From BRMS to Stan

Mitzi Morris

2025-03-25

Stan Development Team

BRMS: Bayesian Regression and Multilevelmodeling 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.

- Stan: specify a model using elements of the Stan probabilistic programming language.
  - `data` block defines `y`, as well as all unmodeled data inputs.
  - `parameters` block defines all distributional parameters.
  - `model` block specifies the likelihood and priors.

- *Goal: make Stan code equally efficient.*

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

  `Reaction ~ Days + (Days|Subject)`
- Initial goal is a simple linear model - complete pooling across subjects

  `Reaction ~ Days`

  (`Reaction ~ 1 + 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);
  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

- Capture the structure in the data
- Sleep study: reaction time varies by subject
  - BRMS formula: `Reaction ~ 1 + Days + (1 + Days|Subject)`
- Expand Stan model - why not just:
  - Add per-subject parameter `vector[J] beta_subject`
  - Add hyperparameters `mu_subj`, `sigma_subj`, `beta_subject ~ normal(mu_subj, sigma_subj)`
  - ???
- Nope, not that easy.

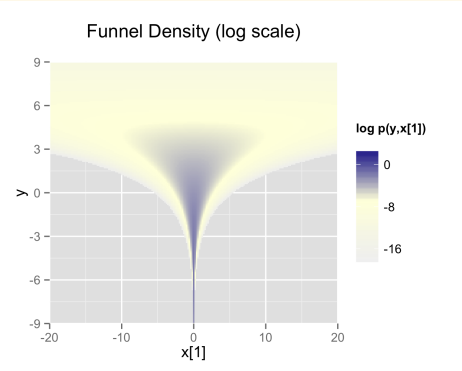
Multilevel models

- **Partial Pooling**: The hierarchical prior controls the pooling between levels:
  - Similar data across levels → Low hierarchical variance → Strong pooling
  - Dissimilar data across levels → High hierarchical variance → 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.

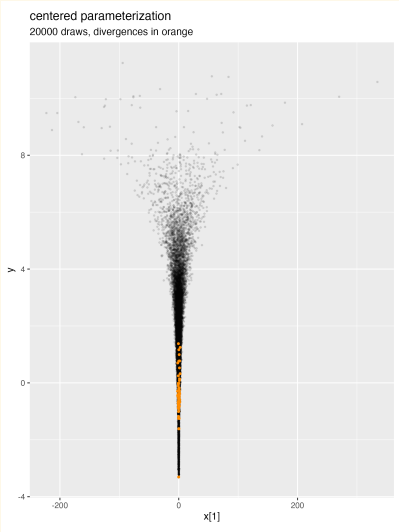
Explore mouth: need large stepsize on x-axis, small stepsize on y-axis.

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

Funnel Example: Hierarchical Logistic Regression

The natural parameterization is the "centered parameterization"

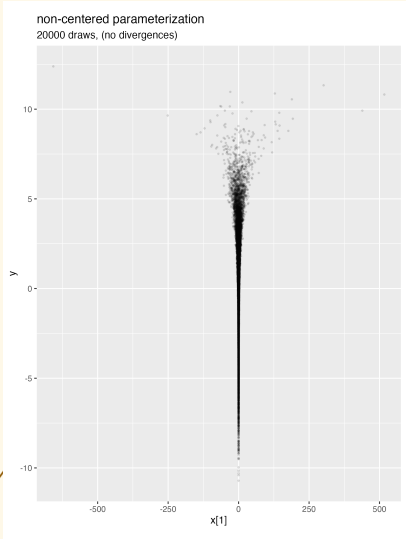
```
parameters {
  real y;
  vector[9] x;
}
model {
  y ~ normal(0, 3);
  x ~ normal(0, exp(y/2));
}
```



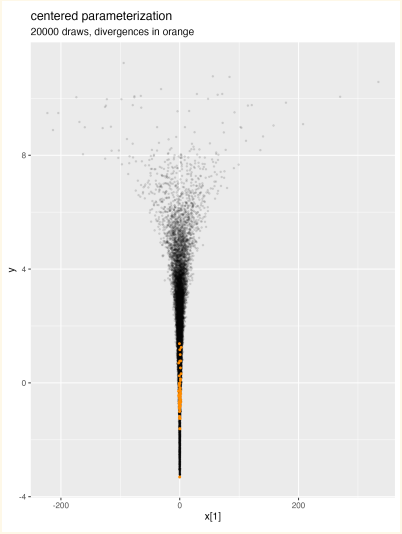
Non-centered reparameterization

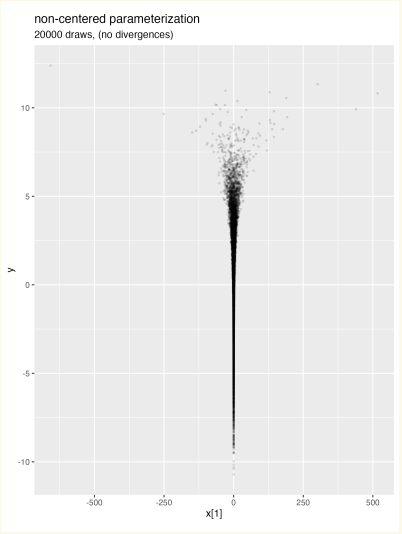
```
parameters {
  real y_raw;
  vector[9] x_raw;
}
transformed parameters {
  real y;
  vector[9] x;

  y = 3.0 * y_raw;
  x = exp(y/2) * x_raw;
}
model {
  y_raw ~ std_normal(); // implies y ~ normal(0, 3)
  x_raw ~ std_normal(); // implies x ~ normal(0, exp(y/
}
```



Compare Funnel Fits





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

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

Efficient multi-variate reparameterization

- Following Stan User's Guide: multivariate regression example

```
parameters {
  real b_intercept;  // global intercept
  real b_day; // global day effect
  real<lower=0> sigma;

  // Subject-level effects - non-centered parameterization
  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 [intercept, slope] by subject
  matrix[2, J] r = diag_pre_multiply(tau, L_Omega) * z;
}
```

Specifying the Likelihood

- `Reaction ~ 1 + Day + (1 + Subject | Day)`

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

- Stan formula terms re-arranged for vectorization
  - b_intercept is population-level intercept
  - `r[1, subj]'` is vector of per-subject intercepts
  - b_day is population-level day effect
  - `r[2, subj]'` is vector of per-subject day effects

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);

  // Get random effect variances
  vector[2] sd_r = tau;
  real sd_intercept = sd_r[1];
  real sd_day = sd_r[2];
  real cor_intercept_day = Omega[1, 2];
```

BRMS to Stan

- In BRMS, it's easy to specify varying intercept, varying slope effects.
- In Stan, it's not so easy, but it is well-documented.
- If you can code this, you can code most anything.

Notebook Demo

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???