A quick intro to brms

30 Oct 2020

Last updated 03 May 2021

By: Chloé

Session info:

- R version 4.0.1 (2020-06-06)
- brms version 2.13.0
- tidybayes version 2.1.1

1. The absolute basics

brms is a package for Bayesian modeling in R that talks to a software called Stan in the background.

In Bayesian models, parameters are estimated using MCMC (Markov chain Monte Carlo) instead of the maximum likelihood type estimation usually seen in your standard regressions (e.g. in lme4).

You can imagine an MCMC chain like a person wandering around a mountain range, and the mountain range is the range of values your parameter could take on. The chains walk all around and up and down for as long as you tell them to. You'll normally have multiple chains running around mountains at once, and they'll converge around a value—that's the estimate you're looking for.

brms is nice because it can handle more complicated models. For example lme4 sometimes gets yelly about singular fits with random slope models, but brms does it no problem.

Anyway moving on…

2. Building a model

Specify your model formula in the **bf( )** function using lme4 syntax:

```
spec <- bf(scale(gene_diversity) ~ hfi + (hfi|species)) # random slope | and intercept

mod <- brm(spec,
           cores = 4, chains = 4,
           iter = 2000, warmup = 1000,
           control = list(adapt_delta = 0.999, max_treedepth = 15),
           data = data)
```

- **chains:** those MCMC chains we were just talking about. Set the number of walkers going around the mountain. 3 or 4 is good.
- **cores:** the number of computer cores to use. If chains = cores, each chain will run on a separate core. If chains > cores, multiple chains will run on the same core, but they have to go one at a time. So it's slower. (*you can check how many CPU cores you have with detectCores() in the 'parallel' package*).
- **iter**: iterations. How many steps the walkers will take around the mountain. Default is 2000. Increase if you run into convergence issues.
- **warmup:** walker's gotta calibrate before it walks. Defaults to iterations/2
- **control = list(** *# these are optional parameters to set)*

  adapt_delta: the size of the steps your walker takes. The Stan help page puts it nicely. If the step is too large, your walker's gonna fall down the mountain, and that'll give you a '**divergent transitions**' error. The default value is 0.95. Increasing adapt_delta will give your walker smaller steps (and consequently make the chains slower). You can set it to any value approaching 1, but not 1, and obviously not above 1.

  max_treedepth: something about efficiency. Not a huge deal. You'll probably get an error about exceeding max tree depth at some point. The default is 10; increase as you wish.

2. More building a model

```
spec <- bf(gene_diversity ~ hfi + (hfi|species)) # random slope | and intercept

mod <- brm(spec,
           cores = 4, chains = 4,
           iter = 2000, warmup = 1000,
           control = list(adapt_delta = 0.999, max_treedepth = 15),
           prior = set_prior("normal(0, 0.5)", class = "b", coef = "scalehfi")
           data = data, family = "beta")
```

- **priors:** the default prior for the fixed effects is an uninformative prior (estimate can be any real number). Note that you can set priors for different parameter classes in the model, e.g. your fixed effects (b), intercept (intercept), standard deviation (sd), correlation matrices (L). You can view your model's priors with **prior_summary()**. Can't say more about that because I don't know more. In the code above I set the prior for my fixed 'scalehfi' effect to be a normal distribution with mean 0 and standard deviation 0.5.

- **family:** model family, default is normal distribution. (list of model families)

- Everything else you could add: https://rdrr.io/cran/brms/man/brm.html

```
prior_summary(mod)
```

```
                  prior     class      coef   group resp dpar nlpar bound
1                               b
2                               b  scalehfi
3  student_t(3, 0, 2.5) Intercept
4  lkj_corr_cholesky(1)         L
5                               L           species
6  student_t(3, 0, 2.5)        sd
7                               sd           species
8                               sd Intercept species
9                               sd  scalehfi species
10 student_t(3, 0, 2.5)     sigma
```

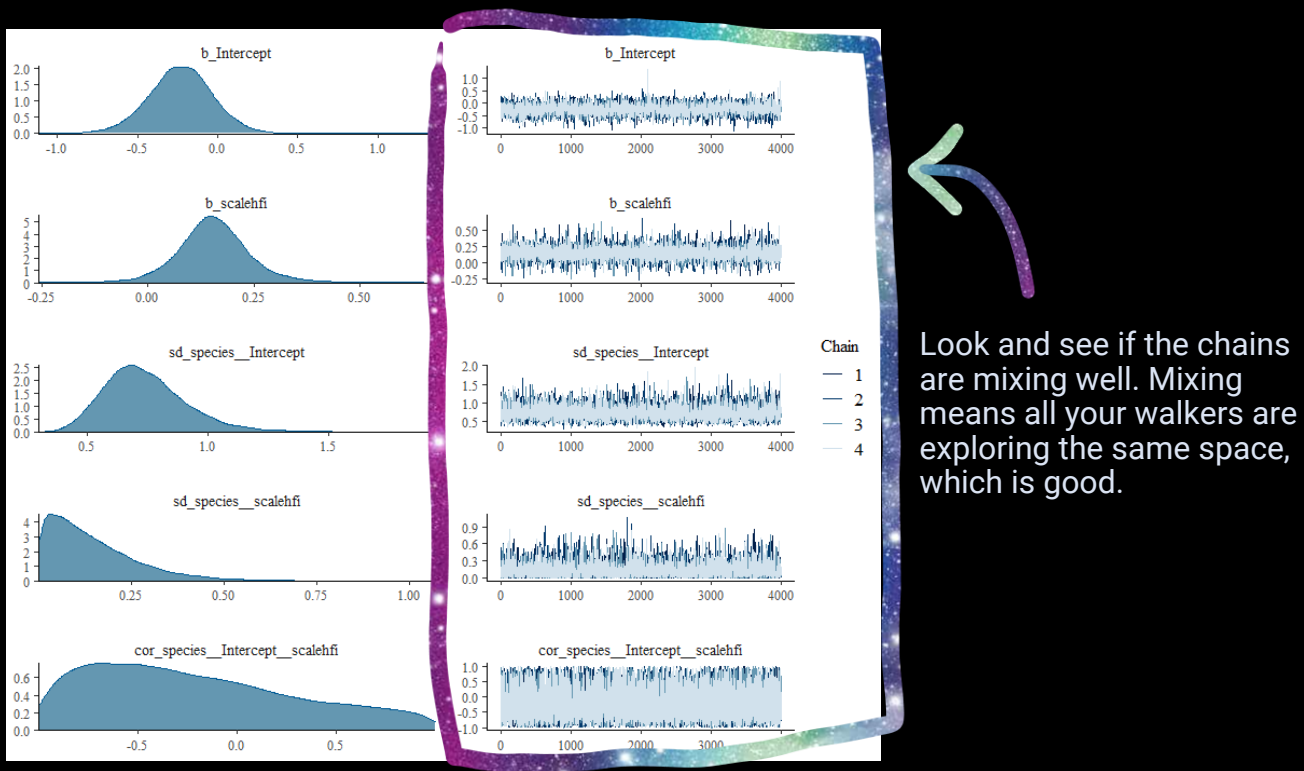
3. Check the model

Plot and check residuals as you normally would:

```
library(tidyverse)
library(tidybayes) # you'll want this to make fun plots with brms models!

data %>%
  add_residual_draws(mod) %>%
  ggplot(aes(x = .row, y = .residual)) +
  stat_pointinterval()
```

Plot and check your chains:

```
plot(mod)
```



Look and see if the chains are mixing well. Mixing means all your walkers are exploring the same space, which is good.

If everything looks fine we can move on…

4. Reading the summary

What in the heck did we just do

```
summary(mod)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: scale(log(Ne)) ~ scale(hfi) + (scale(hfi) | species)
   Data: data_group (Number of observations: 387)
Samples: 4 chains, each with iter = 5000; warmup = 1000; thin = 1;
         total post-warmup samples = 16000

Group-Level Effects:
~species (Number of levels: 18)
                     Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)            0.74      0.17     0.46     1.13 1.00     5024     7740
sd(scalehfi)             0.15      0.12     0.01     0.46 1.00     3633     6515
cor(Intercept,scalehfi)  -0.21      0.51    -0.95     0.87 1.00    13282     9454

Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept    -0.23      0.20    -0.64     0.15 1.00     3844     6321
scalehfi      0.15      0.09    -0.01     0.34 1.00     9424     8381

Family Specific Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     0.83      0.03     0.77     0.89 1.00    26922    11027

Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

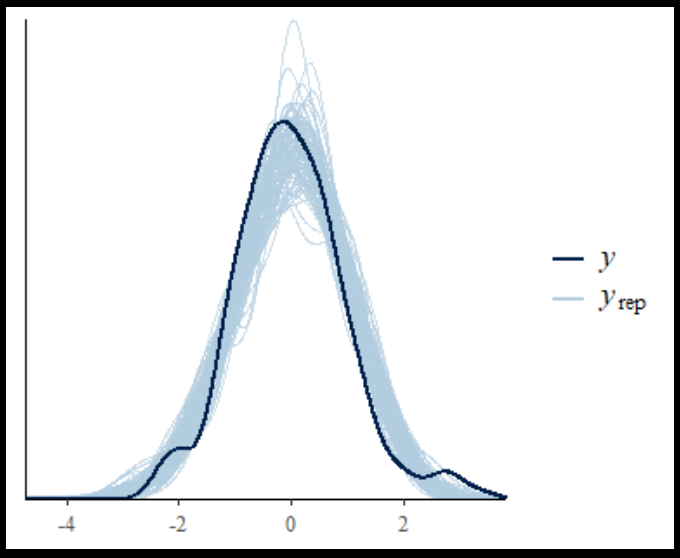
- Your fixed effects
- **Estimate:** the parameter estimate, aka coefficient aka effect size
- **Est.Error:** standard deviation
- **l-95% CI** and **u-95%CI:** lower 95% and upper 95% *credible* intervals. If these overlap 0, as they do here, there's no effect. 95% is the default interval, but you can specify whatever interval you want with 'prob', like: **summary(mod, prob = 0.8)** for 80% intervals.
- **Rhat, Bulk_ESS, Tail_ESS:** These tell you how well the model was able to estimate the parameter. Rhat = 1 when the model converges. If it's a lot bigger than 1, the model needs more time (more iterations). ESS is effective sample size (something about the quality of samples…) and bulk ESS should be > 1000.

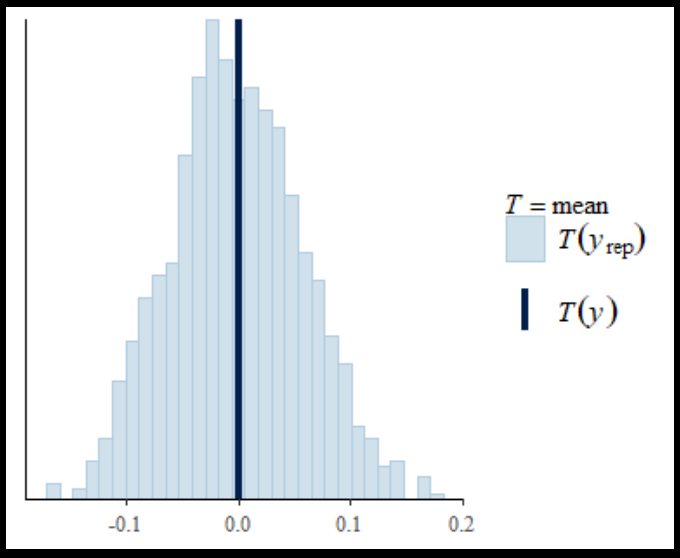
cooooooooool

5. Check it more (posterior predictive checks)

Posterior predictive checks means comparing your data (y) to data that's generated by your model (y_{rep}). Ideally you want your data to match up with what the model predicts.

```
# check predicted values drawn from posterior distribution vs actual y values
# different ways to visualize this:
# density plot
pp_check(mod, type = "dens_overlay", nsamples = 100) # nsamples are # of light blue lines

# compare statistic (e.g., mean)
pp_check(mod, type = "stat", stat = "mean", nsamples = 100)
```





These look pretty good. If the dark blue line falls outside the light blue predicted distributions, you may have a problem

5. MORE checking (and model comparing)

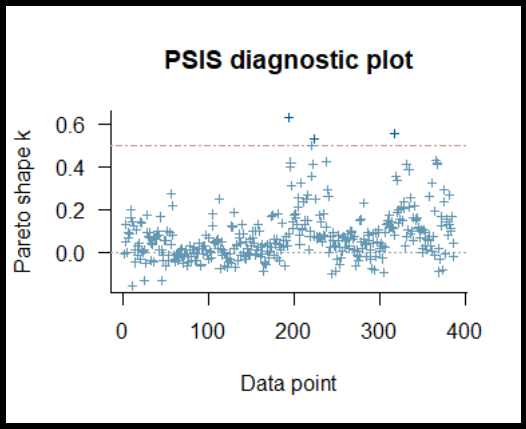
LOO (*leave-one-out cross validation*) is another way to evaluate and compare models. Disclaimer: I don't really know about this. There are some useful resources with a lot more detailed explanations on the last page.

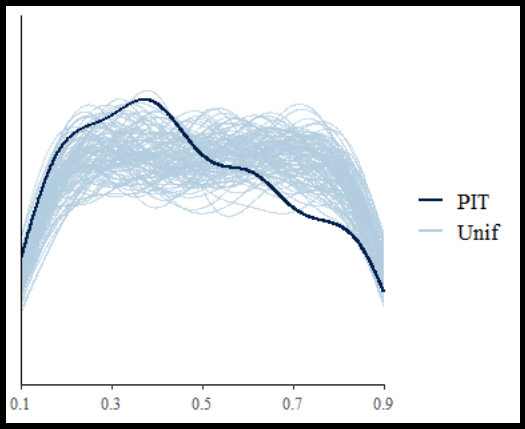
```
loo(mod)
```

```
Computed from 16000 by 387 log-likelihood matrix

         Estimate   SE
elpd_loo   -486.2 18.8
p_loo        19.0  2.2
looic       972.4 37.6
------
Monte Carlo SE of elpd_loo is 0.0.

Pareto k diagnostic values:
                         Count Pct.    Min. n_eff
(-Inf, 0.5]   (good)     384   99.2%   3185
 (0.5, 0.7]   (ok)         3    0.8%   3729
   (0.7, 1]   (bad)        0    0.0%   <NA>
   (1, Inf)   (very bad)   0    0.0%   <NA>

All Pareto k estimates are ok (k < 0.7).
See help('pareto-k-diagnostic') for details.
```

- **elpd_loo:** the "theoretical expected log pointwise predictive density for a new dataset." ....yeeeeeah
- **p_loo:** effective number of parameters. It *should* be lower than the number of data points and the number of parameters in your model.
- **looic:** like AIC, but LOO! Calculated from the elpd.
- **Pareto k:** basically how influential individual data points are on the posterior. High (>0.7) k values mean that observation is very influential. Check to make sure the data is correct (a typo, maybe?) or it might be an outlier. A bad p_loo and lots of bad k's can mean your model is bad, or super flexible. Can get wonky in mixed models if some groups have few data points. Interpret with a grain of salt.

```
# You can also inspect this visually
plot(loo(mod))
pp_check(mod, type = "loo_pit_overlay", nsamples = 100)
```





Seem fine

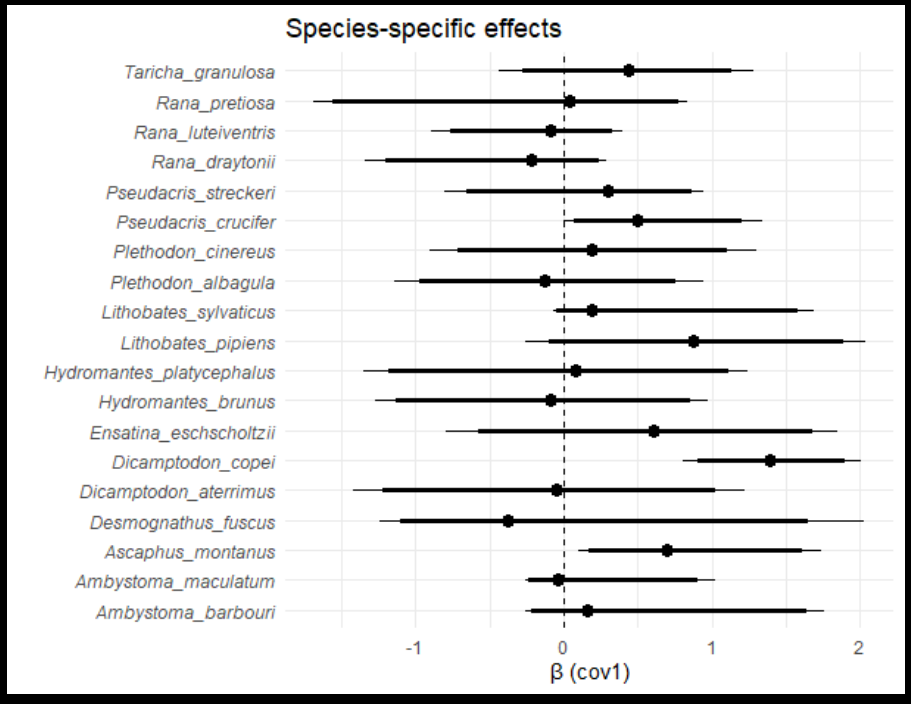
6. Group level effects

With hierarchical models you can use *tidybayes* to plot group level effects (random slopes or intercepts). The vignette for tidybayes tells you all you need to know. First, you gotta get the data you want out of your model. This example gets the parameter estimates for species effects in a random slope + intercept model.

```
# Create new dataframe with species parameter estimates by adding the species effect
# (r_species) to the overall parameter estimate for the effect of interest (b_cov1 below).
toplot <- mod %>%
  spread_draws(b_cov1, r_species[species,]) %>%
  mutate(species_mean = b_cov1 + r_species)

# b_cov1 is the slope (b) for the effect of interest (cov1). Your model might look
# something like: y ~ cov1 + (cov1|species). There's no 'b_cov1'! You can use the
# get_variables() command to find out what it's called:

get_variables(mod)
```

Then your data is ready for ggplot...

```
ggplot(toplot, aes(y = species, x = species_mean)) +
  geom_vline(xintercept = 0, linetype = "dashed", color = "black") + # vertical line at x=0
  stat_pointinterval(.width = c(0.9, 0.95)) + # plots 90% and 95% credible intervals
  theme_minimal() +
  labs(title = "Species-specific effects",
       x = "\u03b2 (cov1)", y = "") +
  scale_y_discrete(limits = rev(levels(toplot$species))) + # species in alphabetical order
  theme(axis.text.y = element_text(face = "italic")) # italicize species names
```



Resources

- brms overview: https://cran.r-project.org/web/packages/brms/readme/README.html
- brms vignette: https://cran.r-project.org/web/packages/brms/vignettes/brms_overview.pdf
- A Brief Guide to Stan's warnings: https://mc-stan.org/misc/warnings.html
- Visualizing brms models with tidybayes: https://mjskay.github.io/tidybayes/articles/tidy-brms.html
- Visualizing the Bayesian workflow in R (prior & posterior predictive checks): https://www.monicaalexander.com/posts/2020-28-02-bayes_viz/
- Cross validation FAQ: https://avehtari.github.io/modelselection/CV-FAQ.html
- Statistical rethinking with brms, ggplot2, and the tidyverse: https://bookdown.org/ajkurz/Statistical_Rethinking_recoded/
- Phylogenetic regressions with brms: https://cran.r-project.org/web/packages/brms/vignettes/brms_phylogenetics.html
- Intro to hierarchical models: http://mfviz.com/hierarchical-models/