Package ‘bml’

February 20, 2026

Type Package

Title Bayesian Multiple-Membership Multilevel Models with Parameterizable Weight Functions

Version 0.9.0

Description

Implements Bayesian multiple-membership multilevel models with parameterizable weight functions via 'JAGS' to model how lower-level units jointly shape higher-level outcomes (micro-macro link) across a range of outcome types (e.g., linear, logit, and survival models). Supports estimation and comparison of alternative aggregation mechanisms, allows weight matrices to be endogenized through parameters and covariates, and accommodates complex dependence structures that extend beyond traditional multilevel frameworks. For details, see Rosche (2026) ``A Multilevel Model for Coalition Governments. Uncovering Party-Level Dependencies Within and Between Governments'' <doi:10.31235/osf.io/4bafr_v2>.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports dplyr, readr, tidyr, stringr, R2jags, ggplot2, ggmcmc, coda, patchwork, tibble, purrr, rlang

URL `https://benrosche.github.io/bml/`

BugReports `https://github.com/benrosche/bml/issues`

RoxygenNote 7.3.3

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

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Benjamin Rosche [aut, cre] (ORCID: <`https://orcid.org/0000-0001-5196-625X`>)

Maintainer Benjamin Rosche <`benrosche@nyu.edu`>

Repository CRAN

Date/Publication 2026-02-20 08:00:02 UTC

Contents

`bml` *Bayesian Multiple-Membership Multilevel Models with Parameterizable Weight Functions Using JAGS*

Description

The **bml** package provides a user-friendly interface for fitting Bayesian multiple-membership multilevel models with parameterizable weight functions via JAGS.

JAGS must be installed separately: `https://sourceforge.net/projects/mcmc-jags/`.

Usage

```
bml(
  formula,
  family = "Gaussian",
  priors = NULL,
  inits = NULL,
  n.iter = 1000,
  n.burnin = 500,
  n.thin = max(1, floor((n.iter - n.burnin)/1000)),
  n.chains = 3,
  seed = NULL,
  run = TRUE,
  parallel = FALSE,
  monitor = TRUE,
  modelfile = FALSE,
  cox_intervals = NULL,
  data = NULL
)
```

Arguments

`formula`	A symbolic model formula. See 'Formula Components' section for details. The general structure is: `outcome ~ 1 + predictors + mm(...) + hm(...)`. For survival models, use `Surv(time, event)` on the left-hand side.
`family`	Character string specifying the outcome distribution and link function. Options: • `"Gaussian"`: Normal distribution with identity link (continuous outcomes) • `"Binomial"`: Binomial distribution with logit link (binary outcomes) • `"Weibull"`: Weibull survival model (requires `Surv(time, event)` outcome) • `"Cox"`: Cox proportional hazards model (requires `Surv(time, event)` outcome)
`priors`	Named list or character vector of JAGS prior specifications. Parameter names follow JAGS naming conventions: • **Main level:** `b[x]` for main-equation coefficients (e.g., `"b[1] ~ dnorm(0, 0.01)"` for the intercept) • **HM level:** `b.hm.k[x]` for hm block `k` coefficients, `tau.hm.k` for hm block `k` random effect precision • **MM level:** `b.mm.k[x]` for mm block `k` coefficients, `b.w.k[x]` for weight function parameters in mm block `k`, `tau.mm.g` for mm random effect precision (indexed by member-group ID group `g`) • **Other:** `shape` (Weibull shape parameter), `lambda0[k]` (Cox baseline hazard intervals) **Note:** Priors on variance components must be specified on the *precision* scale (`tau = 1/sigma^2`), not the standard deviation, since JAGS parameterizes normal distributions using precision. Example: `list("b.mm.1 ~ dnorm(0, 0.01)", "tau.mm.1 ~ dgamma(2, 0.1)")`. Default priors are weakly informative.
`inits`	List of initial values for MCMC chains. Applied to all chains. If `NULL`, JAGS generates initial values automatically. Weight function parameters (`b.w.k`) are always initialized at 0 by default to prevent numerical instability (e.g., `ilogit` with extreme inputs). User-supplied inits override these defaults.
`n.iter`	Total number of MCMC iterations per chain. Default: 1000. Increase for better convergence (e.g., 10000-50000 for production models).
`n.burnin`	Number of burn-in iterations to discard at the start of each chain. Default: 500. Should be sufficient for chains to reach stationarity.
`n.thin`	Thinning rate: save every k-th iteration to reduce autocorrelation. Default: `max(1, floor((n.iter - n.burnin) / 1000))` (targets ~1000 samples). Increase if posterior samples show high autocorrelation.
`n.chains`	Number of MCMC chains. Default: 3. Use 3-4 chains to assess convergence via Gelman-Rubin diagnostics.
`seed`	Integer random seed for reproducibility. If `NULL`, results will vary across runs.
`run`	Logical; if `TRUE` (default), JAGS is executed and the model is fitted. If `FALSE`, returns the model specification without fitting (useful for inspecting generated JAGS code or data structures).
`parallel`	Logical; if `TRUE`, run MCMC chains in parallel using multiple cores. Requires parallel backend setup. Default: `FALSE`.

`monitor`	Logical; if `TRUE`, store full MCMC chains and additional outputs for diagnostic plots. Required for `monetPlot` and `mcmcDiag`. Default: `TRUE`.
`modelfile`	Logical or character path: • `FALSE` (default): JAGS code generated internally • `TRUE`: Save generated JAGS code to `modelstring.txt` in working directory • Character path: Read JAGS code from specified file instead of generating
`cox_intervals`	For Cox models only: controls baseline hazard flexibility and computational efficiency. • `NULL` (default): Non-parametric baseline hazard using all unique event times (maximum flexibility, slower for large datasets) • Integer k: Piecewise constant baseline hazard with k intervals (faster, suitable for datasets with many unique event times). Recommended: k = 10-20 for most applications.
`data`	Data frame in member-level (long) format where each row represents a member-level observation. Must contain all variables referenced in the formula, including identifiers specified in `id()`.

Details

In addition to hierarchical and cross-classified multilevel models, the **bml** package allows users to fit Bayesian multiple-membership models. Unlike tools such as `brms` or MLwiN, **bml** lets users specify and estimate models in which membership weights are parameterized through flexible formula syntax. This enables a more nuanced examination of how effects from member-level units aggregate to group level (the micro-macro link).

The package automatically generates JAGS code to fit the model and processes the output to facilitate interpretation of model parameters and diagnostics.

The package and modeling framework are introduced in: Rosche, B. (2026). *A Multilevel Model for Coalition Governments: Uncovering Party-Level Dependencies Within and Between Governments. Political Analysis*.

For accessible introductions to multiple-membership models, see Fielding and Goldstein (2006) and Beretvas (2010). Advanced treatments include Goldstein (2011, Ch. 13), Rasbash and Browne (2001, 2008), Browne et al. (2001), and Leckie (2013).

Value

A list of class `"bml"` containing:

- `reg.table`: Data frame of posterior summaries with columns `Parameter`, `mean`, `sd`, `lb`, `ub` (95% credible interval bounds).
- `w`: List of weight matrices (one per `mm()` block). Each matrix has rows = groups and columns = members within each group.
- `re.mm`: List of member-level random effects (one per mmid group with `RE = TRUE`). Vector for standard RE, matrix for autoregressive RE.
- `re.hm`: List of nesting-level random effects (one per `hm()` block with `type = "RE"`).

- `pred`: Vector of predicted values (posterior means) for each group.
- `input`: List of model specifications including `family`, `mm` and `hm` block info, sample sizes, and MCMC settings.
- `jags.out`: Full R2jags output object (if `monitor = TRUE`; `NULL` otherwise). Contains posterior samples, MCMC chains, and convergence diagnostics.

Formula Components

- **Outcome (Y):** The dependent variable. For survival models, use `Surv(time, event)`.
- **Intercept:** Follows standard R formula conventions (like `lm()`):
  - `y ~ x`: Includes intercept by default
  - `y ~ 1 + x`: Explicitly includes intercept (same as default)
  - `y ~ 0 + x` or `y ~ -1 + x`: Excludes intercept
- **Main-level predictors (X.main):** Variables defined at the main (group) level, separated by `+`.
- **HM-level predictors (X.hm):** Variables defined at the nesting level, separated by `+`.
- **Multiple membership object (`mm()`):** Defines how member-level units are associated with group-level constructs using a user-specified weighting function. Multiple `mm()` objects can be specified with different weight functions.
- **Hierarchical membership (`hm()`):** Specifies nesting of main-level units within higher-level entities. Cross-classified structures can be modeled by including multiple `hm()` objects.

Formula Features: The main formula and `vars()` specifications support standard R formula syntax:

- **Interactions:** Use `*` for main effects plus interaction, or `:` for interaction only. Example: `y ~ a * b` expands to `y ~ a + b + a:b`.
- **Transformations:** Use `I()` for arithmetic operations. Example: `y ~ I(x^2)` or `y ~ I(a + b)`.

These features work in:

- **Main formula:** `y ~ 1 + a * b + I(x^2)`
- **mm() vars:** `vars(a * b)` or `vars(I(x^2))`
- **hm() vars:** `vars(a:b)` or `vars(I(log(x)))`

Note on weight functions: The `fn()` weight function in `mm()` does NOT support interactions or `I()` transformations. Users must pre-create any needed transformed variables in their data before using them in weight functions. For example, instead of `fn(w ~ b1 * x^2)`, first create `data$x_sq <- data$x^2` and use `fn(w ~ b1 * x_sq)`.

Note on intercepts: Intercept syntax (`1`, `0`, `-1`) only applies to the main formula. Numeric literals in `vars()` are ignored (e.g., `vars(1 + x)` is equivalent to `vars(x)`).

Multiple Membership Object `mm()`

```
mm(
  id   = id(mmid, mainid),
  vars = vars(X.mm),
  fn   = fn(w ~ 1/n, c = TRUE),
  RE   = TRUE,
  ar   = FALSE
)
```

Components:

- `id(mmid, mainid)`: Specifies identifiers linking each member-level unit (`mmid`) to its corresponding group-level entities (`mainid`).
- `vars(X.mm)`: Specifies member-level covariates aggregated across memberships. Use `+` to include multiple variables. Supports interactions (`*`, `:`) and transformations (`I()`). Set to `NULL` for RE-only blocks.
- `fn(w ~ ..., c)`: Defines the weight function (micro-macro link). The `c` parameter controls weight normalization: when `c = TRUE` (default), weights are normalized to sum to 1 within each group ($\tilde{w}_{ik} = w_{ik}/\sum_k w_{ik}$). Set `c = FALSE` for unnormalized weights (e.g., when aggregating sums). Note: Does not support interactions or `I()` - pre-create transformed variables.
- `RE`: Logical; if `TRUE`, include random effects for this block. Automatically `TRUE` if `vars = NULL`.
- `ar`: Logical; if `TRUE`, member-level random effects evolve as a random walk across repeated participations in groups. This captures dynamics where a member's unobserved heterogeneity changes over time. Default: `FALSE`.

Multiple mm() blocks: You can specify multiple `mm()` blocks with different weight functions. However, `RE = TRUE` can only be specified for one `mm()` block.

```
mm(id = id(pid, gid), vars = vars(X.mm.1), fn = fn(w ~ 1/n), RE = FALSE) +
mm(id = id(pid, gid), vars = vars(X.mm.2), fn = fn(w ~ pseat == max(pseat)), RE = FALSE) +
mm(id = id(pid, gid), vars = NULL, fn = fn(w ~ 1/n), RE = TRUE)
```

Hierarchical Membership Object `hm()`

```
hm(id = id(hmid), vars = vars(X.hm), name = hmname, type = "RE", showFE = FALSE)
```

Components:

- `id = id(hmid)`: Variable identifying nesting-level groups.
- `vars = vars(X.hm)`: Nesting-level variables, or `NULL`. Supports interactions (`*`, `:`) and transformations (`I()`).
- `name = hmname`: Optional labels for nesting-level units.
- `type`: `"RE"` (default) or `"FE"`.
- `showFE`: If `TRUE` and `type = "FE"`, report the fixed effects.

Supported Families / Links

- Gaussian (continuous): `family = "Gaussian"`
- Binomial (logistic): `family = "Binomial"`
- Weibull survival: `family = "Weibull"`, outcome: `Surv(time, event)`
- Cox survival: `family = "Cox"`, outcome: `Surv(time, event)`

Priors

Priors can be specified for parameters. With multiple mm() blocks, use indexed names:

```
priors = list(
  "b.mm.1 ~ dnorm(0, 0.01)",
  "b.w.1 ~ dnorm(0, 0.1)",
  "tau.mm.1 ~ dscaled.gamma(25, 1)"
)
```

Author(s)

Benjamin Rosche <benrosche@nyu.edu>

References

Rosche, B. (2026). A Multilevel Model for Coalition Governments: Uncovering Party-Level Dependencies Within and Between Governments. *Political Analysis*.

Browne, W. J., Goldstein, H., & Rasbash, J. (2001). Multiple membership multiple classification (MMMC) models. *Statistical Modelling*, 1(2), 103-124.

See Also

summary.bml for model summaries, monetPlot for posterior visualization, mcmcDiag for convergence diagnostics, mm, hm for model specification helpers

Examples

```
data(coalgov)

# Basic multiple-membership model
# Parties (pid) within governments (gid), nested in countries (cid)
m1 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + majority +
    mm(
      id   = id(pid, gid),
      vars = vars(cohesion),
      fn   = fn(w ~ 1/n, c = TRUE),
      RE   = TRUE
    ) +
    hm(id = id(cid), type = "RE"),
  family = "Weibull",
  data   = coalgov
)

# View results
summary(m1)
monetPlot(m1, "b[2]")  # Plot for majority coefficient

# Multiple mm() blocks with different weight functions
m2 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + majority +
    mm(id = id(pid, gid), vars = vars(cohesion),
       fn = fn(w ~ cohesion == max(cohesion)), RE = FALSE) +
    mm(id = id(pid, gid), vars = NULL, fn = fn(w ~ 1/n), RE = TRUE),
  family = "Weibull",
  data   = coalgov
)

# Cox model with piecewise baseline hazard (faster for large datasets)
m3 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + majority +
    mm(id = id(pid, gid), vars = vars(cohesion), fn = fn(w ~ 1/n), RE = TRUE),
  family = "Cox",
  cox_intervals = 10,  # Use 10 intervals instead of all unique times
  data   = coalgov
)

# Parameterized weight function
# ilogit() bounds raw weights between 0 and 1; c = TRUE normalizes to sum to 1
m4 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + majority +
    mm(
      id   = id(pid, gid),
      vars = vars(cohesion),
      fn   = fn(w ~ ilogit(b0 + b1 * pseat), c = TRUE),
      RE   = FALSE
    ),
  family = "Weibull",
  data   = coalgov
)

# Fixed coefficients (offsets)
m5 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + fix(majority, 1) + # Fix majority coefficient to 1.0
    mm(
      id   = id(pid, gid),
      vars = vars(rile),
      fn   = fn(w ~ 1/n, c = TRUE),
      RE   = FALSE
    ),
  family = "Weibull",
  data   = coalgov
)

# Custom priors
m6 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + majority +
    mm(id = id(pid, gid), vars = vars(cohesion), fn = fn(w ~ 1/n), RE = TRUE),
  family = "Weibull",
  priors = list(
    "b[1] ~ dnorm(0, 0.01)",       # Intercept prior
    "b.mm.1 ~ dnorm(0, 0.1)",      # MM coefficient prior
    "tau.mm.1 ~ dgamma(2, 0.5)"    # MM precision prior
  ),
  data   = coalgov
)

# Cross-classified model (multiple hm blocks)
# Governments are cross-classified by country and election year
m7 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + majority +
    mm(id = id(pid, gid), vars = vars(cohesion), fn = fn(w ~ 1/n), RE = TRUE) +
    hm(id = id(cid), type = "RE") +
    hm(id = id(year), type = "RE"),
  family = "Weibull",
  data   = coalgov |> mutate(year = format(election, "%Y") |> as.integer())
)
```

`coalgov` *Coalition Governments in Western Democracies (1944-2014)*

Description

A dataset containing information on coalition governments and their member parties across 30 parliamentary democracies. The data are in long format where the unit of analysis is parties in governments, making it suitable for multiple-membership multilevel models where governments (groups) are composed of multiple parties (members).

Usage

`coalgov`

Format

A tibble with 2,077 rows and 18 variables. Each row represents a party's participation in a specific coalition government. The sample contains 628 governments formed by 312 unique parties across 29 countries.

Identifiers:

gid Government identifier (group-level unit in `mm()` specification). Range: [3, 1105]

pid Party identifier (member-level unit in `mm()` specification). Range: [11110, 96955]

cid Country identifier (nesting-level unit in `hm()` specification). Range: [11, 96]

cname Three-letter country code (ISO 3166-1 alpha-3)

pname Full party name

Government-level variables:

election Date of the preceding election that led to the government's formation. Range: [1939-04-02, 2014-12-14]

n Number of parties in the coalition (group size for weight functions). Range: [2, 9], mean: 3.31

dur_wkb Government duration in days, measured from investiture to termination (outcome variable for survival models). Range: [7, 1840], mean: 554.5

event_wkb Early termination indicator: 1 = government terminated due to political conflict (voluntary resignation, dissension within government, lack of parliamentary support, or head of state intervention) more than one year before the official end of term; 0 = censored (regular elections, other reasons, or termination within one year of scheduled elections). Range: [0, 1], mean: 0.39

majority Majority government indicator: 1 = coalition controls majority of parliamentary seats, 0 = minority government. Range: [0, 1], mean: 0.80

mwc Minimal winning coalition indicator: 1 = coalition would lose its majority if any party left, 0 = oversized coalition. Range: [0, 1], mean: 0.35

rile_SD Inter-party ideological heterogeneity. Standard deviation of coalition parties' left-right positions (from CMP) relative to the ideological distribution of all parties in parliament. Standardized and inverted so higher values indicate greater ideological cohesion. Range: [-8.40, 2.12], mean: 0.04

Country-level variables:

investiture Investiture vote requirement (time-constant country characteristic): 1 = country requires formal parliamentary investiture vote, 0 = no formal requirement. Range: [0, 1], mean: 0.46

Party-level variables:

pseat Party's relative seat share within the coalition, computed as `pseat / sum(pseat)` within each government. Sums to 1 within each coalition. Range: [0.00, 1.00], mean: 0.33

prime Prime minister party indicator: `TRUE` = party holds prime ministership (n = 628), `FALSE` = junior coalition partner (n = 1,449)

cohesion Intra-party ideological cohesion, measured using an adaptation of the Cowles-Jones ratio. Computed as the ratio of continuous ideological shifts to reversals in a party's left-right position over time. Higher values indicate more consistent ideological trajectories (greater cohesion). Standardized. Range: [-1.13, 3.85], mean: 0.00

rile Party's left-right ideological position (from CMP). Measured on a continuous scale where higher values indicate more right-wing positions and lower values indicate more left-wing positions. Standardized. Range: [-3.21, 3.68], mean: 0.00

finance Party's economic dependence on member contributions (from PPDB). Measured as the share of party funding from member dues relative to total income. Standardized; higher values indicate greater dependence on member financing. Treated as time-constant due to data limitations. Range: [-0.98, 4.40], mean: 0.00

Nmembers Number of party members (from PPDB). Standardized; treated as time-constant due to data limitations. Range: [-0.33, 15.02], mean: 0.00

Details

This dataset demonstrates multiple-membership multilevel modeling where:

- **Members:** Political parties (identified by `pid`)
- **Groups:** Coalition governments (identified by `gid`)

- **Nesting:** Governments nested within countries (identified by `cid`)

Each coalition government comprises multiple parties, and parties can participate in multiple governments over time. This creates a multiple-membership structure where party-level characteristics are aggregated to the government level using weighting functions specified in `mm()` blocks.

Sample: After matching party data across sources and excluding single-party and caretaker governments, the sample comprises 628 governments formed by 312 unique parties across 29 countries: Australia, Austria, Belgium, Bulgaria, Croatia, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Ireland, Israel, Italy, Japan, Latvia, Lithuania, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Spain, Sweden, Switzerland, and United Kingdom.

Measurement notes:

- Government duration follows the WKB convention: time from investiture to termination or new elections
- Early termination events focus on political gridlock (conflict-related endings) and exclude terminations within one year of scheduled elections
- Party-level variables (`cohesion`, `finance`, `Nmembers`) are standardized (mean = 0) for analysis

Source

Data compiled from multiple sources:

- **Coalition governments:** Woldendorp, Keman, and Budge (WKB) dataset, updated by Seki and Williams (2014)
- **Party ideology:** Comparative Manifesto Project (CMP; Volkens et al. 2016)
- **Party organization:** Political Party Database (PPDB; Scarrow, Poguntke, and Webb 2017)

Missing party-level data imputed using multiple imputation by chained equations with predictive mean matching.

References

Seki, K., & Williams, L. K. (2014). Updating the Party Government data set. *Electoral Studies*, 34, 270-279.

Volkens, A., et al. (2016). The Manifesto Data Collection. Manifesto Project (MRG/CMP/MARPOR). Version 2016a. Berlin: Wissenschaftszentrum Berlin fur Sozialforschung.

Scarrow, S. E., Webb, P. D., & Poguntke, T. (Eds.). (2017). *Organizing Political Parties: Representation, Participation, and Power*. Oxford University Press.

See Also

`bml` for modeling examples using this dataset

Examples

```
data(coalgov)

# Explore data structure
str(coalgov)
table(coalgov$cname)

# Number of unique units
length(unique(coalgov$gid))   # Governments
length(unique(coalgov$pid))   # Parties
length(unique(coalgov$cid))   # Countries


# Model: government duration as function of majority status and party characteristics
m1 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + majority +
    mm(id = id(pid, gid), vars = vars(finance), fn = fn(w ~ 1/n), RE = TRUE) +
    hm(id = id(cid), type = "RE"),
  family = "Weibull",
  data = coalgov
)
summary(m1)
```

`fix` *Fix a coefficient to a known value*

Description

Specify a covariate whose coefficient should be held constant at a fixed value rather than estimated from the data. This is useful for offset variables or when you want to impose theoretical constraints. Fixed coefficients are handled efficiently by pre-computing their contribution in R before passing data to JAGS.

Usage

```
fix(var, value)
```

Arguments

`var`	Unquoted variable name from your data
`value`	Numeric value for the coefficient (e.g., `1.0` for a standard offset)

Value

A `bml_fix` object that can be used within `vars`.

See Also

`vars`, `mm`, `hm`

Examples

```
# Fix a coefficient to 1.0 (standard offset)
fix(exposure, 1.0)

# Use within vars() for multiple-membership models
vars(fix(population, 0.5) + income + education)
```

fn — *Specify a weight function for multiple-membership models*

Description

Defines how member-level contributions are weighted when aggregating to the group level (the "micro-macro link"). The weight function can be a simple formula (e.g., `1/n` for equal weights) or can include parameters to be estimated from the data.

Usage

```
fn(w = w ~ 1/n, c = TRUE)
```

Arguments

`w` A two-sided formula specifying the weight function. The left-hand side must be `w`; the right-hand side defines the weighting scheme:

- Simple: `w ~ 1/n` (equal weights based on group size)
- Parameterized: `w ~ b0 + b1 * tenure` (weights depend on member characteristics and estimated parameters)
- With group aggregates: `w ~ b1 * min(x) + (1-b1) * mean(x)` (weights based on group-level summaries; see Details)

Parameters must be named `b0`, `b1`, `b2`, etc.

`c` Logical; if `TRUE` (default), weights are normalized to sum to 1 within each group. Set to `FALSE` for unnormalized weights.

Details

Weight Function Components:

- **Variables** (e.g., `n`, `tenure`): Data from your dataset
- **Parameters** (e.g., `b0`, `b1`): Estimated from the data
- **Operations**: Standard R arithmetic (`+`, `-`, `*`, `/`, `^`, etc.)

Common Weight Functions:

- Equal weights: `w ~ 1/n`
- Duration-based: `w ~ duration`
- Flexible parameterized: `w ~ b0 + b1 * seniority`
- Group aggregates: `w ~ b1 * min(x) + (1-b1) * mean(x)`

When `c = TRUE`, the weights are constrained: $\sum_{k \in group} w_k = 1$.

Group-Level Aggregation Functions:

The weight function supports aggregation functions that compute summaries within each group (mainid). These are pre-computed in R before passing to JAGS. Supported functions:

- `min(var)`, `max(var)`: Minimum/maximum value within the group
- `mean(var)`, `sum(var)`: Mean/sum of values within the group
- `median(var)`, `mode(var)`: Median/mode (most frequent) value within the group
- `sd(var)`, `var(var)`, `range(var)`: Standard deviation/variance/range (max-min) within the group
- `first(var)`, `last(var)`: First/last value (based on data order)
- `quantile(var, prob)`: Quantile at probability `prob` (0 to 1). For example, `quantile(x, 0.25)` computes the 25th percentile.

Example: `fn(w ~ b1 * min(tenure) + (1-b1) * max(tenure))` creates weights that blend the minimum and maximum tenure within each group, with the blend controlled by the estimated parameter `b1`.

Example with quantile: `fn(w ~ quantile(tenure, 0.75) / max(tenure))` uses the 75th percentile relative to the maximum within each group.

Note: Nested aggregation functions (e.g., `min(max(x))`) are not supported.

JAGS Mathematical Functions:

The following mathematical functions are passed directly to JAGS and can be used in weight formulas:

- `exp`, `log`, `log10`, `sqrt`, `abs`, `pow`
- `sin`, `cos`, `tan`, `asin`, `acos`, `atan`
- `sinh`, `cosh`, `tanh`
- `logit`, `ilogit`, `probit`, `iprobit`, `cloglog`, `icloglog`
- `round`, `trunc`, `floor`, `ceiling`

Example: `fn(w ~ 1 / (1 + (n - 1) * exp(-(b1 * x))))` uses an exponential decay function where weights depend on member characteristics.

Ensuring Numerical Stability:

Weight functions with estimated parameters (`b0`, `b1`, ...) must produce bounded, positive values across all plausible parameter values. Unbounded weight functions can cause the MCMC sampler to crash (e.g., `"Error in node w.1[...]: Invalid parent values"`). During sampling, weight parameters can take on extreme values, and if the weight function is not bounded, this will destabilize the likelihood.

Recommendations:

- **Use bounded weight functions.** Two options:
  - `ilogit()`: Bounds weights between 0 and 1 with a zero-point at 0.5: `fn(w ~ ilogit(b0 + b1 * x), c = TRUE)`
  - **Generalized logistic** (Rosche, 2026): Bounds weights between 0 and 1 with a zero-point at $1/n$ (equal weights), so deviations from equal weighting are estimated as a function of covariates: `fn(w ~ 1 / (1 + (n - 1) * exp(-(b0 + b1 * x))), c = TRUE)`
- **Use** `c = TRUE` (weight normalization) to prevent weights from growing without bound
- **Standardize covariates** in the weight function. Variables with large ranges (e.g., income in thousands) can cause `b * x` to overflow
- **Use informative priors** for weight parameters via the `priors` argument in `bml` (e.g., `priors = list("b.w.1[1] ~ dnorm(0, 1)")`)
- **Avoid unbounded functions** like `exp(b * x)` without normalization (`c = TRUE`) or wrapping (e.g., inside `ilogit()`)

Weight parameters are initialized at 0 by default to ensure numerically stable starting values. See `vignette("faq")` (Question 7) for detailed troubleshooting of numerical issues.

Value

A `bml_fn` object containing the parsed weight function specification.

References

Rosche, B. (2026). A Multilevel Model for Theorizing and Estimating the Micro-Macro Link. *Political Analysis*.

Browne, W. J., Goldstein, H., & Rasbash, J. (2001). Multiple membership multiple classification (MMMC) models. *Statistical Modelling*, 1(2), 103-124.

See Also

`mm`, `bml`, `vignette("model")` for the model structure, `vignette("faq")` for troubleshooting

Examples

```
# Equal weights (standard multiple-membership)
fn(w ~ 1/n, c = TRUE)

# Tenure-based weights (proportional to time served)
fn(w ~ tenure, c = TRUE)

# Flexible parameterized weights
fn(w ~ b0 + b1 * seniority, c = TRUE)

# Unconstrained weights
fn(w ~ importance, c = FALSE)

# Weights based on group aggregates
fn(w ~ b1 * min(tenure) + (1 - b1) * mean(tenure), c = TRUE)
```

```
# Combining individual and aggregate measures
fn(w ~ b0 + b1 * (tenure / max(tenure)), c = TRUE)

# Using median for robust central tendency
fn(w ~ tenure / median(tenure), c = TRUE)

# Using quantiles for percentile-based weights
fn(w ~ quantile(tenure, 0.75) - quantile(tenure, 0.25), c = TRUE)
```

---

hm — *Define a hierarchical nesting structure*

Description

Specifies a hierarchical (nesting) level in the model where group-level units are nested within higher-level entities. Unlike multiple-membership structures, each group belongs to exactly one nesting-level unit. Can model either random effects or fixed effects at the nesting level.

Usage

```
hm(id, vars = NULL, name = NULL, type = "RE", showFE = FALSE, ar = FALSE)
```

Arguments

`id`	An `id` object specifying the nesting-level identifier: `id(hmid)` where `hmid` identifies the higher-level units (e.g., countries, regions).
`vars`	A `vars` object specifying nesting-level covariates, or `NULL` for intercept-only effects. Supports interactions (`*`, `:`) and transformations (`I()`).
`name`	Unquoted variable name for nesting-level labels (optional). If provided, these labels will be displayed in model output for fixed effects.
`type`	Character; either `"RE"` for random effects (default) or `"FE"` for fixed effects at the nesting level.
`showFE`	Logical; if `TRUE` and `type = "FE"`, fixed effect estimates for each nesting-level unit are included in output. Default: `FALSE`.
`ar`	Logical; if `TRUE`, random effects evolve autoregressively across participations at the nesting level. Requires sequential participation indicators in the data. Default: `FALSE`.

Details

Hierarchical vs. Multiple-Membership:

Hierarchical structures (`hm`) model strict nesting: each group belongs to exactly one higher-level unit. Use `mm` when groups can have memberships in multiple units.

Random vs. Fixed Effects:

- **Random effects** (`type = "RE"`): Nesting-level units are treated as a random sample from a population. Best when you have many units and want to generalize.
- **Fixed effects** (`type = "FE"`): Each unit gets its own parameter. Best when you have few units or want to estimate unit-specific effects.

Cross-Classification:

Multiple `hm()` blocks create cross-classified models where groups are simultaneously nested within multiple non-nested hierarchies (e.g., schools within both neighborhoods and districts).

Value

A `bml_hm` object containing the hierarchical specification.

References

Goldstein, H. (2011). *Multilevel Statistical Models* (4th ed.). Wiley.

Rabe-Hesketh, S., & Skrondal, A. (2012). *Multilevel and Longitudinal Modeling Using Stata* (3rd ed.). Stata Press.

See Also

`bml`, `mm`, `id`, `vars`

Examples

```
# Random effects with covariates
hm(
  id = id(cid),
  vars = vars(gdp + democracy),
  name = cname,
  type = "RE"
)

# Random intercepts only
hm(
  id = id(cid),
  vars = NULL,
  type = "RE"
)

# Fixed effects
hm(
  id = id(cid),
  vars = NULL,
  name = cname,
  type = "FE",
  showFE = TRUE  # Show estimates for each country
)

# Autoregressive random effects
hm(
  id = id(cid),
  vars = NULL,
  type = "RE",
  ar = TRUE  # Effects evolve over time
)
```

---

id *Specify identifier variables for multiple-membership and hierarchical structures*

Description

Helper function used within mm and hm to specify the identifier variables that define memberships and nesting structures. In multiple-membership models, `id()` links member-level units (e.g., party IDs) to group-level units (e.g., government IDs). In hierarchical models, `id()` specifies the nesting-level identifier (e.g., country ID).

Usage

```
id(...)
```

Arguments

`...`	Unquoted variable names from your data: • For `mm()`: Two identifiers `id(mmid, mainid)` where `mmid` identifies member-level units and `mainid` identifies group-level units • For `hm()`: One identifier `id(hmid)` where `hmid` identifies nesting-level units

Value

A `bml_id` object containing the variable names as character strings.

See Also

mm, hm, bml

Examples

```
# Multiple-membership: parties (pid) within governments (gid)
id(pid, gid)

# Hierarchical: governments within countries
id(cid)
```

`mcmcDiag` *Summarize MCMC convergence diagnostics*

Description

Computes common convergence diagnostics for selected parameters from a JAGS/BUGS fit and returns a compact, report-ready table. The diagnostics include Gelman–Rubin $\hat{R}$, Geweke z-scores, Heidelberger-Welch stationarity p-values, and autocorrelation at a user-specified lag.

Usage

```
mcmcDiag(bml.out, parameters, lag = 50)
```

Arguments

`bml.out`	A model fit object containing JAGS output, typically as returned by `R2jags::jags()`, with component `$jags.out$BUGSoutput`.
`parameters`	Character vector of parameter names (or patterns) to extract. These may be exact names or patterns (e.g., a prefix like `"b"` that matches `"b[1]"`, `"b[2]"`, . . .).
`lag`	Integer specifying the lag at which to compute autocorrelation. Default: 50. Lower values (e.g., 10) capture short-range dependence; higher values assess whether the chain has mixed well over longer intervals.

Details

Internally, the function converts the BUGS/JAGS output to a `coda::mcmc.list`, then computes per-chain diagnostics and averages them across chains for each parameter:

- **Gelman–Rubin** ($\hat{R}$): `coda::gelman.diag()`. Values close to 1 indicate convergence; a common heuristic is $\hat{R} \leq 1.1$.
- **Geweke** z-score: `coda::geweke.diag()`. Large absolute values (e.g., $|z| > 2$) suggest lack of convergence.
- **Heidelberger–Welch** p-value: `coda::heidel.diag()` tests the null of stationarity in the chain segment.
- **Autocorrelation**: `coda::autocorr()` at the specified `lag`, averaged across chains. Values near zero indicate good mixing; persistent autocorrelation suggests the chain needs thinning or reparameterization.

All statistics are rounded to three decimals. The returned table is transposed so that *rows are diagnostics* and *columns are parameters*.

Value

A `data.frame` with one row per diagnostic and one column per parameter; cell entries are the average diagnostic values across chains. Row names include: `"Gelman/Rubin convergence statistic"`, `"Geweke z-score"`, `"Heidelberger/Welch p-value"`, `"Autocorrelation (lag <lag>)"`.

Author(s)

Benjamin Rosche <benrosche@nyu.edu>

References

Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4), 457-472.

Brooks, S. P., & Gelman, A. (1998). General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7(4), 434-455.

See Also

gelman.diag, geweke.diag, heidel.diag, autocorr

Examples

```
data(coalgov)

# Fit model
m1 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + majority +
    mm(id = id(pid, gid), vars = vars(cohesion), fn = fn(w ~ 1/n), RE = TRUE),
  family = "Weibull",
  monitor = TRUE,
  data = coalgov
)

# Check convergence for main parameters
mcmcDiag(m1, parameters = "b")  # All b coefficients

# Check specific parameters
mcmcDiag(m1, parameters = c("b[1]", "b[2]", "shape"))

# Check mm block parameters
mcmcDiag(m1, parameters = c("b.mm.1", "sigma.mm.1"))

# Custom autocorrelation lag
mcmcDiag(m1, parameters = "b", lag = 100)

# Interpreting results:
# - Gelman-Rubin < 1.1: Good convergence
# - |Geweke z| < 2: No evidence against convergence
# - Heidelberger p > 0.05: Chain appears stationary
# - Low autocorrelation: Good mixing
```

`mm` *Define a multiple-membership structure*

Description

Specifies a multiple-membership level in the model where group-level units (e.g., governments) are composed of multiple member-level units (e.g., political parties). Unlike pure hierarchical nesting, members can belong to multiple groups, and their contributions are aggregated using a user-specified weight function.

Usage

```
mm(id, vars = NULL, fn = NULL, RE = NULL, ar = FALSE)
```

Arguments

`id`	An `id` object specifying the member-level and group-level identifiers: `id(mmid, mainid)` where `mmid` identifies members and `mainid` identifies groups.
`vars`	A `vars` object specifying member-level covariates to aggregate, or `NULL` for random effects only. Supports interactions (`*`, `:`) and transformations (`I()`). Variables are weighted according to the function specified in `fn`.
`fn`	A `fn` object specifying the weight function (default: `fn(w ~ 1/n, c = TRUE)` for equal weights). Note: Weight functions do NOT support interactions or `I()` - pre-create any needed transformed variables in your data. See `fn` for details.
`RE`	Logical; if `TRUE`, include random effects for member-level units. Automatically set to `TRUE` if `vars = NULL` (random effects only).
`ar`	Logical; if `TRUE`, random effects evolve autoregressively across participations. Requires members to have sequential participation indicators in the data. Default: `FALSE`.

Details

Multiple-Membership Models:

In standard hierarchical models, each observation belongs to exactly one group. Multiple-membership models relax this assumption, allowing groups to be composed of multiple members, with flexible weighting of member contributions.

Model Structure:

The contribution from mm block k to group j is:

$$\text{mm}_{kj} = \sum_{i \in group_j} w_{ki}(x'_{ki}\beta_k + \alpha_{ki})$$

where:

- w_{ki}: Weight for member i in group j (from `fn`)

- x_{ki}: Member-level covariates (from `vars`)
- β_k: Regression coefficients (estimated)
- α_{ki}: Member-level random effect (if `RE = TRUE`)

Multiple mm() Blocks:

You can specify multiple `mm()` blocks with different weight functions, variables, or random effect specifications. This allows modeling different aggregation mechanisms simultaneously.

Value

A `bml_mm` object containing the multiple-membership specification.

References

Rosche, B. (2026). A Multilevel Model for Theorizing and Estimating the Micro-Macro Link. *Political Analysis*.

Browne, W. J., Goldstein, H., & Rasbash, J. (2001). Multiple membership multiple classification (MMMC) models. *Statistical Modelling*, 1(2), 103-124.

Fielding, A., & Goldstein, H. (2006). *Cross-classified and multiple membership structures in multilevel models: An introduction and review*. Research Report RR791, Department for Education and Skills.

See Also

`bml`, `id`, `vars`, `fn`, `hm`

Examples

```
# Equal weights with variables
mm(
  id = id(pid, gid),
  vars = vars(rile + ipd),
  fn = fn(w ~ 1/n, c = TRUE),
  RE = FALSE
)

# Random effects only (no variables)
mm(
  id = id(pid, gid),
  vars = NULL,
  fn = fn(w ~ 1/n, c = TRUE),
  RE = TRUE  # Automatically TRUE when vars = NULL
)

# Flexible weights with parameter
mm(
  id = id(pid, gid),
  vars = vars(org_structure),
  fn = fn(w ~ ilogit(b0 + b1 * pseat), c = TRUE),
  RE = TRUE
)

# Autoregressive random effects
mm(
  id = id(pid, gid),
  vars = NULL,
  fn = fn(w ~ 1/n, c = TRUE),
  RE = TRUE,
  ar = TRUE  # Random effects evolve over participations
)

# Interactions and transformations in vars
mm(
  id = id(pid, gid),
  vars = vars(rile * ipd),  # Main effects plus interaction
  fn = fn(w ~ 1/n, c = TRUE),
  RE = FALSE
)

mm(
  id = id(pid, gid),
  vars = vars(rile + I(rile^2)),  # Quadratic term
  fn = fn(w ~ 1/n, c = TRUE),
  RE = FALSE
)
```

`monetPlot` *Visualize posterior distributions with density and trace plots*

Description

Creates a combined diagnostic plot showing both the posterior density and MCMC trace plot for a specified parameter. Helps assess convergence and visualize posterior uncertainty. The plot displays the median and 90% highest posterior density (HPD) interval.

Usage

```
monetPlot(bml, parameter, label = NULL, r = 2, yaxis = TRUE)
```

Arguments

`bml`	A fitted model object of class `"bml"` returned by `bml`. Must be fitted with `monitor = TRUE` to store MCMC chains.
`parameter`	Character string specifying the parameter to plot. Must use the internal parameter name (i.e., row names from `bml$reg.table`). Examples: `"b[1]"` (intercept), `"b[2]"` (first covariate), `"b.mm.1"` (first mm block coefficient), `"sigma.mm"` (mm random effect SD).

`label`	Optional character string for the parameter label displayed on the plot. If `NULL` (default), uses the internal parameter name.
`r`	Number of decimal places for displayed quantiles and statistics. Default: 2.
`yaxis`	Logical; if `TRUE` (default), display axis titles ("Density" and "Scans"). If `FALSE`, omit axis titles for cleaner appearance when combining multiple plots.

Details

Interpreting the Plot:

- **Density panel**: Shows the posterior distribution. The dashed line marks the median (central estimate). Shading indicates the 90% credible region.
- **Trace panel**: Shows parameter values across MCMC iterations for each chain. Good mixing looks like "fuzzy caterpillars" with chains overlapping. Poor mixing shows trends, stickiness, or separation between chains.

Convergence Checks:

- Chains should overlap and explore the same space
- No sustained trends or drift
- Rapid mixing (no long autocorrelation)

Use `mcmcDiag` for formal convergence statistics (Gelman-Rubin, Geweke, etc.).

Value

A `ggplot` object (using `patchwork`) combining two panels:

- **Top panel**: Posterior density with shaded 90% HPD interval. Solid vertical line at zero, dashed line at posterior median.
- **Bottom panel**: Trace plot showing MCMC iterations across chains. Same reference lines as top panel. Helps diagnose convergence and mixing.

Author(s)

Benjamin Rosche <benrosche@nyu.edu>

See Also

`bml`, `mcmcDiag`, `summary.bml`

Examples

```
data(coalgov)

# Fit model with monitoring enabled
m1 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + majority +
    mm(id = id(pid, gid), vars = vars(cohesion), fn = fn(w ~ 1/n), RE = TRUE) +
    hm(id = id(cid), type = "RE"),
  family = "Weibull",
  monitor = TRUE,  # Required for monetPlot
  data = coalgov
)

# Plot intercept
monetPlot(m1, parameter = "b[1]", label = "Intercept")

# Plot majority coefficient with custom label
monetPlot(m1, parameter = "b[2]", label = "Majority Government Effect")

# Plot mm coefficient
monetPlot(m1, parameter = "b.mm.1", label = "Party Fragmentation")

# Plot random effect SD
monetPlot(m1, parameter = "sigma.mm.1")

# List available parameters
rownames(m1$reg.table)
```

`summary.bml` *Summarize a fitted bml model*

Description

S3 method for summarizing `bml` model objects. Returns a formatted table of parameter estimates with posterior means, standard deviations, and credible intervals, along with model information and convergence statistics.

Usage

```
## S3 method for class 'bml'
summary(object, r = 3, ...)
```

Arguments

`object`	A fitted model object of class `"bml"` returned by `bml`.
`r`	Number of decimal places for rounding numeric output. Default: 3.
`...`	Additional arguments (currently unused).

Details

The summary method rounds all numeric values for readability while preserving the underlying structure and metadata from the fitted model. All columns remain accessible via standard data frame indexing (e.g., `$Parameter`, `$mean`).

For Cox models with piecewise baseline hazards (when `cox_intervals` is specified), the outcome description includes the number of intervals used.

Value

A data frame of class "bml_summary" containing rounded parameter estimates with the following columns:

- Parameter: Labeled parameter names
- mean: Posterior mean
- sd: Posterior standard deviation
- lb: Lower bound of 95% credible interval
- ub: Upper bound of 95% credible interval

The object includes metadata attributes printed above the table:

- Outcome family and link function
- Estimate type (posterior mean from MCMC)
- Credible interval specification (95% equal-tailed)
- Level specification (mm and hm block details)
- DIC (Deviance Information Criterion) for model comparison

Author(s)

Benjamin Rosche <benrosche@nyu.edu>

See Also

bml, monetPlot, mcmcDiag

Examples

```
data(coalgov)

# Fit model
m1 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + majority +
    mm(id = id(pid, gid), vars = vars(cohesion), fn = fn(w ~ 1/n), RE = TRUE) +
    hm(id = id(cid), type = "RE"),
  family = "Weibull",
  data = coalgov
)

# View summary
summary(m1)

# Summary with more decimal places
summary(m1, r = 4)

# Access specific columns
s <- summary(m1)
s$Parameter  # Parameter names
s$mean       # Posterior means
```

```
s$lb          # Lower credible bounds

# Custom posterior summaries (requires monitor = TRUE)
# Extract posterior draws as a tidy data frame
draws <- coda::as.mcmc.list(m1$jags.out$BUGSoutput) |> as.matrix() |> as_tibble()

# Select specific parameters and compute custom summaries
draws |>
  dplyr::select(dplyr::starts_with("b[")) |>
  tidyr::pivot_longer(everything(), names_to = "param") |>
  dplyr::group_by(param) |>
  dplyr::summarise(
    median = median(value),
    mad    = mad(value),
    q05    = quantile(value, 0.05),
    q95    = quantile(value, 0.95)
  )
```

varDecomp   *Variance decomposition for fitted bml models*

Description

Computes a posterior variance decomposition and intraclass correlation coefficients (ICCs) from a fitted `bml` model. The function automatically discovers all variance components (sigma parameters) in the model, applies weight adjustments for multiple-membership levels, and returns posterior summaries.

Usage

```
varDecomp(model, uncertainty = "sd", r = 2)
```

Arguments

`model`	A fitted model object of class `"bml"` returned by `bml`. Must have been fitted with `monitor = TRUE` (the default).
`uncertainty`	Uncertainty measure to report. One of `"sd"` (posterior standard deviation, the default), `"mad"` (median absolute deviation), or `"ci"` (95% credible interval with lower/upper bounds).
`r`	Number of decimal places for rounding numeric output. Default: 2.

Details

Variance decomposition. The total variance of the outcome is partitioned into additive components, one for each level in the model:

$$\mathrm{Var}(y) = \sigma_1^2 + \sigma_2^2 + \ldots$$

Each component contributes variance σ^2, except for multiple-membership (MM) levels, where the effective variance contribution is scaled by the average of the summed squared weights across groups.

$$\mathrm{Var}_{\mathrm{mm}} = \sigma_{\mathrm{mm}}^2 \cdot \overline{w^2}, \quad \overline{w^2} = \frac{1}{N}\sum_{i=1}^{N}\sum_{k} w_{ik}^2$$

This weight adjustment accounts for the fact that the member-level variance is distributed across multiple members with potentially unequal influence. With equal weights ($w_{ik} = 1/n_i$), the effective variance shrinks as group size increases.

Intraclass Correlation Coefficient (ICC). The ICC for a given level is the proportion of total variance attributable to that level:

$$\rho_l = \frac{\sigma_l^2}{\sum_{l'} \sigma_{l'}^2}$$

Intuitively, the ICC answers: "What fraction of the total variation in the outcome is due to differences between units at this level?" An ICC of 0.30 for the country level, for example, means that 30% of the outcome variation can be attributed to between-country differences.

ICCs are computed per posterior draw and then summarized, properly propagating uncertainty from the MCMC samples.

Family-specific handling:

- **Gaussian / Weibull**: The residual `sigma` from the model is used directly.
- **Binomial**: There is no residual sigma. The latent logistic residual variance $\pi^2/3 \approx 3.29$ is used instead.
- **Cox**: There is no residual variance. ICCs are computed among the non-residual components only.

Value

A data frame of class `"bml_varDecomp"` with one row per variance component. Always includes `Component`, `sigma`, and `ICC` columns. Additional columns depend on `uncertainty`:

- `"sd"`: `sigma_sd` and `ICC_sd`
- `"mad"`: `sigma_mad` and `ICC_mad`
- `"ci"`: `sigma_lb`, `sigma_ub`, `ICC_lb`, `ICC_ub`

Author(s)

Benjamin Rosche <benrosche@nyu.edu>

See Also

`bml`, `summary.bml`

Examples

```
data(coalgov)

m1 <- bml(
  Surv(dur_wkb, event_wkb) ~ 1 + majority +
    mm(id = id(pid, gid), vars = vars(cohesion), fn = fn(w ~ 1/n), RE = TRUE) +
    hm(id = id(cid), type = "RE"),
  family = "Weibull",
  data = coalgov
)

varDecomp(m1)
varDecomp(m1, uncertainty = "ci")
varDecomp(m1, uncertainty = "mad")
```

---

vars *Specify covariates for multiple-membership or hierarchical models*

Description

Helper function used within `mm` and `hm` to specify which variables should be included at each level of the model. Supports both free variables (with coefficients to be estimated) and fixed variables (with coefficients held constant using `fix`).

Usage

```
vars(...)
```

Arguments

`...`	Unquoted variable names from your data, combined using + (formula-style). Supports: • Simple variables: `vars(x + y)` • Interactions: `vars(x * y)` or `vars(x:y)` • Transformations: `vars(I(x^2))` or `vars(I(x + y))` • Fixed coefficients: `vars(fix(x, 1.0) + y)` Note: Numeric literals like `1`, `0`, or `-1` are ignored (no intercept support in mm/hm blocks).

Value

A `bml_vars` object containing:

- `formula`: Formula object for use with `model.matrix()`
- `free`: Character vector of base variable names
- `fixed`: List of variables with fixed coefficients (if any)

Returns `NULL` if no variables are specified.

See Also

fix, mm, hm

Examples

```
# Simple variable specification (formula-style with +)
vars(income + education)

# Single variable
vars(income)

# Interactions
vars(income * education)  # expands to income + education + income:education
vars(income:education)    # interaction only

# Transformations
vars(I(income^2))         # squared term
vars(income + I(income^2)) # linear and squared

# Mix free and fixed variables
vars(fix(exposure, 1.0) + income + education)

# Use in mm() specification
mm(
  id = id(pid, gid),
  vars = vars(rile + ipd),
  fn = fn(w ~ 1/n),
  RE = FALSE
)
```

Index