Package ‘brms.mmrm’

October 2, 2024

Title Bayesian MMRMs using 'brms'

Version 1.1.1

Description The mixed model for repeated measures (MMRM) is a popular model for longitudinal clinical trial data with continuous endpoints, and 'brms' is a powerful and versatile package for fitting Bayesian regression models. The 'brms.mmrm' R package leverages 'brms' to run MMRMs, and it supports a simplified interfaced to reduce difficulty and align with the best practices of the life sciences. References: Bürkner (2017) <doi:10.18637/jss.v080.i01>, Mallinckrodt (2008) <doi:10.1177/009286150804200402>.

License MIT + file LICENSE

URL `https://openpharma.github.io/brms.mmrm/`,
`https://github.com/openpharma/brms.mmrm`

BugReports `https://github.com/openpharma/brms.mmrm/issues`

Depends R (>= 4.0.0)

Imports brms (>= 2.19.0), dplyr, ggplot2, ggridges, MASS, posterior, purrr, rlang, stats, tibble, tidyr, tidyselect, trialr, utils, zoo

Suggests BH, emmeans (>= 1.8.7), fst, gt, gtsummary, knitr (>= 1.30), markdown (>= 1.1), mmrm, parallel, Rcpp, RcppEigen, RcppParallel, rmarkdown (>= 2.4), rstan, StanHeaders, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

NeedsCompilation no

Author William Michael Landau [aut, cre]
(<`https://orcid.org/0000-0003-1878-3253`>),

Kevin Kunzmann [aut] (<https://orcid.org/0000-0002-1140-7143>),
Yoni Sidi [aut],
Christian Stock [aut] (<https://orcid.org/0000-0002-3493-3234>),
Eli Lilly and Company [cph, fnd],
Boehringer Ingelheim Pharma GmbH & Co. KG [cph, fnd]

Maintainer William Michael Landau <will.landau.oss@gmail.com>

Repository CRAN

Date/Publication 2024-10-02 20:10:01 UTC

Contents

`brms.mmrm-package` *brms.mmrm: Bayesian MMRMs using* `brms`

Description

The mixed model for repeated measures (MMRM) is a popular model for longitudinal clinical trial data with continuous endpoints, and `brms` a is powerful and versatile package for fitting Bayesian regression models. The `brms.mmrm` R package leverages `brms` to run MMRMs, and it supports a simplified interfaced to reduce difficulty and align with the best practices of the life sciences.

References

- Bürkner, P.-C. (2017), "brms: An R package for Bayesian multilevel models using Stan," Journal of Statistical Software, 80, 1–28. https://doi.org/10.18637/jss.v080.i01.
- Holzhauer, B., and Weber, S. (2024), "Bayesian mixed effects model for repeated measures," in Applied Modeling in Drug Development, Novartis AG. https://opensource.nibr.com/bamdd/src/02h_mmrm.html.
- Mallinckrodt, C. H., Lane, P. W., Schnell, D., and others (2008), "Recommendations for the primary analysis of continuous endpoints in longitudinal clinical trials," Therapeutic Innovation and Regulatory Science, 42, 303–319. https://doi.org/10.1177/009286150804200402.
- Mallinckrodt, C. H., and Lipkovich, I. (2017), Analyzing longitudinal clinical trial data: A practical guide, CRC Press, Taylor & Francis Group.

`brm_archetype_average_cells` *Cell-means-like time-averaged archetype*

Description

Create a cell-means-like informative prior archetype with a special fixed effect to represent the average across time.

Usage

```
brm_archetype_average_cells(
  data,
  intercept = FALSE,
  baseline = !is.null(attr(data, "brm_baseline")),
 baseline_subgroup = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
 baseline_subgroup_time = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
  baseline_time = !is.null(attr(data, "brm_baseline")),
  covariates = TRUE,
```

```
  prefix_interest = "x_",
  prefix_nuisance = "nuisance_"
)
```

Arguments

`data` A classed data frame from `brm_data()`, or an informative prior archetype from a function like `brm_archetype_successive_cells()`.

`intercept` Logical of length 1. `TRUE` (default) to include an intercept, `FALSE` to omit.

`baseline` Logical of length 1. `TRUE` to include an additive effect for baseline response, `FALSE` to omit. Default is `TRUE` if `brm_data()` previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

`baseline_subgroup` Logical of length 1.

`baseline_subgroup_time` Logical of length 1. `TRUE` to include baseline-by-subgroup-by-time interaction, `FALSE` to omit. Default is `TRUE` if `brm_data()` previously declared baseline and subgroup variables in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

`baseline_time` Logical of length 1. `TRUE` to include baseline-by-time interaction, `FALSE` to omit. Default is `TRUE` if `brm_data()` previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

`covariates` Logical of length 1. `TRUE` (default) to include any additive covariates declared with the `covariates` argument of `brm_data()`, `FALSE` to omit. For informative prior archetypes, this option is set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

`prefix_interest` Character string to prepend to the new columns of generated fixed effects of interest (relating to group, subgroup, and/or time). In rare cases, you may need to set a non-default prefix to prevent name conflicts with existing columns in the data, or rename the columns in your data. `prefix_interest` must not be the same value as `prefix_nuisance`.

`prefix_nuisance` Same as `prefix_interest`, but relating to generated fixed effects NOT of interest (not relating to group, subgroup, or time). Must not be the same value as `prefix_interest`.

Details

This archetype has a special fixed effect for each treatment group to represent the mean response averaged across all the time points.

To illustrate, suppose the dataset has two treatment groups A and B, time points 1, 2, and 3, and no other covariates.

Let `mu_gt` be the marginal mean of the response at group `g` time `t` given data and hyperparameters. The model has fixed effect parameters `beta_1`, `beta_2`, ..., `beta_6` which express the marginal means `mu_gt` as follows:

```
`mu_A1 = 3 * beta_1 - beta_2 - beta_3`
`mu_A2 = beta_2`
`mu_A3 = beta_3`

`mu_B1 = 3 * beta_4 - beta_5 - beta_6`
`mu_B2 = beta_5`
`mu_B3 = beta_6`
```

For group A, `beta_1` is the average response in group A averaged across time points. You can confirm this yourself by expressing the average across time `(mu_A1 + mu_A2 + mu_A3) / 3` in terms of the `beta_*` parameters and confirming that the expression simplifies down to just `beta_1`. `beta_2` represents the mean response in group A at time 2, and `beta_3` represents the mean response in group A at time 3. `beta_4`, `beta_5`, and `beta_6` are analogous for group B.

Value

A special classed `tibble` with data tailored to the cell-means-like time-averaged archetype. The dataset is augmented with extra columns with the `"archetype_"` prefix, as well as special attributes to tell downstream functions like `brm_formula()` what to do with the object.

Prior labeling for `brm_archetype_average_cells()`

Within each treatment group, the initial time point represents the average, and each successive time point represents the response within that actual time. To illustrate, consider the example in the Details section. In the labeling scheme for `brm_archetype_average_cells()`, you can label the prior on `beta_1` using `brm_prior_label(code = "normal(1.2, 5)", group = "A", time = "1")`. Similarly, you cal label the prior on `beta_5` with `brm_prior_label(code = "normal(1.3, 7)", group = "B", time = "2")`. To confirm that you set the prior correctly, compare the `brms` prior with the output of `summary(your_archetype)`. See the examples for details.

Nuisance variables

In the presence of covariate adjustment, functions like `brm_archetype_successive_cells()` convert nuisance factors into binary dummy variables, then center all those dummy variables and any continuous nuisance variables at their means in the data. This ensures that the main model coefficients of interest are not implicitly conditional on a subset of the data. In other words, preprocessing nuisance variables this way preserves the interpretations of the fixed effects of interest, and it ensures informative priors can be specified correctly.

Prior labeling

Informative prior archetypes use a labeling scheme to assign priors to fixed effects. How it works:

```
1. First, assign the prior of each parameter a collection
  of labels from the data. This can be done manually or with
  successive calls to [brm_prior_label()].
2. Supply the labeling scheme to [brm_prior_archetype()].
  [brm_prior_archetype()] uses attributes of the archetype
  to map labeled priors to their rightful parameters in the model.
```

For informative prior archetypes, this process is much more convenient and robust than manually calling `brms::set_prior()`. However, it requires an understanding of how the labels of the priors map to parameters in the model. This mapping varies from archetype to archetype, and it is documented in the help pages of archetype-specific functions such as `brm_archetype_successive_cells()`.

See Also

Other informative prior archetypes: `brm_archetype_average_effects()`, `brm_archetype_cells()`, `brm_archetype_effects()`, `brm_archetype_successive_cells()`, `brm_archetype_successive_effects()`

Examples

```
set.seed(0L)
data <- brm_simulate_outline(
  n_group = 2,
  n_patient = 100,
  n_time = 4,
  rate_dropout = 0,
  rate_lapse = 0
) |>
  dplyr::mutate(response = rnorm(n = dplyr::n())) |>
  brm_data_change() |>
  brm_simulate_continuous(names = c("biomarker1", "biomarker2")) |>
  brm_simulate_categorical(
    names = c("status1", "status2"),
    levels = c("present", "absent")
  )
dplyr::select(
  data,
  group,
  time,
  patient,
  starts_with("biomarker"),
  starts_with("status")
)
archetype <- brm_archetype_average_cells(data)
archetype
summary(archetype)
formula <- brm_formula(archetype)
formula
prior <- brm_prior_label(
```

```
    code = "normal(1, 2.2)",
    group = "group_1",
    time = "time_2"
  ) |>
    brm_prior_label("normal(1, 3.3)", group = "group_1", time = "time_3") |>
    brm_prior_label("normal(1, 4.4)", group = "group_1", time = "time_4") |>
    brm_prior_label("normal(2, 2.2)", group = "group_2", time = "time_2") |>
    brm_prior_label("normal(2, 3.3)", group = "group_2", time = "time_3") |>
    brm_prior_label("normal(2, 4.4)", group = "group_2", time = "time_4") |>
    brm_prior_archetype(archetype)
  prior
  class(prior)
  if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
  tmp <- utils::capture.output(
    suppressMessages(
      suppressWarnings(
        model <- brm_model(
          data = archetype,
          formula = formula,
          prior = prior,
          chains = 1,
          iter = 100,
          refresh = 0
        )
      )
    )
  )
  suppressWarnings(print(model))
  brms::prior_summary(model)
  draws <- brm_marginal_draws(
    data = archetype,
    formula = formula,
    model = model
  )
  summaries_model <- brm_marginal_summaries(draws)
  summaries_data <- brm_marginal_data(data)
  brm_plot_compare(model = summaries_model, data = summaries_data)
  }
```

---

brm_archetype_average_effects
Treatment effect time-averaged archetype

Description

Create a treatment effect informative prior archetype with a special fixed effect to represent the average across time.

Usage

```
brm_archetype_average_effects(
  data,
  intercept = FALSE,
  baseline = !is.null(attr(data, "brm_baseline")),
  baseline_subgroup = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
  baseline_subgroup_time = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
  baseline_time = !is.null(attr(data, "brm_baseline")),
  covariates = TRUE,
  prefix_interest = "x_",
  prefix_nuisance = "nuisance_"
)
```

Arguments

- `data` — A classed data frame from `brm_data()`, or an informative prior archetype from a function like `brm_archetype_successive_cells()`.
- `intercept` — Logical of length 1. `TRUE` (default) to include an intercept, `FALSE` to omit.
- `baseline` — Logical of length 1. `TRUE` to include an additive effect for baseline response, `FALSE` to omit. Default is `TRUE` if `brm_data()` previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.
- `baseline_subgroup` — Logical of length 1.
- `baseline_subgroup_time` — Logical of length 1. `TRUE` to include baseline-by-subgroup-by-time interaction, `FALSE` to omit. Default is `TRUE` if `brm_data()` previously declared baseline and subgroup variables in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.
- `baseline_time` — Logical of length 1. `TRUE` to include baseline-by-time interaction, `FALSE` to omit. Default is `TRUE` if `brm_data()` previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.
- `covariates` — Logical of length 1. `TRUE` (default) to include any additive covariates declared with the `covariates` argument of `brm_data()`, `FALSE` to omit. For informative prior archetypes, this option is set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

`prefix_interest`
: Character string to prepend to the new columns of generated fixed effects of interest (relating to group, subgroup, and/or time). In rare cases, you may need to set a non-default prefix to prevent name conflicts with existing columns in the data, or rename the columns in your data. `prefix_interest` must not be the same value as `prefix_nuisance`.

`prefix_nuisance`
: Same as `prefix_interest`, but relating to generated fixed effects NOT of interest (not relating to group, subgroup, or time). Must not be the same value as `prefix_interest`.

Details

This archetype has a special fixed effect for each treatment group to represent the mean response averaged across all the time points, and treatment effects are explicitly parameterized.

To illustrate, suppose the dataset has two treatment groups A (placebo/reference group) and B (active/non-reference group), time points 1, 2, and 3, and no other covariates. Let `mu_gt` be the marginal mean of the response at group `g` time `t` given data and hyperparameters. The model has fixed effect parameters `beta_1`, `beta_2`, ..., `beta_6` which express the marginal means `mu_gt` as follows:

```
`mu_A1 = 3 * beta_1 - beta_2 - beta_3`
`mu_A2 = beta_2`
`mu_A3 = beta_3`

`mu_B1 = 3 * beta_1 - beta_2 - beta_3 + 3 * beta_4 - beta_5 - beta_6`
`mu_B2 = beta_2 + beta_5`
`mu_B3 = beta_3 + beta_6`
```

For group A, `beta_1` is the average response in group A averaged across time points. You can confirm this yourself by expressing the average across time `(mu_A1 + mu_A2 + mu_A3) / 3` in terms of the `beta_*` parameters and confirming that the expression simplifies down to just `beta_1`. `beta_2` represents the mean response in group A at time 2, and `beta_3` represents the mean response in group A at time 3. `beta_4` is the treatment effect of group B relative to group A, averaged across time points. `beta_5` is the treatment effect of B vs A at time 2, and `beta_6` is analogous for time 3.

Value

A special classed `tibble` with data tailored to the treatment effect time-averaged archetype. The dataset is augmented with extra columns with the `"archetype_"` prefix, as well as special attributes to tell downstream functions like `brm_formula()` what to do with the object.

Prior labeling for `brm_archetype_average_effects()`

Within each treatment group, the initial time point represents the average, and each successive time point represents the response within that actual time. To illustrate, consider the example in the Details section. In the labeling scheme for `brm_archetype_average_effects()`, you can label the prior on `beta_1` using `brm_prior_label(code = "normal(1.2, 5)", group = "A", time = "1")`. Similarly, you cal label the prior on `beta_5` with `brm_prior_label(code = "normal(1.3, 7)",` group = "B", time = "2"). To confirm that you set the prior correctly, compare the brms prior with the output of summary(your_archetype). See the examples for details.

Nuisance variables

In the presence of covariate adjustment, functions like brm_archetype_successive_cells() convert nuisance factors into binary dummy variables, then center all those dummy variables and any continuous nuisance variables at their means in the data. This ensures that the main model coefficients of interest are not implicitly conditional on a subset of the data. In other words, preprocessing nuisance variables this way preserves the interpretations of the fixed effects of interest, and it ensures informative priors can be specified correctly.

Prior labeling

Informative prior archetypes use a labeling scheme to assign priors to fixed effects. How it works:

```
1. First, assign the prior of each parameter a collection
  of labels from the data. This can be done manually or with
  successive calls to [brm_prior_label()].
2. Supply the labeling scheme to [brm_prior_archetype()].
  [brm_prior_archetype()] uses attributes of the archetype
  to map labeled priors to their rightful parameters in the model.
```

For informative prior archetypes, this process is much more convenient and robust than manually calling brms::set_prior(). However, it requires an understanding of how the labels of the priors map to parameters in the model. This mapping varies from archetype to archetype, and it is documented in the help pages of archetype-specific functions such as brm_archetype_successive_cells().

See Also

Other informative prior archetypes: brm_archetype_average_cells(), brm_archetype_cells(), brm_archetype_effects(), brm_archetype_successive_cells(), brm_archetype_successive_effects()

Examples

```
set.seed(0L)
data <- brm_simulate_outline(
  n_group = 2,
  n_patient = 100,
  n_time = 4,
  rate_dropout = 0,
  rate_lapse = 0
) |>
  dplyr::mutate(response = rnorm(n = dplyr::n())) |>
  brm_data_change() |>
  brm_simulate_continuous(names = c("biomarker1", "biomarker2")) |>
  brm_simulate_categorical(
    names = c("status1", "status2"),
    levels = c("present", "absent")
  )
dplyr::select(
```

```
  data,
  group,
  time,
  patient,
  starts_with("biomarker"),
  starts_with("status")
)
archetype <- brm_archetype_average_effects(data)
archetype
summary(archetype)
formula <- brm_formula(archetype)
formula
prior <- brm_prior_label(
  code = "normal(1, 2.2)",
  group = "group_1",
  time = "time_2"
) |>
  brm_prior_label("normal(1, 3.3)", group = "group_1", time = "time_3") |>
  brm_prior_label("normal(1, 4.4)", group = "group_1", time = "time_4") |>
  brm_prior_label("normal(2, 2.2)", group = "group_2", time = "time_2") |>
  brm_prior_label("normal(2, 3.3)", group = "group_2", time = "time_3") |>
  brm_prior_label("normal(2, 4.4)", group = "group_2", time = "time_4") |>
  brm_prior_archetype(archetype)
prior
class(prior)
if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
tmp <- utils::capture.output(
  suppressMessages(
    suppressWarnings(
      model <- brm_model(
        data = archetype,
        formula = formula,
        prior = prior,
        chains = 1,
        iter = 100,
        refresh = 0
      )
    )
  )
)
suppressWarnings(print(model))
brms::prior_summary(model)
draws <- brm_marginal_draws(
  data = archetype,
  formula = formula,
  model = model
)
summaries_model <- brm_marginal_summaries(draws)
summaries_data <- brm_marginal_data(data)
brm_plot_compare(model = summaries_model, data = summaries_data)
}
```

brm_archetype_cells *Cell means archetype*

Description

Create an informative prior archetype for cell means.

Usage

```
brm_archetype_cells(
  data,
  intercept = FALSE,
  baseline = !is.null(attr(data, "brm_baseline")),
 baseline_subgroup = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
 baseline_subgroup_time = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
  baseline_time = !is.null(attr(data, "brm_baseline")),
  covariates = TRUE,
  clda = FALSE,
  prefix_interest = "x_",
  prefix_nuisance = "nuisance_"
)
```

Arguments

`data` A classed data frame from `brm_data()`, or an informative prior archetype from a function like `brm_archetype_successive_cells()`.

`intercept` `TRUE` to make one of the parameters an intercept, `FALSE` otherwise. If `TRUE`, then the interpretation of the parameters in the "Details" section will change, and you are responsible for manually calling `summary()` on the archetype and interpreting the parameters according to the output. In addition, you are responsible for setting an appropriate prior on the intercept. In normal usage, `brms` looks for a model parameter called `"Intercept"` and uses the data to set the prior to help the MCMC runs smoothly. If `intercept = TRUE` for informative prior archetypes, the intercept will be called something else, and `brms` cannot auto-generate a sensible default prior.

`baseline` Logical of length 1. `TRUE` to include an additive effect for baseline response, `FALSE` to omit. Default is `TRUE` if `brm_data()` previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

`baseline_subgroup`
Logical of length 1.

baseline_subgroup_time
: Logical of length 1. TRUE to include baseline-by-subgroup-by-time interaction, FALSE to omit. Default is TRUE if brm_data() previously declared baseline and subgroup variables in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

baseline_time
: Logical of length 1. TRUE to include baseline-by-time interaction, FALSE to omit. Default is TRUE if brm_data() previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

covariates
: Logical of length 1. TRUE (default) to include any additive covariates declared with the covariates argument of brm_data(), FALSE to omit. For informative prior archetypes, this option is set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

clda
: TRUE to opt into constrained longitudinal data analysis (cLDA), FALSE otherwise. To use cLDA, reference_time must have been non-NULL in the call to brm_data() used to construct the data.

  Some archetypes cannot support cLDA (e.g. brm_archetype_average_cells() and brm_archetype_average_effects()).

  In cLDA, the fixed effects parameterization is restricted such that all treatment groups are pooled at baseline. (If you supplied a subgroup variable in brm_data(), then this constraint is applied separately within each subgroup variable.) cLDA may result in more precise estimates when the time variable has a baseline level and the baseline outcomes are recorded before randomization in a clinical trial.

prefix_interest
: Character string to prepend to the new columns of generated fixed effects of interest (relating to group, subgroup, and/or time). In rare cases, you may need to set a non-default prefix to prevent name conflicts with existing columns in the data, or rename the columns in your data. prefix_interest must not be the same value as prefix_nuisance.

prefix_nuisance
: Same as prefix_interest, but relating to generated fixed effects NOT of interest (not relating to group, subgroup, or time). Must not be the same value as prefix_interest.

Details

In this archetype, each fixed effect is a cell mean: the group mean for a given value of treatment group and discrete time (and subgroup level, if applicable).

Value

A special classed tibble with data tailored to the successive differences archetype. The dataset is augmented with extra columns with the "archetype_" prefix, as well as special attributes to tell downstream functions like brm_formula() what to do with the object.

Prior labeling for brm_archetype_cells()

Within each treatment group, each model parameter is a cell mean, and the labeling scheme in brm_prior_label() and brm_prior_archetype() translate easily. For example, brm_prior_label(code = "normal(1.2, 5)", group = "B", time = "VISIT2") declares a normal(1.2, 5) prior on the cell mean of treatment group B at discrete time point VISIT2. To confirm that you set the prior correctly, compare the brms prior with the output of summary(your_archetype). See the examples for details.

Nuisance variables

In the presence of covariate adjustment, functions like brm_archetype_successive_cells() convert nuisance factors into binary dummy variables, then center all those dummy variables and any continuous nuisance variables at their means in the data. This ensures that the main model coefficients of interest are not implicitly conditional on a subset of the data. In other words, preprocessing nuisance variables this way preserves the interpretations of the fixed effects of interest, and it ensures informative priors can be specified correctly.

Prior labeling

Informative prior archetypes use a labeling scheme to assign priors to fixed effects. How it works:

```
1. First, assign the prior of each parameter a collection
  of labels from the data. This can be done manually or with
  successive calls to [brm_prior_label()].
2. Supply the labeling scheme to [brm_prior_archetype()].
  [brm_prior_archetype()] uses attributes of the archetype
  to map labeled priors to their rightful parameters in the model.
```

For informative prior archetypes, this process is much more convenient and robust than manually calling brms::set_prior(). However, it requires an understanding of how the labels of the priors map to parameters in the model. This mapping varies from archetype to archetype, and it is documented in the help pages of archetype-specific functions such as brm_archetype_successive_cells().

See Also

Other informative prior archetypes: brm_archetype_average_cells(), brm_archetype_average_effects(), brm_archetype_effects(), brm_archetype_successive_cells(), brm_archetype_successive_effects()

Examples

```
set.seed(0L)
data <- brm_simulate_outline(
  n_group = 2,
  n_patient = 100,
  n_time = 4,
  rate_dropout = 0,
  rate_lapse = 0
```

```
) |>
  dplyr::mutate(response = rnorm(n = dplyr::n())) |>
  brm_data_change() |>
  brm_simulate_continuous(names = c("biomarker1", "biomarker2")) |>
  brm_simulate_categorical(
    names = c("status1", "status2"),
    levels = c("present", "absent")
  )
dplyr::select(
  data,
  group,
  time,
  patient,
  starts_with("biomarker"),
  starts_with("status")
)
archetype <- brm_archetype_cells(data)
archetype
summary(archetype)
formula <- brm_formula(archetype)
formula
prior <- brm_prior_label(
  code = "normal(1, 2.2)",
  group = "group_1",
  time = "time_2"
) |>
  brm_prior_label("normal(1, 3.3)", group = "group_1", time = "time_3") |>
  brm_prior_label("normal(1, 4.4)", group = "group_1", time = "time_4") |>
  brm_prior_label("normal(2, 2.2)", group = "group_2", time = "time_2") |>
  brm_prior_label("normal(2, 3.3)", group = "group_2", time = "time_3") |>
  brm_prior_label("normal(2, 4.4)", group = "group_2", time = "time_4") |>
  brm_prior_archetype(archetype)
prior
class(prior)
if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
tmp <- utils::capture.output(
  suppressMessages(
    suppressWarnings(
      model <- brm_model(
        data = archetype,
        formula = formula,
        prior = prior,
        chains = 1,
        iter = 100,
        refresh = 0
      )
    )
  )
)
suppressWarnings(print(model))
brms::prior_summary(model)
draws <- brm_marginal_draws(
  data = archetype,
```

```
  formula = formula,
  model = model
)
summaries_model <- brm_marginal_summaries(draws)
summaries_data <- brm_marginal_data(data)
brm_plot_compare(model = summaries_model, data = summaries_data)
}
```

---

brm_archetype_effects *Treatment effect archetype*

Description

Create an informative prior archetype for a simple treatment effect parameterization.

Usage

```
brm_archetype_effects(
  data,
  intercept = FALSE,
  baseline = !is.null(attr(data, "brm_baseline")),
  baseline_subgroup = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
  baseline_subgroup_time = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
  baseline_time = !is.null(attr(data, "brm_baseline")),
  covariates = TRUE,
  clda = FALSE,
  prefix_interest = "x_",
  prefix_nuisance = "nuisance_"
)
```

Arguments

data
: A classed data frame from `brm_data()`, or an informative prior archetype from a function like `brm_archetype_successive_cells()`.

intercept
: `TRUE` to make one of the parameters an intercept, `FALSE` otherwise. If `TRUE`, then the interpretation of the parameters in the "Details" section will change, and you are responsible for manually calling `summary()` on the archetype and interpreting the parameters according to the output. In addition, you are responsible for setting an appropriate prior on the intercept. In normal usage, `brms` looks for a model parameter called `"Intercept"` and uses the data to set the prior to help the MCMC runs smoothly. If `intercept = TRUE` for informative prior archetypes, the intercept will be called something else, and `brms` cannot auto-generate a sensible default prior.

baseline
: Logical of length 1. TRUE to include an additive effect for baseline response, FALSE to omit. Default is TRUE if brm_data() previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

baseline_subgroup
: Logical of length 1.

baseline_subgroup_time
: Logical of length 1. TRUE to include baseline-by-subgroup-by-time interaction, FALSE to omit. Default is TRUE if brm_data() previously declared baseline and subgroup variables in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

baseline_time
: Logical of length 1. TRUE to include baseline-by-time interaction, FALSE to omit. Default is TRUE if brm_data() previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

covariates
: Logical of length 1. TRUE (default) to include any additive covariates declared with the covariates argument of brm_data(), FALSE to omit. For informative prior archetypes, this option is set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

clda
: TRUE to opt into constrained longitudinal data analysis (cLDA), FALSE otherwise. To use cLDA, reference_time must have been non-NULL in the call to brm_data() used to construct the data.

  Some archetypes cannot support cLDA (e.g. brm_archetype_average_cells() and brm_archetype_average_effects()).

  In cLDA, the fixed effects parameterization is restricted such that all treatment groups are pooled at baseline. (If you supplied a subgroup variable in brm_data(), then this constraint is applied separately within each subgroup variable.) cLDA may result in more precise estimates when the time variable has a baseline level and the baseline outcomes are recorded before randomization in a clinical trial.

prefix_interest
: Character string to prepend to the new columns of generated fixed effects of interest (relating to group, subgroup, and/or time). In rare cases, you may need to set a non-default prefix to prevent name conflicts with existing columns in the data, or rename the columns in your data. prefix_interest must not be the same value as prefix_nuisance.

prefix_nuisance
: Same as prefix_interest, but relating to generated fixed effects NOT of interest (not relating to group, subgroup, or time). Must not be the same value as prefix_interest.

Details

In this archetype, each fixed effect is either a placebo response or a treatment effect.

To illustrate, suppose the dataset has two treatment groups A and B, time points 1, 2, and 3, and no other covariates. Assume group A is the reference group (e.g. placebo). Let `mu_gt` be the marginal mean of the response at group `g` time `t` given data and hyperparameters. The model has fixed effect parameters `beta_1`, `beta_2`, ..., `beta_6` which express the marginal means `mu_gt` as follows:

```
`mu_A1 = beta_1`
`mu_A2 = beta_2`
`mu_A3 = beta_3`

`mu_B1 = beta_1 + beta_4`
`mu_B2 = beta_2 + beta_5`
`mu_B3 = beta_3 + beta_6`
```

Above, `beta_2` is the group mean of treatment group A at time 2, and `beta_5` is the treatment effect of B relative to A at time 2.

Value

A special classed `tibble` with data tailored to the successive differences archetype. The dataset is augmented with extra columns with the `"archetype_"` prefix, as well as special attributes to tell downstream functions like `brm_formula()` what to do with the object.

Prior labeling for `brm_archetype_effects()`

In the reference group (e.g. placebo) each fixed effect is a cell mean at a time point. In each non-reference group, each fixed effect is the treatment effect relative to the reference (at a time point). The labeling scheme in `brm_prior_label()` and `brm_prior_archetype()` translate straightforwardly. For example, `brm_prior_label(code = "normal(1.2, 5)", group = "A", time = "2")` declares a `normal(1.2, 5)` on `beta_2` (cell mean of the reference group at time 2). Similarly, `brm_prior_label(code = "normal(1.3, 6)", group = "B", time = "2")` declares a `normal(1.3, 6)` prior on the treatment effect of group `B` relative to group `A` at discrete time point `2`. To confirm that you set the prior correctly, compare the `brms` prior with the output of `summary(your_archetype)`. See the examples for details.

Nuisance variables

In the presence of covariate adjustment, functions like `brm_archetype_successive_cells()` convert nuisance factors into binary dummy variables, then center all those dummy variables and any continuous nuisance variables at their means in the data. This ensures that the main model coefficients of interest are not implicitly conditional on a subset of the data. In other words, preprocessing nuisance variables this way preserves the interpretations of the fixed effects of interest, and it ensures informative priors can be specified correctly.

Prior labeling

Informative prior archetypes use a labeling scheme to assign priors to fixed effects. How it works:

```
1. First, assign the prior of each parameter a collection
  of labels from the data. This can be done manually or with
  successive calls to [brm_prior_label()].
2. Supply the labeling scheme to [brm_prior_archetype()].
  [brm_prior_archetype()] uses attributes of the archetype
  to map labeled priors to their rightful parameters in the model.
```

For informative prior archetypes, this process is much more convenient and robust than manually calling `brms::set_prior()`. However, it requires an understanding of how the labels of the priors map to parameters in the model. This mapping varies from archetype to archetype, and it is documented in the help pages of archetype-specific functions such as `brm_archetype_successive_cells()`.

See Also

Other informative prior archetypes: `brm_archetype_average_cells()`, `brm_archetype_average_effects()`, `brm_archetype_cells()`, `brm_archetype_successive_cells()`, `brm_archetype_successive_effects()`

Examples

```
set.seed(0L)
data <- brm_simulate_outline(
  n_group = 2,
  n_patient = 100,
  n_time = 4,
  rate_dropout = 0,
  rate_lapse = 0
) |>
  dplyr::mutate(response = rnorm(n = dplyr::n())) |>
  brm_data_change() |>
  brm_simulate_continuous(names = c("biomarker1", "biomarker2")) |>
  brm_simulate_categorical(
    names = c("status1", "status2"),
    levels = c("present", "absent")
  )
dplyr::select(
  data,
  group,
  time,
  patient,
  starts_with("biomarker"),
  starts_with("status")
)
archetype <- brm_archetype_effects(data)
archetype
summary(archetype)
formula <- brm_formula(archetype)
formula
prior <- brm_prior_label(
  code = "normal(1, 2.2)",
  group = "group_1",
  time = "time_2"
) |>
```

```
    brm_prior_label("normal(1, 3.3)", group = "group_1", time = "time_3") |>
    brm_prior_label("normal(1, 4.4)", group = "group_1", time = "time_4") |>
    brm_prior_label("normal(2, 2.2)", group = "group_2", time = "time_2") |>
    brm_prior_label("normal(2, 3.3)", group = "group_2", time = "time_3") |>
    brm_prior_label("normal(2, 4.4)", group = "group_2", time = "time_4") |>
    brm_prior_archetype(archetype)
  prior
  class(prior)
  if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
  tmp <- utils::capture.output(
    suppressMessages(
      suppressWarnings(
        model <- brm_model(
          data = archetype,
          formula = formula,
          prior = prior,
          chains = 1,
          iter = 100,
          refresh = 0
        )
      )
    )
  )
  suppressWarnings(print(model))
  brms::prior_summary(model)
  draws <- brm_marginal_draws(
    data = archetype,
    formula = formula,
    model = model
  )
  summaries_model <- brm_marginal_summaries(draws)
  summaries_data <- brm_marginal_data(data)
  brm_plot_compare(model = summaries_model, data = summaries_data)
  }
```

---

brm_archetype_successive_cells

Cell-means-like successive differences archetype

Description

Create an informative prior archetype where the fixed effects are successive differences between adjacent time points.

Usage

```
brm_archetype_successive_cells(
  data,
  intercept = FALSE,
  baseline = !is.null(attr(data, "brm_baseline")),
```

```
  baseline_subgroup = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
  baseline_subgroup_time = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
  baseline_time = !is.null(attr(data, "brm_baseline")),
  covariates = TRUE,
  clda = FALSE,
  prefix_interest = "x_",
  prefix_nuisance = "nuisance_"
)
```

Arguments

- `data` — A classed data frame from `brm_data()`, or an informative prior archetype from a function like `brm_archetype_successive_cells()`.
- `intercept` — `TRUE` to make one of the parameters an intercept, `FALSE` otherwise. If `TRUE`, then the interpretation of the parameters in the "Details" section will change, and you are responsible for manually calling `summary()` on the archetype and interpreting the parameters according to the output. In addition, you are responsible for setting an appropriate prior on the intercept. In normal usage, `brms` looks for a model parameter called `"Intercept"` and uses the data to set the prior to help the MCMC runs smoothly. If `intercept = TRUE` for informative prior archetypes, the intercept will be called something else, and `brms` cannot auto-generate a sensible default prior.
- `baseline` — Logical of length 1. `TRUE` to include an additive effect for baseline response, `FALSE` to omit. Default is `TRUE` if `brm_data()` previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.
- `baseline_subgroup` — Logical of length 1.
- `baseline_subgroup_time` — Logical of length 1. `TRUE` to include baseline-by-subgroup-by-time interaction, `FALSE` to omit. Default is `TRUE` if `brm_data()` previously declared baseline and subgroup variables in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.
- `baseline_time` — Logical of length 1. `TRUE` to include baseline-by-time interaction, `FALSE` to omit. Default is `TRUE` if `brm_data()` previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like `brm_archetype_successive_cells()` rather than in `brm_formula()` in order to make sure columns are appropriately centered and the underlying model matrix has full rank.
- `covariates` — Logical of length 1. `TRUE` (default) to include any additive covariates declared with the `covariates` argument of `brm_data()`, `FALSE` to omit. For informative

prior archetypes, this option is set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

clda
: TRUE to opt into constrained longitudinal data analysis (cLDA), FALSE otherwise. To use cLDA, reference_time must have been non-NULL in the call to brm_data() used to construct the data.

  Some archetypes cannot support cLDA (e.g. brm_archetype_average_cells() and brm_archetype_average_effects()).

  In cLDA, the fixed effects parameterization is restricted such that all treatment groups are pooled at baseline. (If you supplied a subgroup variable in brm_data(), then this constraint is applied separately within each subgroup variable.) cLDA may result in more precise estimates when the time variable has a baseline level and the baseline outcomes are recorded before randomization in a clinical trial.

prefix_interest
: Character string to prepend to the new columns of generated fixed effects of interest (relating to group, subgroup, and/or time). In rare cases, you may need to set a non-default prefix to prevent name conflicts with existing columns in the data, or rename the columns in your data. prefix_interest must not be the same value as prefix_nuisance.

prefix_nuisance
: Same as prefix_interest, but relating to generated fixed effects NOT of interest (not relating to group, subgroup, or time). Must not be the same value as prefix_interest.

Details

In this archetype, each fixed effect is either an intercept on the first time point or the difference between two adjacent time points, and each treatment group has its own set of fixed effects independent of the other treatment groups.

To illustrate, suppose the dataset has two treatment groups A and B, time points 1, 2, and 3, and no other covariates. Let mu_gt be the marginal mean of the response at group g time t given data and hyperparameters. The model has fixed effect parameters beta_1, beta_2, ..., beta_6 which express the marginal means mu_gt as follows:

```
`mu_A1 = beta_1`
`mu_A2 = beta_1 + beta_2`
`mu_A3 = beta_1 + beta_2 + beta_3`

`mu_B1 = beta_4`
`mu_B2 = beta_4 + beta_5`
`mu_B3 = beta_4 + beta_5 + beta_6`
```

For group A, beta_1 is the time 1 intercept, beta_2 represents time 2 minus time 1, and beta_3 represents time 3 minus time 2. beta_4, beta_5, and beta_6 behave analogously for group B.

Value

A special classed `tibble` with data tailored to the successive differences archetype. The dataset is augmented with extra columns with the `"archetype_"` prefix, as well as special attributes to tell downstream functions like `brm_formula()` what to do with the object.

Nuisance variables

In the presence of covariate adjustment, functions like `brm_archetype_successive_cells()` convert nuisance factors into binary dummy variables, then center all those dummy variables and any continuous nuisance variables at their means in the data. This ensures that the main model coefficients of interest are not implicitly conditional on a subset of the data. In other words, preprocessing nuisance variables this way preserves the interpretations of the fixed effects of interest, and it ensures informative priors can be specified correctly.

Prior labeling for `brm_archetype_successive_cells()`

Within each treatment group, each intercept is labeled by the earliest time point, and each successive difference term gets the successive time point as the label. To illustrate, consider the example in the Details section. In the labeling scheme for `brm_archetype_successive_cells()`, you can label the prior on `beta_1` using `brm_prior_label(code = "normal(1.2, 5)", group = "A", time = "1")`. Similarly, you cal label the prior on `beta_5` with `brm_prior_label(code = "normal(1.3, 7)", group = "B", time = "2")`. To confirm that you set the prior correctly, compare the `brms` prior with the output of `summary(your_archetype)`. See the examples for details.

Prior labeling

Informative prior archetypes use a labeling scheme to assign priors to fixed effects. How it works:

```
1. First, assign the prior of each parameter a collection
  of labels from the data. This can be done manually or with
  successive calls to [brm_prior_label()].
2. Supply the labeling scheme to [brm_prior_archetype()].
  [brm_prior_archetype()] uses attributes of the archetype
  to map labeled priors to their rightful parameters in the model.
```

For informative prior archetypes, this process is much more convenient and robust than manually calling `brms::set_prior()`. However, it requires an understanding of how the labels of the priors map to parameters in the model. This mapping varies from archetype to archetype, and it is documented in the help pages of archetype-specific functions such as `brm_archetype_successive_cells()`.

See Also

Other informative prior archetypes: `brm_archetype_average_cells()`, `brm_archetype_average_effects()`, `brm_archetype_cells()`, `brm_archetype_effects()`, `brm_archetype_successive_effects()`

Examples

```
set.seed(0L)
data <- brm_simulate_outline(
  n_group = 2,
  n_patient = 100,
  n_time = 4,
  rate_dropout = 0,
  rate_lapse = 0
) |>
  dplyr::mutate(response = rnorm(n = dplyr::n())) |>
  brm_data_change() |>
  brm_simulate_continuous(names = c("biomarker1", "biomarker2")) |>
  brm_simulate_categorical(
    names = c("status1", "status2"),
    levels = c("present", "absent")
  )
dplyr::select(
  data,
  group,
  time,
  patient,
  starts_with("biomarker"),
  starts_with("status")
)
archetype <- brm_archetype_successive_cells(data)
archetype
summary(archetype)
formula <- brm_formula(archetype)
formula
prior <- brm_prior_label(
  code = "normal(1, 2.2)",
  group = "group_1",
  time = "time_2"
) |>
  brm_prior_label("normal(1, 3.3)", group = "group_1", time = "time_3") |>
  brm_prior_label("normal(1, 4.4)", group = "group_1", time = "time_4") |>
  brm_prior_label("normal(2, 2.2)", group = "group_2", time = "time_2") |>
  brm_prior_label("normal(2, 3.3)", group = "group_2", time = "time_3") |>
  brm_prior_label("normal(2, 4.4)", group = "group_2", time = "time_4") |>
  brm_prior_archetype(archetype)
prior
class(prior)
if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
tmp <- utils::capture.output(
  suppressMessages(
    suppressWarnings(
      model <- brm_model(
        data = archetype,
        formula = formula,
        prior = prior,
        chains = 1,
        iter = 100,
```

```
        refresh = 0
      )
    )
  )
)
suppressWarnings(print(model))
brms::prior_summary(model)
draws <- brm_marginal_draws(
  data = archetype,
  formula = formula,
  model = model
)
summaries_model <- brm_marginal_summaries(draws)
summaries_data <- brm_marginal_data(data)
brm_plot_compare(model = summaries_model, data = summaries_data)
}
```

`brm_archetype_successive_effects`

Treatment-effect-like successive differences archetype

Description

Create an informative prior archetype where the fixed effects are successive differences between adjacent time points and terms in non-reference groups are treatment effects.

Usage

```
brm_archetype_successive_effects(
  data,
  intercept = FALSE,
  baseline = !is.null(attr(data, "brm_baseline")),
  baseline_subgroup = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
  baseline_subgroup_time = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
  baseline_time = !is.null(attr(data, "brm_baseline")),
  covariates = TRUE,
  clda = FALSE,
  prefix_interest = "x_",
  prefix_nuisance = "nuisance_"
)
```

Arguments

`data`	A classed data frame from `brm_data()`, or an informative prior archetype from a function like `brm_archetype_successive_cells()`.
`intercept`	Logical of length 1. `TRUE` (default) to include an intercept, `FALSE` to omit.

baseline
: Logical of length 1. TRUE to include an additive effect for baseline response, FALSE to omit. Default is TRUE if brm_data() previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

baseline_subgroup
: Logical of length 1.

baseline_subgroup_time
: Logical of length 1. TRUE to include baseline-by-subgroup-by-time interaction, FALSE to omit. Default is TRUE if brm_data() previously declared baseline and subgroup variables in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

baseline_time
: Logical of length 1. TRUE to include baseline-by-time interaction, FALSE to omit. Default is TRUE if brm_data() previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

covariates
: Logical of length 1. TRUE (default) to include any additive covariates declared with the covariates argument of brm_data(), FALSE to omit. For informative prior archetypes, this option is set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

clda
: TRUE to opt into constrained longitudinal data analysis (cLDA), FALSE otherwise. To use cLDA, reference_time must have been non-NULL in the call to brm_data() used to construct the data.

  Some archetypes cannot support cLDA (e.g. brm_archetype_average_cells() and brm_archetype_average_effects()).

  In cLDA, the fixed effects parameterization is restricted such that all treatment groups are pooled at baseline. (If you supplied a subgroup variable in brm_data(), then this constraint is applied separately within each subgroup variable.) cLDA may result in more precise estimates when the time variable has a baseline level and the baseline outcomes are recorded before randomization in a clinical trial.

prefix_interest
: Character string to prepend to the new columns of generated fixed effects of interest (relating to group, subgroup, and/or time). In rare cases, you may need to set a non-default prefix to prevent name conflicts with existing columns in the data, or rename the columns in your data. prefix_interest must not be the same value as prefix_nuisance.

prefix_nuisance
: Same as prefix_interest, but relating to generated fixed effects NOT of interest (not relating to group, subgroup, or time). Must not be the same value as prefix_interest.

Details

Within the reference treatment group (e.g. placebo), each fixed effect is either an intercept on the first time point or the difference between two adjacent time points. In each non-reference treatment group, each model parameter is defined as an effect relative to the reference group.

To illustrate, suppose the dataset has two treatment groups A and B, time points 1, 2, and 3, and no other covariates. Say group A is the reference group (e.g. placebo). Let mu_gt be the marginal mean of the response at group g time t given data and hyperparameters. The model has fixed effect parameters beta_1, beta_2, ..., beta_6 which express the marginal means mu_gt as follows:

```
`mu_A1 = beta_1`
`mu_A2 = beta_1 + beta_2`
`mu_A3 = beta_1 + beta_2 + beta_3`

`mu_B1 = beta_1 + beta_4`
`mu_B2 = beta_1 + beta_2 + beta_4 + beta_5`
`mu_B3 = beta_1 + beta_2 + beta_3 + beta_4 + beta_5 + beta_6`
```

For group A, beta_1 is the time 1 intercept, beta_2 represents time 2 minus time 1, and beta_3 represents time 3 minus time 2. beta_4 is the treatment effect of group B relative to group A at time 1. beta_5 is the treatment effect of the difference between times 2 and 1, relative to treatment group A. Similarly, beta_6 is the treatment effect of the difference between times 3 and 2, relative to treatment group A.

Value

A special classed tibble with data tailored to the successive differences archetype. The dataset is augmented with extra columns with the "archetype_" prefix, as well as special attributes to tell downstream functions like brm_formula() what to do with the object.

Prior labeling for brm_archetype_successive_effects()

Within each treatment group, each intercept is labeled by the earliest time point, and each successive difference term gets the successive time point as the label. To illustrate, consider the example in the Details section. In the labeling scheme for brm_archetype_successive_effects(), you can label the prior on beta_1 using brm_prior_label(code = "normal(1.2, 5)", group = "A", time = "1"). Similarly, you cal label the prior on beta_5 with brm_prior_label(code = "normal(1.3, 7)", group = "B", time = "2"). To confirm that you set the prior correctly, compare the brms prior with the output of summary(your_archetype). See the examples for details.

Nuisance variables

In the presence of covariate adjustment, functions like brm_archetype_successive_cells() convert nuisance factors into binary dummy variables, then center all those dummy variables and any continuous nuisance variables at their means in the data. This ensures that the main model coefficients of interest are not implicitly conditional on a subset of the data. In other words, preprocessing nuisance variables this way preserves the interpretations of the fixed effects of interest, and it ensures informative priors can be specified correctly.

Prior labeling

Informative prior archetypes use a labeling scheme to assign priors to fixed effects. How it works:

```
1. First, assign the prior of each parameter a collection
  of labels from the data. This can be done manually or with
  successive calls to [brm_prior_label()].
2. Supply the labeling scheme to [brm_prior_archetype()].
  [brm_prior_archetype()] uses attributes of the archetype
  to map labeled priors to their rightful parameters in the model.
```

For informative prior archetypes, this process is much more convenient and robust than manually calling `brms::set_prior()`. However, it requires an understanding of how the labels of the priors map to parameters in the model. This mapping varies from archetype to archetype, and it is documented in the help pages of archetype-specific functions such as `brm_archetype_successive_cells()`.

See Also

Other informative prior archetypes: `brm_archetype_average_cells()`, `brm_archetype_average_effects()`, `brm_archetype_cells()`, `brm_archetype_effects()`, `brm_archetype_successive_cells()`

Examples

```
set.seed(0L)
data <- brm_simulate_outline(
  n_group = 2,
  n_patient = 100,
  n_time = 4,
  rate_dropout = 0,
  rate_lapse = 0
) |>
  dplyr::mutate(response = rnorm(n = dplyr::n())) |>
  brm_data_change() |>
  brm_simulate_continuous(names = c("biomarker1", "biomarker2")) |>
  brm_simulate_categorical(
    names = c("status1", "status2"),
    levels = c("present", "absent")
  )
dplyr::select(
  data,
  group,
  time,
  patient,
  starts_with("biomarker"),
  starts_with("status")
)
archetype <- brm_archetype_successive_effects(data)
archetype
summary(archetype)
formula <- brm_formula(archetype)
formula
prior <- brm_prior_label(
```

```
    code = "normal(1, 2.2)",
    group = "group_1",
    time = "time_2"
  ) |>
    brm_prior_label("normal(1, 3.3)", group = "group_1", time = "time_3") |>
    brm_prior_label("normal(1, 4.4)", group = "group_1", time = "time_4") |>
    brm_prior_label("normal(2, 2.2)", group = "group_2", time = "time_2") |>
    brm_prior_label("normal(2, 3.3)", group = "group_2", time = "time_3") |>
    brm_prior_label("normal(2, 4.4)", group = "group_2", time = "time_4") |>
    brm_prior_archetype(archetype)
  prior
  class(prior)
  if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
  tmp <- utils::capture.output(
    suppressMessages(
      suppressWarnings(
        model <- brm_model(
          data = archetype,
          formula = formula,
          prior = prior,
          chains = 1,
          iter = 100,
          refresh = 0
        )
      )
    )
  )
  suppressWarnings(print(model))
  brms::prior_summary(model)
  draws <- brm_marginal_draws(
    data = archetype,
    formula = formula,
    model = model
  )
  summaries_model <- brm_marginal_summaries(draws)
  summaries_data <- brm_marginal_data(data)
  brm_plot_compare(model = summaries_model, data = summaries_data)
  }
```

`brm_data` *Create and preprocess an MMRM dataset.*

Description

Create a dataset to analyze with an MMRM.

Usage

```
brm_data(
  data,
```

```
  outcome,
  baseline = NULL,
  group,
  subgroup = NULL,
  time,
  patient,
  covariates = character(0L),
  missing = NULL,
  reference_group,
  reference_subgroup = NULL,
  reference_time = NULL,
  role = NULL,
  level_baseline = NULL,
  level_control = NULL
)
```

Arguments

`data` Data frame or tibble with longitudinal data.

`outcome` Character of length 1, name of the continuous outcome variable. Example possibilities from clinical trial datasets include `"CHG"` and `"AVAL"`. The `outcome` column in the data should be a numeric vector.

`baseline` Character of length 1, name of the baseline response variable (for example, `"BASE"` in many clinical trial datasets). Only relevant if the response variable is change from baseline. Supply `NULL` to ignore or omit.

`group` Character of length 1, name of the treatment group variable. Example possibilities from clinical trial datasets include `"TRT01P"`, `"TREATMENT"`, `"TRT"`, and `"GROUP"`. The `group` column in the data should be a character vector or unordered factor.

`subgroup` Character of length 1, optional name of the a discrete subgroup variable. Set to `NULL` to omit the subgroup (default). If present, the `subgroup` column in the data should be a character vector or unordered factor.

`time` Character of length 1, name of the discrete time variable. Example possibilities from clinical trial datasets include `"AVISIT"` and `"VISIT"`. For most analyses, please ensure the time column in the data is an ordered factor. You can easily turn the time variable into an ordered factor using `brm_data_chronologize()`, either before or immediately after `brm_data()` (but before any `brm_archetype_*()` functions). This ensures the time points sort in chronological order, which ensures the correctness of informative prior archetypes and autoregressive / moving average correlation structures.

Ordinarily, ordered factors automatically use polynomial contrasts from `contr.poly()`. This is undesirable for MMRMs, so if the time variable is an ordered factor, then `brm_data()` manually sets `contrasts(data[[time]])` to a set of treatment contrasts using `contr.treatment()`. If you prefer different contrasts, please manually set `contrasts(data[[time]])` to something else after calling `brm_data()`.

patent
: Character of length 1, name of the patient ID variable. Example possibilities from clinical trial datasets include "USUBJID", "SUBJID", "PATIENT", "PATIENTID", "SUBJECT", "SUBJIDID", "SBJID", "STYSID1A", "SBJ1N", and "ID". The patient column in the data should be a factor or character vector.

covariates
: Character vector of names of other covariates. All these covariates are assumed to be non-time-varying. For time-varying covariates, please manually expand the data to the full grid of patients and time points before you call brm_data(). See the "Preprocessing" section for details.

missing
: Character of length 1, name of an optional variable in a simulated dataset to indicate which outcome values should be missing. Set to NULL to omit.

reference_group
: Atomic value of length 1, Level of the group column to indicate the control group. Example possibilities from clinical trial datasets include "Placebo", "PLACEBO", "PBO", "PLB", "CONTROL", "CTRL", "REFERENCE", and "REF". reference_group only applies to the post-processing that happens in functions like brm_marginal_draws() downstream of the model. It does not control the fixed effect mapping in the model matrix that brms derives from the formula from brm_formula().

reference_subgroup
: Atomic value of length 1, level of the subgroup column to use as a reference for pairwise differences in when computing marginal means downstream of the model. It does not control the fixed effect mapping in the model matrix that brms derives from the formula from brm_formula().

reference_time
: Atomic value of length 1 or NULL, level of the time column to indicate the baseline time point. Leave as NULL if there is no baseline or baseline is not included in data[[time]].

  If reference_time is not NULL, then brm_marginal_draws() will calculate change from baseline, and it will calculate treatment differences as differences between change-from-baseline values. If reference_time is not NULL, then brm_marginal_draws() will not calculate change from baseline, and it will calculate treatment differences as differences between response values.

  Note: reference_time only applies to the post-processing that happens in functions like brm_marginal_draws() downstream of the model. It does not control the fixed effect mapping in the model matrix that brms derives from the formula from brm_formula().

role
: Deprecated as unnecessary on 2024-07-11 (version 1.0.1.9007). Use reference_time to supply a baseline time point value if it exists.

level_baseline
: Deprecated on 2024-01-11 (version 0.2.0.9002). Use reference_time instead.

level_control
: Deprecated on 2024-01-11 (version 0.2.0.9002). Use reference_group instead.

Value

A classed tibble with attributes which denote features of the data such as the treatment group and discrete time variables.

Preprocessing

The preprocessing steps in `brm_data()` are as follows:

- Perform basic assertions to make sure the data and other arguments are properly formatted.
- Convert the group and time columns to character vectors.
- Sanitize the levels of the group and time columns using `make.names(unique = FALSE, allow_ = TRUE)` to ensure agreement between the data and the output of `brms`.
- For each implicitly missing outcome observation, add explicit row with the outcome variable equal to `NA_real_`. Missing values in the predictors are implicitly filled using `zoo::na.locf()` on within each patient, which is not valid for time-varying covariates. If any covariates are time-varying, please manually perform this step before calling `brm_data()`.
- Arrange the rows of the data by group, then patient, then discrete time.
- Select only the columns of the data relevant to an MMRM analysis.

Separation string

Post-processing in `brm_marginal_draws()` names each of the group-by-time marginal means with the delimiting character string from `Sys.getenv("BRM_SEP", unset = "|")`. Neither the column names nor element names of the group and time variables can contain this string. To set a custom string yourself, use `Sys.setenv(BRM_SEP = "YOUR_CUSTOM_STRING")`.

See Also

Other data: `brm_data_change()`, `brm_data_chronologize()`

Examples

```
set.seed(0)
data <- brm_simulate_simple()$data
colnames(data) <- paste0("col_", colnames(data))
data
brm_data(
  data = data,
  outcome = "col_response",
  group = "col_group",
  time = "col_time",
  patient = "col_patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
```

brm_data_change *Convert to change from baseline.*

Description

Convert a dataset from raw response to change from baseline.

Usage

```
brm_data_change(data, name_change = "change", name_baseline = "baseline")
```

Arguments

data	A classed tibble (e.g. from brm_data()) with raw response as the outcome variable and no baseline time point stored in the attributes.
name_change	Character of length 1, name of the new outcome column for change from baseline.
name_baseline	Character of length 1, name of the new column for the original baseline response.

Value

A classed tibble with change from baseline as the outcome variable and the internal attributes modified accordingly. A special baseline column is also created, and the original raw response column is removed. The new baseline column is comprised of the elements of the response variable corresponding to the reference_time argument of brm_data().

If there is a column to denote missing values for simulation purposes, e.g. the "missing" column generated by brm_simulate_outline(), then missing baseline values are propagated accordingly such that change from baseline will be missing if either the post-baseline response is missing or the baseline response is missing.

See Also

Other data: brm_data(), brm_data_chronologize()

Examples

```
set.seed(0)
data <- brm_data(
  data = dplyr::rename(brm_simulate_simple()$data, y_values = response),
  outcome = "y_values",
  group = "group",
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
```

```
data
attr(data, "brm_outcome")
attr(data, "brm_baseline")
attr(data, "brm_reference_time")
changed <- brm_data_change(data = data, name_change = "delta")
changed
attr(changed, "brm_outcome")
attr(changed, "brm_baseline")
attr(data, "brm_reference_time")
```

`brm_data_chronologize` *Chronologize a dataset*

Description

Convert the discrete time variable into an ordered factor.

Usage

```
brm_data_chronologize(
  data,
  order = NULL,
  levels = NULL,
  time = attr(data, "brm_time")
)
```

Arguments

`data`	Data frame or tibble with longitudinal data.
`order`	Optional character string with the name of a variable in the data for ordering the time variable. Either `order` or `levels` must be supplied, but not both together. If `order` is supplied, the levels of `data[[order]]` must have a 1:1 correspondence with those of `data[[time]]`, and `sort(unique(data[[order]]))` must reflect the desired order of the levels of `data[[time]]`. For example, suppose you have a CDISC dataset with categorical time variable `AVISIT` and integer variable `AVISITN`. Then, `brm_data_chronologize(time = "AVISIT", order = "AVISITN")` will turn `AVISIT` into an ordered factor with levels that respect the ordering in `AVISITN`.
`levels`	Optional character vector of levels of `data[[time]]` in chronological order. Used to turn `data[[time]]` into an ordered factor. Either `order` or `levels` must be supplied, but not both together.
`time`	Character string with the name of the discrete time variable in the data. This is the variable that `brm_data_chronologize()` turns into an ordered factor. It needs to be specified explicitly if and only if the `data` argument was not produced by a call to `brm_data()`.

Details

Most MMRMs should use an ordered factor for the time column in the data. This way, individual time points are treated as distinct factor levels for the purposes of fixed effect parameterizations (see the "Contrasts" section), and the explicit ordering ensures that informative prior archetypes and ARMA-like correlation structures are expressed correctly. Without the ordering, problems can arise when character vectors are sorted: e.g. if AVISIT has levels "VISIT1", "VISIT2", ..., "VISIT10", then brms will mistake the the order of scheduled study visits to be "VISIT1", "VISIT10", "VISIT2", ..., which is not chronological.

You can easily turn the time variable into an ordered factor using brm_data_chronologize(). Either supply an explicit character vector of chronologically-ordered factor levels in the levels argument, or supply the name of a time-ordered variable in the order argument.

brm_data_chronologize() can be called either before or just after brm_data(), but in the former case, the discrete time variable needs to be specified explicitly in time argument. And in the latter, brm_data_chronologize() must be called before any of the informative prior archetype functions such as brm_archetype_successive_cells().

Value

A data frame with the time column as an ordered factor.

Contrasts

Ordinarily, ordered factors automatically use polynomial contrasts from contr.poly(). This is undesirable for MMRMs, so if the time variable is an ordered factor, then brm_data() manually sets contrasts(data[[time]]) to a set of treatment contrasts using contr.treatment(). If you prefer different contrasts, please manually set contrasts(data[[time]]) to something else after calling brm_data().

See Also

Other data: brm_data(), brm_data_change()

Examples

```
data <- brm_simulate_outline(n_time = 12, n_patient = 4)
data$AVISIT <- gsub("_0", "_", data$time)
data$AVISITN <- as.integer(gsub("time_", "", data$time))
data[, c("AVISIT", "AVISITN")]
sort(unique(data$AVISIT)) # wrong order
data1 <- brm_data_chronologize(data, time = "AVISIT", order = "AVISITN")
sort(unique(data1$AVISIT)) # correct order
levels <- paste0("time_", seq_len(12))
data2 <- brm_data_chronologize(data, time = "AVISIT", levels = levels)
sort(unique(data2$AVISIT)) # correct order
```

brm_formula *Model formula*

Description

Build a model formula for an MMRM, either for a generic brm_data() dataset or an informative prior archetype.

Usage

```
brm_formula(
  data,
  model_missing_outcomes = FALSE,
  check_rank = TRUE,
  sigma = brms.mmrm::brm_formula_sigma(data = data, check_rank = check_rank),
  correlation = "unstructured",
  autoregressive_order = 1L,
  moving_average_order = 1L,
  residual_covariance_arma_estimation = FALSE,
  ...
)

## Default S3 method:
brm_formula(
  data,
  model_missing_outcomes = FALSE,
  check_rank = TRUE,
  sigma = brms.mmrm::brm_formula_sigma(data = data, check_rank = check_rank),
  correlation = "unstructured",
  autoregressive_order = 1L,
  moving_average_order = 1L,
  residual_covariance_arma_estimation = FALSE,
  intercept = TRUE,
  baseline = !is.null(attr(data, "brm_baseline")),
 baseline_subgroup = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
 baseline_subgroup_time = !is.null(attr(data, "brm_baseline")) && !is.null(attr(data,
    "brm_subgroup")),
  baseline_time = !is.null(attr(data, "brm_baseline")),
  covariates = TRUE,
  group = TRUE,
  group_subgroup = !is.null(attr(data, "brm_subgroup")),
  group_subgroup_time = !is.null(attr(data, "brm_subgroup")),
  group_time = TRUE,
  subgroup = !is.null(attr(data, "brm_subgroup")),
  subgroup_time = !is.null(attr(data, "brm_subgroup")),
  time = TRUE,
```

```
  center = TRUE,
  ...,
  effect_baseline = NULL,
  effect_group = NULL,
  effect_time = NULL,
  interaction_baseline = NULL,
  interaction_group = NULL
)

## S3 method for class 'brms_mmrm_archetype'
brm_formula(
  data,
  model_missing_outcomes = FALSE,
  check_rank = TRUE,
  sigma = brms.mmrm::brm_formula_sigma(data = data, check_rank = check_rank),
  correlation = "unstructured",
  autoregressive_order = 1L,
  moving_average_order = 1L,
  residual_covariance_arma_estimation = FALSE,
  ...,
  warn_ignored = TRUE
)
```

Arguments

data
: A classed data frame from `brm_data()`, or an informative prior archetype from a function like `brm_archetype_successive_cells()`.

model_missing_outcomes
: Logical of length 1, `TRUE` to impute missing outcomes during model fitting as described in the "Imputation during model fitting" section of https://paulbuerkner.com/brms/articles/brms_missings.html. Specifically, if the outcome variable is `y`, then the formula will begin with `y | mi() ~ ...` instead of simply `y ~ ...`. Set to `FALSE` (default) to forgo this kind of imputation and discard missing observations from the data just prior to fitting the model inside `brm_model()`. See https://opensource.nibr.com/bamdd/src/02h_mmrm.html#what-estimand-does-mmrm-addr #nolint to understand the standard assumptions and decisions regarding MMRMs and missing outcomes.

check_rank
: `TRUE` to check the rank of the model matrix and throw an error if rank deficiency is detected. `FALSE` to skip this check. Rank-deficient models may have non-identifiable parameters and it is recommended to choose a full-rank mapping.

sigma
: A formula produced by `brm_formula_sigma()`. The formula is a base R formula with S3 class `"brms_mmrm_formula_sigma"`, and it controls the parameterization of the residual standard deviations `sigma`.

correlation
: Character of length 1, name of the correlation structure. The correlation matrix is a square `T x T` matrix, where `T` is the number of discrete time points in the data. This matrix describes the correlations between time points in the same patient, as modeled in the residuals. Different patients are modeled as independent. The `correlation` argument controls how this matrix is parameter-

ized, and the choices given by brms are listed at https://paulbuerkner.com/brms/reference/autocor-terms.html, and the choice is ultimately encoded in the main body of the output formula through terms like unstru() and arma(), some of which are configurable through arguments autoregressive_order, moving_average_order, and residual_covariance_arma_estimation of brm_formula(). Choices in brms.mmrm:

- "unstructured": the default/recommended option, a fully parameterized covariance matrix with a unique scalar parameter for each unique pair of discrete time points. C.f. https://paulbuerkner.com/brms/reference/unstr.html.
- "autoregressive_moving_average": autoregressive moving average (ARMA), c.f. https://paulbuerkner.com/brms/reference/arma.html.
- "autoregressive": autoregressive (AR), c.f. https://paulbuerkner.com/brms/reference/ar.html.
- "moving_average": moving average (MA), c.f. https://paulbuerkner.com/brms/reference/ma.html.
- "compound_symmetry: compound symmetry, c.f. https://paulbuerkner.com/brms/reference/cosy.html.
- "diagonal": declare independent time points within patients.

autoregressive_order
: Nonnegative integer, autoregressive order for the "autoregressive_moving_average" and "autoregressive" correlation structures.

moving_average_order
: Nonnegative integer, moving average order for the "autoregressive_moving_average" and "moving_average" correlation structures.

residual_covariance_arma_estimation
: TRUE or FALSE, whether to estimate ARMA effects using residual covariance matrices. Directly supplied to the cov argument in brms for "autoregressive_moving_average", "autoregressive", and "moving_average" correlation structures. C.f. https://paulbuerkner.com/brms/reference/arma.html.

...
: Named arguments to specific brm_formula() methods.

intercept
: Logical of length 1. TRUE (default) to include an intercept, FALSE to omit.

baseline
: Logical of length 1. TRUE to include an additive effect for baseline response, FALSE to omit. Default is TRUE if brm_data() previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

baseline_subgroup
: Logical of length 1.

baseline_subgroup_time
: Logical of length 1. TRUE to include baseline-by-subgroup-by-time interaction, FALSE to omit. Default is TRUE if brm_data() previously declared baseline and subgroup variables in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order

to make sure columns are appropriately centered and the underlying model matrix has full rank.

baseline_time
: Logical of length 1. TRUE to include baseline-by-time interaction, FALSE to omit. Default is TRUE if brm_data() previously declared a baseline variable in the dataset. Ignored for informative prior archetypes. For informative prior archetypes, this option should be set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

covariates
: Logical of length 1. TRUE (default) to include any additive covariates declared with the covariates argument of brm_data(), FALSE to omit. For informative prior archetypes, this option is set in functions like brm_archetype_successive_cells() rather than in brm_formula() in order to make sure columns are appropriately centered and the underlying model matrix has full rank.

group
: Logical of length 1. TRUE (default) to include additive effects for treatment groups, FALSE to omit.

group_subgroup
: Logical of length 1. TRUE to include group-by-subgroup interaction, FALSE to omit. Default is TRUE if brm_data() previously declared a subgroup variable in the dataset.

group_subgroup_time
: Logical of length 1. TRUE to include group-by-subgroup-by-time interaction, FALSE to omit. Default is TRUE if brm_data() previously declared a subgroup variable in the dataset.

group_time
: Logical of length 1. TRUE (default) to include group-by-time interaction, FALSE to omit.

subgroup
: Logical of length 1. TRUE to include additive fixed effects for subgroup levels, FALSE to omit. Default is TRUE if brm_data() previously declared a subgroup variable in the dataset.

subgroup_time
: Logical of length 1. TRUE to include subgroup-by-time interaction, FALSE to omit. Default is TRUE if brm_data() previously declared a subgroup variable in the dataset.

time
: Logical of length 1. TRUE (default) to include a additive effect for discrete time, FALSE to omit.

center
: TRUE to center the columns of the model matrix before fitting the model if the model formula includes an intercept term controlled by brms. FALSE to skip centering. Centering usually leads to more computationally efficient sampling in the presence of an intercept, but it changes the interpretation of the intercept parameter if included in the model (as explained in the help file of brms::brmsformula()). Informative prior archetypes always use center = FALSE and use an intercept not controlled by brms.mmrm to ensure the intercept parameter is interpretable and compatible with user-defined priors.

effect_baseline
: Deprecated on 2024-01-16 (version 0.0.2.9002). Use baseline instead.

effect_group
: Deprecated on 2024-01-16 (version 0.0.2.9002). Use group instead.

effect_time
: Deprecated on 2024-01-16 (version 0.0.2.9002). Use time instead.

interaction_baseline
: Deprecated on 2024-01-16 (version 0.0.2.9002). Use baseline_time instead.

interaction_group
: Deprecated on 2024-01-16 (version 0.0.2.9002). Use group_time instead.

warn_ignored
: Set to TRUE to throw a warning if ignored arguments are specified, FALSE otherwise.

Value

An object of class "brmsformula" returned from brms::brmsformula(). It contains the fixed effect mapping, correlation structure, and residual variance structure.

brm_data() formulas

For a brm_data() dataset, brm_formula() builds an R formula for an MMRM based on the details in the data and your choice of mapping. Customize your mapping by toggling on or off the various TRUE/FALSE arguments of brm_formula(), such as intercept, baseline, and group_time. All plausible additive effects, two-way interactions, and three-way interactions can be specified. The following interactions are not supported:

- Any interactions with the concomitant covariates you specified in the covariates argument of brm_data().
- Any interactions which include baseline response and treatment group together. Rationale: in a randomized controlled experiment, baseline and treatment group assignment should be uncorrelated.

Formulas for informative prior archetypes

Functions like brm_archetype_successive_cells() tailor datasets to informative prior archetypes. For these specialized tailored datasets, brm_formula() works differently. It still applies the variance and correlation structure of your choosing, and it still lets you choose whether to adjust for nuisance covariates, but it no longer lets you toggle on/off individual terms in the model, such as intercept, baseline, or group. Instead, to ensure the correct interpretation of the parameters, brm_formula() uses the x_* and nuisance_* columns generated by brm_archetype_successive_cells(prefix_interest = "x_", prefix_nuisance = "nuisance_").

Parameterization

For a formula on a brm_data() dataset, the formula is not the only factor that determines the fixed effect mapping. The ordering of the categorical variables in the data, as well as the contrast option in R, affect the construction of the model matrix. To see the model matrix that will ultimately be used in brm_model(), run brms::make_standata() and examine the X element of the returned list. See the examples below for a demonstration.

See Also

Other models: brm_formula_sigma(), brm_model()

Examples

```
set.seed(0)
data <- brm_data(
  data = brm_simulate_simple()$data,
  outcome = "response",
  group = "group",
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
brm_formula(data)
brm_formula(data = data, intercept = FALSE, baseline = FALSE)
formula <- brm_formula(
  data = data,
  intercept = FALSE,
  baseline = FALSE,
  group = FALSE
)
formula
# Standard deviations of residuals are distributional parameters that can
# regress on variables in the data.
homogeneous <- brm_formula_sigma(data, time = FALSE)
by_group <- brm_formula_sigma(data, group = TRUE, intercept = TRUE)
homogeneous
by_group
brm_formula(data, sigma = homogeneous)
brm_formula(data, sigma = by_group)
# Optional: set the contrast option, which determines the model matrix.
options(contrasts = c(unordered = "contr.SAS", ordered = "contr.poly"))
# See the fixed effect mapping you get from the data:
head(brms::make_standata(formula = formula, data = data)$X)
# Specify a different contrast method to use an alternative
# mapping when fitting the model with brm_model():
options(
  contrasts = c(unordered = "contr.treatment", ordered = "contr.poly")
)
# different model matrix than before:
head(brms::make_standata(formula = formula, data = data)$X)
# Formula on an informative prior archetype:
data <- brm_simulate_outline(
  n_group = 2,
  n_patient = 100,
  n_time = 4,
  rate_dropout = 0,
  rate_lapse = 0
) |>
  dplyr::mutate(response = rnorm(n = dplyr::n())) |>
  brm_data_change() |>
  brm_simulate_continuous(names = c("biomarker1", "biomarker2")) |>
  brm_simulate_categorical(
    names = "biomarker3",
```

```
    levels = c("present", "absent")
  )
archetype <- brm_archetype_successive_cells(data)
formula <- brm_formula(data = archetype)
formula
```

---

brm_formula_sigma   *Formula for standard deviation parameters*

Description

Parameterize standard deviations using a formula for the `sigma` argument of `brm_formula()`.

Usage

```
brm_formula_sigma(
  data,
  check_rank = TRUE,
  intercept = FALSE,
  baseline = FALSE,
  baseline_subgroup = FALSE,
  baseline_subgroup_time = FALSE,
  baseline_time = FALSE,
  covariates = FALSE,
  group = FALSE,
  group_subgroup = FALSE,
  group_subgroup_time = FALSE,
  group_time = FALSE,
  subgroup = FALSE,
  subgroup_time = FALSE,
  time = TRUE
)
```

Arguments

`data` A classed data frame from `brm_data()`, or an informative prior archetype from a function like `brm_archetype_successive_cells()`.

`check_rank` `TRUE` to check the rank of the model matrix for `sigma` and throw an error if rank deficiency is detected. `FALSE` to skip this check. Rank-deficiency may cause `sigma` to be non-identifiable, may prevent the MCMC from converging.

`intercept` Logical of length 1. `TRUE` (default) to include an intercept, `FALSE` to omit.

`baseline` Logical of length 1. `TRUE` to include an additive effect for baseline response, `FALSE` to omit. If `TRUE`, then effect size will be omitted from the output of `brm_marginal_draws()`.

`baseline_subgroup`
Logical of length 1.

baseline_subgroup_time
: Logical of length 1. TRUE to include baseline-by-subgroup-by-time interaction, FALSE to omit. If TRUE, then effect size will be omitted from the output of brm_marginal_draws().

baseline_time
: Logical of length 1. TRUE to include baseline-by-time interaction, FALSE to omit. If TRUE, then effect size will be omitted from the output of brm_marginal_draws().

covariates
: Logical of length 1. TRUE (default) to include any additive covariates declared with the covariates argument of brm_data(), FALSE to omit. If TRUE, then effect size will be omitted from the output of brm_marginal_draws().

group
: Logical of length 1. TRUE (default) to include additive effects for treatment groups, FALSE to omit.

group_subgroup
: Logical of length 1. TRUE to include group-by-subgroup interaction, FALSE to omit.

group_subgroup_time
: Logical of length 1. TRUE to include group-by-subgroup-by-time interaction, FALSE to omit.

group_time
: Logical of length 1.

subgroup
: Logical of length 1. TRUE to include additive fixed effects for subgroup levels, FALSE to omit.

subgroup_time
: Logical of length 1. TRUE to include subgroup-by-time interaction, FALSE to omit.

time
: Logical of length 1.

Details

In brms, the standard deviations of the residuals are modeled through a parameter vector called sigma. brms.mmrm always treats sigma as a distributional parameter (https://paulbuerkner.com/brms/articles/brms_distreg.html). brm_formula_sigma() lets you control the parameterization of sigma. The output of brm_formula_sigma() serves as input to the sigma argument of brm_formula().

The default sigma formula is sigma ~ 0 + time, where time is the discrete time variable in the data. This is the usual heterogeneous variance structure which declares one standard deviation parameter for each time point in the data. Alternatively, you could write brm_formula_sigma(data, intercept = TRUE, time = FALSE). This will produce sigma ~ 1, which yields a single scalar variance (a structure termed "homogeneous variance").

With arguments like baseline and covariates, you can specify extremely complicated variance structures. However, if baseline or covariates are used, then the output of brm_marginal_draws() omit effect size due to the statistical challenges of calculating marginal means of draws of sigma for this uncommon scenario.

Value

A base R formula with S3 class "brms_mmrm_formula_sigma". This formula controls the parameterization of sigma, the linear-scale brms distributional parameters which represent standard deviations.

See Also

Other models: brm_formula(), brm_model()

Examples

```
set.seed(0)
data <- brm_data(
  data = brm_simulate_simple()$data,
  outcome = "response",
  group = "group",
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
homogeneous <- brm_formula_sigma(data, time = FALSE, intercept = TRUE)
by_group <- brm_formula_sigma(data, group = TRUE, intercept = TRUE)
homogeneous
by_group
brm_formula(data, sigma = homogeneous)
brm_formula(data, sigma = by_group)
```

---

brm_marginal_data *Marginal summaries of the data.*

Description

Marginal summaries of the data.

Usage

```
brm_marginal_data(
  data,
  level = 0.95,
  use_subgroup = !is.null(attr(data, "brm_subgroup"))
)
```

Arguments

data	A classed data frame from brm_data(), or an informative prior archetype from a function like brm_archetype_successive_cells().
level	Numeric of length 1 from 0 to 1, level of the confidence intervals.
use_subgroup	Logical of length 1, whether to summarize the data by each subgroup level.

Value

A tibble with one row per summary statistic and the following columns:

- group: treatment group.
- subgroup: subgroup level. Only included if the subgroup argument of brm_marginal_data() is TRUE.
- time: discrete time point.
- statistic: type of summary statistic.
- value: numeric value of the estimate.

The statistic column has the following possible values:

- mean: observed mean response after removing missing values.
- median: observed median response after removing missing values.
- sd: observed standard deviation of the response after removing missing values.
- lower: lower bound of a normal equal-tailed confidence interval with confidence level determined by the level argument.
- upper: upper bound of a normal equal-tailed confidence interval with confidence level determined by the level argument.
- n_observe: number of non-missing values in the response.
- n_total: number of total records in the data for the given group/time combination, including both observed and missing values.

See Also

Other marginals: brm_marginal_draws(), brm_marginal_draws_average(), brm_marginal_grid(), brm_marginal_probabilities(), brm_marginal_summaries()

Examples

```
set.seed(0L)
data <- brm_data(
  data = brm_simulate_simple()$data,
  outcome = "response",
  group = "group",
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
brm_marginal_data(data = data)
```

brm_marginal_draws *MCMC draws from the marginal posterior of an MMRM*

Description

Get marginal posterior draws from a fitted MMRM.

Usage

```
brm_marginal_draws(
  model,
  data = model$brms.mmrm_data,
  formula = model$brms.mmrm_formula,
  transform = brms.mmrm::brm_transform_marginal(data = data, formula = formula,
    average_within_subgroup = average_within_subgroup),
  effect_size = attr(formula, "brm_allow_effect_size"),
  average_within_subgroup = NULL,
  use_subgroup = NULL,
  control = NULL,
  baseline = NULL
)
```

Arguments

model	A fitted model object from brm_model().
data	A classed data frame from brm_data(), or an informative prior archetype from a function like brm_archetype_successive_cells().
formula	An object of class "brmsformula" from brm_formula() or brms::brmsformula(). Should include the full mapping of the model, including fixed effects, residual correlation, and heterogeneity in the discrete-time-specific residual variance components.
transform	Matrix with one row per marginal mean and one column per model parameter. brm_marginal_draws() uses this matrix to map posterior draws of model parameters to posterior draws of marginal means using matrix multiplication. Please use brm_transform_marginal() to compute this matrix and then modify only if necessary. See the methods vignettes for details on this matrix, as well as how brms.mmrm computes marginal means more generally.
effect_size	Logical, TRUE to derive posterior samples of effect size (treatment effect divided by residual standard deviation). FALSE to omit. brms.mmrm does not support effect size when baseline or covariates are included in the brm_formula_sigma() formula. If effect_size is TRUE in this case, then brm_marginal_draws() will automatically omit effect size and throw an informative warning.
average_within_subgroup	TRUE, FALSE, or NULL to control whether nuisance parameters are averaged within subgroup levels in brm_transform_marginal(). Ignored if the transform argument is manually supplied by the user. See the help page of brm_transform_marginal() for details on the average_within_subgroup argument.

use_subgroup	Deprecated. No longer used. brm_marginal_draws() no longer marginalizes over the subgroup declared in brm_data(). To marginalize over the subgroup, declare that variable in covariates instead.
control	Deprecated. Set the control group level in brm_data().
baseline	Deprecated. Set the control group level in brm_data().

Value

A named list of tibbles of MCMC draws of the marginal posterior distribution of each treatment group and time point. These marginals are also subgroup-specific if brm_formula() included fixed effects that use the subgroup variable originally declared in brm_data(). In each tibble, there is 1 row per posterior sample and one column for each type of marginal distribution (i.e. each combination of treatment group and discrete time point. The specific tibbles in the returned list are described below:

- response: on the scale of the response variable.
- difference_time: change from baseline: the response at a particular time minus the response at baseline (reference_time). Only returned if the reference_time argument of brm_data() was not NULL (i.e. if a baseline value for the time variable was identified).
- difference_group: treatment effect: These samples depend on the values of reference_group and reference_time which were originally declared in brm_data(). reference_group is the control group, and reference_time is baseline. If baseline was originally given (via reference_time in brm_data()), then difference_time is the change-from-baseline value of each active group minus that of the control group. Otherwise, if baseline is omitted (i.e. reference_time = NULL (default) in brm_data()), then difference_time is the raw response at each active group minus that of the control group.
- difference_subgroup: subgroup differences: the difference_group at each subgroup level minus the difference_group at the subgroup reference level (reference_subgroup). Only reported if a subgroup analysis was specified through the appropriate arguments to brm_data() and brm_formula().
- effect: effect size, defined as the treatment difference divided by the residual standard deviation. Omitted if the effect_size argument is FALSE or if the brm_formula_sigma() includes baseline or covariates.
- sigma: posterior draws of linear-scale marginal standard deviations of residuals. Omitted if the effect_size argument is FALSE or if the brm_formula_sigma() includes baseline or covariates.

Baseline

The returned values from brm_marginal_draws() depend on whether a baseline time point was declared through the reference_time argument of brm_data(). If reference_time was not NULL, then brm_marginal_draws() will calculate change from baseline, and it will calculate treatment differences as differences between change-from-baseline values. If reference_time was not NULL, then brm_marginal_draws() will not calculate change from baseline, and it will calculate treatment differences as differences between response values.

Separation string

Post-processing in brm_marginal_draws() names each of the group-by-time marginal means with the delimiting character string from Sys.getenv("BRM_SEP", unset = "|"). Neither the column names nor element names of the group and time variables can contain this string. To set a custom string yourself, use Sys.setenv(BRM_SEP = "YOUR_CUSTOM_STRING").

See Also

Other marginals: brm_marginal_data(), brm_marginal_draws_average(), brm_marginal_grid(), brm_marginal_probabilities(), brm_marginal_summaries()

Examples

```
if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
set.seed(0L)
data <- brm_data(
  data = brm_simulate_simple()$data,
  outcome = "response",
  group = "group",
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
formula <- brm_formula(
  data = data,
  baseline = FALSE,
  baseline_time = FALSE
)
tmp <- utils::capture.output(
  suppressMessages(
    suppressWarnings(
      model <- brm_model(
        data = data,
        formula = formula,
        chains = 1,
        iter = 100,
        refresh = 0
      )
    )
  )
)
brm_marginal_draws(data = data, formula = formula, model = model)
}
```

---

brm_marginal_draws_average

Average marginal MCMC draws across time points.

Description

Simple un-weighted arithmetic mean of marginal MCMC draws across time points.

Usage

```
brm_marginal_draws_average(draws, data, times = NULL, label = "average")
```

Arguments

draws	List of posterior draws from brm_marginal_draws().
data	A classed data frame from brm_data(), or an informative prior archetype from a function like brm_archetype_successive_cells().
times	Character vector of discrete time point levels over which to average the MCMC samples within treatment group levels. Set to NULL to average across all time points. Levels are automatically sanitized with make.names(unique = FALSE, allow_ = TRUE) to ensure agreement with brms variable names in downstream computations.
label	Character of length 1, time point label for the averages. Automatically sanitized with make.names(unique = FALSE, allow_ = TRUE). Must not conflict with any existing time point labels in the data after the label and time points are sanitized.

Value

A named list of tibbles of MCMC draws of the marginal posterior distribution of each treatment group and time point (or group-by-subgroup-by-time, if applicable). See brm_marginal_draws() for the full details of the return value. The only difference is that brm_marginal_draws_average() returns a single pseudo-time-point to represent the average across multiple real time points.

Separation string

Post-processing in brm_marginal_draws() names each of the group-by-time marginal means with the delimiting character string from Sys.getenv("BRM_SEP", unset = "|"). Neither the column names nor element names of the group and time variables can contain this string. To set a custom string yourself, use Sys.setenv(BRM_SEP = "YOUR_CUSTOM_STRING").

See Also

Other marginals: brm_marginal_data(), brm_marginal_draws(), brm_marginal_grid(), brm_marginal_probabilities(), brm_marginal_summaries()

Examples

```
if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
set.seed(0L)
data <- brm_data(
  data = brm_simulate_simple()$data,
  outcome = "response",
  group = "group",
```

```
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
formula <- brm_formula(
  data = data,
  baseline = FALSE,
  baseline_time = FALSE
)
tmp <- utils::capture.output(
  suppressMessages(
    suppressWarnings(
      model <- brm_model(
        data = data,
        formula = formula,
        chains = 1,
        iter = 100,
        refresh = 0
      )
    )
  )
)
draws <- brm_marginal_draws(data = data, formula = formula, model = model)
brm_marginal_draws_average(draws = draws, data = data)
brm_marginal_draws_average(
  draws = draws,
  data = data,
  times = c("time_1", "time_2"),
  label = "mean"
)
}
```

`brm_marginal_grid` *Marginal names grid.*

Description

Describe the column names of the data frames output by `brm_marginal_draws()`.

Usage

```
brm_marginal_grid(data, formula)
```

Arguments

data	A classed data frame from `brm_data()`, or an informative prior archetype from a function like `brm_archetype_successive_cells()`.

formula — An object of class "brmsformula" from brm_formula() or brms::brmsformula(). Should include the full mapping of the model, including fixed effects, residual correlation, and heterogeneity in the discrete-time-specific residual variance components.

Details

Useful for creating custom posterior summaries from the draws.

Value

A data frame with a name column with the names of columns of data frames in brm_marginal_draws(), along with metadata to describe which groups, subgroups, and time points those columns correspond to.

See Also

Other marginals: brm_marginal_data(), brm_marginal_draws(), brm_marginal_draws_average(), brm_marginal_probabilities(), brm_marginal_summaries()

Examples

```
data <- brm_simulate_outline()
brm_marginal_grid(data, brm_formula(data))
data <- brm_simulate_outline(n_subgroup = 2L)
brm_marginal_grid(data, brm_formula(data))
```

---

brm_marginal_probabilities

Marginal probabilities on the treatment effect for an MMRM.

Description

Marginal probabilities on the treatment effect for an MMRM.

Usage

```
brm_marginal_probabilities(draws, direction = "greater", threshold = 0)
```

Arguments

draws — Posterior draws of the marginal posterior obtained from brm_marginal_draws().

direction — Character vector of the same length as threshold. "greater" to compute the marginal posterior probability that the treatment effect is greater than the threshold, "less" to compute the marginal posterior probability that the treatment effect is less than the threshold. Each element direction[i] corresponds to threshold[i] for all i from 1 to length(direction).

threshold — Numeric vector of the same length as direction, treatment effect threshold for computing posterior probabilities. Each element direction[i] corresponds to threshold[i] for all i from 1 to length(direction).

Value

A tibble of probabilities of the form Prob(treatment effect > threshold | data) and/or Prob(treatment effect < threshold | data). It has one row per probability and the following columns: * group: treatment group. * subgroup: subgroup level, if applicable. * time: discrete time point, * direction: direction of the comparison in the marginal probability: "greater" for >, "less" for < * threshold: treatment effect threshold in the probability statement. * value: numeric value of the estimate of the probability.

See Also

Other marginals: brm_marginal_data(), brm_marginal_draws(), brm_marginal_draws_average(), brm_marginal_grid(), brm_marginal_summaries()

Examples

```
if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
set.seed(0L)
data <- brm_data(
  data = brm_simulate_simple()$data,
  outcome = "response",
  group = "group",
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
formula <- brm_formula(
  data = data,
  baseline = FALSE,
  baseline_time = FALSE
)
tmp <- utils::capture.output(
  suppressMessages(
    suppressWarnings(
      model <- brm_model(
        data = data,
        formula = formula,
        chains = 1,
        iter = 100,
        refresh = 0
      )
    )
  )
)
draws <- brm_marginal_draws(data = data, formula = formula, model = model)
brm_marginal_probabilities(draws, direction = "greater", threshold = 0)
}
```

brm_marginal_summaries *Summary statistics of the marginal posterior of an MMRM.*

Description

Summary statistics of the marginal posterior of an MMRM.

Usage

```
brm_marginal_summaries(draws, level = 0.95)
```

Arguments

draws	Posterior draws of the marginal posterior obtained from brm_marginal_draws().
level	Numeric of length 1 between 0 and 1, credible level for the credible intervals.

Value

A tibble with one row per summary statistic and the following columns:

- marginal: type of marginal distribution. If outcome was "response" in brm_marginal_draws(), then possible values include "response" for the response on the raw scale, "change" for change from baseline, and "difference" for treatment difference in terms of change from baseline. If outcome was "change", then possible values include "response" for the response one the change from baseline scale and "difference" for treatment difference.
- statistic: type of summary statistic. "lower" and "upper" are bounds of an equal-tailed quantile-based credible interval.
- group: treatment group.
- subgroup: subgroup level, if applicable.
- time: discrete time point.
- value: numeric value of the estimate.
- mcse: Monte Carlo standard error of the estimate. The statistic column has the following possible values:
- mean: posterior mean.
- median: posterior median.
- sd: posterior standard deviation of the mean.
- lower: lower bound of an equal-tailed credible interval of the mean, with credible level determined by the level argument.
- upper: upper bound of an equal-tailed credible interval with credible level determined by the level argument.

See Also

Other marginals: brm_marginal_data(), brm_marginal_draws(), brm_marginal_draws_average(), brm_marginal_grid(), brm_marginal_probabilities()

Examples

```
if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
set.seed(0L)
data <- brm_data(
  data = brm_simulate_simple()$data,
  outcome = "response",
  group = "group",
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
formula <- brm_formula(
  data = data,
  baseline = FALSE,
  baseline_time = FALSE
)
tmp <- utils::capture.output(
  suppressMessages(
    suppressWarnings(
      model <- brm_model(
        data = data,
        formula = formula,
        chains = 1,
        iter = 100,
        refresh = 0
      )
    )
  )
)
draws <- brm_marginal_draws(data = data, formula = formula, model = model)
suppressWarnings(brm_marginal_summaries(draws))
}
```

---

brm_model *Fit an MMRM.*

Description

Fit an MMRM model using brms.

Usage

```
brm_model(
  data,
  formula,
  ...,
  prior = NULL,
  family = brms::brmsfamily(family = "gaussian"),
  imputed = NULL
)
```

Arguments

data A classed data frame from brm_data(), or an informative prior archetype from a function like brm_archetype_successive_cells(). Unless you supplied model_missing_outcomes = TRUE in brm_formula(), brm_model() automatically rows with missing outcomes just prior to fitting the model with brms::brm(). The brms.mmrm_data attribute in the output object is always the version of the data prior to removing these rows. See the data element of the returned brms object for the final data actually supplied to the model.

If you supply a non-NULL value for the imputed argument, then the data argument is ignored and the MMRM is fit successively to each dataset in imputed using brms::brm_multiple(). Posterior draws are combined automatically for downstream post-processing unless you set combine = FALSE in brm_model().

formula An object of class "brmsformula" from brm_formula() or brms::brmsformula(). Should include the full mapping of the model, including fixed effects, residual correlation, and heterogeneity in the discrete-time-specific residual variance components.

... Arguments to brms::brm() or brms::brm_multiple() other than data, formula, prior, and family.

prior Either NULL for default priors or a "brmsprior" object from brms::prior().

family A brms family object generated by brms::brmsfamily(). Must fit a continuous outcome variable and have the identity link.

imputed Either NULL (default), list of datasets generated with multiple imputation, or a "mids" object from the mice package. The rbmi package may offer a more appropriate method for imputation for MMRMs than mice. It is your responsibility to choose an imputation method appropriate for the data and model.

If not NULL, then the MMRM is fit successively to each dataset in imputed using brms::brm_multiple(). Posterior draws are combined automatically for downstream post-processing unless you set combine = FALSE in brm_model(), so everything at the level of brm_marginal_draws() will be exactly the same as a non-imputation workflow.

Even if you supply imputed, please also supply the original non-imputed dataset in the data argument to help with downstream post-processing.

Value

A fitted model object from brms, with new list elements brms.mmrm_data and brms.mmrm_formula to capture the data and formula supplied to brm_model(). See the explanation of the data argument for how the data is handled and how it relates to the data returned in the brms.mmrm_data attribute.

Parameterization

For a formula on a brm_data() dataset, the formula is not the only factor that determines the fixed effect mapping. The ordering of the categorical variables in the data, as well as the contrast option in R, affect the construction of the model matrix. To see the model matrix that will ultimately be used in brm_model(), run brms::make_standata() and examine the X element of the returned list. See the examples below for a demonstration.

See Also

Other models: brm_formula(), brm_formula_sigma()

Examples

```
if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
set.seed(0L)
data <- brm_data(
  data = brm_simulate_simple()$data,
  outcome = "response",
  group = "group",
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
formula <- brm_formula(
  data = data,
  baseline = FALSE,
  baseline_time = FALSE
)
# Optional: set the contrast option, which determines the model matrix.
options(contrasts = c(unordered = "contr.SAS", ordered = "contr.poly"))
# See the fixed effect mapping you get from the data:
head(brms::make_standata(formula = formula, data = data)$X)
# Specify a different contrast method to use an alternative
# mapping when fitting the model with brm_model():
options(
  contrasts = c(unordered = "contr.treatment", ordered = "contr.poly")
)
# different model matrix than before:
head(brms::make_standata(formula = formula, data = data)$X)
tmp <- utils::capture.output(
  suppressMessages(
    suppressWarnings(
      model <- brm_model(
        data = data,
```

```
        formula = formula,
        chains = 1,
        iter = 100,
        refresh = 0
      )
    )
  )
)
# The output is a brms model fit object with added list
# elements "brms.mmrm_data" and "brms.mmrm_formula" to track the dataset
# and formula used to fit the model.
model$brms.mmrm_data
model$brms.mmrm_formula
# Otherwise, the fitted model object acts exactly like a brms fitted model.
suppressWarnings(print(model))
brms::prior_summary(model)
}
```

`brm_plot_compare` *Visually compare the marginals of multiple models and/or datasets.*

Description

Visually compare the marginals of multiple models and/or datasets.

Usage

```
brm_plot_compare(
  ...,
  marginal = "response",
  compare = "source",
  axis = "time",
  facet = c("group", "subgroup")
)
```

Arguments

`...`	Named `tibble`s of marginals posterior summaries from `brm_marginal_summaries()` and/or `brm_marginal_data()`.
`marginal`	Character of length 1, which kind of marginal to visualize. Must be a value in the `marginal` column of the supplied `tibble`s in the `...` argument. Only applies to MCMC output, the data is always on the scale of the response variable.
`compare`	Character of length 1 identifying the variable to display using back-to-back interval plots of different colors. This is the primary comparison of interest. Must be one of `"source"` (the source of the marginal summaries, e.g. a model or dataset), `"time"` or `"group"` (in the non-subgroup case). Can also be `"subgroup"` if all the marginal summaries are subgroup-specific. The value must not be in `axis` or `facet`.

axis
: Character of length 1 identifying the quantity to put on the horizontal axis. Must be be one of "source" (the source of the marginal summaries, e.g. a model or dataset), "time", or "group" (in the non-subgroup case). If the marginals are subgroup-specific, then axis can also be "subgroup". The value must not be in compare or facet.

facet
: Character vector of length 1 or 2 with quantities to generate facets. Each element must be "source" (the source of the marginal summaries, e.g. a model or dataset), "time", "group", or "subgroup", and c(axis, facet) must all have unique elements. "subgroup" is automatically removed if not all the marginal summaries have a subgroup column. If facet has length 1, then faceting is wrapped. If facet has length 2, then faceting is in a grid, and the first element is horizontal facet.

Details

By default, brm_plot_compare() compares multiple models and/or datasets side-by-side. The compare argument selects the primary comparison of interest, and arguments axis and facet control the arrangement of various other components of the plot. The subgroup variable is automatically included if and only if all the supplied marginal summaries have a subgroup column.

Value

A ggplot object.

See Also

Other visualization: brm_plot_draws()

Examples

```
if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
set.seed(0L)
data <- brm_data(
  data = brm_simulate_simple()$data,
  outcome = "response",
  group = "group",
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
formula <- brm_formula(
  data = data,
  baseline = FALSE,
  baseline_time = FALSE
)
tmp <- utils::capture.output(
  suppressMessages(
    suppressWarnings(
      model <- brm_model(
        data = data,
```

```
        formula = formula,
        chains = 1,
        iter = 100,
        refresh = 0
      )
    )
  )
)
draws <- brm_marginal_draws(data = data, formula = formula, model = model)
suppressWarnings(summaries_draws <- brm_marginal_summaries(draws))
summaries_data <- brm_marginal_data(data)
brm_plot_compare(
  model1 = summaries_draws,
  model2 = summaries_draws,
  data = summaries_data
)
brm_plot_compare(
  model1 = summaries_draws,
  model2 = summaries_draws,
  marginal = "difference"
)
}
```

`brm_plot_draws` *Visualize posterior draws of marginals.*

Description

Visualize posterior draws of marginals.

Usage

```
brm_plot_draws(draws, axis = "time", facet = c("group", "subgroup"))
```

Arguments

`draws`	A data frame of draws from an element of the output list of `brm_marginal_summaries()`.
`axis`	Character of length 1 identifying the quantity to put on the horizontal axis. Must be be one of `"time"` or `"group"` if the marginal summaries are not subgroup-specific. If the marginals are subgroup-specific, then `axis` must be one of `"time"`, `"group"`, or `"subgroup"`.
`facet`	Character vector of length 1 or 2 with quantities to generate facets. Each element must be `"time"`, `"group"`, or `"subgroup"`, and `c(axis, facet)` must all have unique elements. `"subgroup"` is automatically removed if the marginals have no subgroup. If `facet` has length 1, then faceting is wrapped. If `facet` has length 2, then faceting is in a grid, and the first element is horizontal facet.

Value

A ggplot object.

See Also

Other visualization: brm_plot_compare()

Examples

```
if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
set.seed(0L)
data <- brm_data(
  data = brm_simulate_simple()$data,
  outcome = "response",
  group = "group",
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
formula <- brm_formula(
  data = data,
  baseline = FALSE,
  baseline_time = FALSE
)
tmp <- utils::capture.output(
  suppressMessages(
    suppressWarnings(
      model <- brm_model(
        data = data,
        formula = formula,
        chains = 1,
        iter = 100,
        refresh = 0
      )
    )
  )
)
draws <- brm_marginal_draws(data = data, formula = formula, model = model)
brm_plot_draws(draws = draws$difference_time)
}
```

---

brm_prior_archetype *Informative priors for fixed effects in archetypes*

Description

Create a brms prior for fixed effects in an archetype.

Usage

```
brm_prior_archetype(label, archetype)
```

Arguments

label	A data frame with one row per model parameter in the archetype and columns to indicate the mapping between priors and labels. Generate using `brm_prior_label()` or manually. See the examples and the informative prior archetypes vignette for details.
archetype	An informative prior archetype generated by a function like `brm_archetype_successive_cells()`.

Value

A `brms` prior object that you can supply to the `prior` argument of `brm_model()`.

Prior labeling

Informative prior archetypes use a labeling scheme to assign priors to fixed effects. How it works:

```
1. First, assign the prior of each parameter a collection
  of labels from the data. This can be done manually or with
  successive calls to [brm_prior_label()].
2. Supply the labeling scheme to [brm_prior_archetype()].
  [brm_prior_archetype()] uses attributes of the archetype
  to map labeled priors to their rightful parameters in the model.
```

For informative prior archetypes, this process is much more convenient and robust than manually calling `brms::set_prior()`. However, it requires an understanding of how the labels of the priors map to parameters in the model. This mapping varies from archetype to archetype, and it is documented in the help pages of archetype-specific functions such as `brm_archetype_successive_cells()`.

See Also

Other priors: `brm_prior_label()`, `brm_prior_simple()`, `brm_prior_template()`

Examples

```
set.seed(0L)
data <- brm_simulate_outline(
  n_group = 2,
  n_patient = 100,
  n_time = 3,
  rate_dropout = 0,
  rate_lapse = 0
) |>
  dplyr::mutate(response = rnorm(n = dplyr::n())) |>
  brm_simulate_continuous(names = c("biomarker1", "biomarker2")) |>
  brm_simulate_categorical(
    names = c("status1", "status2"),
    levels = c("present", "absent")
```

```
  )
archetype <- brm_archetype_successive_cells(data)
dplyr::distinct(data, group, time)
prior <- NULL |>
  brm_prior_label("normal(1, 1)", group = "group_1", time = "time_1") |>
  brm_prior_label("normal(1, 2)", group = "group_1", time = "time_2") |>
  brm_prior_label("normal(1, 3)", group = "group_1", time = "time_3") |>
  brm_prior_label("normal(2, 1)", group = "group_2", time = "time_1") |>
  brm_prior_label("normal(2, 2)", group = "group_2", time = "time_2") |>
  brm_prior_label("normal(2, 3)", group = "group_2", time = "time_3") |>
  brm_prior_archetype(archetype = archetype)
prior
class(prior)
```

`brm_prior_label` *Label a prior with levels in the data.*

Description

Label an informative prior for a parameter using a collection of levels in the data.

Usage

```
brm_prior_label(label = NULL, code, group, subgroup = NULL, time)
```

Arguments

`label`	A `tibble` with the prior labeling scheme so far, with one row per model parameter and columns for the Stan code, treatment group, subgroup, and discrete time point of each parameter.
`code`	Character of length 1, Stan code for the prior. Could be a string like `"normal(1, 2.2)"`. The full set of priors is given in the Stan Function Reference at https://mc-stan.org/docs/functions-reference/ in the "Unbounded Continuous Distributions" section. See the documentation `brms::set_prior()` for more details.
`group`	Value of length 1, level of the treatment group column in the data to label the prior. The treatment group column is the one you identified with the `group` argument of `brm_data()`.
`subgroup`	Value of length 1, level of the subgroup column in the data to label the prior. The subgroup column is the one you identified with the `subgroup` argument of `brm_data()`, if applicable. Not every dataset has a subgroup variable. If yours does not, please either ignore this argument or set it to `NULL`.
`time`	Value of length 1, level of the discrete time column in the data to label the prior. The discrete time column is the one you identified with the `time` argument of `brm_data()`.

Value

A tibble with one row per model parameter and columns for the Stan code, treatment group, subgroup, and discrete time point of each parameter. You can supply this tibble to the label argument of brm_prior_archetype().

Prior labeling

Informative prior archetypes use a labeling scheme to assign priors to fixed effects. How it works:

```
1. First, assign the prior of each parameter a collection
  of labels from the data. This can be done manually or with
  successive calls to [brm_prior_label()].
2. Supply the labeling scheme to [brm_prior_archetype()].
  [brm_prior_archetype()] uses attributes of the archetype
  to map labeled priors to their rightful parameters in the model.
```

For informative prior archetypes, this process is much more convenient and robust than manually calling brms::set_prior(). However, it requires an understanding of how the labels of the priors map to parameters in the model. This mapping varies from archetype to archetype, and it is documented in the help pages of archetype-specific functions such as brm_archetype_successive_cells().

See Also

Other priors: brm_prior_archetype(), brm_prior_simple(), brm_prior_template()

Examples

```
set.seed(0L)
data <- brm_simulate_outline(
  n_group = 2,
  n_patient = 100,
  n_time = 3,
  rate_dropout = 0,
  rate_lapse = 0
) |>
  dplyr::mutate(response = rnorm(n = dplyr::n())) |>
  brm_simulate_continuous(names = c("biomarker1", "biomarker2")) |>
  brm_simulate_categorical(
    names = c("status1", "status2"),
    levels = c("present", "absent")
  )
archetype <- brm_archetype_successive_cells(data)
dplyr::distinct(data, group, time)
label <- NULL |>
  brm_prior_label("normal(1, 1)", group = "group_1", time = "time_1") |>
  brm_prior_label("normal(1, 2)", group = "group_1", time = "time_2") |>
  brm_prior_label("normal(1, 3)", group = "group_1", time = "time_3") |>
  brm_prior_label("normal(2, 1)", group = "group_2", time = "time_1") |>
  brm_prior_label("normal(2, 2)", group = "group_2", time = "time_2") |>
  brm_prior_label("normal(2, 3)", group = "group_2", time = "time_3")
label
```

brm_prior_simple *Simple prior for a* brms *MMRM*

Description

Generate a simple prior for a brms MMRM.

Usage

```
brm_prior_simple(
  data,
  formula,
  intercept = "student_t(3, 0, 2.5)",
  coefficients = "student_t(3, 0, 2.5)",
  sigma = "student_t(3, 0, 2.5)",
  unstructured = "lkj(1)",
  autoregressive = "",
  moving_average = "",
  compound_symmetry = "",
  correlation = NULL
)
```

Arguments

data	A classed data frame from brm_data(), or an informative prior archetype from a function like brm_archetype_successive_cells().
formula	An object of class "brmsformula" from brm_formula() or brms::brmsformula(). Should include the full mapping of the model, including fixed effects, residual correlation, and heterogeneity in the discrete-time-specific residual variance components.
intercept	Character of length 1, Stan code for the prior to set on the intercept parameter.
coefficients	Character of length 1, Stan code for the prior to set independently on each of the non-intercept model coefficients.
sigma	Character of length 1, Stan code for the prior to set independently on each of the log-scale standard deviation parameters. Should be a symmetric prior in most situations.
unstructured	Character of length 1, Stan code for an unstructured correlation prior. Supply the empty string "" to set a flat prior (default). Applies to the "cortime parameter class in brms priors. Used for formulas created with brm_formula(correlation = "unstructured"). LKJ is recommended. See also brms::unstr().
autoregressive	Character of length 1, Stan code for a prior on autoregressive correlation parameters. Supply the empty string "" to set a flat prior (default). Applies to the "ar parameter class in brms priors. Used for formulas created with brm_formula(correlation = "autoregressive") and brm_formula(correlation = "autoregressive_moving_average"). See also brms::ar() and brms::arma().

moving_average Character of length 1, Stan code for a prior on moving average correlation parameters. Supply the empty string "" to set a flat prior (default). Applies to the "ma parameter class in brms priors. Used for formulas created with brm_formula(correlation = "moving_average") and brm_formula(correlation = "autoregressive_moving_average"). See also brms::ma() and brms::arma().

compound_symmetry
Character of length 1, Stan code for a prior on compound symmetry correlation parameters. Supply the empty string "" to set a flat prior (default). Applies to the "cosy parameter class in brms priors. Used for formulas created with brm_formula(correlation = "compound_symmetry"). See also brms::cosy().

correlation Deprecated on 2024-04-22 (version 0.1.0.9004). Please use arguments like "unstructured", and/or "autoregressive" to supply correlation-specific priors.

Details

In brm_prior_simple(), you can separately choose priors for the intercept, model coefficients, log-scale standard deviations, and pairwise correlations between time points within patients. However, each class of parameters is set as a whole. In other words, brm_prior_simple() cannot assign different priors to different fixed effect parameters.

Value

A classed data frame with the brms prior.

See Also

Other priors: brm_prior_archetype(), brm_prior_label(), brm_prior_template()

Examples

```
set.seed(0L)
data <- brm_simulate_outline()
data <- brm_simulate_continuous(data, names = c("age", "biomarker"))
formula <- brm_formula(
  data = data,
  baseline = FALSE,
  baseline_time = FALSE,
  check_rank = FALSE
)
brm_prior_simple(
  data = data,
  formula = formula,
  intercept = "student_t(3, 0, 2.5)",
  coefficients = "normal(0, 10)",
  sigma = "student_t(2, 0, 4)",
  unstructured = "lkj(2.5)"
)
```

brm_prior_template *Label template for informative prior archetypes*

Description

Template for the label argument of brm_prior_archetype().

Usage

```
brm_prior_template(archetype)
```

Arguments

archetype	An informative prior archetype generated by a function like brm_archetype_successive_cells().

Details

The label argument of brm_prior_archetype() is a tibble which maps Stan code for univariate priors to fixed effect parameters in the model. Usually this tibble is built gradually using multiple calls to brm_prior_label(), but occasionally it is more convenient to begin with a full template and manually write Stan code in the code column. brm_prior_template() creates this template.

Value

A tibble with one row per fixed effect parameter and columns to map Stan code to each parameter. After manually writing Stan code in the code column of the template, you can supply the result to the label argument of brm_prior_archetype() to build a brms prior for your model.

Prior labeling

Informative prior archetypes use a labeling scheme to assign priors to fixed effects. How it works:

```
1. First, assign the prior of each parameter a collection
  of labels from the data. This can be done manually or with
  successive calls to [brm_prior_label()].
2. Supply the labeling scheme to [brm_prior_archetype()].
  [brm_prior_archetype()] uses attributes of the archetype
  to map labeled priors to their rightful parameters in the model.
```

For informative prior archetypes, this process is much more convenient and robust than manually calling brms::set_prior(). However, it requires an understanding of how the labels of the priors map to parameters in the model. This mapping varies from archetype to archetype, and it is documented in the help pages of archetype-specific functions such as brm_archetype_successive_cells().

See Also

Other priors: brm_prior_archetype(), brm_prior_label(), brm_prior_simple()

Examples

```
set.seed(0L)
data <- brm_simulate_outline(
  n_group = 2,
  n_patient = 100,
  n_time = 3,
  rate_dropout = 0,
  rate_lapse = 0
) |>
  dplyr::mutate(response = rnorm(n = dplyr::n())) |>
  brm_simulate_continuous(names = c("biomarker1", "biomarker2")) |>
  brm_simulate_categorical(
    names = c("status1", "status2"),
    levels = c("present", "absent")
  )
archetype <- brm_archetype_successive_cells(data)
label <- brm_prior_template(archetype)
label$code <- c(
  "normal(1, 1)",
  "normal(1, 2)",
  "normal(1, 3)",
  "normal(2, 1)",
  "normal(2, 2)",
  "normal(2, 3)"
)
brm_prior_archetype(label = label, archetype = archetype)
```

---

brm_recenter_nuisance *Recenter nuisance variables*

Description

Change the center of a nuisance variable of an informative prior archetype.

Usage

```
brm_recenter_nuisance(data, nuisance, center)
```

Arguments

data	An informative prior archetype data frame output from brm_archetype_cells() or similar.
nuisance	Character of length 1, name of the nuisance column in the data to shift the center.
center	Numeric of length 1, value of the center to shift the column in nuisance. The affected column in the returned archetype data frame will look as if it were centered by this value to begin with.

Details

By "centering vector y at scalar x", we mean taking the difference z = y - x. If x is the mean, then mean(z) is 0. Informative prior archetypes center nuisance variables at their means so the parameters can be interpreted correctly for setting informative priors. This is appropriate most of the time, but sometimes it is better to center a column at a pre-specified scientifically meaningful fixed number. If you want a nuisance column to be centered at a fixed value other than its mean, use brm_recenter_nuisance() to shift the center. This function can handle any nuisance variable

Value

An informative prior archetype data frame with one of the variables re-centered.

Examples

```
set.seed(0L)
data <- brm_simulate_outline(
  n_group = 2,
  n_patient = 100,
  n_time = 4,
  rate_dropout = 0,
  rate_lapse = 0
) |>
  dplyr::mutate(response = rnorm(n = dplyr::n())) |>
  brm_data_change() |>
  brm_simulate_continuous(names = c("biomarker1", "biomarker2")) |>
  brm_simulate_categorical(
    names = c("status1", "status2"),
    levels = c("present", "absent")
  )
archetype <- brm_archetype_cells(data)
mean(archetype$nuisance_biomarker1) # after original centering
center <- mean(data$biomarker1)
center # original center, before the centering from brm_archetype_cells()
attr(archetype$nuisance_biomarker1, "brm_center") # original center
max(abs((data$biomarker1 - center) - archetype$nuisance_biomarker1))
# Re-center nuisance_biomarker1 at 0.75.
archetype <- brm_recenter_nuisance(
  data = archetype,
  nuisance = "nuisance_biomarker1",
  center = 0.75
)
attr(archetype$nuisance_biomarker1, "brm_center") # new center
mean(archetype$nuisance_biomarker1) # no longer equal to the center
# nuisance_biomarker1 is now as though we centered it at 0.75.
max(abs((data$biomarker1 - 0.75) - archetype$nuisance_biomarker1))
```

---

brm_simulate_categorical
Append simulated categorical covariates

Description

Simulate and append non-time-varying categorical covariates to an existing brm_data() dataset.

Usage

```
brm_simulate_categorical(data, names, levels, probabilities = NULL)
```

Arguments

data	Classed tibble as from brm_data() or brm_simulate_outline().
names	Character vector with the names of the new covariates to simulate and append. Names must all be unique and must not already be column names of data.
levels	Character vector of unique levels of the simulated categorical covariates.
probabilities	Either NULL or a numeric vector of length length(levels) with levels between 0 and 1 where all elements sum to 1. If NULL, then all levels are equally likely to be drawn. If not NULL, then probabilities is a vector of sampling probabilities corresponding to each respective level of levels.

Details

Each covariate is a new column of the dataset with one independent random categorical draw for each patient, using a fixed set of levels (via base::sample() with replace = TRUE). All covariates simulated this way are independent of everything else in the data, including other covariates (to the extent that the random number generators in R work as intended).

Value

A classed tibble, like from brm_data() or brm_simulate_outline(), but with new categorical covariate columns and with the names of the new covariates appended to the brm_covariates attribute. Each new categorical covariate column is a character vector, not the factor type in base R.

See Also

Other simulation: brm_simulate_continuous(), brm_simulate_outline(), brm_simulate_prior(), brm_simulate_simple()

Examples

```
data <- brm_simulate_outline()
brm_simulate_categorical(
  data = data,
  names = c("site", "region"),
  levels = c("area1", "area2")
)
brm_simulate_categorical(
  data = data,
  names = c("site", "region"),
  levels = c("area1", "area2"),
  probabilities = c(0.1, 0.9)
)
```

brm_simulate_continuous
Append simulated continuous covariates

Description

Simulate and append non-time-varying continuous covariates to an existing brm_data() dataset.

Usage

```
brm_simulate_continuous(data, names, mean = 0, sd = 1)
```

Arguments

data	Classed tibble as from brm_data() or brm_simulate_outline().
names	Character vector with the names of the new covariates to simulate and append. Names must all be unique and must not already be column names of data.
mean	Numeric of length 1, mean of the normal distribution for simulating each covariate.
sd	Positive numeric of length 1, standard deviation of the normal distribution for simulating each covariate.

Details

Each covariate is a new column of the dataset with one independent random univariate normal draw for each patient. All covariates simulated this way are independent of everything else in the data, including other covariates (to the extent that the random number generators in R work as intended).

Value

A classed tibble, like from brm_data() or brm_simulate_outline(), but with new numeric covariate columns and with the names of the new covariates appended to the brm_covariates attribute.

See Also

Other simulation: brm_simulate_categorical(), brm_simulate_outline(), brm_simulate_prior(), brm_simulate_simple()

Examples

```
data <- brm_simulate_outline()
brm_simulate_continuous(
  data = data,
  names = c("age", "biomarker")
)
brm_simulate_continuous(
```

```
    data = data,
    names = c("biomarker1", "biomarker2"),
    mean = 1000,
    sd = 100
  )
```

---

brm_simulate_outline *Start a simulated dataset*

Description

Begin creating a simulated dataset.

Usage

```
brm_simulate_outline(
  n_group = 2L,
  n_subgroup = NULL,
  n_patient = 100L,
  n_time = 4L,
  rate_dropout = 0.1,
  rate_lapse = 0.05
)
```

Arguments

n_group	Positive integer of length 1, number of treatment groups.
n_subgroup	Positive integer of length 1, number of subgroup levels. Set to NULL to omit the subgroup entirely.
n_patient	Positive integer of length 1. If n_subgroup is NULL, then n_patient is the number of patients per treatment group. Otherwise, n_patient is the number of patients per treatment group *per subgroup*. In both cases, the total number of patients in the whole simulated dataset is usually much greater than the n_patients argument of brm_simulate_outline().
n_time	Positive integer of length 1, number of discrete time points (e.g. scheduled study visits) per patient.
rate_dropout	Numeric of length 1 between 0 and 1, post-baseline dropout rate. A dropout is an intercurrent event when data collection for a patient stops permanently, causing the outcomes for that patient to be missing during and after the dropout occurred. The first time point is assumed to be baseline, so dropout is there. Dropouts are equally likely to occur at each of the post-baseline time points.
rate_lapse	Numeric of length 1, expected proportion of post-baseline outcomes that are missing. Missing outcomes of this type are independent and uniformly distributed across the data.

Value

A classed data frame from brm_data(). The data frame has one row per patient per time point and the following columns:

- group: integer index of the treatment group.
- patient: integer index of the patient.
- time: integer index of the discrete time point.

See Also

Other simulation: brm_simulate_categorical(), brm_simulate_continuous(), brm_simulate_prior(), brm_simulate_simple()

Examples

```
brm_simulate_outline()
```

---

brm_simulate_prior *Prior predictive draws.*

Description

Simulate the outcome variable from the prior predictive distribution of an MMRM using brms.

Usage

```
brm_simulate_prior(
  data,
  formula,
  prior = brms.mmrm::brm_prior_simple(data = data, formula = formula),
  ...
)
```

Arguments

data	A classed data frame from brm_data(), or an informative prior archetype from a function like brm_archetype_successive_cells().
formula	An object of class "brmsformula" from brm_formula() or brms::brmsformula(). Should include the full mapping of the model, including fixed effects, residual correlation, and heterogeneity in the discrete-time-specific residual variance components.
prior	A valid brms prior object with proper priors for parameters b (model coefficients), b_sigma (log residual standard deviations for each time point), and cortime (residual correlations among time points within patients). See the brm_prior_simple() function for an example.
...	Named arguments to specific brm_formula() methods.

Details

brm_simulate_prior() calls brms::brm() with sample_prior = "only", which sets the default intercept prior using the outcome variable and requires at least some elements of the outcome variable to be non-missing in advance. So to provide feasible and consistent output, brm_simulate_prior() temporarily sets the outcome variable to all zeros before invoking brms::brm().

Value

A list with the following elements:

- data: a classed tibble with the outcome variable simulated as a draw from the prior predictive distribution (the final row of outcome in the output). If you simulated a missingness pattern with brm_simulate_outline(), then that missingness pattern is applied so that the appropriate values of the outcome variable are set to NA.
- model: the brms model fit object.
- model_matrix: the model matrix of the fixed effects, obtained from brms::make_standata().
- outcome: a numeric matrix with one column per row of data and one row per saved prior predictive draw.
- parameters: a tibble of saved parameter draws from the prior predictive distribution.

See Also

Other simulation: brm_simulate_categorical(), brm_simulate_continuous(), brm_simulate_outline(), brm_simulate_simple()

Examples

```
if (identical(Sys.getenv("BRM_EXAMPLES", unset = ""), "true")) {
set.seed(0L)
data <- brm_simulate_outline()
data <- brm_simulate_continuous(data, names = c("age", "biomarker"))
data$response <- rnorm(nrow(data))
formula <- brm_formula(
  data = data,
  baseline = FALSE,
  baseline_time = FALSE
)
tmp <- utils::capture.output(
  suppressMessages(
    suppressWarnings(
      out <- brm_simulate_prior(
        data = data,
        formula = formula
      )
    )
  )
)
out$data
}
```

brm_simulate_simple *Simple MMRM simulation.*

Description

Simple function to simulate a dataset from a simple specialized MMRM.

Usage

```
brm_simulate_simple(
  n_group = 2L,
  n_patient = 100L,
  n_time = 4L,
  hyper_beta = 1,
  hyper_tau = 0.1,
  hyper_lambda = 1
)
```

Arguments

n_group	Positive integer of length 1, number of treatment groups.
n_patient	Positive integer of length 1, number of patients per treatment group.
n_time	Positive integer of length 1, number of discrete time points (e.g. scheduled study visits) per patient.
hyper_beta	Positive numeric of length 1, hyperparameter. Prior standard deviation of the fixed effect parameters `beta`.
hyper_tau	Positive numeric of length 1, hyperparameter. Prior standard deviation parameter of the residual log standard deviation parameters `tau`
hyper_lambda	Positive numeric of length 1, hyperparameter. Prior shape parameter of the LKJ correlation matrix of the residuals among discrete time points.

Details

Refer to the methods vignette for a full model specification. The `brm_simulate_simple()` function simulates a dataset from a simple pre-defined MMRM. It assumes a cell means structure for fixed effects, which means there is one fixed effect scalar parameter (element of vector `beta`) for each unique combination of levels of treatment group and discrete time point. The elements of `beta` have independent univariate normal priors with mean 0 and standard deviation `hyper_beta`. The residual log standard deviation parameters (elements of vector `tau`) have normal priors with mean 0 and standard deviation `hyper_tau`. The residual correlation matrix parameter `lambda` has an LKJ correlation prior with shape parameter `hyper_lambda`.

Value

A list of three objects:

- data: A tidy dataset with one row per patient per discrete time point and columns for the outcome and ID variables.
- model_matrix: A matrix with one row per row of data and columns that represent levels of the covariates.
- parameters: A named list of parameter draws sampled from the prior:
  - beta: numeric vector of fixed effects.
  - tau: numeric vector of residual log standard parameters for each time point.
  - sigma: numeric vector of residual standard parameters for each time point. sigma is equal to exp(tau).
  - lambda: correlation matrix of the residuals among the time points within each patient.
  - covariance: covariance matrix of the residuals among the time points within each patient. covariance is equal to diag(sigma) %*% lambda %*% diag(sigma).

See Also

Other simulation: brm_simulate_categorical(), brm_simulate_continuous(), brm_simulate_outline(), brm_simulate_prior()

Examples

```
set.seed(0L)
simulation <- brm_simulate_simple()
simulation$data
```

---

brm_transform_marginal *Marginal mean transformation*

Description

Transformation from model parameters to marginal means.

Usage

```
brm_transform_marginal(
  data,
  formula,
  average_within_subgroup = NULL,
  prefix = "b_"
)
```

Arguments

data
: A classed data frame from brm_data(), or an informative prior archetype from a function like brm_archetype_successive_cells().

formula
: An object of class "brmsformula" from brm_formula() or brms::brmsformula(). Should include the full mapping of the model, including fixed effects, residual correlation, and heterogeneity in the discrete-time-specific residual variance components.

average_within_subgroup
: TRUE to average concomitant covariates proportionally within subgroup levels, FALSE to average these covariates across the whole dataset. If average_within_subgroup is NULL (default), and if the model has a subgroup and nuisance variables, then brm_transform_marginal() prints and informative message (once per session) and sets average_within_subgroup to FALSE. If you see this message, please read https://openpharma.github.io/brms.mmrm/articles/inference.html, decide whether to set average_within_subgroup to TRUE or FALSE in brm_transform_marginal(), and then manually supply the output of brm_transform_marginal() to the transform argument of brm_marginal_draws().

  To create marginal means, brms.mmrm conditions the nuisance covariates on their averages across the whole dataset (average_within_subgroup = FALSE or NULL in brm_transform_marginal()). This may be reasonable in some cases, and it mitigates the kind of hidden confounding between the subgroup and other variables which may otherwise cause Simpson's paradox. However, for subgroup-specific marginal means, it may not be realistic to condition on a single point estimate for all levels of the reference grid (for example, if the subgroup is female vs male, but all marginal means condition on a single overall observed pregnancy rate of 5%). In these situations, it may be appropriate to instead condition on subgroup-specific averages of nuisance variables (average_within_subgroup = TRUE in brm_transform_marginal()). But if you do this, it is your responsibility to investigate and understand the hidden interactions and confounding in your dataset. For more information, please visit https://openpharma.github.io/brms.mmrm/articles/inference.html and https://cran.r-project.org/package=emmeans/vignettes/interactions.html.

prefix
: Character of length 1, prefix to add to the model matrix ("X") from brms::make_standata() in order to reconstruct the brms model parameter names. This argument should only be modified for testing purposes.

Details

The matrix from brm_transform_marginal() is passed to the transform_marginal argument of brm_marginal_draws(), and it transforms posterior draws of model parameters to posterior draws of marginal means. You may customize the output of brm_transform_marginal() before passing it to brm_marginal_draws(). However, please do not modify the dimensions, row names, or column names.

Value

A matrix to transform model parameters (columns) into marginal means (rows).

Examples

```
set.seed(0L)
data <- brm_data(
  data = brm_simulate_simple()$data,
  outcome = "response",
  group = "group",
  time = "time",
  patient = "patient",
  reference_group = "group_1",
  reference_time = "time_1"
)
formula <- brm_formula(
  data = data,
  baseline = FALSE,
  baseline_time = FALSE
)
transform <- brm_transform_marginal(data = data, formula = formula)
equations <- summary(transform)
print(equations)
summary(transform, message = FALSE)
class(transform)
print(transform)
```

Index