Package: glmbayesCore (via r-universe)

June 12, 2026

Type Package

Title Core C++ Sampling Engine for glmbayes

Version 0.1.0

Date 2026-06-03

Description Core C++ engine for glmbayes: envelope-based iid samplers, Gibbs building blocks, and optional OpenCL. Developer backend for glmbayes, lmebayes, and extensions. End users should use glmbayes for lm/glm modelling.

License GPL-2

URL https://github.com/knygren/glmbayesCore

BugReports https://github.com/knygren/glmbayesCore/issues

Imports stats, Rcpp (>= 1.1.1), RcppParallel, Rdpack (>= 0.11-0), opencltools (>= 0.8.1), nmathopencl

RdMacros Rdpack

LinkingTo Rcpp, RcppArmadillo, RcppParallel

Depends MASS, R (>= 3.5.0)

Suggests glmbayes, coda, ggplot2, bayesrules, bayestestR, LearnBayes, testthat (>= 3.0.0), spelling, knitr, rmarkdown

VignetteBuilder knitr

SystemRequirements Optional OpenCL support. If available, GPU acceleration will be used; otherwise, computation runs on CPU.

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/testthat/edition 3

LazyData true

Language en-US

Config/pak/sysreqs make ocl-icd-opencl-dev

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

Date/Publication 2026-06-12 05:14:39 UTC

RemoteUrl https://github.com/knygren/glmbayesCore

RemoteRef HEAD

RemoteSha 3df73962ba3755f5106c460dea52dd15c6d22838

Contents

`glmbayesCore-package` *glmbayesCore: Core C++ Sampling Engine for glmbayes*

Description

Core C++ engine for envelope-based iid samplers, Gibbs building blocks, and optional OpenCL acceleration. Developer backend for **glmbayes**, **lmebayes**, and related extensions. End users should install **glmbayes** for formula-based modelling and S3 methods.

Details

Low-level entry points include envelope construction (`EnvelopeBuild`, `EnvelopeOrchestrator`), registered simulation pipelines (`rNormalGLM_std`, `rIndepNormalGammaReg_std`), matrix-input samplers (`rglmb`, `rlmb`), and OpenCL kernel loaders. Formula interfaces `glmb` and `lmb` live in **glmbayes**; **glmbayesCore** supplies the sampling engine.

OpenCL startup checks

In interactive sessions, attaching the package with `library(glmbayesCore)` may emit a short `packageStartupMessage` when `glmbayesCore_has_opencl()` is `FALSE` but a GPU or OpenCL stack appears available on the host. Set `options(glmbayes.quiet_opencl_startup = TRUE)` to suppress attach notes (recommended for CI and `R CMD check`).

Author(s)

Maintainer: Kjell Nygren <kjell.a.nygren@gmail.com>

Other contributors:

- The R Core Team (R Mathlib sources, R stats modeling code, and derived/adapted routines) [contributor, copyright holder]
- The R Foundation (Portions of R Mathlib and R source code) [copyright holder]
- Ross Ihaka (R Mathlib and original R modeling infrastructure) [contributor, copyright holder]
- Robert Gentleman (Portions of R Mathlib) [contributor, copyright holder]

- Simon Davies (Original R glm implementation) [contributor]
- Morten Welinder (Portions of R Mathlib) [contributor, copyright holder]
- Martin Maechler (Portions of R Mathlib) [contributor]

References

There are no references for Rd macro `\insertAllCites` on this help page.

See Also

glmbayes for the end-user modelling package.

`AMI` *Amitriptyline overdose data*

Description

Data with information on 17 overdoses of the drug amitriptyline

Usage

```
data(AMI)
```

Format

This data frame contains the following columns:

`TOT` total TCAD plasma level

`AMI` amount of amitriptyline present in the TCAD plasma level

`GEN` gender (male = 0, female = 1)

`AMT` amount of drug taken at time of overdose

`PR` PR wave measurement

`DIAP` diastolic blood pressure

`QRS` QRS wave measurement

Details

Each row is one overdose episode. Variables include total tricyclic antidepressant level, amitriptyline component, gender, reported amount ingested, and ECG-related measures (PR interval, QRS duration, diastolic blood pressure). The dataset is used in package examples for binomial and related regression; see (Dobson 1990) for analogous generalized linear modelling of clinical outcomes.

References

Dobson A~J (1990). *An Introduction to Generalized Linear Models*. Chapman and Hall, London.

Examples

```
############################## Start of AMI dataset example ###################

data(AMI)
summary(AMI)

###############################################################################
## End of AMI dataset example
###############################################################################
```

`BikeSharing` *Bike Sharing Dataset (Processed)*

Description

A processed version of the UCI Bike Sharing Dataset (hourly data). The data include derived variables for part of day, quarter, and Fourier terms for cyclic effects of hour and month.

Usage

```
BikeSharing
```

Format

A data frame with 17,379 observations and 25 variables (original plus derived):

instant Record index.

dteday Date.

season Season (1: spring, 2: summer, 3: fall, 4: winter).

yr Year (0: 2011, 1: 2012).

mnth Month (1–12).

hr Hour (0–23).

holiday Whether the day is a holiday (0/1).

weekday Day of week (0–6).

workingday Working day (0/1).

weathersit Weather situation (1–4).

temp Normalized temperature.

atemp Normalized feeling temperature.

hum Normalized humidity.

windspeed Normalized wind speed.

casual Count of casual users.

registered Count of registered users.

cnt Total count (casual + registered).

hr_num Hour as numeric (same as hr).

month_num Month as integer.

part_of_day Factor: Night (0–5h), Morning (6–11h), Afternoon (12–17h), Evening (18–23h).

quarter Factor: Q1–Q4.

hr_sin Sine term for 24-hour cycle.

hr_cos Cosine term for 24-hour cycle.

mon_sin Sine term for 12-month cycle.

mon_cos Cosine term for 12-month cycle.

Source

UCI Machine Learning Repository: Bike Sharing Dataset. https://archive.ics.uci.edu/dataset/275/bike+sharing+dataset

Examples

```
############################## Start of BikeSharing dataset example ###################

data("BikeSharing")
head(BikeSharing)
dim(BikeSharing)

###########################################################################
## End of BikeSharing dataset example
###########################################################################
```

---

block_rNormalGLM_update

One Gibbs block update via block_rNormalGLM

Description

Draw a single blockwise GLM posterior sample (n = 1) and return latent-level draws (e.g. updated theta) for two-block Gibbs samplers where each observation or group is its own block with scalar (or vector) coefficients per block.

Draw a single blockwise Gaussian posterior sample (n = 1) and return coefficient draws for two-block Gibbs samplers where each group is its own block (e.g. school-level random effects b_j).

Draw from blockwise full conditionals when the posterior factorizes across observation blocks. block_rNormalReg uses .block_rNormalReg_cpp() (Gaussian; each block calls rNormalReg). block_rNormalGLM uses .block_rNormalGLM_cpp() (GLM envelope; C++ partition and prior payload; each block calls rNormalGLM). Typical use is **block Gibbs** (n = 1 per outer step); n > 1 gives iid draws from the product conditional.

Usage

```
block_rNormalGLM_update(
  mu_all,
  sigma_theta_sq = NULL,
  y,
  x,
  block,
  family = poisson(),
  prior_list = NULL,
  prior_lists = NULL,
  offset = NULL,
  weights = 1,
  Gridtype = 2L,
  n_envopt = 1L,
  use_parallel = TRUE,
  use_opencl = FALSE,
  verbose = FALSE,
  progbar = FALSE,
  seed = NULL,
  theta_coef_col = 1L
)

block_rNormalReg_update(
  mu_all,
  P = NULL,
  dispersion = NULL,
  y,
  x,
  block,
  prior_list = NULL,
  prior_lists = NULL,
  offset = NULL,
  weights = 1,
  Gridtype = 2L,
  seed = NULL,
  coef_cols = NULL
)

block_rNormalGLM(
  n,
  y,
  x,
  block,
  prior_list = NULL,
  prior_lists = NULL,
  offset = NULL,
  weights = 1,
  family = gaussian(),
```

```
  Gridtype = 2L,
  n_envopt = NULL,
  use_parallel = TRUE,
  use_opencl = FALSE,
  verbose = FALSE,
  progbar = FALSE
)

block_rNormalReg(
  n,
  y,
  x,
  block,
  prior_list = NULL,
  prior_lists = NULL,
  offset = NULL,
  weights = 1,
  Gridtype = 2L
)
```

Arguments

`mu_all`	Numeric matrix `l1` x `k` or vector of length `l1` (recycled to all blocks): prior means per block (e.g. $X_{\text{hyper}}\gamma$). Required unless `prior_lists` or `prior_list` is supplied.
`sigma_theta_sq`	Shared prior variance for scalar blocks when building `prior_lists` from `mu_all`.
`y`	Response vector of length `nrow(x)`.
`x`	Design matrix `nrow(x)` by `ncol(x)`; same `ncol` in every block.
`block`	Block partition: `factor`/integer length `l2`, `l2_blocks` counts summing to `l2`, or list of row index vectors.
`family`	GLM `family` (not `gaussian()`).
`prior_list`	Single prior specification recycled to all blocks, or with `mu` as `l1` by `k` matrix or `blocks` sublist.
`prior_lists`	Optional list of length `k` (or 1) of per-block `prior_list` objects.
`offset`	Optional numeric vector (length 1 or `length(y)`); partitioned across blocks like `y`.
`weights`	Optional weights; same recycling and blocking as `offset`.
`Gridtype`	Passed to each block's sampler (Armadillo Gridtype).
`n_envopt`	Passed to each block; defaults to 1 when `NULL`.
`use_parallel`, `use_opencl`, `verbose`, `progbar`	Passed to each block's GLM sampler.
`seed`	Optional; passed to `set.seed` before sampling.
`theta_coef_col`	Column index of `coefficients` to return as `theta` (default 1 for scalar intercept blocks).

P	Prior precision matrix l1 x l1 (inverse of Σ_{ranef}). Shared across all blocks when building per-block priors from mu_all.
dispersion	Residual variance σ^2. Shared across all blocks when building per-block priors from mu_all.
coef_cols	Column indices of coefficients to return as b_draws (default NULL returns all columns).
n	Number of iid draws per block (n = 1 typical for Gibbs).

Details

This is a thin wrapper around block_rNormalGLM that always uses n = 1. When prior_lists and prior_list are both omitted, per-block dNormal priors are built from mu_all and sigma_theta_sq (scalar intercept per block).

This is a thin wrapper around block_rNormalReg that always uses n = 1. When prior_lists and prior_list are both omitted, per-block dNormal priors are built from mu_all, P, and dispersion.

Output layout: coefficients and coef.mode are matrices with **rows = blocks** and **columns = predictors**.

See inst/DESIGN_RGLM_BLOCKS.md.

Per-block prior mean: pass prior_list$mu as an l1 x k matrix (one column per block). Standard for Block~1 of the lmebayes Gibbs sampler, where $\mu_j = X_{\text{hyper}}\gamma$ depends on the current fixed-effects draw.

Dispersion: must be supplied via prior_list$dispersion (residual variance σ^2).

Value

A list with:

theta Vector of length nrow(x) (one draw per row/block).

coefficients,coef.mode Matrices from block_rNormalGLM.

block_rNormalGLM Full block sampler output.

A list with:

b_draws Matrix k x length(coef_cols) of coefficient draws.

coefficients,coef.mode Matrices from block_rNormalReg.

block_rNormalReg Full block sampler output.

A list with class "block_rNormalGLM" including:

coefficients Matrix k * p; row b is the draw for block b.

coef.mode Matrix k * p; posterior mode per block.

block_info Block partition metadata.

block_results List of length k with each block's sampler output.

A list with class "block_rNormalReg" including coefficients, coef.mode, dispersion, and block_info.

Functions

- `block_rNormalReg()`: Gaussian blockwise full conditionals via `.block_rNormalReg_cpp()` (C++ partition, prior payload, and `rNormalReg()` per block). This is the Gaussian counterpart of `block_rNormalGLM`.

See Also

`block_rNormalGLM`, `normalize_block`

`block_rNormalReg`, `normalize_block`

`rNormal_reg`, `simfunction`, `normalize_block`, `inst/DESIGN_RGLM_BLOCKS.md`

Examples

```
## One Gibbs block update: scalar intercept per observation (Poisson)

set.seed(42)

n <- 5L
y <- rpois(n, 2)
x <- matrix(1, n, 1)
mu <- rep(0, n)

upd <- block_rNormalGLM_update(
  mu_all = mu,
  sigma_theta_sq = 1,
  y = y,
  x = x,
  block = seq_len(n),
  family = poisson(),
  use_parallel = FALSE
)

upd$theta
upd$coefficients
## One block_rNormalReg_update step: draw b_j given current fixed effects

set.seed(7)

## Simulate data: 3 schools, 10 students each
n_schools <- 3L
n_per     <- 10L
school    <- rep(seq_len(n_schools), each = n_per)
Z         <- cbind(1, rnorm(n_schools * n_per))  # within-school design
b_true    <- matrix(c(5, 0.5, 3, -0.2, 7, 0.3), nrow = n_schools, byrow = TRUE)
sigma2    <- 1.5

y <- rowSums(Z * b_true[school, ]) + rnorm(nrow(Z), sd = sqrt(sigma2))

## Precision matrix for the random-effect prior (shared across schools)
l1  <- ncol(Z)
P_b <- diag(0.01, l1)   # vague prior
```

```
## Current "fixed-effects" prior mean (one vector recycled to all blocks)
mu_current <- rep(0, l1)

out <- block_rNormalReg_update(
  mu_all     = mu_current,   ## recycled to all k blocks (prior_list path)
  P          = P_b,
  dispersion = sigma2,
  y          = y,
  x          = Z,
  block      = school
)

out$b_draws         ## k x l1 matrix of updated random effects
out$coefficients    ## same content (all columns)
## Conditionally independent block GLM draw (Poisson, 3 outcome groups)

set.seed(42)

## Dobson (1990) p. 93 RCT
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
d.AD <- data.frame(outcome, treatment, counts)

ps <- glmbayesCore::Prior_Setup(counts ~ treatment, family = poisson(), data = d.AD)
y <- ps$y
x <- ps$x
block <- outcome

out <- block_rNormalGLM(
  n = 1L,
  y = y,
  x = x,
  block = block,
  prior_list = list(mu = ps$mu, Sigma = ps$Sigma),
  family = poisson(),
  use_parallel = FALSE
)

out$coefficients
out$coef.mode
## Conditionally independent block Gaussian draw (3 school groups)

set.seed(42)

## Simulate a simple two-level data set: 3 schools, ~10 students each
n_schools <- 3L
n_per     <- 10L
school    <- rep(seq_len(n_schools), each = n_per)
x         <- cbind(1, rnorm(n_schools * n_per))  # intercept + covariate
b_true    <- matrix(c(5, 0.5, 3, -0.2, 7, 0.3), nrow = n_schools, byrow = TRUE)
sigma2    <- 1.5    # residual variance
```

```
y <- rowSums(x * b_true[school, ]) + rnorm(nrow(x), sd = sqrt(sigma2))

## Flat prior (large Sigma) shared across all schools; dispersion = sigma2
l1         <- ncol(x)
prior_list <- list(
  mu         = rep(0, l1),
  Sigma      = diag(100, l1),
  dispersion = sigma2,
  ddef       = FALSE
)

out <- block_rNormalReg(
  n          = 1L,
  y          = y,
  x          = x,
  block      = school,
  prior_list = prior_list
)

out$coefficients   ## k x l1 matrix: one row of b_j draws per school
out$coef.mode
```

`Boston_centered` *Boston housing data with mean-centered predictors*

Description

A copy of `Boston` where all predictors (every column except `medv`) have been mean-centered (subtract column means, no scaling).

Usage

```
data("Boston_centered")
```

Format

A data frame with 506 observations and 14 variables (same names as `Boston`). See `?MASS::Boston` for variable descriptions.

Source

Derived from `MASS::Boston`. Original data described in Harrison and Rubinfeld (1978); see `?Boston` in **MASS**.

`build_mu_all` *Build per-group random-effect prior means for Block 1 sampling*

Description

Forms the `mu_all` matrix passed to `block_rNormalReg_update`: for each grouping level j and each random-effect column k of the level-1 design `Z`,

$$\mu\_\text{all}[k, j] = X_{\text{hyper},k}[j,]^{\top} \gamma_k,$$

where γ_k is the current hyper-parameter vector for RE k (Block 2 state).

Usage

```
build_mu_all(design, fixef, group_levels = NULL)
```

Arguments

`design`	List with components `X_hyper`, `re_coef_names`, and `groups` (typically supplied by a downstream mixed-effects model setup step).
`fixef`	Named list of hyper-parameter vectors, one entry per RE column of `Z`. Names must match `design$re_coef_names`. Each `fixef[[k]]` is a numeric vector of length `ncol(X_hyper[[k]])` with names matching `colnames(X_hyper[[k]])`.
`group_levels`	Character vector of grouping levels defining the *column order* of `mu_all`. Defaults to `levels(design$groups)`, which is the canonical ordering used in two-block mixed models and consistent with `lmer` and `block_rNormalReg` (which preserves the input factor's level order).

Value

A list with:

`mu_all` Numeric matrix `p_re` x `J` (`p_re` = number of RE columns, `J` = number of groups). Row names are `design$re_coef_names`; column names are `group_levels` in the order supplied.

`re_coef_names` Copy of `design$re_coef_names`.

`group_levels` Grouping levels used for columns.

See Also

`lmerb_posterior_mean`, `two_block_rNormal_reg`, `block_rNormalReg_update`

carinsca *Canadian Automobile Insurance Claims for 1957-1958*

Description

The data give the Canadian automobile insurance experience for policy years 1956 and 1957 as of June 30, 1959. The data includes virtually every insurance company operating in Canada and was collated by the Statistical Agency (Canadian Underwriters' Association - Statistical Department) acting under instructions from the Superintendent of Insurance. The data given here is for private passenger automobile liability for non-farmers for all of Canada excluding Saskatchewan.

Usage

```
data(carinsca)
```

Format

A data frame with 20 observations on the following 6 variables:

`Merit` Merit Rating:
- 3 - licensed and accident free 3 or more years
- 2 - licensed and accident free 2 years
- 1 - licensed and accident free 1 year
- 0 - all others

`Class` 1 - pleasure, no male operator under 25
- 2 - pleasure, non-principal male operator under 25
- 3 - business use
- 4 - unmarried owner or principal operator under 25
- 5 - married owner or principal operator under 25

`Insured` Earned car years

`Premium` Earned premium in 1000's
(adjusted to what the premium would have been had all cars been written at 01 rates)

`Claims` Number of claims

`Cost` Total cost of the claim in 1000's of dollars

Details

One could apply Poisson regression to the number of claims and gamma regression to the cost per claim.

Source

Bailey, R. A., and Simon, LeRoy J. (1960). Two studies in automobile insurance ratemaking. ASTIN Bulletin, 192-217.

References

Data downloaded from http://www.statsci.org/data/general/carinsca.html. That site also contains classical Poisson and Gamma regression examples.

Examples

```
############################## Start of carinsca dataset example ##################

data(carinsca)
str(carinsca)
head(carinsca)

###############################################################################
## End of carinsca dataset example
###############################################################################
```

`Cleveland` *Cleveland Heart Disease Dataset*

Description

A cleaned version of the Cleveland heart disease dataset from the UCI Machine Learning Repository. This version contains only complete cases and includes a derived binary outcome variable `hd` indicating the presence (`"Yes"`) or absence (`"No"`) of heart disease.

Usage

```
data("Cleveland")
```

Format

A data frame with 297 observations and 15 variables:

age Age in years (numeric).

sex Sex (0 = female, 1 = male).

cp Chest pain type (numeric code 1–4).

trestbps Resting blood pressure (mm Hg).

chol Serum cholesterol (mg/dl).

fbs Fasting blood sugar > 120 mg/dl (1 = true, 0 = false).

restecg Resting electrocardiographic results (numeric code).

thalach Maximum heart rate achieved.

exang Exercise-induced angina (1 = yes, 0 = no).

oldpeak ST depression induced by exercise relative to rest.

slope Slope of the peak exercise ST segment.

ca Number of major vessels colored by fluoroscopy (0–3).

thal Thalassemia status (numeric code).

num Original UCI disease score (0–4).

hd Binary heart disease indicator: `"No"` (num = 0) or `"Yes"` (num > 0).

Source

UCI Machine Learning Repository: Heart Disease Data Set. https://archive.ics.uci.edu/dataset/45/heart+disease

---

`compute_gaussian_prior` *Compute Calibrated Gaussian Normal–Gamma Prior Components*

Description

Internal Gaussian calibration routine used by `Prior_Setup`. Given weighted Gaussian regression inputs and a dispersion–independent coefficient–scale prior covariance Σ_0, this function computes all Normal–Gamma quantities required by the Gaussian prior families: `dispersion`, `shape`, `shape_ING`, `rate`, `rate_gamma`, and the calibrated coefficient–scale covariance `Sigma`. The input `Sigma_0` is returned unchanged.

Usage

```
compute_gaussian_prior(
  X,
  Y,
  weights,
  offset,
  dispersion = NULL,
  n_effective,
  bhat,
  mu,
  Sigma_0,
  Sigma = NULL,
  n_prior,
  k = 1
)
```

Arguments

`X`	Numeric model matrix with `nrow(X) == length(Y)`.
`Y`	Numeric response vector.
`weights`	Numeric vector of case weights.
`offset`	Numeric offset vector.
`dispersion`	Optional scalar dispersion. If supplied, overrides the calibrated value.

n_effective	Effective sample size (typically sum(weights)).
bhat	Numeric coefficient vector, usually the weighted least–squares estimate.
mu	Numeric prior mean vector.
Sigma_0	Dispersion–independent prior covariance matrix $[p \times p]$.
Sigma	Optional coefficient–scale covariance matrix. If supplied, overrides the calibrated Sigma.
n_prior	Effective prior sample size.
k	Non–negative scalar with $k+p \geq 2$. Controls tail behavior of the variance prior; does not affect posterior means.

Details

The computation follows a structured pipeline:

1. Validate dimensions and numeric inputs.
2. Compute the weighted residual sum of squares at bhat.
3. Form the weighted Gram matrix $X^\top W X$, invert it, and construct the marginal quadratic term S_{marg}.
4. Map n_prior and k to the Normal–Gamma shape and rate.
5. Calibrate the implied dispersion and coefficient covariance.
6. Return all calibrated prior components.

The function assumes the Chapter 11 convention that Σ_0 is *dispersion–free*: it encodes prior structure on the precision–weighted coefficient scale. The returned Sigma is the corresponding coefficient–scale covariance after calibration.

A common choice is the Zellner–type form

$$\Sigma_0 = \frac{1 - \mathrm{pwt}}{\mathrm{pwt}} (X^\top W X)^{-1},$$

where pwt is a scalar prior weight. More generally, Sigma_0 may be any positive–definite matrix.

The function computes:

- The marginal quadratic term

$$S_{marg} = RSS_w + (\hat{\beta} - \mu)^\top \left(\Sigma_0 + (X^\top W X)^{-1}\right)^{-1} (\hat{\beta} - \mu).$$

- Prior Gamma shape: $a_0 = (n_{\mathrm{prior}} + k)/2$.
- Posterior Gamma shape: $a_n = (n_{\mathrm{prior}} + n_w + k)/2$, where $n_w = n_{\mathrm{effective}}$.
- Calibrated dispersion:

$$E[\sigma^2 \mid y] = \frac{S_{marg}}{n_w - p},$$

  the usual weighted residual–df estimator.

- Prior Gamma rate:

$$b_0 = \frac{1}{2} S_{marg} \frac{n_{\mathrm{prior}} + k + p - 2}{n_w - p},$$

  ensuring $E[\sigma^2 \mid y] = S_{marg}/(n_w - p)$.

- Calibrated coefficient covariance:

$$\Sigma = \frac{n_w}{n_{\text{prior}}} E[\sigma^2 \mid y](X^\top W X)^{-1}.$$

Limiting behavior.

- As $n_{\text{prior}} \to \infty$, the prior becomes increasingly concentrated and dominates the likelihood.
- As $n_{\text{prior}} \to 0^+$ with $n_w > p$, $S_{marg} \to RSS_w$ and $E[\sigma^2 \mid y] \to RSS_w/(n_w - p)$, matching the classical weighted Gaussian estimator.
- Strict positivity of the prior rate requires $n_{\text{prior}} + k + p > 2$.
- The prior mean μ is never altered; this function calibrates only scale parameters.

Value

A list with components:

- `dispersion` — calibrated Gaussian dispersion.
- `shape` — Gamma shape for residual precision.
- `shape_ING` — shape for the independent Normal–Gamma prior.
- `rate` — Gamma rate for residual precision.
- `rate_gamma` — Gamma rate for the fixed–β path (`dGamma`).
- `Sigma` — calibrated coefficient–scale covariance.
- `Sigma_0` — the input dispersion–free covariance.

References

There are no references for Rd macro `\insertAllCites` on this help page.

`diagnose_glmbayes` *GPU and OpenCL diagnostics for* **glmbayes**

Description

Compile-time OpenCL probing for **glmbayes**, plus `diagnose_glmbayes()` — a readable report that combines **opencltools** host/runtime checks with this package's OpenCL build status (`glmbayesCore_has_opencl`).

Workstation probes (GPU vendor detection, drivers, ICD/PATH, and related helpers) live in **opencltools**; call `opencltools::. . .` or see `?opencltools`.

Usage

```
diagnose_glmbayes()

glmbayesCore_has_opencl()
```

Details

GPU acceleration speeds up envelope construction and grid evaluation when you pass use_opencl = TRUE in rglmb, rlmb, and related functions. CPU-only builds remain fully usable for standard modelling.

Start with diagnose_glmbayes() for a single readable report; use glmbayesCore_has_opencl() for a quick boolean when scripting. Full install notes: vignette("Chapter-16", package = "glmbayes") ((Nygren 2025)).

Diagnostics exported from glmbayes

- diagnose_glmbayes() — full report including compile-time OpenCL status for this package.
- glmbayesCore_has_opencl() — TRUE if this build was compiled with OpenCL support.
- opencltools::has_opencl() — compile-time flag for **opencltools** (distinct).

Host / runtime checks (opencltools)

- detect_environment_and_gpus()
- detect_compute_runtimes()
- verify_opencl_runtime()
- check_runtime_env()
- get_opencl_core_count()
- load_kernel_source(), load_kernel_library() (pass package = "glmbayesCore" for kernels under inst/cl/)
- add_to_path_windows() and related PATH helpers

References

Nygren K (2025). "Chapter 16: Large models — GPU acceleration using OpenCL." Vignette in the glmbayes R package. R vignette name: Chapter-16.

See Also

diagnose_glmbayes, glmbayesCore_has_opencl, **opencltools**, rglmb, rlmb.

EnvelopeBuild *GPU-Accelerated Envelope Construction for Posterior Simulation*

Description

GPU-Accelerated Envelope Construction for Posterior Simulation

Usage

```
EnvelopeBuild(bStar,A,y,x,mu,P,alpha,wt,family = "binomial",link = "logit",
Gridtype = 2L,n = 1L,n_envopt=NULL,sortgrid = FALSE,use_opencl = FALSE,verbose = FALSE)

EnvelopeSetGrid(GridIndex, cbars, Lint)

EnvelopeSetLogP(logP, NegLL, cbars, G3)
```

Arguments

bStar	Point at which envelope should be centered (typically posterior mode).
A	Diagonal precision matrix for the log-likelihood in standard form.
y	A vector of observations of length m.
x	A design matrix of dimension m * p.
mu	A vector giving the prior means of the variables.
P	Prior precision matrix of the variables (positive-definite).
alpha	Offset vector.
wt	A vector of weights.
family	Family for the envelope: binomial, quasibinomial, poisson, quasipoisson, or Gamma.
link	Link function ("logit", "probit", "cloglog" for binomial; "log" for Poisson/Gamma).
Gridtype	Method to determine the number of subgradient densities in the grid.
n	Number of draws from the posterior (used for grid sizing).
n_envopt	Effective sample size passed to EnvelopeOpt for grid construction. Defaults to match n. Larger values encourage tighter envelopes.
sortgrid	Logical; if TRUE, sort the envelope descending by component probability.
use_opencl	Logical; if TRUE, use OpenCL for gradient evaluations.
verbose	Logical; if TRUE, print progress messages.
GridIndex	A matrix indicating, for each grid component, whether the component lies in the left tail, center, or right tail of the density. Rows correspond to grid components; columns correspond to standardized variables.
cbars	A matrix containing the subgradient of the (adjusted) negative log-likelihood at each grid component.
Lint	A matrix storing the lower and upper bounds for each grid component, depending on whether sampling is from the left, center, or right.
logP	A matrix (typically two columns) with information for each grid component. The first column usually holds the output from EnvelopeSetGrid(), corresponding to the restricted normal density.
NegLL	A vector of negative log-likelihood evaluations at each grid component.
G3	A matrix of tangency points used in the grid.

Details

Constructs an enveloping function for posterior simulation using a grid of tangency points. The envelope is used in accept-reject sampling to guarantee iid draws from the posterior distribution. The implementation follows (Nygren and Nygren 2006), with extensions for GPU acceleration (via OpenCL), dynamic grid optimization, and parallelized evaluation.

The envelope is typically built around the posterior mode $\theta^\star$ for a model in standard form (which in this context means a model with a diagonal posterior precision matrix and prior identity precision matrix - `glmb_Standardize_Model`). It uses dimension-specific width parameters ω_i derived from the precision matrix. Tangency points are selected per dimension, and the full grid is formed via Cartesian expansion. Negative log-likelihood and gradient values are computed at each grid point, either on CPU or GPU depending on the `use_opencl` flag. These values are used to construct a piecewise envelope function that dominates the posterior density.

Value

`EnvelopeBuild()` A list of envelope components used for accept-reject sampling:

- `GridIndex` Integer matrix encoding sampling type (tail, center, line) per dimension and region.
- `thetabars` Matrix of tangency points $\bar{\theta}_j$ for each grid region.
- `cbars` Matrix of subgradients $c(\bar{\theta}_j)$ of the negative log-likelihood at tangency.
- `loglt` Matrix of log left-tail probabilities per dimension and region.
- `logrt` Matrix of log right-tail probabilities per dimension and region.
- `logU` Matrix of selected per-dimension log-density contributions (tail/center) for each region.
- `logP` Matrix of total log-probabilities per region (first column); used to derive mixture weights.
- `PLSD` Vector of normalized mixture weights over grid regions used to draw region indices.
- `LLconst` Vector of acceptance-test constants per region used in the inequality for rejection sampling.

`EnvelopeSetGrid()` A list of matrices computed for grid-based log-density evaluation:

- `Down` Lower bounds for truncated-normal evaluation per dimension and region.
- `Up` Upper bounds for truncated-normal evaluation per dimension and region.
- `lglt` Log left-tail probabilities (from $(-\infty, \mathrm{Up}]$) per dimension and region.
- `lgrt` Log right-tail probabilities (from $[\mathrm{Down}, \infty)$) per dimension and region.
- `lgct` Log central-interval probabilities (from $[\mathrm{Down}, \mathrm{Up}]$) per dimension and region.
- `logU` Selected log-probability per grid cell based on `GridIndex` (tail or center).
- `logP` Matrix with row-wise sums of `logU` (first column) used to form mixture weights.

`EnvelopeSetLogP()` A list with updated mixture-weight and acceptance constants:

- `logP` Input `logP` with its second column populated by the log of unnormalized visit probabilities per region (mixture denominators).
- `LLconst` Vector of acceptance constants $-\log f(y \mid \bar{\theta}_j) - c(\bar{\theta}_j)^T \bar{\theta}_j$ used in the accept-reject test.

Models in standard form

The standard-form restriction and its closed-form truncated-normal integrals follow (Nygren and Nygren 2006). See (Nygren 2025) for the full theoretical details (standard form restriction, closed-form truncated-normal integrals, and the resulting log-scale tractability).

In the implementation, these standard-form quantities determine the grid-based tangency shifts and the precomputed region constants (e.g., the log-CDF pieces) that drive the mixture weights used by the envelope sampler.

Construction of restricted subgradient densities

For the full restricted density construction and the resulting envelope constants, see (Nygren 2025).

In the implementation, these theory objects become the precomputed region log-constants (via closed-form CDF pieces) that are used to normalize the envelope mixture weights for the accept-reject sampler.

Mixture construction and tractable probabilities

The mixture construction and its tractable region probabilities are derived in (Nygren 2025). In the implementation, these theory objects become the precomputed region log-constants and the mixture weights (`PLSD`) used by the envelope-based accept-reject sampler.

Log-scale properties of the envelope function

The log-scale form of the envelope factor and the subgradient inequality that imply envelope dominance are given in (Nygren 2025). In the implementation, these properties allow pointwise evaluation in the log-domain and provide the theoretical basis for the rejection test inside the sampler.

Use of the envelope during sampling

The standardized sampler called through `.rNormalGLM_std_cpp()` uses the envelope to generate posterior samples via rejection sampling. Although not exported, this routine is called internally by `.rNormalGLM_cpp()`, which in turn is invoked by the user-facing function `rNormal_reg()`. Together, these routines implement envelope-based sampling for generalized linear models with log-concave likelihood functions and multivariate normal priors.

The envelope provides a mixture of restricted likelihood-subgradient densities, each defined over a region A_i, with associated mixture weights $\tilde{p}_i$ stored in `PLSD`. The sampling proceeds as follows:

1. A region index $J(i)$ is drawn from the discrete distribution defined by `PLSD`.
2. A candidate θ_i is drawn from the restricted density $q_{A_{J(i)}}^{\bar{\theta}_{J(i)}}$, using the normal CDF bounds `loglt` and `logrt`, and subgradient vector `cbars`. Simulation for each dimension uses the internal C++ function `ctrnorm_cpp()`, which explicitly uses these inputs.
3. The log-likelihood $\log f(y \mid \theta_i)$ is computed and stored in `testll[0]` using the appropriate likelihood function `f2`.

The acceptance test is performed using the inequality

$$\log(U_2) \leq \text{LLconst}[J(i)] + \text{cbars}[J(i),]^T\theta_i + \log f(y \mid \theta_i),$$

which is equivalent to

$$\log(U_2) \leq \log f(y \mid \theta_i) - \left(\log f(y \mid \bar{\theta}_{J(i)}) - c(\bar{\theta}_{J(i)})^T(\theta_i - \bar{\theta}_{J(i)})\right),$$

where:

- `LLconst[J(i)]` stores the precomputed quantity $-\log f(y \mid \bar{\theta}_{J(i)}) - c(\bar{\theta}_{J(i)})^T\bar{\theta}_{J(i)}$, computed during envelope construction via `EnvelopeSet_LogP_C2()`.
- `cbars[J(i), ]` is the precomputed subgradient vector $c(\bar{\theta}_{J(i)})$, extracted via `cbars(J(i), _)`. It defines the exponential tilt direction used to evaluate the envelope.
- `testll[0]` is the log-likelihood at the candidate draw θ_i, evaluated using the model specified by `family` and `link`.
- $-\log(U_2)$ is the threshold from a uniform draw $U_2 \sim \text{Unif}(0, 1)$.

The right-hand side of this inequality is always non-positive, and equals zero when $\theta_i = \bar{\theta}_{J(i)}$. This reflects the fact that the envelope is tangent to the log-likelihood at each $\bar{\theta}_j$, and lies above it elsewhere.

This procedure guarantees that accepted samples are drawn from the posterior $\pi(\theta \mid y)$. The envelope ensures bounded rejection probability, and the mixture structure allows efficient sampling across regions. The output `out` contains accepted draws, and `draws` records the number of attempts per sample.

The components returned by `EnvelopeBuild()` are used in specific steps of the sampling procedure as follows:

- `PLSD` is used to randomly select a region index $J(i)$ from the envelope mixture.
- `loglt` and `logrt` define the truncated normal bounds for each dimension, used together with `cbars` to generate candidate values θ_i.
- `cbars` provides the subgradient vectors $c(\bar{\theta}_j)$ used both for candidate generation and for computing the acceptance test.
- `LLconst` stores precomputed constants used in the acceptance inequality, avoiding recomputation of posterior terms at tangency points.
- `logU` stores the per-dimension log-density contributions for each region, computed during envelope setup. These values are summed to produce `logP`, which determines the mixture weights `PLSD`.
- `logP` contains the total log-probabilities for each grid component, which are normalized to form the mixture weights `PLSD`.
- `thetabars` stores the tangency points $\bar{\theta}_j$ used to define subgradients and region-specific densities.
- `GridIndex` encodes the sampling type (tail, center, line) used for each dimension and region, guiding how each coordinate is simulated.

Algorithmic steps (linked to theory)

The implementation of `EnvelopeBuild` follows the envelope construction in (Nygren and Nygren 2006) for models in standard form (see Section 3–3.3 there). Each computational step corresponds to a theoretical guarantee:

1. **Compute width parameters ω_i from the diagonal precision matrix.** In particular, let θ^* denote the unique posterior mode. For each dimension i, define

$$\omega_i := \frac{\sqrt{2} - \exp\big(-1.20491 - 0.7321\sqrt{0.5 - \partial^2 \log f(\theta^* \mid y)/\partial\theta_i^2}\big)}{\sqrt{1 - \partial^2 \log f(\theta^* \mid y)/\partial\theta_i^2}}.$$

   As seen from the above, the widths ω_i are derived from the local curvature of the log-likelihood at the posterior mode. This ensures that the three-interval construction per dimension below yields an envelope whose efficiency does not deteriorate with sample size.

2. **Use the width parameters to construct intervals around the posterior mode $\theta^\star$.** Specifically, we set

$$\ell_{i,1} = \theta_i^* - 0.5\,\omega_i, \quad \ell_{i,2} = \theta_i^* + 0.5\,\omega_i,$$

   and construct three intervals per dimension:

$$A_{i,1} = (-\infty, \ell_{i,1}), \quad A_{i,2} = [\ell_{i,1}, \ell_{i,2}], \quad A_{i,3} = (\ell_{i,2}, \infty).$$

For each dimension i, let $J_i = \{1, 2, 3\}$ and define $J = \prod_{i=1}^{p} J_i$, which has 3^p elements. Each $j \in J$ is a vector $(j_1, \ldots, j_p)$, and we define

$$A_j^* = \prod_{i=1}^{p} A_{i,j_i}.$$

The collection $A^* = \{A_j^* : j \in J\}$ forms a partition of Θ.

1. **For each member of the partition, select tangency points $\theta^\star \pm \omega_i$.**

For each $j \in J$, define index sets

$$C_{j1} = \{i : j_i = 1\}, \quad C_{j2} = \{i : j_i = 2\}, \quad C_{j3} = \{i : j_i = 3\}.$$

The tangency points $\bar{\theta}_j$ are then defined componentwise by

$$\bar{\theta}_{j,i} = \begin{cases} \theta_i^* - \omega_i, & i \in C_{j1}, \\ \theta_i^*, & i \in C_{j2}, \\ \theta_i^* + \omega_i, & i \in C_{j3}. \end{cases}$$

The tangency points are hence chosen so that the envelope touches the log-likelihood at representative points in each interval, guaranteeing dominance and tightness.

1. **Build the full grid of tangency points (Cartesian product across dimensions).**
   The Cartesian product of per-dimension partitions yields the 3^p restricted densities described in the paper, ensuring coverage of the full parameter space.

2. **Evaluate negative log-likelihood and gradients at each grid point to construct the likelihood subgradient densities and to facilitate accept rejection sampling**
   The subgradients $c(\bar{\theta})$ enter the likelihood-subgradient density construction ((Nygren and Nygren 2006); see also (Nygren 2025)), and both subgradients and negative log-likelihoods (through $h_{\bar{\theta}}(\cdot)$) are used in the accept-reject procedure. CPU and GPU routines compute these values efficiently.

- On CPU: via `f2_f3_non_opencl`.
- On GPU: via `f2_f3_opencl`, which computes these in parallel across faces

3. **Call** `EnvelopeSet_Grid_C2_pointwise` **to evaluate restricted multivariate normal log-densities.** Each restricted density corresponds to a subset of the partition, normalized as in Remark 5 of (Nygren and Nygren 2006).
4. **Call** `EnvelopeSet_LogP_C2` **to compute component log-probabilities and constants.** The constants $\tilde{a}$ and mixture weights $\tilde{p}_i$ are computed explicitly as in Remark 6 of the paper, ensuring that the mixture envelope is properly normalized.
5. **Normalize probabilities (**`PLSD`**) and optionally sort grid components.** Normalization implements Claim 2 of the paper so the mixture forms a valid dominating density for the posterior. Sorting is an implementation detail to improve sampling efficiency.

Theory reference (JASA paper and vignette)

Definitions, claims, theorems, remarks, and examples through Remark 16 (including standard form, the 3^p partition, and sampling remarks) are in (Nygren and Nygren 2006). An expanded narrative is in `vignette("Chapter-A08", package = "glmbayes")`.

Subgradient density formulation

Each grid component corresponds to a tilted multivariate normal density, normalized using the moment-generating function (MGF). In the single-point case, centered at the posterior mode $\theta^\star$, the density is:

$$f(\theta) = \frac{1}{(2\pi)^{p/2}|A|^{-1/2} \cdot \text{MGF}_A(c)} \exp\left(-\frac{1}{2}(\theta-\mu)^T A(\theta-\mu) + c^T(\theta-\theta^\star)\right)$$

where:

- A is the precision matrix,
- μ is the prior mean vector,
- c is the gradient of the log-likelihood at $\theta^\star$,
- $\text{MGF}_A(c)$ is the moment-generating function:

$$\text{MGF}_A(c) = \exp\left(\frac{1}{2}c^T A^{-1} c\right)$$

This closed-form density dominates the posterior locally and is used when `Gridtype = 1`. For richer envelopes, multiple such components are constructed at tangency points θ_j, each with its own gradient c_j, and combined into a mixture:

$$f_{\text{env}}(\theta) = \sum_{j=1}^{K} p_j f_j(\theta)$$

where the weights p_j are computed using log-CDF differences and constants:

$$\log p_j = \log \Phi(U_j) - \log \Phi(L_j) - \text{NegLL}_j + \text{LLconst}_j$$

Gridtype logic

The `Gridtype` argument controls how many tangency points are used per dimension:

- 1: Threshold rule. If $1 + a_i \leq 2/\sqrt{\pi}$, use a single-point envelope at the mode; otherwise use three points.
- 2: Dynamic optimization via `EnvelopeOpt`, which balances grid build cost and expected acceptance rate. Grid size is scaled by `n` and the number of OpenCL cores when GPU is enabled.
- 3: Always use three points per dimension.
- 4: Always use a single point (mode only).

Supported families and links

The following families and link functions are supported:

- Binomial: logit, probit, cloglog
- Quasibinomial: logit, probit
- Poisson: log
- Quasipoisson: log
- Gamma: log
- Gaussian: identity

GPU acceleration (`use_opencl = TRUE`) is available for all of the above except Gaussian, which is always evaluated on CPU.

GPU acceleration

When `use_opencl = TRUE`, likelihood and gradient evaluations are offloaded to the GPU using OpenCL. This can substantially reduce runtime for high-dimensional models or large grids. Results are mathematically equivalent to the CPU version, but small numerical differences may occur due to floating-point arithmetic. If reproducibility across hardware is critical, prefer the CPU path.

If OpenCL support was not detected at compile time, the flag is ignored and the CPU implementation is used. Diagnostic messages are printed when `verbose = TRUE`.

Verbose output

When `verbose = TRUE`, the function prints:

- Grid type, number of draws, OpenCL usage, and detected core count.
- Grid size after expansion.
- Time-stamped messages when entering the grid loop, starting likelihood evaluations, starting gradient evaluations, and invoking GPU kernels.
- Messages when setting grid values, computing log-probabilities, and sorting.

Any constants needed by the sampling are added to a list and returned.

References

Nygren K (2025). "Chapter A08: Overview of Envelope Related Functions." Vignette in the glmbayes R package. R vignette name: Chapter-A08.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

See Also

EnvelopeSize, EnvelopeEval, EnvelopeSort, glmb_Standardize_Model; rNormal_reg, rglmb, rlmb. Theory and vignettes: (Nygren and Nygren 2006); (Nygren 2025, 2025).

Examples

```
data(menarche,package="MASS")
Age2=menarche$Age-13

summary(menarche)
plot(Menarche/Total ~ Age, data=menarche)


x<-matrix(as.numeric(1.0),nrow=length(Age2),ncol=2)
x[,2]=Age2

y=menarche$Menarche/menarche$Total
wt=menarche$Total

mu<-matrix(as.numeric(0.0),nrow=2,ncol=1)
mu[2,1]=(log(0.9/0.1)-log(0.5/0.5))/3

V1<-1*diag(as.numeric(2.0))

# 2 standard deviations for prior estimate at age 13 between 0.1 and 0.9
## Specifies uncertainty around the point estimates

V1[1,1]<-((log(0.9/0.1)-log(0.5/0.5))/2)^2
V1[2,2]=(3*mu[2,1]/2)^2  # Allows slope to be up to 1 times as large as point estimate

famfunc<-glmbfamfunc(binomial(logit))

f1<-famfunc$f1
f2<-famfunc$f2
f3<-famfunc$f3
f5<-famfunc$f5
f6<-famfunc$f6

dispersion2<-as.numeric(1.0)
start <- mu
offset2=rep(as.numeric(0.0),length(y))
P=solve(V1)
n=1000
```

```
###### Adjust weight for dispersion

wt2=wt/dispersion2

######################### Shift mean vector to offset so that adjusted model has 0 mean

alpha=x%*%as.vector(mu)+offset2
mu2=0*as.vector(mu)
P2=P
x2=x


#####  Optimization step to find posterior mode and associated Precision

parin=start-mu

opt_out=optim(parin,f2,f3,y=as.vector(y),x=as.matrix(x),mu=as.vector(mu2),
              P=as.matrix(P),alpha=as.vector(alpha),wt=as.vector(wt2),
              method="BFGS",hessian=TRUE
)

bstar=opt_out$par  ## Posterior mode for adjusted model
bstar
bstar+as.vector(mu)  # mode for actual model
A1=opt_out$hessian # Approximate Precision at mode

## Standardize Model

Standard_Mod=glmb_Standardize_Model(y=as.vector(y), x=as.matrix(x),P=as.matrix(P),
                                    bstar=as.matrix(bstar,ncol=1), A1=as.matrix(A1))

bstar2=Standard_Mod$bstar2
A=Standard_Mod$A
x2=Standard_Mod$x2
mu2=Standard_Mod$mu2
P2=Standard_Mod$P2
L2Inv=Standard_Mod$L2Inv
L3Inv=Standard_Mod$L3Inv

Env2=EnvelopeBuild(as.vector(bstar2), as.matrix(A),y, as.matrix(x2),
as.matrix(mu2,ncol=1),as.matrix(P2),as.vector(alpha),as.vector(wt2),
family="binomial",link="logit",Gridtype=as.integer(3), n=as.integer(n),
sortgrid=TRUE)

## These now seem to match

Env2
```

EnvelopeCentering *Envelope Centering for Bayesian Gaussian Regression*

Description

EnvelopeCentering() computes an initial dispersion and the expected posterior weighted RSS (closed form under the Normal posterior for coefficients) for use in envelope construction when the dispersion is unknown. The dispersion-anchoring loop updates dispersion from the Gamma posterior using that expected RSS each iteration. This step is typically called inside rIndepNormalGammaReg() before EnvelopeOrchestrator, but may be used directly for diagnostics or custom workflows.

Usage

```
EnvelopeCentering(
  y,
  x,
  mu,
  P,
  offset,
  wt,
  shape,
  rate,
  Gridtype = 2L,
  verbose = FALSE
)
```

Arguments

y	Numeric response vector of length m.
x	Numeric design matrix of dimension m * p.
mu	Numeric vector of prior means (length p).
P	Numeric matrix of prior precision (p * p).
offset	Numeric vector of length m. Use rep(0, m) for none.
wt	Numeric vector of prior weights.
shape	Numeric. Shape parameter of the Gamma prior for the dispersion.
rate	Numeric. Rate parameter of the Gamma prior for the dispersion.
Gridtype	Integer. Grid construction method (default 2).
verbose	Logical. Reserved for API compatibility; currently unused in C++.

Details

The function first obtains an initial dispersion via lm.wfit residual variance, then iteratively: (1) computes the expected weighted RSS under the Normal posterior (closed form), (2) updates the dispersion via the Gamma posterior using the expected RSS. The result is used as dispersion2 and RSS_Post2 in downstream envelope construction (e.g., EnvelopeOrchestrator).

This anchors the joint Normal–Gamma accept–reject construction in (Nygren and Nygren 2006); see vignettes Chapter-A07, Chapter-A11, and (Nygren 2025, 2025).

Value

A list with components:

`dispersion` Numeric. Anchored dispersion value.

`RSS_post` Numeric. Expected posterior weighted RSS (closed form; last iteration).

References

Nygren K (2025). "Chapter A08: Overview of Envelope Related Functions." Vignette in the glmbayes R package. R vignette name: Chapter-A08.

Nygren K (2025). "Independent Normal–Gamma Regression Sampler." Vignette in the glmbayes R package. R vignette name: independent-norm-gamma.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

See Also

`EnvelopeOrchestrator` for envelope construction; `EnvelopeBuild`, `EnvelopeDispersionBuild`; `rindepNormalGamma_reg` for the full simulation routine; `rlmb` for the user-facing linear-model interface.

Examples

```
############################### Start of EnvelopeCentering example ###################

# This example demonstrates EnvelopeCentering in isolation. It computes an
# initial dispersion and posterior RSS for use in envelope construction when
# the dispersion is unknown (Gaussian regression with Normal-Gamma prior).
# This is Step A of the full pipeline in Ex_EnvelopeDispersionBuild and
# Ex_rIndepNormalGammaReg_std.

ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)

ps <- Prior_Setup(weight ~ group, gaussian())

x <- as.matrix(ps$x)
y <- as.vector(ps$y)
mu <- ps$mu
Sigma <- ps$Sigma
shape <- ps$shape
rate <- ps$rate

n_obs <- length(y)
wt <- rep(1, n_obs)
offset2 <- rep(0, n_obs)
```

```
# Reconstruct coefficient precision P (matches rindepNormalGamma_reg)
Rchol <- chol(Sigma)
Pinv <- chol2inv(Rchol)
P <- 0.5 * (Pinv + t(Pinv))

Gridtype_core <- as.integer(2)

########################################################################
# EnvelopeCentering: initial dispersion + dispersion anchoring loop
########################################################################
centering <- EnvelopeCentering(
  y = y,
  x = x,
  mu = as.vector(mu),
  P = P,
  offset = offset2,
  wt = wt,
  shape = shape,
  rate = rate,
  Gridtype = Gridtype_core,
  verbose = FALSE
)

centering$dispersion
centering$RSS_post

########################################################################
# End of EnvelopeCentering example
########################################################################
```

`EnvelopeDispersionBuild` *Builds Dispersion-Aware Envelope for Simulation*

Description

Constructs a dispersion-aware envelope for simulation in Gaussian models with uncertain variance. This function extrapolates the coefficient envelope across a high-probability interval for the dispersion parameter `sigma^2`, and builds a global upper bound for the log-posterior remainder. It also computes mixture weights for envelope faces and adjusts the Gamma proposal for precision.

The envelope is constructed using the slopes of the face constants with respect to dispersion, evaluated at an anchor point. The resulting structure supports exact i.i.d. sampling via accept-reject correction.

The procedure follows these steps:

1. **Posterior precision (Gamma) parameters.** Using the prior and posterior-predictive RSS, set

$$\text{shape2} = \text{Shape} + n_{\text{obs}}/2, \quad \text{rate3} = \text{Rate} + \text{RSS}_{\text{post}}/2$$

These parameterize the posterior precision $v = 1/\sigma^2 \sim \text{Gamma}(\text{shape2}, \text{rate3})$.

2. **Central credible interval for dispersion (low, upp).** Choose a central mass level `max_disp_perc` (e.g., 0.99) for precision, then invert the corresponding Gamma quantiles to dispersion:

$$\text{low} = 1/Q_{\Gamma}(\text{max\_disp\_perc}; \text{shape2}, \text{rate3}), \quad \text{upp} = 1/Q_{\Gamma}(1-\text{max\_disp\_perc}; \text{shape2}, \text{rate3})$$

   The interval $[\text{low}, \text{upp}]$ is the domain over which all envelopes must dominate.

3. **Face slopes at an anchor (dispstar).**

$$\text{dispstar} = \text{rate3}/(\text{shape2} - 1)$$

   (posterior mean of σ^2). Compute New_LL_Slope_j for each face j.

4. **Linear extrapolation of face constants.**

$$\theta_{\bar{j}}^{\text{low}} = \theta_{\bar{j}}^{\text{base}} + (\text{low} - \text{dispstar}) \cdot \text{New\_LL\_Slope}_j$$

$$\theta_{\bar{j}}^{\text{upp}} = \theta_{\bar{j}}^{\text{base}} + (\text{upp} - \text{dispstar}) \cdot \text{New\_LL\_Slope}_j$$

5. **Global upper line and endpoint maxima.**

$$\text{max\_low} = \max_j \theta_{\bar{j}}^{\text{low}}, \quad \text{max\_upp} = \max_j \theta_{\bar{j}}^{\text{upp}}$$

$$\text{new\_slope} = (\text{max\_upp}-\text{max\_low})/(\text{upp}-\text{low}), \quad \text{new\_int} = \text{max\_low}-\text{new\_slope}\cdot\text{low}$$

6. **Face slack and mixture weights.**

$$\text{lg\_prob\_factor}_j = \max\left(\theta_{\bar{j}}^{\text{upp}} - \text{max\_upp},\ \theta_{\bar{j}}^{\text{low}} - \text{max\_low}\right)$$

   Combine with $\text{New\_logP2}_j = \text{logP}_j + \frac{1}{2}\|\bar{c}_j\|^2$ to form mixture weights $\text{PLSD}_j \propto \exp\big(\text{New\_logP2}_j + \text{lg\_prob\_factor}_j\big)$.

7. **Gamma tilt and dispersion-axis envelope.**

$$\text{dispstar} = (\text{upp} - \text{low})/\log(\text{upp}/\text{low})$$

$$\text{lm\_log2} = \text{new\_slope}\cdot\text{dispstar}, \quad \text{lm\_log1} = \text{new\_int}+\text{new\_slope}\cdot\text{dispstar}-\text{new\_slope}\cdot\log(\text{dispstar})$$

   Tilt the Gamma proposal via $\text{shape3} = \text{shape2} - \text{lm\_log2}$.

Usage

```
EnvelopeDispersionBuild(
  Env, Shape, Rate, P, y, x, alpha, n_obs, RSS_post, RSS_ML,
  mu, wt, max_disp_perc = 0.99,
  disp_lower = NULL, disp_upper = NULL,
  verbose = FALSE, use_parallel = TRUE
)
```

Arguments

Env	Envelope object from EnvelopeBuild, containing tangency points and gradients
Shape	Prior shape parameter for precision v = 1 / sigma^2
Rate	Prior rate parameter for precision
P	Prior precision matrix for coefficients
y	Numeric response vector of length m
x	a design matrix of dimension m * p
alpha	Numeric offset vector of length m
n_obs	Number of observations
RSS_post	Expected posterior weighted residual sum of squares (i.e., $\mathbb{E}[\mathrm{RSS}(\beta) \mid y, \phi]$ under the Normal posterior for β at fixed dispersion $\phi = d$). This value is used for dispersion anchoring / Gamma updates.
RSS_ML	Residual sum of squares associated with MLE estimate
mu	Prior mean parameter
wt	weight vector
max_disp_perc	Truncation level for dispersion (default 0.99)
disp_lower	lower bound truncation for dispersion
disp_upper	upper bound truncation for dispersion
verbose	Option to have verbose output
use_parallel	Logical. Whether to use parallel processing.

Details

This function is designed to complement EnvelopeBuild for Gaussian models with Normal-Gamma priors. It enables exact sampling of both coefficients and dispersion by constructing a joint envelope that respects posterior curvature in both dimensions.

The dispersion anchor point is chosen as the log-scale center of the credible interval, and the Gamma proposal is tilted to match the envelope slope at this point. Theory and narrative: (Nygren and Nygren 2006); vignettes Chapter-A07, Chapter-A11; (Nygren 2025, 2025).

Value

EnvelopeDispersionBuild() A list containing:

- Env_out Envelope object with updated mixture weights (PLSD)
- gamma_list Posterior Gamma tilt parameters
  - shape3 Adjusted shape parameter after slope correction
  - rate2 Posterior rate parameter, defined as Rate + rss_min_global/2
  - disp_upper Upper bound of the dispersion interval σ^2
  - disp_lower Lower bound of the dispersion interval σ^2
- UB_list Upper-bound diagnostics
  - RSS_ML Residual sum of squares at the maximum-likelihood estimate
  - RSS_Min Minimum residual sum of squares across envelope faces

max_New_LL_UB Maximum extrapolated face constant at the upper dispersion bound

max_LL_log_disp Log-posterior upper bound evaluated at disp_upper

lm_log1 Intercept term of the global upper line approximation

lm_log2 Slope term of the global upper line approximation

lg_prob_factor Per-face slack factors used in mixture weighting

lmc1 Linear extrapolation constant (intercept)

lmc2 Linear extrapolation constant (slope)

UB2min Minimum UB2 value across faces, used for diagnostics

diagnostics Internal diagnostic values

dispstar Anchor dispersion value (posterior mean or geometric mean)

New_LL_Slope Vector of slopes of face constants at dispstar

shape2 Posterior shape parameter before tilt correction

rate3 Posterior rate parameter before tilt correction

shape3 Adjusted shape parameter (same as in gamma_list)

max_low Maximum extrapolated face constant at the lower dispersion bound

max_upp Maximum extrapolated face constant at the upper dispersion bound

new_slope Slope of the global upper line across dispersion bounds

new_int Intercept of the global upper line across dispersion bounds

prob_factor Normalized mixture weights across faces

UB2min Minimum UB2 diagnostic value (duplicated for consistency)

EnvBuildLinBound() Numeric vector of slopes of face constants with respect to dispersion, evaluated at the anchor dispstar

thetabar_const() Numeric vector of base face constants computed from tangency points and gradient vectors under prior precision P

Inv_f3_with_disp() Numeric matrix of inverse function evaluations at a given dispersion and face subset, returned by the C++ routine _glmbayes_Inv_f3_with_disp

UB2() Numeric scalar representing the UB2 upper-bound criterion for a given dispersion and face, defined as $(1/dispersion) * (RSS - rss\_min\_global)$

rss_face_at_disp() Numeric scalar giving the residual sum of squares for a specified face at a given dispersion, computed from cached matrices and the inverse function evaluation

Use in accept/reject procedure

The accept/reject sampler relies on a decomposition of the log-posterior into a test statistic and several bounding terms. Each component is constructed so that its sign is controlled, ensuring the validity of the accept/reject step.

test1 (log-likelihood bound) Placeholder: explain how test1 is formed and why it is non-positive.

UB1 (if applicable) Placeholder: describe UB1's role and why it is non-negative.

UB2 (residual sum of squares bound) Placeholder: explain how UB2 is constructed from RSS differences and why it is non-negative.

UB3A (face-wise quadratic/linear envelope surplus) Placeholder: explain how lg_prob_factor, lmc1, and lmc2 are derived and why UB3A >= 0.

UB3B (dispersion-axis envelope surplus) Placeholder: explain how lm_log1, lm_log2, and max_New_LL_UB are used and why UB3B >= 0.

Together, these components define

$$test = test1 - UB2 - UB3A - UB3B,$$

with $test1 \leq 0$ and each UB term ≥ 0, ensuring the accept/reject procedure is valid and unbiased.

References

Nygren K (2025). "Chapter A08: Overview of Envelope Related Functions." Vignette in the glmbayes R package. R vignette name: Chapter-A08.

Nygren K (2025). "Independent Normal–Gamma Regression Sampler." Vignette in the glmbayes R package. R vignette name: independent-norm-gamma.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

See Also

EnvelopeBuild, EnvelopeOrchestrator, EnvelopeCentering (for obtaining RSS_post and anchored dispersion), rindepNormalGamma_reg, rlmb; rglmb, glmbfamfunc.

Examples

```
############################### Start of EnvelopeDispersionBuild example ###################

# This example mirrors the current C++ algorithm path for Gaussian regression
# with an independent Normal-Gamma prior:
#   rIndepNormalGammaReg:
#     - Step A: EnvelopeCentering (initial dispersion + dispersion anchoring loop)
#     - Step B: optimize posterior mode for coefficients (optim + f2/f3)
#     - Step C: standardize the model (glmb_Standardize_Model)
#     - Step D: build coefficient envelope (EnvelopeBuild)
#     - Step E: build dispersion-aware envelope (EnvelopeDispersionBuild)
#     - Step F: sort envelope components (EnvelopeSort)
# It stops after envelope construction (no standardized-envelope sampling).

ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)

ps <- Prior_Setup(weight ~ group, gaussian())

x <- as.matrix(ps$x)
y <- as.vector(ps$y)
mu <- ps$mu
Sigma <- ps$Sigma
shape <- ps$shape
```

```
rate <- ps$rate

n_obs <- length(y)
wt <- rep(1, n_obs)
offset2 <- rep(0, n_obs)

# Reconstruct coefficient precision P (matches rindepNormalGamma_reg)
Rchol <- chol(Sigma)
Pinv <- chol2inv(Rchol)
P <- 0.5 * (Pinv + t(Pinv))

famfunc <- glmbfamfunc(gaussian())
f2 <- famfunc$f2
f3 <- famfunc$f3

Gridtype_core <- as.integer(2)

##############################################################################
# Step A: EnvelopeCentering (initial dispersion + dispersion anchoring loop)
##############################################################################
centering <- EnvelopeCentering(
  y = y,
  x = x,
  mu = as.vector(mu),
  P = P,
  offset = offset2,
  wt = wt,
  shape = shape,
  rate = rate,
  Gridtype = Gridtype_core,
  verbose = FALSE
)


dispersion2 <- centering$dispersion
RSS_Post2 <- centering$RSS_post

n_w <- sum(wt)

##############################################################################
# Step B: Coefficient posterior mode optimization (optim + f2/f3)
##############################################################################
dispstar <- dispersion2

wt2_opt <- wt / dispstar
alpha <- as.vector(x %*% as.vector(mu) + offset2)

mu2 <- rep(0, length(as.vector(mu)))   # mu2 = 0 * mu (as in C++)
parin <- rep(0, length(as.vector(mu))) # parin = 0 vector (mu - mu)

opt_out <- optim(
  par = parin,
  fn = f2,
```

```
  gr = f3,
  y = as.vector(y),
  x = as.matrix(x),
  mu = as.vector(mu2),
  P = as.matrix(P),
  alpha = as.vector(alpha),
  wt = as.vector(wt2_opt),
  method = "BFGS",
  hessian = TRUE
)

bstar <- opt_out$par
A1 <- opt_out$hessian

##############################################################################
# Step C: Standardize model (glmb_Standardize_Model)
##############################################################################
Standard_Mod <- glmb_Standardize_Model(
  y = as.vector(y),
  x = as.matrix(x),
  P = as.matrix(P),
  bstar = as.matrix(bstar, ncol = 1),
  A1 = as.matrix(A1)
)

bstar2 <- Standard_Mod$bstar2
A <- Standard_Mod$A
x2_std <- Standard_Mod$x2
mu2_std <- Standard_Mod$mu2
P2_std <- Standard_Mod$P2

##############################################################################
# Step D: EnvelopeBuild (coefficient envelope at Gridtype = 3)
##############################################################################
max_disp_perc <- 0.99
n_env <- as.integer(200) # used by EnvelopeBuild for diagnostics/overhead
Gridtype_env <- as.integer(3) # EnvelopeOrchestrator overrides to 3

shape2_env <- shape + n_w / 2.0
rate3_env <- rate + RSS_Post2 / 2.0
d1_star <- rate3_env / (shape2_env - 1.0)

wt2_env <- wt / d1_star

Env2 <- EnvelopeBuild(
  bStar = as.vector(bstar2),
  A = as.matrix(A),
  y = as.vector(y),
  x = as.matrix(x2_std),
  mu = as.matrix(mu2_std, ncol = 1),
  P = as.matrix(P2_std),
  alpha = as.vector(alpha),
  wt = as.vector(wt2_env),
```

```
  family = "gaussian",
  link = "identity",
  Gridtype = Gridtype_env,
  n = n_env,
  n_envopt = as.integer(1),
  sortgrid = FALSE,
  use_opencl = FALSE,
  verbose = FALSE
)

################################################################################
# Step E: EnvelopeDispersionBuild (dispersion-aware envelope)
################################################################################
disp_env_out <- EnvelopeDispersionBuild(
  Env = Env2,
  Shape = shape,
  Rate = rate,
  P = as.matrix(P2_std),
  y = as.vector(y),
  x = as.matrix(x2_std),
  alpha = as.vector(alpha),
  n_obs = as.integer(n_obs),
  RSS_post = RSS_Post2,
  RSS_ML = NA_real_,
  mu = as.matrix(mu2_std, ncol = 1),
  wt = as.vector(wt),
  max_disp_perc = max_disp_perc,
  disp_lower = NULL,
  disp_upper = NULL,
  verbose = FALSE,
  use_parallel = TRUE
)

################################################################################
# Step F: EnvelopeSort (mirror EnvelopeOrchestrator: disp_grid_type = 2)
################################################################################
Env3_raw <- disp_env_out$Env_out
UB_list_new <- disp_env_out$UB_list
gamma_list_new <- disp_env_out$gamma_list

cbars <- Env3_raw$cbars
l1 <- ncol(cbars)
l2 <- nrow(cbars)

logP_vec <- Env3_raw$logP
logP_mat <- matrix(logP_vec, nrow = length(logP_vec), ncol = 1)

Env3 <- EnvelopeSort(
  l1 = l1,
  l2 = l2,
  GIndex = Env3_raw$GridIndex,
  G3 = Env3_raw$thetabars,
  cbars = cbars,
```

```
  logU = Env3_raw$logU,
  logrt = Env3_raw$logrt,
  loglt = Env3_raw$loglt,
  logP = logP_mat,
  LLconst = Env3_raw$LLconst,
  PLSD = Env3_raw$PLSD,
  a1 = Env3_raw$a1,
  E_draws = Env3_raw$E_draws,
  lg_prob_factor = UB_list_new$lg_prob_factor,
  UB2min = UB_list_new$UB2min
)

UB_list_final <- UB_list_new
UB_list_final$lg_prob_factor <- Env3$lg_prob_factor
UB_list_final$UB2min <- Env3$UB2min

env_final <- list(
  Env = Env3,
  gamma_list = gamma_list_new,
  UB_list = UB_list_final,
  diagnostics = disp_env_out$diagnostics,
  low = gamma_list_new$disp_lower,
  upp = gamma_list_new$disp_upper
)

print(env_final$low)
print(env_final$upp)
print(env_final$gamma_list[c("shape3", "rate2")])

env_final

################################################################################
# End: envelope construction only
################################################################################
```

`EnvelopeEval` *Evaluate Negative Log-Likelihood and Gradients*

Description

`EnvelopeEval()` evaluates the negative log-likelihood and gradients at a grid of parameter values, optionally using OpenCL acceleration.

Usage

```
EnvelopeEval(G4, y, x, mu, P, alpha, wt,
                    family, link,
                    use_opencl = FALSE, verbose = FALSE)
```

Arguments

G4	Numeric matrix of parameter values (parameters * grid points).
y	Numeric response vector.
x	Numeric design matrix.
mu	Numeric matrix of offsets or prior means.
P	Numeric matrix representing the portion of the prior precision shifted into the likelihood.
alpha	Numeric offset vector of length m
wt	Numeric vector of weights.
family	Character string; model family (e.g. "gaussian").
link	Character string; link function (e.g. "identity").
use_opencl	Logical; if TRUE, attempt OpenCL acceleration.
verbose	Logical; if TRUE, print diagnostic output.

Details

The lower-level helpers f2_f3_non_opencl and f2_f3_opencl are internal C++ kernels used by the CPU and OpenCL backends. The internal routine run_opencl_pilot benchmarks OpenCL performance on a pilot subset of the grid to estimate runtime before full evaluation.

These functions implement the grid evaluation logic used in envelope construction for rejection sampling. They make use of the theory described in (Nygren and Nygren 2006) and the general implementation outlined in (Nygren 2025).

The evaluation workflow has several layers: **1. High-level dispatch (**EnvelopeEval**)**

- EnvelopeEval() is the user-facing entry point. It accepts a grid of parameter values (G4) and the data (y, x, mu, P, alpha, wt).
- If the grid is large (>= 14 columns), it first calls run_opencl_pilot to benchmark OpenCL performance and optionally report estimated runtime.
- It then dispatches to either the CPU or GPU backend:
  - If use_opencl = TRUE and the family is not "gaussian", it calls f2_f3_opencl (an internal C++ kernel).
  - Otherwise, it calls f2_f3_non_opencl (the CPU kernel).

2. CPU backend (f2_f3_non_opencl**)**

- This function evaluates the negative log-likelihood and gradients using standard CPU routines.
- It inspects the family and link arguments and routes to the correct pair of kernels (f2_* for the likelihood, f3_* for the gradient).
- For example:
  - "binomial" with "logit" calls f2_binomial_logit() and f3_binomial_logit().
  - "poisson" calls f2_poisson() and f3_poisson().
  - "gaussian" calls f2_gaussian() and f3_gaussian().

- These kernels ultimately rely on the same C math routines that R itself uses (from the `nmath`/`rmath` libraries), ensuring numerical consistency with base R functions like `dnorm`, `dpois`, etc.

3. GPU backend (`f2_f3_opencl`)

- This function mirrors the CPU backend but executes the likelihood and gradient calculations on an OpenCL device (GPU or CPU).
- It flattens the input matrices/vectors and allocates output buffers.
- It then constructs a full OpenCL program by concatenating:
  - a generic OpenCL support header (`OPENCL.CL`),
  - OpenCL ports of R's `rmath`, `nmath`, and `dpq` libraries,
  - and the family/link-specific kernel source (e.g. `f2_f3_binomial_logit.cl`).
- The resulting program is compiled and passed to a kernel runner (`f2_f3_kernel_runner`) which executes the likelihood and gradient calculations in parallel on the device.
- This ensures that the GPU backend produces results consistent with the CPU backend, but can scale to much larger grids efficiently.

4. Pilot timing (`run_opencl_pilot`)

- This helper runs a small subset of the grid through the OpenCL backend to estimate runtime.
- It is used by `EnvelopeEval()` to inform users (when `verbose = TRUE`) whether OpenCL acceleration is likely to be beneficial.

5. Returned values

- All backends return a list with:
  - `NegLL`: numeric vector of negative log-likelihood values.
  - `cbars`: numeric matrix of gradients (parameters * grid points).

6. Role of likelihood and gradients in sampling

- The outputs of `EnvelopeEval()` - the negative log-likelihood values (`NegLL`) and the gradient matrix (`cbars`) - are not endpoints in themselves. They form the *envelope* used in the rejection sampler implemented by internal functions such as `.rNormalGLM_std_cpp()`.
- This routine is called by `.rNormalGLM_cpp()`, which underlies the user-facing function `rNormal_reg()`. Together they implement envelope-based posterior sampling for GLMs with log-concave likelihoods and multivariate normal priors.

7. Simulation execution (accept/reject procedure)

The acceptance test is performed using

$$\log(U_2) \leq \log f(y \mid \theta_i) - \Big(\log f(y \mid \bar{\theta}_{J(i)}) - c(\bar{\theta}_{J(i)})^T(\theta_i - \bar{\theta}_{J(i)}) \Big) \leq 0$$

Connections between code and notation:

- The arguments `G4` (in `EnvelopeEval`) and `b` (in `f2_f3_*`) both represent the grid of tangency points $\bar{\theta}_j$.
- The output `NegLL` corresponds to $-\log f(y \mid \bar{\theta}_{J(i)})$, i.e. the negative log-likelihood evaluated at each tangency point.

- The output `cbars` corresponds to the subgradient vectors $c(\bar{\theta}_{J(i)})$, which define the tangent hyperplanes used in the envelope construction.

Precomputation for efficiency:

- Both `NegLL` and `cbars` are computed once during envelope construction, prior to the simulation stage.
- This means the sampler does not need to recompute likelihoods or gradients at every candidate draw - it simply reuses the stored values (`NegLL`, `cbars`, and `LLconst`) in the acceptance inequality.

This design ensures that the envelope is tangent to the log-likelihood at each $\bar{\theta}_j$, lies above it elsewhere, and that the accept-reject procedure can run efficiently while still producing samples from the true posterior $\pi(\theta \mid y)$.

Value

EnvelopeEval List with components `NegLL` (numeric vector of negative log-likelihood values) and `cbars` (numeric matrix of gradients).

f2_f3_non_opencl List with components `qf` (negative log-likelihood) and `grad` (gradients) from the CPU kernel.

f2_f3_opencl List with components `qf` and `grad` from the OpenCL kernel.

run_opencl_pilot Numeric scalar giving estimated runtime (seconds) for OpenCL evaluation on a pilot subset of the grid.

References

Nygren K (2025). "Chapter A05: Simulation Methods – Likelihood Subgradient Densities." Vignette in the glmbayes R package. R vignette name: Chapter-A05.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

See Also

`EnvelopeBuild`, `EnvelopeSize`, `EnvelopeSort`; `rNormal_reg`, `rglmb`. Vignettes: (Nygren 2025, 2025).

Examples

```
############################## Start of EnvelopeEval example ###################

# This example demonstrates EnvelopeEval in isolation. EnvelopeEval evaluates
# the negative log-likelihood and gradients at a grid of parameter values.
# It is called internally by EnvelopeBuild. Here we build the same inputs
# (grid G4, standardized model) using EnvelopeSize and expand.grid, then
# call EnvelopeEval directly. The setup mirrors Ex_EnvelopeBuild through
# the standardization step.

data(menarche, package = "MASS")
```

```
Age2 <- menarche$Age - 13

x <- matrix(as.numeric(1.0), nrow = length(Age2), ncol = 2)
x[, 2] <- Age2

y <- menarche$Menarche / menarche$Total
wt <- menarche$Total

mu <- matrix(as.numeric(0.0), nrow = 2, ncol = 1)
mu[2, 1] <- (log(0.9 / 0.1) - log(0.5 / 0.5)) / 3

V1 <- 1 * diag(as.numeric(2.0))
V1[1, 1] <- ((log(0.9 / 0.1) - log(0.5 / 0.5)) / 2)^2
V1[2, 2] <- (3 * mu[2, 1] / 2)^2

famfunc <- glmbfamfunc(binomial(logit))
f2 <- famfunc$f2
f3 <- famfunc$f3

dispersion2 <- as.numeric(1.0)
start <- mu
offset2 <- rep(as.numeric(0.0), length(y))
P <- solve(V1)
n <- 1000

wt2 <- wt / dispersion2
alpha <- x %*% as.vector(mu) + offset2
mu2 <- 0 * as.vector(mu)
P2 <- P
x2 <- x

parin <- start - mu
opt_out <- optim(parin, f2, f3,
  y = as.vector(y), x = as.matrix(x), mu = as.vector(mu2),
  P = as.matrix(P), alpha = as.vector(alpha), wt = as.vector(wt2),
  method = "BFGS", hessian = TRUE
)

bstar <- opt_out$par
A1 <- opt_out$hessian

Standard_Mod <- glmb_Standardize_Model(
  y = as.vector(y), x = as.matrix(x), P = as.matrix(P),
  bstar = as.matrix(bstar, ncol = 1), A1 = as.matrix(A1)
)

bstar2 <- Standard_Mod$bstar2
A <- Standard_Mod$A
x2 <- Standard_Mod$x2
mu2 <- Standard_Mod$mu2
P2 <- Standard_Mod$P2

##############################################################################
```

```
# Build grid G4 via EnvelopeSize and expand.grid (as EnvelopeBuild does)
################################################################################
a <- diag(A)
omega <- (sqrt(2) - exp(-1.20491 - 0.7321 * sqrt(0.5 + a))) / sqrt(1 + a)
b2 <- as.vector(bstar2)
G1 <- rbind(b2 - omega, b2, b2 + omega)

size_info <- EnvelopeSize(a, G1, Gridtype = 3L, n = n)
G2 <- size_info$G2

G3 <- as.matrix(do.call(expand.grid, G2))
G4 <- t(G3)

################################################################################
# EnvelopeEval: negative log-likelihood and gradients at grid points
################################################################################
eval_out <- EnvelopeEval(
  G4 = G4,
  y = y,
  x = as.matrix(x2),
  mu = as.matrix(mu2, ncol = 1),
  P = as.matrix(P2),
  alpha = as.vector(alpha),
  wt = as.vector(wt2),
  family = "binomial",
  link = "logit",
  use_opencl = FALSE,
  verbose = FALSE
)

eval_out$NegLL
eval_out$cbars

################################################################################
# End of EnvelopeEval example
################################################################################
```

`EnvelopeOrchestrator` *Envelope Construction Orchestrator for Bayesian Gaussian Regression*

Description

`EnvelopeOrchestrator()` provides a unified interface for constructing the fixed-dispersion and dispersion-aware envelopes used in likelihood-subgradient simulation for Bayesian Gaussian regression with Normal–Gamma priors.

This function coordinates:

- fixed-dispersion envelope construction via EnvelopeBuild,
- dispersion-refined envelope construction via EnvelopeDispersionBuild,

- envelope sorting and reindexing via EnvelopeSort, and
- UB-list alignment (reordered `lg_prob_factor` and `UB2min`).

It is typically used inside *.cpp routines such as `rIndepNormalGammaReg()`, but may also be called directly for diagnostics, envelope visualization, or custom simulation workflows.

Usage

```
EnvelopeOrchestrator(
  bstar2,
  A,
  y,
  x2,
  mu2,
  P2,
  alpha,
  wt,
  n,
  Gridtype,
  n_envopt,
  shape,
  rate,
  RSS_Post2,
  RSS_ML,
  max_disp_perc,
  disp_lower,
  disp_upper,
  use_parallel = TRUE,
  use_opencl = FALSE,
  verbose = FALSE
)
```

Arguments

`bstar2`	Numeric vector. Posterior mode of the standardized regression coefficients (from the standardized model).
`A`	Numeric matrix. Posterior precision matrix (Hessian) at the mode.
`y`	Numeric response vector of length `m`.
`x2`	Numeric matrix of standardized predictors (`m` × `p`).
`mu2`	Numeric vector. Standardized prior mean (typically a zero vector).
`P2`	Numeric matrix. Standardized prior precision component moved into the log-likelihood.
`alpha`	Numeric vector. Offset-adjusted mean component.
`wt`	Numeric vector of prior weights.
`n`	Integer. Number of envelope grid points or simulation draws.
`Gridtype`	Integer specifying the envelope grid construction method for API compatibility. The C++ orchestrator **overrides** this to `3L` (full 3^p grid): unknown dispersion does not use smaller grids.

n_envopt	Optional integer. Effective sample size passed to EnvelopeOpt during grid construction. Larger values encourage tighter envelopes.
shape	Numeric. Shape parameter of the Gamma prior for the dispersion.
rate	Numeric. Rate parameter of the Gamma prior for the dispersion.
RSS_Post2	Numeric. Expected posterior weighted RSS used to anchor the dispersion axis (typically centering_out$RSS_post from EnvelopeCentering inside rindepNormalGamma_reg; see vignette Chapter-A11).
RSS_ML	Numeric. Maximum-likelihood residual sum of squares.
max_disp_perc	Numeric in (0,1). Tail probability used to determine dispersion bounds when not explicitly supplied.
disp_lower	Optional numeric. Lower bound for the dispersion (σ^2). If supplied, overrides quantile-based bounds.
disp_upper	Optional numeric. Upper bound for the dispersion (σ^2). Must be strictly greater than disp_lower.
use_parallel	Logical. Whether to allow parallel computation inside EnvelopeDispersionBuild.
use_opencl	Logical. Whether to allow OpenCL acceleration inside EnvelopeBuild.
verbose	Logical. Whether to print detailed progress and timing messages.

Details

EnvelopeOrchestrator() is the **envelope-construction stage** for Bayesian **Gaussian regression with an independent Normal–Gamma prior** on (β, ϕ) (dispersion ϕ; precision $\tau = 1/\phi$ in much of the theory). It is implemented in 'src/EnvelopeOrchestrator.cpp' and composes direct C++ calls to EnvelopeBuild and EnvelopeDispersionBuild with an R call to EnvelopeSort.

What this function does not do. It does **not** run the iterative dispersion centering loop (EnvelopeCentering), **not** optimize the posterior mode or Hessian, **not** standardize the model (glmb_Standardize_Model), and **not** draw posterior samples. Those steps are performed by rindepNormalGamma_reg (see vignette Chapter-A11) before and after the orchestrator. Inputs such as bstar2, A, x2, mu2, and P2 must therefore already be in **standard form** for the coefficient subproblem, exactly as passed from that workflow.

What the return value is for. The returned Env, gamma_list, and UB_list are consumed by the internal standardized samplers rIndepNormalGammaReg_std and rIndepNormalGammaReg_std_parallel in 'src/rIndepNormalGammaReg.cpp', which implement the joint accept–reject procedure over (β, ϕ). Theory for the dispersion envelope and bounding arguments is in vignette Chapter-A07; the end-to-end implementation map is in Chapter-A11. The coefficient-only likelihood-subgradient envelope ((Nygren and Nygren 2006)) is documented under EnvelopeBuild and vignette Chapter-A08.

The function does **not** perform simulation. Simulation is carried out afterward via .rIndepNormalGammaReg_std_cpp() or .rIndepNormalGammaReg_std_parallel_cpp(), depending on use_parallel.

Value

A list with components:

Env The fully constructed and sorted envelope, including the PLSD component inserted by the dispersion-aware refinement step.

`gamma_list` Updated Gamma-prior parameters for the dispersion (shape, rate, and dispersion bounds).

`UB_list` Updated UB-list including reordered `lg_prob_factor` and `UB2min`.

`diagnostics` Diagnostic quantities returned by EnvelopeDispersionBuild, useful for debugging or envelope visualization.

`low` Lower dispersion bound used.

`upp` Upper dispersion bound used.

Use of the envelope during sampling

After `EnvelopeOrchestrator()` returns, `rindepNormalGamma_reg` delegates iid simulation to `rIndepNormalGammaReg_st` (serial) or `rIndepNormalGammaReg_std_parallel` (parallel). These routines are **not exported**; they are the direct analogues of the fixed-dispersion path `.rNormalGLM_std_cpp()` for GLMs, but for the **joint** posterior $\pi(\beta, \phi \mid y)$ under the independent Normal–Gamma prior.

Dominating proposal (conceptual). The envelope list `Env` still describes a **mixture of restricted multivariate Normal** proposal pieces for the **standardized** regression coefficients, with mixture weights $\tilde{p}_j$ stored in `PLSD`. After `EnvelopeDispersionBuild`, those weights and the per-face constants are adjusted so that, together with a **truncated inverse-Gamma** (dispersion) proposal derived from `gamma_list`, the joint proposal dominates the target posterior on the truncated dispersion interval `[low, upp]`. `vignette("Chapter-A07", package = "glmbayes")` derives the dispersion-related bounds; `vignette("Chapter-A11", package = "glmbayes")` records how `UB_list` entries enter the code.

One accept–reject iteration (standardized coordinates) proceeds as follows:

1. **Draw a mixture component (face)** J. An index J is drawn from the discrete distribution with probabilities `PLSD`.
2. **Propose coefficients** $\beta^\star$. Conditional on J, each coordinate is drawn from the **restricted Normal** used in the fixed-dispersion construction: cumulative-normal tail probabilities `loglt[J, ]`, `logrt[J, ]`, and subgradient shift `-cbars[J, ]` (internal `rnorm_ct` truncated Normal sampling, same structural role as `ctrnorm_cpp()` in the GLM path).
3. **Propose dispersion** ϕ. A draw is taken from the truncated inverse-Gamma / Gamma piece defined by `shape3`, `rate2`, `disp_lower`, and `disp_upper` in `gamma_list` (`rinvgamma_ct_safe`).
4. **Re-weight the likelihood for** ϕ. Observation weights in the Gaussian log-likelihood are scaled by $1/\phi$ (`wt2 = wt / dispersion` in the C++ sources).
5. **Dispersion-adjusted tangency.** Because the tangency point for the linear upper bound depends on dispersion, the code recomputes a **face-specific** $\bar{\theta}_J(\phi)$ via `Inv_f3_with_disp` (using a one-time cache from `Inv_f3_precompute_disp` built from `cbars` and the data). The negative log-likelihood at that point feeds the **UB1** tangent term.
6. **Log-likelihood at the proposal.** Compute $-\log f(y \mid \beta^\star, \phi)$ with the same ϕ and scaled weights (output `LL_Test` in the serial implementation).

Acceptance inequality (structure). Write $\ell(\beta, \phi)$ for the Gaussian log-likelihood (weighted, with offset). The serial sampler forms UB1 from the tangent to $-\ell$ at $\bar{\theta}_J(\phi)$ along subgradient $c_J =$`cbars[J, ]`:

$$\mathrm{UB1} = -\ell\big(\bar{\theta}_J(\phi), \phi\big) - c_J^\top \big(\beta^\star - \bar{\theta}_J(\phi)\big).$$

Additional terms bound RSS variation along ϕ (**UB2**, using `RSS_Min` and `UB2min` from `UB_list`) and dispersion-axis majorization (**UB3A**, **UB3B**) built from `lg_prob_factor`, `lmc1`, `lmc2`, `lm_log1`,

`lm_log2`, `max_New_LL_UB`, and `max_LL_log_disp`. With $L_{\text{test}} = -\ell(\beta^\star, \phi)$, define $T_1 = L_{\text{test}} - \text{UB1}$ and $T = T_1 - (\text{UB2} + \text{UB3A} + \text{UB3B})$. The code draws $U_2 \sim \text{Unif}(0,1)$ and **accepts** $(\beta^\star, \phi)$ when

$$T - \log(U_2) \geq 0.$$

Serial and parallel workers use the same logical decomposition up to implementation detail. Under the construction in `EnvelopeDispersionBuild`, the terms are arranged so that $T_1 \leq 0$ and UB2, UB3A, UB3B are nonnegative up to controlled numerical slack. Iteration counts are stored in `iters_out`.

Mapping orchestrator outputs to the sampler.

- `Env$PLSD`: mixture probabilities over envelope faces for Step 1.
- `Env$loglt`, `Env$logrt`, `Env$cbars`: restricted Normal proposal for $\beta^\star$ in Step 2.
- `Env$GridIndex`, `Env$thetabars`, `Env$logU`, `Env$logP`: same role as in `EnvelopeBuild` for the coefficient mixture; dispersion refinement may update `PLSD` before sorting.
- `gamma_list`: truncated dispersion proposal parameters (`shape3`, `rate2`, bounds) for Step 3.
- `UB_list`: global and per-face constants (`RSS_Min`, `UB2min`, `lg_prob_factor`, linear `lmc`/`lm_log` pieces) for UB2, UB3A, UB3B.
- `low`, `upp`: dispersion interval endpoints (duplicated from `gamma_list` for convenience).

Unlike the fixed-dispersion GLM sampler, this path does **not** apply the stored `LLconst` vector directly in the acceptance test; the tangent piece is recomputed as UB1 once ϕ and $\bar{\theta}_J(\phi)$ are known.

Algorithmic steps

The orchestrator implements the **independent Normal–Gamma** envelope pipeline: first a **coefficient** envelope at a **dispersion anchor** ((Nygren and Nygren 2006); vignette `Chapter-A08`), then **dispersion-aware** refinement (`Chapter-A07`), then sorting. Steps 3–8 repeat the internal logic of `EnvelopeBuild` (same formulas on that help page); here the likelihood is **Gaussian** with **identity** link, weights are $w_i/d_\star$ with $d_\star$ from the anchor below, and the first pass uses `sortgrid = FALSE` so sorting runs **after** dispersion refinement.

1. **Force full grid for unknown dispersion.** The argument `Gridtype` is overridden to `3L` so the coefficient grid always uses the full 3^p partition (implementation policy in '`src/EnvelopeOrchestrator.cpp`').
2. **Anchor dispersion and rescale weights for** `EnvelopeBuild`. Let $n_w = \sum_i w_i$. With prior hyperparameters `shape` (a_0) and `rate` (b_0) and centered RSS `RSS_Post2`, define $s = a_0 + n_w/2$ and $r = b_0 + \text{RSS}_{\text{post}}/2$ where RSS_{post} denotes `RSS_Post2` (the C++ code names the scalars `shape2` and `rate3`). The dispersion anchor is $d_\star = r/(s-1)$, and observation weights w_i passed into the embedded `EnvelopeBuild` call are scaled by $1/d_\star$. This ties the coefficient envelope to the Gamma posterior for the precision conditional on the centered RSS (Chapters A07, A11).
3. **Compute width parameters ω_i from the diagonal precision matrix.** Let θ^* be the standardized posterior mode. For each dimension i,

$$\omega_i := \frac{\sqrt{2} - \exp\big(-1.20491 - 0.7321\sqrt{0.5 - \partial^2 \log f(\theta^* \mid y)/\partial\theta_i^2}\big)}{\sqrt{1 - \partial^2 \log f(\theta^* \mid y)/\partial\theta_i^2}}.$$

Here f is the weighted Gaussian log-posterior for β at the anchored dispersion.

4. **Construct intervals and the 3^p partition** around $\theta^\star$. Set

$$\ell_{i,1} = \theta_i^* - 0.5\,\omega_i, \quad \ell_{i,2} = \theta_i^* + 0.5\,\omega_i,$$

and

$$A_{i,1} = (-\infty, \ell_{i,1}), \quad A_{i,2} = [\ell_{i,1}, \ell_{i,2}], \quad A_{i,3} = (\ell_{i,2}, \infty).$$

With $J = \prod_{i=1}^{p}\{1,2,3\}$ and $j = (j_1, \ldots, j_p)$, $A_j^* = \prod_{i=1}^{p} A_{i,j_i}$ partitions standardized coefficient space.

5. **Select tangency points $\bar{\theta}_j$ per cell** (left / mode / right of each interval). For index sets C_{j1}, C_{j2}, C_{j3} by coordinate,

$$\bar{\theta}_{j,i} = \begin{cases} \theta_i^* - \omega_i, & i \in C_{j1}, \\ \theta_i^*, & i \in C_{j2}, \\ \theta_i^* + \omega_i, & i \in C_{j3}. \end{cases}$$

6. **Evaluate negative log-likelihood and gradient at each grid point.** Subgradients $c(\bar{\theta}_j)$ and negative log-likelihoods define the likelihood-subgradient envelope pieces ((Nygren and Nygren 2006); `Chapter-A08`). CPU: `f2_f3_non_opencl`; GPU (optional): `f2_f3_opencl`.
7. **Call** `EnvelopeSet_Grid_C2_pointwise` **and** `EnvelopeSet_LogP_C2` **(C++ pipeline)** to obtain restricted Normal log-densities, mixture log-probabilities, and constants as in Remarks 5–6 of the JASA paper (same as `EnvelopeBuild`).
8. **Normalize to** `PLSD` **without sorting.** The embedded `EnvelopeBuild` call sets `sortgrid = FALSE` so an intermediate sort is not wasted before `EnvelopeDispersionBuild` revises mixture weights for the joint (β, ϕ) target and `EnvelopeSort` runs once at the end.
9. **Call** `EnvelopeDispersionBuild` **(C++).** Pass the coefficient envelope list, prior `shape`, `rate`, standardized `P2`, data `y`, `x2`, `alpha`, `mu2`, `wt`, `RSS_Post2`, `RSS_ML`, dispersion controls (`max_disp_perc`, optional bounds), and `use_parallel`. This constructs the dispersion truncation interval, updates Gamma proposal parameters (`gamma_list`), computes `UB_list`, and returns `Env_out` with adjusted `PLSD`. See `Chapter-A07` and `Chapter-A11`, Section 3.3.
10. **Call** `EnvelopeSort` **(R).** Reorder envelope components and align `lg_prob_factor` and `UB2min` with the sorted indexing. If sorting cannot allocate safely, the implementation falls back to unsorted `Env_out` with UB fields patched (‘`src/EnvelopeOrchestrator.cpp`’).
11. **Return** `Env`, `gamma_list`, `UB_list`, `diagnostics`, `low`, and `upp` for the standardized samplers.

References

Nygren K~N, Nygren L~M (2006). “Likelihood Subgradient Densities.” *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

See Also

- EnvelopeBuild – fixed-dispersion envelope construction
- EnvelopeDispersionBuild – dispersion-aware envelope refinement
- EnvelopeSort – envelope sorting and reindexing
- EnvelopeCentering – `RSS_Post2` and dispersion anchor

- glmb_Standardize_Model – standardized inputs for the orchestrator
- rindepNormalGamma_reg – full Normal–Gamma workflow (R + C++)
- rlmb, rglmb, simfuncs – higher-level sampling entry points
- Vignettes Chapter-A07, Chapter-A08, Chapter-A11; cited as (Nygren and Nygren 2006; Nygren 2025, 2025)

Examples

```
############################## Start of EnvelopeOrchestrator example ###################

# This example demonstrates calling EnvelopeOrchestrator directly for Gaussian
# regression with an independent Normal-Gamma prior. It mirrors the algorithm
# path used inside rIndepNormalGammaReg:
#   - Step A: Initial dispersion via weighted lm.wfit residual variance
#   - Step B: EnvelopeCentering loop (closed-form expected RSS, update
#             dispersion2 via Gamma posterior) to anchor the envelope
#   - Step C: Coefficient posterior mode optimization (optim + f2/f3)
#   - Step D: Standardize the model (glmb_Standardize_Model)
#   - Step E: EnvelopeOrchestrator (EnvelopeBuild + EnvelopeDispersionBuild
#             + EnvelopeSort in one call)
# It stops after envelope construction (no sampling).

ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)

ps <- Prior_Setup(weight ~ group, gaussian())

x <- as.matrix(ps$x)
y <- as.vector(ps$y)
mu <- ps$mu
Sigma <- ps$Sigma
shape <- ps$shape
rate <- ps$rate

n_obs <- length(y)
wt <- rep(1, n_obs)
offset2 <- rep(0, n_obs)

# Reconstruct coefficient precision P (matches rindepNormalGamma_reg)
Rchol <- chol(Sigma)
Pinv <- chol2inv(Rchol)
P <- 0.5 * (Pinv + t(Pinv))

famfunc <- glmbfamfunc(gaussian())
f2 <- famfunc$f2
f3 <- famfunc$f3

Gridtype_core <- as.integer(2)

################################################################################
```

```
# Step A/B: EnvelopeCentering (starting at weighted lm.wfit dispersion and
# iteratively refining it via closed-form expected RSS)
##############################################################################
centering <- EnvelopeCentering(
  y = as.vector(y),
  x = as.matrix(x),
  mu = as.vector(mu),
  P = as.matrix(P),
  offset = as.vector(offset2),
  wt = as.vector(wt),
  shape = shape,
  rate = rate,
  Gridtype = Gridtype_core,
  verbose = FALSE
)

dispersion2 <- centering$dispersion
RSS_Post2   <- centering$RSS_post

##############################################################################
# Step C: Coefficient posterior mode optimization (optim + f2/f3)
##############################################################################
dispstar <- dispersion2
wt2_opt <- wt / dispstar
alpha <- as.vector(x %*% as.vector(mu) + offset2)

mu2_opt <- rep(0, length(as.vector(mu)))
parin <- rep(0, length(as.vector(mu)))

opt_out <- optim(
  par = parin,
  fn = f2,
  gr = f3,
  y = as.vector(y),
  x = as.matrix(x),
  mu = as.vector(mu2_opt),
  P = as.matrix(P),
  alpha = as.vector(alpha),
  wt = as.vector(wt2_opt),
  method = "BFGS",
  hessian = TRUE
)

bstar <- opt_out$par
A1 <- opt_out$hessian

##############################################################################
# Step D: Standardize model (glmb_Standardize_Model)
##############################################################################
Standard_Mod <- glmb_Standardize_Model(
  y = as.vector(y),
  x = as.matrix(x),
  P = as.matrix(P),
```

```
  bstar = as.matrix(bstar, ncol = 1),
  A1 = as.matrix(A1)
)

bstar2 <- Standard_Mod$bstar2
A <- Standard_Mod$A
x2_std <- Standard_Mod$x2
mu2_std <- Standard_Mod$mu2
P2_std <- Standard_Mod$P2

############################################################################
# Step E: EnvelopeOrchestrator (EnvelopeBuild + EnvelopeDispersionBuild
#         + EnvelopeSort in one call)
############################################################################
max_disp_perc <- 0.99
n_env <- as.integer(200)
Gridtype_env <- as.integer(3)  # EnvelopeOrchestrator overrides to 3 for unknown dispersion

env_out <- EnvelopeOrchestrator(
  bstar2 = as.vector(bstar2),
  A = as.matrix(A),
  y = as.vector(y),
  x2 = as.matrix(x2_std),
  mu2 = as.matrix(mu2_std, ncol = 1),
  P2 = as.matrix(P2_std),
  alpha = as.vector(alpha),
  wt = as.vector(wt),
  n = n_env,
  Gridtype = Gridtype_env,
  n_envopt = as.integer(1),
  shape = shape,
  rate = rate,
  RSS_Post2 = RSS_Post2,
  RSS_ML = NA_real_,
  max_disp_perc = max_disp_perc,
  disp_lower = NULL,
  disp_upper = NULL,
  use_parallel = TRUE,
  use_opencl = FALSE,
  verbose = FALSE
)

# Output structure matches that from the step-by-step Ex_EnvelopeDispersionBuild
print(env_out$low)
print(env_out$upp)
print(env_out$gamma_list[c("shape3", "rate2")])

env_out

############################################################################
# End: envelope construction only
############################################################################
```

EnvelopeSize *Envelope Sizing and Optimization*

Description

EnvelopeSize() is the high-level entry point that constructs per-dimension grids and expected draw counts, while EnvelopeOpt() performs the adaptive optimization used when Gridtype = 2.

Usage

```
EnvelopeSize(a, G1, Gridtype = 2L, n = 1000L, n_envopt = -1,
                    use_opencl = FALSE, verbose = FALSE)

EnvelopeOpt(a1,n,core_cnt=1L)
```

Arguments

a	Numeric vector of diagonal precisions for the log-likelihood (posterior precision is $1 + a_i$).
G1	Numeric matrix of candidate grid points (3 * 11).
Gridtype	Integer code controlling grid sizing logic: • 1 = static threshold test • 2 = adaptive optimization via EnvelopeOpt() • 3 = always three-point grid • 4 = always single-point grid
n	Integer; number of posterior draws to generate (used for grid sizing).
n_envopt	Integer; effective sample size passed to EnvelopeOpt. Defaults to -1, which means "use n".
use_opencl	Logical; if TRUE, attempt GPU acceleration.
verbose	Logical; if TRUE, print progress messages.
a1	Numeric vector of diagonal elements of the data precision matrix (used by EnvelopeOpt).
core_cnt	Integer; number of OpenCL cores or parallel workers available (default 1). When >1, envelope build cost is scaled down to reflect parallel construction.

Details

These functions implement the grid sizing logic used in envelope construction for rejection sampling. They make use of the theory described in (Nygren and Nygren 2006) and the general implementation outlined in (Nygren 2025).

EnvelopeSize() returns the constructed grid (G2), index vectors (GIndex1), expected draw count (E_draws), and the per-dimension grid index.

EnvelopeOpt() implements the adaptive optimization used in Gridtype = 2, ranking dimensions by posterior variance and promoting them to three-point tangents when the tradeoff is favorable.

Value

`EnvelopeSize()` A list with components `G2`, `GIndex1`, `E_draws`, and `gridindex`.

`EnvelopeOpt()` An integer vector of length $l1$ with entries 1 (single-point) or 3 (three-point).

Gridtype Logic and Candidates per Draw

The envelope sizing logic follows the analysis of (Nygren and Nygren 2006).

Gridtype 1: Static Threshold For each dimension i, if $\sqrt{1+a_i} \leq 2/\sqrt{\pi} \approx 1.128379$, then a single tangent at the posterior mode suffices. Expected candidates per draw in that dimension: $\sqrt{1+a_i}$. Otherwise, a symmetric three-point envelope is used at $(\theta_i^\star - \omega_i, \theta_i^\star, \theta_i^\star + \omega_i)$, with expected candidates per draw bounded above by $2/\sqrt{\pi}$.

Gridtype 2: Adaptive Optimization Each dimension is assigned either a single-point or three-point envelope by minimizing

$$T_{\text{total}}(g_i) = T_{\text{build}}(g_i) + T_{\text{sample}}(n, acc_i(g_i)).$$

The optimizer balances build cost (grows with number of tangents) against sampling cost (decreases as acceptance improves). Expected candidates per draw: $\prod_j \text{scaleest}_{i,j}$, where each factor is either $\sqrt{1+a_j}$ (single-point) or $2/\sqrt{\pi}$ (three-point), depending on the optimization outcome.

Gridtype 3: Always Three-Point Every dimension uses a symmetric three-point envelope. Expected candidates per draw:

$$\left(\tfrac{2}{\sqrt{\pi}}\right)^k$$

for k dimensions, as shown in Theorem 3 of (Nygren and Nygren 2006).

Gridtype 4: Always Single-Point Every dimension uses a single tangent at the posterior mode. Expected candidates per draw:

$$\prod_{i=1}^{k} \sqrt{1+a_i}$$

(Example 1 in (Nygren and Nygren 2006)).

References

Nygren K (2025). "Chapter A05: Simulation Methods – Likelihood Subgradient Densities." Vignette in the glmbayes R package. R vignette name: Chapter-A05.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

See Also

`EnvelopeBuild`, `EnvelopeEval`, `EnvelopeSort`; `rNormal_reg`, `rglmb` for user-facing sampling that uses these grids. Vignettes: (Nygren 2025, 2025).

Examples

```
data(menarche,package="MASS")

summary(menarche)
plot(Menarche/Total ~ Age, data=menarche)

Age2=menarche$Age-13

x<-matrix(as.numeric(1.0),nrow=length(Age2),ncol=2)
x[,2]=Age2

y=menarche$Menarche/menarche$Total
wt=menarche$Total

mu<-matrix(as.numeric(0.0),nrow=2,ncol=1)
mu[2,1]=(log(0.9/0.1)-log(0.5/0.5))/3

V1<-1*diag(as.numeric(2.0))

# 2 standard deviations for prior estimate at age 13 between 0.1 and 0.9
## Specifies uncertainty around the point estimates

V1[1,1]<-((log(0.9/0.1)-log(0.5/0.5))/2)^2
V1[2,2]=(3*mu[2,1]/2)^2  # Allows slope to be up to 1 times as large as point estimate

famfunc<-glmbfamfunc(binomial(logit))

f1<-famfunc$f1
f2<-famfunc$f2
f3<-famfunc$f3
f5<-famfunc$f5
f6<-famfunc$f6

dispersion2<-as.numeric(1.0)
start <- mu
offset2=rep(as.numeric(0.0),length(y))
P=solve(V1)
n=1000



###### Adjust weight for dispersion

wt2=wt/dispersion2

######################### Shift mean vector to offset so that adjusted model has 0 mean

alpha=x%*%as.vector(mu)+offset2
mu2=0*as.vector(mu)
P2=P
x2=x
```

```
#####  Optimization step to find posterior mode and associated Precision

parin=start-mu

opt_out=optim(parin,f2,f3,y=as.vector(y),x=as.matrix(x),mu=as.vector(mu2),
              P=as.matrix(P),alpha=as.vector(alpha),wt=as.vector(wt2),
              method="BFGS",hessian=TRUE
)

bstar=opt_out$par  ## Posterior mode for adjusted model
bstar
bstar+as.vector(mu)  # mode for actual model
A1=opt_out$hessian # Approximate Precision at mode

## Standardize Model

Standard_Mod=glmb_Standardize_Model(y=as.vector(y), x=as.matrix(x),
P=as.matrix(P),bstar=as.matrix(bstar,ncol=1), A1=as.matrix(A1))

bstar2=Standard_Mod$bstar2
A=Standard_Mod$A
x2=Standard_Mod$x2
mu2=Standard_Mod$mu2
P2=Standard_Mod$P2
L2Inv=Standard_Mod$L2Inv
L3Inv=Standard_Mod$L3Inv

## Derive a and G1 (as EnvelopeBuild does internally)
a <- diag(A)
omega <- (sqrt(2) - exp(-1.20491 - 0.7321*sqrt(0.5 + a))) / sqrt(1 + a)
b2 <- as.vector(bstar2)
G1 <- rbind(b2 - omega, b2, b2 + omega)

## EnvelopeOpt: standalone call (used by EnvelopeSize when Gridtype=2)
grid_opt <- EnvelopeOpt(a, n)
grid_opt

## EnvelopeSize for each Gridtype
size_1 <- EnvelopeSize(a, G1, Gridtype=1L, n=n)   # static threshold
size_2 <- EnvelopeSize(a, G1, Gridtype=2L, n=n)   # uses EnvelopeOpt
size_3 <- EnvelopeSize(a, G1, Gridtype=3L, n=n)   # always 3-point
size_4 <- EnvelopeSize(a, G1, Gridtype=4L, n=n)   # always single-point

## EnvelopeBuild for each Gridtype
Env_1 <- EnvelopeBuild(as.vector(bstar2), as.matrix(A), y, as.matrix(x2),
  as.matrix(mu2,ncol=1), as.matrix(P2), as.vector(alpha), as.vector(wt2),
  family="binomial", link="logit", Gridtype=1L, n=as.integer(n), sortgrid=FALSE)
Env_2 <- EnvelopeBuild(as.vector(bstar2), as.matrix(A), y, as.matrix(x2),
  as.matrix(mu2,ncol=1), as.matrix(P2), as.vector(alpha), as.vector(wt2),
  family="binomial", link="logit", Gridtype=2L, n=as.integer(n), sortgrid=FALSE)
Env_3 <- EnvelopeBuild(as.vector(bstar2), as.matrix(A), y, as.matrix(x2),
  as.matrix(mu2,ncol=1), as.matrix(P2), as.vector(alpha), as.vector(wt2),
```

```
  family="binomial", link="logit", Gridtype=3L, n=as.integer(n), sortgrid=FALSE)
Env_4 <- EnvelopeBuild(as.vector(bstar2), as.matrix(A), y, as.matrix(x2),
  as.matrix(mu2,ncol=1), as.matrix(P2), as.vector(alpha), as.vector(wt2),
  family="binomial", link="logit", Gridtype=4L, n=as.integer(n), sortgrid=FALSE)

Env_3
```

`EnvelopeSort` *Sorts Envelope function for simulation*

Description

Sorts Enveloping function for simulation. how frequently each component of the resulting grid should be sampled during simulation.

Usage

```
EnvelopeSort(
  l1,
  l2,
  GIndex,
  G3,
  cbars,
  logU,
  logrt,
  loglt,
  logP,
  LLconst,
  PLSD,
  a1,
  E_draws,
  lg_prob_factor = NULL,
  UB2min = NULL
)
```

Arguments

`l1`	dimension for model (number of independent variables in X matrix)
`l2`	dimension for Envelope (number of components)
`GIndex`	matrix containing information on how each dimension should be sampled (1 means left tail of a restricted normal, 2 center, 3 right tail, and 4 the entire line)
`G3`	A matrix containing the points of tangencies associated with each component of the grid
`cbars`	A matrix containing the gradients for the negative log-likelihood at each tangency

logU	A matrix containing the log of the cummulative probability associated with each dimension
logrt	A matrix containing the log of the probability associated with the right tail (i.e. that to the right of the lower bound)
loglt	A matrix containing the log of the probability associated with the left tail (i.e., that to the left of the upper bound)
logP	A matrix containing log-probabilities related to the components of the grid
LLconst	A vector containing constant for each component of the grid used during the accept-reject procedure
PLSD	A vector containing the probability of each component in the Grid
a1	A vector containing the diagonal of the standardized precision matrix
E_draws	Bound on Expected number of candidates per accepted draw
lg_prob_factor	vector of lg_prob_factors used for the Envelope connected to the independent normal gamma prior
UB2min	Vector containing min for UB2 for each component (relevant for EnvelopeDispersionBuild)

Details

This function sorts the envelope in descending order based on the probability associated with each component in the Grid. Sorting helps speed up simulation once the envelope is constructed. If memory allocation fails (e.g. for very large grids), the function returns the unsorted envelope with sort_ok = FALSE; the sampler remains valid but may have poorer acceptance.

Used after EnvelopeBuild and (for Normal–Gamma models) EnvelopeDispersionBuild; see (Nygren and Nygren 2006; Nygren 2025).

Value

The function returns a list consisting of the following components (the first six of which are matrics with number of rows equal to the number of components in the Grid and columns equal to the number of parameters):

GridIndex	A matrix containing information on how each dimension should be sampled (1 means left tail of a restricted normal, 2 center, 3 right tail, and 4 the entire line)
thetabars	A matrix containing the points of tangencies associated with each component of the grid
cbars	A matrix containing the gradients for the negative log-likelihood at each tangency
logU	A matrix containing the log of the cummulative probability associated with each dimension
logrt	A matrix containing the log of the probability associated with the right tail (i.e. that to the right of the lower bound)
loglt	A matrix containing the log of the probability associated with the left tail (i.e., that to the left of the upper bound)

LLconst	A vector containing constant for each component of the grid used during the accept-reject procedure
logP	A matrix containing log-probabilities related to the components of the grid
PLSD	A vector containing the probability of each component in the Grid
E_draws	A containing a computed theoretical bound on the expected number of draws
sort_ok	Logical; TRUE if sort succeeded, FALSE if memory allocation failed and unsorted envelope was returned (sampler remains valid)

References

Nygren K (2025). "Chapter A08: Overview of Envelope Related Functions." Vignette in the glmbayes R package. R vignette name: Chapter-A08.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

See Also

EnvelopeBuild, EnvelopeOrchestrator, EnvelopeDispersionBuild, rNormal_reg, rglmb.

Examples

```
data(menarche,package="MASS")
Age2=menarche$Age-13

summary(menarche)
plot(Menarche/Total ~ Age, data=menarche)


x<-matrix(as.numeric(1.0),nrow=length(Age2),ncol=2)
x[,2]=Age2

y=menarche$Menarche/menarche$Total
wt=menarche$Total

mu<-matrix(as.numeric(0.0),nrow=2,ncol=1)
mu[2,1]=(log(0.9/0.1)-log(0.5/0.5))/3

V1<-1*diag(as.numeric(2.0))

# 2 standard deviations for prior estimate at age 13 between 0.1 and 0.9
## Specifies uncertainty around the point estimates

V1[1,1]<-((log(0.9/0.1)-log(0.5/0.5))/2)^2
V1[2,2]=(3*mu[2,1]/2)^2  # Allows slope to be up to 1 times as large as point estimate

famfunc<-glmbfamfunc(binomial(logit))

f1<-famfunc$f1
f2<-famfunc$f2
```

```
f3<-famfunc$f3
f5<-famfunc$f5
f6<-famfunc$f6

dispersion2<-as.numeric(1.0)
start <- mu
offset2=rep(as.numeric(0.0),length(y))
P=solve(V1)
n=1000

## Appears that the type for some of these arguments are important/problematic

### This example constructs and sorts an envelope without calling the
### lower-level sampler directly.


###### Adjust weight for dispersion

wt2=wt/dispersion2

######################### Shift mean vector to offset so that adjusted model has 0 mean

alpha=x%*%as.vector(mu)+offset2
mu2=0*as.vector(mu)
P2=P
x2=x


#####  Optimization step to find posterior mode and associated Precision

parin=start-mu

opt_out=optim(parin,f2,f3,y=as.vector(y),x=as.matrix(x),mu=as.vector(mu2),
              P=as.matrix(P),alpha=as.vector(alpha),wt=as.vector(wt2),
              method="BFGS",hessian=TRUE
)

bstar=opt_out$par  ## Posterior mode for adjusted model
bstar
bstar+as.vector(mu)  # mode for actual model
A1=opt_out$hessian # Approximate Precision at mode

## Standardize Model

Standard_Mod=glmb_Standardize_Model(y=as.vector(y), x=as.matrix(x),
P=as.matrix(P),bstar=as.matrix(bstar,ncol=1), A1=as.matrix(A1))

bstar2=Standard_Mod$bstar2
A=Standard_Mod$A
x2=Standard_Mod$x2
mu2=Standard_Mod$mu2
P2=Standard_Mod$P2
L2Inv=Standard_Mod$L2Inv
```

```
L3Inv=Standard_Mod$L3Inv

Env2=EnvelopeBuild(as.vector(bstar2), as.matrix(A),y, as.matrix(x2),
as.matrix(mu2,ncol=1),as.matrix(P2),as.vector(alpha),as.vector(wt2),
family="binomial",link="logit",Gridtype=as.integer(3),
n=as.integer(n),sortgrid=FALSE)

## Extract l1, l2 from envelope (as done in C++) and call EnvelopeSort
l1 <- ncol(Env2$cbars)
l2 <- nrow(Env2$cbars)
logP_mat <- matrix(Env2$logP, ncol = 1)

Env_sorted <- EnvelopeSort(l1, l2,
  GIndex  = Env2$GridIndex,
  G3      = Env2$thetabars,
  cbars   = Env2$cbars,
  logU    = Env2$logU,
  logrt   = Env2$logrt,
  loglt   = Env2$loglt,
  logP    = logP_mat,
  LLconst = Env2$LLconst,
  PLSD    = Env2$PLSD,
  a1      = Env2$a1,
  E_draws = Env2$E_draws
)

Env_sorted
```

`formula.summary.rglmb` *Model Formulae for* `summary.rglmb` *Objects*

Description

Extract a formula for a `summary.rglmb` object by refitting a reference `glm` with the stored response and design matrix.

Usage

```
## S3 method for class 'summary.rglmb'
formula(x, ...)
```

Arguments

`x`	an object of class `summary.rglmb`, typically from `summary.rglmb`.
`...`	further arguments passed to or from other methods.

Value

A model formula.

See Also

rglmb, summary.rglmb, rlmb, formula.

Gamma_ct *The Central Gamma Distribution*

Description

Distribution function and random generation for the center (between a lower and an upper bound) of the Gamma distribution with shape and rate parameters. These functions provide numerically stable evaluation and sampling when the truncation interval is narrow or when the Gamma density is highly skewed.

Usage

```
rgamma_ct(n, shape, rate, lower_prec = NULL, upper_prec = NULL)
```

Arguments

n	Number of draws to generate. If length(n) > 1, the length is taken to be the number required.
shape	Shape parameter of the Gamma distribution.
rate	Rate parameter of the Gamma distribution.
lower_prec	Lower truncation point on the precision scale. If NULL, no lower truncation is applied.
upper_prec	Upper truncation point on the precision scale. If NULL, no upper truncation is applied.

Details

The function pgamma_ct computes the probability mass between a lower bound a and an upper bound b under a Gamma density with the specified shape and rate parameters. This is particularly useful when the interval b - a is small, where the naive computation pgamma(b) - pgamma(a) may underflow to zero even when the true probability is positive.

The function ctrgamma provides a numerically robust sampler for the Gamma distribution under one-sided or two-sided truncation. It handles:

- no truncation (reducing to rgamma)
- lower truncation only
- upper truncation only
- two-sided truncation with lower_prec < upper_prec

- exact degeneracy when the truncation interval collapses
- numerical degeneracy when the Gamma CDF collapses in floating point

All computations are performed on the log scale using stable log–CDF and log–sum–exp transformations. This avoids the catastrophic cancellation that occurs when the Gamma CDF values at the truncation points are extremely close.

These functions are primarily intended for use in hierarchical Bayesian models where precision parameters are updated under tight truncation constraints, and where numerical stability is essential for reliable sampling performance. They are used in envelope-based dispersion sampling (Nygren and Nygren 2006).

Value

For `pgamma_ct`, a vector of probabilities corresponding to the mass of the Gamma distribution between `a` and `b`. For `rgamma_ct` or `ctrgamma`, a vector of length `nn` containing random draws from the Gamma distribution restricted to the interval `[a, b]` (or `[lower_prec, upper_prec]` on the precision scale).

References

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

See Also

`Normal_ct`, `InvGamma_ct`, `EnvelopeDispersionBuild`

Examples

```
################################ Start of Gamma_ct example ###################

## Basic usage: rgamma_ct samples from Gamma truncated to [lower_prec, upper_prec]
shape <- 2
rate  <- 1
lower <- 0.999
upper <- 1.001

## Naive pgamma(b) - pgamma(a) can underflow when a and b are very close
pgamma(upper, shape, rate) - pgamma(lower, shape, rate)

## rgamma_ct samples correctly from the narrow interval
set.seed(42)
x <- rgamma_ct(100, shape, rate, lower_prec = lower, upper_prec = upper)
range(x)
mean(x)

## Example where difference between two pgamma calls fails (catastrophic cancellation)
## but rgamma_ct still samples correctly from the narrow interval
a <- 1.0
b <- 1.0 + 1e-14
pgamma(b, 2, 1) - pgamma(a, 2, 1)
```

```
## Stable computation via log-space (as used inside rgamma_ct)
log_F_a <- pgamma(a, 2, 1, log.p = TRUE)
log_F_b <- pgamma(b, 2, 1, log.p = TRUE)
exp(log_F_b + log(-expm1(log_F_a - log_F_b)))

## rgamma_ct samples from [a, b] even when the interval is extremely narrow
set.seed(123)
rgamma_ct(5, 2, 1, lower_prec = a, upper_prec = b)

##############################################################################
## End of Gamma_ct example
##############################################################################
```

`glmb_Standardize_Model` *Standardize A Non-Gaussian Model*

Description

Standardizes a Non-Gaussian Model prior to Envelope Creation

Usage

```
glmb_Standardize_Model(y, x, P, bstar, A1)
```

Arguments

`y`	a vector of observations of length m
`x`	a design matrix of dimension m*p
`P`	a positive-definite symmetric matrix specifying the prior precision matrix of the variables.
`bstar`	a matrix containing the posterior mode from an optimization step
`A1`	a matrix containing the posterior precision matrix at the posterior mode

Details

This functions starts with basic information about the model in the argument list and then uses the following steps to further standardize the model (the model is already assumed to have a 0 prior mean vector when this step is applied).

1. An eigenvalue composition is applied to the posterior precision matrix, and the model is (as an interim step) standardized to have a posterior precision matrix equal to the identity matrix. Please note that this means that the prior precision matrix after this step is `"smaller"` than the identity matrix.

2. A diagonal matrix epsilon is pulled out from the standardized prior precision matrix so that the remaining part of the prior precision matrix still is positive definite. That part is then treated as part of the likelihood for the rest of the standardization and simulation and only the part connected to epsilon is treated as part of the prior. Note that the exact epsilon chosen seems not to matter. Hence there are many possible ways of doing this standardization and future versions of this package may tweak the current approach if it helps improve numerical accuracy or acceptance rates.
3. The model is next standardized (using a second eigenvalue decomposition) so that the prior (i.e., the portion connected to epsilon) is the identity matrix. The standardized model then simutaneously has the feature that the prior precision matrix is the identity matrix and that the data precision A (at the posterior mode) is a diagonal matrix. Hence the variables in the standardized model are approximately independent at the posterior mode.

The steps here are based on the procedure described in (Nygren and Nygren 2006).

Value

A list with the following components

`bstar2`	Standardized Posterior Mode
`A`	Standardized Data Precision Matrix
`x2`	Standardized Design Matrix
`mu2`	Standardized Prior Mean vector
`P2`	Standardized Precision Matrix Added to log-likelihood
`L2Inv`	A matrix used when undoing the first step in standardization described below
`L3Inv`	A matrix used when undoing the second step in standardization described below

References

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

Examples

```
data(menarche,package="MASS")
Age2=menarche$Age-13

summary(menarche)
plot(Menarche/Total ~ Age, data=menarche)


x<-matrix(as.numeric(1.0),nrow=length(Age2),ncol=2)
x[,2]=Age2

y=menarche$Menarche/menarche$Total
wt=menarche$Total

mu<-matrix(as.numeric(0.0),nrow=2,ncol=1)
mu[2,1]=(log(0.9/0.1)-log(0.5/0.5))/3
```

```
V1<-1*diag(as.numeric(2.0))

# 2 standard deviations for prior estimate at age 13 between 0.1 and 0.9
## Specifies uncertainty around the point estimates

V1[1,1]<-((log(0.9/0.1)-log(0.5/0.5))/2)^2
V1[2,2]=(3*mu[2,1]/2)^2  # Allows slope to be up to 1 times as large as point estimate

famfunc<-glmbfamfunc(binomial(logit))

f1<-famfunc$f1
f2<-famfunc$f2
f3<-famfunc$f3
f5<-famfunc$f5
f6<-famfunc$f6

dispersion2<-as.numeric(1.0)
start <- mu
offset2=rep(as.numeric(0.0),length(y))
P=solve(V1)
n=1000

## Appears that the type for some of these arguments are important/problematic



###### Adjust weight for dispersion

wt2=wt/dispersion2

######################### Shift mean vector to offset so that adjusted model has 0 mean

alpha=x%*%as.vector(mu)+offset2
mu2=0*as.vector(mu)
P2=P
x2=x


#####  Optimization step to find posterior mode and associated Precision

parin=start-mu

opt_out=optim(parin,f2,f3,y=as.vector(y),x=as.matrix(x),mu=as.vector(mu2),
              P=as.matrix(P),alpha=as.vector(alpha),wt=as.vector(wt2),
              method="BFGS",hessian=TRUE
)

bstar=opt_out$par  ## Posterior mode for adjusted model
bstar
bstar+as.vector(mu)  # mode for actual model
A1=opt_out$hessian # Approximate Precision at mode
```

```
## Standardize Model

Standard_Mod=glmb_Standardize_Model(y=as.vector(y), x=as.matrix(x),P=as.matrix(P),
                                    bstar=as.matrix(bstar,ncol=1), A1=as.matrix(A1))

bstar2=Standard_Mod$bstar2
A=Standard_Mod$A
x2=Standard_Mod$x2
mu2=Standard_Mod$mu2
P2=Standard_Mod$P2
L2Inv=Standard_Mod$L2Inv
L3Inv=Standard_Mod$L3Inv
```

`glmbfamfunc` *Return family functions used during simulation and post processing*

Description

This function takes as input a `family` object and returns a set of functions that are used during simulation and summarization of models using the `rglmb` and `rlmb` functions.

Usage

```
glmbfamfunc(family, lik_shape = 1)

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

Arguments

`family`	an object of class `family`
`lik_shape`	Known shape parameter of the Gamma likelihood; used only for the `Gamma(link = "identity")` branch where the regression coefficient is the Gamma *rate*. Ignored for all other family/link combinations. Defaults to `1` (i.e.\ exponential likelihood).
`x`	an object of class `"glmbfamfunc"` for which a printed output is desired.
`...`	additional optional arguments

Details

`glmbfamfunc` is the canonical R closure bundle for likelihood, posterior, gradient, and deviance quantities across all supported family/link combinations.

Registration requirement. A branch must exist inside `glmbfamfunc` for every family/link combination that is used in the package. If a combination is missing, the closures `f1`–`f4` will never be assigned and any downstream code that accesses them will fail with *"object 'f1' not found"*. New family/link combinations must therefore be explicitly added here before they can produce valid DIC or log-likelihood output.

Currently implemented family/link combinations:

Family	Link
gaussian	identity
poisson, quasipoisson	log
poisson, quasipoisson	identity
binomial, quasibinomial	logit
binomial, quasibinomial	probit
binomial, quasibinomial	cloglog
binomial, quasibinomial	identity
Gamma	log
Gamma	identity

Any family/link not in this table will fall through all branches silently and produce the error above at the point of first use. For `Gamma(link = "identity")`, pass the known Gamma likelihood shape `lik_shape` (default 1) so that `f1`–`f4` and `f7` use the correct parameterization (coefficient = Gamma rate β).

Relationship to C++ simulation paths. Many simulation procedures in the package have been fully or partially migrated to `*.cpp` routines, which receive their own objective functions directly. For those paths `glmbfamfunc` may not be called at all during sampling. However, R-side post-processing (e.g.\ `summary.rglmb`) may still use `famfunc\$f1` and `famfunc\$f4`, so a registered branch is still required for those outputs even when the sampler itself has moved to C++.

Value

A list (class `"glmbfamfunc"`) whose first four components are **always** present for every supported `family` and `link`. The names `f1`–`f4` are stable: they mean the same roles across families (only the internal formulas change).

`f1` Negative log-likelihood as a function of coefficients `b` (arguments typically `b`, `y`, `x`, optional `alpha`, `wt`).

`f2` Negative log-posterior (likelihood plus Normal prior quadratic form in `b` with precision `P` and mean `mu`).

`f3` Gradient of `f2` with respect to `b` (same argument pattern as `f2`).

`f4` Deviance-related quantity (twice negative log-likelihood contrast vs.\ saturated model, with a `dispersion` argument for quasi-families); used in DIC-style summaries.

`f7` Family-specific matrix: weighted sum of outer products of predictor rows, i.e.\ a curvature / expected negative Hessian of the log-likelihood w.r.t.\ `b` at the supplied `b` (used e.g.\ for multivariate prior–posterior diagnostics in **glmbayes**).

Slots `f5` and `f6` are **not** returned: they were reserved for alternate or C++-aligned likelihood/posterior routines and remain commented out in the implementation (only `f1`, `f2`, `f3`, `f4`, and `f7` are assigned in the returned list).

Examples

```
famfunc <- glmbfamfunc(binomial(logit))

print(famfunc)

## f1--f4 and f7 are always present for supported families; f5 and f6 are not returned.
f1 <- famfunc$f1
f2 <- famfunc$f2
f3 <- famfunc$f3
f4 <- famfunc$f4
f7 <- famfunc$f7
stopifnot(is.function(f1), is.function(f2), is.function(f3),
          is.function(f4), is.function(f7))
```

---

glmerb_posterior_mode *Joint posterior mode of the two-block GLMM (Gaussian case)*

Description

Finds the joint posterior mode of the two-block model using an *iterated conditional modes* (ICM) algorithm. When the response model is Gaussian and variance components are fixed, the joint posterior is multivariate normal so the posterior mode equals the posterior mean; in that case `glmerb_posterior_mode()` returns the same result as `lmerb_posterior_mean`.

Usage

```
glmerb_posterior_mode(
  design,
  family = gaussian(),
  measurement_prior_list,
  tol = 1e-10,
  maxit = 200L
)
```

Arguments

`design`	Design list with `y`, `Z`, `groups`, `X_hyper`, and `re_coef_names`.
`family`	A `family` object. Defaults to `gaussian()`. Reserved for future non-Gaussian response models.
`measurement_prior_list`	List with `Sigma_ranef` and `prior_list`. `dispersion_ranef` (σ^2) is required for `family = gaussian()` and omitted otherwise. Each `prior_list[[k]]` must contain `mu_fixef`, `Sigma_fixef`, and `dispersion_fixef`.
`tol`	Convergence tolerance on the ℓ_∞ change in `fixef` between successive iterations. Default `1e-10`.
`maxit`	Maximum number of ICM iterations. Default `200L`.

Details

Algorithm. For any jointly Gaussian distribution the conditional mean of each block is an affine function of the other block's value. ICM alternates between the two closed-form conditional mean updates:

Block 1 mode For each group j, the conditional posterior mode of b_j given γ is obtained via `rglmb` with `n = 1L` and a `dNormal` prior with mean μ_j and covariance Σ_b. The mode is read from `coef.mode`. When `family = gaussian()`, this matches the closed-form normal update used in `lmerb_posterior_mean`.

Block 2 mean For each RE component k:

$$E[\gamma_k \mid b_k] = \left(X_k^\top X_k/\tau_k^2 + P_{\gamma_k}\right)^{-1}\left(X_k^\top b_k/\tau_k^2 + P_{\gamma_k}\mu_{\gamma_k}\right)$$

where b_k is the k-th column of the current Block 1 mean matrix, $X_k =$ `design$X_hyper[[k]]`, $\tau_k^2 =$ `dispersion_fixef`, and $P_{\gamma_k} = \Sigma_{\gamma_k}^{-1}$ from `prior_list`.

Value

A list with components `fixef`, `b_mean`, `converged`, `iterations`, and `delta`.

See Also

`lmerb_posterior_mean`, `rglmb`, `two_block_rNormal_reg`, `build_mu_all`

`InvGamma_ct` *The Central Inverse-Gamma Distribution*

Description

Distribution function, quantile function, and random generation for the inverse-Gamma distribution on the dispersion scale. These functions provide numerically stable evaluation and sampling when the dispersion parameter is restricted to lie between a lower and an upper bound.

Usage

```
pinvgamma_ct(dispersion, shape, rate)

qinvgamma_ct(p, shape, rate, disp_upper, disp_lower)

rinvgamma_ct(n, shape, rate, disp_upper, disp_lower)
```

Arguments

`dispersion`	Value(s) at which the inverse-Gamma distribution function is evaluated.
`shape`	Shape parameter of the inverse-Gamma distribution.
`rate`	Rate parameter of the inverse-Gamma distribution.

p	Probability value(s) for the quantile function.
disp_upper	Upper bound of the dispersion parameter.
disp_lower	Lower bound of the dispersion parameter.
n	Number of random draws to generate. If length(n) > 1, the length is taken to be the number required.

Details

The inverse-Gamma distribution is defined by the transformation $D = 1/X$, where X follows a Gamma distribution with the same shape and rate parameters. The functions pinvgamma_ct and qinvgamma_ct therefore compute probabilities and quantiles by mapping the dispersion value D to the corresponding Gamma scale and applying the Gamma CDF or quantile function.

The function rinvgamma_ct generates random draws from a truncated inverse-Gamma distribution by sampling a uniform probability and inverting the truncated CDF on the Gamma scale. This approach avoids numerical instability when the truncation interval is narrow or when the dispersion parameter is close to zero.

These functions are primarily intended for hierarchical Bayesian models in which dispersion parameters are updated under tight truncation constraints. They provide a stable alternative to direct manipulation of the Gamma distribution when working on the dispersion scale is more natural or more numerically robust. They are used in envelope-based dispersion sampling (Nygren and Nygren 2006).

Value

For pinvgamma_ct, a vector of distribution function values evaluated at dispersion. For qinvgamma_ct, a vector of quantiles corresponding to the probabilities p. For rinvgamma_ct, a vector of length n containing random draws from the inverse-Gamma distribution restricted to the interval [disp_lower, disp_upper].

References

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

See Also

Gamma_ct, Normal_ct, EnvelopeDispersionBuild

Examples

```
############################## Start of InvGamma_ct example ###################

## pinvgamma_ct: CDF on the dispersion scale
shape <- 2
rate  <- 1
pinvgamma_ct(1.5, shape, rate)

## Equivalent via pgamma on the precision scale
1 - pgamma(1 / 1.5, shape, rate)
```

```
## Example where interval mass pinvgamma_ct(disp_upper) - pinvgamma_ct(disp_lower)
## fails due to catastrophic cancellation when bounds are very close
disp_lower <- 0.999
disp_upper <- 0.999 + 1e-14
pinvgamma_ct(disp_upper, shape, rate) - pinvgamma_ct(disp_lower, shape, rate)

## On the precision scale this is pgamma(1/disp_lower) - pgamma(1/disp_upper);
## the same cancellation affects the Gamma CDF when precision bounds are close

## rinvgamma_ct samples from truncated inverse-Gamma on the dispersion scale
disp_lower <- 0.99
disp_upper <- 1.01
set.seed(42)
y <- rinvgamma_ct(100, shape = 2, rate = 1,
                  disp_upper = disp_upper, disp_lower = disp_lower)
range(y)
mean(y)

################################################################################
## End of InvGamma_ct example
################################################################################
```

`lmerb_posterior_mean` *Joint posterior mean of the two-block Gaussian model*

Description

Finds the joint posterior mean (= joint mode, since the posterior is exactly multivariate normal when variance components are fixed) of the two-block model sampled by `two_block_rNormal_reg`, using an *iterated conditional means* (ICM) algorithm.

Usage

```
lmerb_posterior_mean(design, measurement_prior_list, tol = 1e-10, maxit = 200L)
```

Arguments

`design`	Design list with `y`, `Z`, `groups`, `X_hyper`, and `re_coef_names`.
`measurement_prior_list`	List with `dispersion_ranef` (Gaussian only), `Sigma_ranef`, and `prior_list`. Each `prior_list[[k]]` must contain `mu_fixef`, `Sigma_fixef`, and `dispersion_fixef`.
`tol`	Convergence tolerance on the ℓ_∞ change in `fixef` between successive iterations. Default `1e-10`.
`maxit`	Maximum number of ICM iterations. Default `200L`.

Details

Algorithm. For any jointly Gaussian distribution the conditional mean of each block is an affine function of the other block's value. ICM alternates between the two closed-form conditional mean updates:

Block 1 mean For each group j:

$$E[b_j \mid \gamma] = \left(Z_j^\top Z_j/\sigma^2 + P_b\right)^{-1}\left(Z_j^\top y_j/\sigma^2 + P_b\,\mu_j(\gamma)\right)$$

where $P_b = \Sigma_b^{-1}$ and $\mu_j(\gamma)$ is the Block 2 prior mean from `build_mu_all`.

Block 2 mean For each RE component k:

$$E[\gamma_k \mid b_k] = \left(X_k^\top X_k/\tau_k^2 + P_{\gamma_k}\right)^{-1}\left(X_k^\top b_k/\tau_k^2 + P_{\gamma_k}\mu_{\gamma_k}\right)$$

where b_k is the k-th column of the current Block 1 mean matrix, $X_k =$ `design$X_hyper[[k]]`, $\tau_k^2 =$ `dispersion_fixef`, and $P_{\gamma_k} = \Sigma_{\gamma_k}^{-1}$ from `prior_list`.

Value

A list with components `fixef`, `b_mean`, `converged`, `iterations`, and `delta`.

See Also

`two_block_rNormal_reg`, `build_mu_all`

`multi_prior_setup` *Prior setup for multiple Gaussian responses*

Description

Prior setup for multiple Gaussian responses

Usage

```
multi_prior_setup(
  formula,
  family = gaussian(),
  data = NULL,
  weights = NULL,
  subset = NULL,
  na.action = na.fail,
  offset = NULL,
  contrasts = NULL,
  pwt = NULL,
  pwt_default_low = 0.01,
  pwt_default_high = 0.05,
  n_prior = NULL,
```

```
  sd = NULL,
  dispersion = NULL,
  intercept_source = c("null_model", "full_model"),
  effects_source = c("null_effects", "full_model"),
  mu = NULL,
  k = 1,
  ...
)
```

Arguments

formula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
family	a description of the error distribution and link function to be used in the model.
data	an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which glm is called.
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector.
subset	an optional vector specifying a subset of observations to be used in the fitting process. (See additional details about how this argument interacts with data-dependent bases in the 'Details' below.)
na.action	how NAs are treated. The default is first, any na.action attribute of data, second a na.action setting of options, and third na.fail if that is unset. The factory-fresh default is na.omit. Another possible value is NULL.
offset	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See model.offset.
contrasts	an optional list. See the contrasts.arg of model.matrix.default.
pwt	Weight on the prior relative to the likelihood function at the maximum likelihood estimate. If supplied, this value is used directly (scalar or one value per coefficient). If n_prior is provided and pwt is still a **scalar** and sd was **not** supplied, pwt is set to n_prior / (n_prior + n_effective). If length(pwt) > 1 (including from sd) or sd was supplied, n_prior does **not** overwrite pwt; it is used only as a scalar for Gamma / S_marg steps. If sd is provided, pwt is computed from the prior standard deviations. If none of these are supplied, pwt defaults to pwt_default_low for models with fewer than 14 coefficients, and pwt_default_high otherwise.
pwt_default_low	Default prior weight used when pwt is not supplied and the model dimension is below 14. Defaults to 0.01.

pwt_default_high
: Default prior weight used when pwt is not supplied and the model dimension is 14 or greater. Defaults to 0.05.

n_prior
: Optional scalar effective prior sample size (on the n_effective scale). If provided with scalar pwt and without sd, pwt is recomputed from n_prior. With vector pwt or with sd, pwt is left unchanged and n_prior is used for the Gamma prior on precision and related Gaussian calibration only. If missing and pwt is scalar, n_prior = (pwt/(1-pwt))*n_effective.

sd
: Optional vector argument with the prior standard deviations for the coefficients

dispersion
: Optional scalar dispersion override (default NULL). For now, this is documented as an optional argument used to scale the Sigma (variance-covariance) matrix; see Details for additional context.

intercept_source
: Specifies the method through which the prior mean for the intercept term is set. Options are based on the null intercept only model (null_model) or full_models. The default is the null model which is safer if variables are not centered.

effects_source
: Specifies the method through which the prior means for the effects terms are set. Options are null_effects (prior means set to zero) or full_model (effect means set to match maximum likelihood estimates).

mu
: Optional vector argument with the prior means for the coefficients

k
: Scalar (default 1), non-negative ($k \geq 0$), with $k+p \geq 2$ where p is the number of coefficients (columns of the model matrix). k controls the tail behavior and effective degrees of freedom of the variance prior. It does not change the posterior mean of σ^2 or the covariance of β, but larger k makes the prior and posterior for σ^2 more concentrated and less heavy-tailed. Not yet used in calibration; passed through to compute_gaussian_prior for future use.

...
: For glm: arguments to be used to form the default control argument if it is not supplied directly.

  For weights: further arguments passed to or from other methods.

Value

A named list of class "multi_PriorSetup". Each element is a Prior_Setup result for one column of the response (names from colnames(y) or Y1, Y2, ...).

See Also

Other prior: Prior_Check(), Prior_Setup()

---

multi_rNormal_reg *Multi-response Normal regression simulation*

Description

Runs rNormal_reg once per column of y. Argument x may be a single shared design matrix or a **list of matrices** with one entry per response column. The list path allows each column of y to have a different number of predictors, which is required for Block 2 of two_block_rNormal_reg.

Usage

```
multi_rNormal_reg(
  n,
  y,
  x,
  prior_list,
  offset = NULL,
  weights = 1,
  family = gaussian(),
  Gridtype = 2,
  n_envopt = NULL,
  use_parallel = TRUE,
  use_opencl = FALSE,
  verbose = FALSE,
  progbar = TRUE
)
```

Arguments

n	Number of draws per column of y.
y	Numeric matrix with one column per random-effect component.
x	Shared design matrix, or list of per-column design matrices.
prior_list	List of per-column prior lists (mu, Sigma or P, optional dispersion).
offset	Optional numeric vector of length m specifying known components of the linear predictor.
weights	Optional numeric vector of prior weights.
family	A description of the error distribution and link function (see family).
Gridtype	Optional integer specifying the method used to construct the envelope function.
n_envopt	Effective sample size passed to EnvelopeOpt for grid construction. Defaults to match n. Larger values encourage tighter envelopes.
use_parallel	Logical. Whether to use parallel processing.
use_opencl	Logical. Whether to use OpenCL acceleration.
verbose	Logical. Whether to print progress messages.
progbar	Logical. Whether to display a progress base during simulation.

Value

For shared x, an object of class "mrglmb". For list x, a plain named list of rNormal_reg results.

See Also

rNormal_reg, two_block_rNormal_reg

glmbayes Simulation Functions simfuncs, two_block_l_for_tv(), two_block_mode_weights(), two_block_rNormal_reg(), two_block_rNormal_reg_v2(), two_block_rate(), two_block_rate_v2(), two_block_tv_bound()

Normal_ct *The Central Normal Distribution*

Description

Distribution function and random generation for the center (between a lower and an upper bound) of the normal distribution with mean equal to mu and standard deviation equal to sigma.

Usage

```
pnorm_ct(a = -Inf, b = Inf, mu = 0, sigma = 1, log.p = TRUE, Diff = FALSE)

rnorm_ct(n, lgrt, lglt, mu = 0, sigma = 1)
```

Arguments

a	Lower bound for the interval. Numeric vector; must be finite when used in rnorm_ct.
b	Upper bound for the interval. Numeric vector; must be finite when used in rnorm_ct.
mu	mean parameter of the underlying normal distribution.
sigma	standard deviation of the underlying normal distribution.
log.p	Logical argument. If TRUE, the log probability is provided
Diff	Logical argument. If TRUE the second parameter is the difference between the lower and upper bound
n	number of draws to generate. If length(n) > 1, the length is taken to be the number required
lgrt	log of the distribution function between the lower bound and infinity. Numeric vectors of length n; used internally to avoid cancellation when b - a is small.
lglt	log of the distribution function between negative infinity and the upper bound. Numeric vectors of length n; used internally to avoid cancellation when b - a is small.

Details

The distribution function pnorm_ct finds the probability of the center of a normal density (the probability of the area between a lower bound a and an upper bound b) while the random number generator rnorm_ct samples from a restricted normal density where lgrt is the log of the distribution between the lower bound and infinity and lglt is the log of the distribution function between negative infinity and the upper bound. The sum of the exponentiated values for the two (exp(lgrt)+exp(lglt)) must sum to more than 1.

These functions are mainly used to handle cases where the differences between the upper and lower bounds b-a are small. In such cases, using pnorm(b)-pnorm(a) may result in 0 being returned even when the difference is supposed to be positive. They are used in envelope-based accept-reject sampling for Bayesian GLMs (Nygren and Nygren 2006).

Value

For pnorm_ct, vector of length equal to length of a and for rnorm_ct, a vector with length determined by n containing draws from the center of the normal distribution.

References

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

See Also

Gamma_ct, EnvelopeBuild

Examples

```
pnorm_ct(0.2,0.4)
exp(pnorm_ct(0.2,0.4))
pnorm_ct(0.2,0.4,log.p=FALSE)
log(pnorm_ct(0.2,0.4,log.p=FALSE))
## Example where difference between two pnorm calls fail
## but call to pnorm_ct works
pnorm(0.5)-pnorm(0.4999999999999999)
pnorm_ct(0.4999999999999999,0.5,log.p=FALSE)
```

---

normalize_block *Normalize a row-block partition for BY-style fits*

Description

Normalize a row-block partition for BY-style fits

Usage

```
normalize_block(block, l2)
```

Arguments

block	Block partition: factor or integer vector of length l2, l2_blocks counts summing to l2, or a list of disjoint row-index vectors covering 1:l2.
l2	Number of observations (rows) after model.frame.

Value

List with k, ids, l2_blocks, starts, and rows (per-block row indices).

Examples

```
## Block partition helpers (factor, l2_blocks counts, or row-index list)

l2 <- 10L

## One level per row (length l2)
normalize_block(factor(rep(c("A", "B"), each = 5L)), l2)

## Contiguous counts summing to l2
normalize_block(c(4L, 6L), l2)

## Explicit row-index list
normalize_block(list(A = 1:4, B = 5:10), l2)
```

---

pfamily *Prior Family Objects for Bayesian Models*

Description

Prior family objects provide a convenient way to specify the details of the priors used by matrix-input samplers such as rglmb and rlmb. See the documentation for rglmb and rlmb for the details of how such model fitting takes place.

Under a Beta(shape1, shape2) prior on the binomial probability θ and a Binomial(n_i, θ) likelihood with identity link ($\theta = \beta$ directly), the posterior is:

$$\theta \mid y \sim \text{Beta}(\text{shape1} + \sum n_i y_i,\ \text{shape2} + \sum n_i(1 - y_i)).$$

Usage

```
pfamily(object, ...)

dNormal(mu, Sigma, dispersion = NULL)

dGamma(
  shape,
  rate,
  beta,
  Inv_Dispersion = TRUE,
  lik_shape = 1,
  max_disp_perc = 0.99,
  disp_lower = NULL,
  disp_upper = NULL
)

dBeta(shape1, shape2, beta)

## S3 method for class 'pfamily'
```

```
print(x, ...)

dNormal_Gamma(mu, Sigma_0, shape, rate)

dIndependent_Normal_Gamma(
  mu,
  Sigma,
  shape,
  rate,
  max_disp_perc = 0.99,
  disp_lower = NULL,
  disp_upper = NULL
)
```

Arguments

object	the function pfamily accesses the pfamily objects which are stored within objects created by modelling functions (e.g., rglmb).
mu	a prior mean vector for the the modeling coefficients used in several pfamilies
Sigma	a prior variance-covariance matrix for dNormal() and dIndependent_Normal_Gamma().
dispersion	the dispersion to be assumed when it is not given a prior. Should be provided when the Normal prior is for the gaussian(), Gamma(), quasibinomial, or quasipoisson families. The binomial() and poisson() families do not have dispersion coefficients. Omitted or NULL uses the internal default 1 and sets ddef in prior_list (see Details).
shape	The prior shape parameter for the gamma piece (inverse dispersion / precision). When taking defaults from Prior_Setup, use ps$shape with dNormal_Gamma() and dGamma(), and ps$shape_ING with dIndependent_Normal_Gamma() on the Gaussian calibrated path (see Details).
rate	The prior rate parameter paired with shape. With Gaussian Prior_Setup, dNormal_Gamma() and dIndependent_Normal_Gamma() use ps$rate; for dGamma() with fixed beta, prefer ps$rate_gamma when that field is non-NULL (see Details).
beta	Initial coefficient matrix (1×1); typically set to the prior mean shape1/(shape1+shape2).
Inv_Dispersion	Logical (default TRUE). Controls which of the two Gamma prior roles dGamma() plays: • TRUE (default) — prior on the *inverse dispersion* (precision / shape parameter $k = 1/\phi$). This is the classical path used for dispersion estimation in Gaussian and Gamma(log) regression (simfun = rGamma_reg). • FALSE — conjugate prior on the Gamma or Poisson *rate* β directly (intercept-only, identity link). The posterior is a closed-form Gamma draw (simfun = rGamma_Conjugate_reg).
lik_shape	Known shape parameter $k > 0$ of the Gamma likelihood. Only used when Inv_Dispersion = FALSE and family = Gamma(link = "identity"). The intercept coefficient is then the Gamma *rate* β, and the conjugate posterior is $\beta \mid y \sim \text{Gamma}(\alpha_0 + nk,\ \beta_0 + \sum y_i)$. Defaults to 1 (exponential distribution). Ignored for Poisson families and whenever Inv_Dispersion = TRUE.

max_disp_perc	Specifies the percentile used to truncate the posterior dispersion distribution when constructing the envelope for accept-reject sampling. This determines the lower and upper bounds for the dispersion (σ^2) used in the simulation. A value of 0.99 corresponds to using the central 98 percent of the posterior dispersion mass (i.e., excluding the outer 1 percent in each tail). Smaller values yield tighter bounds and may improve acceptance rates, while larger values allow broader dispersion support but may increase envelope complexity.
disp_lower	lower bound truncation for dispersion
disp_upper	upper bound truncation for dispersion
shape1	First shape parameter $\alpha > 0$ of the Beta prior (prior successes + 1).
shape2	Second shape parameter $\beta > 0$ of the Beta prior (prior failures + 1).
x	an object, a pfamily function that is to be printed
Sigma_0	prior variance-covariance on the precision-weighted coefficient scale for dNormal_Gamma() only (Gaussian). Stored in prior_list$Sigma for compatibility with downstream samplers.
...	additional argument(s) for methods.

Details

pfamily is a generic with methods for fitted objects such as rglmb and rlmb. The dNormal() prior is supported for all response families. The gaussian() family additionally supports dNormal_Gamma(), dIndependent_Normal_Gamma(), and dGamma() (precision prior). Intercept-only models with an identity link support two closed-form conjugate priors: dBeta() for binomial(link = "identity") and dGamma(Inv_Dispersion = FALSE) for poisson(link = "identity") and Gamma(link = "identity").

A pfamily object represents a structured prior specification for use in Bayesian generalized linear modeling. Each constructor function (e.g., dNormal(), dGamma(), dNormal_Gamma(), dBeta()) returns an object of class "pfamily" containing the prior parameters, supported likelihood families, compatible link functions, and a simulation function for posterior sampling.

These priors are designed to integrate seamlessly with modeling functions such as rglmb() and rlmb() in the **glmbayes** package, which consume the pfamily object to define the prior distribution over model parameters. The pfamily() generic retrieves the embedded prior from a fitted model object, while print.pfamily() displays its structure.

prior_list **and** simfun**.** The named list prior_list holds the hyperparameters for the chosen prior family. When a model function draws from the posterior, it passes prior_list into the element simfun (e.g., rNormal_reg, rGamma_reg) so the low-level sampler receives one consistent list structure regardless of which constructor built the pfamily.

Prior_Setup **and default hyperparameters.** Prior_Setup() fits an auxiliary GLM and returns default mu, Sigma / Sigma_0, dispersion, Gamma shape and rate, and related fields aligned with the data and prior-weight (pwt) choices. Those values can be supplied as arguments to the pfamily constructors when you want package-default priors on the same scale as the model matrix. Recommended use of shape and rate is not identical across constructors: for dIndependent_Normal_Gamma(), pass shape = ps$shape_ING from Prior_Setup (not the scalar ps$shape used by dNormal_Gamma()). For dGamma() with fixed coefficients (beta), pass rate = ps$rate_gamma when that field is present (otherwise ps$rate); see Prior_Setup and compute_gaussian_prior.

Prior Families:

- dNormal(): Specifies a multivariate normal prior over regression coefficients. It is conjugate for Gaussian likelihoods with an identity link function, and serves as the primary implemented prior for all other supported likelihood families in the current framework. This structure facilitates efficient posterior sampling and analytical tractability. The returned prior_list includes ddef: TRUE when dispersion was omitted or NULL (so the default 1 was used), FALSE when dispersion was supplied explicitly (including 1).
  For models with log-concave likelihood functions-such as Poisson, Binomial, and Gamma families- posterior sampling under a dNormal prior is performed using a (Nygren and Nygren 2006) likelihood subgradient approach. This method constructs tight enveloping functions around the posterior using subgradients of the log-likelihood, enabling efficient accept-reject sampling even in high dimensions.
  When the posterior distribution is approximately normal (typically the case for large sample sizes), the area under the enveloping function is bounded above by a constant factor-approximately $2/\sqrt{\pi} \approx 1.128$ in the univariate case, and $(2/\sqrt{\pi})^k$ in k-dimensional models. These bounds ensure that the rejection rate remains manageable and that the sampler remains computationally efficient.
  The concept of conjugate priors was first formalized by (Raiffa and Schlaifer 1961), and further developed for regression models using g-prior structures by (Zellner 1986).
- dGamma(): A Gamma prior with two distinct roles controlled by Inv_Dispersion:
  - Inv_Dispersion = TRUE (default): prior on the inverse dispersion (precision $1/\phi$ or shape k). Used for dispersion estimation in Gaussian and Gamma(log) models, typically in a Gibbs step with beta held fixed (Gelman et al. 2013; Dobson 1990; McCullagh and Nelder 1989). With Gaussian Prior_Setup output, prefer rate_gamma for rate (see Details above).
  - Inv_Dispersion = FALSE: conjugate Gamma prior on the rate parameter β directly. Supports intercept-only models with an identity link: Poisson (Gamma–Poisson conjugacy) and Gamma (Gamma–Gamma conjugacy). Posterior draws are closed-form IID samples via rGamma_Conjugate_reg. The lik_shape argument specifies the known Gamma likelihood shape (default 1, i.e.\ exponential). Prior_Setup returns calibrated conj_poisson hyperparameters for this path.
- dBeta(): A Beta prior on the binomial probability θ for intercept-only binomial(link = "identity") models. The posterior is a closed-form Beta draw (Beta–Binomial conjugacy) produced by rBeta_reg. Arguments shape1 and shape2 are the prior pseudo-success and pseudo-failure counts. Prior_Setup returns calibrated conj_beta hyperparameters for this path.
- dNormal_Gamma(): Combines a multivariate normal prior on coefficients with a gamma prior on precision, forming a conjugate structure for Gaussian models with unknown variance. The second argument is Sigma_0 (precision-weighted scale); it is aliased internally to Sigma in prior_list. This formulation parallels classical Normal-Gamma models and is compatible with hierarchical extensions (Gelman et al. 2013; Raiffa and Schlaifer 1961).
- dIndependent_Normal_Gamma(): Similar to dNormal_Gamma(), but assumes independence between the coefficient and precision priors. This structure is useful for models where prior independence is desired or analytically convenient. With Prior_Setup on a Gaussian model, pass shape_ING as the shape argument (see Details above).

Each pfamily object includes:

- pfamily, prior_list, okfamilies, plinks, and simfun (see Value).

mu / Sigma: the surrogate Normal mean is shape1/(shape1+shape2) and the surrogate variance is the Beta variance shape1*shape2/((shape1+shape2)^2*(shape1+shape2+1)).

Value

An object of class "pfamily" (with a concise print method). A list with elements:

pfamily	Character string: the constructor name ("dNormal", "dGamma", "dNormal_Gamma", "dIndependent_Normal_Gamma", or "dBeta").
prior_list	Named list of prior hyperparameters. It is passed into simfun when sampling so the relevant low-level routine receives the prior in a fixed list form. Contents depend on the constructor: dNormal: mu, Sigma, dispersion, and logical ddef (TRUE if dispersion was omitted or NULL, so the default 1 was used; FALSE if set explicitly). dGamma: shape, rate, beta, Inv_Dispersion, max_disp_perc, disp_lower, disp_upper. When Inv_Dispersion = FALSE, also includes surrogate mu and Sigma (computed from the Gamma prior moments) and lik_shape. dNormal_Gamma: mu, Sigma (the Sigma_0 precision-weighted input), shape, rate. dIndependent_Normal_Gamma: mu, Sigma (coefficient-scale covariance), shape, rate, max_disp_perc, disp_lower, disp_upper. dBeta: shape1, shape2, beta, and surrogate mu and Sigma computed from the Beta prior moments (mu = shape1/(shape1+shape2), Sigma = shape1*shape2/((shape1+shape2
okfamilies	Character vector of implemented family names for which this pfamily may be used.
plinks	Function of one family argument returning allowed link names for that family.
simfun	Function used to generate posterior draws (e.g., rNormal_reg, rGamma_reg, rGamma_Conjugate_reg, rNormalGamma_reg, rindepNormalGamma_reg); for standard use these produce i.i.d.\ posterior samples for the implemented settings.

Author(s)

The design of the pfamily set of functions was developed by Kjell Nygren and was inspired by the family used by rglmb to specify the likelihood function. That design in turn was inspired by S functions of the same names from the statistical modeling literature.

References

Dobson A~J (1990). *An Introduction to Generalized Linear Models*. Chapman and Hall, London.

Gelman A, Carlin JB, Stern HS, Dunson DB, Vehtari A, Rubin DB (2013). *Bayesian Data Analysis*, 3rd edition. CRC Press.

McCullagh P, Nelder J~A (1989). *Generalized Linear Models*. Chapman and Hall, London.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

Raiffa H, Schlaifer R (1961). *Applied Statistical Decision Theory*. Clinton Press, Inc., Boston.

Zellner A (1986). "On Assessing Prior Distributions and Bayesian Regression Analysis with g-Prior Distributions." In Goel P~K, Zellner A (eds.), *Bayesian Inference and Decision Techniques: Essays in Honor of Bruno de Finetti*, volume 6 of *Studies in Bayesian Econometrics and Statistics*, 233–243. Elsevier.

See Also

`rglmb`, `rlmb` for modeling functions that consume `pfamily` objects.

`rNormal_reg`, `rNormalGamma_reg`, `rGamma_reg`, `rGamma_Conjugate_reg`, `rindepNormalGamma_reg` for lower-level sampling functions used by `pfamily` constructors.

`Prior_Setup`, `Prior_Check` for initializing and validating prior specifications.

`EnvelopeBuild` for envelope construction methods used in likelihood subgradient sampling (Nygren and Nygren 2006).

See also (Hastie and Pregibon 1992) for the original S modeling framework that inspired the design of `pfamily`.

Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts    <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome   <- gl(3, 1, 9)
treatment <- gl(3, 3)
print(d.AD <- data.frame(treatment, outcome, counts))

## Set up Prior for Poisson Model
ps <- Prior_Setup(counts ~ outcome + treatment, family = poisson())
ps

## Normal prior for rglmb
rglmb.D93 <- rglmb(
  n = 1000,
  y = ps$y,
  x = as.matrix(ps$x),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma),
  family = poisson(),
  weights = rep(1, nrow(ps$x))
)

pfamily(rglmb.D93)

## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group  <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)
```

```
## Set up prior for gaussian model
ps2 <- Prior_Setup(weight ~ group, family = gaussian())
ps2

y <- ps2$y
x <- as.matrix(ps2$x)
wt <- rep(1, length(y))

## Conjugate Normal Prior (fixed dispersion)
rlmb.D9 <- rlmb(
  n = 1000,
  y = y,
  x = x,
  pfamily = dNormal(mu = ps2$mu, ps2$Sigma, dispersion = ps2$dispersion),
  weights = wt
)
pfamily(rlmb.D9)

## Conjugate Normal_Gamma Prior
rlmb.D9_v2 <- rlmb(
  n = 1000,
  y = y,
  x = x,
  pfamily = dNormal_Gamma(
    ps2$mu,
    Sigma_0 = ps2$Sigma_0,
    shape = ps2$shape,
    rate = ps2$rate
  ),
  weights = wt
)
pfamily(rlmb.D9_v2)

## Independent_Normal_Gamma_Prior
rlmb.D9_v3 <- rlmb(
  n = 1000,
  y = y,
  x = x,
  pfamily = dIndependent_Normal_Gamma(
    ps2$mu,
    ps2$Sigma,
    shape = ps2$shape_ING,
    rate = ps2$rate
  ),
  weights = wt
)
pfamily(rlmb.D9_v3)
```

`pfamily_list` *Build a named list of pfamily objects*

Description

Generic for constructing a named list of pfamily prior objects from a prior-specification object. Packages that define prior setup containers (e.g. Prior_Setup_lmebayes() in **lmebayes**) provide methods that map each component of the container to a pfamily constructor such as dNormal or dIndependent_Normal_Gamma.

Usage

```
pfamily_list(object, ...)
```

Arguments

object	A prior-specification object.
...	Additional arguments passed to methods (e.g. ptypes).

Value

A named list whose elements are objects of class "pfamily".

See Also

pfamily, dNormal, dIndependent_Normal_Gamma

---

print.two_block_mode_weights

Print method for two_block_mode_weights objects

Description

Print method for two_block_mode_weights objects

Usage

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

Arguments

x	Object of class "two_block_mode_weights".
...	Ignored.

Value

x invisibly.

print.two_block_rate *Print method for two_block_rate objects*

Description

Print method for two_block_rate objects

Usage

```
## S3 method for class 'two_block_rate'
print(x, tols = c(0.01, 0.001, 1e-06), ...)
```

Arguments

x	Object of class "two_block_rate".
tols	Numeric vector of TV tolerances for the implied m_convergence table.
...	Ignored.

Value

x invisibly.

Prior_Check *Checks for Prior-data conflicts*

Description

Checks if the credible intervals for the prior overlap with the implied confidence intervals from the classical model (obtained via glm). The approach relates to prior-data conflict checks (Evans and Moshonov 2006).

Usage

```
Prior_Check(
  formula,
  family,
  pfamily,
  level = 0.95,
  data = NULL,
  weights,
  subset,
  na.action,
  start = NULL,
  etastart,
  mustart,
```

```
  offset,
  control = list(...),
  model = TRUE,
  method = "glm.fit",
  x = FALSE,
  y = TRUE,
  contrasts = NULL,
  ...
)
```

Arguments

formula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
family	a description of the error distribution and link function to be used in the model. For glm this can be a character string naming a family function, a family function or the result of a call to a family function. For glm.fit only the third option is supported. (See family for details of family functions.)
pfamily	a pfamily object specifying the prior distribution to check against the data.
level	the confidence level at which the Prior-data conflict should be checked.
data	an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which glm is called.
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector.
subset	an optional vector specifying a subset of observations to be used in the fitting process. (See additional details about how this argument interacts with data-dependent bases in the 'Details' below.)
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The 'factory-fresh' default is na.omit. Another possible value is NULL, no action. Value na.exclude can be useful.
start	starting values for the parameters in the linear predictor.
etastart	starting values for the linear predictor.
mustart	starting values for the vector of means.
offset	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See model.offset.
control	a list of parameters for controlling the fitting process. For glm.fit this is passed to glm.control.

model	a logical value indicating whether *model frame* should be included as a component of the returned value.
method	the method to be used in fitting the model. The default method "glm.fit" uses iteratively reweighted least squares (IWLS): the alternative "model.frame" returns the model frame and does no fitting. User-supplied fitting functions can be supplied either as a function or a character string naming a function, with a function which takes the same arguments as glm.fit. If specified as a character string it is looked up from within the **stats** namespace.
x, y	For glm: logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value. For glm.fit: x is a design matrix of dimension n * p, and y is a vector of observations of length n.
contrasts	an optional list. See the contrasts.arg of model.matrix.default.
...	For glm: arguments to be used to form the default control argument if it is not supplied directly. For weights: further arguments passed to or from other methods.

Value

A vector where each item provided the ratio of the absolue value for the difference between the prior and maximum likelihood estimate divided by the length of the sum of half of the two intervals (where normality is assumed)

References

Evans M, Moshonov H (2006). "Checking for prior-data conflict." *Bayesian Analysis*, **1**(4), 893–914. doi:10.1214/06BA129.

See Also

Prior_Setup, rglmb; see (Nygren 2025) for prior tailoring; (Nygren 2025) for full derivations.

Other prior: Prior_Setup(), multi_prior_setup()

Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
print(d.AD <- data.frame(treatment, outcome, counts))
## Step 1: Set up Prior
ps=Prior_Setup(counts ~ outcome + treatment)
mu=ps$mu
V=ps$Sigma
# Step2A: Check the Prior
Prior_Check(counts ~ outcome + treatment,family = poisson(),
```

```
            pfamily=dNormal(mu=mu,Sigma=V))
# Step2B: Update and Re-Check the Prior
mu[1,1]=log(mean(counts))
Prior_Check(counts ~ outcome + treatment,family = poisson(),
            pfamily=dNormal(mu=mu,Sigma=V))
```

`Prior_Setup` *Setup Prior Objects*

Description

Helper function to facilitate the Setup of Prior Distributions for glm models.

Usage

```
Prior_Setup(
  formula,
  family = gaussian(),
  data = NULL,
  weights = NULL,
  subset = NULL,
  na.action = na.fail,
  offset = NULL,
  contrasts = NULL,
  pwt = NULL,
  pwt_default_low = 0.01,
  pwt_default_high = 0.05,
  n_prior = NULL,
  sd = NULL,
  dispersion = NULL,
  intercept_source = c("null_model", "full_model"),
  effects_source = c("null_effects", "full_model"),
  mu = NULL,
  k = 1,
  ...
)

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

Arguments

`formula`	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`family`	a description of the error distribution and link function to be used in the model.

`data` an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `glm` is called.

`weights` an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.

`subset` an optional vector specifying a subset of observations to be used in the fitting process. (See additional details about how this argument interacts with data-dependent bases in the 'Details' below.)

`na.action` how `NA`s are treated. The default is first, any `na.action` attribute of data, second a `na.action` setting of options, and third `na.fail` if that is unset. The `factory-fresh` default is `na.omit`. Another possible value is `NULL`.

`offset` this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be `NULL` or a numeric vector of length equal to the number of cases. One or more `offset` terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See `model.offset`.

`contrasts` an optional list. See the `contrasts.arg` of `model.matrix.default`.

`pwt` Weight on the prior relative to the likelihood function at the maximum likelihood estimate. If supplied, this value is used directly (scalar or one value per coefficient). If `n_prior` is provided and `pwt` is still a **scalar** and `sd` was **not** supplied, `pwt` is set to `n_prior / (n_prior + n_effective)`. If `length(pwt) > 1` (including from `sd`) or `sd` was supplied, `n_prior` does **not** overwrite `pwt`; it is used only as a scalar for Gamma / S_marg steps. If `sd` is provided, `pwt` is computed from the prior standard deviations. If none of these are supplied, `pwt` defaults to `pwt_default_low` for models with fewer than 14 coefficients, and `pwt_default_high` otherwise.

`pwt_default_low`
Default prior weight used when `pwt` is not supplied and the model dimension is below 14. Defaults to 0.01.

`pwt_default_high`
Default prior weight used when `pwt` is not supplied and the model dimension is 14 or greater. Defaults to 0.05.

`n_prior` Optional scalar effective prior sample size (on the `n_effective` scale). If provided with scalar `pwt` and without `sd`, `pwt` is recomputed from `n_prior`. With vector `pwt` or with `sd`, `pwt` is left unchanged and `n_prior` is used for the Gamma prior on precision and related Gaussian calibration only. If missing and `pwt` is scalar, `n_prior = (pwt/(1-pwt))*n_effective`.

`sd` Optional vector argument with the prior standard deviations for the coefficients

`dispersion` Optional scalar dispersion override (default `NULL`). For now, this is documented as an optional argument used to scale the `Sigma` (variance-covariance) matrix; see Details for additional context.

`intercept_source`
Specifies the method through which the prior mean for the intercept term is set. Options are based on the null intercept only model (null_model) or full_models. The default is the null model which is safer if variables are not centered.

effects_source	Specifies the method through which the prior means for the effects terms are set. Options are null_effects (prior means set to zero) or full_model (effect means set to match maximum likelihood estimates).
mu	Optional vector argument with the prior means for the coefficients
k	Scalar (default 1), non-negative ($k \geq 0$), with $k+p \geq 2$ where p is the number of coefficients (columns of the model matrix). k controls the tail behavior and effective degrees of freedom of the variance prior. It does not change the posterior mean of σ^2 or the covariance of β, but larger k makes the prior and posterior for σ^2 more concentrated and less heavy-tailed. Not yet used in calibration; passed through to compute_gaussian_prior for future use.
...	For glm: arguments to be used to form the default control argument if it is not supplied directly. For weights: further arguments passed to or from other methods.
x	An object of class "PriorSetup"

Details

Inputs to the function

The inputs to Prior_Setup() fall into three conceptual categories:

1. Model specification

- formula: structure of the GLM (response and predictors).
- family: error distribution and link.
- data, weights, subset, na.action, offset, contrasts, control, ...: as in glm.

2. Prior variance–covariance specification

- pwt: prior weight relative to the likelihood. If scalar, used to construct a Zellner-type g-prior. If vector, applied elementwise.
- n_prior: optional scalar effective prior sample size. Replaces scalar pwt only when pwt is scalar and sd is not used; otherwise supplies precision-prior / calibration only.
- sd: optional vector of prior standard deviations. If provided, used to compute pwt from the diagonal of vcov(glm_full).
- pwt_default_low, pwt_default_high: defaults for pwt when not supplied.

3. Prior mean specification

- intercept_source: method for setting the prior mean of the intercept ("null_model" or "full_model").
- effects_source: method for setting the prior mean of the effects ("null_effects" or "full_model").
- mu: optional user-specified prior mean vector; overrides other centering logic if provided.

Prior covariance and Zellner scaling

Let $V_0 = \mathrm{vcov}(\hat{\beta})$ be the covariance matrix of the full-model GLM coefficients. For non-Gaussian families, the prior covariance is:

$$\Sigma = \begin{cases} \dfrac{1-\mathrm{pwt}}{\mathrm{pwt}} V_0, & \text{scalar pwt,} \\ V_0 \circ \left[\sqrt{\dfrac{1-\mathrm{pwt}_i}{\mathrm{pwt}_i}} \sqrt{\dfrac{1-\mathrm{pwt}_j}{\mathrm{pwt}_j}} \right], & \text{vector pwt,} \end{cases}$$

where $\circ$ denotes elementwise multiplication.

Intercept-only Poisson(link = "identity") conjugate prior on the rate

When the design is a single column (intercept only), `family = poisson()`, `link = "identity"`, scalar `pwt`, and offsets are zero, the effective conjugate prior observation count n_{prior} already satisfies $n_{\mathrm{prior}}/(n_{\mathrm{prior}} + n_{\mathrm{effective}}) = \mathrm{pwt}$ (otherwise `conj_poisson` remains `NULL` with a warning). Writing the weighted mean $\bar{y}_w = \sum_i w_i y_i / \sum_i w_i$, the output list component `conj_poisson` stores `shape` $= n_{\mathrm{prior}} \bar{y}_w$ and `rate` $= n_{\mathrm{prior}}$, so the prior mean for the rate matches $\bar{y}_w$. Omitting optional `mu` resets the surrogate Normal summaries `mu` and the sole diagonal element of `Sigma` to these Gamma moments (`Sigma_{11}` = $\bar{y}_w / n_{\mathrm{prior}}$).

For Gaussian families, `Prior_Setup()` also constructs the dispersion-free covariance

$$\Sigma_0 = \Sigma / \texttt{dispersion},$$

which under scalar `pwt` and the default calibration reduces to

$$\Sigma_0 = \frac{1-\mathrm{pwt}}{\mathrm{pwt}} (X^\top W X)^{-1}.$$

Gaussian Normal–Gamma calibration and S_{marg}

For `family = gaussian()`, the function performs the Normal–Gamma calibration described in (Nygren 2025). Let:

- $p = \texttt{ncol}(x)$,
- $n_{\mathrm{effective}} = \sum_i w_i$,
- $\hat{\beta}$ the weighted least-squares estimator,
- Σ_0 the dispersion-free prior covariance.

The marginal quadratic term is

$$S_{\mathrm{marg}} = \mathrm{RSS}_w + (\hat{\beta} - \mu)^\top \left(\Sigma_0 + (X^\top W X)^{-1} \right)^{-1} (\hat{\beta} - \mu),$$

where RSS_w is the weighted residual sum of squares at $\hat{\beta}$. Under the default scalar-`pwt` Zellner mapping $\Sigma_0 = \frac{1-\mathrm{pwt}}{\mathrm{pwt}} (X^\top W X)^{-1}$, this simplifies to

$$S_{\mathrm{marg}} = \mathrm{RSS}_w + \mathrm{pwt}\, (\hat{\beta} - \mu)^\top (X^\top W X)(\hat{\beta} - \mu),$$

which makes the limiting behavior as $\mathrm{pwt} \to 0$ transparent.

The calibrated dispersion is

$$\texttt{dispersion} = \frac{S_{\mathrm{marg}}}{n_{\mathrm{effective}} - p},$$

and the Normal–Gamma hyperparameters are

$$\text{shape} = \frac{n_{\text{prior}} + k}{2}, \qquad \text{rate} = \frac{1}{2} S_{\text{marg}} \frac{n_{\text{prior}} + k + p - 2}{n_{\text{effective}} - p}.$$

The independent Normal–Gamma shape is

$$\text{shape}_{ING} = \text{shape} + \frac{p}{2}.$$

Posterior summaries for the conjugate Normal–Gamma prior

Under the conjugate Normal–Gamma prior (used by `dNormal_Gamma()`), the posterior has:

- Posterior mean
$$E[\beta \mid y] = (1 - \text{pwt})\, \hat{\beta} + \text{pwt}\, \mu.$$
- Posterior expectation of σ^2
$$E[\sigma^2 \mid y] = \frac{S_{\text{marg}}}{n_{\text{effective}} - p}.$$
- Posterior covariance
$$\text{Cov}(\beta \mid y) = E[\sigma^2 \mid y] \left(\Sigma_0^{-1} + X^\top W X\right)^{-1}.$$

Weak-prior limits (Theorems 2 and 3)

As $n_{\text{prior}} \to 0^+$ (equivalently $\text{pwt} \to 0$), $S_{\text{marg}} \to \text{RSS}_w$, and the conjugate Normal–Gamma posterior converges to the classical weighted least-squares limit:

$$E[\beta \mid y] \to \hat{\beta}, \qquad E[\sigma^2 \mid y] \to \frac{\text{RSS}_w}{n_{\text{effective}} - p}, \qquad \text{Cov}(\beta \mid y) \to \frac{\text{RSS}_w}{n_{\text{effective}} - p} (X^\top W X)^{-1}.$$

For the independent Normal–Gamma prior used by `dIndependent_Normal_Gamma()`, neither the posterior mean nor the posterior covariance is available in closed form; the posterior must be obtained by numerical integration or sampling (e.g., `rindepNormalGamma_reg()`). Theorem 3 in (Nygren 2025) shows that the ING posterior has the same weak-prior limit as the conjugate Normal–Gamma posterior:

$$E[\beta \mid y] \to \hat{\beta}, \qquad \text{Cov}(\beta \mid y) \to \frac{\text{RSS}_w}{n_{\text{effective}} - p} (X^\top W X)^{-1}.$$

Value

A list of class `"PriorSetup"` with components:

`mu`	Prior mean vector (length equal to the number of coefficients).
`Sigma`	Coefficient-scale prior variance–covariance matrix.
`Sigma_0`	For `family = gaussian()` only: dispersion-independent prior covariance on the precision-weighted coefficient scale (the `Sigma_0` passed to `compute_gaussian_prior`). Under scalar `pwt`, $\Sigma_0^{-1} = \frac{\text{pwt}}{1-\text{pwt}} X^\top W X$.
`dispersion`	Calibrated dispersion (Gaussian models only), equal to $S_{\text{marg}}/(n_{\text{effective}} - p)$ under the default calibration.

`shape` Derived prior Gamma shape parameter for the Normal–Gamma prior on precision (Gaussian only), $(n_{\text{prior}} + k)/2$.

`shape_ING` For `gaussian()` only when `shape` is available: dedicated shape parameter for `dIndependent_Normal_Gamma()`, `shape` $+ p/2$.

`rate` Derived prior Gamma rate parameter (Gaussian only), using the calibrated S_{marg}.

`rate_gamma` For `gaussian()` only, when Gaussian calibration runs: prior Gamma rate for `dGamma()` / fixed-β use, based on $\text{RSS}_w(\beta_\star)$ at the Zellner blend.

`coefficients` Named numeric vector of returned coefficient values. For `gaussian()` with scalar or vector `pwt`, this is the closed-form posterior-mean blend $(1 - \text{pwt})\hat{\beta} + \text{pwt}\mu$ when inputs are valid; otherwise it falls back to the full-model GLM coefficients.

`model` The model frame used to construct the design matrix (if `model = TRUE`).

`x` The model matrix used (if `x = TRUE`).

`y` The response vector used (if `y = TRUE`).

`call` The matched call to `Prior_Setup()`.

`PriorSettings` A list containing prior configuration details, including `pwt`, `n_prior`, `n_effective`, `n_likelihood`, `intercept_source`, and `effects_source`.

`conj_poisson` `NULL` unless `family = poisson(link = "identity")` with a single-column design (intercept-only), scalar `pwt`, finite positive scalar `n_prior`, and negligible `offset`. Then a named list with conjugate `Gamma(shape, rate)` hyperparameters on the Poisson rate, with `shape` $= n_{\text{prior}}\,\bar{y}_w$, `rate` $= n_{\text{prior}}$, and `beta` centered at the weighted sample mean $\bar{y}_w$; pass to `dGamma` with `Inv_Dispersion = FALSE`. See Details.

References

Nygren K (2025). "Chapter A12: Technical Derivations for Priors Returned by Prior_ Setup." Vignette in the glmbayes R package. R vignette name: Chapter-A12.

See Also

`pfamily` for prior-family objects and the constructors `dNormal`, `dNormal_Gamma`, `dGamma`, and `dIndependent_Normal_Gamma`.

`rglmb`, `rlmb` for matrix-input fits with a `pfamily` built from `Prior_Setup()` output; `rglmb`, `rlmb` for matrix-based sampling that consumes the same prior structure; `simfuncs` for functions that take a `prior_list` assembled from those components (including `rindepNormalGamma_reg` for `dIndependent_Normal_Gamma()`). `multi_prior_setup` for a matrix/cbind response with Gaussian; use with `rlmb` `Prior_Setup` per column.

(Zellner 1986); (Raiffa and Schlaifer 1961); (Gelman et al. 2013); (McCullagh and Nelder 1989); (Nygren 2025); (Nygren 2025).

Other prior: `Prior_Check()`, `multi_prior_setup()`

Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
print(d.AD <- data.frame(treatment, outcome, counts))

## Set up Prior for Poisson Model
ps <- Prior_Setup(counts ~ outcome + treatment, family = poisson())
ps

## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)

## Set up prior for gaussian model
ps2 <- Prior_Setup(weight ~ group, family = gaussian())
ps2

## ----------------------------------------------------------------------------
## Matrix-input bridge: use Prior_Setup outputs with rglmb() and rlmb()
## ----------------------------------------------------------------------------
y <- ps2$y
x <- as.matrix(ps2$x)
wt <- rep(1, length(y))

rglmb.D9 <- rglmb(
  n = 1000,
  y = y,
  x = x,
  pfamily = dIndependent_Normal_Gamma(
    ps2$mu,
    ps2$Sigma,
    shape = ps2$shape_ING,
    rate = ps2$rate
  ),
  weights = wt,
  family = gaussian()
)

rlmb.D9 <- rlmb(
  n = 1000,
  y = y,
  x = x,
  pfamily = dIndependent_Normal_Gamma(
    ps2$mu,
    ps2$Sigma,
    shape = ps2$shape_ING,
    rate = ps2$rate
```

```
  ),
  weights = wt
)

## ---------------------------------------------------------------------------
## Prior-list templates for lower-level samplers
## ---------------------------------------------------------------------------
prior_list_rNormalGamma <- list(
  mu = ps2$mu,
  Sigma = ps2$Sigma_0,
  shape = ps2$shape,
  rate = ps2$rate
)

prior_list_rindepNormalGamma <- list(
  mu = ps2$mu,
  Sigma = ps2$Sigma,
  dispersion = ps2$dispersion,
  shape = ps2$shape_ING,
  rate = ps2$rate,
  Precision = solve(ps2$Sigma),
  max_disp_perc = 0.99
)

rate_dg <- if (!is.null(ps2$rate_gamma)) ps2$rate_gamma else ps2$rate
prior_list_rGamma <- list(
  beta = ps2$coefficients,
  shape = ps2$shape,
  rate = rate_dg
)

## Note: for a full dGamma run across rGamma_reg/rglmb/rlmb, see:
## example("summary.rGamma_reg")

## ---------------------------------------------------------------------------
## dGamma prior illustration: Prior_Setup(shape, rate_gamma or rate) + fixed beta
## ---------------------------------------------------------------------------
out.rGamma_reg <- rGamma_reg(
  n = 1000,
  y = y,
  x = x,
  prior_list = prior_list_rGamma,
  offset = rep(0, length(y)),
  weights = wt,
  family = gaussian()
)

## ---------------------------------------------------------------------------
## Poisson(link = "identity"), intercept-only: `conj_poisson` + dGamma(Inv_Dispersion=FALSE)
## ---------------------------------------------------------------------------
y_p <- c(rep(1L, 3L), rep(0L, 6L))
df_p <- data.frame(y = y_p)
ps_p <- Prior_Setup(
```

```
  y ~ 1,
  family = poisson(link = "identity"),
  data = df_p,
  pwt = 0.4
)
if (!is.null(ps_p$conj_poisson)) {
  cp <- ps_p$conj_poisson
 pf_conj <- dGamma(shape = cp$shape, rate = cp$rate, beta = cp$beta, Inv_Dispersion = FALSE)
  ## rglmb(n = 500, y = ps_p$y, x = as.matrix(ps_p$x),
 ##     pfamily = pf_conj, family = poisson(link = "identity"), weights = rep(1, length(ps_p$y)))
}
```

`residuals.rglmb` *Deviance Residuals for* `rglmb` *and* `summary.rglmb` *Objects*

Description

Returns a matrix of deviance residuals across posterior draws, using the fitted object's `family$dev.resids` function as in `residuals.glm`.

Usage

```
## S3 method for class 'rglmb'
residuals(object, ysim = NULL, ...)

## S3 method for class 'summary.rglmb'
residuals(object, ysim = NULL, ...)

## S3 method for class 'rlmb'
residuals(object, ysim = NULL, ...)
```

Arguments

`object`	an object of class `rglmb`, `rlmb`, or `summary.rglmb`.
`ysim`	optional simulated responses (matrix with one row per draw).
`...`	further arguments (currently unused).

Value

A numeric matrix of deviance residuals with one row per draw.

See Also

`rglmb`, `summary.rglmb`, `residuals.glm`.

rglmb *The Bayesian Generalized Linear Model Distribution*

Description

rglmb is used to generate iid samples for Bayesian Generalized Linear Models. The model is specified by providing a data vector, a design matrix, the family (determining the likelihood function) and the pfamily (determining the prior distribution).

Usage

```
rglmb(
  n = 1,
  y,
  x,
  family = gaussian(),
  pfamily,
  offset = NULL,
  weights = 1,
  Gridtype = 2,
  n_envopt = NULL,
  use_parallel = TRUE,
  use_opencl = FALSE,
  verbose = FALSE
)

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

Arguments

n	number of draws to generate. If length(n) > 1, the length is taken to be the number required.
y	a vector of observations of length m.
x	for rglmb a design matrix of dimension m * p and for print.rglmb the object to be printed.
family	a description of the error distribution and link function to be used in the model.
pfamily	a description of the prior distribution (see pfamily).
offset	this can be used to specify an a priori known component to be included in the linear predictor.
weights	an optional vector of prior weights to be used in the fitting process.
Gridtype	an optional argument specifying the method used to determine tangent points for the envelope.
n_envopt	Effective sample size passed to EnvelopeOpt for grid construction. Defaults to match n. Larger values encourage tighter envelopes.

use_parallel	Logical. Whether to use parallel processing during simulation.
use_opencl	Logical. Whether to use OpenCL acceleration during Envelope construction.
verbose	Logical. Whether to print progress messages.
digits	the number of significant digits to use when printing.
...	further arguments passed to or from other methods.

Details

The function rglmb is a minimalistic engine for Bayesian generalized linear model simulation. It is designed to generate independent draws from the posterior distribution of a GLM, given a design matrix, Unlike formula-based interfaces in **glmbayes**, rglmb operates directly on numeric inputs and is optimized for speed, transparency, and integration into simulation workflows.

The original R implementation of glm was written by Simon Davies (under Ross Ihaka at the University of Auckland) and has since been extensively rewritten by members of the R Core Team; its design was inspired by the S function described in (Hastie and Pregibon 1992), which in turn relies on the formula framework described in (Wilkinson and Rogers 1973).

The design of the pfamily family of functions was created by Kjell Nygren and is modeled on how glm uses family to specify the likelihood. For any implemented combination of family, link, and pfamily, rglmb generates independent draws from the posterior density-no MCMC chains are required.

A helper, Prior_Setup, assists users in choosing prior parameters. It ships with sensible defaults but also allows full customization. In particular, the default for dNormal is a reparameterization of Zellner's g-prior (Zellner 1986).

Currently supported response families are gaussian (identity link), poisson and quasipoisson (log link), gamma (log link), and binomial and quasibinomial (logit, probit, cloglog). All families support a dNormal prior; the Gaussian family also offers dNormalGamma and dIndependent_Normal_Gamma.

For the Gaussian family, draws under dNormal and dNormalGamma come from posterior distributions resulting from conjugate prior distributions (Raiffa and Schlaifer 1961). For all other priors or response families, we use an accept-reject sampler built on the likelihood-subgradient envelope method (Nygren and Nygren 2006). The Gridtype argument controls how many tangent points are used in the envelope-trading off envelope tightness against construction cost-and iters reports candidate counts before acceptance.

By default, rglmb draws n = 1 sample, uses parallel CPU simulation, and-if use_opencl = TRUE-GPU-accelerated envelope building. The function returns a list containing posterior samples, prior specifications, dispersion estimates, and the envelope used during sampling. It does not return a full model object, and does not support formula-based modeling or method dispatch. It is intended for Gibbs sampling implementations or other workflows where full model reconstruction is unnecessary.

Value

rglmb returns a object of class "rglmb". The generic accessor functions coefficients and fitted.values can be used to extract useful features of the value returned by rglmb. An object of class "rglmb" is a list containing at least the following components:

coefficients	a matrix of dimension n by length(mu) with one sample in each row

coef.mode	a vector of length(mu) with the estimated posterior mode coefficients
dispersion	Either a constant provided as part of the call, or a vector of length n with one sample in each row.
Prior	A list with the priors specified for the model in question. Items in the list may vary based on the type of prior
prior.weights	a vector of weights specified or implied by the model
y	a vector with the dependent variable
x	a matrix with the implied design matrix for the model
famfunc	Family functions used during estimation process
iters	an n by 1 matrix giving the number of candidates generated before acceptance for each sample.
Envelope	the envelope that was used during sampling

Objects of class "rglmb" are normally of class c("rglmb","glmb","glm","lm"), meaning they inherit from glmb, glm, and lm. This allows methods defined for these upstream classes to be applied to "rglmb" objects when appropriate, while supporting extensions for regularized Bayesian GLMs with structured priors.

Author(s)

The R implementation of rglmb has been written by Kjell Nygren and was built to be a Bayesian version of the glm function but with a more minimalistic interface than a formula wrapper. It also borrows some of its structure from other random generating function like rnorm and hence the r prefix.

References

Hastie T~J, Pregibon D (1992). "Generalized Linear Models." In Chambers J~M, Hastie T~J (eds.), *Statistical Models in S*, chapter 6. Wadsworth & Brooks/Cole, Belmont, CA.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

Raiffa H, Schlaifer R (1961). *Applied Statistical Decision Theory*. Clinton Press, Inc., Boston.

Wilkinson GN, Rogers CE (1973). "Symbolic Descriptions of Factorial Models for Analysis of Variance." *Applied Statistics*, **22**(3), 392–399. doi:10.2307/2346786.

Zellner A (1986). "On Assessing Prior Distributions and Bayesian Regression Analysis with g-Prior Distributions." In Goel P~K, Zellner A (eds.), *Bayesian Inference and Decision Techniques: Essays in Honor of Bruno de Finetti*, volume 6 of *Studies in Bayesian Econometrics and Statistics*, 233–243. Elsevier.

See Also

rlmb for the Gaussian specialization; lm and glm for classical modeling functions.

EnvelopeBuild, EnvelopeSize, EnvelopeEval for envelope construction and grid evaluation used in non-conjugate sampling.

family for documentation of family functions used to specify priors. pfamily for documentation of pfamily functions used to specify priors.

Prior_Setup, Prior_Check for functions used to initialize and to check priors,

Further reading: (Nygren and Nygren 2006); (Nygren 2025, 2025); OpenCL/GPU: (Nygren 2025, 2025). summary.rglmb and residuals.rglmb provide posterior summaries for matrix-input fits; additional formula-based methods are in **glmbayes**.

glmbayes Modeling Functions rlmb()

Examples

```
set.seed(333)
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
print(d.AD <- data.frame(treatment, outcome, counts))

## Classical Model
glm.D93 <- glm(counts ~ outcome + treatment, family = poisson(link = log))
summary(glm.D93)

## Poisson prior and rglmb (Prior_Setup supplies default g-prior hyperparameters)
ps <- Prior_Setup(counts ~ outcome + treatment, family = poisson(), data = d.AD)
rglmb.D93 <- rglmb(
  n = 1000,
  y = ps$y,
  x = as.matrix(ps$x),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma),
  family = poisson(),
  weights = rep(1, nrow(ps$x))
)
print(rglmb.D93)


## Menarche model with informative prior. See \code{\link{Prior_Setup}}
## for default g-priors; fitted curves for the menarche data are shown below.
## See \code{vignette("Chapter-05", package = "glmbayes")}
## and \code{vignette("Chapter-06", package = "glmbayes")}.
data(menarche, package = "MASS")

summary(menarche)
design_df <- data.frame(
  Age = menarche$Age,
  Age2 = menarche$Age - 13,
  Proportion = menarche$Menarche / menarche$Total,
  Total = menarche$Total
)
x <- model.matrix(~ Age2, data = design_df)
y <- design_df$Proportion
```

```
wt <- design_df$Total

# Extract coefficient names from design matrix
coef_names <- colnames(x)

# Set up prior mean with names
mu <- matrix(0, nrow = length(coef_names), ncol = 1)
mu[2, 1] <- (log(0.9 / 0.1) - log(0.5 / 0.5)) / 3
rownames(mu) <- coef_names

# Set up prior covariance matrix with named rows and columns
V1 <- 1 * diag(as.numeric(2.0))

# 2 standard deviations for prior estimate at age 13 between 0.1 and 0.9
## Specifies uncertainty around the point estimates
V1[1, 1] <- ((log(0.9 / 0.1) - log(0.5 / 0.5)) / 2)^2
V1[2, 2] <- (3 * mu[2, 1] / 2)^2 # Allows slope to be up to 3 times as large as point estimate
rownames(V1) <- coef_names
colnames(V1) <- coef_names

out <- rglmb(
  n = 1000, y = y, x = x, pfamily = dNormal(mu = mu, Sigma = V1), weights = wt,
  family = binomial(logit)
)
print(out)

## rglmb with dGamma prior (dispersion-only; coefficients fixed)
ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)
ps_dg <- Prior_Setup(weight ~ group, family = gaussian())
rate_dg <- if (!is.null(ps_dg$rate_gamma)) ps_dg$rate_gamma else ps_dg$rate
out_dGamma <- rglmb(
  n = 100, y = ps_dg$y, x = as.matrix(ps_dg$x),
  pfamily = dGamma(shape = ps_dg$shape, rate = rate_dg, beta = ps_dg$coefficients),
  weights = rep(1, length(ps_dg$y)), family = gaussian()
)
print(out_dGamma)
```

`rIndepNormalGammaReg_std` *The Bayesian Gaussian Regression with Independent Normal-Gamma Prior in Standard Form*

Description

`rIndepNormalGammaReg_std` generates iid samples from a Bayesian Gaussian regression model with an independent Normal-Gamma prior, in standard form. The function should only be called after standardization and envelope construction (e.g., via `EnvelopeOrchestrator`).

Usage

```
rIndepNormalGammaReg_std(
  n,
  y,
  x,
  mu,
  P,
  alpha,
  wt,
  f2,
  Envelope,
  gamma_list,
  UB_list,
  family,
  link,
  progbar = TRUE,
  verbose = FALSE
)
```

Arguments

n	Number of draws to generate. If length(n) > 1, the length is taken to be the number required.
y	A vector of observations of length m.
x	A design matrix of dimension m * p.
mu	A matrix of prior means (typically standardized to zero) of dimension p * 1.
P	A positive-definite matrix of dimension p * p specifying the prior precision component shifted into the log-likelihood.
alpha	A numeric vector of length m for the offset-adjusted mean component. See model.offset.
wt	An optional vector of prior weights. Should be NULL or a numeric vector.
f2	Function used to calculate the negative of the log-posterior (kept for signature parity with rNormalGLM_std).
Envelope	An envelope object containing PLSD, loglt, logrt, cbars, and related components from EnvelopeOrchestrator.
gamma_list	A list with shape3, rate2, disp_lower, and disp_upper from the Gamma posterior for the dispersion.
UB_list	A list with lg_prob_factor, UB2min, RSS_Min, and other bounds from EnvelopeOrchestrator.
family	Character vector specifying the family (e.g., "gaussian").
link	Character vector specifying the link (e.g., "identity").
progbar	Logical. Whether to display a progress bar during simulation.
verbose	Logical. Whether to print diagnostic messages.

Details

This function uses the envelope and dispersion bounds from EnvelopeOrchestrator to sample from the joint posterior of coefficients and dispersion via rejection sampling. It is typically called internally by rindepNormalGamma_reg(), but may be used directly for custom split workflows (e.g., after constructing the envelope separately).

Value

A list with components:

beta_out	A matrix of simulated regression coefficients in standardized space. Each row is one draw.
disp_out	A vector of dispersion draws for each sample.
iters_out	A vector of iteration counts (candidates per acceptance) for each draw.
weight_out	A vector of weights (typically all ones).

See Also

EnvelopeOrchestrator for envelope construction, rNormalGLM_std for the non-Gaussian standardized sampler, rindepNormalGamma_reg for the full simulation routine.

Examples

```
############################## Start of rIndepNormalGammaReg_std example ###################

# This example demonstrates calling rIndepNormalGammaReg_std directly for Gaussian
# regression with an independent Normal-Gamma prior. It uses Ex_EnvelopeDispersionBuild
# as a starting point (Steps A through F: EnvelopeCentering, mode optimization,
# standardization, EnvelopeBuild, EnvelopeDispersionBuild, EnvelopeSort), then
# adds sampling and back-transformation to unstandardized form (like the C++ code
# and rNormalGLM_std).

ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)

ps <- Prior_Setup(weight ~ group, gaussian())

x <- as.matrix(ps$x)
y <- as.vector(ps$y)
mu <- ps$mu
Sigma <- ps$Sigma
shape <- ps$shape
rate <- ps$rate

n_obs <- length(y)
wt <- rep(1, n_obs)
offset2 <- rep(0, n_obs)
```

```
# Reconstruct coefficient precision P (matches rindepNormalGamma_reg)
Rchol <- chol(Sigma)
Pinv <- chol2inv(Rchol)
P <- 0.5 * (Pinv + t(Pinv))

famfunc <- glmbfamfunc(gaussian())
f2 <- famfunc$f2
f3 <- famfunc$f3

Gridtype_core <- as.integer(2)

##############################################################################
# Step A: EnvelopeCentering (initial dispersion + dispersion anchoring loop)
##############################################################################
centering <- EnvelopeCentering(
  y = y,
  x = x,
  mu = as.vector(mu),
  P = P,
  offset = offset2,
  wt = wt,
  shape = shape,
  rate = rate,
  Gridtype = Gridtype_core,
  verbose = FALSE
)

dispersion2 <- centering$dispersion
RSS_Post2 <- centering$RSS_post

n_w <- sum(wt)

##############################################################################
# Step B: Coefficient posterior mode optimization (optim + f2/f3)
##############################################################################
dispstar <- dispersion2

wt2_opt <- wt / dispstar
alpha <- as.vector(x %*% as.vector(mu) + offset2)

mu2 <- rep(0, length(as.vector(mu)))   # mu2 = 0 * mu (as in C++)
parin <- rep(0, length(as.vector(mu))) # parin = 0 vector (mu - mu)

opt_out <- optim(
  par = parin,
  fn = f2,
  gr = f3,
  y = as.vector(y),
  x = as.matrix(x),
  mu = as.vector(mu2),
  P = as.matrix(P),
  alpha = as.vector(alpha),
  wt = as.vector(wt2_opt),
```

```
  method = "BFGS",
  hessian = TRUE
)

bstar <- opt_out$par
A1 <- opt_out$hessian

############################################################################
# Step C: Standardize model (glmb_Standardize_Model)
############################################################################
Standard_Mod <- glmb_Standardize_Model(
  y = as.vector(y),
  x = as.matrix(x),
  P = as.matrix(P),
  bstar = as.matrix(bstar, ncol = 1),
  A1 = as.matrix(A1)
)

bstar2 <- Standard_Mod$bstar2
A <- Standard_Mod$A
x2_std <- Standard_Mod$x2
mu2_std <- Standard_Mod$mu2
P2_std <- Standard_Mod$P2
L2Inv <- Standard_Mod$L2Inv
L3Inv <- Standard_Mod$L3Inv

############################################################################
# Step D: EnvelopeBuild (coefficient envelope at Gridtype = 3)
############################################################################
max_disp_perc <- 0.99
n_env <- as.integer(200) # used by EnvelopeBuild for diagnostics/overhead
Gridtype_env <- as.integer(3) # EnvelopeOrchestrator overrides to 3

shape2_env <- shape + n_w / 2.0
rate3_env <- rate + RSS_Post2 / 2.0
d1_star <- rate3_env / (shape2_env - 1.0)

wt2_env <- wt / d1_star

Env2 <- EnvelopeBuild(
  bStar = as.vector(bstar2),
  A = as.matrix(A),
  y = as.vector(y),
  x = as.matrix(x2_std),
  mu = as.matrix(mu2_std, ncol = 1),
  P = as.matrix(P2_std),
  alpha = as.vector(alpha),
  wt = as.vector(wt2_env),
  family = "gaussian",
  link = "identity",
  Gridtype = Gridtype_env,
  n = n_env,
  n_envopt = as.integer(1),
```

```
  sortgrid = FALSE,
  use_opencl = FALSE,
  verbose = FALSE
)

##############################################################################
# Step E: EnvelopeDispersionBuild (dispersion-aware envelope)
##############################################################################
disp_env_out <- EnvelopeDispersionBuild(
  Env = Env2,
  Shape = shape,
  Rate = rate,
  P = as.matrix(P2_std),
  y = as.vector(y),
  x = as.matrix(x2_std),
  alpha = as.vector(alpha),
  n_obs = as.integer(n_obs),
  RSS_post = RSS_Post2,
  RSS_ML = NA_real_,
  mu = as.matrix(mu2_std, ncol = 1),
  wt = as.vector(wt),
  max_disp_perc = max_disp_perc,
  disp_lower = NULL,
  disp_upper = NULL,
  verbose = FALSE,
  use_parallel = TRUE
)

##############################################################################
# Step F: EnvelopeSort (mirror EnvelopeOrchestrator: disp_grid_type = 2)
##############################################################################
Env3_raw <- disp_env_out$Env_out
UB_list_new <- disp_env_out$UB_list
gamma_list_new <- disp_env_out$gamma_list

cbars <- Env3_raw$cbars
l1 <- ncol(cbars)
l2 <- nrow(cbars)

logP_vec <- Env3_raw$logP
logP_mat <- matrix(logP_vec, nrow = length(logP_vec), ncol = 1)

Env3 <- EnvelopeSort(
  l1 = l1,
  l2 = l2,
  GIndex = Env3_raw$GridIndex,
  G3 = Env3_raw$thetabars,
  cbars = cbars,
  logU = Env3_raw$logU,
  logrt = Env3_raw$logrt,
  loglt = Env3_raw$loglt,
  logP = logP_mat,
  LLconst = Env3_raw$LLconst,
```

```
  PLSD = Env3_raw$PLSD,
  a1 = Env3_raw$a1,
  E_draws = Env3_raw$E_draws,
  lg_prob_factor = UB_list_new$lg_prob_factor,
  UB2min = UB_list_new$UB2min
)

UB_list_final <- UB_list_new
UB_list_final$lg_prob_factor <- Env3$lg_prob_factor
UB_list_final$UB2min <- Env3$UB2min

env_final <- list(
  Env = Env3,
  gamma_list = gamma_list_new,
  UB_list = UB_list_final,
  diagnostics = disp_env_out$diagnostics,
  low = gamma_list_new$disp_lower,
  upp = gamma_list_new$disp_upper
)

########################################################################################
# Step G: Sample via rIndepNormalGammaReg_std (standardized space)
########################################################################################
n <- as.integer(100)
sim <- rIndepNormalGammaReg_std(
  n = n,
  y = as.vector(y),
  x = as.matrix(x2_std),
  mu = as.matrix(mu2_std, ncol = 1),
  P = as.matrix(P2_std),
  alpha = as.vector(alpha),
  wt = as.vector(wt),
  f2 = f2,
  Envelope = env_final$Env,
  gamma_list = env_final$gamma_list,
  UB_list = env_final$UB_list,
  family = "gaussian",
  link = "identity",
  progbar = FALSE,
  verbose = FALSE
)

########################################################################################
# Step H: Back-transform to unstandardized form (mirror C++ and rNormalGLM_std)
########################################################################################
# beta_out is n x p (one draw per row); t(beta_out) is p x n
coef_unstd <- L2Inv %*% L3Inv %*% t(sim$beta_out)
for (i in seq_len(n)) {
  coef_unstd[, i] <- coef_unstd[, i] + as.vector(mu)
}
coefficients <- t(coef_unstd)  # n x p, one draw per row
colnames(coefficients) <- colnames(x)
```

```
##############################################################################
# Summary output
##############################################################################
summary(coefficients)
mean(sim$iters_out)
sim$disp_out[1:5]

##############################################################################
# End of rIndepNormalGammaReg_std example
##############################################################################
```

---

rlmb *The Bayesian Linear Model Distribution*

Description

rlmb is used to generate iid samples from Bayesian Linear Models with multivariate normal priors. The model is specified by providing a data vector, a design matrix, and a pfamily (determining the prior distribution).

Usage

```
rlmb(
  n = 1,
  y,
  x,
  pfamily,
  offset = rep(0, nobs),
  weights = NULL,
  Gridtype = 2,
  n_envopt = NULL,
  use_parallel = TRUE,
  use_opencl = FALSE,
  verbose = FALSE,
  progbar = FALSE
)

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

Arguments

n	number of draws to generate. If length(n) > 1, the length is taken to be the number required.
y	a vector of observations of length m.
x	for rlmb a design matrix of dimension m * p and for print.rlmb the object to be printed.

pfamily	a description of the prior distribution and associated constants to be used in the model. This should be a pfamily function (see pfamily for details of pfamily functions.)
offset	an optional numeric vector of known offset to be included in the linear predictor.
weights	an optional numeric vector of prior weights for the observations.
Gridtype	an optional argument specifying tangent points for envelope construction.
n_envopt	Effective sample size passed to EnvelopeOpt for grid construction.
use_parallel	Logical. Whether to use parallel processing during simulation.
use_opencl	Logical. Whether to use OpenCL acceleration during Envelope construction.
verbose	Logical. Whether to print progress messages.
progbar	Logical. Whether to display a progress base during simulation.
digits	the number of significant digits to use when printing.
...	further arguments passed to or from other methods.

Details

The function rlmb is a minimalistic Bayesian simulation engine for Gaussian linear models. It bypasses classical model fitting and formula parsing, operating directly on numeric inputs such as the design matrix, response vector, and prior specification via the pfamily argument. Internally, rlmb generates independent draws from the posterior distribution using multivariate normal simulation when conjugate priors are specified.

The modeling framework follows (Wilkinson and Rogers 1973), and the prior structure builds on the S system (Chambers 1992), Zellner's g-prior (Zellner 1986), and the conjugate prior formulation of Raiffa and Schlaifer (Raiffa and Schlaifer 1961).

Prior specification is handled via the pfamily argument, which defines the prior mean, covariance, and dispersion. The design of the pfamily family of functions was created by Kjell Nygren and is modeled on how glm uses family to specify the likelihood. A helper function, Prior_Setup, assists users in choosing prior parameters. It ships with sensible defaults but also allows full customization. Available priors include the dNormal, dNormalGamma and dIndependent_Normal_Gamma priors. The last of these allows for more flexible prior structures including independent priors on variance components.

Posterior draws are generated using standard simulation procedures for conjugate priors (Raiffa and Schlaifer 1961). For non-conjugate setups, the function uses envelope-based accept-reject sampling via the likelihood-subgradient method (Nygren and Nygren 2006). The Gridtype parameter controls how many tangent points are used to construct the envelope-trading off tightness against computational cost- and the iters component reports the number of candidate samples generated before acceptance.

The output includes posterior samples, prior specifications, dispersion estimates, and envelope diagnostics. Use summary.rlmb (same implementation as summary.rglmb) for posterior summaries and DIC-style output. Formula-based predict methods are provided by **glmbayes** when that package is installed.

Value

rlmb returns a object of class "rlmb". The generic accessor functions coefficients and fitted.values can be used to extract useful features of the value returned by rlmb. An object of class "rlmb" is a list containing at least the following components:

coefficients	a matrix of dimension n by length(mu) with one sample in each row
coef.mode	a vector of length(mu) with the estimated posterior mode coefficients
dispersion	Either a constant provided as part of the call, or a vector of length n with one sample in each row.
Prior	A list with the priors specified for the model in question. Items in the list may vary based on the type of prior
prior.weights	a vector of weights specified or implied by the model
y	a vector with the dependent variable
x	a matrix with the implied design matrix for the model
famfunc	Family functions used during estimation process
iters	an n by 1 matrix giving the number of candidates generated before acceptance for each sample.
Envelope	the envelope that was used during sampling

Objects of class "rlmb" are normally of class c("rlmb","rglmb","glmb","glm","lm"), meaning they inherit from rglmb, glmb, glm, and lm. Well-designed methods for these classes will be applied when appropriate, allowing "rlmb" objects to benefit from existing infrastructure while supporting specialized behavior for restricted linear model priors.

References

Chambers JM (1992). "Linear Models." In Chambers JM, Hastie TJ (eds.), *Statistical Models in S*, chapter 4, 85–124. Wadsworth & Brooks/Cole, Pacific Grove, CA.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

Raiffa H, Schlaifer R (1961). *Applied Statistical Decision Theory*. Clinton Press, Inc., Boston.

Wilkinson GN, Rogers CE (1973). "Symbolic Descriptions of Factorial Models for Analysis of Variance." *Applied Statistics*, **22**(3), 392–399. doi:10.2307/2346786.

Zellner A (1986). "On Assessing Prior Distributions and Bayesian Regression Analysis with g-Prior Distributions." In Goel P~K, Zellner A (eds.), *Bayesian Inference and Decision Techniques: Essays in Honor of Bruno de Finetti*, volume 6 of *Studies in Bayesian Econometrics and Statistics*, 233–243. Elsevier.

See Also

The classical modeling functions lm and glm.

rglmb for the GLM sampler; EnvelopeBuild, EnvelopeOrchestrator for envelope stages used in non-conjugate Gaussian sampling.

pfamily for documentation of pfamily functions used to specify priors.

Prior_Setup, Prior_Check for functions used to initialize and to check priors,

Further reading: (Nygren and Nygren 2006); (Nygren 2025, 2025). User-facing S3 methods for fitted models are provided by **glmbayes**.

glmbayes Modeling Functions rglmb()

Examples

```
## Main Example based on Dobson Plant Weight Data
## Use demo(Ex_07_Schools) for a longer/more complex model

## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.

ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group  <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)

ps <- Prior_Setup(weight ~ group)
x  <- ps$x
mu <- ps$mu
V  <- ps$Sigma
y <- ps$y
shape    <- ps$shape
rate     <- ps$rate
rate_dg  <- if (!is.null(ps$rate_gamma)) ps$rate_gamma else rate


## Two-Block Gibbs sampler for Plant Weight regression model
set.seed(180)

## Note: iteration counts reduced for CRAN checks; increase for production use
n_burnin  <- 200
n_samples <- 200

## Initilize dispersion to ML estimate
dispersion2 <- ps$dispersion

## Run  burn-in iterations
for (i in 1:n_burnin) {
  ## Update block for regression coefficients
  out1 <- rlmb( n = 1, y = y, x = x,
    pfamily = dNormal(mu = mu, Sigma = V, dispersion = dispersion2) )

  ## Update block for dispersion
  out2 <- rlmb(n = 1, y = y, x = x,
    pfamily = dGamma(shape = shape, rate = rate_dg, beta = out1$coefficients[1, ]))
  dispersion2 <- out2$dispersion
```

```
}

## Create Objects to store outputs
beta_out <- matrix(0, nrow = n_samples, ncol = 2)
disp_out <- rep(0, n_samples)

for (i in 1:n_samples) {
  ## Update block for regression coefficients
  out1 <- rlmb( n = 1, y = y, x = x,
                pfamily = dNormal(mu = mu, Sigma = V, dispersion = dispersion2) )

  ## Update block for dispersion
  out2 <- rlmb(n = 1, y = y, x = x,
           pfamily = dGamma(shape = shape, rate = rate_dg, beta = out1$coefficients[1, ]))
  dispersion2 <- out2$dispersion

  ## Store output

  beta_out[i, 1:2] <- out1$coefficients[1, 1:2]
  disp_out[i]      <- out2$dispersion
}

mcmc_two_block <- coda::mcmc(cbind(    beta1 = beta_out[, 1],beta2 = beta_out[, 2],
                                       dispersion = disp_out  ))

## Review output
cat("\nCODA summary (Two-block Gibbs):\n")
print(summary(mcmc_two_block))
cat("\nEffective sample size (dispersion):\n")
print(coda::effectiveSize(mcmc_two_block)["dispersion"])

## rlmb with dGamma prior (dispersion-only; coefficients fixed)
out_rlmb_dGamma <- rlmb(n = 100, y = y, x = x,
  pfamily = dGamma(shape = shape, rate = rate_dg, beta = ps$coefficients),
  weights = rep(1, length(y)))
print(out_rlmb_dGamma)
```

`rNormal_reg.wfit` *Bayesian Weighted Fitting Engines*

Description

These functions provide the Bayesian analogue of `lm.wfit`. They implement the core weighted least squares step used inside Bayesian linear linear models, incorporating prior precision and posterior mode information.

Usage

```
rNormal_reg.wfit(
```

```
  x,
  y,
  P,
  mu,
  w,
  offset = NULL,
  method = "qr",
  tol = 1e-07,
  singular.ok = TRUE,
  ...
)

glmb.wfit(
  x,
  y,
  weights = rep.int(1, nobs),
  offset = rep.int(0, nobs),
  family = gaussian(),
  Bbar,
  P,
  betastar,
  method = "qr",
  tol = 1e-07,
  singular.ok = TRUE,
  ...
)
```

Arguments

x	design matrix of dimension n * p.
y	vector of observations of length n, or a matrix with n rows.
P	Prior precision matrix of dimension p * p.
mu	Prior mean vector of length p.
w	vector of weights (length n) to be used in the fitting process for the wfit functions. Weighted least squares is used with weights w, i.e., sum(w * e^2) is minimized.
offset	(numeric of length n). This can be used to specify an *a priori* known component to be included in the linear predictor during fitting.
method	currently, only method = "qr" is supported.
tol	tolerance for the qr decomposition. Default is 1e-7.
singular.ok	logical. If FALSE, a singular model is an error.
...	currently disregarded.
weights	an optional vector of *prior weights* to be used in the fitting process. Should be NULL or a numeric vector.
family	a description of the error distribution and link function to be used in the model. Should be a family function. (see family for details of family functions.)

Bbar	Prior mean vector of length p.
betastar	Posterior mode vector of length p which has already been estimated.

Details

rNormal_reg.wfit performs the Bayesian weighted least squares update for linear models under a Normal prior.

glmb.wfit performs the corresponding update for generalized linear models, reconstructing the weighted least squares step using the posterior mode and the GLM family functions.

Value

a list with components:

a list wih components:

Examples

```
set.seed(333)
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)

ps <- Prior_Setup(counts ~ outcome + treatment, family = poisson())
mu <- ps$mu
V0 <- ps$Sigma
out <- rglmb(
  n = 1000,
  y = ps$y,
  x = as.matrix(ps$x),
  pfamily = dNormal(mu = mu, Sigma = V0),
  family = poisson(),
  weights = rep(1, nrow(ps$x))
)

betastar <- out$coef.mode
x <- out$x
y <- out$y
offset2 <- 0 * y
weights2 <- out$prior.weights

fit <- glmb.wfit(x, y, weights2, offset2, family = poisson(), Bbar = mu, P = solve(V0), betastar)
influence.measures(fit)

print(fit)
print(out$coef.mode)

mu1 <- 0 * mu
V1 <- 0.1 * V0
out2 <- rglmb(
  n = 1000,
```

```
  y = ps$y,
  x = as.matrix(ps$x),
  pfamily = dNormal(mu = mu1, Sigma = V1),
  family = poisson(),
  weights = rep(1, nrow(ps$x))
)

Bbar2 <- mu1
betastar2 <- out2$coef.mode
fit2 <- glmb.wfit(x, y, weights2, offset2, family = poisson(), Bbar2, P = solve(V1), betastar2)

influence.measures(fit2)

print(fit2)
print(out2$coef.mode)
```

---

rNormalGLM_std *The Bayesian Generalized Linear Model Distribution in Standard Form*

Description

rNormalGLM_std is used to generate iid samplers from Non-Gaussian Generalized Linear Models in standard form. The function should only be called after standardization of a Generalized Linear Model.

Usage

```
rNormalGLM_std(
  n,
  y,
  x,
  mu,
  P,
  alpha,
  wt,
  f2,
  Envelope,
  family,
  link,
  progbar = 1L
)
```

Arguments

n	number of draws to generate. If length(n) > 1, the length is taken to be the number required.
y	a vector of observations of length m.

x	a design matrix of dimension m * p.
mu	a vector of length p giving the prior means of the variables in the design matrix.
P	a positive-definite symmetric matrix of dimension p * p specifying the prior precision matrix of the variable.
alpha	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See model.offset.
wt	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector.
f2	function used to calculate the negative of the log-posterior function
Envelope	an object of type glmbenvelope.
family	family used for simulation. Used that this is different from the family used in other functions.
link	link function used for simulation.
progbar	dummy for flagging if a progressbar should be produced during the call

Details

This function uses the information contained in the constructed envelope list in order to sample from a model in standard form. The simulation proceeds as follows in order to generate each draw in the required number of samples.

1. A random number between 0 and 1 is generated and is used together with the information in the PLSD vector (from the envelope) in order to identify the part of the grid from which a candidate is to be generated.
2. For the part of the grid selected, the dimensions are looped through and a candidate component for each dimension is generated from a restricted normal using information from the Envelope (in particular, the values for logrt, loglt, and cbars corresponding to that the part of the grid selected and the dimension sampled)
3. The log-likelihood for the standardized model is evaluated for the generated candidate (note that the log-likelihood here includes the portion of the prior that was shifted to the log-likelihood)
4. An additional random number is generated and the log of this random number is compared to a log-acceptance rate that is calculated based on the candidate and the LLconst component from the Envelope component selected in order to determine if the candidate should be accepted or rejected
5. If the candidate was not accepted, the process above is repeated from step 1 until a candidate is accepted

Value

A list consisting of the following:

out	A matrix with simulated draws from a model in standard form. Each row represents one draw from the density
draws	A vector with the number of candidates required before acceptance for each draw

Examples

```
############################### Start of rNormalGLM_std examples ###################
data(menarche,package="MASS")

summary(menarche)
plot(Menarche/Total ~ Age, data=menarche)

Age2=menarche$Age-13

x<-matrix(as.numeric(1.0),nrow=length(Age2),ncol=2)
x[,2]=Age2

y=menarche$Menarche/menarche$Total
wt=menarche$Total

mu<-matrix(as.numeric(0.0),nrow=2,ncol=1)
mu[2,1]=(log(0.9/0.1)-log(0.5/0.5))/3

V1<-1*diag(as.numeric(2.0))

# 2 standard deviations for prior estimate at age 13 between 0.1 and 0.9
## Specifies uncertainty around the point estimates

V1[1,1]<-((log(0.9/0.1)-log(0.5/0.5))/2)^2
V1[2,2]=(3*mu[2,1]/2)^2  # Allows slope to be up to 1 times as large as point estimate

famfunc<-glmbfamfunc(binomial(logit))

f1<-famfunc$f1
f2<-famfunc$f2  # Used in optim and glmbsim_cpp
f3<-famfunc$f3  # Used in optim
f5<-famfunc$f5
f6<-famfunc$f6

dispersion2<-as.numeric(1.0)
start <- mu
offset2=rep(as.numeric(0.0),length(y))
P=solve(V1)
n=1000


###### Adjust weight for dispersion

wt2=wt/dispersion2

######################### Shift mean vector to offset so that adjusted model has 0 mean
```

```
alpha=x%*%as.vector(mu)+offset2
mu2=0*as.vector(mu)
P2=P
x2=x


#####  Optimization step to find posterior mode and associated Precision

parin=start-mu

opt_out=optim(parin,f2,f3,y=as.vector(y),x=as.matrix(x),mu=as.vector(mu2),
              P=as.matrix(P),alpha=as.vector(alpha),wt=as.vector(wt2),
              method="BFGS",hessian=TRUE
)

bstar=opt_out$par  ## Posterior mode for adjusted model
bstar
bstar+as.vector(mu)  # mode for actual model
A1=opt_out$hessian # Approximate Precision at mode

## Standardize Model

Standard_Mod=glmb_Standardize_Model(y=as.vector(y), x=as.matrix(x),P=as.matrix(P),
                                    bstar=as.matrix(bstar,ncol=1), A1=as.matrix(A1))

bstar2=Standard_Mod$bstar2
A=Standard_Mod$A
x2=Standard_Mod$x2
mu2=Standard_Mod$mu2
P2=Standard_Mod$P2
L2Inv=Standard_Mod$L2Inv
L3Inv=Standard_Mod$L3Inv

Env2=EnvelopeBuild(as.vector(bstar2), as.matrix(A),y, as.matrix(x2),
                   as.matrix(mu2,ncol=1),as.matrix(P2),as.vector(alpha),as.vector(wt2),
                   family="binomial",link="logit",Gridtype=as.integer(3),
                   n=as.integer(n),sortgrid=TRUE)

## These now seem to match

Env2

# The low-level sampler is called below with explicit type coercions.

### Note: getting the types correct here is important but potentially difficult for users
### May be better to call an R function wrapper that checks (and converts when possible)
##  to correct types

sim=rNormalGLM_std(n=as.integer(n),y=as.vector(y),x=as.matrix(x2),mu=as.matrix(mu2,ncol=1),
                   P=as.matrix(P2),alpha=as.vector(alpha),wt=as.vector(wt2),
                   f2=f2,Envelope=Env2,family="binomial",link="logit",as.integer(0))
```

```
out=L2Inv%*%L3Inv%*%t(sim$out)

for(i in 1:n){
  out[,i]=out[,i]+mu
}

summary(t(out))
mean(sim$draws)
```

---

simfuncs *Simulation Functions for Bayesian Generalized Linear Models*

Description

Simulation functions provide a unified interface for generating posterior samples from Bayesian GLMs. These functions are typically used within model fitting routines such as rglmb and rlmb, and are also suitable for use in Block Gibbs sampling and other simulation-based inference techniques.

Usage

```
simfunction(object, ...)

rNormal_reg(n, y, x, prior_list, offset = NULL, weights = 1,
            family = gaussian(), Gridtype = 2, n_envopt = NULL,
          use_parallel = TRUE, use_opencl = FALSE, verbose = FALSE,progbar=FALSE)

rNormalGamma_reg(n, y, x, prior_list, offset = NULL, weights = 1, family = gaussian(),
                 Gridtype = 2,n_envopt = NULL,
            use_parallel = TRUE, use_opencl = FALSE, verbose = FALSE,progbar=FALSE)

rindepNormalGamma_reg(n, y, x, prior_list, offset = NULL, weights = 1,
                              family = gaussian(), Gridtype = 2,n_envopt = NULL,
                        use_parallel = TRUE, use_opencl = FALSE, verbose = FALSE,
                              progbar = TRUE)

rGamma_reg(n, y, x, prior_list, offset = NULL, weights = 1, family = gaussian(),
           Gridtype = 2,n_envopt = NULL,
          use_parallel = TRUE, use_opencl = FALSE, verbose = FALSE,progbar=FALSE)

## S3 method for class 'rGamma_reg'
print(x, digits = max(3, getOption("digits") - 3), ...)

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

rGamma_Conjugate_reg(n, y, x, prior_list, offset = NULL, weights = 1, family = gaussian(),
           Gridtype = 2,n_envopt = NULL,
          use_parallel = TRUE, use_opencl = FALSE, verbose = FALSE,progbar=FALSE)
```

```
rBeta_reg(n, y, x, prior_list, offset = NULL, weights = 1,
                family = gaussian(), Gridtype = 2, n_envopt = NULL,
                use_parallel = TRUE, use_opencl = FALSE,
                verbose = FALSE, progbar = FALSE)
```

Arguments

object	A fitted model object containing a pfamily component. The generic function simfunction() accesses the simulation metadata stored within such objects.
n	Number of draws to generate. If length(n) > 1, the length is taken to be the number required.
y	A vector of observations of length m.
x	for the simulation functions a design matrix of dimension m * p and for the print functions the object to be printed.
prior_list	A list with prior parameters (e.g., shape, rate, beta) used in the simulation.
offset	Optional numeric vector of length m specifying known components of the linear predictor.
weights	Optional numeric vector of prior weights.
family	A description of the error distribution and link function (see family).
Gridtype	Optional integer specifying the method used to construct the envelope function.
n_envopt	Effective sample size passed to EnvelopeOpt for grid construction. Defaults to match n. Larger values encourage tighter envelopes.
use_parallel	Logical. Whether to use parallel processing.
use_opencl	Logical. Whether to use OpenCL acceleration.
verbose	Logical. Whether to print progress messages.
progbar	Logical. Whether to display a progress base during simulation.
digits	Number of significant digits to use for printed output.
...	Additional arguments passed to or from other methods.

Details

The low-level simulation functions rNormal_reg(), rNormalGamma_reg(), rindepNormalGamma_reg(), and rGamma_reg() generate iid samples from posterior distributions for specific model components. These model functions are used internally by the functions rglmb() and rlmb() to generate samples.

The simfunction() generic extracts metadata from simulation objects, including the function name, call, and arguments used. This is useful for introspection, reproducibility, and diagnostics.

The lower-level simulation functions generate iid samples from posterior distributions for specific model components. These functions are used internally by pfamily constructors and model fitting routines.

Simulation Functions:

- rNormal_reg(): Produces iid draws for regression coefficients in models with multivariate normal priors and log-concave likelihood functions. For Gaussian likelihoods, these are conjugate priors and standard simulation procedures for multivariate normal distributions are utilized (Lindley and Smith 1972; Diaconis and Ylvisaker 1979). For all other families/link functions, the likelihood subgradient approach of (Nygren and Nygren 2006) is used to generate iid samples.
- rNormalGamma_reg(): Produces iid draws for regression coefficients and the dispersion parameter in models with Normal-Gamma priors and Gaussian likelihoods, where this is a conjugate prior distribution. Standard simulation procedures for gamma and multivariate normal distributions are utilized (Raiffa and Schlaifer 1961; Lindley and Smith 1972).
- rindepNormalGamma_reg(): Produces iid draws for regression coefficients and the dispersion parameter in models with independent Normal and truncated Gamma priors. This is a non-conjugate specification but can still be sampled using accept-reject procedures based on an enveloping approach (see vignette (Nygren 2025)).
- rGamma_reg(): Simulates dispersion parameters for Gaussian and Gamma families using either standard gamma sampling or accept-reject methods based on likelihood subgradients (Chen 1979; Nygren 2025).

Value

simfunction() An object of class "simfunction" containing:

name Character string with the name of the simulation function.

call The matched call used to generate the simulation.

args A named list of arguments passed to the simulation function.

rNormal_reg() A list object with classes "rglmb", "glmb", "glm", and "lm". Elements include:

coefficients Matrix (n * p) of simulated regression coefficients, with column names from x.

coef.mode Posterior mode of the coefficients. Gaussian: from lm.fit; non-Gaussian: BFGS mode shifted by prior mean.

dispersion Scalar dispersion used. Poisson/Binomial: 1; otherwise the supplied value. Quasi families: mean residual-based dispersion computed in the wrapper.

Prior List with mean (prior mean vector) and Precision (prior precision matrix P).

prior.weights Vector of prior weights used in the simulation (unscaled).

offset Offset vector passed to the C++ sampler.

offset2 Offset used internally by the wrapper (copy of input or a zero vector).

y Response vector.

x Design matrix.

fit Fitted/diagnostic object. Gaussian: result of lm.fit (class "lm"). Non-Gaussian: result of glmb.wfit(...).

iters Vector of iteration counts per sample. Gaussian: vector of ones; non-Gaussian: counts from the sampler.

Envelope Envelope list used for accept-reject sampling (non-Gaussian); NULL for Gaussian.

family Family object describing distribution and link.

famfunc Processed family functions used internally (e.g., f2, f3).

call Matched call to rNormal_reg().

formula Formula reconstructed from y and x.

model Model frame corresponding to formula.

data Data frame combining y and x.

rNormalGamma_reg() A list with class "rglmb" containing:

coefficients Matrix (n * p) of simulated regression coefficients; row i equals Btilde + IR %*% rnorm(p) * sqrt(dispersion[i]). Column names are set to colnames(x).

coef.mode Posterior mean/mode vector Btilde from rNormal_reg.wfit().

dispersion Numeric vector of length n with draws from the inverse-gamma posterior 1/rgamma(shape = shape + nobs/2, rate = rate + 0.5*S).

Prior List with mean (as numeric vector mu) and Precision (matrix P).

offset Offset vector as supplied.

prior.weights Vector of prior weights wt.

y Response vector.

x Design matrix.

fit Result from rNormal_reg.wfit(), including fields such as Btilde, IR, S, and k.

famfunc Processed family functions for Gaussian models (from glmbfamfunc(gaussian())).

iters Numeric vector (length n) of ones indicating per-draw iteration counts.

Envelope NULL; no envelope is constructed in this conjugate setup.

call Matched call to rNormalGamma_reg().

rindepNormalGamma_reg() A list with class "rglmb" containing:

coefficients Matrix (n * p) of simulated regression coefficients, back-transformed to the original scale; column names set to colnames(x).

coef.mode Vector with the conditional posterior mode used for envelope anchoring (from the Gaussian fit).

dispersion Numeric vector of length n with simulated dispersion draws.

Prior List with prior components: mean (prior mean mu), Sigma (prior covariance), shape and rate (Gamma prior for dispersion), Precision (solve(Sigma)).

family The gaussian() family object.

prior.weights Vector of prior weights used in the simulation.

y Response vector.

x Design matrix.

call Matched call to rindepNormalGamma_reg().

famfunc Processed family functions for Gaussian models (from glmbfamfunc).

iters Vector with per-draw iteration counts returned by the joint sampler.

Envelope NULL; envelope diagnostics are not returned by this function.

loglike NULL; placeholder for log-likelihood values.

weight_out Numeric vector of per-draw weights returned by the C++ routine.

sim_bounds List with low and upp, the dispersion bounds used by the shared envelope.

offset2 Offset vector used internally (copy of input or a zero vector).

rGamma_reg() An object of class "rGamma_reg" containing:

coefficients A 1 * p matrix of assumed regression coefficients.

coef.mode Currently NULL; reserved for future use.

`dispersion` A vector of simulated dispersion values.

`Prior` A list with prior parameters: `shape` and `rate`.

`prior.weights` Vector of prior weights used in the simulation.

`y` The response vector.

Author(s)

The simulation framework was developed by Kjell Nygren as part of the **glmbayes** package. It builds on the likelihood subgradient approach described in (Nygren and Nygren 2006), and extends classical Bayesian GLM sampling techniques.

References

Chen C (1979). "Bayesian Inference for a Normal Dispersion Matrix and Its Application to Stochastic Multiple Regression Analysis." *Journal of the Royal Statistical Society. Series B (Methodological)*, **41**(2), 235–248. doi:10.1111/j.25176161.1979.tb01078.x.

Diaconis P, Ylvisaker D (1979). "Conjugate Priors for Exponential Families." *Annals of Statistics*, **7**(2), 269–281. doi:10.1214/aos/1176344069.

Lindley DV, Smith AFM (1972). "Bayes Estimates for the Linear Model." *Journal of the Royal Statistical Society. Series B (Methodological)*, **34**(1), 1–41. doi:10.1111/j.25176161.1972.tb00899.x.

Nygren K (2025). "Independent Normal–Gamma Regression Sampler." Vignette in the glmbayes R package. R vignette name: independent-norm-gamma.

Nygren K (2025). "Gamma Dispersion Sampling in glmbayes." Vignette in the glmbayes R package. R vignette name: gamma-dispersion.

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

Raiffa H, Schlaifer R (1961). *Applied Statistical Decision Theory*. Clinton Press, Inc., Boston.

See Also

`pfamily`, `rglmb`, `rlmb` for modeling functions that consume simulation functions.

`rNormal_reg`, `rNormalGamma_reg`, `rGamma_reg` for individual simulation functions.

`EnvelopeBuild`, `EnvelopeEval`, `EnvelopeSize` for envelope construction and grid evaluation used in likelihood-subgradient sampling.

Theory and implementation narrative: (Nygren and Nygren 2006); (Nygren 2025, 2025).

glmbayes Simulation Functions `multi_rNormal_reg()`, `two_block_l_for_tv()`, `two_block_mode_weights()`, `two_block_rNormal_reg()`, `two_block_rNormal_reg_v2()`, `two_block_rate()`, `two_block_rate_v2()`, `two_block_tv_bound()`

glmbayes Simulation Functions `multi_rNormal_reg()`, `two_block_l_for_tv()`, `two_block_mode_weights()`, `two_block_rNormal_reg()`, `two_block_rNormal_reg_v2()`, `two_block_rate()`, `two_block_rate_v2()`, `two_block_tv_bound()`

glmbayes Simulation Functions multi_rNormal_reg(), two_block_l_for_tv(), two_block_mode_weights(), two_block_rNormal_reg(), two_block_rNormal_reg_v2(), two_block_rate(), two_block_rate_v2(), two_block_tv_bound()

glmbayes Simulation Functions multi_rNormal_reg(), two_block_l_for_tv(), two_block_mode_weights(), two_block_rNormal_reg(), two_block_rNormal_reg_v2(), two_block_rate(), two_block_rate_v2(), two_block_tv_bound()

glmbayes Simulation Functions multi_rNormal_reg(), two_block_l_for_tv(), two_block_mode_weights(), two_block_rNormal_reg(), two_block_rNormal_reg_v2(), two_block_rate(), two_block_rate_v2(), two_block_tv_bound()

glmbayes Simulation Functions multi_rNormal_reg(), two_block_l_for_tv(), two_block_mode_weights(), two_block_rNormal_reg(), two_block_rNormal_reg_v2(), two_block_rate(), two_block_rate_v2(), two_block_tv_bound()

Examples

```
################################ Start of rNormal_reg examples ###################
set.seed(333)

## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
print(d.AD <- data.frame(treatment, outcome, counts))

## Poisson Prior and rNormal_reg call (using Prior_Setup for x, y, and prior values)
ps <- Prior_Setup(counts ~ outcome + treatment, family = poisson(), data = d.AD)

out_pois <- rNormal_reg(
  n = 1000,
  y = ps$y,
  x = ps$x,
  prior_list = list(mu = ps$mu, Sigma = ps$Sigma),
  family = poisson(link = "log"),
  weights = rep(1, nrow(ps$x))
)
print(out_pois)


## Menarche Binomial Data Example
data(menarche, package = "MASS")
menarche$Age2 <- menarche$Age - 13

## Logit Prior and rNormal_reg call (use proportion + trial weights)
ps1 <- Prior_Setup(
  Menarche / Total ~ Age2,
  family = binomial(logit),
  data = menarche,
  weights = menarche$Total
)

out_logit <- rNormal_reg(
```

```
  n = 1000,
  y = ps1$y,
  x = ps1$x,
  prior_list = list(mu = ps1$mu, Sigma = ps1$Sigma),
  family = binomial(logit),
  weights = menarche$Total
)
print(out_logit)

## Probit Prior and rNormal_reg call
ps2 <- Prior_Setup(
  Menarche / Total ~ Age2,
  family = binomial(probit),
  data = menarche,
  weights = menarche$Total
)

out_probit <- rNormal_reg(
  n = 1000,
  y = ps2$y,
  x = ps2$x,
  prior_list = list(mu = ps2$mu, Sigma = ps2$Sigma),
  family = binomial(probit),
  weights = menarche$Total
)
print(out_probit)

## clog-log Prior and rNormal_reg call
ps3 <- Prior_Setup(
  Menarche / Total ~ Age2,
  family = binomial(cloglog),
  data = menarche,
  weights = menarche$Total
)

out_cloglog <- rNormal_reg(
  n = 1000,
  y = ps3$y,
  x = ps3$x,
  prior_list = list(mu = ps3$mu, Sigma = ps3$Sigma),
  family = binomial(cloglog),
  weights = menarche$Total
)
print(out_cloglog)


## Gamma regression
data(carinsca)
carinsca$Merit <- ordered(carinsca$Merit)
carinsca$Class <- factor(carinsca$Class)
oldopt <- options(contrasts = c("contr.treatment", "contr.treatment"))

psg <- Prior_Setup(
```

```
  Cost / Claims ~ Merit + Class,
  family = Gamma(link = "log"),
  data = carinsca,
  weights = carinsca$Claims
)

out_gamma <- rNormal_reg(
  n = 1000,
  y = psg$y,
  x = psg$x,
  prior_list = list(mu = psg$mu, Sigma = psg$Sigma, dispersion = psg$dispersion),
  family = Gamma(link = "log"),
  weights = carinsca$Claims
)
print(out_gamma)
options(oldopt)
############################### Start of rNormalGamma_reg examples ###################
## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
weight <- c(ctl, trt)

ps=Prior_Setup(weight ~ group)
mu <- ps$mu
shape <- ps$shape
rate <- ps$rate

y <- ps$y
x <- as.matrix(ps$x)
prior_list <- list(mu = mu, Sigma = ps$Sigma_0, shape = shape, rate = rate)
ngamma.D9 <- rNormalGamma_reg(n = 1000, y = y, x = x,
  prior_list = prior_list)

print(ngamma.D9)
############################### Start of rindepNormalGamma_reg examples ###################
## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
weight <- c(ctl, trt)
p_setup <- Prior_Setup(weight ~ group, family = gaussian())

mu <- p_setup$mu
Sigma_prior <- p_setup$Sigma
dispersion <- p_setup$dispersion
shape <- p_setup$shape
rate <- p_setup$rate
y <- p_setup$y
x <- p_setup$x
```

```
prior_list <- list(
  mu = mu,
  Sigma = Sigma_prior,
  dispersion = dispersion,
  shape = shape,
  rate = rate,
  Precision = solve(Sigma_prior),
  max_disp_perc = 0.99
)


set.seed(360)

sim2 <- rindepNormalGamma_reg(n = 1000, y, x, prior_list = prior_list)
print(sim2)




################################ Start of rGamma_reg examples ###################
## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)

## Set up prior hyperparameters (shape/rate) and model matrix via Prior_Setup
ps <- Prior_Setup(weight ~ group, family = gaussian())
y <- ps$y
x <- as.matrix(ps$x)

## rGamma_reg uses a dGamma-style prior on dispersion with fixed beta.
## Use coefficients from Prior_Setup (full-model GLM MLE by default).
prior_list <- list(beta = ps$coefficients, shape = ps$shape, rate = ps$rate)

out <- rGamma_reg(n = 1000, y = y, x = x, prior_list = prior_list, family = gaussian())
summary(out)
```

SimulationPipeline *Low-Level Simulation Pipeline for Bayesian GLMs*

Description

A detailed overview of the low-level simulation pipeline used by `rglmb()` and related functions. These routines implement the optimization -> standardization -> envelope sizing -> envelope construction -> sampling -> back-transformation workflow described in (Nygren and Nygren 2006).

Details

(summaries of each step)

References

Nygren K~N, Nygren L~M (2006). "Likelihood Subgradient Densities." *Journal of the American Statistical Association*, **101**(475), 1144–1156. doi:10.1198/016214506000000357.

Examples

```
############################### Start of rNormalGLM_std examples ###################
data(menarche,package="MASS")

summary(menarche)
plot(Menarche/Total ~ Age, data=menarche)

Age2=menarche$Age-13

x<-matrix(as.numeric(1.0),nrow=length(Age2),ncol=2)
x[,2]=Age2

y=menarche$Menarche/menarche$Total
wt=menarche$Total

mu<-matrix(as.numeric(0.0),nrow=2,ncol=1)
mu[2,1]=(log(0.9/0.1)-log(0.5/0.5))/3

V1<-1*diag(as.numeric(2.0))

# 2 standard deviations for prior estimate at age 13 between 0.1 and 0.9
## Specifies uncertainty around the point estimates

V1[1,1]<-((log(0.9/0.1)-log(0.5/0.5))/2)^2
V1[2,2]=(3*mu[2,1]/2)^2  # Allows slope to be up to 1 times as large as point estimate

famfunc<-glmbfamfunc(binomial(logit))

f1<-famfunc$f1
f2<-famfunc$f2  # Used in optim and glmbsim_cpp
f3<-famfunc$f3  # Used in optim
f5<-famfunc$f5
f6<-famfunc$f6

dispersion2<-as.numeric(1.0)
start <- mu
offset2=rep(as.numeric(0.0),length(y))
P=solve(V1)
n=1000


###### Adjust weight for dispersion
```

```
wt2=wt/dispersion2

######################### Shift mean vector to offset so that adjusted model has 0 mean

alpha=x%*%as.vector(mu)+offset2
mu2=0*as.vector(mu)
P2=P
x2=x


#####  Optimization step to find posterior mode and associated Precision

parin=start-mu

opt_out=optim(parin,f2,f3,y=as.vector(y),x=as.matrix(x),mu=as.vector(mu2),
              P=as.matrix(P),alpha=as.vector(alpha),wt=as.vector(wt2),
              method="BFGS",hessian=TRUE
)

bstar=opt_out$par  ## Posterior mode for adjusted model
bstar
bstar+as.vector(mu)  # mode for actual model
A1=opt_out$hessian # Approximate Precision at mode

## Standardize Model

Standard_Mod=glmb_Standardize_Model(y=as.vector(y), x=as.matrix(x),P=as.matrix(P),
                                    bstar=as.matrix(bstar,ncol=1), A1=as.matrix(A1))

bstar2=Standard_Mod$bstar2
A=Standard_Mod$A
x2=Standard_Mod$x2
mu2=Standard_Mod$mu2
P2=Standard_Mod$P2
L2Inv=Standard_Mod$L2Inv
L3Inv=Standard_Mod$L3Inv

Env2=EnvelopeBuild(as.vector(bstar2), as.matrix(A),y, as.matrix(x2),
                   as.matrix(mu2,ncol=1),as.matrix(P2),as.vector(alpha),as.vector(wt2),
                   family="binomial",link="logit",Gridtype=as.integer(3),
                   n=as.integer(n),sortgrid=TRUE)

## These now seem to match

Env2

# The low-level sampler is called below with explicit type coercions.

### Note: getting the types correct here is important but potentially difficult for users
### May be better to call an R function wrapper that checks (and converts when possible)
##  to correct types

sim=rNormalGLM_std(n=as.integer(n),y=as.vector(y),x=as.matrix(x2),mu=as.matrix(mu2,ncol=1),
```

```
                    P=as.matrix(P2),alpha=as.vector(alpha),wt=as.vector(wt2),
                    f2=f2,Envelope=Env2,family="binomial",link="logit",as.integer(0))

out=L2Inv%*%L3Inv%*%t(sim$out)

for(i in 1:n){
  out[,i]=out[,i]+mu
}

summary(t(out))
mean(sim$draws)
```

---

summary.rGamma_reg *Summarizing Bayesian gamma_reg Distribution Functions*

Description

These functions are all methods for class rGamma_reg or summary.rGamma_reg objects.

Usage

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

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

Arguments

object	an object of class "rGamma_reg" for which a summary is desired.
x	an object of class "summary.rGamma_reg" for which a printed output is desired.
digits	the number of significant digits to use when printing.
...	Additional optional arguments

Value

summary.rGamma_reg() returns an object of class "summary.rGamma_reg", a list containing summaries of posterior draws for the dispersion and precision parameters. Components include:

call	the matched call from the fitted object.
n	number of posterior draws.
coefficients1	matrix of prior means and standard deviations for precision and dispersion.
coefficients	matrix of posterior means, posterior standard deviations, Monte Carlo errors, and empirical tail probabilities.
Percentiles	matrix of posterior percentiles for dispersion draws.

`implied_disp_point`
: dispersion point estimate implied by the Gamma prior on precision, computed as `rate / shape`.

`print.summary.rGamma_reg()` prints the summary object and returns `x` invisibly.

Examples

```
## summary.rGamma_reg: dGamma prior (dispersion-only; coefficients fixed)
## All three functions (rGamma_reg, rglmb, rlmb) use summary.rGamma_reg when
## prior is dGamma.
##
## This example uses the Boston data: Prior_Setup() for hyperparameters and
## ps$coefficients as fixed beta for dGamma / rGamma_reg-style runs.
data("Boston", package = "MASS")

predictors <- setdiff(names(Boston), "medv")
Boston_centered <- Boston
Boston_centered[predictors] <- scale(Boston[predictors], center = TRUE, scale = FALSE)

form <- medv ~
  crim + zn +
  indus + chas + nox + age + dis + rad + tax + ptratio + black + lstat + rm

ps.boston <- Prior_Setup(form, gaussian(), data = Boston_centered)
rate_dg <- if (!is.null(ps.boston$rate_gamma)) ps.boston$rate_gamma else ps.boston$rate

y <- ps.boston$y
x <- as.matrix(ps.boston$x)
wt <- rep(1, length(y))

## 1. rGamma_reg
out1 <- rGamma_reg(
  n = 1000,
  y = y,
  x = x,
 prior_list = list(beta = ps.boston$coefficients, shape = ps.boston$shape, rate = rate_dg),
  offset = rep(0, length(y)),
  weights = wt,
  family = gaussian()
)
summary(out1)

## 2. rglmb
out2 <- rglmb(n = 1000, y = y, x = x,
 pfamily = dGamma(shape = ps.boston$shape, rate = rate_dg, beta = ps.boston$coefficients),
  weights = wt, family = gaussian())
summary(out2)

## 3. rlmb
out3 <- rlmb(n = 1000, y = y, x = x,
 pfamily = dGamma(shape = ps.boston$shape, rate = rate_dg, beta = ps.boston$coefficients),
  weights = wt)
```

```
summary(out3)
```

---

summary.rglmb *Summarizing Bayesian Generalized Linear Model Distribution Functions*

Description

These functions are all methods for class rglmb, rlmb, or summary.rglmb objects.

Usage

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

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

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

Arguments

object	an object of class "rglmb" or "rlmb" for which a summary is desired.
x	an object of class "summary.rglmb" for which a printed output is desired.
digits	the number of significant digits to use when printing.
...	Additional optional arguments

Details

summary.rglmb summarizes output from rglmb or rlmb. For dGamma rate-prior Poisson fits, it delegates to summary.rGamma_reg.

Value

summary.rglmb returns an object of class "summary.rglmb", a list with posterior summaries, DIC-related quantities, and tables suitable for print.summary.rglmb.

See Also

rglmb, rlmb, summary.rGamma_reg, summary.glm, summary.lm.

Examples

```
data(menarche, package = "MASS")

summary(menarche)

Age2 <- menarche$Age - 13

x <- matrix(as.numeric(1.0), nrow = length(Age2), ncol = 2)
x[, 2] <- Age2

y <- menarche$Menarche / menarche$Total
wt <- menarche$Total

mu <- matrix(as.numeric(0.0), nrow = 2, ncol = 1)
mu[2, 1] <- (log(0.9 / 0.1) - log(0.5 / 0.5)) / 3

V1 <- 1 * diag(as.numeric(2.0))

V1[1, 1] <- ((log(0.9 / 0.1) - log(0.5 / 0.5)) / 2)^2
V1[2, 2] <- (3 * mu[2, 1] / 2)^2

out <- rglmb(
  n = 1000, y = y, x = x, pfamily = dNormal(mu = mu, Sigma = V1),
  weights = wt, family = binomial(logit)
)
summary(out)
```

---

two_block_l_for_tv *Sweeps required to reach a TV tolerance*

Description

Smallest `l` such that `two_block_tv_bound(rate, l, method, D0) <= tol`. The bound is decreasing in `l`, so a doubling search followed by bisection is exact.

Usage

```
two_block_l_for_tv(
  rate,
  tol,
  method = c("theorem3", "corollary1"),
  D0 = 0,
  l_max = 1000000L
)
```

Arguments

`rate`	Object from `two_block_rate`.

tol	Target total-variation tolerance in (0, 1).
method	"theorem3" (exact terms) or "corollary1" (geometric envelope).
D0	Optional squared standardized distance of the starting point from the posterior mean, $(x^{(0)} - \mu)^\top \Sigma_{11}^{-1}(x^{(0)} - \mu)$. Default 0 (start at the posterior mean).
l_max	Search cap (error if the bound stays above tol).

Value

Integer: the required number of sweeps.

See Also

two_block_tv_bound

glmbayes Simulation Functions multi_rNormal_reg(), simfuncs, two_block_mode_weights(), two_block_rNormal_reg(), two_block_rNormal_reg_v2(), two_block_rate(), two_block_rate_v2(), two_block_tv_bound()

---

two_block_mode_weights

Likelihood precision weights at the posterior mode

Description

Evaluates the per-observation likelihood precisions (IRLS/Fisher weights) of a GLMM Block 1 model at a supplied random-effects value – typically the joint posterior mode from glmerb_posterior_mode – and assembles the corresponding per-group likelihood precision blocks $Z_j^\top W_j Z_j$.

Usage

```
two_block_mode_weights(
  x,
  block,
  b_mode,
  family = gaussian(),
  wt = 1,
  offset = 0,
  dispersion = NULL,
  group_levels = levels(block)
)
```

Arguments

x	Level-1 RE design matrix Z (l2 x p_re), as passed to two_block_rNormal_reg.
block	Grouping factor or block partition of length l2.
b_mode	J x p_re matrix of random-effect values at which to evaluate the curvature (rows aligned to group_levels), e.g. glmerb_posterior_mode(...)$b_mean.

family	Response family object (default gaussian()).
wt	Prior weights (length 1 or l2; e.g. binomial trial counts). Default 1.
offset	Linear-predictor offset (length 1 or l2). Default 0.
dispersion	Dispersion ϕ. Required for gaussian() and Gamma families; defaults to 1 for poisson and binomial.
group_levels	Character vector defining group order (default levels(block)); must match the rows of b_mode.

Details

The weight for observation i is

$$w_i = \frac{wt_i\,\mu'(\eta_i)^2}{V(\mu_i)\,\phi}, \qquad \eta_i = \text{offset}_i + z_i^\top b_{j(i)},$$

the (expected) negative curvature of its log-likelihood with respect to its own linear predictor. For canonical links (gaussian-identity, poisson-log, binomial-logit) this equals the exact observed Hessian at b_mode; for other links it is the expected (Fisher) information. Examples: gaussian $w_i = wt_i/\phi$; poisson-log $w_i = wt_i\,\lambda_i$; binomial-logit $w_i = wt_i\,p_i(1-p_i)$; binomial-probit $w_i = wt_i\,\phi(\eta_i)^2/[p_i(1-p_i)]$.

The returned weights vector is designed to be passed directly to two_block_rate(weights =) to obtain the local-Gaussian heuristic convergence rate for non-Gaussian families. Note that for non-Gaussian responses the joint posterior is not normal, so rates and TV bounds derived from these weights are a heuristic approximation, not a theorem.

Value

Object of class "two_block_mode_weights": a list with

weights Length-l2 per-observation precisions w_i; pass as two_block_rate(weights =).

eta, mu Linear predictor and fitted means at b_mode.

B_lik Named list (per group) of p_re x p_re likelihood precision blocks $Z_j^\top W_j Z_j$ (the likelihood part of B_j before adding the Block 1 prior precision).

info_total p_re x p_re total $Z^\top W Z$.

family, link, dispersion, group_levels, b_mode, call Inputs echoed for reference.

See Also

two_block_rate, glmerb_posterior_mode, glmbfamfunc

glmbayes Simulation Functions multi_rNormal_reg(), simfuncs, two_block_l_for_tv(), two_block_rNormal_reg(), two_block_rNormal_reg_v2(), two_block_rate(), two_block_rate_v2(), two_block_tv_bound()

two_block_rate *Two-block Gibbs sampler convergence rate (Remark 8 eigenvalues)*

Description

Computes the eigenvalues of $A = P_{11}^{-1/2} P_{12} P_{22}^{-1} P_{21} P_{11}^{-1/2}$ for the joint Gaussian posterior targeted by `two_block_rNormal_reg` with `family = gaussian()` and fixed variance components. The maximal eigenvalue λ^* is the geometric contraction rate of the two-block Gibbs sampler (Nygren 2020, Claim 2 / Remark 8 / Corollary 1): the total variation distance between the l-step kernel and the target decays like $(\lambda^*)^l$.

Usage

```
two_block_rate(
  x,
  block,
  x_hyper,
  prior_list_block1,
  prior_list_block2,
  weights = NULL,
  family = gaussian(),
  group_levels = levels(block)
)
```

Arguments

`x`	Level-1 RE design matrix `Z` (`l2 x p_re`), as passed to `two_block_rNormal_reg`.
`block`	Grouping factor or block partition of length `l2`.
`x_hyper`	Named list of group-level design matrices `X_k` (`J x q_k`), one per column of `x`.
`prior_list_block1`	Block 1 prior: `P` or `Sigma` (`p_re x p_re`); `dispersion` required for `gaussian()` when `weights` is `NULL`.
`prior_list_block2`	Named list of Block 2 prior lists (each with `P` or `Sigma`, `q_k x q_k`).
`weights`	Optional per-observation working weights (length 1 or `l2`). Default for `gaussian()`: `1 / prior_list_block1$dispersion`. Required for non-Gaussian families.
`family`	Response family (default `gaussian()`).
`group_levels`	Character vector defining group order (default `levels(block)`); must match the row order of `x_hyper` when rownames are absent.

Details

Block convention: the paper's x_1 (the block updated second in each sweep) is the Block 2 hyper vector γ of dimension $q = \sum_k q_k$; the paper's x_2 is the Block 1 random-effect stack. A is therefore

$q \times q$ and is computed without ever forming the $Jp_{re} \times Jp_{re}$ Block 1 precision: per group only $p_{re} \times p_{re}$ solves are needed, followed by a single $q \times q$ symmetric eigendecomposition.

For non-Gaussian families the joint posterior is not normal; supplying IRLS-style `weights` evaluated at the posterior mode yields a local-Gaussian heuristic rate (no theorem applies).

Value

Object of class `"two_block_rate"`: a list with `lambda_star` (the Remark 8 rate), `eigenvalues` (full spectrum $a_1 \geq \cdots \geq a_q$ in $[0, 1)$), `m_for_tol` (function: iterations needed so that $(\lambda^*)^m \leq$ `tol`), `S` ($P_{12}P_{22}^{-1}P_{21}$), `P11`, `dims`, `re_names`, `gamma_names`, `group_levels`, `family`, `weights_source`, and `call`.

References

Nygren, K. (2020). *On the total variation distance between multivariate normal densities with applications to two-block Gibbs samplers.* Unpublished manuscript.

See Also

`two_block_rNormal_reg`

glmbayes Simulation Functions `multi_rNormal_reg()`, `simfuncs`, `two_block_l_for_tv()`, `two_block_mode_weights()` `two_block_rNormal_reg()`, `two_block_rNormal_reg_v2()`, `two_block_rate_v2()`, `two_block_tv_bound()`

---

two_block_rate_v2 *Convergence rate for the v2 (pfamily) two-block sampler*

Description

Thin wrapper around `two_block_rate` that accepts the Block~2 priors as `pfamily` objects. For `dNormal` components the fixed `dispersion` is used; for `dIndependent_Normal_Gamma` components the conservative `disp_lower` plug-in is used (the rate is then an upper bound over the truncated tau^2 range).

Usage

```
two_block_rate_v2(
  x,
  block,
  x_hyper,
  prior_list_block1,
  pfamily_list,
  weights = NULL,
  family = gaussian(),
  group_levels = levels(block)
)
```

Arguments

x, block, x_hyper, prior_list_block1, weights, family, group_levels	As in two_block_rate.
pfamily_list	Named list of pfamily objects (one per RE component), as in two_block_rNormal_reg_v2.

Value

Object of class "two_block_rate".

See Also

two_block_rate, two_block_rNormal_reg_v2

glmbayes Simulation Functions multi_rNormal_reg(), simfuncs, two_block_l_for_tv(), two_block_mode_weights() two_block_rNormal_reg(), two_block_rNormal_reg_v2(), two_block_rate(), two_block_tv_bound()

---

two_block_rNormal_reg *Two-block Gibbs sampler for hierarchical regression*

Description

Runs the coupled Block~1 / Block~2 Gibbs sampler for two-block mixed models. Block~1 draws group-level random effects b_j; Block~2 updates hyper means γ_k via multi_rNormal_reg (always Gaussian).

Usage

```
two_block_rNormal_reg(
  n,
  y,
  x,
  block,
  x_hyper,
  prior_list_block1,
  prior_list_block2,
  fixef_start,
  re_coef_names = colnames(x),
  group_levels = levels(block),
  group_name = NULL,
  m_convergence = 10L,
  sampling = c("replicate", "chain"),
  family = gaussian(),
  offset = NULL,
  weights = 1,
  Gridtype = 2L,
  n_envopt = NULL,
  use_parallel = TRUE,
```

```
  use_opencl = FALSE,
  verbose = FALSE,
  seed = NULL,
  progbar = TRUE
)
```

Arguments

n	Number of stored draws.
y	Response vector of length nrow(x).
x	Level-1 design matrix Z (l2 x p_re).
block	Grouping factor or block partition of length l2.
x_hyper	Named list of group-level design matrices X_k (J x q_k), one per column of x.
prior_list_block1	Prior for Block~1: P or Sigma, dispersion (required for gaussian()), optional ddef. mu is updated internally.
prior_list_block2	Named list of Block~2 prior lists passed to multi_rNormal_reg.
fixef_start	Named list of hyper-parameter vectors at which each inner chain is initialised.
re_coef_names	Character vector naming columns of x.
group_levels	Character vector defining row order of Block~1 draws.
group_name	Name for the grouping column in coefficients.
m_convergence	Number of inner Gibbs steps per stored draw.
sampling	Sampling scheme; only "replicate" is implemented.
family	Response family for Block~1 (default gaussian()). Block~2 always uses gaussian().
offset, weights	Passed to Block~1 (length l2 or recycled).
Gridtype, n_envopt, use_parallel, use_opencl, verbose	Passed to Block~1 when family is not Gaussian.
seed	Optional RNG seed.
progbar	Logical; show a text progress bar.

Details

Block~1 follows the same path as rNormal_reg: block_rNormalReg when family = gaussian(), otherwise block_rNormalGLM for the GLM envelope path.

Value

Object of class "two_block_rNormal_reg".

See Also

build_mu_all, block_rNormalReg, block_rNormalGLM, multi_rNormal_reg, rNormal_reg

glmbayes Simulation Functions multi_rNormal_reg(), simfuncs, two_block_l_for_tv(), two_block_mode_weights() two_block_rNormal_reg_v2(), two_block_rate(), two_block_rate_v2(), two_block_tv_bound()

two_block_rNormal_reg_v2 *Two-block Gibbs sampler with pfamily Block 2 priors (development v2)*

Description

Development version of two_block_rNormal_reg where the Block~2 priors are supplied as pfamily objects instead of bare prior lists. Each component of pfamily_list may be a dNormal prior (conjugate gamma_k draw at fixed dispersion, identical to v1) or a dIndependent_Normal_Gamma prior, in which case Block~2 makes a joint (gamma_k, tau^2_k) draw via the likelihood-subgradient envelope sampler (the same path as rglmb with an ING pfamily) and the sampled tau^2_k is fed back into the Block~1 prior precision on the next inner step.

Usage

```
two_block_rNormal_reg_v2(
  n,
  y,
  x,
  block,
  x_hyper,
  prior_list_block1,
  pfamily_list,
  fixef_start,
  re_coef_names = colnames(x),
  group_levels = levels(block),
  group_name = NULL,
  m_convergence = 10L,
  sampling = c("replicate", "chain"),
  family = gaussian(),
  offset = NULL,
  weights = 1,
  Gridtype = 2L,
  n_envopt = NULL,
  use_parallel = TRUE,
  use_opencl = FALSE,
  verbose = FALSE,
  seed = NULL,
  progbar = TRUE
)
```

Arguments

n	Number of stored draws.
y	Response vector of length nrow(x).

x	Level-1 design matrix Z (l2 x p_re).
block	Grouping factor or block partition of length l2.
x_hyper	Named list of group-level design matrices X_k (J x q_k), one per column of x.
prior_list_block1	Prior for Block~1: P or Sigma, dispersion (required for gaussian()), optional ddef. mu is updated internally.
pfamily_list	Named list of pfamily objects, one per column of x: dNormal or dIndependent_Normal_Gamma.
fixef_start	Named list of hyper-parameter vectors at which each inner chain is initialised.
re_coef_names	Character vector naming columns of x.
group_levels	Character vector defining row order of Block~1 draws.
group_name	Name for the grouping column in coefficients.
m_convergence	Number of inner Gibbs steps per stored draw.
sampling	Sampling scheme; only "replicate" is implemented.
family	Response family for Block~1 (default gaussian()). Block~2 always uses gaussian().
offset, weights	Passed to Block~1 (length l2 or recycled).
Gridtype, n_envopt, use_parallel, use_opencl, verbose	Passed to Block~1 when family is not Gaussian.
seed	Optional RNG seed.
progbar	Logical; show a text progress bar.

Details

With dNormal priors throughout, this function produces draws that are identical to two_block_rNormal_reg under the same seed; that equivalence is the regression gate for the v2 development track.

Value

Object of class c("two_block_rNormal_reg_v2", "two_block_rNormal_reg"). Same fields as two_block_rNormal_reg, plus dispersion_fixef_draws: an n x p_re matrix of the Block~2 dispersion (tau^2_k) at each stored draw (constant columns for dNormal components).

See Also

two_block_rNormal_reg, dNormal, dIndependent_Normal_Gamma

glmbayes Simulation Functions multi_rNormal_reg(), simfuncs, two_block_l_for_tv(), two_block_mode_weights() two_block_rNormal_reg(), two_block_rate(), two_block_rate_v2(), two_block_tv_bound()

two_block_tv_bound *Total-variation bound for the two-block Gibbs sampler*

Description

Evaluates the bound on the total variation distance between the l-step kernel of the two-block Gibbs sampler and its target (Nygren 2020), from the eigenvalue spectrum computed by two_block_rate.

Usage

```
two_block_tv_bound(rate, l, method = c("theorem3", "corollary1"), D0 = 0)
```

Arguments

rate	Object from two_block_rate.
l	Integer vector of sweep counts (each >= 1).
method	"theorem3" (exact terms) or "corollary1" (geometric envelope).
D0	Optional squared standardized distance of the starting point from the posterior mean, $(x^{(0)} - \mu)^\top \Sigma_{11}^{-1} (x^{(0)} - \mu)$. Default 0 (start at the posterior mean).

Details

method = "theorem3" evaluates the exact Theorem 3 terms $d_i^{(l)}$ using the closed form $\mathrm{erf}_n(x) = P(\chi_n^2 \le 2x^2)$ with $r_i^{(l)} = (1 - a_{i-1}^{2l})/(1 - a_i^{2l})$. method = "corollary1" evaluates the looser geometric envelope of Corollary 1 (via Remarks 5 and 17), which decays like a_i^{2l} with explicit constants.

When the chain is started at the exact posterior mean (as lmerb does), D0 = 0 and the mean term of both bounds vanishes identically; only the variance-convergence sum remains. The returned bound is capped at 1.

Note the bound applies to the block updated *second* in each sweep (the Block 2 hyper vector γ); the stored Block 1 draw lags by a half-step, so evaluate at l - 1 when calibrating m_convergence for the random-effect draws.

Value

Numeric vector of TV bounds, one per element of l, capped at 1.

References

Nygren, K. (2020). *On the total variation distance between multivariate normal densities with applications to two-block Gibbs samplers.* Unpublished manuscript.

See Also

two_block_rate, two_block_l_for_tv

glmbayes Simulation Functions multi_rNormal_reg(), simfuncs, two_block_l_for_tv(), two_block_mode_weights() two_block_rNormal_reg(), two_block_rNormal_reg_v2(), two_block_rate(), two_block_rate_v2()

Index