

Package: tulpaObs (via r-universe)

October 12, 2026

Title Bayesian Occupancy, Abundance and Detection Models via 'tulpa'

Version 0.6.1

Description Fits Bayesian models for species observed with imperfect detection: single-season and dynamic site occupancy (MacKenzie et al. (2002) [doi:10.1890/0012-9658\(2002\)083\[2248:ESORWD\]2.0.CO;2](https://doi.org/10.1890/0012-9658(2002)083[2248:ESORWD]2.0.CO;2)), N-mixture abundance from replicated counts (Royle (2004) [doi:10.1111/j.0006-341X.2004.00142.x](https://doi.org/10.1111/j.0006-341X.2004.00142.x)), distance sampling, removal and double-observer counts, false-positive and time-to-detection occupancy, integrated models combining several data sources, multi-species and joint species distribution models, and a hurdle model for vegetation cover. Linear predictors take fixed, random, spatial and temporal effects. Each model supplies its observation likelihood to the 'tulpa' engine, which fits it by Laplace approximation, by nested Laplace integration over the hyperparameters, or by Hamiltonian Monte Carlo with the No-U-Turn Sampler (NUTS) using automatic differentiation. Simulation, prediction and diagnostic functions (posterior predictive checks, leave-one-out cross-validation, simulation-based calibration) accompany each model.

URL <https://gillescolling.com/tulpaObs/>,
<https://github.com/gcol33/tulpaObs>

BugReports <https://github.com/gcol33/tulpaObs/issues>

SystemRequirements C++17

Roxygen list(markdown = TRUE)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports graphics, grDevices, Matrix, methods, Rcpp (>= 1.0.12), tulpa (>= 0.7.3), tulpaMesh (>= 0.1.3), parallel, loo (>= 2.7.0), stats, utils

Suggests testthat (>= 3.0.0), betareg, dplyr, ggplot2, INLA, knitr,
ps, rmarkdown, spAbundance, spOccupancy, unmarked, withr

Additional_repositories <https://inla.r-inla-download.org/R/stable>,
<https://gcol33.r-universe.dev>

VignetteBuilder knitr

LinkingTo Rcpp, RcppEigen, tulpa (>= 0.7.2)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake

Repository <https://gcol33.r-universe.dev>

Date/Publication 2026-10-12 11:08:11 UTC

RemoteUrl <https://github.com/gcol33/tulpaobs>

RemoteRef HEAD

RemoteSha c47095846ba6204143ebb5e315116f83db2ea5df

Contents

\$.tobs_fit	5
abun	5
check_model.tobs_fit	6
coef.tobs_batch	7
coef.tobs_fit	8
convergence	8
count	10
cover	11
cover_priors	16
distance	17
distance_laplace	19
distsamp_open	20
double_observer	22
dyn_abun	24
dyn_abun_laplace	25
dyn_int_occu	26
dyn_occu	27
fitted.tobs_fit	28
foray_counts	29
fp_occu	30
fp_occu_laplace	31
gdistremoval	32
glance.tobs_fit	33
int_occu	34
jsdm	35
logLik.tobs_fit	36
meadow_cover	36

ms_abun	37
ms_count	39
ms_distance	40
ms_dyn_occu	42
ms_int_occu	43
ms_occu	44
ms_occu_cover	45
nmix_laplace	48
nmix_laplace_bym2	50
nmix_laplace_car_proper	53
nmix_laplace_icar	55
nmix_laplace_re	57
nmix_site_marginal	62
nobs.tobs_fit	64
occu	64
occu_aggregation_scan	65
occu_categorical	67
occu_cover	68
occu_cover_inputs	75
occu_multi	77
occu_multiscale_cover	78
occu_priors	79
occu_ttd	81
peatland_occu	82
pit_residuals.tobs_fit	83
plot.tobs_aggregation_scan	84
plot.tobs_data	84
ppc	85
predict.cover_fit	86
predict.occu_categorical_fit	87
predict.tobs_fit	88
print.tobs_batch	92
print.tobs_family	92
print.tobs_fit	93
ranef.tobs_fit	93
removal	94
removal_laplace	95
residuals.tobs_fit	97
royle_nichols	97
sbc.tobs_fit	98
simulate.tobs_fit	102
simulate_abun	103
simulate_count	104
simulate_cover	105
simulate_cover_joint	106
simulate_distance	108
simulate_distsamp_open	109
simulate_double_observer	111

simulate_dyn_abun	112
simulate_dyn_int_occu	113
simulate_dyn_occu	114
simulate_fp_occu	116
simulate_gdistremoval	117
simulate_int_occu	118
simulate_jsdm	119
simulate_ms_abun	120
simulate_ms_count	122
simulate_ms_distance	123
simulate_ms_dyn_occu	124
simulate_ms_int_occu	126
simulate_ms_occu	127
simulate_ms_occu_cover	128
simulate_ms_occu_cover_spatial	129
simulate_occu	131
simulate_occu_categorical	132
simulate_occu_cover	133
simulate_occu_multi	136
simulate_occu_multiscale_cover	137
simulate_occu_ttd	138
simulate_removal	139
simulate_royle_nichols	140
simulate_t_occu	142
summary.tobs_data	143
summary.tobs_fit	143
t_occu	144
tidy.tobs_fit	145
tobs	145
tobs_associations	154
tobs_check_id	156
tobs_criteria	157
tobs_data	158
tobs_fit_intervals	160
tobs_format	161
tobs_format_ms	162
tobs_get	163
tobs_gof_tests	164
tobs_marginal_effect	165
tobs_multiarm_methods	166
tobs_predict_spatial	167
tobs_richness	169
tobs_stack	169
tobs_terms	171
update.tobs_fit	175
within_between	176

\$.tobs_fit	<i>Access spOccupancy-compatible fields from tobs fits</i>
-------------	--

Description

Allows accessing spOccupancy-style fields (e.g., `$beta.samples`, `$psi.samples`) on `tobs_fit` objects. Since `tobs` stores actual posterior draws, this is a thin remapping layer.

Usage

```
## S3 method for class 'tobs_fit'
x$name
```

Arguments

<code>x</code>	A <code>tobs_fit</code> object.
<code>name</code>	Field name to access.

Value

The requested field value.

<code>abun</code>	<i>N-mixture abundance family</i>
-------------------	-----------------------------------

Description

Latent Poisson (or NB) abundance with binomial detection per visit (Royle 2004).

Usage

```
abun(K.max = NULL, mixture = c("poisson", "negbin", "zip", "zinb"))
```

Arguments

<code>K.max</code>	upper bound for the latent-abundance marginal sum (the exact integration over N is truncated at $K.max$). <code>NULL</code> (default) lets the engine pick $\max(y) + 100$ (matching <code>unmarked::pcount()</code>); raise it if a fit warns that the posterior over N puts mass on the boundary.
<code>mixture</code>	latent-abundance distribution, both fitted via tulpa's closed-form marginal Laplace engine. "poisson" is the default; "negbin" (negative binomial, $\text{Var}(N) = \lambda + \lambda^2 / r$) adds an overdispersion parameter r . Non-spatially the log size <code>log_r</code> is estimated jointly with the coefficients and reported with an SE; on the areal-spatial path (<code>method = "nested_laplace"</code>) r is integrated over the outer hyperparameter grid and reported as a posterior mean / sd. Named mixture (after <code>unmarked::pcount()</code>) to avoid collision with the model-type family argument of <code>tobs()</code> .

Details

Royle's (2004) N-mixture model: latent abundance $N_i \sim \text{Poisson}(\lambda_i)$ with $\log \lambda_i = X_{\lambda} \beta_{\lambda}$ and counts $y_{ij} | N_i \sim \text{Binomial}(N_i, p_{ij})$ with $\text{logit } p_{ij} = X_p \beta_p$. The abundance formula is the `tobs()` formula; the per-visit detection formula is `detection`. The marginal likelihood integrates N out exactly, so no route runs an EM over N .

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – a direct Laplace approximation over the closed-form marginal with an observed-Fisher covariance; a grouped random effect on one arm is integrated by adaptive Gauss-Hermite quadrature.
- `method = "nested_laplace"` – an areal `icar()` / `bym2()` / `car_proper()` field on the abundance arm, its hyperparameters integrated on an outer grid (grid-integrated coefficient covariance).
- `method = "nuts"` – the in-tree C++ NUTS sampler over the same marginal, Poisson or negative binomial (`log_r` sampled), warm-started at the Laplace mode; with an areal term a fixed-hyper non-centered field on the abundance arm (`icar()` / `car_proper()` / `bym2()`); "`zip`" / "`zinb`" are "`laplace`"-only, and a field does not combine with a random effect here.

Examples

```
sim <- simulate_abun(N = 120, J = 4, n.abund.covs = 1, n.det.covs = 1, seed = 1)
fit <- tobs(~ abund_cov1, data = sim$data, family = abund(),
           detection = ~ det_cov1, y = sim$y, method = "laplace")
summary(fit)
```

check_model.tobs_fit *Comprehensive model checking*

Description

The `tulpa::check_model()` method for `tobs_fit`: the criteria, the posterior-predictive check and the goodness-of-fit tests in one call, printed as a report and drawn as the diagnostic panel.

Usage

```
## S3 method for class 'tobs_fit'
check_model(
  object,
  coords = NULL,
  n.samples = 250,
  nsim = NULL,
```

```

    seed = NULL,
    plot = TRUE,
    ...
)

```

Arguments

object	A tobs_fit object.
coords	Optional n_sites x 2 coordinate matrix. Adds Moran's I on the occupancy residuals and the correlogram panel.
n.samples	Posterior samples for simulation tests.
nsim	The same budget under the name <code>tulpa::check_model()</code> 's default uses; overrides n.samples when given.
seed	Optional random seed for the simulation tests; the caller's RNG stream is re-stored afterwards.
plot	Draw the panel. FALSE prints the report alone, for a log or a headless run.
...	Must be empty; an unrecognised argument is an error.

Details

The engine's default method builds its panel from `fitted()` and `residuals()` as numeric vectors, which a latent-state fit does not have – `fitted()` returns `list(psi, p, z)` and `residuals()` one series per process, so the response cannot be resolved. This method reads the same quantities through the family's own doors instead: `pit_residuals()`, `ppc()`, `test_dispersion()` and `tulpa::moran_i()`.

Value

Invisibly, the diagnostic results: `waic`, `dic`, `cpo`, `ppc`, `zero_inflation`, `dispersion`, `pit`, `uniformity`, and `moran` when `coords` is supplied.

coef.tobs_batch	<i>Per-species coefficients from a batched occu_cover fit.</i>
-----------------	--

Description

Per-species coefficients from a batched occu_cover fit.

Usage

```

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

```

Arguments

object	A tobs_batch returned by <code>tobs()</code> on multi-response y.
...	Passed to each per-species <code>coef()</code> .

Value

A named list of per-species coefficient vectors.

<code>coef.tobs_fit</code>	<i>Coefficients for a tobs_fit</i>
----------------------------	------------------------------------

Description

The fixed-effect estimates as one named numeric vector, named `<arm>_<term>` as `stats::vcov()` and `stats::confint()` name them: `psi_(Intercept)`, `p_det_cov1`, `lambda_elev`, ... The visit-level detection coefficients (`p_visit_<cov>`) belong to the detection arm. A coordinate that enters no single arm, such as the `log_r` overdispersion, keeps its own name.

Usage

```
## S3 method for class 'tobs_fit'
coef(object, arm = NULL, ...)
```

Arguments

<code>object</code>	A <code>tobs_fit</code> object.
<code>arm</code>	Optional name of one linear predictor (" <code>psi</code> ", " <code>p</code> ", ...). When given, only that arm's coefficients are returned, named by their design column without the arm prefix.
<code>...</code>	Ignored.

Value

A named numeric vector.

<code>convergence</code>	<i>Convergence record for a fitted model</i>
--------------------------	--

Description

The public accessor for whether a fit converged, with one return shape across every `tobs()` family. Every family's optimiser / EM / sampler verdict is recorded under `fit$convergence`, and a Laplace-family fit that stops without meeting its criterion warns once, naming `control$max.iter`. `convergence()` returns the full record (`converged`, `n_iter`, `sla_status` when the simplified-Laplace marginals were used, and the per-group AGHQ solve status on the families fitted by that engine), and `converged()` returns the single logical.

Usage

```
convergence(object, ...)

## S3 method for class 'tobs_fit'
convergence(object, ...)

converged(object, ...)

## S3 method for class 'tobs_fit'
converged(object, ...)
```

Arguments

<code>object</code>	A fitted <code>tobs_fit</code> (occupancy / abundance / cover / ...).
<code>...</code>	Ignored.

Value

`convergence()`: a list with `converged` (logical), `n_iter` (integer, NA for grid / closed-form fits with no iteration count), and `sla_status` (character, when present). A fit whose random effects were integrated by adaptive Gauss-Hermite quadrature also carries `group_ok` (logical, one entry per group: FALSE where that group's posterior solve failed and its estimates are NA), `groups_failed` (their indices) and `groups_failed_names` (their labels where the family names its groups – species, for the community models). `converged` is FALSE whenever any group failed, since the quantities that group contributes to are not estimates of anything; filter on these rather than on the text of the warning the engine raises. A single-species fit whose grouped random effects were integrated that way carries `re_boundary`: one record per random-effect SD (`sigma_<g>_<coef>`) and, on a correlated block, per correlation (`cor_<g>_<ci>_<cj>`), with the boundary Wald statistic, its critical value and distinguishable; FALSE means the SD cannot be told from zero or the correlation from ± 1 , and the fit says so in one message ("boundary (singular) fit"). `converged()`: a single TRUE / FALSE.

Examples

```
sim <- simulate_occu(N = 100, J = 3, n.occ.covs = 1, n.det.covs = 1,
                    seed = 1)
fit <- tobs(~ occ_cov1, data = sim$data, family = occu(),
           detection = ~ det_cov1, y = sim$y, method = "laplace",
           control = list(verbose = FALSE, progress = FALSE))
converged(fit)
convergence(fit)$n_iter
```

count

*Count / relative-abundance GLMM family (no detection process)***Description**

A generalized linear model on an observed count (or continuous) response directly, with no detection sub-model and no latent abundance to marginalize. The abundance analogue of `jsdm()`: where `abun()` fits Royle's N-mixture (latent N plus imperfect detection), `count()` fits the observed response as a plain GLMM. This is the "relative abundance" model of `spAbundance` (`abund`): one value per site, log-link Poisson / negative-binomial counts or an identity-link Gaussian response.

Usage

```
count(response = c("poisson", "negbin", "gaussian", "binomial"))
```

Arguments

response The response distribution: "poisson" (log link), "negbin" (negative binomial, log link, an estimated size / dispersion), "gaussian" (identity link, an estimated residual variance), or "binomial" (logit link, k successes out of n trials per site). The binomial response is the detection-free binomial GLMM of `spOccupancy` (`svcPGBinom`): supply the per-site trial count as `trials = on` `tobs()` (default 1, i.e. a Bernoulli response). Unlike the count / Gaussian areal fit, a binomial areal field is identified (the variance is pinned by n), so an `icar()` / `car_proper()` term composes with any trials.

Details

The response is a numeric vector (one value per site), supplied via `y =` or on a two-sided formula left-hand side (`count.value ~ predictors`).

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – one tulpa GLMM block; the negative-binomial size and the Gaussian residual variance are estimated by an outer dispersion loop.
- `method = "nested_laplace"` – a plain areal field (`icar()` / `car_proper()`) on the formula as a latent GMRF, integrated over its hyperparameters by the nested-Laplace EM; the areal term is required on this route.

See Also

`abun()` (N-mixture: latent abundance + detection), `jsdm()` (occurrence GLMM, no detection).

Examples

```
f <- count("poisson")
f
```

cover	<i>Cover hurdle family (vegetation cover, MOTIVATE pattern)</i>
-------	---

Description

Latent presence (Bernoulli) plus conditional positive cover (beta or lognormal). Does not share the replicate-detection assumption of the other families – see `vignette("families")` for the conceptual caveat.

Usage

```
cover(
  response = c("beta", "beta_oi", "lognormal", "lognormal_trunc", "ordinal", "gaussian"),
  breaks = NULL
)
```

Arguments

response	likelihood for the positive cover part: • "beta" – cover in (0, 1) with a logit link; • "beta_oi" – a one-inflated Beta: a point mass at cover = 1 (plots recorded at full cover, a genuine boundary mass rather than a near-1 continuous value) plus a Beta on the interior (0, 1). The inflation probability is a constant, estimated as the share of positive plots at the ceiling; the interior Beta is fit on the (0, 1) plots. Reported as <code>pi_one</code>. (With the zero hurdle this is the zero-one-inflated Beta.); • "lognormal" – a Gaussian on log-cover (unbounded above); • "lognormal_trunc" – a Gaussian on log-cover upper-truncated at $\log(1) = 0$, the bounded-support form for cover in (0, 1] (it cannot place mass above cover = 1 the way "lognormal" can). Requires <code>method = "nested_laplace"</code>; • "ordinal" – an interval-censored Gaussian on log-cover with KNOWN class thresholds (an ordered probit for Braun-Blanquet-style class data), set with <code>breaks</code>. Requires <code>method = "nested_laplace"</code>; • "gaussian" – an identity-Gaussian magnitude (the delta-normal hurdle): Bernoulli presence times a Gaussian on the raw response. For a pre-transformed or otherwise unbounded response on a real scale (it permits negative fitted values), NOT for cover fractions in (0, 1); those stay on "beta" / "lognormal" / "ordinal". Presence is the nonzero sentinel ($y \neq 0$) rather than $y > 0$.
breaks	for positive = "ordinal" only, the interior cover-class boundaries on the (0, 1) cover-fraction scale (strictly ascending, all in (0, 1)); the open outer classes are added automatically. NULL for the other families.

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – two independent `tulpa_laplace()` fits, one per arm (presence, positive), non-spatial; the default.
- `method = "laplace_sla"` – the same two fits with simplified-Laplace skew-corrected marginals.
- `method = "nested_laplace"` – the joint nested-Laplace engine: shared or arm-specific areal fields, weighted trend fields, `temporal()` and `re()` blocks, every hyperparameter integrated on an outer grid; the route the `"lognormal_trunc"` and `"ordinal"` arms require.
- `method = "nested_laplace_sla"` – the nested route with skew-corrected marginals.
- `method = "nuts"` – the non-spatial sampler over the exact two-arm coefficient vector `c(beta_presence, beta_positive, log_disp)`, giving calibrated intervals and a per-draw pointwise likelihood for WAIC / LOO; `"beta"`, `"lognormal"` and `"gaussian"` arms only, and any structured term errors with a pointer to `"nested_laplace"`.

Response on the formula left-hand side

The cover response is a single length-N vector, so it may sit on the top formula left-hand side and `y` is dropped. The two calls are equivalent:

```
# response on the LHS (y = omitted)
tobs(cover.flat ~ time.sc + habitat +
      spatial(~ 1 + time.sc || cell_idx, graph = adj),
      data = dat, family = cover(response = "beta"),
      method = "nested_laplace")

# the same fit with a one-sided formula and an explicit y =
tobs(~ time.sc + habitat +
      spatial(~ 1 + time.sc || cell_idx, graph = adj),
      y = dat$cover.flat, data = dat, family = cover(response = "beta"),
      method = "nested_laplace")
```

Naming the response makes the per-arm spatial labels read naturally: `cover()` splits `cover.flat` into a presence arm and a positive arm, the arm names that per-arm formulas and `share()` address. The LHS is evaluated against data (then the calling environment), so it may be a bare column or an expression.

Joint nested-Laplace engine – spatial-prior parameterisation

When fitted with `method = "nested_laplace"` and an areal spatial term in the latent-presence formula (`bym2(graph = adj)`, or `car()` / `car_proper()`), the engine identifies a single latent field z per region and parameterises the two arms as $\eta_{\text{occ}} = X \beta_{\text{occ}} + \sigma * z$ and $\eta_{\text{pos}} = X \beta_{\text{pos}} + \alpha * \sigma * z$. Both BYM2 sub-blocks (ϕ , θ) are subject to hard sum-to-zero constraints (see `.joint_inner_var()` for the math): the cover-arm intercept `beta_pos[1]` is identified as the population mean of η_{pos} under $\text{mean}(z) = 0$. Any

simulator generating data for this engine must demean both `phi_f` and `theta_f` before scaling, otherwise the estimator targets $\beta_{\text{pos}_0\text{truth}} + \alpha * \text{mean}(w_{s\text{sim}})$ and coverage of the *population* truth collapses with α . See `simulate_cover_joint()` for a ready-made demeaned simulator.

Spatially varying trend (a weighted areal term)

A spatially varying trend is model structure, so it is declared in the formula as a `SECOND`, weighted areal term on the same graph as the intercept field, not through `control`:

```
~ time.sc +
  icar(graph = adj, group.var = "cell_idx") +
  icar(graph = adj, weight = time.sc, group.var = "cell_idx")
```

The unweighted term is the shared intercept field; the weighted term `icar(..., weight = col)` is the trend field, whose contribution to each arm's predictor is $\text{weight}_i * z[\text{cell}_i]$, coupled onto the cover arm with its own scale (`alpha_trend`, reported in `fit$alpha_trend / fit$sigma_trend`) integrated over the outer grid. The umbrella spelling `spatial(graph = adj, model = "icar", weight = col)` resolves identically. The trend block inherits the intercept block's amplitude; give it its own with `share(spatial(), terms = list(intercept = ..., trend = ...))`. Requires method = "nested_laplace". A coupled trend cannot currently combine with `temporal()` / `re()` blocks in the same fit.

The axis the amplitude rides carries prior structure – a point mass at $\alpha = 0$ ("no coupling") and a log-spaced slab above it – so it is set in one of two ways. `share(alpha = grid(...))` STATES its nodes, and with them that structure. `share(alpha = grid(n = 9))` states a RESOLUTION: the engine re-reads its own axis with that many slab nodes, point mass and slab bounds unchanged, so sharpening the axis never restates its structure. `terms =` gives either, per block. The resolution is the only way to raise this axis, because it does not densify when the donor `control$sigma.grid` does: on an informative data set the outer grid's quadrature effective sample size saturates on the copy amplitude while every other axis tracks the request (measured engine-side, `NOTES_measurements.md`). One block takes one of the two; giving both is an error. `control$alpha.grid[.trend]` and `control$alpha.n[.trend]` are the wire format these compile into and are refused as user input.

Set it when the fit's hyperparameter intervals are reported. On the coupled SBC fixture the declared resolution leaves the field SD miscalibrated once the data are informative – at 10 visits per site its uniformity p-value is $9.1e-05$, against 0.17 at 3 visits – and `control$alpha.n = 21` returns every scored quantity to nominal at both. Thirteen slab nodes do not (the field SD still reads $4.5e-03$), and the price is a 2.3-3x longer fit, since the outer grid is a tensor. Point estimates are not affected the way the intervals are; the measurement is in `NOTES_measurements.md`.

Varying-coefficient spatial bar (the compact single-term form)

The intercept field plus its weighted trend field can also be written as one `spatial()` term carrying an lme4-style coefficient formula:

```
~ time.sc + habitat +
  spatial(~ 1 + time.sc || cell_idx, graph = adj)
```

The bar left-hand side spells the coefficient fields: the intercept column (1) is the unweighted field; each covariate column (time.sc) is a weight-scaled coefficient field ($\text{weight}_i * z[\text{cell}_i]$). The bar right-hand side (cell_idx) is the graph node index (the areal group.var); || requests independent intercept and slope fields, a single | makes them correlated. This desugars to exactly the two-term weighted-areal form above, so the two spellings give the same fit.

Choosing a field's arm

placement and share(): The cover hurdle's two arms are presence (the $y > 0$ Bernoulli arm) and positive (the $y | y > 0$ arm); summary() and the coefficient output print these same labels. A field is placed on an arm by writing it in that arm's per-arm formula (presence = ~ ..., positive = ~ ...), and shared across arms with share(). A field in the single shared formula reaches both arms.

Both arms (shared / copied). A field in the shared formula is one presence-anchored latent copied onto the positive arm with an estimated coupling per coefficient field (alpha for the intercept field, alpha_trend for the trend field), marginalized on the outer grid:

```
# one latent field, shared by both arms
tobs(cover.flat ~ time.sc + habitat +
      spatial(~ 1 + time.sc || cell_idx, graph = adj),
      data = dat, family = cover(response = "beta"), method = "nested_laplace")
```

```
presence: eta_presence = ... + u_cell + time.sc * s_cell
positive: eta_positive = ... + alpha * u_cell + alpha_trend * time.sc * s_cell
```

Per-arm formulas make the shared field explicit: place it on presence and copy it onto positive. share(spatial()) estimates the coupling on the default axis; share(spatial(), alpha = grid(c(...))) integrates it over supplied nodes, alpha = grid(n = 9) over the engine's own axis read at n slab nodes, and alpha = 0.5 fixes it. share(spatial(), prior = list(...)) regularizes the coefficient itself; a fit copying both the intercept and a weighted trend block is refused it, since one prior reaches the engine per fit.

```
tobs(presence = ~ time.sc + habitat +
      spatial(~ 1 + time.sc || cell_idx, graph = adj),
      positive = ~ time.sc + habitat + share(spatial()),
      data = dat, family = cover(response = "beta"), method = "nested_laplace")
```

One arm only (free / separate). A field written in one arm's formula is a separate latent on that arm alone, with its own precision and no cross-arm coupling. A field in each arm's formula gives two independent latents:

```
tobs(presence = ~ time.sc + spatial(~ 1 + time.sc || cell_idx, graph = adj),
      positive = ~ time.sc + spatial(~ 1 + time.sc || cell_idx, graph = adj),
      data = dat, family = cover(response = "beta"), method = "nested_laplace")
```

```
presence: eta_presence = ... + u_presence_cell + time.sc * s_presence_cell
positive: eta_positive = ... + u_positive_cell + time.sc * s_positive_cell
```

The free fit reports `sigma_armspecific`. Arm-specific fields are their own spatial structure: they do not combine with a shared field, a weighted trend, or `temporal()` / `re()` in the same formula, and at most one targets each arm.

The `||` and `|` axis is separate from shared / free: `||` makes the intercept and slope fields independent, while a single `|` makes them correlated (a free cross-covariance, MCAR). A correlated `|` bar shared across both arms (in the shared formula, or on presence with `share()` on positive) is copied with one estimated amplitude; placed on one arm's formula it is a free-Sigma correlated field on that arm alone, no cross-arm copy.

field in the shared formula, no <code>share()</code>	one presence-arm latent, pinned at $\alpha = 0$
field + <code>share()</code> (either spelling)	one shared / copied latent (presence anchor, coupling estimated)
a field in each arm's formula	separate / free latents, no coupling
<code> </code>	independent intercept and slope coefficient fields
<code> </code>	correlated (MCAR) coefficient fields, copy-only

Without a `share()` (and without `control$alpha.grid`) the presence field is NOT carried onto the cover arm: the amplitude is pinned at $\alpha = 0$ and the field rides the presence arm alone. Structure nobody wrote is not in the model, the reading `occu_cover()` and `occu_multiscale_cover()` also have. Write `+ share(spatial())` in the shared formula, or place the field on presence and `share(spatial())` on positive, to couple the two arms. A correlated `|` bar is the exception: that spelling is copy-only by definition, so the bar itself states the coupling.

A formula bar is a random effect, not a spatial field

A bare `lme4` bar in the formula – `(1 | cell)`, `(1 + x | cell)`, `(x || cell)` – is a grouped random effect, not a spatial field, even when the grouping factor names the areal graph nodes. The engine's inline-MCAR call `tulpa::spatial(graph, ~ 1 + x | cell)` reads `| cell` as a separable spatial field, but the same spelling inside a `cover()` formula is parsed as an IID random effect on `cell`. For a spatial field on the cells write either the compact `spatial()` bar of the section above, `spatial(~ 1 + x || cell, graph = adj)`, or the two-term weighted-areal form, `icar(graph = adj, group.var = "cell") + icar(graph = adj, weight = x, group.var = "cell")`. When a formula carries a bar whose grouping factor is also an areal term's `group.var`, `cover()` emits a one-time message noting the bar is fitted as a random effect; suppress it with `base::suppressMessages()`.

Checkpoint / resume

A spatial cover-hurdle fit integrates over a large outer hyperparameter grid and can run for hours. `control$checkpoint = list(path = "fit.ckpt", resume = TRUE)` makes the joint engine append each completed grid cell to path; a `resume = TRUE` run loads the finished cells and solves only the rest, reproducing the from-scratch fit, so a killed or rebooted fit resumes instead of restarting. `resume = FALSE` starts a fresh file. Forwarded to `tulpa::tulpa_nested_laplace_joint()`.

Examples

```
sim <- simulate_cover(N = 200, seed = 1)
fit <- tobs(~ x, data = sim$data, family = cover("beta"), y = sim$y)
summary(fit)
```

cover_priors

Weakly-informative priors for cover-hurdle Laplace fits

Description

Builds an opt-in fixed-effect prior for `cover()` models fit with a Laplace method. The cover hurdle has two arms with their own coefficients – an occurrence arm (Bernoulli presence) and a positive-cover arm (beta or lognormal) – each with an intercept and a slope bucket. The penalty is the quadratic + $\sum((\text{beta}_j - \mu_j)^2 / (2 \text{sd}_j^2))$ added to each arm's negative log-likelihood, the same mechanism `occu_priors()` uses.

Usage

```
cover_priors(
  occ.intercept = list(mean = 0, sd = 2),
  occ.slope = list(mean = 0, sd = 2.5),
  pos.intercept = list(mean = 0, sd = 3),
  pos.slope = list(mean = 0, sd = 2.5)
)
```

Arguments

<code>occ.intercept</code>	Prior on the occurrence (presence) intercept, logit scale. <code>list(mean, sd)</code> . Default <code>list(mean = 0, sd = 2)</code> .
<code>occ.slope</code>	Prior on occurrence slopes. Default <code>list(mean = 0, sd = 2.5)</code> .
<code>pos.intercept</code>	Prior on the positive-cover intercept (logit scale for "beta", log scale for "lognormal"). Default <code>list(mean = 0, sd = 3)</code> .
<code>pos.slope</code>	Prior on positive-cover slopes. Default <code>list(mean = 0, sd = 2.5)</code> .

Details

Cover priors are opt-in: without priors, a cover fit is unpenalised. The main use is regularising perfect separation in the occurrence arm at small N. Set any `sd = Inf` to drop that component.

Value

A `cover_priors` object, ready to pass to `tobs(..., priors = ...)`.

Coverage

Applied on the separate-Laplace path (`method = "laplace" / "laplace_sla"`) when the formula has no spatial term. Both arms are penalised: the occurrence and lognormal-positive arms through `tulpa::tulpa_laplace()`, and the beta-positive arm through `tulpa::tulpa_laplace_beta()`. The joint nested-Laplace path (`method = "nested_laplace" / "nested_laplace_sla"`, used for spatial cover formulas) also threads the priors, as a per-arm `beta_prior_mean / beta_prior_prec` on the joint engine's occurrence and positive responses. The separate-Laplace path keeps rejecting a prior alongside a spatial term (that solver carries its own); add the spatial term through the nested-Laplace path to combine the two.

See Also

[occu_priors\(\)](#)

Examples

```
# regularise the occurrence arm, leave the positive arm unpenalised
priors <- cover_priors(pos.intercept = list(mean = 0, sd = Inf),
                     pos.slope     = list(mean = 0, sd = Inf))

priors

sim <- simulate_cover(N = 150, seed = 1)
fit <- tobs(cover ~ x, data = sim$data,
            family = cover(response = "lognormal"),
            method = "laplace", priors = priors)

coef(fit)
```

distance	<i>Binned distance-sampling family</i>
----------	--

Description

Latent abundance $N_i \sim \text{Poisson}(\lambda_i)$ (or negative binomial) in a covered region, observed through a half-normal or hazard-rate detection function over distance bins. With B bins the detected counts are multinomial over (bin 1, ..., bin B , undetected) with cell probabilities $\pi_b = \int_{\text{bin } b} g(x; \sigma) f(x) dx$ and $1 - \sum_b \pi_b$, where $f(x)$ is the distance density (uniform for a line transect, proportional to distance for a point transect). The latent N is summed out in closed form (truncation $K_{\max}$), so the fit is a direct Laplace approximation (no EM), with a NUTS path over the same marginal; the per-bin detection integrals are evaluated by Gauss-Legendre quadrature.

Usage

```
distance(
  key = c("halfnorm", "hazard"),
  transect = c("line", "point"),
  cutpoints = NULL,
  K.max = NULL,
  mixture = c("poisson", "negbin")
)
```

Arguments

key	detection-function key. "halfnorm" (default) or "hazard" (the hazard-rate shape b is estimated as a scalar, reported as <code>log_shape</code>).
transect	"line" (default; perpendicular distances uniform) or "point" (radial distances, density proportional to distance).

cutpoints	numeric distance-bin edges, length $n_bins + 1$ ($0 = c_0 < c_1 < \dots < c_B$). Required.
K.max	upper bound for the latent-abundance marginal sum (the exact integration over N is truncated at K.max). NULL (default) lets the engine pick $\max(y) + 100$ (matching <code>unmarked::pcount()</code>); raise it if a fit warns that the posterior over N puts mass on the boundary.
mixture	latent-abundance distribution, both fitted via tulpa's closed-form marginal Laplace engine. "poisson" is the default; "negbin" (negative binomial, $\text{Var}(N) = \lambda + \lambda^2 / r$) adds an overdispersion parameter r . Non-spatially the log size $\log_r$ is estimated jointly with the coefficients and reported with an SE; on the areal-spatial path (<code>method = "nested_laplace"</code>) r is integrated over the outer hyperparameter grid and reported as a posterior mean / sd. Named mixture (after <code>unmarked::pcount()</code>) to avoid collision with the model-type family argument of <code>tobs()</code> .

Details

The `tobs()` formula is the abundance ($\log \lambda$) model; detection is the site-level detection-scale ($\log \sigma$) model. The response y is an $n_sites \times n_bins$ integer matrix of per-bin detected counts. The bin edges and transect geometry travel with the family: `distance(cutpoints = ...)`.

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – a direct Laplace approximation over the closed-form marginal (Poisson or negative binomial); a grouped random effect on the abundance arm is integrated by adaptive Gauss-Hermite quadrature.
- `method = "nested_laplace"` – an areal `icar()` / `car_proper()` / `bym2()` field on the abundance arm, optionally composed with a `temporal()` field, through the shared areal-BFGS driver; a hazard-rate fit recovers the shape alongside the field.
- `method = "nuts"` – the in-tree C++ NUTS sampler over the same marginal, with a fixed-hyper areal or temporal field block on the abundance arm (one of the two, not both); the hazard-rate log-shape rides alongside.

References

Buckland, S. T., Anderson, D. R., Burnham, K. P., Laake, J. L., Borchers, D. L., Thomas, L. (2001). Introduction to Distance Sampling. Oxford. Royle, J. A., Dawson, D. K., Bates, S. (2004). Modeling abundance effects in distance sampling. *Ecology* 85, 1591-1597.

Examples

```
sim <- simulate_distance(N = 200, key = "halfnorm", transect = "line",
                        n.abund.covs = 1, n.sigma.covs = 1, seed = 1)
fit <- tobs(~ abund_cov1, data = sim$data,
```

```

family = distance(key = "halfnorm", transect = "line",
                  cutpoints = sim$cutpoints),
detection = ~ sigma_cov1, y = sim$y, method = "laplace")
summary(fit)

```

distance_laplace

*Laplace fit of the binned distance-sampling abundance model***Description**

Maximum-likelihood fit (non-spatial, fixed effects only) of a binned distance-sampling abundance model with a Poisson or negative-binomial abundance mixing distribution and a half-normal or hazard-rate detection key. Latent abundance $N_i \sim \text{Poisson}(\lambda_i)$ (or $\text{NegBin}(\lambda_i, r)$) is summed out in closed form (truncation $K_{\max}$); the detection integrals over distance bins are evaluated by Gauss-Legendre quadrature.

Usage

```

distance_laplace(
  y,
  X_lambda,
  X_sigma,
  cutpoints,
  key = c("halfnorm", "hazard"),
  transect = c("line", "point"),
  mixture = c("P", "NB"),
  beta_lambda_init = NULL,
  beta_sigma_init = NULL,
  eta_b_init = NULL,
  log_r_init = NULL,
  r_max = 1e+05,
  K_max = NULL,
  headroom = NULL,
  quad_order = 64L,
  max_iter = 100L,
  tol = 1e-06,
  verbose = FALSE,
  quad_xptr = NULL
)

```

Arguments

<code>y</code>	Integer matrix [$n_{\text{sites}} \times n_{\text{bins}}$] of per-bin detected counts.
<code>X_lambda</code>	Numeric matrix [$n_{\text{sites}} \times p_{\text{lambda}}$] of abundance covariates.
<code>X_sigma</code>	Numeric matrix [$n_{\text{sites}} \times p_{\text{sigma}}$] of detection-scale covariates (the log sigma linear predictor).

cutpoints	Numeric bin edges, length $n_{\text{bins}} + 1$ ($0 = c_0 < c_1 < \dots < c_B$).
key	"halfnorm" (default) or "hazard" detection key.
transect	"line" (default, distances uniform) or "point" (radial, density proportional to distance).
mixture	"P" (Poisson, default) or "NB" (negative binomial).
beta_lambda_init, beta_sigma_init, eta_b_init, log_r_init	Optional warm starts.
r_max	Upper bound on the NB size r (NB only, default $1e5$).
K_max	Marginal-sum truncation; defaults to $3 * \max(\text{site total}) + 100$ (applied per site, see headroom).
headroom	Latent- N states summed above each site's own detected total. NULL (default) derives it from K_{max} : an unset K_{max} caps each site at its own total plus $2 * \max(\text{site total}) + 100$, an explicit K_{max} truncates globally and uncapped. A negative value disables the per-site cap.
quad_order	Gauss-Legendre nodes per bin (default 64).
max_iter	Newton iteration budget (default 100).
tol	Gradient-norm convergence tolerance (default $1e-6$).
verbose	Print per-iteration progress.
quad_xptr	Optional pre-built quadrature (from <code>cpp_distance_build_quad()</code>) for a caller fitting several species/starts against the same cutpoints / transect / quad_order; built once and reused instead of every call rebuilding it. Internal; defaults to building it fresh.

Value

A list of class `distance_fit` with `beta_lambda`, `beta_sigma`, `shape / eta_b` (hazard-rate), `log_r / r` (NB), `log_lik`, `vcov`, `H_obs`, per-site `mean_N / var_N / p_det / boundary_weight`.

References

Buckland, S. T., et al. (2001). Introduction to Distance Sampling. Oxford. Royle, J. A., Dawson, D. K., Bates, S. (2004). Modeling abundance effects in distance sampling. *Ecology* 85, 1591-1597.

distsamp_open	<i>Open-population distance-sampling family</i>
---------------	---

Description

The unmarked `distsampOpen` model: a Dail-Madsen open N -mixture (as `dyn_abun()`) observed by distance sampling at each primary period – the open-population counterpart of the single-season `gdistremoval()`. Initial abundance $N_1 \sim \text{Poisson}(\lambda)$, then $N_t = \text{Binomial}(N_{t-1}, \omega) + \text{Poisson}(\gamma)$ (apparent survival ω + recruitment γ); at each primary period the detected birds are distance-sampled into bins.

Usage

```
distsamp_open(
  transect = c("line", "point"),
  cutpoints = NULL,
  K.max = NULL,
  mixture = c("poisson", "negbin", "zip", "zinb"),
  dynamics = c("constant", "notrend", "trend", "autoreg", "ricker", "gompertz")
)
```

Arguments

transect	Transect geometry: "line" (default) or "point".
cutpoints	Distance-bin edges, length $\dim(y)[2] + 1$, strictly increasing and starting at ≥ 0 .
K.max	Truncation for the latent abundance HMM. Defaults to $3 * \max(\text{period total}) + 40$.
mixture	Initial-abundance mixing distribution. "poisson" (default), "negbin" (negative binomial, $\text{Var}(N_1) = \lambda + \lambda^2 / r$, with an overdispersion r estimated jointly and reported as $\log_r$), "zip" (zero-inflated Poisson) or "zinb" (zero-inflated negative binomial). A structural-zero site is never occupied across any primary period, so all its band counts are zero; the observed per-site marginal is the two-component mixture $\omega * 1\{\text{all zero}\} + (1 - \omega) * L_{\text{open}}$, with L_{open} the exact open-population distance marginal and ω an intercept-only structural-zero probability reported as zi_logit (distinct from ω , the survival arm). The negative-binomial size / zero-inflation is layered over the same forward-HMM marginal; the Poisson path is unchanged.
dynamics	Population-dynamics form for the transition $N_t \mid N_{\{t-1\}}$ (following <code>unmarked::distsampOpen()</code>): "constant" (default) is $N_t = \text{Binomial}(N_{\{t-1\}}, \omega) + \text{Poisson}(\gamma)$; "notrend" ties recruitment so the expected abundance is stationary ($\gamma = (1 - \omega) * \lambda$, no free γ arm); "trend" is an exponential-growth process $N_t \sim \text{Poisson}(N_{\{t-1\}} * \gamma)$ with no survival term (ω unused); "autoreg" adds density-dependent recruitment $\text{Poisson}(N_{\{t-1\}} * \gamma)$ on top of survival; "ricker" and "gompertz" are the density-regulated recruitment forms with an estimated carrying capacity K (reported as K , on the log scale, modelled through the $\omega = \sim \dots$ formula slot) and a growth rate reported as r (modelled through the $\gamma = \sim \dots$ slot). "constant" / "notrend" fit with the exact analytic gradient; the density-dependent forms (trend / autoreg / ricker / gompertz) use the exact forward-HMM marginal with a numeric gradient.

Details

The distance-band allocation is conditional on the period total detected, so it factors out of the abundance HMM (the `gdistremoval()` trick): the marginal is the `dyn_abun()` forward recursion with the detection probability set to the overall distance detection `pdist`, plus the per-period band multinomials.

Four site-level arms: log abundance λ (the `tobs()` formula), log distance scale σ (detection), logit survival ω ($\omega = \sim \dots$), and log recruitment γ ($\gamma = \sim \dots$), the last two

intercept-only by default.

Value

A `tobs_family` object.

Inputs

`y` is a 3D array [`n_sites` x `n_bins` x `n_seasons`] of per-distance-band counts at each primary period (secondary occasions absorbed into the period total).

Engines

- `method = "laplace"` – the forward-HMM marginal maximised (BFGS) with an observed-information covariance; the only route (no field, no sampler).

References

Dail, D., Madsen, L. (2011). Models for estimating abundance from repeated counts of an open metapopulation. *Biometrics* 67, 577-587. Sollmann, R., Gardner, B., Chandler, R. B., Royle, J. A., Sillett, T. S. (2015). An open-population hierarchical distance sampling model. *Ecology* 96, 325-331.

See Also

[gdistremoval\(\)](#) (single-season), [dyn_abun\(\)](#) (open N-mixture), [distance\(\)](#).

Examples

```
f <- distsamp_open(transect = "line", cutpoints = c(0, 10, 20, 30, 40),
                  dynamics = "notrend")
f
```

double_observer

Double-observer abundance family

Description

Abundance from a double-observer protocol (**unmarked** `multinomPois` with a double-observer `pi`-function). Site abundance $N \sim \text{Poisson}(\lambda)$ is surveyed by two observers with detection p_1 / p_2 . The state formula models $\log \lambda$; detection the shared site-level per-observer detection design (observers 1 and 2 carry separate coefficients). By Poisson-multinomial thinning the observable cell counts are independent Poissons, so the marginal is closed form with no latent-abundance summation.

Usage

```
double_observer(type = c("independent", "dependent"))
```

Arguments

type "independent" (default; $N \times 3$ cell counts) or "dependent" (removal-style, $N \times 2$ cell counts, needs primary = for identifiability).

Value

A `tobs_family` object for `tobs()`.

Independent vs dependent protocol

type = "independent" (the default): the two observers detect independently and each individual is recorded by observer 1 only, observer 2 only, or both, so y is an $N \times 3$ matrix of cell counts in that column order and the three cells are $\pi_i = (p_1(1 - p_2), (1 - p_1)p_2, p_1 p_2)$.

type = "dependent": a removal-style protocol where a *primary* observer records what it detects and a *secondary* observer records only what the primary missed, so y is an $N \times 2$ matrix (primary-detected, secondary-only) with cells $\pi_i = (p_{pri}, (1 - p_{pri})p_{sec})$. A single fixed primary observer gives only two cells for three parameters (λ, p_1, p_2), a ridge that does not separate the two observer detections; observer **role-swapping** identifies them – pass primary = (a length- N vector in $\{1, 2\}$ naming each site's primary observer), and with observer 1 primary at some sites and observer 2 at others (p_{pri}, p_{sec}) alternates between (p_1, p_2) and (p_2, p_1) , giving the four cell means that recover (λ, p_1, p_2) . With $p_1 = p_2$ (an interchangeable pair) the dependent protocol reduces to a two-pass `removal()` model.

Engines

- method = "laplace" – the closed-form Poisson-multinomial marginal maximised (BFGS) with an observed-information covariance; the only route (no field, no sampler).

Examples

```
sim <- simulate_double_observer(N = 200, seed = 1)
fit <- tobs(~ abund_cov1, data = sim$data, family = double_observer(),
           detection = ~ det_cov1, y = sim$y, control = list(verbose = FALSE))
coef(fit)

# Dependent (role-swapping) protocol:
sd <- simulate_double_observer(N = 300, type = "dependent", seed = 1)
fd <- tobs(~ abund_cov1, data = sd$data, family = double_observer("dependent"),
           detection = ~ det_cov1, y = sd$y, primary = sd$primary,
           control = list(verbose = FALSE))
coef(fd)
```

dyn_abun

*Open-population (Dail-Madsen) N-mixture family***Description**

Latent abundance evolves across primary seasons via apparent survival and recruitment (Dail & Madsen 2011): $N_1 \sim \text{Poisson}(\lambda)$; for $t \geq 2$, $N_t = S_t + G_t$ with survivors $S_t \sim \text{Binomial}(N_{t-1}, \omega)$ and recruits $G_t \sim \text{Poisson}(\gamma)$; observed via $\text{Binomial}(N_t, p)$ over secondary visits. The latent abundance sequence is summed out by an exact HMM forward recursion (it is not closed form, unlike the static `abun()`); analytic gradients come from forward-mode differentiation of the scaled forward algorithm, so the fit is a direct maximum-likelihood / Laplace fit with a NUTS path over the same marginal.

Usage

```
dyn_abun(K.max = NULL, mixture = c("poisson", "negbin", "zip", "zinb"))
```

Arguments

K.max	abundance-state truncation for the forward recursion (states $0 \dots K_{\text{max}}$). NULL (default) uses <code>max(count) + 40</code> ; raise it if abundance may exceed that (the forward cost is roughly cubic in K.max).
mixture	initial-abundance distribution: "poisson" (default), "negbin" (negative-binomial $N_1 \sim \text{NB}(\text{mean} = \lambda, \text{size} = r)$), or their zero-inflated counterparts "zip" / "zinb" (a structural-zero share ω of sites is never occupied across any season; the remaining sites follow the Dail-Madsen open-population process). Zero-inflation is non-spatial laplace with an intercept-only ω ; a field / RE / NUTS stay Poisson / negbin.

Details

Four arms: initial abundance λ (the `tobs()` formula) and detection p (detection) are site-level; apparent survival ω (the ω argument, default ~ 1) and recruitment γ (the γ argument, default ~ 1) span the $T - 1$ transition intervals. A constant ω / γ is shared across a site's seasons; supplying a season-varying covariate (a $[n_{\text{sites}} \times (T - 1)]$ matrix column of data, one column per transition interval) on ω / γ gives interval-specific vital rates. The response y is a 3D array $[n_{\text{sites}} \times \text{max_visits} \times n_{\text{seasons}}]$ (or a list of per-season count matrices); missing visits are NA.

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – analytic-gradient BFGS over the forward marginal with an observed-information covariance; a grouped random effect on the initial-abundance arm is integrated by adaptive Gauss-Hermite quadrature.

- `method = "nested_laplace"` – an areal `icar()` / `car_proper()` field on the initial-abundance arm (BFGS over the forward marginal plus the CAR prior), optionally composed with a `temporal()` field, through the shared areal-BFGS driver.
- `method = "nuts"` – the in-tree C++ NUTS sampler over the same forward marginal, with a fixed-hyper areal field block on the initial-abundance arm; a `temporal()` term is not sampled.

References

Dail, D., Madsen, L. (2011). Models for estimating abundance from repeated counts of an open metapopulation. *Biometrics* 67, 577-587.

Examples

```
f <- dyn_abun(mixture = "negbin")
f
```

dyn_abun_laplace	<i>Maximum-likelihood fit of the Dail-Madsen open N-mixture</i>
------------------	---

Description

Fits the Dail-Madsen (2011) open-population N-mixture (Poisson initial abundance, binomial survival, Poisson recruitment, binomial detection) by maximising the exact HMM forward marginal likelihood with an analytic gradient (BFGS). The observed-information covariance is the inverse of the negative finite-difference Jacobian of the analytic gradient at the mode. All four arms (initial abundance λ , detection p , survival ω , recruitment γ) are site-level.

Usage

```
dyn_abun_laplace(
  y_flat,
  n_sites,
  T,
  J,
  K_max,
  X_lambda,
  X_p,
  X_omega,
  X_gamma,
  mixture = "poisson",
  log_r_init = NULL,
  max_iter = 300L,
  tol = 1e-08,
  verbose = FALSE
)
```

Arguments

y_flat	Integer vector, flattened detection counts (site-major, then season, then visit; -1 marks a missing visit).
n_sites, T, J	Sites, primary seasons, secondary visits per season.
K_max	Abundance-state truncation (states 0..K_max).
X_lambda, X_p, X_omega, X_gamma	Numeric [n_sites x p_arm] design matrices.
mixture	Initial-abundance distribution: "poisson" (default) or "negbin" (negative-binomial).
log_r_init	Optional starting value (log scale) for the negative-binomial size parameter r; used only when mixture = "negbin". Default NULL starts from log(2).
max_iter	BFGS iteration budget (default 300).
tol	Convergence tolerance (optim reltol, default 1e-8).
verbose	Print convergence status.

Value

A list of class `dyn_abun_fit` with `beta_lambda`, `beta_p`, `beta_omega`, `beta_gamma`, `log_lik`, `vcov`, `H_obs`, per-site `mean_N1`, and `converged`.

References

Dail, D., Madsen, L. (2011). Models for estimating abundance from repeated counts of an open metapopulation. *Biometrics* 67, 577-587.

dyn_int_occu	<i>Multi-season integrated occupancy family</i>
--------------	---

Description

Dynamic (multi-season) occupancy observed by several detection sources (`spOccupancy tIntPGOcc`): the product of a dynamic occupancy HMM (season-1 occupancy `psi1`, colonization `gamma`, extinction `eps`) and integrated occupancy (a per-season emission that pools multiple detection sources). Pooling sources across seasons is how colonization / extinction estimates come out of individually-sparse opportunistic data. The latent occupancy sequence integrates out by the two-state HMM forward recursion; the exact marginal is maximised with an observed-information `vcov`.

Usage

```
dyn_int_occu()
```

Details

y is a length-S list of [sites x visits x seasons] detection arrays (0/1/NA). The state formula models `logit psi1; colonization = ~ ...` and `extinction = ~ ...` the site-level transitions (required, as in `dyn_occu()`); detection the shared per-source detection design (each source carries its own coefficients). `v1`: every source covers all sites and the same season grid, constant transitions, site-level detection.

Value

A `tobs_family` object for `tobs()`.

Engines

- `method = "laplace"` – the non-spatial fit: the HMM-forward marginal maximised with analytic forward-backward gradients and an observed-information covariance.
- `method = "nested_laplace"` – a shared `icar()` field on the first-season occupancy formula (`spOccupancy stIntPGOcc`) through the shared areal-BFGS driver, the field hyperparameters integrated on an outer grid; that field is the only structured term accepted and is required on this route.

Examples

```
sim <- simulate_dyn_int_occu(N = 200, T.seasons = 4, S = 2, seed = 1)
fit <- tobs(~ 1, data = sim$data, family = dyn_int_occu(),
           detection = ~ 1, colonization = ~ 1, extinction = ~ 1,
           y = sim$y, sources = sim$sources, control = list(verbose = FALSE))
coef(fit)
```

dyn_occu

Dynamic (multi-season) occupancy family

Description

Bernoulli occupancy state with colonisation + extinction transitions across seasons (the MacKenzie et al. dynamic model). Colonisation / extinction are constant across a site's seasons by default; a season-varying rate is supplied as a $[n_sites \times (T - 1)]$ matrix column of data (one column per transition interval) on colonization / extinction. Detection is site-level by default; a $[n_sites \times T]$ matrix column of data (one column per primary season) on the detection formula gives per-season detection. All season-varying paths run under `method = "laplace"` (the exact HMM-forward marginal refine calibrates the coefficients); they are gated under "nuts" with a pointer.

Usage

```
dyn_occu()
```

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – EM + Laplace over the exact HMM-forward marginal, the default and the route every season-varying path runs on.
- `method = "laplace_sla"` – the Laplace fit with simplified-Laplace skew-corrected marginals.
- `method = "laplace_gibbs" / method = "laplace_mi"` – the Laplace fit with a post-EM Gibbs / multiple-imputation variance correction.
- `method = "nested_laplace"` – an areal field on the occupancy arm through the nested-Laplace EM, its hyperparameters integrated on an outer grid.
- `method = "nuts"` – the unified C++ NUTS sampler over the HMM marginal; season-varying colonisation / extinction / detection error with a pointer.

Examples

```
f <- dyn_occu()
f
```

fitted.tobs_fit	<i>Fitted values (occupancy and detection probabilities)</i>
-----------------	--

Description

Fitted values (occupancy and detection probabilities)

Usage

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

Arguments

<code>object</code>	A <code>tobs_fit</code> object.
<code>...</code>	Ignored.

Value

A list with `psi` (occupancy probabilities), `p` (detection probabilities), and `z` (posterior state probability at posterior-mean parameters). For an `int_occu()` fit `p` is a named list, one detection probability vector per source (matching `y`'s names), since each source carries its own detection design and, when present, its own detection-arm spatial field. For single-season, `int_occu()` and community models `z` is the per-site Bayes posterior $P(z=1 \mid y)$ (an `int_occu()` site is occupied with certainty if ANY source detected it, and otherwise pools every source's non-detection evidence); for dynamic models `z` is the forward-backward (HMM smoothing) state posterior $P(z_{t=1} \mid y_{\{1:T\}})$ as an $[n_sites \times n_seasons]$ matrix.

foray_counts	<i>Repeated point-count abundance survey (synthetic)</i>
--------------	--

Description

A binomial N-mixture dataset (Royle 2004) of repeated counts at 100 sites over 3 visits, drawn from a known generative model. Latent abundance increases with shrub cover and decreases with elevation; per-visit detection increases with survey effort. Synthetic, not field data; the generative coefficients and the realised latent abundances are in truth.

Usage

```
foray_counts
```

Format

A list with components:

y 100 x 3 integer matrix of counts.

occ.covs Data frame of site covariates: elevation, shrub (both standardised).

det.covs Named list with one 100 x 3 visit covariate matrix, effort (standardised).

truth List of generative quantities: beta_lambda, beta_p, the realised latent abundance N, and per-site lambda.

See Also

[abun\(\)](#), [tobs\(\)](#), [simulate_abun\(\)](#)

Examples

```
data(foray_counts)
summary(as.vector(foray_counts$y))

fit <- tobs(~ elevation + shrub, data = foray_counts$occ.covs,
           family = abun(), detection = ~ effort,
           y = foray_counts$y, visits = foray_counts$det.covs)
coef(fit)
```

fp_occu

*Multistate false-positive occupancy family***Description**

Occupancy with both false negatives and false positives in the detection process (e.g. acoustic classifiers, citizen-science misidentification), using the Miller et al. (2011) confirmed-detection design that makes the model robustly identifiable. Each visit yields a detection state y in $\{0, 1, 2\}$: 0 = no detection, 1 = ambiguous detection (a true detection OR a false positive), 2 = certain / confirmed detection (only possible when the site is truly occupied). The latent occupancy z marginalises in closed form (two states), so the fit maximises the exact marginal likelihood directly (analytic-gradient BFGS, observed-information covariance) with a NUTS path over the same marginal.

Usage

fp_occu()

Details

Four site-level logit arms: occupancy ψ (the `tobs()` formula), true detection $p11$ (detection), false-positive rate $p10$, and the probability a true detection is certain b . The $p10$ and b predictors default to intercept-only and are set with the $p10 = \sim \dots$ and $certainty = \sim \dots$ arguments to `tobs()` ($certainty$ is the b arm). The response y is an $n_sites \times J$ integer matrix in $\{0, 1, 2\}$ (NA visits dropped).

ValueA `tobs_family` object.**Engines**

- `method = "laplace"` – analytic-gradient BFGS over the exact marginal with an observed-information covariance; a grouped random effect on the ψ or $p11$ arm is integrated by adaptive Gauss-Hermite quadrature.
- `method = "nested_laplace"` – an areal `icar()` / `car_proper()` field on the occupancy arm (BFGS over the marginal plus the CAR prior), optionally composed with a `temporal()` field, through the shared areal-BFGS driver.
- `method = "nuts"` – the in-tree C++ NUTS sampler over the same marginal, with a fixed-hyper areal or temporal field block on the occupancy arm (one of the two, not both).

References

Miller, D. A. W., Nichols, J. D., McClintock, B. T., Grant, E. H. C., Bailey, L. L., Weir, L. A. (2011). Improving occupancy estimation when two types of observational error occur. *Ecology* 92, 1422-1428. Royle, J. A., Link, W. A. (2006). Generalized site occupancy models allowing for false positive and false negative errors. *Ecology* 87, 835-841.

Examples

```
f <- fp_occu()
f
```

fp_occu_laplace	<i>Maximum-likelihood fit of the multistate false-positive occupancy model</i>
-----------------	--

Description

Fits the Miller et al. (2011) false-positive occupancy model with confirmed detections (y in $\{0, 1, 2\}$) by maximising the exact two-state marginal likelihood with an analytic gradient (BFGS). The observed-information covariance is the inverse of the negative finite-difference Jacobian of the analytic gradient at the mode. All four arms (occupancy ψ , true detection p_{11} , false-positive p_{10} , certain-classification b) are site-level logit predictors.

Usage

```
fp_occu_laplace(
  y,
  site_idx,
  X_psi,
  X_p11,
  X_p10,
  X_b,
  sigma_beta = NULL,
  max_iter = 200L,
  tol = 1e-08,
  verbose = FALSE
)
```

Arguments

<code>y</code>	Integer vector of detection states (0/1/2), long form (valid visits only).
<code>site_idx</code>	Integer vector, same length as <code>y</code> , 1-based site index.
<code>X_psi, X_p11, X_p10, X_b</code>	Numeric [<code>n_sites</code> x <code>p_arm</code>] design matrices for the four arms.
<code>sigma_beta</code>	Optional Gaussian prior SD on the coefficients (a mild ridge for stability); NULL (default) is the unpenalised MLE.
<code>max_iter</code>	BFGS iteration budget (default 200).
<code>tol</code>	Convergence tolerance (<code>optim reltol</code> , default $1e-8$).
<code>verbose</code>	Print convergence status.

Value

A list of class `fp_occu_fit` with `beta_psi`, `beta_p11`, `beta_p10`, `beta_b`, `log_lik`, `vcov`, `H_obs`, per-site posterior occupancy `w1`, and `converged`.

References

Miller, D. A. W., et al. (2011). Improving occupancy estimation when two types of observational error occur. *Ecology* 92, 1422-1428.

gdistremoval	<i>Joint distance + removal sampling family</i>
--------------	---

Description

The unmarked gdistremoval model (Amundson et al. 2014): a **single-season** point-count design in which the detected individuals are recorded two ways at once – their distance band (distance sampling) and the removal period of first detection (removal sampling). This is NOT the open-population distance model (distsampOpen); the abundance is static.

Usage

```
gdistremoval(transect = c("line", "point"), cutpoints = NULL)
```

Arguments

transect	Transect geometry: "line" (default) or "point".
cutpoints	Distance-bin edges, length ncol(y) + 1, strictly increasing and starting at >= 0.

Details

Site abundance $N_i \sim \text{Poisson}(\lambda_i)$; the detected birds are cross-classified by a distance band and a removal period. Writing p_{dist_i} for the overall distance detection, p_{rem_i} for the overall removal detection, the total detected is a binomial thinning of N_i , and Poisson is closed under binomial thinning, so the marginal is closed-form:

$$y_{\text{sum}_i} \sim \text{Poisson}(\lambda_i p_i^{\text{dist}} p_i^{\text{rem}}),$$

with the distance-band counts and removal-period counts as two conditional multinomials – the [double_observer\(\)](#) Poisson-multinomial pattern, here with a distance multinomial (half-normal key band integrals) and a depleting-removal multinomial $\pi_k = r (1 - r)^{k-1}$.

Three site-level arms: log abundance λ (the [tobs\(\)](#) formula), log distance scale σ (detection), and logit per-period removal capture r (removal = ~ ..., default intercept-only).

Value

A `tobs_family` object.

Inputs

`y` is an `n_sites` x `n_bins` integer matrix of per-distance-band counts; `y.rem` an `n_sites` x `n_periods` integer matrix of per-removal-period counts. The per-site row totals must match (the same detected birds cross-classified).

Engines

- method = "laplace" – the closed-form marginal maximised (BFGS) with an observed-information covariance; the only route (no field, no sampler).

References

Amundson, C. L., Royle, J. A., Handel, C. M. (2014). A hierarchical model combining distance sampling and time removal to estimate detection probability during avian point counts. *The Auk* 131, 476-494.

See Also

[distance\(\)](#), [removal\(\)](#), [double_observer\(\)](#).

Examples

```
f <- gdistremoval(transect = "point", cutpoints = c(0, 10, 20, 30, 40))
f
```

glance.tobs_fit	<i>One-row model summary for a tobs_fit</i>
-----------------	---

Description

Every family, and every method within a family, glances to one column set: nobs, df, logLik, n_fixed, n_samples, n_divergent, mean_accept and converged, the sampler columns NA on a fit with no draws. The joint nested-Laplace fits add the outer grid placement and, when requested, the outer Pareto-k diagnostic. The joint-coupled families (occu_cover(), cover(), occu() spatial, occu_multiscale_cover()) carry both at the fit top level.

Usage

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

Arguments

x	A fitted tobs_fit.
...	Ignored.

Value

The one-row glance data frame. A joint-coupled fit adds two outer-grid placement columns, reported whether or not the grid moved so an inert auto-recenter is visible in a batch summary:

outer_grid_placement "fixed" (the grid the fit was given) or "auto_recentered" (the engine moved it onto the hyperparameter mode).

outer_grid_recenter_declined On a "fixed" placement, why the recenter did not apply – for instance a user-pinned axis, or "auto_recenter_disabled". NA when the grid was recentered.

A fit that also requested the Pareto-k diagnostic (control\$diagnose.k = TRUE, off by default) adds three more:

pareto_k The outer importance-sampling $\hat{k}$ for the hyperparameter Gaussian summary; < 0.7 indicates a reliable summary, NA when the diagnostic did not run or the proposal was degenerate.

pareto_k_is_ess The importance-sampling effective sample size on the PSIS-smoothed weights (numeric); pareto_k_is_ess / control\$k.samples is the relative IS efficiency.

pareto_k_proposal_source How the importance proposal was built: "mode_hessian" from the Laplace curvature at the hyperparameter mode – curvature-backed, so the $\hat{k}$ stays trustworthy even when a sharp posterior collapses the integration grid to ~ 1 cell; "grid_moment" from the grid-weighted covariance of the integration nodes (with "moment_matched" its refinement); or "grid_mixture", the local-bump-per-cell mixture matching what the engine samples on a spread grid.

int_occu

Integrated occupancy family

Description

Multiple data sources informing a shared latent occupancy state, each with its own detection process.

Usage

```
int_occu()
```

Details

A continuous-mesh spde() term on the detection formula loads a spatially-structured detection field, broadcast to every source, under method = "laplace". The per-source fields are reported as a named list in fit\$spatial_field_det. All sources must share one detection formula; areal terms belong on the state arm.

Value

A tobs_family object.

Engines

- method = "laplace" – EM + Laplace over the multi-source two-state marginal, the default.
- method = "laplace_sla" – the Laplace fit with simplified-Laplace skew-corrected marginals.
- method = "laplace_gibbs" / method = "laplace_mi" – the Laplace fit with a post-EM Gibbs / multiple-imputation variance correction.

- `method = "nested_laplace"` – an areal field on the state arm through the nested-Laplace EM, its hyperparameters integrated on an outer grid.
- `method = "nuts"` – the unified C++ NUTS sampler over the same marginal.

Examples

```
f <- int_occu()
f
```

jsdm

Joint species distribution family (no detection process)

Description

Multivariate occurrence with shared latent factors. No observation replication – treats observed presence/absence as the response.

Usage

```
jsdm()
```

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – the community Laplace-EM shared with `ms_count()` (a logit-link community GLMM); `latent()` factors alone (lfJSDM) run here through the block-coordinate driver.
- `method = "nuts"` – samples the exact joint community posterior (community means, per-species deviations, community covariance) over the Bernoulli response, warm-started at the Laplace-EM mode; non-spatial, an areal or `latent()` term errors with a pointer.
- `method = "pg_gibbs"` – a per-species Polya-Gamma conjugate Gibbs with community mean and Inverse-Gamma variance updates, giving a calibrated community-variance posterior; non-spatial.
- `method = "nested_laplace"` – a shared areal field (`icar()` / `car_proper()` / `bym2()`), with or without `latent()` factors (sfJSDM), by block coordinate ascent; the areal term is required on this route.

Examples

```
f <- jsdm()
f
```

<code>logLik.tobs_fit</code>	<i>Log-likelihood of a tobs_fit</i>
------------------------------	-------------------------------------

Description

The value and degrees of freedom are the engine's (the `tulpa_fit` method); the "nobs" attribute is `nobs.tobs_fit()` on the same fit, so `stats::BIC()` and anything else reading the attribute counts the observations `nobs()` reports.

Usage

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

Arguments

<code>object</code>	A <code>tobs_fit</code> object.
<code>...</code>	Passed on to the next method.

Value

A `logLik` object.

<code>meadow_cover</code>	<i>Grassland vegetation-cover panel (synthetic)</i>
---------------------------	---

Description

A vegetation cover-hurdle dataset for 150 grassland plots surveyed across three years, drawn from a known generative model: the species occurs with a probability driven by moisture, and where it occurs, its conditional cover (a Beta-distributed proportion in $(0, 1)$) increases with moisture and decreases with grazing. Roughly half the plots record zero cover. `year_c` is the centred survey year, so `within_between()` can split the cross-plot baseline from the within-plot temporal deviation. Synthetic, not field data; the generative coefficients are attached as `attr(meadow_cover, "truth")`.

Usage

```
meadow_cover
```

Format

A data frame with 150 rows (one per plot) and columns:

plot Plot identifier (factor).

year Survey year (2018-2020).

year_c Survey year centred on its mean.

moisture Standardised soil-moisture index.

grazing Standardised grazing-pressure index.

cover Recorded cover as a proportion in $[\emptyset, 1]$ (0 where absent).

See Also

[cover\(\)](#), [tobs\(\)](#), [within_between\(\)](#), [simulate_cover\(\)](#)

Examples

```
data(meadow_cover)
mean(meadow_cover$cover > 0)

fit <- tobs(cover ~ moisture, data = meadow_cover,
            family = cover(response = "beta"))
coef(fit)
```

ms_abun

Multispecies N-mixture family

Description

Per-species N-mixture with shared community-level hyperparameters.

Usage

```
ms_abun(K.max = NULL, mixture = c("poisson", "negbin", "zip", "zinb"))
```

Arguments

K.max	upper bound for the latent-abundance marginal sum (the exact integration over N is truncated at K.max). NULL (default) lets the engine pick $\max(y) + 100$ (matching <code>unmarked::pcount()</code>); raise it if a fit warns that the posterior over N puts mass on the boundary.
mixture	Abundance mixing distribution: "poisson" (default), "negbin", or their zero-inflated counterparts "zip" / "zinb". The negative-binomial dispersion and the zero-inflation structural-zero share are each a per-species random effect ($\log_r_s \sim N(\mu_log_r, \sigma_log_r)$, $\logit_omega_s \sim N(\mu_omega, \sigma_omega)$) integrated by the community AGHQ path alongside the abundance / detection coefficients. Zero-inflation is a non-spatial laplace fit with an intercept-only structural-zero logit; a shared field stays Poisson / negbin.

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – the in-tree C++ community Laplace-EM (per-species coefficient random effects, closed-form community covariance M-step, AGHQ variance debias), the default; `latent()` factors run here through the block-coordinate driver.
- `method = "nested_laplace"` – a shared areal field (`icar()` / `bym2()` / `car_proper()`) on the abundance arm (the `spAbundance sfmsNMix` model), Poisson or grid-integrated negative-binomial size; with `latent()` factors beside the field the spatial-factor model by block coordinate ascent.
- `method = "nuts"` – the non-spatial community sampler over the closed-form per-(species, site) marginal, sampling the community means, per-species deviations and community covariances jointly (Poisson, or per-species negative-binomial `log_r_s`), warm-started at the Laplace-EM mode; with a shared `car_proper()` field (Poisson) a fixed-hyper non-centered field joins the sampler, while `icar()` / `bym2()`, `temporal()` and `re()` terms stay on "nested_laplace".

Reading the negbin community dispersion

Under `mixture = "negbin"` the community mean `mu_log_r` reaches `coef()` / `vcov()` / `confint()` with a marginal Wald SE, and the variance component `sigma_log_r` is reported on `fit$ms_dispersion`. That interval is calibrated **conditional on the variance component being recovered**, and the two are not independent: `sigma_log_r` is a scalar variance over species and at few species it can settle near its lower boundary, which shifts `mu_log_r` and narrows its SE at the same time.

Measured on simulated data (39 laplace fits at 8 and 36 species, `sigma.logr = 0.5`): where `sigma_log_r` came back at least 0.30, the nominal 95% interval covered 33 of 34 with a `sqrt(mean(z^2))` of 0.88; where it came back below, it covered 2 of 5, the point estimate was 2.2x further from the truth and the SE 28% narrower. So read `fit$ms_dispersion$sigma_log_r` before quoting a `mu_log_r` interval: a value near zero, well under what the data should support, means the interval is not trustworthy even though the fit converged and reported no warning. Those thresholds are the ones this fixture separated on and are not a general rule; what transfers is the conditioning, not the number.

Away from that boundary the interval is calibrated, and calibrated at every group count measured. Over 97 fits at 8 / 18 / 36 species the scale `sd(mu_log_r) / mean(se)` reads 1.066 / 1.265 / 0.981, and the middle figure is the seed block rather than the estimator: `mu_log_r` is a POPULATION mean, each seed draws `S` log-dispersions around it, and that draw supplies about two thirds of the across-seed spread. The 18-species blocks drew theirs 18-21% wider than `sigma.logr / sqrt(S)`. Put the draw at its expectation and the scale is 1.077 / 1.101 / 0.977; rebuild the SE at the simulated `sigma` as well and it is 0.990 / 1.035 / 0.963 (NOTES_measurements.md). The residual there is the attenuation of `sigma_log_r` itself (0.418 / 0.448 / 0.487 against 0.5), which shrinks as species are added and which the SE inherits, since $se^2 = (\sigma_{log_r}^2 + c) / S$.

A simulation study measuring this needs `simulate_ms_abun()`'s `truth$mu_log_r_real` – the mean of the log-dispersions that seed actually drew – beside `truth$mu_log_r`. Score coverage of a community mean against the constant; score point recovery and interval SCALE against the realized mean, or the draw is charged to the estimator.

control\$logr.sigma.prior puts a Penalized-Complexity prior on that variance, which adds curvature at $\sigma_{\log_r} \rightarrow 0$. Measured on the same fixture at $c(1, 0.05)$, it is worth reaching for when a fit reports a $\sigma_{\log_r}$ near zero or fails outright with a singular marginal Hessian – one seed that errored under pure maximum likelihood converged and covered under the prior, and variance components near 0.01-0.06 lifted by an order of magnitude. It does **not** repair coverage: on 19 paired seeds the nominal 95% interval covered 17 either way, with the same two seeds missing, because the prior bites near the boundary while those misses sit at $\sigma_{\log_r}$ around 0.2. It is off by default because the fits that were already calibrated pick up a small systematic shift under it ($\mu_{\log_r}$ by -0.006, $p = 0.001$).

Examples

```
f <- ms_abun(mixture = "negbin")
f
```

ms_count

Multispecies (community) relative-abundance GLMM family

Description

The community analogue of `count()`: a per-species GLMM on an observed count or continuous response with Gaussian community hyperpriors on the per-species coefficients (the `spAbundance` `msAbund` model). No detection and no latent state – the abundance counterpart of `ms_occu()` without the occupancy layer. Poisson / negative-binomial (log link) or Gaussian (identity). `y` is an $n_{\text{sites}} \times n_{\text{species}}$ matrix (or a named list of n_{species} count vectors), one value per site per species; species names the columns.

Usage

```
ms_count(response = c("poisson", "negbin", "gaussian", "binomial"))
```

Arguments

response	One of "poisson", "negbin", "gaussian", or "binomial". The binomial response is the community k-of-n GLMM (community <code>svcPGBinom</code>): supply the per-site (or per-site $\times$ species) trial count as <code>trials = on tobs()</code> (default 1, i.e. Bernoulli, which is the <code>jsdm()</code> response). With <code>trials > 1</code> the community-mean intercept carries a small first-order-Laplace bias of order $1 / n_{\text{species}}$ (a few hundredths on the logit scale at 20 species, shrinking with more species; the slope and the <code>trials = 1</code> case are unbiased) – the same character as the negative-binomial slope attenuation noted above.
----------	--

Details

As with the other community Laplace-EM families (`ms_occu()`, `ms_occu_cover()`), the community means are recovered essentially unbiased for the Gaussian (identity link, exact Laplace) and Poisson responses; the negative-binomial slope carries a mild first-order-Laplace (PQL) attenuation

of order the community variance (a few percent, shrinking with the number of species and observations per species). The community-mean Wald intervals are calibrated to the package rubric (pooled coverage at least 0.85).

method = "nuts" samples the exact joint posterior (community means, per-species deviations, and the community covariance) for all three responses – the negative binomial carrying a per-species dispersion random effect, the Gaussian a per-species free residual variance – which removes the Laplace-EM's negative-binomial attenuation and returns calibrated, non-Gaussian community intervals. A shared areal field or latent factors are available through method = "nested_laplace" / the latent() term; see the package overview for the spatial and factor variants.

Value

A tobs_family object.

Engines

- method = "laplace" – the shared community Laplace-EM, the default; latent() factors alone run here through the block-coordinate driver.
- method = "nested_laplace" – a shared areal field icar() on the abundance formula (the spAbundance sFMsAbund model), with or without latent() factors, by block coordinate ascent; the field is required on this route.
- method = "nuts" – the exact joint community posterior sampled by the in-tree C++ target, warm-started at the Laplace-EM mode, as described above; non-spatial.
- method = "pg_gibbs" – a per-species Polya-Gamma conjugate Gibbs with community mean and Inverse-Gamma variance updates, for the logistic "binomial" response only (another response errors with a pointer); non-spatial.

See Also

[count\(\)](#) (single species), [ms_occu\(\)](#), [ms_abun\(\)](#)

Examples

```
sim <- simulate_ms_count(N = 120, n.species = 8, seed = 1)
fit <- tobs(~ x, data = sim$data, family = ms_count(), y = sim$y,
           species = colnames(sim$y), method = "laplace")
summary(fit)
```

ms_distance

Community (multispecies) binned distance-sampling family

Description

The community analogue of [distance\(\)](#): per-species binned distance sampling with Gaussian community hyperpriors on the per-species abundance and detection-scale coefficients, so rare species borrow strength from common ones through the shared community means and covariances – the same pooling that stabilises [ms_abun\(\)](#), on the distance-sampling response.

Usage

```
ms_distance(
  key = c("halfnorm", "hazard"),
  transect = c("line", "point"),
  cutpoints = NULL,
  K.max = NULL,
  mixture = c("poisson", "negbin")
)
```

Arguments

key	Detection function: "halfnorm" (default) or "hazard". Under the hazard-rate key the scalar log-shape is shared across species.
transect	Transect geometry: "line" (default) or "point".
cutpoints	Distance-bin edges, length $\dim(y)[2] + 1$, strictly increasing and starting at ≥ 0 .
K.max	Truncation for the latent abundance sum. Defaults to $3 * \max(\text{rowSums}(y)) + 100$.
mixture	Abundance mixing distribution. "poisson" only; the negative-binomial size is not yet carried as a per-species random effect.

Details

Per species s , site i , distance bin b :

$$\begin{aligned}
 N_{\{s,i\}} &\sim \text{Poisson}(\lambda_{\{s,i\}}) \\
 y_{\{s,i,\cdot\}} \mid N &\sim \text{Multinomial}(N_{\{s,i\}}; \pi_{\{s,i,1..B\}}, 1 - p_{\text{det}}) \\
 \log \lambda_{\{s,i\}} &= X_i \cdot (\mu_{\lambda} + b_{\lambda_s}) \\
 \log \sigma_{\{s,i\}} &= X_{\text{sig}_i} \cdot (\mu_{\sigma} + b_{\sigma_s}) \\
 b_{\lambda_s} &\sim N(0, \text{Sigma}_{\lambda}), \quad b_{\sigma_s} \sim N(0, \text{Sigma}_{\sigma})
 \end{aligned}$$

with π_b the integral of the detection function over bin b . The latent $N_{\{s,i\}}$ integrates out per species-site in closed form, exactly as for [distance\(\)](#); the per-species coefficient deviations are the random effects, integrated by the shared community Laplace-EM.

Adding latent(n) to the abundance formula gives residual species co-occurrence through n per-site latent factors with per-species loadings; adding an areal term (`icar()` / `car_proper()` / `bym2()`) or `spde()` alongside gives a shared spatial field. Both route through the same block-coordinate engine as the other community families.

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – the shared community Laplace-EM over the existing distance kernel, plain or with `latent()` factors alone (lfMsDS).
- `method = "nested_laplace"` – a shared field (`icar()` / `car_proper()` / `bym2()` / `spde()`) on the abundance formula, with or without factors (sfMsDS), by block coordinate ascent; the field is required on this route.

Poisson only, and no sampler route.

Inputs

`y` is a 3D array [`n_sites` x `n_bins` x `n_species`] or a named list of `n_sites` x `n_bins` per-bin count matrices. `formula` is the abundance (log λ) predictor, detection the detection-scale (log σ) predictor, and `species` the species labels.

See Also

[distance\(\)](#) (single species), [ms_abun\(\)](#) (community N-mixture).

Examples

```
f <- ms_distance(key = "halfnorm", transect = "line",
                 cutpoints = c(0, 25, 50, 75, 100))
f
```

ms_dyn_occu

Community (multispecies) dynamic occupancy family

Description

Per-species dynamic (HMM) occupancy with Gaussian community hyperpriors on the per-species first-season occupancy and detection coefficients, and shared community-wide colonisation / extinction transition coefficients.

Usage

```
ms_dyn_occu()
```

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – the shared community Laplace-EM, the default: the shared colonisation / extinction dynamics and the per-species first-season occupancy / detection components recover across seeds (see `tests/testthat/test-ms-dyn-occu.R`, community-mean 95% CI coverage measured ~0.98).
- `method = "pg_gibbs"` – the community Polya-Gamma machinery of `msPGOcc` with a two-state HMM forward-filtering backward-sampling latent step (`spOccupancy` `tMsPGOcc`), giving a calibrated community-variance posterior; constant transitions, site-level detection, no structured term.
- `method = "nested_laplace"` – a shared areal field on the first-season occupancy formula ($\sim 1 + \text{icar}(\text{graph} = \text{adj})$) fits the `spOccupancy` `stMsPGOcc` model: the field is shared across species, and because the first-season occupancy `psi1` only sets the initial mixing weight of each species' HMM, the block-coordinate driver alternates the community EM (field as a `psi1` offset) with an areal field Newton, the field recovering cleanly (`cor` ~0.94). A spatially-varying-coefficient bar ($\sim \text{spatial}(\sim 1 + w \mid \mid \text{cell}, \text{graph} = \text{adj})$) adds a shared covariate-weighted field alongside the intercept field, fitting the `svcTMsPGOcc` model through the same K-field weighted-ICAR solve as the community count SVC (both fields recover, `cor` ~0.90 / ~0.89). `icar()` only; `bym2()` / `car_proper()` fields are a follow-up. The field is required on this route.
- `method = "nuts"` – the non-spatial community sampler over the exact per-species HMM-forward marginal (in-tree C++), sampling the community means, per-species first-season / detection deviations, both per-arm community covariances and the shared colonisation / extinction coefficients jointly, warm-started at the Laplace-EM mode; an areal term errors with a pointer to "nested_laplace".

See Also

[dyn_occu\(\)](#), [ms_occu\(\)](#)

Examples

```
f <- ms_dyn_occu()
f
```

ms_int_occu

Community (multispecies) integrated occupancy family

Description

Multiple detection data sources share one latent occupancy state per species, with Gaussian community hyperpriors on the per-species coefficients of the occupancy arm and of every per-source detection arm. Fit by a shared community Laplace-EM.

Usage

```
ms_int_occu()
```

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – the shared community Laplace-EM, the default: the shared occupancy mean and the per-source detection community components recover near nominal across seeds and more than one source (see `tests/testthat/test-ms-int-occu.R`). The binary community-mean intervals carry the mild Laplace under-dispersion typical of occupancy data (measured 95% CI coverage ~ 0.89 at small per-species n).
- `method = "pg_gibbs"` – the community integrated Polya-Gamma Gibbs sampler (one latent state per species-site observed by every source), giving a calibrated community-variance posterior; site-level per-source detection.
- `method = "nuts"` – the community integrated sampler over the exact multi-source two-state per-(species, site) marginal (in-tree C++), sampling the community means, per-species deviations and the $D + 1$ per-arm community covariances jointly, warm-started at the Laplace-EM mode.

No route takes a structured term: a field or random effect on either formula errors with a pointer, and there is no nested-Laplace path for this family.

See Also

`int_occu()`, `ms_occu()`

Examples

```
f <- ms_int_occu()
f
```

ms_occu

Multispecies (community) single-season occupancy family

Description

Per-species occupancy and detection with Gaussian community hyperpriors on the per-species coefficients of each arm (the `spOccupancy msPGOcc` model). The occupancy and detection per-species deviations are independent, each with its own community covariance, fit by a community Laplace-EM.

Usage

```
ms_occu()
```

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – the shared community Laplace-EM, the default; `latent()` factors alone (`lfMsPGOcc`) run here through the block-coordinate driver.
- `method = "nuts"` – the non-spatial community sampler over the closed-form per-(species, site) two-state marginal, sampling the community means, per-species deviations and both per-arm community covariances jointly, warm-started at the Laplace-EM mode; an areal term errors with a pointer to `"nested_laplace"`.
- `method = "pg_gibbs"` – the hierarchical Polya-Gamma Gibbs sampler (`spOccupancy msPGOcc`), giving a calibrated community-variance posterior; non-spatial.
- `method = "nested_laplace"` – a shared areal field (`icar()` / `bym2()` / `car_proper()`) on the occupancy arm: a plain intercept field runs the in-tree C++ community-spatial nested Laplace-EM, a varying-coefficient bar (`svcMsPGOcc`) or `latent()` factors beside the field (`sfMsPGOcc`) run the block-coordinate fitter; the field is required on this route.

See Also

`occu()`, `ms_dyn_occu()`, `ms_int_occu()`

Examples

```
sim <- simulate_ms_occu(N = 80, J = 4, n.species = 8, seed = 1)
fit <- tobs(~ x, data = sim$data, family = ms_occu(), detection = ~ 1,
           y = sim$y, species = paste0("sp", seq_len(8)), method = "laplace")
summary(fit)
```

ms_occu_cover

Community (multispecies) joint occupancy-detection + cover family

Description

The community version of `occu_cover()`: a per-species joint occupancy-cover model with Gaussian community hyperpriors on the per-species coefficients of all three arms (occupancy `psi`, detection `p`, and positive cover). Rare species borrow strength from common ones through the shared community means and covariances, the same pooling that stabilises `ms_occu()` and `ms_abun()` but on the joint occupancy + vegetation-cover response.

Usage

```
ms_occu_cover(response = c("beta", "lognormal", "gaussian"))
```

Arguments

<code>response</code>	likelihood for the positive cover arm. <code>"beta"</code> (cover in (0, 1)), <code>"lognormal"</code> (log-cover Gaussian), or <code>"gaussian"</code> (an identity-link Gaussian magnitude, the delta-normal hurdle; for a pre-transformed / unbounded positive response, not raw cover fractions).
-----------------------	---

Details

Per species s , cell i , visit j :

```

z_{s,i} ~ Bernoulli(psi_{s,i})
y_{s,i,j} | z ~ Bernoulli(p_{s,i,j})
c_{s,i,j} | y ~ f_pos(eta_pos_{s,i,j}, disp)
logit psi_{s,i} = X_occ_i . (mu_occ + b_occ_s)
logit p_{s,i,j} = X_p_{ij} . (mu_p + b_p_s)
g(cover) = X_pos_{ij} . (mu_pos + b_pos_s)
b_occ_s ~ N(0, Sigma_occ), b_p_s ~ N(0, Sigma_p),
b_pos_s ~ N(0, Sigma_pos)

```

The latent presence z integrates out per species-cell in closed form (the same two-state mixture as `occu_cover()`); the per-species coefficient deviations are the random effects, integrated by a Laplace-EM. The positive-arm dispersion is a shared community parameter.

Value

A `tobs_family` object.

Inputs

`y` and `y.pos` are 3D arrays [`n_sites` x `max_visits` x `n_species`] (or named lists of `n_sites` x `max_visits` matrices, one per species); `species` = is required. The occupancy formula, the detection formula, and the cover positive formula carry community covariates shared across species. `coef()` returns the community means; `ranef()` the per-species coefficient deviations.

Engines

- `method = "laplace"` – the community Laplace-EM (per-species deviations integrated, closed-form community covariance M-step), non-spatial or with the reduced-rank spatial factors below.
- `method = "nuts"` – the community sampler over the exact per-(species, cell) two-state marginal, warm-started at the Laplace-EM mode: non-spatial it samples the community means, per-species deviations, the three per-arm community covariances and the shared log-dispersion jointly; with a shared areal field it samples the spatial-factor model below.

Reduced-rank spatial factors

A single areal field term (`icar(graph = adj)`, `car_proper(graph = adj)`, or `bym2(graph = adj)`) on the occupancy formula fits the reduced-rank (HMSC / `spatial-gllvm`) community model: K shared latent fields with per-species loadings on the occupancy predictor, $\text{logit } \psi_{sc} = X_c (\mu_{\text{occ}} + b_{\text{occ}_s}) + \sum$. A single shared field with only a species intercept can shift each map's level but not its shape; the loadings give each species its own spatial shape as a combination of the shared factors, borrowing strength so rare taxa get a calibrated map. The number of factors is set by `control = list(n.factors = K)`, or chosen automatically by `control = list(n.factors = "auto", n.factors.max = M)`, which fits the identified (lower-triangular loading) ladder and picks the rank that maximises the empirical-Bayes Laplace marginal likelihood (the field is integrated out, so its prior supplies

the Occam penalty – latent-level criteria such as WAIC track the field’s effective dimension, not the rank, and under-select). The per-rank evidence is returned in `fit$spatial$K_selection`. `sd.load` sets the loading prior scale.

Adding the SAME `icar(graph = adj)` term to the cover positive formula shares the latent fields across the two processes: the fields then also load on the cover predictor through a free loading matrix, $g(cover)_{sc} = X_c (\mu_{pos} + b_{pos_s}) + \sum_k L_{pos_{sk}} w_{kc}$, so a species’ spatial occupancy pattern and its spatial cover pattern are linked through one set of factors. The cover loadings are returned in `fit$spatial$loadings_cover`. The field is shared, so the cover-arm term must match the occupancy arm (same type and graph), and a cover-arm field without a matching occupancy field errors.

The field structure is set by the term: `icar()` (improper intrinsic CAR, the default), `car_proper(graph = adj)` (a proper CAR whose per-factor correlation ρ is returned in `fit$spatial$rho_w`), or `bym2(graph = adj)` (the Riebler 2016 convolution whose per-factor spatial-variance fraction ϕ is returned in `fit$spatial$phi_w`); the field hyperparameter is estimated by EM. All three share one engine; `fit$spatial$field_type` records the choice. Structured terms on the detection arm, or an unsupported field term, error from the dispatcher.

The shared fields imply a residual species-association matrix (the spatial-JSDM / HMSC output): `tobs_associations()` returns the occupancy association $\text{corr}(L L')$ and, with a cover-arm factor, the cover association and the cross-arm occupancy-vs-cover association, each marginalised over the loading posterior so an interval accompanies the estimate. `predict()` returns the other JSDM output – the per-species per-cell occupancy maps (ψ with $\psi_{lower} / \psi_{upper}$), marginalised over the loading + field posterior so a rare species borrows strength across the shared factors for a calibrated map.

Scope

The non-spatial fit is Laplace-EM. A community spatial `occu_cover` – a shared latent field coupled across the occupancy and cover arms with per-species RE on all three arms – is the reduced-rank spatial-factor fit: an `icar()` / `car_proper()` / `bym2()` term on the occupancy formula (and, with a matching term on the cover formula, a cover-arm factor) fits per-species loadings on the shared field via Laplace-EM and NUTS, with the per-species community covariances $\Sigma_{occ} / \Sigma_p / \Sigma_{pos}$ on top. The free per-species loading form generalises a single common field amplitude, so it subsumes the common-amplitude coupling of `occu_cover()`’s joint-coupled engine. A community model whose per-arm variance components are integrated on an outer grid (the engine route of `occu_cover()`’s joint-coupled path) is not used: the joint nested-Laplace engine integrates every variance component on its outer grid, so per-arm community RE variances plus the field hyperparameters exceed any tractable grid size – the closed-form covariance M-step of the Laplace-EM is the scaling route for community variance components. Structured terms beyond the shared field error from the dispatcher rather than being silently dropped.

Community variance debias

The community-MEAN estimates (`coef()`, `vcov()`, `confint()`) are unbiased. The community-VARIANCE components – the per-arm covariance matrices in `fit$ms_community$Sigma_occ / Sigma_p / Sigma_pos` and their `sd_*`, which set the spread of the per-species deviations – pick up Laplace small-cluster attenuation at small per-species n under the EM M-step. They are debiased by default with an adaptive Gauss-Hermite quadrature of the exact per-species RE posterior (`control$re.aghq = TRUE`, the same correction the single-arm AGHQ path applies; disable with

`re.aghq = FALSE`, set nodes with `control$n.quad`). The cover hurdle ties a species' ψ_i / p_i / cover coefficients through the data, so the per-species RE posterior is not separable across arms and the AGHQ is a tensor product over the joint RE vector – $n.quad^P$ nodes per species in the total RE dimension P . That cost is exponential in P , so the debias is a hard scope limit: it runs only up to a small total RE dimension (`control$re.aghq.maxdim`, default 4, e.g. intercept-only on all three arms), and larger RE designs keep the EM variance. The EM (Laplace) variance is a documented lower bound on the true component – attenuated toward zero at small per-species n , monotonically less attenuated as n grows – not a bias of unknown sign. `fit$ms_community$var_attenuation$debias` records "aghq" or "none", `$affects` names the lower-bounded components, and `print()` flags the EM case.

See Also

`occu_cover()` (single species), `ms_occu()` (community occupancy, no cover), `ms_abun()` (community N-mixture).

Examples

```
f <- ms_occu_cover(response = "lognormal")
f
```

nmix_laplace

Laplace fit of the Royle (2004) N-mixture model

Description

Maximum-likelihood fit (non-spatial, fixed effects only) of the Royle (2004) N-mixture model with a Poisson or negative-binomial abundance mixing distribution:

$$N_i \sim \text{Poisson}(\lambda_i) \quad \text{or} \quad N_i \sim \text{NegBin}(\text{mean} = \lambda_i, \text{size} = r), \quad y_{ij}|N_i \sim \text{Binomial}(N_i, p_{ij}),$$

with abundance linear predictor $\log \lambda_i = X_{\lambda}^{(i)} \beta_{\lambda}$ and detection linear predictor $\text{logit } p_{ij} = X_p^{(ij)} \beta_p$. The NB uses the `neg_binomial_2` convention (size r , variance $\lambda + \lambda^2/r$); Poisson is the $r \rightarrow \infty$ limit.

Optimisation uses inner Newton on $(\beta_{\lambda}, \beta_p)$ with the marginal observed Fisher information matrix as the curvature (complete-data Fisher fallback at iterates where the observed-info Hessian is not PSD). Under `mixture = "NB"` the dispersion $\theta = \log r$ is a single global scalar, profiled outside the beta-Newton by block coordinate ascent on its analytic profile score. All gradients and Hessians are analytical (no numerical differentiation), so the fit converges in a small number of iterations and is typically 20-50x faster than `unmarked::pcount()`'s BFGS-with-numerical-derivatives path on equivalent problems.

This is the non-spatial entry point; a spatial nested-Laplace path that places a latent prior block on the abundance arm is a separate entry, to be added (see `tulpa_nested_laplace_nmix()` once shipped).

Usage

```
nmix_laplace(
  y,
  site_idx,
  X_lambda,
  X_p,
  mixture = c("P", "NB"),
  beta_lambda_init = NULL,
  beta_p_init = NULL,
  log_r_init = NULL,
  r_max = 1e+05,
  K_max = NULL,
  headroom = NULL,
  max_iter = 100L,
  tol = 1e-06,
  verbose = FALSE
)
```

Arguments

<code>y</code>	Integer vector of observed counts, one entry per visit (long form).
<code>site_idx</code>	Integer vector, same length as <code>y</code> , 1-based site index indicating which site each visit belongs to.
<code>X_lambda</code>	Numeric matrix [<code>n_sites</code> x <code>p_lambda</code>] of abundance covariates.
<code>X_p</code>	Numeric matrix [<code>n_obs</code> x <code>p_p</code>] of detection covariates (long form, row order matches <code>y</code>).
<code>mixture</code>	Abundance mixing distribution: "P" (Poisson, default) or "NB" (negative binomial). Under "NB" an extra dispersion parameter <code>log_r</code> (log NB size) is estimated jointly and reported with its standard error.
<code>beta_lambda_init</code>	Optional numeric [<code>p_lambda</code>] warm start. Defaults to <code>c(log(mean(y) + 0.1), 0, 0, ...)</code> .
<code>beta_p_init</code>	Optional numeric [<code>p_p</code>] warm start. Defaults to zeros.
<code>log_r_init</code>	Optional scalar warm start for <code>log_r</code> (NB only). Defaults to a method-of-moments estimate from the site count totals, clamped to a sensible range.
<code>r_max</code>	Upper bound on the NB size <code>r</code> (NB only, default 1e5). If the optimiser pins <code>log_r</code> at <code>log(r_max)</code> the data are consistent with Poisson and a warning recommends <code>mixture = "P"</code> .
<code>K_max</code>	Marginal-sum truncation. Defaults to <code>max(y) + 100</code> (matches <code>unmarked::pcount</code>), applied per site (see <code>headroom</code>). The returned <code>boundary_weight</code> per site flags any site whose posterior over <code>N</code> puts non-trivial mass on its truncation; raise <code>K_max</code> if any such weight exceeds $\sim 1e-4$.
<code>headroom</code>	Latent- <code>N</code> states summed above each site's own <code>max(y_i)</code> . NULL (default) derives it from <code>K_max</code> : an unset <code>K_max</code> caps each site at <code>max(y_i) + 100</code> , an explicit one truncates globally and uncapped. A negative value disables the per-site cap.

max_iter	Newton iteration budget (default 100).
tol	Gradient-norm convergence tolerance (default 1e-6).
verbose	Print per-iteration (log_lik, grad_norm, boundary_max).

Value

A list of class `nmix_fit`:

- `beta_lambda`, `beta_p` – MLE coefficient vectors
- `mixture` – "P" or "NB"
- `log_r`, `r` – estimated log NB size and the NB size $\exp(\log_r)$ (NB only; NA under Poisson). `vcov` carries `log_r` as its last coordinate, so $\sqrt{\text{diag}(\text{vcov})["\log_r"]}$ is its standard error.
- `log_lik` – marginal log-likelihood at the mode
- `vcov` – variance-covariance matrix (marginal observed Fisher inverse), over (`beta_lambda`, `beta_p`) and, under NB, `log_r`
- `vcov_ok` – whether the final observed-info Cholesky succeeded
- `H_obs` – final marginal observed Fisher information matrix
- `n_iter` – (outer) iterations consumed
- `converged` – whether the convergence tolerance was reached
- `grad_norm` – gradient norm at termination
- `mean_N`, `var_N` – per-site posterior mean and variance of $N \mid y$
- `boundary_weight` – per-site posterior weight on $N = K_{\max}$
- `call` – matched call

References

Royle, J. A. (2004). N-mixture models for estimating population size from spatially replicated counts. *Biometrics* 60, 108-115. Dennis, E. B., Morgan, B. J. T., Ridout, M. S. (2015). Computational aspects of N-mixture models. *Biometrics* 71, 237-246.

nmix_laplace_bym2	<i>BYM2 Royle (2004) N-mixture model via nested Laplace</i>
-------------------	---

Description

Nested-Laplace fit of the spatial N-mixture model with a BYM2 (Riebler et al. 2016) prior on the abundance-arm spatial offset. The offset decomposes as

$$\phi_u = \sigma \left(\sqrt{\rho/s} v_u + \sqrt{1 - \rho} w_u \right),$$

with $v \sim \text{ICAR}$ (unscaled, sum-to-zero) and $w \sim N(0, I)$ iid. s is the Riebler scaling factor, the geometric mean of the marginal variances $\text{diag}(Q^+)$ of the intrinsic ICAR field; σ is then the joint marginal standard deviation of ϕ , and $\rho \in [0, 1]$ is the spatial fraction of variance.

The inner Newton works in the joint state $x = (\beta_\lambda, \beta_p, v, w)$ (dimension $p_\lambda + p_p + 2n_{\text{spatial}}$). At the converged mode the Laplace log-marginal is accumulated; the outer 2D grid integrates over (σ, ρ) .

Usage

```
nmix_laplace_bym2(
  y,
  site_idx,
  map_site_to_unit,
  X_lambda,
  X_p,
  adj_row_ptr,
  adj_col_idx,
  n_neighbors,
  n_spatial,
  sigma_grid = NULL,
  rho_grid = NULL,
  mixture = c("P", "NB"),
  r_grid = NULL,
  scale_factor = NULL,
  beta_lambda_init = NULL,
  beta_p_init = NULL,
  v_init = NULL,
  w_init = NULL,
  K_max = NULL,
  max_iter = 100L,
  tol = 1e-06,
  verbose = FALSE
)
```

Arguments

<code>y</code>	Integer vector of observed counts (long form, one entry per visit).
<code>site_idx</code>	Integer vector, 1-based site index for each visit, same length as <code>y</code> .
<code>map_site_to_unit</code>	Integer vector of length <code>n_sites</code> , 1-based spatial unit index for each site. Sites can share spatial units; the data contribution to <code>z[u]</code> aggregates across all sites that map to <code>u</code> .
<code>X_lambda</code>	Numeric matrix [<code>n_sites</code> × <code>p_lambda</code>] of abundance covariates.
<code>X_p</code>	Numeric matrix [<code>n_obs</code> × <code>p_p</code>] of detection covariates.
<code>adj_row_ptr, adj_col_idx, n_neighbors</code>	CSR adjacency for the ICAR graph on the <code>n_spatial</code> units. <code>adj_row_ptr</code> has length <code>n_spatial + 1</code> , <code>adj_col_idx</code> lists 0-based neighbours, and <code>n_neighbors[s]</code> is the row degree of unit <code>s</code> .
<code>n_spatial</code>	Number of spatial units.
<code>sigma_grid</code>	Numeric vector of σ (joint sd) grid points. Defaults to <code>exp(seq(log(0.2), log(3), length.out = 5L))</code> .
<code>rho_grid</code>	Numeric vector of spatial-fraction grid points in $[0, 1]$. Defaults to <code>c(0.05, 0.3, 0.5, 0.7, 0.95)</code> .

mixture	Abundance mixing distribution "P" (default) or "NB"; under "NB" the NB size r is integrated as an additional outer grid axis.
r_grid	Optional NB size grid (NB only); see nmix_laplace_icar() .
scale_factor	Optional scalar Riebler scaling factor s , the geometric mean of the marginal variances $\text{diag}(Q^+)$ of the intrinsic ICAR field. If NULL, it is computed from the adjacency via dense eigendecomposition.
beta_lambda_init	Optional warm start; default $c(\log(\text{mean}(y)+0.1), 0, \dots)$.
beta_p_init	Optional warm start; default $\text{rep}(0, p_p)$.
v_init, w_init	Optional warm starts (each length n_{spatial}) for the two BYM2 latent components – the structured (spatial) and unstructured (iid) effects. NULL (default) starts both at zero.
K_max	Truncation for the per-site marginal sum over N . Defaults to $\max(y) + 100$. Returned <code>boundary_max</code> holds, per grid point, the worst site's posterior mass on K_{max} ; the fit warns when the grid-weighted mass $\text{sum}(\text{weights} * \text{boundary_max})$ exceeds $1e-4$ – raise K_{max} then.
max_iter, tol	Inner Newton iteration budget and gradient-norm tol.
verbose	Print per-iteration and per-grid-point progress.

Value

A list of class `nmix_spatial_fit`:

- `theta_grid` – $[n_{\text{grid}} \times 2]$ matrix of (σ, ρ) per grid point
- `sigma_grid, rho_grid` – input axes
- `scale_factor` – the Riebler scaling factor
- `log_marginal, weights, sigma_mean, sigma_sd, rho_mean, rho_sd` – posterior summaries of the joint hyperparameters
- `modes` – $[n_{\text{grid}} \times (p_{\text{lambda}} + p_p + 2 n_{\text{spatial}})]$ per-grid modes; columns $(p_{\text{lambda}} + p_p + 1) \dots (p_{\text{lambda}} + p_p + n_{\text{spatial}})$ are v , the next n_{spatial} columns are w
- `beta_lambda_mean, beta_p_mean` – weighted-mean coefficients
- `v_mean, w_mean` – weighted-mean ICAR and iid components
- `phi_mean` – weighted-mean total offset ϕ
- other diagnostic fields as in [nmix_laplace_icar\(\)](#)

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, 1145-1165.
 Morris, M., Wheeler-Martin, K., Simpson, D., Mooney, S. J., Gelman, A., DiMaggio, C. (2019). Bayesian hierarchical spatial models: Implementing the BYM2 model in Stan. *Spatial and Spatio-temporal Epidemiology* 31.

nmix_laplace_car_proper

Proper CAR Royle (2004) N-mixture model via nested Laplace

Description

Nested-Laplace fit of the spatial N-mixture model with a proper conditional autoregressive prior on the abundance-arm spatial offset:

$$N_i \sim \text{Poisson}(\lambda_i), \quad y_{ij}|N_i \sim \text{Binomial}(N_i, p_{ij}),$$

$\log \lambda_i = X_{\lambda}^{(i)} \beta_{\lambda} + z_{u(i)}$, where $z \mid \tau, \rho \sim N(0, [\tau(D - \rho W)]^{-1})$. Both hyperparameters are integrated over an outer 2D grid; the inner Newton step shares the kernel with [nmix_laplace_icar\(\)](#) (ICAR is the $\rho = 1$ limit).

Unlike the ICAR fit, no sum-to-zero centering is applied – $Q(\rho)$ is full rank for $\rho < 1$. The per-rho $\log |Q(\rho)|$ is precomputed once via a dense Cholesky on the $n_{\text{spatial}} \times n_{\text{spatial}}$ precision matrix.

Usage

```
nmix_laplace_car_proper(
  y,
  site_idx,
  map_site_to_unit,
  X_lambda,
  X_p,
  adj_row_ptr,
  adj_col_idx,
  n_neighbors,
  n_spatial,
  tau_grid = NULL,
  rho_grid = NULL,
  mixture = c("P", "NB"),
  r_grid = NULL,
  beta_lambda_init = NULL,
  beta_p_init = NULL,
  z_init = NULL,
  K_max = NULL,
  max_iter = 100L,
  tol = 1e-06,
  verbose = FALSE
)
```

Arguments

y	Integer vector of observed counts (long form, one entry per visit).
site_idx	Integer vector, 1-based site index for each visit, same length as y.

map_site_to_unit	Integer vector of length n_{sites} , 1-based spatial unit index for each site. Sites can share spatial units; the data contribution to $z[u]$ aggregates across all sites that map to u .
X_lambda	Numeric matrix [n_{sites} x p_{lambda}] of abundance covariates.
X_p	Numeric matrix [n_{obs} x p_{p}] of detection covariates.
adj_row_ptr, adj_col_idx, n_neighbors	CSR adjacency for the ICAR graph on the n_{spatial} units. <code>adj_row_ptr</code> has length $n_{\text{spatial}} + 1$, <code>adj_col_idx</code> lists 0-based neighbours, and <code>n_neighbors[s]</code> is the row degree of unit s .
n_spatial	Number of spatial units.
tau_grid	Optional numeric vector of τ grid points (defaults to <code>exp(seq(log(0.3), log(30), length.out = 7L))</code>).
rho_grid	Optional numeric vector of ρ grid points in the valid eigenvalue interval. Defaults to a 5-point grid in $(0, 1)$ – callers that want eigenvalue-derived bounds should compute them via <code>tulpa::spatial_car_proper(adjacency)\$rho_bounds</code> and pass an explicit grid in that interval.
mixture	Abundance mixing distribution "P" (default) or "NB"; under "NB" the NB size r is integrated as an additional outer grid axis.
r_grid	Optional NB size grid (NB only); see nmix_laplace_icar() .
beta_lambda_init	Optional warm start; default <code>c(log(mean(y)+0.1), 0, ...)</code> .
beta_p_init	Optional warm start; default <code>rep(0, p_p)</code> .
z_init	Optional warm start for the spatial field; default zeros.
K_max	Truncation for the per-site marginal sum over N . Defaults to $\max(y) + 100$. Returned <code>boundary_max</code> holds, per grid point, the worst site's posterior mass on K_{max} ; the fit warns when the grid-weighted mass <code>sum(weights * boundary_max)</code> exceeds $1e-4$ – raise K_{max} then.
max_iter, tol	Inner Newton iteration budget and gradient-norm tol.
verbose	Print per-iteration and per-grid-point progress.

Value

A list of class `nmix_spatial_fit`:

- `theta_grid` – [n_{grid} x 2] matrix of (τ, ρ) per grid point
- `tau_grid, rho_grid` – input axes
- `log_det_Q_rho` – precomputed log determinants per ρ
- `log_marginal, weights, tau_mean, tau_sd, rho_mean, rho_sd` – as in [nmix_laplace_icar\(\)](#) but with the ρ marginal added
- other diagnostic fields and named coefficient means as in ICAR

References

Cressie, N. (1993). *Statistics for Spatial Data*. Wiley. Rue, H., Held, L. (2005). *Gaussian Markov Random Fields*. CRC.

nmix_laplace_icar *Spatial Royle (2004) N-mixture model via nested Laplace*

Description

Nested-Laplace fit of the spatial N-mixture model

$$N_i \sim \text{Poisson}(\lambda_i), \quad y_{ij}|N_i \sim \text{Binomial}(N_i, p_{ij}),$$

with abundance linear predictor $\log \lambda_i = X_\lambda^{(i)} \beta_\lambda + z_{u(i)}$ where $z \sim \text{ICAR}(\tau)$ is an intrinsic conditional autoregressive field on the user-supplied adjacency graph, and detection linear predictor $\text{logit } p_{ij} = X_p^{(ij)} \beta_p$.

The hyperparameter τ (ICAR precision) is integrated out by an outer grid: at each τ_k the inner Newton finds the joint mode of $(\beta_\lambda, \beta_p, z)$ and the Laplace log-marginal $\log p(y \mid \tau_k)$ is accumulated. Posterior weights over τ normalise the grid.

Inner Newton uses the marginal observed Fisher information matrix for curvature (with the $\text{Var}[N \mid y_i]$ rank-1 correction encoding cross-arm coupling) and falls back to the complete-data Fisher block when the observed-info matrix is not PSD. A small diagonal ridge keeps the Cholesky stable through the (intercept, constant- z) structural null direction; z is centered to sum zero after every step.

Usage

```
nmix_laplace_icar(
  y,
  site_idx,
  map_site_to_unit,
  X_lambda,
  X_p,
  adj_row_ptr,
  adj_col_idx,
  n_neighbors,
  n_spatial,
  tau_grid = NULL,
  mixture = c("P", "NB"),
  r_grid = NULL,
  beta_lambda_init = NULL,
  beta_p_init = NULL,
  z_init = NULL,
  K_max = NULL,
  max_iter = 100L,
  tol = 1e-06,
  verbose = FALSE
)
```

Arguments

<code>y</code>	Integer vector of observed counts (long form, one entry per visit).
<code>site_idx</code>	Integer vector, 1-based site index for each visit, same length as <code>y</code> .
<code>map_site_to_unit</code>	Integer vector of length <code>n_sites</code> , 1-based spatial unit index for each site. Sites can share spatial units; the data contribution to <code>z[u]</code> aggregates across all sites that map to <code>u</code> .
<code>X_lambda</code>	Numeric matrix [<code>n_sites</code> x <code>p_lambda</code>] of abundance covariates.
<code>X_p</code>	Numeric matrix [<code>n_obs</code> x <code>p_p</code>] of detection covariates.
<code>adj_row_ptr, adj_col_idx, n_neighbors</code>	CSR adjacency for the ICAR graph on the <code>n_spatial</code> units. <code>adj_row_ptr</code> has length <code>n_spatial + 1</code> , <code>adj_col_idx</code> lists 0-based neighbours, and <code>n_neighbors[s]</code> is the row degree of unit <code>s</code> .
<code>n_spatial</code>	Number of spatial units.
<code>tau_grid</code>	Optional numeric vector of τ grid points. Defaults to <code>exp(seq(log(0.3), log(30), length.out = 9))</code> .
<code>mixture</code>	Abundance mixing distribution: "P" (Poisson, default) or "NB" (negative binomial). Under "NB" the NB size r is integrated as an additional outer grid dimension (alongside τ); the posterior <code>r_mean / r_sd</code> are reported from the grid weights.
<code>r_grid</code>	Optional numeric vector of NB size grid points (NB only). Defaults to <code>exp(seq(log(0.5), log(40), length.out = 6))</code> . Ignored under Poisson.
<code>beta_lambda_init</code>	Optional warm start; default <code>c(log(mean(y)+0.1), 0, ...)</code> .
<code>beta_p_init</code>	Optional warm start; default <code>rep(0, p_p)</code> .
<code>z_init</code>	Optional warm start for the spatial field; default zeros.
<code>K_max</code>	Truncation for the per-site marginal sum over <code>N</code> . Defaults to <code>max(y) + 100</code> . Returned <code>boundary_max</code> holds, per grid point, the worst site's posterior mass on <code>K_max</code> ; the fit warns when the grid-weighted mass <code>sum(weights * boundary_max)</code> exceeds <code>1e-4</code> – raise <code>K_max</code> then.
<code>max_iter, tol</code>	Inner Newton iteration budget and gradient-norm tol.
<code>verbose</code>	Print per-iteration and per-grid-point progress.

Value

A list of class `nmix_spatial_fit`:

- `tau_grid` – input grid
- `log_marginal` – log marginal at each `tau` (up to a `tau`-independent constant)
- `weights` – normalised grid weights (sum to 1)
- `tau_mean, tau_sd` – posterior moments of `tau`
- `modes` – [`n_grid` x (`p_lambda` + `p_p` + `n_spatial`)] matrix of inner modes
- `beta_lambda_mean, beta_p_mean` – weighted-mean coefficient estimates

- z_mean – weighted-mean spatial field
- n_iter, converged, grad_norm, log_lik, boundary_max – per-grid diagnostics
- p_lambda, p_p, n_spatial, K_max – echoed dimensions
- call – matched call

References

Royle, J. A. (2004). N-mixture models for estimating population size from spatially replicated counts. *Biometrics* 60, 108-115. Besag, J., York, J., Mollie, A. (1991). Bayesian image restoration with two applications in spatial statistics. *Ann. Inst. Statist. Math.* 43, 1-20. Rue, H., Martino, S., Chopin, N. (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *JRSS-B* 71, 319-392.

nmix_laplace_re

Community / multispecies N-mixture by Laplace

Description

Fits the community (spAbundance msNMix) N-mixture model: a per-species Royle (2004) N-mixture with Gaussian community hyperpriors on the per-species abundance and detection coefficients,

$$\begin{aligned}
 N_{s,i} &\sim \text{Poisson}(\lambda_{s,i}), \quad y_{s,i,j} | N \sim \text{Binomial}(N_{s,i}, p_{s,i,j}), \\
 \log \lambda_{s,i} &= X_{\lambda}^{(i)}(\mu_{\lambda} + b_s^{\lambda}), \quad \text{logit } p_{s,i,j} = X_p^{(ij)}(\mu_p + b_s^p), \\
 b_s^{\lambda} &\sim N(0, \Sigma_{\lambda}), \quad b_s^p \sim N(0, \Sigma_p).
 \end{aligned}$$

The latent abundances integrate out per species-site in closed form (the shared N-mixture kernel exposed by [nmix_site_marginal\(\)](#)); the per-species coefficient deviations $b_s = (b_s^{\lambda}, b_s^p)$ are the random effects. Both solvers assemble the per-species marginal in compiled code via a native oracle (no per-group round trip into R). At n_quad = 1 (the default, the joint Laplace / glmer nAGQ = 1) the fit is a Laplace-EM (block-coordinate Newton mode + closed-form covariance M-step + Schur- complement SEs) – the fast production path. A higher n_quad routes the same native oracle through the shared compiled AGHQ engine ([tulpa_re_aghq\(\)](#)), replacing each species' Laplace integral with adaptive Gauss-Hermite quadrature to reduce the small-cluster (few species) downward bias of the community covariances Sigma_lambda / Sigma_p – at a n_quad^(p_lambda + p_p) per-species grid cost. Each species' marginal – value, abundance/detection score, and the per-site observed- information block carrying the $\text{Var}(N_i | y_i)$ abundance/ detection coupling – is supplied as the per-group oracle, so there is one marginal/quadrature/covariance implementation across the package. The Gaussian community priors pin (μ_{λ}, μ_p) as fixed effects, so no sum-to-zero constraint is needed; their standard errors come from the marginal observed-information Hessian.

The abundance mixing distribution is Poisson (mixture = "P") or negative binomial (mixture = "NB"). Under NB the dispersion is itself a per-species random effect, $\log_r_s \sim N(\mu_{\log_r}, \sigma_{\log_r})$ (i.e. $r_s \sim \text{LogNormal}$): the per-species RE vector widens to $b_s = (b_{\lambda_s}, b_{p_s}, b_{\log_r_s})$, the community log-dispersion $\mu_{\log_r}$ joins (μ_{λ}, μ_p) as a fixed effect, and $\sigma_{\log_r}$ joins the community covariances as a third (scalar) block, all integrated jointly by the same AGHQ engine. The per-species size is $r_s = \exp(\mu_{\log_r} + b_s^{\log_r})$. Poisson is the $r \rightarrow \infty$ limit (no dispersion coordinate).

Usage

```
nmix_laplace_re(
  y,
  site_idx,
  species_idx,
  X_lambda,
  X_p,
  n_sites,
  n_species,
  mu_lambda_init = NULL,
  mu_p_init = NULL,
  Sigma_lambda_init = NULL,
  Sigma_p_init = NULL,
  K_max = NULL,
  headroom = NULL,
  max_iter = 200L,
  optimizer = c("em", "joint_fd", "joint_grad"),
  mixture = c("P", "NB", "ZIP", "ZINB"),
  r_init = 10,
  sigma_logr_init = 0.5,
  omega_init = 0.2,
  sigma_omega_init = 0.5,
  n_quad = 1L,
  n_quad_scalar = .TOBS_MIN_SCALAR_NQUAD,
  lkj_eta = 1,
  sigma_beta = 100,
  omega_sigma_prior = c(1, 0.05),
  logr_sigma_prior = NULL,
  verbose = FALSE
)
```

Arguments

<code>y</code>	Integer vector of counts, one entry per observed visit (long form, all species stacked).
<code>site_idx</code>	Integer vector (same length as <code>y</code>), 1-based site index.
<code>species_idx</code>	Integer vector (same length as <code>y</code>), 1-based species index.
<code>X_lambda</code>	Numeric matrix [<code>n_sites</code> x <code>p_lambda</code>] of abundance covariates (shared across species; one row per site).
<code>X_p</code>	Numeric matrix [<code>n_obs</code> x <code>p_p</code>] of detection covariates (long form, row order matching <code>y</code>).
<code>n_sites, n_species</code>	Integer counts.
<code>mu_lambda_init, mu_p_init</code>	Optional warm starts for the community means. Default: the column means of independent per-species <code>nmix_laplace()</code> fits.

Sigma_lambda_init, Sigma_p_init	Optional warm starts for the community covariances. Default: the (ridge-regularized) sample covariance of the per-species coefficient estimates.
K_max	Marginal-sum truncation (default $\max(y) + 100$), applied per site (see headroom).
headroom	Latent-N states summed above each site's own $\max(y_i)$. NULL (default) derives it from K_max: an unset K_max caps each site at $\max(y_i) + 100$, an explicit one truncates globally and uncapped. A caller that resolved the ceiling itself passes both.
max_iter	Optimizer iteration cap (default 200).
optimizer	Outer optimize driver over the shared native oracle: "em" (default) is the fast Laplace-EM (block-coordinate Newton mode + closed-form covariance M-step + Schur SE), the exact $n_{\text{quad}} = 1$ solver. "joint_grad" and "joint_fd" are the joint (theta, log-Cholesky Sigma) optimizers (tulpa_re_aghq()) and both do the $n_{\text{quad}} > 1$ AGHQ debias of the community covariances. "joint_grad" (the fast debias path) supplies the analytic Fisher-identity gradient – one group sweep per step, no per-coordinate objective re-solve – and requires $n_{\text{quad}} > 1$ (at $n_{\text{quad}} = 1$ use the EM). "joint_fd" finite-differences the objective (slower; the FD sweep re-solves every per-species mode per coordinate) and is kept for correctness / architecture validation and as the $n_{\text{quad}} = 1$ joint reference.
mixture	Abundance mixing distribution: "P" (Poisson, default), "NB" (negative binomial with a per-species dispersion random effect $\log_r_s \sim N(\mu_{\log_r}, \sigma_{\log_r})$), or their zero-inflated counterparts "ZIP" / "ZINB" (a per-species structural-zero random effect $\logit_omega_s \sim N(\mu_{\omega}, \sigma_{\omega})$; a share ω_{share} of a species' sites is structurally empty). None of "NB" / "ZIP" / "ZINB" has a closed-form EM, so each defaults optimizer to "joint_grad" and errors on optimizer = "em". The default n_{quad} when unsupplied is 3 for "NB" and the zero-inflated "ZIP" / "ZINB" (the tensor grid is $n_{\text{quad}}^{(p_{\lambda} + p_p + [\text{NB}] + [\text{ZI}])}$, so a coarser grid keeps it tractable; #355).
r_init	Initial community-mean negative-binomial size for the joint optimizer (mixture = "NB" only; default 10, a moderate overdispersion start). The optimizer carries $\mu_{\log_r} = \log(r_{\text{init}})$ as the $(p_{\lambda} + p_p + 1)$ -th fixed effect.
sigma_logr_init	Initial standard deviation of the per-species log-dispersion random effect $\log_r_s$ (mixture = "NB" / "ZINB" only; default 0.5). Seeds the scalar $\log_r$ covariance block.
omega_init	Initial community-mean structural-zero probability for the joint optimizer (mixture = "ZIP" / "ZINB" only; default 0.2). The optimizer carries $\mu_{\omega} = \text{qlogis}(\omega_{\text{init}})$ as a trailing fixed effect.
sigma_omega_init	Initial standard deviation of the per-species structural-zero-logit random effect $\logit_omega_s$ (mixture = "ZIP" / "ZINB" only; default 0.5). Seeds the scalar ω covariance block.
n_quad	Quadrature points per random-effect dimension passed to tulpa_re_aghq() (default 1, the joint Laplace). A higher n_{quad} debiases the community covariances at a $n_{\text{quad}}^{(p_{\lambda} + p_p)}$ per-species grid cost, so keep it modest

	when the coefficient dimension is large. This sets the order of the correlated coefficient blocks (λ , p); the scalar nuisance blocks use <code>n_quad_scalar</code> .
<code>n_quad_scalar</code>	Quadrature points for the scalar nuisance random-effect blocks – the negative-binomial dispersion <code>log_r</code> and the zero-inflation <code>logit_omega</code> (default 3). These integrate a 1-D posterior, so <code>tulpa_re_aghq()</code> 's per-block quadrature runs them at this coarser order rather than the full <code>n_quad</code> , trimming the <code>n_quad^(NB + ZI)</code> factor the nuisance axes would otherwise multiply into the tensor grid. Floored at 3 (and the floor wins over a smaller <code>n_quad</code>): the rule is converged there, and below it the reported community-mean SE on that block can collapse by an order of magnitude with no other symptom.
<code>lkj_eta</code>	LKJ shape regularizing each <i>correlated</i> community covariance block's correlation off the boundary (default 1, no penalty); passed through to <code>tulpa_re_aghq()</code> . Does not touch the marginal SDs.
<code>sigma_beta</code>	Weak Gaussian ridge SD on the community means (default 100, i.e. $\tau = 1e-4$, matching the other Laplace paths); stabilizes a weakly-identified community mean without materially shifting it.
<code>omega_sigma_prior</code>	Penalized-Complexity prior $c(U, \alpha)$ ($P(\sigma_{\omega} > U) = \alpha$) on the per-species structural-zero random effect SD (mixture = "ZIP" / "ZINB" only; default $c(1, 0.05)$, interior mode ~ 0.33). σ_{ω} is the softest AGHQ direction and at few species can collapse to the boundary, flattening the marginal Hessian and attenuating the recovered SD; the weak prior adds curvature there (passed to <code>tulpa_re_aghq()</code> 's <code>sigma_prior</code>) without biasing an identified fit. NULL restores pure ML on the structural-zero variance. Ignored for Poisson / NB.
<code>logr_sigma_prior</code>	Penalized-Complexity prior $c(U, \alpha)$ ($P(\sigma_{\log_r} > U) = \alpha$) on the per-species log-dispersion random effect SD (mixture = "NB" / "ZINB" only; default NULL, pure ML). $\sigma_{\log_r}$ is the same shape of parameter as σ_{ω} – one scalar variance over species – and settles near its lower boundary the same way at few species. At 8 and 36 species with a simulated $\sigma_{\log_r} = 0.5$, <code>n_quad = 3</code> and <code>n_quad_scalar = 3</code>, fits recovering $\sigma_{\log_r} \geq 0.30$ covered $\mu_{\log_r}$ 33/34 while those below covered 2/5. That split is a property of those two group counts and does not carry to a third. Re-measured at 18 species on the same fixture and seeds (19 fits), one fit sits below 0.30, the point-estimate error is uncorrelated with the recovered SD (Spearman -0.08), and the reported SE is very nearly a deterministic multiple of it (Spearman +0.98, $R^2 = 0.99$). So what a threshold split picks up at 18 species is the SE side alone: a low recovered SD buys a proportionally narrower interval around an error that did not shrink with it. Both halves moving together is what the 8-and-36 pool showed and what is absent there. The SE itself is $se^2 = (\hat{\sigma}_{\log_r}^2 + c)/S$, with c the per-species inverse information for $\log_r$s. Measured over 97 fits at 8 / 18 / 36 species, c is 0.130 / 0.133 / 0.141 – one constant across a factor of 4.5 in S – so the shape is right and $\sigma_{\hat{}}$ is the input that is off. Its attenuation (0.418 / 0.448 / 0.487 against a simulated 0.5) is monotone in S and passes straight into the interval. A 1.28x-too-narrow $\mu_{\log_r}$ interval at 18 species on this fixture is mostly the seed block, not the estimator. $\mu_{\log_r}$ is a population mean and each seed

draws S log-dispersions around it, so the across-seed error carries a $\sigma_{\log r}^2 / S$ term that the SE includes and that is itself measured on ~ 19 seeds: it supplies about two thirds of the spread, and the 18-species blocks drew it 18-21% wide (chi-square $p = 0.12$ and 0.03). Putting that draw at its expectation and rebuilding the SE at the simulated σ gives a scale of $0.990 / 1.035 / 0.963$ at $8 / 18 / 36$ species. Use `simulate_ms_abun()`'s `truth$mu_log_r_real` to score against the seed's own realized species mean and keep the two apart.

A fit now says whether its dispersion variance is distinguishable from zero at all: `fit$ms_dispersion$sigma_log_r_boundary` carries the boundary test and a fit that fails it warns.

What the penalty reaches on this block appears to be the boundary rather than the calibration. Measured on 20 seeds of that fixture at 8 species with `c(U, alpha) = c(1, 0.05)`: a $\sigma_{\log r}$ of 0.01-0.06 lifts by an order of magnitude and a seed that stopped at a singular marginal Hessian under pure ML converges and covers, while paired coverage holds at 17 of 19 with the same two seeds missing, both of them at $\sigma_{\log r}$ around 0.2 where the penalty is weak. The fits that were already calibrated take a small systematic shift ($\mu_{\log r}$ by -0.006 , $p = 0.001$; SEs a median 4.2% narrower). Hence the NULL default: it is a lever for a fit whose dispersion variance came back near zero or that failed outright, rather than a correction expected to hold across fits.

When both this and `omega_sigma_prior` are set they must be equal: the engine applies one Penalized-Complexity prior across every block it regularizes. Ignored for Poisson / ZIP.

`verbose` Print the per-iteration community-covariance change of the optimizer = "em" solver; the joint AGHQ optimizers are silent.

Value

A list of class `nmix_re_fit`: `mu_lambda`, `mu_p` (community means), `vcov` (their joint covariance from the AGHQ marginal Hessian, $(p_{\lambda} + p_p)$ square; marginalizes the community-covariance uncertainty rather than plugging in Σ), `Sigma_lambda`, `Sigma_p` (community covariances), `b_lambda`, `b_p` (per-species BLUP deviations, `n_species` rows), `blup_cov_g` (per-species list of the posterior covariance of the species' stacked RE vector, `lambda` then `p`, then the `log_r` / `logit_omega` deviation when present), `log_lik` (AGHQ marginal), `converged`, `K_max`, `n_quad`, `lkj_eta`, and (when `mixture = "NB"`) the dispersion summaries: `mu_log_r` (community-mean log-dispersion; its SE is the trailing `vcov` diagonal), `sigma_log_r` (the per-species log-dispersion SD), `b_logr` (per-species deviations), `r_s` (per-species sizes $\exp(\mu_{\log r} + b_s^{\log r})$), and `r` equal to $\exp(\mu_{\log r})$ (the community-mean size, the LogNormal median).

References

Royle, J. A. (2004). N-mixture models for estimating population size from spatially replicated counts. *Biometrics* 60, 108-115. Doser, J. et al. (2023). `spAbundance.msNMix()`.

See Also

`nmix_laplace()` (single species), `nmix_site_marginal()` (the per-species marginal primitive), `tulpa_re_aghq()` (the shared random-effect integrator).

nmix_site_marginal	<i>Per-site N-mixture marginal as a composable random-effect callback</i>
--------------------	---

Description

Exposes the Royle (2004) N-mixture per-site marginal – the latent abundance N_i summed out in closed form – as a reusable building block for integrating grouped random effects over the abundance and/or detection linear predictors. It is the bridge between the marginal kernel (which already differentiates through both arms) and a random-effect integrator such as `tulpa_re_aghq()`.

Unlike `nmix_laplace()`, which fixes the coefficients and returns a point fit, this helper returns a *closure object* that evaluates the marginal and its eta-level derivatives at arbitrary linear predictors. A random-effect integrator perturbs the predictors by Zb per group and calls `eval()` at each quadrature point; the per-site value, gradient and observed- information block are everything the per-group integral needs.

The abundance arm is per site ($\log \lambda_i = \eta_i^\lambda$) and the detection arm is per visit ($\text{logit } p_{ij} = \eta_{ij}^p$); the two are coupled through the shared latent N_i . The per-site marginal observed information in the eta coordinates $(\eta_i^\lambda, \eta_{i1}^p, \dots, \eta_{iJ_i}^p)$ is

$$B_i = \text{diag}(I_i^\lambda, I_{ij}^p) - \text{Var}(N_i | y_i) v_i v_i^\top, \quad v_i = (-w_i, p_{i1}, \dots, p_{iJ_i}),$$

where I_i^λ, I_{ij}^p are the complete-data Fisher diagonal, w_i is the abundance score weight (1 Poisson, $1 - q_i$ NB) and $p_{ij} = \text{plogis}(\eta_{ij}^p)$. The off-diagonal $\text{Var}(N_i) w_i p_{ij}$ is the abundance/detection coupling an integrator placing random effects on both arms must carry (Louis 1982). This is the eta-level form of the curvature `nmix_laplace()` sandwiches with the design matrices.

The single- vs multi-arm `make_site` adapter that wires this into a specific integrator is deliberately not built here – this object is the arm-agnostic foundation it sits on.

Usage

```
nmix_site_marginal(
  y,
  site_idx,
  X_lambda,
  X_p,
  mixture = c("P", "NB"),
  K_max = NULL,
  headroom = NULL,
  K_site = NULL
)
```

Arguments

<code>y</code>	Integer vector of observed counts, one entry per visit (long form).
<code>site_idx</code>	Integer vector, same length as <code>y</code> , 1-based site index indicating which site each visit belongs to.
<code>X_lambda</code>	Numeric matrix [<code>n_sites</code> x <code>p_lambda</code>] of abundance covariates.

X_p	Numeric matrix [n_obs x p_p] of detection covariates (long form, row order matches y).
mixture	Abundance mixing distribution: "P" (Poisson, default) or "NB" (negative binomial). Under "NB" an extra dispersion parameter log_r (log NB size) is estimated jointly and reported with its standard error.
K_max	Marginal-sum truncation. Defaults to $\max(y) + 100$ (matches <code>unmarked::pcount</code>), applied per site (see headroom). The returned <code>boundary_weight</code> per site flags any site whose posterior over N puts non-trivial mass on its truncation; raise K_max if any such weight exceeds $\sim 1e-4$.
headroom	Latent-N states summed above each site's own $\max(y_i)$. NULL (default) derives it from K_max: an unset K_max caps each site at $\max(y_i) + 100$, an explicit one truncates globally and uncapped. A negative value disables the per-site cap. Callers that resolve the ceiling themselves (the community fitters share one K_max across species) pass both.
K_site	Per-site latent-N ceiling, an integer vector of length n_sites (NULL = none). The same ceiling headroom derives from $\max(y_i)$, stated directly instead: a caller holding a fitted λ_i keys each site's ceiling to that site's abundance scale, which $\max(y_i)$ understates wherever detection is low. Takes precedence over headroom, and must be at least each site's own $\max(y_i)$.

Value

An object of class `nmix_marginal`: a list of the validated data plus closures

- `eval(eta_lambda, eta_p, r = Inf)` – evaluate at the abundance log predictor `eta_lambda` (length n_sites) and detection logit predictor `eta_p` (length n_obs, visit order matching y). Returns a list with `log_lik` (total), `log_lik_site`, `grad_eta_lambda`, `grad_eta_p`, `grad_theta`, the complete-data Fisher `info_eta_lambda / info_eta_p`, the abundance score weight score `wt_lambda`, `p` (= `plogis(eta_p)`), `mean_N`, `var_N`, `boundary_weight`, and (NB) the dispersion-coupling pieces `info_theta / info_lambda_theta / cov_N_stheta / var_stheta`.
- `eval_beta(beta_lambda, beta_p, r = Inf)` – the same, with the predictors formed from the stored design matrices.
- `obs_info_block(s, ev)` – the $(1 + J_s) \times (1 + J_s)$ per-site marginal observed-information matrix B_s (above) for site `s`, from an `eval()` / `eval_beta()` result `ev`. Coordinates are `(eta_lambda_s, eta_p over site s's visits in input order)`. Plus `n_sites`, `n_obs`, `p_lambda`, `p_p`, `mixture`, `K_max`, `obs_by_site` (per-site visit row indices).

References

Royle, J. A. (2004). N-mixture models for estimating population size from spatially replicated counts. *Biometrics* 60, 108-115. Louis, T. A. (1982). Finding the observed information matrix when using the EM algorithm. *JRSS-B* 44, 226-233.

See Also

`nmix_laplace()` for the fixed-effects fit, `tulpa_re_aghq()` for the grouped random-effect integrator this feeds.

nobs.tobs_fit	<i>Number of observations</i>
---------------	-------------------------------

Description

The count of observed response entries the fit's likelihood reads, which is the sample size `stats::AIC()` and `stats::BIC()` form an information criterion at. What one entry is follows the family's own response layout: a surveyed (site, visit) cell for the occupancy families, a (site, band) count for distance sampling, a (site, visit, source) cell for the integrated ones.

Usage

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

Arguments

object	A tobs_fit object.
...	Ignored.

Value

Integer count of the fit's observed response entries. A model type with no registered count is an error naming that type.

occu	<i>Single-season occupancy family</i>
------	---------------------------------------

Description

Latent Bernoulli occupancy state with binomial detection per visit. The MacKenzie et al. (2002) single-season model.

Usage

```
occu()
```

Value

A tobs_family object.

Engines

- `method = "laplace"` – penalized EM + Laplace with Gaussian marginals, the default.
- `method = "laplace_sla"` – the Laplace fit with simplified-Laplace skew-corrected marginals.
- `method = "laplace_gibbs" / method = "laplace_mi"` – the Laplace fit with a post-EM Gibbs / multiple-imputation variance correction (Rubin-pooled refits, not an MCMC chain).
- `method = "pg_gibbs"` – a Polya-Gamma Gibbs sampler over the exact posterior (the spOccupancy PGOcc engine): site-level detection, an areal `icar()` field is accepted; a `temporal()` / `re()` / `latent()` / `svc()` term errors with a pointer.
- `method = "nested_laplace"` – multi-block nested Laplace: areal, temporal and random-effect hyperparameters are integrated on an outer grid, and `predict()` interpolates all-NA detection sites from the field.
- `method = "nested_laplace_sla"` – the nested route with skew-corrected marginals.
- `method = "nuts"` – the unified C++ NUTS sampler over the structured terms (spatial, temporal, `re()`, `svc()`, `latent()`), reporting `Rhat` / `ESS`.

Examples

```
f <- occu()
f
```

`occu_aggregation_scan` *Suggest spatial and temporal aggregation for an identifiable occupancy model*

Description

Scans candidate spatial cell sizes and yearly clusterings of long-format, typically single-visit plot data and scores each by how well a single-season occupancy model separates occupancy (`psi`) from detection (`p`). Replication is manufactured by pooling plots into a spatial cell (assumes plots in a cell share `psi`) and years into a contiguous block (assumes occupancy is closed across the block); an occasion is any record in a (cell, block) bucket, so single-visit plots, repeat-visited plots, and the no-pooling limit all flow through one code path.

Usage

```
occu_aggregation_scan(
  data,
  response,
  coords,
  year,
  plot = NULL,
  cell.sizes = NULL,
  block.lengths = NULL,
  score = c("info", "count"),
  family = c("occupancy"),
  control = list()
)
```

Arguments

<code>data</code>	Long data.frame, one row per plot-year record (or plot-year-visit).
<code>response</code>	Name of the 0/1 detection column.
<code>coords</code>	Length-2 character vector naming the x and y coordinate columns.
<code>year</code>	Name of the integer year column.
<code>plot</code>	Optional name of the plot-id column; enables the within-cell homogeneity proxy.
<code>cell.sizes</code>	Numeric vector of candidate cell edge lengths. NULL auto-proposes a geometric ladder from the nearest-neighbour spacing to half the extent.
<code>block.lengths</code>	Integer vector of candidate contiguous block lengths (in years). NULL runs mean-shift changepoint segmentation of the per-year naive occupancy series and uses the resulting blocks as the single temporal candidate.
<code>score</code>	"info" (curvature-based, default) or "count" (structural).
<code>family</code>	Observation family. Only "occupancy" is supported; the per-family constant-model scorer is the extension point for others.
<code>control</code>	List of tuning knobs: <code>n_cell_sizes</code> (auto ladder length, 6), <code>nn_sample</code> (NN-spacing sample cap, 1500), <code>min_block</code> (min years per segment, 1), <code>cp_penalty</code> (explicit changepoint penalty, else auto), <code>cp_penalty_mult</code> (auto-penalty scale, 1), <code>max_condition</code> (conditioning cap for the identifiable flag, 1e6), <code>optim_control</code> (passed to <code>optim</code>).

Details

Two scoring modes:

- "info" fits the constant (intercept-only) occupancy model per candidate and reports the posterior SEs of ψ_i/p and the smallest eigenvalue and condition number of the 2×2 ($\logit \psi_i$, $\logit p$) observed information. A confounding ridge shows up directly as a near-zero eigenvalue.
- "count" checks only the structural necessary conditions (units with two or more occasions, detected units, a non-degenerate naive detection rate).

The recommended candidate is the **least-pooling** identifiable one (smallest mean occasions per unit, ties broken by largest information eigenvalue): pool only as much as identifiability requires. When no candidate is identifiable the data cannot support a standard occupancy model and a Royle-Nichols or count model is the alternative.

Value

A `tobs_aggregation_scan` object: `candidates` (scored data.frame), `recommended` (chosen row or NULL), `segmentation` (auto changepoint blocks, if used), and scan metadata.

Examples

```
set.seed(1)
d <- data.frame(east = runif(200), north = runif(200),
               year = sample(2001:2006, 200, replace = TRUE),
               detected = rbinom(200, 1, 0.3))
scan <- occu_aggregation_scan(d, response = "detected",
```

```

                                coords = c("east", "north"), year = "year",
                                cell.sizes = c(0.2, 0.4),
                                block.lengths = c(2, 3))

scan

```

occu_categorical	<i>Presence + nominal class hurdle family</i>
------------------	---

Description

A hurdle for a response that is either absent or, when present, one of K **nominal (unordered)** classes – a colour morph, microhabitat / substrate use, or a cryptic-species / classifier label observed given the organism is present. The presence/absence part is a Bernoulli arm; the class given present is a baseline-category multinomial logit (the last class is the baseline). The two arms factorise the likelihood exactly:

$$P(y = 0) = 1 - \psi, \quad P(y = k) = \psi p_k,$$

with $p = \text{softmax}(X\beta_{\text{class}})$.

Usage

```
occu_categorical(classes = NULL)
```

Arguments

classes	optional character vector of class labels (length K), used only to name the coefficient blocks; when <code>NULL</code> , K is taken from <code>max(y)</code> and the classes are labelled <code>1..K</code> .
---------	--

Details

This is the categorical counterpart of `cover()` (presence + a magnitude): here the positive part is an unordered class, so it uses a multinomial logit rather than beta / lognormal. For an *ordered* class response (Braun-Blanquet cover bands) use `cover(response = "ordinal")`, which exploits the ordering; this family is for classes with no ordering.

Value

A `tobs_family` object.

Response

y is a length- N integer vector in $0..K$: 0 is absent, k is class k . It may sit on the formula left-hand side ($y \sim \dots$), so y can be dropped. The formula predictors are shared by both arms.

Engines

Non-spatial Laplace (method = "laplace"), the only route. The multinomial math is the FD-validated tulpa kernel (multinomial_logit.h); the non-spatial fit is the vectorised R Newton over the same closed forms. Spatial fields / NUTS (the native multi-process likelihood) and the latent-class *misclassification* variant (the K-class generalisation of fp_occu(), a confusion matrix on the observed label) are documented follow-ups.

See Also

cover() (presence + magnitude), fp_occu() (two-state false-positive detection).

Examples

```
f <- occu_categorical(classes = c("red", "green", "blue"))
f
```

occu_cover	<i>Joint occupancy-detection + cover hurdle family</i>
------------	--

Description

Cell-level latent presence psi, per-visit binomial detection p, and per-visit positive cover f_pos (beta or lognormal) on a third linear predictor. The N-mixture analogue for vegetation cover: where abun() couples occupancy to counts via Royle’s binomial-N marginal, occu_cover() couples it to a positive-cover observation at each detected visit. The latent presence z marginalises out in closed form (two states), so the marginal log-likelihood is exact.

Usage

```
occu_cover(
  response = c("beta", "lognormal", "gaussian"),
  cover.aggregate = NULL
)
```

Arguments

- response likelihood for the positive cover arm. "beta" (cover in (0, 1)), "lognormal" (log-cover Gaussian), or "gaussian" (an identity-Gaussian magnitude on a real, unbounded scale – the delta-normal hurdle; not for cover fractions, which stay on "beta"/"lognormal").
- cover.aggregate how the cover arm collapses per occupancy unit on the shared-field spatial path: "mean", "median", "latent" (a per-unit cover random effect integrated out), or "none" (per-visit). NULL (default) is "mean" on the spatial path and "none" on the non-spatial path. See the *Cell-aggregated cover* section.

Details

Per-cell likelihood:

```
any_det_i : L_i = psi_i * prod_j h_ij
no_det_i  : L_i = psi_i * prod_j (1 - p_ij) + (1 - psi_i)

h_ij = (1 - p_ij) * 1{y_ij = 0}
      + p_ij      * f_pos(y_pos_ij; eta_pos_ij, ...) * 1{y_ij = 1}
```

A detected visit ($y_{ij} = 1$) with a missing cover ($y_{pos_ij} = NA$) keeps its detection term but drops the f_pos factor: cover is taken missing-at-random, so the cover likelihood runs over the detected visits with an observed cover.

Reduces to `occu()` when the cover arm is degenerate, and to the plot-level cover hurdle (`cover()`) when $J = 1$ and detection is perfect.

Value

A `tobs_family` object.

Engines

The non-spatial fit is a direct Laplace approximation (`method = "laplace"`) or a NUTS sampler over the same exact two-state marginal (`method = "nuts"`, beta or lognormal cover), giving calibrated intervals and a per-draw pointwise likelihood for WAIC / LOO. A shared areal field across the occupancy and cover arms is the `method = "nested_laplace"` path (a structured `icar()` / `bym2()` term on the ψ_i formula); a structured term on a non-spatial route errors from the dispatcher with a pointer to it.

`method = "nuts"` also samples the coupled areal field(s) (`icar()` / `bym2()` / `car_proper()`), written either as areal terms or as the bar `spatial(~ 1 || cell, graph = adj)`, TOGETHER WITH their hyperparameters: the field SD, the mixing (`bym2`) or spatial-correlation (`car_proper`) parameter, and - where the formula asks for one - the cover-arm copy amplitude. Each is bounded to the span the `nested_laplace` path's outer quadrature integrates that axis over, in that grid's own coordinate, and the same `control$sigma.grid / alpha.grid / rho.car.grid` knobs set it on both routes - so the sampler is an independent reference for the hyperparameter layer rather than a fit conditioned on that layer's point estimate. The measure over that span is the one the outer grid declares for the axis: flat in the grid's coordinate for the field SD and the mixing parameter, and for the copy amplitude the penalized-complexity Exponential slab, with `control$copy.slab = "flat"` asking for the flat alternative the grid also accepts. The grid weighs that slab against a point mass at $\alpha = 0$ ("no coupling"), which a gradient sampler cannot visit, so the sampled copy amplitude is the posterior conditional on a coupled field; the point mass's share of the prior is read on the `nested_laplace` route. `fit$nuts$sampled_hyper` and `fit$nuts$fixed_hyper` name which is which per fit (`icar` pins ρ at 1: the intrinsic precision has no mixing parameter; a field with no `share()` pins the copy amplitude at 0), `fit$hyper_draws` carries their posterior alongside `field_sd`, the geometric-mean marginal SD the field implies. `control$fixed.hyper = TRUE` conditions on the warm nested-Laplace estimate instead.

The grid-integrated `nested_laplace` path reports `field_sd` too (`fit$spatial$field_sd_mean / field_sd_sd`, alongside `sigma_mean`), in the SAME geo-mean-marginal-SD convention as the

NUTS path's `field_sd`. It is a fixed multiple of `sigma` (not an independent grid axis), so it is NOT in `fit$means / fit$vcov` – folding it in there would make the joint parameter `vcov` exactly singular. Do not compare fits, or a fit against `simulate_occu_cover()`'s `sigma`, by reading the raw `sigma` on this path: it is the field's amplitude against the unscaled intrinsic ICAR precision $Q = D - W$, which differs from `field_sd` by `sqrt(scale_q)`, a graph-size-dependent factor (about 2.1 for a 30-node chain graph). `field_sd` is the number comparable across fits and to a simulation truth; `sigma` is the raw amplitude the engine's grid axis is spelled in.

A spatially-varying coefficient - a *weighted* areal term beside the unweighted intercept field, or a bar column `spatial(~ 1 + year || cell, graph = adj)` - is sampled as a SECOND field block, with its own whitened surface and its own (SD, mixing, copy amplitude) coordinates: two fields share no hyperparameter. The surfaces are `fit$spatial_field` and `fit$trend_field / fit$trend_fields` (named by the weight column), and each block's hypers carry that block's suffix in `fit$hyper_draws` and in `fit$nuts$sampled_hyper` (`sigma_trend`, `alpha_trend`, indexed when there are several). Each field's amplitude is addressed on its own through `share(spatial(), terms = list(...))` – with stated nodes, or `grid(n = )` for the engine's axis read more finely – or through `control$alpha.grid[.trend] / control$alpha.n[.trend]`, the same knobs the grid-integrated route reads. A second field multiplies the warm fit's outer grid, so a defaulted axis is thinned to three nodes over the same span - the sampled prior is unchanged, the warm fit stays affordable. A correlated bar (`|`, one free-Sigma MCAR block across the fields) stays on the grid-integrated `nested_laplace` path; the community spatial-factor route is `ms_occu_cover()`.

On the shared-field `nested_laplace` path the field-coupled occupancy slope Wald interval is mildly anti-conservative at small N (the grid-integrated Laplace under-disperses that coefficient; the pooled coverage across the three arms stays near nominal). The non-spatial nuts path gives fully calibrated occupancy intervals.

What a bare areal term means for the cover arm

A term's process is the formula it sits in. An areal term written in the occurrence formula therefore loads on the OCCURRENCE arm alone: the cover arm sees no field, and the copy amplitude `alpha` is 0. This holds under `method = "nested_laplace"` and `method = "nuts"` alike, so the same input fits the same model on either engine.

To carry the occurrence field onto the cover arm as well, write the copy in the positive formula:

```
positive = ~ pos_cov + share(spatial()) # amplitude estimated
positive = ~ pos_cov + share(spatial(), alpha = grid(c(...))) # over given nodes
positive = ~ pos_cov + share(spatial(), alpha = grid(n = 9)) # engine axis, finer
positive = ~ pos_cov + share(spatial(), alpha = 0.5) # fixed
positive = ~ pos_cov + share(spatial(), prior = list("pc.prec", c(4, 0.01)))
```

The amplitude is integrated over that axis on the `nested_laplace` path and sampled over its span on the nuts path; a scalar `alpha` = pins it on both. `prior` = regularizes the coefficient itself, in the joint driver's `list(<family>, <params>)` shape (see `prior_alpha` in `tulpa::tulpa_nested_laplace_joint()`); one reaches the engine per fit, so a fit copying both an intercept and a trend block is refused it rather than given it on the first.

`control$alpha.grid[.trend]` and `control$alpha.n[.trend]` are the WIRE FORMAT these compile into, not user surface: a fit that sets one is refused and told which formula form to write. `control$prior.alpha` is still the lower-level spelling of `prior` =; set it in one place, not both.

Without a `share()` the occurrence field is NOT carried onto the cover arm – there is no implicit coupling – so the amplitude is not a parameter of the model, and neither engine reports one. Under `nested_laplace` `alpha` is absent from `coef()`, `vcov()` and the `n_params` count; under `nuts` it is absent from `fit$nuts$sampled_hyper`, `fit$nuts$fixed_hyper` and the `fit$hyper_draws` columns. `predict()` returns a cover arm the occurrence field does not enter. Add `share(spatial())` to couple the two arms.

The axis the amplitude rides carries prior structure – a point mass at `alpha = 0` ("no coupling") and a log-spaced slab above it – so it is set in one of two ways. `share(alpha = grid(...))` STATES its nodes, and with them that structure. `share(alpha = grid(n = 9))` states a RESOLUTION: the engine re-reads its own axis with that many slab nodes, point mass and slab bounds unchanged, so sharpening the axis never restates its structure. `terms =` gives either, per block. The resolution is the only way to raise this axis, because it does not densify when the donor `control$sigma.grid` does: on an informative data set the outer grid's quadrature effective sample size saturates on the copy amplitude while every other axis tracks the request (measured engine-side, `NOTES_measurements.md`). One block takes one of the two; giving both is an error. `control$alpha.grid[.trend]` and `control$alpha.n[.trend]` are the wire format these compile into and are refused as user input.

Set it when the fit's hyperparameter intervals are reported. On the coupled SBC fixture the declared resolution leaves the field SD miscalibrated once the data are informative – at 10 visits per site its uniformity p-value is $9.1e-05$, against 0.17 at 3 visits – and `control$alpha.n = 21` returns every scored quantity to nominal at both. Thirteen slab nodes do not (the field SD still reads $4.5e-03$), and the price is a 2.3-3x longer fit, since the outer grid is a tensor. Point estimates are not affected the way the intervals are; the measurement is in `NOTES_measurements.md`.

Coupled fields and spatially-varying trends

The spatial engines (`method = "nested_laplace"`, default `control$engine = "joint"`, and `method = "nuts"`) share one areal field (the cell intercept, an unweighted `icar()` / `bym2()` term) across the occupancy and cover arms, when the positive formula copies it. ADDITIONAL coupled fields - spatially-varying coefficients, e.g. a temporal trend - are added as *weighted* areal terms in the formula:

```
~ elev + icar(graph = adj) + icar(graph = adj, weight = year)
```

Each weighted term `icar(graph, weight = col)` is a second field whose contribution to a predictor row is `weight_i * z[cell_i]`. All such fields share the same graph; a `share()` carries one onto the cover arm with its own scale (`alpha` for the intercept field, `alpha_trend` for a trend field), integrated over the outer grid, and a field the positive formula does not copy stays on occurrence. `share(spatial(), terms = list(...))` gives a per-block amplitude and must address every block. The intercept field is reported in `fit$spatial_field`, trend fields in `fit$trend_field` / `fit$trend_fields`. The trend coupling grid defaults to the intercept block's amplitude; give the trend its own with `share(spatial(), terms = list(intercept = ..., trend = ...))`. Under `method = "nuts"` each field is its own sampled block, with the scales sampled over those same spans rather than integrated over the grid.

A single trend field may also be requested out-of-band with `control = list(trend = list(weight = "<col>"))`, naming a numeric per-cell covariate in the cell data; this is the equivalent of one weighted formula term. Specify the trend field one way or the other, not both.

Sites larger than cells (group.var)

By default each site (one row of y / data, one latent occupancy state) is its own field node, so the graph must have one node per site. Passing `group.var = "<col>"` to the `icar()` / `bym2()` term maps each site to a field node named by that integer column, so several sites can share one node. The field then stays length `n_cells` (the graph) while occupancy, detection, and cover run over `n_sites`. The motivating layout is a site = cell x time-period: plots in a cell-period are the detection replicates, occupancy is per cell-period, and a per-site time weight on a coupled trend field (`icar(graph, weight = time, group.var = "cell")`) gives a detection-corrected occupancy trend on a shared cell field.

Per-group random intercept on the shared-field path (`re()` / `(1 | g)`)

On the spatial `nested_laplace` path a single random INTERCEPT term on the `psi` formula – `re(g)` or the `(1 | g)` bar – layers a per-group occupancy offset on top of the shared field. It joins the joint fit as one iid prior block whose latent rides the occupancy arm only; its variance integrates on the outer grid alongside the field `sigma / alpha` and is reported as the `sigma_re` hyperparameter, with the per-group BLUPs in `fit$re` and via `ranef()`. A default `re.sigma.grid` (log-spaced) sets the grid; pass `control$re.sigma.grid` to override. Scope: one random-intercept term (a scalar per group maps onto the one iid block); a random slope, a correlated multi-coefficient block, or an RE without a shared field is rejected (the joint engine integrates every variance component on its grid, so multiple variance components do not scale – richer RE is the non-spatial cover-hurdle EM's route). This is the single-species analogue of the community spatial `occu_cover` (`ms_occu_cover()`) and the `tulpaObs` consumer of `tulpa`'s field + per-group RE engine composition.

A bar is therefore always a random effect, never a spatial field, even when the grouping factor names the areal graph nodes. The engine's inline-MCAR call `tulpa::spatial(graph, ~ 1 + x | cell)` reads `| cell` as a separable spatial field, but the same spelling in an `occu_cover()` formula is an IID random effect on `cell`. For a spatial field on the cells use an areal term – `icar(graph = adj, group.var = "cell")` (plus `icar(graph = adj, weight = x, group.var = "cell")` for a spatially-varying trend) – not a bar. When a formula carries a bar whose grouping factor is also an areal term's `group.var`, `occu_cover()` emits a one-time message noting the bar is fitted as a random effect; suppress it with `base::suppressMessages()`.

Independent field on the cover arm (placement)

A `share(spatial())` in the positive formula shares the occupancy field: the cover arm sees it as `alpha * (occupancy field)`, the coregionalization copy on the outer (`sigma`, `alpha`) grid. When the cover trend is spatially structured but is not a scalar multiple of the occupancy field, `alpha` collapses toward 0 and the cover arm inherits no field, so per-cell conditional cover (and its change over time, `delta_cover_cond`) comes out flat. A `spatial()` bar placed in the positive formula adds an INDEPENDENT, non-copied areal field on the cover arm alone:

```
tobs(occurrence = ~ x + icar(graph = adj, group.var = "cell"),
      detection   = ~ 1,
      positive    = ~ time + spatial(~ 1 + time || cell, graph = adj),
      data        = cell_dat, y = y, y.pos = y_pos,
      family      = occu_cover("lognormal"), method = "nested_laplace")
```

The occupancy field still drives `psi` and, via the `alpha` copy, `delta_cover_exp`; the independent cover field carries a cover-specific structure the `alpha` copy cannot express. Its intercept + coefficient

columns become separate ICAR blocks on the cover arm, each with its own precision integrated on the outer grid and reported as `sigma_pos_field(intercept) / sigma_pos_field_<col>` (a covariate column). The field SD grid defaults to `control$sigma.grid`; set `control$sigma.grid.pos.field` to integrate it over its own (usually coarser) grid, which keeps the added axis from multiplying the outer-grid cost. The per-cell field posterior is in `fit$pos_field / fit$pos_field_table` (and `fit$pos_field_tables` per column). Scope: it composes with the shared occupancy field but uses per-visit cover (`cover.aggregate = "none"`) and does not combine with the correlated | MCAR field, the latent cover RE, or the batched fused path; like the shared field it is fitted as ICAR (bym2/car read as ICAR). A static intercept cover field is only weakly identified against the occupancy field's alpha copy (both are per-cell cover offsets); a time-weighted trend cover field is identified separately and is what makes `delta_cover_cond` spatially varying.

The same placement works on the detection arm: a spatial-field term in the detection formula (`detection = ~ 1 + spatial(~ 0 + time || cell, graph = adj)`) fits an independent field on the detection predictor, for a spatially-structured detection probability. Its SD is reported as `sigma_p_field / sigma_p_field_<col>`.

Checkpoint / resume

A full-field `occu_cover()` fit integrates over a large outer hyperparameter grid and can run for hours, so a reboot or OOM kill otherwise loses the whole run. `control$checkpoint = list(path = "fit.ckpt", resume = TRUE)` makes the joint engine append each completed grid cell to path; a `resume = TRUE` run loads the finished cells and solves only the remaining ones, reproducing the from-scratch fit. `resume = FALSE` starts a fresh file (any stale checkpoint at path is removed first). A torn tail from a killed write is discarded and re-solved, and a checkpoint written for different data or settings is rejected rather than resumed onto. Forwarded to [tulpa::tulpa_nested_laplace_joint\(\)](#).

Cell-aggregated cover (`cover.aggregate`)

On the shared-field spatial path the cover arm has one observation per detected visit, so a cell with many detected plots informs the shared field far more than the single occupancy observation for that cell; the field is then driven almost entirely by the cover arm and the detection-corrected occupancy surface flattens. `cover.aggregate` collapses the cover arm to a single response per occupancy unit (cell-period) so the two arms inform the field with comparable weight:

- "mean" – model the mean cover over the unit's detected visits;
- "median" – model the median cover;
- "latent" – model a per-unit cover random effect $u_i \sim N(0, \sigma_u^2)$ shared across the unit's detected visits and integrated out per unit, so the cover arm contributes one marginal observation per unit while keeping every detected visit (the principled counterpart of mean / median aggregation). The lognormal arm integrates in closed form (compound-symmetry); the beta arm uses adaptive Gauss-Hermite quadrature (`control$n.quad`, default 15). The within-unit dispersion is pre-fit from the within-unit spread and held fixed; σ_u is integrated on the outer grid (`control$sigma.u.grid`).
- "none" – the per-visit cover arm (one cover observation per detected visit).

NULL (the default) selects "mean" on the shared-field spatial nested_laplace path and "none" on the non-spatial laplace path (which has no shared field to over-weight). Aggregation needs a cell-level positive design: a visit-level covariate in the positive formula cannot be collapsed to

one value per cell and errors. Aggregation is currently wired on the spatial path only; requesting "mean" / "median" on the non-spatial laplace fit errors rather than silently using per-visit cover.

Random effects on the observation arms

Under `method = "nested_laplace"` (shared-field spatial) and `method = "nuts"` (non-spatial, which samples the group SD), random intercepts may be written on the **detection** or **positive-cover** formula with the usual `lme4` bar or `re()` spelling, e.g. `detection = ~ effort + (1 | habitat)`. The grouping is per visit – one code per (`site`, `visit`) entry, distinct from the per-site occupancy-arm grouping – so a many-level categorical visit covariate (an EUNIS habitat class, an observer) enters as a partially pooled random intercept. **Crossed** (`((1 | habitat) + (1 | observer))`) and **nested** (`((1 | region/site)`, which expands to `re(region) + re(region:site)`) groupings are supported: each term joins the joint fit as its own iid latent block. Each block's variance integrates on the outer grid (`sigma_re_p / sigma_re_pos` for a lone term on the detection / cover arm, suffixed by the grouping variable – `sigma_re_p_habitat` – when several terms share an arm; tune with `control$re.sigma.grid.p / re.sigma.grid.pos`). The per-group BLUPs are reported in `fit$re` (one entry per term, keyed by arm or "`<arm>:<var>`") and via `ranef()`. `predict()` sums every term's BLUP offset on the predicted arm when `newdata` carries the grouping columns and `type = "detection"` (or "`occurrence`" / "`cover_cond`"); an unseen / held-out level shrinks that term to the population mean. Several crossed terms multiply the outer grid, so set `control$integration = "ccd"` (and/or coarsen the RE grids) at scale.

Random **slopes** are also supported (tulpa engine $\geq 0.0.39$): an uncorrelated slope (`((x | g), (0 + x | g))`) is one per-row weighted iid block per coefficient, and a correlated slope (`((1 + x | g))`) one multivariate free-Sigma iid block. A slope term's `fit$re` entry carries an `[n_groups x n_coefs]` BLUP matrix, a per-coefficient sigma, and (correlated) a cor matrix; the hyperparameters are `sigma_re_p_<coef>` and `cor_re_p_<ci>_<cj>`, marginalized over the grid. `predict()` weights each coefficient by its covariate column in `newdata` (intercept = 1). Each slope covariate is **standardized** to unit SD before fitting (mirroring the fixed-effect design), so the variance grid is scale-invariant – the reported slope BLUPs and sigma are back-transformed to the covariate's natural units (correlation is scale-free). A correlated slope adds $p(p+1)/2$ Sigma axes to the outer grid, so its free-Sigma grid uses a compact principled default (symmetric correlation nodes including 0); widen it with `control$re.logchol.grid.p / re.logchol.grid.pos`.

The positive-cover RE needs per-visit cover (`cover.aggregate = "none"`). As with the occupancy-arm RE, each grid-integrated variance carries the binary / small-cluster inner-Laplace attenuation and is a lower bound on the truth; the BLUPs recover the per-group structure.

`method = "nuts"` samples an observation-arm random **intercept** instead of integrating it on a grid: each grouping factor is a non-centered block (`b_g = sigma_re z_g, z_g ~ N(0, 1)`) whose group SD is a coordinate of the sampled vector under a $N(0, 1.5^2)$ prior on `log sigma_re` – the same weakly informative width the other observation-family samplers use, and the reason the sampled SD carries no small-cluster attenuation to correct for. Crossed and nested groupings are simply several blocks, on either or both observation arms. `fit$re` keys and `ranef()` match the `nested_laplace` path; each term additionally reports `sigma_median` and the per-draw `sigma_draws`, the summaries to quote for a right-skewed variance component, with per-SD convergence in `fit$nuts$re_sigma_rhat / re_sigma_ess`. The sampled coefficient surface (`coef()`, `vcov()`, `fit$draws`) is unchanged. Still on `nested_laplace` only: a random slope, and an observation-arm RE composed with the coupled areal field. `waic()` / `cpo()` on a sampled fit read the coefficient draws, so they score the RE at zero – the same limitation the sampled coupled field has.

See Also

[occu\(\)](#) (no cover), [cover\(\)](#) (plot-level hurdle, no detection), [abun\(\)](#) (counts not cover).

Examples

```
N <- 120; J <- 4
sim <- simulate_occu_cover(N = N, J = J, n.occ.covs = 1, n.det.covs = 1,
                           n.pos.covs = 1, positive = "beta", seed = 1)
long <- data.frame(site_id = rep(seq_len(N), each = J),
                  visit    = rep(seq_len(J), times = N),
                  y        = as.vector(t(sim$y)),
                  det_cov1 = sim$visit_data$det_cov1,
                  pos_cov1 = sim$visit_data$pos_cov1)
od <- tobs_data(long, y = "y", site = "site_id", visit = "visit",
               det.covs = c("det_cov1", "pos_cov1"))
cell_dat <- cbind(data.frame(site_id = seq_len(N)), sim$data)
y_pos <- sim$y_pos; y_pos[is.na(y_pos)] <- 0
fit <- tobs(~ occ_cov1, data = cell_dat, family = occu_cover("beta"),
           detection = ~ det_cov1, positive = ~ pos_cov1,
           y = od$y, y.pos = y_pos, visits = od$det.covs, method = "laplace")
summary(fit)
```

occu_cover_inputs

*Build occu_cover() model inputs from a long, plot-level frame***Description**

Assemble the paired occurrence / cover response arms, the visit-level covariate grid, and the site-level design frame that an [occu_cover\(\)](#) fit consumes, from one long (one row per site-visit / plot) data frame. This is the same construction [tobs\(\)](#) runs internally when a single [occu_cover\(\)](#) fit is handed a long frame; call it directly when you want to inspect the arms before fitting, or pass them on as `y = , y.pos = , visits = , data =` to [tobs\(\)](#).

The occurrence and cover arms are pivoted onto one shared (sites, visits) grid, so they align row-for-row, and the site-level design is the first row per site. The cover arm is positive-only (meaningful where the species is detected): in the dense grid absent / unsampled cover cells are set to 0, and in the compact (ragged) carrier they are NA in the values and never read by the joint engine.

Usage

```
occu_cover_inputs(
  data,
  site,
  visit,
  response,
  y.pos,
  occ.covs = NULL,
  det.covs = NULL,
```

```

  coords = NULL,
  compact = TRUE,
  positive = "beta"
)

```

Arguments

<code>data</code>	A long / plot-level data frame: one row per site-visit.
<code>site, visit, response, y.pos</code>	Column names: the site identifier, the visit / replicate, the 0/1 detection response, and the continuous cover response (used only where <code>response == 1</code>). The accepted range of <code>y.pos</code> follows <code>positive</code> .
<code>occ.covs</code>	Optional character vector of site-level covariate columns. The default (NULL) keeps the full first-row-per-site frame and lets the occurrence / detection / positive formulas select what they reference.
<code>det.covs</code>	Optional character vector of visit-level covariate columns.
<code>coords</code>	Optional length-2 character vector of coordinate columns.
<code>compact</code>	Logical (default TRUE). Build the compact (ragged) arms the joint nested-Laplace <code>occu_cover()</code> engine consumes (no per-site visit cap), or the dense [<code>n_sites</code> x <code>max_visits</code>] grid.
<code>positive</code>	The cover-arm distribution, matching <code>occu_cover()</code> : "beta" (default) stores <code>y.pos</code> as a proportion in $[0, 1]$ (<code>tobs_data()</code> type "cover"); "lognormal" / "gamma" store it as a positive real $(0, \text{Inf})$ (type "positive", validated non-negative with no upper bound). Pick the value you pass to <code>occu_cover(response =)</code> .

Value

A list with `y`, `y_pos`, `visits`, `site_data`, `coords`, `n_sites`, `max_visits`, `n_visits`, and `compact`.

See Also

[tobs\(\)](#), [tobs_data\(\)](#).

Examples

```

sim <- simulate_occu_cover(N = 20, J = 3, seed = 1)
long <- data.frame(site = rep(1:20, each = 3), visit = rep(1:3, 20),
  occur = as.vector(t(sim$y)),
  cover = as.vector(t(ifelse(is.na(sim$y_pos), 0,
    sim$y_pos))),
  det_cov1 = sim$visit_data$det_cov1,
  occ_cov1 = rep(sim$data$occ_cov1, each = 3))
inp <- occu_cover_inputs(long, site = "site", visit = "visit",
  response = "occur", y.pos = "cover",
  det.covs = "det_cov1", compact = FALSE,
  positive = "lognormal")

dim(inp$y)

```

occu_multi

Multi-species co-occurrence occupancy family

Description

Joint occupancy of S species where the occupancy state z in $\{0, 1\}^S$ follows a log-linear model with first-order (per species) and second-order (per species pair) natural parameters (Rota et al. 2016; **unmarked** occuMulti). The second-order parameters are the species interactions (positive = co-occur more than independent; negative = avoid). The state formula is the shared occupancy covariate design (each natural parameter carries its own coefficients); detection the shared site-level per-species detection design. The latent state integrates out by enumerating the 2^S states, so the exact marginal is maximised with an observed-information vcov.

Usage

```
occu_multi()
```

Details

y is a length- S list of $N \times J$ 0/1/NA detection matrices (or a 3D [sites $\times$ visits $\times$ species] array); species names the arms.

Value

A `tobs_family` object for `tobs()`.

Engines

- `method = "laplace"` – the enumerated 2^S -state marginal maximised (BFGS) with an observed-information covariance; the only route (no field, no sampler).

Examples

```
sim <- simulate_occu_multi(S = 2, N = 300, seed = 1)
fit <- tobs(~ scov1, data = sim$data, family = occu_multi(),
           detection = ~ 1, y = sim$y, species = sim$species,
           control = list(verbose = FALSE))
coef(fit)
```

occu_multiscale_cover *Three-level (multiscale) occupancy + cover hurdle family*

Description

A cell-level occupancy gate, a plot-level availability gate, per-visit detection, and the cover hurdle, for data where a site's "visits" are spatially distinct plots aggregated into a (cell, period) rather than temporal revisits (the EVA / MOTIVATE vegetation layout; Nichols et al. 2008; Mordecai et al. 2011). `occu_cover()` treats plots as detection replicates of one occupancy state, which on spatial subunits conflates within-cell prevalence into the detection arm (Kendall & White 2009); this family separates them with an explicit middle level:

Usage

```
occu_multiscale_cover(response = c("beta", "lognormal", "gaussian"))
```

Arguments

response likelihood for the positive cover arm. "beta" (cover in (0, 1)), "lognormal" (log-cover Gaussian), or "gaussian" (an identity-link Gaussian magnitude, the delta-normal hurdle; for a pre-transformed / unbounded positive response, not raw cover fractions).

Details

```
z_c      ~ Bernoulli(psi_c)           # cell / range occupancy
a_cj | z=1 ~ Bernoulli(theta_cj)      # plot availability / use
y_cjv|a=1 ~ Bernoulli(p_cjv)         # detection
cover|y=1 ~ f_pos(.; eta_pos, disp)   # cover hurdle (beta / lognormal)
```

Both z (over cells) and a (over plots) marginalize in closed form (two states each), so the joint marginal log-likelihood is exact and reuses the same nested-Laplace cell-coupling machinery as `occu_cover()`.

Value

A `tobs_family` object.

Inputs

$y / y.pos$ are $[n_plots \times max_visits]$ matrices (plots are the rows, the availability units; visits the columns). The state-process formula is the cell-level occupancy predictor and MUST carry an areal field naming the per-plot cell column, `icar(graph = adj, group.var = "cell")`. `availability = ~ ...` is the plot-level theta predictor (default ~ 1); `detection` the per-visit p predictor; `positive = ~ ...` the cover predictor (default the detection formula). $y.pos$ is read only where $y == 1$.

A detected visit ($y_cjv = 1$) with a missing cover ($y_pos_cjv = NA$) keeps its detection term but drops the f_pos factor: cover is taken missing-at-random, so the cover likelihood runs over the detected visits with an observed cover. This is the rule `occu_cover()` applies, and it holds on every engine below.

Identifiability

The availability (θ) and detection (p) levels separate only with replication WITHIN a plot. Single releves supply none, so a plain fit identifies ψ (cell) and the product $\theta * p$ (plot) – it reduces to `occu_cover()` with $p := \theta * p$. Within-plot temporal replication (e.g. a resurvey of the same plot in a later period) makes the third level estimable.

Engines

Three engines. `method = "nested_laplace"` carries a single shared areal field coupled across the occupancy (σ) and cover ($\alpha * \sigma$) arms, integrated over the outer (σ , α) grid. `method = "laplace"` and `method = "nuts"` are the non-spatial path (iid cells, no field): the exact three-level marginal (z over cells, a over plots both summed in closed form) optimised directly ("laplace", a Gaussian observed-Fisher posterior) or sampled ("nuts", the exact coefficient posterior with calibrated intervals and WAIC / LOO). Cells are declared the same way on every path, via an `icar(graph = adj, group.var = "<cell>")` term (the graph drives the field under "nested_laplace" and supplies only the plot $\rightarrow$ cell map under "laplace" / "nuts"). On the "nested_laplace" path additional weighted areal terms in the ψ formula (`icar(graph = adj, group.var = "<cell>", w` add spatially-varying-coefficient trend fields, each coupled onto the cover arm with its own `alpha_trend`; the fitted fields are in `fit$trend_field / fit$trend_fields`. The coupled / trend field is not sampled, so `method = "nuts"` takes a single cell-declaring areal term only.

The coupling is written as `share(spatial())` in the positive formula, the same spelling `occu_cover()` takes; it needs a field to copy, so it is accepted under `method = "nested_laplace"` and refused on the two non-spatial engines, which fix the field at 0. Omitting it decouples the arms: the block is pinned at $\alpha = 0$ and the field rides occupancy alone, the same meaning the absence has on `occu_cover()`. There is no implicit default coupling.

The copy amplitude's axis carries prior structure – a point mass at $\alpha = 0$ and a log-spaced slab above it – so `share(alpha = grid(...))` STATES its nodes, while `share(alpha = grid(n = ))` states a RESOLUTION: the engine re-reads its own axis with that many slab nodes, point mass and bounds unchanged. `terms =` gives either, per block; one block takes one of the two. `control$alpha.grid[.trend] / control$alpha.n[.trend]` are the wire format these compile into and are refused as user input.

See Also

`occu_cover()` (two-level), `cover()` (plot hurdle, no detection).

Examples

```
f <- occu_multiscale_cover(response = "beta")
f
```

Description

Constructs a prior specification consumed by `tobs()` when a Laplace method (`method = "laplace"` etc.) is used for the occupancy family. The penalty is applied as a quadratic term $+\sum((\beta_j - \mu_j)^2 / (2 * \sigma_j^2))$ added to the negative-log-posterior in each M-step submodel block by `tulpa::tulpa_laplace()` (via the per-block `beta_prior`).

Usage

```
occu_priors(
  p.intercept = list(mean = 0, sd = 1.5),
  p.slope = list(mean = 0, sd = 2.5),
  beta.occ.intercept = list(mean = 0, sd = 2),
  beta.occ.slope = list(mean = 0, sd = 5)
)
```

Arguments

`p.intercept` Prior on the detection intercept (logit scale). A list `list(mean = numeric(1), sd = numeric(1))`. Default `list(mean = 0, sd = 1.5)`.

`p.slope` Prior on detection slopes (every non-intercept detection coefficient). Default `list(mean = 0, sd = 2.5)`.

`beta.occ.intercept` Prior on the occupancy (ψ / ψ_1) intercept. Default `list(mean = 0, sd = 2)`.

`beta.occ.slope` Prior on occupancy slopes. Default `list(mean = 0, sd = 5)`.

Details

The defaults are weakly informative: they pull the detection intercept toward $p = 0.5$ strongly enough to break the ψ - p identifiability ridge at small J , but stay wide enough not to bias estimates at $N \geq 600$ with informative covariates. Setting any `sd` to `Inf` disables that component of the prior (same penalised objective, just $1/\text{Inf}^2 = 0$).

Value

An `occu_priors` object, ready to pass to `tobs(..., priors = ...)`.

Examples

```
# disable the detection-slope penalty
priors <- occu_priors(p.slope = list(mean = 0, sd = Inf))
priors

sim <- simulate_occu(N = 100, J = 3, n.occ.covs = 1, n.det.covs = 1,
  seed = 1)
fit <- tobs(~ occ_cov1, data = sim$data, family = occu(),
  detection = ~ det_cov1, y = sim$y, method = "laplace",
  priors = occu_priors(),
  control = list(verbose = FALSE, progress = FALSE))
```

```
coef(fit)
```

```
occu_ttd
```

```
Time-to-detection occupancy family
```

Description

Occupancy where a survey records the TIME to first detection rather than a 0/1 outcome (Garrard et al. 2008; **unmarked** occuTTD). At an occupied site the time-to-detection is exponential with rate λ (constant hazard); a survey of length `survey.length` that reaches its end without a detection is censored. An unoccupied site never detects. The state formula models $\text{logit } \psi$ (occupancy); detection models $\text{log } \lambda$ (the site-level detection rate). The latent occupancy state integrates out in closed form (two states), so the fit maximises the exact marginal with an observed-information `vcov`.

Usage

```
occu_ttd(survey.length = 1)
```

Arguments

`survey.length` Survey length $T_{\max}$ (the censoring time): a scalar, a length- N vector, or an $N \times J$ matrix. Default 1.

Details

`y` is an $N \times J$ matrix of detection times: a value in $(0, \text{survey.length})$ is a detection; a value $\geq \text{survey.length}$ is a non-detection (censored); NA is a survey not conducted.

Value

A `tobs_family` object for `tobs()`.

Engines

- `method = "laplace"` – the exact marginal maximised (BFGS) with an observed-information covariance; the only route (no field, no sampler).

Examples

```
sim <- simulate_occu_ttd(N = 200, J = 4, Tmax = 3, seed = 1)
fit <- tobs(~ psi_cov1, data = sim$data, family = occu_ttd(survey.length = 3),
  detection = ~ rate_cov1, y = sim$y, control = list(verbose = FALSE))
coef(fit)
```

peatland_occu	<i>Peatland occupancy survey (synthetic)</i>
---------------	--

Description

A single-season occupancy detection-history dataset for 120 wetland sites surveyed on 4 visits, drawn from a known generative model (MacKenzie et al. 2002). Occupancy increases with site wetness and decreases with elevation; per-visit detection increases with survey effort. Twelve detection cells are set to NA to mimic incomplete histories. Synthetic, not field data; the generative coefficients are in truth.

Usage

```
peatland_occu
```

Format

A list with components:

y 120 x 4 integer matrix of detections (0/1/NA).

occ.covs Data frame of site covariates: elevation, wetness (both standardised).

det.covs Named list with one 120 x 4 visit covariate matrix, effort (standardised).

coords 120 x 2 matrix of x / y coordinates in the unit square.

truth List of generative quantities: beta_psi, beta_p, the realised latent occupancy z, and per-site psi.

See Also

[occu\(\)](#), [tobs\(\)](#), [simulate_occu\(\)](#)

Examples

```
data(peatland_occu)
str(peatland_occu$y)

fit <- tobs(~ elevation + wetness, data = peatland_occu$occ.covs,
            family = occu(), detection = ~ effort,
            y = peatland_occu$y, visits = peatland_occu$det.covs)
coef(fit)
```

pit_residuals.tobs_fit

PIT residuals for a tobs_fit

Description

The `tulpa::pit_residuals()` method for `tobs_fit`: the family's own predictive CDF, with the randomisation step a discrete response needs.

Usage

```
## S3 method for class 'tobs_fit'
pit_residuals(
  object,
  n.samples = 250,
  nsim = NULL,
  seed = NULL,
  observed = NULL,
  ...
)
```

Arguments

<code>object</code>	A <code>tobs_fit</code> object.
<code>n.samples</code>	Number of posterior samples (default 250).
<code>nsim</code>	Alias for <code>n.samples</code> , and the name <code>tulpa::test_uniformity()</code> forwards under. An explicit <code>nsim</code> wins, so a uniformity test run at <code>nsim = 2000</code> draws 2000 samples rather than the default 250.
<code>seed</code>	Optional RNG seed for the posterior-draw selection, forwarded by <code>tulpa::test_uniformity()</code> . Set it and repeated calls return the same residuals; the caller's RNG stream is restored afterwards.
<code>observed</code>	Must be <code>NULL</code> , which is what <code>tulpa::test_uniformity()</code> forwards by default: a latent-state fit scores the response it was fitted to.
<code>...</code>	Must be empty; an unrecognised argument is an error.

Value

Numeric vector of PIT residuals.

See Also

`tulpa::test_uniformity()` for the uniformity test on the result.

```
plot.tobs_aggregation_scan
```

Plot an occupancy aggregation scan

Description

Heatmap of the identifiability score over the scanned (cell size x temporal block) grid: information smallest-eigenvalue for score = "info", fraction of replicated units for score = "count". Identifiable cells are outlined.

Usage

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

Arguments

x	A tobs_aggregation_scan object.
...	Ignored.

Value

Invisible NULL.

```
plot.tobs_data
```

Plot detection history patterns

Description

2x2 panel: detection frequency histogram, per-visit detection rates, visit completeness, and spatial detection map (if coordinates available).

Usage

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

Arguments

x	An tobs_data object.
...	Ignored.

Value

Invisible NULL.

ppc

*Posterior predictive check***Description**

The discrepancy-statistic check (the `spOccupancy` `ppcOcc` construction): latent occupancy is drawn from its full conditional, a detection replicate from the fitted model, and the two are compared through a discrepancy statistic to give a Bayesian p-value. This is a test, not a plot, so it is its own generic rather than a method on `tulpa::pp_check()`, which draws the graphical check.

Usage

```
ppc(object, ...)

## S3 method for class 'tobs_fit'
ppc(object, fit.stat = c("freeman-tukey", "chi-squared"), n.samples = 500, ...)
```

Arguments

<code>object</code>	A <code>tobs_fit</code> object.
<code>...</code>	Passed to methods.
<code>fit.stat</code>	"freeman-tukey" (default) or "chi-squared".
<code>n.samples</code>	Number of posterior samples (default 500).

Value

A list with `fit.y`, `fit.y.rep`, and `bayesian.p`.

Examples

```
sim <- simulate_occu(N = 60, J = 3, seed = 1)
fit <- tobs(~ occ_cov1, data = sim$data, family = occu(),
           detection = ~ det_cov1, y = sim$y, method = "laplace",
           control = list(verbose = FALSE))
ppc(fit, n.samples = 100)$bayesian.p
```

predict.cover_fit	<i>Predict cover from a cover_fit</i>
-------------------	---------------------------------------

Description

Occurrence probability is always $p = \text{plogis}(X * \text{beta_occ})$. The conditional positive cover μ depends on the positive-part family:

Usage

```
## S3 method for class 'cover_fit'
predict(
  object,
  newdata = NULL,
  type = NULL,
  include.RE = FALSE,
  times = NULL,
  time.col = NULL,
  level = 0.95,
  nsim = 1000L,
  draws = TRUE,
  weights = NULL,
  aggregate = FALSE,
  mc.tol = 0.05,
  nsim.max = 10000L,
  mc.floor = 0.001,
  ...
)
```

Arguments

object	A cover_fit.
newdata	A data frame of covariates matching the original formula. For the nested-Laplace fit, one row per spatial unit (or a cell column).
type	Separate-Laplace fit: one of "expected", "occupancy", "conditional". Nested-Laplace fit: "occurrence", "cover_cond", "cover_exp", or "change"; the separate-Laplace names "expected", "occupancy" and "conditional" map to "cover_exp", "occurrence" and "cover_cond".
include.RE	Ignored for the separate-Laplace fit (no spatial projection); the nested-Laplace fit always projects the shared field.
times, time.col, level, nsim, draws	Nested-Laplace fit only: times = c(t1, t2) and time.col drive the "change" map; level is the credible level, nsim the draw count (or "auto" for a trajectory, as in predict.tobs_fit()), draws whether to attach the draw matrices.

weights	Nested-Laplace fit only: pool several newdata rows into one prediction per cell, as in <code>predict.tobs_fit()</code> . A numeric vector (one weight per row) or the name of a newdata column.
aggregate	Nested-Laplace fit, type = "trajectory" only: TRUE averages each quantity over the prediction units per posterior draw and returns one row per time, as in <code>predict.tobs_fit()</code> .
mc.tol, nsim.max, mc.floor	Nested-Laplace fit, trajectory only: the Monte Carlo precision target, the draw cap and the SD floor the error is measured against, as in <code>predict.tobs_fit()</code> .
...	Unused.

Details

- positive = "lognormal": $\mu = \exp(\eta_{\text{pos}} + \sigma_{\text{pos}}^2 / 2)$ (lognormal back-transform on log-cover).
- positive = "beta": $\mu = \text{plogis}(\eta_{\text{pos}})$ (mean of the beta on the natural cover scale with logit link).

Expected cover is $E[y] = p * \mu$ under both positive parts.

The separate-Laplace fit (method = "laplace") returns a fixed-effects-only numeric vector. The nested-Laplace shared-field fit (method = "nested_laplace") instead projects the shared occupancy-cover field and returns a tobs_prediction of posterior draws – the same tidy / change contract as `predict.tobs_fit()` for `occu_cover()`: pass type = "change" with times = c(t1, t2) and time.col for a per-cell delta map. Each prediction unit is a row of newdata (or a cell column indexing the field cells), and every quantity is marginalized per draw over the grid-integrated joint posterior.

Value

Separate-Laplace fit: a numeric vector. Nested-Laplace fit: a tobs_prediction.

predict.occu_categorical_fit

Predict from an occu_categorical_fit

Description

Returns, per row of newdata, the presence probability psi, the conditional class probabilities $p_1 \dots p_K$ (given present), and the unconditional class probabilities $\text{psi} * p_k$ plus the absent probability $1 - \text{psi}$.

Usage

```
## S3 method for class 'occu_categorical_fit'
predict(object, newdata, ...)
```

Arguments

object	an occu_categorical_fit.
newdata	a data frame of predictors for the shared formula.
...	ignored.

Value

A list with psi (length n), cond (n x K conditional class probabilities), and joint (n x (K+1): absent, then class 1..K).

predict.tobs_fit	<i>Predict from occupancy model</i>
------------------	-------------------------------------

Description

Four modes:

- **In-sample:** predict(fit) returns fitted values.
- **State posterior / NA-response:** predict(fit, type = "state") returns the marginalised per-site occupancy posterior of a "nested_laplace" fit: the per-row eta posterior is a Gaussian mixture over the hyperparameter grid (per-cell fitted linear predictor and predictive variance), so psi is its Gauss-Hermite mean and psi_lower / psi_upper are the mixture-CDF quantiles – a calibrated 95% credible interval, exact for every latent prior. Rows with heldout = TRUE are the INLA-style NA-response prediction targets (single-season sites whose detection history was all-missing), where occupancy is interpolated from the latent field rather than informed by detections.
- **Design-matrix:** predict(fit, X.0 = ...) predicts at new covariate values.
- **Terms-based:** predict(fit, terms = "elev") varies one covariate, others at mean.
- **Joint occu_cover:** for a occu_cover joint-coupled fit, predict(fit, newdata, type = "occurrence" | "cover_cond" | "cover_exp" | "change") samples the joint latent from the grid-integrated posterior (the outer-grid mixture via [tulpa::tulpa_posterior_draws\(\)](#)) and marginalises every derived quantity per draw. type = "change" with times = c(t1, ..., tK) (K >= 2) returns a per-cell change table. Each quantity – occupancy psi, conditional cover cover_cond, expected cover cover_exp – has a level column per step (psi_T1..psi_TK, ...) carrying the posterior mean with .sd / .lwr / .upr at level, and a delta per step against t1 (delta_psi, delta_cover_cond, delta_cover_exp) with .lwr / .upr and .prob_pos, the posterior probability P(delta > 0) per cell. The expected-cover change is split exactly into its occupancy- and abundance-driven parts (delta_cover_from_occ, delta_cover_from_ab), reported the same way. With two times the delta columns are unsuffixed; with K > 2 each carries _T<k>. Every step is evaluated on ONE draw set, so levels and deltas share a posterior and are jointly valid. A standalone occu() joint fit returns the same layout with occupancy only. The result is a tobs_prediction table (one row per cell) carrying per-unit [cell x nsim] draw matrices in attr(, "draws"); map it yourself, e.g. left_join(cents, pr, by = "cell") then geom_tile(aes(x, y, fill = delta_psi)) (or geom_sf() on polygon cells). type = "trajectory" with times = c(t1, ..., tK) predicts occupancy psi, conditional cover

cover_cond and expected cover cover_exp at each time, for any times the user chooses, as a long table with the posterior mean, median and level interval of each (one row per cell and time); aggregate = TRUE averages over the cells per draw and gives the overall trajectory, one row per time. All times share one draw set.

- **Community families:** ms_occu(), ms_dyn_occu(), ms_int_occu(), ms_abun(), ms_distance() and ms_occu_cover() predict per-species matrices [rows x species], in-sample from fitted() and at newdata from the per-species coefficients. Each family answers to its own response types: "occupancy" / "detection" (ms_occu, ms_dyn_occu, ms_int_occu), "abundance" / "detection" (ms_abun), "lambda" / "sigma" (ms_distance), and "occupancy" / "detection" / "cover_cond" / "cover_exp" (ms_occu_cover). A fit carrying an areal field or a latent-factor loading is in-sample only, the field having no value at an unseen site.
- **Spatial-factor community:** for a reduced-rank spatial-factor ms_occu_cover() fit, predict(fit, type = "occupancy" | "cover_cond" | "cover_exp") returns the per-species per-cell posterior as a long table (one row per cell x species): "occupancy" gives psi, "cover_cond" the conditional cover mean $E[\text{cover} \mid \text{present}]$, "cover_exp" the unconditional expected cover $\text{psi} * E[\text{cover} \mid \text{present}]$, each with a _median and a _lower / _upper interval. The maps are marginalised over the loading + field posterior, so a rare species borrows strength across the shared factors for a calibrated map. The latent fields are tied to the cell graph, so X.0 / newdata prediction is not supported.

Usage

```
## S3 method for class 'tobs_fit'
predict(
  object,
  X.0 = NULL,
  type = c("occupancy", "detection", "both", "state"),
  quantiles = c(0.025, 0.5, 0.975),
  terms = NULL,
  n.points = 50L,
  newdata = NULL,
  times = NULL,
  level = 0.95,
  nsim = 1000L,
  draws = TRUE,
  time.col = NULL,
  X.det.0 = NULL,
  weights = NULL,
  aggregate = FALSE,
  mc.tol = 0.05,
  nsim.max = 10000L,
  mc.floor = 0.001,
  ...
)
```

Arguments

object A tobs_fit object.

<code>X.0</code>	Optional design matrix for occupancy prediction.
<code>type</code>	"occupancy" (default), "detection", "both", or "state" (nested-Laplace marginalised per-site psi, incl. held-out sites). For an <code>occu_cover</code> fit: "occurrence", "cover_cond", "cover_exp", "change" or "trajectory". "detection" / "both" on a <code>single/int_occu()</code> fit needs <code>X.det.0</code> .
<code>quantiles</code>	Quantile levels for credible intervals: a strictly increasing length-3 vector (lower, middle, upper), each in (0, 1). The levels name the columns they fill, so <code>c(0.05, 0.5, 0.95)</code> reports q5 / q50 / q95, and the levels used travel on the returned table in <code>attr(, "quantiles")</code> .
<code>terms</code>	Name of a single term to vary: one column of the predicted process's design matrix, every other column held at its column mean. A vector of more than one term is an error, since a second term would be held at its mean rather than grouped over its levels. For a grid over more than one covariate, build the design matrix and pass it as <code>X.0</code> .
<code>n.points</code>	Number of prediction points per continuous term.
<code>newdata</code>	data.frame of prediction units. For <code>occu_cover</code> : one row per field cell (or carrying a cell column mapping rows to field cells), defaulting to the training data. For the families predicting from a design matrix (<code>occu()</code> , <code>dyn_occu()</code> , <code>int_occu()</code> , <code>abun()</code> , <code>removal()</code> , <code>distance()</code> , <code>fp_occu()</code> , <code>dyn_abun()</code>): the covariates of the arm type selects, expanded through that arm's fitted formula (factor levels and transforms as fitted) in place of <code>X.0 / X.det.0</code> ; an <code>int_occu()</code> detection prediction expands it once per source. A random-effect term on the predicted arm whose grouping variable is a column of <code>newdata</code> adds each row's fitted group effect (its BLUP / posterior mean, times the slope covariate for a random slope); a level the fit did not see adds 0, the population mean. Without that column, and always through <code>X.0 / X.det.0</code> , the prediction is population-level. Give <code>newdata</code> or <code>X.0 / X.det.0 / terms</code> , not both.
<code>times</code>	<code>occu_cover</code> type = "change" / "trajectory": numeric values of the time covariate. For "change", at least two: <code>c(t1, t2)</code> differences one against the other, <code>c(t1, ..., tK)</code> differences every step against <code>t1</code> . For "trajectory", one or more times to predict at, any values of the covariate (between or beyond the fitted times; beyond them the prediction follows the fitted time effect).
<code>level</code>	<code>occu_cover</code> only: credible level for the interval columns (default 0.95).
<code>nsim</code>	<code>occu_cover</code> only: number of joint posterior draws (default 1000), or "auto" (type = "trajectory" on a joint nested-Laplace fit) to choose the count by Monte Carlo precision: draws are taken until the Monte Carlo standard error of every reported median and interval bound, over every unit, time and quantity, is at most <code>mc.tol</code> times that quantity's posterior SD, or <code>nsim.max</code> draws are reached. Each round redraws the whole table at the count the $1/\sqrt{n}$ scaling projects, so memory is that of one run at the final count and all quantities share one draw set.
<code>draws</code>	<code>occu_cover</code> only: if TRUE (default), carry the per-unit [<code>cell</code> x <code>nsim</code>] draw matrices in <code>attr(, "draws")</code> .
<code>time.col</code>	<code>occu_cover</code> only: name of the time covariate weighting the trend field / driving the change map; auto-resolved from the fit's stored trend weight when omitted.

X.det.0	Optional detection design for out-of-sample "detection" / "both" prediction on a single-season or <code>int_occu()</code> fit: a plain design matrix for a single-season fit, or a list of one design matrix per source (named by source, or in source order) for an <code>int_occu()</code> fit – matching <code>fitted()</code> \$p's per-source shape. In-sample locations only: like the state-arm <code>svc()/spde()</code> surfaces, this does not krig a spatial detection field to unseen points.
weights	<code>occu_cover</code> (and the joint <code>occu()</code> SVC route) only: pool several newdata rows into one prediction per cell. newdata then carries more than one row per cell (for example one row per habitat level), and <code>weights</code> gives each row's share of its cell, as a numeric vector with one entry per row or the name of a newdata column. Weights are normalised within each cell. Every quantity is pooled per posterior draw before it is summarised or differenced: occupancy, detection and expected cover are the weighted means of the rows, and conditional cover is the pooled expected cover divided by the pooled occupancy (the weighted mean of the rows' conditional cover when occupancy does not vary across them). The result has one row per cell, in order of first appearance, and a change map's occupancy and cover columns share one draw set. NULL (default) predicts each row on its own.
aggregate	<code>type = "trajectory"</code> only: FALSE (default) returns one row per prediction unit and time; TRUE averages each quantity over the units per posterior draw first and returns one row per time (the overall trend, with its own interval).
mc.tol	<code>nsim = "auto"</code> only: the Monte Carlo standard error allowed on each reported median and bound, as a fraction of its posterior SD (default 0.05, about 2,900 draws for a Gaussian-shaped quantity at level = 0.95).
nsim.max	<code>nsim = "auto"</code> only: the most draws to take (default 10000).
mc.floor	Trajectory only: the smallest posterior SD a Monte Carlo error is measured against, on the quantity's own scale (default 0.001). Where occupancy saturates near 0 or 1 its posterior SD goes to about 0, and an error relative to it would be unstable; the error is measured against <code>max(SD, mc.floor)</code> instead. <code>psi</code> , <code>cover_cond</code> and <code>cover_exp</code> are probabilities or cover fractions, so one floor serves all three.
...	Ignored.

Value

Depends on mode. In-sample: `fitted()` result. "state": a data.frame with row, `psi` (marginalised posterior mean), `psi_lower` / `psi_upper` (equal-tailed 95% credible interval; NA when the engine did not return per-cell predictive variance), and `heldout`. Design-matrix/ terms: data.frame with estimate and CIs; `type = "detection"` on an `int_occu()` fit returns a named list of such data.frames (one per source), and `type = "both"` returns `list(occupancy = , detection = )`. `occu_cover`: a `tobs_prediction` table (one row per cell) with per-unit draw matrices in `attr(, "draws")`; `type = "trajectory"` a long table (cell, time, then `<q>_mean` / `_median` / `_lwr` / `_upr` for `psi`, `cover_cond` and `cover_exp`), one row per cell and time, or per time with `aggregate = TRUE`. A trajectory states its Monte Carlo precision in `attr(, "mc_se_max")` (the largest MC standard error of any reported median or bound, in posterior SDs) and `attr(, "nsim_used")`; under `nsim = "auto"` also `attr(, "mc_tol")` and `attr(, "nsim_capped")` (TRUE when `nsim.max` stopped the draws before `mc.tol` was met). `attr(, "mc_binding")` names the bound that sets `mc_se_max`: a

one-row data frame with its cell (NA with aggregate = TRUE), time, quantity, bound ("median", "lwr" or "upr") and mc_se.

print.tobs_batch	<i>Print a batched multi-response occu_cover fit.</i>
------------------	---

Description

Print a batched multi-response occu_cover fit.

Usage

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

Arguments

x	A tobs_batch returned by <code>tobs()</code> on multi-response y.
...	Ignored.

Value

x, invisibly.

print.tobs_family	<i>Print method for tobs_family</i>
-------------------	-------------------------------------

Description

Print method for tobs_family

Usage

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

Arguments

x	a tobs_family object.
...	ignored.

Value

x, invisibly.

print.tobs_fit	<i>Print method for tobs_fit</i>
----------------	----------------------------------

Description

Print method for tobs_fit

Usage

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

Arguments

x	a tobs_fit object.
...	forwarded to underlying print methods.

Value

x, invisibly.

ranef.tobs_fit	<i>Random-effect estimates (BLUPs) for a tobs_fit</i>
----------------	---

Description

Returns the per-group random-effect posterior summaries. Under NUTS the non-centred draws are reconstructed to the natural BLUP scale ($b_{\{g,c\}} = \sigma_c * (L_{z_g})_c$) and summarised; the deterministic Laplace path returns the variance-component EM modes and their Schur-complement standard errors. A community fit reports its per-species deviations from the community mean. Every family and engine returns the same columns.

Usage

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

Arguments

object	A tobs_fit object.
...	Ignored.

Value

A data frame with one row per random-effect level and coefficient and the columns arm (the linear predictor the effect enters: `psi`, `p`, `lambda`, ...; `psi+p` for an effect shared across arms; NA when the fit does not distinguish arms), `group` (the grouping variable; "species" for the per-species deviations of a community fit), `level` (the group level, or the species), `term` (the coefficient the effect shifts, "(Intercept)" for a random intercept), `estimate` and `std.error` (NA where the fitter reports no posterior SD). Zero rows when the fit carries no random effects.

On a community fit, `std.error` is the per-species posterior SD of the deviation, the square root of the matching diagonal entry of the species' posterior covariance `fit$ms_community$Cinv[[s]]` (the analogue of `lme4`'s conditional variance). On the Laplace routes that covariance is conditional on the community means; on a spatial fit integrated over an outer hyperparameter grid it combines the per-node covariances and the spread of the per-node BLUPs (law of total covariance); on the NUTS and Polya-Gamma Gibbs routes it is the covariance of the sampled deviations. The `ms_abun()` `log_r` (negative binomial) and `logit_omega` (zero-inflated) deviations carry an SD as well: every route that fits them (joint AGHQ, and NUTS for `log_r`) keeps them in the covariance. `std.error` is NA for a species whose per-species posterior solve failed.

removal

*Removal-sampling family***Description**

Sequential-depletion removal sampling: latent abundance $N_i \sim \text{Poisson}(\lambda_i)$ (or negative binomial) observed through K ordered removal passes, where pass k removes $\text{Binomial}(N_i - \sum_{l < k} y_{il}, p_{ik})$ of the individuals still present. The declining catch sequence identifies detection p and abundance N ; the latent N is summed out in closed form (truncation $K.\text{max}$), so the fit is a direct Laplace approximation (no EM), with a NUTS path over the same marginal.

Usage

```
removal(K.max = NULL, mixture = c("poisson", "negbin"))
```

Arguments

<code>K.max</code>	upper bound for the latent-abundance marginal sum (the exact integration over N is truncated at $K.\text{max}$). <code>NULL</code> (default) lets the engine pick $\max(y) + 100$ (matching <code>unmarked::pcount()</code>); raise it if a fit warns that the posterior over N puts mass on the boundary.
<code>mixture</code>	latent-abundance distribution, both fitted via tulpa's closed-form marginal Laplace engine. "poisson" is the default; "negbin" (negative binomial, $\text{Var}(N) = \lambda + \lambda^2 / r$) adds an overdispersion parameter r . Non-spatially the log size <code>log_r</code> is estimated jointly with the coefficients and reported with an SE; on the areal-spatial path (<code>method = "nested_laplace"</code>) r is integrated over the outer hyperparameter grid and reported as a posterior mean / sd. Named mixture (after <code>unmarked::pcount()</code>) to avoid collision with the model-type family argument of <code>tobs()</code> .

Details

The `tobs()` formula is the abundance (log lambda) model; detection is the per-pass detection (logit p) model. The response `y` is an `n_sites` x `K` integer matrix of per-pass removals with the passes in column order; complete pass sequences are required (no NA).

Value

A `tobs_family` object.

Engines

- `method = "laplace"` – a direct Laplace approximation over the closed-form marginal (Poisson or negative binomial); a grouped random effect is integrated by adaptive Gauss-Hermite quadrature.
- `method = "nested_laplace"` – an areal `icar()` / `car_proper()` field on the abundance arm, optionally composed with a `temporal()` field, through the shared areal-BFGS driver.
- `method = "nuts"` – the in-tree C++ NUTS sampler over the same marginal, with a fixed-hyper areal or temporal field block on the abundance arm (one of the two, not both).

References

Royle, J. A. (2004). N-mixture models for estimating population size from spatially replicated counts. *Biometrics* 60, 108-115. Dorazio, R. M., Jelks, H. L., Jordan, F. (2005). Improving removal-based estimates of abundance. *Biometrics* 61, 1093-1101.

Examples

```
f <- removal(K.max = 100)
f
```

removal_laplace

Laplace fit of the removal-sampling abundance model

Description

Maximum-likelihood fit (non-spatial, fixed effects only) of the removal / sequential-depletion abundance model with a Poisson or negative-binomial abundance mixing distribution. Pass `k` removes $y_{ik} \sim \text{Binomial}(N_i - \sum_{l < k} y_{il}, p_{ik})$ of the individuals still present, and the latent $N_i \sim \text{Poisson}(\lambda_i)$ (or $\text{NegBin}(\lambda_i, r)$) is summed out in closed form (truncation `K_max`). Optimisation is the same analytic inner-Newton / observed-Fisher path the N-mixture uses; the per-site detection arm sees the depleting available count rather than the full `N`.

Usage

```
removal_laplace(
  y,
  site_idx,
  X_lambda,
  X_p,
  mixture = c("P", "NB"),
  beta_lambda_init = NULL,
  beta_p_init = NULL,
  log_r_init = NULL,
  r_max = 1e+05,
  K_max = NULL,
  max_iter = 100L,
  tol = 1e-06,
  verbose = FALSE
)
```

Arguments

<code>y</code>	Integer vector of per-pass removals (long form), passes in order.
<code>site_idx</code>	Integer vector, same length as <code>y</code> , 1-based site index; a site's passes must appear in pass order (the depletion order).
<code>X_lambda</code>	Numeric matrix [<code>n_sites</code> x <code>p_lambda</code>] of abundance covariates.
<code>X_p</code>	Numeric matrix [<code>n_obs</code> x <code>p_p</code>] of per-pass detection covariates.
<code>mixture</code>	"P" (Poisson, default) or "NB" (negative binomial).
<code>beta_lambda_init</code> , <code>beta_p_init</code> , <code>log_r_init</code>	Optional warm starts.
<code>r_max</code>	Upper bound on the NB size <code>r</code> (NB only, default 1e5).
<code>K_max</code>	Marginal-sum truncation; defaults to <code>max(site removal total) + 100</code> . Must be $\geq$ the largest per-site removal total.
<code>max_iter</code>	Newton iteration budget (default 100).
<code>tol</code>	Gradient-norm convergence tolerance (default 1e-6).
<code>verbose</code>	Print per-iteration progress.

Value

A list of class `nmix_fit` (the shared count-marginal fit shape): `beta_lambda`, `beta_p`, `mixture`, `log_r / r` (NB), `log_lik`, `vcov`, `H_obs`, `n_iter`, `converged`, `grad_norm`, per-site `mean_N / var_N / boundary_weight`.

References

Royle, J. A. (2004). N-mixture models for estimating population size from spatially replicated counts. *Biometrics* 60, 108-115. Dorazio, R. M., Jelks, H. L., Jordan, F. (2005). Improving removal-based estimates of abundance. *Biometrics* 61, 1093-1101.

residuals.tobs_fit	<i>Residuals from occupancy model</i>
--------------------	---------------------------------------

Description

One contract for every family: a list with occ, the STATE-level residual series (per site, or per site and season for a multi-season family), and det, the OBSERVATION-level series (per visit / pass / distance bin). A family carrying only one of the two levels fills that one and leaves the other NULL; nothing returns a bare matrix, so a caller reading \$occ gets the state residual or NULL, never an error.

Usage

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

Arguments

object	A tobs_fit object.
type	One of "deviance" (default), "pearson", or "response".
...	Ignored.

Value

A list with occ (state-level) and det (observation-level) residuals, either of which may be NULL for a family that has no such level. A model type with no registered handler is an error naming it.

royle_nichols	<i>Royle-Nichols occupancy family</i>
---------------	---------------------------------------

Description

Occupancy with abundance-induced detection heterogeneity (Royle & Nichols 2003; **unmarked** occuRN). Latent abundance $N_i \sim \text{Poisson}(\lambda_i)$ drives the per-visit detection probability $1 - (1 - r_{ij})^{N_i}$, where r_{ij} is the per-individual detection probability. The state formula models $\log \lambda$ (abundance); detection models $\text{logit } r$. Detection is site-level by default; passing visits makes it visit-varying ($\text{logit } r_{ij}$ gains the visit-level covariates), exactly as for the occupancy / N-mixture front doors. The latent N marginalises in closed form (a Poisson sum to $K.\text{max}$), so the fit maximises the exact marginal with an observed-information vcov.

Usage

```
royle_nichols(K.max = NULL)
```

Arguments

K.max Upper summation bound for the latent abundance (default: a data-driven Poisson-tail guess).

Value

A `tobs_family` object for `tobs()`.

Engines

- `method = "laplace"` – the exact marginal maximised (BFGS) with an observed-information covariance; the only route (no field, no sampler).

Examples

```
sim <- simulate_royle_nichols(N = 150, J = 5, seed = 1)
fit <- tobs(~ x, data = sim$data, family = royle_nichols(),
           detection = ~ 1, y = sim$y, control = list(verbose = FALSE))
coef(fit)

# Visit-varying detection via `visits`:
sv <- simulate_royle_nichols(N = 150, J = 5, beta.r.visit = 0.8, seed = 1)
fv <- tobs(~ x, data = sv$data, family = royle_nichols(),
           detection = ~ w, y = sv$y, visits = sv$visits,
           control = list(verbose = FALSE))
coef(fv)
```

sbc.tobs_fit

Simulation-based calibration for a fitted tobs model

Description

Runs the POSTERIOR simulation-based calibration experiment (Talts et al. 2018; Sailyoja, Schmitt, Buerkner and Vehtari 2026, Algorithm 2) on a fitted `tobs_fit`, through the engine's `tulpa::sbc()` front door. The truth is drawn from the fit's own posterior at the observed data, a replicate is simulated at that truth on FRESH cells, the model is refitted on the two data sets together, and the truth's rank under that augmented posterior is recorded. Under exact inference those ranks are `Uniform(0, 1)`; the departure from uniformity is a verdict on the approximation.

Usage

```
## S3 method for class 'tobs_fit'
sbc(
  object,
  n.sim = 100L,
  n.draws = 1000L,
  n.ref = 200L,
```

```

quantities = NULL,
controls = character(),
bad.factor = 1.25,
level = 0.95,
seed = 0L,
model.only = FALSE,
fit.control = list(),
control = list(),
...
)

```

Arguments

object	A fitted <code>tobs_fit</code> . Its family must be registered; see Details for the roster.
n.sim	Number of simulations. Each costs one refit on the pooled data.
n.draws	Posterior draws per fit backing the reported predictives.
n.ref	Reference draws behind the <code>log_lik</code> rank. 0 drops that arm.
quantities	Optional character vector restricting what is scored.
controls	Mis-scaled control arms to add: any of "wide", "narrow".
bad.factor	The control arms' SD multiplier.
level	Simultaneous band level.
seed	Seed offset. Simulation <i>i</i> draws its truth at <code>seed + i</code> .
model.only	Return the callback list instead of running the experiment, for inspection or for passing to <code>tulpa::sbc()</code> directly.
fit.control	Merged into the control list of every refit.
control	Passed to <code>tulpa::sbc()</code> (progress, rand_seed).
...	Unused.

Details

Registered families: `occu_cover` (the coupled occupancy + cover hurdle on the joint nested-Laplace engine), and `occu`, `abun`, `count`, `removal`, `distance`, `fp_occu`, `royle_nichols`, `occu_ttd` and `double_observer`, whose site marginals multiply. For those the replicate is the family's own `simulate()` kernel at the drawn theta, the rank arm is the family's exact marginal log-likelihood, and two kinds of fit are refused rather than approximated: a structured term (its field is a latent quantity shared across sites that theta does not hold, which is what the coupled `occu_cover` route handles by drawing on fresh cells) and a visit-level observation design (the replicate kernel assembles the observation arm from the site-level design, so the two sides would carry different models).

`dyn_occu` (constant-rate only) pools on the SITE axis and leaves the season axis alone, since its response is 3D [`n_sites` x `max_visits` x `n_seasons`]: a bespoke replicate generator forward-simulates the colonization/extinction HMM, but the posterior draws and the rank statistic are the same shared machinery every other registered family uses.

`int_occu` (full source-site overlap only) pools on the SITE axis and leaves the per-source axis alone, since its response is a list of one detection matrix per source; the rank statistic is the shared machinery unchanged. Registering it surfaced a real bug (since fixed): the detection intercept was silently

left in standardized-covariate units on every fit with an autoscaled detection covariate, independent of SBC.

gdistremoval pools two response matrices (yDist band counts, yRem period counts) together on the SITE axis, reusing the same list-of-matrices pool int_occu uses; its replicate generator is its own simulate() handler.

dyn_abun (constant-rate only) shares dyn_occu's 3D [n_sites x max_visits x n_seasons] response and site-axis pooling, but unlike dyn_occu it already has a working simulate() handler (the Dail-Madsen open N-mixture forward is not a two-state HMM that needs a bespoke generator), so its replicate is the same family-simulate() route as occu/abun/etc.

occu_categorical is a tobs_multiarm_fit with two independent Laplace-Gaussian blocks (presence, class) rather than a joint MVN draw matrix; draws() samples the two blocks independently and loglik_many() scores the two-arm likelihood directly, since this family has no .tobs_pointwise_loglik dispatch to reuse.

distsamp_open (constant-dynamics, Poisson only) shares dyn_abun's 3D response and site-axis pooling, and its fit\$means/fit\$draws are the standard .tobs_bfgs_marginal_fit() shape, so draws/simulate/loglik_many are all the shared generic ones.

occu_multi's response is a list of S per-species matrices, the same shape int_occu() pools; its simulate() is custom, since species share one joint multi-species state rather than being independently observed.

dyn_int_occu is the product of the multi-season and multi-source shapes: a named list of S per-source 3D arrays, pooled by a new .tobs_sbc_pool_named_3d that composes both existing pooling rules.

t_occu is a pg_gibbs family whose fit\$draws is already the real pooled posterior sample; it has no simulate() handler and no .tobs_pointwise_loglik dispatch, so both are custom – simulate draws a fresh ARI year effect at the theta's own (sigma, rho), and loglik_many is a Laplace approximation to that year effect's marginal (FD-validated, brute-force cross-checked at T = 2).

cover is a tobs_multiarm_fit with the same two-independent-block shape as occu_categorical; positive = "lognormal" only for v1, with dispersion held fixed (no SE is reported for it anywhere in the package).

occu_multiscale_cover is a standard single-block fit (unlike cover's two-block one); the exchangeable unit is the CELL rather than the plot, so pooling and site labeling track cell indices.

A family is registered by adding one entry to the internal registry – a replicate generator, a refit call and optionally a joint statistic; the pooling, the grouping labels, the arm construction and the controls are shared.

Value

An object of class sbc from **tulpa**, carrying the per-quantity rank ECDFs, the exact simultaneous band, and the uniformity tests. When model.only = TRUE, the callback list instead.

What is scored

The arm fixed effects, the areal field SD on each arm, the copy scale alpha, and the dispersion when the fit estimates one. The per-cell field itself is not scored – it is integrated out by the fit and carries no truth theta could hold. The cover-arm field SD and alpha are read per draw off the outer

hyperparameter grid, whose cells are sampled by their own normalized weight, so what is ranked is the grid-marginalized posterior of each rather than a function of component modes.

A quantity whose posterior has no spread – a dispersion the joint engine holds fixed in its cell-coupling spec, for instance – is dropped rather than scored, and named in `attr(x, "fixed")`.

An extra `log_lik` quantity ranks a joint statistic of the whole parameter vector, which catches a posterior getting each marginal right while getting the dependence between them wrong.

Fresh cells

The replicate is drawn on new cells whose areal graph is block-diagonal against the observed one. `occu_cover` couples its occurrence and cover arms through one shared field, and `theta` carries no per-cell field value, so a replicate re-observing the observed cells would depend on the observed data given `theta` and break the factorization posterior SBC rests on. `group_ids` is supplied to `tulpa::sbc()`, which verifies the observable half of that (disjoint group labels) rather than assuming it.

Controls

`controls = c("wide", "narrow")` adds arms reporting the same draws rescaled about their mean by `bad.factor` and its reciprocal. They are deliberately mis-scaled posteriors and are expected to leave the band: a calibration read that nothing can fail is not evidence.

References

Talts, S., Betancourt, M., Simpson, D., Vehtari, A. and Gelman, A. (2018). Validating Bayesian inference algorithms with simulation-based calibration. *arXiv:1804.06788*.

Sailynoja, T., Schmitt, M., Buerkner, P.-C. and Vehtari, A. (2026). Posterior SBC: simulation-based calibration checking conditional on data. *Statistics and Computing* 36:78.

See Also

`tulpa::sbc()`, `waic()`

Examples

```
N <- 20L; J <- 3L
adj <- matrix(0L, N, N)
for (s in seq_len(N)) {
  if (s > 1L) adj[s, s - 1L] <- 1L
  if (s < N)  adj[s, s + 1L] <- 1L
}
sim <- simulate_occu_cover(N = N, J = J, positive = "lognormal",
                          adj = adj, sigma = 0.7, alpha = 1, seed = 1L)
long <- data.frame(site_id = rep(seq_len(N), each = J),
                  visit = rep(seq_len(J), times = N),
                  y = as.vector(t(sim$y)),
                  det_cov1 = sim$visit_data$det_cov1,
                  pos_cov1 = sim$visit_data$pos_cov1)
od <- tobs_data(long, y = "y", site = "site_id", visit = "visit",
               det.covs = c("det_cov1", "pos_cov1"))
```

```

y_pos <- sim$y_pos; y_pos[is.na(y_pos)] <- 0
# The dispersion goes on the outer grid so it is estimated and scored, and
# the field-SD grid is pinned so both stages integrate the same support.
ctl <- list(engine = "joint", verbose = FALSE,
            sigma.grid = exp(seq(log(0.15), log(2.0), length.out = 9)),
            phi.grid.pos = exp(seq(log(0.20), log(0.90), length.out = 7)))
fit <- tobs(~ occ_cov1 + icar(graph = adj),
            data = cbind(data.frame(site_id = seq_len(N)), sim$data),
            family = occu_cover("lognormal"),
            detection = ~ det_cov1,
            positive = ~ pos_cov1 + share(spatial()),
            y = od$y, y.pos = y_pos, visits = od$det.covs,
            method = "nested_laplace", control = ctl)
sbc(fit, n.sim = 20L, controls = "narrow", fit.control = ctl)

```

simulate.tobs_fit	<i>Simulate replicate datasets from the posterior</i>
-------------------	---

Description

Each replicate draws a parameter vector from the fit's posterior draws and simulates the latent state and the response from it, so the replicates carry parameter uncertainty.

Usage

```

## S3 method for class 'tobs_fit'
simulate(object, nsim = 1, seed = NULL, ...)

```

Arguments

object	A tobs_fit object.
nsim	Number of simulated datasets (default 1).
seed	Optional random seed. As for stats::simulate() , a supplied seed is set for the call and the caller's random number stream is restored afterwards.
...	Must be empty.

Value

A list of length nsim, one simulated response per element (named sim_1, sim_2, ...), each in the layout the family's y takes: a detection matrix, a [sites x visits x seasons] array, a list of source or species matrices, and so on. The list carries the "seed" attribute [stats::simulate\(\)](#) specifies: the .Random.seed in force when seed is NULL, otherwise seed with its RNGkind() as the "kind" attribute. Every family returns this layout, whatever nsim is.

simulate_abun

*Simulate Royle (2004) N-mixture abundance data***Description**

Latent abundance $N_i \sim \text{Poisson}(\lambda_i)$ (or, under mixture = "negbin", $N_i \sim \text{NegBin}(\text{mean} = \lambda_i, \text{size} = \text{size})$ with variance $\lambda_i + \lambda_i^2 / \text{size}$) with $\log \lambda_i = X_{\lambda} \beta_{\lambda}$, and replicate counts $y_{ij} \sim \text{Binomial}(N_i, p_i)$ with $\text{logit } p_i = X_p \beta_p$ (site-level detection). The returned y is an $N \times J$ integer count matrix suitable for `tobs()` with `abun()`.

Usage

```
simulate_abun(
  N = 100,
  J = 4,
  n.abund.covs = 2,
  n.det.covs = 1,
  beta.lambda = NULL,
  beta.p = NULL,
  mixture = c("poisson", "negbin", "zip", "zinb"),
  size = 2,
  omega = 0.3,
  seed = NULL
)
```

Arguments

<code>N</code>	Number of sites (default 100).
<code>J</code>	Number of replicate visits (default 4).
<code>n.abund.covs</code>	Number of abundance covariates (default 2).
<code>n.det.covs</code>	Number of detection covariates (default 1).
<code>beta.lambda</code>	Abundance coefficients $c(\text{intercept}, \text{slopes} \dots)$ on the log scale. Default $c(\log(3), \text{runif}(n.abund.covs, -0.5, 0.5))$.
<code>beta.p</code>	Detection coefficients $c(\text{intercept}, \text{slopes} \dots)$ on the logit scale. Default $c(0, \text{runif}(n.det.covs, -0.5, 0.5))$ (intercept 0 = $p = 0.5$).
<code>mixture</code>	Abundance mixing distribution: "poisson" (default), "negbin" (negative binomial, overdispersed), or their zero-inflated counterparts "zip" / "zinb" (a structural-zero share ω of sites have $N = 0$ regardless of λ).
<code>size</code>	Negative-binomial size r (the "negbin" / "zinb" dispersion; variance $\lambda + \lambda^2 / r$). Smaller r means more overdispersion; as r grows large the model approaches Poisson. Default 2. Ignored under Poisson.
<code>omega</code>	Structural-zero probability for "zip" / "zinb" (the share of sites with $N = 0$ independent of λ). Default 0.3. Ignored otherwise.
<code>seed</code>	Optional random seed.

Value

A list with y (N x J count matrix), data (covariate data frame), and truth (the coefficients, per-site lambda, p, latent N, and the mixture / size used).

Examples

```
sim <- simulate_abun(N = 50, J = 3, seed = 1)
dim(sim$y)
```

simulate_count	<i>Simulate count / relative-abundance GLMM data</i>
----------------	--

Description

One observed value per site from a GLMM with no detection process, matching the `count()` family: log-link Poisson / negative-binomial counts, an identity-link Gaussian response, or a logit-link binomial response (k successes out of n trials per site). The mean predictor is X beta with beta the fixed-effect coefficients (intercept first).

Usage

```
simulate_count(
  N = 200,
  beta = c(1, 0.5),
  response = c("poisson", "negbin", "gaussian", "binomial"),
  size = 2,
  sd = 1,
  trials = 10,
  seed = NULL
)
```

Arguments

N	Number of sites (default 200).
beta	Fixed-effect coefficients, length 1 + n_covs (intercept first).
response	The response distribution: "poisson", "negbin", "gaussian", or "binomial".
size	Negative-binomial size (dispersion); larger is closer to Poisson. Used only for response = "negbin".
sd	Gaussian residual SD. Used only for response = "gaussian".
trials	Per-site trial count for response = "binomial": a scalar (recycled) or a length-N vector. Default 10 (a scalar 1 gives Bernoulli data, the <code>svcPGBinom</code> trials = 1 setting).
seed	Random seed.

Value

A list with `y` (a length-`N` numeric response vector; success counts for the binomial response), `data` (the site covariates), and `truth` (the coefficients, response, dispersion, and the binomial trials).

Examples

```
sim <- simulate_count(N = 100, beta = c(1, 0.5), seed = 1)
length(sim$y)
```

simulate_cover	<i>Simulate cover-hurdle data (lognormal positive part)</i>
----------------	---

Description

Generates synthetic data matching the `cover(response = "lognormal")` generative model:

Usage

```
simulate_cover(
  N = 200L,
  beta.occ = c(-0.5, 0.8),
  beta.pos = c(-1, 0.3),
  sigma.pos = 0.4,
  response = c("lognormal", "gaussian"),
  spatial.range = NULL,
  spatial.var = 1,
  seed = NULL
)
```

Arguments

<code>N</code>	Number of sites (default 200).
<code>beta.occ</code>	Length-2 occurrence coefficients (intercept, slope on <code>x</code>).
<code>beta.pos</code>	Length-2 log-cover coefficients (intercept, slope on <code>x</code>).
<code>sigma.pos</code>	Lognormal residual standard deviation (default 0.4).
<code>response</code>	Positive-arm likelihood: "lognormal" (default) or "gaussian".
<code>spatial.range</code>	Length scale of the exponential spatial field (in unit-square distance). <code>NULL</code> disables the spatial layer.
<code>spatial.var</code>	Marginal variance of the spatial field (default 1).
<code>seed</code>	Optional integer seed.

Details

```

eta_occ = X %*% beta_occ + spatial_occ
eta_pos = X %*% beta_pos + spatial_pos
p = plogis(eta_occ)
occur ~ Bernoulli(p)
log_cover ~ Normal(eta_pos, sigma_pos^2) | occur = 1
cover = exp(log_cover) when occur = 1, else 0

```

Both layers share the same design matrix by default (one continuous covariate plus an intercept). When `spatial.range` is supplied, a simple exponential-kernel Gaussian field on a random unit-square coordinate set is added to both linear predictors; the two layers see independently simulated draws of that field, which matches the two-independent-arm fit. The joint engine's shared field is a different generative model.

Value

A list with:

data A data frame with cover, covariate x, and lon, lat.
y data\$cover (length-N numeric vector, for passing to `tobs()`).
coords An N x 2 matrix of coordinates.
truth A list with beta_occ, beta_pos, sigma_pos, p, mu, occur, spatial_occ, spatial_pos.

Examples

```

sim <- simulate_cover(N = 200, seed = 1)
head(sim$data)

```

simulate_cover_joint *Simulate cover-hurdle data with a shared BYM2 spatial field*

Description

Generates synthetic data matching the joint nested-Laplace cover-hurdle parameterisation ($\sigma_{occ} * z$ on the occurrence arm, $\alpha * \sigma * z$ on the cover arm) where z is a BYM2 latent field on the supplied region adjacency. The cover-arm field amplitude $\sigma_{pos} = \alpha * \sigma$ is the derived ratio surfaced as the "alpha" column of `fit$joint$theta_grid`.

Usage

```

simulate_cover_joint(
  N = 300L,
  adj,
  beta.occ = c(-0.3, 0.7),
  beta.pos = c(0.4, -0.5),
  sigma = 0.6,

```

```

    rho = 0.7,
    alpha = 1,
    positive = c("beta", "lognormal", "gaussian"),
    phi = 30,
    sigma.pos.resid = 0.4,
    seed = NULL
  )

```

Arguments

<code>N</code>	Number of sites (default 300).
<code>adj</code>	<code>n_s</code> x <code>n_s</code> integer adjacency matrix for the BYM2 field.
<code>beta.occ</code>	Length-2 occurrence coefficients (intercept + slope).
<code>beta.pos</code>	Length-2 cover-arm coefficients (intercept + slope).
<code>sigma</code>	Marginal spatial-field amplitude (default 0.6); the occurrence-arm linear predictor adds $\sigma * z[\text{region}]$.
<code>rho</code>	BYM2 mixing parameter, in $[0, 1]$ (default 0.7). $\rho = 1$ is pure ICAR; $\rho = 0$ is pure IID.
<code>alpha</code>	Cover-arm scaling: cover-arm linear predictor adds $\alpha * \sigma * z[\text{region}]$ (so $\sigma_{\text{pos}} = \alpha * \sigma$).
<code>positive</code>	Likelihood for the positive arm: "beta" or "lognormal".
<code>phi</code>	Beta precision when <code>positive = "beta"</code> (default 30).
<code>sigma.pos.resid</code>	Lognormal residual SD when <code>positive = "lognormal"</code> (default 0.4); independent of the spatial <code>sigma</code> .
<code>seed</code>	Optional integer seed.

Details

Both BYM2 sub-blocks (`phi`, `theta`) are drawn IID Normal **and demeaned to mean zero** before scaling, matching the sum-to-zero constraint applied to both sub-blocks by `tulpa::spatial_bym2()` inside the nested-Laplace engine (see `.joint_inner_var()`). Without the demean each seed carries $\text{mean}(w_s) \sim N(0, \sigma^2 / n_s)$ and the constrained intercept identified by the engine targets $\text{beta_pos_0_truth} + \alpha * \text{mean}(w_s)$ rather than `beta_pos_0_truth`; coverage of the *population* truth then collapses with `alpha` while the engine-reported posterior is correctly calibrated for the constrained parameter. See `example/validation/SUMMARY.md` in INLAabun (Demo 3) for the diagnosis.

Value

A list with:

- data** Data frame with `x` and `region` (factor).
- y** Length-`N` numeric vector of cover values (zeros where occur = 0).
- adj** The adjacency matrix passed in (for downstream `tulpa::spatial_bym2()`).
- truth** Named list with `beta_occ`, `beta_pos`, `sigma`, `rho`, `alpha`, `sigma_pos = alpha * sigma`, `positive`, plus the simulated `phi_f`, `theta_f`, `w_s`, `region`.

Examples

```
adj <- matrix(0L, 10, 10)
for (s in 1:10) for (j in setdiff(c(s - 1L, s + 1L), c(0L, 11L)))
  adj[s, j] <- 1L
sim <- simulate_cover_joint(N = 100, adj = adj, alpha = 1.0, seed = 1)
head(sim$data)
```

simulate_distance	<i>Simulate binned distance-sampling abundance data</i>
-------------------	---

Description

Latent abundance $N_i \sim \text{Poisson}(\lambda_i)$ (or $\text{NegBin}(\lambda_i, \text{size})$) with $\log \lambda_i = X_{\lambda} \beta_{\lambda}$ observed through a half-normal or hazard-rate detection function with $\log \sigma_i = X_{\sigma} \beta_{\sigma}$. Each individual lands in a distance bin (or goes undetected) by the multinomial cell probabilities. Returns an $N \times n_{\text{bins}}$ integer matrix of per-bin counts suitable for `tobs()` with `distance()`.

Usage

```
simulate_distance(
  N = 200,
  cutpoints = seq(0, 1, length.out = 6),
  key = c("halfnorm", "hazard"),
  transect = c("line", "point"),
  n.abund.covs = 1,
  n.sigma.covs = 1,
  beta.lambda = NULL,
  beta.sigma = NULL,
  shape = 3,
  mixture = c("poisson", "negbin"),
  size = 5,
  seed = NULL
)
```

Arguments

N	Number of sites (default 200).
cutpoints	Distance-bin edges (length $n_{\text{bins}} + 1$). Default <code>seq(0, 1, length.out = 6)</code> (five bins out to 1).
key	"halfnorm" (default) or "hazard".
transect	"line" (default) or "point".
n.abund.covs, n.sigma.covs	Number of abundance / detection covariates.
beta.lambda	Abundance coefficients (log scale). Default <code>c(log(40), runif(n.abund.covs, -0.4, 0.4))</code> .

beta.sigma	Detection-scale coefficients (log scale). Default $c(\log(0.4), \text{runif}(n.\text{sigma.covs}, -0.3, 0.3))$.
shape	Hazard-rate shape b (key = "hazard" only, default 3).
mixture	"poisson" (default) or "negbin".
size	Negative-binomial size r (mixture = "negbin" only, default 5).
seed	Optional random seed.

Value

A list with y ($N \times n_{\text{bins}}$ count matrix), data (covariates), cutpoints, and truth (coefficients, per-site λ / σ , latent N , key/transect/mixture/shape/size).

Examples

```
sim <- simulate_distance(N = 50, seed = 1)
head(sim$y)
```

```
simulate_distsamp_open
```

Simulate an open-population distance-sampling data set

Description

Draws from the `distsamp_open()` model: an open metapopulation ($N_1 \sim \text{Poisson} / \text{NB}(\lambda)$, $N_t = \text{Binomial}(N_{t-1}, \omega) + \text{Poisson}(\gamma)$) observed by distance sampling (half-normal key, scale σ) at each primary period.

Usage

```
simulate_distsamp_open(
  N = 200,
  cutpoints = c(0, 10, 20, 30, 40),
  n.seasons = 4L,
  transect = "line",
  n.abund.covs = 1,
  n.det.covs = 1,
  beta.lambda = NULL,
  beta.sigma = NULL,
  omega = 0.7,
  gamma = 2.5,
  mixture = c("poisson", "negbin", "zip", "zinb"),
  size = 3,
  zi = 0.3,
  dynamics = c("constant", "notrend", "trend", "autoreg", "ricker", "gompertz"),
  K = NULL,
  r = 0.3,
  seed = NULL
)
```

Arguments

N	Number of sites (default 200).
cutpoints	Distance-bin edges $\emptyset = c_{\emptyset} < \dots < c_B$.
n.seasons	Number of primary periods (default 4).
transect	"line" (default) or "point".
n.abund.covs, n.det.covs	Number of abundance / distance covariates.
beta.lambda, beta.sigma	Coefficients on the log-abundance and log-scale arms. Defaults give moderate abundance / detection.
omega, gamma	Apparent survival probability and recruitment rate (intercept-only defaults 0.7 / 2.5).
mixture	Initial-abundance distribution: "poisson" (default), "negbin" / "zinb" (draws N_1 negative-binomial with size size), or "zip" / "zinb" (a zi fraction of sites are structural zeros).
size	Negative-binomial size (overdispersion r) when mixture draws an NB initial abundance (default 3).
zi	Structural-zero probability when mixture is "zip" / "zinb" (default 0.3).
dynamics	Population-dynamics form (see distsamp_open()): "constant" (default), "notrend", "trend", "autoreg", "ricker", or "gompertz". The alternative dynamics draw the abundance sequence from the matching transition (Poisson initial abundance only).
K	Carrying capacity for "ricker" / "gompertz" (default $4 * \text{lambda}\emptyset$); r the intrinsic growth rate for those forms (default 0.3). For "trend" / "autoreg" the recruitment multiplier is gamma.
r	Intrinsic growth rate for "ricker" / "gompertz" (default 0.3).
seed	Optional random seed.

Value

A list with y ([n_sites x n_bins x n_seasons] distance-band counts), data, and truth.

Examples

```
sim <- simulate_distsamp_open(N = 40, n.seasons = 3, seed = 1)
dim(sim$y)
```

simulate_double_observer

Simulate a double-observer abundance data set

Description

Draws from the `double_observer()` model: site abundance $N \sim \text{Poisson}(\lambda)$ observed by two observers with detection p_1 / p_2 . For `type = "independent"` each individual is recorded by observer 1 only, observer 2 only, or both (three cell counts). For `type = "dependent"` a primary observer records what it detects and a secondary observer records only the primary's misses (two cell counts, primary-detected and secondary-only); the primary observer alternates across sites (role-swapping) so p_1 and p_2 are identifiable, and the per-site primary indicator is returned as `primary`.

Usage

```
simulate_double_observer(
  N = 200,
  type = c("independent", "dependent"),
  n.abund.covs = 1,
  n.det.covs = 1,
  beta.lambda = NULL,
  beta.p1 = NULL,
  beta.p2 = NULL,
  seed = NULL
)
```

Arguments

<code>N</code>	Number of sites (default 200).
<code>type</code>	"independent" (default) or "dependent" (role-swapping).
<code>n.abund.covs, n.det.covs</code>	Number of abundance / detection covariates.
<code>beta.lambda</code>	Log-abundance coefficients <code>c(intercept, slopes...)</code> . Default <code>c(log(8), runif(n.abund.covs, -0.5, 0.5))</code> .
<code>beta.p1, beta.p2</code>	Per-observer detection coefficients (logit). Default moderate detection.
<code>seed</code>	Optional random seed.

Value

A list with `y` ($N \times 3$ cell counts for "independent", $N \times 2$ for "dependent"), `data`, `primary` (the per-site primary observer, for "dependent"), and `truth`.

Examples

```
sim <- simulate_double_observer(N = 50, seed = 1)
head(sim$y)
```

simulate_dyn_abun	<i>Simulate Dail-Madsen open N-mixture data</i>
-------------------	---

Description

Latent $N_{1t} \sim \text{Poisson}(\lambda_{1t})$; for $t \geq 2$, $N_{1t} = \text{Binomial}(N_{1,t-1}, \omega_{1t}) + \text{Poisson}(\gamma_{1t})$; observed via $\text{Binomial}(N_{1t}, p)$ over J secondary visits in each of T primary seasons. Returns a 3D array $N \times J \times T$ suitable for `tobs()` with `dyn_abun()`.

Usage

```
simulate_dyn_abun(
  N = 150,
  T = 4,
  J = 3,
  n.abund.covs = 1,
  beta.lambda = NULL,
  p = 0.5,
  omega = 0.6,
  gamma = 1,
  beta.omega = NULL,
  beta.gamma = NULL,
  mixture = c("poisson", "negbin"),
  r = 2,
  zi = 0,
  seed = NULL
)
```

Arguments

<code>N</code>	Number of sites (default 150).
<code>T</code>	Number of primary seasons (default 4).
<code>J</code>	Number of secondary visits per season (default 3).
<code>n.abund.covs</code>	Number of initial-abundance covariates (default 1).
<code>beta.lambda</code>	Initial-abundance coefficients (log). Default <code>c(log(5), runif(n.abund.covs, -0.4, 0.4))</code> .
<code>p, omega, gamma</code>	Detection, apparent-survival, and recruitment parameters (scalars; defaults 0.5, 0.6, 1.0). <code>omega / gamma</code> set the constant rate used when <code>beta.omega / beta.gamma</code> are NULL.
<code>beta.omega, beta.gamma</code>	Optional length-2 coefficients <code>c(intercept, slope)</code> for a season-varying survival (logit link) or recruitment (log link) rate driven by a per-(site, interval) season covariate. NULL (default) keeps the constant <code>omega / gamma</code> .
<code>mixture</code>	Initial-abundance distribution: "poisson" (default) or "negbin" (negative-binomial $N_{1t} \sim \text{NB}(\text{mean} = \lambda_{1t}, \text{size} = r)$).

<code>r</code>	Negative-binomial size for mixture = "negbin" (default 2).
<code>zi</code>	Structural-zero share for a zero-inflated model (default 0). With probability <code>zi</code> a site is never occupied ($N_t = 0$ for all seasons), so all its counts are zero; fit such data with <code>dyn_abun(mixture = "zip") / "zinb"</code> .
<code>seed</code>	Optional random seed.

Details

Survival and recruitment are constant across seasons by default. Supplying `beta.omega` (logit-scale) or `beta.gamma` (log-scale) instead makes the rate depend on a per-(site, interval) season covariate `z`: $\omega_{i,t} = \text{plogis}(\text{beta_omega}[1] + \text{beta_omega}[2] * z_{i,t})$, $\gamma_{i,t} = \exp(\text{beta_gamma}[1] + \text{beta_gamma}[2] * z_{i,t})$. The covariate is returned as a $[N \times (T-1)]$ matrix column `season_cov` of data, ready for $\omega = \sim \text{season_cov}$.

Value

A list with `y` ($N \times J \times T$ count array), `data` (covariates, including the $[N \times (T-1)]$ `season_cov` matrix column when a season-varying rate is used), and `truth` (coefficients, per-site λ , the realised ω / γ rates, and `r` under "negbin").

Examples

```
sim <- simulate_dyn_abun(N = 40, T = 3, J = 3, seed = 1)
dim(sim$y)
```

`simulate_dyn_int_occu` *Simulate a multi-season integrated occupancy data set*

Description

Draws from the `dyn_int_occu()` model: a dynamic (multi-season) occupancy process (ψ_1 , colonization γ , extinction ϵ) observed by S detection sources with per-source detection probability.

Usage

```
simulate_dyn_int_occu(
  N = 200,
  T.seasons = 4,
  S = 2,
  J = 3,
  psi1 = 0.5,
  gamma = 0.3,
  eps = 0.2,
  p = c(0.4, 0.6),
  field = NULL,
  trend = NULL,
```

```

    source.seasons = NULL,
    seed = NULL
  )

```

Arguments

N	Number of sites (default 200).
T.seasons	Number of seasons (default 4).
S	Number of detection sources (default 2).
J	Visits per source (scalar or length-S; default 3).
psi1, gamma, eps	Season-1 occupancy, colonization, extinction (defaults 0.5 / 0.3 / 0.2).
p	Per-source detection probabilities (length-S; default 0.4 / 0.6).
field	Optional per-site shared areal field (length N) added to the first-season occupancy logit – the shared field of the multi-season integrated spatial model (stIntP-GOcc). Default NULL (no field).
trend	Optional per-site varying-coefficient (SVC) areal field (length N); with a covariate $w \sim N(0, 1)$ stored in data, the first-season logit gains $w * \text{trend}$ on top of field – the svcTIntP-GOcc surface. Default NULL. When set, data carries a cell node index (1..N) and the covariate w for the bar spatial($\sim 1 + w \text{cell}$, graph).
source.seasons	Optional length-S list; source.seasons[[s]] is the integer vector of seasons source s observes (partial season overlap). The seasons a source does not cover are set to NA in its array – the staggered survey where sources rarely share the full season grid. Default NULL (every source observes every season).
seed	Optional random seed.

Value

A list with y (a length-S list of $[N \times J \times T]$ arrays), data, sources, and truth.

Examples

```

sim <- simulate_dyn_int_occu(N = 40, T.seasons = 3, seed = 1)
lapply(sim$y, dim)

```

simulate_dyn_occu	<i>Simulate temporal (multi-season) occupancy data</i>
-------------------	--

Description

Colonization and extinction are constant across the $n_seasons - 1$ transition intervals by default. Supplying beta.gamma and/or beta.epsilon makes the corresponding rate SEASON-VARYING: a per-(site, interval) covariate is drawn into an $[N \times (n_seasons - 1)]$ matrix column (gamma_cov / eps_cov) of the returned data, and the rate is $\text{plogis}(\text{beta}[1] + \text{beta}[2] * \text{cov})$. Fit these with colonization = ~ gamma_cov / extinction = ~ eps_cov (the matrix column drives the interval-indexed rate).

Usage

```
simulate_dyn_occu(
  N = 100,
  J = 4,
  n.seasons = 5,
  beta.occ = c(0.5),
  beta.det = c(0),
  gamma = 0.2,
  epsilon = 0.1,
  beta.gamma = NULL,
  beta.epsilon = NULL,
  beta.det.season = NULL,
  seed = NULL
)
```

Arguments

N	Number of sites (default 100).
J	Number of visits per season (default 4).
n.seasons	Number of seasons (default 5).
beta.occ	Initial occupancy coefficients.
beta.det	Detection coefficients.
gamma	Colonization probability (default 0.2); ignored when beta.gamma is given.
epsilon	Extinction probability (default 0.1); ignored when beta.epsilon is given.
beta.gamma	Optional c(intercept, slope) for a season-varying colonization logit driven by a drawn per-(site, interval) covariate.
beta.epsilon	Optional c(intercept, slope) for a season-varying extinction logit driven by a drawn per-(site, interval) covariate.
beta.det.season	Optional c(intercept, slope) for a season-varying detection logit driven by a drawn per-(site, season) covariate det_cov (a [N x T] matrix column of data); fit with detection = ~ det_cov.
seed	Random seed.

Value

A list with y (3D array), data, and truth.

Examples

```
sim <- simulate_dyn_occu(N = 40, J = 3, n.seasons = 4, seed = 1)
dim(sim$y)
```

simulate_fp_occu	<i>Simulate multistate false-positive occupancy data</i>
------------------	--

Description

Latent occupancy $z_i \sim \text{Bernoulli}(\psi_i)$ with logit $\psi_i = X_{\psi_i} \beta_{\psi_i}$, observed through J replicate visits emitting a detection state y in $\{0, 1, 2\}$: at an occupied site a visit detects with probability p_{11} and the detection is certain (state 2) with probability b else ambiguous (state 1); at an unoccupied site a visit yields a false-positive ambiguous detection (state 1) with probability p_{10} . Returns an $N \times J$ integer matrix in $\{0, 1, 2\}$ suitable for `tobs()` with `fp_occu()`.

Usage

```
simulate_fp_occu(
  N = 300,
  J = 5,
  n.occ.covs = 1,
  beta.psi = NULL,
  p11 = 0.6,
  p10 = 0.05,
  b = 0.5,
  seed = NULL
)
```

Arguments

<code>N</code>	Number of sites (default 300).
<code>J</code>	Number of visits (default 5).
<code>n.occ.covs</code>	Number of occupancy covariates (default 1).
<code>beta.psi</code>	Occupancy coefficients (logit). Default <code>c(qlogis(0.5), runif(n.occ.covs, -0.6, 0.6))</code> .
<code>p11, p10, b</code>	True detection, false-positive, and certain-classification probabilities (scalars; defaults 0.6, 0.05, 0.5).
<code>seed</code>	Optional random seed.

Value

A list with `y` ($N \times J$ state matrix), `data` (occupancy covariates), and `truth` (coefficients, per-site ψ_i , scalar $p_{11}/p_{10}/b$, latent z).

Examples

```
sim <- simulate_fp_occu(N = 50, J = 4, seed = 1)
table(sim$y)
```

simulate_gdistremoval *Simulate a joint distance + removal data set*

Description

Draws from the `gdistremoval()` model: site abundance $N \sim \text{Poisson}(\lambda)$, the detected birds cross-classified by a distance band (half-normal key, scale σ) and by a removal period (per-period capture r).

Usage

```
simulate_gdistremoval(
  N = 200,
  cutpoints = c(0, 10, 20, 30, 40),
  n.periods = 4L,
  transect = "line",
  n.abund.covs = 1,
  n.det.covs = 1,
  n.rem.covs = 1,
  beta.lambda = NULL,
  beta.sigma = NULL,
  beta.r = NULL,
  seed = NULL
)
```

Arguments

<code>N</code>	Number of sites (default 200).
<code>cutpoints</code>	Distance-bin edges $0 = c_0 < \dots < c_B$.
<code>n.periods</code>	Number of removal periods.
<code>transect</code>	"line" (default) or "point".
<code>n.abund.covs</code> , <code>n.det.covs</code> , <code>n.rem.covs</code>	Number of abundance / distance / removal covariates.
<code>beta.lambda</code> , <code>beta.sigma</code> , <code>beta.r</code>	Coefficients on the log-abundance, log-scale, and logit-removal arms (<code>c(intercept, slopes...)</code>). Defaults give moderate abundance / detection.
<code>seed</code>	Optional random seed.

Value

A list with `y` (distance-band counts), `y_rem` (removal-period counts), `data`, and `truth`.

Examples

```
sim <- simulate_gdistremoval(N = 50, seed = 1)
dim(sim$y)
dim(sim$y_rem)
```

simulate_int_occu	<i>Simulate integrated (multi-source) occupancy data</i>
-------------------	--

Description

Simulate integrated (multi-source) occupancy data

Usage

```
simulate_int_occu(
  N.total = 150,
  n.data = 2,
  J = c(4, 3),
  n.shared = 20,
  beta.occ = c(0.5, 0.3),
  beta.det = list(c(0.2, -0.4), c(-0.1, 0.3)),
  seed = NULL
)
```

Arguments

N.total	Total number of unique sites (default 150).
n.data	Number of data sources (default 2).
J	Vector of visits per source (default c(4, 3)).
n.shared	Number of sites shared across sources (default 20).
beta.occ	Occupancy coefficients (default c(0.5, 0.3)).
beta.det	List of detection coefficient vectors per source.
seed	Random seed.

Value

A list with y (list of matrices, each row named by the site of data it measures), data, site_maps, and truth.

Examples

```
sim <- simulate_int_occu(N.total = 60, n.shared = 10, seed = 1)
lapply(sim$y, dim)
```

simulate_jsdm	<i>Simulate joint species distribution (presence/absence) data</i>
---------------	--

Description

Presence $y_{\{i,s\}} \sim \text{Bernoulli}(\psi_{\{i,s\}})$ with $\text{logit } \psi_i = X \beta_{\cdot s}$, the per-species occupancy coefficients drawn from Gaussian community hyperpriors. Matches the `jsdm()` family, which observes presence directly (no detection process), so the response is an $N \times n_{\text{species}}$ presence matrix.

Usage

```
simulate_jsdm(  
  N = 100,  
  n.species = 10,  
  beta.comm.mean = c(0, 0.5),  
  beta.comm.sd = c(0.5, 0.3),  
  seed = NULL  
)
```

Arguments

<code>N</code>	Number of sites (default 100).
<code>n.species</code>	Number of species (default 10).
<code>beta.comm.mean</code>	Community-mean occupancy coefficients, length $1 + n_{\text{occ_covs}}$ (intercept first).
<code>beta.comm.sd</code>	Between-species SD of each occupancy coefficient (same length as <code>beta.comm.mean</code>).
<code>seed</code>	Random seed.

Value

A list with `y` (an $N \times n_{\text{species}}$ 0/1 presence matrix), `data`, and `truth` (per-species coefficients and the community hyperparameters).

Examples

```
sim <- simulate_jsdm(N = 60, n.species = 5, seed = 1)  
dim(sim$y)  # 60 sites x 5 species presence matrix
```

simulate_ms_abun

*Simulate community (multispecies) N-mixture abundance data***Description**

Per-species Royle (2004) N-mixture with Gaussian community hyperpriors: $\text{beta_lambda_s} \sim N(\mu_lambda, \text{Sigma_lambda}), \text{beta_p_s} \sim N(\mu_p, \text{Sigma_p})$, then $N_{\{s,i\}} \sim \text{Poisson}(\text{lambda}_{\{s,i\}})$ (or $\text{NegBin}(\mu = \text{lambda}, \text{size} = r_s)$ with a per-species size $r_s = \exp(\mu_log_r + b_logr_s)$) and $y_{\{s,i,j\}} \sim \text{Binomial}(N_{\{s,i\}}, p_{\{s,i,j\}})$. The returned y is a 3D array [$n_sites \times J \times n_species$] suitable for `tobs()` with `ms_abun()`.

Usage

```
simulate_ms_abun(
  n.species = 12,
  N = 80,
  J = 4,
  n.abund.covs = 1,
  n.det.covs = 1,
  mu.lambda = NULL,
  mu.p = NULL,
  sd.lambda = 0.5,
  sd.p = 0.4,
  mixture = c("poisson", "negbin", "zip", "zinb"),
  size = 3,
  sigma.logr = 0.4,
  omega = 0.3,
  sigma.omega = 0.4,
  graph = NULL,
  sigma.field = 0.6,
  seed = NULL
)
```

Arguments

<code>n.species</code>	Number of species (default 12).
<code>N</code>	Number of sites (default 80).
<code>J</code>	Number of replicate visits (default 4).
<code>n.abund.covs</code>	Number of abundance covariates (default 1).
<code>n.det.covs</code>	Number of detection covariates (default 1).
<code>mu.lambda</code>	Community-mean abundance coefficients <code>c(intercept, slopes...)</code> on the log scale. Default <code>c(log(3), rep(0.4, n.abund.covs))</code> .
<code>mu.p</code>	Community-mean detection coefficients on the logit scale. Default <code>c(0.3, rep(-0.3, n.det.covs))</code> .

sd.lambda	Per-coefficient community SD for the abundance arm (the sqrt-diagonal of Sigma_lambda). Length 1 (recycled) or 1 + n.abund.covs. Default 0.5.
sd.p	Per-coefficient community SD for the detection arm. Default 0.4.
mixture	Abundance mixing distribution: "poisson" (default), "negbin", or their zero-inflated counterparts "zip" / "zinb" (a per-species structural-zero share).
size	Community-mean negative-binomial size, equal to $\exp(\mu_{\log r})$ (ignored under Poisson). Default 3. The per-species sizes are $r_s = \exp(\mu_{\log r} + b_s^{\log r})$ with $b_s^{\log r} \sim N(0, \sigma_{\log r}^2)$.
sigma.logr	Standard deviation of the per-species log-dispersion random effect log_r_s (used only under mixture = "negbin" / "zinb"). Default 0.4. Set to 0 for a shared (single-r) community.
omega	Community-mean structural-zero probability, equal to $\text{plogis}(\mu_\omega)$ (used only under mixture = "zip" / "zinb"). Default 0.3. The per-species structural-zero probabilities are $\omega_s = \text{plogis}(\mu_\omega + b_s^\omega)$ with $b_s^\omega \sim N(0, \sigma_\omega^2)$.
sigma.omega	Standard deviation of the per-species structural-zero-logit random effect logit_omega_s (used only under zero-inflation). Default 0.4. Set to 0 for a shared (single-omega) community.
graph	Optional N x N 0/1 adjacency matrix. When supplied, a single shared spatial field f (one value per site) is drawn from a proper GMRF on the graph, centered and scaled to standard deviation sigma.field, and added to every species' abundance log-linear predictor ($\log \lambda_{\{s,i\}} = X_{\lambda_{i,s}} \cdot \beta_{\lambda_{i,s}} + f_i$). N is taken from nrow(graph). The truth field is returned in truth\$field.
sigma.field	Standard deviation of the shared spatial field (used only when graph is supplied). Default 0.6.
seed	Optional random seed.

Value

A list with y (3D count array), data (site covariate frame), species (species names), and truth (community means / SDs, the per-species coefficients, lambda, p, latent N, the shared field when graph is given, and – under negbin – mu_log_r, sigma_log_r, the per-species b_logr / r_s).

truth carries each community mean twice. mu_lambda, mu_p, mu_log_r and mu_omega are the POPULATION constants the per-species values were drawn around. mu_lambda_real, mu_p_real, mu_log_r_real and mu_omega_real are the mean of the values this seed actually drew, which sits $\text{sd} / \sqrt{n_species}$ from the constant – 0.12 on mu_log_r at sigma_logr = 0.5 and 18 species.

Which one to score against follows from the question. An interval for a community mean targets the POPULATION constant and its width includes that $\text{sd}^2 / n_species$ term, so a coverage measurement scores against mu_log_r. A point-recovery or interval-SCALE measurement scores against mu_log_r_real: against the constant, the draw carries about two thirds of the spread across seeds, and a seed block that happens to draw wide reads as a miscalibrated interval. That is what the 18-species fixture measured – its seed block drew the species means 18-21% wider than sigma_logr / sqrt(18), and both arms calibrate once the draw is put at its expectation. The _real entries are supplied so the split does not have to be rebuilt at each call site; tests/testthat/helper-community-mean.R is the assertion that consumes them.

Examples

```
sim <- simulate_ms_abun(n.species = 4, N = 30, J = 3, seed = 1)
dim(sim$y)
```

simulate_ms_count	<i>Simulate community relative-abundance (count / continuous) data</i>
-------------------	--

Description

Per-species GLMM with Gaussian community hyperpriors on the coefficients (the `ms_count()` / `spAbundance msAbund` model): $g(\mu_{\{s,i\}}) = X_i \beta_s$, $\beta_s \sim N(\beta_{\text{comm_mean}}, \text{diag}(\beta_{\text{comm_sd}}^2))$, $y_{\{s,i\}}$ drawn Poisson / negative-binomial (log link) or Gaussian (identity). No detection.

Usage

```
simulate_ms_count(
  N = 120,
  n.species = 10,
  beta.comm.mean = c(1, 0.5),
  beta.comm.sd = c(0.4, 0.3),
  response = c("poisson", "negbin", "gaussian", "binomial"),
  size = 2,
  size.log.sd = 0.3,
  sd = 1,
  trials = 10,
  seed = NULL
)
```

Arguments

<code>N</code>	Number of sites.
<code>n.species</code>	Number of species.
<code>beta.comm.mean</code> , <code>beta.comm.sd</code>	Community mean and SD of the per-species coefficients (intercept first). Length sets the number of covariates.
<code>response</code>	One of "poisson", "negbin", "gaussian", "binomial".
<code>size</code>	Negative-binomial community size (mean of the per-species <code>log_r</code>); <code>size.log.sd</code> is its across-species SD.
<code>size.log.sd</code>	Across-species SD of <code>log(size)</code> (negbin only).
<code>sd</code>	Gaussian residual SD.
<code>trials</code>	Binomial trial count (<code>response = "binomial"</code>): a scalar (shared across sites and species) or a length- <code>N</code> per-site vector. Default 10.
<code>seed</code>	Optional RNG seed.

Value

A list with `y` (an $N \times n_species$ matrix), `data`, and `truth`.

Examples

```
sim <- simulate_ms_count(N = 50, n.species = 4, seed = 1)
head(sim$y)
```

<code>simulate_ms_distance</code>	<i>Simulate community (multispecies) binned distance-sampling data</i>
-----------------------------------	--

Description

Per-species binned distance sampling with Gaussian community hyperpriors: $\beta_{\lambda_s} \sim N(\mu_{\lambda}, \text{diag}(\text{sd}_{\lambda}^2))$, $\beta_{\sigma_s} \sim N(\mu_{\sigma}, \text{diag}(\text{sd}_{\sigma}^2))$, then $N_{\{s,i\}} \sim \text{Poisson}(\lambda_{\{s,i\}})$ and the detected individuals allocated to distance bins by the half-normal detection function. The returned `y` is a 3D array [$n_sites \times n_bins \times n_species$] suitable for `tobs()` with `ms_distance()`.

Usage

```
simulate_ms_distance(
  n.species = 10,
  N = 100,
  cutpoints = c(0, 25, 50, 75, 100),
  transect = c("line", "point"),
  key = c("halfnorm", "hazard"),
  shape = 0,
  n.abund.covs = 1,
  n.det.covs = 0,
  mu.lambda = NULL,
  mu.sigma = NULL,
  sd.lambda = 0.4,
  sd.sigma = 0.2,
  n.factors = 0,
  load.sd = 0.5,
  field = NULL,
  quad.order = 64L,
  seed = NULL
)
```

Arguments

<code>n.species</code>	Number of species (default 10).
<code>N</code>	Number of sites (default 100).
<code>cutpoints</code>	Distance-bin edges (default <code>c(0, 25, 50, 75, 100)</code>).

transect	Transect geometry: "line" (default) or "point".
key	Detection function: "halfnorm" (default) or "hazard".
shape	Hazard-rate log-shape, shared across species (ignored under the half-normal key). Default 0.
n.abund.covs, n.det.covs	Number of abundance / detection-scale covariates (default 1 and 0).
mu.lambda	Community-mean abundance coefficients on the log scale. Default $c(\log(30), \text{rep}(0.4, n.abund.covs))$.
mu.sigma	Community-mean detection-scale coefficients on the log scale. Default $c(\log(40), \text{rep}(0, n.det.covs))$.
sd.lambda, sd.sigma	Per-coefficient community SDs. Default 0.4 and 0.2.
n.factors	If > 0 , add $n.factors$ per-site latent factors with per-species loadings to log lambda (the lfMsDS truth). Default 0.
load.sd	SD of the factor loadings (default 0.5).
field	Optional length-N shared spatial field added to every species' log lambda. When given alongside $n.factors > 0$ the loadings are centred across species, so the field owns the shared spatial mean.
quad.order	Gauss-Legendre nodes per bin used to integrate the per-bin detection probabilities (default 64, matching <code>ms_distance()</code>). Set it to the quad.order the model will be fit at: the rule is (cutpoints, transect, quad.order), so a different order integrates a different pi and the data would come from a model the fit does not use.
seed	Optional random seed.

Value

A list with y, data, species, cutpoints, and truth (community means / SDs, per-species coefficients, lambda, sigma, and – with factors – the loadings, factors and the implied residual correlation cor_res). The latent N is drawn inside the shared C++ simulator and is not returned.

Examples

```
sim <- simulate_ms_distance(n.species = 4, N = 30, seed = 1)
dim(sim$y)
```

simulate_ms_dyn_occu *Simulate temporal multi-species occupancy data*

Description

Simulate temporal multi-species occupancy data

Usage

```
simulate_ms_dyn_occu(
  N = 50,
  J = 3,
  n.species = 5,
  n.seasons = 4,
  beta.comm.mean = c(0),
  beta.comm.sd = c(0.5),
  gamma = 0.15,
  epsilon = 0.1,
  field = NULL,
  trend = NULL,
  seed = NULL
)
```

Arguments

N	Number of sites (default 50).
J	Visits per season (default 3).
n.species	Number of species (default 5).
n.seasons	Number of seasons (default 4).
beta.comm.mean	Community mean for occupancy (default c(0)).
beta.comm.sd	Community SD for occupancy (default c(0.5)).
gamma	Colonization probability (default 0.15).
epsilon	Extinction probability (default 0.1).
field	Optional per-site shared areal field (length N) added to the first-season occupancy logit of every species – the shared field of the community dynamic-spatial model (stMsPGOcc). Default NULL (no field).
trend	Optional per-site shared varying-coefficient field (length N), added to the first-season occupancy logit as $\text{trend}_i * x_i$ (weighted by the site covariate x) – the varying-coefficient field of the community dynamic-spatial model (svcTM-sPGOcc). Default NULL.
seed	Random seed.

Value

A list with y (4D array), data, and truth.

See Also

[ms_dyn_occu\(\)](#), the family this simulates for.

Examples

```
sim <- simulate_ms_dyn_occu(N = 30, J = 2, n.species = 3, n.seasons = 3,
                             seed = 1)
dim(sim$y)
```

simulate_ms_int_occu *Simulate integrated multi-species occupancy data*

Description

Simulate integrated multi-species occupancy data

Usage

```
simulate_ms_int_occu(  
  N = 100,  
  J = c(3, 4),  
  n.species = 5,  
  n.data = 2,  
  seed = NULL  
)
```

Arguments

N	Number of sites (default 100).
J	Vector of visits per source (default c(3, 4)).
n.species	Number of species (default 5).
n.data	Number of data sources (default 2).
seed	Random seed.

Value

A list with y (list of 3D arrays), data, and truth.

See Also

[ms_int_occu\(\)](#), the family this simulates for.

Examples

```
sim <- simulate_ms_int_occu(N = 40, n.species = 3, seed = 1)  
lapply(sim$y, dim)
```

simulate_ms_occu*Simulate multi-species occupancy data*

Description

Simulate multi-species occupancy data

Usage

```
simulate_ms_occu(  
  N = 100,  
  J = 4,  
  n.species = 10,  
  beta.comm.mean = c(0, 0.5),  
  beta.comm.sd = c(0.5, 0.3),  
  alpha.comm.mean = c(0),  
  alpha.comm.sd = c(0.5),  
  seed = NULL  
)
```

Arguments

N	Number of sites (default 100).
J	Number of visits (default 4).
n.species	Number of species (default 10).
beta.comm.mean	Community mean for occupancy (default c(0, 0.5)).
beta.comm.sd	Community SD for occupancy (default c(0.5, 0.3)).
alpha.comm.mean	Community mean for detection (default c(0)).
alpha.comm.sd	Community SD for detection (default c(0.5)).
seed	Random seed.

Value

A list with y (3D array), data, and truth.

Examples

```
sim <- simulate_ms_occu(N = 40, J = 3, n.species = 4, seed = 1)  
dim(sim$y)
```

simulate_ms_occu_cover

Simulate community (multispecies) joint occupancy-cover data

Description

Per-species joint occupancy-detection + cover model with Gaussian community hyperpriors on the per-species coefficients of all three arms: $\text{beta_occ_s} \sim N(\mu_{\text{occ}}, \text{diag}(\text{sd_occ}^2))$, $\text{beta_p_s} \sim N(\mu_{\text{p}}, \text{diag}(\text{sd_p}^2))$, $\text{beta_pos_s} \sim N(\mu_{\text{pos}}, \text{diag}(\text{sd_pos}^2))$, then per cell $z_{\{s,i\}} \sim \text{Bernoulli}(\psi_{\{s,i\}})$, per visit $y_{\{s,i,j\}} \mid z = 1 \sim \text{Bernoulli}(p_{\{s,i,j\}})$, and per detected visit $c_{\{s,i,j\}} \sim \text{f_pos}(\eta_{\text{pos}_{\{s,i,j\}}}, \text{diag}(\text{sd_pos}^2))$. The site and visit covariates are shared across species (community covariates). The returned y / y_{pos} are 3D arrays [$n_{\text{sites}} \times J \times n_{\text{species}}$] suitable for `tobs()` with `ms_occu_cover()` (y_{pos} is NA where not detected).

Usage

```
simulate_ms_occu_cover(
  n.species = 12,
  N = 120,
  J = 5,
  n.occ.covs = 1,
  n.det.covs = 1,
  n.pos.covs = 1,
  mu.occ = NULL,
  mu.p = NULL,
  mu.pos = NULL,
  sd.occ = 0.5,
  sd.p = 0.4,
  sd.pos = 0.4,
  positive = c("lognormal", "beta", "gaussian"),
  phi = 30,
  sigma.pos = 0.4,
  seed = NULL
)
```

Arguments

<code>n.species</code>	Number of species (default 12).
<code>N</code>	Number of sites / cells (default 120).
<code>J</code>	Number of replicate visits (default 5).
<code>n.occ.covs</code> , <code>n.det.covs</code> , <code>n.pos.covs</code>	Number of covariates on each arm (drawn IID standard normal, shared across species).
<code>mu.occ</code> , <code>mu.p</code> , <code>mu.pos</code>	Community-mean coefficient vectors <code>c(intercept, slopes...)</code> on each arm's link scale. Defaults pick weakly-informative values.

sd.occ, sd.p, sd.pos	Per-coefficient community SD on each arm (length 1, recycled, or one per coefficient). Default 0.5 / 0.4 / 0.4.
positive	"lognormal" (default) or "beta".
phi	Beta precision when positive = "beta" (default 30).
sigma.pos	Lognormal residual SD when positive = "lognormal" (default 0.4).
seed	Optional integer seed.

Value

A list with `y` (3D detection array), `y_pos` (3D cover array, NA where not detected), `data` (per-cell covariate frame), `visit_data` (per-visit covariate frame, N*J rows in site-major order), `species` (species names), and `truth` (community means / SDs, per-species coefficients, the dispersion, and the latent state).

Examples

```
sim <- simulate_ms_occu_cover(n.species = 4, N = 30, J = 3, seed = 1)
dim(sim$y)
```

```
simulate_ms_occu_cover_spatial
```

Simulate a reduced-rank spatial-factor community occu_cover data set ($K = 1$)

Description

Generates occupancy / detection / cover data for `n.species` species sharing one latent ICAR spatial factor `w` on the cell graph, with per-species loadings `L_s` on the occupancy state predictor and Gaussian community priors on the per-species arm coefficients. This is the ground-truth generator for the Stage-1 reduced-rank spatial JSMD.

Usage

```
simulate_ms_occu_cover_spatial(
  adj,
  n.species = 8L,
  J = 4L,
  K = 1L,
  n.occ.covs = 1L,
  n.det.covs = 1L,
  n.pos.covs = 1L,
  mu.occ = NULL,
  mu.p = NULL,
  mu.pos = NULL,
  sd.occ = 0.4,
```

```

sd.p = 0.4,
sd.pos = 0.3,
mean.load = 0,
sd.load = 1,
cover.factor = FALSE,
mean.load.pos = 0,
sd.load.pos = 1,
field = c("icar", "car_proper", "bym2"),
rho = 0.9,
phi = 0.7,
sigma.pos = 0.4,
positive = c("lognormal", "beta"),
seed = NULL
)

```

Arguments

<code>adj</code>	N x N 0/1 adjacency matrix of the cell graph (required); N cells.
<code>n.species</code>	Number of species.
<code>J</code>	Number of detection visits per cell.
<code>K</code>	Number of shared latent spatial factors ($1 \leq K \leq n.species$). $K = 1$ is the Stage-1 single-field case (loading vector, field vector); $K > 1$ draws K ICAR fields with lower-triangular, positive-diagonal loadings and returns the $S \times K$ loading matrix / $N \times K$ field matrix.
<code>n.occ.covs, n.det.covs, n.pos.covs</code>	Number of (Gaussian) covariates on the occupancy, detection, and cover arms; each arm also has an intercept.
<code>mu.occ, mu.p, mu.pos</code>	Community mean coefficient vectors (intercept first). NULL picks sensible defaults of the right length.
<code>sd.occ, sd.p, sd.pos</code>	Community RE SDs (diagonal <code>Sigma_.</code>); scalar (recycled) or per-coefficient.
<code>mean.load, sd.load</code>	Mean and SD of the per-species loadings <code>L_s</code> .
<code>cover.factor</code>	Logical; when TRUE the same shared fields <code>W</code> also load on the cover (positive) predictor through a free $S \times K$ loading matrix <code>L_pos</code> (the cover-arm factor). The cover-factor draws are gated, so FALSE (the default) reproduces the no-factor RNG stream exactly. <code>truth\$L_pos</code> carries the generating cover loadings.
<code>mean.load.pos, sd.load.pos</code>	Mean and SD of the cover-arm loadings <code>L_pos</code> (used only when <code>cover.factor = TRUE</code>).
<code>field</code>	Areal structure of the shared latent factors: "icar" (improper intrinsic CAR, the default), "car_proper" (proper CAR with a correlation ρ , field precision τ ($D - \rho A$)), or "bym2" (the Riebler 2016 reparameterised convolution with a spatial-variance fraction ϕ). "icar" reproduces the Stage 1-2 RNG stream byte for byte; <code>truth\$field</code> / <code>truth\$rho</code> / <code>truth\$phi</code> carry the choice.

rho	Proper-CAR correlation in $[\emptyset, 1)$, used only when field = "car_proper" (the rho $\rightarrow 1$ limit is the ICAR field).
phi	BYM2 spatial-variance fraction in $[\emptyset, 1]$, used only when field = "bym2" (phi = 1 is the pure ICAR field, phi = $\emptyset$ pure iid).
sigma.pos	Lognormal cover residual SD (on the log scale).
positive	Cover family; only "lognormal" in Stage 1.
seed	Optional RNG seed.

Value

A list with y (N x J x n_species 0/1 detections), y_pos (N x J x n_species cover, NA off detected visits), data (cell-level covariate frame), species, and truth (the canonical-form generating parameters: μ_* , sd_* , per-species b_* , loadings L, field w, ψ , z).

Examples

```
adj <- matrix(0, 10, 10)
adj[cbind(1:9, 2:10)] <- 1
adj <- adj + t(adj)
sim <- simulate_ms_occu_cover_spatial(adj, n.species = 4, J = 3, seed = 1)
dim(sim$y)
```

simulate_occu	<i>Simulate single-species occupancy data</i>
---------------	---

Description

Simulate single-species occupancy data

Usage

```
simulate_occu(
  N = 100,
  J = 4,
  n.occ.covs = 2,
  n.det.covs = 1,
  beta.occ = NULL,
  beta.det = NULL,
  n.visit.groups = 0L,
  sigma.visit = 1,
  seed = NULL
)
```

Arguments

N	Number of sites (default 100).
J	Number of visits (default 4).
n.occ.covs	Number of occupancy covariates (default 2).
n.det.covs	Number of detection covariates (default 1).
beta.occ	Occupancy coefficients (auto-generated if NULL).
beta.det	Detection coefficients (auto-generated if NULL).
n.visit.groups	Number of groups a visit can belong to (an observer, say), each adding an effect to the detection logit of the visits it made. Visits are assigned to groups uniformly at random. 0 (default) simulates no such effect.
sigma.visit	Standard deviation of the per-group detection effects, drawn from $N(0, \text{sigma.visit}^2)$.
seed	Random seed.

Value

A list with y, data, and truth. With n_visit_groups > 0 it also carries visits, a data frame with one row per site-visit in site-major order and the factor column visit_group, and truth gains b_visit (the group effects) and sigma_visit.

Examples

```
sim <- simulate_occu(N = 50, J = 3, seed = 1)
dim(sim$y)

obs <- simulate_occu(N = 50, J = 3, n.visit.groups = 5, seed = 1)
table(obs$visits$visit_group)
```

```
simulate_occu_categorical
```

Simulate from an occu_categorical model

Description

Simulate from an occu_categorical model

Usage

```
simulate_occu_categorical(
  N = 500L,
  beta.occ = c(0.2, 0.8),
  beta.class = NULL,
  seed = NULL
)
```

Arguments

N	number of units.
beta.occ	length-2 presence coefficients (intercept, slope on x).
beta.class	a 2 × (K-1) matrix of class coefficients (rows intercept / slope on x; columns the non-baseline classes), or NULL for a built-in 3-class default.
seed	optional RNG seed.

Value

A list with data (a data frame with x), y (length-N integer in 0..K), and truth.

Examples

```
sim <- simulate_occu_categorical(N = 100, seed = 1)
table(sim$y)
```

simulate_occu_cover	<i>Simulate joint occupancy-detection + cover data</i>
---------------------	--

Description

Per-cell mixture: latent $z_i \sim \text{Bernoulli}(\psi_i)$, per-visit detection $y_{ij} \mid z_i = 1 \sim \text{Bernoulli}(p_{ij})$, per-visit cover $y_{\text{pos}_{ij}} \mid y_{ij} = 1$ drawn from positive ("beta" or "lognormal") on the cover-arm linear predictor. Used by the recovery test and as the generator for the joint occupancy + cover hurdle family (see [occu_cover\(\)](#)).

Usage

```
simulate_occu_cover(
  N = 200L,
  J = 4L,
  n.occ.covs = 1L,
  n.det.covs = 1L,
  n.pos.covs = 1L,
  beta.occ = NULL,
  beta.p = NULL,
  beta.pos = NULL,
  positive = c("lognormal", "beta", "gaussian"),
  phi = 30,
  sigma.pos = 0.4,
  adj = NULL,
  sigma = 0.6,
  alpha = 1,
  trend = FALSE,
  sigma.trend = 0.6,
  alpha.trend = 1,
```

```

pos.field = FALSE,
sigma.pos.int = 0.5,
sigma.pos.trend = 0.6,
det.field = FALSE,
sigma.p.int = 0.5,
sigma.p.trend = 0.6,
re.det.groups = NULL,
sigma.re.p = 0.7,
re.pos.groups = NULL,
sigma.re.pos = 0.7,
re.det = NULL,
seed = NULL
)

```

Arguments

N	Number of sites (cells).
J	Number of visits per site.
n.occ.covs, n.det.covs, n.pos.covs	Number of covariates on each arm (drawn IID standard normal).
beta.occ, beta.p, beta.pos	Coefficient vectors c(intercept, slopes). Defaults pick weakly-informative values: psi intercept at logit(0.4), p intercept at logit(0.5), cover intercept on the appropriate link.
positive	"beta" or "lognormal".
phi	Beta precision when positive = "beta" (default 30).
sigma.pos	Lognormal residual SD when positive = "lognormal" (default 0.4).
adj	Optional N x N adjacency matrix. When supplied, generates the shared ICAR field; when NULL, the simulator is non-spatial (matches v1).
sigma	Spatial field amplitude (used only when adj is supplied), in the geo-mean-marginal-SD (Sorbye-Rue) convention: the field is drawn with geometric-mean marginal variance 1 before scaling by sigma, so sigma is directly comparable to a fitted field_sd (fit\$spatial\$field_sd_mean on the nested_laplace path, fit\$hyper_draws[, "field_sd"] on nuts) – never to the raw sigma a nested_laplace fit reports, which is the amplitude against the unscaled intrinsic precision and differs by sqrt(scale_q).
alpha	Cover-arm scaling on the shared field (used only when adj is supplied). 1.0 = arms see the field identically; positive = same sign, negative = opposite.
trend	Logical; when TRUE (and adj is supplied) a SECOND shared ICAR field f2 (a spatially-varying temporal trend) is generated on the same graph, weighted by a per-cell covariate time drawn IID standard normal. The trend enters the occupancy and cover predictors as sigma.trend * time_i * f2[i] (occupancy) and alpha.trend * sigma.trend * time_i * f2[i] (cover); the detection predictor is unaffected. The time covariate is per-cell and broadcast to every visit of that cell.
sigma.trend	Trend-field amplitude (used only when trend = TRUE).

alpha.trend	Cover-arm scaling on the trend field (used only when trend = TRUE).
pos.field	Logical; when TRUE (and adj is supplied) draw an INDEPENDENT areal field on the cover (positive) arm only – an intercept field plus a time-weighted trend field, each unrelated to the occupancy field and with no alpha copy. Adds a time column to the returned data and reports g_0 / g_1 (the two fields) and their SDs in truth. Fit by placing <code>spatial(~ 1 + time cell, graph = adj)</code> in the positive formula.
sigma.pos.int	Cover-arm intercept-field SD (used only when pos.field = TRUE).
sigma.pos.trend	Cover-arm trend-field SD (used only when pos.field = TRUE).
det.field	Logical; when TRUE (and adj is supplied) draw an INDEPENDENT areal field on the detection arm only – an intercept field plus a time-weighted trend field, unrelated to the occupancy and cover fields and with no alpha copy, so detection varies spatially on its own. Fit by placing <code>spatial(~ 0 + time cell, graph = adj, to = "detection")</code> in the detection formula.
sigma.p.int	Detection-arm intercept-field SD (used only when det.field = TRUE).
sigma.p.trend	Detection-arm trend-field SD (used only when det.field = TRUE).
re.det.groups	Optional integer ≥ 2 : the number of levels of a per-visit detection random intercept (a habitat factor on <code>visit_data</code> , levels <code>hab1..K</code>), drawn $b_g \sim N(0, \sigma_{re.p}^2)$ and centred. NULL (default) adds no detection RE. Recover it with <code>detection = ~ ... + (1 habitat)</code> .
sigma.re.p	SD of the <code>re.det.groups</code> random intercept (default 0.7).
re.pos.groups	Optional integer ≥ 2 : the number of levels of a per-visit COVER-arm random intercept (a habitat factor on <code>visit_data</code> , levels <code>hab1..K</code>), drawn $b_g \sim N(0, \sigma_{re.pos}^2)$ and centred, added to the positive-cover linear predictor. NULL (default) adds no cover RE. Recover it with <code>positive = ~ ... + (1 habitat)</code> under <code>cover.aggregate = "none"</code> . Truth in <code>truth\$b_pos_re / truth\$sigma_re_pos</code> .
sigma.re.pos	SD of the <code>re.pos.groups</code> random intercept (default 0.7).
re.det	Optional named list of FURTHER per-visit detection random effects, for crossed / nested / slope designs. Each element <code>list(K =, sigma =, prefix =, nested_in =, slope_cov =, sigma_slope =, rho =)</code> adds a factor column (levels <code><prefix>1..K</code>). Without <code>slope_cov</code> it is a centred $N(0, \sigma^2)$ random intercept; <code>nested_in = "<name>"</code> nests its codes within a previously listed grouping (matching (1 parent/child)), otherwise crossed. With <code>slope_cov = "<column>"</code> (a per-visit covariate, generated $N(0, \sigma_{sd})$ if absent, <code>slope_sd</code> default 1) it is a random slope: a slope-only uncorrelated block when <code>rho</code> is unset, or a correlated intercept + slope block (covariance from <code>sigma / sigma_slope / rho</code>) when <code>rho</code> is given. Truth is returned in <code>truth\$re_det[[name]]</code> (named by the level label a fit reconstructs): <code>b / b_slope</code> BLUP vectors, or the <code>B</code> BLUP matrix plus <code>s0 / s1 / rho</code> for a correlated slope.
seed	Optional integer seed.

Details

When `adj` is supplied (a square adjacency matrix), an ICAR field $f[1..N]$ is drawn from $MVN(0, Q^-)$ (with sum-to-zero constraint), and the linear predictors become

```
eta_psi_i = X_psi[i, ] %*% beta_occ + sigma * f[i]
eta_pos_ij = X_pos[i, ] %*% beta_pos + alpha * sigma * f[i]
```

This is the shared-field parameterisation (field SD sigma, cover-arm copy amplitude alpha) that the nested-Laplace `occu_cover()` fit estimates.

Value

A list with `y` (N x J detection matrix), `y_pos` (N x J cover matrix, NA where not detected), `data` (per-site covariate frame, gaining a time column when `trend = TRUE`), `visit_data` (per-visit covariate frame, N*J rows in site-major order), and `truth` (the coefficients, dispersion, and field(s) if generated; `f2`, `sigma_trend`, `alpha_trend`, and time when `trend = TRUE`).

Examples

```
sim <- simulate_occu_cover(N = 50, J = 3, seed = 1)
dim(sim$y)
```

<code>simulate_occu_multi</code>	<i>Simulate a multi-species co-occurrence occupancy data set</i>
----------------------------------	--

Description

Draws from the `occu_multi()` model: a joint occupancy state for `S` species from the log-linear (first + second order) occupancy model, then per-species site-level detections at the observed visit pattern.

Usage

```
simulate_occu_multi(
  S = 2,
  N = 300,
  J = 4,
  n.state.covs = 1,
  beta.first = NULL,
  beta.second = NULL,
  beta.p = NULL,
  seed = NULL
)
```

Arguments

<code>S</code>	Number of species (default 2).
<code>N</code>	Number of sites (default 300).
<code>J</code>	Number of replicate visits (default 4).
<code>n.state.covs</code>	Number of shared occupancy covariates (default 1).

beta.first	Length-S list of first-order natural-parameter coefficients $c(\text{intercept}, \text{slopes} \dots)$. Default moderate occupancy.
beta.second	Length-choose(S, 2) list of second-order (interaction) coefficients (in $\text{combn}(S, 2)$ order). Default a single positive interaction for $S = 2$, else 0.
beta.p	Length-S list of detection coefficients (logit). Default $c(\text{qlogis}(0.5), \dots)$.
seed	Optional random seed.

Value

A list with y (a length-S list of $N \times J$ matrices), data, species, and truth.

Examples

```
sim <- simulate_occu_multi(S = 2, N = 50, J = 3, seed = 1)
length(sim$y)
```

```
simulate_occu_multiscale_cover
```

Simulate a three-level occupancy + cover hurdle data set

Description

Draws cell occupancy $z_c \sim \text{Bernoulli}(\psi_c)$, plot availability $a_{cj} \mid z_c=1 \sim \text{Bernoulli}(\theta_{cj})$, visit detection $y_{cjk} \mid a_{cj}=1 \sim \text{Bernoulli}(p_{cjk})$, and the cover hurdle $\text{cover}_{cjk} \mid y_{cjk}=1 \sim f_{\text{pos}}$, with a shared areal ICAR field on the occupancy (σ) and cover ($\alpha * \sigma$) arms. The data-generating process for `occu_multiscale_cover()` recovery tests.

Usage

```
simulate_occu_multiscale_cover(
  n.cells = 60L,
  plots.per.cell = 4L,
  visits.per.plot = 2L,
  beta.psi = c(0.4, 0.6),
  beta.theta = c(0.2, 0.5),
  beta.p = c(0, 0.5),
  beta.pos = c(log(0.1), -0.4),
  positive = c("lognormal", "beta", "gaussian"),
  phi = 0.35,
  adj = NULL,
  sigma = 0.7,
  alpha = 1,
  trend = FALSE,
  sigma.trend = 0.7,
  alpha.trend = 1,
  seed = NULL
)
```

Arguments

<code>n.cells</code>	number of areal cells (graph nodes).
<code>plots.per.cell</code>	plots (the availability units) per cell.
<code>visits.per.plot</code>	detection replicate visits per plot.
<code>beta.psi, beta.theta, beta.p, beta.pos</code>	length-2 <code>c(intercept, slope)</code> coefficients for the four arms (one covariate each).
<code>positive</code>	"lognormal" or "beta" cover arm.
<code>phi</code>	cover dispersion (lognormal log-scale SD, or beta precision).
<code>adj</code>	<code>n.cells</code> x <code>n.cells</code> adjacency; default a 1-D chain.
<code>sigma</code>	areal-field marginal SD on the occupancy arm.
<code>alpha</code>	cover-arm field scaling (<code>alpha * sigma</code> on the cover arm).
<code>trend</code>	add a second, spatially-varying-coefficient (SVC) areal field weighted per cell by a cell-level covariate <code>tcov</code> (added to data): <code>tcov_c * sigma.trend * f_trend[c]</code> on occupancy and <code>tcov_c * alpha.trend * sigma.trend * f_trend[c]</code> on cover. Default FALSE.
<code>sigma.trend, alpha.trend</code>	trend-field marginal SD and its cover-arm scaling (used only when <code>trend = TRUE</code>).
<code>seed</code>	optional RNG seed.

Value

A list with `y`, `y_pos` (`[n_plots x visits_per_plot]`), the plot-level data (cell id `cell`, covariates), `adj`, and `truth`.

Examples

```
sim <- simulate_occu_multiscale_cover(n.cells = 10, plots.per.cell = 2,
                                     visits.per.plot = 2, seed = 1)
dim(sim$y)
```

`simulate_occu_ttd`

Simulate a time-to-detection occupancy data set

Description

Draws from the `occu_ttd()` model: site occupancy $z \sim \text{Bernoulli}(\psi)$ and, at occupied sites, exponential time-to-detection at rate `lambda` over a survey of length `Tmax` (a non-detection is censored at `Tmax`).

Usage

```
simulate_occu_ttd(  
  N = 200,  
  J = 4,  
  n.psi.covs = 1,  
  n.rate.covs = 1,  
  beta.psi = NULL,  
  beta.rate = NULL,  
  Tmax = 3,  
  seed = NULL  
)
```

Arguments

N	Number of sites (default 200).
J	Number of replicate surveys per site (default 4).
n.psi.covs, n.rate.covs	Number of occupancy / rate covariates.
beta.psi	Occupancy coefficients $c(\text{intercept}, \text{slopes} \dots)$ (logit). Default $c(\text{qlogis}(0.6), \text{runif}(n.\text{psi.covs}, -0.5, 0.5))$.
beta.rate	Log detection-rate coefficients $c(\text{intercept}, \text{slopes} \dots)$. Default $c(\log(0.6), \text{runif}(n.\text{rate.covs}, -0.4, 0.4))$.
Tmax	Survey length (scalar). Default 3.
seed	Optional random seed.

Value

A list with y (N x J TTD matrix), data, Tmax, and truth.

Examples

```
sim <- simulate_occu_ttd(N = 50, J = 3, seed = 1)  
dim(sim$y)
```

simulate_removal	<i>Simulate removal-sampling abundance data</i>
------------------	---

Description

Latent abundance $N_i \sim \text{Poisson}(\lambda_i)$ (or $\text{NegBin}(\lambda_i, \text{size})$) with $\log \lambda_i = X_{\lambda} \beta_{\lambda}$ observed through K ordered removal passes: pass k removes $\text{Binomial}(N_i - \sum_{l < k} y_{il}, p_{ik})$ of the individuals still present, with $\logit p_{ik} = X_p \beta_p$ (site-level detection here). The returned y is an N x K integer matrix of per-pass removals suitable for `tobs()` with `removal()`.

Usage

```
simulate_removal(
  N = 100,
  K = 4,
  n.abund.covs = 2,
  n.det.covs = 1,
  beta.lambda = NULL,
  beta.p = NULL,
  mixture = c("poisson", "negbin"),
  size = 3,
  seed = NULL
)
```

Arguments

N	Number of sites (default 100).
K	Number of removal passes (default 4).
n.abund.covs	Number of abundance covariates (default 2).
n.det.covs	Number of detection covariates (default 1).
beta.lambda	Abundance coefficients on the log scale. Default <code>c(log(6), runif(n.abund.covs, -0.5, 0.5))</code> .
beta.p	Detection coefficients on the logit scale. Default <code>c(0.4, runif(n.det.covs, -0.5, 0.5))</code> .
mixture	"poisson" (default) or "negbin".
size	Negative-binomial size <i>r</i> (mixture = "negbin" only, default 3).
seed	Optional random seed.

Value

A list with `y` (N x K removal matrix), `data` (covariate data frame), and `truth` (coefficients, per-site `lambda`, `p`, latent `N`, mixture/size).

Examples

```
sim <- simulate_removal(N = 50, K = 3, seed = 1)
dim(sim$y)
```

simulate_royle_nichols

Simulate Royle-Nichols occupancy data

Description

Latent abundance $N_i \sim \text{Poisson}(\lambda_i)$, $\log \lambda_i = X_i \beta_\lambda$, and per-visit detection $y_{ij} \sim \text{Bernoulli}(1 - (1 - r_{ij})^{N_i})$ with per-individual detection logit $r_{ij} = \beta_r$. Detection is site-level by default; supplying `beta.r.visit` draws a per-visit covariate w and makes logit $r_{ij} = \beta_r + \beta_{r_visit} * w_{ij}$ visit-varying. Matches the `royle_nichols()` family; the response is an $N \times J$ 0/1 detection-history matrix.

Usage

```
simulate_royle_nichols(
  N = 200,
  J = 5,
  beta.lambda = c(0.3, 0.5),
  beta.r = -0.8,
  beta.r.visit = NULL,
  seed = NULL
)
```

Arguments

<code>N</code>	Number of sites (default 200).
<code>J</code>	Number of visits per site (default 5).
<code>beta.lambda</code>	Log-abundance coefficients <code>c(intercept, slope_on_x)</code> .
<code>beta.r</code>	Per-individual detection logit (a scalar intercept).
<code>beta.r.visit</code>	Optional scalar slope on a drawn per-visit detection covariate w . When supplied, detection varies by visit and the returned list carries <code>visits = list(w =)</code> (an $N \times J$ matrix) for <code>tobs(..., visits =)</code> .
<code>seed</code>	Random seed.

Value

A list with `y` ($N \times J$ 0/1 matrix), `data`, `truth` (`beta.lambda`, `beta.r`, `beta.r.visit`, realised abundance N , λ , per-site or per-visit r), and, when `beta.r.visit` is supplied, `visits`.

Examples

```
sim <- simulate_royle_nichols(N = 100, J = 4, seed = 1)
dim(sim$y)
# Visit-varying detection:
sv <- simulate_royle_nichols(N = 100, J = 4, beta.r.visit = 0.8, seed = 1)
names(sv$visits)
```

simulate_t_occu

*Simulate multi-season occupancy with an AR1 year effect (tPGOcc)***Description**

Simulate multi-season occupancy with an AR1 year effect (tPGOcc)

Usage

```
simulate_t_occu(
  N = 150,
  T.seasons = 8,
  J = 3,
  beta.occ = c(0.2),
  p = 0.4,
  rho = 0.6,
  sigma = 0.7,
  seed = NULL
)
```

Arguments

N	Number of sites (default 150).
T.seasons	Number of seasons / years (default 8).
J	Visits per season (default 3).
beta.occ	Occupancy coefficients (intercept first; default $c(0.2)$).
p	Detection probability (default 0.4).
rho	AR1 correlation of the year effect (default 0.6).
sigma	AR1 innovation SD of the year effect (default 0.7).
seed	Optional random seed.

Value

A list with y (3D array [N x T x J]), data, and truth.

See Also

[t_occu\(\)](#), the family this simulates for.

Examples

```
sim <- simulate_t_occu(N = 30, T.seasons = 4, J = 2, seed = 1)
dim(sim$y)
```

summary.tobs_data	<i>Summarise occupancy data</i>
-------------------	---------------------------------

Description

Returns detection statistics including naive occupancy, naive detection, per-visit rates, and detection frequency table.

Usage

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

Arguments

object	An tobs_data object.
...	Ignored.

Value

An tobs_data_summary object (printed automatically).

summary.tobs_fit	<i>Summary for tobs_fit, with skewness column when simplified Laplace is used</i>
------------------	---

Description

Extends summary.tulpa_fit with an extra skew column populated from object\$skew when the fit was produced with an SLA method (method = "laplace_sla" / "nested_laplace_sla"). All quantile / mean / sd columns come from object\$draws, which under simplified Laplace are skew-normal samples – so 2.5%/97.5% quantiles are already SLA-corrected.

Usage

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

Arguments

object	A tobs_fit object.
...	Forwarded to summary.tulpa_fit.

Value

A `summary.tulpa_fit` object as `tulpa::summary.tulpa_fit()` returns it. Its coefficient table, `coef(summary(fit))`, has one row per coefficient named `<arm>_<term>` as `coef.tobs_fit()` names it, with columns `estimate`, `std.error` and the two interval bounds, plus a `skew` column when `$skew` is present and `rhat`, `ess_bulk`, `ess_tail` on a sampled fit. Every family, and every method within a family, returns this layout.

t_occu	<i>Multi-season occupancy with a temporal (AR1) year effect (tPGOcc)</i>
--------	--

Description

The `spOccupancy tPGOcc` model: per-(site, season) Bernoulli occupancy with a shared AR1 year random effect on the occupancy logit and NO colonization / extinction dynamics (that is `dyn_occu()`). Use it for an occupancy trend over years with a temporal random effect. Fit with `method = "pg_gibbs"` (the Polya-Gamma Gibbs sampler `spOccupancy` uses); `y` is a 3D array `[n_sites x n_seasons x max_visits]` (or a list of per-season `[n_sites x max_visits]` matrices), the occupancy formula and detection formula are site-level.

Usage

```
t_occu()
```

Value

A `tobs_family` object.

Engines

- `method = "pg_gibbs"` – the Polya-Gamma Gibbs sampler: conditional on the year effects the seasons factorise, so the joint (coefficients, AR1 year effect) update is one exact Gaussian Markov random field draw; the only route.

Examples

```
f <- t_occu()
f
```

tidy.tobs_fit	<i>Coefficient table for a tobs_fit</i>
---------------	---

Description

Coefficient table for a tobs_fit

Usage

```
## S3 method for class 'tobs_fit'
tidy(x, conf.level = 0.95, ...)
```

Arguments

x	A fitted tobs_fit.
conf.level	Interval level (default 0.95).
...	Ignored.

Value

A data frame with one row per coefficient: arm (the linear predictor it enters, NA for a coordinate that enters none), term (its design column, without the arm prefix), estimate, std.error, conf.low and conf.high. Every family returns this layout.

tobs	<i>Fit a hierarchical latent-state observation model</i>
------	--

Description

Unified entry point for occupancy, abundance, distance, removal, and related models that share the latent-state-plus-imperfect-detection generative template. The specific model is chosen via the family argument; the engine (Laplace / nested Laplace / NUTS) via engine.

Usage

```
tobs(
  formula,
  data,
  family,
  occurrence = NULL,
  detection = NULL,
  positive = NULL,
  y = NULL,
  visits = NULL,
  method = c("auto", "laplace", "laplace_sla", "laplace_gibbs", "laplace_mi", "pg_gibbs",
```

```

    "nested_laplace", "nested_laplace_sla", "nuts"),
  priors = NULL,
  control = list(),
  by = NULL,
  ...
)

```

Arguments

formula

state-process formula, e.g. `~ elev + forest`. For occupancy this is the occupancy probability formula; for N-mixture the abundance formula; for the cover hurdle the latent-presence formula (also given as occurrence, which reads symmetrically with detection and positive).

Single-vector-response families (the cover hurdle, `cover()`) also accept the response on the left-hand side, `response ~ predictors`, in which case `y =` is omitted (e.g. `cover.flat ~ time + habitat`). The LHS is evaluated against data (then the calling environment), so it may be a bare column or an expression. Matrix / array / list response families (`occu()`, `abun()`, the `ms_*` families, ...) keep the one-sided form and supply the response via `y =`; a two-sided formula for those errors.

Structured effects are written as terms inside the formula, the way `lme4`, `mgcv`, and `INLA` do: spatial fields `icar(graph = adj)`, `bym2(graph = adj)`, `gp(lon, lat, prior.range = c(r0, alpha))`, `spde(lon, lat)`; random effects `re(group)`; temporal fields `temporal(time)`; spatially varying coefficients `spatial(~ 1 + w || cell, graph = adj)` over a graph or `spatial(lon, lat, model = "svc", coefficients = ..., prior.range = c(r0, alpha))` over coordinates; community latent factors `latent(k)`. A term enters whichever linear predictor it is written in (occupancy formula or detection). To share one realization across both predictors, tag the term with `id = "u"` and write `share("u")` in the other formula.

The spatial fields also have a single-verb umbrella `spatial(..., model = ...)` that selects the field type by name, mirroring `temporal(time, type = ...)` and `INLA's f(i, model = ...)`: `spatial(graph = adj, model = "bym2")` is `bym2(graph = adj)` and `spatial(lon, lat, model = "spde")` is `spde(lon, lat)`. `model` is one of `"icar"`, `"bym2"`, `"car"`, `"car_proper"`, `"gp"`, `"multiscale_gp"`, `"spde"`, `"svc"`; per-model arguments pass through unchanged. The umbrella also reads a varying-coefficient bar (`spatial(~ 1 + w || cell, graph = adj)`), so both flavours of spatially-varying coefficient – areal over a graph, continuous over coordinates (`model = "svc"`) – are written through the same verb.

The continuous fields (`gp()`, `svc()`, `spde()`) require `prior.range = c(r0, alpha)`, a penalized-complexity prior on the spatial range encoding $P(\text{range} < r0) = \alpha$ (Fuglstad et al. 2019). The range is in the units of the coordinates – the kernel is $\exp(-d / \text{range})$ – so choose `r0` as a distance below which the field's correlation would be surprisingly short: on unit-square coordinates `prior.range = c(0.1, 0.05)` reads as "a 5% chance the range is under 0.1". There is no default, deliberately. The range is weakly identified by the likelihood alone, so a default would be an invented prior doing real work on the posterior rather than a convenience.

Random effects also accept lme4 bar syntax as shorthand for `re()`: `(1 | g)` is `re(g)`, `(x | g)` is a correlated random intercept and slope `re(g, type = "slope", covariate = x)`, and `(x || g)` drops the correlation (`correlated = FALSE`). Use `re()` directly for AR1/RW structures or other options.

data	data frame of site-level covariates with <code>nrow(data) == nrow(y)</code> , or a <code>tobs_data</code> frame (from <code>tobs_data()</code> / <code>tobs_format()</code>) bundling the response, site covariates, and visit covariates. When a frame is passed, <code>y</code> and <code>visits</code> are taken from it and must not also be supplied.
family	a <code>tobs_family</code> object (see <code>obs_family()</code> and the concrete constructors <code>occu()</code> , <code>abun()</code> , <code>cover()</code> , ...).
occurrence	state-process (occupancy / latent-presence) formula for the cover hurdle (<code>occu_cover()</code> / <code>cover()</code>), reading symmetrically with detection and positive. The front-door name for formula; supply one of the two.
detection	detection-process formula, e.g. <code>~ observer + effort</code> . Family-dependent: required for <code>occu()</code> and <code>abun()</code> , ignored for <code>jsdm()</code> and (currently) <code>cover()</code> .
positive	positive-arm (cover) formula for <code>occu_cover()</code> , e.g. <code>~ time.sc + habitat</code> . The occurrence spatial field is carried onto this arm with a <code>share()</code> selector, the INLA-style cross-arm edge written in the formula. The selector is a constructor, so no field name is needed in the common case: • <code>share(spatial(), alpha = grid(g))</code> copies the occurrence arm's spatial effect and marginalizes the coupling amplitude over the nodes <code>g</code>; <code>alpha = grid(n = 9)</code> marginalizes it over the engine's own axis read at <code>n</code> slab nodes instead, which sharpens the axis without restating the prior structure it carries; <code>alpha = <scalar></code> fixes it. • <code>share(spatial(), terms = list(intercept = grid(g0), time.sc = grid(g1)))</code> gives a per-component amplitude, keyed by the field's own block names (the intercept block and a <code> </code>-declared trend column, or its alias <code>trend</code>); <code>terms =</code> must address every block. • <code>share(spatial(), prior = list("pc.prec", c(4, 0.01)))</code> regularizes the copy coefficient itself (<code>prior_alpha</code> in <code>tulpa::tulpa_nested_laplace_joint()</code>). One reaches the engine per fit. • <code>share(spatial(cell_idx), ...)</code> disambiguates by grouping variable when the occurrence arm carries several spatial effects. • decoupling is structural: omit <code>share()</code> so the field rides occupancy only (a block with no <code>share()</code> is pinned at zero coupling), or write a <code>spatial()</code> term for the cover arm's own field.

Defaults to detection when unset (a per-visit cover design matching the detection design).

y	response. Shape depends on family: • <code>occu()</code> – $N \times J$ detection-history matrix. • <code>abun()</code> – $N \times J$ integer count matrix. • <code>ms_*</code> – $S \times N \times J$ array. • <code>cover()</code> – length-N vector of cover proportions in $[0, 1]$.
---	--

For a single-vector-response family (`cover()`) the response may instead be written on the formula left-hand side (`response ~ predictors`), in which case `y =` is omitted. Supplying the response both on the LHS and via `y = errors`. Matrix / array / list response families take the response via `y = only`.

visits

optional visit-level detection covariates. Accepts either:

- a named list of `[n_sites, max_visits]` matrices (the shape returned by `tobs_data()` in `det.covs`) – flattened internally to a long data frame in site-major order;
- a data frame with `nrow(y) * ncol(y)` rows in site-major order.

When `visits` is provided without a "formula" attribute, the detection argument is interpreted as the visit-level detection formula and the site-level detection design matrix is an intercept only. To split visit-level and site-level detection covariates (e.g. visit-level effort plus site-level observer category), pass a long data frame with `attr(visits, "formula") = ~ effort` and use `detection = ~ observer` for the site-level terms.

For `occu()`, a random-effect bar in the visit-level formula is a random effect over visit rows: `detection = ~ effort + (1 | observer)` with `observer` a column of `visits` gives each observer its own shift of the detection logit on the visits it made. It is fitted with `method = "laplace"` or `"nuts"`, and other families refuse it.

method

inference route, naming a fully-specified path rather than a pair of orthogonal knobs. Each family accepts a subset of the routes below (listed per route, from the package's support table); an unsupported method errors with the list of methods that family supports, and the structured terms a route carries for a given family are on that family's help page.

- "auto" – the family's default route (see `default_engine`).
- "laplace" – EM + Laplace with Gaussian marginals (fast default). Every family except `t_occu()`.
- "laplace_sla" – Laplace with skew-corrected (simplified-Laplace) marginals. `occu()`, `dyn_occu()`, `int_occu()`, `cover()`.
- "laplace_gibbs" / "laplace_mi" – Laplace with a post-EM Gibbs / multiple-imputation correction. The fixed-effect prior threads into the correction re-fits, so these use the same weakly-informative default prior as "laplace"; pass `priors = FALSE` for the unpenalised correction. `occu()`, `dyn_occu()`, `int_occu()`.
- "pg_gibbs" – a Polya-Gamma Gibbs sampler over the exact occupancy posterior (the `spOccupancy` `PGOcc` engines). A real MCMC chain (with `Rhat` / `ESS` diagnostics), distinct from "laplace_gibbs" (a stochastic-EM variance correction). Sampler controls (`n.iter`, `n.warmup`, `n.chains`, `n.thin`, `seed`, `sigma.beta`). `occu()` (site-level detection; an `icar()` field on occupancy), `ms_occu()`, `ms_dyn_occu()`, `ms_int_occu()`, `jsdm()`, `ms_count()`, and `t_occu()`, whose only route it is.
- "nested_laplace" – multi-block nested Laplace: the structured terms (areal fields, shared cover-hurdle fields, latent factors) are integrated over a grid of their hyperparameters. `occu()`, `dyn_occu()`, `int_occu()`, `dyn_int_occu()`, `ms_occu()`, `ms_dyn_occu()`, `jsdm()`, `count()`, `ms_count()`, `abun()`, `ms_abun()`,

	<code>dyn_abun(), removal(), distance(), ms_distance(), fp_occu(), cover(), occu_cover(), occu_multiscale_cover().</code> • "nested_laplace_sla" – nested Laplace with skew-corrected marginals. <code>occu(), cover().</code> • "nuts" – HMC / NUTS sampler (reports Rhat / ESS). <code>occu(), dyn_occu(), int_occu(), ms_occu(), ms_dyn_occu(), ms_int_occu(), jsdm(), ms_count(), abun(), ms_abun(), dyn_abun(), removal(), distance(), fp_occu(), cover(), occu_cover(), ms_occu_cover(), occu_multiscale_cover().</code>
priors	optional prior specification. For occupancy families fit with a Laplace method (method = "laplace", "laplace_sla", "nested_laplace"), pass a list or <code>occu_priors()</code> object to set weakly-informative quadratic priors on the fixed-effect coefficients (defaults pull the detection intercept toward $p = 0.5$ and break the psi-p identifiability ridge at small J). Pass <code>priors = FALSE</code> to disable the default prior and recover the unpenalised MAP. The "laplace_gibbs" / "laplace_mi" routes apply the same default prior and thread it through the correction refits. For NUTS, this is forwarded to the underlying tulpa engine.
control	list of low-level engine controls. Names follow the dotted-separator convention. Every default below is resolved from one table (<code>.TOBS_ENGINE_DEFAULTS</code>), so it is a property of the fitting engine rather than of the family. Where a default differs between the single-species and community entries, both are given. Sampler controls (method = "nuts"): • <code>n.iter</code> – post-warmup sampling iterations kept per chain (default 1000); the total run per chain is <code>n.iter + n.warmup</code>. • <code>n.warmup</code> – warmup / adaptation iterations per chain, discarded (default 1000). • <code>n.thin</code> – keep every <code>n.thin</code>-th post-warmup draw (default 1). The kept draws and the per-iteration diagnostics (divergent, accept_prob, treedepth) are thinned by the same stride. • <code>n.chains</code> – number of chains, run with offset seeds and pooled (default 1). Split-Rhat / bulk / tail ESS are reported on <code>\$convergence</code>. • <code>n.threads</code> – chains to run in parallel (default 1, sequential). Values > 1 use a PSOCK cluster and require tulpaObs to be installed. This is chain parallelism; it does not change the thread count inside a single gradient evaluation. • <code>n.threads.grad</code> – OpenMP threads inside ONE gradient evaluation of a community NUTS target (<code>ms_occu(), ms_count() / jsdm(), ms_abun(), ms_dyn_occu()</code>), whose per-species loop is parallel. Default 0 leaves the count to OpenMP. The per-species reduction is serial and order-fixed, so the gradient is the same at any count. Rejected on the other families. • <code>adapt.delta</code> – target acceptance probability (default 0.8 on the single-species families, 0.9 on the community samplers). • <code>max.treedepth</code> – NUTS maximum tree depth (default 10). • <code>seed</code> – base RNG seed; chain <code>c</code> uses <code>seed + c - 1</code> (default 42 on the single-species families, 1 on the community samplers). The resolved per-chain seeds are stored on <code>\$seeds</code>.

- `sigma.beta` – prior SD on the coefficients (default 10 on the single-species families and on the log-link community samplers `ms_count()` / `jsdm()` / `ms_abun()`, whose unit change is multiplicative; 5 on the logit-link community samplers `ms_occu()`, `ms_dyn_occu()`, `ms_int_occu()`, `ms_occu_cover()`).
- `sigma.logr` – prior SD on the community-mean log-dispersion `mu_log_r` (default 1.5), on the negative-binomial samplers that carry one (`ms_abun()`, `ms_count()`, `jsdm()`), and on the log-dispersion and grouped random-effect log-SD of the `abun()` / `removal()` / `distance()` samplers. At the default this is an informative prior on the dispersion scale, so raise it to compare the sampler against a maximum-likelihood target. Ignored under a Poisson mixture; rejected on the other families.

Sampler controls (`method = "pg_gibbs"`). A conjugate sweep is far cheaper than a NUTS trajectory, so the chain is longer and two chains run by default. `n.iter` **counts differently here**: it is the TOTAL number of sweeps and warmup comes out of it, so the chain keeps `n.iter - n.warmup` draws, where under "nuts" it is the kept count and the run is `n.iter + n.warmup` long.

- `n.iter` – total sweeps per chain (default 3000).
- `n.warmup` – sweeps discarded from the front (default 1500), leaving 1500 kept draws.
- `n.thin` – keep every `n.thin`-th post-warmup sweep (default 1).
- `n.chains` – number of chains (default 2, so split-Rhat is available without a second call).
- `seed` – base RNG seed (default 1).
- `sigma.beta` – coefficient prior SD (default 2.5; tighter than the NUTS one because a conjugate update has no step-size adaptation to absorb a wide prior). There is deliberately no `adapt.delta / max.treedepth`: those are HMC knobs.
- `n.seeds` – number of seed-offset refits to fit and LOO-stack into a `tobs_stack` ensemble (default 1, a single fit). Member `k` uses base seed `seed + k - 1`. Only meaningful for the stochastic routes ("nuts", "pg_gibbs", "laplace_gibbs", "laplace_mi"); the deterministic Laplace methods reject it. Seed-variants are statistically identical, so their stacking weights come out roughly uniform (this is a Monte-Carlo robustness device) – pass distinct fits to `tobs_stack()` for a genuine model average.

Laplace controls (`method = "laplace" / "laplace_sla" / "nested_laplace"`): `max.iter`, `tol`, `damping`, `sigma.beta`.

- `logr.sigma.prior` – Penalized-Complexity prior $c(U, \alpha)$ ($P(\text{sigma_log_r} > U) = \alpha$) on the per-species log-dispersion SD under `ms_abun(mixture = "negbin" / "zinb")`. Default NULL, pure maximum likelihood. `sigma_log_r` is one scalar variance over species and at few species can settle near its lower boundary, which shifts `mu_log_r` and narrows its interval together (see `?ms_abun`); the prior adds curvature there. `omega.sigma.prior` is the same knob for the structural-zero variance and does default to `c(1, 0.05)`; when both are set they must be equal, since one Penalized-Complexity prior is applied across every regularized block.
- `re.aghq` – for a formula random effect under `method = "laplace"`, run the adaptive Gauss-Hermite debias of the variance components after the EM

converges (default TRUE). Removes the Laplace small-cluster attenuation of sigma / the RE correlation for binary occupancy; set FALSE for the raw EM (Laplace, nAGQ = 1) fit.

- `n.quad` – quadrature points. One name across several routes, each integrating a different marginal over a different latent dimension, so the default is per route rather than one number. `n.quad = 1` is always the plain Laplace (nAGQ = 1) marginal; higher values refine it toward the exact one. Formula random effect under `method = "laplace"`: 9. Adaptive Gauss-Hermite over the exact per-group marginal, debiasing the Laplace small-cluster attenuation of a binary occupancy variance component. Binary data carries little information per group, so the refine wants many nodes. Community N-mixture (`ms_abun()`): 1, i.e. the EM default. Each species' count marginal is already informative, so the AGHQ refine barely moves the community covariances; it is opt-in via `optimizer = "joint_fd"` for the sparse / rare-species regime. `n.quad.scalar` (default 3, and floored there) is the trailing per-species log-dispersion / structural-zero-logit coordinate, integrated separately. Community joint occupancy-cover (`ms_occu_cover()`): 5. Tensor AGHQ over the joint per-species RE vector, so the node count is raised to a power of the RE dimension and stays small. Latent cover-per-unit (`cover.aggregate = "latent"`): 15 for a beta cover arm, 1 for lognormal, whose per-unit marginal is closed form and needs no quadrature at all. Community latent() factors: 5, the Gauss-Hermite nodes the joint site marginal integrates the factor scores on. Both the loading magnitude and the score-matched offset are insensitive to it (argmax stable to < 0.4% against 21 nodes).
- `max.outer` – for a community family whose latent() factors or shared areal field are fit by block coordinate ascent, the cap on the outer alternation between the community EM and the field / factor update. A field block reaches `tol` and stops early, so its default 25 is only a cap; a factor block does not, and each family sets its own budget from a measured bias curve (150 on `ms_count()` / `jsdm()` / `ms_occu()`, 25 elsewhere).
- `factor.starts` – candidate starting directions the first factor pass selects over, on the joint marginal. Each costs a full loading EM against that family's oracle, so the default is per family: 1 on `ms_abun()` (measured to buy nothing there against a 2.0-2.3x cost), 8 elsewhere. Accepted only by the families fit this way; the resolved value, and `max.outer / n.quad` alongside it, is reported as `fit$latent_control`.
- `re.lkj` – LKJ shape (eta) regularizing a *correlated* random slope's correlation in the `re.aghq` refine (default 1.5). Pulls a weakly-identified RE correlation off the +-1 boundary toward 0 without touching the marginal SDs; `re.lkj = 1` disables it (uniform). No effect on intercept / uncorrelated terms.
- `sd.load` – prior SD on a spatial-factor loading in the community occupancy-cover fit (default 1). The auto-rank ladder selects K by marginal evidence under this prior, so the selection fit and the final fit necessarily read the same value.
- `inner.solver` – for a spatial community N-mixture (`ms_abun()` with an `icar()` / `bym2()` / `car_proper()` field on the abundance arm), the inner

solver integrating the shared field given the community: "em" (default) the closed-form Laplace-EM M-step, or "newton" the exact- Newton shared-field solve alternated with a tulpa AGHQ community debias. Both integrate the field hyperparameter on the outer grid and return the same fit object; "newton" is Poisson- and areal-only, and markedly slower (an FD-gradient profile loop per grid node) – an accuracy / validation alternative, not the production default.

- integration – how the in-package spatial / community nested-Laplace fitters integrate the outer field hyperparameters (tau, rho, sigma, range): "grid" (default) a fixed tensor grid, or "ccd" a mode-centred central-composite design placed at the marginal-likelihood mode and scaled by the outer posterior covariance, with the outer PSIS Pareto-k reported on `fit$spatial_pareto_k`. CCD declines to the grid when the outer curvature is ill-conditioned (a weakly-identified axis) or for a single positive hyperparameter; `fit$spatial_integration` records which ran. Each outer node is a full inner solve, so "ccd" adds a mode-find without a node saving on these coarse grids – it is opt-in, most useful when a multi-axis hyperparameter posterior is well identified.
- `adaptive.grid.cutoff`, `adaptive.grid.stride`, `adaptive.grid.max.frac`, `adaptive.grid.min.cells` – tuning for integration = "grid_adaptive" on the joint-coupled spatial families (`occu_cover()`, `cover()`, `occu()` spatial, `occu_multiscale_cover()`), which the engine hosts alongside "grid" and "ccd". That integrator evaluates a strict subset of the same tensor lattice – it floods outward from the posterior mode and keeps the cells within `adaptive.grid.cutoff` log-density of the peak – and declines back to the dense tensor when the kept region would rival it, so it trades inner solves for nothing but resolution of the hyperparameter tails. `adaptive.grid.cutoff` is the keep / expand radius (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 it declines, and `adaptive.grid.min.cells` the smallest dense tensor worth the machinery. Unset leaves each at the engine default. These are a DIFFERENT mechanism from `adaptive.grid / adaptive.grid.edge.thresh / adaptive.grid.max.passes`, which refine an already-integrated grid; the two compose and neither implies the other. The saving is largest where a fit carries several outer axes – two spatial arms (a shared field plus an arm-specific one) put three on the grid, which integration = "auto" resolves to the dense tensor.
- `max.grid.cells` – for the same joint-coupled spatial families, an opt-in ceiling on the dense outer tensor's cell count (the engine's `max_grid_cells`, unset by default). A grid past it is refused before any inner solve, with the axes that produced the count. The count is the number of inner solves: two copied field blocks at the default field-SD and copy axes cross to 2500 latent cells, and an active positive-arm dispersion grid multiplies that by its node count. Unset, a tensor of any size is solved, including the one a declined integration = "grid_adaptive" falls back to, and one past 2048 cells is announced with a warning first.
- `diagnose.k` – for the joint-coupled spatial families (`occu_cover()`, `occu()`)

spatial, `occu_multiscale_cover()`), whether to score the outer hyperparameter Gaussian summary with an importance-sampling Pareto-k. Defaults FALSE (it re-solves the inner Laplace on the full field `k.samples` times). When TRUE, the fit carries `pareto_k` (< 0.7 = reliable summary), `pareto_k_is_ess` (the importance-sampling ESS on the PSIS-smoothed weights; `k.samples` is the relative IS efficiency), and `pareto_k_proposal_source` at the top level (and in `glance()`). `pareto_k_proposal_source` is "mode_hessian" when the importance proposal came from the Laplace curvature at the hyperparameter mode – curvature-backed, so the $\hat{k}$ stays trustworthy even when a sharp posterior collapses the integration grid – or "grid_moment" when it came from the grid-weighted node covariance, the regime to watch as the grid concentrates. Stochastic-correction controls ("laplace_gibbs" / "laplace_mi"): `n.gibbs / n.imputations` (Rubin-pooled draw count) and seed (stored on `$seed`).

Progress controls (every method): `progress` toggles the console iteration / grid bar (default TRUE in an interactive session, FALSE otherwise), `progress.every` and `progress.throttle` set its emit cadence, and `progress.file` names a heartbeat file rewritten with "<done> <total> <elapsed_s> <eta_s>". The file is the channel that survives a detached run, where a console flush does not, and it is written whenever it is set regardless of progress. The environment variable `TULPAOBS_PROGRESS` sets the console default for the whole session: 0 (also false / no / off) turns it off, any other value turns it on – for a batch or CI run whose caller cannot pass control to each individual fit; an explicit `control$progress` still wins.

Control names are validated against the chosen method: passing a sampler control (e.g. `n.chains`) to a Laplace method, a Laplace control (e.g. `max.iter`) to "nuts", seed to a deterministic route, or an unrecognized name raises an error rather than being silently ignored.

by optional name of a species column for a per-species batched fit (`occu_cover()` / `cover()` only). When supplied, `data` is a long, plot-level frame: one row per site-visit (`occu_cover()`) or per site (`cover()`), with `by` giving the species. `tobs()` splits `data` by that column, builds each species' response onto one shared site x visit grid (via `tobs_data()`), fits the B species independently, and returns a `tobs_batch`. Per-species fits are statistically independent (no pooling) – identical to fitting each species with a separate single-species `tobs()` call; for a community model that borrows strength across species use `ms_occu_cover()`. The long->response pivot needs the column names: `pass site =` and `response =` (the detection 0/1 column for `occu_cover()`, the cover column for `cover()`), plus, for `occu_cover()`, `visit =` (the replicate column) and `y.pos =` (the cover column) and any visit-level `det.covs =`. Site-level covariates (those the formula / detection reference at the cell level) are read as the first row per site, the way `tobs_data()(occ.covs = )` does.

The same long-frame contract drives a SINGLE `occu_cover()` fit when `by` is omitted: `pass site =`, `visit =`, `response =` (the 0/1 detection column) and `y.pos =` (the cover column), plus any visit-level `det.covs =`, with a long, plot-level `data`, and `tobs()` builds the paired occurrence / cover arms and the site-level design for you – the `by=` batch reduced to one species, so you no longer hand-build the two responses and align them. The arms default to the

compact (ragged) layout on the nested-Laplace route (no per-site visit cap); set `control$compact = FALSE` for the dense grid. See `occu_cover_inputs()` to build and inspect the arms without fitting.

The long-frame `response` = here names a COLUMN of data (the pivot key holding the detection / cover values). It is a separate argument from the positive-part distribution the family constructor takes (`cover(response = "beta")`, `occu_cover(response = "lognormal")`), which selects the cover-arm likelihood rather than a data column. Both may appear in one call – `tobs(long, family = cover(response = "beta"), site = , response = "cover.flat", ...)` – and do not interact.

... family-specific named arguments forwarded to the underlying engine builder.

Value

An object of class `c("tobs_fit", "<family>_fit", "tulpa_fit")`. When `control$n.seeds > 1`, a `tobs_stack` ensemble of the seed-offset refits is returned instead (see `tobs_stack()`). When `by` is supplied, a `tobs_batch` of per-species fits is returned (see `tobs_get()`).

Examples

```
# Single-season occupancy
sim <- simulate_occu(N = 100, J = 3, n.occ.covs = 1, n.det.covs = 1,
                    seed = 1)

fit <- tobs(
  formula = ~ occ_cov1,
  data     = sim$data,
  family   = occu(),
  detection = ~ det_cov1,
  y        = sim$y,
  method   = "laplace",
  control  = list(verbose = FALSE, progress = FALSE)
)
summary(fit)
```

<code>tobs_associations</code>	<i>Residual species-association matrices from a spatial-factor community fit</i>
--------------------------------	--

Description

For a reduced-rank spatial-factor community occupancy-cover fit (a `ms_occu_cover()` model carrying an `icar()`, `car_proper()`, or `bym2()` field on the occupancy arm), return the residual species associations the shared latent fields imply – the spatial-JSDM / HMSC output. The K unit-scale fields induce a residual occupancy covariance $L L'$ across species; the reported matrix is its correlation form, identified (invariant to the factor rotation, sign, and column scale). When the fit also carries a cover-arm factor, the cover association `corr(L_pos L_pos')` and the joint cross-arm association (standardized `L_occ L_pos'`, species occupancy vs species cover) are available too.

Usage

```
tobs_associations(
  object,
  type = c("occupancy", "cover", "cross"),
  summary = NULL
)
```

Arguments

object	A fitted <code>tobs_fit</code> from a spatial-factor <code>ms_occu_cover()</code> .
type	Which association to return: "occupancy" (default), "cover", or "cross" (occupancy vs cover). "cover" and "cross" require a cover-arm factor (an <code>icar()</code> / <code>car_proper()</code> / <code>bym2()</code> term on the cover formula).
summary	One of "median", "estimate" (rotation-invariant point estimate at the posterior mode), "lower", "upper"; or NULL (default) to return the full list of all four $S \times S$ matrices plus the interval prob and draw count.

Details

Each matrix is summarised from draws of the loading posterior (not the plug-in mode), so a central interval accompanies the point estimate.

Value

An $S \times S$ correlation matrix (when summary is given) or a list of the point estimate, posterior median, and interval bounds.

Examples

```
adj <- matrix(0, 16, 16)
adj[cbind(1:15, 2:16)] <- 1
adj <- adj + t(adj)
sim <- simulate_ms_occu_cover_spatial(adj, n.species = 6, K = 2, J = 4,
                                       seed = 1)
fit <- tobs(~ occ_cov1 + icar(graph = adj), data = sim$data,
            family = ms_occu_cover("lognormal"), detection = ~ det_cov1,
            positive = ~ pos_cov1, y = sim$y, y.pos = sim$y_pos,
            species = sim$species, method = "laplace",
            control = list(n.factors = 2, max.iter = 30, verbose = FALSE))
round(tobs_associations(fit, summary = "median"), 2)
```

tobs_check_id	<i>Check model identifiability</i>
---------------	------------------------------------

Description

Diagnostics for potential identifiability issues. Checks for confounded covariates, low detection or detection-free sites, and sparse replication.

Usage

```
tobs_check_id(model, fit = NULL)
```

Arguments

model	A tobs_model object (before fitting).
fit	Optional tobs_fit object (for post-fit diagnostics).

Details

Applies to the families whose response is a site-by-visit grid: `occu()`, `abun()`, `removal()` and `fp_occu()`. On any other family the pre-fit checks are reported as not applicable rather than silently skipped – an empty issues on a family the checks never ran on is not a clean bill of health. The post-fit sampler checks apply to any NUTS fit.

Value

A list with `identifiable`, the issues messages, and `prefit_checked` (FALSE where the family carries no site-by-visit grid).

Examples

```
sim <- simulate_occu(N = 100, J = 3, n.occ.covs = 1, n.det.covs = 1,
                    seed = 1)
fit <- tobs(~ occ_cov1, data = sim$data, family = occu(),
           detection = ~ det_cov1, y = sim$y, method = "laplace",
           control = list(verbose = FALSE, progress = FALSE))
chk <- tobs_check_id(fit$model, fit)
chk$identifiable
```

tobs_criteria

*Model criteria for occupancy / cover models***Description**

`waic()`, `loo()`, `dic()`, and `cpo()` on a `tobs_fit` build the family pointwise log-likelihood matrix once and hand it to the engine's single criteria layer `tulpa::tulpa_criteria()`, which derives WAIC / DIC / CPO / LPML / PSIS-LOO. DIC additionally evaluates the deviance at the posterior mean of the parameters, supplied by the family-specific `.tobs_loglik_at_mean()`.

Usage

```
## S3 method for class 'tobs_fit'
waic(x, n.draws = 1000L, loo.unit = c("obs", "cell"), n.threads = NULL, ...)

## S3 method for class 'tobs_fit'
loo(x, n.draws = 1000L, loo.unit = c("obs", "cell"), n.threads = NULL, ...)

## S3 method for class 'tobs_fit'
dic(object, n.draws = 1000L, n.threads = NULL, ...)

## S3 method for class 'tobs_fit'
cpo(
  object,
  n.draws = 1000L,
  loo.unit = c("obs", "cell"),
  n.threads = NULL,
  ...
)

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

Arguments

<code>x, object</code>	A <code>tobs_fit</code> object.
<code>n.draws</code>	Posterior draws used to build the pointwise log-likelihood (the <code>cover</code> / <code>occu_cover</code> paths sample this many; the draw-matrix families use the first <code>n.draws</code> stored draws). Default 1000.
<code>loo.unit</code>	The cross-validation unit for <code>waic()</code> / <code>loo()</code> / <code>cpo()</code> . "obs" (default) is the family's pointwise unit – one column of the log-likelihood per plot (<code>cover</code>) or site (<code>occu_cover</code>) – and is byte-identical to the call without the argument. "cell" switches to leave-one-group-out cross-validation (LOGO-CV): the fit's own per-observation cell map folds the log-likelihood columns of a spatial cell into one, so each cell is a fold instead of each plot / site. <code>waic()</code> and <code>cpo()</code> hand the map to <code>tulpa::tulpa_criteria()</code> as group; <code>loo()</code> applies the same fold itself

and runs PSIS on the resulting $[n_draws \times n_cells]$ matrix, whose column is the cell's joint conditional log-likelihood per draw, so the importance ratio inverts the whole cell's likelihood. A whole cell is a larger perturbation of the posterior than a single row, so the Pareto k values are correspondingly higher and are the diagnostic to read before trusting the cell-level number. Implemented for `cover()` (the areal field node, when sites are grouped via `group.var`) and `occu_cover()` (the `site_cell` map); a non-spatial fit has no cells, so "cell" errors there. Equivalent to passing `group =` the cell map directly, without hand-building it. `dic()` has no cross-validation unit – it is a plug-in deviance over all observations – and rejects `loo.unit`.

<code>n.threads</code>	Threads for the parallel <code>occu_cover()</code> pointwise log-likelihood (the compact / ragged path). The draw loop is embarrassingly parallel, so this is the WAIC / LOO analogue of the fit's <code>n.threads.outer</code> . NULL (default) uses all but four logical cores, at most two under R CMD check; other families ignore it.
<code>...</code>	For <code>dic()</code> and <code>cpo()</code> , forwarded to <code>tulpa::tulpa_criteria()</code> (e.g. <code>chunk_size</code> , or an explicit <code>group =</code> for a custom leave-one-group-out unit). <code>waic()</code> and <code>loo()</code> accept only <code>group =</code> , which folds the pointwise matrix before the loo call, and reject any other argument.
<code>ndraws</code>	For <code>pointwise_loglik()</code> , the posterior draws used to build the matrix, as <code>n.draws</code> above. NULL (default) uses 1000.

Details

`waic()` and `loo()` are the **loo** package's generics and build their objects through `loo::waic()` and `loo::loo()` on the same pointwise matrix (`loo()` via PSIS with relative effective sample sizes), so `loo::loo_compare()` and the rest of that ecosystem read them directly. `dic()` and `cpo()` are **tulpa**'s, and return a `tulpa_criteria` object.

Value

`dic()` and `cpo()` return a `tulpa_criteria` object. `waic()` returns a `waic` object and `loo()` a `psis_loo` object from the **loo** package; read the estimates from `$estimates` (e.g. `waic(fit)$estimates["waic", "Estimate"]`). Both carry the cross-validation unit they scored in their `"loo_unit"` attribute.

See Also

`tulpa::tulpa_criteria()`, `loo::loo_compare()`

tobs_data

Convert long-format data to a site x visit observation object

Description

Convert long-format data to a site x visit observation object

Usage

```
tobs_data(
  df,
  y,
  site,
  visit,
  type = c("occurrence", "abundance", "cover", "positive"),
  occ.covs = NULL,
  det.covs = NULL,
  coords = NULL,
  sites = NULL,
  visits = NULL,
  cover.floor = 0,
  compact = FALSE
)
```

Arguments

<code>df</code>	Data.frame in long format (one row per site-visit).
<code>y</code>	Character, name of the response column. Its meaning is set by <code>type</code> : a 0/1 detection ("occurrence"), an integer count ("abundance"), or a continuous cover proportion in $[0, 1]$ ("cover").
<code>site</code>	Character, name of site identifier column.
<code>visit</code>	Character, name of visit/replicate column.
<code>type</code>	Response kind. "occurrence" (default) and "abundance" build an integer site x visit matrix; "cover" and "positive" build a double matrix and preserve continuous values (no coercion to integer). "cover" is a proportion in $[0, 1]$ (the beta cover arm); "positive" is a positive real $(0, \text{Inf})$ (the lognormal / gamma cover arm), validated as non-negative with no upper bound. Both apply the <code>cover.floor</code> absence policy below.
<code>occ.covs</code>	Character vector of site-level covariate names.
<code>det.covs</code>	Character vector of visit-level covariate names. A named column that is a factor or character is preserved as categorical: the downstream detection / positive design expands it to $k - 1$ dummies for a k -level factor, with the factor's first level (for a character column, the first level after sorting the unique values) as the reference. Numeric columns are kept as a continuous covariate.
<code>coords</code>	Character vector of length 2 for coordinate columns.
<code>sites</code>	Optional site level set. When supplied, the site x visit grid uses these site identifiers in this order (rather than <code>df</code> 's first-appearance order), so a subset of <code>df</code> pivots onto a fixed, externally-defined site grid; site values in <code>df</code> outside the set error.
<code>visits</code>	Optional visit level set, analogous to <code>sites</code> for the visit axis (default the sorted unique visits in <code>df</code>).
<code>cover.floor</code>	For <code>type = "cover" / "positive"</code> , the threshold at or below which a value is stored as NA rather than as a positive observation (default 0). The positive arm

of a hurdle is positive-only, so a 0 is an absence handled by the occurrence arm, not a positive observation; sending it to NA keeps a sampled-absent or unsampled cell from entering the positive arm as a fabricated zero (which, padded across a grid, flattens the spatial field). Set `cover.floor = -Inf` to keep every value verbatim.

compact Logical (default FALSE). When TRUE, return a compact (ragged) `tobs_data`: the response is stored as one row per valid site-visit (a `tobs_ragged` carrier) rather than as a padded `[n_sites x max_visits]` matrix, and each detection covariate as a length-V vector in the same canonical order(`site`, `visit`). The compact layout has no per-site visit cap (its memory is the number of observations, not the padded grid) and is consumed by the joint nested-Laplace `occu_cover()` engine, which works one valid visit at a time. Two compact calls on the same `df / site / visit` align row-for-row, so an occurrence response and a cover response can be paired directly.

Value

An `tobs_data` object. With `compact = TRUE` its `$y` is a `tobs_ragged` carrier and its `$det.covs` are length-V vectors.

Examples

```
sim <- simulate_occu(N = 30, J = 3, seed = 1)
long <- data.frame(site = rep(seq_len(30), times = 3),
  visit = rep(1:3, each = 30),
  det = as.vector(sim$y),
  occ_cov1 = sim$data$occ_cov1,
  effort = runif(90))
dat <- tobs_data(long, y = "det", site = "site", visit = "visit",
  occ.covs = "occ_cov1", det.covs = "effort")
dat
```

<code>tobs_fit_intervals</code>	<i>Fixed-effect covariance and credible intervals for tobs_fit</i>
---------------------------------	--

Description

On a nested-Laplace fit that keeps the per-cell pieces of its outer grid (for example `occu_cover()` or `abun()` with an areal field), the covariance is the grid-marginalized one and the bounds invert the Gaussian mixture the grid defines, rather than being read off the Monte Carlo draws the fit also carries; `coef()`, `summary()` and `tidy()` read the same mixture. Every other fit reads as `tulpa::vcov.tulpa_fit() / tulpa::confint.tulpa_fit()` describe.

Usage

```
## S3 method for class 'tobs_fit'
vcov(object, ...)
```

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

Arguments

object	A tobs_fit object.
...	Ignored.
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 posterior draws when they carry a column of that name.
level	Interval level (default 0.95).

Value

vcov(): the fixed-effect covariance matrix. confint(): a two-column matrix of lower and upper bounds, one row per requested parameter, carrying the interval_source, interval_declined, retained_mass and skew_applied attributes [tulpa::confint.tulpa_fit\(\)](#) documents.

tobs_format	<i>Format occupancy data from matrices/lists</i>
-------------	--

Description

Format occupancy data from matrices/lists

Usage

```
tobs_format(y, occ.covs = NULL, det.covs = NULL, coords = NULL, species = NULL)
```

Arguments

y	Detection history matrix (n_sites x max_visits) or 3D array.
occ.covs	Data.frame of site-level covariates, or named list.
det.covs	Named list of detection covariates. Each element is a vector (length n_sites, constant across visits) or matrix (n_sites x max_visits).
coords	Optional n_sites x 2 coordinate matrix.
species	Optional character or integer species identifier.

Value

An tobs_data object.

Examples

```

sim <- simulate_occu(N = 50, J = 3, seed = 1)
dat <- tobs_format(sim$y,
                  occ.covs = sim$data[, c("occ_cov1", "occ_cov2")],
                  det.covs = list(det_cov1 = sim$data$det_cov1))
dat

```

tobs_format_ms	<i>Format multi-species occupancy data</i>
----------------	--

Description

Format multi-species occupancy data

Usage

```

tobs_format_ms(
  y,
  occ.covs = NULL,
  det.covs = NULL,
  coords = NULL,
  species.names = NULL
)

```

Arguments

y	3D array (n_sites x max_visits x n_species) or named list of matrices.
occ.covs	Data.frame of site-level covariates.
det.covs	Named list of detection covariates.
coords	Optional n_sites x 2 coordinate matrix.
species.names	Optional character vector of species names.

Value

An tobs_data object with multi-species structure.

Examples

```

sim <- simulate_ms_occu(N = 40, J = 3, n.species = 4, seed = 1)
dat <- tobs_format_ms(sim$y, occ.covs = sim$data,
                    species.names = paste0("sp", 1:4))
dat$n_species

```

tobs_get	<i>Extract one species' fit from a batched occu_cover fit.</i>
----------	--

Description

Extract one species' fit from a batched occu_cover fit.

Usage

```
tobs_get(x, species)
```

Arguments

x	A tobs_batch.
species	Species label (character) or index (integer).

Value

The single-species tobs_fit for that species.

Examples

```
sim_a <- simulate_occu_cover(N = 30, J = 3, seed = 1)
sim_b <- simulate_occu_cover(N = 30, J = 3, seed = 2)
long <- rbind(
  data.frame(site = rep(1:30, each = 3), visit = rep(1:3, 30), sp = "a",
    occur = as.vector(t(sim_a$y)),
    cover = as.vector(t(ifelse(is.na(sim_a$y_pos), 0,
      sim_a$y_pos)))),
  data.frame(site = rep(1:30, each = 3), visit = rep(1:3, 30), sp = "b",
    occur = as.vector(t(sim_b$y)),
    cover = as.vector(t(ifelse(is.na(sim_b$y_pos), 0,
      sim_b$y_pos))))))
long$det_cov1 <- rep(sim_a$visit_data$det_cov1, 2)
long$occ_cov1 <- rep(rep(sim_a$data$occ_cov1, each = 3), 2)
fit <- tobs(~ occ_cov1, data = long, family = occu_cover("lognormal"),
  detection = ~ det_cov1, positive = ~ 1, method = "laplace",
  by = "sp", site = "site", visit = "visit", response = "occur",
  y.pos = "cover", det.covs = "det_cov1",
  control = list(verbose = FALSE))
fit_a <- tobs_get(fit, "a")
coef(fit_a)
```

tobs_gof_tests	<i>Goodness-of-fit tests for a tobs_fit</i>
----------------	---

Description

The **tulpa** goodness-of-fit generics for `tobs_fit`: dispersion, zero-inflation, and outlier counts compared against the fitted model's simulated replicates. For the Kolmogorov-Smirnov uniformity test, hand `pit_residuals()` to `tulpa::test_uniformity()`, which takes a PIT vector directly.

Usage

```
## S3 method for class 'tobs_fit'
test_dispersion(
  object,
  n.samples = 250,
  nsim = NULL,
  seed = NULL,
  observed = NULL,
  alternative = c("greater", "two.sided", "less"),
  ...
)

## S3 method for class 'tobs_fit'
test_zero_inflation(
  object,
  n.samples = 250,
  nsim = NULL,
  seed = NULL,
  observed = NULL,
  ...
)

## S3 method for class 'tobs_fit'
test_outliers(
  object,
  n.samples = 250,
  nsim = NULL,
  seed = NULL,
  observed = NULL,
  ...
)
```

Arguments

<code>object</code>	A fitted <code>tobs_fit</code> .
<code>n.samples</code>	Number of posterior-predictive replicates to simulate.

nsim	The same budget under the name the tulpa default methods use; overrides <code>n.samples</code> when given.
seed	Optional random seed for the replicates; the caller's RNG stream is restored afterwards.
observed	Must be NULL: a tobs fit scores the response it was fitted to. Accepted so a call spelled for the tulpa default reaches the method.
alternative	Tail of the dispersion p-value: "greater" (default, over-dispersion), "two.sided" or "less".
...	Must be empty; an unrecognised argument is an error.

Value

A list with the observed statistic (`observed`), its posterior-predictive expectation (`expected`), their ratio (`ratio`) and a tail p-value (`p.value`). The names are the same on every family, so a caller can read the statistic without branching on the model type.

tobs_marginal_effect	<i>Compute marginal effect of a covariate</i>
----------------------	---

Description

Returns the fitted response over the range of one covariate, holding the others at their means, on the process's response scale (occupancy / detection probability, or abundance intensity).

Usage

```
tobs_marginal_effect(
  object,
  covariate,
  process = c("occupancy", "detection", "abundance"),
  n.points = 100L
)
```

Arguments

object	A <code>tobs_fit</code> object.
covariate	Name of covariate.
process	"occupancy" (default), "detection", or "abundance" (the state process of a count family, on the intensity scale).
n.points	Number of prediction points (default 100).

Value

A `data.frame` with covariate values and the predicted response.

Examples

```
sim <- simulate_abun(N = 100, J = 4, n.abund.covs = 2, n.det.covs = 1,
                    seed = 1)
fit <- tobs(~ abund_cov1 + abund_cov2, data = sim$data,
           family = abun(K.max = 50), detection = ~ det_cov1, y = sim$y,
           control = list(verbose = FALSE))
me <- tobs_marginal_effect(fit, "abund_cov1", process = "abundance")
head(me) # estimate is on the abundance (lambda) scale, not a probability
```

tobs_multiarm_methods *S3 methods for multi-arm fits*

Description

Coefficient, covariance, and summary accessors for fits that store independent per-arm coefficient blocks (cover_fit, occu_categorical_fit) rather than a flat posterior. The arms are fit independently, so the joint covariance is block-diagonal. When a full posterior is present (the NUTS cover path) the inference methods defer to the draw-based tulpa_fit methods.

Usage

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

## S3 method for class 'tobs_multiarm_fit'
coef(object, arm = NULL, ...)

## S3 method for class 'tobs_multiarm_fit'
vcov(object, ...)

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

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

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

## S3 method for class 'tobs_multiarm_fit'
tidy(x, conf.level = 0.95, ...)

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

## S3 method for class 'tobs_multiarm_fit'
```

```

plot(x, ...)

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

## S3 method for class 'tobs_multiarm_fit'
residuals(object, ...)

```

Arguments

<code>object, x</code>	A <code>tobs_multiarm_fit</code> (a <code>cover_fit</code> or <code>occu_categorical_fit</code>).
<code>...</code>	Ignored, or forwarded to <code>NextMethod()</code> on the posterior path.
<code>arm</code>	Optional arm name ("presence", "positive" or "class") for <code>coef()</code> ; see coef.tobs_fit() .
<code>parm</code>	Ignored (present for <code>confint()</code> generic compatibility).
<code>level</code>	Confidence level for <code>confint()</code> and <code>summary()</code> (default 0.95).
<code>conf.level</code>	Interval level for <code>tidy()</code> (default 0.95).

Value

The layout every `tobs_fit` returns: `nobs()` an integer; `coef()` a named vector (`presence_(Intercept)`, `positive_x`, `class_2:x`, ...); `vcov()` the block-diagonal covariance and `confint()` a two-column matrix, both named as `coef()`; `logLik()` a `logLik` object; `tidy()` the arm / term / estimate / std.error / conf.low / conf.high table; `summary()` a `summary.tulpa_fit` object whose `coef()` is the estimate / std.error / interval table, rows named as `coef()`; `glance()` the one-row column set of [glance.tobs_fit\(\)](#); `plot()` one Gaussian-density panel per coefficient (up to 4), invisibly.

<code>tobs_predict_spatial</code>	<i>Predict the state process at new spatial locations</i>
-----------------------------------	---

Description

Generates state-process predictions at new coordinates, including the fitted spatial field interpolated to those locations by inverse-distance weighting over its five nearest nodes. The returned scale follows the family's state-process link: occupancy probability for occupancy families, abundance intensity (λ) for the count families.

Usage

```

tobs_predict_spatial(
  object,
  newcoords,
  newocc.covs = NULL,
  quantiles = c(0.025, 0.5, 0.975),
  node.coords = NULL
)

```

Arguments

<code>object</code>	A <code>tobs_fit</code> object fitted with a spatial component.
<code>newcoords</code>	Matrix of new coordinates (<code>n_new</code> x 2).
<code>newocc.covs</code>	Optional data.frame of state-process covariates at the new locations, one row per row of <code>newcoords</code> , on the scale and in the variables the fit's state formula names.
<code>quantiles</code>	Quantile levels for credible intervals (default 0.025, 0.5, 0.975). Strictly increasing, each in (0, 1); one column is reported per level, named from that level (q2.5, q50, q97.5).
<code>node.coords</code>	Optional matrix of coordinates for the fitted field's nodes (<code>n_nodes</code> x 2), required for an areal field and ignored when the term already carries its own coordinates.

Details

The field is taken from the sampled field columns of `object$draws` when the fit has them (the NUTS paths), so the reported `sd` and `quantiles` carry the field's own posterior variance. On the deterministic backends (`laplace` / `nested_laplace`) the draws hold only the coefficients, so the posterior-mean surface in `object$spatial_field` is used instead: it enters every draw as the same offset, and the reported spread is then the coefficients' alone.

A continuous term (`gp()`, `spde()`, `svc()`) carries the coordinates of its own nodes, so nothing extra is needed. An areal term (`icar()`, `bym2()`, `car()`) has graph nodes and no geometry, so interpolating it to a new point is only defined once the nodes are placed: pass `node.coords`, one row per element of `object$spatial_field`. Without it the call is an error rather than a prediction that quietly drops the field.

`newocc.covs` is expanded through the fit's own state-process formula and its columns are then keyed to the fitted coefficient names, so the frame may hold its covariates in any order and may carry factors, interactions and transforms. It needs one row per row of `newcoords`.

`newocc.covs = NULL` predicts at the fitted covariate MEANS, which the intercept-plus-zeros design expresses only where the state covariates were centred. A fit whose state design is not centred is an error rather than a prediction at covariate values the caller did not choose.

Value

A data.frame with `mean`, `sd`, and one column per requested quantile level (on the response scale: occupancy probability or abundance intensity), carrying the levels used in `attr(, "quantiles")`.

Examples

```
sim <- simulate_occu(N = 30, J = 3, n.occ.covs = 1, seed = 1)
d <- transform(sim$data, cell = seq_len(30))
adj <- matrix(0L, 30, 30)
adj[cbind(1:29, 2:30)] <- 1L
adj[cbind(2:30, 1:29)] <- 1L
fit <- tobs(~ occ_cov1 + icar(graph = adj, group.var = "cell"),
            data = d, family = occu(), detection = ~ 1, y = sim$y,
            method = "nested_laplace",
            control = list(verbose = FALSE, progress = FALSE))
```

```
# the chain graph's nodes placed at 1, ..., 30 on a line
tobs_predict_spatial(fit, newcoords = cbind(c(5.5, 20.5), 0),
                     newocc.covs = data.frame(occ_cov1 = c(0, 1)),
                     node.coords = cbind(1:30, 0))
```

tobs_richness	<i>Estimate species richness from community model</i>
---------------	---

Description

Per-site expected species richness $\sum_s \psi_{s,i}$ with a posterior credible interval propagated from the community-mean occupancy draws. For the dynamic community family this is the first-season richness (from ψ_1).

Usage

```
tobs_richness(object)
```

Arguments

object	A tobs_fit object from a community occupancy model (ms_occu(), ms_dyn_occu(), or ms_int_occu()).
--------	--

Value

A data.frame with site-level richness estimates.

Examples

```
sim <- simulate_ms_occu(N = 40, J = 3, n.species = 5, seed = 1)
fit <- tobs(~ x, data = sim$data, family = ms_occu(), detection = ~ 1,
           y = sim$y, species = paste0("sp", 1:5),
           control = list(verbose = FALSE))
head(tobs_richness(fit))
```

tobs_stack	<i>Stack fitted models into a LOO-weighted ensemble</i>
------------	---

Description

Combines two or more fitted models into one predictive distribution with leave-one-out predictive stacking. Each member contributes its Pareto-smoothed importance-sampling LOO predictive density; the weights maximize the stacked leave-one-out predictive density and so favour members that predict held-out observations best. Works across every family (single-season / community / dynamic / integrated occupancy, JSMD, and the cover hurdle), each scored by the marginal likelihood in `.tobs_pointwise_loglik()`.

Usage

```
tobs_stack(..., method = c("stacking", "pseudobma"))

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

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

## S3 method for class 'tobs_stack'
predict(
  object,
  X.0 = NULL,
  quantiles = c(0.025, 0.5, 0.975),
  n.draws = 4000L,
  ...
)
```

Arguments

<code>...</code>	ignored.
<code>method</code>	weighting scheme passed to <code>loo::loo_model_weights()</code> : "stacking" (default) or "pseudobma".
<code>x</code>	a <code>tobs_stack</code> object.
<code>object</code>	a <code>tobs_stack</code> object.
<code>X.0</code>	optional occupancy design matrix for out-of-sample prediction. When supplied, all members must share the same occupancy design.
<code>quantiles</code>	credible-interval quantile levels.
<code>n.draws</code>	size of the pooled stacked-predictive sample (design-matrix mode); each member contributes a share proportional to its weight.

Details

The members must be fit to the same observation set. They may otherwise differ freely – different covariates, priors, or inference methods. (An ensemble of pure seed-variants, e.g. from `tobs(..., control = list(n.seeds = K))`, is a valid but degenerate case: the members are statistically identical, so the weights come out roughly uniform.)

Value

An object of class `tobs_stack`: a list with `weights` (named numeric, summing to 1), `fits` (the members), `loo` (per-member `loo::loo()` objects), `comparison` (a data frame of `elpd_loo`, `weight`, and worst Pareto-k per member), and `method`. `stats::predict()` / `stats::fitted()` on the result return the weight-combined predictive.

Examples

```
sim <- simulate_occu(N = 100, J = 3, n.occ.covs = 2, n.det.covs = 1,
                    seed = 1)
ctrl <- list(verbose = FALSE, progress = FALSE)
f1 <- tobs(~ occ_cov1, data = sim$data, family = occu(),
          detection = ~ det_cov1, y = sim$y, method = "laplace",
          control = ctrl)
f2 <- tobs(~ occ_cov1 + occ_cov2, data = sim$data, family = occu(),
          detection = ~ det_cov1, y = sim$y, method = "laplace",
          control = ctrl)
ens <- tobs_stack(simple = f1, full = f2)
ens$weights
# weight-combined in-sample psi / p / z
str(predict(ens))
```

tobs_terms

Structured terms inside a tobs() formula

Description

Spatial fields, random effects, temporal structure, varying coefficients and latent factors are written inside the model formula, in the process they belong to, rather than passed as `tobs()` arguments. A term entered in the state formula acts on occupancy or abundance; the same term written in `detection = ~ ...` acts on detection.

Details

These are formula specials, not exported functions: they are resolved against an internal registry while the formula is parsed, and calling them outside a `tobs()` formula is an error. This page documents the call shapes and their arguments.

Areal spatial fields

`icar(graph)`, `bym2(graph, scale.factor)`, `car(graph)` and `car_proper(graph)` put a field over the nodes of a symmetric adjacency matrix graph. `bym2()` computes its Riebler `scale.factor` from the graph when it is not supplied. All four accept:

group.var Column naming each observation's graph node, when the graph is over regions rather than rows.

weight Per-node numeric column turning the field into a spatially varying coefficient (`weight_i * z[node_i]`) instead of an intercept field. `car()` does not take it.

id Name for the realization, so `share("id")` can share it with another process.

The umbrella form `spatial(~ 1 + w || cell, graph = adj, model = "icar")` forwards to the same terms and additionally reads a bar, which is how areal varying coefficients are usually written (see below).

An intrinsic areal field (`icar`, `bym2`, `car`) has one constant null direction per CONNECTED COMPONENT of its graph, and each of them is identified separately: the engine augments the precision with one sum-to-zero per component (INLA's `adjust.for.con.comp = TRUE`), and the components share one field variance. A graph with islands or with separate survey regions is therefore fitted as intended. Because the component count is also what an adjacency looks like when a spatial join has quietly dropped edges, and because it is not visible in the fitted object, a graph with more than one component reports its component count and sizes at fit time.

A component of a single node has no neighbours to borrow from. The areal cover paths reject one outright, with an error naming the node; where such a node is accepted it keeps a proper $N(0, 1/\tau)$ effect, because the identification augments the precision rather than imposing a hard constraint, which on a lone node would pin it to exactly zero.

Continuous spatial fields

`gp(lon, lat, ...)` is an NNGP-approximated Gaussian process over coordinates, `multiscale_gp(lon, lat, ...)` a two-scale (local plus regional) version, and `spde(lon, lat, ...)` a Matern field over a triangular mesh. Coordinates come either from bare column names or from `coords =`, a two-column matrix.

`cov` Covariance function: "exponential" (default), "matern", "gaussian", "spherical".

`nu` Matern smoothness.

`nn` Number of nearest neighbours in the Vecchia approximation, capped at $n - 1$. `multiscale_gp()` splits this into `nn.local` and `nn.regional`.

`prior.range` Length-2 $c(r_0, \alpha)$ PC prior on the range, read as $P(\text{range} < r_0) = \alpha$. Required by `gp()` and `svc()`, which ship no default; `spde()` defaults to $c(0.5, 0.5)$ alongside `prior.sigma`.

`mesh, max.edge, cutoff` `spde()` mesh controls, passed to `tulpa::spatial_spde()`.

Spatially varying coefficients

A coefficient that varies over space is a spatial field like any other, so it is written under `spatial()` and the `model = choice` says whether space is a graph or a set of coordinates. In both forms you name the coefficients that vary; the difference is only what they vary over.

Areal, over graph nodes, written as a bar:

```
~ 1 + w + spatial(~ 1 + w || cell, graph = adj)
```

with `method = "nested_laplace"`. The bar's left-hand side names the varying coefficients (one field per column, the intercept included), its right-hand side the graph node. Broad family support (single-season, dynamic, integrated and the community routes) and recovery-tested throughout. This is what most spatially-varying-coefficient questions want. `model =` selects the areal structure ("`icar`" by default, or "`bym2`" / "`car`" / "`car_proper`").

Continuous, over coordinates, as NNGP surfaces:

```
~ elevation + spatial(lon, lat, model = "svc",
                      coefficients = "elevation", prior.range = c(50, 0.05))
```

There is no node index to group on, so the coefficients are named directly instead of through a bar.

coefficients Names of the design columns whose coefficients vary, matched against the design of the arm the surfaces load on. One surface per entry; "(Intercept)" gives a plain intercept surface.

indices The same selection by column position, for a design column with no usable name. Give coefficients or indices, not both.

coords Two-column coordinate matrix, as an alternative to bare lon, lat column names. One coordinate pair per site.

cov "exponential" (default), "matern" or "gaussian".

nn Nearest neighbours in the Vecchia approximation, default 15, capped at $n - 1$.

prior.range Required, no default: $c(r_0, \alpha)$ with $P(\text{range} < r_0) = \alpha$.

sigma2.prior.scale Scale of the prior on the surface's marginal variance.

`svc(lon, lat, coefficients = )` is the direct constructor and stays available, the way `icar()` does alongside `spatial(model = "icar")`.

The continuous form fits on single-season `occu()` under "laplace", "nested_laplace" and "nuts", and on `removal()`, `distance()`, `fp_occu()` and `dyn_abun()` under "laplace" and "nested_laplace". The surfaces load on the state arm (the occupancy logit, or log lambda for the count families) and come back as `fit$svc_field`, with the integrated hyperparameters in `fit$svc_hyper` and the resolved design columns in `fit$svc_indices`. On any other family, method or arm the term errors and points at the areal bar rather than being silently dropped.

Random effects

`re(group, ...)` is a grouped random effect, and lme4 bars are sugar over it: $(1 | g)$, $(x | g)$, $(x || g)$, $(0 + x | g)$, $(1 | g:h)$ and $(1 | g/h)$ are rewritten to `re()` calls before the formula is evaluated.

type "intercept" (default), "slope" or "iid".

covariate Slope column(s): a column name, several names, a `cbind()` matrix or a one-sided formula. Required for type = "slope".

model Structure over the group levels: "iid" (default), "ar1", "rw1", "rw2".

correlated Whether slopes share a correlated covariance with the group intercept.

intercept FALSE drops the implicit group intercept, giving a slope-only block.

Temporal structure

`temporal(time, ...)` puts a field over the levels of time.

type "ar1" (default), "rw1", "rw2" or "iid".

group Optional grouping, for one series per group.

cyclic Wraps the last level onto the first.

tau.shape, tau.rate Gamma prior on the field precision.

Latent factors and sharing

`latent(n.factors)` adds per-site latent factors with per-species loadings to a community model, giving residual co-occurrence. `constraint` and `sigma.prior.rate` control the identification anchor and the loading prior.

`share("id")` shares one realization of the term named `id` across processes, optionally scaled: `alpha = 0.5` pins the coupling amplitude, `alpha = grid(c(0.25, 0.5, 1))` marginalizes over it on the outer integration.

`residual =` carries an arm-specific deviation beside the copy, so the sharing arm's surface is $\alpha * w + \delta$ rather than $\alpha * w$ alone. Use it where the two arms respond to related but not identical surfaces: a plain copy has no way to bend, so a pattern the shared field does not have is not fitted at all. `residual = "full"` gives the deviation one latent per field node, spanning the same surfaces as a field placed on the arm itself. `residual = r` gives it r basis functions on a basis orthogonalized against the shared field, so the latent problem is $K + r$ rather than $2K$, and the deviation cannot reproduce the copied component. Because the two halves are orthogonal, the reported amplitude is identified: it is the projection of the fitted arm surface onto the occurrence surface, taken on the linear-predictor scale. A rank- r deviation is pinned at the shared field's own SD, which costs one warm fit of the model without it. Available on `occu_cover()` under `method = "nested_laplace"`, one per fit, and it needs a shared field to deviate from.

`grid()` states the axis's NODES, and the span the fit works over is wider than the nodes themselves, for two separate reasons.

First, each node represents the cell around it, so the axis runs half a node step past the outermost node at each end, in the axis's own log coordinate. That is what makes the quadrature weights sum over a whole axis rather than stopping on the last node, and under `method = "nuts"` the sampled amplitude takes the same span as the support of its flat prior, so one stated grid moves both backends over one region. For k equally log-spaced nodes this is exact: $k - 1$ steps between the nodes plus a half step at each end, so the span is $k / (k - 1)$ times the stated range in the log coordinate. It is therefore largest for the shortest grid – $2x$ at two nodes, $1.125x$ at nine. With refinement off, `c(0.2, 0.5)` gives $[0.127, 0.791]$.

Second, refinement. `control$adaptive.grid` is TRUE by default, and what it does depends on who placed the axis. An axis YOU stated is DENSIFIED: further nodes are placed inside the range you wrote, never past its ends, so the nodes you give are a bound and the half step above is the whole of the widening. An axis the fit placed for you – the default copy axis, a field SD axis – is EXTENDED when a boundary cell still holds weight (`adaptive.grid.edge.thresh`, default 0.02), which is what lets a fit whose default grid misses the mode go and find it.

Refining an axis changes how densely its nodes sit, not the region they cover: the reported span stays the declared span unless a refinement point lands past it. A stated `c(0.2, 0.5)` is only densified, so it spans $[0.127, 0.791]$ with refinement on as with it off. Holding an axis at exactly the nodes stated takes BOTH `control$adaptive.grid = FALSE` and `control$var.of.means.consistency = FALSE`: refinement reaches an axis from two independent passes, and the consistency pass runs whatever the adaptive grid is set to, so a fit that turns off only the first still integrates nodes placed inside the range it wrote. Under `method = "nuts"` a fit reports the span each sampled hyperparameter was worked over in `fit$nuts$hyper_support`, which is the region that hyperparameter's flat prior is supported on.

See Also

[tobs](#) for the model front door.

Examples

```
# `occu()` takes the detection history as `y` and a ONE-sided state formula:
# a two-sided formula would be resolved against a response the family does
# not carry. `data` is site-level, `visits` visit-level.
sim <- simulate_occu(N = 30, J = 3, seed = 1)
d <- transform(sim$data, cell = seq_len(30),
               lon = runif(30), lat = runif(30))

# A chain graph over the 30 cells: neighbours are consecutive indices.
adj <- matrix(0L, 30, 30)
adj[cbind(1:29, 2:30)] <- 1L
adj[cbind(2:30, 1:29)] <- 1L

ctrl <- list(verbose = FALSE, progress = FALSE)

# Areal field on occupancy
fit <- tobs(~ occ_cov1 + icar(graph = adj, group.var = "cell"),
           data = d, family = occu(), detection = ~ 1, y = sim$y,
           method = "nested_laplace", control = ctrl)
fit$spatial_field[1:5]

# Detection random effect by observer, who changes between visits
obs <- simulate_occu(N = 60, J = 3, n.visit.groups = 4, seed = 1)
fit_re <- tobs(~ occ_cov1, data = obs$data, family = occu(),
              detection = ~ (1 | visit_group), y = obs$y,
              visits = obs$visits, method = "laplace", control = ctrl)
coef(fit_re)

# Areal spatially varying coefficient on `occ_cov2`
fit_svc <- tobs(~ occ_cov2 + spatial(~ 1 + occ_cov2 || cell, graph = adj),
               data = d, family = occu(), detection = ~ 1, y = sim$y,
               method = "nested_laplace", control = ctrl)
coef(fit_svc)

# Continuous NNGP varying coefficient on the `occ_cov2` slope
fit_nngp <- tobs(~ occ_cov2 + spatial(lon, lat, model = "svc",
                                     coefficients = "occ_cov2",
                                     prior.range = c(0.5, 0.05)),
               data = d, family = occu(), detection = ~ 1, y = sim$y,
               method = "laplace", control = ctrl)
coef(fit_nngp)
```

update.tobs_fit

Update and refit a tobs model

Description

Re-enters `tobs()` with the arguments the fit was originally built from (`$tobs_call`, recorded by every family), so the refit keeps the fit's family dispatcher, method and control instead of drop-

ping to a bare Laplace refit of the occupancy state alone. A name in ... that is one of `tobs()`'s own arguments (formula, data, family, occurrence, detection, positive, y, visits, method, priors, control) replaces that argument outright; any other name (`n.iter`, `seed`, `verbose`, ...) is merged into `control`, matching where the recorded fit's own value for it lives.

Usage

```
## S3 method for class 'tobs_fit'
update(object, ..., evaluate = TRUE)
```

Arguments

- `object` A `tobs_fit` object (built via `tobs()`).
- `...` Named arguments to override; see Details on where each lands.
- `evaluate` If TRUE (default), refit the model. If FALSE, return the unevaluated `tobs()` call.

Value

Updated `tobs_fit` object (or an unevaluated call if `evaluate = FALSE`).

<code>within_between</code>	<i>Within / between (Mundlak) decomposition for longitudinal data</i>
-----------------------------	---

Description

For each variable in `vars`, adds two new columns to data:

Usage

```
within_between(data, group, vars, suffix = c("_btw", "_wtm"), na.rm = TRUE)
```

Arguments

- `data` A data frame.
- `group` Character name (or character vector of names) identifying the grouping column(s). With multiple names, grouping is by the interaction.
- `vars` Character vector of numeric columns to decompose. Must all exist in data.
- `suffix` Length-2 character vector giving the suffixes for the between- and within-group columns. Default `c("_btw", "_wtm")`.
- `na.rm` Logical; passed to `mean()` when computing per-group means. Default TRUE.

Details

- `<var><suffix[1]>` - the per-group mean of `var` (the **between**-group component, constant within each group),
- `<var><suffix[2]>` - `var - <var><suffix[1]>` (the **within**-group deviation, summing to zero within each group).

The decomposition is exact: `data[[var]] == data[[btw]] + data[[wtn]]`. Including both columns in a regression (`~ x_btw + x_wtn`) separates the cross-group association from the within-group association without imposing that they be the same coefficient (the random-effects assumption).

Typical use in the MOTIVATE cover-resurvey workflow:

```
plots <- within_between(plots, group = "plot_id", vars = "year")
fit <- tobs(~ year_btw + year_wtn + bym2(graph = adj), data = plots,
           family = cover("beta"), y = plots$cover,
           method = "nested_laplace")
```

`year_wtn` then carries the within-plot temporal trend; `year_btw` absorbs across-plot timing heterogeneity (older versus newer baseline plots).

Value

The input data with `2 * length(vars)` new columns appended.

References

- Mundlak, Y. (1978). On the pooling of time series and cross section data. *Econometrica*, 46(1), 69-85.
- Bell, A. & Jones, K. (2015). Explaining fixed effects: random effects modeling of time-series cross-sectional and panel data. *Political Science Research and Methods*, 3(1), 133-153.

Examples

```
set.seed(1)
plots <- data.frame(
  plot_id = rep(1:5, each = 4),
  year    = rep(2000:2003, times = 5) + rep(c(0, 5, 10, 15, 20), each = 4),
  cover   = stats::runif(20)
)
decomposed <- within_between(plots, group = "plot_id", vars = "year")
head(decomposed)
```

Index

* datasets

- foray_counts, 29
- meadow_cover, 36
- peatland_occu, 82

`$.tobs_fit`, 5

`abun`, 5

`abun()`, 10, 24, 29, 68, 75, 103, 146–149, 156, 160

`base::suppressMessages()`, 15, 72

`bym2 (tobs_terms)`, 171

`car_proper (tobs_terms)`, 171

`check_model.tobs_fit`, 6

`coef()`, 160

`coef.tobs_batch`, 7

`coef.tobs_fit`, 8

`coef.tobs_fit()`, 144, 167

`coef.tobs_multiarm_fit (tobs_multiarm_methods)`, 166

`confint.tobs_fit (tobs_fit_intervals)`, 160

`confint.tobs_multiarm_fit (tobs_multiarm_methods)`, 166

`converged (convergence)`, 8

`convergence`, 8

`count`, 10

`count()`, 39, 40, 104, 148

`cover`, 11

`cover()`, 16, 37, 67–69, 75, 79, 146–149

`cover_priors`, 16

`cpo.tobs_fit (tobs_criteria)`, 157

`dic.tobs_fit (tobs_criteria)`, 157

`distance`, 17, 173

`distance()`, 22, 33, 40–42, 108, 149

`distance_laplace`, 19

`distsamp_open`, 20

`distsamp_open()`, 109, 110

`double_observer`, 22

`double_observer()`, 32, 33, 111

`dyn_abun`, 24, 173

`dyn_abun()`, 20–22, 112, 149

`dyn_abun_laplace`, 25

`dyn_int_occu`, 26

`dyn_int_occu()`, 113, 148

`dyn_occu`, 27

`dyn_occu()`, 26, 43, 144, 148, 149

`fitted.tobs_fit`, 28

`fitted.tobs_multiarm_fit (tobs_multiarm_methods)`, 166

`fitted.tobs_stack (tobs_stack)`, 169

`foray_counts`, 29

`fp_occu`, 30, 173

`fp_occu()`, 68, 116, 149, 156

`fp_occu_laplace`, 31

`gdistremoval`, 32

`gdistremoval()`, 20–22, 117

`glance()`, 153

`glance.tobs_fit`, 33

`glance.tobs_fit()`, 167

`glance.tobs_multiarm_fit (tobs_multiarm_methods)`, 166

`gp (tobs_terms)`, 171

`icar (tobs_terms)`, 171

`int_occu`, 34

`int_occu()`, 28, 44, 91, 148, 149

`jsdm`, 35

`jsdm()`, 10, 39, 119, 147–149

`latent (tobs_terms)`, 171

`logLik.tobs_fit`, 36

`logLik.tobs_multiarm_fit (tobs_multiarm_methods)`, 166

`loo.tobs_fit (tobs_criteria)`, 157

`loo::loo()`, 158, 170

loo::loo_compare(), 158
 loo::loo_model_weights(), 170
 loo::waic(), 158

 meadow_cover, 36
 ms_abun, 37
 ms_abun(), 40, 42, 45, 48, 120, 148, 149
 ms_count, 39
 ms_count(), 35, 148, 149
 ms_distance, 40
 ms_distance(), 123, 124, 149
 ms_dyn_occu, 42
 ms_dyn_occu(), 45, 125, 148, 149
 ms_int_occu, 43
 ms_int_occu(), 45, 126, 148, 149
 ms_occu, 44
 ms_occu(), 39, 40, 43–45, 48, 148, 149
 ms_occu_cover, 45
 ms_occu_cover(), 39, 70, 72, 128, 149, 153, 154
 multiscale_gp(tobs_terms), 171

 nmix_laplace, 48
 nmix_laplace(), 58, 61–63
 nmix_laplace_bym2, 50
 nmix_laplace_car_proper, 53
 nmix_laplace_icar, 55
 nmix_laplace_icar(), 52–54
 nmix_laplace_re, 57
 nmix_site_marginal, 62
 nmix_site_marginal(), 57, 61
 nobs.tobs_fit, 64
 nobs.tobs_fit(), 36
 nobs.tobs_multiarm_fit
 (tobs_multiarm_methods), 166

 obs_family(), 147
 occu, 64, 173
 occu(), 45, 69, 75, 82, 146–149, 156
 occu_aggregation_scan, 65
 occu_categorical, 67
 occu_cover, 68
 occu_cover(), 45–48, 68, 75, 76, 78, 79, 133, 136, 147, 149, 160
 occu_cover_inputs, 75
 occu_cover_inputs(), 154
 occu_multi, 77
 occu_multi(), 136
 occu_multiscale_cover, 78

 occu_multiscale_cover(), 137, 149
 occu_priors, 79
 occu_priors(), 16, 17, 149
 occu_ttd, 81
 occu_ttd(), 138

 peatland_occu, 82
 pit_residuals(), 7, 164
 pit_residuals.tobs_fit, 83
 plot.tobs_aggregation_scan, 84
 plot.tobs_data, 84
 plot.tobs_multiarm_fit
 (tobs_multiarm_methods), 166
 pointwise_loglik.tobs_fit
 (tobs_criteria), 157
 ppc, 85
 ppc(), 7
 predict.cover_fit, 86
 predict.occu_categorical_fit, 87
 predict.tobs_fit, 88
 predict.tobs_fit(), 86, 87
 predict.tobs_stack(tobs_stack), 169
 print.tobs_batch, 92
 print.tobs_family, 92
 print.tobs_fit, 93
 print.tobs_stack(tobs_stack), 169

 ranef(), 72, 74
 ranef.tobs_fit, 93
 re(tobs_terms), 171
 removal, 94, 173
 removal(), 23, 33, 139, 149, 156
 removal_laplace, 95
 residuals.tobs_fit, 97
 residuals.tobs_multiarm_fit
 (tobs_multiarm_methods), 166
 royle_nichols, 97
 royle_nichols(), 141

 sbc.tobs_fit, 98
 simulate.tobs_fit, 102
 simulate_abun, 103
 simulate_abun(), 29
 simulate_count, 104
 simulate_cover, 105
 simulate_cover(), 37
 simulate_cover_joint, 106
 simulate_cover_joint(), 13
 simulate_distance, 108

simulate_distsamp_open, 109
 simulate_double_observer, 111
 simulate_dyn_abun, 112
 simulate_dyn_int_occu, 113
 simulate_dyn_occu, 114
 simulate_fp_occu, 116
 simulate_gdistremoval, 117
 simulate_int_occu, 118
 simulate_jsdm, 119
 simulate_ms_abun, 120
 simulate_ms_count, 122
 simulate_ms_distance, 123
 simulate_ms_dyn_occu, 124
 simulate_ms_int_occu, 126
 simulate_ms_occu, 127
 simulate_ms_occu_cover, 128
 simulate_ms_occu_cover_spatial, 129
 simulate_occu, 131
 simulate_occu(), 82
 simulate_occu_categorical, 132
 simulate_occu_cover, 133
 simulate_occu_cover(), 70
 simulate_occu_multi, 136
 simulate_occu_multiscale_cover, 137
 simulate_occu_ttd, 138
 simulate_removal, 139
 simulate_royle_nichols, 140
 simulate_t_occu, 142
 spatial (tobs_terms), 171
 spde (tobs_terms), 171
 stats::confint(), 8
 stats::fitted(), 170
 stats::predict(), 170
 stats::simulate(), 102
 stats::vcov(), 8
 summary(), 160
 summary.tobs_data, 143
 summary.tobs_fit, 143
 summary.tobs_multiarm_fit
 (tobs_multiarm_methods), 166
 svc (tobs_terms), 171

 t_occu, 144
 t_occu(), 142, 148
 temporal (tobs_terms), 171
 test_dispersion(), 7
 test_dispersion.tobs_fit
 (tobs_gof_tests), 164
 test_outliers.tobs_fit
 (tobs_gof_tests), 164
 test_zero_inflation.tobs_fit
 (tobs_gof_tests), 164
 tidy(), 160
 tidy.tobs_fit, 145
 tidy.tobs_multiarm_fit
 (tobs_multiarm_methods), 166
 tobs, 145, 174
 tobs(), 5, 7, 10, 18, 21, 23, 27, 29, 30, 32, 37,
 39, 75–77, 80–82, 92, 94, 98, 103,
 108, 112, 116, 120, 123, 128, 139,
 175, 176
 tobs_associations, 154
 tobs_associations(), 47
 tobs_check_id, 156
 tobs_criteria, 157
 tobs_data, 158
 tobs_data(), 76, 147, 153
 tobs_fit_intervals, 160
 tobs_format, 161
 tobs_format(), 147
 tobs_format_ms, 162
 tobs_get, 163
 tobs_get(), 154
 tobs_gof_tests, 164
 tobs_marginal_effect, 165
 tobs_multiarm_methods, 166
 tobs_predict_spatial, 167
 tobs_richness, 169
 tobs_stack, 169
 tobs_stack(), 150, 154
 tobs_terms, 171
 tulpa::check_model(), 6, 7
 tulpa::confint.tulpa_fit(), 160, 161
 tulpa::moran_i(), 7
 tulpa::pit_residuals(), 83
 tulpa::pp_check(), 85
 tulpa::sbc(), 101
 tulpa::summary.tulpa_fit(), 144
 tulpa::test_uniformity(), 83, 164
 tulpa::tulpa_criteria(), 157, 158
 tulpa::tulpa_laplace(), 16, 80
 tulpa::tulpa_laplace_beta(), 16
 tulpa::tulpa_nested_laplace_joint(),
 15, 70, 73, 147
 tulpa::tulpa_posterior_draws(), 88
 tulpa::vcov.tulpa_fit(), 160

`tulpa_re_aghq()`, [57](#), [59–63](#)

`update.tobs_fit`, [175](#)

`vcov.tobs_fit(tobs_fit_intervals)`, [160](#)

`vcov.tobs_multiarm_fit`
 (`tobs_multiarm_methods`), [166](#)

`waic()`, [101](#)

`waic.tobs_fit(tobs_criteria)`, [157](#)

`within_between`, [176](#)

`within_between()`, [36](#), [37](#)