Chapter 3: Computational Bayesian data analysis

Shravan Vasishth (vasishth.github.io)

June 2025

Contents

Textbook

Introduction to Bayesian Data Analysis for Cognitive Science

Nicenboim, Schad, Vasishth

- Online version: https://bruno.nicenboim.me/bayescogsci/
- Source code: https://github.com/bnicenboim/bayescogsci
- Physical book: here

Be sure to read the textbook's chapters 1-3 in addition to watching this lecture.

Linear modeling

Suppose y is a vector of continuous responses; assume for now that the y are independent and identically distributed:

$y \overset{iid}{\sim} Normal(\mu, \sigma)$

This is the simple linear model:

$y = \mu + \varepsilon$ where $\varepsilon \sim Normal(0, \sigma)$

There are two parameters, μ, σ, so we need priors on these. We expand on this simple model next.

The way we will conduct data analysis is as follows:

- Given data, specify a **likelihood function**.
- Specify **prior distributions** for model parameters.
- Evaluate whether model makes sense, using simulated data, **prior predictive** and

posterior predictive checks, and (if you want to claim a discovery) calibrating true and false discovery rates.

- Using software, derive **marginal posterior distributions** for parameters given a likelihood function and prior density. I.e., simulate parameters to get **samples from posterior distributions** of parameters using some **Markov Chain Monte Carlo (MCMC) sampling algorithm**.
- Check that the model converged using **model convergence** diagnostics,
- Summarize **posterior distributions** of parameter samples and make your scientific decision.

We will now work through some specific examples to illustrate how the data analysis process works.

Example 1: A single subject pressing a button repeatedly

As a first example, we will fit a simple linear model to some reaction time data.

The data frame df_spacebar contains data of a subject pressing the space bar without reading in a self-paced reading experiment.

Install the library

Install the library bcogsci. See:

https://github.com/bnicenboim/bcogsci

Load the data

```
library(bcogsci)
data("df_spacebar")
head(df_spacebar)
```

```
## # A tibble: 6 x 2
##       t trial
##   <int> <int>
## 1   141     1
## 2   138     2
## 3   128     3
## 4   132     4
## 5   126     5
## 6   134     6
```

Visualizing the data

It is a good idea to look at the distribution of the data before doing anything else. See Figure 1.

```
plot(density(df_spacebar$t),
     main="Button-press data",
     xlab="Tapping time")
```

The data looks a bit skewed, but we ignore this for the moment.

Define the likelihood function

Let's model the data with the following assumptions:

- There is a true underlying time, μ, that the participant needs to press the space-bar.
- There is some noise around μ.

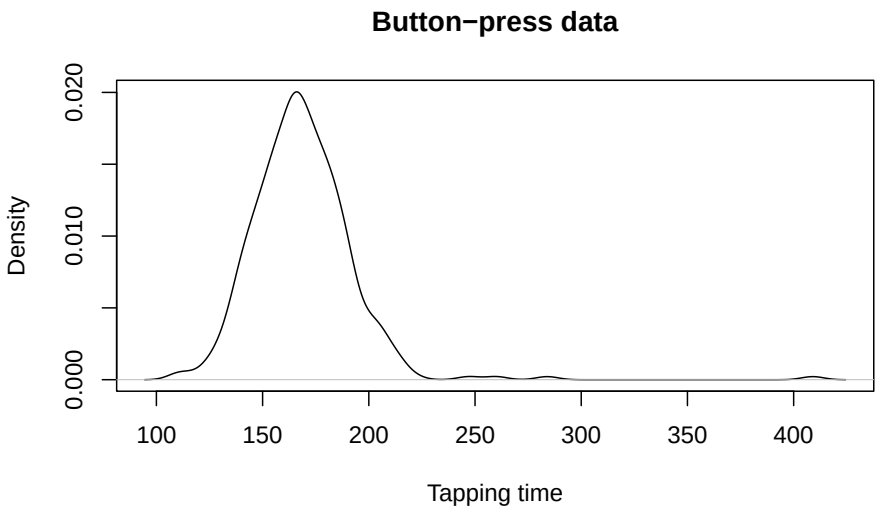


Figure 1: Visualizing the data.

- The noise is normally distributed (this assumption is questionable given the skew but; we fix this assumption later).

This means that the likelihood for each observation n will be:

$$t_n \sim Normal(\mu, \sigma) \tag{1}$$

where $n = 1 \ldots N$.

This is just the simple linear model:

$$t = \mu + \varepsilon \text{ where } \varepsilon \sim Normal(0, \sigma) \tag{2}$$

Define the priors for the parameters

We are going to use the following priors for the two parameters in this model:

$$\begin{aligned} &\mu \sim Normal(0, 2000) \\ &\sigma \sim Normal(0, 500) \text{ truncated so that } \sigma > 0 \end{aligned} \tag{3}$$

In order to decide on a prior for the parameters, always visualize them first. See the Figure below.

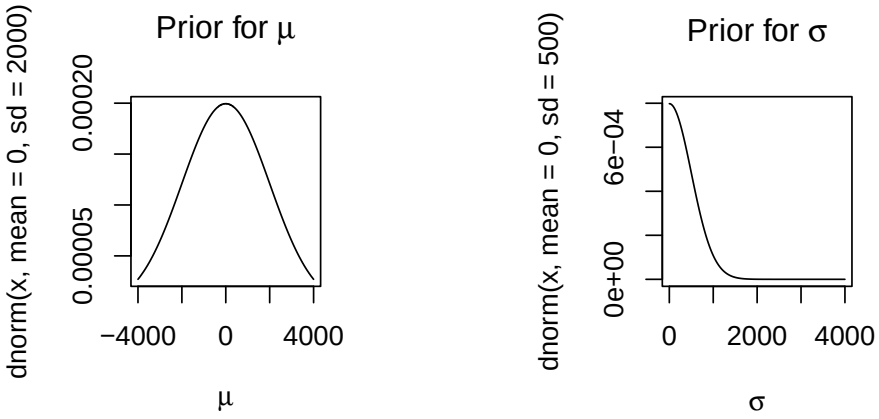


Figure 2: Visualizing the priors for example 1.

Prior predictive distribution

With these priors, we are going to generate something called the **prior predictive distribution**. This helps us check whether the priors make sense.

Formally, we want to know the density $f(\cdot)$ of data points $t_1, \ldots, t_n$, given a vector of priors Θ. In our example, $\Theta = \langle \mu, \sigma \rangle$. The prior predictive density is:

$$f(t_1, \ldots, t_n) = \int f(t_1) \cdot f(t_2) \cdots f(t_n) f(\Theta)\, d\Theta \tag{4}$$

In essence, we integrate out the parameters. Here is one way to do it in R:

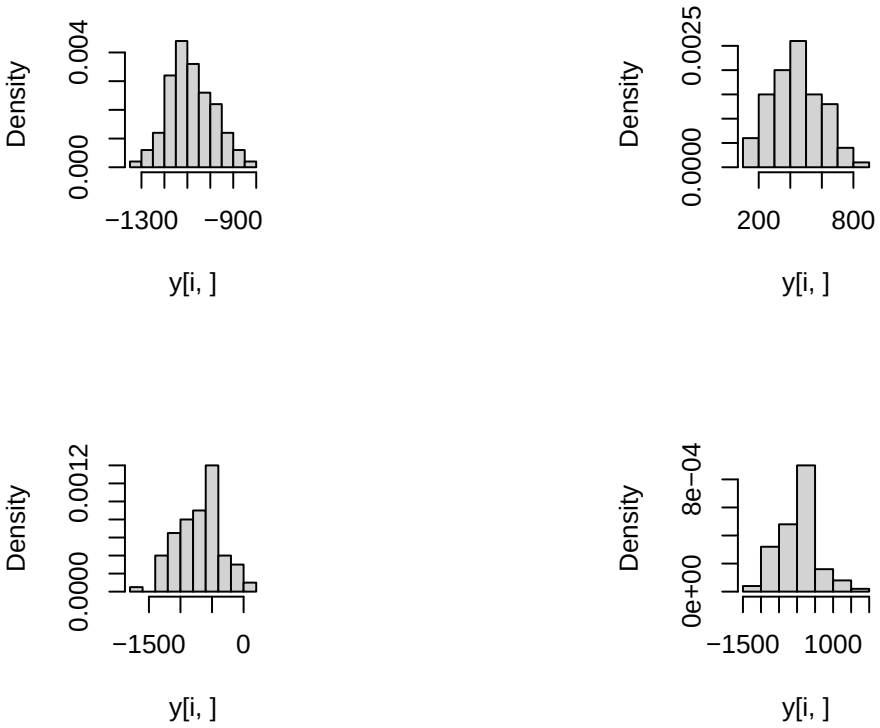
- Take one sample from each of the priors
- Generate *nobs* data points using those samples

This would give us a matrix containing nsim * nobs generated data. We can then plot the prior

predictive densities generated.

```
## needed for half-normal distribution
library(extraDistr)
## number of simulations
nsim<-1000
## number of observations generated
## each time:
nobs<-100
y<-matrix(rep(NA,nsim*nobs),ncol = nobs)
mu<-rnorm(nsim,mean=0,sd=2000)
## truncated normal, cut off at 0:
sigma<-rtnorm(nsim,mean=0,sd=500,a=0)

for(i in 1:nsim){
y[i,]<-rnorm(nobs,mean=mu[i],sd=sigma[i])
}
```

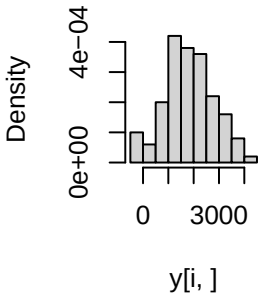


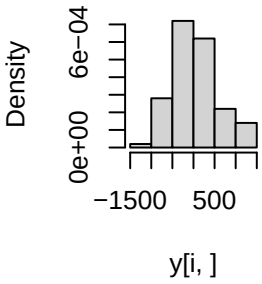
We can try to redefine the prior for μ to have only positive values, and then check again. We still get some negative values, but that is because we are assuming that

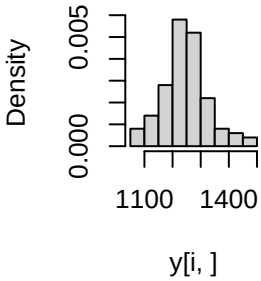
$$y \sim Normal(\mu, \sigma)$$

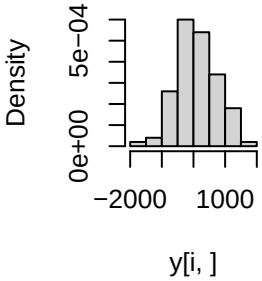
which will have negative values for small μ and large σ.

```
y<-matrix(rep(NA,nsim*nobs),ncol = nobs)
mu<-rtnorm(nsim,mean=0,sd=2000,a=0)
for(i in 1:nsim){
y[i,]<-rnorm(nobs,mean=mu[i],sd=sigma[i])
}
```









Generating prior predictive distributions using Stan

We can generate a prior predictive distribution using Stan as follows.

First, we define a Stan model that defines the priors and defines how the data are to be generated.

Documentation on Stan is available at mc-stan .org.

```
priorpred<-"data {
  int N;
}
parameters {
real<lower=0> mu;
real<lower=0> sigma;
}
model {
    mu ~ normal(0,2000);
    sigma ~ normal(0,500);
}
generated quantities {
  vector[N] y_sim;
  for(i in 1:N) {
    y_sim[i] = normal_rng(mu,sigma);
  }}"
```

Load RStan and brms.

```
## load rstan
library(rstan)
options(mc.cores = parallel::detectCores())
library(brms)
```

Then we generate the data:

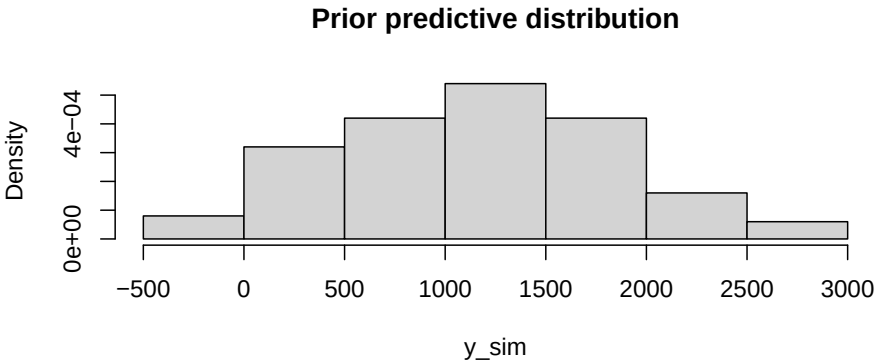
```
## generate 100 data-points
dat<-list(N=100)

## fit model:
m1priorpred<-stan(model_code=priorpred,
                  data=dat,
                  chains = 4,
                  warmup = 1000,
                iter = 2000)
```

```
## extract and plot one of the data-sets:
y_sim<-extract(m1priorpred,pars="y_sim")
str(y_sim)
```

```
## List of 1
##  $ y_sim: num [1:4000, 1:100] 3167 3058 534 2178 711 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ iterations: NULL
##   .. ..$           : NULL
```

```
## plot 100th simulated data set:
hist(y_sim$y_sim[100,],
     main="Prior predictive distribution",
     xlab="y_sim",freq=FALSE)
```



Fit the model in Stan

Having satisfied ourselves that the priors mostly make sense (do they, though?), we now fit the model to simulated data generated from known parameter values. The goal here is to ensure that the model recovers the true underlying parameters.

Next, we write the Stan model, adding a likelihood in the model block:

```
m1<-"data {
  int N;
```

```
  real y[N]; // data
}
parameters {
real<lower=0> mu;
real<lower=0> sigma;
}
model {
mu ~ normal(0,2000);
sigma ~ normal(0,500);
y ~ normal(mu,sigma);
}
generated quantities {
  vector[N] y_sim;
  for(i in 1:N) {
    y_sim[i] = normal_rng(mu,sigma);
  }}
"
```

Then generate simulated data with known parameter values (we decide what these are):

```
set.seed(123)
N <- 500
true_mu <- 400
true_sigma <- 125
y <- rnorm(N, true_mu, true_sigma)

y <- round(y)
sim_data <- data.frame(y=y)
dat<-list(y=y,N=N)
```

Finally, we fit the model:

```
## fit model:
m1rstan<-stan(model_code=m1,
```

```
                    data=dat,
                    chains = 4,
                  iter = 2000)

## extract posteriors:
posteriors<-extract(m1rstan,
              pars=c("mu","sigma"))
```

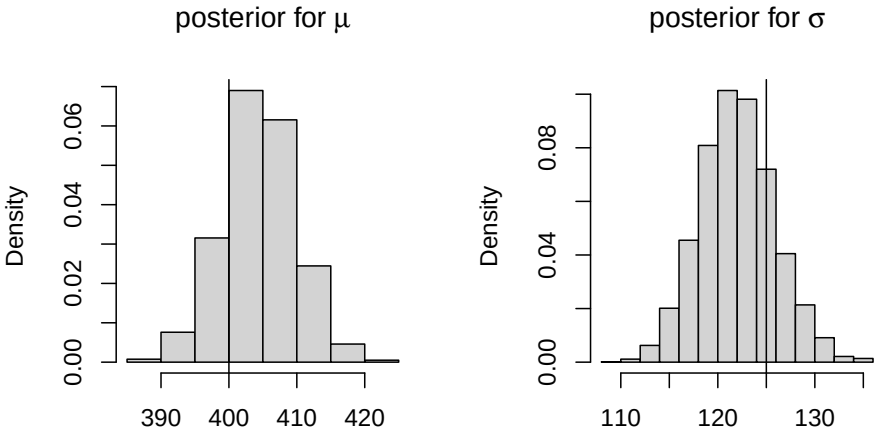


Figure 3: Posteriors from fake data, model m1. Vertical lines show the true values of the parameters.

Posterior predictive checks

Once we have the posterior distribution $f(\Theta \mid y)$, we can derive the predictions based on this posterior distribution:

$$p(y_{pred} \mid y) = \int p(y_{pred}, \Theta \mid y)\, d\Theta = \int p(y_{pred} \mid \Theta, y) p(\Theta \mid y)\, d\Theta \tag{5}$$

Assuming that past and future observations are conditionally independent given Θ, i.e., $p(y_{pred} \mid \Theta, y) = p(y_{pred} \mid \Theta)$, we can write:

$$p(y_{pred} \mid y) = \int p(y_{pred} \mid \Theta) p(\Theta \mid y)\, d\Theta \tag{6}$$

Note that we are conditioning y_{pred} only on y, we do not condition on what we don't know (Θ); **we integrate out the unknown parameters**.

This posterior predictive distribution is different from the frequentist approach, which gives only a predictive distribution of y_{pred} given our estimate of θ (a point value).

In the Stan code above, we have already generated the posterior predictive distribution, in the generated quantities block.

Implementing the model in brms

This model is expressed in `brms` in the following way. First, define the priors:

```
priors <- c(set_prior("normal(0, 2000)",
                      class = "Intercept"),
            set_prior("normal(0, 500)",
                      class = "sigma"))
```

Then, define the generative process assumed:

```
m1brms<-brm(t~1,df_spacebar,prior = priors,
       iter = 2000,
       warmup = 1000,
       chains = 4,
       family = gaussian(),
       control = list(adapt_delta = 0.99))
```

Summarizing the posteriors, and convergence diagnostics

A graphical summary of posterior distributions of model m1 is shown below:

The trace plots below show how well the four

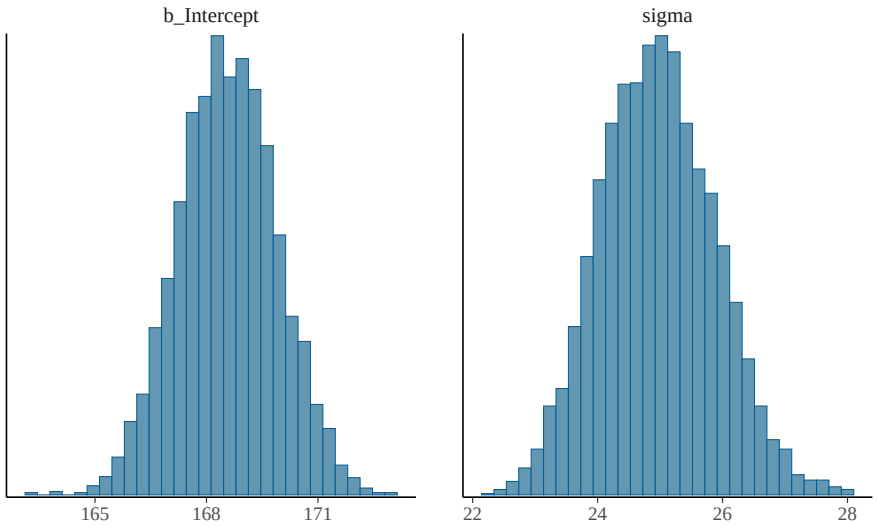


Figure 4: Posterior distributions of the parameters in model m1.

chains are mixing:

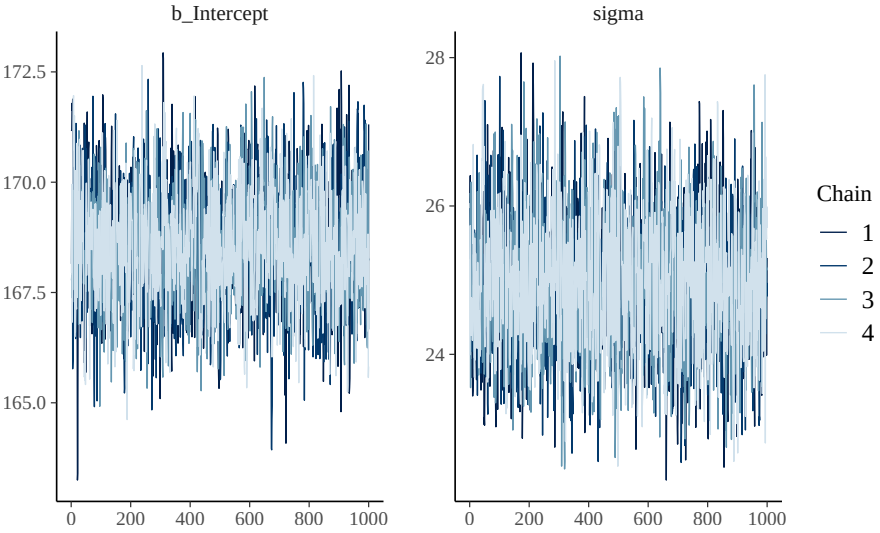


Figure 5: Trace plots in model m1.

An alternative way to plot is shown below.

Fitting the brms model on simulated data

```
m1_fakebrms<-brm(y~1,sim_data,prior = priors,
       iter = 2000, chains = 4,family = gaussian(),
       control = list(adapt_delta = 0.99))
```

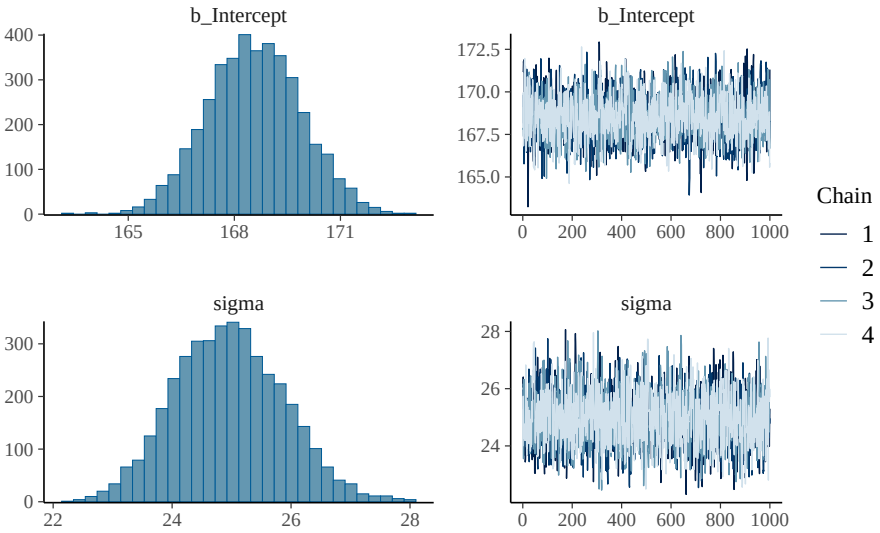


Figure 6: Posterior distributions and trace plots in model m1.

Summarizing the posterior distribution

We are assuming that there's a true underlying time it takes to press the space bar, μ, and there is normally distributed noise with a Normal(0,σ) truncated at 0 that generates the different button tapping times. All this is encoded in our likelihood by assuming that the tapping times are normally distributed with an unknown true mean μ (and an unknown standard deviation σ).

The objective of the Bayesian model is to learn about the plausible values of μ, or in other words, to get a distribution that encodes what we know about the true mean of the distribution of RTs, and about the true standard deviation, σ, of the distribution of RTs.

Our model allows us to answer questions such as:

What is the probability that the underlying value of the mindless press of the space bar would be over, say 170 ms?**

As an example, consider this model that we ran above.

```
priors <- c(set_prior("normal(0, 2000)",
                      class = "Intercept"),
            set_prior("normal(0, 500)",
                      class = "sigma"))

m1brms<-brm(t~1,df_spacebar,prior = priors,
       iter = 2000,
       warmup = 1000,
       chains = 4,
       family = gaussian(),
       control = list(adapt_delta = 0.99))
```

Now compute the posterior probability $Prob(\mu > 170)$:

```
mu_post<-posterior_samples(m1brms,
         variable=c("b_Intercept"))$b_Int
mean(mu_post>170)
```

```
## [1] 0.14875
```

The credible interval

The 95% credible interval can be extracted for μ as follows:

```
posterior_interval(m1brms,
    variable=c("b_Intercept"))
```

```
##                  2.5%    97.5%
## b_Intercept 166.0053 171.3141
```

```
posterior_summary(m1brms,
    variable=c("b_Intercept"))
```

```
##             Estimate Est.Error     Q2.5    Q97.5
## b_Intercept 168.6377  1.335129 166.0053 171.3141
```

This type of interval is also known as a **credible interval**.

A credible interval demarcates the range within which we can be certain with a certain probability that the "true value" of a parameter lies given the data and the model.

Aside: This is very different from the frequentist confidence interval, which has the following meaning: If you were (counterfactually) to run the same experiment over and over again, say 100 times, then 95% of the 100 confidence intervals you would generate would contain the true value of the parameter. The frequentist confidence interval does not tell you the range over which you can be 95% certain that the true value of the parameter lies: the parameter is not a random variable but rather is a point value, so one cannot make probability statements about it.

The percentile interval is a type of credible interval (the most common one), where we assign equal probability mass to each tail.

We generally report 95% credible intervals. But we can extract any interval; a 73% interval, for example, leaves 13.5% of the probability mass on each tail, and we can calculate it like this:

```
round(quantile(mu_post,
   prob=c(0.135,0.865)))
```

```
## 13.5% 86.5%
##   167   170
```

The influence of priors and sensitivity analysis

$$\begin{aligned} \mu &\sim Uniform(0, 5000) \\ \sigma &\sim Uniform(0, 500) \end{aligned} \tag{7}$$

```
priors <- c(set_prior("uniform(0, 5000)",
                      class = "Intercept",
                      lb = 0,
                      ub = 5000),
            set_prior("uniform(0, 500)",
                      class = "sigma",
                      lb=0,
                      ub=500))
```

```
m2<-brm(t~1,df_spacebar,prior = priors,
        iter = 2000, chains = 4,
        family = gaussian(),
        control = list(adapt_delta = 0.99))
```

For looking at the results:

```
short_summary(m2)
```

```
## ...
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   168.64      1.32   166.02   171.23 1.00     2197     1855
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    24.99      0.93    23.28    26.83 1.00     1959     1993
##
## ...
```

In general, we don't want our priors to have too much influence on our posterior.

This is unless we have *very* good reasons for having informative priors, such as a very small sample and/or a lot of prior information; an example would be if we have data from an impaired population, which makes it hard to increase our

sample size.

We usually center the priors on 0 and we let the likelihood dominate in determining the posterior.

This type of prior is sometimes called a *weakly informative prior*. Notice that a uniform prior is not a weakly informative prior, it assumes that every value is equally likely, zero is as likely as 5000.

You should always do a *sensitivity analysis* to check how influential the prior is: try different priors and verify that the posterior doesn't change drastically.

Example 2: Investigating adaptation effects

More realistically, we might have run the small experiment to find out whether the participant tended to speedup (practice effect) or slowdown (fatigue effect) while pressing the space bar.

Preprocessing the data

- We need to have data about the number of times the space bar was pressed for each observation, and add it to our list.
- It's a good idea to center the number of presses (a covariate) to have a clearer interpretation of the intercept.
- In general, centering predictors is always a good idea, for interpretability and for computational reasons.

- See the contrast coding chapters in the book for details on this point.

```
df_spacebar <- df_spacebar %>%
  mutate(c_trial = trial - mean(trial))
```

Probability model

Our model changes, because we have a new parameter.

$$t_n \sim Normal(\alpha + c\_trial_n \cdot \beta, \sigma) \quad (8)$$

where $n = 1 \dots N$.

We could use the following priors.

$$\begin{aligned} \alpha &\sim Normal(0, 2000) \\ \beta &\sim Normal(0, 500) \\ \sigma &\sim Normal(0, 500) \text{ truncated so that } \sigma > 0 \end{aligned} \quad (9)$$

We are basically fitting a linear model, α represents the intercept (namely, the grand mean of the RTs), and β represents the slope.

What information are the priors encoding?

Do the priors make sense?

We'll write this in brms as follows.

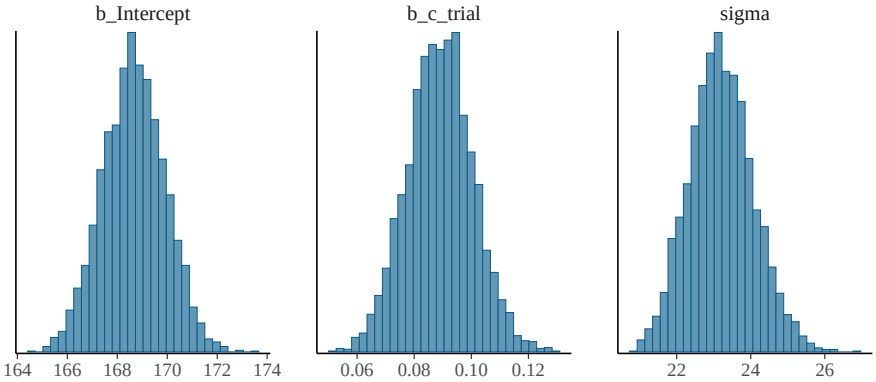
```
priors <- c(set_prior("normal(0, 2000)",
                      class = "Intercept"),
            set_prior("normal(0, 500)",
                      class = "b",
                      coef="c_trial"),
```

```
            set_prior("normal(0, 500)",
                      class = "sigma"))
```

```
m2<-brm(t~1+c_trial,df_spacebar,
        prior = priors,
       iter = 2000, chains = 4,family = gaussian(),
       control = list(adapt_delta = 0.99))
```

Posteriors

```
library(bayesplot)
#bayesplot::theme_default()
stanplot(m2,type="hist")
```



We'll need to examine what happens with β. The summary gives us the relevant information.

```
m2_post_samp_b <- posterior_samples(m2, "^b")
str(m2_post_samp_b)
```

```
## 'data.frame':    4000 obs. of  2 variables:
##  $ b_Intercept: num  169 169 170 168 171 ...
##  $ b_c_trial  : num  0.0737 0.0979 0.0995 0.0994 0.0736 ...
```

```
beta_samples <- m2_post_samp_b$b_c_trial
beta_mean<-mean(beta_samples)
quantiles_beta <- quantile(beta_samples,
                           prob=c(0.025,0.975))
beta_low<-quantiles_beta[1]
beta_high<-quantiles_beta[2]
```

Examine what happens with β. The summary gives us the relevant information:

```
m2_post_samp_b2 <- posterior_summary(m2, "^b")
round(m2_post_samp_b2,2)
```

```
##             Estimate Est.Error   Q2.5  Q97.5
## b_Intercept   168.66      1.25 166.17 171.07
## b_c_trial       0.09      0.01   0.07   0.11
```

Let's say we know that our model is working as expected, since we already used simulated data to test the recovery of the parameters.

Posterior predictive checks

To do posterior predictive checks for our last example, using **brms**, we need to do:

```
pp_check(m2,nsamples=100)+
  theme(text = element_text(size=16),
        legend.text=element_text(size=16))
```

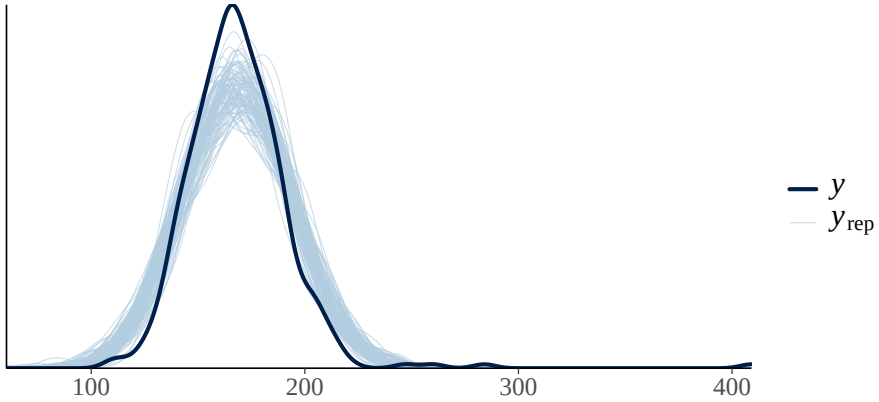


Figure 7: Posterior predictive check of model m2.

Using the log-normal likelihood

```
mu <- 6
sigma <- 0.5
N <- 100000
# Generate N random samples
## from a log-normal distribution
sl <- rlnorm(N, mu, sigma)
```

```
lognormal_plot <- ggplot(data.frame(samples=sl),
aes(sl)) + geom_histogram() +
      ggtitle("Log-normal distribution\n") +
      ylim(0,25000) + xlim(0,2000) + theme_bw()
# Generate N random samples
# from a normal distribution,
# and then exponentiate them
sn <- exp(rnorm(N, mu, sigma))
normalplot <- ggplot(data.frame(samples=sn),
aes(sn)) + geom_histogram() +
      ggtitle("Exponentiated samples of
      \n a normal distribution") +
      ylim(0,25000) + xlim(0,2000) + theme_bw()
```

```
plot(lognormal_plot)
```

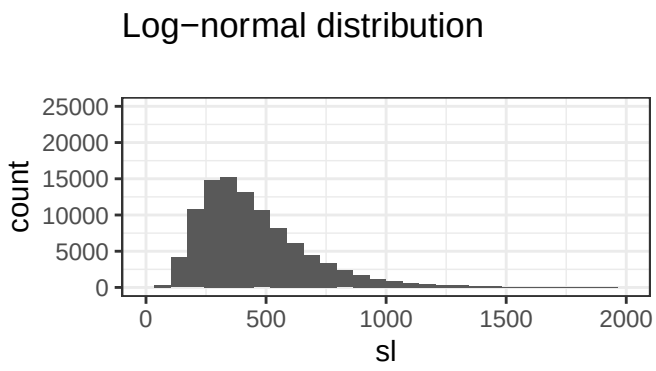


Figure 8: The log-normal distribution.

```
plot(normalplot)
```

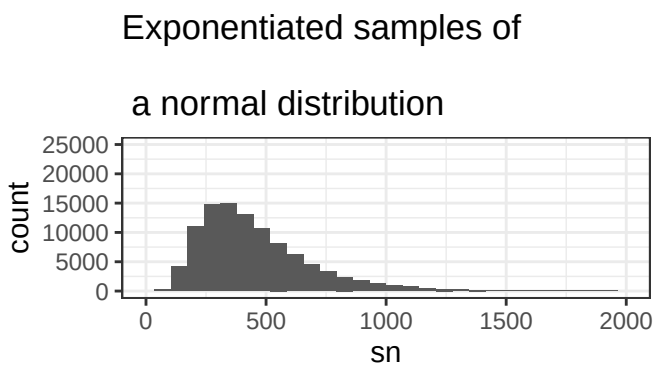


Figure 9: Exponentiated samples from a log-normal distribution.

Re-fit the model assuming a log-normal likelihood

If we assume that RTs are log-normally distributed, we'll need to change our model:

$$t_n \sim LogNormal(\alpha + c\_trial_n \cdot \beta, \sigma) \quad (10)$$

where $n = 1 \dots N$

But now the scale of our priors needs to change! They are no longer in milliseconds.

$$\begin{aligned} \alpha &\sim Normal(0, 10) \\ \beta &\sim Normal(0, 1) \\ \sigma &\sim Normal(0, 2) \text{ truncated so that } \sigma > 0 \end{aligned} \quad (11)$$

```
priors_log <- c(set_prior("normal(0, 10)",
                          class = "Intercept"),
                set_prior("normal(0, 1)",
                          class = "b",
                          coef="c_trial"),
                set_prior("normal(0, 2)",
                          class = "sigma"))
```

```
m2_logn<-brm(t~1+ c_trial,df_spacebar,
             prior = priors_log,
       iter = 2000, chains = 4,family = lognormal(),
       control = list(adapt_delta = 0.99,
                      max_treedepth=15))
```

```
## Compiling Stan program...
## Start sampling
```

Summarizing the posterior and inference

Next, we turn to the question of what we can report as our results, and what we can conclude from the data.

- We can summarize the posterior and do inference as discussed in Example 1.
- If we want to talk about the effect estimated by the model, we summarize the posterior of β in the following way:
- $\hat{\beta} = 0.0873272$, 95% CrI $= [0.06676, 0.1077809]$, $P(\beta > 0) \approx 1$

Posterior predictive checks and distribution of summary statistics

We can now verify whether our predicted datasets look more similar to the real dataset. See rgw figure below.

```
pp_check(m2_logn, nsamples = 100)+
  theme(text = element_text(size=16),
  legend.text=element_text(size=16))
```

```
m2_logn<-brm(t~1+c_trial,df_spacebar,
             prior = priors_log,
       iter = 2000, chains = 4,
       family = lognormal(),
       control = list(adapt_delta = 0.99,
                      max_treedepth=15))
```

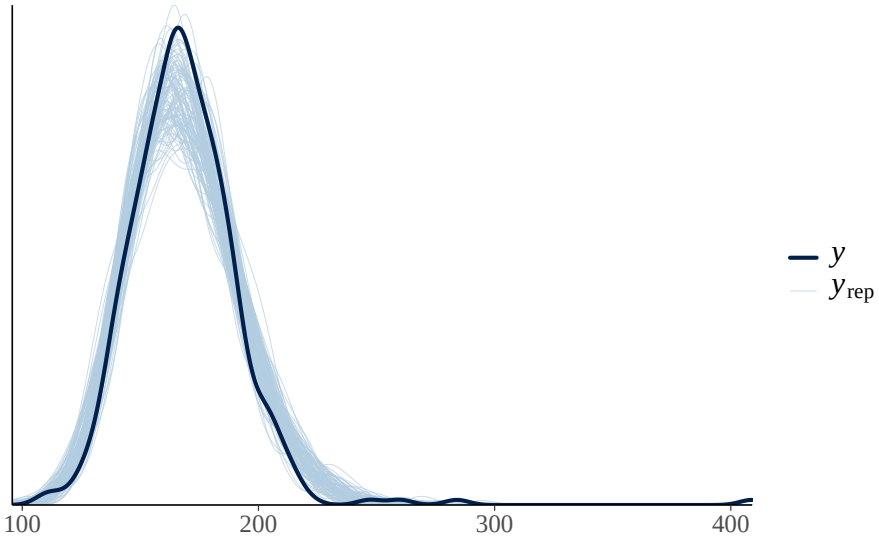


Figure 10: Posterior predictive check.

General workflow

This is the general workflow that we suggest for a Bayesian model.

1. Define the full probability model:
   a. Decide on the likelihood.
   b. Decide on the priors.
   c. Write the `brms` or Stan model.
2. Do prior predictive checks to determine if priors make sense.

3. Check model using simulated data:
   a. Simulate data with known values for the parameters.
   b. Fit the model and do MCMC diagnostics.
   c. Verify that it recovers the parameters from simulated data.
4. Fit the model with real data and do MCMC diagnostics.
5. Evaluate the model's fit (e.g., posterior predictive checks, distribution of summary

statistics). This may send you back to 1.

6. Inference/prediction/decisions.
7. Conduct model comparison if there's an alternative model (to be discussed later).