Package: gretaR (via r-universe)

November 27, 2026

Title Bayesian Statistical Modelling with Torch

Version 0.5.2

Description A probabilistic programming package for Bayesian statistical modelling in R using native R syntax. Models are defined interactively with 'gretaR_array' objects, then compiled to 'torch' tensors for Hamiltonian Monte Carlo and No-U-Turn Sampler inference. Runs on a native 'torch' (libtorch) backend with no Python dependency required.

License MIT + file LICENSE

URL https://github.com/max578/gretaR, https://max578.github.io/gretaR/

BugReports https://github.com/max578/gretaR/issues

Depends R (>= 4.1.0)

Imports R6 (>= 2.5.0), torch (>= 0.12.0), cli (>= 3.6.0), posterior (>= 1.5.0), rlang (>= 1.1.0), lifecycle (>= 1.0.0), methods

Suggests Matrix, lme4, reformulas, mgcv, cmdstanr, testthat (>= 3.0.0), knitr, rmarkdown, bayesplot, ggplot2, loo, withr

VignetteBuilder knitr

Additional_repositories https://stan-dev.r-universe.dev

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-AU

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Repository https://max578.r-universe.dev

Date/Publication 2026-10-16 21:46:52 UTC

RemoteUrl https://github.com/max578/gretaR

RemoteRef main

RemoteSha c4a1515f527054691ee8a132a2ddf0647d539c40

Contents

[.gretaR_array . . . 3
%*% . . . 3
as_data . . . 4
bernoulli . . . 4
beta_dist . . . 5
binomial_dist . . . 6
cauchy . . . 6
coef.gretaR_fit . . . 7
compile_to_stan . . . 8
custom_distribution . . . 9
dirichlet . . . 10
distribution . . . 10
distribution<- . . . 11
exponential . . . 12
gamma_dist . . . 12
gretaR_glm . . . 13
half_cauchy . . . 15
half_normal . . . 16
hmc . . . 17
joint_density . . . 17
laplace . . . 18
lkj_correlation . . . 19
lognormal . . . 20
mcmc . . . 20
mean.gretaR_array . . . 23
mixture . . . 23
model . . . 24
model_from_arrays . . . 25
multivariate_normal . . . 27
negative_binomial . . . 27
normal . . . 28
nuts . . . 29
opt . . . 29
parse_re_bars . . . 31
plot.gretaR_fit . . . 31
poisson_dist . . . 32
print.gretaR_fit . . . 33
process_smooths . . . 33
random_effect . . . 34
remove_re_bars . . . 36
reset_gretaR_env . . . 37
student_t . . . 37
sum.gretaR_array . . . 38
summary.gretaR_fit . . . 38
uniform . . . 39
variable . . . 40

variational 40
wishart 42

Index **43**

`[.gretaR_array` *Extract elements from a gretaR_array*

Description

Enables indexing into group-level parameters for hierarchical models. Supports integer vector indexing (e.g., `alpha[group_id]`).

Usage

```
## S3 method for class 'gretaR_array'
x[i, j, ..., drop = TRUE]
```

Arguments

`x`	A gretaR_array.
`i`	Index: an integer vector or a gretaR_array of integer indices.
`j`	Not supported. Supplying a column index raises an error; only single-index row/element selection is currently implemented.
`...`	Additional arguments (ignored).
`drop`	Logical (ignored; always returns a gretaR_array).

Value

A new gretaR_array with elements selected by the index.

`%*%` *Matrix multiplication with gretaR_array support*

Description

Overrides base `%*%` to dispatch to sparse-aware matrix multiplication when either argument is a `gretaR_array`.

Usage

```
x %*% y
```

Arguments

`x`	A matrix, gretaR_array, or numeric.
`y`	A matrix, gretaR_array, or numeric.

Value

A gretaR_array (if either input is gretaR_array) or the base result.

---

as_data *Wrap Observed Data as a gretaR Array*

Description

Convert numeric vectors, matrices, or arrays into a `gretaR_array` data node for use in model definitions. Data nodes are fixed (not sampled) during inference.

Usage

```
as_data(x)
```

Arguments

`x`	A numeric vector, matrix, or array of observed data.

Value

A `gretaR_array` representing fixed (observed) data.

Examples

```
## Not run:
y <- as_data(iris$Sepal.Length)
X <- as_data(matrix(rnorm(30), ncol = 3))

## End(Not run)
```

---

bernoulli *Bernoulli Distribution*

Description

Create a Bernoulli-distributed variable with the specified probability of success. Support is on `{0, 1}`.

Usage

```
bernoulli(prob, dim = NULL)
```

Arguments

`prob`	Probability of success (numeric or `gretaR_array`, 0 to 1).
`dim`	Integer vector of dimensions (default scalar).

Value

A gretaR_array.

Examples

```
## Not run:
z <- bernoulli(0.5)

## End(Not run)
```

---

beta_dist *Beta Distribution*

Description

Create a beta-distributed variable with the specified shape parameters. Support is on the interval (0, 1). Named beta_dist to avoid conflict with base::beta.

Usage

```
beta_dist(alpha, beta, dim = NULL, truncation = NULL)
```

Arguments

alpha	First shape parameter (positive numeric or gretaR_array).
beta	Second shape parameter (positive numeric or gretaR_array).
dim	Integer vector of dimensions (default scalar).
truncation	Optional length-2 numeric vector c(lower, upper) specifying truncation bounds. Default NULL (no truncation).

Value

A gretaR_array.

Examples

```
## Not run:
p <- beta_dist(alpha = 2, beta = 5)

## End(Not run)
```

binomial_dist *Binomial Distribution*

Description

Create a binomially-distributed variable with the specified number of trials and probability of success per trial.

Named `binomial_dist` to avoid masking `stats::binomial` (the GLM family constructor).

Usage

```
binomial_dist(size, prob, dim = NULL)
```

Arguments

`size`	Number of trials (positive integer or `gretaR_array`).
`prob`	Probability of success per trial (numeric or `gretaR_array`, 0 to 1).
`dim`	Integer vector of dimensions (default scalar).

Value

A `gretaR_array`.

Examples

```
## Not run:
y <- binomial_dist(size = 10, prob = 0.3)

## End(Not run)
```

cauchy *Cauchy Distribution*

Description

The Cauchy distribution (Student-t with 1 degree of freedom). Heavy-tailed; useful as a weakly informative prior.

Usage

```
cauchy(location = 0, scale = 1, dim = NULL, truncation = NULL)
```

Arguments

location	Location parameter.
scale	Scale parameter (positive).
dim	Dimensions.
truncation	Optional length-2 numeric vector c(lower, upper) specifying truncation bounds. Default NULL (no truncation).

Value

A gretaR_array.

Examples

```
## Not run:
x <- cauchy(0, 1)

## End(Not run)
```

---

coef.gretaR_fit *Extract Coefficients from a gretaR Fit*

Description

Extract posterior means (for MCMC/VI) or MAP estimates as a named numeric vector.

Usage

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

Arguments

object	A gretaR_fit object.
...	Ignored.

Value

A named numeric vector of point estimates.

Examples

```
## Not run:
mu <- normal(0, 10)
y <- as_data(rnorm(50, 3, 1))
distribution(y) <- normal(mu, 1)
m <- model(mu)
fit <- opt(m)
coef(fit)

## End(Not run)
```

`compile_to_stan` *Generate Stan Code from a gretaR Model*

Description

Translate a compiled `gretaR_model` into Stan code suitable for use with `cmdstanr`. The generated code preserves the model structure defined via the gretaR DSL.

Usage

```
compile_to_stan(model)
```

Arguments

`model`	A `gretaR_model` object created by `model()`.

Value

A character string containing valid Stan code.

Examples

```
## Not run:
mu <- normal(0, 10)
sigma <- half_cauchy(2)
y <- as_data(rnorm(50, 3, 1.5))
distribution(y) <- normal(mu, sigma)
m <- model(mu, sigma)
cat(compile_to_stan(m))

## End(Not run)
```

custom_distribution *Custom Distribution*

Description

Define a distribution with a user-supplied log-probability density function. The function must accept a torch tensor and return a scalar torch tensor. It must be differentiable via torch autograd.

Usage

```
custom_distribution(
  log_prob_fn,
  constraint = NULL,
  dim = NULL,
  name = "custom"
)
```

Arguments

log_prob_fn	A function `f(x) -> scalar torch tensor` computing the log-probability density at `x`.
constraint	Optional list with `lower` and `upper` bounds for the parameter space. Determines the transform for HMC sampling.
dim	Dimensions of the variable.
name	Optional name for display purposes.

Value

A `gretaR_array` representing a variable with the custom distribution.

Examples

```
## Not run:
# Laplace distribution (not built in)
x <- custom_distribution(
  log_prob_fn = function(x) -torch_sum(torch_abs(x)),
  name = "laplace"
)

# Truncated normal (positive only)
x <- custom_distribution(
  log_prob_fn = function(x) {
    torch_sum(-0.5 * x^2)  # kernel of N(0,1)
  },
  constraint = list(lower = 0, upper = Inf),
  name = "truncated_normal"
)

## End(Not run)
```

dirichlet *Dirichlet distribution*

Description

Creates a variable distributed according to the Dirichlet distribution, the multivariate generalisation of the Beta distribution. Values lie on the probability simplex (non-negative, sum to one).

Usage

```
dirichlet(concentration, dim = NULL)
```

Arguments

concentration	Concentration parameter vector (positive numeric or gretaR_array). Length determines the dimensionality of the simplex.
dim	Dimensions of the variable (inferred from concentration if NULL).

Value

A gretaR_array with support on the simplex.

Note

Latent (sampled) Dirichlet variables are not yet supported because a correct simplex transform (stick-breaking or centered softmax with the matching Jacobian) is not yet implemented. model() will error if a Dirichlet variable is reachable as a free parameter. The distribution can still be evaluated (log_prob) and is planned to be sampler-ready in v0.3.

Examples

```
## Not run:
theta <- dirichlet(c(1, 1, 1))
theta <- dirichlet(c(2, 5, 1))

## End(Not run)
```

distribution *Get the Distribution of a gretaR Array*

Description

Retrieve the distribution object associated with a gretaR_array variable node, or NULL if none is set.

Usage

```
distribution(x)
```

Arguments

x	A gretaR_array.

Value

The GretaRDistribution object, or NULL.

Examples

```
## Not run:
mu <- normal(0, 1)
distribution(mu)

## End(Not run)
```

---

distribution<- *Assign a Distribution (Likelihood) to Observed Data*

Description

Define the likelihood by assigning a distribution to a data gretaR_array. This registers the distribution as a likelihood term in the model's log-joint density.

Usage

```
distribution(x) <- value
```

Arguments

x	A data gretaR_array (created with as_data).
value	A distribution gretaR_array (e.g., from normal).

Value

The data gretaR_array x, invisibly.

Examples

```
## Not run:
y <- as_data(rnorm(100))
mu <- normal(0, 10)
sigma <- half_cauchy(1)
distribution(y) <- normal(mu, sigma)

## End(Not run)
```

exponential *Exponential Distribution*

Description

Create an exponentially-distributed variable with the specified rate. Support is on the positive reals.

Usage

```
exponential(rate = 1, dim = NULL, truncation = NULL)
```

Arguments

rate	Rate parameter (positive numeric or gretaR_array).
dim	Integer vector of dimensions (default scalar).
truncation	Optional length-2 numeric vector c(lower, upper) specifying truncation bounds. Default NULL (no truncation).

Value

A gretaR_array.

Examples

```
## Not run:
lambda <- exponential(rate = 1)

## End(Not run)
```

gamma_dist *Gamma Distribution*

Description

Create a gamma-distributed variable with the specified shape and rate. Support is on the positive reals. Named gamma_dist to avoid conflict with base::gamma.

Usage

```
gamma_dist(shape, rate, dim = NULL, truncation = NULL)
```

Arguments

shape	Shape parameter (positive numeric or gretaR_array).
rate	Rate parameter (positive numeric or gretaR_array).
dim	Integer vector of dimensions (default scalar).
truncation	Optional length-2 numeric vector c(lower, upper) specifying truncation bounds. Default NULL (no truncation).

Value

A gretaR_array.

Examples

```
## Not run:
tau <- gamma_dist(shape = 2, rate = 1)

## End(Not run)
```

---

gretaR_glm *Fit a Bayesian GLM Using Formula Syntax*

Description

[Experimental]

A high-level interface for specifying and fitting generalised linear models (including mixed / hierarchical models) using standard R formula syntax. Internally translates to the gretaR DSL, constructs the model, and runs MCMC inference.

lme4-style random effects are supported via regex parsing (lme4 is **not** required). Recognised patterns:

- (1|group) — random intercepts by group (supported)
- (0 + x|group) — independent random slopes, no intercept (supported)
- (x|group) — correlated random intercept + slope (**not yet supported**; correct support requires an LKJ correlation transform planned for v0.3)

As a workaround for correlated intercept + slope, split the term: (1|group) + (0 + x|group) fits both as independent random effects.

Multiple random effect terms are permitted, e.g. y ~ x + (1|site) + (1|year).

A non-centred parameterisation is used by default for superior HMC geometry.

Usage

```
gretaR_glm(
  formula,
  data,
  family = c("gaussian", "binomial", "poisson"),
  prior = NULL,
  sampler = c("nuts", "hmc", "vi", "map"),
  chains = 4L,
  iter = 2000L,
  warmup = NULL,
  formula_style = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

formula	A formula specifying the model (e.g., y ~ x1 + x2, or y ~ x + (1|group) for mixed models).
data	A data frame containing the variables referenced in the formula.
family	Distribution family: "gaussian" (default), "binomial", or "poisson".
prior	A named list of gretaR distribution objects for parameter priors. Recognised names: "beta" (regression coefficients), "intercept", "sigma" (residual SD, gaussian only), "tau" (random effect SD). Use NULL for default priors.
sampler	Sampler: "nuts" (default), "hmc", "vi", or "map".
chains	Number of MCMC chains (default 4).
iter	Total iterations per chain (warmup + samples, default 2000).
warmup	Number of warmup iterations (default half of iter).
formula_style	Optional explicit formula style hint: "base", "lme4", "brms", "mgcv". If NULL (default), auto-detected.
verbose	Logical; print progress (default TRUE).
...	Additional arguments passed to the sampler.

Value

A gretaR_glm_fit object with components:

draws Posterior draws (from MCMC or VI).

model The compiled gretaR_model.

formula The original formula.

family The family used.

data The original data.

design_matrix The model matrix (fixed effects).

col_names Named mapping of fixed-effect parameters.

random_effects List of parsed random effect specifications (NULL for base-style formulas).

Examples

```
## Not run:
# Gaussian linear model
fit <- gretaR_glm(Sepal.Length ~ Sepal.Width + Petal.Length,
                  data = iris, family = "gaussian")
summary(fit$draws)

# Logistic regression
dat <- data.frame(y = rbinom(100, 1, 0.6), x = rnorm(100))
fit <- gretaR_glm(y ~ x, data = dat, family = "binomial")

# Custom priors
fit <- gretaR_glm(y ~ x, data = dat, family = "gaussian",
                  prior = list(beta = normal(0, 1), sigma = half_cauchy(1)))

# Random intercepts model (lme4-style)
sleepstudy <- data.frame(
  Reaction = rnorm(180, 300, 50),
  Days = rep(0:9, each = 18),
  Subject = factor(rep(1:18, times = 10))
)
fit <- gretaR_glm(Reaction ~ Days + (1 | Subject),
                  data = sleepstudy, family = "gaussian")

# Random intercept + independent random slope (workaround for (Days|Subject))
fit <- gretaR_glm(Reaction ~ Days + (1 | Subject) + (0 + Days | Subject),
                  data = sleepstudy, family = "gaussian")

## End(Not run)
```

`half_cauchy` *Half-Cauchy Distribution*

Description

Create a half-Cauchy-distributed variable. A popular weakly informative prior for scale parameters (Gelman, 2006), with support on the positive reals.

Usage

```
half_cauchy(scale = 1, dim = NULL, truncation = NULL)
```

Arguments

`scale`	Scale parameter (positive numeric or `gretaR_array`).
`dim`	Integer vector of dimensions (default scalar).
`truncation`	Optional length-2 numeric vector `c(lower, upper)` specifying truncation bounds. Default `NULL` (no truncation).

Value

A gretaR_array with support on the positive reals.

Examples

```
## Not run:
sigma <- half_cauchy(1)
tau <- half_cauchy(5)

## End(Not run)
```

---

half_normal *Half-Normal Distribution*

Description

Create a half-normal-distributed variable. The half-normal is the absolute value of a normal distribution, with support on the positive reals.

Usage

```
half_normal(sd = 1, dim = NULL, truncation = NULL)
```

Arguments

sd	Scale parameter (positive numeric or gretaR_array).
dim	Integer vector of dimensions (default scalar).
truncation	Optional length-2 numeric vector c(lower, upper) specifying truncation bounds. Default NULL (no truncation).

Value

A gretaR_array with support on the positive reals.

Examples

```
## Not run:
sigma <- half_normal(1)

## End(Not run)
```

hmc *Run HMC Sampling*

Description

[Experimental]

Convenience wrapper around `mcmc` that selects the static Hamiltonian Monte Carlo sampler.

Usage

```
hmc(model, n_samples = 1000L, warmup = 1000L, chains = 4L, ...)
```

Arguments

`model`	A `gretaR_model` object created by `model`.
`n_samples`	Number of post-warmup samples per chain (default 1000).
`warmup`	Number of warmup (adaptation) iterations per chain (default 1000).
`chains`	Number of independent chains (default 4).
`...`	Additional arguments passed to `mcmc`.

Value

A `gretaR_fit` object.

Examples

```
## Not run:
m <- model(normal(0, 1))
fit <- hmc(m, n_samples = 500, warmup = 500)
coef(fit)

## End(Not run)
```

joint_density *Get the Log Joint Density Function*

Description

Extract a torch-compatible function that evaluates the log joint density of a compiled `gretaR_model` at a given parameter vector in unconstrained space.

Usage

```
joint_density(model)
```

Arguments

model	A gretaR_model object created by model.

Value

A function f(theta) that takes a 1-D torch tensor of unconstrained parameter values and returns a scalar torch tensor (the log joint density).

Examples

```
## Not run:
mu <- normal(0, 10)
y <- as_data(rnorm(50, 3))
distribution(y) <- normal(mu, 1)
m <- model(mu)
ld <- joint_density(m)
ld(torch::torch_zeros(1))

## End(Not run)
```

---

laplace *Laplace Approximation*

Description

[Experimental]

Approximate the posterior distribution using a multivariate normal centred at the MAP estimate with covariance equal to the inverse of the negative Hessian of the log-joint density.

Usage

```
laplace(model, map_fit = NULL, ...)
```

Arguments

model	A gretaR_model object.
map_fit	Optional output from opt(). If NULL, MAP is computed first.
...	Additional arguments passed to opt() if map_fit is NULL.

Value

A list with components:

mean Named numeric vector of posterior means (constrained).

mean_unconstrained Posterior means in unconstrained space.

covariance Posterior covariance matrix (unconstrained space).

sd Named numeric vector of posterior standard deviations (unconstrained).

log_evidence Approximate log marginal likelihood.

map The MAP fit used.

Examples

```
## Not run:
mu <- normal(0, 10)
y <- as_data(rnorm(50, 3, 1))
distribution(y) <- normal(mu, 1)
m <- model(mu)
la <- laplace(m)
la$mean
la$sd

## End(Not run)
```

`lkj_correlation` *LKJ Correlation distribution*

Description

The LKJ distribution over correlation matrices (Lewandowski, Kurowicka, and Joe, 2009). The density is proportional to $\det(R)^{\eta-1}$ where R is a correlation matrix. When $\eta = 1$, the distribution is uniform over valid correlation matrices.

Usage

```
lkj_correlation(eta = 1, dim = 2L)
```

Arguments

`eta`	Shape parameter (positive). Values > 1 concentrate mass around the identity matrix; values < 1 favour extreme correlations.
`dim`	Dimension of the correlation matrix (integer >= 2).

Value

A `gretaR_array` representing a correlation matrix.

Note

Latent (sampled) LKJ variables are not yet supported because a correct correlation-matrix transform (LKJ-Cholesky with the matching Jacobian) is not yet implemented. `model()` will error if an LKJ variable is reachable as a free parameter. The distribution can still be evaluated (`log_prob`); sampler-ready support is planned for v0.3.

Examples

```
## Not run:
R <- lkj_correlation(eta = 2, dim = 3)

## End(Not run)
```

lognormal *Log-Normal Distribution*

Description

A variable whose logarithm is normally distributed.

Usage

```
lognormal(meanlog = 0, sdlog = 1, dim = NULL, truncation = NULL)
```

Arguments

meanlog	Mean of the log-scale distribution.
sdlog	Standard deviation on the log scale (positive).
dim	Dimensions.
truncation	Optional length-2 numeric vector `c(lower, upper)` specifying truncation bounds. Default `NULL` (no truncation).

Value

A `gretaR_array` with support on the positive reals.

Examples

```
## Not run:
x <- lognormal(0, 1)

## End(Not run)
```

mcmc *Draw MCMC Samples from a gretaR Model*

Description

[Experimental]

Run Hamiltonian Monte Carlo or the No-U-Turn Sampler on a compiled `gretaR_model` and return posterior draws in a format compatible with the `posterior` and `bayesplot` packages.

Usage

```
mcmc(
  model,
  n_samples = 1000L,
  warmup = 1000L,
  chains = 4L,
  sampler = c("nuts", "hmc"),
  backend = c("torch", "stan"),
  step_size = NULL,
  max_treedepth = 10L,
  n_leapfrog = 25L,
  target_accept = NULL,
  metric = c("diag", "dense"),
  init_values = NULL,
  seed = NULL,
  batched = FALSE,
  trajectory = c("fixed", "chees"),
  device = "cpu",
  verbose = TRUE
)
```

Arguments

`model`	A `gretaR_model` object created by `model`.
`n_samples`	Number of post-warmup samples per chain (default 1000).
`warmup`	Number of warmup (adaptation) iterations per chain (default 1000).
`chains`	Number of independent chains (default 4).
`sampler`	Sampler to use: `"nuts"` (default) or `"hmc"`.
`backend`	Inference backend: `"torch"` (default, native R + torch) or `"stan"` (generates Stan code, compiles and runs via cmdstanr).
`step_size`	Initial step size for the leapfrog integrator. If `NULL` (default), automatically tuned during warmup.
`max_treedepth`	Maximum tree depth for NUTS (default 10).
`n_leapfrog`	Safety cap on leapfrog steps per static-HMC iteration (default 25). HMC integrates for a random time $T \sim U(0, 2\pi]$ and takes $\mathrm{round}(T/\epsilon)$ steps, capped at `10 * n_leapfrog`, to avoid the resonance that cripples fixed-length HMC.
`target_accept`	Target average acceptance probability (default 0.8 for NUTS, 0.65 for HMC).
`metric`	Mass-matrix metric for the single-chain samplers, estimated during warmup. `"diag"` (default) is a diagonal metric (inverse posterior variance) – the robust, Stan-default choice. `"dense"` is a dense metric (inverse posterior covariance) that captures linear parameter correlations a diagonal metric is blind to; it can substantially improve mixing on correlated posteriors (e.g. regression with correlated predictors), but it is opt-in because it does not help – and can hurt – funnel-shaped posteriors such as hierarchical latent blocks, and it adds $O(P^2)$ cost. Falls back to diagonal (with a message) when the dimension is too large

	or warmup draws are too few to estimate a covariance. Ignored on the batched path.
`init_values`	Optional list of initial parameter vectors (one per chain).
`seed`	Optional integer seed for reproducibility. Sets both R and torch random number generators.
`batched`	Logical (default `FALSE`). When `TRUE` (and `sampler = "hmc"`), all chains advance together as one set of batched `torch` tensor operations rather than chain-by-chain. Wall-clock is then roughly flat in the number of chains, so many-chain runs are much faster (e.g. ~2x at 8 chains, ~4x at 16 on CPU). Statistically equivalent to the single-chain HMC. Batched NUTS is not yet supported.
`trajectory`	Trajectory-length rule for the batched path (`batched = TRUE`). `"fixed"` (default) is integration-time HMC – a random time $T \sim U(0, 2\pi]$ per iteration. `"chees"` is ChEES-HMC (Hoffman, Radul & Sountsov 2021), which adapts the trajectory length during warmup using a criterion computed across the chain ensemble, so it batches where NUTS's per-chain tree recursion cannot. ChEES is opt-in and most useful in a specific regime: it needs a reasonably large ensemble (use **at least ~8 chains**; below that its criterion is noisy and it can mix worse than NUTS) and a well-conditioned posterior. On hierarchical models, pair it with a near-centred `random_effect` – in our tests ChEES on the non-centred funnel was no better than NUTS, but ChEES on a near-centred parameterisation gave several times the effective sample size per second of the NUTS default, the two acting together (the centring conditions the geometry, the adaptive trajectory then traverses it efficiently). Single- chain NUTS remains the robust default; `"chees"` is ignored unless `batched = TRUE`. Adaptive trajectory length means wall-clock is not flat in chain count as it is for `"fixed"`.
`device`	Character device for the batched path: `"cpu"` (default), `"mps"`, or `"cuda"`. The batched code is device-generic, but on typical models CPU is fastest – gretaR's log-density is many small ops, so GPU kernel-launch overhead dominates; GPU is for future large-model use.
`verbose`	Logical; print progress information (default `TRUE`).

Value

A `gretaR_fit` object with components:

draws Posterior draws as `posterior::draws_array`.

model The compiled `gretaR_model`.

summary Posterior summary table (mean, sd, quantiles, R-hat, ESS).

convergence List: `n_eff`, `rhat`, `max_rhat`, `min_ess`, `n_divergences`.

call_info List of sampling arguments for reproducibility.

run_time Elapsed seconds.

method `"nuts"` or `"hmc"`.

Use `coef()` for point estimates, `summary()` for full table, `plot()` for diagnostics.

Examples

```
## Not run:
# Simple normal model
mu <- normal(0, 10)
sigma <- half_cauchy(2)
y <- as_data(rnorm(50, 3, 1.5))
distribution(y) <- normal(mu, sigma)
m <- model(mu, sigma)
draws <- mcmc(m, n_samples = 500, warmup = 500)
summary(draws)

## End(Not run)
```

---

mean.gretaR_array *Mean of a gretaR_array*

Description

Build an operation node that reduces a gretaR_array to a scalar by averaging over all elements.

Usage

```
## S3 method for class 'gretaR_array'
mean(x, ...)
```

Arguments

x	A gretaR_array.
...	Ignored.

Value

A scalar gretaR_array.

---

mixture *Mixture Distribution*

Description

Define a finite mixture of distributions. The mixture is marginalised over the discrete component indicator using the log-sum-exp trick, enabling gradient-based inference (HMC/NUTS).

Usage

```
mixture(distributions, weights)
```

Arguments

distributions	A list of gretaR distribution objects (e.g., list(normal(mu1, sigma1), normal(mu2, sigma2))). Each must be a gretaR_array with a distribution attached.
weights	A gretaR_array or numeric vector of mixture weights (must sum to 1). Typically from dirichlet() or softmax().

Value

A distribution object suitable for use with distribution().

Examples

```
## Not run:
# Two-component Gaussian mixture
w <- dirichlet(c(1, 1))
mu1 <- normal(-2, 1); mu2 <- normal(2, 1)
sigma <- half_cauchy(1)

mix <- mixture(
  distributions = list(normal(mu1, sigma), normal(mu2, sigma)),
  weights = w
)
y <- as_data(rnorm(100))
distribution(y) <- mix
m <- model(w, mu1, mu2, sigma)

## End(Not run)
```

---

model *Create a gretaR Model*

Description

Compile the computation graph defined by the specified target variables into a differentiable log-joint-density function suitable for HMC/NUTS inference.

Usage

```
model(..., precision = c("float32", "float64"))
```

Arguments

...	gretaR_array objects representing the parameters of interest.
precision	Torch dtype: "float32" (default) or "float64".

Value

A gretaR_model object with log_prob() and grad_log_prob() methods.

Examples

```
## Not run:
alpha <- normal(0, 10)
beta <- normal(0, 5)
sigma <- half_cauchy(1)
y <- as_data(rnorm(100))
x <- as_data(rnorm(100))
mu <- alpha + beta * x
distribution(y) <- normal(mu, sigma)
m <- model(alpha, beta, sigma)

## End(Not run)
```

`model_from_arrays` *Build a gretaR Model from Explicit Arrays (Programmatic Interface)*

Description

Compile a computation graph into a `gretaR_model` the same way `model` does, but with target parameters, their names, and the likelihood passed explicitly rather than recovered from the call. This makes model construction safe to perform programmatically – from `do.call()`, a code generator, or another package building gretaR models from its own representation – and re-entrant, since the likelihood is scoped to the supplied data nodes instead of the whole session.

Usage

```
model_from_arrays(
  targets,
  likelihood = NULL,
  names = NULL,
  precision = c("float32", "float64")
)
```

Arguments

`targets`	A list of `gretaR_array` variable nodes – the free parameters of interest. These come first in the parameter vector, in the order given. A single `gretaR_array` is accepted and wrapped.
`likelihood`	Optional. The data `gretaR_array`(s) whose assigned likelihood should enter this model, as a single array or a list of arrays. Each must already have a distribution attached via `distribution(y) <- ...`. When `NULL` (default), the session-global likelihood registry is used, reproducing `model`'s behaviour.
`names`	Optional parameter names, one entry per element of `targets`. A character vector gives one label per target (a vector target expands to `label[1]`, `label[2]`, ...). A list gives finer control: each entry is either a single label or a character vector of per-element names whose length matches that target's number of elements – so a length-p coefficient vector can be labelled `c("(Intercept)", "x1",`

`...`) directly, and those labels flow straight through to the posterior draws. When `NULL`, the names of `targets` are used if it is a named list; otherwise each target falls back to its internal node id.

`precision` Torch dtype: `"float32"` (default) or `"float64"`.

Details

`model(alpha, beta, sigma)` reads the names `"alpha"`, `"beta"`, `"sigma"` by deparsing its call. A programmatic caller has no stable way to reproduce that, and an interactive one cannot override it. `model_from_arrays()` takes the targets as a list and their names as a character vector (or as the names of the list), so the parameter labels are whatever the caller chooses – the hook a host package needs to map gretaR's internal names onto its own canonical scheme.

`model()` additionally folds in *every* `distribution(y) <- ...` assignment registered in the session, so two models built in the same session share each other's likelihood terms unless the environment is reset between them. `model_from_arrays()` instead includes only the likelihood of the data nodes named in `likelihood`, leaving the global registry untouched. Independent models can therefore be compiled in sequence without interference.

The two front-ends delegate to the same internal compiler, so a model built with `model_from_arrays()` is indistinguishable from the equivalent `model()` call and is accepted everywhere a `gretaR_model` is – `mcmc`, `opt`, `joint_density`.

Value

A `gretaR_model` object, identical in structure to one produced by `model`.

See Also

`model` for the interactive front-end.

Examples

```
## Not run:
# Programmatic construction — names supplied, not deparsed.
mu <- normal(0, 10)
sigma <- half_cauchy(1)
y <- as_data(rnorm(100, 3))
distribution(y) <- normal(mu, sigma)
m <- model_from_arrays(
  targets = list(mu, sigma),
  likelihood = y,
  names = c("mu", "sigma")
)

# Re-entrant: a second, independent model in the same session.
theta <- normal(0, 1)
z <- as_data(rnorm(50))
distribution(z) <- normal(theta, 1)
m2 <- model_from_arrays(targets = list(theta), likelihood = z, names = "theta")

## End(Not run)
```

multivariate_normal *Multivariate Normal Distribution*

Description

Create a multivariate-normal-distributed variable with the specified mean vector and covariance matrix. If `dim` is `NULL`, it is inferred from the length of `mean`.

Usage

```
multivariate_normal(mean, covariance, dim = NULL)
```

Arguments

`mean`	Numeric mean vector or `gretaR_array`.
`covariance`	Covariance matrix (positive definite numeric matrix or `gretaR_array`).
`dim`	Integer vector of dimensions (inferred from `mean` if `NULL`).

Value

A `gretaR_array`.

Examples

```
## Not run:
mu <- c(0, 0)
Sigma <- diag(2)
x <- multivariate_normal(mean = mu, covariance = Sigma)

## End(Not run)
```

negative_binomial *Negative Binomial distribution*

Description

Parameterised by the number of successes `size` (r) and the probability of success `prob` (p). Models the number of failures before `size` successes occur.

Usage

```
negative_binomial(size, prob, dim = NULL)
```

Arguments

size	Target number of successes (positive numeric or gretaR_array).
prob	Probability of success per trial (0 to 1).
dim	Dimensions.

Value

A gretaR_array (discrete, non-negative integers).

Examples

```
## Not run:
y <- negative_binomial(size = 5, prob = 0.5)

## End(Not run)
```

---

normal *Normal Distribution*

Description

Create a normally-distributed variable with the specified mean and standard deviation. Support is on the entire real line.

Usage

```
normal(mean = 0, sd = 1, dim = NULL, truncation = NULL)
```

Arguments

mean	Mean of the distribution (numeric or gretaR_array).
sd	Standard deviation (numeric or gretaR_array, positive).
dim	Integer vector of dimensions (default scalar).
truncation	Optional length-2 numeric vector c(lower, upper) specifying truncation bounds. Default NULL (no truncation). Compatible with greta's truncation syntax.

Value

A gretaR_array representing a normally-distributed variable.

Examples

```
## Not run:
x <- normal(0, 1)
beta <- normal(0, 5, dim = c(3, 1))
x_pos <- normal(0, 1, truncation = c(0, Inf))

## End(Not run)
```

nuts *Run NUTS Sampling*

Description

[Experimental]

Convenience wrapper around `mcmc` that selects the No-U-Turn Sampler (NUTS).

Usage

```
nuts(model, n_samples = 1000L, warmup = 1000L, chains = 4L, ...)
```

Arguments

`model`	A `gretaR_model` object created by `model`.
`n_samples`	Number of post-warmup samples per chain (default 1000).
`warmup`	Number of warmup (adaptation) iterations per chain (default 1000).
`chains`	Number of independent chains (default 4).
`...`	Additional arguments passed to `mcmc`.

Value

A `gretaR_fit` object.

Examples

```
## Not run:
m <- model(normal(0, 1))
fit <- nuts(m, n_samples = 500, warmup = 500)
coef(fit)

## End(Not run)
```

opt *Find the Maximum A Posteriori (MAP) Estimate*

Description

[Experimental]

Optimise the log-joint density using gradient-based methods to find the posterior mode. Uses the Adam optimiser via torch.

Usage

```
opt(
  model,
  max_iter = 2000L,
  learning_rate = 0.01,
  tolerance = 1e-06,
  init = NULL,
  verbose = TRUE,
  backend = c("torch", "stan"),
  seed = NULL
)
```

Arguments

model	A gretaR_model object created by model().
max_iter	Maximum number of optimisation iterations (default 2000).
learning_rate	Adam learning rate (default 0.01).
tolerance	Convergence tolerance on relative change in log-prob (default 1e-6).
init	Optional initial values (numeric vector in unconstrained space).
verbose	Logical; print progress (default TRUE).
backend	Inference backend: "torch" (default) or "stan".
seed	Optional integer seed for reproducibility.

Value

A gretaR_fit object (method = "map") with components:

par Named numeric vector of MAP estimates (constrained space).

par_unconstrained Numeric vector of MAP in unconstrained space.

log_prob Log-joint density at the MAP.

convergence List with convergence info.

iterations Number of iterations used.

Examples

```
## Not run:
mu <- normal(0, 10)
sigma <- half_cauchy(2)
y <- as_data(rnorm(50, 3, 1.5))
distribution(y) <- normal(mu, sigma)
m <- model(mu, sigma)
fit <- opt(m)
fit$par

## End(Not run)
```

parse_re_bars *Parse random effects bar terms from an lme4-style formula*

Description

Extracts all `(expr | group)` terms and classifies them as `"intercept"`, `"slope_only"`, or `"intercept_slope"`.

Usage

```
parse_re_bars(formula)
```

Arguments

`formula`	A formula potentially containing bar terms.

Value

A list of parsed random effect specifications. Each element is a list with components: `raw` (original text), `lhs` (left of bar), `group` (grouping variable), `type` (one of `"intercept"`, `"slope_only"`, `"intercept_slope"`), and `slope_vars` (character vector of slope variable names, may be empty).

Examples

```
parse_re_bars(y ~ x + (1 | group))
parse_re_bars(y ~ x + (x | group))
parse_re_bars(y ~ x + (0 + x | group))
parse_re_bars(y ~ x + (1 | site) + (1 | year))
```

plot.gretaR_fit *Plot Diagnostics for a gretaR Fit*

Description

Generate diagnostic plots for MCMC or VI posterior draws. Requires the `bayesplot` package.

Usage

```
## S3 method for class 'gretaR_fit'
plot(x, type = c("trace", "density", "pairs", "rhat", "neff"), ...)
```

Arguments

`x`	A `gretaR_fit` object.
`type`	Plot type: `"trace"` (default), `"density"`, `"pairs"`, `"rhat"`, `"neff"`.
`...`	Additional arguments passed to the bayesplot function.

Value

A ggplot object.

Examples

```
## Not run:
mu <- normal(0, 10)
y <- as_data(rnorm(50, 3, 1))
distribution(y) <- normal(mu, 1)
m <- model(mu)
fit <- mcmc(m, n_samples = 200, warmup = 200, chains = 2)
plot(fit, type = "trace")

## End(Not run)
```

---

poisson_dist *Poisson Distribution*

Description

Create a Poisson-distributed variable with the specified rate. Named `poisson_dist` to avoid conflict with `stats::poisson`.

Usage

```
poisson_dist(rate, dim = NULL)
```

Arguments

`rate`	Rate parameter (positive numeric or `gretaR_array`).
`dim`	Integer vector of dimensions (default scalar).

Value

A `gretaR_array`.

Examples

```
## Not run:
y <- poisson_dist(rate = 5)

## End(Not run)
```

print.gretaR_fit *Print a gretaR Fit Object*

Description

Display a concise summary of the model fit including method, parameter count, convergence diagnostics, and timing.

Usage

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

Arguments

x	A gretaR_fit object.
...	Ignored.

Value

x, invisibly.

process_smooths *Process Smooth Terms from an mgcv-Style Formula*

Description

Extract and construct smooth terms from a formula containing s(), te(), ti(), or t2() terms. Requires the mgcv package.

Uses the smooth2random decomposition (same approach as brms) to convert penalised smooth bases into fixed + random effect components suitable for HMC/NUTS sampling.

Usage

```
process_smooths(formula, data, knots = NULL)
```

Arguments

formula	A formula potentially containing smooth terms.
data	A data frame with the covariates.
knots	Optional named list of knot locations.

Value

A list with components:

fixed_formula The formula with smooth terms removed (parametric part).

smooth_Xf Combined fixed-effects design matrix from all smooths (n x p_fixed).

smooth_Zs List of random-effects design matrices (each n x p_j).

smooth_info List of metadata for each smooth (label, type, rank, etc.).

n_smooth_fixed Number of smooth fixed-effect columns.

n_smooth_random List of dimensions per random-effect block.

Examples

```
## Not run:
library(mgcv)
dat <- data.frame(y = rnorm(100), x = rnorm(100))
sm <- process_smooths(y ~ s(x, k = 10), data = dat)
str(sm)

## End(Not run)
```

---

random_effect *Partially-Centred Random-Effect Block*

Description

Construct a grouped random-effect term with an explicit centring weight that interpolates between the non-centred and centred parameterisations. The weight controls only the sampling geometry seen by the HMC/NUTS sampler – the implied prior on the group effects, $u_j \sim N(\text{mean}, \text{sd}^2)$, is identical for every weight – so it is a pure efficiency control for hierarchical models whose mixing is bottlenecked by the funnel coupling between a group-scale parameter and its latent effects.

Usage

```
random_effect(group, n_groups, sd, centring = 0, mean = 0)
```

Arguments

group	Integer vector of length n mapping each observation to a group in `seq_len(n_groups)`, or the equivalent data `gretaR_array`. Used to gather the group effects to the observations.
n_groups	Number of groups J (a positive integer).
sd	A scalar `gretaR_array` – the random-effect standard deviation (commonly a `half_normal` or `half_cauchy` group-scale parameter). The funnel partner of the latent block.

`centring` The centring weight w. Either a single number in $[0, 1]$ applied to every group, or a numeric vector of length `n_groups` giving a per-group weight. Defaults to `0` (non-centred), gretaR's historical hierarchical parameterisation.

`mean` The random-effect mean, a numeric scalar or scalar `gretaR_array`. Defaults to `0`: the standard zero-mean deviation block whose global level lives in a separate intercept term.

Details

A centring weight `w = 0` gives the non-centred form $u_j = \text{mean} + \text{sd}\,\xi_j$ with $\xi_j \sim N(0, 1)$, which mixes well when groups are weakly informed. A weight `w = 1` gives the centred form $u_j = \text{mean} + \xi_j$ with $\xi_j \sim N(0, \text{sd}^2)$, which mixes well when groups are strongly informed. Intermediate and per-group weights interpolate:

$$\xi_j \sim N(0, \text{sd}^{w_j}), \qquad u_j = \text{mean} + \text{sd}^{\,1-w_j}\,\xi_j,$$

which leaves $u_j \sim N(\text{mean}, \text{sd}^2)$ unchanged. The informativeness-optimal weight for a balanced group of n_j observations with residual scale σ is $w_j^\star = (n_j/\sigma^2)/(n_j/\sigma^2 + 1/\text{sd}^2)$; a data-adaptive sampler estimates it during warmup, but a fixed value may be supplied directly here.

The returned latent block `$latent` is the free parameter the sampler moves and must be passed to `model_from_arrays` (or `model`); the returned `$effect` is the per-observation effect to add into the linear predictor. The latent block carries the group-level names, not the observation-level ones.

Value

A list of class `gretaR_random_effect` with elements

`latent` The J-element latent `gretaR_array` to include in the model's target list.

`effect` The n-element per-observation effect `gretaR_array` for the linear predictor.

`centring` The resolved per-group weight vector.

`n_groups` The number of groups.

See Also

`model_from_arrays` for the programmatic model front-end the latent block is designed for.

Examples

```
## Not run:
# Random-intercept model, near-centred (informative groups).
n <- 1000L
J <- 20L
g <- rep(seq_len(J), length.out = n)
x <- rnorm(n)
y <- as_data(1.5 - 0.8 * x + rnorm(n))

b0 <- normal(0, 5)
b1 <- normal(0, 5)
s <- half_normal(5)
tau <- half_normal(2)
```

```
re <- random_effect(g, n_groups = J, sd = tau, centring = 0.9)
eta <- b0 + b1 * as_data(x) + re$effect
distribution(y) <- normal(eta, s)

m <- model_from_arrays(
  list(b0, b1, s, tau, re$latent),
  likelihood = y,
  names = list("b0", "b1", "s", "tau", paste0("u[", seq_len(J), "]"))
)

## End(Not run)
```

`remove_re_bars` *Remove random effects bar terms from a formula*

Description

Returns the fixed-effects-only formula by stripping all `(expr|group)` terms. If **reformulas** is installed, delegates to `reformulas::nobars()`; otherwise falls back to `lme4::nobars()`, and finally to regex substitution.

Usage

```
remove_re_bars(formula)
```

Arguments

`formula`	A formula potentially containing bar terms.

Value

A formula with bar terms removed.

Examples

```
remove_re_bars(y ~ x + (1 | group))
# y ~ x
```

reset_gretaR_env *Reset the gretaR Global Environment*

Description

Clear all tracked nodes, distributions, and reset the node counter. Called automatically on package load; call manually to start a fresh model definition.

Usage

```
reset_gretaR_env()
```

Value

Invisible NULL.

Examples

```
## Not run:
reset_gretaR_env()

## End(Not run)
```

student_t *Student-t Distribution*

Description

Create a Student-t-distributed variable with the specified degrees of freedom, location, and scale. Useful as a robust alternative to the normal distribution.

Usage

```
student_t(df = 3, mu = 0, sigma = 1, dim = NULL, truncation = NULL)
```

Arguments

df	Degrees of freedom (positive numeric or gretaR_array).
mu	Location parameter (numeric or gretaR_array).
sigma	Scale parameter (positive numeric or gretaR_array).
dim	Integer vector of dimensions (default scalar).
truncation	Optional length-2 numeric vector c(lower, upper) specifying truncation bounds. Default NULL (no truncation).

Value

A gretaR_array.

Examples

```
## Not run:
x <- student_t(df = 3, mu = 0, sigma = 1)

## End(Not run)
```

---

sum.gretaR_array *Sum of a gretaR_array*

Description

Build an operation node that reduces a gretaR_array to a scalar by summing over all elements.

Usage

```
## S3 method for class 'gretaR_array'
sum(x, ..., na.rm = FALSE)
```

Arguments

x	A gretaR_array.
...	Ignored.
na.rm	Ignored (no NAs in torch).

Value

A scalar gretaR_array.

---

summary.gretaR_fit *Summarise a gretaR Fit Object*

Description

Compute or display detailed posterior summary statistics and convergence diagnostics.

Usage

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

Arguments

object	A gretaR_fit object.
...	Additional arguments passed to posterior::summarise_draws().

Value

A data frame of posterior summaries (from posterior::summarise_draws()), or a list for MAP/Laplace fits.

Examples

```
## Not run:
mu <- normal(0, 10)
y <- as_data(rnorm(50, 3, 1))
distribution(y) <- normal(mu, 1)
m <- model(mu)
fit <- mcmc(m, n_samples = 200, warmup = 200, chains = 2)
summary(fit)

## End(Not run)
```

---

uniform *Uniform Distribution*

Description

Create a uniformly-distributed variable on the interval [lower, upper].

Usage

```
uniform(lower = 0, upper = 1, dim = NULL)
```

Arguments

lower	Lower bound (numeric).
upper	Upper bound (numeric).
dim	Integer vector of dimensions (default scalar).

Value

A gretaR_array.

Examples

```
## Not run:
p <- uniform(0, 1)

## End(Not run)
```

variable *Create a Free Variable*

Description

Create an unconstrained or constrained variable without a distributional prior. Useful for deterministic transformations or parameters that do not require a prior.

Usage

```
variable(lower = -Inf, upper = Inf, dim = NULL)
```

Arguments

lower	Lower bound (default -Inf, unconstrained).
upper	Upper bound (default Inf, unconstrained).
dim	Integer vector of dimensions (default c(1, 1)).

Value

A gretaR_array variable node.

Examples

```
## Not run:
x <- variable()
x_pos <- variable(lower = 0)
x_bounded <- variable(lower = 0, upper = 1)

## End(Not run)
```

variational *Variational Inference (ADVI)*

Description

[Experimental]

Fit a model using Automatic Differentiation Variational Inference. Approximates the posterior with a multivariate Gaussian in unconstrained parameter space.

Usage

```
variational(
  model,
  n_samples = 1L,
  max_iter = 5000L,
  learning_rate = 0.01,
  tolerance = 1e-04,
  method = c("meanfield", "fullrank"),
  init_from_map = TRUE,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

model	A gretaR_model object created by model().
n_samples	Number of Monte Carlo samples per ELBO gradient estimate (default 1).
max_iter	Maximum number of optimisation iterations (default 5000).
learning_rate	Adam learning rate (default 0.01).
tolerance	Convergence tolerance on relative ELBO change (default 1e-4).
method	Variational family: "meanfield" (default) or "fullrank".
init_from_map	Logical; initialise from MAP estimate (default TRUE).
seed	Optional integer seed for reproducibility. Sets both R and torch random number generators.
verbose	Logical; print progress (default TRUE).

Value

A gretaR_fit object (method = "vi") with standard fields plus:

par Named vector of posterior means (constrained space).

sd Named vector of posterior SDs (unconstrained space).

covariance Posterior covariance matrix (unconstrained, fullrank only).

elbo Vector of ELBO values per iteration.

draws A posterior::draws_array of samples from the variational posterior.

converged Logical; did the optimiser converge?

Examples

```
## Not run:
mu <- normal(0, 10)
sigma <- half_cauchy(2)
y <- as_data(rnorm(50, 3, 1.5))
distribution(y) <- normal(mu, sigma)
m <- model(mu, sigma)
```

```
fit <- variational(m)
coef(fit)

## End(Not run)
```

`wishart` *Wishart Distribution*

Description

The Wishart distribution over positive-definite matrices. Used as a prior for covariance or precision matrices.

Usage

```
wishart(df, scale_matrix)
```

Arguments

`df`	Degrees of freedom (must be >= dimension of scale matrix).
`scale_matrix`	Scale matrix (positive definite, p x p).

Value

A `gretaR_array` representing a positive-definite matrix.

Note

Latent (sampled) Wishart variables are not yet supported because a correct positive-definite-matrix transform (Cholesky factor with positive diagonal plus the matching Jacobian) is not yet implemented. `model()` will error if a Wishart variable is reachable as a free parameter. The distribution can still be evaluated (`log_prob`); sampler-ready support is planned for v0.3.

Examples

```
## Not run:
Sigma <- wishart(df = 5, scale_matrix = diag(3))

## End(Not run)
```

Index

[.gretaR_array, 3
%*%, 3

as_data, 4, *11*

bernoulli, 4
beta_dist, 5
binomial_dist, 6

cauchy, 6
coef.gretaR_fit, 7
compile_to_stan, 8
custom_distribution, 9

dirichlet, 10
distribution, 10
distribution<-, 11

exponential, 12

gamma_dist, 12
gretaR_glm, 13

half_cauchy, 15, *34*
half_normal, 16, *34*
hmc, 17

joint_density, 17, *26*

laplace, 18
lkj_correlation, 19
lognormal, 20

mcmc, *17*, 20, *26*, *29*
mean.gretaR_array, 23
mixture, 23
model, *17, 18*, *21*, 24, *25, 26*, *29*, *35*
model(), *8*, *30*, *41*
model_from_arrays, 25, *35*
multivariate_normal, 27

negative_binomial, 27
normal, *11*, 28
nuts, 29

opt, *26*, 29
opt(), *18*

parse_re_bars, 31
plot.gretaR_fit, 31
poisson_dist, 32
print.gretaR_fit, 33
process_smooths, 33

random_effect, *22*, 34
remove_re_bars, 36
reset_gretaR_env, 37

student_t, 37
sum.gretaR_array, 38
summary.gretaR_fit, 38

uniform, 39

variable, 40
variational, 40

wishart, 42