

From BRMS to Stan

Mitzi Morris

2025-03-25

Stan Development Team



BRMS: Bayesian Regression and Multilevel modeling in Stan

Recap from last week * BRMS extended formula syntax for multi-level regressions

`response ~ pterms + (gterms | group)`

- *pterms* define *population-level* effects; same for all observations.
- *gterms* define *group-level* effects; vary across *group* variable.
- The intercept term is 1 or 0 for no intercept; if unspecified, default is 1.

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

equivalent to

```
fit1 <- brm(Reaction ~ 1 + Days + (1 + Days | Subject), data = sleepstudy)
```

Stan Program File (review)

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 (review)

- 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.

From BRMS to Stan

- BRMS: specify arguments to the `brm` function: formula, data, family, prior.
 - BRMS generates Stan model code.
 - BRMS code is not quite human-readable, but efficiently coded.
- *Goal: write efficient, robust Stan program*
 - map regression formula to Stan program's sampling distribution statement.
 - data block defines all data inputs - outcomes and predictors, plus dimensions.
 - transformed data block mean-centers predictor variables.
 - parameters block defines all distributional parameters.
 - model block specifies the likelihood and priors.
- When should you do this?
 - When model specification in BRMS is long / complicated / not quite possible.

Stepwise 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.
 - **Pro tip: always start with “Hello, World!”**
- End goal is a efficient and maintainable multi-level model
 $\text{Reaction} \sim \text{Days} + (\text{Days} | \text{Subject})$
- Initial goal is a simple linear model - complete pooling across subjects
 $\text{Reaction} \sim \text{Days}$
($\text{Reaction} \sim 1 + \text{Days}$ - by default, model includes global intercept.)
- Carry over BRMS efficiencies to Stan model

Stan Model sleep_simple.stan

```
data {  
  int<lower=0> N;    vector[N] day;    vector[N] y;  // reaction time  
}  
transformed data {  
  real day_mean = mean(day);  
  vector[N] day_centered = day - day_mean;  
}  
parameters {  
  real alpha;  real b_day;  // intercept, slope  
  real<lower=0> sigma;  // residual standard deviation  
}  
model {  
  y ~ normal(alpha + day_centered * b_day, sigma);  // Reaction ~ 1 + day  
  alpha ~ normal(250, 50);  // informed prior for human reaction times in ms  
  b_day ~ normal(10, 10);  // weakly informed prior for per-day effect  
  sigma ~ normal(0, 10);  // very weakly informative prior  
}  
generated quantities {  
  real b_intercept = alpha - b_day * day_mean;  
  array[N] real y_rep = normal_rng(alpha + day_centered * b_day, sigma);  
}
```

Notebook - Stan, BRMS complete pooling model

```
sleep_data = list(  
  N = nrow(sleepstudy), J = length(unique(sleepstudy$Subject)),  
  subj = as.integer(sleepstudy$Subject), day = sleepstudy$Days,  
  y = as.double(sleepstudy$Reaction)  
)  
  
sleep_simple = cmdstan_model("stan/sleep_simple.stan")  
sleep_simple_stanfit = sleep_simple$sample(data = sleep_data)  
as.data.frame(sleep_simple_stanfit$summary(variables = c('b_intercept', 'b_day', 'sigma')))  
  
priors <- c(set_prior("normal(250, 50)", class = "Intercept"),  
  set_prior("normal(10, 10)", class = "b"),  
  set_prior("normal(0, 10)", class = "sigma"))  
  
sleep_simple_brmsfit = brm(Reaction ~ Days, data = sleepstudy, prior = priors)  
sleep_simple_brmsfit
```


Multilevel models

- Specify model in terms of the inherent structure of the data
- Sleep study: reaction time varies by subject
 - BRMS formula: `Reaction ~ 1 + Days + (1 + Days|Subject)`
- Expand Stan model:
 - Predictor vector β is *multivariate normal*
 - `b_subj ~ multi_normal(mu_subj, sigma_subj)`
 - `mu_subj` is vector, `sigma_subj` is covariance matrix.
- *Problems*
 - Stan distribution `multi_normal` - requires inverting covariance matrix at every evaluation - computationally expensive.
 - Two sources of variance: `sigma` and hierarchical variance `sigma_subj`
difficult to estimate from small number of observations per group.

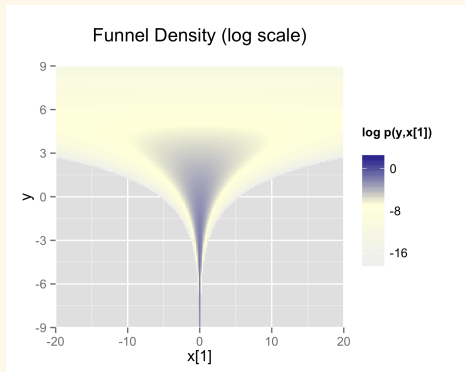
Multilevel models

- **Partial Pooling:** The hierarchical prior controls the pooling between levels:
 - Similar data across levels \rightarrow Low hierarchical variance \rightarrow Strong pooling
 - Dissimilar data across levels \rightarrow High hierarchical variance \rightarrow Weak pooling
- **Problem:** For low numbers of observations, MCMC sampler cannot resolve residual variance `sigma` and variance of hierarchical prior `sigma_subj`; many divergences
- **Solution:** Reparameterization, following code example in Stan User's Guide section [Hierarchical models and the non-centered parameterization](#)

Multilevel models and Neal's Funnel

- Neal's Funnel: extreme example of a challenging hierarchical model

$$p(y, x) = \text{normal}(y \mid 0, 3) \times \prod_{n=1}^9 \text{normal}(x_n \mid 0, \exp(y/2)).$$



Explore neck: need small stepsize on x-axis,
large stepsize on y-axis.

Explore mouth: need large stepsize on x-axis,
small stepsize on y-axis.

But stepsize is same for all axes; cannot
adequately sample either.

Funnel Example: Hierarchical Logistic Regression

Centered parameterization

- “Natural” parameterization.

```
parameters {  
  real y;  
  vector[9] x;  
}  
  
model {  
  y ~ normal(0, 3);  
  x ~ normal(0, exp(y/2));  
}
```

Non-centered parameterization

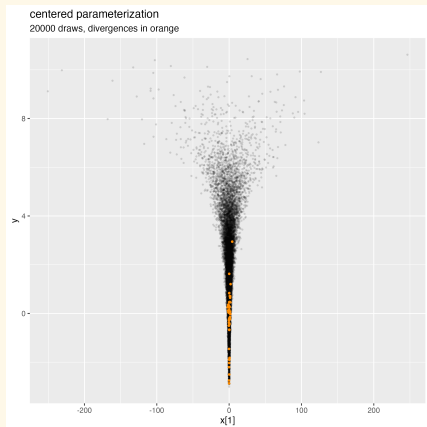
- Parameters block: declare standardized parameters.
- Transformed parameters: declare *variables* add offset (location), multiply by scale.

```
parameters {  
  real y_raw;  
  vector[9] x_raw;  
}  
  
transformed parameters {  
  // offset is 0, just multiply by scale  
  real y = 3.0 * y_raw;  
  vector[9] x = exp(y/2) * x_raw;  
}  
  
model {  
  y_raw ~ std_normal(); // y ~ normal(0, 3)  
  x_raw ~ std_normal(); // x ~ normal(0, exp(y/2))  
}
```

Compare Funnel Fits

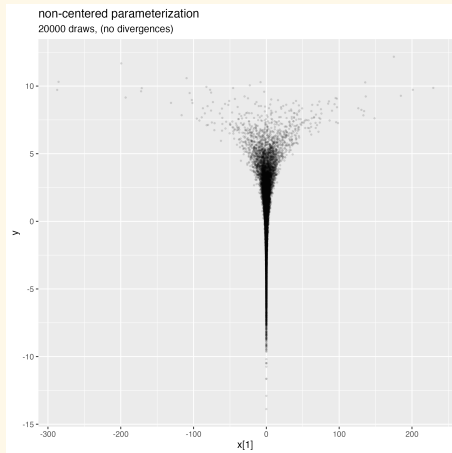
Centered parameterization

- Cannot explore neck of funnel
many divergences (in orange)



Non-centered parameterization

- Explores further; no divergences



BRMS to Stan

- Recap: formula `Reaction ~ 1 + Days` becomes distribution statement:
`y ~ normal(alpha + day_centered * b_day, sigma);`
- Coding `Reaction ~ 1 + Days + (1 + Days | Subject)`
 - Global intercept and slope for Days
 - Subject-specific random intercepts and slopes
- Correlation between random effects: prior on subject effect is a multivariate normal with mean vector μ and covariance matrix Σ .
- Multivariate reparameterization

Multi-variate reparameterization

Stan User's Guide [Multivariate regression reparameterization example](#)

Centered parameterization

```
parameters {  
  vector[K] beta;  
  vector[K] mu;  
  cov_matrix[K] Sigma;  
  // ...  
}  
model {  
  beta ~ multi_normal(mu, Sigma);  
  // ...  
}
```

Non-centered parameterization

```
parameters {  
  matrix[2, J] z; // standardized random effects  
  cholesky_factor_corr[2] L_Omega;  
  vector<lower=0>[2] tau; // scale of re  
  // ...  
}  
transformed parameters {  
  // random effects matrix scaled, transposed  
  matrix[J, 2] r =  
    (diag_pre_multiply(tau, L_Omega) * z)';  
}  
model {  
  to_vector(z) ~ std_normal();  
  // ...  
}
```

Specifying the Likelihood

- $\text{Reaction} \sim 1 + \text{Day} + (1 + \text{Subject} \mid \text{Day})$

```
y ~ normal(r[subj, 1] + b_intercept + (r[subj, 2] + b_day) .* day,  
           sigma);
```

- Arrange formula terms by intercept, slope
 - $r[\text{subj}, 1]$ is vector of per-subject intercepts
 - $b_intercept$ is population-level intercept
 - $r[\text{subj}, 2]$ is vector of per-subject day effects
 - b_day is population-level day effect

Parameters, transformed parameters, model blocks

```
parameters {  
  // population effects  
  real b_intercept;  real b_day;  real<lower=0> sigma;  
  // group-level effects  
  matrix[2, J] z; // standardized random effects  
  cholesky_factor_corr[2] L_Omega; // Cholesky factor of correlation matrix  
  vector<lower=0>[2] tau; // scale of random effects  
}  
transformed parameters {  
  // random effects matrix scaled, transposed  
  matrix[J, 2] r = (diag_pre_multiply(tau, L_Omega) * z)';  
}  
model {  
  y ~ normal(r[subj, 1] + b_intercept + (r[subj, 2] + b_day) .* day, sigma);  
  // priors  
  to_vector(z) ~ std_normal();  
  tau ~ cauchy(0, 25);  L_Omega ~ lkj_corr_cholesky(2);  sigma ~ exponential(1);  
  b_intercept ~ normal(250, 50);  b_day ~ normal(10, 10);  
}
```

Recovering the Quantities of Interest

```
generated quantities {  
  // Reconstruct correlation matrix from Cholesky factor  
  matrix[2, 2] Omega;  
  Omega = multiply_lower_tri_self_transpose(L_Omega);  
  real cor_intercept_day = Omega[1, 2];  
  
  // Get random effect variances  
  real sd_intercept = tau[1];  
  real sd_day = tau[2];  
}
```

Notebook Demo

```
# Stan multilevel model
```

```
sleep_mlm = cmdstan_model(stan_file = "stan/sleep_mlm.stan")
```

```
sleep_mlm_stanfit = sleep_mlm$sample(data = sleep_data)
```

```
as.data.frame(sleep_mlm_stanfit$summary(variables = c('b_intercept', 'b_day
```

```
# BRMS multilevel model
```

```
sleep_mlm_brmsfit <- brm(Reaction ~ Days + (Days|Subject), data = sleepst
```

```
sleep_mlm_brmsfit
```

Discussion

- BRMS formula syntax provides concise description of the regression.
 - function `brm` generates Stan code given both the formula and the data
 - default is a simple linear model.
- There is a point beyond which writing a model in Stan becomes easier than coding up the equivalent statements using BRMS.
- An efficient Stan program makes it easy for the sampler to converge and sample from the posterior.
 - zero-center predictors, make sure they are on the same scale.
 - the choice of the centered vs. non-centered parameterization depends on the amount of observations per group-level predictor.
 - use the non-centered parameterization for low-data regimes.

References

Stan User's Guide:

- Efficiency Tuning, [Hierarchical models and the non-centered parameterization](#)
- Efficiency Tuning, [Multivariate reparameterization](#)
- Regression, [Multivariate regression example](#)

Many Thanks!

Questions???