

Package: tulpaRatio (via r-universe)

October 13, 2026

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

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

Title Bayesian Hierarchical Models for Ratios, Rates, and Proportions

Version 1.7.3

Description An opinionated framework for Bayesian hierarchical modelling of ratios, rates, and proportions. Ratios are treated as derived quantities, with all inference performed on the latent processes generating the numerator and denominator. Supports shared random effects, latent factors for unmeasured confounders, spatial structure ('CAR'/'BYM2'/'GP'), spatiotemporal interaction (Knorr-Held Type I-IV), zero-inflation variants, and full uncertainty propagation to ratio posteriors. Provides multiple inference backends including custom 'HMC'/'NUTS' (default), 'Laplace' approximation, and 'Polya-Gamma' Gibbs sampling.

Language en-US

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

Imports bayesplot, posterior, loo, stats, tulpa (>= 0.7.5), Rcpp (>= 1.0.0)

LinkingTo Rcpp, RcppEigen, tulpa (>= 0.7.5)

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

Suggests patchwork, ggplot2, gridExtra, stars, sf, Matrix, MASS, testthat (>= 3.0.0), knitr, rmarkdown, svglite

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.1.0
Config/pak/sysreqs cmake libicu-dev
Repository <https://gcol33.r-universe.dev>
Date/Publication 2026-10-13 04:08:24 UTC
RemoteUrl <https://github.com/gcol33/tulparatio>
RemoteRef HEAD
RemoteSha 19b38ce69cd8413d0618686be812d0200a33050b

Contents

as.data.frame.ratiod_fit	4
as_draws.ratiod_fit	5
check_diagnostics	6
diagnostic_summary	7
diagnostics.ratiod_fit	8
draws_interval	9
draws_long	11
draws_wide	12
fitted.ratiod_fit	13
geweke_test	15
gpu-support	16
gpu_available	16
gpu_info	17
hurdle_binomial	18
hurdle_negbin	18
hurdle_poisson	19
inference_mode_info	20
latent_factor	20
loo.ratiod_fit	23
n_divergent	24
oi_binomial	25
plot_acf	25
plot_divergences	26
plot_energy	27
plot_ess	28
plot_map	29
plot_map_panel	31
plot_pairs	32
plot_rhat	33
posterior_predict.ratiod_fit	34
pp_check.ratiod_fit	35
predict.ratiod_fit	36
print.ratiod_sghmc	39
prior_beta	41
prior_exponential	42
prior_gamma	42

prior_half_cauchy	43
prior_half_normal	43
prior_normal	44
prior_pc	44
prior_predict	45
priors_default	47
ratio	48
ratio_contrast	49
ratiod_average	51
ratiod_benchmark	53
ratiod_benchmark_compare	54
ratiod_beta_binomial	55
ratiod_binomial	56
ratiod_compare	57
ratiod_families	58
ratiod_gamma_gamma	58
ratiod_hurdle_binomial	60
ratiod_hurdle_negbin	60
ratiod_hurdle_pois	62
ratiod_latent	63
ratiod_lognormal	63
ratiod_negbin_gamma	64
ratiod_negbin_negbin	66
ratiod_oibinomial	67
ratiod_poisson_gamma	68
ratiod_priors	69
ratiod_spatial	71
ratiod_temporal	71
ratiod_threads	72
ratiod_validate	72
ratiod_zi_families	73
ratiod_zibinomial	73
ratiod_zinegbin	74
ratiod_zipois	76
ratiod_zoibinomial	77
sim_ratiod	78
sim_ratiod_sbc	81
spatial_bym2	82
spatial_car	84
spatial_gp	87
spatial_hsgp	89
spatial_multiscale	92
spatial_rsr	94
spatial_svc	97
spatiotemporal	100
svc	104
temporal	106
temporal_ar1	107

temporal_gp	109
temporal_iid	112
temporal_multiscale	113
temporal_rtr	115
temporal_rw1	118
temporal_rw2	120
temporal_tvc	122
tratio	124
tvc	129
validate_mode	131
waic.ratiod_fit	132
zi_binomial	133
zi_negbin	133
zi_poisson	134
zi_specification	135
zoib	136
Index	137

as.data.frame.ratiod_fit	<i>Posterior Draws of a Fitted Ratio Model as a Data Frame</i>
--------------------------	--

Description

Posterior Draws of a Fitted Ratio Model as a Data Frame

Usage

```
## S3 method for class 'ratiod_fit'  
as.data.frame(x, ...)
```

Arguments

- x A ratiod_fit object
- ... Ignored.

Value

A draws_df data frame: one row per posterior draw, one column per parameter, plus the .chain, .iteration and .draw columns.

Examples

```

sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
draws <- as.data.frame(fit)
dim(draws)
names(draws)[1:6]

```

as_draws.ratiod_fit	<i>Convert a Fitted Ratio Model to a Draws Object</i>
---------------------	---

Description

Convert a fitted ratio model to a format compatible with the posterior and tidybayes packages. Returns draws in a tidy format suitable for further analysis and visualization.

Usage

```

## S3 method for class 'ratiod_fit'
as_draws(x, ...)

```

Arguments

x	A ratiod_fit object
...	Additional arguments passed to posterior::as_draws_df

Details

The returned draws object is compatible with:

- **posterior** package functions (summarise_draws, rvar, etc.)
- **tidybayes** package functions (spread_draws, gather_draws, etc.)
- **bayesplot** package visualization functions

To work in tidybayes, extract the draws with this function and pass them on to its verbs.

Value

A draws_df object from the posterior package.

See Also

[posterior::as_draws_df\(\)](#), [ratio\(\)](#) for ratio-specific extraction

Examples

```
# Generate synthetic data
set.seed(111)
n <- 48
site <- rep(letters[1:12], each = 4)
u <- rnorm(12, sd = 0.5)[match(site, letters[1:12])]
depth <- rnorm(n)
df <- data.frame(
  count = rpois(n, exp(3 + 0.3 * depth + u)),
  effort = rgamma(n, shape = 11, rate = exp(-u)),
  depth = depth,
  site = site
)

# Fit model
fit <- tratio(
  count | effort ~ depth + (1 | site),
  data = df,
  family = ratioid_poisson_gamma(),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Convert to draws format
draws <- as_draws(fit)

# Use with posterior package
if (requireNamespace("posterior", quietly = TRUE)) {
  posterior::summarise_draws(draws)
}
```

check_diagnostics

*Check MCMC Diagnostics and Report Problems***Description**

Checks MCMC diagnostics (Rhat, ESS, divergences) and prints actionable warnings with suggestions for fixing issues.

Usage

```
check_diagnostics(fit, quiet = FALSE)
```

Arguments

fit	A ratioid_fit object
quiet	Logical; if TRUE, suppress output and just return status.

Value

Invisibly returns a list with diagnostic status:

ok Logical; TRUE if all diagnostics pass
n_divergent Number of divergent transitions
n_bad_rhat Number of parameters with Rhat > 1.01
n_low_ess Number of parameters with ESS < 400

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
check_diagnostics(fit)
```

diagnostic_summary	<i>Diagnostic Summary of a Fitted Ratio Model</i>
--------------------	---

Description

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

Usage

```
diagnostic_summary(fit)
```

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

Arguments

fit	A <code>ratiod_fit</code> object.
x	A <code>ratiod_diagnostic_summary</code> object.
...	Ignored.

Value

A list with class `ratiod_diagnostic_summary` containing:

status Overall status: "PASS", "WARN", or "FAIL"
n_divergent Number of divergent transitions
divergent_pct Percentage of divergent transitions

worst_rhat Data frame of parameters with worst Rhat
worst_ess Data frame of parameters with worst ESS
e_bfmi E-BFMI value (HMC only)
recommendations Character vector of recommendations

See Also

[check_diagnostics\(\)](#), [diagnostics\(\)](#), [plot_diagnostics\(\)](#)

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
diagnostic_summary(fit)
```

diagnostics.ratiod_fit

Posterior Diagnostics for a Fitted Ratio Model

Description

Computes per-parameter improved Rhat and bulk / tail effective sample size by delegating to the engine's [tulpa::diagnostics\(\)](#), the single source of truth for convergence diagnostics in the tulpa ecosystem. Works with all backends.

Usage

```
## S3 method for class 'ratiod_fit'
diagnostics(fit, pars = NULL, ...)

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

Arguments

<code>fit</code>	A <code>ratiod_fit</code> object
<code>pars</code>	Character vector of parameter names (default: all).
<code>...</code>	Ignored.
<code>x</code>	A <code>ratiod_diagnostics</code> object

Details

Rhat is the improved value of Vehtari et al. (2021): the maximum of rank-normalized split-Rhat and folded split-Rhat. Split-Rhat is defined for a single chain (the chain is split in half), so single-chain fits also get a value.

Value

A data frame with columns: parameter, rhat, ess_bulk, ess_tail.

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
diagnostics(fit)
```

draws_interval

Point Estimates and Credible Intervals for Selected Parameters

Description

Summarize the draws of the named parameters as a point estimate (median or mean) and a credible interval (quantile or highest-density), one row per parameter per interval width.

Usage

```
draws_interval(
  object,
  ...,
  .width = 0.95,
  .point = c("median", "mean"),
  .interval = c("qi", "hdi")
)

## S3 method for class 'ratiod_fit'
draws_interval(
  object,
  ...,
  .width = 0.95,
  .point = c("median", "mean"),
  .interval = c("qi", "hdi")
)
```

Arguments

<code>object</code>	A <code>ratiod_fit</code> object
<code>...</code>	Parameter names to summarize (unquoted).
<code>.width</code>	Width(s) of credible intervals. Default 0.95.
<code>.point</code>	Point estimate function: "median" (default) or "mean".
<code>.interval</code>	Interval type: "qi" for quantile interval (default), "hdi" for highest density interval.

Value

A data frame with columns for the variable name, point estimate, and interval bounds.

Examples

```
# Generate synthetic data
set.seed(444)
n <- 48
site <- rep(letters[1:12], each = 4)
u <- rnorm(12, sd = 0.5)[match(site, letters[1:12])]
depth <- rnorm(n)
df <- data.frame(
  count = rpois(n, exp(3 + 0.3 * depth + u)),
  effort = rgamma(n, shape = 9, rate = exp(-u)),
  depth = depth,
  site = site
)

# Fit model
fit <- tratio(
  count | effort ~ depth + (1 | site),
  data = df,
  family = ratiod_poisson_gamma(),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Get point estimates and 95% intervals
draws_interval(fit, beta_num, beta_denom)

# Multiple interval widths
draws_interval(fit, beta_num, .width = c(0.5, 0.9, 0.95))
```

draws_long

*Draws for Selected Parameters in Long Format***Description**

The long counterpart of `draws_wide()`: one row per draw per parameter, with the parameter in `.variable` and its value in `.value`.

Usage

```
draws_long(object, ..., regex = FALSE, ndraws = NULL)
```

```
## S3 method for class 'ratiod_fit'
```

```
draws_long(object, ..., regex = FALSE, ndraws = NULL)
```

Arguments

<code>object</code>	A <code>ratiod_fit</code> object
<code>...</code>	Parameter names to extract (unquoted).
<code>regex</code>	Logical; if TRUE, treat parameter names as regex patterns.
<code>ndraws</code>	Number of draws to return. If NULL, returns all.

Value

A data frame with columns `.chain`, `.iteration`, `.draw`, `.variable`, and `.value`.

Examples

```
# Generate synthetic data
set.seed(333)
n <- 48
site <- rep(letters[1:12], each = 4)
u <- rnorm(12, sd = 0.5)[match(site, letters[1:12])]
depth <- rnorm(n)
df <- data.frame(
  count = rpois(n, exp(3 + 0.3 * depth + u)),
  effort = rgamma(n, shape = 12, rate = exp(-u)),
  depth = depth,
  site = site
)

# Fit model
fit <- tratio(
  count | effort ~ depth + (1 | site),
  data = df,
  family = ratiod_poisson_gamma(),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
```

```

)

# Gather all fixed effects
draws_long(fit, beta_num, beta_denom)

# Use for plotting
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  draws_long(fit, beta_num) |>
    ggplot(aes(x = .value)) +
    geom_histogram(bins = 40) +
    facet_wrap(~ .variable)
}

```

draws_wide

*Draws for Selected Parameters in Wide Format***Description**

Extract posterior draws for the named parameters into a wide data frame, one row per draw and one column per parameter, alongside the `.chain`, `.iteration` and `.draw` bookkeeping columns.

Parameters are named unquoted, and a name matches every indexed element it prefixes, so `beta_num` reaches `beta_num[1]`, `beta_num[2]` and so on.

Usage

```

draws_wide(object, ..., regex = FALSE, ndraws = NULL)

## S3 method for class 'ratioid_fit'
draws_wide(object, ..., regex = FALSE, ndraws = NULL)

```

Arguments

<code>object</code>	A <code>ratioid_fit</code> object
<code>...</code>	Parameter names to extract (unquoted). Supports patterns like <code>beta_num</code> or <code>beta_num[i]</code> for indexed parameters.
<code>regex</code>	Logical; if <code>TRUE</code> , treat parameter names as regex patterns.
<code>ndraws</code>	Number of draws to return. If <code>NULL</code> , returns all.

Value

A data frame with columns `.chain`, `.iteration`, `.draw`, and one column per requested parameter.

Examples

```
# Generate synthetic data
set.seed(222)
n <- 48
site <- rep(letters[1:12], each = 4)
u <- rnorm(12, sd = 0.5)[match(site, letters[1:12])]
depth <- rnorm(n)
df <- data.frame(
  count = rpois(n, exp(3 + 0.3 * depth + u)),
  effort = rgamma(n, shape = 10, rate = exp(-u)),
  depth = depth,
  site = site
)

# Fit model
fit <- tratio(
  count | effort ~ depth + (1 | site),
  data = df,
  family = ratioid_poisson_gamma(),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Extract fixed effects
draws_wide(fit, beta_num, beta_denom)

# Subsample draws
draws_wide(fit, sigma_re, ndraws = 100)
```

fitted.ratioid_fit	<i>Fitted Values of a Ratio Model</i>
--------------------	---------------------------------------

Description

Compute posterior distributions of fitted values (expected values) for both the numerator and denominator processes from a fitted ratio model.

Usage

```
## S3 method for class 'ratioid_fit'
fitted(
  object,
  component = c("all", "numerator", "denominator", "ratio"),
  summary = TRUE,
  probs = c(0.025, 0.5, 0.975),
  ...
)
```

```
## S3 method for class 'ratioid_fitted'
print(x, n = 10, ...)
```

Arguments

<code>object</code>	A <code>ratioid_fit</code> object
<code>component</code>	Which component to return: "numerator", "denominator", "ratio", or "all" (default).
<code>summary</code>	Logical; if TRUE, return summary statistics instead of full posterior draws (default TRUE).
<code>probs</code>	Quantiles to compute if <code>summary = TRUE</code> .
<code>...</code>	Ignored
<code>x</code>	A <code>ratioid_fitted</code> object
<code>n</code>	Number of rows to print

Value

If `summary = TRUE`, a data frame with posterior summaries. If `summary = FALSE`, a list with matrices of posterior draws.

Examples

```
# Generate synthetic data
set.seed(789)
n <- 40
df <- data.frame(
  count = rpois(n, lambda = 15),
  effort = rgamma(n, shape = 8, rate = 1),
  depth = rnorm(n)
)

# Fit model
fit <- tratio(
  count | effort ~ depth,
  data = df,
  family = ratioid_poisson_gamma(),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Get fitted value summaries
fitted(fit)

# Get just numerator fitted values
fitted(fit, component = "numerator")

# Get full posterior draws
fitted(fit, summary = FALSE)
```

geweke_test

Geweke Convergence Test

Description

Performs Geweke's convergence diagnostic, comparing the mean of the first portion of a chain to the last portion. Useful for single-chain diagnostics.

Usage

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

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

Arguments

fit	A ratioid_fit object.
frac1	Fraction of chain for first window (default: 0.1).
frac2	Fraction of chain for second window (default: 0.5).
pars	Character vector of parameter names (default: all main params).
x	A ratioid_geweke object.
...	Ignored.

Details

The Geweke test computes a z-score comparing the means of early and late portions of a chain. Large z-scores ($|z| > 2$) indicate the chain has not converged.

Value

A data frame with columns: parameter, z_score, p_value.

See Also

[diagnostics\(\)](#), [check_diagnostics\(\)](#)

Examples

```
sim <- sim_ratioid(n = 100, family = ratioid_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratioid_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
geweke_test(fit)
```

 gpu-support

GPU Support Utilities

Description

Functions to check GPU availability and capabilities for accelerated GP computations.

Examples

```
gpu_available()
ratio_d_threads()
```

 gpu_available

Check Whether GPU Support Is Available

Description

Checks whether tulpaRatio was compiled with GPU support (CUDA or OpenCL) and whether compatible GPU hardware is detected.

Usage

```
gpu_available()
```

Details

GPU support requires:

1. tulpaRatio compiled with GPU support (-DUSE_CUDA or -DUSE_OPENCL)
2. Compatible GPU hardware (NVIDIA for CUDA, any GPU for OpenCL)
3. Appropriate drivers installed

The solver = "gpu" option in [spatial_gp\(\)](#) will automatically fall back to PCG if GPU support is unavailable.

Value

Logical; TRUE if GPU acceleration is available, FALSE otherwise.

See Also

[spatial_gp\(\)](#) for using GPU-accelerated GP models

Examples

```
# Check if GPU is available
gpu_available()

# Get detailed GPU info
if (gpu_available()) {
  gpu_info()
}
```

gpu_info

*GPU Device Information***Description**

Returns information about available GPU devices including name, memory, compute capability, and backend (CUDA or OpenCL).

Usage

```
gpu_info()

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

Arguments

x	GPU info object from <code>gpu_info()</code>
...	Ignored

Value

A list with GPU information, or NULL if no GPU is available.

available Logical; whether GPU is available

backend Character; "cuda", "opencl", or "none"

device_count Integer; number of GPU devices

devices List of device info (name, memory, compute capability)

Examples

```
info <- gpu_info()
if (!is.null(info) && info$available) {
  cat("GPU backend:", info$backend, "\n")
  cat("Devices:", info$device_count, "\n")
}
```

hurdle_binomial	<i>Hurdle Binomial Specification</i>
-----------------	--------------------------------------

Description

Hurdle Binomial Specification

Usage

```
hurdle_binomial(formula = NULL)
```

Arguments

formula	Optional formula for the hurdle probability ($P(Y > 0)$). If NULL, uses intercept-only model.
---------	---

Value

A `ratiod_zi` object

Examples

```
# Intercept-only hurdle
hurdle_binomial()
```

hurdle_negbin	<i>Hurdle Negative Binomial Specification</i>
---------------	---

Description

Hurdle Negative Binomial Specification

Usage

```
hurdle_negbin(formula = NULL)
```

Arguments

formula	Optional formula for the hurdle probability ($P(Y > 0)$). If NULL, uses intercept-only model.
---------	---

Value

A `ratiod_zi` object

Examples

```
# Intercept-only hurdle
hurdle_negbin()

# Hurdle probability depends on sampling effort
hurdle_negbin(~ log_effort)
```

hurdle_poisson	<i>Hurdle Poisson Specification</i>
----------------	-------------------------------------

Description

Hurdle Poisson Specification

Usage

```
hurdle_poisson(formula = NULL)
```

Arguments

formula	Optional formula for the hurdle probability ($P(Y > 0)$). If NULL, uses intercept-only model.
---------	---

Value

A `ratiod_zi` object

Examples

```
# Intercept-only hurdle
hurdle_poisson()

# Hurdle probability depends on effort
hurdle_poisson(~ effort)
```

inference_mode_info	<i>Inference Mode Information</i>
---------------------	-----------------------------------

Description

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

Usage

```
inference_mode_info()
```

Examples

```
inference_mode_info()
```

latent_factor	<i>Latent Factor Specification</i>
---------------	------------------------------------

Description

Define latent factors for capturing unmeasured structure shared between numerator and denominator. Each observation carries factor scores, and each process loads on them with its own loading, so the factors model the residual covariance of an observation's two linear predictors.

Usage

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

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

## S3 method for class 'ratiod_fit'
latent_factors(fit, summary = TRUE, probs = c(0.025, 0.5, 0.975), ...)
```

Arguments

n_factors	Integer; number of latent factors. Default is 1. With shared = TRUE at most 2 factors are identified, with shared = FALSE at most 1 (see Details).
prior	Prior for the diagonal loadings. Default is a PC prior with $P(\text{loading} > 1) = 0.01$, which shrinks toward simpler models.
shared	Logical; if TRUE (default), the factors load on both the numerator and the denominator linear predictor. If FALSE, they load on the numerator only.
constraint	Identifiability constraint for the factor scores: • "sum_to_zero" (default): each factor's scores sum to zero across observations • "first_zero": the first observation's scores are fixed at zero
x	A ratioid_latent object
...	Ignored
fit	A ratioid_fit sampled with latent = latent_factor(...).
summary	Logical; TRUE (default) returns one row per observation and factor with the posterior mean, sd and probs quantiles of its score, FALSE the draws matrix with columns factor[observation, factor].
probs	Quantiles for the summary.

Details**Why Use Latent Factors?:**

When modeling ratios, the numerator and denominator often share unmeasured drivers. For example, in relative abundance data, both the focal species count and total count might be affected by:

- Observer skill (unmeasured)
- Local microhabitat conditions (unmeasured)
- Weather on sampling day (unmeasured)

Latent factors capture this shared structure without requiring the drivers to be measured.

Mathematical Model:

For observation i with K latent factors:

$$\eta_i^{num} = X_i^{num} \beta^{num} + \sum_{k=1}^K \lambda_{1k} f_{ik} + \dots$$

$$\eta_i^{denom} = X_i^{denom} \beta^{denom} + \sum_{k=1}^K \lambda_{2k} f_{ik} + \dots$$

where:

- $f_{ik} \sim N(0, 1)$ are factor scores, with $\sum_i f_{ik} = 0$ for each k
- $\Lambda = (\lambda_{pk})$ is the loading matrix, lower triangular with a positive diagonal

The factors add the covariance $\Lambda\Lambda^\top$ between the two linear predictors of an observation. A 2 x 2 covariance has three free entries: one factor (two loadings) fills it up to rank one and two factors (three loadings) fill it completely, so a third factor would not be identified. Fixing Λ lower triangular with a positive diagonal makes it the Cholesky factor of that covariance, which is unique; without it the loadings and scores rotate freely.

The ratio's linear predictor gains $\sum_k (\lambda_{1k} - \lambda_{2k}) f_{ik}$: a factor that loads equally on both processes cancels in the ratio, one that loads unequally moves it.

Priors:

Each diagonal loading carries prior (a PC prior by default). Each off-diagonal loading carries a normal prior centred at zero, with the standard deviation that gives $|\lambda|$ the same 99% quantile as the diagonal prior.

Choosing n_factors:

- `n_factors = 1`: one unmeasured driver acting on both processes, with process-specific strength
- `n_factors = 2`: an unrestricted residual covariance between the two processes

Relationship to Random Effects:

Latent factors differ from random effects in several ways:

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

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

Value

A `ratiod_latent` object for use in `tratio()`.

See Also

`tratio()` for model fitting, `prior_pc()` for prior specification

Examples

```
# Basic latent factor (single factor on both processes)
latent_factor()

# Two latent factors: an unrestricted residual covariance
latent_factor(n_factors = 2)

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

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

# Use in model fitting
```

```

set.seed(1)
n <- 60
df <- data.frame(
  species_count = rnbino(n, size = 5, mu = 15),
  total_count = rnbino(n, size = 8, mu = 100),
  habitat = factor(rep(c("forest", "grassland"), each = n / 2)),
  site = factor(rep(1:10, each = n / 10))
)
fit <- tratio(
  species_count | total_count ~ habitat + (1 | site),
  data = df,
  family = ratiod_negbin_negbin(),
  latent = latent_factor(n_factors = 1),
  control = list(iter = 400, chains = 1, seed = 1)
)
head(latent_factors(fit))

```

loo.ratiod_fit

Leave-One-Out Cross-Validation

Description

Compute leave-one-out cross-validation using Pareto-smoothed importance sampling (PSIS-LOO).

Usage

```

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

```

Arguments

x	A ratiod_fit object
...	Additional arguments passed to loo::loo

Details

The pointwise log-likelihood is reconstructed from the posterior draws and the model (see [ratiod_compare\(\)](#)); it works on any fit produced by the hmc, ess, sghmc, or vi backend, for every family including the zero-inflated, one-inflated and hurdle ones. Each observation's likelihood is evaluated at the full linear predictor: fixed, random and structured effects.

Value

A loo object

Examples

```

sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
loo(fit)

```

n_divergent	<i>Number of Divergent Transitions</i>
-------------	--

Description

Number of Divergent Transitions

Usage

```
n_divergent(fit)
```

Arguments

fit A ratiod_fit object

Value

Integer count of divergent transitions

Examples

```

sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
n_divergent(fit)

```

oi_binomial	<i>One-Inflated Binomial Specification</i>
-------------	--

Description

One-Inflated Binomial Specification

Usage

```
oi_binomial(formula = NULL)
```

Arguments

formula	Optional formula for the one-inflation probability (excess at the upper boundary, $y = n$). If NULL, uses intercept-only model.
---------	--

Value

A `ratiod_zi` object

Examples

```
# Intercept-only OI
oi_binomial()
```

plot_acf	<i>Plot Autocorrelation Functions</i>
----------	---------------------------------------

Description

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

Usage

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

Arguments

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

Details

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

Value

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

See Also

`plot_ess()`, `diagnostics()`

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
plot_acf(fit)
plot_acf(fit, pars = c("beta_num[2]", "sigma_re"))
```

plot_divergences

Plot Divergent Transitions

Description

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

Usage

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

Arguments

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

Details

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

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

Value

A ggplot object or base R plot (invisible).

See Also

[plot_pairs\(\)](#), [n_divergent\(\)](#)

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
n_divergent(fit)
plot_divergences(fit)
```

plot_energy

Plot the Energy Diagnostic (E-BFMI)

Description

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

Usage

```
plot_energy(fit)
```

Arguments

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

Details

E-BFMI (Energy Bayesian Fraction of Missing Information) compares the distribution of energy levels to energy transitions. Values below 0.3 indicate the sampler may not be exploring the full posterior. The energy is the Hamiltonian after each iteration's momentum draw; it is centred and differenced within each chain, and the subtitle reports the lowest per-chain E-BFMI ([tulpa::energy_bfmi\(\)](#)).

Value

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

See Also

[diagnostic_summary\(\)](#), [check_diagnostics\(\)](#)

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
plot_energy(fit)
```

plot_ess

*Plot the Effective Sample Size Diagnostic***Description**

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

Usage

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

Arguments

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

Details

ESS/iter ratio interpretation:

- Green: ESS $\geq$ threshold
- Yellow: threshold/2 $\leq$ ESS < threshold
- Red: ESS < threshold/2

Value

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

See Also

[plot_rhat\(\)](#), [diagnostic_summary\(\)](#), [diagnostics\(\)](#)

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
plot_ess(fit)
```

plot_map

Plot Spatial Predictions as a Map

Description

Create publication-ready maps from tulpaRatio spatial predictions. Supports plotting ratio estimates, uncertainty (credible interval width), and individual process predictions (numerator/denominator).

Usage

```
plot_map(
  x,
  coords = NULL,
  what = c("ratio", "numerator", "denominator", "uncertainty"),
  summary = c("median", "mean", "q2.5", "q97.5", "sd"),
  newdata = NULL,
  title = NULL,
  palette = "viridis",
  points = FALSE,
  point_color = "grey30",
  point_size = 0.5,
  na_color = "transparent",
  crs = NULL,
  legend_title = NULL,
  ...
)
```

Arguments

x	A <code>ratiod_fit</code> object with spatial structure, or a data frame containing predictions with coordinate columns.
coords	A data frame or matrix with spatial coordinates (columns named <code>'x'/'y'</code> , <code>'X'/'Y'</code> , <code>'lon'/'lat'</code> , <code>'longitude'/'latitude'</code> , or <code>'Easting'/'Northing'</code>). Required if <code>x</code> is a data frame.
what	What to plot: "ratio" (default), "numerator", "denominator", or "uncertainty".
summary	Which summary statistic: "median" (default), "mean", "q2.5", "q97.5", or "sd".

<code>newdata</code>	Optional data frame with prediction locations and covariates. If NULL and <code>x</code> is a <code>ratiod_fit</code> , uses fitted values at observed locations.
<code>title</code>	Plot title. If NULL, auto-generated based on what.
<code>palette</code>	Color palette: "viridis" (default), "magma", "plasma", "inferno", "cividis", "mako", "rocket", or a custom vector of colors.
<code>points</code>	Logical; if TRUE, overlay observation points. Default FALSE.
<code>point_color</code>	Color for observation points. Default "grey30".
<code>point_size</code>	Size for observation points. Default 0.5.
<code>na_color</code>	Color for NA values. Default "transparent".
<code>crs</code>	Coordinate reference system (proj4 string or EPSG code). If NULL, uses planar coordinates.
<code>legend_title</code>	Title for the color legend. If NULL, auto-generated.
<code>...</code>	Additional arguments passed to ggplot2 theme functions.

Details

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

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

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

Value

A ggplot2 object that can be further customized.

Required packages

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

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

See Also

[ratio\(\)](#) for extracting ratio posteriors, [predict.ratiod_fit\(\)](#) for predictions at new locations

Examples

```
# Twenty sites on a chain, each visited three times
set.seed(123)
n_sites <- 20
sites <- data.frame(site = factor(seq_len(n_sites)),
                    x = sort(runif(n_sites)), y = runif(n_sites))
df <- sites[rep(seq_len(n_sites), 3), ]
df$count <- rpois(nrow(df), lambda = 8)
df$effort <- rgamma(nrow(df), shape = 4, rate = 1)
df$elevation <- rnorm(nrow(df))
adj <- matrix(0, n_sites, n_sites)
for (i in 1:(n_sites - 1)) adj[i, i + 1] <- adj[i + 1, i] <- 1
fit <- tratio(
  count | effort ~ elevation,
  data = df,
  family = ratioid_poisson_gamma(),
  spatial = spatial_car(adj, group_var = "site"),
  mode = "laplace"
)
if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot_map(fit, coords = df[, c("x", "y")])
}
```

plot_map_panel

*Plot Several Maps in a Grid***Description**

Create a multi-panel figure with ratio estimate and uncertainty maps side by side.

Usage

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

Arguments

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

Value

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

Examples

```
# Twenty sites on a chain, each visited three times
set.seed(123)
n_sites <- 20
sites <- data.frame(site = factor(seq_len(n_sites)),
                    x = sort(runif(n_sites)), y = runif(n_sites))
df <- sites[rep(seq_len(n_sites), 3), ]
df$count <- rpois(nrow(df), lambda = 8)
df$effort <- rgamma(nrow(df), shape = 4, rate = 1)
df$elevation <- rnorm(nrow(df))
adj <- matrix(0, n_sites, n_sites)
for (i in 1:(n_sites - 1)) adj[i, i + 1] <- adj[i + 1, i] <- 1
fit <- tratio(
  count | effort ~ elevation,
  data = df,
  family = ratioid_poisson_gamma(),
  spatial = spatial_car(adj, group_var = "site"),
  mode = "laplace"
)
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("patchwork", quietly = TRUE)) {
  plot_map_panel(fit, coords = df[, c("x", "y")], ncol = 1)
}
```

plot_pairs

Plot Bivariate Parameter Posteriors (Pairs Plot)

Description

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

Usage

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

Arguments

fit	A ratioid_fit object.
pars	Character vector of parameter names. If NULL, selects main variance parameters.

highlight_divergent	Logical; highlight divergent transitions in red (default: TRUE).
n_pars	Maximum number of parameters (default: 5).
alpha	Point transparency (default: 0.3).

Details

Pairs plots help identify:

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

Value

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

See Also

[plot_divergences\(\)](#), [diagnostics\(\)](#)

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
plot_pairs(fit, pars = c("beta_num[1]", "beta_num[2]", "sigma_re"))
```

plot_rhat

Plot the Rhat Convergence Diagnostic

Description

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

Usage

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

Arguments

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

Details

Color coding:

- Green: $R_{\text{hat}} < 1.01$ (converged)
- Yellow: $1.01 \leq R_{\text{hat}} < 1.05$ (borderline)
- Red: $R_{\text{hat}} \geq 1.05$ (not converged)

Value

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

See Also

[plot_ess\(\)](#), [diagnostic_summary\(\)](#), [diagnostics\(\)](#)

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
plot_rhat(fit)
```

posterior_predict.ratiod_fit

Posterior Predictive Draws

Description

Draw replicated numerator and denominator responses from the posterior predictive distribution: at each posterior draw the two fitted means are assembled from that draw's parameters, and a response is drawn from the family at those means. The replicates are what [pp_check\(\)](#) compares the observed data against.

Usage

```
## S3 method for class 'ratiod_fit'
posterior_predict(object, ndraws = NULL, ...)
```

Arguments

object	A ratiod_fit object
ndraws	Number of posterior draws to replicate. NULL (the default) uses every draw; a smaller number takes a random subsample.
...	Unused.

Value

A list with `y_num_rep` and `y_denom_rep`, each a draws-by-observation matrix, plus `y_num` and `y_denom`, the observed responses. A family with a fixed denominator (the binomial arms, whose denominator is the trial count) reports the trials as `y_denom_rep`.

See Also

[pp_check\(\)](#)

Examples

```
set.seed(4)
n <- 40
df <- data.frame(
  count = rpois(n, 12),
  total = rpois(n, 80),
  x = rnorm(n)
)
fit <- tratio(count | total ~ x, data = df,
  family = ratiod_poisson_gamma(),
  control = list(iter = 400, chains = 1, seed = 1))
rep <- posterior_predict(fit, ndraws = 50)
dim(rep$y_num_rep)
```

pp_check.ratiod_fit *Posterior Predictive Check*

Description

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

Usage

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

Arguments

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

Value

A ggplot object

Examples

```
# Simulate data and fit model
set.seed(123)
n <- 50
df <- data.frame(
  count = rnbino(n, size = 5, mu = 15),
  total = rnbino(n, size = 8, mu = 100),
  x = rnorm(n),
  site = factor(rep(1:10, each = 5))
)
fit <- tratio(
  count | total ~ x + (1 | site),
  data = df,
  family = ratiod_negbin_negbin(),
  control = list(iter = 400, chains = 1, seed = 1)
)
# Density overlay of replicated against observed numerators
pp_check(fit, type = "dens_overlay")
```

predict.ratiod_fit *Predictions from a Ratio Model*

Description

Compute posterior predictions for new data from a fitted ratio model.

Usage

```
## S3 method for class 'ratiod_fit'
predict(
  object,
  newdata,
```

```

    type = c("response", "link"),
    component = c("ratio", "numerator", "denominator", "all"),
    summary = TRUE,
    probs = c(0.025, 0.5, 0.975),
    re_formula = NULL,
    allow_new_levels = FALSE,
    coords.0 = NULL,
    include_spatial = TRUE,
    return_spatial = FALSE,
    n_samples = NULL,
    ...
)

## S3 method for class 'ratioid_prediction'
print(x, n = 10, ...)

```

Arguments

object	A ratioid_fit object
newdata	Data frame with new observations. Must contain all predictor variables used in the original model.
type	Type of prediction: "response" for fitted values on the response scale, "link" for linear predictor scale.
component	Which component to predict: "ratio" (default), "numerator", "denominator", or "all".
summary	Logical; if TRUE, return summary statistics instead of full posterior draws (default TRUE).
probs	Quantiles to compute if summary = TRUE.
re_formula	Formula for random effects in prediction. Use NULL to include all random effects (if new levels match), NA or ~ 0 to exclude random effects (population-level prediction).
allow_new_levels	Logical; if TRUE, allows new group levels not in the original data (predictions use population mean).
coords.0	Matrix of spatial coordinates for new prediction locations (required for GP/HSGP spatial models). Should be N x 2 matrix with same coordinate system as training data.
include_spatial	Logical; if TRUE (default), include spatial random effects in predictions. Set to FALSE for fixed-effects-only predictions.
return_spatial	Logical; if TRUE, return spatial effect samples at new locations as w.0. samples in the output (similar to spOccupancy).
n_samples	Number of posterior samples to draw for backends that don't store samples directly (e.g., Laplace, VI). Default uses all available samples for HMC/PG.
...	Ignored
x	A ratioid_prediction object
n	Number of rows to print

Value

If `summary = TRUE`, a data frame with posterior summaries. If `summary = FALSE`, a matrix of posterior draws (rows = draws, cols = obs). If `return_spatial = TRUE`, returns a list with predictions and `w.0.samples`.

Examples

```
# A smooth spatial field w shared by count and effort
set.seed(321)
n <- 60
x <- runif(n)
y <- runif(n)
w <- 0.6 * sin(2 * pi * x) + 0.6 * cos(2 * pi * y)
depth <- rnorm(n)
season <- factor(sample(c("summer", "winter"), n, replace = TRUE))
df <- data.frame(
  count = rpois(n, exp(2.5 + 0.3 * depth + 0.4 * (season == "winter") + w)),
  effort = rgamma(n, shape = 9, rate = exp(-w)),
  depth = depth,
  season = season,
  x = x,
  y = y
)
fit <- tratio(
  count | effort ~ depth + season,
  data = df,
  family = ratioid_poisson_gamma(),
  control = list(iter = 400, chains = 1, seed = 1)
)

# Predict for new data; levels of `season` follow the training data
new_df <- data.frame(depth = c(-1, 0, 1), season = "summer")
predict(fit, newdata = new_df)

# Full posterior draws of the predicted ratio (draws x rows)
dim(predict(fit, newdata = new_df, summary = FALSE)$ratio)

# Spatial prediction at new locations (kriging) from a GP fit
fit_gp <- tratio(
  count | effort ~ depth,
  data = df,
  spatial = spatial_gp(coords = c("x", "y")),
  family = ratioid_poisson_gamma(),
  control = list(iter = 400, chains = 1, seed = 1)
)
new_coords <- matrix(runif(10), ncol = 2)
predict(fit_gp, newdata = data.frame(depth = rnorm(5)),
  coords.0 = new_coords)
```

print.ratioid_sghmc	<i>Summarise, Print and Plot a Fitted Ratio Model</i>
---------------------	---

Description

print() gives a short description of a fit: how it was inferred, the family and the data. summary() returns the full report: posterior mean, sd and credible interval of every main parameter with its convergence diagnostics, which its print() method lays out. coef() returns the posterior means of the fixed effects and vcov() their posterior covariance; coef(summary(fit)) returns the summary table itself. nobs() and formula() return the number of observations and the model formula.

Usage

```
## S3 method for class 'ratioid_sghmc'
print(x, digits = 3, ...)

## S3 method for class 'ratioid_sgld'
print(x, digits = 3, ...)

## S3 method for class 'ratioid_vi'
print(x, digits = 3, ...)

## S3 method for class 'ratioid_vi'
summary(object, prob = 0.95, ...)

## S3 method for class 'summary.ratioid_vi'
print(x, digits = 3, ...)

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

## S3 method for class 'ratioid_fit'
summary(object, prob = 0.95, ...)

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

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

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

## S3 method for class 'ratioid_fit'
vcov(object, ...)

## S3 method for class 'ratioid_fit'
```

```
nobs(object, ...)

## S3 method for class 'ratiod_fit'
formula(x, ...)

## S3 method for class 'ratiod_fit'
plot(x, pars = NULL, type = c("trace", "dens", "both"), n_col = NULL, ...)
```

Arguments

<code>x</code>	A fit returned by <code>tratio()</code> : a <code>ratiod_fit</code> , or one of the backend subclasses <code>ratiod_vi</code> , <code>ratiod_sghmc</code> and <code>ratiod_sgld</code> , or the summary of one.
<code>digits</code>	Number of significant digits to print (default 3).
<code>...</code>	Ignored
<code>object</code>	A <code>ratiod_fit</code> object
<code>prob</code>	Probability mass for credible intervals (default 0.95)
<code>pars</code>	Character vector of parameter names to plot. If <code>NULL</code> (default), plots the first 12 main parameters (fixed effects, variance components).
<code>type</code>	Type of plot: "trace" (default), "dens" (density), or "both".
<code>n_col</code>	Number of columns in the plot grid (default: auto).

Value

An object of class `summary.ratiod_vi`.

The input `x`, invisibly.

`summary()`: a `summary.ratiod_fit` object. `coef()`: a named vector, numerator coefficients prefixed `num:` and denominator ones `denom:`. `vcov()`: the matching covariance matrix.

Invisibly returns the plot object (base graphics or `ggplot2`).

Examples

```
# Create simple dataset
set.seed(456)
n <- 48
site <- factor(rep(1:6, each = 8))
u <- rnorm(6, sd = 0.5)[site]
depth <- rnorm(n)
df <- data.frame(
  count = rpois(n, exp(2 + 0.3 * depth + u)),
  effort = rgamma(n, shape = 2, rate = 0.5 * exp(-u)),
  depth = depth,
  site = site
)

fit <- tratio(
  count | effort ~ depth + (1|site),
```

```
data = df,
family = ratiod_poisson_gamma(),
control = list(iter = 400, chains = 2, seed = 1)
)
plot(fit)
plot(fit, pars = "beta_num", type = "both")
```

prior_beta	<i>Beta Prior</i>
------------	-------------------

Description

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

Usage

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

Arguments

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

Details

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

Value

A ratiod_prior object

Examples

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

prior_exponential	<i>Exponential Prior</i>
-------------------	--------------------------

Description

Specify an exponential prior for a positive parameter.

Usage

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

Arguments

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

Value

A `ratiod_prior` object

Examples

```
prior_exponential(1)
```

prior_gamma	<i>Gamma Prior</i>
-------------	--------------------

Description

Specify a gamma prior for a positive parameter.

Usage

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

Arguments

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

Details

Mean = shape/rate, Variance = shape/rate².

Value

A `ratiod_prior` object

Examples

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

prior_half_cauchy	<i>Half-Cauchy Prior</i>
-------------------	--------------------------

Description

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

Usage

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

Arguments

scale	Scale parameter. Must be positive.
-------	------------------------------------

Value

A `ratiod_prior` object

Examples

```
prior_half_cauchy(2.5)
```

prior_half_normal	<i>Half-Normal Prior</i>
-------------------	--------------------------

Description

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

Usage

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

Arguments

sd	Scale parameter. Must be positive.
----	------------------------------------

Value

A ratioid_prior object

Examples

```
prior_half_normal(1)
```

prior_normal	<i>Normal Prior</i>
--------------	---------------------

Description

Specify a normal prior for a parameter.

Usage

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

Arguments

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

Value

A ratioid_prior object

Examples

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

prior_pc	<i>Penalized Complexity (PC) Prior</i>
----------	--

Description

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

Usage

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

Arguments

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

Details

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

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

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

Value

A `ratiod_prior` object

References

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

Examples

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

prior_predict	<i>Prior Predictive Simulation</i>
---------------	------------------------------------

Description

Simulate numerator and denominator responses from the prior predictive distribution: parameters are drawn from priors, the two linear predictors are assembled on the design formula and data imply, and a response is drawn from family at each draw. Comparing the resulting ratios against what is plausible for the system under study is the check on whether the priors are reasonable before any data are fitted.

Random-effect terms contribute at a standard deviation drawn from `priors$sigma`, one per intercept and one per slope column. Correlation between slopes within a term is not drawn, so a correlated term simulates at its marginal scales.

Usage

```
prior_predict(formula, family, data, priors = NULL, n = 100)

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

Arguments

formula	A model formula, using the combined <code>num denom ~ x</code> syntax or a single response with <code>formula_denom</code> supplied through <code>ratiod_formula()</code> .
family	A <code>ratiod_family</code> object.
data	Data frame supplying the predictors, the grouping factors and, for the binomial families, the trial counts. Response columns need not be present; where one is missing a placeholder is used, since nothing about the prior predictive distribution depends on the observed response.
priors	A <code>ratiod_priors</code> object. Defaults to <code>ratiod_priors()</code> .
n	Number of prior predictive draws.
x	A <code>ratiod_priorpred</code> object
...	Ignored

Value

An object of class `ratiod_priorpred`: a list with the $n \times N$ matrices `y_num`, `y_denom`, `ratio`, `mu_num` and `mu_denom`, the drawn coefficients `beta_num` and `beta_denom`, the drawn random-effect scales `sigma`, the drawn dispersions `phi_num` and `phi_denom`, and the family and data the draws were taken on.

`x`, invisibly

See Also

`sim_ratiod()` for simulating from a known parameterisation, `ratiod_priors()` for the prior specification

Examples

```
set.seed(1)
df <- data.frame(
  count = rep(0L, 40),
  total = rep(0L, 40),
  x = rnorm(40),
  site = factor(rep(1:8, each = 5))
)

pp <- prior_predict(
  count | total ~ x + (1 | site),
  family = ratiod_negbin_negbin(),
  data = df,
  n = 50
)
dim(pp$ratio)
summary(as.vector(pp$ratio))

# A tighter coefficient prior narrows the predicted ratios
pp_tight <- prior_predict(
  count | total ~ x + (1 | site),
```

```

family = ratiod_negbin_negbin(),
data = df,
priors = ratiod_priors(beta = prior_normal(0, 0.5)),
n = 50
)

```

priors_default

Default Priors for a Model Family

Description

Display the priors a fit uses for every component [ratiod_priors\(\)](#) leaves unspecified, as a starting point for customization.

Usage

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

Arguments

family	A tulpaRatio family object (e.g., ratiod_negbin_negbin()). If NULL (default), shows the dispersion default of every family.
spatial	Logical; if TRUE, include the BYM2 mixing prior.
temporal	Logical; if TRUE, include the AR1 correlation prior.

Details

- **Fixed effects (beta):** Normal(0, 10) on the link scale.
- **Random effect SD (sigma):** Half-Cauchy(2.5); the Laplace backend uses a PC prior with $P(\text{sigma} > 1) = 0.01$.
- **Dispersion (phi):** Gamma(2, 2) for the lognormal log-scale SD, Gamma(2, 0.5) for a Gamma shape (gamma_gamma, poisson_gamma), Gamma(2, 0.1) for a negative binomial size or beta-binomial precision.
- **Temporal correlation (rho):** Beta(2, 2) on $(\text{rho} + 1) / 2$, symmetric about $\text{rho} = 0$.
- **Spatial mixing (rho_spatial):** Beta(1, 1), uniform on (0, 1).

Value

Invisibly, a `ratiod_priors` object holding the defaults (for `family = NULL`, with the negative binomial dispersion default).

See Also

[ratiod_priors\(\)](#) for creating custom priors

Examples

```
priors_default(ratiod_negbin_negbin())
priors_default(ratiod_poisson_gamma(), spatial = TRUE, temporal = TRUE)
priors_default()
```

ratio

Ratio Posteriors from a Ratio Model

Description

Compute posterior distributions of ratios from a fitted ratio model. Ratios are derived quantities: $E[\text{numerator}] / E[\text{denominator}]$, computed from each posterior draw, so they carry the full posterior uncertainty.

Usage

```
ratio(
  object,
  newdata = NULL,
  type = c("response", "log", "logit"),
  by = NULL,
  summary = FALSE,
  probs = c(0.025, 0.5, 0.975),
  ...
)

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

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

## S3 method for class 'ratiod_ratio_summary'
print(x, n = 10, ...)
```

Arguments

object	A <code>ratiod_fit</code> object
newdata	Optional data frame for prediction. If <code>NULL</code> , uses original data.
type	Scale for ratio: "response" (default), "log", or "logit".
by	Optional grouping variable name to aggregate ratios by group.
summary	Logical; if <code>TRUE</code> , return summary statistics instead of full posterior draws.
probs	Quantiles to compute if <code>summary = TRUE</code> .
...	Ignored
x	A <code>ratiod_ratio</code> object, or the <code>ratiod_ratio_summary</code> its <code>summary()</code> returns.
n	Number of rows to print

Value

A `ratiod_ratio` object containing posterior draws or summaries.

Examples

```
# Create a simple family
fam <- ratiod_poisson_gamma()

# Generate synthetic data: a site effect shared by count and effort
set.seed(123)
n <- 60
site <- rep(letters[1:6], each = 10)
u <- rnorm(6, sd = 0.5)[match(site, letters[1:6])]
depth <- rnorm(n)
df <- data.frame(
  count = rpois(n, exp(3 + 0.3 * depth + u)),
  effort = rgamma(n, shape = 10, rate = exp(-u)),
  depth = depth,
  site = site
)

# Fit model
fit <- tratio(
  count | effort ~ depth + (1 | site),
  data = df,
  family = ratiod_poisson_gamma(),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Extract all observation-level ratios
r <- ratio(fit)
summary(r)

# Log-ratio scale
r_log <- ratio(fit, type = "log")

# Group-level ratios
r_site <- ratio(fit, by = "site")
```

ratio_contrast

Contrasts Between Ratio Conditions

Description

Compute posterior distributions of contrasts (differences or ratios) between conditions for the derived ratio quantity.

Usage

```
ratio_contrast(
  object,
  contrast,
  type = c("difference", "ratio"),
  ref = NULL,
  ...
)

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

Arguments

object	A ratiod_fit object
contrast	A formula specifying the contrast, e.g., ~ treatment
type	Type of contrast: "difference" (default) or "ratio"
ref	Reference level for factor contrasts (default: first level)
...	Ignored
x	A ratiod_contrast object

Value

A data frame with posterior summaries of contrasts

Examples

```
# Generate synthetic data with grouping variable
set.seed(456)
n <- 60
df <- data.frame(
  count = rpois(n, lambda = 25),
  effort = rgamma(n, shape = 12, rate = 1),
  season = sample(c("summer", "winter"), n, replace = TRUE)
)

# Fit model
fit <- tratio(
  count | effort ~ season,
  data = df,
  family = ratiod_poisson_gamma(),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Compare CPUE between seasons
ratio_contrast(fit, ~ season)

# Ratio of ratios (multiplicative comparison)
```

```
ratio_contrast(fit, ~ season, type = "ratio")
```

ratiod_average

Model Averaging for Ratio Models

Description

Compute model-averaged predictions using stacking or pseudo-BMA weights. Model weights are derived from LOO-CV or WAIC, and predictions are combined accordingly.

Usage

```
ratiod_average(
  ...,
  weights = c("loo", "waic", "pbma", "pbma+"),
  newdata = NULL,
  type = c("ratio", "numerator", "denominator"),
  summary = TRUE
)

## S3 method for class 'ratiod_average'
print(x, digits = 3, ...)

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

## S3 method for class 'ratiod_average'
weights(object, ...)
```

Arguments

...	Multiple <code>ratiod_fit</code> objects to average
weights	Method for computing weights: "loo" (stacking, default), "waic", "pbma" (pseudo-BMA), or "pbma+" (pseudo-BMA+ with Bayesian bootstrap).
newdata	Optional new data for predictions. If <code>NULL</code> , uses fitted values.
type	Type of prediction: "ratio" (default), "numerator", or "denominator".
summary	Logical; if <code>TRUE</code> (default), return summary statistics. If <code>FALSE</code> , return full posterior draws.
x	A <code>ratiod_average</code> object
digits	Number of digits for printing
object	A <code>ratiod_average</code> object

Details

Model averaging accounts for model uncertainty by combining predictions from multiple candidate models. The weighting methods are:

- **stacking** (weights = "loo"): Optimal linear combination minimizing leave-one-out prediction error. Recommended default.
- **pseudo-BMA** (weights = "pbma"): Weights proportional to $\exp(-\text{ELPD})$. Can be unstable with similar models.
- **pseudo-BMA+** (weights = "pbma+"): Bayesian bootstrap variant that accounts for estimation uncertainty.

For ratio predictions, averaging is performed on the log scale to respect the multiplicative nature of ratios.

Value

A `ratiod_average` object containing:

- `weights`: Model weights
- `predictions`: Model-averaged predictions
- `models`: List of model names

Predictions data frame or matrix

Named numeric vector of weights

See Also

`ratiod_compare()` for model comparison, `loo::stacking_weights()`

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
fit0 <- tratio(y_num | y_denom ~ 1 + (1 | group), data = sim$data,
              family = ratiod_poisson_gamma(),
              control = list(iter = 1000, chains = 2, seed = 1))
avg <- ratiod_average(fit0, fit)
avg
weights(avg)
```

ratiod_benchmark	<i>Benchmark Sampler Performance</i>
------------------	--------------------------------------

Description

Utilities for benchmarking ratio model fitting performance across different dataset sizes and configurations.

Fits a model on simulated data and reports timing and performance metrics.

Usage

```
ratiod_benchmark(
  N = 500,
  p = 3,
  n_groups = 0,
  family = ratiod_negbin_negbin(),
  n_iter = 1000,
  n_warmup = 500,
  n_chains = 1,
  n_threads = 1,
  verbose = FALSE
)

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

Arguments

N	Number of observations
p	Number of predictors
n_groups	Number of random effect groups (0 for none)
family	Model family (default: ratiod_negbin_negbin())
n_iter	Total iterations
n_warmup	Warmup iterations
n_chains	Number of chains
n_threads	Number of threads per chain
verbose	Print progress (default: FALSE)
x	A ratiod_benchmark object
...	Ignored

Value

A list with timing and diagnostic information

Examples

```
# Quick benchmark
bench <- ratiobenchmark(N = 100, p = 3, n_iter = 200, n_warmup = 100)
print(bench)
```

ratiobenchmark_compare

Compare Benchmarks Across Configurations

Description

Run benchmarks across multiple configurations to compare performance.

Usage

```
ratiobenchmark_compare(configs, n_iter = 500, n_warmup = 250, n_threads = 1)
```

Arguments

configs	A list of configuration lists, each with N, p, n_groups
n_iter	Iterations per benchmark
n_warmup	Warmup iterations
n_threads	Number of threads

Value

A data frame with benchmark results

Examples

```
# Compare benchmarks
configs <- list(
  list(N = 50, p = 2, n_groups = 0),
  list(N = 100, p = 2, n_groups = 0)
)
comparison <- ratiobenchmark_compare(configs, n_iter = 200, n_warmup = 100)
print(comparison)
```

ratioid_beta_binomial	<i>Beta-Binomial Family for Overdispersed Proportions</i>
-----------------------	---

Description

Two-process model where the numerator follows a beta-binomial distribution, allowing for overdispersion in binomial data. The denominator is the known number of trials.

Use cases:

- Overdispersed proportions (e.g., infection rates across sites)
- Hierarchical binomial with extra-binomial variation
- Proportions with correlation within clusters

Usage

```
ratioid_beta_binomial(link = "logit")
```

Arguments

link	Link function for the success probability: "logit" (default), "probit" or "cloglog".
------	--

Details

The beta-binomial distribution arises when the success probability itself follows a beta distribution. This creates overdispersion (variance > binomial variance) and is useful when:

- There is unmodelled heterogeneity in success probability
- Observations within clusters are correlated
- The binomial assumption of independent trials is violated

The overdispersion parameter ρ (intra-class correlation) ranges from 0 (binomial) to 1 (maximum overdispersion).

Value

A `ratioid_family` object

Examples

```
# Create family object
fam <- ratioid_beta_binomial()
print(fam)

# Simulate overdispersed proportion data
set.seed(123)
n <- 40
df <- data.frame(
  infected = rbinom(n, size = 20, prob = rbeta(n, 2, 5)),
```

```

  tested = rep(20L, n),
  treatment = factor(rep(c("control", "treated"), each = n/2)),
  site = factor(rep(1:8, each = n/8))
)

# Fit model
fit <- tratio(
  infected | tested ~ treatment + (1 | site),
  data = df,
  family = ratiod_beta_binomial(),
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)

```

ratiod_binomial

Binomial Family for Trial-Based Ratios

Description

Trial-based model where the numerator represents successes out of a known number of trials, the denominator.

Use cases:

- Detection / availability (occupancy-style)
- Successes / trials (hierarchical binomial)
- Proportions with known denominators

Usage

```
ratiod_binomial(link = "logit")
```

Arguments

link Link function for the success probability: "logit" (default), "probit" or "cloglog".

Value

A ratiod_family object

Examples

```
# Create family object
fam <- ratiod_binomial()
print(fam)

# Simulate detection/availability data
set.seed(123)
n <- 50
df <- data.frame(
  detections = rbinom(n, size = 10, prob = 0.4),
  trials = rep(10L, n),
  effort = rnorm(n),
  site = factor(rep(1:10, each = 5))
)

# Fit model
fit <- tratio(
  detections | trials ~ effort + (1 | site),
  data = df,
  family = ratiod_binomial(),
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)
```

ratiod_compare

Compare Ratio Models

Description

Compare multiple ratio models using LOO-CV or WAIC.

Usage

```
ratiod_compare(..., criterion = c("loo", "waic"))
```

Arguments

...	Multiple <code>ratiod_fit</code> objects to compare
criterion	Comparison criterion: "loo" (default) or "waic"

Value

A `loo_compare` object

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
fit0 <- tratio(y_num | y_denom ~ 1 + (1 | group), data = sim$data,
              family = ratiod_poisson_gamma(),
              control = list(iter = 1000, chains = 2, seed = 1))
ratiod_compare(fit0, fit)
```

ratiod_families	<i>Model Families</i>
-----------------	-----------------------

Description

Distribution families for the numerator and denominator processes. Each family specifies the likelihood structure for joint modelling.

Usage

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

Arguments

x	A ratiod_family object
...	Ignored

Examples

```
ratiod_poisson_gamma()
ratiod_binomial()
```

ratiod_gamma_gamma	<i>Gamma-Gamma Family for Continuous Ratio Data</i>
--------------------	---

Description

Two-process model where both numerator and denominator follow gamma distributions. This is the natural family for ratios of positive continuous quantities.

Use cases:

- Biomass ratios (species biomass / total biomass)
- Concentration ratios (analyte / reference)
- Duration ratios (time in activity / total time)

Usage

```
ratioid_gamma_gamma(link_num = "log", link_denom = "log")
```

Arguments

link_num	Link function for the numerator mean: "log" (default), "identity" or "inverse".
link_denom	Link function for the denominator mean: "log" (default), "identity" or "inverse".

Details

The gamma distribution is appropriate for positive continuous data with right skew. The ratio of two gamma random variables does not have a simple closed form, which is why ratio models them jointly.

Shape parameters are estimated for both numerator and denominator, allowing different amounts of variability in each process.

Value

A `ratioid_family` object

Examples

```
# Create family object
fam <- ratioid_gamma_gamma()
print(fam)

# Simulate biomass ratio data
set.seed(123)
n <- 50
df <- data.frame(
  species_biomass = rgamma(n, shape = 3, rate = 0.5),
  total_biomass = rgamma(n, shape = 10, rate = 0.3),
  habitat = factor(rep(c("forest", "grassland"), each = n/2)),
  site = factor(rep(1:10, each = n/10))
)

# Fit model
fit <- tratio(
  species_biomass | total_biomass ~ habitat + (1 | site),
  data = df,
  family = ratioid_gamma_gamma(),
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)
```

ratiod_hurdle_binomial

Hurdle Binomial Family

Description

Hurdle model for binomial data where zero counts are modeled separately from positive counts using a truncated binomial.

Usage

```
ratiod_hurdle_binomial(link_num = "logit", link_hurdle = "logit")
```

Arguments

link_num	Link function for the success probability: "logit" (default), "probit" or "cloglog".
link_hurdle	Link function for the hurdle probability $P(Y > 0)$: "logit" (default), "probit" or "cloglog".

Value

A ratiod_family object with hurdle structure

Examples

```
fam <- ratiod_hurdle_binomial()
print(fam)
```

ratiod_hurdle_negbin *Hurdle Negative Binomial Family*

Description

Hurdle model where zero vs positive counts are modelled as separate processes.

$$P(Y = 0) = 1 - \theta$$

$$P(Y = y | Y > 0) = P_{NB}^+(y), \quad y > 0$$

where θ is the probability of a positive count and P_{NB}^+ is the truncated-at-zero negative binomial.

Usage

```
ratiod_hurdle_negbin(
  link_num = "log",
  link_denom = "log",
  link_hurdle = "logit",
  denom_family = "negbin"
)
```

Arguments

<code>link_num</code>	Link function for the count mean: "log".
<code>link_denom</code>	Link function for the denominator mean: "log" (default); a gamma denominator also takes "identity" or "inverse".
<code>link_hurdle</code>	Link function for the hurdle probability: "logit" (default), "probit" or "cloglog".
<code>denom_family</code>	Denominator distribution: "negbin" (default), "gamma" for continuous effort, or "fixed" for a known exposure. A known exposure is data: it scales the count mean, which is then the rate times the exposure, and the ratio is the rate.

Details

The hurdle model differs from zero-inflation in interpretation:

Zero-inflated: Zeros come from two sources (structural + sampling)

- π : probability of structural zero
- Some zeros also come from the count process

Hurdle: Zeros are all from one source (the "hurdle" process)

- $1 - \theta$: probability of zero
- All positive counts from truncated count process

Choose hurdle when zeros represent a distinct biological/sampling state.

Value

A `ratiod_family` object with hurdle structure

Examples

```
# Create family object
fam <- ratiod_hurdle_negbin()
print(fam)

# Simulate hurdle data (presence/absence + abundance)
set.seed(123)
n <- 60
presence_prob <- 0.7
df <- data.frame(
  count = ifelse(runif(n) > presence_prob, 0,
```

```

      rnbinom(n, size = 3, mu = 10) + 1),
total = rnbinom(n, size = 5, mu = 50),
habitat = factor(rep(c("forest", "grassland"), each = n/2)),
site = factor(rep(1:10, each = n/10))
)

# Fit model
fit <- tratio(
  count | total ~ habitat + (1 | site),
  data = df,
  family = ratioid_hurdle_negbin(),
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)

```

ratioid_hurdle_pois	<i>Hurdle Poisson Family</i>
---------------------	------------------------------

Description

Hurdle model with truncated Poisson for positive counts.

Usage

```

ratioid_hurdle_pois(
  link_num = "log",
  link_denom = "log",
  link_hurdle = "logit",
  denom_family = "gamma"
)

```

Arguments

link_num	Link function for the count mean: "log".
link_denom	Link function for the denominator mean: "log" (default); a gamma denominator also takes "identity" or "inverse".
link_hurdle	Link function for the hurdle probability: "logit" (default), "probit" or "cloglog".
denom_family	Denominator distribution: "gamma" (default), "negbin" for a count denominator, or "fixed" for a known exposure.

Value

A ratioid_family object with hurdle structure

Examples

```
ratiod_hurdle_pois()
```

ratiod_latent	<i>Latent Factors for Unmeasured Confounders</i>
---------------	--

Description

Specify latent factors to capture shared unmeasured confounders between numerator and denominator processes. Latent factors are particularly useful when you suspect that both processes are driven by common unmeasured variables.

Examples

```
latent_factor(n_factors = 2)
```

ratiod_lognormal	<i>Log-Normal Family for Multiplicative Processes</i>
------------------	---

Description

Two-process model where both numerator and denominator follow log-normal distributions. Appropriate when the underlying process is multiplicative and residuals are log-normally distributed.

Use cases:

- Abundance indices (CPUE with continuous effort)
- Economic ratios (costs, prices)
- Body condition indices

Usage

```
ratiod_lognormal(link_num = "log", link_denom = "log", denom_fixed = FALSE)
```

Arguments

link_num	Link function for the numerator median: "log".
link_denom	Link function for the denominator median: "log".
denom_fixed	If TRUE, the denominator is a known exposure rather than a modelled response: the numerator's median is the exposure times $\exp(\eta)$, and the ratio is $\exp(\eta)$.

Details

The log-normal distribution arises when a variable is the product of many independent positive factors. If $Y = e^X$ where X is normal, then Y is log-normal.

For ratios, if both numerator and denominator are log-normal, the ratio is also log-normal (difference of normals on log scale). This can simplify interpretation when working on the log scale.

Value

A `ratiod_family` object

Examples

```
# Create family object
fam <- ratiod_lognormal()
print(fam)

# Simulate body condition data
set.seed(123)
n <- 60
df <- data.frame(
  weight = rlnorm(n, meanlog = 3, sdlog = 0.3),
  length_cubed = rlnorm(n, meanlog = 2, sdlog = 0.2),
  age = sample(1:5, n, replace = TRUE),
  sex = factor(rep(c("M", "F"), each = n/2)),
  cohort = factor(rep(1:6, each = n/6))
)

# Fit model
fit <- tratio(
  weight | length_cubed ~ age + sex + (1 | cohort),
  data = df,
  family = ratiod_lognormal(),
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)
```

ratiod_negbin_gamma	<i>Negative Binomial-Gamma Family for Overdispersed Count/Effort Ratios</i>
---------------------	---

Description

Two-process model where the numerator follows a negative binomial distribution and the denominator (effort/exposure) follows a Gamma distribution. This fills the gap between Poisson-Gamma (equidispersed counts) and NegBin-NegBin (integer denominator): use it when counts are overdispersed and effort is continuous.

Use cases:

- Overdispersed catch per unit effort (CPUE)
- Overdispersed observations per continuous exposure
- Any ratio of overdispersed counts over continuous positive effort

Usage

```
ratioid_negbin_gamma(link_num = "log", link_denom = "log")
```

Arguments

link_num	Link function for the count rate: "log".
link_denom	Link function for the effort mean: "log" (default), "identity" or "inverse".

Value

A `ratioid_family` object

Examples

```
# Create family object
fam <- ratioid_negbin_gamma()
print(fam)

# Simulate overdispersed CPUE data
set.seed(123)
n <- 60
df <- data.frame(
  catch = rnbinom(n, size = 5, mu = 8),
  effort_hours = rgamma(n, shape = 4, rate = 1),
  depth = rnorm(n),
  vessel = factor(rep(1:6, each = n/6))
)

# Fit model
fit <- tratio(
  catch | effort_hours ~ depth + (1 | vessel),
  data = df,
  family = ratioid_negbin_gamma(),
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)
```

ratiod_negbin_negbin *Negative Binomial-Negative Binomial Family*

Description

Two-process model where both numerator and denominator follow negative binomial distributions. This is the core family for count-count ratios where both processes exhibit overdispersion.

Use cases:

- Species counts / total counts (relative abundance)
- Events / opportunities (both counts)
- Any ratio where numerator and denominator are overdispersed counts

Usage

```
ratiod_negbin_negbin(link_num = "log", link_denom = "log")
```

Arguments

link_num	Link function for the numerator mean: "log".
link_denom	Link function for the denominator mean: "log".

Value

A `ratiod_family` object

Examples

```
# Create family object
fam <- ratiod_negbin_negbin()
print(fam)

# Simulate data for relative abundance
set.seed(123)
n <- 60
df <- data.frame(
  sp_count = rnbino(n, size = 5, mu = 15),
  total_count = rnbino(n, size = 8, mu = 100),
  habitat = factor(rep(c("forest", "grassland"), each = n/2)),
  site = factor(rep(1:10, each = n/10))
)

# Fit model
fit <- tratio(
  sp_count | total_count ~ habitat + (1 | site),
  data = df,
  family = ratiod_negbin_negbin(),
```

```
control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)
```

ratiod_oibinomial	<i>One-Inflated Binomial Family</i>
-------------------	-------------------------------------

Description

One-inflated binomial for proportions with excess ones (100% success). Useful when perfect detection/success has a structural component.

$$P(Y = n) = \psi + (1 - \psi) \cdot p^n$$

$$P(Y = y) = (1 - \psi) \cdot \binom{n}{y} p^y (1 - p)^{n-y}, \quad y < n$$

Usage

```
ratiod_oibinomial(link_num = "logit", link_oi = "logit")
```

Arguments

link_num	Link function for the success probability: "logit" (default), "probit" or "cloglog".
link_oi	Link function for the one-inflation probability: "logit" (default), "probit" or "cloglog".

Details

Use this when excess 100% success rates occur due to:

- Perfect detection in highly suitable habitat
- Saturation effects
- Structural constraints ensuring full success

Value

A `ratiod_family` object with one-inflation

Examples

```
# Create family object
fam <- ratiod_oibinomial()
print(fam)
```

ratioid_poisson_gamma	<i>Poisson-Gamma Family for Count/Effort Ratios</i>
-----------------------	---

Description

Two-process model where the numerator follows a Poisson distribution and the denominator (effort/exposure) follows a Gamma distribution. This is the natural family for CPUE-type data.

Use cases:

- Catch per unit effort (CPUE)
- Observations per hour
- Events per continuous exposure

Usage

```
ratioid_poisson_gamma(link_num = "log", link_denom = "log")
```

Arguments

link_num	Link function for the count rate: "log".
link_denom	Link function for the effort mean: "log" (default), "identity" or "inverse".

Value

A ratioid_family object

Examples

```
# Create family object
fam <- ratioid_poisson_gamma()
print(fam)

# Simulate CPUE data
set.seed(123)
n <- 60
season <- factor(rep(c("spring", "summer", "fall"), each = n/3))
vessel <- factor(rep(1:10, each = n/10))
u <- rnorm(10, sd = 0.5)[vessel]
depth <- rnorm(n)
df <- data.frame(
  catch = rpois(n, exp(1.5 + 0.3 * depth + 0.4 * (season == "summer") + u)),
  effort_hours = rgamma(n, shape = 4, rate = exp(-u)),
  depth = depth,
  season = season,
  vessel = vessel
)
```

```
# Fit model
fit <- tratio(
  catch | effort_hours ~ depth + season + (1 | vessel),
  data = df,
  family = ratiod_poisson_gamma(),
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)
```

ratiod_priors	<i>Prior Specification for Ratio Models</i>
---------------	---

Description

Specify priors for model parameters. Supports both PC (penalized complexity) priors for variance components and standard distributions for other parameters.

Usage

```
ratiod_priors(
  beta = NULL,
  sigma = NULL,
  phi = NULL,
  rho_temporal = NULL,
  rho_spatial = NULL
)

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

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

Arguments

beta	Prior for fixed effects. Default: <code>prior_normal(0, 10)</code> .
sigma	Prior for random effect SDs. Default: <code>prior_half_cauchy(2.5)</code> (the Laplace backend: <code>prior_pc(U = 1, alpha = 0.01)</code>).
phi	Prior for the dispersion parameter. Default: a Gamma prior set by family: <code>prior_gamma(2, 2)</code> for lognormal, <code>prior_gamma(2, 0.5)</code> for gamma_gamma and <code>poisson_gamma</code> , <code>prior_gamma(2, 0.1)</code> otherwise.
rho_temporal	Prior for a temporal autocorrelation, placed on $(\rho + 1) / 2$ so ρ lives on $(-1, 1)$. Default: <code>prior_beta(2, 2)</code> , symmetric about $\rho = 0$. Applies to <code>temporal_ar1()</code> , a <code>spatiotemporal()</code> interaction's AR1 time margin, the TVC arm and the multi-scale short-term arm; each block's own <code>rho_prior</code> overrides it.

<code>rho_spatial</code>	Prior for spatial proportion (BYM2). Default: <code>prior_beta(1, 1)</code> (uniform).
<code>x</code>	A <code>ratiod_priors</code> object from <code>ratiod_priors()</code> , or a single <code>ratiod_prior</code> from a <code>prior_*</code> () constructor.
<code>...</code>	Ignored

Details

The MCMC samplers (`mode = "hmc", "ess", "pg", "gibbs", "vi", "sghmc", "sgld"`) evaluate a Normal prior with mean 0 on fixed effects, a Half-Cauchy prior on random-effect SDs and a Gamma prior on the dispersion parameter; a component specified with another family is refused when the model is fitted. The Laplace backend takes `prior_pc()` or `prior_half_normal()` on sigma. The constructors `prior_normal()`, `prior_half_normal()`, `prior_half_cauchy()`, `prior_gamma()`, `prior_beta()`, `prior_exponential()` and `prior_pc()` build each component; `prior_predict()` draws from any of them.

Value

A `ratiod_priors` object. A component left NULL takes its default, the same for every sampler and for `prior_predict()`.

References

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

Examples

```
# Default priors
ratiod_priors()

# Custom fixed effect prior
ratiod_priors(beta = prior_normal(0, 1))

# Tighter random effect prior
ratiod_priors(sigma = prior_half_cauchy(0.5))

# PC prior on the random effect SD (Laplace backend, prior_predict())
ratiod_priors(sigma = prior_pc(U = 0.5, alpha = 0.01))

# Informative prior for temporal correlation
ratiod_priors(rho_temporal = prior_beta(5, 2)) # Prior mode at ~0.8
```

ratiod_spatial	<i>Spatial Structure Specifications</i>
----------------	---

Description

Functions to specify spatial random effects for ratio models. Spatial effects are shared between numerator and denominator by default, which helps prevent spurious ratio effects from spatially-structured unmeasured confounders.

Usage

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

Arguments

x	A ratiod_spatial object
...	Ignored

Examples

```
adj <- matrix(0, 4, 4)  
adj[cbind(1:3, 2:4)] <- 1  
adj <- adj + t(adj)  
spatial_car(adj, group_var = "region")  
spatial_gp(coords = c("x", "y"))
```

ratiod_temporal	<i>Temporal Structure Specifications</i>
-----------------	--

Description

Functions to specify temporal random effects for ratio models. Temporal effects are shared between numerator and denominator by default, which helps prevent spurious ratio effects from temporally-structured unmeasured confounders.

Usage

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

Arguments

x	A ratiod_temporal object
...	Ignored

Examples

```
temporal_rw1("year")
temporal_ar1("year")
```

ratiod_threads	<i>OpenMP Thread Count</i>
----------------	----------------------------

Description

Returns the maximum number of OpenMP threads available.

Usage

```
ratiod_threads()
```

Value

Integer number of threads

Examples

```
ratiod_threads()
```

ratiod_validate	<i>Posterior Predictive Checks for Ratio Models</i>
-----------------	---

Description

Visual and numerical checks comparing observed data to posterior predictive distributions. Essential for assessing model fit.

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
if (requireNamespace("ggplot2", quietly = TRUE)) {
  pp_check(fit, type = "dens_overlay")
}
loo(fit)
```

ratiod_zi_families	<i>Zero-Inflated Model Families</i>
--------------------	-------------------------------------

Description

Distribution families for handling excess zeros in count data. Zero-inflation occurs when there are more zeros than expected under the standard count distribution.

Two model types are available:

- **Zero-inflated (ZI)**: Mixture of point mass at zero and count distribution
- **Hurdle**: Separate processes for zero vs positive counts

Usage

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

Arguments

x	A ratiod_family_zi object
...	Ignored

Examples

```
ratiod_zipois()
ratiod_hurdle_negbin()
```

ratiod_zibinomial	<i>Zero-Inflated Binomial Family</i>
-------------------	--------------------------------------

Description

Zero-inflated binomial for the numerator when modeling proportions with excess zeros (e.g., species detection/non-detection).

$$P(Y = 0) = \pi + (1 - \pi) \cdot (1 - p)^n$$

$$P(Y = y) = (1 - \pi) \cdot \binom{n}{y} p^y (1 - p)^{n-y}, \quad y > 0$$

Usage

```
ratiod_zibinomial(link_num = "logit", link_zi = "logit")
```

Arguments

link_num	Link function for the success probability: "logit" (default), "probit" or "cloglog".
link_zi	Link function for the zero-inflation probability: "logit" (default), "probit" or "cloglog".

Details

Use this family when modeling proportions (successes/trials) where excess zeros occur beyond what binomial predicts. Common applications:

- Species occupancy with false negatives
- Survey data with non-response
- Epidemiological data with under-reporting

Value

A `ratiod_family` object with zero-inflation

See Also

[ratiod_binomial\(\)](#), [ratiod_zinegbin\(\)](#)

Examples

```
# Create family object
fam <- ratiod_zibinomial()
print(fam)
```

ratiod_zinegbin	<i>Zero-Inflated Negative Binomial Family</i>
-----------------	---

Description

Zero-inflated negative binomial for the numerator process. Models excess zeros as a mixture:

$$P(Y = 0) = \pi + (1 - \pi) \cdot P_{NB}(0)$$

$$P(Y = y) = (1 - \pi) \cdot P_{NB}(y), \quad y > 0$$

where π is the zero-inflation probability and P_{NB} is the negative binomial PMF.

Usage

```
ratiod_zinegbin(
  link_num = "log",
  link_denom = "log",
  link_zi = "logit",
  denom_family = "negbin"
)
```

Arguments

link_num	Link function for the count mean: "log".
link_denom	Link function for the denominator mean: "log" (default); a gamma denominator also takes "identity" or "inverse".
link_zi	Link function for the zero-inflation probability: "logit" (default), "probit" or "cloglog".
denom_family	Denominator distribution: "negbin" (default), "gamma" for continuous effort, or "fixed" for a known exposure. A known exposure is data: it scales the count mean, which is then the rate times the exposure, and the ratio is the rate.

Details

The zero-inflation probability can have its own linear predictor via the zi argument in `tratio()`. If not specified, a single intercept is estimated.

For ecological data, zero-inflation often represents:

- Structural zeros (species truly absent vs not detected)
- Sampling zeros (inadequate effort)
- False negatives

Value

A `ratiod_family` object with zero-inflation

See Also

[ratiod_hurdle_negbin\(\)](#) for hurdle model alternative

Examples

```
# Create family object
fam <- ratiod_zinegbin()
print(fam)

# Simulate zero-inflated count data
set.seed(123)
n <- 60
zi_prob <- 0.3
habitat <- factor(rep(c("forest", "grassland"), each = n/2))
site <- factor(rep(1:10, each = n/10))
u <- rnorm(10, sd = 0.5)[site]
df <- data.frame(
  count = ifelse(runif(n) < zi_prob, 0,
    rnbinom(n, size = 3, mu = exp(2 + 0.5 * (habitat == "grassland") + u))),
  total = rnbinom(n, size = 5, mu = 50 * exp(u)),
  habitat = habitat,
  site = site
)
```

```
# Fit model
fit <- tratio(
  count | total ~ habitat + (1 | site),
  data = df,
  family = ratiod_zinegbin(),
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)
```

ratiod_zipois	<i>Zero-Inflated Poisson Family</i>
---------------	-------------------------------------

Description

Zero-inflated Poisson for the numerator process. Useful when overdispersion is primarily due to excess zeros.

Usage

```
ratiod_zipois(
  link_num = "log",
  link_denom = "log",
  link_zi = "logit",
  denom_family = "gamma"
)
```

Arguments

link_num	Link function for the count mean: "log".
link_denom	Link function for the denominator mean: "log" (default); a gamma denominator also takes "identity" or "inverse".
link_zi	Link function for the zero-inflation probability: "logit" (default), "probit" or "cloglog".
denom_family	Denominator distribution: "gamma" (default), "negbin" for a count denominator, or "fixed" for a known exposure.

Details

Use `ratiod_zipois()` when:

- Excess zeros are the main source of overdispersion
- The non-zero counts follow Poisson (no additional overdispersion)

Use `ratiod_zinegbin()` when:

- Both excess zeros AND overdispersion in positive counts

Value

A ratioid_family object with zero-inflation

Examples

```
# Create family object
fam <- ratioid_zipois()
print(fam)

# Simulate zero-inflated CPUE data
set.seed(123)
n <- 60
zi_prob <- 0.25
df <- data.frame(
  catch = ifelse(runif(n) < zi_prob, 0, rpois(n, lambda = 5)),
  effort = rgamma(n, shape = 4, rate = 1),
  depth = rnorm(n),
  vessel = factor(rep(1:6, each = n/6))
)

# Fit model
fit <- tratio(
  catch | effort ~ depth + (1 | vessel),
  data = df,
  family = ratioid_zipois(denom_family = "gamma"),
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)
```

ratioid_zoibinomial	<i>Zero-and-One-Inflated Binomial Family</i>
---------------------	--

Description

Zero-and-one inflated binomial for proportions with excess zeros AND ones. Commonly needed in ecological applications where both absence (structural) and perfect detection (saturation) have distinct processes.

Usage

```
ratioid_zoibinomial(link_num = "logit", link_zi = "logit", link_oi = "logit")
```

Arguments

link_num	Link function for the success probability: "logit" (default), "probit" or "cloglog".
link_zi	Link function for the zero-inflation probability: "logit" (default), "probit" or "cloglog".
link_oi	Link function for the one-inflation probability: "logit" (default), "probit" or "cloglog".

Details

The ZOIB (Zero-and-One Inflated Binomial) model has three components:

1. Zero process: $P(\text{structural zero}) = \pi_0$
2. One process: $P(\text{structural one} \mid \text{not zero}) = \pi_1$
3. Binomial process: $P(Y=y \mid \text{non-structural})$ follows binomial

Use when both boundaries (0 and n) have excess observations.

Value

A `ratiod_family` object with zero-and-one inflation

Examples

```
# Create family object
fam <- ratiod_zoibinomial()
print(fam)
```

sim_ratiod	<i>Simulate Data from Ratio Models</i>
------------	--

Description

Generate simulated datasets from prior or posterior predictive distributions. Useful for prior predictive checks, simulation-based calibration, and understanding model behavior.

Usage

```
sim_ratiod(
  n = 100,
  family = ratiod_negbin_negbin(),
  formula = NULL,
  beta_num = NULL,
  beta_denom = NULL,
  sigma_re = 0.5,
  phi_num = 5,
  phi_denom = 5,
```

```

    n_groups = 10,
    n_per_group = NULL,
    spatial = NULL,
    seed = NULL
)

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

Arguments

n	Number of observations to simulate.
family	A tulpaRatio family object specifying the distributions.
formula	Optional formula for generating covariates and structure. If NULL, generates intercept-only data.
beta_num	Numeric vector of true numerator coefficients. If NULL, drawn from prior.
beta_denom	Numeric vector of true denominator coefficients. If NULL, drawn from prior.
sigma_re	Standard deviation of random effects. Default 0.5.
phi_num	Overdispersion for numerator (negbin/gamma families). Default 5.
phi_denom	Overdispersion for denominator (negbin/gamma families). Default 5.
n_groups	Number of random effect groups. Default 10.
n_per_group	Observations per group. If NULL, groups are unbalanced.
spatial	Optional spatial structure specification.
seed	Random seed for reproducibility.
x	A ratiod_simdata object
...	Ignored

Details

The simulation generates data according to the ratio model structure:

For **negbin_negbin**:

- $y_{\text{num}} \sim \text{NegBin}(\mu_{\text{num}}, \phi_{\text{num}})$
- $y_{\text{denom}} \sim \text{NegBin}(\mu_{\text{denom}}, \phi_{\text{denom}})$
- $\log(\mu_{\text{num}}) = X \%*\% \beta_{\text{num}} + \text{re}$
- $\log(\mu_{\text{denom}}) = X \%*\% \beta_{\text{denom}} + \text{re}$ (shared RE)

For **binomial**:

- $y_{\text{num}} \sim \text{Binomial}(y_{\text{denom}}, p)$
- $\text{logit}(p) = X \%*\% \beta + \text{re}$

For **poisson_gamma**:

- $y_{\text{num}} \sim \text{Poisson}(\mu_{\text{num}})$
- $y_{\text{denom}} \sim \text{Gamma}(\text{shape}, \text{rate})$ where $\text{rate} = \text{shape} / \mu_{\text{denom}}$

Value

A list with class `ratiod_simdata` containing:

data Data frame with simulated observations
true_params List of true parameter values used
family Family specification
n Number of observations

See Also

[tratio\(\)](#) for model fitting, [prior_predict\(\)](#) for prior checks

Examples

```
# Simulate CPUE-type data
sim <- sim_ratiod(
  n = 200,
  family = ratiod_poisson_gamma(),
  beta_num = c(2, 0.5),      # intercept + depth effect
  beta_denom = c(1, 0.2),    # effort model
  sigma_re = 0.3,
  n_groups = 20,
  seed = 123
)

# View simulated data
head(sim$data)

# Check true ratio distribution
true_ratio <- sim$data$y_num / sim$data$y_denom
hist(true_ratio, main = "Simulated ratios")

# Simulate binomial proportion data
sim_binom <- sim_ratiod(
  n = 100,
  family = ratiod_binomial(),
  beta_num = c(-0.5, 1),     # logit-scale coefficients
  n_groups = 10,
  seed = 456
)

# Fit model to simulated data
fit <- tratio(
  y_num | y_denom ~ x1 + (1 | group),
  data = sim$data,
  family = ratiod_poisson_gamma(),
  control = list(iter = 400, chains = 1, seed = 1)
)

# Compare estimated vs. true parameters
# summary(fit)
```

```
# sim$true_params
```

```
sim_ratiod_sbc
```

```
Simulate Datasets for Simulation-Based Calibration
```

Description

Generate multiple simulated datasets for simulation-based calibration (SBC). Each dataset is generated with parameters drawn from the prior.

Usage

```
sim_ratiod_sbc(
  n_sims = 100,
  n = 100,
  family = ratiod_negbin_negbin(),
  priors = NULL,
  n_groups = 10,
  seed = NULL
)

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

Arguments

n_sims	Number of simulations to run.
n	Number of observations per simulation.
family	A tulpaRatio family object.
priors	Prior specification for parameter generation.
n_groups	Number of random effect groups.
seed	Random seed.
x	A ratiod_sbc_sims object
...	Ignored

Value

A list of ratiod_simdata objects.

See Also

[sim_ratiod\(\)](#) for single dataset simulation

Examples

```
# Generate datasets for SBC (returns immediately)
sims <- sim_ratiod_sbc(
  n_sims = 3,
  n = 30,
  family = ratiod_negbin_negbin(),
  seed = 42
)
print(sims)

# Fit model to each simulation
# ranks <- lapply(sims, function(sim) {
#   fit <- tratio(y_num | y_denom ~ x1 + (1 | group),
#   data = sim$data,
#   family = ratiod_negbin_negbin(),
#   control = list(iter = 400, chains = 1, seed = 1))
#   sum(as.matrix(fit$draws)[, "beta_num[1]"] < sim$true_params$beta_num[1])
# })
```

spatial_bym2

BYM2 Spatial Structure

Description

Specify a Besag-York-Mollie 2 (BYM2) spatial random effect. BYM2 decomposes the spatial effect into a structured (ICAR) component and an unstructured (IID) component, with a mixing parameter controlling the proportion of variance attributable to spatial structure.

BYM2 is preferred over plain CAR when you want to:

- Distinguish structured vs unstructured spatial variation
- Have an interpretable spatial fraction parameter
- Use the scaling from Riebler et al. (2016)

Usage

```
spatial_bym2(
  adjacency,
  level = c("group", "obs"),
  group_var = NULL,
  shared = TRUE,
  scale_factor = NULL,
  parameterization = c("standard", "collapsed")
)
```

Arguments

adjacency	Adjacency matrix (sparse or dense). A symmetric matrix where entry (i,j) is 1 if areas i and j are neighbors, 0 otherwise.
level	Level at which spatial structure applies: • "group": Spatial effect at the grouping variable level (e.g., sites). Requires group_var to be specified. • "obs": Spatial effect at the observation level.
group_var	Name of the grouping variable in data (required if level = "group").
shared	Logical; if TRUE (default), spatial effect enters both numerator and denominator linear predictors identically.
scale_factor	Scaling factor for the ICAR component. If NULL (default), computed from the adjacency matrix following Riebler et al.
parameterization	Parameterization of the spatial random effect. Either "standard" (default) or "collapsed" to marginalize the spatial variance for improved mixing in sparse-graph regimes.

Value

A `ratiod_spatial` object

References

Riebler, A., Sørbye, S. H., Simpson, D., & Rue, H. (2016). An intuitive Bayesian spatial model for disease mapping that accounts for scaling. *Statistical Methods in Medical Research*, 25(4), 1145-1165.

Examples

```
# Create adjacency matrix for 10 regions (chain structure)
adj <- matrix(0, 10, 10)
for (i in 1:9) {
  adj[i, i+1] <- adj[i+1, i] <- 1
}

# Create BYM2 spatial structure
bym2 <- spatial_bym2(adj, level = "group", group_var = "region")
print(bym2)

# Generate synthetic epidemiological data
set.seed(456)
n_regions <- 10
epi_data <- data.frame(
  region = factor(1:n_regions),
  age = rnorm(n_regions, 50, 10),
  cases = rbinom(n_regions, size = 100, prob = 0.15),
  population = rep(100L, n_regions)
```

```
)

# Disease mapping with BYM2 spatial smoothing
fit <- tratio(
  cases | population ~ age,
  spatial = spatial_bym2(adj, level = "group", group_var = "region"),
  data = epi_data,
  family = ratioid_binomial(),
  mode = "laplace"
)
summary(fit)
```

spatial_car	<i>CAR Spatial Structure</i>
-------------	------------------------------

Description

Specify a conditional autoregressive (CAR) spatial random effect. Supports both intrinsic CAR (ICAR) and proper CAR variants.

Usage

```
spatial_car(
  adjacency,
  level = c("group", "obs"),
  group_var = NULL,
  proper = FALSE,
  shared = TRUE,
  parameterization = c("standard", "collapsed"),
  center = TRUE
)
```

Arguments

adjacency	Adjacency matrix (sparse or dense). A symmetric matrix where entry (i,j) is 1 if areas i and j are neighbors, 0 otherwise.
level	Level at which spatial structure applies: • "group": Spatial effect at the grouping variable level (e.g., sites). Requires group_var to be specified. • "obs": Spatial effect at the observation level.
group_var	Name of the grouping variable in data (required if level = "group").
proper	Logical; if FALSE (default), uses Intrinsic CAR (ICAR) with rho = 1 fixed. If TRUE, uses proper CAR with rho estimated. See Details.
shared	Logical; if TRUE (default), spatial effect enters both numerator and denominator linear predictors identically.

parameterization

Parameterization of the spatial random effect. Either "standard" (default) or "collapsed" to marginalize the spatial variance for improved mixing in sparse-graph regimes.

center

Logical; if TRUE (default), the field's mean is removed on its way into the linear predictor, so the intercept carries the level and the field carries deviations from it. The prior still reads the uncentred field. center = FALSE requires proper = TRUE and leaves the mean in the linear predictor, where it is jointly identified with the intercept: on a 36-unit fixture the two then correlate at -0.977 and the intercept's bulk ESS is 71 of 2000 against 1953 under the default, for the same posterior on the level. An intrinsic field (proper = FALSE) has no such choice, since its mean is unidentified until it is removed.

Details

The CAR model specifies that:

$$\phi_i | \phi_{-i} \sim N \left(\rho \frac{\sum_{j \sim i} \phi_j}{n_i}, \frac{\sigma^2}{n_i} \right)$$

where n_i is the number of neighbors of area i and $j \sim i$ denotes that j is a neighbor of i .

This leads to a precision matrix:

$$Q = \tau(D - \rho W)$$

where D is the diagonal matrix of neighbor counts and W is the adjacency matrix.

ICAR (proper = FALSE, default)

- Sets $\rho = 1$ (fixed)
- Improper prior (rank-deficient Q)
- Requires sum-to-zero constraint for identifiability
- Simpler: one fewer parameter to estimate
- Standard choice for disease mapping
- Equivalent to RW1 on a graph

Proper CAR (proper = TRUE)

- Estimates ρ in $(0, 1)$ from data
- Proper prior (integrates to 1)
- ρ measures spatial autocorrelation strength
- $\rho \rightarrow 0$: approaches independence (IID)
- $\rho \rightarrow 1$: approaches ICAR
- More flexible but additional parameter to estimate
- Prior: $\rho \sim \text{Beta}(1, 1)$ (uniform on $(0, 1)$)

Value

A `ratiod_spatial` object

See Also

[spatial_bym2\(\)](#) for decomposed spatial + IID effects, [spatial_gp\(\)](#) for continuous spatial effects

Examples

```
# Create adjacency matrix for 10 regions (chain structure)
adj <- matrix(0, 10, 10)
for (i in 1:9) {
  adj[i, i+1] <- adj[i+1, i] <- 1
}

# ICAR (default, rho = 1 fixed)
icar <- spatial_car(adj, level = "group", group_var = "site")
print(icar)

# Proper CAR (rho estimated)
proper_car <- spatial_car(adj, level = "group", group_var = "site",
                          proper = TRUE)
print(proper_car)

# Generate synthetic data with spatial structure
set.seed(123)
n_sites <- 10
n_per_site <- 5
df <- data.frame(
  site = rep(1:n_sites, each = n_per_site),
  x = rnorm(n_sites * n_per_site),
  count = rpois(n_sites * n_per_site, 20),
  effort = rgamma(n_sites * n_per_site, shape = 5, rate = 1)
)

# ICAR model (standard disease mapping approach)
fit_icar <- tratio(
  count | effort ~ x,
  data = df,
  family = ratiod_poisson_gamma(),
  spatial = spatial_car(adj, level = "group", group_var = "site"),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Proper CAR model (estimate spatial autocorrelation)
fit_car <- tratio(
  count | effort ~ x,
  data = df,
  family = ratiod_poisson_gamma(),
  spatial = spatial_car(adj, level = "group", group_var = "site",
                        proper = TRUE),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)
```

```
)

# Extract rho from proper CAR
summary(fit_car) # Shows rho_spatial parameter
```

spatial_gp

*Gaussian Process Spatial Structure***Description**

Specify a Gaussian Process (GP) spatial random effect using coordinate-based distances. Uses Nearest Neighbor Gaussian Process (NNGP) approximation for computational efficiency with large datasets (scales to millions of observations).

This provides a continuous spatial effect that captures smooth spatial variation, unlike CAR/BYM2 which require discrete areal units.

Usage

```
spatial_gp(
  coords,
  cov = c("exponential", "matern", "gaussian", "spherical"),
  nu = 1.5,
  nn = 15,
  shared = TRUE,
  scale_coords = TRUE,
  parameterization = c("centered", "noncentered", "collapsed")
)

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

Arguments

coords	A one-sided formula specifying coordinate columns (e.g., $\sim \text{lon} + \text{lat}$), or a character vector of length 2 with column names.
cov	Covariance function: "exponential" (default), "matern", "gaussian", or "spherical".
nu	Smoothness parameter for Matern covariance. Common values: • 0.5: Equivalent to exponential (rough, once mean-square differentiable) • 1.5: Once differentiable (moderate smoothness) • 2.5: Twice differentiable (smooth) Ignored for non-Matern covariance functions.
nn	Number of nearest neighbors for NNGP approximation. Default 15. Larger values give better approximation but slower computation.

shared	Logical; if TRUE (default), spatial effect enters both numerator and denominator. Set to FALSE for process-specific spatial effects (triggers warning about potential confounding).
scale_coords	Logical; if TRUE (default), coordinates are scaled to unit variance before computing distances.
parameterization	Parameterization of the latent field. One of "centered" (default), "noncentered" (Matt trick, better for diffuse prior regimes), or "collapsed" (marginalize spatial variance for improved mixing). Centered is robust for informative priors.
x	A <code>ratiod_gp</code> object
...	Ignored

Details

The GP spatial model adds a spatially-correlated random effect to the linear predictor:

$$\eta(s) = X\beta + w(s)$$

where $w(s)$ follows a Gaussian process:

$$w(s) \sim GP(0, \sigma^2 C(d; \phi))$$

The correlation function $C(d; \phi)$ depends on distance d and range parameter ϕ :

- **Exponential:** $C(d) = \exp(-d/\phi)$
- **Matern:** $C(d) = \frac{2^{1-\nu}}{\Gamma(\nu)} (\sqrt{2\nu}d/\phi)^\nu K_\nu(\sqrt{2\nu}d/\phi)$
- **Gaussian:** $C(d) = \exp(-(d/\phi)^2)$
- **Spherical:** $C(d) = 1 - 1.5(d/\phi) + 0.5(d/\phi)^3$ for $d < \phi$

NNGP approximation: For computational tractability, we use the Nearest Neighbor Gaussian Process (Datta et al., 2016), which conditions each location on its k nearest neighbors. This reduces complexity from $O(n^3)$ to $O(n \cdot k^2)$, enabling models with millions of observations.

Value

A `ratiod_gp` object

References

Datta, A., Banerjee, S., Finley, A. O., & Gelfand, A. E. (2016). Hierarchical nearest-neighbor Gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, 111(514), 800-812.

See Also

`spatial_car()`, `spatial_bym2()` for areal spatial effects, `spatial_svc()` for spatially-varying coefficients, `spatial_multiscale()` for multi-scale spatial effects

Examples

```

# Create GP spatial structure
gp <- spatial_gp(~ lon + lat)
print(gp)

# Generate synthetic spatial data with a smooth field w shared by
# count and effort
set.seed(789)
n <- 40
lon <- runif(n, 0, 10)
lat <- runif(n, 0, 10)
w <- 0.6 * sin(lon / 2) + 0.6 * cos(lat / 2)
depth <- rnorm(n)
temp <- rnorm(n)
df <- data.frame(
  lon = lon,
  lat = lat,
  depth = depth,
  temp = temp,
  count = rpois(n, exp(3 + 0.3 * depth - 0.2 * temp + w)),
  effort = rgamma(n, shape = 4, rate = exp(-w))
)

# Continuous spatial effect with exponential covariance
fit <- tratio(
  count | effort ~ depth + temp,
  data = df,
  family = ratioid_poisson_gamma(),
  spatial = spatial_gp(~ lon + lat),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1, adapt_delta = 0.95)
)
summary(fit)

# Smoother spatial field with Matern covariance
fit2 <- tratio(
  count | effort ~ depth + temp,
  data = df,
  family = ratioid_poisson_gamma(),
  spatial = spatial_gp(~ lon + lat, cov = "matern", nu = 1.5),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1, adapt_delta = 0.95)
)

```

Description

Specify a Hilbert Space Gaussian Process approximation for spatial effects. HSGP approximates a GP using Laplacian eigenfunctions, providing $O(NM^2)$ complexity with analytical gradients instead of $O(Nk^2)$ with numerical gradients for NNGP.

This gives approximately 50x speedup over standard NNGP while maintaining high accuracy for smooth spatial fields.

Usage

```
spatial_hsgp(coords, m = 6, c = 1.5, shared = TRUE, scale_coords = TRUE)
```

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

Arguments

coords	A one-sided formula specifying coordinate columns (e.g., ~lon + lat), or a character vector of length 2 with column names.
m	Number of basis functions per dimension. Total basis functions will be m^2 . Default 6. Higher values give better approximation but slower computation. Recommended range: 5-15.
c	Boundary factor controlling domain extension beyond data range. Default 1.5. The domain is extended to $[-cL, cL]$ where L is half the data range. Larger values improve approximation at boundaries.
shared	Logical; if TRUE (default), spatial effect enters both numerator and denominator. Set to FALSE for process-specific spatial effects (triggers warning about potential confounding).
scale_coords	Logical; if TRUE (default), coordinates are scaled to unit variance before computing basis functions.
x	A ratiod_hsgp object
...	Ignored

Details

HSGP approximates a GP as:

$$f(x) = \sum_{j=1}^{M^2} \phi_j(x) \sqrt{S(\lambda_j)} \beta_j$$

where:

- $\phi_j(x)$ are Laplacian eigenfunctions (products of sines)
- $S(\lambda_j)$ is the spectral density of the squared exponential kernel
- $\beta_j \sim N(0, 1)$ are basis coefficients

The hyperparameters are:

- σ^2 : marginal variance (PC prior: $P(\text{sigma} > 1) = 0.01$)
- ℓ : lengthscale (LogNormal(0, 1) prior)

Advantages over NNGP:

- Analytical gradients enable ~50x speedup
- Simple parameter interpretation
- Works well for smooth spatial fields

Limitations:

- Assumes squared exponential (smooth) covariance
- Less accurate for rough fields (Matern with low nu)
- Approximation quality depends on m and c choices

Value

A `ratiod_hsgp` object

References

Riutort-Mayol, G., Bürkner, P. C., Andersen, M. R., Solin, A., & Vehtari, A. (2023). Practical Hilbert space approximate Bayesian Gaussian processes for probabilistic programming. *Statistics and Computing*, 33(1), 17.

See Also

[spatial_gp\(\)](#) for NNGP-based GP, [spatial_car\(\)](#) for areal effects

Examples

```
# Create HSGP spatial structure
hsgp <- spatial_hsgp(~ lon + lat)
print(hsgp)

# Generate synthetic spatial data
set.seed(42)
n <- 100
lon <- runif(n, 0, 10)
lat <- runif(n, 0, 10)
w <- 0.6 * sin(lon / 2) + 0.6 * cos(lat / 2)
x <- rnorm(n)
df <- data.frame(
  lon = lon,
  lat = lat,
  x = x,
  count = rpois(n, exp(3 + 0.3 * x + w)),
  effort = rgamma(n, shape = 4, rate = exp(-w))
)
```

```
# Fast spatial effect with HSGP (much faster than spatial_gp)
fit <- tratio(
  count | effort ~ x,
  data = df,
  family = ratiod_poisson_gamma(),
  spatial = spatial_hsgp(~ lon + lat, m = 6),
  mode = "hmc",
  control = list(iter = 500, warmup = 250, chains = 2, seed = 1)
)
summary(fit)
```

spatial_multiscale	<i>Multi-Scale Gaussian Process Spatial Structure</i>
--------------------	---

Description

Specify a multi-scale spatial random effect that decomposes spatial variation into local (fine-scale) and regional (broad-scale) components. Each scale has its own range and variance parameters, and both are expanded on one Hilbert-space basis (Solin and Sarkka 2020).

Usage

```
spatial_multiscale(
  coords,
  scales = c("local", "regional"),
  range_local = c(0.01, 1),
  range_regional = c(1, 10),
  m = 10,
  c_boundary = 1.5,
  shared = TRUE,
  scale_coords = TRUE,
  sampler = c("auto", "lbfgs")
)

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

Arguments

coords	A one-sided formula specifying coordinate columns (e.g., <code>~ lon + lat</code>), or a character vector of length 2 with column names.
scales	Character vector specifying scale names. Default: <code>c("local", "regional")</code> .
range_local	Prior range for local scale as <code>c(lower, upper)</code> in coordinate units. Default: <code>c(0.01, 1)</code> (after scaling).
range_regional	Prior range for regional scale as <code>c(lower, upper)</code> . Default: <code>c(1, 10)</code> (after scaling).

m	Basis functions per dimension, giving m^2 coefficients per scale. Default 10.
c_boundary	Boundary factor: the domain is extended by this multiple before the basis is built. Default 1.5.
shared	Logical; if TRUE (default), spatial effects enter both numerator and denominator.
scale_coords	Logical; if TRUE (default), coordinates are scaled to unit variance before computing distances.
sampler	Warmup mass-matrix adaptation for the two GP blocks: "auto" (default) leaves the sampler's own adaptation in place, "lbfgs" runs an L-BFGS pass during warmup to initialize it.
x	A <code>ratiod_multiscale</code> object
...	Ignored

Details

The multi-scale model decomposes spatial variation additively:

$$\eta(s) = X\beta + w_{local}(s) + w_{regional}(s)$$

where each component follows an independent Gaussian process with a squared-exponential covariance:

$$w_{local}(s) \sim GP(0, \sigma_{local}^2 C(\ell_{local}))$$

$$w_{regional}(s) \sim GP(0, \sigma_{regional}^2 C(\ell_{regional}))$$

The range priors centre each scale's lengthscale prior.

Each field is the expansion of m^2 basis functions of the domain with standard normal coefficients scaled by the covariance's spectral density, so its cost does not grow with the number of locations.

Identifiability: the data inform the sum of the two fields, and the priors on the variances and ranges decide how it is split between them. Non-overlapping range priors give each scale its own band of the spectrum. A field with one value per location for each scale (a nearest-neighbour GP per scale) leaves that split along a ridge in as many dimensions as there are locations, which NUTS does not traverse, so the basis is the only representation offered.

Value

A `ratiod_multiscale` object

References

Solin, A. and Sarkka, S. (2020). Hilbert space methods for reduced-rank Gaussian process regression. *Statistics and Computing*, 30(2), 419-446.

See Also

`spatial_gp()` for single-scale GP, `temporal_multiscale()` for multi-scale temporal effects

Examples

```
# Create multi-scale spatial structure
ms <- spatial_multiscale(
  ~ lon + lat,
  range_local = c(0.1, 0.5),
  range_regional = c(1, 5)
)
print(ms)

# A regional trend plus local variation, shared by count and effort
set.seed(101)
n <- 60
lon <- runif(n, 0, 10)
lat <- runif(n, 0, 10)
w <- 0.5 * sin(lon / 3) + 0.5 * cos(lat / 3) + 0.3 * sin(2 * lon) * cos(2 * lat)
depth <- rnorm(n)
temp <- rnorm(n)
df <- data.frame(
  lon = lon,
  lat = lat,
  depth = depth,
  temp = temp,
  count = rpois(n, exp(3 + 0.3 * depth - 0.2 * temp + w)),
  effort = rgamma(n, shape = 5, rate = exp(-w))
)

# Multi-scale spatial with local and regional components
fit <- tratio(
  count | effort ~ depth + temp,
  data = df,
  family = ratiod_poisson_gamma(),
  spatial = spatial_multiscale(
    ~ lon + lat,
    range_local = c(0.1, 0.5),
    range_regional = c(1, 5),
    m = 6
  ),
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)
```

spatial_rsr

Restricted Spatial Regression (RSR)

Description

Apply Restricted Spatial Regression to mitigate spatial confounding. RSR orthogonalizes the spatial effect to the covariate space, preventing the spatial random effect from absorbing covariate

information.

This is important when covariates are spatially smooth (e.g., climate variables, elevation) because the spatial random effect can "steal" variance from these covariates, leading to biased coefficient estimates.

Usage

```
spatial_rsr(spatial, restrict_to)

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

Arguments

<code>spatial</code>	A spatial specification (<code>spatial_gp</code> , <code>spatial_car</code> , etc.)
<code>restrict_to</code>	Formula specifying which covariates to orthogonalize against (e.g., <code>~ depth + temp</code>). The spatial effect will be constrained to be orthogonal to the column space of these covariates.
<code>x</code>	A <code>ratiod_rsr</code> object
<code>...</code>	Passed to underlying print method

Details

The RSR approach (Reich et al., 2006; Hodges & Reich, 2010) modifies the spatial random effect to be orthogonal to the fixed effect design matrix:

$$w_{RSR} = (I - P_X)w$$

where $P_X = X(X'X)^{-1}X'$ is the projection matrix onto the column space of X .

When to use RSR:

- Covariates are spatially smooth (environmental gradients)
- Interested in causal interpretation of covariate effects
- Coefficients appear attenuated toward zero

When NOT to use RSR:

- Covariates are spatially uncorrelated
- Spatial effect is the primary quantity of interest
- Prediction is the main goal (not causal inference)

Value

A modified spatial specification with RSR enabled

References

- Reich, B. J., Hodges, J. S., & Zadnik, V. (2006). Effects of residual smoothing on the posterior of the fixed effects in disease-mapping models. *Biometrics*, 62(4), 1197-1206.
- Hodges, J. S., & Reich, B. J. (2010). Adding spatially-correlated errors can mess up the fixed effect you love. *The American Statistician*, 64(4), 325-334.

See Also

[spatial_gp\(\)](#), [spatial_car\(\)](#)

Examples

```
# Create RSR spatial structure
rsr <- spatial_rsr(
  spatial_gp(~ lon + lat),
  restrict_to = ~ depth + temp
)
print(rsr)

# Generate synthetic spatial data with confounding
set.seed(404)
n <- 40
lon <- runif(n, 0, 10)
lat <- runif(n, 0, 10)
# Smooth field w shared by count and effort
w <- 0.6 * sin(lon / 2) + 0.6 * cos(lat / 2)
# Make depth and temp spatially correlated
depth <- lon/5 + rnorm(n, 0, 0.5) # Correlated with lon
temp <- lat/5 + rnorm(n, 0, 0.5)  # Correlated with lat
df <- data.frame(
  lon = lon,
  lat = lat,
  depth = depth,
  temp = temp,
  count = rpois(n, exp(2.5 + 0.3 * depth - 0.2 * temp + w)),
  effort = rgamma(n, shape = 4, rate = exp(-w))
)

# Standard GP (may have spatial confounding)
fit1 <- tratio(
  count | effort ~ depth + temp,
  data = df,
  family = ratioid_poisson_gamma(),
  spatial = spatial_gp(~ lon + lat),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# RSR to protect depth and temp coefficients
fit2 <- tratio(
```

```

count | effort ~ depth + temp,
data = df,
family = ratiod_poisson_gamma(),
spatial = spatial_rsr(
  spatial_gp(~ lon + lat),
  restrict_to = ~ depth + temp
),
mode = "hmc",
control = list(iter = 400, chains = 1, seed = 1)
)

# Compare coefficient estimates
summary(fit1) # May be attenuated
summary(fit2) # Protected from spatial confounding

```

spatial_svc

Spatially Varying Coefficients (SVC)

Description

Specify spatially-varying coefficients for ratio models. SVCs allow regression coefficients to vary smoothly across space using a Gaussian process prior. This captures local effects that may differ from global relationships.

Uses Nearest Neighbor Gaussian Process (NNGP) approximation for computational efficiency with large datasets.

Usage

```

spatial_svc(
  coords,
  terms = 1,
  cov = c("exponential", "matern", "gaussian", "spherical"),
  nn = 15,
  shared = TRUE,
  scale_coords = TRUE,
  approx = c("nngp", "hsgp"),
  m = 6,
  c_boundary = 1.5,
  range = svc_default_range(),
  parameterization = c("noncentered", "centered")
)

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

```

Arguments

coords	A one-sided formula specifying coordinate columns (e.g., $\sim \text{lon} + \text{lat}$), or a character vector of length 2 with column names.
terms	Which coefficients should vary spatially. Options: • Integer vector: Column indices of design matrix (1 = intercept) • Character vector: Coefficient names (e.g., "(Intercept)", "depth") • Formula: $\sim 1 + \text{depth}$ for intercept and depth
cov	Covariance function: "exponential" (default), "matern", "gaussian", or "spherical".
nn	Number of nearest neighbors for NNGP approximation. Default 15. Larger values give better approximation but slower computation.
shared	Logical; if TRUE (default), SVC effects enter both numerator and denominator. Set to FALSE for process-specific SVCs (triggers warning about potential confounding).
scale_coords	Logical; if TRUE (default), coordinates are scaled to unit variance before computing distances.
approx	Spatial GP approximation. One of "nngp" (default, Nearest Neighbor GP) or "hsgp" (Hilbert Space GP, basis-function approximation suitable for moderate N with strong smoothness).
m	Number of basis functions per coordinate for HSGP (used only when approx = "hsgp"). Default 6.
c_boundary	Boundary scaling factor for HSGP. The HSGP domain is extended to c_boundary times the half-range of scaled coordinates. Default 1.5.
range	Prior range for the SVC range parameter ϕ as c(lower, upper), in the units of the scaled coordinates. The prior is uniform on that interval and the density is -Inf outside it, so a posterior that piles up against either end is reporting that the data want a range the interval excludes. Default c(0.3, 30). Applies to approx = "nngp"; the HSGP arm carries a LogNormal(0, 1) prior on its lengthscale instead and rejects a range set here.
parameterization	Coordinate each NNGP term is sampled in. "noncentered" (default) samples $z_j \sim N(0, I)$ and builds $w_j = L(\sigma_j^2, \phi_j)z_j$ through the NNGP autoregression, so the variance reaches the field through the transform rather than through the prior's quadratic form. "centered" samples the field w_j directly under its NNGP prior, where the field's level and σ_j^2 form a funnel that mixes slowly on small data. Both place the same prior on w_j . Applies to approx = "nngp"; the HSGP arm already samples standardised basis coefficients.
x	A ratioid_svc object
...	Ignored

Details

The SVC model extends the linear predictor:

$$\eta(s) = X\beta + \tilde{X}(s)w(s)$$

where:

- β are global (non-spatial) coefficients
- $\tilde{X}(s)$ is the subset of covariates with SVCs
- $w(s)$ are spatially-varying adjustments at location s

Each SVC follows an independent Gaussian process:

$$w_j(s) \sim GP(0, \sigma_j^2 \cdot C(\phi_j))$$

where $C(\phi)$ is the correlation function with range parameter ϕ .

NNGP approximation: For computational tractability, we use the Nearest Neighbor Gaussian Process (Datta et al., 2016), which conditions each location on its k nearest neighbors. This reduces complexity from $O(n^3)$ to $O(n \cdot k^2)$, enabling models with thousands of locations.

Interpretation: A positive SVC for depth at location s means the depth effect is stronger at s than the global average. The spatial variance σ_j^2 quantifies how much the effect varies across space.

Value

A `ratiod_svc` object

References

Datta, A., Banerjee, S., Finley, A. O., & Gelfand, A. E. (2016). Hierarchical nearest-neighbor Gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, 111(514), 800-812.

See Also

[spatial_car\(\)](#), [spatial_bym2\(\)](#) for areal spatial effects, [svc\(\)](#) for extracting SVC posteriors

Examples

```
# Create SVC specification
svc_spec <- spatial_svc(~ lon + lat, terms = 1)
print(svc_spec)

# Generate synthetic spatial data
set.seed(202)
n <- 30
df <- data.frame(
  lon = runif(n, 0, 10),
  lat = runif(n, 0, 10),
  depth = rnorm(n),
  temp = rnorm(n),
  count = rpois(n, 20),
  effort = rgamma(n, shape = 4, rate = 1)
)

# Spatially-varying intercept (random spatial field)
fit <- tratio(
```

```

count | effort ~ depth,
data = df,
family = ratiod_poisson_gamma(),
spatial = spatial_svc(~ lon + lat, terms = 1),
control = list(iter = 200, chains = 1, seed = 1)
)
summary(fit)

# Extract and plot the spatially-varying coefficients
svc_effects <- svc(fit)
plot(svc_effects)

```

spatiotemporal

Spatiotemporal Interaction Specifications

Description

Functions to specify spatiotemporal interaction effects for ratio models. These capture dependencies that arise when spatial patterns vary over time, or when temporal trends differ across space.

Specify a spatiotemporal interaction effect for ratio models. The interaction captures structured or unstructured deviation from the additive spatial + temporal model.

Usage

```

spatiotemporal(
  spatial,
  temporal,
  type = c("I", "II", "III", "IV", "iid", "separable", "nonsep_gp"),
  shared = TRUE,
  parameterization = c("centered", "noncentered", "collapsed"),
  nonsep_type = c("product", "gneiting"),
  coords = NULL,
  cov_space = NULL,
  cov_time = NULL,
  nn = NULL
)

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

```

Arguments

spatial	A spatial specification from spatial_car() , spatial_bym2() , or spatial_gp() .
temporal	A temporal specification from temporal_rw1() , temporal_rw2() , temporal_ar1() , or temporal_gp() .

type	Interaction type: • "I" or "iid": Unstructured interaction (IID) • "II": Structured time at each location • "III": Structured space at each time point • "IV": Fully structured (Kronecker product of spatial and temporal) • "separable": Separable space-time GP, the product covariance $C_s(h) C_t(u)$ over the grid • "nonsep_gp": Non-separable space-time GP, see nonsep_type
shared	Logical; if TRUE (default), spatiotemporal effect enters both numerator and denominator. Set to FALSE for process-specific effects (triggers warning about potential confounding).
parameterization	With the HMC backend: "centered" (default) samples the interaction directly; "noncentered" samples a standard normal z and reconstructs the interaction from it, as $z / \sqrt{\tau}$ for the Knorr-Held types ("I" to "IV") and through the nearest-neighbour GP autoregression, $\delta_i = m_i + \sqrt{d_i} z_i$, for the GP types. When the data carry little or no space-time interaction, the posterior for τ drifts toward large values and the centered field's conditional scale shrinks with it (a funnel), which NUTS cannot span with a single step size; "noncentered" decouples the two and should be tried first when ρ for the interaction (or an intercept sharing its geometry) will not drop with more warmup. "collapsed", for the GP types, integrates the interaction out with a Laplace approximation inside HMC (Margossian et al. 2020): the sampler moves the hyperparameters and every other block, and the interaction is drawn from its Gaussian approximation at each saved iteration. That removes the interaction's field-against-hyperparameter funnel; the marginal it targets is an approximation, exact for a Gaussian likelihood. It needs a log-concave family with canonical links (binomial, Poisson-gamma, negative binomial, gamma-gamma, lognormal) without zero inflation. An HSGP spatial margin is sampled centered. Ignored outside the HMC backend.
nonsep_type	For type = "nonsep_gp": "product" (the separable covariance) or "gneiting" (Gneiting 2002). An additive covariance is not among them: on a complete $S \times T$ grid its rank is $S + T - 1$, and the directions it drops are the space-time interactions the block carries.
coords	Required for "separable" and "nonsep_gp": a formula ($\sim \text{lon} + \text{lat}$) or a character vector of two column names, giving the coordinates the interaction's covariance is built on. Where a spatial unit's rows carry different positions, the unit's coordinate is their centroid. The interaction's geometry is its own, so an areal spatial main effect pairs with a continuous interaction.
cov_space, cov_time	For the GP types: the spatial and temporal kernels, one of "exponential" (default), "matern", "gaussian", "spherical". ϕ is a lengthscale in all four.
nn	For the GP types: NNGP neighbours per grid cell (default 15, capped at $S * T - 1$), chosen on a joint space-time metric.
x	A ratioid_spatiotemporal object
...	Ignored

Details

Spatiotemporal interactions extend the basic additive model:

$$\eta_{st} = X\beta + f_s(space) + f_t(time)$$

to include interactions:

$$\eta_{st} = X\beta + f_s(space) + f_t(time) + \delta_{st}$$

where δ_{st} captures space-time interactions.

Interaction Types (following Knorr-Held, 2000):

- **Type I:** Unstructured interaction - IID $\delta_{st} \sim N(0, \sigma^2)$
- **Type II:** Structured time, unstructured space - temporal structure at each location
- **Type III:** Structured space, unstructured time - spatial structure at each time
- **Type IV:** Structured space AND time - full Kronecker interaction

Separable Models:

- **Separable:** Covariance is Kronecker product $C_{st} = C_s \otimes C_t$
- **Non-separable:** GP with joint space-time metric

Type I (IID)

Independent random effect for each space-time combination:

$$\delta_{st} \stackrel{iid}{\sim} N(0, \sigma_\delta^2)$$

This is the simplest form, requiring S*T parameters but capturing no structured interaction.

Type II (Temporal structure per location)

Each location has its own temporal random effect:

$$\delta_{.t}^{(s)} \sim RW(\sigma^2)$$

This captures location-specific temporal trends but assumes independence across locations.

Type III (Spatial structure per time point)

Each time point has its own spatial random effect:

$$\delta_{s.}^{(t)} \sim ICAR(\tau)$$

This captures time-specific spatial patterns but assumes independence across time points.

Type IV (Full structure)

Kronecker product of spatial and temporal precision matrices:

$$Q_\delta = Q_s \otimes Q_t$$

This is the most constrained model, assuming the interaction has the same structure as the marginal effects.

Separable

For GP-based effects, assumes separable covariance:

$$C(\mathbf{s}_1, t_1; \mathbf{s}_2, t_2) = C_s(\mathbf{s}_1, \mathbf{s}_2) \cdot C_t(t_1, t_2)$$

Value

A `ratiod_spatiotemporal` object

References

Knorr-Held, L. (2000). Bayesian modelling of inseparable space-time variation in disease risk. *Statistics in Medicine*, 19(17-18), 2555-2567.

Margossian, C. C., Vehtari, A., Simpson, D. and Agrawal, R. (2020). Hamiltonian Monte Carlo using an adjoint-differentiated Laplace approximation: Bayesian inference for latent Gaussian models and beyond. *Advances in Neural Information Processing Systems* 33.

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

See Also

[spatial_car\(\)](#), [spatial_gp\(\)](#), [temporal_rw1\(\)](#), [temporal_ar1\(\)](#)

Examples

```
# Create adjacency matrix for 10 regions
adj <- matrix(0, 10, 10)
for (i in 1:9) adj[i, i+1] <- adj[i+1, i] <- 1

# Type I: Unstructured interaction
st1 <- spatiotemporal(
  spatial = spatial_car(adj, level = "group", group_var = "region"),
  temporal = temporal_rw1("year"),
  type = "I"
)
print(st1)

# Type IV: Fully structured interaction
st4 <- spatiotemporal(
  spatial = spatial_car(adj, level = "group", group_var = "region"),
  temporal = temporal_rw1("year"),
  type = "IV"
)
print(st4)

# Generate synthetic spatiotemporal data
set.seed(123)
n_regions <- 10
n_years <- 8
df <- expand.grid(
  region = 1:n_regions,
  year = 2015:(2015 + n_years - 1)
)
df$x <- rnorm(nrow(df))
```

```

df$count <- rpois(nrow(df), lambda = 20)
df$effort <- rgamma(nrow(df), shape = 4, rate = 1)

# Fit model with spatiotemporal interaction
fit <- tratio(
  count | effort ~ x,
  data = df,
  family = ratioid_poisson_gamma(),
  spatiotemporal = spatiotemporal(
    spatial = spatial_car(adj, level = "group", group_var = "region"),
    temporal = temporal_rw1("year"),
    type = "IV"
  ),
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)

```

 svc

Spatially Varying Coefficients of a Fitted Model

Description

Extract posterior distributions of spatially-varying coefficients (SVCs) from a fitted ratio model with SVC specification.

Usage

```

svc(object, terms = NULL, summary = FALSE, probs = c(0.025, 0.5, 0.975), ...)

## S3 method for class 'ratioid_fit'
svc(object, terms = NULL, summary = FALSE, probs = c(0.025, 0.5, 0.975), ...)

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

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

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

```

Arguments

object	A <code>ratioid_fit</code> object fitted with <code>svc</code> argument
terms	Which SVC terms to extract. If <code>NULL</code> (default), extracts all.
summary	Logical; if <code>TRUE</code> , return summary statistics instead of full posterior draws.

probs	Quantiles to compute if summary = TRUE.
...	Ignored
x	A <code>ratiod_svc_posterior</code> object
term	Which term to plot (name or index). Default: first term.
type	Plot type: "mean" (default), "sd", or quantile (e.g., "q50")

Value

A `ratiod_svc_posterior` object containing:

- `draws`: Array of posterior draws (draws x locations x terms)
- `coords`: Coordinate matrix
- `term_names`: Names of SVC terms

See Also

[spatial_svc\(\)](#), [plot.ratiod_svc_posterior\(\)](#)

Examples

```
# Generate synthetic spatial data
set.seed(303)
n <- 30
lon <- runif(n, 0, 10)
lat <- runif(n, 0, 10)
depth <- rnorm(n)
# Spatially varying intercept w0 and depth slope w1, shared by count and
# effort
w0 <- 0.5 * sin(lon / 2) + 0.5 * cos(lat / 2)
w1 <- 0.4 * cos(lon / 3)
df <- data.frame(
  lon = lon,
  lat = lat,
  depth = depth,
  count = rpois(n, exp(3 + 0.3 * depth + w0 + w1 * depth)),
  effort = rgamma(n, shape = 4, rate = exp(-(w0 + w1 * depth)))
)

# Fit model with SVC
fit <- tratio(
  count | effort ~ depth,
  data = df,
  family = ratiod_poisson_gamma(),
  spatial = spatial_svc(~ lon + lat, terms = c(1, 2)),
  control = list(iter = 400, chains = 1, seed = 1)
)

# Extract SVC posteriors
svc_post <- svc(fit)
summary(svc_post)
```

temporal

*Temporal Effects of a Fitted Model***Description**

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

Usage

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

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

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

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

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

Arguments

object	A ratiod_fit object fitted with temporal argument
component	Which component to extract for multi-scale models: "all" (default), "trend", "seasonal", or "short_term".
summary	Logical; if TRUE, return summary statistics instead of full posterior draws.
probs	Quantiles to compute if summary = TRUE.

...	Ignored
x	A <code>ratiod_temporal_posterior</code> object
type	Plot type: "ribbon" (default) or "line"

Value

A `ratiod_temporal_posterior` object

See Also

[temporal_multiscale\(\)](#), [temporal_rw1\(\)](#)

Examples

```
# Fit model with multi-scale temporal effects
set.seed(131)
year <- 1:40
# Trend plus a 12-year cycle, shared by count and effort
eff <- 0.4 * sin(year / 8) + 0.3 * sin(2 * pi * year / 12)
x <- rnorm(40)
df <- data.frame(
  year = year,
  x = x,
  count = rpois(40, exp(3 + 0.3 * x + eff)),
  effort = rgamma(40, shape = 4.5, rate = exp(-eff))
)

fit <- tratio(
  count | effort ~ x,
  data = df,
  family = ratiod_poisson_gamma(),
  temporal = temporal_multiscale("year", trend = "rw2", seasonal = 12),
  control = list(iter = 400, chains = 1, seed = 1)
)

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

temporal_ar1

AR1 Temporal Structure (First-Order Autoregressive)

Description

Specify a first-order autoregressive temporal random effect. AR1 models temporal correlation where each time point depends on the previous one: $\phi_i[t] = \rho * \phi_i[t-1] + \epsilon[t]$.

Unlike RW1/RW2, AR1 is a proper (stationary) model with an estimated autocorrelation parameter ρ .

Usage

```
temporal_ar1(
  time_var,
  group_var = NULL,
  shared = TRUE,
  rho_prior = NULL,
  parameterization = c("centered", "noncentered")
)
```

Arguments

<code>time_var</code>	Name of the time variable in data. Can be a formula (e.g., <code>~ year</code>) or a character string (e.g., <code>"year"</code>).
<code>group_var</code>	Optional name of grouping variable for panel data. If provided, a separate AR1 process is estimated for each group.
<code>shared</code>	Logical; if TRUE (default), temporal effect enters both numerator and denominator linear predictors identically.
<code>rho_prior</code>	Prior on the autocorrelation, as a prior_beta() on $(\rho + 1) / 2$. NULL takes the model-wide <code>ratiod_priors(rho_temporal =)</code> , which defaults to $\text{Beta}(2, 2)$.
<code>parameterization</code>	Coordinate the field is sampled in. "centered" (default) samples the effects themselves; "noncentered" samples the $N(0, I)$ innovations the AR1 recursion builds the effects from, so both the precision and ρ reach the linear predictor through the transform. AR1 is proper and stationary, so this is a different construction from the intrinsic walks' $\sigma^2 * A^{-1} z$.

Details

The AR1 process has marginal variance $\sigma^2 / (1 - \rho^2)$ and correlation between time points t and s of $\rho^{|t-s|}$.

The precision matrix is tridiagonal and full rank, so no constraints are needed.

ρ is estimated on $(-1, 1)$, the range every AR1 correlation in the package is sampled on, so an alternating series is reachable. The sampled coordinate is $\text{logit}((\rho + 1) / 2)$, which is what `rho_prior` places its Beta on.

Value

A `ratiod_temporal` object

Examples

```
# Create temporal AR1 specification
temporal_ar1("year")
temporal_ar1("year", group = "site")

# AR1 temporal correlation
```

```

set.seed(127)
df <- data.frame(
  year = rep(2005:2024, each = 3),
  x = rnorm(60),
  count = rpois(60, lambda = 18),
  effort = rgamma(60, shape = 3.5, rate = 1)
)

fit <- tratio(
  count | effort ~ x,
  data = df,
  family = ratioid_poisson_gamma(),
  temporal = temporal_ar1("year"),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Panel data with group-specific AR1
set.seed(128)
df_panel <- data.frame(
  year = rep(2012:2027, each = 3),
  site = rep(paste0("site", 1:3), times = 16),
  x = rnorm(48),
  count = rpois(48, lambda = 12),
  effort = rgamma(48, shape = 2.5, rate = 1)
)

fit2 <- tratio(
  count | effort ~ x,
  data = df_panel,
  family = ratioid_poisson_gamma(),
  temporal = temporal_ar1("year", group_var = "site"),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

```

temporal_gp

*Gaussian Process Temporal Structure***Description**

Specify a Gaussian Process (GP) temporal random effect for irregularly-spaced or continuous time points. Unlike RW1/RW2/AR1 which assume equally-spaced observations, GP temporal effects model correlation as a function of time distance.

This is particularly useful for:

- Irregularly-spaced time series
- Continuous time (e.g., exact timestamps)
- Smooth temporal trends with uncertainty

Usage

```
temporal_gp(
  time_var,
  cov = c("exponential", "matern", "gaussian", "periodic"),
  nu = 1.5,
  period = NULL,
  group_var = NULL,
  shared = TRUE,
  scale_coords = TRUE,
  parameterization = c("noncentered", "centered")
)

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

Arguments

time_var	Name of the time variable in data. Can be a formula (e.g., ~ year) or a character string (e.g., "year"). Should be numeric (continuous time) or convertible to numeric.
cov	Covariance function: "exponential" (default, rough), "matern" (tunable smoothness), "gaussian" (very smooth), or "periodic" (for seasonal patterns).
nu	Smoothness parameter for Matern covariance. Common values: • 0.5: Equivalent to exponential (rough) • 1.5: Once differentiable (moderate smoothness) • 2.5: Twice differentiable (smooth) Ignored for non-Matern covariance functions.
period	Period for periodic covariance (e.g., 12 for monthly, 365 for daily data with annual cycle). Only used when cov = "periodic".
group_var	Optional name of grouping variable for panel data. If provided, separate GPs are estimated for each group.
shared	Logical; if TRUE (default), temporal effect enters both numerator and denominator.
scale_coords	Logical; if TRUE (default), time values are scaled to unit variance before computing distances.
parameterization	Parameterization for GP effects: "noncentered" (default) stores $z \sim N(0,1)$ and scales by covariance (better for weakly-informed effects); "centered" stores effects directly (better for strongly-informed effects).
x	A ratioid_temporal_gp object
...	Ignored

Details

The GP temporal model adds a time-correlated random effect:

$$\eta(t) = X\beta + f(t)$$

where $f(t)$ follows a Gaussian process:

$$f(t) \sim GP(0, \sigma^2 C(|t - t'|; \phi))$$

The correlation function $C(d; \phi)$ depends on time distance d :

- **Exponential:** $C(d) = \exp(-d/\phi)$ - continuous but not differentiable
- **Matern:** Smooth with tunable roughness via ν
- **Gaussian:** $C(d) = \exp(-(d/\phi)^2)$ - infinitely differentiable
- **Periodic:** $C(d) = \exp(-2 \sin^2(\pi d/p)/\phi^2)$ - for seasonal data

Implementation: Uses a state-space representation for $O(n)$ computational complexity when possible (exponential, Matern with half-integer ν).

Value

A `ratiod_temporal_gp` object

See Also

[temporal_rw1\(\)](#), [temporal_ar1\(\)](#) for equally-spaced temporal effects, [spatial_gp\(\)](#) for spatial GP effects

Examples

```
# Create GP temporal specification
temporal_gp("timestamp")
temporal_gp("day", cov = "matern", nu = 1.5)
temporal_gp("month", cov = "periodic", period = 12)

# Irregularly-spaced time series
set.seed(140)
times <- sort(runif(30, 0, 100))
df <- data.frame(
  time = times,
  x = rnorm(30),
  count = rpois(30, lambda = 20),
  effort = rgamma(30, shape = 4, rate = 1)
)

fit <- tratio(
  count | effort ~ x,
  data = df,
  family = ratiod_poisson_gamma(),
  temporal = temporal_gp("time"),
  control = list(iter = 400, chains = 1, seed = 1)
)
```

temporal_iid	<i>IID Temporal Structure (Independent Effects)</i>
--------------	---

Description

Specify an independent temporal random effect: each time point gets its own coefficient with $\phi_i[t] \sim N(0, \sigma^2)$, no dependence between them.

Use it for a year or season effect that varies without a trend, and as the unstructured comparison against which `temporal_rw1()`, `temporal_rw2()` and `temporal_ar1()` add smoothness.

Usage

```
temporal_iid(time_var, group_var = NULL, shared = TRUE)
```

Arguments

<code>time_var</code>	Name of the time variable in data. Can be a formula (e.g., <code>~ year</code>) or a character string (e.g., "year").
<code>group_var</code>	Optional name of grouping variable for panel data. If provided, separate random walks are estimated for each group. Each group's walk sums to zero over time, so it carries the group's trend and not its level; add a group intercept such as <code>(1 site)</code> for the levels.
<code>shared</code>	Logical; if TRUE (default), temporal effect enters both numerator and denominator linear predictors identically.

Details

The precision matrix is $\tau \times I$: diagonal and full rank, so no constraint is needed and no ordering of the time points is used. A model whose time points carry no meaningful order fits the same way here as one whose do.

Value

A `ratiod_temporal` object

Examples

```
# Create temporal IID specification
temporal_iid("year")

# Unstructured year effects
set.seed(129)
df <- data.frame(
  year = rep(2010:2029, each = 2),
  x = rnorm(40),
  count = rpois(40, lambda = 22),
```

```

    effort = rgamma(40, shape = 4, rate = 1)
  )

  fit <- tratio(
    count | effort ~ x,
    data = df,
    family = ratiod_poisson_gamma(),
    temporal = temporal_iid("year"),
    mode = "hmc",
    control = list(iter = 400, chains = 1, seed = 1)
  )

```

temporal_multiscale *Multi-Scale Temporal Structure*

Description

Specify a multi-scale temporal random effect that decomposes temporal variation into trend, seasonal, and short-term components. Each component has its own variance parameter and structure.

This is particularly useful for long time series where patterns exist at multiple temporal scales (e.g., annual cycles, decadal trends).

Usage

```

temporal_multiscale(
  time_var,
  trend = c("rw2", "rw1", "none"),
  seasonal = NULL,
  short_term = c("ar1", "iid", "none"),
  group_var = NULL,
  shared = TRUE,
  parameterization = c("centered", "noncentered")
)

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

```

Arguments

time_var	Name of the time variable in data. Can be a formula (e.g., ~ year) or a character string (e.g., "year").
trend	Type of trend component: "rw2" (default, smooth), "rw1" (less smooth), or "none".
seasonal	Period for seasonal component (integer). Set to NULL or 0 for no seasonal component. Common values: 12 (monthly), 52 (weekly).

short_term	Type of short-term residual component: "ar1" (default, correlated), "iid" (independent), or "none".
group_var	Optional name of grouping variable for panel data.
shared	Logical; if TRUE (default), temporal effects enter both numerator and denominator.
parameterization	Coordinate the intrinsic trend and seasonal arms are sampled in. "centered" (default) samples the effects themselves. "noncentered" samples $z \sim N(0, I)$ and reaches the effects through $\sigma * A^{-1} z$, where A stacks the arm's difference operator on $1' / \sqrt{n}$, so the variance enters the linear predictor through the transform rather than through a quadratic form. The two describe the same model; what differs is the geometry the sampler sees. Prefer "noncentered" when an arm's variance is shrunk toward zero, where a centred coordinate leaves a funnel it cannot descend: on a 500-observation fixture whose true trend is zero, the divergence count goes 30 of 2000 to 3, the bulk ESS of σ^2_{trend} 21 to 310 and its $\hat{r}$ 1.13 to 1.01, and the centred run reports a posterior median four times larger because it never reaches the small end. The short-term arm is proper and is the same block either way.
x	A <code>ratiod_temporal_multiscale</code> object
...	Ignored

Details

The multi-scale temporal model decomposes variation additively:

$$\eta(t) = trend(t) + seasonal(t) + short(t)$$

where:

- **trend(t)**: Long-term smooth change via RW1 or RW2
- **seasonal(t)**: Repeating pattern with period P via cyclic RW1
- **short(t)**: Residual temporal correlation via AR(1) or IID

Each component has a separate variance parameter with PC priors to favor simpler models (fewer active components).

Sum-to-zero constraints are applied to trend and seasonal components for identifiability with the intercept.

Value

A `ratiod_temporal_multiscale` object

See Also

`temporal_rw1()`, `temporal_rw2()`, `temporal_ar1()` for single-component temporal effects, `spatial_multiscale()` for multi-scale spatial effects

Examples

```
# Create multi-scale temporal specification
temporal_multiscale("month_id", trend = "rw2", seasonal = 12, short_term = "ar1")
temporal_multiscale("year", trend = "rw2", seasonal = NULL, short_term = "none")

# Decompose into trend + seasonal + short-term
set.seed(129)
month_id <- 1:48
# Trend, 12-month cycle and AR(1) noise, shared by count and effort
eff <- 0.4 * sin(month_id / 15) + 0.4 * sin(2 * pi * month_id / 12) +
  as.numeric(arima.sim(list(ar = 0.6), 48, sd = 0.15))
x <- rnorm(48)
df <- data.frame(
  month_id = month_id,
  x = x,
  count = rpois(48, exp(3 + 0.3 * x + eff)),
  effort = rgamma(48, shape = 4, rate = exp(-eff))
)

fit <- tratio(
  count | effort ~ x,
  data = df,
  family = ratioid_poisson_gamma(),
  temporal = temporal_multiscale(
    time_var = "month_id",
    trend = "rw2",
    seasonal = 12,
    short_term = "ar1"
  ),
  control = list(iter = 1000, chains = 1, seed = 1)
)

# Extract and plot components
temporal_effects <- temporal(fit)
plot(temporal_effects, component = "trend")
```

temporal_rtr

Restricted Temporal Regression (RTR)

Description

Apply Restricted Temporal Regression to mitigate temporal confounding. RTR orthogonalizes the temporal effect to the covariate space, preventing the temporal random effect from absorbing covariate information.

This is important when covariates have temporal trends (e.g., increasing over time) because the temporal random effect can "steal" variance from these covariates, leading to biased coefficient estimates.

Usage

```
temporal_rtr(temporal, restrict_to)

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

Arguments

temporal	A temporal specification (temporal_rw1, temporal_gp, etc.)
restrict_to	Formula specifying which covariates to orthogonalize against (e.g., ~ year_effect + trend). The temporal effect will be constrained to be orthogonal to the column space of these covariates.
x	A ratiod_rtr object
...	Passed to underlying print method

Details

The RTR approach (analogous to Restricted Spatial Regression) modifies the temporal random effect to be orthogonal to the fixed effect design matrix:

$$f_{RTR}(t) = (I - P_X)f(t)$$

where $P_X = X(X'X)^{-1}X'$ is the projection matrix onto the column space of X .

When to use RTR:

- Covariates have temporal trends (e.g., increasing/decreasing over time)
- Interested in causal interpretation of covariate effects
- Coefficients appear attenuated toward zero
- Time and covariates are correlated

When NOT to use RTR:

- Covariates are temporally uncorrelated (random sampling over time)
- Temporal effect is the primary quantity of interest
- Prediction is the main goal (not causal inference)

Value

A modified temporal specification with RTR enabled

References

Hanks, E. M., Schliep, E. M., Hooten, M. B., & Hoeting, J. A. (2015). Restricted spatial regression in practice: geostatistical models, confounding, and robustness under model misspecification. *Environmetrics*, 26(4), 243-254.

See Also

[temporal_rw1\(\)](#), [temporal_rw2\(\)](#), [temporal_gp\(\)](#), [spatial_rsr\(\)](#) for the spatial analog

Examples

```
# Create RTR temporal structure
rtr <- temporal_rtr(
  temporal_rw2("year"),
  restrict_to = ~ temperature + precipitation
)
print(rtr)

# Generate data with temporal confounding
set.seed(170)
years <- 2000:2023
n_per_year <- 4
df <- data.frame(
  year = rep(years, each = n_per_year),
  # Temperature increasing over time (confounded with year)
  temperature = rep(seq(15, 18, length.out = length(years)), each = n_per_year) +
    rnorm(length(years) * n_per_year, 0, 0.5),
  count = rpois(length(years) * n_per_year, lambda = 20),
  effort = rgamma(length(years) * n_per_year, shape = 4, rate = 1)
)

# Standard RW2 (may have temporal confounding)
fit1 <- tratio(
  count | effort ~ temperature,
  data = df,
  family = ratiod_poisson_gamma(),
  temporal = temporal_rw2("year"),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# RTR to protect temperature coefficient
fit2 <- tratio(
  count | effort ~ temperature,
  data = df,
  family = ratiod_poisson_gamma(),
  temporal = temporal_rtr(
    temporal_rw2("year"),
    restrict_to = ~ temperature
  ),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Compare coefficient estimates
summary(fit1) # May be attenuated
summary(fit2) # Protected from temporal confounding
```

temporal_rw1

*RW1 Temporal Structure (First-Order Random Walk)***Description**

Specify a first-order random walk temporal random effect. RW1 assumes that the difference between consecutive time points is normally distributed: $\phi[t] - \phi[t-1] \sim N(0, \sigma^2)$.

This is an intrinsic model (improper prior) that captures smooth temporal trends. It is equivalent to an ICAR model on a 1D chain.

Usage

```
temporal_rw1(
  time_var,
  group_var = NULL,
  cyclic = FALSE,
  shared = TRUE,
  parameterization = c("centered", "noncentered")
)
```

Arguments

time_var	Name of the time variable in data. Can be a formula (e.g., <code>~ year</code>) or a character string (e.g., <code>"year"</code>).
group_var	Optional name of grouping variable for panel data. If provided, separate random walks are estimated for each group. Each group's walk sums to zero over time, so it carries the group's trend and not its level; add a group intercept such as <code>(1 site)</code> for the levels.
cyclic	Logical; if TRUE, the random walk wraps around (first and last time points are neighbors). Useful for seasonal data.
shared	Logical; if TRUE (default), temporal effect enters both numerator and denominator linear predictors identically.
parameterization	Coordinate the walk is sampled in. "centered" (default) samples the effects themselves; "noncentered" samples $z \sim N(0, I)$ and reaches the effects through $\sigma * A^{-1} z$, where A stacks the walk's difference operator on $1' / \sqrt{n}$, so the variance enters the linear predictor through the transform rather than through a quadratic form. The two describe the same model and differ in the geometry the sampler sees: non-centred helps where the variance is shrunk toward zero and leaves a funnel the centred coordinate cannot descend, and can be worse where the data pin the field. Measured on a 500-observation fixture with no temporal signal, an RW1's bulk ESS for <code>tau_temporal</code> goes 409 to 805, while an RW2 picks up 23 divergences of 2000 for no gain, which is why this is

an argument rather than a default. With `group_var` the prior still augments one global constant, so the transform carries the group contrasts as free directions beside the scaled ones and the group levels stay unshrunk, as they are in the centred coordinate.

Details

The RW1 precision matrix Q has the form (for T time points):

$$Q[t, t] = 1 \text{ for } t = 1, T$$

$$Q[t, t] = 2 \text{ for } 1 < t < T$$

$$Q[t, t - 1] = Q[t - 1, t] = -1$$

This is a rank-deficient matrix (rank $T-1$), so a sum-to-zero constraint is applied for identifiability.

Value

A `ratiod_temporal` object

Examples

```
# Create temporal RW1 specification
temporal_rw1("year")
temporal_rw1("month", cyclic = TRUE)

# Simple temporal trend: a random walk shared by count and effort
set.seed(123)
year <- rep(2010:2021, each = 4)
trend <- cumsum(rnorm(12, sd = 0.25))[year - 2009]
x <- rnorm(48)
df <- data.frame(
  year = year,
  x = x,
  count = rpois(48, exp(3 + 0.3 * x + trend)),
  effort = rgamma(48, shape = 5, rate = exp(-trend))
)

fit <- tratio(
  count | effort ~ x,
  data = df,
  family = ratiod_poisson_gamma(),
  temporal = temporal_rw1("year"),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Panel data: separate trends per site around each site's level
set.seed(124)
site <- rep(paste0("site", 1:5), times = 12)
s <- match(site, paste0("site", 1:5))
```

```

t <- rep(1:12, each = 5)
walks <- apply(matrix(rnorm(12 * 5, sd = 0.2), 12, 5), 2, cumsum)
eff <- rnorm(5, sd = 0.5)[s] + walks[cbind(t, s)]
x <- rnorm(60)
df_panel <- data.frame(
  year = 2014 + t,
  site = site,
  x = x,
  count = rpois(60, exp(2.5 + 0.3 * x + eff)),
  effort = rgamma(60, shape = 3, rate = exp(-eff))
)

# The per-site walks carry the trends, (1 | site) the site levels
fit2 <- tratio(
  count | effort ~ x + (1 | site),
  data = df_panel,
  family = ratioid_poisson_gamma(),
  temporal = temporal_rw1("year", group_var = "site"),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Cyclic for monthly seasonal data
set.seed(125)
month <- rep(1:12, times = 4)
season <- 0.5 * sin(2 * pi * month / 12)
x <- rnorm(48)
df_monthly <- data.frame(
  month = month,
  x = x,
  count = rpois(48, exp(2.5 + 0.3 * x + season)),
  effort = rgamma(48, shape = 2, rate = exp(-season))
)

fit3 <- tratio(
  count | effort ~ x,
  data = df_monthly,
  family = ratioid_poisson_gamma(),
  temporal = temporal_rw1("month", cyclic = TRUE),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

```

Description

Specify a second-order random walk temporal random effect. RW2 penalizes deviations from linearity, resulting in smoother trends than RW1: $\phi[t] - 2\phi[t-1] + \phi[t-2] \sim N(0, \sigma^2)$.

Usage

```
temporal_rw2(
  time_var,
  group_var = NULL,
  cyclic = FALSE,
  shared = TRUE,
  parameterization = c("centered", "noncentered")
)
```

Arguments

time_var	Name of the time variable in data. Can be a formula (e.g., <code>~ year</code>) or a character string (e.g., <code>"year"</code>).
group_var	Optional name of grouping variable for panel data. If provided, separate random walks are estimated for each group. Each group's walk sums to zero over time, so it carries the group's trend and not its level; add a group intercept such as <code>(1 site)</code> for the levels.
cyclic	Logical; if TRUE, the random walk wraps around (first and last time points are neighbors). Useful for seasonal data.
shared	Logical; if TRUE (default), temporal effect enters both numerator and denominator linear predictors identically.
parameterization	Coordinate the walk is sampled in. "centered" (default) samples the effects themselves; "noncentered" samples $z \sim N(0, I)$ and reaches the effects through $\sigma * A^{-1} z$, where A stacks the walk's difference operator on $1'/\sqrt{n}$, so the variance enters the linear predictor through the transform rather than through a quadratic form. The two describe the same model and differ in the geometry the sampler sees: non-centred helps where the variance is shrunk toward zero and leaves a funnel the centred coordinate cannot descend, and can be worse where the data pin the field. Measured on a 500-observation fixture with no temporal signal, an RW1's bulk ESS for <code>tau_temporal</code> goes 409 to 805, while an RW2 picks up 23 divergences of 2000 for no gain, which is why this is an argument rather than a default. With <code>group_var</code> the prior still augments one global constant, so the transform carries the group contrasts as free directions beside the scaled ones and the group levels stay unshrunk, as they are in the centred coordinate.

Details

RW2 produces smoother curves than RW1 because it penalizes the second derivative (curvature) rather than the first derivative (slope). It requires at least 3 time points.

The precision matrix is rank $T-2$ (two constraints needed).

Value

A `ratiod_temporal` object

Examples

```
# Create temporal RW2 specification
temporal_rw2("year")

# Smooth temporal trend
set.seed(126)
df <- data.frame(
  year = rep(2008:2027, each = 2),
  x = rnorm(40),
  count = rpois(40, lambda = 25),
  effort = rgamma(40, shape = 4, rate = 1)
)

fit <- tratio(
  count | effort ~ x,
  data = df,
  family = ratiod_poisson_gamma(),
  temporal = temporal_rw2("year"),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)
```

`temporal_tvc`

Temporally Varying Coefficients (TVC)

Description

Specify temporally-varying coefficients for ratio models. TVCs allow regression coefficients to change smoothly over time using a Gaussian process or random walk prior. This captures how effects evolve temporally.

Usage

```
temporal_tvc(
  time_var,
  terms = 1,
  structure = c("rw1", "rw2", "ar1", "iid", "gp"),
  group_var = NULL,
  shared = TRUE
)

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

Arguments

time_var	Name of the time variable in data. Can be a formula (e.g., <code>~ year</code>) or a character string (e.g., <code>"year"</code>).
terms	Which coefficients should vary over time. Options: • Integer vector: Column indices of design matrix (1 = intercept) • Character vector: Coefficient names (e.g., <code>"(Intercept)"</code>, <code>"depth"</code>) • Formula: <code>~ 1 + depth</code> for intercept and depth
structure	Temporal structure for the varying coefficients: • <code>"rw1"</code>: First-order random walk (default) • <code>"rw2"</code>: Second-order random walk (smoother) • <code>"ar1"</code>: First-order autoregressive • <code>"iid"</code>: Independent draws at each time point (no temporal smoothing) • <code>"gp"</code>: Gaussian process (for irregular time spacing). Not yet implemented; <code>temporal_tvc()</code> errors if selected.
group_var	Optional name of grouping variable for panel data.
shared	Logical; if TRUE (default), TVC effects enter both numerator and denominator. Set to FALSE for process-specific TVCs (triggers warning about potential confounding).
x	A <code>ratiod_tvc</code> object
...	Ignored

Details

The TVC model extends the linear predictor:

$$\eta(t) = X\beta + \tilde{X}(t)w(t)$$

where:

- β are global (non-time-varying) coefficients
- $\tilde{X}(t)$ is the subset of covariates with TVCs
- $w(t)$ are time-varying adjustments

Each TVC follows a temporal process:

$$w_j(t) \sim RW1(\sigma_j^2) \text{ or other temporal structure}$$

Interpretation: A positive TVC for depth at time t means the depth effect is stronger at t than the global average. The temporal variance σ_j^2 quantifies how much the effect varies over time.

Identifiability: TVCs are centered (sum-to-zero constraint) to separate the time-varying component from the global effect.

Value

A `ratiod_tvc` object

See Also

[spatial_svc\(\)](#) for spatially-varying coefficients, [temporal_rw1\(\)](#) for temporal random effects, [tvc\(\)](#) for extracting TVC posteriors

Examples

```
# Create TVC specification
tvc <- temporal_tvc("year", terms = 1)
print(tvc)

# Generate synthetic data with time-varying effect
set.seed(150)
df <- data.frame(
  year = rep(2010:2025, each = 5),
  x = rnorm(80),
  count = rpois(80, lambda = 20),
  effort = rgamma(80, shape = 4, rate = 1)
)

# Time-varying intercept (random temporal field)
fit <- tratio(
  count | effort ~ x,
  data = df,
  family = ratiod_poisson_gamma(),
  temporal = temporal_tvc("year", terms = 1),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)
summary(fit)

# Time-varying effect of x
fit2 <- tratio(
  count | effort ~ x,
  data = df,
  family = ratiod_poisson_gamma(),
  temporal = temporal_tvc("year", terms = c("(Intercept)", "x"), structure = "rw2"),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
)

# Extract and plot the time-varying coefficients
tvc_effects <- tvc(fit2)
plot(tvc_effects, "x")
```

Description

Fit a Bayesian hierarchical model for ratios, rates, or proportions. The model jointly estimates the numerator and denominator processes with optional shared latent structure.

`tratio()` is `tulpaRatio`'s front door, the ratio-model member of the `tulpa*` family of fitting verbs (`tulpa::tulpa()` for the engine, `tulpaObs::tobs()` for observation processes).

Core principle: Ratios are derived quantities, computed post hoc from the posterior of the joint model. The ratio is never modelled directly.

Usage

```
tratio(
  formula,
  data,
  family = ratioid_negbin_negbin(),
  formula_num = NULL,
  formula_denom = NULL,
  shared = NULL,
  spatial = NULL,
  temporal = NULL,
  spatiotemporal = NULL,
  zi = NULL,
  latent = NULL,
  priors = NULL,
  mode = c("auto", "exact", "structured", "optimized", "hmc", "ess", "pg", "gibbs",
    "sghmc", "sgld", "laplace", "vi"),
  control = list()
)
```

Arguments

formula	Model formula specifying the response and predictors. Two syntax options: Combined (recommended) <code>num denom ~ predictors + (1 group)</code> where both processes share the same predictors and random effects; Separate <code>num ~ predictors</code> with the <code>formula_denom</code> argument.
data	Data frame containing all variables.
family	A <code>tulpaRatio</code> family object specifying the distributions: <code>ratioid_negbin_negbin()</code> for both count processes (default), <code>ratioid_binomial()</code> for successes/trials, <code>ratioid_poisson_gamma()</code> for count/continuous effort (CPUE).
formula_num	Optional one-sided formula for additional numerator predictors: <code>~ extra_terms</code> . Added to the main formula.
formula_denom	Optional formula for denominator. Required if main formula has single response. Can be <code>denom ~ predictors</code> .
shared	Formula specifying shared random effects structure: <code>NULL</code> (default) infers from matching random effects in both processes; <code>~ (1 group)</code> for explicit shared structure; <code>~ 0</code> for independence assumption (triggers warning).
spatial	Optional spatial structure specification. See <code>spatial_car()</code> , <code>spatial_bym2()</code> .

temporal	Optional temporal structure specification. See temporal_rw1() , temporal_rw2() , temporal_ar1() .
spatiotemporal	Optional spatiotemporal interaction specification. See spatiotemporal() .
zi	Optional zero-inflation specification. See zi_poisson() , zi_negbin() , hurdle_poisson() , hurdle_negbin() .
latent	Optional latent factor specification for unmeasured confounders. See latent_factor() .
priors	Prior specification. See ratiod_priors() .
mode	Inference mode or backend. Accepts either: Tier names (epistemic guarantees): • "auto" (default): Selects between Exact and Structured based on model. Never silently chooses Optimized. • "exact" (Tier 1): Asymptotically correct posterior inference. Credible intervals are interpretable as posterior uncertainty. • "structured" (Tier 2): Accurate inference conditional on structural assumptions (latent Gaussian, conditional independence). • "optimized" (Tier 3): No general correctness guarantee. Point estimates usually good, uncertainty often underestimated. Requires explicit opt-in. Backend names (specific implementation): • "hmc" (Tier 1): Native HMC/NUTS sampler • "ess" (Tier 1): Elliptical Slice Sampling • "pg" (Tier 1): Pólya-Gamma Gibbs (binomial only) • "sghmc" (Tier 1): Stochastic Gradient HMC (large datasets) • "sgld" (Tier 1): Stochastic Gradient Langevin Dynamics (large datasets) • "laplace" (Tier 2): Laplace approximation • "vi" (Tier 3): Variational Inference Structures each backend fits. "hmc" fits every structure. The rest carry a subset, and a model naming a structure its backend cannot fit is refused with that structure named: • "ess", "sghmc", "sgld", "vi": spatial, temporal, zi • "gibbs": spatial, and temporal as temporal_tvc() • "pg", "laplace": spatial, or temporal as temporal_multiscale(), one of the two per model "auto" selects a backend that fits the whole model, so it reaches "laplace" or "gibbs" only for models those carry. See inference_mode_info() for details on the tier system.
control	Named list of perf / numerical / tuning knobs. Statistical arguments stay in the signature above; everything that tunes <i>how</i> the fit runs lives here. Unknown names are an error, not a silent no-op. Sampling budget (all backends): • chains: number of MCMC chains (default 4). • iter: total iterations per chain (default 2000). • warmup: warmup iterations per chain (default floor(iter / 2)).

- `thin`: thinning interval (default 1).
- `cores`: cores for parallel chains (default `getOption("mc.cores", chains)`).
- `seed`: random seed for reproducibility.
- `verbose`: print progress messages and sampler adaptation traces during fitting (default `FALSE`).

NUTS / HMC (mode = "hmc"):

- `adapt_delta`: target average acceptance probability for step-size adaptation. Higher values give smaller steps, reducing divergences but slowing sampling. `NULL` (default) selects on model complexity: 0.80 simple, 0.85 ICAR, 0.90 BYM2 and correlated random slopes. Range 0.80–0.99.
- `max_treedepth`: maximum NUTS tree depth. `NULL` (default) uses 10. Raise if you see many max-treedepth warnings; costs computation per iteration.
- `metric`: mass matrix. "auto" (default) picks on model complexity; "dense" full matrix with Ledoit-Wolf shrinkage (correlated posteriors: random slopes, BYM2, HSGP, TVC); "diag" diagonal (faster per step, may need deeper trees); "block_diag" per parameter group. Auto falls back to diagonal when $p > 2000$.
- `walnuts`: run each NUTS trajectory as a WALNUTS transition (Bou-Rabee et al. 2025, tulpa's <tulpa/walnuts.h>), which integrates each macro step at the coarsest leapfrog subdivision that keeps the joint log density within tolerance and keeps it only if the reversed step picks the same one. The kernel is exact, and its step follows the local curvature, which is what a funnel-shaped posterior (a hierarchical scale near zero) needs. `FALSE` (default) runs NUTS.
- `gradient_mode`: "auto" (default) selects the fastest available; "H" hand-coded analytical (fastest); "A_r" arena reverse-mode autodiff (fast, $O(N)$); "A" forward-mode ($O(p \cdot N)$, thread-safe); "A_t" tape-based reverse-mode (slow, legacy); "N" numerical finite differences (slowest, always works).
- `re_param`: random-effect parameterization. "noncentered" (default) stores $z \sim N(0, 1)$ and computes $re = \sigma * z$ (or $\text{diag}(\sigma) * L * z$ when correlated), suiting weakly-informed effects or small groups; "centered" stores re directly under $re \sim N(0, \sigma^2)$, suiting strongly-informed effects or large groups.
- `L`: fixed leapfrog steps. Default 0 leaves NUTS to choose.

Variational inference (mode = "vi"):

- `vi_variant`: "auto" (default) selects on parameter count; "meanfield" diagonal covariance (fastest); "lowrank" low-rank plus diagonal (balanced); "fullrank" full Cholesky (best for small models).

Stochastic gradient backends (mode = "sghmc" / "sgld"):

- `batch_size`: minibatch size (default `ceiling(sqrt(n))`).
- `epsilon`: step size.
- `alpha`: friction coefficient ("sghmc" only).
- `schedule_a`, `schedule_b`, `schedule_gamma`, `use_schedule`: step-size decay schedule ("sgld" only).

Value

A `ratiod_fit` object containing:

draws Posterior draws matrix
formula Parsed formula specification
family Model family
data Original data
backend Inference backend used
diagnostics MCMC diagnostics (if applicable)

See Also

- [ratio\(\)](#) for extracting ratio posteriors
- [ratio_contrast\(\)](#) for comparing ratios between groups
- [pp_check\(\)](#) for posterior predictive checks
- [loo::loo\(\)](#) and [loo::waic\(\)](#) for model comparison

Examples

```
# Quick example: create model object (no fitting)
# A site effect u shared by catch and effort
set.seed(123)
n <- 60
site <- factor(rep(1:10, each = 6))
u <- rnorm(10, sd = 0.5)[site]
depth <- rnorm(n)
season <- factor(sample(c("spring", "summer"), n, replace = TRUE))
df <- data.frame(
  catch = rpois(n, exp(1.5 + 0.3 * depth + 0.4 * (season == "summer") + u)),
  effort = rgamma(n, shape = 2, rate = 0.5 * exp(-u)),
  depth = depth,
  season = season,
  site = site
)

# CPUE example with combined formula
fit <- tratio(
  catch | effort ~ depth + season + (1 | site),
  data = df,
  family = ratiod_poisson_gamma(),
  control = list(iter = 400, chains = 1, seed = 1)
)

# Different predictors for each process
fit2 <- tratio(
  catch | effort ~ (1 | site),
  formula_num = ~ depth + season,
  formula_denom = ~ season,
```

```

    data = df,
    family = ratiod_poisson_gamma(),
    control = list(iter = 400, chains = 1, seed = 1)
  )

  # Extract ratio posteriors
  cpue <- ratio(fit)
  summary(cpue)

  # Compare ratios between groups
  ratio_contrast(fit, ~ season)

```

tvc

Temporally Varying Coefficients of a Fitted Model

Description

Extract posterior distributions of temporally-varying coefficients (TVCs) from a fitted ratio model with TVC specification.

Usage

```

tvc(
  object,
  terms = NULL,
  group = 1L,
  summary = FALSE,
  probs = c(0.025, 0.5, 0.975),
  ...
)

## S3 method for class 'ratiod_fit'
tvc(
  object,
  terms = NULL,
  group = 1L,
  summary = FALSE,
  probs = c(0.025, 0.5, 0.975),
  ...
)

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

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

```

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

Arguments

object	A <code>ratiod_fit</code> object fitted with <code>tvc</code> argument
terms	Which TVC terms to extract. If <code>NULL</code> (default), extracts all.
group	Which level of the temporal grouping variable to report, as an index into its levels; a model without temporal groups has the one level 1.
summary	Logical; if <code>TRUE</code> , return summary statistics instead of full posterior draws.
probs	Quantiles to compute if <code>summary = TRUE</code> .
...	Ignored
x	A <code>ratiod_tvc_posterior</code> object
term	Which term to plot (name or index). Default: first term.
type	Plot type: "ribbon" (default) or "line"

Value

A `ratiod_tvc_posterior` object containing:

- `draws`: Array of posterior draws (draws x times x terms)
- `time_levels`: Time point labels
- `term_names`: Names of TVC terms

See Also

[temporal_tvc\(\)](#), [plot.ratiod_tvc_posterior\(\)](#)

Examples

```
# Generate synthetic data
set.seed(160)
df <- data.frame(
  year = rep(2015:2024, each = 5),
  x = rnorm(50),
  count = rpois(50, lambda = 18),
  effort = rgamma(50, shape = 3.5, rate = 1)
)

# Fit model with TVC
fit <- tratio(
  count | effort ~ x,
  data = df,
  family = ratiod_poisson_gamma(),
  temporal = temporal_tvc("year", terms = c(1, 2)),
  mode = "hmc",
  control = list(iter = 400, chains = 1, seed = 1)
```

```
)

# Extract TVC posteriors
tvc_post <- tvc(fit)
summary(tvc_post)

# Plot temporal evolution
plot(tvc_post, "x")
```

validate_mode

Validate That a Fit Used the Expected Mode

Description

Ensures the fit object was created with the expected inference mode. Useful for enforcing mode requirements in downstream analysis.

Usage

```
validate_mode(fit, expected_mode, error = TRUE)
```

Arguments

fit	A <code>ratiod_fit</code> object
expected_mode	Mode that should have been used: a tier ("exact", "structured", "optimized") or a backend ("hmc", "laplace", ...), as passed to <code>tratio(mode =)</code> .
error	If TRUE (default), error on mismatch. If FALSE, return logical.

Value

If `error = FALSE`, returns TRUE/FALSE. Otherwise errors on mismatch.

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
validate_mode(fit, "hmc")
validate_mode(fit, "laplace", error = FALSE)
```

waic.ratiod_fit	<i>Widely Applicable Information Criterion (WAIC)</i>
-----------------	---

Description

Compute Widely Applicable Information Criterion (WAIC).

Usage

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

Arguments

x	A ratiod_fit object
...	Additional arguments passed to loo::waic

Details

Uses the same reconstructed pointwise log-likelihood as [loo.ratiod_fit\(\)](#); supported on the hmc, ess, sghmc, and vi backends and conditional on the fixed-effect linear predictor.

Value

A waic object

Examples

```
sim <- sim_ratiod(n = 100, family = ratiod_poisson_gamma(), seed = 1)
fit <- tratio(y_num | y_denom ~ x1 + (1 | group), data = sim$data,
             family = ratiod_poisson_gamma(),
             control = list(iter = 1000, chains = 2, seed = 1))
# waic() warns when an observation's p_waic exceeds 0.4; loo() is then
# the estimate to report
waic(fit)
loo(fit)
```

zi_binomial	<i>Zero-Inflated Binomial Specification</i>
-------------	---

Description

Zero-Inflated Binomial Specification

Usage

```
zi_binomial(formula = NULL)
```

Arguments

formula	Optional formula for the zero-inflation probability. If NULL, uses intercept-only model for P(structural zero).
---------	---

Value

A `ratiod_zi` object

Examples

```
# Intercept-only ZI
zi_binomial()

# ZI probability depends on a covariate
zi_binomial(~ habitat)
```

zi_negbin	<i>Zero-Inflated Negative Binomial Specification</i>
-----------	--

Description

Zero-Inflated Negative Binomial Specification

Usage

```
zi_negbin(formula = NULL)
```

Arguments

formula	Optional formula for the zero-inflation probability. If NULL, uses intercept-only model for P(structural zero).
---------	---

Value

A `ratiod_zi` object

Examples

```
# Intercept-only ZI
zi_negbin()

# ZI probability depends on multiple covariates
zi_negbin(~ habitat + season)
```

zi_poisson

Zero-Inflated Poisson Specification

Description

Zero-Inflated Poisson Specification

Usage

```
zi_poisson(formula = NULL)
```

Arguments

formula	Optional formula for the zero-inflation probability. If NULL, uses intercept-only model for P(structural zero).
---------	---

Value

A `ratiod_zi` object

Examples

```
# Intercept-only ZI
zi_poisson()

# ZI probability depends on a covariate
zi_poisson(~ habitat)
```

zi_specification	<i>Zero-Inflation and Hurdle Model Specifications</i>
------------------	---

Description

Functions to specify zero-inflation or hurdle model components for ratio models. These handle excess zeros in count data where the standard count distribution (Poisson, negative binomial) doesn't adequately capture the zero proportion.

Note on naming convention: These functions (`zi_poisson()`, `zi_negbin()`, `hurdle_poisson()`, `hurdle_negbin()`) are **ZI specification** functions for the `zi` argument in `tratio()`. They are distinct from the **family functions** (`ratiod_zipois()`, `ratiod_zinegbin()`, `ratiod_hurdle_pois()`, `ratiod_hurdle_negbin()`) which specify the complete model family. Use ZI specifications when you want to add zero-inflation to an existing family; use ZI family functions when you want a pre-configured zero-inflated family.

Usage

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

Arguments

<code>x</code>	A <code>ratiod_zi</code> object
<code>...</code>	Additional arguments (ignored)

Details

Zero-Inflated Models: Zero-inflated models assume two data-generating processes:

1. A process that generates "structural zeros" with probability π
2. A count process that generates counts (including zeros) with probability $1 - \pi$

The probability of observing y is:

$$P(Y = 0) = \pi + (1 - \pi) \cdot P_{count}(0)$$

$$P(Y = y|y > 0) = (1 - \pi) \cdot P_{count}(y)$$

Hurdle Models: Hurdle models treat the process as:

1. A binary process determining zero vs non-zero
2. A truncated count process (conditional on $y > 0$)

$$P(Y = 0) = 1 - \theta$$

$$P(Y = y|y > 0) = \theta \cdot P_{count}(y|y > 0)$$

When to use:

- `zi_poisson()`: Excess zeros with equidispersed positive counts
- `zi_negbin()`: Excess zeros with overdispersed positive counts
- `hurdle_poisson()`: Zero hurdle with equidispersed positive counts
- `hurdle_negbin()`: Zero hurdle with overdispersed positive counts

Value

Invisibly returns the input

Examples

```
zi_poisson()
hurdle_negbin()
```

zoib

Zero-and-One-Inflated Binomial (ZOIB) Specification

Description

Zero-and-One-Inflated Binomial (ZOIB) Specification

Usage

```
zoib(zi_formula = NULL, oi_formula = NULL)
```

Arguments

<code>zi_formula</code>	Optional formula for the zero-inflation probability. If NULL, uses intercept-only model.
<code>oi_formula</code>	Optional formula for the one-inflation probability. If NULL, uses intercept-only model.

Value

A `ratiod_zi` object

Examples

```
# Intercept-only for both zero and one inflation
zoib()
```

Index

- * **models**
 - spatiotemporal, [100](#)
 - zi_specification, [135](#)
- as.data.frame.ratioid_fit, [4](#)
- as_draws.ratioid_fit, [5](#)
- check_diagnostics, [6](#)
- check_diagnostics(), [8](#), [15](#), [27](#)
- coef.ratioid_fit (print.ratioid_sghmc), [39](#)
- coef.summary.ratioid_fit (print.ratioid_sghmc), [39](#)
- diagnostic_summary, [7](#)
- diagnostic_summary(), [27](#), [28](#), [34](#)
- diagnostics(), [8](#), [15](#), [26](#), [28](#), [33](#), [34](#)
- diagnostics.ratioid_fit, [8](#)
- draws_interval, [9](#)
- draws_long, [11](#)
- draws_wide, [12](#)
- draws_wide(), [11](#)
- fitted.ratioid_average (ratioid_average), [51](#)
- fitted.ratioid_fit, [13](#)
- formula.ratioid_fit (print.ratioid_sghmc), [39](#)
- geweke_test, [15](#)
- gpu-support, [16](#)
- gpu_available, [16](#)
- gpu_info, [17](#)
- gpu_info(), [17](#)
- hurdle_binomial, [18](#)
- hurdle_negbin, [18](#)
- hurdle_negbin(), [126](#)
- hurdle_poisson, [19](#)
- hurdle_poisson(), [126](#)
- inference_mode_info, [20](#)
- inference_mode_info(), [126](#)
- latent_factor, [20](#)
- latent_factor(), [126](#)
- latent_factors.ratioid_fit (latent_factor), [20](#)
- loo.ratioid_fit, [23](#)
- loo.ratioid_fit(), [132](#)
- loo::loo(), [128](#)
- loo::stacking_weights(), [52](#)
- loo::waic(), [128](#)
- n_divergent, [24](#)
- n_divergent(), [27](#)
- nobs.ratioid_fit (print.ratioid_sghmc), [39](#)
- oi_binomial, [25](#)
- plot.ratioid_fit (print.ratioid_sghmc), [39](#)
- plot.ratioid_svc_posterior (svc), [104](#)
- plot.ratioid_svc_posterior(), [105](#)
- plot.ratioid_temporal_posterior (temporal), [106](#)
- plot.ratioid_tvc_posterior (tvc), [129](#)
- plot.ratioid_tvc_posterior(), [130](#)
- plot_acf, [25](#)
- plot_diagnostics(), [8](#)
- plot_divergences, [26](#)
- plot_divergences(), [33](#)
- plot_energy, [27](#)
- plot_ess, [28](#)
- plot_ess(), [26](#), [34](#)
- plot_map, [29](#)
- plot_map_panel, [31](#)
- plot_pairs, [32](#)
- plot_pairs(), [27](#)
- plot_rhat, [33](#)
- plot_rhat(), [28](#)
- posterior::as_draws_df(), [5](#)
- posterior_predict.ratioid_fit, [34](#)

pp_check(), [34](#), [35](#), [128](#)
 pp_check.ratioid_fit, [35](#)
 predict.ratioid_fit, [36](#)
 predict.ratioid_fit(), [30](#)
 print.ratioid_average (ratioid_average),
 [51](#)
 print.ratioid_benchmark
 (ratioid_benchmark), [53](#)
 print.ratioid_contrast (ratio_contrast),
 [49](#)
 print.ratioid_diagnostic_summary
 (diagnostic_summary), [7](#)
 print.ratioid_diagnostics
 (diagnostics.ratioid_fit), [8](#)
 print.ratioid_family (ratioid_families),
 [58](#)
 print.ratioid_family_zi
 (ratioid_zi_families), [73](#)
 print.ratioid_fit (print.ratioid_sghmc),
 [39](#)
 print.ratioid_fitted
 (fitted.ratioid_fit), [13](#)
 print.ratioid_geweke (geweke_test), [15](#)
 print.ratioid_gp (spatial_gp), [87](#)
 print.ratioid_hsgp (spatial_hsgp), [89](#)
 print.ratioid_latent (latent_factor), [20](#)
 print.ratioid_multiscale
 (spatial_multiscale), [92](#)
 print.ratioid_prediction
 (predict.ratioid_fit), [36](#)
 print.ratioid_prior (ratioid_priors), [69](#)
 print.ratioid_priorpred (prior_predict),
 [45](#)
 print.ratioid_priors (ratioid_priors), [69](#)
 print.ratioid_ratio (ratio), [48](#)
 print.ratioid_ratio_summary (ratio), [48](#)
 print.ratioid_rsr (spatial_rsr), [94](#)
 print.ratioid_rtr (temporal_rtr), [115](#)
 print.ratioid_sbc_sims (sim_ratioid_sbc),
 [81](#)
 print.ratioid_sghmc, [39](#)
 print.ratioid_sgld (print.ratioid_sghmc),
 [39](#)
 print.ratioid_simdata (sim_ratioid), [78](#)
 print.ratioid_spatial (ratioid_spatial),
 [71](#)
 print.ratioid_spatiotemporal
 (spatiotemporal), [100](#)
 print.ratioid_svc (spatial_svc), [97](#)
 print.ratioid_svc_posterior (svc), [104](#)
 print.ratioid_temporal
 (ratioid_temporal), [71](#)
 print.ratioid_temporal_gp (temporal_gp),
 [109](#)
 print.ratioid_temporal_multiscale
 (temporal_multiscale), [113](#)
 print.ratioid_temporal_posterior
 (temporal), [106](#)
 print.ratioid_tvc (temporal_tvc), [122](#)
 print.ratioid_tvc_posterior (tvc), [129](#)
 print.ratioid_vi (print.ratioid_sghmc), [39](#)
 print.ratioid_zi (zi_specification), [135](#)
 print.summary.ratioid_fit
 (print.ratioid_sghmc), [39](#)
 print.summary.ratioid_vi
 (print.ratioid_sghmc), [39](#)
 print.tulpaRatio_gpu_info (gpu_info), [17](#)
 prior_beta, [41](#)
 prior_beta(), [108](#)
 prior_exponential, [42](#)
 prior_gamma, [42](#)
 prior_half_cauchy, [43](#)
 prior_half_normal, [43](#)
 prior_normal, [44](#)
 prior_pc, [44](#)
 prior_pc(), [22](#)
 prior_predict, [45](#)
 prior_predict(), [70](#), [80](#)
 priors_default, [47](#)
 ratio, [48](#)
 ratio(), [5](#), [30](#), [128](#)
 ratio_contrast, [49](#)
 ratio_contrast(), [128](#)
 ratioid_average, [51](#)
 ratioid_benchmark, [53](#)
 ratioid_benchmark_compare, [54](#)
 ratioid_beta_binomial, [55](#)
 ratioid_binomial, [56](#)
 ratioid_binomial(), [74](#), [125](#)
 ratioid_compare, [57](#)
 ratioid_compare(), [23](#), [52](#)
 ratioid_families, [58](#)
 ratioid_formula(), [46](#)
 ratioid_gamma_gamma, [58](#)
 ratioid_hurdle_binomial, [60](#)
 ratioid_hurdle_negbin, [60](#)

- ratiod_hurdle_negbin(), 75
- ratiod_hurdle_pois, 62
- ratiod_latent, 63
- ratiod_lognormal, 63
- ratiod_negbin_gamma, 64
- ratiod_negbin_negbin, 66
- ratiod_negbin_negbin(), 125
- ratiod_oibinomial, 67
- ratiod_poisson_gamma, 68
- ratiod_poisson_gamma(), 125
- ratiod_priors, 69
- ratiod_priors(), 46, 47, 70, 126
- ratiod_spatial, 71
- ratiod_temporal, 71
- ratiod_threads, 72
- ratiod_validate, 72
- ratiod_zi_families, 73
- ratiod_zibinomial, 73
- ratiod_zinegbin, 74
- ratiod_zinegbin(), 74
- ratiod_zipois, 76
- ratiod_zoibinomial, 77

- sim_ratiod, 78
- sim_ratiod(), 46, 81
- sim_ratiod_sbc, 81
- spatial_bym2, 82
- spatial_bym2(), 86, 88, 99, 100, 125
- spatial_car, 84
- spatial_car(), 88, 91, 96, 99, 100, 103, 125
- spatial_gp, 87
- spatial_gp(), 16, 86, 91, 93, 96, 100, 103, 111
- spatial_hsgp, 89
- spatial_multiscale, 92
- spatial_multiscale(), 88, 114
- spatial_rsr, 94
- spatial_rsr(), 117
- spatial_svc, 97
- spatial_svc(), 88, 105, 124
- spatiotemporal, 100
- spatiotemporal(), 126
- summary.ratiod_fit
 - (print.ratiod_sghmc), 39
- summary.ratiod_ratio(ratio), 48
- summary.ratiod_svc_posterior(svc), 104
- summary.ratiod_temporal_posterior
 - (temporal), 106
- summary.ratiod_tvc_posterior(tvc), 129

- summary.ratiod_vi(print.ratiod_sghmc), 39
- svc, 104
- svc(), 99

- temporal, 106
- temporal_ar1, 107
- temporal_ar1(), 100, 103, 111, 114, 126
- temporal_gp, 109
- temporal_gp(), 100, 117
- temporal_iid, 112
- temporal_multiscale, 113
- temporal_multiscale(), 93, 107, 126
- temporal_rtr, 115
- temporal_rw1, 118
- temporal_rw1(), 100, 103, 107, 111, 114, 117, 124, 126
- temporal_rw2, 120
- temporal_rw2(), 100, 114, 117, 126
- temporal_tvc, 122
- temporal_tvc(), 126, 130
- tratio, 124
- tratio(), 22, 40, 80, 135
- tulpa::diagnostics(), 8
- tulpa::energy_bfmi(), 27
- tulpa::tulpa(), 125
- tvc, 129
- tvc(), 124

- validate_mode, 131
- vcov.ratiod_fit(print.ratiod_sghmc), 39

- waic.ratiod_fit, 132
- weights.ratiod_average
 - (ratiod_average), 51

- zi_binomial, 133
- zi_negbin, 133
- zi_negbin(), 126
- zi_poisson, 134
- zi_poisson(), 126
- zi_specification, 135
- zoib, 136