From BRMS to Stan

Mitzi Morris

2025-03-18

Stan Development Team

BRMS: Bayesian Regression and Multilevelmodeling in Stan

brms

Bayesian regression models using Stan

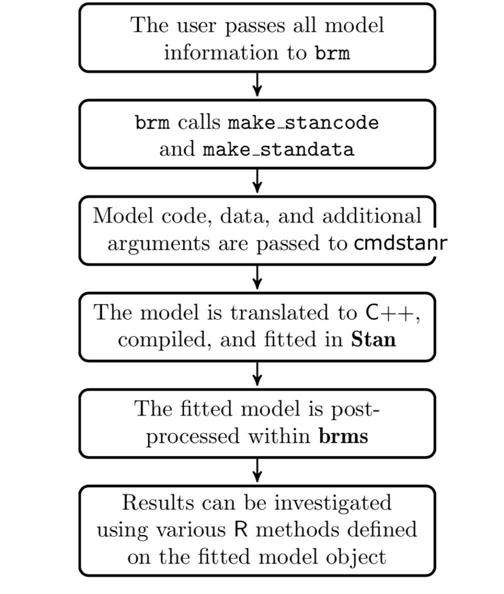
The **brms** package provides an interface to fit Bayesian generalized (non-)linear multivariate multilevel models using Stan. The formula syntax is very similar to that of the package lme4 to provide a familiar and simple interface for performing regression analyses.

https://paul-buerkner.github.io/brms/

My Questions for You

- What kinds of models do you use for research? For teaching?
- BRMS
  - How often do you use it: never, sometimes, always?
  - What do you like/dislike/find confusing about BRMS?
- Stan
  - Have you run Stan models through an interface, RStan, CmdStanR, CmdStanPy?
  - Have you written / tried to write or modify Stan models?
- What other tools do you use?

BRMS Processing



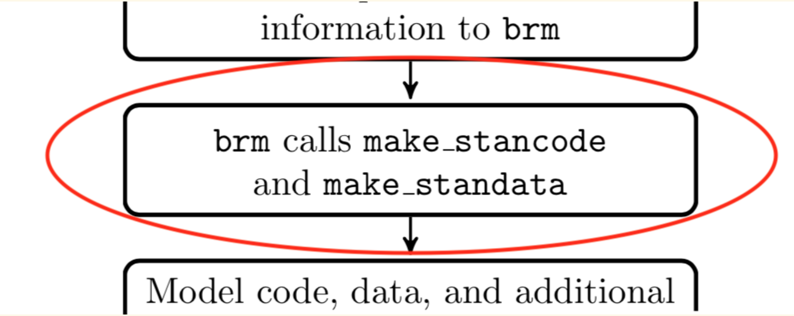
Specify and fit a model in BRMS

```
fit1 <- brm(Reaction ~ Days + (Days|Subject),
            data = sleepstudy)

sleep_model <- stancode(fit1)

sleep_data <- standata(fit1)
```

What's the problem?

BRMS Processing

This is the problem!



The Stan program created by `make_stancode` is not easy to understand or modify.

BRMS formula syntax

- The BRMS formula specifies the regression.
- For a multilevel model, BRMS extends the R formula syntax, following lme4.

$$\text{response} \sim \text{pterms} + (\text{gterms} \mid \text{group})$$

- The `pterms` contain *population-level* effects, assumed to be the same across observations.
- The `gterms` contain *group-level* effects, assumed to vary across the grouping variables specified as `group`.
- The intercept term is either 1 or 0 for no intercept; if unspecified, default is 1.

Function brm

```
fit1 <- brm(Reaction ~ Days + (Days|Subject),
            data = sleepstudy)
```

- First argument - required: the extended-syntax R formula for the linear predictor.
- Second argument (`data`) - required: a dataframe containing columns names which correspond to the terms in the formula.
- Additional arguments to the `brm` function allow futher model specification and configure the call to the inference algorithm.
  - family (default “gaussian”) - response distribution and link function
  - prior - override BRMS defaults
  - brm function

BRMS Description of a Multilevel Model

$$y_i \sim \mathrm{D}(f^{-1}(\eta_i), \theta)$$

- `brm` formula argument specifies η, the linear predictor
  - η can be rewritten $\mathrm{X}\,\beta + \mathrm{Z}\,\mathrm{u}$ where
    β and u are population-level and group-level coefficients respectively
- `data` argument specifies design matrics X, Z and the modeled data y.
- `family` argument specifies both
  - D, the distributional family
  - f, the *inverse link function*
- `prior` argument overrides defaults on θ, the family-specific parameter(s)

The Stan Language

- A Stan program
  - Declares data and (constrained) parameter variables
  - Defines log posterior (or penalized likelihood)
  - Computes quantities of interest
- Syntax
  - Influenced by BUGS (plus Java/C++ punctuation)
  - Explicit variable types (like Java/C++, not like Python, R)
  - Named program blocks - distinguish between data and parameters (not like BUGS)
  - Control flow: `if` statements - dynamic branch points (more powerful than BUGS)
- Mathematical operations
  - Stan language includes a very large set of probability distributions and mathematical functions from Stan's math library
  - Efficient, vectorized (almost always)

Stan Example: Laplace's Model of Birth Rate by Sex

Laplace's data on live births in Paris from 1745–1770:

sex	*live births*
female	241 945
male	251 527

- *Question 1* (Estimation): What is the birth rate of boys vs. girls?
- *Question 2* (Event Probability): Is a boy more likely to be born than a girl?

Laplace computed this analytically. Let's use Stan's NUTS-HMC sampler instead.

Laplace's Model in Stan

```
transformed data {
  int male = 251527;
  int female = 241945;
}
parameters {
  real<lower=0, upper=1> theta;
}
model {
  male ~ binomial(male + female, theta);
}
generated quantities {
  int<lower=0, upper=1> theta_gt_half = (theta > 0.5);
}
```

Laplace's Answer

```
births_model = cmdstan_model("laplace.stan")   # compile model
births_fit = births_model$sample()             # run inference algorithm
as.data.frame(births_fit$summary())            # manage results
```

variable	mean	median	sd	q5	q95
theta	0.51	0.51	0.000725	0.509	0.511
theta_gt_half	1.00	1.00	0.000000	1.000	1.000

- *Question 1* (Estimation): What is the birth rate of boys vs. girls?

  θ is 90% certain to lie in $(0.509, 0.511)$

- *Question 2* (Event Probability): Is a boy more likely to be born than a girl?

  Laplace "morally certain" boys more prevalent

Stan Program File

A Stan program consists of one or more named program blocks, strictly ordered

```
functions {
  // declare, define functions
} data {
  // declare input data
} transformed data {
   // transform inputs, define program data
} parameters {
   // declare (continuous) parameters
} transformed parameters {
   // define derived parameters
} model {
   // compute the log joint distribution
} generated quantities {
   // define quantities of interest
}
```

Stan Program Blocks - Execution During Sampling

- `data`, `transformed data` blocks - executed once on startup
- `parameters` -
  - on startup: initialize parameters
  - at every step of inference algorithm: validate constraints
- `transformed parameters`, `model` blocks - executed every *step* of the sampler
- `generated quantities` - executed every *iteration* of the sampler
- After every sampler iteration, program outputs current values of all variables in parameters, transformed parameters, and generated quantities blocks.

The Stan Language

- Variables have strong, static types
  - Strong: only values of that type will be assignable to the variable, (type promotion allowed, e.g., `int` to `real`, `complex`)
  - Static: variable type is constant throughout program
  - Types: vector, row_vector, matrix, complex, array, and tuple
  - Variables can be declared with constraints on upper and/or lower bounds
- Motivation
  - Static types make programs easier to comprehend, debug, and maintain
  - Programming errors can be detected at compile-time
  - Constrained types catch runtime errors

```
int<lower=0> N, N_time;                         // constraint applies to both N, N_time
vector[3]<lower=0> sigma;
array[3] matrix[M, N] some_xs;
array[N] int<lower=0, upper=N_time> time;  // use arrays for structured int data
```

The Stan Language

The `model` block

- defines the joint log probability density function given parameters
- this function is the sum of all distribution and log probability increment statements in the model block

Distribution statements

```
y ~ normal(mu, sigma);
mu ~ normal(0, 10);
sigma ~ normal(0, 1);
```

Log probability increment statements

```
target += normal_lupdf(y | mu, sigma);
target += normal_lupdf(mu | 0, 10);
target += normal_lupdf(sigma | 0, 1);
```

Processing Steps

- `births_model = cmdstan_model("laplace.stan")    # compile model`
  - Stan compiler translates Stan file to C++ file
  - C++ file is compiled to executable program, via GNU Make
- `births_fit = births_model$sample()              # do inference`
  - Interfaces run the compiled executable (as a subprocess) and manage the per-chain outputs (modified CSV format files)
- `as.data.frame(births_fit$summary())             # manage results`
  - Parse CSV outputs into in-memory object
  - Access individual parameters and quantities of interest
  - Run summary and diagnostic reports

From BRMS to Stan

- In BRMS, specify arguments to the `brm` function: formula, data, family, prior. BRMS supplies defaults for all elements of the model

$$y_i \sim \mathrm{D}(f^{-1}(\eta_i), \theta)$$

  - $\mathrm{D}, f^{-1}, \theta$ are the distributional family, link function, and distributional parameters
  - η is the regression formula

- In Stan, specify a model using elements of the Stan probabalistic programming language.n

  - `data` block defines `y`, as well as all unmodeled data inputs
  - `parameters` block defines all distributional parameters.
  - `model` block specifies the likelihood and priors. (formula, distributional family, priors).

Model Development: Hello, World!

- A "Hello, World!" program is the name given to the first, simplest possible program written when learning a new programming language.
- Our "Hello, World!" model is the **complete pooling model**
  all subjects are alike; reaction time is predicted by days since study start.
- BRMS formula: `Reaction ~ Days`
- Stan likelihood: `y ~ normal(alpha + days_c * beta, sigma);`
  - center predictors
  - use weakly informative priors

Stan Model

```
data {
  int<lower=0> N;   vector[N] days;   vector[N] y;  // reaction time
}
transformed data {  // center predictor variable "days"
  real days_mean = mean(days); vector[N] days_c = days - days_mean;
}
parameters {
  real alpha;  real beta;  // intercept, slope
  real<lower=0> sigma; // residual standard deviation
}
model {
  y ~ normal(alpha + days_c * beta, sigma);
  alpha ~ normal(250, 50);  // informed prior for human reaction times in ms
  beta ~ normal(10, 10);  // weakly informed prior for per-day effect
  sigma ~ normal(0, 10);  // very weakly informative prior
}
generated quantities {  // adjust for centered predictor values
  real intercept = alpha - beta * days_mean;
}
```

Reading for Next Time

- Gelman blogpost varying-slope/varying-intercept models
  "If you want to program this directly in Stan, once you have varying intercepts and slopes, you have to deal with covariance-matrix decompositions and arrays of coefficient vectors, and it's all a hairy mess."
- Bob Carpenter's response Varying slopes and intercepts in Stan: still painful in 2024
  "I have to confess up front that Andrew's right—this is still painful"
- Optimization through Cholesky Factorization Stan User's Guide, Regression Models
- QR Decomposition Michael Betancourt case study