demo brms

Fabian Jörg Fischer

1. Introduction

This is an R Markdown Notebook to give a short demo of how to use the *brms* package in R.

What this is:

- a quick introduction
- overview over typical workflow

What this is not:

- an introduction to Bayesian statistics

Useful modelling maxims:

- use common sense and biological knowledge to build a model
- try to prove yourself and your model wrong

Useful books/tutorials:

- McElreath: Statistical Rethinking
- Gelman et al.: Regression and Other Stories
- other tutorials (https://ourcodingclub.github.io/tutorials/brms/, but beware: pp_check assessment is odd)

2. Sample data: Tallo

Throughout this document, I will use Tommaso's Tallo database (https://zenodo.org/records/6637599) as an example to work with. For simplicity, I have added a few additional variables to the database: biome and biogeographic realm (Dinerstein et al. 2017, BioScience), as well as climatic predictors from CHELSA/BIOCLIM+ (Karger et al., 2017, Scientific Data; Brun et al., 2022, Earth System Science Data). We will load it directly from my github page.

The climatic predictors are yearly averages: *t_chelsa* (temperature, degree), *ppt_chelsa* (precipitation, mm), *ws_chelsa* (wind speed, m/s), *swb_chelsa* (soil water balance, kg/m2), *ai_chelsa* (aridity index, unitless).

```r
# standard libraries for our purposes
library(data.table) # fast data manipulation
library(brms) # Bayesian modelling
library(rnaturalearth) # for subsetting geographic data
library(rnaturalearthdata) # for subsetting geographic data
library(terra) # for subsetting/plotting geographic data
library(ggplot2) # plotting
library(viridis) # plotting
library(patchwork) # plotting

# set the working directory to where the current file is
path = dirname(rstudioapi::getSourceEditorContext()$path)
setwd(path)

# define paths
url_tallo = "https://github.com/fischer-fjd/DemoBRMS/blob/main/tallo.RData?raw=true"
dir_data = "data"
if(!dir.exists(dir_data)) dir.create(dir_data)
path_tallo = file.path(dir_data, "tallo.RData")

# create directory
if(!dir.exists(dir_data)) dir.create(dir_data)

# download file
# download.file(url_tallo, destfile = path_tallo,quiet = F) # commented out so that we don't have to re

# load tallo (read as data.table)
load(path_tallo)
print(tallo) # always look at your data first for a while
```

3. Sample question: How are tree height and crown radius related?

Motivation:

- ecologically: how do trees allocate resources to growth?
- practically: can we measure only one of them and predict the other?

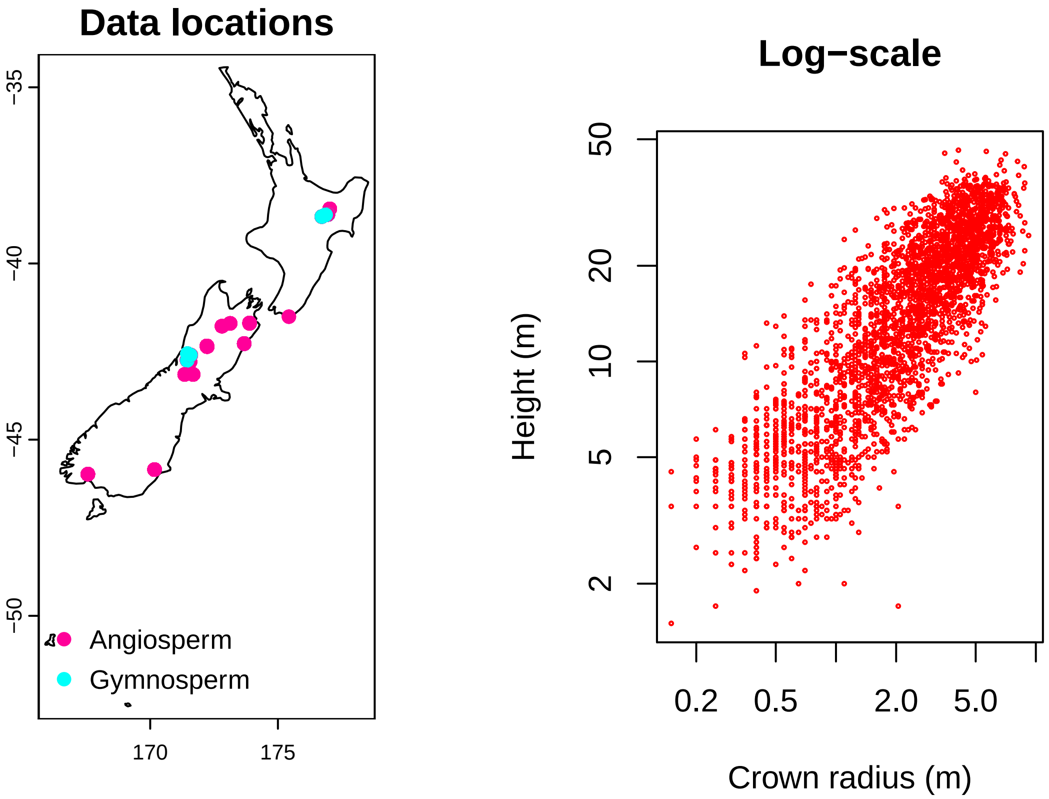
Expectations:

- positive relationship: taller trees need to sustain their biomass through more photosynthesis, so they need larger crowns
- linear relationship on log-scales: growth and size of organisms often follow lognormal distributions

To make life simpler, we use a Tallo subset (New Zealand).

```r
# we add log-scale transformations to tallo
tallo$stem_diameter_log = log(tallo$stem_diameter_cm)
tallo$height_log = log(tallo$height_m)
tallo$crown_radius_log = log(tallo$crown_radius_m)

# get shapefile for New Zealand
nz_vect = vect(ne_countries(geounit = "New Zealand",scale = 50))
nz_vect = aggregate(crop(nz_vect, ext(150,180,-90,90)))

# convert tallo to terra vector
tallo_vect = vect(tallo, geom = c("longitude","latitude"), crs = "EPSG:4326", keepgeom = T)
tallo_vect_nz = intersect(tallo_vect, nz_vect)
```

```
# backconvert to data.table for easier manipulation
tallo_nz = as.data.table(tallo_vect_nz)

# plot
{
  par(mfrow = c(1,2))
  plot(nz_vect, main = "Data locations")
  plot(tallo_vect_nz, "division", add = T, legend = "bottomleft")
  plot(tallo_vect_nz$crown_radius_m, tallo_vect_nz$height_m, cex = 0.25, col = "red", xlab = "Crown radi
}
```



We build a simple model and find, indeed, a positive effect of height on crown radius (log-scale).

```
m1_nz = lm(crown_radius_log ~ height_log, tallo_nz)
summary(m1_nz)
```

```
##
## Call:
## lm(formula = crown_radius_log ~ height_log, data = tallo_nz)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.55538 -0.27495  0.03437  0.30466  2.10660
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.93797    0.03385  -57.24   <2e-16 ***
## height_log   1.03502    0.01270   81.51   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4389 on 2690 degrees of freedom
##   (24 observations deleted due to missingness)
## Multiple R-squared:  0.7118, Adjusted R-squared:  0.7117
## F-statistic:  6644 on 1 and 2690 DF,  p-value: < 2.2e-16
```

We note the coefficients for later:

```
print(m1_nz)
```

```
##
## Call:
## lm(formula = crown_radius_log ~ height_log, data = tallo_nz)
##
## Coefficients:
## (Intercept)   height_log
##      -1.938        1.035
```

4 How to evaluate this model? How do we prove it wrong?

Before we attempt to do create the same model and refine it with *brms*, we think about how we could evaluate models more generally. Any ideas?

4.1 Suggestion: Data splitting One great way of testing models is data splitting:

1. Split data into a number of subsets
2. Remove one subset (test data)
3. Fit model to the remaining subsets (training data)
4. Predict the test data
5. Repeat with each subset
6. Evaluate goodness of prediction (R2, RMSE, MAE)

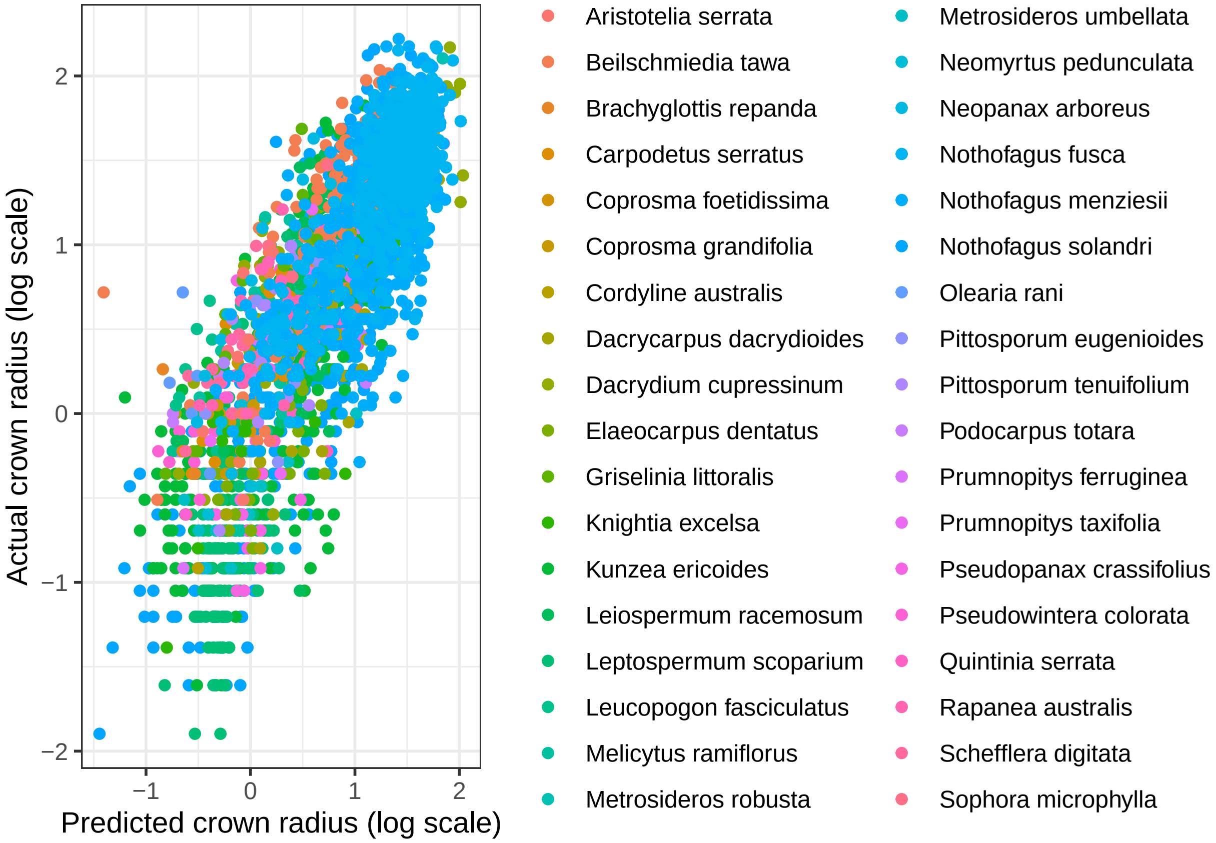
Advantages:

- summarizes scientific process: if I had to use my model in a new context, would I expect it to generalize?
- model agnostic: works with any model (linear regression, linear mixed models, random forest, deep learning)
- flexible: split by taxonomic group, by geography, by temporal unit

Recommended paper: Ploton et al. 2020, Nat. Comm.

4.2 Example: split by species We could call this a "leave-one-species-out" validation:

```
species_nz = unique(tallo_nz$species)
predictions_byspecies = data.table(species = character(), crown_radius_log = numeric(), crown_radius_log

for(species_current in species_nz){
  cat("Current species:",species_current,"\n")
  # split data by species
  tallo_test = tallo_nz[species == species_current]
  tallo_train = tallo_nz[species != species_current]

  # train model on training data
  m1_train = lm(crown_radius_log ~ height_log, tallo_train)

  # predict to test data
  crown_radius_log_pred = predict(m1_train, newdata = tallo_test)
  predictions_test = data.table(species = species_current, crown_radius_log = tallo_test$crown_radius_l
  predictions_byspecies = rbind(predictions_byspecies, predictions_test)
}

ggplot(data = predictions_byspecies, aes(x = crown_radius_log_pred, y = crown_radius_log, col = species
```



We evaluate how good the predictions are numerically:

```
(R2 = round(cor(predictions_byspecies$crown_radius_log,predictions_byspecies$crown_radius_log_pred, use
```

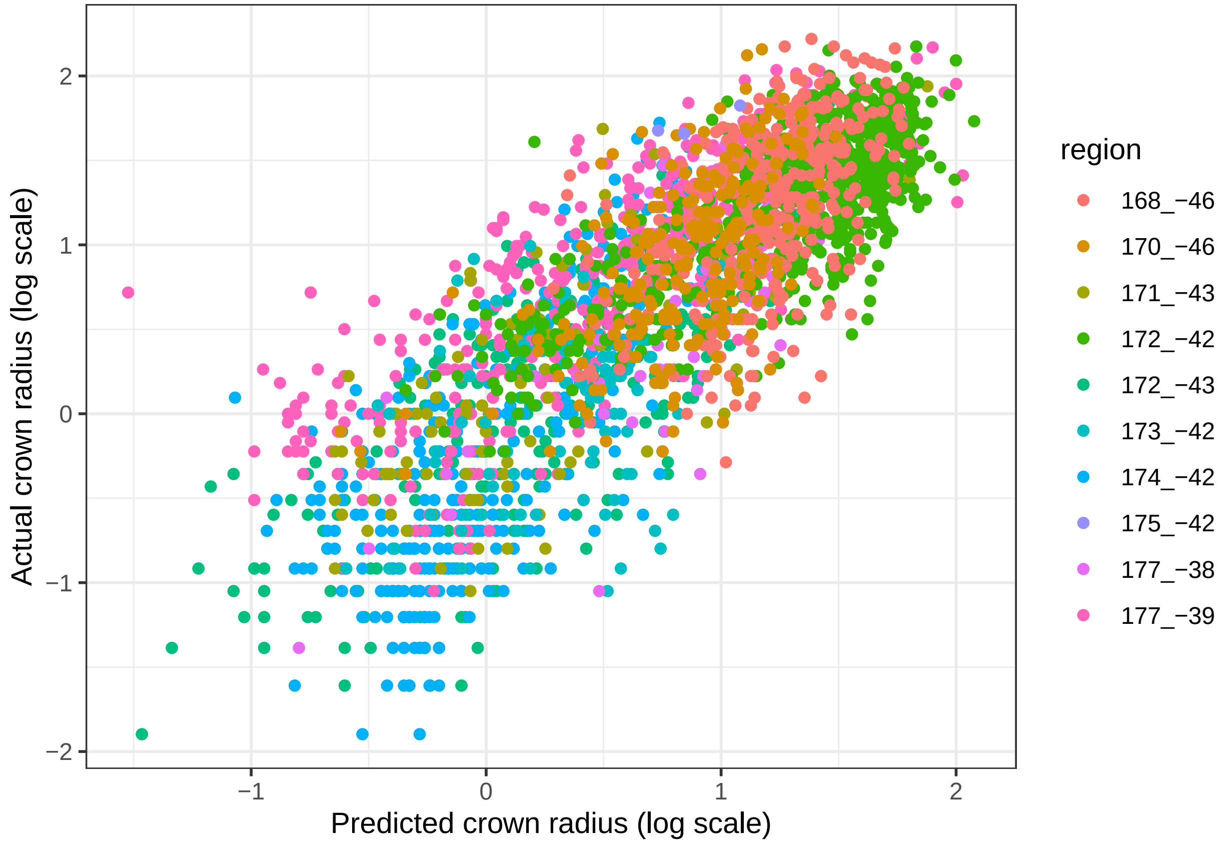
```
## [1] 0.7
```

```
(RMSE = round(sqrt(mean((predictions_byspecies$crown_radius_log-predictions_byspecies$crown_radius_log_
```

```
## [1] 0.45
```

4.3 Example: split by region We group by rounded longitude/latitude. We could call this a "leave-one-region-out" validation:

```
tallo_nz[, region := paste0(round(longitude),"_",round(latitude))]
region_nz = unique(tallo_nz$region)
predictions_byregion = data.table(region = character(), crown_radius_log = numeric(), crown_radius_log_p

for(region_current in region_nz){
  cat("Current region:",region_current,"\n")

  # split data by region
  tallo_test = tallo_nz[region == region_current]
  tallo_train = tallo_nz[region != region_current]

  # train model on training data
  m1_train = lm(crown_radius_log ~ height_log, tallo_train)

  # predict to test data
  crown_radius_log_pred = predict(m1_train, newdata = tallo_test)
  predictions_test = data.table(region = region_current, crown_radius_log = tallo_test$crown_radius_log
  predictions_byregion = rbind(predictions_byregion, predictions_test)
}

ggplot(data = predictions_byregion, aes(x = crown_radius_log_pred, y = crown_radius_log, col = region))
```



```
(R2 = round(cor(predictions_byregion$crown_radius_log,predictions_byregion$crown_radius_log_pred, use =
```

```
## [1] 0.69
```

```
(RMSE = round(sqrt(mean((predictions_byregion$crown_radius_log-predictions_byregion$crown_radius_log_pre
```

```
## [1] 0.45
```

5 Finally: brms

5.1 What is brms?

brms is a package that links to the *STAN* software (https://mc-stan.org/), which is mostly used for Bayesian inference.

5.2 What is Bayesian inference/modelling and why should we care?

Conceptually: in a Bayesian context, we can think of a model as a probability distribution of the model parameters. What we want to know is: given our prior knowledge and the data, how likely are specific parameter combinations.

Comparison with other statistical frameworks:

- priors: narrow down the parameter combinations before applying a mathematical model –> useful both for small data sets (reduce uncertainty) and large data sets (reduce model complexity)
- flexibility: Bayesian models typically rely on sampling algorithms –> easy to extend to non-linear problems / non-normal error distributions

So why have they not been used before?

5.3 How do we set up a model in brms

In the past, most Bayesian models had to be hardcoded in *STAN*, but brms provides a translation into typical R syntax, so we can fit the exact same model as before:

```
# probably faster to run outside the code chunk
m1brm_nz = brm(
  crown_radius_log ~ height_log # same as in lm(), lmer(), etc.
  , data = tallo_nz
  , iter = 2500 # total number of iterations
  , warmup = 1000 # iterations that are used for "warmup"
  , chains = 4 # we sample the surface 4 times separately for robustness
  , cores = 4 # parallelization
  , control = list(adapt_delta = 0.95, max_treedepth = 10) # HMC parameters
  , silent = F
  , seed = 1 # for reproducibility, fix seed
)
```

```
## Compiling Stan program...
```

```
## Trying to compile a simple C file
```

```
## Start sampling
```

Now we can evaluate the results just as before.

```
summary(m1brm_nz)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: crown_radius_log ~ height_log
##    Data: tallo_nz (Number of observations: 2692)
##   Draws: 4 chains, each with iter = 2500; warmup = 1000; thin = 1;
##          total post-warmup draws = 6000
##
## Regression Coefficients:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept    -1.94      0.03    -2.00    -1.87 1.00     5264     3750
## height_log     1.03      0.01     1.01     1.06 1.00     5480     3628
##
## Further Distributional Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.44      0.01     0.43     0.45 1.00     5986     3609
##
## Draws were sampled 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).
```

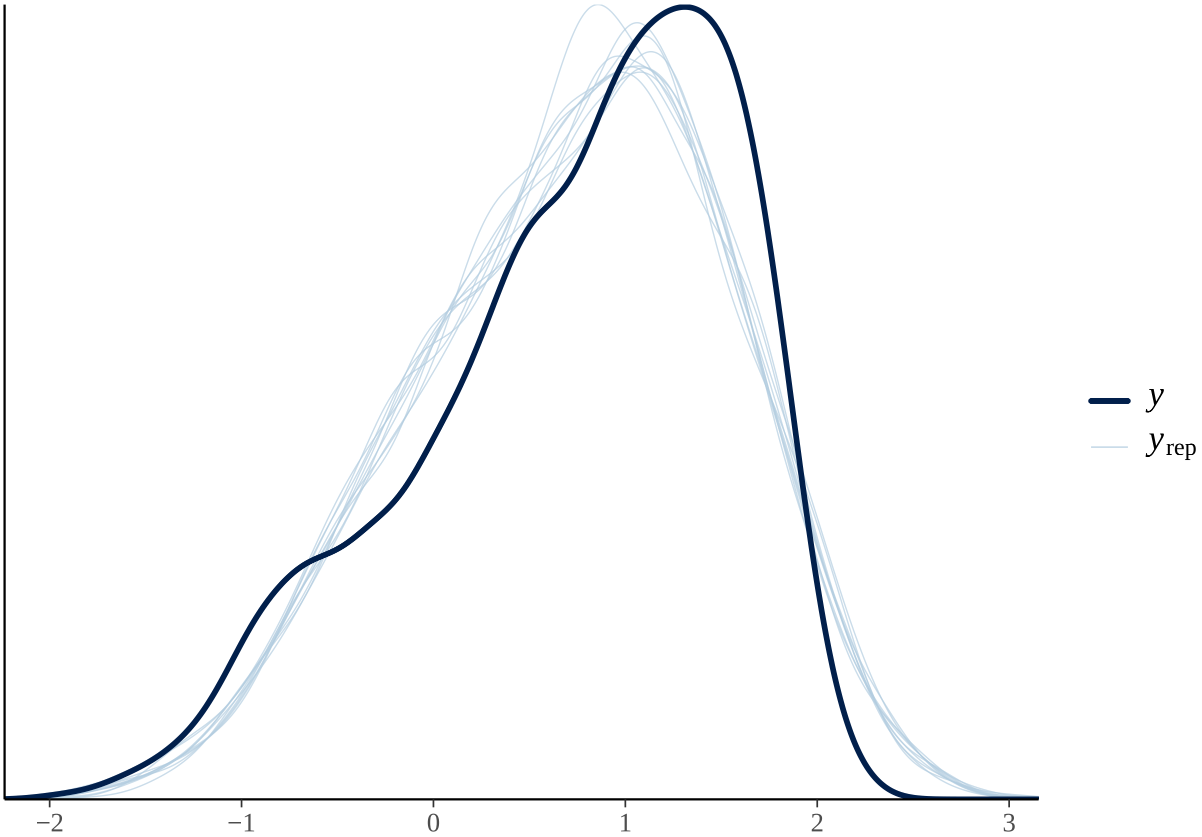
5.4 How do we evaluate a model? Key things to look at first:

- warnings: there should be no warnings, no divergent transitions, etc.!
- Rhat: needs to be < 1.01-1.02
- ESS: should be > 1000, but better larger

Then we check how well the model reproduces the data (posterior predictive check)

```
pp_check(m1brm_nz)
```

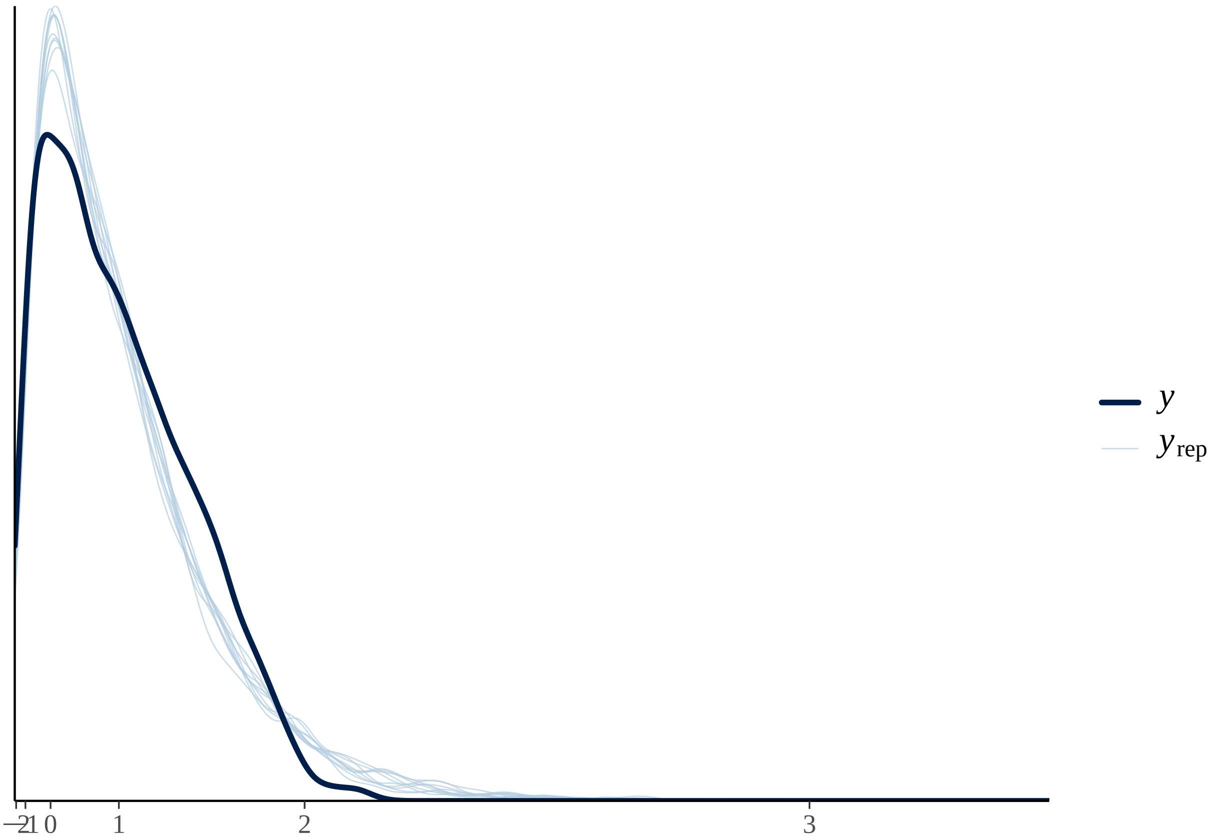
```
## Using 10 posterior draws for ppc type 'dens_overlay' by default.
```



On original scales

```
pp_check(m1brm_nz) + scale_x_continuous(trans = "exp")
```

```
## Using 10 posterior draws for ppc type 'dens_overlay' by default.
```



5.5 Could we improve this model? Potentially, the slope varies between species (we have seen above that it probably does not vary massively), so we try this with a mixed effects model.

```
# probably faster to run outside the code chunk
m2brm_nz = brm(
  crown_radius_log ~ height_log + (height_log | species) # same as in lm(), lmer(), etc.
  , data = tallo_nz
  , iter = 2500 # total number of iterations
  , warmup = 1000 # iterations that are used for "warmup"
  , chains = 4 # we sample the surface 4 times separately for robustness
  , cores = 4 # parallelization
  , control = list(adapt_delta = 0.95, max_treedepth = 10) # HMC parameters
  , silent = F
  , seed = 1 # for reproducibility, fix seed
)
```

```
## Compiling Stan program...
```

```
## Trying to compile a simple C file
```

```
## Start sampling
```

Note: A model is typically easy to fit/unproblematic when all chains run in a similar time.

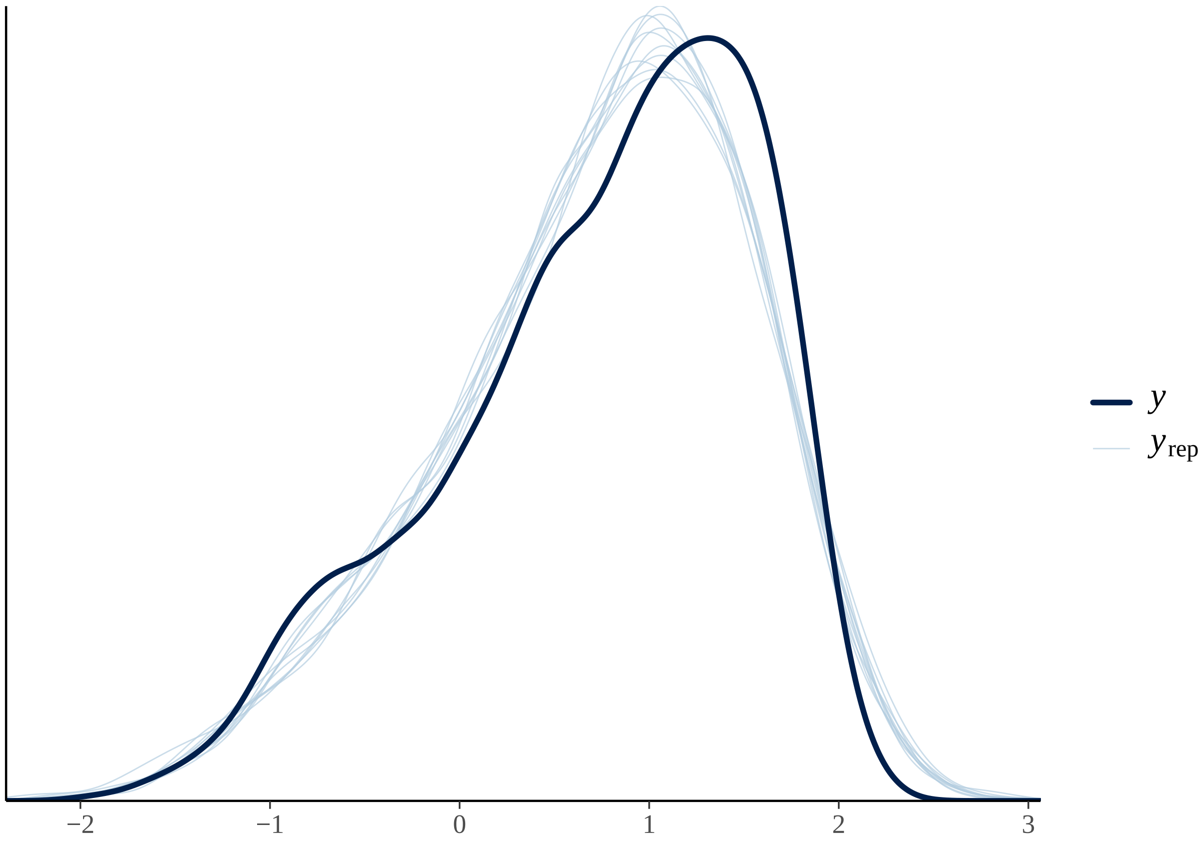
Again, we check the results.

```
summary(m2brm_nz)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: crown_radius_log ~ height_log + (height_log | species)
##    Data: tallo_nz (Number of observations: 2692)
##   Draws: 4 chains, each with iter = 2500; warmup = 1000; thin = 1;
##          total post-warmup draws = 6000
##
## Multilevel Hyperparameters:
## ~species (Number of levels: 36)
##                           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
## sd(Intercept)                 0.62      0.10     0.46     0.85 1.00     1509
## sd(height_log)                0.22      0.04     0.15     0.31 1.00     1559
## cor(Intercept,height_log)    -0.95      0.03    -0.98    -0.88 1.00     2119
##                           Tail_ESS
## sd(Intercept)                 2369
## sd(height_log)                2315
## cor(Intercept,height_log)     3440
##
## Regression Coefficients:
##            Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     -1.60      0.13    -1.85    -1.34 1.00     1488     2450
## height_log     0.94      0.05     0.84     1.03 1.00     1388     2381
##
## Further Distributional Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.39      0.01     0.38     0.40 1.00     9166     4186
##
## Draws were sampled 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).
```

```r
pp_check(m2brm_nz)
```

```
## Using 10 posterior draws for ppc type 'dens_overlay' by default.
```



This did not seem to massively improve the fitting, as anticipated.

5.6 What about the priors? *brms* actually runs the models implicitly with what's called "flat priors", i.e., very few prior assumptions about the parameters.

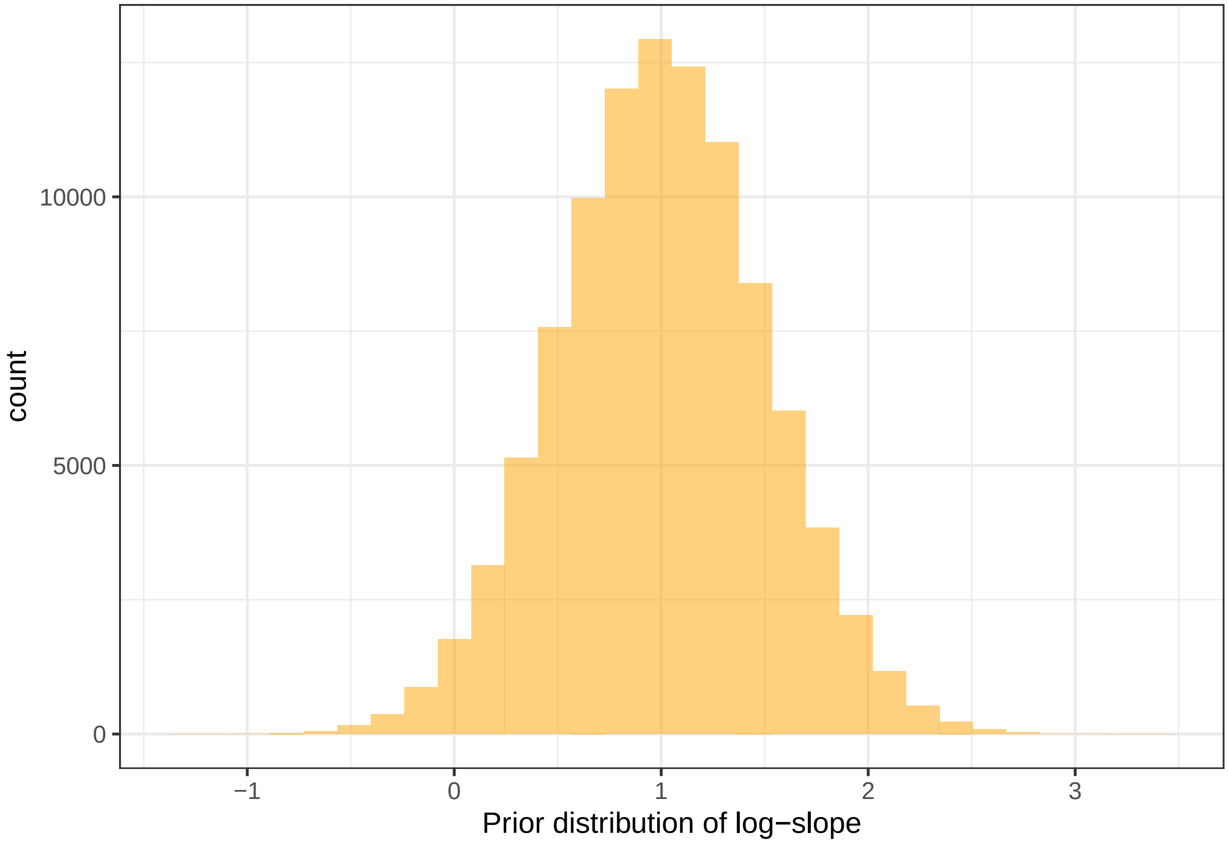
```
m2brm_nz$prior
```

```
##                  prior     class       coef   group resp dpar nlpar lb ub
##                 (flat)         b                                         
##                 (flat)         b height_log                              
##  student_t(3, 0.9, 2.5) Intercept                                        
##   lkj_corr_cholesky(1)         L                                         
##   lkj_corr_cholesky(1)         L            species                      
##    student_t(3, 0, 2.5)        sd                                    0   
##    student_t(3, 0, 2.5)        sd            species                 0   
##    student_t(3, 0, 2.5)        sd height_log species                 0   
##    student_t(3, 0, 2.5)        sd  Intercept species                 0   
##    student_t(3, 0, 2.5)     sigma                                    0   
##        source
##       default
##  (vectorized)
##       default
##       default
##  (vectorized)
##       default
##  (vectorized)
##  (vectorized)
##  (vectorized)
##       default
```

But we can explicitly set the priors to become more informative. For example:

```
# for example, we might now that the slope has to be around 1
normal_truncated = rnorm(100000,mean = 1.0, sd = 0.5)
ggplot() + geom_histogram(aes(x = normal_truncated), fill = "orange", alpha = 0.5) + theme_bw() + xlab(
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We rerun:

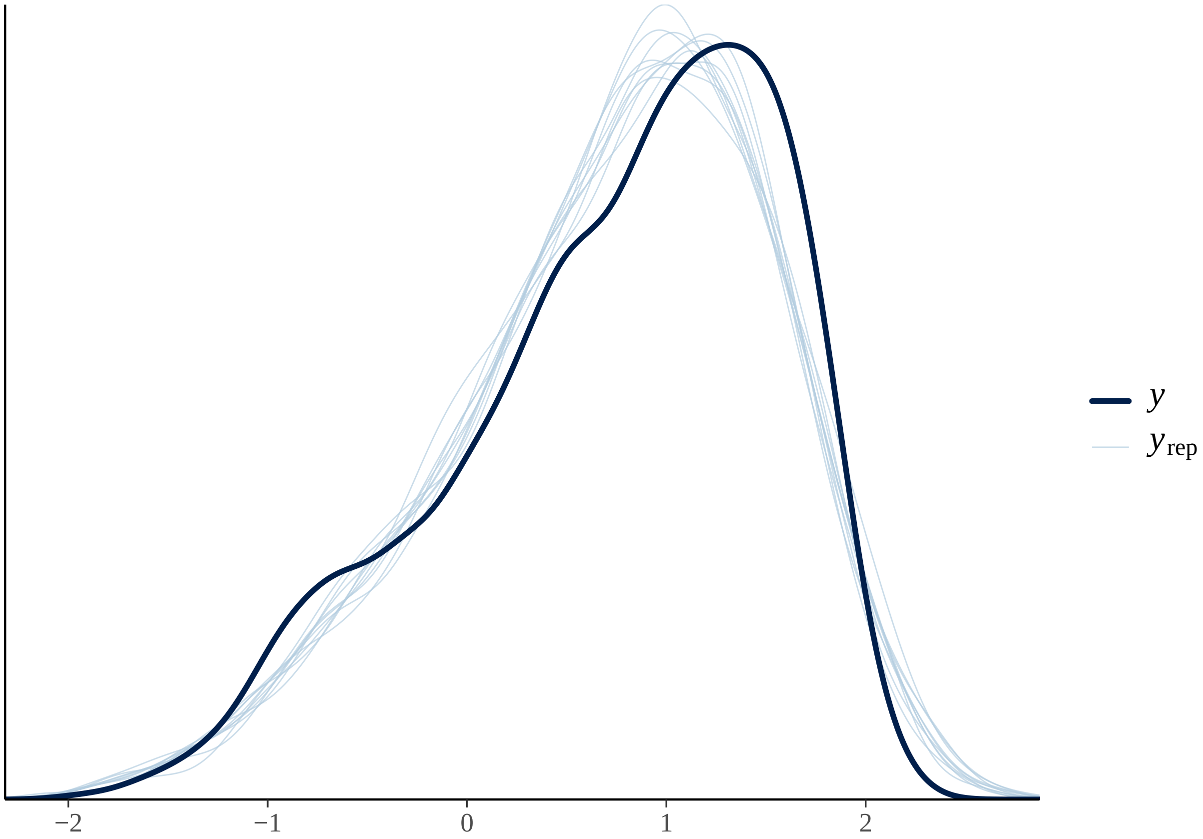
```
# for example, we might now that
prior_m2brms_nz = c(
  prior(normal(1.0,0.5), class = "b", coef = "height_log")
)

m2brm_nz_withprior = brm(
  crown_radius_log ~ height_log + (height_log | species) # same as in lm(), lmer(), etc.
  , data = tallo_nz
  , prior = prior_m2brms_nz
  , iter = 2500 # total number of iterations
  , warmup = 1000 # iterations that are used for "warmup"
  , chains = 4 # we sample the surface 4 times separately for robustness
  , cores = 4 # parallelization
  , control = list(adapt_delta = 0.95, max_treedepth = 10) # HMC parameters
  , silent = F
  , seed = 1 # for reproducibility, fix seed
)
```

```
## Compiling Stan program...

## Trying to compile a simple C file

## Start sampling
```

And check:

```r
pp_check(m2brm_nz_withprior)
```

```
## Using 10 posterior draws for ppc type 'dens_overlay' by default.
```



5.7 Could we maybe fit this as a non-linear model? Behind the linear regression on log-scales is actually a non-linear function (a power law). In *brms* and *STAN*, we can directly write model formulas via the bf() formula interface.

```r
# we specify model beforehand to be clearer
m3brm = bf(
  crown_radius_m ~ exp(intercept + slope * height_log) # same as in lm(), lmer(), etc.
  , intercept ~ 1 # we only model the parameter as intercept (i.e., independent of other predictors)
  , slope ~ 1 # we only model the parameter as intercept (i.e., independent of other predictors)
  , nl = T
)

# we check the priors
get_prior(m3brm, data = tallo_nz)

prior_m3brm_nz = c(
  prior(normal(-2.0,2.0), class = "b", nlpar = "intercept")
  , prior(normal(1.0,1.0), class = "b", nlpar = "slope", lb = 0)
)

# probably faster to run outside the code chunk
m3brm_nz = brm(
  formula = m3brm
  , prior = prior_m3brm_nz
  , data = tallo_nz
  , iter = 2500 # total number of iterations
  , warmup = 1000 # iterations that are used for "warmup"
  , chains = 4 # we sample the surface 4 times separately for robustness
  , cores = 4 # parallelization
  , control = list(adapt_delta = 0.95, max_treedepth = 10) # HMC parameters
  , silent = F
  , seed = 1 # for reproducibility, fix seed
)
```

```
## Compiling Stan program...
```

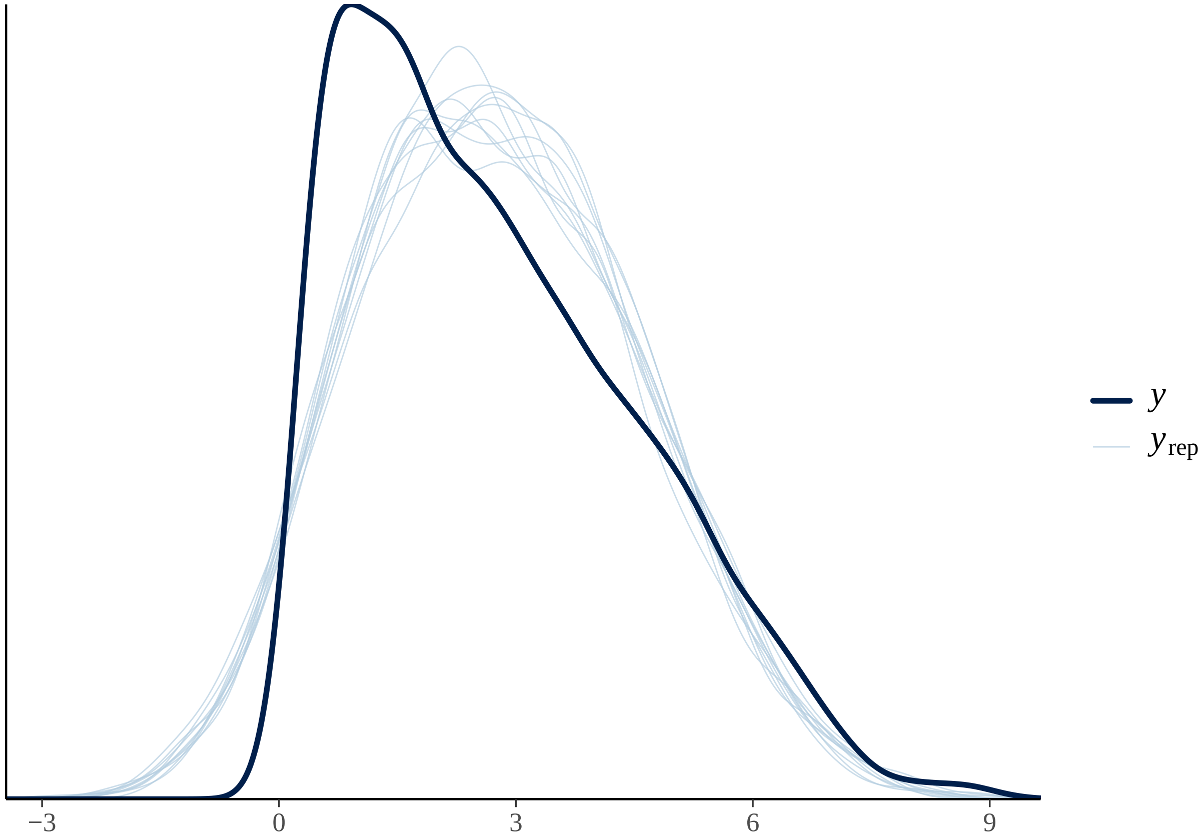
```
## Trying to compile a simple C file
```

```
## Start sampling
```

And check:

```
pp_check(m3brm_nz)
```

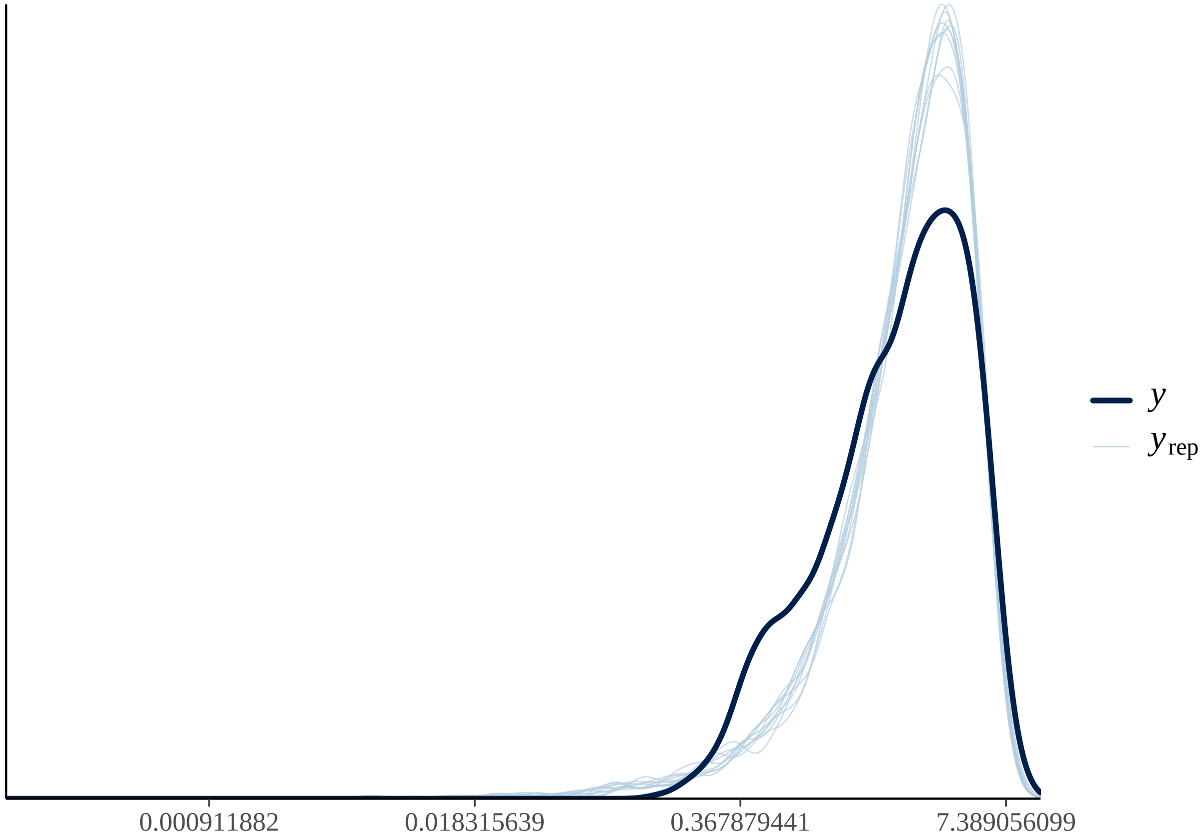
```
## Using 10 posterior draws for ppc type 'dens_overlay' by default.
```



Better log-transform for comparability:

```r
pp_check(m3brm_nz) + scale_x_continuous(trans = "log")
```

```
## Using 10 posterior draws for ppc type 'dens_overlay' by default.
```



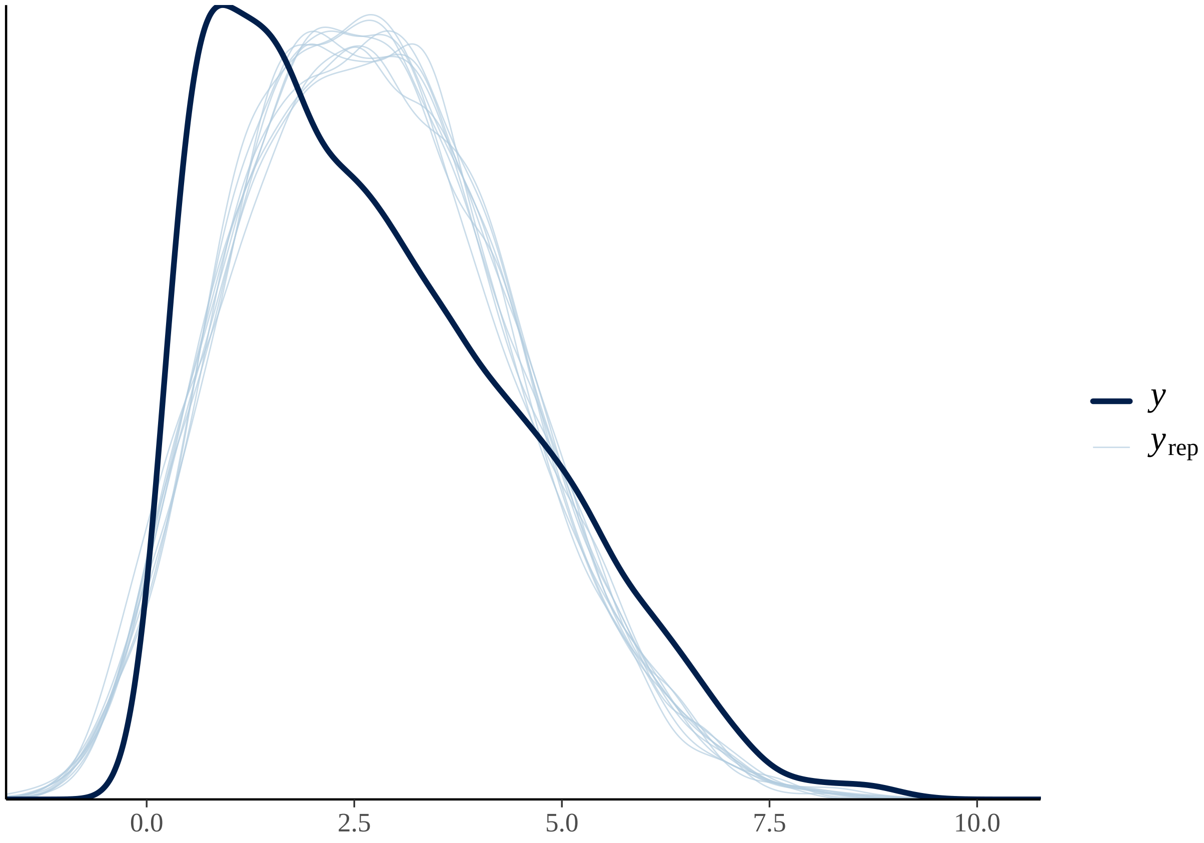
5.8 Maybe with an asymmetric error distribution and species-specific values? Maybe we just need a better error distribution

```r
# we specify model beforehand to be clearer
m4brm = bf(
  crown_radius_m ~ exp(intercept + slope * height_log) # same as in lm(), lmer(), etc.
  , intercept ~ 1 + (1 | species) # we let the intercept vary by species (random effect)
  , slope ~ 1 + (1 | species) # we let the slope vary by species (random effect)
  , nl = T
)

# we check the priors
get_prior(m4brm, data = tallo_nz)

prior_m4brm_nz = c(
  prior(normal(-2.0,2.0), class = "b", nlpar = "intercept")
  , prior(normal(1.0,1.0), class = "b", nlpar = "slope", lb = 0)
)

# probably faster to run outside the code chunk
m4brm_nz = brm(
  formula = m4brm
  , prior = prior_m4brm_nz
  , data = tallo_nz
  , family = "skew_normal"
  , iter = 2500 # total number of iterations
  , warmup = 1000 # iterations that are used for "warmup"
  , chains = 4 # we sample the surface 4 times separately for robustness
  , cores = 4 # parallelization
  , control = list(adapt_delta = 0.95, max_treedepth = 10) # HMC parameters
  , silent = F
  , seed = 1 # for reproducibility, fix seed
)
```

```
## Compiling Stan program...

## Trying to compile a simple C file

## Start sampling
```

```
pp_check(m4brm_nz)
```

```
## Using 10 posterior draws for ppc type 'dens_overlay' by default.
```



5.9 What when things go wrong?

Things to check

- Check the data
- Model too complex? Remove components?
- Different error distribution?
- Standardize your variables?
- Reparameterize your variables? [advanced, e.g. exp(intercept) = intercept_star]

5.10 Words of warning Bayesian models are no silver bullet. They offer many advantages (bronze bullet?), but common issues with model building continue to exist, e.g.:

- models that can reproduce/predict the data are not always useful (correlation != causation)
- models cannot correct for biased or noisy data
- models will not be able to differentiate between highly correlated predictors