Package 'glmbayes'

May 18, 2026

Type Package

Title Bayesian Generalized Linear Models (IID Samples)

Version 0.9.5

Date 2026-05-18

Description Provides Bayesian linear and generalized linear model fitting with independent and identically distributed (iid) posterior samples. The main functions mirror R's lm() and glm() interfaces while adding prior family specifications for Gaussian, Poisson, binomial, and Gamma models with log-concave likelihoods. Sampling for supported non-conjugate models uses accept-reject methods based on likelihood subgradients as in Nygren and Nygren (2006) <doi:10.1198/016214506000000357>. The package also includes tools for prior setup, posterior summaries, prediction, diagnostics, simulation, vignettes, and optional 'OpenCL' acceleration for larger models.

License GPL-2

URL https://CRAN.R-project.org/package=glmbayes, https://github.com/knygren/glmbayes, https://knygren.r-universe.dev/glmbayes

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

Imports stats, coda, Rcpp (>= 1.1.1), RcppParallel, Rdpack (>= 0.11-0)

RdMacros Rdpack

LinkingTo Rcpp, RcppArmadillo, RcppParallel

Depends MASS, R (>= 3.5.0)

Suggests knitr, rmarkdown, testthat (>= 3.0.0), spelling

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

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/edition 3

LazyData true

Language en-US

NeedsCompilation yes

Author Kjell Nygren [aut, cre],
The R Core Team [ctb, cph] (R Mathlib sources, R stats modeling code, and derived/adapted routines),
The R Foundation [cph] (Portions of R Mathlib and R source code),
Ross Ihaka [ctb, cph] (R Mathlib and original R modeling infrastructure),
Robert Gentleman [ctb, cph] (Portions of R Mathlib),
Simon Davies [ctb] (Original R glm implementation),
Morten Welinder [ctb, cph] (Portions of R Mathlib),
Martin Maechler [ctb] (Portions of R Mathlib)

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

Repository CRAN

Date/Publication 2026-05-18 06:00:02 UTC

Contents

`glmbayes-package` *glmbayes: Bayesian Generalized Linear Models with iid Sampling*

Description

`glmbayes` provides independent and identically distributed (iid) samples for Bayesian generalized linear models (GLMs), serving as a Bayesian analogue to the base `glm()` function. Supported likelihood families include Gaussian, Poisson, Binomial, and Gamma models with log-concave likelihoods.

Details

The main user-facing interface is `glmb()`, which mirrors the structure of `glm()` and supports prior specification through `pfamily` objects. Lower-level functions such as `rglmb()` and `rGamma_reg()` provide direct access to the underlying samplers and can be used in block Gibbs sampling or hierarchical model implementations.

For an introduction to the package, examples, and a complete set of vignettes, see:

- README: https://github.com/knygren/glmbayes#readme
- All vignettes: `browseVignettes("glmbayes")`

The package includes extensive documentation on model fitting, prior construction, diagnostics, and optional GPU acceleration using OpenCL.

Releases: version **0.9.5** is on CRAN (`install.packages("glmbayes")`). Source is available from GitHub; R-Universe (https://knygren.r-universe.dev/glmbayes) also builds binaries from that source. Prebuilt CRAN and R-Universe binaries do not include OpenCL; for GPU support with the CRAN release, install from source (see vignette *Chapter 12*).

IID posterior simulation for non-Gaussian GLMs and several non-conjugate linear-model setups uses the likelihood-subgradient envelope method of (Nygren and Nygren 2006). Introductory material and worked examples are in (Nygren 2025, 2025); estimation and simulation background in (Nygren 2025, 2025); prior derivations for `Prior_Setup()` in (Nygren 2025); GPU/OpenCL topics in (Nygren 2025, 2025).

Author(s)

Kjell Nygren

References

Nygren K (2025). "Chapter 00: Introduction." Vignette in the glmbayes R package. R vignette name: Chapter-00.

Nygren K (2025). "Chapter A01: A detailed overview of the glmbayes package." Vignette in the glmbayes R package. R vignette name: Chapter-A01.

Nygren K (2025). "Chapter A02: Overview of Estimation Procedures." Vignette in the glmbayes R package. R vignette name: Chapter-A02.

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

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

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

Nygren K (2025). "Chapter 12: Large Models: GPU Acceleration using OpenCL." Vignette in the glmbayes R package. R vignette name: Chapter-12.

Nygren K (2025). "Chapter A10: Accelerated EnvelopeBuild Implementation using OpenCL." Vignette in the glmbayes R package. R vignette name: Chapter-A10.

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

Main interfaces: glmb, lmb, rglmb, rlmb; low-level simulation API simfuncs; envelope construction EnvelopeBuild.

Useful links:

- CRAN: https://CRAN.R-project.org/package=glmbayes
- GitHub: https://github.com/knygren/glmbayes
- R-Universe: https://knygren.r-universe.dev/glmbayes

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))

## Call to glm
glm.D93 <- glm(counts ~ outcome + treatment,
               family = poisson())

## Using glmb
## Step 1: Set up Prior
ps=Prior_Setup(counts ~ outcome + treatment,family = poisson())
mu=ps$mu
V=ps$Sigma

# Step 2: Call the glmb function
glmb.D93<-glmb(counts ~ outcome + treatment, family=poisson(),
               pfamily=dNormal(mu=mu,Sigma=V))

summary(glmb.D93)

## ----Printed_Views-----------------------------------------------------------
## Printed view of the output from the glm function
print(glm.D93)
## Printed view of the output from the glmb function
print(glmb.D93)

## ----Methods-----------------------------------------------------------------
## Methods for class "lm"
methods(class="lm")

## Methods for class "glm"
methods(class="glm")

## Methods for class "glmb"
methods(class="glmb")

## ----summary-----------------------------------------------------------------
```

```
## summary output for the "glm" class
summary(glm.D93)

## summary output for the "glm" class
summary(glmb.D93)

## ----fitted outputs-------------------------------------------------------
## fitted outputs for the glm function
fitted(glm.D93)

## ----glmb fitted outputs--------------------------------------------------------
## mean of fitted outputs for the glm function
colMeans(fitted(glmb.D93))

## ----predictions----------------------------------------------------------
## predictions for the glm function
predict(glm.D93)

## predictions for the glmb function
colMeans(predict(glmb.D93))

## ----residuals------------------------------------------------------------
## residuals for the glm function
residuals(glm.D93)

## residuals for the glmb function
colMeans(residuals(glmb.D93))

## ----vcov-----------------------------------------------------------------
## vcov for the glm function
vcov(glm.D93)

## vcov for the glm function
vcov(glmb.D93)

## ----confint--------------------------------------------------------------
## confint for the glm function
confint(glm.D93)

## confint for the glm function
confint(glmb.D93)

## ----AIC/DIC--------------------------------------------------------------
## AIC for the glm function (equivalent degrees of freedom and the AIC)
extractAIC(glm.D93)

## DIC for the glmb function (estimated effective number of parameters and the DIC)
extractAIC(glmb.D93)

## ----Deviance-------------------------------------------------------------
## Deviance for the glm function
deviance(glm.D93)
```

```
## Deviance for the glmb function
mean(deviance(glmb.D93))

## ----logLik-------------------------------------------------------------
## Deviance for the glm function
logLik(glm.D93)

## Deviance for the glmb function
mean(logLik(glmb.D93))

## ----Model Frame--------------------------------------------------------
## Model Frame for the glm function
model.frame(glm.D93)

## Model Frame for the glmb function
model.frame(glmb.D93$glm)

## ----formula------------------------------------------------------------
## formula for the glm function
formula(glm.D93)

## ----formula------------------------------------------------------------
## formula for the glmb function
formula(glmb.D93)

## ----family-------------------------------------------------------------
## family for the glm function
family(glm.D93)

## family for the glmb function
family(glmb.D93$glm)

## ----nobs---------------------------------------------------------------
## nobs for the glm function
nobs(glm.D93)

## nobs for the glmb function
nobs(glmb.D93)

## ----show---------------------------------------------------------------
## show for the glm function
show(glm.D93)

## show for the glmb function
show(glmb.D93)
```

---

add_to_path *Add Directories to PATH or LD_LIBRARY_PATH*

Description

These helper functions allow you to add missing directories to the PATH or library search environment variables in a permanent way, minimizing manual editing.

Usage

```
add_to_path_windows(dirs)

add_to_path_linux(dirs)

add_to_libpath_linux(dirs)
```

Arguments

`dirs`	Character vector of directories to add.

Details

- On **Windows**, updates the user-level PATH via PowerShell.
- On **Linux/WSL**, appends export lines to ~/.bashrc for PATH or LD_LIBRARY_PATH.

Value

No return value; called for side effects.

See Also

Sys.getenv, Sys.setenv

Examples

```
############################## Start of add_to_path example ###################


if (interactive()) {
  add_to_path_windows(c("C:/OpenCL/bin"))
  add_to_path_linux(c("/usr/local/cuda/bin"))
  add_to_libpath_linux(c("/usr/local/cuda/lib64"))
}


################################################################################
## End of add_to_path example
################################################################################
```

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

anova.glmb *Analysis of Deviance for Bayesian Generalized Linear Model Fits*

Description

Compute an analysis of deviance table for one (current implementation) or more (future) Bayesian generalized linear model fits. The structure follows the sequential analysis of deviance (McCullagh and Nelder 1989), with Bayesian extensions for DIC, pD, Mahalanobis shift, and directional tail probability (Spiegelhalter et al. 2002).

Usage

```
## S3 method for class 'glmb'
anova(object, ...)
```

Arguments

object	an object of class glmb, typically the result of a call to glmb
...	Other arguments passed to or from other methods.

Details

Specifying a single object (currently only implementation) gives a sequential analysis of deviance table for that fit. The reductions in residual deviance as each term of the formula is added in turn are given as the rows of a table, plus the residual deviances themselves. The Mahalanobis shift and pDirectional columns report prior-posterior disagreement diagnostics; see directional_tail and (Nygren 2025).

Value

An object of class "anova" inheriting from class "data.frame".

References

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

Nygren K (2025). "Chapter A04: Directional Tail Diagnostics for Prior-Posterior Disagreement." Vignette in the glmbayes R package. R vignette name: Chapter-A04.

Spiegelhalter DJ, Best NG, Carlin BP, van der Linde A (2002). "Bayesian Measures of Model Complexity and Fit." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **64**(4), 583–639. doi:10.1111/14679868.00353.

See Also

directional_tail, summary.glmb, glmb, glmbayes-package; rglmb, rlmb, lmb; anova.glm

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())
glmb.D93 <- glmb(
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma)
)
summary(glmb.D93)

# anova for Bayesian Model
# Sequential classical ANOVA (see glm anova below): adding outcome reduces residual deviance;
# adding treatment barely changes it.
# By DIC, the model with outcome but not treatment has the lowest DIC;
# treatment does not improve the criterion.
# Use of traditional anova is questionable in a Bayesian context.
# One may instead use Bayes factors or other approaches.
anova(glmb.D93)


glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())
summary(glm.D93)

# corresponding
anova(glm.D93)
```

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

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**.

Examples

```
############################### Boston_centered dataset example ###################

data("Boston_centered")
head(Boston_centered)
summary(Boston_centered)

## Predictors are mean-centered (column means ~0)
predictors <- setdiff(names(Boston_centered), "medv")
colMeans(Boston_centered[predictors])

form <- medv ~
  crim + zn +
  indus + chas + nox + age + dis + rad + tax + ptratio + black + lstat + rm

lm.boston <- lm(form, data = Boston_centered, x = TRUE, y = TRUE)
summary(lm.boston)

ps <- Prior_Setup(form, gaussian(), data = Boston_centered)

lmb.boston <- lmb(
  form,
  data = Boston_centered,
  pfamily = dNormal(
    mu = ps$mu,
    Sigma = ps$Sigma,
```

```
    dispersion = ps$dispersion
  )
)
summary(lmb.boston)

lmb.boston_v2 <- lmb(
  form,
  data = Boston_centered,
  pfamily = dNormal_Gamma(
    mu = ps$mu,
    Sigma_0 = ps$Sigma_0,
    shape = ps$shape,
    rate = ps$rate
  )
)
summary(lmb.boston_v2)

## Independent Normal-Gamma (OpenCL path when available)

  if (has_opencl()) {
    lmb.boston_v3 <- lmb(
      n = 1000L,
      form,
      data = Boston_centered,
      pfamily = dIndependent_Normal_Gamma(
        ps$mu,
        ps$Sigma,
        shape = ps$shape_ING,
        rate = ps$rate
      ),
      use_parallel = TRUE,
      use_opencl = TRUE,
      verbose = FALSE
    )
    summary(lmb.boston_v3)
  }

################################################################################
## End of Boston_centered dataset example
################################################################################
```

---

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

case.names.glmb *Case and Variable Names of Fitted Models*

Description

Simple utilities returning (non-missing) case names, and (non-eliminated) variable names.

Usage

```
## S3 method for class 'glmb'
case.names(object, full = FALSE, ...)

## S3 method for class 'glmb'
variable.names(object, full = FALSE, ...)
```

Arguments

object	a fitted model object, typically of class "glmb".
full	logical; if TRUE, all names (including zero weights, ...) are returned.
...	further arguments passed to or from other methods.

Value

A character vector

See Also

glmb, glmbayes-package; rglmb, rlmb, lmb; case.names.

Examples

```
## ----dobson----------------------------------------------------------------
## 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())
## Call to glmb
glmb.D93 <- glmb(
  n = 1000,
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma)
)

## ----glmb vcov-------------------------------------------------------------
case.names(glmb.D93)
```

`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

Examples

```
############################## Start of Cleveland dataset example ##################

data("Cleveland")
head(Cleveland)
summary(Cleveland)

# OpenCL-accelerated Bayesian logistic regression example
# This only runs if OpenCL is available

if (has_opencl()) {
  ps <- Prior_Setup(
    hd ~ age + sex + cp + trestbps + chol +
      fbs + restecg + thalach + exang + oldpeak + slope + ca + thal,
    family = binomial(logit),
    data = Cleveland
  )

  fit <- glmb(
    hd ~ age + sex + cp + trestbps + chol +
      fbs + restecg + thalach + exang + oldpeak + slope + ca + thal,
    family       = binomial(link = "logit"),
    pfamily      = dNormal(mu = ps$mu, Sigma = ps$Sigma),
    data         = Cleveland,
    n            = 1000,
    Gridtype     = 2,
    use_parallel = TRUE,
    use_opencl   = TRUE,
    verbose      = FALSE
  )
  summary(fit)
}

##########################################################################################
## End of Cleveland dataset example
##########################################################################################
```

`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

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

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

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

  the usual weighted residual–df estimator.
- Prior Gamma rate:

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

  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.

---

confint.glmb *Credible Intervals for Model Parameters*

Description

Computes Bayesian credible intervals for model parameters from posterior draws. Intervals are constructed as quantiles of the posterior distribution, following the standard approach for posterior summaries (Gelman et al. 2013).

Usage

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

Arguments

object	a fitted model object of class "glmb". Typically the result of a call to glmb.
parm	a specification (not yet implemented) of which parameters are to be given credible sets, either a vector of numbers or a vector of names. If missing, all parameters are considered.
level	the credible interval required.
...	additional argument(s) for methods.

Value

A matrix (or vector) with columns giving lower and upper credible limits for each parameter. These will be labeled (1-level)/2 and 1-(1-level)/2 in \

References

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

See Also

summary.glmb, vcov.glmb, glmb, glmbayes-package; rglmb, rlmb, lmb; confint for classical confidence intervals

Examples

```
## ----dobson----------------------------------------------------------------
## 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())
## Call to glmb
glmb.D93 <- glmb(
  n = 1000,
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma)
)
## ----glmb confint----------------------------------------------------------
confint(glmb.D93)
```

`deviance.rglmb` *Model Deviance*

Description

Returns the deviance of a fitted Bayesian Generalized Linear Model

Usage

```
## S3 method for class 'rglmb'
deviance(object, ...)
```

Arguments

`object`	an object of class `"rglmb"`, typically the result of a call to rglmb
`...`	further arguments to or from other methods

Value

A vector with the deviance extracted from the `object`.

See Also

`rglmb`; `glmb`, `glmbayes-package`; `rlmb`, `lmb`; `deviance`.

Examples

```
## ----dobson----------------------------------------------------------------
## 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())
## Call to glmb
glmb.D93 <- glmb(
  n = 1000,
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma)
)
## ----rglmb deviance--------------------------------------------------------
deviance(glmb.D93)
```

`diagnose_glmbayes` *GPU and OpenCL Diagnostics for glmbayes*

Description

A collection of tools for detecting GPU hardware, verifying OpenCL availability, checking driver installation, validating environment configuration, and diagnosing whether **glmbayes** can use GPU acceleration. These functions provide both high-level diagnostic summaries and low-level checks of system components such as PATH, library directories, OpenCL headers, and the ICD loader.

The diagnostic workflow is centered around `diagnose_glmbayes()`, which orchestrates all other checks and prints a detailed, human-readable report. Lower-level helpers can be called individually for programmatic inspection or automated testing.

Usage

```
diagnose_glmbayes()

detect_environment_and_gpus()

gpu_names()

detect_or_install_gpu_drivers(info)

detect_compute_runtimes(info)

has_opencl()

verify_opencl_runtime(lib_dirs = NULL)

check_runtime_env(runtime_info)
```

Arguments

`info`	A list returned by `detect_environment_and_gpus()`. The list must contain the following elements: **environment** One of "windows", "msys2", "linux", "wsl", or "unknown". **nvidia** A list with elements `present` (logical) and `names` (character). **amd** A list with elements `present` (logical) and `names` (character). **intel** A list with elements `present` (logical) and `names` (character).
`lib_dirs`	A list of OpenCL directories
`runtime_info`	The structured list returned by `detect_compute_runtimes()`.

Details

GPU acceleration speeds up **envelope construction and grid evaluation** (e.g. large 3^p grids or many tangency evaluations) when you pass `use_opencl = TRUE` in modeling and envelope functions such as `glmb` and `rglmb`. OpenCL is **vendor-neutral** (NVIDIA, AMD, Intel); CPU-only builds remain valid and are often used when no OpenCL stack is present.

Practical setup (summary). The CRAN release and prebuilt R-Universe binaries are built **without** OpenCL GPU support; enabling the GPU path usually requires installing the development **glmbayes from source** on a machine with OpenCL **headers**, a linkable **OpenCL library / ICD loader**, and a working **vendor runtime** (GPU driver). You need a normal C/C++ toolchain (e.g. Rtools on Windows, `build-essential` and `r-base-dev` on Linux, Xcode CLT plus GCC on macOS for source installs). Vendor-specific notes (CUDA Toolkit vs Intel SDK vs Khronos headers on Windows, `opencl-headers` and `ocl-icd` packages on Linux, etc.) are spelled out in (Nygren 2025).

What this help page checks. A usable OpenCL environment requires:

1. OpenCL headers (e.g., `CL/cl.h`) at compile time,
2. the OpenCL ICD loader (e.g., `libOpenCL.so.1`) at runtime,
3. correct PATH and library search paths (especially on Linux/WSL),
4. a functional OpenCL platform and device (driver installed).

The functions here inspect these pieces. On Linux and WSL, `verify_opencl_runtime()` tries to create a platform, device, context, queue, and compile a minimal kernel. On Windows, that probe is skipped because platform-creation failures are often uninformative; rely on `diagnose_glmbayes()` and driver/runtime detection instead.

Start with `diagnose_glmbayes()` for a single readable report; use `has_opencl()` for a quick boolean when scripting.

Value

A structured list with diagnostics for each runtime, including:

- installed: whether the runtime was detected
- found_path_dirs: directories already present in PATH
- missing_path_dirs: directories that should be added to PATH

- found_lib_dirs: directories already present in LD_LIBRARY_PATH
- missing_lib_dirs: directories that should be added to LD_LIBRARY_PATH
- include_dirs: include directories detected for headers

Most functions return structured lists describing detected hardware, drivers, runtimes, or environment issues. `diagnose_glmbayes()` prints a formatted report and invisibly returns a named list containing all intermediate diagnostic results.

High-level diagnostic

- `diagnose_glmbayes()` — full GPU/OpenCL diagnostic report.

Environment and hardware detection

- `detect_environment_and_gpus()` — detect OS and GPU vendor.
- `gpu_names()` — enumerate available GPU device names.
- `detect_compute_runtimes()` — detect CUDA/OpenCL runtimes.

OpenCL availability and runtime checks

- `has_opencl()` — quick check for OpenCL support.
- `verify_opencl_runtime()` — probe OpenCL platform/device availability.
- `check_runtime_env()` — validate PATH and library directories.

Driver installation helpers

- `detect_or_install_gpu_drivers()` — detect driver presence and issues.

PATH and library path utilities

These are optional helpers used by the diagnostic pipeline.

- `add_to_path_windows()`
- `add_to_path_linux()`
- `add_to_libpath_linux()`

References

Nygren K (2025). "Chapter 12: Large Models: GPU Acceleration using OpenCL." Vignette in the glmbayes R package. R vignette name: Chapter-12.

See Also

`diagnose_glmbayes`, `detect_environment_and_gpus`, `detect_compute_runtimes`, `verify_opencl_runtime`, `has_opencl`.

Modeling with `use_opencl`: `glmb`, `rglmb`. Envelope helpers: `EnvelopeBuild`, `EnvelopeEval`.

Full install and troubleshooting: `vignette("Chapter-12", package = "glmbayes")` ((Nygren 2025)); implementation notes: (Nygren 2025).

`directional_tail` *Directional Tail Diagnostic*

Description

Computes the directional tail probability based on posterior draws and prior mean, using whitening transformation and projection onto the direction of disagreement. This diagnostic identifies directional disagreement between posterior and prior, and is especially useful for visualizing rejection regions in whitened space. The whitening uses Mahalanobis distance (Mahalanobis 1936) in posterior-precision-scaled coordinates.

Usage

```
directional_tail(fit, mu0 = NULL)

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

Arguments

`fit`	A fitted model object of class 'glmb' or 'lmb'
`mu0`	An optional argument containing a reference vector relative to which the directional tail is computed. Defaults to the prior mean.
`x`	An object of class `directional_tail`
`...`	Additional arguments passed to or from other methods.

Details

Whitening is performed using the posterior precision matrix. The direction vector is computed as the mean shift in whitened space. Tail probability is the proportion of draws with negative projection onto this direction. For theory, interpretation, and relation to t/F statistics, see (Nygren 2025).

Value

An object of class 'directional_tail' containing:

`mahalanobis_shift`	Measures the standardized Mahalanobis distance between the posterior and prior means, using posterior precision for scaling. In the Gaussian case, this directly determines the directional tail probability via Phi(-‖w‖).
`p_directional`	Directional tail probability (proportion of draws in the direction of disagreement)
`delta`	Mean shift in whitened space
`draws`	List containing whitened draws, raw draws, and tail flags

References

Mahalanobis PC (1936). "On the generalized distance in statistics." *Proceedings of the National Institute of Sciences of India*, **2**(1), 49–55.

Nygren K (2025). "Chapter A04: Directional Tail Diagnostics for Prior-Posterior Disagreement." Vignette in the glmbayes R package. R vignette name: Chapter-A04.

See Also

summary.glmb, anova.glmb

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)

dat  <- data.frame(weight, group)
dat2 <- rbind(dat)

lm.D9_null <- lm(weight ~ 1, data = dat2)
summary(lm.D9_null)

lm.D9_default <- lm(weight ~ group, data = dat2)
summary(lm.D9_default)

ps_null <- Prior_Setup(
  weight ~ group,
  family = gaussian(),
  pwt = 0.01,
  intercept_source = "null_model",
  effects_source   = "null_effects",
  data = dat2
)
mu_null     <- ps_null$mu
V_null      <- ps_null$Sigma
disp_ML_null <- ps_null$dispersion

lmb.D9_null <- lmb(
  weight ~ group,
  dNormal(mu_null, V_null, dispersion = disp_ML_null),
  data = dat2,
  n = 10000
)
summary(lmb.D9_null)

glmb.D9_default <- glmb(
  weight ~ group,
  family  = gaussian(),
```

```
  pfamily = dNormal(mu_null, V_null, dispersion = disp_ML_null),
  data    = dat2
)
summary(glmb.D9_default)

colMeans(residuals(lmb.D9_null))

glm.D9_default <- glm(weight ~ group, family = gaussian(), data = dat2)
summary(glm.D9_default)
disp_D9_default <- summary(glm.D9_default)$dispersion

solve(vcov(lmb.D9_null$lm))
t(lmb.D9_null$lm$x) %*% lmb.D9_null$lm$x / disp_D9_default
t(lmb.D9_null$lm$x) %*% lmb.D9_null$lm$x * (20 / 18)

tailprobs <- directional_tail(lmb.D9_null)
tailprobs

summary(lm.D9_null)
summary(lm.D9_null)$sigma^2

tailprobs$Prec_lik * summary(lm.D9_null)$sigma^2
t(lmb.D9_null$x) %*% lmb.D9_null$x
tailprobs$p_directional

########################################

Z     <- tailprobs$draws$Z
flag  <- tailprobs$draws$is_tail
delta <- tailprobs$delta
w     <- tailprobs$delta

asp <- 1

## Plot posterior draws in whitened space
plot(
  Z,
  col = ifelse(flag, "red", "blue"),
  pch = 19,
  xlab = "Z1",
  ylab = "Z2",
  main = "Directional Tail Diagnostic"
)

abline(a = 0, b = -w[1] / w[2], col = "darkgreen", lty = 2)

## Add radius boundary centered at posterior mode (delta)
r <- sqrt(sum(delta^2))
symbols(
  delta[1], delta[2],
  circles = r,
  inches  = FALSE,
  add     = TRUE,
```

```
  lwd     = 2,
  fg      = "gray"
)

## Add prior mean at origin
points(0, 0, pch = 4, col = "black", lwd = 2)

## Add posterior mode at delta
points(delta[1], delta[2], pch = 3, col = "purple", lwd = 2)

## Add legend
legend(
  "topright",
  legend = c(
    "Tail draws", "Non-tail draws", "Direction vector",
    "Radius boundary", "Prior mean", "Posterior mode"
  ),
  col = c("red", "blue", "darkgreen", "gray", "black", "purple"),
  pch = c(19, 19, NA, NA, 4, 3),
  lty = c(NA, NA, 1, 1, NA, NA),
  lwd = c(NA, NA, 2, 2, 2, 2),
  bty = "n"
)

############################## Original Scales ###################

B       <- tailprobs$draws$B
flag    <- tailprobs$draws$is_tail
mu0     <- as.numeric(lmb.D9_null$Prior$mean)
mu_post <- colMeans(B)
x_range <- range(B[, 1])  # Intercept values
padding <- diff(x_range) * 0.1  # 10% margin

oldpar <- par(no.readonly = TRUE)
par(mar = c(5, 6, 4, 2))  # bottom, left, top, right

plot(
  B,
  col  = ifelse(flag, "red", "blue"),
  pch  = 19,
  xlab = "Intercept",
  ylab = "groupTrt",
  xlim = c(x_range[1] - padding, x_range[2] + padding),
  main = "Directional Tail Diagnostic (Raw Space)"
)

points(mu0[1], mu0[2], pch = 4, col = "black", cex = 1.5)       # Prior mean
points(mu_post[1], mu_post[2], pch = 3, col = "darkgreen", cex = 1.5)  # Posterior mean

legend(
  "topright",
  legend = c("Tail draws", "Non-tail draws", "Prior", "Posterior"),
  col    = c("red", "blue", "black", "darkgreen"),
```

```
  pch    = c(19, 19, 4, 3)
)

par(oldpar)
```

`dummy.coef.glmb` *Extract Coefficients in Original Coding*

Description

Extracts coefficients in terms of the original factor levels rather than the coded variables. The logic mirrors `dummy.coef` for `lm` objects (Chambers 1992; Venables and Ripley 2002).

Usage

```
## S3 method for class 'glmb'
dummy.coef(object, use.na = FALSE, ...)

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

Arguments

`object`	a `glmb` model fit
`use.na`	logical flag for coefficients in a singular model. If use.na is true, undetermined coefficients will be missing; if false they will get one possible value.
`x`	object to be printed
`...`	arguments passed to or from other methods

Details

A fitted linear model has coefficients for the contrasts of the factor terms, usually one less in number than the number of levels. This function re-expresses the coefficients in the original coding; as the coefficients will have been fitted in the reduced basis, any implied constraints (e.g., zero sum for contr.helmert or contr.sum) will be respected. There will be little point in using dummy.coef for contr.treatment contrasts, as the missing coefficients are by definition zero.

Value

A list giving for each term the draws for the coefficients.

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.

Venables W~N, Ripley B~D (2002). *Modern Applied Statistics with S*. Springer, New York.

See Also

summary.glmb, glmb, glmbayes-package; rglmb, rlmb, lmb; dummy.coef

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())
glmb.D93 <- glmb(
  n = 1000,
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma)
)
summary(glmb.D93)


myd <- dummy.coef(glmb.D93)
print(myd)
```

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

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

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 $U_2 \sim \mathrm{Unif}(0, 1)$.

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, glmb. 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., $\mathbb{E}[\text{RSS}(\beta) \mid y, \phi]$ 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; glmb, 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 $T_1 = L_{\text{test}} - \mathrm{UB1}$ and $T = T_1 - (\mathrm{UB2} + \mathrm{UB3A} + \mathrm{UB3B})$. The code draws $U_2 \sim \mathrm{Unif}(0, 1)$ 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 $r = b_0 + \mathrm{RSS}_{\mathrm{post}}/2$ where $\mathrm{RSS}_{\mathrm{post}}$ 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 * l1).
`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:

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

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
```

`extractAIC.glmb` *Extract DIC from a Fitted Bayesian Model*

Description

Computes the Deviance Information Criterion (DIC) and the effective number of parameters for Bayesian generalized linear models fitted via `glmb()` or `rglmb()`. The DIC, introduced by (Spiegelhalter et al. 2002), provides a Bayesian analog to the AIC (Akaike 1974).

Usage

```
## S3 method for class 'glmb'
extractAIC(fit, ...)

## S3 method for class 'rglmb'
extractAIC(fit, ...)

extractDIC(fit, ...)
```

Arguments

`fit`	A fitted model of class "`glmb`" or "`rglmb`".
`...`	Additional arguments passed to or from methods.

Value

A named numeric vector with components:

pD Estimated effective number of parameters

DIC Deviance Information Criterion

References

Akaike H (1974). "A new look at the statistical model identification." *IEEE Transactions on Automatic Control*, **19**(6), 716–723. doi:10.1109/TAC.1974.1100705.

Spiegelhalter DJ, Best NG, Carlin BP, van der Linde A (2002). "Bayesian Measures of Model Complexity and Fit." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **64**(4), 583–639. doi:10.1111/14679868.00353.

See Also

summary.glmb, glmb, glmbayes-package, rglmb, rlmb, lmb; extractAIC for the classical AIC computation on lm/glm fits

Examples

```
## ----dobson-------------------------------------------------------------------
## 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())
## Call to glmb
glmb.D93 <- glmb(
  n = 1000,
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma)
)
## ----glmb extractAIC----------------------------------------------------------
extractAIC(glmb.D93)
```

---

formula.summary.rglmb *Model Formulae*

Description

This function is a method function for the "summary.rglmb" class used to Extract a formulae for the objective and the family

Usage

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

Arguments

x	an object of class summary.rglmb, typically the result of a call to summary.rglmb
...	further arguments to or from other methods

Value

The function returns model formulae

See Also

rglmb, summary.rglmb; glmb, glmbayes-package; rlmb, lmb; 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
##############################################################################
```

---

get_opencl_core_count *Get the number of available OpenCL compute units*

Description

Returns the number of compute units (cores) available on the default OpenCL device. This can be useful for diagnostics or performance tuning.

Usage

```
get_opencl_core_count()
```

Value

Integer count of compute units.

---

glmb *Fitting Bayesian Generalized Linear Models*

Description

glmb is used to fit Bayesian generalized linear models, specified by giving a symbolic descriptions of the linear predictor, the error distribution, and the prior distribution.

Usage

```
glmb(
  formula,
  family = binomial,
  pfamily = dNormal(mu, Sigma, dispersion = 1),
  n = 1000,
  data,
  weights,
```

```
  use_parallel = TRUE,
  use_opencl = FALSE,
  verbose = FALSE,
  subset,
  offset,
  na.action,
  Gridtype = 2,
  n_envopt = NULL,
  start = NULL,
  etastart,
  mustart,
  control = list(...),
  model = TRUE,
  method = "glm.fit",
  x = FALSE,
  y = TRUE,
  contrasts = NULL,
  ...
)

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

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 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).
n	number of draws to generate. If length(n) > 1, the length is taken to be the number required.
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.
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.

subset	an optional vector specifying a subset of observations to be used in the fitting process.
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 documentation for model.offset at model.extract.
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 stats{na.omit}. Another possible value is NULL, no action. Value stats{na.exclude} can be useful.
Gridtype	an optional argument specifying the method used to determine the number of tangent points used to construct the enveloping function.
n_envopt	Effective sample size passed to EnvelopeOpt for grid construction. Defaults to match n. Larger values encourage tighter envelopes.
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.
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.
digits	the number of significant digits to use when printing.

Details

The function glmb is a Bayesian version of the classical glm function. 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).

Setup (including the use of formulas and families) mirrors that of glm but adds a required pfamily argument to specify the prior distribution. 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, glmb generates independent draws from the posterior density- no MCMC chains are required. Results can be printed or summarized with methods that mirror those for glm (e.g.\ print.glmb, summary.glmb), as well as all the usual glm/lm generics (predict, residuals, etc.).

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, glmb draws n = 1000 samples, uses parallel CPU simulation, and-if use_opencl = TRUE-GPU-accelerated envelope building. "glmb" comes with many of the same kinds of method functions that come with "glm" and "lm", so you can still call extractAIC, fitted.values, or any other standard method.

The lmb function is a Bayesian version of the lm function that can be used to estimate models from the Gaussian family without the need for a family argument.

rglmb and rlmb are functions with more minimalistic interfaces for estimating the same models without most of the internal overhead (these functions are called internally by glmb and lmb). The reduced overhead may be beneficial for Gibbs sampling implementations.

Value

glmb returns an object of class "glmb". The function summary (i.e., summary.glmb) can be used to obtain or print a summary of the results. The generic accessor functions coefficients, fitted.values, residuals, and extractAIC can be used to extract various useful features of the value returned by glmb.

An object of class "glmb" is a list containing at least the following components:

glm	an object of class "glm" containing the output from a call to the function glm
coefficients	a matrix of dimension n by length(mu) with one sample in each row
coef.means	a vector of length(mu) with the estimated posterior mean coefficients
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 list may vary based on the type of prior
fitted.values	a matrix of dimension n by length(y) with one sample for the mean fitted values in each row
family	the family object used.
linear.predictors	an n by length(y) matrix with one sample for the linear fit on the link scale in each row
deviance	an n by 1 matrix with one sample for the deviance in each row
pD	An Estimate for the effective number of parameters
Dbar	Expected value for minus twice the log-likelihood function
Dthetabar	Value of minus twice the log-likelihood function evaluated at the mean value for the coefficients
DIC	Estimated Deviance Information criterion
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
model	if requested (the default),the model frame
call	the matched call
formula	the formula supplie
terms	the terms object used
data	the data argument
famfunc	Family functions used during estimation process
iters	an n by 1 matrix giving the number of candidates generated before acceptance for each sample.
contrasts	(where relevant) the contrasts used.
xlevels	(where relevant) a record of the levels of the factors used in fitting
digits	the number of significant digits to use when printing.

In addition, non-empty fits will have (yet to be implemented) components qr, R and effects relating to the final weighted linear fit for the posterior mode. Objects of class "glmb" are normall of class c("glmb","glm","lm"), that is inherit from classes glm and lm and well-designed methods from those classed will be applied when appropriate.

If a binomial glmb model was specified by giving a two-column response, the weights returned by prior.weights are the total number of cases (factored by the supplied case weights) and the component of y of the result is the proportion of successes.

Author(s)

The R implementation of glmb has been written by Kjell Nygren and was built to be a Bayesian version of the glm function and hence tries to mirror the features of the glm function to the greatest extent possible. For details on the author(s) for the glm function see the documentation for glm.

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

See Also

`lm`, `glm`, `family`, `formula` for classical modeling functions, family objects, and formula syntax

`pfamily` for documentation of pfamily functions used to specify priors.

`Prior_Setup`, `Prior_Check` for functions used to initialize and to check priors,

`EnvelopeBuild` for envelope construction methods.

Further reading: (Nygren and Nygren 2006); (Nygren 2025, 2025); OpenCL/GPU: (Nygren 2025, 2025).

`summary.glmb`, `predict.glmb`, `residuals.glmb`, `simulate.glmb`, `extractAIC.glmb`, `dummy.coef.glmb` and methods(class="glmb") for `glmb` and the methods and generic functions for classes `glm` and `lm` from which class `glmb` inherits.

glmbayes Modeling Functions `lmb()`, `rglmb()`, `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 Model
ps=Prior_Setup(counts ~ outcome + treatment,family = poisson())
mu=ps$mu
V=ps$Sigma
```

```
# Step 2: Call the glmb function
glmb.D93<-glmb(counts ~ outcome + treatment, family=poisson(),
               pfamily=dNormal(mu=mu,Sigma=V))
summary(glmb.D93)


# Menarche Binomial Data Example
data(menarche,package="MASS")
Age2=menarche$Age-13

## Classical Model

glm.out<-glm(cbind(Menarche, Total-Menarche) ~ Age2, family=binomial(logit), data=menarche)
summary(glm.out)

## Logit Prior and Model
ps1=Prior_Setup(cbind(Menarche, Total-Menarche) ~ Age2,family=binomial(logit), data=menarche)

glmb.out1<-glmb(cbind(Menarche, Total-Menarche) ~ Age2,
family=binomial(logit),pfamily=dNormal(mu=ps1$mu,Sigma=ps1$Sigma), data=menarche)
summary(glmb.out1)

## Posterior mean fitted probabilities on the response scale (see also
## \code{vignette("Chapter-04", package = "glmbayes")})
require(graphics)
pred1 <- predict(glmb.out1, type = "response")
pred1_m <- colMeans(pred1)
plot(
  Menarche / Total ~ Age,
  data = menarche,
  main = "Proportion with menarche (data and posterior mean fit)"
)
lines(menarche$Age, pred1_m, col = "blue", lwd = 2)

## Probit Prior and Model
ps2=Prior_Setup(cbind(Menarche, Total-Menarche) ~ Age2,family=binomial(probit), data=menarche)

glmb.out2<-glmb(cbind(Menarche, Total-Menarche) ~ Age2,
           family=binomial(probit),pfamily=dNormal(mu=ps2$mu,Sigma=ps2$Sigma), data=menarche)
summary(glmb.out2)

## clog-log Prior and Model
ps3=Prior_Setup(cbind(Menarche, Total-Menarche) ~ Age2,family=binomial(cloglog), data=menarche)

glmb.out3<-glmb(cbind(Menarche, Total-Menarche) ~ Age2,
           family=binomial(cloglog),pfamily=dNormal(mu=ps3$mu,Sigma=ps3$Sigma), data=menarche)
summary(glmb.out3)

## Comparison of DIC Statistics

DIC_Out=rbind(extractAIC(glmb.out1),extractAIC(glmb.out2),extractAIC(glmb.out3))
rownames(DIC_Out)=c("logit","probit","clog-log")
```

```
colnames(DIC_Out)=c("pD","DIC")
DIC_Out

### Gamma regression

data(carinsca)
carinsca$Merit <- ordered(carinsca$Merit)
carinsca$Class <- factor(carinsca$Class)
oldopt <- options(contrasts = c("contr.treatment", "contr.treatment"))
Claims=carinsca$Claims
Insured=carinsca$Insured
Merit=carinsca$Merit
Class=carinsca$Class
Cost=carinsca$Cost


out <- glm(Cost/Claims~Merit+Class,family=Gamma(link="log"),weights=Claims,x=TRUE)
summary(out)
disp=gamma.dispersion(out)
ps=Prior_Setup(Cost/Claims~Merit+Class,family=Gamma(link="log"),weights=Claims)
mu=ps$mu
V=ps$Sigma

out3 <- glmb(Cost/Claims~Merit+Class,family=Gamma(link="log"),
             pfamily=dNormal(mu=mu,Sigma=V,dispersion=disp),weights=Claims)

summary(out)
summary(out3)

options(oldopt)


## glmb 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$rate
out_glmb_dGamma <- glmb(weight ~ group, family = gaussian(),
  pfamily = dGamma(shape = ps_dg$shape, rate = rate_dg, beta = ps_dg$coefficients))
summary(out_glmb_dGamma)
```

---

glmb.influence.measures

Bayesian Regression Diagnostics

Description

These functions compute Cook's distance, dfbetas, dffits, covratio, and standardized/studentized residuals for fitted Bayesian GLM objects. They delegate to the corresponding **stats** functions applied to the fitted GLM component.

Usage

```
glmb.influence.measures(model, infl = influence(model))

## S3 method for class 'glmb'
rstandard(model, ..., infl = influence(model))

## S3 method for class 'glmb'
rstudent(model, ..., infl = influence(model))

glmb.dffits(model, infl = influence(model))

## S3 method for class 'glmb'
dfbetas(model, ..., infl = influence(model))

glmb.covratio(model, infl = influence(model))

## S3 method for class 'glmb'
cooks.distance(model, ..., infl = influence(model))
```

Arguments

model	an R object, typically returned by lm or glm.
infl	influence structure as returned by influence.glmb (the latter only for the glm method of rstudent and cooks.distance).
...	further arguments passed to or from other methods.

Value

a list with components as returned by the underlying **stats** functions.

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
glmb.D93 <- glmb(
  counts ~ outcome + treatment,
```

```
  family = poisson(),
  pfamily = dNormal(mu = mu, Sigma = V0)
)

## Start setup here [First output from earlier optim optimization]
betastar <- glmb.D93$coef.mode # Posterior mode from optim
x <- glmb.D93$x
y <- glmb.D93$y
# The fitted object does not currently store an offset, so use zeros here.
offset2 <- 0 * y # Should return this from lower level functions
weights2 <- glmb.D93$prior.weights

## Check influence measures for original model
fit <- glmb.wfit(x, y, weights2, offset2, family = poisson(), Bbar = mu, P = solve(V0), betastar)
influence.measures(fit)

print(fit)
print(glmb.D93$coef.mode)

### Now try a strong prior with poorly chosen intercept
mu1 <- 0 * mu
V1 <- 0.1 * V0
glmb2.D93 <- glmb(
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = mu1, Sigma = V1)
)

Bbar2 <- mu1 # Prior mean
betastar2 <- glmb2.D93$coef.mode # Posterior mode from optim
fit2 <- glmb.wfit(x, y, weights2, offset2, family = poisson(), Bbar2, P = solve(V1), betastar2)

influence.measures(fit2)

print(fit2)
print(glmb2.D93$coef.mode)



influence(glmb2.D93)
influence(glmb2.D93, do.coef = TRUE)
influence(glmb2.D93, do.coef = FALSE)

glmb.influence.measures(glmb2.D93, influence(glmb2.D93))


## Do not need method functions
dfbeta(glmb2.D93, influence(glmb2.D93))
dfbeta(glmb2.D93$fit, influence(glmb2.D93))
hatvalues(glmb2.D93, influence(glmb2.D93))
hatvalues(glmb2.D93$fit, influence(glmb2.D93))

# Implemented methods
```

```
rstandard(glmb2.D93, infl = influence(glmb2.D93))
rstandard(glmb2.D93)


# Methods requiring dedicated handling

## Needs a method function
dfbetas(glmb2.D93)
dfbetas(glmb2.D93, influence(glmb2.D93, do.coef = TRUE))
dfbetas(glmb2.D93$fit, influence(glmb2.D93))



# Needs a method function
cooks.distance(glmb2.D93)
cooks.distance(glmb2.D93$fit, influence(glmb2.D93))



# The rstandard method now works directly on glmb objects.


# Needs a method function
rstudent(glmb2.D93)
rstudent(glmb2.D93$fit, influence(glmb2.D93))


# Not a method, separate function
glmb.dffits(glmb2.D93)
dffits(glmb2.D93$fit, influence(glmb2.D93))

# Not a method - separate function
glmb.covratio(glmb2.D93)
covratio(glmb2.D93$fit, influence(glmb2.D93))



# Needs a method function but requires a different approach
# The influence function actually stores this measure
## This seems to require more work to create a method function

infl <- influence(glmb2.D93)
hat2 <- infl$hat
hat2
```

---

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 glmb, and rglmb functions.

Usage

```
glmbfamfunc(family)

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

Arguments

family	an object of class family
x	an object of class "glmbfamfunc" for which a printed output is desired.
...	additional optional arguments

Details

For simulation, many code paths now pass closed-form objectives into C++ directly; glmbfamfunc remains the canonical R closure bundle for the same likelihood/prior/deviance quantities and for post-processing (e.g.\ logLik.glmb, summary.rglmb, directional_tail).

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.\ by directional_tail for glmb fits).

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))
```

---

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
```

`influence.glmb` *Bayesian Regression Diagnostics*

Description

This function provides the basic quantities which are used in forming a wide variety of diagnostics for checking the quality of Bayesian regression fits. These methods delegate to `influence`, `cooks.distance`, `dfbetas`, and related functions in the **stats** package, applied to the fitted GLM component stored in the model object.

Usage

```
## S3 method for class 'glmb'
influence(model, ...)
```

Arguments

`model`	an object as returned by `lm` or `glm`.
`...`	further arguments passed to or from other methods.

Details

Cook's distance was introduced by (Cook 1977). The dfbetas, dffits, and covratio diagnostics follow the framework of (Belsley et al. 1980). Because `glmb` and `lmb` store coefficient draws rather than a single mode, these methods use the fitted `fit` component (from the underlying `glm`/`lm` fit at the posterior mode) for influence calculations.

Value

a `list` with components as returned by `influence`.

References

Belsley DA, Kuh E, Welsch RE (1980). *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. Wiley, New York. doi:10.1002/0471725153.

Cook RD (1977). "Detection of influential observation in linear regression." *Technometrics*, **19**(1), 15–18. doi:10.2307/1268249.

See Also

glmb, glmbayes-package; lmb, rglmb, rlmb; summary.glmb; influence, influence.measures, cooks.distance, dfbetas

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
glmb.D93 <- glmb(
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = mu, Sigma = V0)
)

## Start setup here [First output from earlier optim optimization]
betastar <- glmb.D93$coef.mode # Posterior mode from optim
x <- glmb.D93$x
y <- glmb.D93$y
# The fitted object does not currently store an offset, so use zeros here.
offset2 <- 0 * y # Should return this from lower level functions
weights2 <- glmb.D93$prior.weights

## Check influence measures for original model
fit <- glmb.wfit(x, y, weights2, offset2, family = poisson(), Bbar = mu, P = solve(V0), betastar)
influence.measures(fit)

print(fit)
print(glmb.D93$coef.mode)

### Now try a strong prior with poorly chosen intercept
mu1 <- 0 * mu
V1 <- 0.1 * V0
glmb2.D93 <- glmb(
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = mu1, Sigma = V1)
```

```
)

Bbar2 <- mu1 # Prior mean
betastar2 <- glmb2.D93$coef.mode # Posterior mode from optim
fit2 <- glmb.wfit(x, y, weights2, offset2, family = poisson(), Bbar2, P = solve(V1), betastar2)

influence.measures(fit2)

print(fit2)
print(glmb2.D93$coef.mode)



influence(glmb2.D93)
influence(glmb2.D93, do.coef = TRUE)
influence(glmb2.D93, do.coef = FALSE)

glmb.influence.measures(glmb2.D93, influence(glmb2.D93))


## Do not need method functions
dfbeta(glmb2.D93, influence(glmb2.D93))
dfbeta(glmb2.D93$fit, influence(glmb2.D93))
hatvalues(glmb2.D93, influence(glmb2.D93))
hatvalues(glmb2.D93$fit, influence(glmb2.D93))

# Implemented methods

rstandard(glmb2.D93, infl = influence(glmb2.D93))
rstandard(glmb2.D93)


# Methods requiring dedicated handling

## Needs a method function
dfbetas(glmb2.D93)
dfbetas(glmb2.D93, influence(glmb2.D93, do.coef = TRUE))
dfbetas(glmb2.D93$fit, influence(glmb2.D93))



# Needs a method function
cooks.distance(glmb2.D93)
cooks.distance(glmb2.D93$fit, influence(glmb2.D93))



# The rstandard method now works directly on glmb objects.


# Needs a method function
rstudent(glmb2.D93)
rstudent(glmb2.D93$fit, influence(glmb2.D93))
```

```
# Not a method, separate function
glmb.dffits(glmb2.D93)
dffits(glmb2.D93$fit, influence(glmb2.D93))

# Not a method - separate function
glmb.covratio(glmb2.D93)
covratio(glmb2.D93$fit, influence(glmb2.D93))



# Needs a method function but requires a different approach
# The influence function actually stores this measure
## This seems to require more work to create a method function

infl <- influence(glmb2.D93)
hat2 <- infl$hat
hat2
```

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

---

lmb *Fitting Bayesian Linear Models*

Description

lmb is used to fit Bayesian linear models, specified by giving a symbolic descriptions of the linear predictor and the prior distribution.

Usage

```
lmb(
  formula,
  pfamily,
  n = 1000,
  data,
  subset,
  weights,
  na.action,
  method = "qr",
  model = TRUE,
  x = TRUE,
  y = TRUE,
  qr = TRUE,
  singular.ok = TRUE,
  contrasts = NULL,
  offset,
  Gridtype = 2,
  n_envopt = NULL,
  use_parallel = TRUE,
  use_opencl = FALSE,
  verbose = FALSE,
  ...
)
```

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

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'.
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).
n	number of draws to generate. If length(n) > 1, the length is taken to be the number required.
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 lm is called.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
weights	an optional vector of weights to be used in the fitting process. Should be NULL or a numeric vector. If non-NULL, weighted least squares is used with weights weights (that is, minimizing sum(w*e^2)); otherwise ordinary least squares is used. See also 'Details',
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 stats{na.omit}. Another possible value is NULL, no action. Value stats{na.exclude} can be useful.
method	the method to be used in fitting the classical model during a call to glm. 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.
model, x, y, qr	logicals. If TRUE the corresponding components of the fit (the model frame, the model matrix, the response, the QR decomposition) are returned.
singular.ok	logical. If FALSE (the default in S but not in R) a singular fit is an error.
contrasts	an optional list. See the contrasts.arg of model.matrix.default.
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 or matrix of extents matching those of the response. One or more offset terms can be included in the formula instead or as well, and if more than one are specified their sum is used. See model.offset.
Gridtype	an optional argument specifying the method used to determine the number of tangent points used to construct the enveloping 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 during simulation.
use_opencl	Logical. Whether to use OpenCL acceleration during Envelope construction.
verbose	Logical. Whether to print progress messages.
...	For lm(): additional arguments to be passed to the low level regression fitting functions (see below).
digits	the number of significant digits to use when printing.

Details

The function lmb is a Bayesian extension of the classical lm function. It retains the familiar formula interface and model setup used in lm, while introducing posterior simulation and prior specification via the pfamily argument. Internally, lmb calls lm to obtain the classical least squares fit, then generates independent draws from the posterior distribution using either multivariate normal simulation (for Gaussian priors) or accept-reject sampling via likelihood-subgradient envelopes (Nygren and Nygren 2006).

The symbolic formula interface follows (Wilkinson and Rogers 1973), and the overall design of lm was inspired by the S system (Chambers 1992). lmb comes with many of the same types of generic methods that are available to lm and glm, including predict, residuals, extractAIC, and summary. Many of these are inherited from glmb.

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. All models support the dNormal prior; the Gaussian family also supports dNormalGamma and dIndependent_Normal_Gamma, which allow for more flexible prior structures including independent priors on variance components.

Posterior draws are generated using the prior specification provided via pfamily. For Gaussian models with conjugate priors, draws are obtained directly from the posterior distribution using standard simulation procedures for multivariate normal densities (Raiffa and Schlaifer 1961). For non-conjugate setups, the function uses envelope-based accept-reject sampling, where the Gridtype parameter controls the granularity of the envelope construction. The number of candidates generated before acceptance is returned in the iters component.

The output includes both the classical lm fit and Bayesian diagnostics such as the Deviance Information Criterion (DIC), effective number of parameters (pD), and posterior summaries. The DIC, introduced by (Spiegelhalter et al. 2002), provides a Bayesian analog to AIC by balancing model fit and complexity using posterior expectations. This dual structure allows users to compare classical and Bayesian fits side-by-side, and to leverage familiar modeling workflows while gaining access to richer inferential tools.

The lmb function is a specialized version of glmb for Gaussian models, and does not require a family argument. For conjugate models, it uses standard simulation methods for posterior draws, avoiding the need for envelope construction or subgradient sampling. Like glmb, it returns objects compatible with many standard methods from lm and glm, including extractAIC, fitted.values, and residuals.

For more minimalistic workflows, rlmb and rglmb offer stripped-down interfaces for posterior sampling without the overhead of full model objects. rlmb is called from within lmb. The functions rlmb might be useful in Gibbs sampling or simulation-heavy contexts.

Value

lmb returns an object of class "lmb". The function summary (i.e., summary.glmb) can be used to obtain or print a summary of the results. The generic accessor functions coefficients, fitted.values, residuals, and extractAIC can be used to extract various useful features of the value returned by lmb.

An object of class "lmb" is a list containing at least the following components:

lm	an object of class "lm" containing the output from a call to the function lm
coefficients	a matrix of dimension n by length(mu) with one sample in each row
coef.means	a vector of length(mu) with the estimated posterior mean coefficients
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 list may vary based on the type of prior
residuals	a matrix of dimension n by length(y) with one sample for the deviance residuals in each row
fitted.values	a matrix of dimension n by length(y) with one sample for the fitted values in each row
linear.predictors	an n by length(y) matrix with one sample for the linear fit on the link scale in each row
deviance	an n by 1 matrix with one sample for the deviance in each row
pD	An Estimate for the effective number of parameters
Dbar	Expected value for minus twice the log-likelihood function
Dthetabar	Value of minus twice the log-likelihood function evaluated at the mean value for the coefficients
DIC	Estimated Deviance Information criterion
weights	a vector of weights specified or implied by the model
prior.weights	a vector of weights specified or implied by the model
y	a vector of observations of length m.
x	a design matrix of dimension m * p
model	if requested (the default),the model frame
call	the matched call
formula	the formula supplied
terms	the terms object used
data	the data argument

famfunc	family functions used during estimation and post processing
iters	an n by 1 matrix giving the number of candidates generated before acceptance for each sample.
contrasts	(where relevant) the contrasts used.
xlevels	(where relevant) a record of the levels of the factors used in fitting
pfamily	the prior family specified
digits	the number of significant digits to use when printing.

In addition, non-empty fits will have (yet to be implemented) components qr, R and effects relating to the final weighted linear fit for the posterior mode. Objects of class "lmb" are normally of class c("lmb","glmb","glm","lm"), that is inherit from classes glmb. glm and lm and well-designed methods from those classed will be applied when appropriate.

Author(s)

The R implementation of lmb has been written by Kjell Nygren and was built to be a Bayesian version of the lm function and hence tries to mirror the features of the lm function to the greatest extent possible while also taking advantage of some of the method functions developed for the glmb function. For details on the author(s) for the lm function see the documentation for lm.

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.

Spiegelhalter DJ, Best NG, Carlin BP, van der Linde A (2002). "Bayesian Measures of Model Complexity and Fit." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **64**(4), 583–639. doi:10.1111/14679868.00353.

Wilkinson GN, Rogers CE (1973). "Symbolic Descriptions of Factorial Models for Analysis of Variance." *Applied Statistics*, **22**(3), 392–399. doi:10.2307/2346786.

See Also

The classical modeling functions lm and glm.

glmb, rglmb, rlmb for related Bayesian GLM/linear interfaces; EnvelopeBuild for envelope construction when accept–reject sampling is used.

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); independent Normal–Gamma sampler: (Nygren 2025).

summary.glmb, predict.glmb, simulate.glmb, extractAIC.glmb, dummy.coef.glmb and methods(class="glmb") for methods inherited from class glmb and the methods and generic functions for classes glm and lm from which class lmb also inherits.

glmbayes Modeling Functions glmb(), rglmb(), rlmb()

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, gaussian())
# Prior_Setup supplies the prior mean and covariance components used below.

## Classical model
lm.D9 <- lm(weight ~ group, x = TRUE, y = TRUE)
summary(lm.D9)
vcov(lm.D9)
s <- summary(lm.D9)
disp_classical <- s$sigma^2
cat("Classical lm dispersion (sigma^2 = RSS/(n-p)):", disp_classical, "\n")

## Conjugate Normal Prior (fixed dispersion)
lmb.D9 <- lmb(
  weight ~ group,
  pfamily = dNormal(mu = ps$mu, ps$Sigma, dispersion = ps$dispersion)
)
summary(lmb.D9)
vcov(lmb.D9)

## Conjugate Normal_Gamma Prior (second argument is Sigma_0 from Prior_Setup)
lmb.D9_v2 <- lmb(
  weight ~ group,
  pfamily = dNormal_Gamma(
    ps$mu,
    Sigma_0 = ps$Sigma_0,
    shape = ps$shape,
    rate  = ps$rate
  )
)
summary(lmb.D9_v2)
vcov(lmb.D9_v2)

## Independent_Normal_Gamma_Prior (same mu, Sigma, rate as dNormal_Gamma; shape = ps$shape_ING)

lmb.D9_v3 <- lmb(
  weight ~ group,
  dIndependent_Normal_Gamma(
    ps$mu,
```

```
    ps$Sigma,
    shape = ps$shape_ING,
    rate  = ps$rate
  )
)
summary(lmb.D9_v3)


## anova
anova(lmb.D9)

## lmb with dGamma prior (dispersion-only; coefficients fixed)
rate_dg <- if (!is.null(ps$rate_gamma)) ps$rate_gamma else ps$rate
out_lmb_dGamma <- lmb(
  weight ~ group,
  pfamily = dGamma(shape = ps$shape, rate = rate_dg, beta = ps$coefficients))
summary(out_lmb_dGamma)
```

`load_kernel_source` *Load OpenCL Kernel Source Files*

Description

These functions provide a user-facing interface for loading OpenCL kernel source files and kernel libraries from the package's `cl/` directory. They call internal C++ routines that perform file lookup, dependency resolution, and concatenation of kernel sources.

Usage

```
load_kernel_source(relative_path, package = "glmbayes")

load_kernel_library(subdir, package = "glmbayes", verbose = FALSE)
```

Arguments

`relative_path`	A file path inside the package's `cl/` directory. Used by `load_kernel_source()` to load a single `.cl` file.
`package`	Name of the package containing the OpenCL sources. Defaults to `"glmbayes"`.
`subdir`	A subdirectory inside `cl/` containing a set of `.cl` files annotated with `@provides` and `@depends` tags. Used by `load_kernel_library()` to construct a dependency-resolved kernel library.
`verbose`	Logical; print diagnostic information during dependency resolution (default: `FALSE`).

Details

OpenCL support is optional. If the package was built without OpenCL (e.g., on systems lacking OpenCL headers or drivers), these functions return a clear error message.

Value

A character string containing the kernel source code or combined kernel library.

OpenCL Availability

Use has_opencl to check whether OpenCL support is available in the current build of **glmbayes**.

How These Functions Assemble an OpenCL Program

The functions load_kernel_source() and load_kernel_library() are the fundamental tools used by **glmbayes** to construct complete OpenCL programs from modular components. OpenCL kernels in this package are not stored as monolithic .cl files. Instead, they are built dynamically by concatenating several layers of source code, each serving a distinct purpose in the final GPU program.

A typical OpenCL program used by **glmbayes** is assembled in the following order:

1. **Global configuration header** The file OPENCL.cl defines global extensions, IEEE constants, helper macros, and device-side utilities. It plays a role analogous to a C/C++ header file and must appear at the very top of every combined kernel module. It enables features such as double-precision arithmetic (cl_khr_fp64) and device-side debugging (cl_khr_printf).
2. **Mathematical library modules** Subdirectories such as "rmath", "dpq", and "nmath" contain collections of .cl files implementing mathematical functions used throughout the GLM likelihood and gradient computations.

   ```
   Each file may declare \code{@provides} and \code{@depends} tags.
   \code{load_kernel_library()} reads all files in a subdirectory,
   parses these annotations, performs a dependency-aware topological sort,
   and concatenates the files in an order that guarantees that upstream
   functions appear before downstream callers.

   This mechanism is conceptually similar to a sequence of
   \code{#include} statements in C/C++, but with automatic dependency
   resolution.
   ```

3. **Model-specific helper functions** Some kernels require additional device-side utilities that are not part of the shared libraries. These are typically loaded using load_kernel_source() and appended after the library modules.
4. **Final kernel entry function** The last component is the model-specific kernel that OpenCL will execute on the device. For example, in the function f2_f3_opencl(), the final kernel is selected based on the GLM family and link function (e.g., "f2_f3_binomial_logit").

   ```
   This kernel must appear last in the combined program, after all helper
   functions and libraries have been defined, because OpenCL requires that
   all functions be defined before they are referenced.
   ```

The resulting program is a single, syntactically valid OpenCL source string that is passed directly to the OpenCL compiler (e.g., via clBuildProgram). The ordering performed by load_kernel_library() is essential for successful compilation and ensures that the GPU kernels used by **glmbayes** are reproducible, modular, and maintainable.

The function `f2_f3_opencl()` provides a concrete example of this assembly process: it loads the global configuration header, then the mathematical libraries, then the model-specific kernel file, and finally concatenates these components into a complete OpenCL program that is sent to the GPU for evaluation of the log-likelihood and gradient.

Preparing a Global Configuration Header

Most OpenCL programs begin with a small configuration header that enables device extensions, defines IEEE constants, and provides utility macros used throughout the kernel code. The file `OPENCL.cl` included in the examples illustrates one such header, but users are free to design their own.

A configuration header typically includes:

- **Extension declarations** such as `#pragma OPENCL EXTENSION cl_khr_fp64 : enable` to activate double-precision arithmetic.
- **IEEE constants** (e.g., `ML_NAN`, `ML_POSINF`), which many statistical kernels rely on.
- **Utility macros** such as `INLINE` or `R_UNUSED` to improve readability and suppress warnings.
- **Optional helper definitions** such as work-item macros, typedefs, or device-side debugging tools.

This header should appear at the very top of every combined OpenCL program. The function `load_kernel_source()` is typically used to load it.

Library-Level Header Files

In addition to individual `.cl` files that define functions, most OpenCL libraries also require a *library-level header file*. This file contains constants, macros, error-handling definitions, and other shared utilities that apply to the entire library. It is conceptually similar to a C/C++ header such as `<math.h>` or `Rmath.h`.

A library-level header file:

- defines numerical constants (e.g., `ML_NAN`, `ML_POSINF`),
- provides OpenCL-safe versions of R's mathlib macros,
- stubs out error and warning hooks for device-side execution,
- defines validation macros such as `ISNAN` and `R_FINITE`,
- declares error codes and error-handling helpers, and
- provides any library-wide constants (e.g., `WILCOX_MAX`).

This file is typically named after the library itself (e.g., `"nmath.cl"`) and is placed in the same directory as the other library files. It should be loaded *before* any of the function-level files in the library, because those files may rely on macros or constants defined here.

A simplified example of such a header is shown below:

```
// nmath.cl - OpenCL math constants, macros & remaps for GPU kernels

// Stub out R's error/warning hooks for OpenCL
```

```
#ifndef MATHLIB_ERROR
  #define MATHLIB_ERROR(fmt, ...) /* no-op */
#endif

// Numerical constants
#define ML_POSINF (1.0 / 0.0)
#define ML_NEGINF (-1.0 / 0.0)
#define ML_NAN    (0.0 / 0.0)

// Error codes
#define ME_DOMAIN    1
#define ME_RANGE     2
#define ME_NOCONV    4
#define ME_PRECISION 8

// Validation macros
#define ISNAN(x)    (isnan(x))
#define R_FINITE(x) (isfinite(x))

// Error-handling macros
#define ML_ERR_return_NAN \
    do { ML_ERROR(ME_DOMAIN, fname); return ML_NAN; } while(0)

// Library-wide constants
#define WILCOX_MAX 50
```

When `load_kernel_library()` processes a library directory, it first loads this header file (if present) and then loads the remaining files in dependency-correct order. This ensures that all macros and constants are available before any function-level code is compiled.

Library-level headers are optional but strongly recommended. They provide a clean, centralized place to define constants and macros that would otherwise need to be duplicated across multiple files. This structure is general and applies to any OpenCL project, not only to statistical or mathematical libraries.

Preparing Library Files

A library file is a single `.cl` source file containing one or more device-side functions. To allow `load_kernel_library()` to assemble these files in a dependency-correct order, each file should begin with a small annotation block describing what the file provides and what it depends on.

A typical library file begins with three comment lines:

```
// @provides: symbol1, symbol2, ...
// @depends:  fileA, fileB, ...
// @includes: library_name
```

`@provides` A comma-separated list of function names defined in this file. These names are used by the dependency resolver to determine which files satisfy which requirements.

`@depends` A comma-separated list of *file names* (excluding the `.cl` extension) that this file depends on. Only *direct* dependencies should be listed. For example, if functions in file A call functions in file B, and functions in file B call functions in file C, then file A should list only `B` in its `@depends` tag—not `C`. Every file in a library must list the library's header file (the file whose name matches the library directory) in its `@depends` tag, along with any additional files it directly relies on. This ensures that the header is always loaded first and that all dependencies are resolved in the correct order.

`@includes` The name of the library to which this file belongs (e.g., `"nmath"`). This tag does *not* affect dependency ordering. Instead, it serves as metadata that groups related files into a coherent library. The `load_kernel_library()` function uses this information for diagnostics and consistency checks.

The following example illustrates the recommended format:

```
// expm1.cl - OpenCL adaptation of expm1.c
// @provides: expm1
// @depends:  nmath, log1p
// @includes: nmath

#ifndef HAVE_EXPM1
double expm1(double x) {
    ...
}
#endif
```

The dependency resolver in `load_kernel_library()` reads the `@provides` and `@depends` tags, constructs a directed graph of file dependencies, and performs a topological sort to ensure that:

- all required files are loaded before they are referenced,
- files with no dependencies (typically the library header) are loaded first,
- files depending on others are loaded later, and
- circular or missing dependencies are detected and reported.

The `@includes` tag is not used for dependency resolution; it simply identifies the library grouping to which the file belongs.

This annotation format is general and applies to any OpenCL project. It allows users to port existing C/C++ mathematical libraries into OpenCL simply by translating the functions into `.cl` files and adding the appropriate `@provides` and `@depends` tags.

Dependency Resolution, Circular Logic, and Verbose Output

The `load_kernel_library()` function performs a dependency-aware topological sort of all `.cl` files in the specified library directory. Each file is promoted into the sorted load order only when all of the file names listed in its `@depends` tag have already been promoted. Files with an empty `@depends` list (typically the library header file) are promoted first.

If, during the sorting process, no additional files can be promoted, the function concludes that the remaining files have either:

- circular dependencies (e.g., file A depends on file B, and file B depends on file A), or
- missing dependencies (e.g., a file lists a dependency that does not correspond to any `.cl` file in the directory).

In such cases, the function throws a runtime error:

```
"Dependency sort failed: unresolved dependencies remain."
```

This error prevents the construction of an invalid or incomplete OpenCL program.

When `verbose = TRUE`, `load_kernel_library()` prints detailed diagnostic information describing each stage of the dependency-resolution process. The verbose output includes:

- the list of all `.cl` files discovered in the library,
- the initial set of files with no dependencies,
- the set of unsorted files and their dependency counts,
- each pass of the while-loop used for topological sorting,
- for each file, whether each dependency has already been satisfied,
- which files are promoted on each pass, and
- a final summary of the sorted load order.

If circular or missing dependencies are detected while `verbose = TRUE`, the function prints the list of files that could not be promoted before raising the error. This makes it straightforward for users to identify incorrect or incomplete `@depends` tags. In rare cases, files may need to be split or functions rewritten to eliminate genuine circular dependencies.

Writing Kernel Files

After preparing a configuration header and any required libraries, users typically write one or more OpenCL *kernel* files. A kernel is the entry point executed on the device and is usually designed for tasks that are embarrassingly parallel.

A kernel file should:

- begin with any required `#pragma OPENCL EXTENSION` lines,
- include any constants or local macros needed by the computation,
- define a `__kernel void` function that operates on global buffers, and
- avoid dependencies on host-side libraries (OpenCL C is a restricted subset of C99).

The example below illustrates a kernel that computes a quadratic form and gradient for each grid point in parallel. It demonstrates typical OpenCL idioms such as:

- using `get_global_id(0)` to index work-items,
- reading from `__global` buffers,
- accumulating results in local arrays, and
- writing results back to global memory.

Kernel files are usually loaded with `load_kernel_source()` and placed at the end of the assembled program so that all helper functions and library routines are defined before the kernel is compiled.

Kernel Runners and Kernel Wrappers

Although not strictly required, it is strongly recommended to separate OpenCL execution into two layers: a *kernel runner* and a *kernel wrapper*. This is the design used throughout the **glmbayes** implementation and provides a clean, modular structure for preparing inputs, launching OpenCL kernels, and post-processing results.

Kernel Runners: A kernel runner is a low-level C++ function that interacts directly with the OpenCL runtime. It is responsible for:

- selecting the OpenCL platform and device,
- creating the context and command queue,
- compiling the combined program source,
- creating device buffers and transferring data,
- setting kernel arguments,
- launching the kernel via `clEnqueueNDRangeKernel()`,
- reading results back to host memory, and
- releasing all OpenCL resources.

Kernel runners contain *no R-specific logic*. They operate entirely on flattened numeric arrays and primitive types. This makes them reusable across many kernels and easy to test in isolation.

In **glmbayes**, the function `f2_f3_kernel_runner()` is the primary example of a kernel runner. It takes fully prepared inputs, executes the OpenCL kernel, and returns raw numeric outputs.

Kernel Wrappers: A kernel wrapper is an R-facing function (typically exported via `[[Rcpp::export]]`) that prepares inputs for the runner and performs any necessary post-processing. A wrapper is responsible for:

- validating R inputs and extracting dimensions,
- flattening matrices and vectors into contiguous arrays,
- selecting the appropriate kernel name and kernel file based on model family and link function,
- assembling the full OpenCL program source by concatenating the core OpenCL header, library sources, and the model-specific kernel,
- invoking the kernel runner with the prepared inputs, and
- reshaping or converting the runner's outputs into R-friendly structures (e.g., `NumericVector`, `arma::mat`).

Kernel wrappers contain all R-level logic and none of the OpenCL plumbing. They provide a stable, user-facing API while delegating GPU execution to the runner.

In **glmbayes**, the function `f2_f3_opencl()` is the kernel wrapper corresponding to `f2_f3_kernel_runner()`. It flattens inputs, selects the correct kernel based on the GLM family and link, assembles the program source, calls the runner, and returns the results as R objects.

Why Separate Runners and Wrappers?: This two-layer design is recommended because it:

- isolates OpenCL resource management from R-level logic,
- makes kernel runners reusable across many models,
- simplifies debugging by separating data preparation from GPU execution,
- allows wrappers to evolve independently of the low-level runner,
- enables consistent program assembly across kernels, and

- keeps exported R functions clean, readable, and easy to maintain.

While users are free to design their own structure, adopting this pattern generally leads to clearer code and more maintainable OpenCL integrations.

Examples

```
############################### Start of load_kernel_source example ###################


if (has_opencl()) {
  src <- load_kernel_source("nmath/bd0.cl")
  lib <- load_kernel_library("nmath")
  nchar(src)
  nchar(lib)
}


##############################################################################
## End of load_kernel_source example
##############################################################################
```

`logLik.glmb` *Extract Log-Likelihood*

Description

This function is a method function for the `"glmb"` class used to Extract the log-Likelihood from a Bayesian Generalized Linear Model.

Usage

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

Arguments

`object`	an object of class `glmb`, typically the result of a call to glmb
`...`	further arguments to or from other methods

Value

The function returns a vector, `logLikout` with the estimated log-likelihood for each draw.

See Also

`glmb`, `glmbayes-package`; `rglmb`, `rlmb`, `lmb`; `logLik`.

Examples

```
## ----dobson----------------------------------------------------------------
## 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())
## Call to glmb
glmb.D93 <- glmb(
  n = 1000,
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma)
)
## ----glmb logLik-----------------------------------------------------------
colMeans(logLik(glmb.D93))
```

`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)
```

pfamily *Prior Family Objects for Bayesian Models*

Description

Prior family objects provide a convenient way to specify the details of the priors used by functions such as glmb. See the documentations for lmb, glmb, glmb, and rglmb for the details of how such model fitting takes place.

Usage

```
pfamily(object, ...)

dNormal(mu, Sigma, dispersion = NULL)

dGamma(
  shape,
  rate,
  beta,
  max_disp_perc = 0.99,
  disp_lower = NULL,
  disp_upper = NULL
)

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
)

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

Arguments

object	the function pfamily accesses the pfamily objects which are stored within objects created by modelling functions (e.g., glmb).
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	the regression coefficients to be assumed when it is not given a prior. Needs to be provided when the Gamma prior is used for the dispersion. This specification is typically only used as part of Gibbs sampling where the beta and dispersion parameters are updated separately.
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
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.
x	an object, a pfamily function that is to be printed
...	additional argument(s) for methods.

Details

pfamily is a generic with methods for fitted objects such as glmb and lmb. Many glmb models currently only implement the dNormal() prior family. The Gamma() response family works with dGamma(); the gaussian() family works with dGamma() and dNormal_Gamma().

A pfamily object represents a structured prior specification for use in Bayesian generalized linear modeling. Each constructor function (e.g., dNormal(), dGamma(), dNormal_Gamma()) 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 glmb() 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(): Defines a gamma prior over a scalar precision parameter, often used in hierarchical models or variance components. This prior is particularly relevant for Gamma likelihoods and dispersion modeling in exponential families (Gelman et al. 2013; Dobson 1990; McCullagh and Nelder 1989). With Gaussian Prior_Setup output, prefer rate_gamma for rate when updating dispersion with fixed beta (see Details above).
- 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).

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", or "dIndependent_Normal_Gamma").
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, max_disp_perc, disp_lower, disp_upper. 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.
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, 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 the glmb function 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

glmb, rlmb, lmb, rglmb for modeling functions that consume pfamily objects.

rNormal_reg, rNormalGamma_reg, rGamma_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 glmb
glmb.D93 <- glmb(
  counts ~ outcome + treatment,
  family  = poisson(),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma)
)

## Extract pfamily and pfamily settings for call to glmb
pfamily(glmb.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

## Conjugate Normal Prior (fixed dispersion)
lmb.D9 <- lmb(
  weight ~ group,
  pfamily = dNormal(mu = ps2$mu, ps2$Sigma, dispersion = ps2$dispersion)
)
pfamily(lmb.D9)
```

```
## Conjugate Normal_Gamma Prior
lmb.D9_v2 <- lmb(
  weight ~ group,
  pfamily = dNormal_Gamma(
    ps2$mu,
    Sigma_0 = ps2$Sigma_0,
    shape = ps2$shape,
    rate  = ps2$rate
  )
)
pfamily(lmb.D9_v2)

## Independent_Normal_Gamma_Prior
lmb.D9_v3 <- lmb(
  weight ~ group,
  dIndependent_Normal_Gamma(
    ps2$mu,
    ps2$Sigma,
    shape = ps2$shape_ING,
    rate  = ps2$rate
  )
)
pfamily(lmb.D9_v3)
```

`plot.glmb` *Plot posterior draws for objects of class* `glmb`.

Description

This converts `x$coefficients` to a `coda::mcmc` object. If `x$dispersion` is a draw vector (length equal to the number of draws), it is appended and plotted too.

Usage

```
## S3 method for class 'glmb'
plot(x, which = c("coefficients", "dispersion", "both"), ...)
```

Arguments

`x`	Object inheriting from class `glmb`.
`which`	One of `"coefficients"`, `"dispersion"`, or `"both"`.
`...`	Forwarded to `coda::plot.mcmc`.

Value

Invisibly `NULL`.

See Also

glmb, glmbayes-package; rglmb, rlmb, lmb; plot.mcmc; plot.lm and termplot for classical diagnostic plots of lm/glm fits.

`predict.glmb` *Predict Method for Bayesian GLM Fits*

Description

Obtains predictions from a fitted Bayesian generalized linear model object. Predictions are computed on the link scale or response scale following the GLM framework (McCullagh and Nelder 1989), with posterior draws used to produce a matrix of predictions.

Usage

```
## S3 method for class 'glmb'
predict(
  object,
  newdata = NULL,
  type = "link",
  se.fit = FALSE,
  dispersion = NULL,
  terms = NULL,
  na.action = na.pass,
  olddata = NULL,
  ...
)
```

Arguments

object	a fitted object of class inheriting from "glmb".
newdata	optionally, a data frame in which to look for variables with which to predict. If omitted, the fitted linear predictors are used.
type	the type of prediction required. The default is on the scale of the linear predictors; the alternative "response" is on the scale of the response variable. Thus for a default binomial model the default predictions are of log-odds (probabilities on logit scale) and type = "response" gives the predicted probabilities. The "terms" option returns a matrix giving the fitted values of each term in the model formula on the linear predictor scale (not implemented).
se.fit	logical switch indicating if standard errors are required (not implemented).
dispersion	the dispersion of the Bayesian GLM fit to be assumed in computing the standard errors. If omitted, that returned by the summary applied to the object is used.
terms	with type="terms" by default all terms are returned. A character vector specifies which terms are to be returned.

na.action	function determining what should be done with missing values in newdata. The default is to predict NA.
olddata	a data frame that should contain all the variables used in the original model specification. Must currently be provided whenever newdata is provided. Both olddata and newdata are subsetted to the model variables extracte from the object model formula and rbind(olddata,newdata) must be valid after this step. A check is also run to verify if the resulting x matrix from olddata is consistent with that from the original model object.
...	further arguments passed to or from other methods.

Details

If newdata is omitted the predictions are based on the data used for the fit. In that case how cases with missing values in the original fit is determined by the na.action argument of that fit. If na.action = na.omit omitted cases will not appear in the residuals, whereas if na.action = na.exclude they will appear (in predictions and standard errors), with residual value NA. See also napredict.

Value

A matrix of predictions where the rows correspond to the draws from the estimated model, and the columns to the observations in the newdata dataset (or the original data if newdata is missing).

Note

Variables are first looked for in newdata and then searched for in the usual way (which will include the environment of the formula used in the fit). A warning will be given if the variables found are not of the same length as those in newdata if it was supplied.

References

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

See Also

simulate.glmb, residuals.glmb, glmb, glmbayes-package; rglmb, rlmb, lmb; predict.glm. See (Nygren 2025) for predictions and model statistics.

Examples

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

## ----Analysis Setup------------------------------------------------------------
## Number of variables in model
Age=menarche$Age
nvars=2
## Reference Ages for setting of priors and Age_Difference
ref_age1=13  # user can modify this
ref_age2=15  ## user can modify this
## Define variables used later in analysis
```

```
Age2=Age-ref_age1
Age_Diff=ref_age2-ref_age1
mu1=as.matrix(c(0,1.098612),ncol=1)
V1<-1*diag(nvars)
V1[1,1]=0.18687882
V1[2,2]=0.10576217
V1[1,2]=-0.03389182
V1[2,1]=-0.03389182
Menarche_Model_Data=data.frame(Age=menarche$Age,Total=menarche$Total,
                               Menarche=menarche$Menarche,Age2)
glmb.out1<-glmb(n=1000,cbind(Menarche, Total-Menarche) ~Age2,family=binomial(logit),
                pfamily=dNormal(mu=mu1,Sigma=V1),data=Menarche_Model_Data)

# Prediction from original model
pred0=predict(glmb.out1,type="link")
colMeans(pred0)

pred1=predict(glmb.out1,type="response")
colMeans(pred1)


## Generate new data
Age_New <- seq(8, 20, 0.25)
Age2_New=Age_New-13
mod_Object=glmb.out1
obs_size=median(menarche$Total) ## Counts for sim from Binomial
olddata=data.frame(Age=menarche$Age,
                   Menarche=menarche$Menarche,Total=menarche$Total,Age2=Age2)

newdata=data.frame(Age=Age_New,Age2=Age2_New)

# Simulate for newdata

pred_menarche=predict(mod_Object,newdata=newdata,olddata=olddata,type="response")
pred_m=colMeans(pred_menarche)

n=nrow(mod_Object$coefficients)
pred_y=matrix(0,nrow=n,ncol=length(Age_New))
for(i in 1:n){
  pred_y[i,1:length(Age_New)]=rbinom(length(Age_New),size=obs_size,
  prob=pred_menarche[i,1:length(Age_New)
                                                                                ])
}

# Produce various predictions
pred_y_m=colMeans(pred_y/obs_size)
quant1_m=apply(pred_menarche,2,FUN=quantile,probs=c(0.025))
quant2_m=apply(pred_menarche,2,FUN=quantile,probs=c(0.975))
quant1_m_y=apply(pred_y/obs_size,2,FUN=quantile,probs=c(0.025))
quant2_m_y=apply(pred_y/obs_size,2,FUN=quantile,probs=c(0.975))

#Plot Predictions for newdata
plot(Menarche/Total ~ Age, data=menarche,
```

```
main="Percentage of girls Who have had their first period")
lines(Age_New, pred_m,lty=1)
lines(Age_New, quant1_m,lty=2)
lines(Age_New, quant2_m,lty=2)
lines(Age_New, quant1_m_y,lty=2)
lines(Age_New, quant2_m_y,lty=2)
```

`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 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).

`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.

`na.action` a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The 'factory-fresh' default is `stats{na.omit}`. Another possible value is `NULL`, no action. Value `stats{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 documentation for `model.offset` at `model.extract`.

`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, glmb; see (Nygren 2025) for prior tailoring; (Nygren 2025) for full derivations.

Other prior: 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 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.

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.

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:

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

The marginal quadratic term is

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

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

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

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

The calibrated dispersion is

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

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.

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.

glmb, lmb for formula-based 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()).

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

Other prior: Prior_Check()

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 glmb
glmb.D93 <- glmb(
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma)
)

## 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

## Conjugate Normal Prior (fixed dispersion)
lmb.D9 <- lmb(
  weight ~ group,
  pfamily = dNormal(mu = ps2$mu, ps2$Sigma, dispersion = ps2$dispersion)
)

## Conjugate Normal_Gamma Prior
lmb.D9_v2 <- lmb(
  weight ~ group,
  pfamily = dNormal_Gamma(
    ps2$mu,
    Sigma_0 = ps2$Sigma_0,
    shape = ps2$shape,
    rate = ps2$rate
  )
)

## Independent_Normal_Gamma_Prior
lmb.D9_v3 <- lmb(
  weight ~ group,
  dIndependent_Normal_Gamma(
    ps2$mu,
    ps2$Sigma,
```

```
    shape = ps2$shape_ING,
    rate = ps2$rate
  )
)

## ------------------------------------------------------------------------
## 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/glmb/lmb, see:
## example("summary.rGamma_reg")

## ------------------------------------------------------------------------
## dGamma prior illustration: Prior_Setup(shape, rate_gamma or rate) + fixed beta
## ------------------------------------------------------------------------
## For gaussian models, Prior_Setup() provides Gamma hyperparameters for the
## precision: tau = 1/dispersion ~ Gamma(shape, rate). The lmb() / rGamma_reg()
## constructors consume these via dGamma(pfamily) and rGamma_reg(prior_list),
## respectively.

## dGamma via pfamily for lmb()
lmb.D9_dGamma <- lmb(
  weight ~ group,
  pfamily = dGamma(shape = ps2$shape, rate = rate_dg, beta = ps2$coefficients)
)

## dGamma via prior_list for rGamma_reg()
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()
)
```

`residuals.glmb` *Accessing Bayesian Generalized Linear Model Fits*

Description

Extract deviance residuals from fitted Bayesian GLM objects. The residuals use the family's deviance residuals function as in `residuals.glm` (McCullagh and Nelder 1989).

Usage

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

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

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

Arguments

object	an object of class glmb, typically the result of a call to glmb
ysim	Optional simulated data for the data y.
...	further arguments to or from other methods

Details

These functions are all methods for class glmb, lmb, or summary.glmb objects.

Value

A matrix DevRes of dimension n times p containing the Deviance residuals for each draw. If ysim is provided, the residuals are based on a comparison to the simulated data instead. The credible intervals for residuals based on simulated data should be a more appropriate measure of whether individual residuals represent outliers or not.

References

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

See Also

predict.glmb, summary.glmb, glmb, glmbayes-package; rglmb, rlmb, lmb; residuals.glm

Examples

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

## ----Analysis Setup------------------------------------------------------------
## Number of variables in model
Age=menarche$Age
nvars=2
## Reference Ages for setting of priors and Age_Difference
ref_age1=13  # user can modify this
ref_age2=15  ## user can modify this
## Define variables used later in analysis
Age2=menarche$Age-ref_age1
Age_Diff=ref_age2-ref_age1
mu1=as.matrix(c(0,1.098612),ncol=1)
```

```
V1<-1*diag(nvars)
V1[1,1]=0.18687882
V1[2,2]=0.10576217
V1[1,2]=-0.03389182
V1[2,1]=-0.03389182
Menarche_Model_Data=data.frame(Age=menarche$Age,Total=menarche$Total,
                               Menarche=menarche$Menarche,Age2)
glmb.out1<-glmb(n=1000,cbind(Menarche, Total-Menarche) ~Age2,family=binomial(logit),
                pfamily=dNormal(mu=mu1,Sigma=V1),data=Menarche_Model_Data)

# Prediction from original model
pred1=predict(glmb.out1,type="response")

## Get Original Residuals, their means, and credible bounds
res_out=residuals(glmb.out1)
colMeans(res_out, na.rm=TRUE)

## Set up to simulate new data and residuals
res_mean=colMeans(res_out, na.rm=TRUE)
res_low1=apply(res_out,2,FUN=quantile,probs=c(0.025),na.rm=TRUE)
res_high1=apply(res_out,2,FUN=quantile,probs=c(0.975),na.rm=TRUE)

## Simulate new data and get residuals for simulated data

ysim1=simulate(glmb.out1,nsim=1,seed=10401L,pred=pred1,family="binomial",
               prior.weights=weights(glmb.out1))


res_ysim_out1=residuals(glmb.out1,ysim=ysim1)
res_low=apply(res_ysim_out1,2,FUN=quantile,probs=c(0.025),na.rm=TRUE)
res_high=apply(res_ysim_out1,2,FUN=quantile,probs=c(0.975),na.rm=TRUE)


oldpar <- par(no.readonly = TRUE)
par(mar = c(5, 4, 4, 2) + 0.1)  # Standard margin setup

# Plot Credible Interval bounds for Deviance Residuals

plot(res_mean~Age,ylim=c(-2.5,2.5),
main="Credible Interval Bound for Menarche - Logit Model Deviance Residuals",
xlab = "Age", ylab = "Avg. Dev. Res")
lines(Age, 0*res_mean,lty=1)
lines(Age, res_low,lty=1)
lines(Age, res_high,lty=1)
lines(Age, res_low1,lty=2)
lines(Age, res_high1,lty=2)

par(oldpar)
```

---

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. 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 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	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 documentation for model.offset at model.extract.
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector.

Gridtype	an optional argument specifying the method used to determine the number of tangent points used to construct the enveloping 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 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.
...	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.

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, response vector, likelihood family, and prior specification. Unlike glmb, which wraps formula parsing, model setup, and method dispatch, 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. Instead, it is called internally by glmb and and may be useful for Gibbs sampling implementations or other workflows where full model reconstruction is unnecessary.

Value

rglmb returns a object of class "rglmb". The function summary (i.e., summary.rglmb) can be used to obtain or print a summary of the results. The generic accessor functions coefficients, fitted.values, residuals, and extractAIC can be used to extract various 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 the glmb function. 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

`glmb` for the formula interface; `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.glmb`, `predict.glmb`, `residuals.glmb`, `simulate.glmb`, `extractAIC.glmb`, `dummy.coef.glmb` and methods(class="glmb") for glmb and the methods and generic functions for classes `glm` and `lm` from which class glmb inherits.

glmbayes Modeling Functions `glmb()`, `lmb()`, `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 (same prior as \code{\link{glmb}} with \code{\link{Prior_Setup}})
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))
)
summary(rglmb.D93)


## Menarche model with informative prior. See \code{\link{glmb}} and \code{\link{Prior_Setup}}
## for default g-priors; for data and fitted curves using \code{predict.glmb}, see the
## menarche block in \code{\link{glmb}} and \code{vignette("Chapter-04", 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)
)
summary(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()
)
summary(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	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 or matrix of extents matching those of the response. One or more offset terms can be included in the formula instead or as well, and if more than one are specified their sum is used. See model.offset.
weights	an optional vector of weights to be used in the fitting process. Should be NULL or a numeric vector. If non-NULL, weighted least squares is used with weights weights (that is, minimizing sum(w*e^2)); otherwise ordinary least squares is used. See also 'Details',
Gridtype	an optional argument specifying the method used to determine the number of tangent points used to construct the enveloping 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 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.
...	For lm(): additional arguments to be passed to the low level regression fitting functions (see below).

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. While rlmb does not return a full model object or support generic methods like predict or summary, it is designed for efficient posterior simulation in Gaussian models where full model reconstruction is unnecessary.

The rlmb function called from within lmb. It is intended for simulation-heavy workflows such as Gibbs sampling or posterior predictive checks where minimal overhead is preferred.

Value

rlmb returns a object of class "rlmb". The function summary (i.e., summary.rglmb) can be used to obtain or print a summary of the results. The generic accessor functions coefficients, fitted.values, residuals, and extractAIC can be used to extract various 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`.

`lmb`, `glmb`, `rglmb` for related interfaces; `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).

`summary.glmb`, `predict.glmb`, `simulate.glmb`, `extractAIC.glmb`, `dummy.coef.glmb` and methods(class="glmb") for methods inherited from class `glmb` and the methods and generic functions for classes `glm` and `lm` from which class `lmb` also inherits.

glmbayes Modeling Functions `glmb()`, `lmb()`, `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"])


## Same model using the lmb function

lmb.D9 <- lmb(n= 1000,weight  ~ group,
 pfamily = dIndependent_Normal_Gamma(ps$mu, ps$Sigma, shape = ps$shape_ING, rate  = ps$rate))

## lmb summary
summary(lmb.D9)


## 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)))
summary(out_rlmb_dGamma)
```

---

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)
```

---

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
glmb.D93 <- glmb(
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = mu, Sigma = V0)
)

## Start setup here [First output from earlier optim optimization]
betastar <- glmb.D93$coef.mode # Posterior mode from optim
x <- glmb.D93$x
y <- glmb.D93$y
# The fitted object does not currently store an offset, so use zeros here.
offset2 <- 0 * y # Should return this from lower level functions
weights2 <- glmb.D93$prior.weights

## Check influence measures for original model
fit <- glmb.wfit(x, y, weights2, offset2, family = poisson(), Bbar = mu, P = solve(V0), betastar)
influence.measures(fit)

print(fit)
print(glmb.D93$coef.mode)

### Now try a strong prior with poorly chosen intercept
mu1 <- 0 * mu
V1 <- 0.1 * V0
glmb2.D93 <- glmb(
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = mu1, Sigma = V1)
)

Bbar2 <- mu1 # Prior mean
betastar2 <- glmb2.D93$coef.mode # Posterior mode from optim
fit2 <- glmb.wfit(x, y, weights2, offset2, family = poisson(), Bbar2, P = solve(V1), betastar2)

influence.measures(fit2)

print(fit2)
print(glmb2.D93$coef.mode)



influence(glmb2.D93)
influence(glmb2.D93, do.coef = TRUE)
influence(glmb2.D93, do.coef = FALSE)

glmb.influence.measures(glmb2.D93, influence(glmb2.D93))


## Do not need method functions
dfbeta(glmb2.D93, influence(glmb2.D93))
```

```
dfbeta(glmb2.D93$fit, influence(glmb2.D93))
hatvalues(glmb2.D93, influence(glmb2.D93))
hatvalues(glmb2.D93$fit, influence(glmb2.D93))

# Implemented methods

rstandard(glmb2.D93, infl = influence(glmb2.D93))
rstandard(glmb2.D93)


# Methods requiring dedicated handling

## Needs a method function
dfbetas(glmb2.D93)
dfbetas(glmb2.D93, influence(glmb2.D93, do.coef = TRUE))
dfbetas(glmb2.D93$fit, influence(glmb2.D93))



# Needs a method function
cooks.distance(glmb2.D93)
cooks.distance(glmb2.D93$fit, influence(glmb2.D93))



# The rstandard method now works directly on glmb objects.


# Needs a method function
rstudent(glmb2.D93)
rstudent(glmb2.D93$fit, influence(glmb2.D93))


# Not a method, separate function
glmb.dffits(glmb2.D93)
dffits(glmb2.D93$fit, influence(glmb2.D93))

# Not a method - separate function
glmb.covratio(glmb2.D93)
covratio(glmb2.D93$fit, influence(glmb2.D93))



# Needs a method function but requires a different approach
# The influence function actually stores this measure
## This seems to require more work to create a method function

infl <- influence(glmb2.D93)
hat2 <- infl$hat
hat2
```

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, ...)
```

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, glmb, lmb, 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).

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))
)
summary(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
)
summary(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
)
summary(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
```

```
)
summary(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
)
summary(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)

summary(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)
summary(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)
```

simulate.glmb *Simulate Responses*

Description

Simulate responses from the posterior predictive distribution corresponding to a fitted glmb object (Gelman et al. 2013).

Usage

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

Arguments

object	An object of class glmb, typically the result of a call to the function glmb.
nsim	Defunct (see below).
seed	an object specifying if and how the random number generator should be initialized (seeded).
...	Additional arguments passed to the function. Will frequently include a matrix pred of simulated predictions from the predict function, the family (e.g., binomial) and an optional vector of weights specifying prior.weights for the simulated values (default is 1)

Value

Simulated values for data corresponding to simulated model predictions that correspond either to the original data or to a newdata data frame provided to the predict function.

References

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

See Also

predict.glmb, glmb, glmbayes-package; rglmb, rlmb, lmb; simulate (e.g. simulate.glm, simulate.lm for classical fits); see (Nygren 2025) for model statistics.

Examples

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

## ----Analysis Setup------------------------------------------------------------
## Number of variables in model
Age=menarche$Age
nvars=2
```

```
## Reference Ages for setting of priors and Age_Difference
ref_age1=13  # user can modify this
ref_age2=15  ## user can modify this
## Define variables used later in analysis
Age2=menarche$Age-ref_age1
Age_Diff=ref_age2-ref_age1
mu1=as.matrix(c(0,1.098612),ncol=1)
V1<-1*diag(nvars)
V1[1,1]=0.18687882
V1[2,2]=0.10576217
V1[1,2]=-0.03389182
V1[2,1]=-0.03389182
Menarche_Model_Data=data.frame(Age=menarche$Age,Total=menarche$Total,
                               Menarche=menarche$Menarche,Age2)
glmb.out1<-glmb(n=1000,cbind(Menarche, Total-Menarche) ~Age2,family=binomial(logit),
                pfamily=dNormal(mu=mu1,Sigma=V1),data=Menarche_Model_Data)

# Prediction from original model
pred1=predict(glmb.out1,type="response")

## Get Original Residuals, their means, and credible bounds
res_out=residuals(glmb.out1)
colMeans(res_out, na.rm=TRUE)

## Set up to simulate new data and residuals
res_mean=colMeans(res_out, na.rm=TRUE)
res_low1=apply(res_out,2,FUN=quantile,probs=c(0.025),na.rm=TRUE)
res_high1=apply(res_out,2,FUN=quantile,probs=c(0.975),na.rm=TRUE)

## Simulate new data and get residuals for simulated data

ysim1=simulate(glmb.out1,nsim=1,seed=10401L,pred=pred1,family="binomial",
               prior.weights=weights(glmb.out1))


res_ysim_out1=residuals(glmb.out1,ysim=ysim1)
res_low=apply(res_ysim_out1,2,FUN=quantile,probs=c(0.025),na.rm=TRUE)
res_high=apply(res_ysim_out1,2,FUN=quantile,probs=c(0.975),na.rm=TRUE)


oldpar <- par(no.readonly = TRUE)
par(mar = c(5, 4, 4, 2) + 0.1)  # Standard margin setup

# Plot Credible Interval bounds for Deviance Residuals

plot(res_mean~Age,ylim=c(-2.5,2.5),
main="Credible Interval Bound for Menarche - Logit Model Deviance Residuals",
xlab = "Age", ylab = "Avg. Dev. Res")
lines(Age, 0*res_mean,lty=1)
lines(Age, res_low,lty=1)
lines(Age, res_high,lty=1)
lines(Age, res_low1,lty=2)
lines(Age, res_high1,lty=2)
```

```
par(oldpar)
```

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.glmb` *Summarizing Bayesian Generalized Linear Model Fits*

Description

These functions are all `methods` for class `glmb` or `summary.glmb` objects.

Usage

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

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

Arguments

`object`	an object of class `"glmb"` for which a summary is desired.
`x`	an object of class `"summary.glmb"` for which a printed output is desired.
`digits`	the number of significant digits to use when printing.
`...`	Additional optional arguments

Details

The summary.glmb function summarizes the output from the glmb function. Key output includes mean residuals, information related to the prior, mean coefficients with associated stats, percentiles for the coefficients, as well as the effective number of parameters and the DIC statistic. The dir_tail component reports the directional tail diagnostic; see directional_tail and (Nygren 2025) for interpretation.

Value

summary.glmb returns a object of class "summary.glmb", a list with components:

call	the component from object
n	number of draws generated
residuals	vector of mean deviance residuals
coefficients1	Matrix with the prior mean and maximum likelihood coefficients with associated standard deviations
coefficients	Matrix with columns for the posterior mode, posterior mean, posterior standard deviation, monte carlo error, and tail probabilities (posterior probability of observing a value for the coefficient as extreme as the prior mean)
dir_tail	List containing information related to the directional tail relative to the Prior
dir_tail_null	List containing information related to the directional tail relative to the Null Model
Percentiles	Matrix with estimated percentiles associated with the posterior density
pD	Estimated effective number of parameters
deviance	Vector with draws for the deviance
DIC	Estimated DIC statistic
iters	Average number of candidates per generated draws

References

Nygren K (2025). "Chapter A04: Directional Tail Diagnostics for Prior-Posterior Disagreement." Vignette in the glmbayes R package. R vignette name: Chapter-A04.

See Also

directional_tail, glmb, glmbayes-package, lmb, rglmb, rlmb, summary, summary.lm, summary.glm

Examples

```
############################  Example for lmb function

## 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, family = gaussian())

lmb.D9 <- lmb(
  weight ~ group,
  pfamily = dNormal_Gamma(
    ps$mu,
    Sigma_0 = ps$Sigma_0,
    shape = ps$shape,
    rate  = ps$rate
  )
)
summary(lmb.D9)

##########################  Example for glmb function

## 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))

ps <- Prior_Setup(counts ~ outcome + treatment, family = poisson(), data = d.AD)

glmb.D93 <- glmb(
  counts ~ outcome + treatment,
  family  = poisson(),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma)
)
summary(glmb.D93)

## Menarche logit model with default (non-informative) Prior_Setup prior
data(menarche, package = "MASS")
Age2 <- menarche$Age - 13

ps_m <- Prior_Setup(
  cbind(Menarche, Total - Menarche) ~ Age2,
  family = binomial(logit),
  data = menarche
)

glmb.out1 <- glmb(
  cbind(Menarche, Total - Menarche) ~ Age2,
  family  = binomial(logit),
  pfamily = dNormal(mu = ps_m$mu, Sigma = ps_m$Sigma),
  data    = menarche
)
summary(glmb.out1)

## Posterior mean fitted probabilities on response scale
require(graphics)
pred1 <- predict(glmb.out1, type = "response")
```

```
pred1_m <- colMeans(pred1)
plot(
  Menarche / Total ~ Age,
  data = menarche,
  main = "Proportion with menarche (data and posterior mean fit)"
)
lines(menarche$Age, pred1_m, col = "blue", lwd = 2)
```

---

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 five functions (rGamma_reg, rglmb, rlmb, glmb, lmb) 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)

## 4. glmb
out4 <- glmb(form, data = Boston_centered, family = gaussian(),
 pfamily = dGamma(shape = ps.boston$shape, rate = rate_dg, beta = ps.boston$coefficients))
summary(out4)
```

```
## 5. lmb
out5 <- lmb(form, data = Boston_centered,
  pfamily = dGamma(shape = ps.boston$shape, rate = rate_dg, beta = ps.boston$coefficients))
summary(out5)
```

---

summary.rglmb *Summarizing Bayesian Generalized Linear Model Distribution Functions*

Description

These functions are all methods for class rglmb or summary.rglmb objects.

Usage

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

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

Arguments

object	an object of class "rglmb" 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

The summary.rglmb function summarizes the output from the rglmb function. It takes an object of class rglmb as an input. The output to a large extent mirrors the output from the summary.glm function. This is particularly true for the print output from the function (i.e. the output of the function print.summary.rglmb).

Value

summary.rglmb returns a object of class "summary.rglmb", a list with components:

n	number of draws generated
coefficients1	Matrix with the prior mean and approximate weight for the prior relative to the data
coefficients	Matrix with columns for the posterior mode, posterior mean, posterior standard deviation, monte carlo error, and tail probabilities (posterior probability of observing a value for the coefficient as extreme as the prior mean)
Percentiles	Matrix with estimated percentiles associated with the posterior density

See Also

rglmb; glmb, glmbayes-package, rlmb, lmb, summary, summary.lm, summary.glm.

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))

# 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

out<-rglmb(n = 1000, y=y, x=x, pfamily=dNormal(mu=mu,Sigma=V1), weights = wt,
           family = binomial(logit))
summary(out)
```

---

vcov.glmb *Calculate Variance-Covariance Matrix for a Fitted Model Object*

Description

Returns the posterior variance-covariance matrix of the regression coefficients from a fitted Bayesian GLM object (Gelman et al. 2013).

Usage

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

Arguments

object	fitted model object, typically the result of a call to glmb.
...	additional arguments for method functions.

Value

A matrix of estimated covariances between the parameter estimates in the linear or non-linear predictor of the model. This should have row and column names corresponding to the parameter names given by the coef method.

References

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

See Also

confint.glmb, summary.glmb, glmb, glmbayes-package; rglmb, rlmb, lmb; vcov

Examples

```
## ----dobson----------------------------------------------------------------
## 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())
## Call to glmb
glmb.D93 <- glmb(
  n = 1000,
  counts ~ outcome + treatment,
  family = poisson(),
  pfamily = dNormal(mu = ps$mu, Sigma = ps$Sigma)
)

## ----glmb vcov-------------------------------------------------------------
vcov(glmb.D93)
```

Index