# Bayesian Statistics – Slides based on Nicenboim, Schad and Vasishth (2021)

Vera Demberg

Saarland University

1.2.2022

*In our reasonings concerning matter of fact, there are all imaginable degrees of assurance, from the highest certainty to the lowest species of moral evidence. A wise man, therefore, proportions his belief to the evidence.*

– David Hume

# Where are we?

1. Bayes Theorem
2. Let's fit a model!
   - Hamiltonian Monte Carlo Sampling
   - Back to fitting the model
3. Choosing the prior
4. Old example, with different prior
5. Posterior predictive distribution
6. Choosing a different likelihood function

# Table of Contents

# How does this connect to the Bayes rule?

## Bayesian Statistics

Notation: p stands for probability distribution; y stands for data, $\theta$ is the parameter to be estimated.

$$p(\Theta|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\Theta) \cdot p(\Theta)}{p(\boldsymbol{y})} \tag{1}$$

Last time, we said $p(y)$ is just a normalizing constant. It indeed doesn't contribute much to the theory.

In practice, however, we DO need to calculate also the normalizing constant in order to get a probability distribution for the posterior. For continuous distributions, we need to integrate the numerator:

$$p(\Theta|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\Theta) \cdot p(\Theta)}{\int_{\Theta} p(\boldsymbol{y}|\Theta) \cdot p(\Theta) d\Theta} \tag{2}$$

# What held Bayesian methods back?

Calculating this integral is very difficult and was the major bottleneck of Bayesian analysis in the past.

$$\int_{\Theta} p(\boldsymbol{y}|\Theta) \cdot p(\Theta) d\Theta$$

Instead, we need to sample from the distribution in order to approximate it. Probabilistic programming languages make it a lot easier to do this nowadays.

# Enabling step

The development of probabilistic programming languages (such as WinBUGS, JAGS, pymc3, Turing, Stan) has made it possible to carry out the sampling process easily.

in R: user-friendly interfaces to Stan:

- rstanarm (Goodrich et al., 2018)
- brms (Bürkner 2019)

We will use the brms package (its syntax is most similar to frequentist models).

# Table of Contents

# How to fit a simple linear model

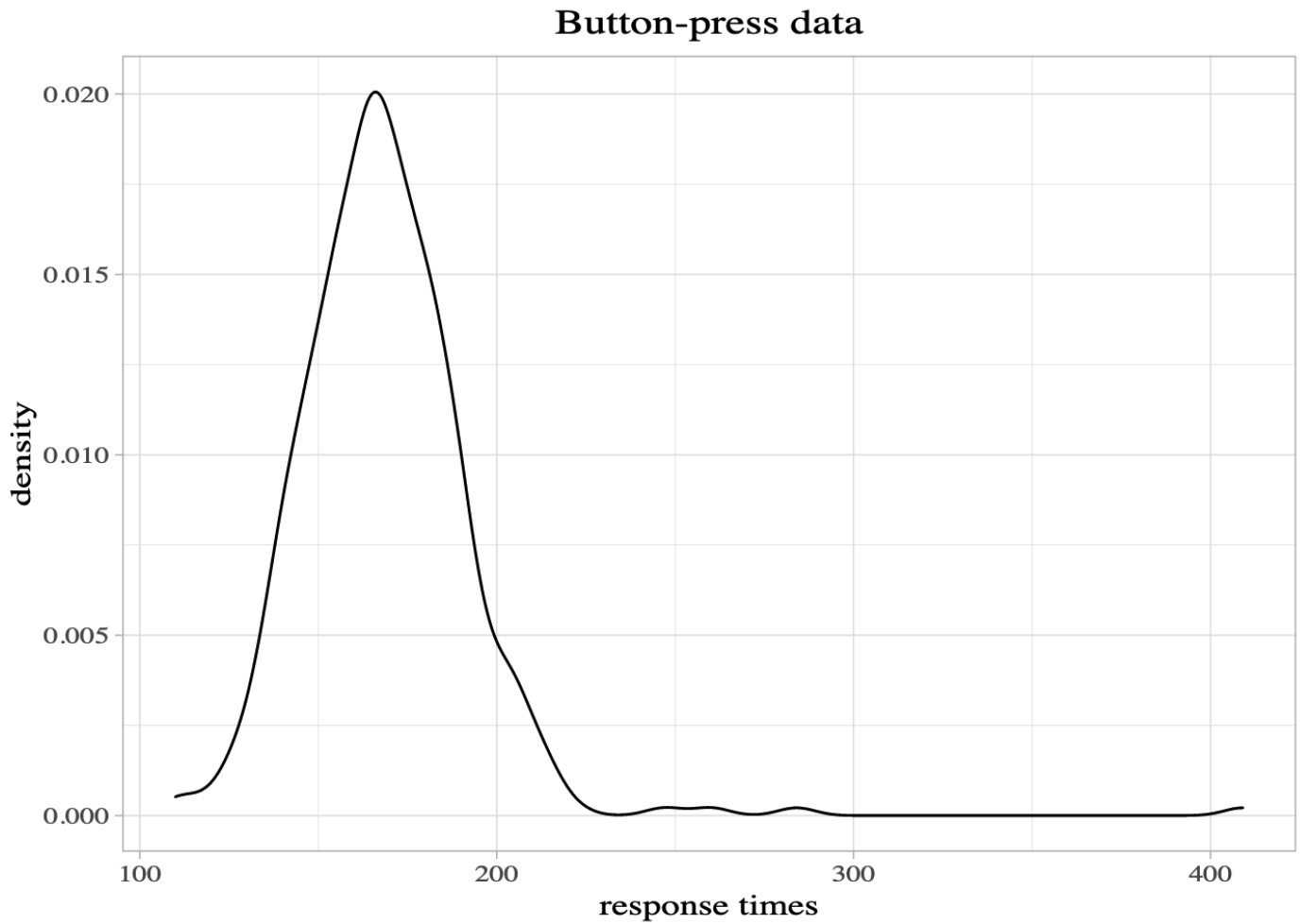
## Example

We want to estimate how long it takes to press a button from the data of a human repeatedly pressing the space bar as fast as he/she can.

$$rt_n \sim Normal(\mu, \sigma) \tag{3.2}$$

- $\mu$ is the underlying true time the person needs to press the space bar.
- $\sigma$ is the noise in the process
- the noise is normally distributed
- $rt_n$ are reaction time measurements n=1..N

# Looking at the data




Actually, it's a bit skewed.

# How to estimate this model?

## Frequentist regression

lm(rt $\sim$ 1,data=df_spacebar)

## Bayesian Regression model

Need to define priors for each of the parameters ($\mu$ and $\sigma$) in the model:

$$\begin{aligned} \mu &\sim \mathit{Uniform}(0, 60000) \\ \sigma &\sim \mathit{Uniform}(0, 2000) \end{aligned} \tag{3.3}$$

This means:

- we believe that the mean reaction time is between 0 and 60 seconds, and we have no idea which of the values in this range is most likely.
- the variance is between 0 and 2 seconds.

# How to estimate this model?

## Frequentist regression

fit_press_freq←lm(rt ∼ 1,data=df_spacebar)

## Bayesian Regression model

```
fit_press <- brm(rt ~ 1,
  data = df_spacebar,
  family = gaussian(),
  prior = c(
    prior(uniform(0, 60000), class = Intercept),
    prior(uniform(0, 2000), class = sigma)
  ),
  chains = 4,
  iter = 2000,
  warmup = 1000
)
```

# Chains? iter? warmup?

these specifications set the parameters for the sampling algorithm.

```
  ),
  chains = 4,
  iter = 2000,
  warmup = 1000
)
```

- chains: number of independent runs for sampling (4 is the default).
- iter: number of iterations that the sampler makes to sample from the posterior distribution of each parameter.
- warmup: number of iterations that will be discarded (by default, half of iter)

Sampling algorithm:
the No-U-Turn Sampler (NUTS; Hoffman and Gelman 2014) extension of Hamiltonian Monte Carlo (Duane et al. 1987; Neal 2011).

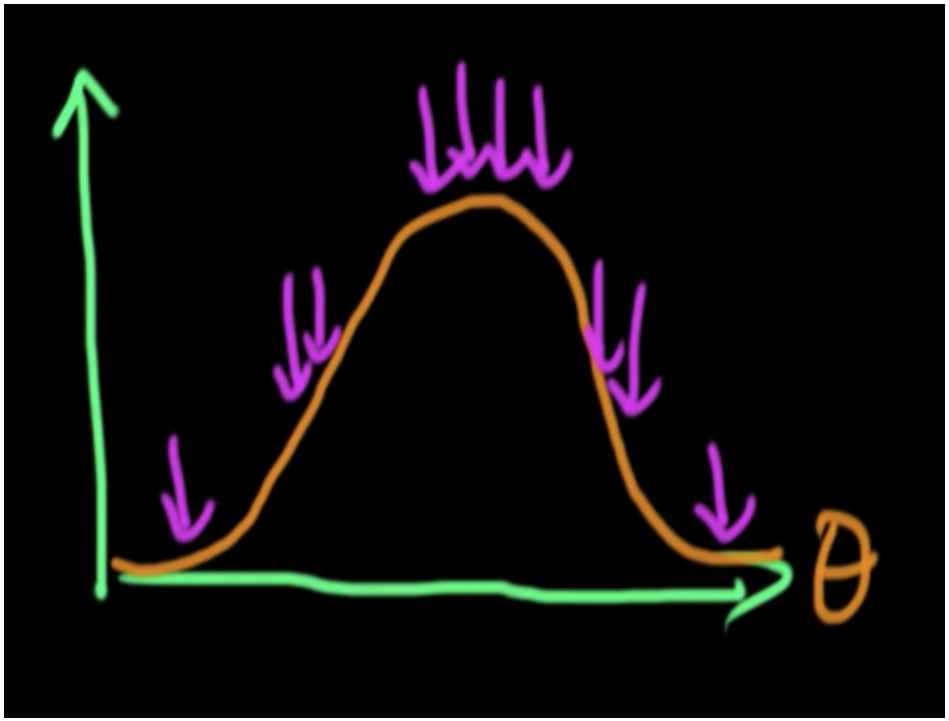
# Sampling and convergence

- chains are initialized randomly and independently from each other
- at each iteration, each chain takes one sample
- R uses the No-U-Turn Sampler (Nuts),
  which is an extension of Hamiltonian Monte Carlo,
  which in turn is a variant of Metropolis-Hastings,
  which is a type of Markov-Chain Monte Carlo (MCMC) sampling.

We'll next look at how Hamiltonian Monte Carlo works.

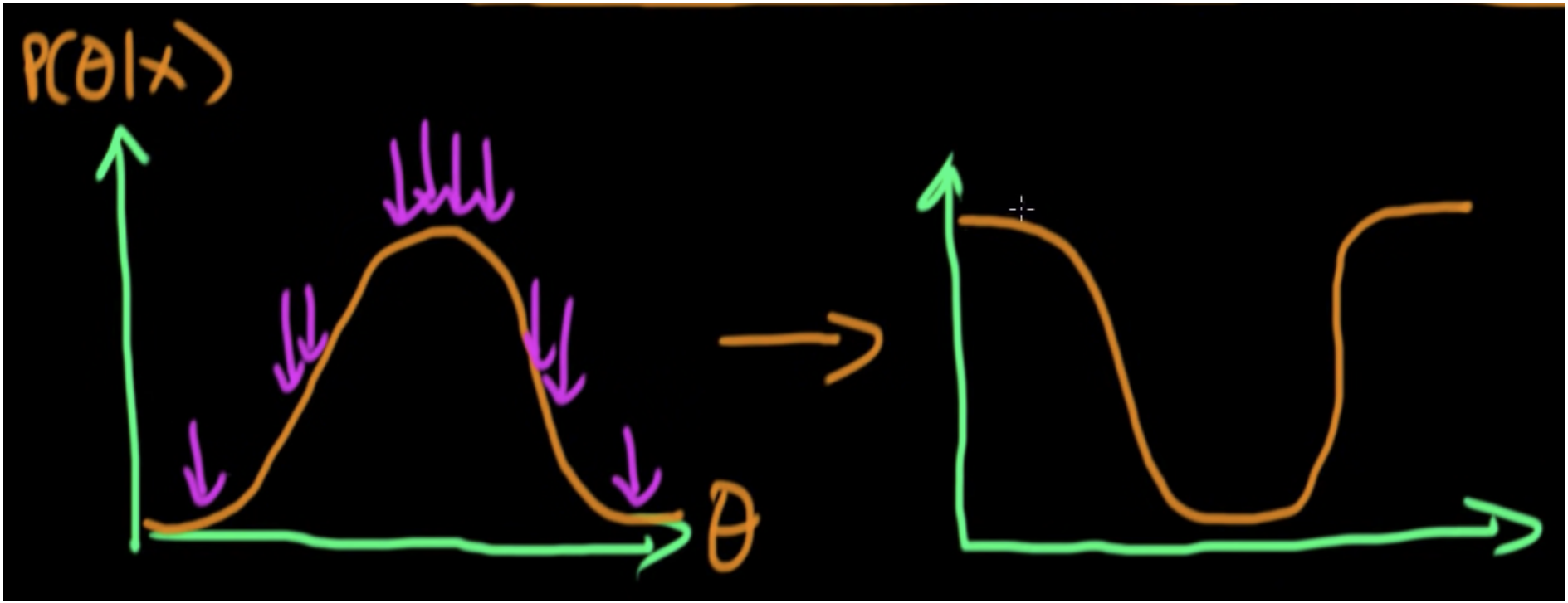
# We want to sample a distribution in proportion to its density

If there is a lot of probability mass in one area, we want to sample more densely from that area than from an area where there is very little probability mass.



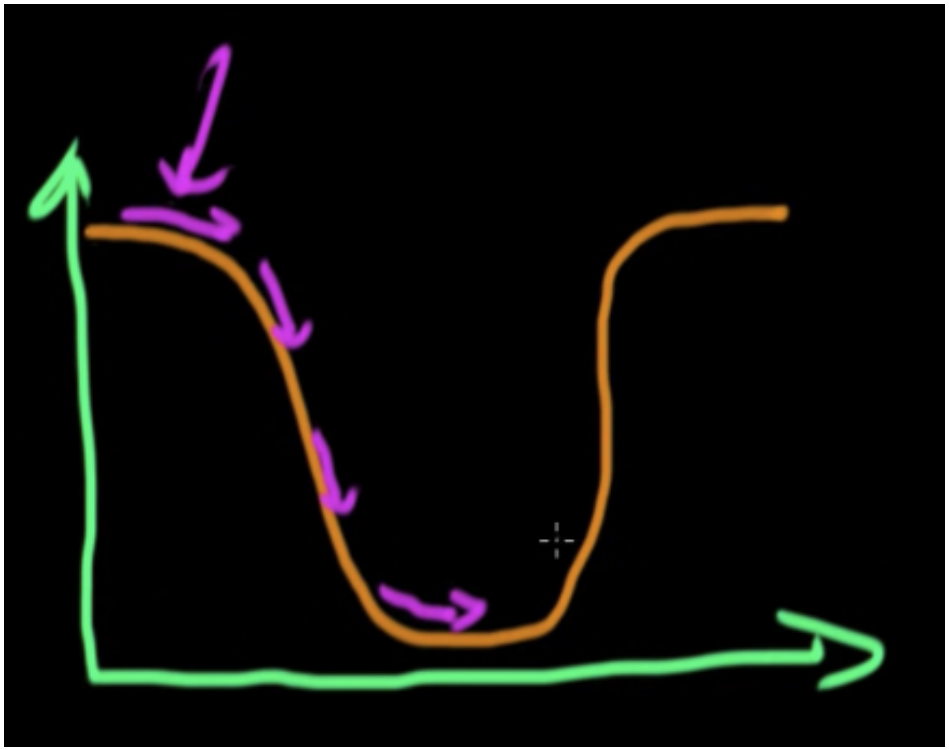
# We want to sample a distribution in proportion to its density

In order to do this, we first flip the density (and calculate its log).



# We want to sample a distribution in proportion to its density

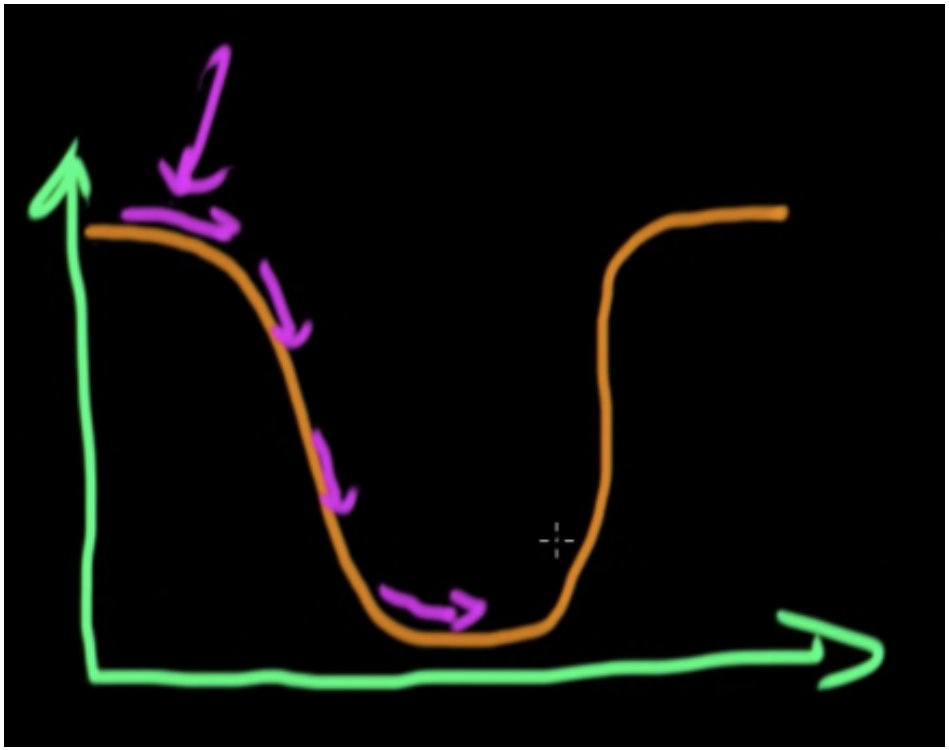
Physical analogy: we now imagine there's a sled that can run on this landscape. It would of course tend to move down.



So this will give us more samples from the valley (which corresponds to the peak of the original distribution).

# We want to sample a distribution in proportion to its density

We give the sledge a random kick every now and then so it will move up to one of the sides (but these will be visited less frequently).



# How is this useful to us?

We imagine that our sledge on the hill system has an energy level.
It is described by the position and the momentum of the sledge. There is no friction (i.e. we never lose energy).

$$E(\theta, k) = U(\theta) + KE(k) = constant$$

- $E$ stands for energy (aka " Hamiltonian")
- $\theta$ (position on the surface, a vector)
- $k$ (momentum of the sledge, also a vector)
- $U$ (gravitational potential energy, depends on position)
- $KE$ kinetic energy of the particle (depends on momentum)

# Sledge with Hamiltonian dynamics

Imagine the sledge moving over a frictionless surface of varying heights.

- The sledge moves at constant velocity (momentum) k on flat surface
- When the sledge moves up an incline, it's kinetic energy goes down, and its potential energy goes up.
- When the sledge slows down and comes to a halt, kinetic energy becomes 0.
- When the sledge slides back, kinetic energy goes up, potential energy goes down.

# Background: Hamiltonian dynamics

## Potential energy (position)

Define the potential energy of the sledge as

$$U(\theta) = -\log(p(X|\theta)p(\theta))$$

Thus:

- $U(\theta)$ is defined to be the negative log posterior density
- it is defined to be the inverse of the posterior space.

## Kinetic energy (speed)

Kinetic energy is $\frac{1}{2}mv^2$, with m=mass, v= velocity
Assuming q dimensions and m=1

$$KE(k) = \sum_{i=1}^{q} \frac{k_i^2}{2}$$

# How does the sledge move?

The equations of motion:
Let there be i=1,...,d parameters.
Given the equation:

$$H(\theta, k) = U(\theta) + K(m)$$

Classical mechanics defines these equatios of motion:

- position:

$$\frac{d\theta_i}{dt} = \frac{\delta H}{\delta k_i}$$

- momentum:

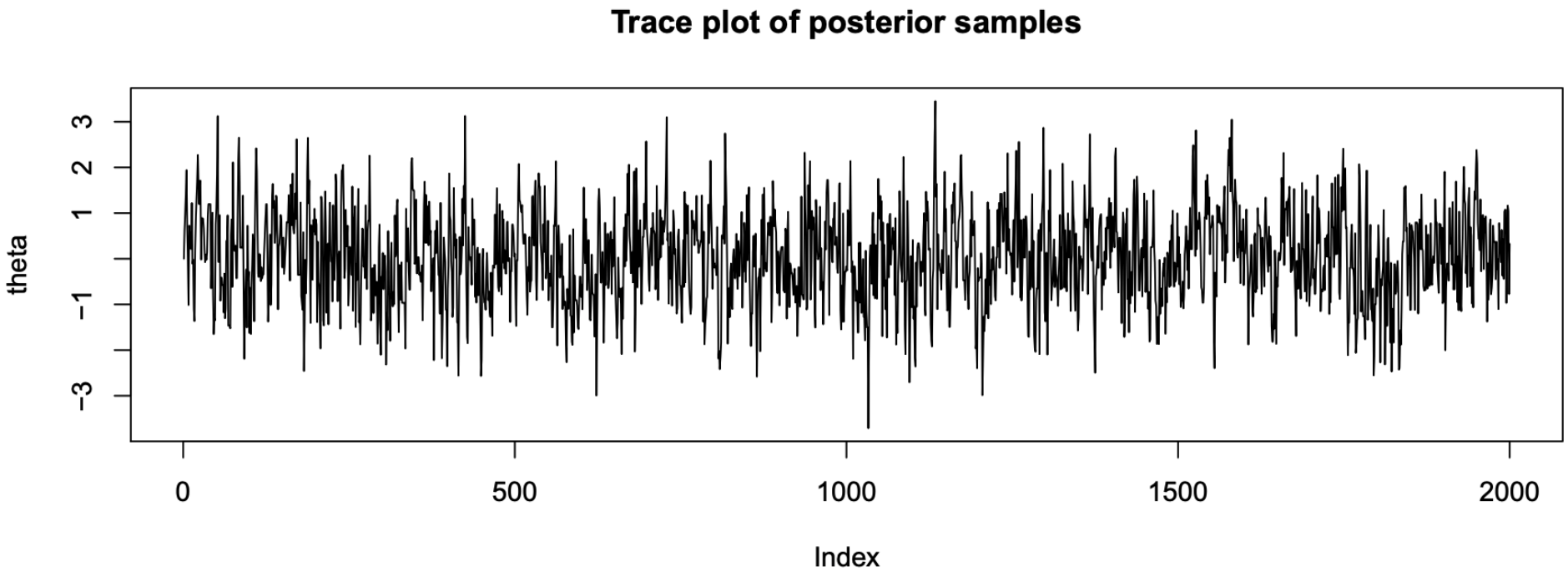
$$\frac{dk_i}{dt} = \frac{\delta H}{\delta \theta_i}$$

These equations define the mapping from state of the sledge at time $t$ to time $t + s$.

# Simplified algorithm

- Choose initial momentum $k \sim N(0, \Sigma)$
- Record sledge position (value of $\theta$)
- Record sledge momentum (value of $k$)
- The sledge's position and momentum lead to an accept / reject rule that yields samples from the posterior with a high probability of acceptance.

# How is this useful to us?

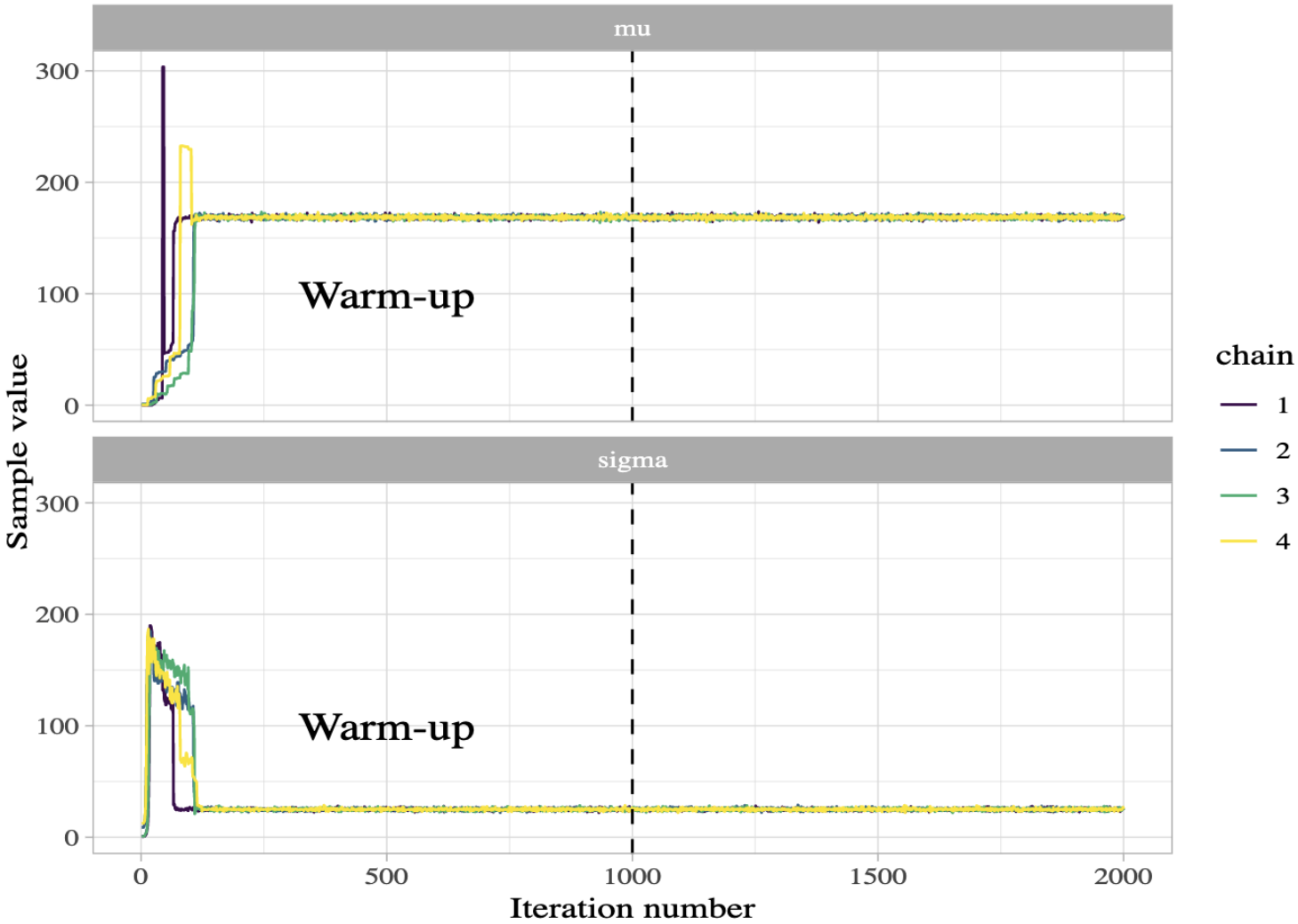


# Back to sampling and convergence

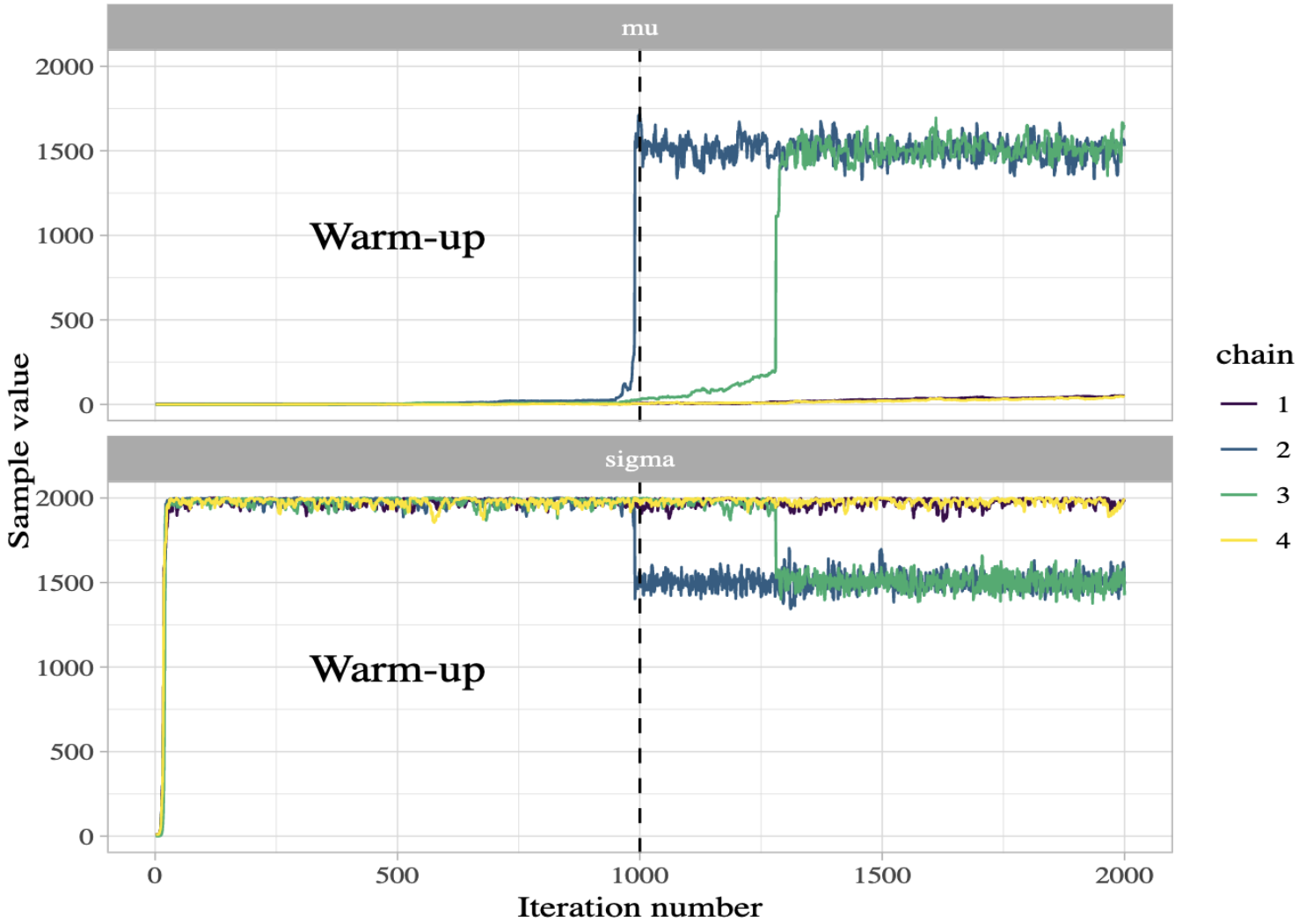
- chains are initialized randomly and independently from each other
- at each iteration, each chain takes one sample
- R uses the No-U-Turn Sampler (Nuts),
  which is an extension of Hamiltonian Monte Carlo,
  which in turn is a variant of Metropolis-Hastings,
  which is a type of Markov-Chain Monte Carlo (MCMC) sampling.

Let's now get back to our example.

# The chains should converge during warm-up



# Convergence failure



# Model output

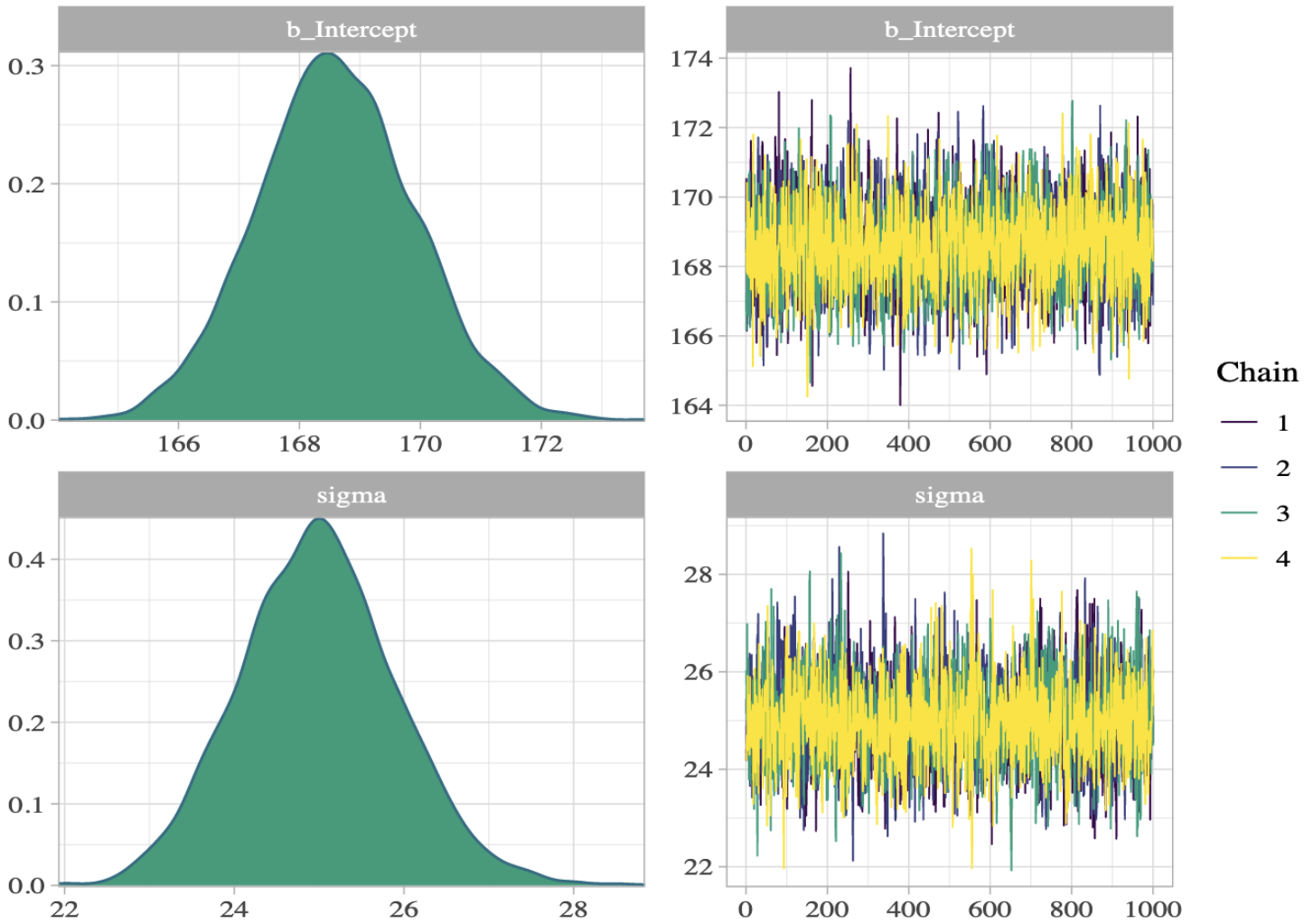
We can actually ask R to output the samples from the posterior distribution after fitting the model:

```
as_draws_df(fit_press)
```

```
## # A draws_df: 1000 iterations, 4 chains, and 3 variables
##    b_Intercept sigma  lp__
## 1          169    25 -1688
## 2          168    26 -1688
## 3          168    24 -1689
## 4          169    25 -1688
## 5          168    25 -1688
## 6          169    25 -1688
## 7          168    26 -1689
## 8          168    23 -1689
## 9          166    25 -1690
## 10         168    25 -1688
## # ... with 3990 more draws
## # ... hidden reserved variables {'.chain', '.iteration', '.draw'}
```

# Visualisation of samples

```
plot(fit_press)
```



# Model summary

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: rt ~ 1
##    Data: df_spacebar (Number of observations: 361)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   168.63      1.27   166.13   171.14 1.00     3489     2795
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    25.00      0.94    23.20    27.02 1.00     3843     2465
##
## 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).
```

# We fit a first model!

- But did our choice for the prior actually make sense? That seemed pretty random.
- Have we chosen the right likelihood function for this data (normal distribution)?

# Table of Contents

# Choosing the prior

$$p(\Theta|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\Theta) \cdot p(\Theta)}{p(\boldsymbol{y})} \tag{3}$$

The prior here is denoted as $p(\Theta)$, and we had set $\Theta =< \mu, \sigma >$ with $\mu \sim Uniform(0, 60000)$ and $\sigma \sim Uniform(0, 2000)$.
What would data look like that was generated based on this distribution?
Prior predictive density:

$$\begin{aligned} p(\boldsymbol{y_{pred}}) &= p(y_{pred_1}, \ldots, y_{pred_n}) \\ &= \int_{\Theta} p(y_{pred_1}|\Theta) \cdot p(y_{pred_2}|\Theta) \cdots p(y_{pred_N}|\Theta) p(\Theta)\, d\Theta \end{aligned} \tag{4}$$

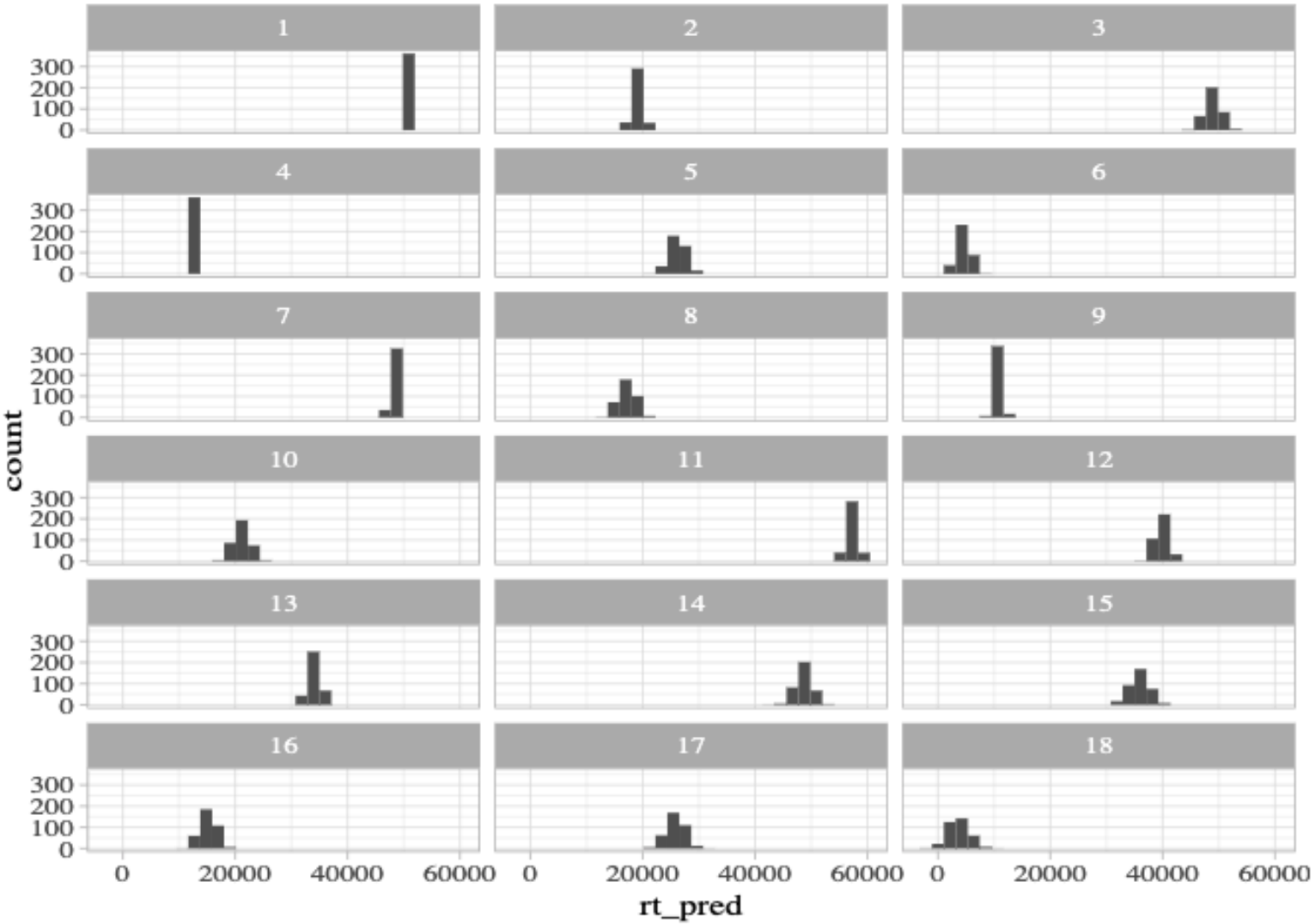
Now we can calculate the integral or sample to figure out the shape of the prior predictive distribution.

# Sampling from the prior predictive distribution

Repeat many times:

- take a sample $\mu$ from the distribution Uniform(0,60000)
- take a sample $\sigma$ from distribution Uniform(0,2000)
- randomly sample n observations from a normal distribution (since we assumed that the likelihood function is a normal distribution) with the sampled parameters $\mu$ and $\sigma$: rnorm(n,mu,sigma)
- save those samples together with the samples of the other iterations.

# Let's look at some samples



# All samples together yield the prior predictive distribution

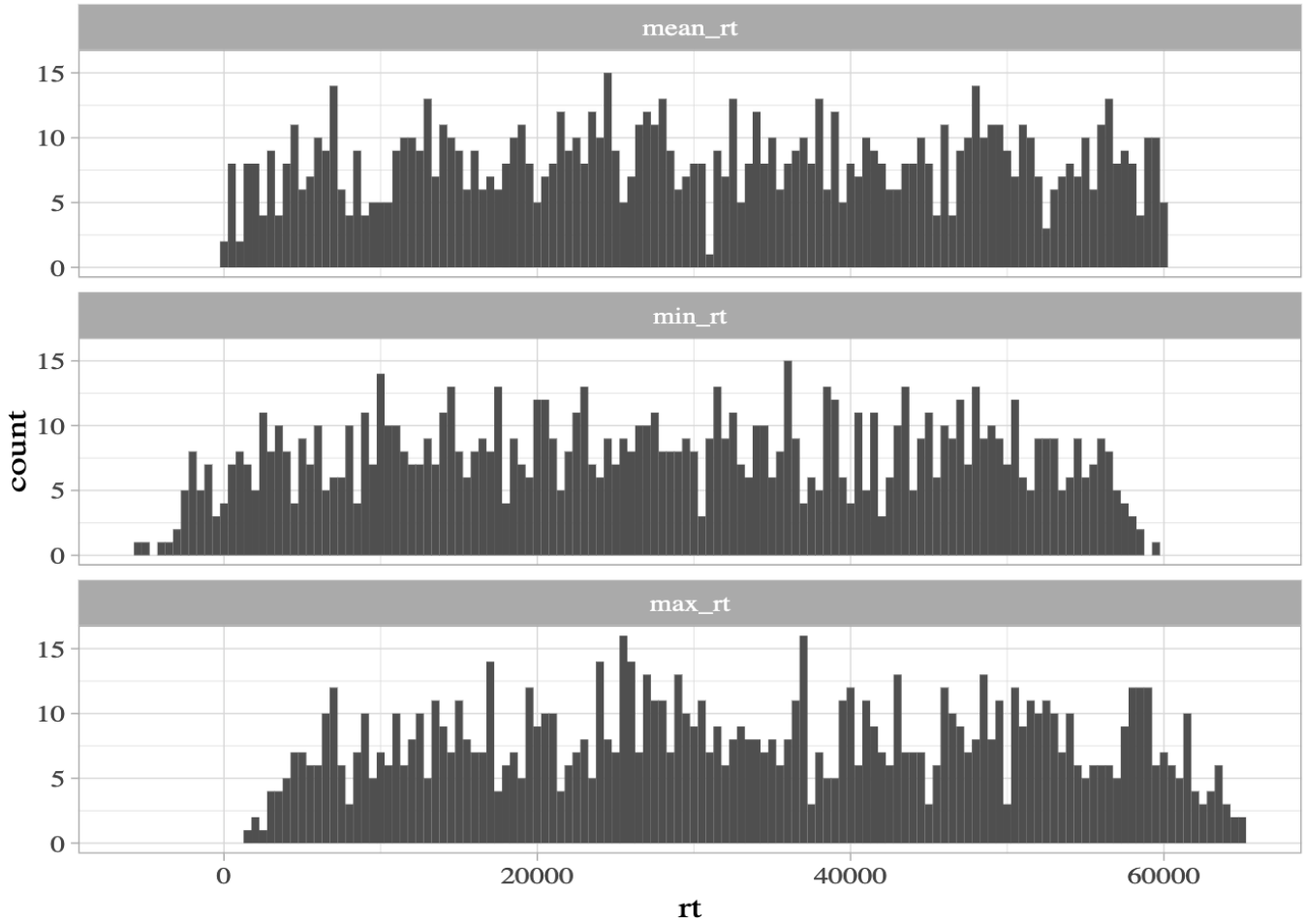



FIGURE 3.6: Prior predictive distribution of mean, minimum, and maximum value of the button-pressing model defined in section 3.1.1.1.

# Types of priors

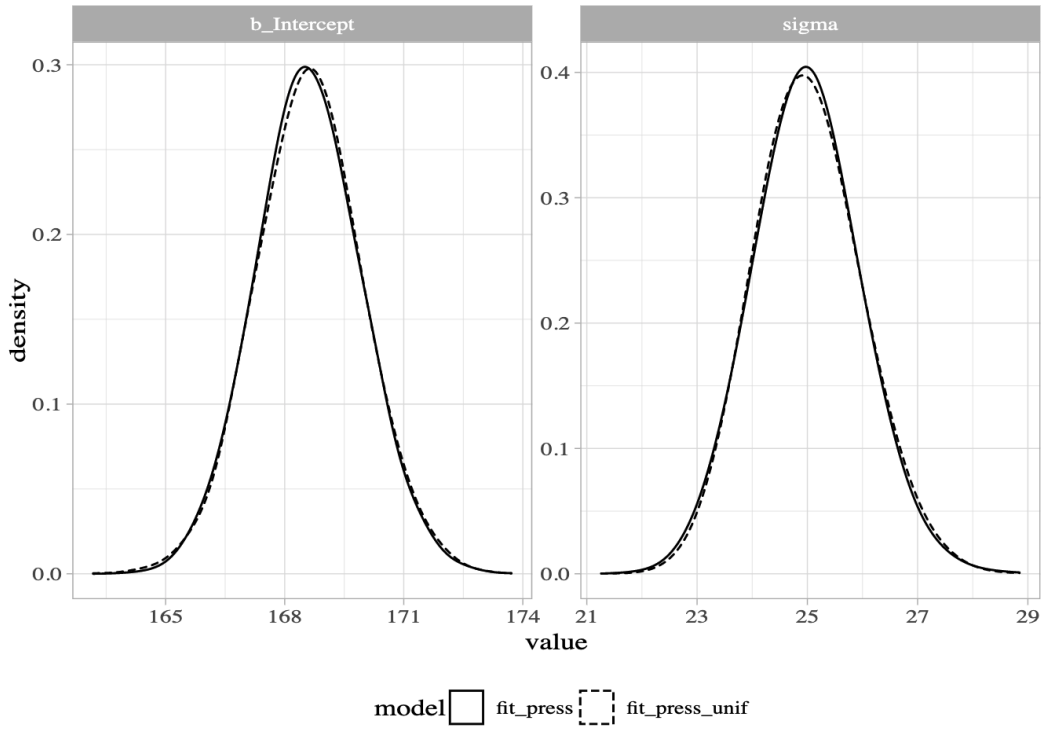
- **flat, uninformative priors** (e.g.: uniform distribution)
  Expresses that we don't know anything about likely parameter values.
  Disadvantages:
  - slower sampling
  - possible convergence problems
- **regularizing priors**
  downweight extreme values; stabilize computation, but still theory-neutral.
- **principled priors**
  encode all theory-neutral information; reflect the real properties of the data.
- **informative priors**
  useful when we have a lot of prior knowledge and little data.
  Usually want to avoid this because the prior should not influence the posterior too much.

# Table of Contents

# Let's try a different prior for the button-press experiment!

What would happen if we would use extremely unrealistic priors?

$$\mu \sim Uniform(-10^{10}, 10^{10})$$
$$\sigma \sim Uniform(0, 10^{10}) \tag{5}$$



## Principled prior

Based on previous similar experiments, we expect that the mean reaction time would be around 200 ms, but we have a large degree of uncertainty regarding that value, so we decide on a relatively wide prior:

$$\mu \sim Normal(200, 100)$$

Given that we only have one subject and the task is very simple, we don't expect a large standard deviation:

$$\sigma \sim Normal_+(50, 50)$$

## Informative prior

Here: expresses a specific (but wrong) idea about what the parameters.

$$\begin{aligned} \mu &\sim Normal(400, 10) \\ \sigma &\sim Normal_+(100, 10) \end{aligned} \tag{6}$$

# Comparison of posterior estimates

| prior: | $\mu$ | $\sigma$ |
|---|---|---|
| flat (unrealistic) | 168.66 | 25.01 |
| flat (regularizing) | 168.63 | 25.00 |
| principled | 168.67 | 25.04 |
| informative (but wrong) | 172 | 26.08 |

The estimates for the posterior are quite stable in this case. The informative (but incorrect) prior had the largest effect on the posterior estimates.

# Table of Contents

# Posterior predictive distribution

Basically, we obtain this by generating samples based on our posterior $p(\Theta|y)$:

$$p(\boldsymbol{y_{pred}} \mid \boldsymbol{y}) = \int_{\Theta} p(\boldsymbol{y_{pred}}, \Theta \mid \boldsymbol{y})\, d\Theta = \int_{\Theta} p(\boldsymbol{y_{pred}} \mid \Theta, \boldsymbol{y}) p(\Theta \mid \boldsymbol{y})\, d\Theta \tag{7}$$

We assume that past and future observations ar conditionally independent given $\Theta$:

$$p(\boldsymbol{y_{pred}} \mid \boldsymbol{y}) = \int_{\Theta} p(\boldsymbol{y_{pred}} \mid \Theta) p(\Theta \mid \boldsymbol{y})\, d\Theta \tag{8}$$

We can sample from this just like from the prior predictive distribution. But $\mu$ and $\sigma$ are now sampled from the posterior.

# in R

Convenient function: `posterior_predict(fit_press)`[1]

Why should we do this? Sanity check:

- Check whether the model predictions look similar to the data.

Convenient visualisation command:

```
pp_check(fit_press, ndraws = 11, type = "hist")
```

[1] fitpress is the name of the variable that stores the model.

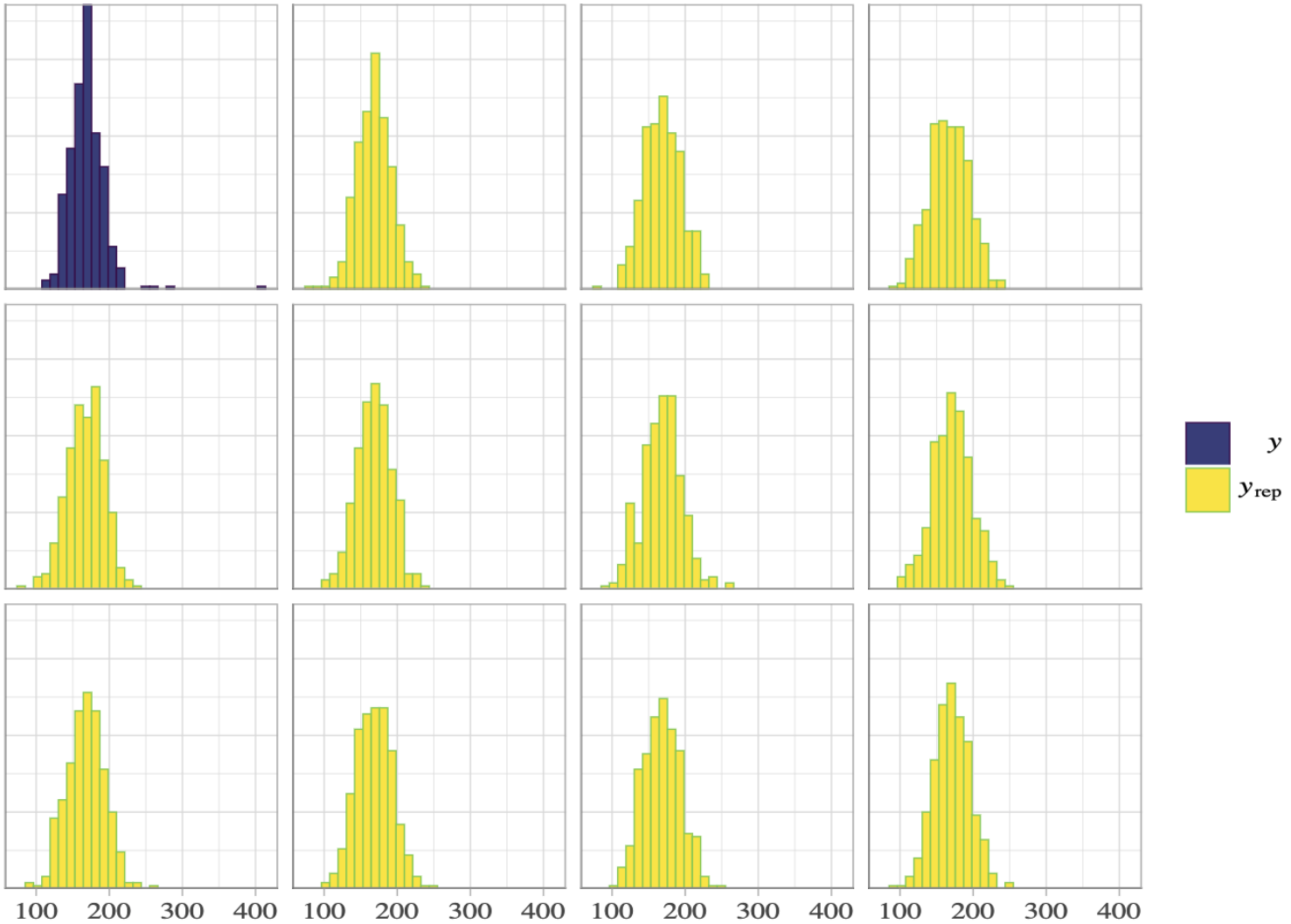



FIGURE 3.8: Histograms of eleven samples from the posterior predictive distribution of the model `fit_press` ($y_{rep}$).

```
pp_check(fit_press, ndraws = 100, type = "dens_overlay")
```

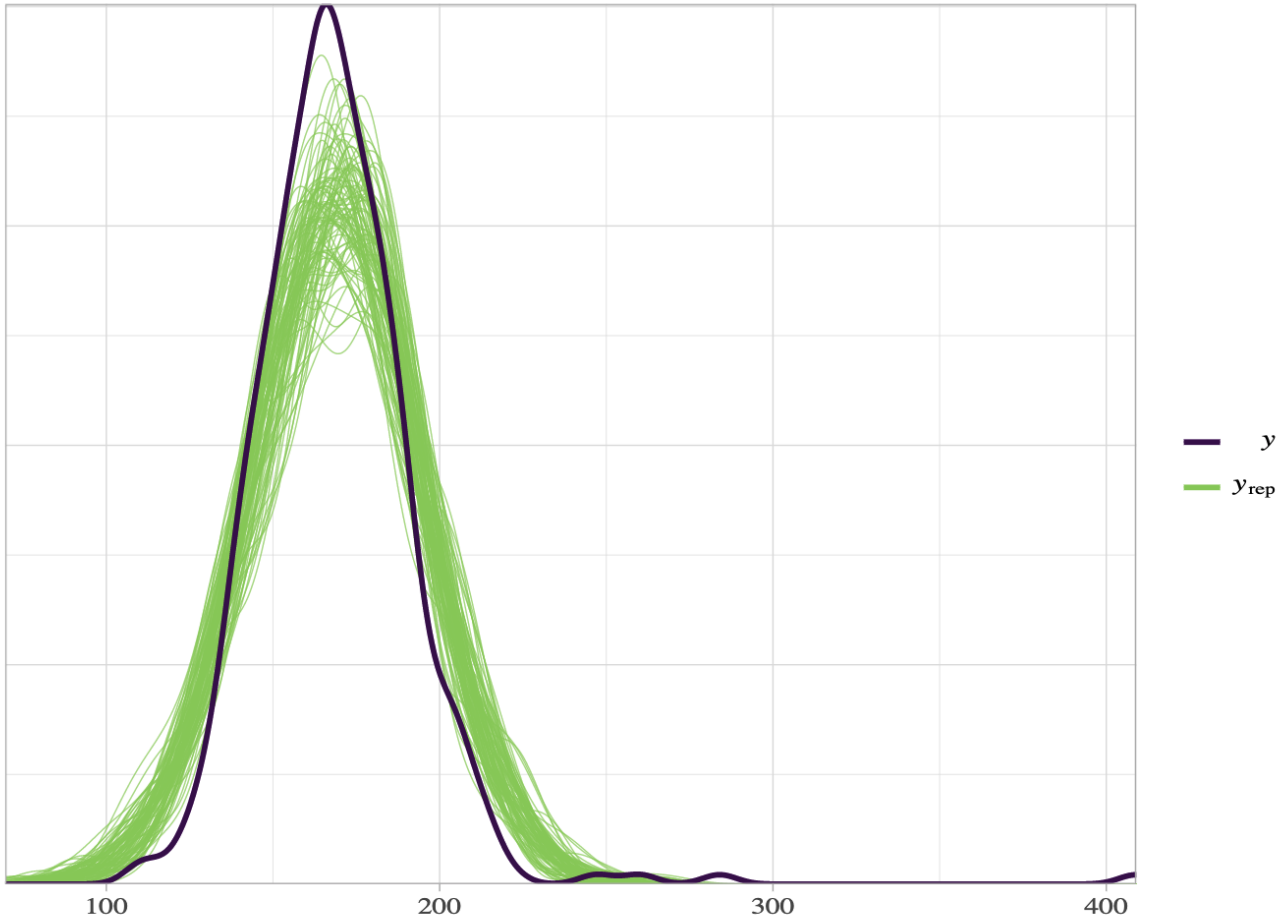



FIGURE 3.9: Posterior predictive check that shows the fit of the model `fit_press` in comparison to data sets from the posterior predictive distribution using an overlay of density plots.

Fit looks ok but could it be better?

# Table of Contents

# Choosing a different likelihood function

We made several decisions:

- How to set $\mu$
- How to set $\sigma$
- What likelihood function to choose (we assumed a normal distribution).

What if we choose a different likelihood function?

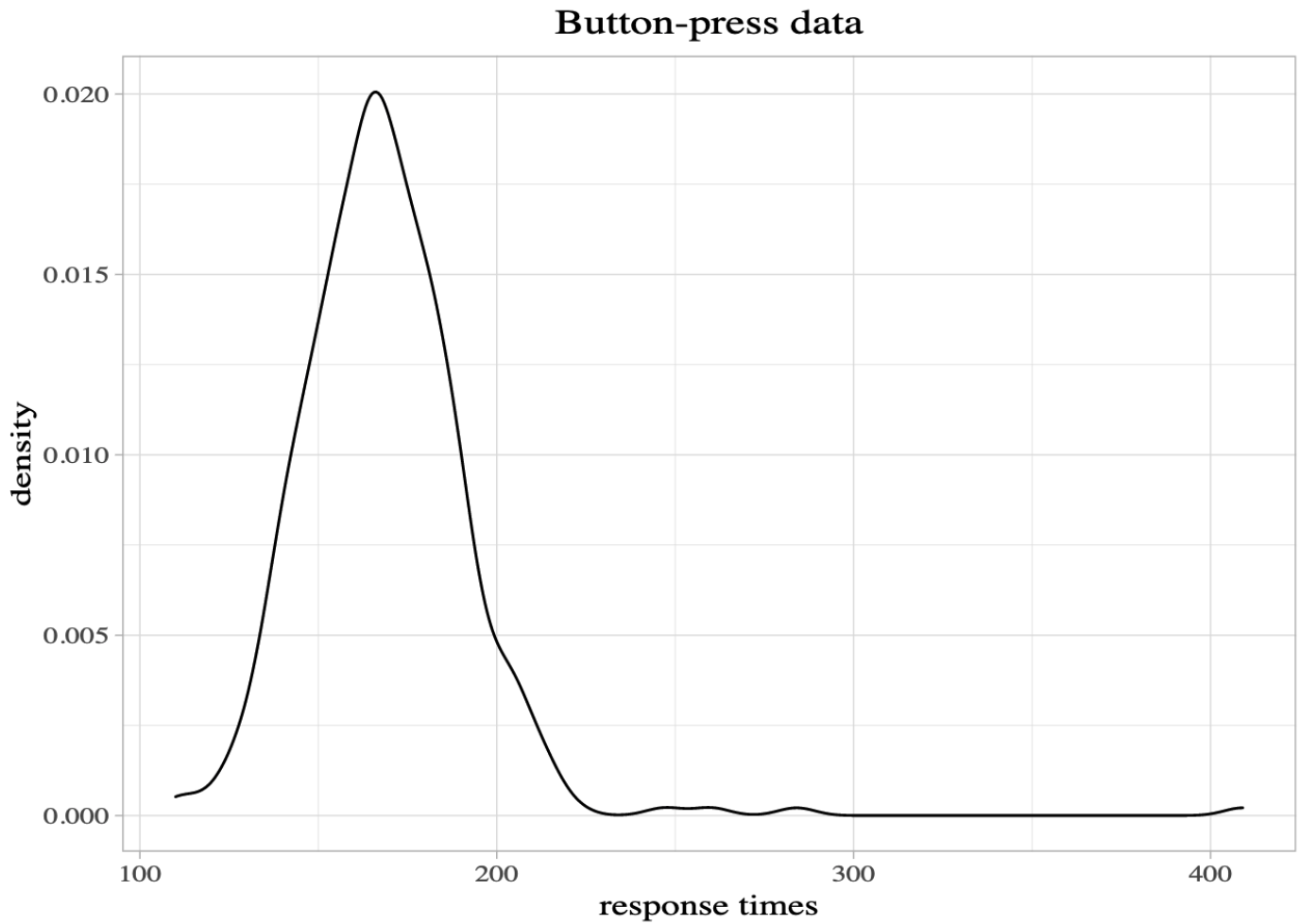
# Choosing a different likelihood function

We made several decisions:

- How to set $\mu$
- How to set $\sigma$
- What likelihood function to choose (we assumed a normal distribution).

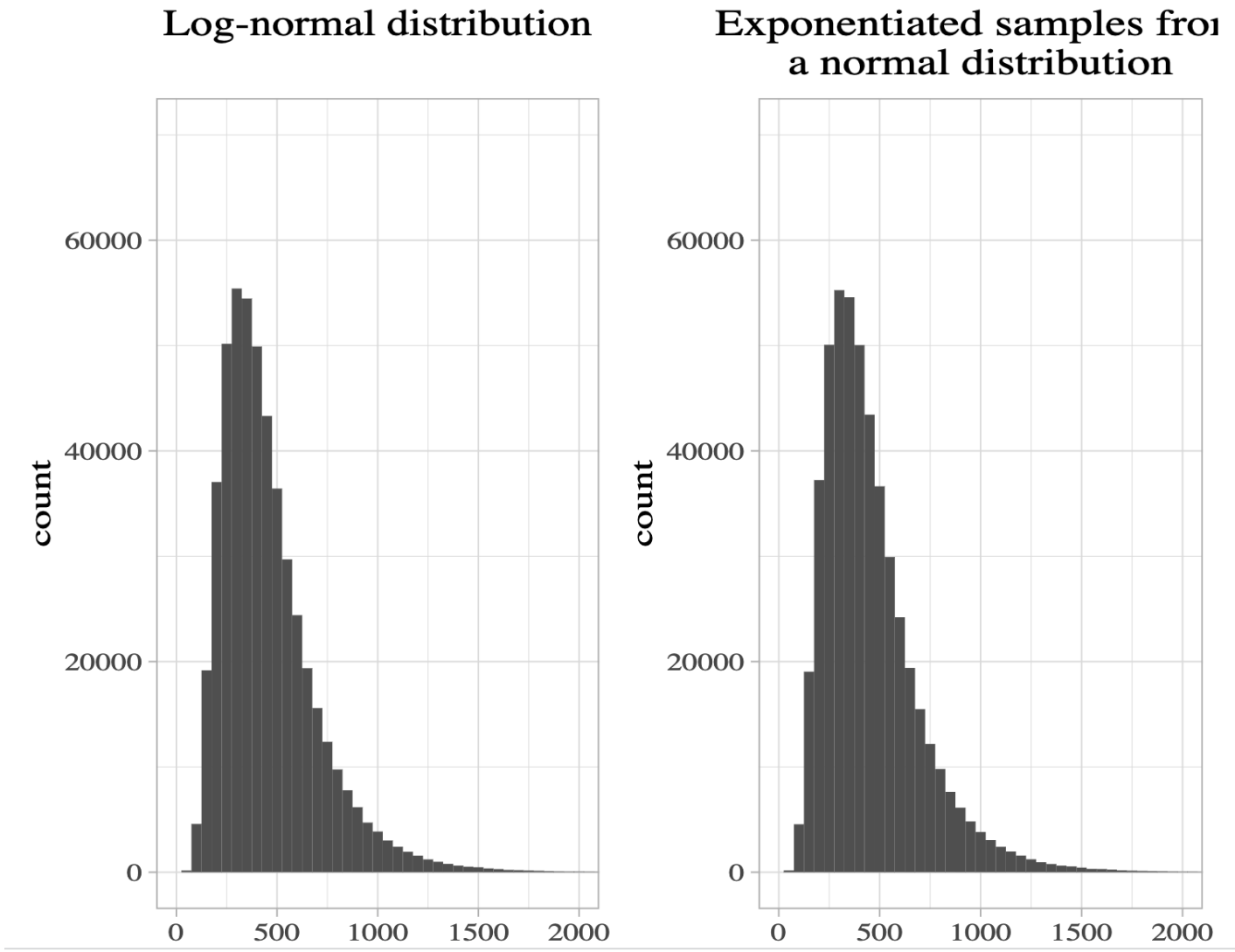
What if we choose a different likelihood function?
Let's take another look at our data.

# Looking at the data




Actually, it's a bit skewed.

# The lognormal distribution (log(y) is normally distributed)




Maybe this will fit better.

# Priors in lognormal distribution

Choosing the lognormal distribution has implications for how to set our priors.
Model specification:

$$rt_n \sim LogNormal(\mu, \sigma)$$
$$\mu \sim Uniform(0, 11)$$
$$\sigma \sim Uniform(0, 1)$$

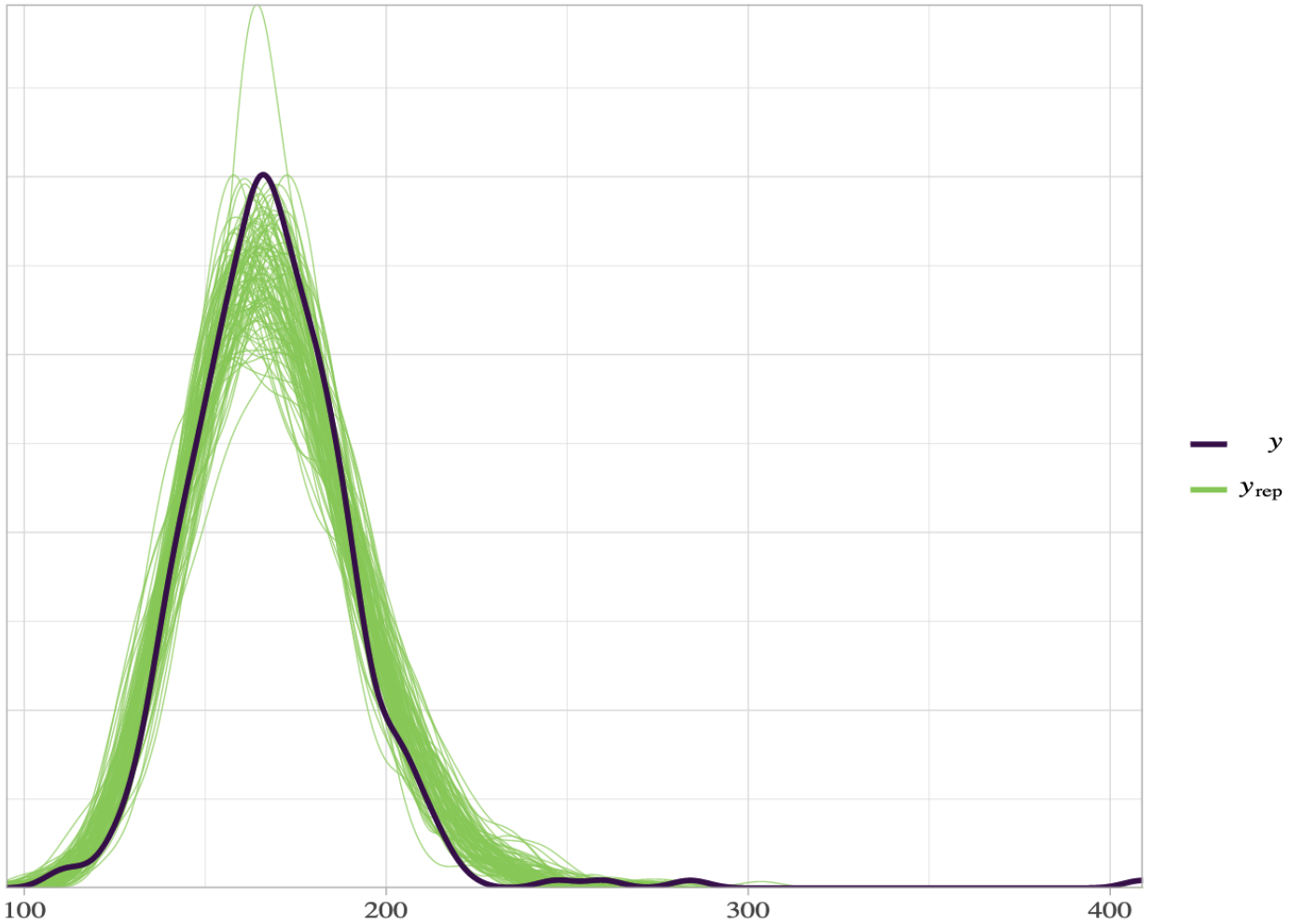
Prior of $\mu$ is now in log millisecond scale, not millisecond scale.
So if we expect the mean to be at 200ms, we need to calculate $exp(\mu) = 200$, so $\mu = 5.3$

It's similar (but slightly more complicated) for $\sigma$.[2]

[2] we'll skip this here.

# Posterior of the lognormal model: fits better!

```
pp_check(fit_press_ln, ndraws = 100)
```



# Summary

Important concepts:

- sampling from a distribution
- specifying iterations, warmup and chains.
- convergence checks
- choosing the likelihood and the prior
- effects of choosing different priors
- calculating the prior predictive distribution
- calculating the posterior predictive distribution
- checking how well these distributions reflect the properties of the data

for reference, see `https://vasishth.github.io/bayescogsci/book/`