Statistical Inference, pt. IV

ECON 3640–001

Marcio Santetti
Spring 2022

Motivation

Housekeeping

Notes based on `Johnson et al. (2022)`:

- Chapter 6
- Available `here`

Also, make sure to install the `rstan` package by following the directions in the book's `Preface`, *Getting set up* section.

Motivation

> *"When we can't know something, we approximate it."*

This is the motivation for today's lecture.

By this point, you have probably noticed that, in order to answer more **complex/flexible** questions, the **complexity** of statistical models grows in a rather absurd way.

However, the mathematical complexity does not change anything about the **logic of the process**.

Motivation

Recall the **proportion** (θ) of Bayesian researchers within the Social Sciences we were interested in a few lectures ago.

What if we are curious about the *same* proportion, but for *other* areas as well?

- e.g., Natural Sciences, Hard Sciences, etc.

With a quite simple question in mind, our problem goes from

$$P(\theta \mid y) = \frac{P(\theta)\ P(y \mid \theta)}{\int_{\theta} P(\theta)\ P(y \mid \theta)\ d\theta}$$

to

$$P(\vec{\theta} \mid y) = \frac{P(\theta)\ P(y \mid \theta)}{\int_{\theta_1} \int_{\theta_2} \int_{\theta_3} \int_{\theta_4} \int_{\theta_5} \ldots \int_{\theta_k} P(\theta)\ P(y \mid \theta)\ d\theta_k \ldots d\theta_2 d\theta_1}$$

Motivation

Even today's computers will **not** be able to properly calculate such posteriors.

But the **good news** is that we can approximate the posterior **via simulation**.

Here, we will explore two simulation techniques:

1. *Grid approximation*;
2. *Markov Chain Monte Carlo (MCMC)*.

When properly motivated, these techniques provide a **sample** of θ values whose properties reflect those of the "true" posterior distribution.

Grid approximation

Grid approximation

Grid approximation

Our "target image" is the posterior distribution.

if we are able to evaluate its PDF at a **discrete** and **finite** grid of possible θ values, we may get a *nice image* of what we are looking for.

The **recipe** 👨‍🍳 👨‍🍳:

1. Define a discrete grid of possible θ values;
2. Evaluate the **prior** and **likelihood** at each θ grid value;
3. Multiply the prior by the likelihood;
4. Normalize the product to get a posterior distribution that adds up/integrates to 1.

Grid approximation

In practice:

```
data_grid ← tibble(
  theta_grid = seq(from = 0, to = 1, by = 0.01),
  prior = dbeta(theta_grid, shape1 = 20, shape2 = 60),
  likelihood = dbinom(x = 30, size = 100, prob = theta_grid)
)
```

Grid approximation

In practice:

```
data_grid ← data_grid %>%
  mutate(unstd_posterior = prior * likelihood,
         std_posterior = unstd_posterior/sum(unstd_posterior))
```

Grid approximation

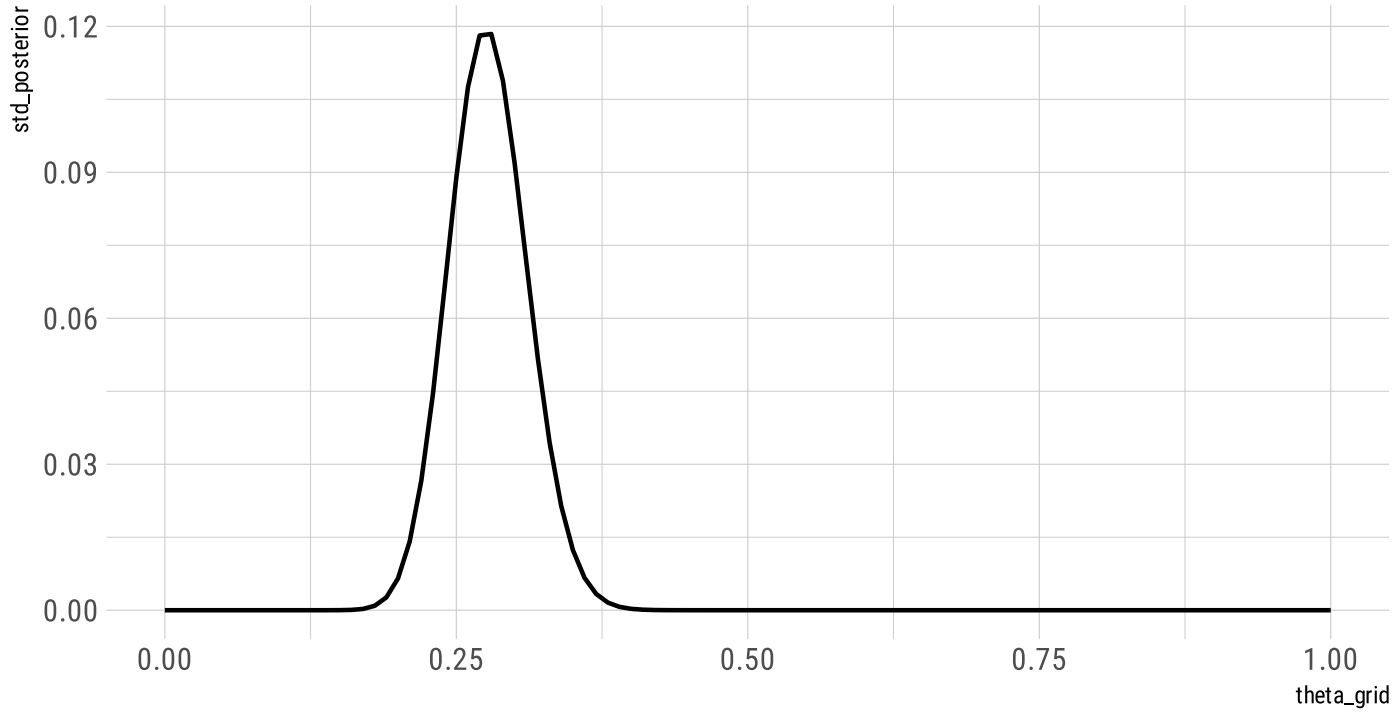
In practice:

```
data_grid %>%
  summarize(sum_posterior = sum(std_posterior))
```

```
#> # A tibble: 1 × 1
#>   sum_posterior
#>           <dbl>
#> 1             1
```

Grid approximation

```
data_grid %>%
  ggplot(aes(x = theta_grid, y = std_posterior)) +
  geom_line(size = .8)
```



Grid approximation

Check out section `6.1.2` of the `Bayes Rules!` book for a **Gamma-Poisson** example of grid approximation.

Grid approximation

Grid approximation is simple.

However...

As we try to increase the *number of parameters* we are curious about, grid approximation is not very effective.

It suffers from the "**curse of dimensionality**."

Markov Chain Monte Carlo

Markov Chain Monte Carlo

Given the limitations of grid approximation, **Markov Chain Monte Carlo** (MCMC) methods are the **most efficient and used** techniques to properly *approximate a posterior distribution*, even when it is too complicated to go over the Math.

The idea is *very simple*.

As a first step, we want to simulate a sample of θ values:

$$\{\theta^1, \theta^2, \theta^3, \ldots, \theta^N\}$$

And then, as a second step, use this sample to *approximate* the **main features** of the posterior distribution.

Markov Chain Monte Carlo

One way to start is through **Monte Carlo** simulations.

Monte Carlo simulations produce **random samples** of size *N*, where each value is *independent* of one another.

As a quick example, suppose we are interested in the how many cars pass by the *roundabout* close to the Stadium here at the U from 4:00 to 5:00 pm.

This is a **count** random variable case, which can be analyzed through a **Gamma-Poisson** model.

Markov Chain Monte Carlo

Suppose we have a prior belief that, on average, 250 cars pass by there within this hour, but we may allow for some uncertainty.

And we stop by the roundabout today at this time slot and collect some new data.

Here's a simulation:

```
set.seed(123)

dd ← tibble(

  lambda_par = rgamma(10000, shape = 500, rate = 2),
  y = rpois(10000, lambda = lambda_par)


)
```

Markov Chain Monte Carlo

```
# How many values from this simulation are equal to 250?

dd %>%
  filter(y == 250) %>%
  nrow()
```

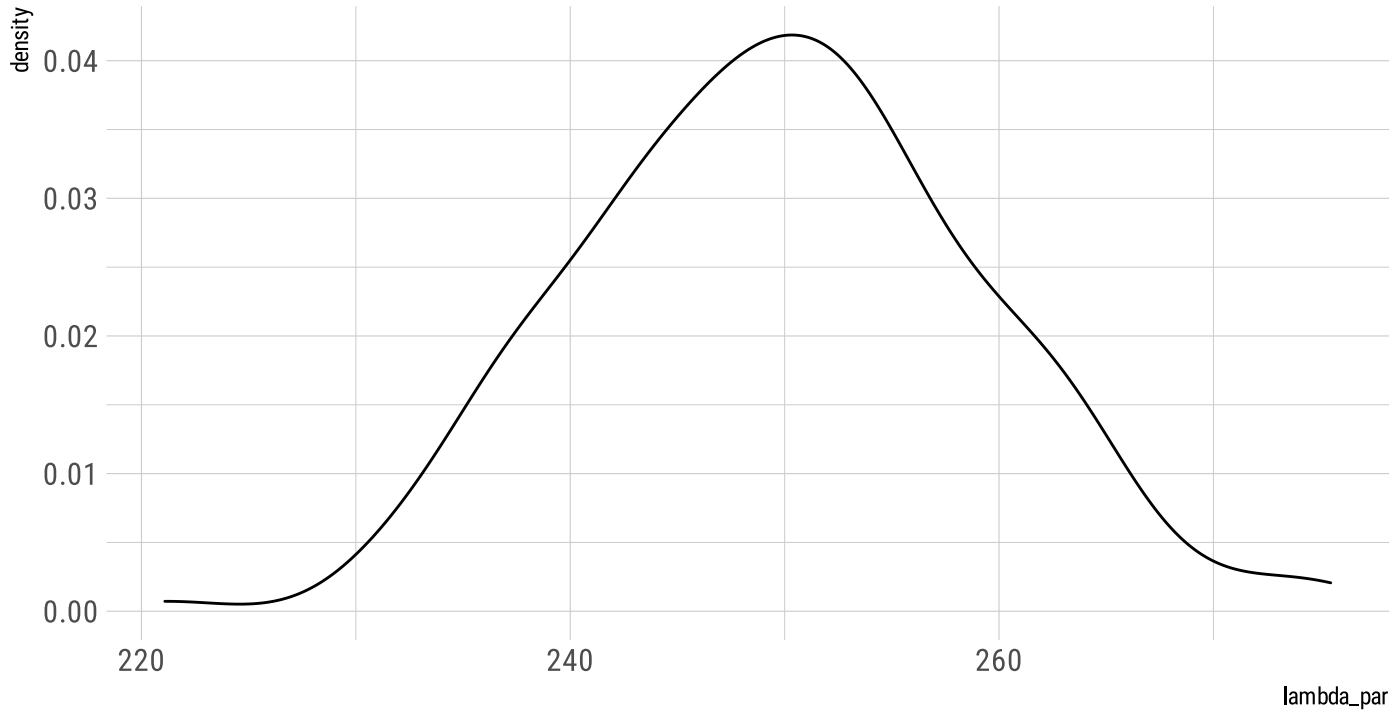
```
#> [1] 192
```

```
dd %>%
  filter(y == 250) %>%
  head(4)
```

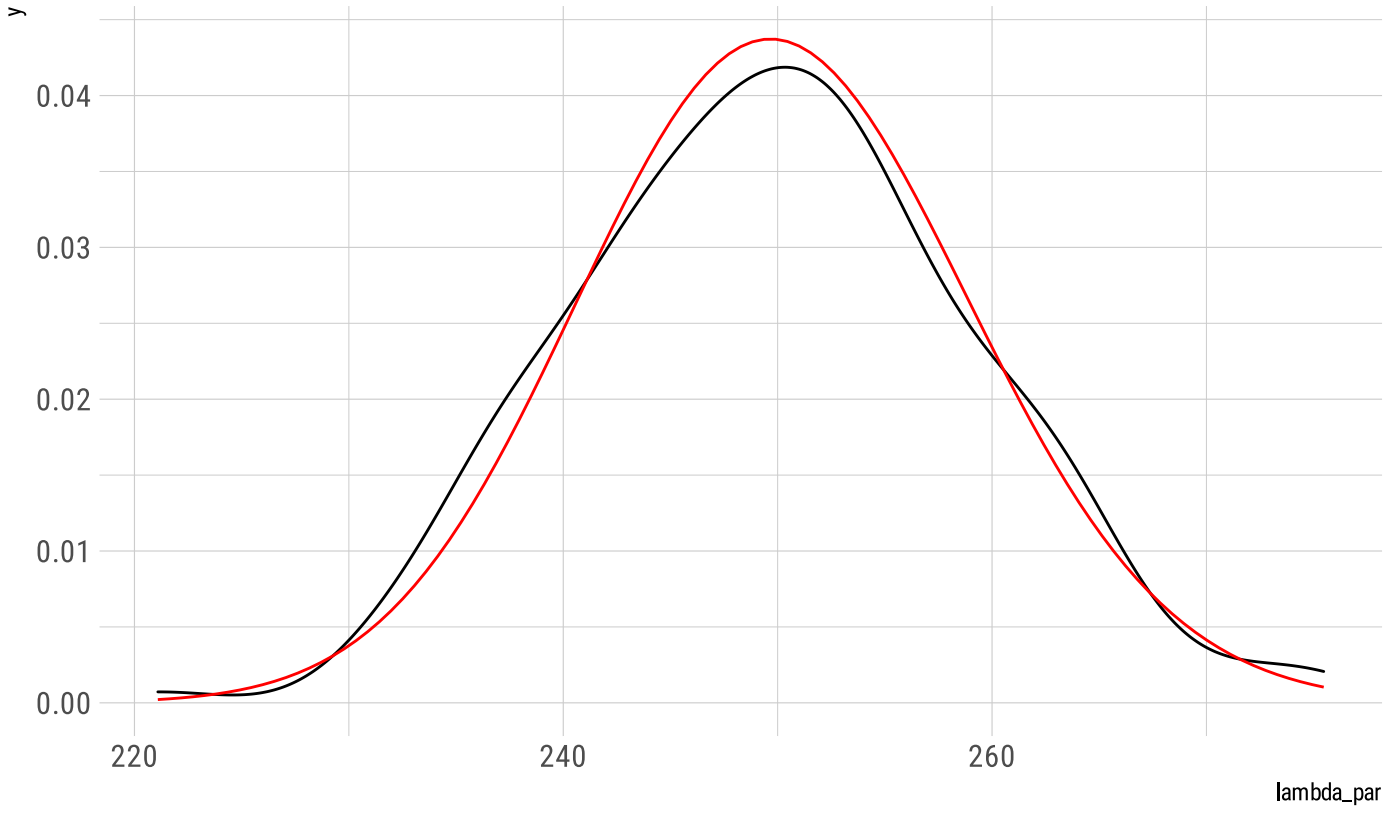
```
#> # A tibble: 4 × 2
#>   lambda_par     y
#>        <dbl> <int>
#> 1       261.   250
#> 2       243.   250
#> 3       238.   250
#> 4       252.   250
```

Markov Chain Monte Carlo

```
dd %>%
  filter(y == 250) %>%
  ggplot(aes(x = lambda_par)) +
  geom_density()
```



```r
dd %>%
  filter(y == 250) %>%
  ggplot(aes(x = lambda_par)) +
  geom_density() +
  stat_function(fun = dgamma, args = list(shape = 750, rate = 3), color = "red")
```



Markov Chain Monte Carlo

We can check the validity of a Monte Carlo simulation if we **know** the posterior's form.

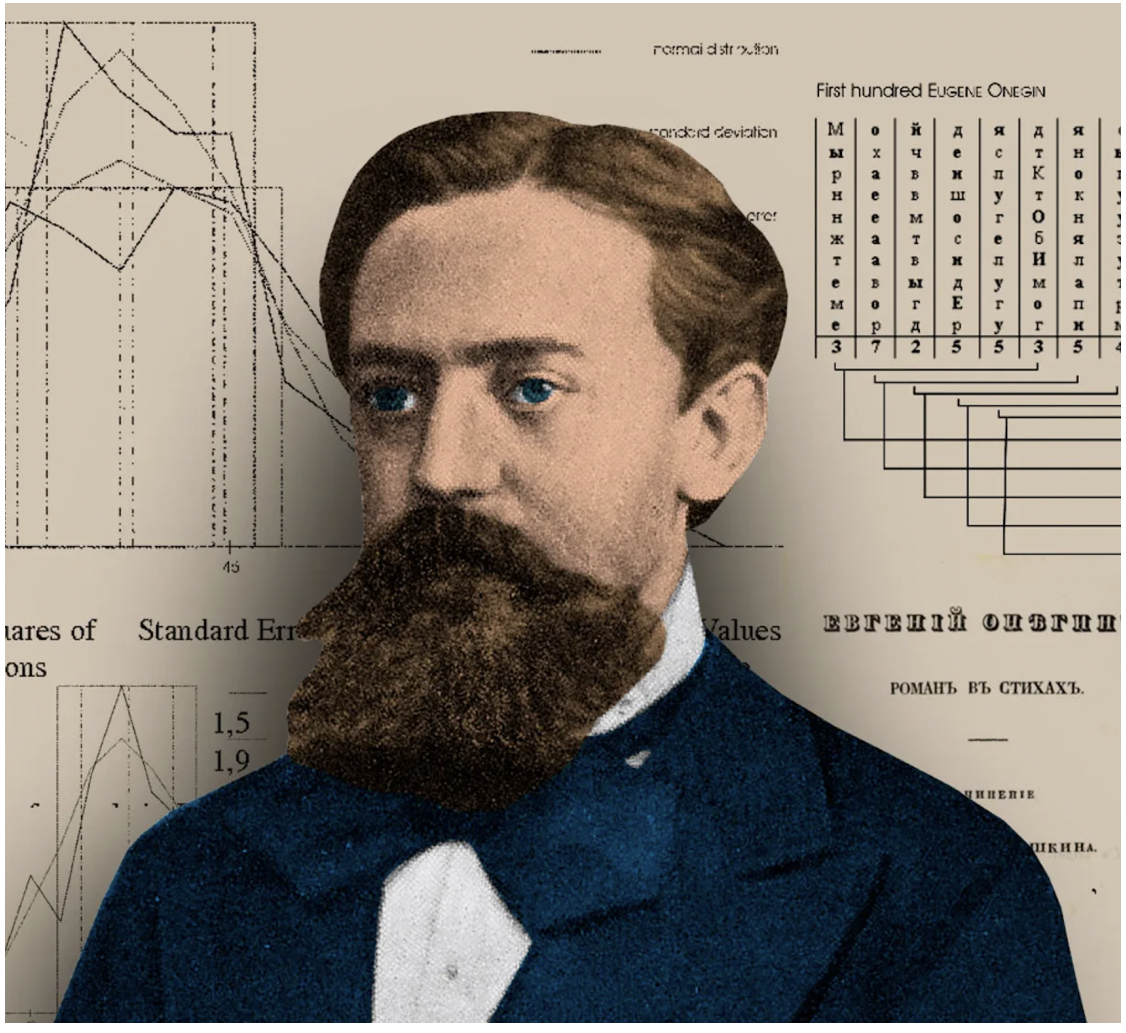
But how can we check our sanity if we **don't know** its form?

We introduce **Markov chains**.

MCMC follows the same principle as Monte Carlo simulation, but each value θ^{1+i} in the chain $\{\theta^1, \theta^2, \theta^3, \ldots, \theta^N\}$ **depends on** the previous value θ^i, but is **independent** of all other past values.

The chains visit various spots of the *sample space*, so we come up with a *nice picture* of the posterior distribution.

Markov Chain Monte Carlo



Markov Chain Monte Carlo

Markov Chain Monte Carlo

`Stan official website`

We will use the `rstan` R package to approximate teh posterior via MCMC.

In order to properly use this package, go back to the first slide and follow the directions to set it up.

MCMC in practice

MCMC in practice

The two main steps for approximating the posterior with MCMC methods using `rstan` are:

1. **Defining** the model's structure;

2. **Simulating** the posterior.

MCMC in practice

Defining the posterior:

```
your_model ← "
  data {
  }
  parameters {
  }
  model {
  }
"
```

Check out the `rstan-model-starter.R` file on Canvas!

MCMC in practice

We always start with the `model` portion:

```
your_model ← "
  data {
  }
  parameters {
  }
  model {
    Y ~ binomial(100, theta);  // the likelihood
    theta ~ beta(20, 60);     // the prior

  }
"
```

MCMC in practice

We then move on to **defining the parameters** we have included in the `model` section.

```
your_model ← "
  data {
  }
  parameters {
  real<lower=0, upper=1> theta;  // our parameter of interest is theta
  }
  model {
    Y ~ binomial(100, theta);  // the likelihood
    theta ~ beta(20, 60);     // the prior

  }
"
```

Now, we deal with the remaining **data** and **hyperparameters** in the `data` section.

```
your_model ← "

  data {

    int<lower=1> n;                // the number of trials
    int<lower=0, upper=n> Y;     // the number of sucesses
    real<lower=0> alpha;         // the beta prior's alpha hyperparameter
    real<lower=0> beta;         // the beta prior's beta hyperparameter
  }

  parameters {

    real<lower=0, upper=1> theta;  // our parameter of interest is theta

  }

  model {

    Y ~ binomial(100, theta);  // the likelihood

    theta ~ beta(20, 60);      // the prior

  }

"
```

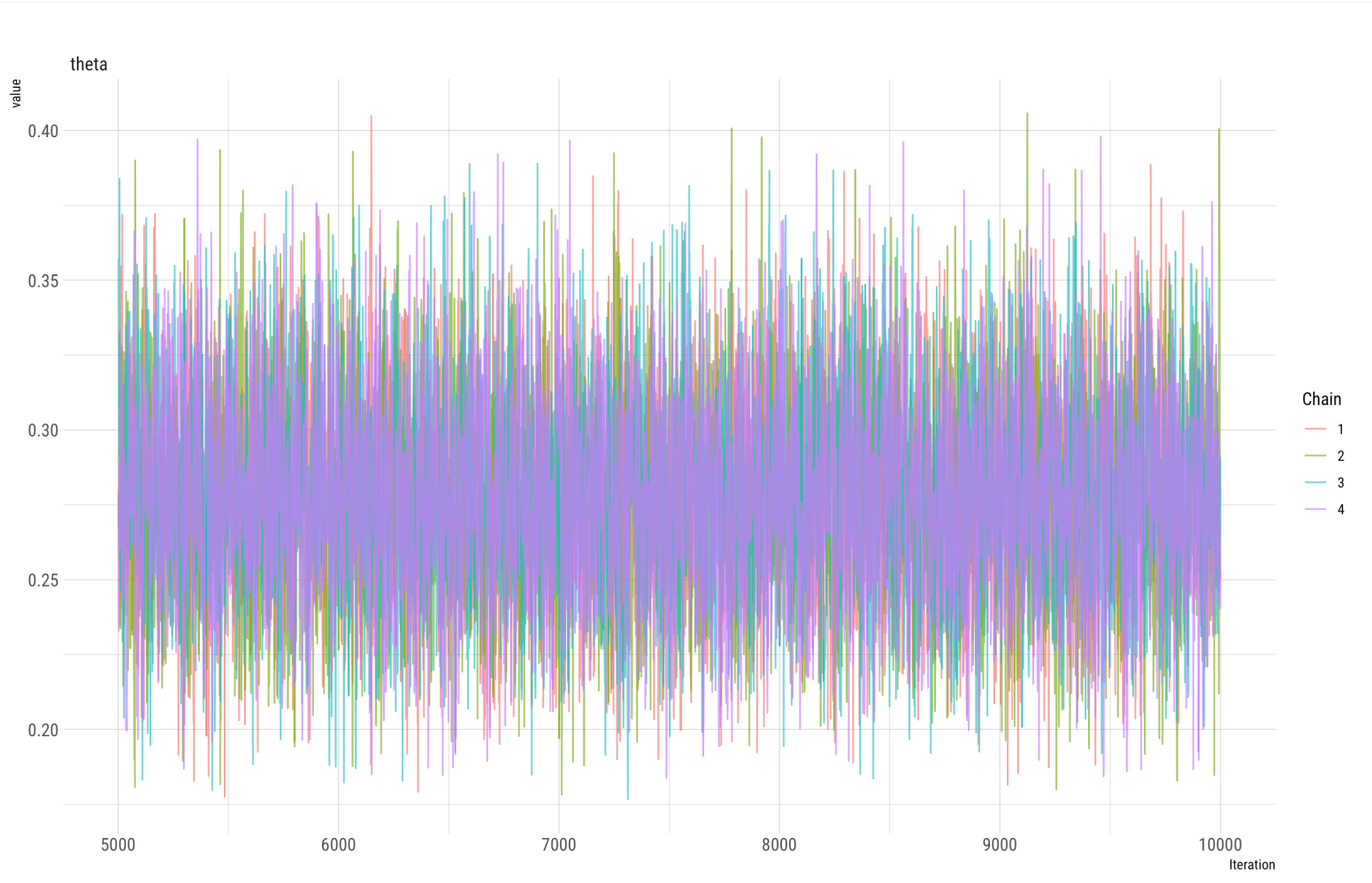
MCMC in practice

Defining the model is the **hardest** part.

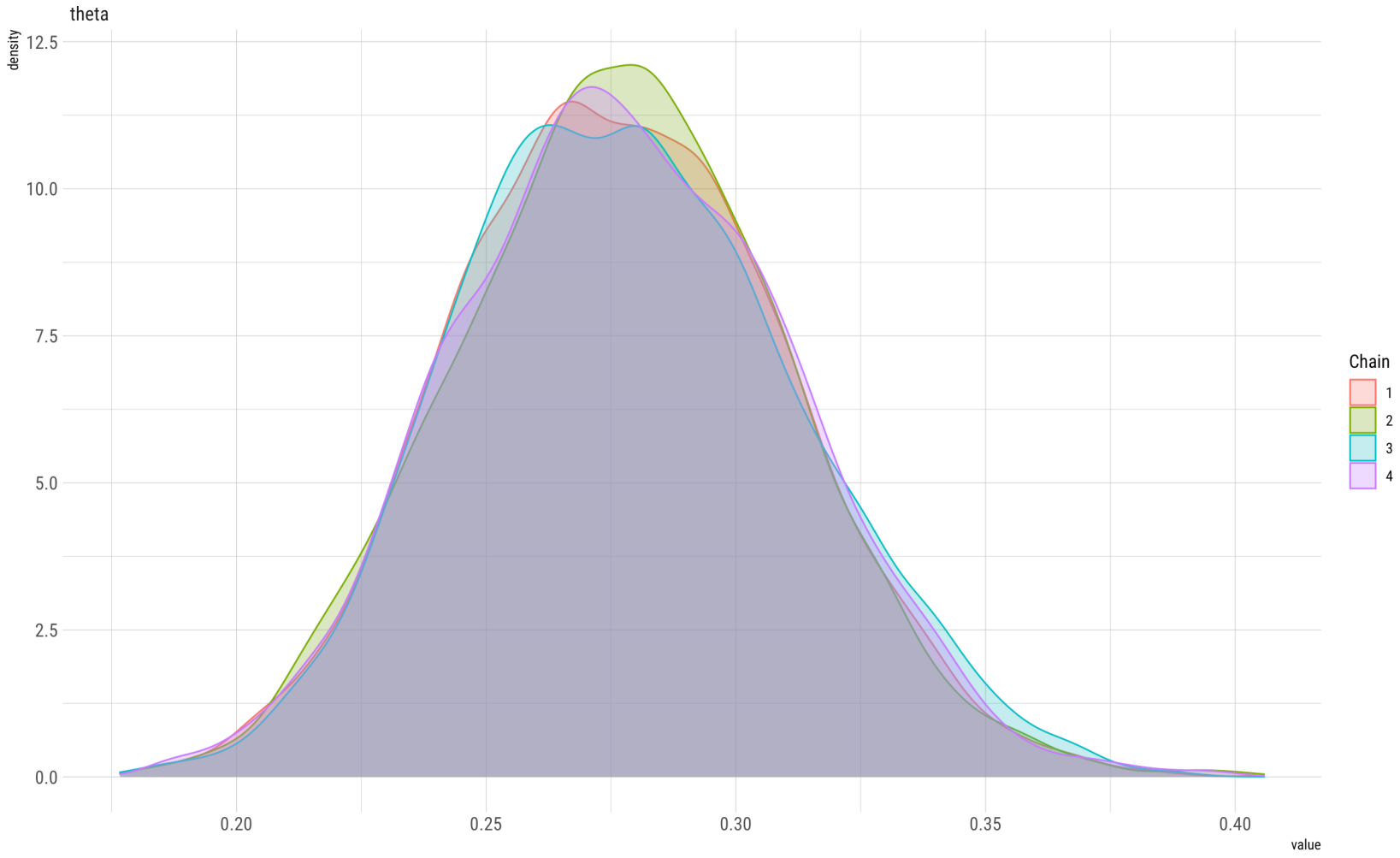
After this gets done, `rstan` does the heavy lifting for us.

```
library(rstan)
options(mc.cores = parallel::detectCores())  ## using your computer cores.


# Simulating the posterior:

set.seed(123)    ## don't forget to set a seed!

model_sim ← stan(

  model_code = your_model,    ## the model from before

  data = list(alpha = 20, beta = 60, Y = 30, n = 100),  ## the actual data and hyperparameters

  chains = 4, iter = 5000 * 2     ## run 4 parallel Markov chains, with 10,000 simulations

)
```

MCMC in practice



MCMC in practice



Next time: Posterior inference