ISYE 6420 - BUGS to Stan

Josh Blakely

2/15/23

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

```
1 + 1
```

```
[1] 2
```

install.packages("cmdstanr", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))

Part I

UNIT 1

About this course and website

The course was originally developed at Georgia Tech in 2004 by Professor Brani Vidakovic, who is now the head of the Texas A&M Department of Statistics. Many of the individual examples are related to biostatistics (Prof. Vidakovic wrote the textbook *Engineering Biostatistics*), but the methods are broadly applicable.

> See also Professor Vidakovic's publication history.
>
> Professor Joseph's.

This site mostly follows the original course outline. Each example lists the corresponding lecture video and contains a download link to the original code file(s). Only the lecture video where the professor goes over the example code will be listed, but there may be other relevant lectures that you'll need to watch. Any necessary data will either have a download link or, if the data is compact enough, will be included in the code.

This is a supplement to the Canvas site, not a replacement!

Other recommended resources

These are not required for the class, but they might be helpful. Prof. Vidakovic's lectures often assume that the student has a certain amount of background knowledge, so if you feel lost or if you just want to dive deeper into the subject check them out.

Textbooks and courses

Statistical Rethinking by Richard McElreath is a great book for gaining intuition about Bayesian inference and modeling in general.

- main site
- lecture videos
- R and Stan code examples.

Bayesian Data Analysis by Gelman, Carlin, Stern, Dunson, Vehtari, and Rubin goes into more mathematical theory than *Statistical Rethinking.* I use it as a reference - not planning to try reading this one all the way through!

- main site
- pdf

Blogs

- Andrew Gelman, author of *Bayesian Data Analysis* (above), has a blog: Statistical Modeling, Causal Inference, and Social Science.
- Dan Simpson, one of the maintainers of the Stan PPL, has a blog called Un garçon pas comme les autres (Bayes) with opinionated and funny deep dives into various Bayesian topics. Warning: NSFW language.
- Count Bayesie: Probably a probability blog by Will Kurt.
- Michael Betancourt, another Stan developer, has a series of incredibly in-depth posts and notebooks on Bayesian modeling.
- PyMC developer Austin Rochford's blog has a lot of good posts.
- Another PyMC developer, Oriol Abril, posts some really helpful PyMC examples.

Podcasts

- Learn Bayes Stats by Alex Andorra, one of the PyMC developers.

Other

- Stan User's Guide
- Aaron Reding's PyMC

Part II

UNIT 2

1 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```r
1 + 1
```

```
[1] 2
```

```
install.packages("rstan", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))
install.packages("cmdstanr", repos = c("https://mc-stan.org/r-packages/", getOption("repos"))
```

Part III

UNIT 3

2 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
1 + 1
```

```
[1] 2
```

```
install.packages("rstan", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))
install.packages("cmdstanr", repos = c("https://mc-stan.org/r-packages/", getOption("repos"))
```

Part IV

UNIT 4

3 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```r
1 + 1
```

```
[1] 2
```

```
install.packages("rstan", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))
install.packages("cmdstanr", repos = c("https://mc-stan.org/r-packages/", getOption("repos"))
```

Part V

UNIT 5

4 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
1 + 1
```

```
[1] 2
```

```
install.packages("rstan", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))
install.packages("cmdstanr", repos = c("https://mc-stan.org/r-packages/", getOption("repos"))
```

Part VI

UNIT 6

Stress, Diet, and Plasma Acids

```r
library(cmdstanr)
```

This example introduces tracking of deterministic variables and shows how to recreate the BUGS step function in Stan.

It is adapted from Unit 6: stressacids.odc.

Associated lecture video: Unit 6 lesson 5

Problem Statement

In the study Interrelationships Between Stress, Dietary Intake, and Plasma Ascorbic Acid During Pregnancy conducted at the Virginia Polytechnic Institute and State University, the plasma ascorbic acid levels of pregnant women were compared for smokers versus non-smokers. Thirty-two women in the last three months of pregnancy, free of major health disorders, and ranging in age from 15 to 32 years were selected for the study. Prior to the collection of 20 ml of blood, the participants were told to avoid breakfast, forego their vitamin supplements, and avoid foods high in ascorbic acid content. From the blood samples, the plasma ascorbic acid values of each subject were determined in milligrams per 100 milliliters.

I start with the data pasted from `stressacids.odc`, then create one list for smokers and one for nonsmokers.

```r
plasma <- c(0.97, 0.72, 1.00, 0.81, 0.62, 1.32, 1.24, 0.99, 0.90, 0.74,
            0.88, 0.94, 1.06, 0.86, 0.85, 0.58, 0.57, 0.64, 0.98, 1.09,
            0.92, 0.78, 1.24, 1.18, 0.48, 0.71, 0.98, 0.68, 1.18, 1.36,
            0.78, 1.64)

smo <- c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2)
```

```
nonsmoker_index <- which(smo == 1)
plasma_smokers <- plasma[-nonsmoker_index]
plasma_nonsmokers <- plasma[nonsmoker_index]
```

BUGS `step` function

BUGS defines the step function like this:

$$step(e) = \begin{cases} 1, & e \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Stan follows what you would expect in a programming language. We implement this like:

```
e >= 0 ? 1 : 0;
```

How do I track non-random variables in Stan?

One nice thing about BUGS is you can easily track both deterministic and non-deterministic variables while sampling. For Stan, you add the variable to the `generated quantities` block.

```
mod <- cmdstan_model(stress_diet_stan)
```

Stress, Diet, and Plasma Acids Model

```
data {
  int<lower=0> N_smoker;
  int<lower=0> N_nonsmoker;
  vector[N_smoker] plasma_smoker;
  vector[N_nonsmoker] plasma_nonsmoker;
}

parameters {
  real<lower=0> tau_nonsmoker;
  real mu_nonsmoker;
  real<lower=0> tau_smoker;
  real mu_smoker;
}

model {
  tau_nonsmoker ~ gamma(0.0001, 0.0001);
  tau_smoker ~ gamma(0.0001, 0.0001);
  mu_nonsmoker ~ normal(0, 100); // equivalent to BUGS tau = 0.0001
  mu_smoker ~ normal(0, 100);

  plasma_smoker ~ normal(mu_smoker, 1 / sqrt(tau_smoker));
  plasma_nonsmoker ~ normal(mu_nonsmoker, 1 / sqrt(tau_nonsmoker));
}

generated quantities {
  real testmu;
  real r;

  testmu = (mu_smoker >= mu_nonsmoker ? 1 : 0);
  r = tau_nonsmoker / tau_smoker;
}
```

Sampling

Let us prepare the data to be passed into sample

```
input_data <- list(N_smoker=length(plasma_smokers),
                   N_nonsmoker=length(nonsmoker_index),
                   plasma_smoker=plasma_smokers,
                   plasma_nonsmoker=plasma_nonsmokers
)
```

Now that we have the data to pass into our sampling, let's proceed.

```
fit <- mod$sample(
  data = input_data,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh = 500,   # print update every 500 iterations
  iter_warmup = 1000,
  iter_sampling = 5000
)
```

```
fit$summary()
```

```
# A tibble: 7 x 10
  variable        mean median     sd    mad     q5    q95  rhat ess_bulk ess_tail
  <chr>          <num>  <num>  <num>  <num>  <num>  <num> <num>    <num>    <num>
1 lp__         27.8   28.1   1.52   1.31   24.9   29.6    1.00    7997.    9743.
2 tau_nonsmok~ 22.6   21.9   6.68   6.53   12.9   34.5    1.00   17854.   12231.
3 mu_nonsmoker  0.912  0.912 0.0449 0.0436  0.839  0.986  1.00   17455.   12780.
4 tau_smoker    6.53   5.92  3.50   3.20    2.03  13.1    1.00   13257.   10335.
5 mu_smoker     0.977  0.977 0.162  0.145   0.721  1.24   1.00   13512.    9947.
6 testmu        0.667  1     0.471  0       0      1      1.00   16274.      NA
7 r             4.83   3.72  4.27   2.19    1.43  11.7    1.00   14304.   11261.
```

These results are very similar to PyMC results.

Dugongs

```r
library(cmdstanr)
```

This example is the first example of dealing with missing data.

It is adapted from Unit 6: dugongsmissing.odc.

Associated lecture video: Unit 6 lesson 6

Problem Statement

Carlin and Gelfand (1991) investigated the age (x) and length (y) of 27 captured dugongs (sea cows). Estimate parameters in a nonlinear growth model.

References

Data provided by Ratkowsky (1983).

Carlin, B. and Gelfand, B. (1991). An Iterative Monte Carlo Method for Nonconjugate Bayesian Analysis, Statistics and Computing,

Ratkowsky, D. (1983). Nonlinear regression modeling: A unified practical approach. M. Dekker, NY, viii, 276 p.

Input Data

```r
X <- c(1.0, 1.5, 1.5, 1.5, 2.5, 4.0, 5.0, 5.0, 7.0, 8.0, 8.5, 9.0, 9.5,
     9.5, 10.0, 12.0, 12.0, 13.0, 13.0, 14.5, 15.5, 15.5, 16.5, 17.0,
     22.5, 29.0, 31.5)
y <- c(1.80, 1.85, 1.87, -1, 2.02, 2.27, 2.15, 2.26, 2.47, 2.19, 2.26,
     2.40, 2.39, 2.41, 2.50, 2.32, 2.32, 2.43, 2.47, 2.56, 2.65, 2.47,
     2.64, 2.56, 2.70, 2.72, -1)
```

Stan imputes missing values differently from PyMC and Bugs. We have to pass in the indices for the missing values as well as the count of observed and missing values.

```r
mod <- cmdstan_model(dugongs_stan)
```

now that we have compiled our stan file, we can print out the model that we will use for this:

```r
mod$print()
```

```
data {
  int<lower=0> N_obs;
  int<lower=0> N_mis;
  array[N_obs] int<lower=1, upper=N_obs + N_mis> iy_obs;  // index of obs y's
  array[N_mis] int<lower=1, upper=N_obs + N_mis> iy_mis;  // index of mis y's
  vector[N_obs + N_mis] X;
  vector[N_obs] y_obs;    // actual y values that were observed
}

transformed data {
  int<lower=0> N = N_obs + N_mis;  // total size of the input
}

parameters {
  real alpha;
  real beta;
  real gamma;
  real sigma;
  vector[N_mis] y_mis;  // account for missing y
}

transformed parameters {
  vector[N] Y;         // Imputed Y values
  Y[iy_obs] = y_obs;   // actual y values
  Y[iy_mis] = y_mis;   // missing y values
}

model {
  // priors
  alpha ~ uniform(0, 100);
  beta ~ uniform(0, 100);
  gamma ~ uniform(0, 1);
  sigma ~ uniform(-10, 10);
  Y ~ normal(alpha - beta * pow(gamma, X), sigma);
}
```

Sampling

Let us prepare the data to be passed into sample

```
# Initial data has `-1` where missing.
idx.mis <- which(y == -1)
idx.obs <- which(y != -1)

input_data <- list(N_obs=length(idx.obs), N_mis=length(idx.mis),
                  iy_obs=idx.obs, iy_mis=idx.mis,
                  X=X,
                  y_obs=y[idx.obs]
)
```

Now that we have the data to pass into our sampling, let's proceed.

```
fit <- mod$sample(
  data = input_data,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh = 500,   # print update every 500 iterations
  iter_warmup = 1000,
  iter_sampling = 5000
)
```

```
fit$summary()
```

```
# A tibble: 34 x 10
   variable   mean  median     sd    mad      q5    q95  rhat ess_bulk ess_tail
   <chr>     <num>   <num>  <num>  <num>   <num>  <num> <num>    <num>    <num>
 1 lp__     49.5   49.9    2.09   1.90    45.5    52.2    1.00     5446.    7258.
 2 alpha     2.73   2.71   0.125  0.106    2.57    2.96   1.00     3806.    2567.
 3 beta      0.986  0.978  0.105  0.0925   0.833   1.16   1.00     5014.    2863.
 4 gamma     0.886  0.891  0.0358 0.0313   0.823   0.936  1.00     4033.    3205.
 5 sigma     0.100  0.0980 0.0165 0.0153   0.0771  0.130  1.00     8360.    8205.
 6 y_mis[1]  1.91   1.91   0.114  0.111    1.72    2.09   1.00    11392.   11007.
 7 y_mis[2]  2.69   2.69   0.128  0.123   2.48    2.90   1.00    6343.    6816.
 8 Y[1]      1.8    1.8    0      0       1.8     1.8   NA          NA       NA
 9 Y[2]      1.85   1.85   0      0       1.85    1.85  NA          NA       NA
10 Y[3]      1.87   1.87   0      0       1.87    1.87  NA          NA       NA
# i 24 more rows
```

These results are very similar to PyMC results.

Equivalence of Generic and Brand-name Drugs

```r
library(cmdstanr)
```

This example introduces tracking of deterministic variables and shows how to recreate the BUGS step function in Stan.

It is adapted from Unit 6: equivalence.odc.

Associated lecture video: Unit 6 lesson 7

Problem Statement

The manufacturer wishes to demonstrate that their generic drug for a particular metabolic disorder is equivalent to a brand name drug. One of indication of the disorder is an abnormally low concentration of levocarnitine, an amino acid derivative, in the plasma. The treatment with the brand name drug substantially increases this concentration.

A small clinical trial is conducted with 43 patients, 18 in the Brand Name Drug arm and 25 in the Generic Drug arm. The increases in the log-concentration of levocarnitine are in the data below.

The FDA declares that bioequivalence among the two drugs can be established if the difference in response to the two drugs is within 2 units of log-concentration. Assuming that the log-concentration measurements follow normal distributions with equal population variance, can these two drugs be declared bioequivalent within a tolerance +/-2 units?

Input Data

Starting with the data pasted from `equivalence.odc`.

```r
increase.1 <- c(7, 8, 4, 6, 10, 10, 5, 7, 9, 8, 6, 7, 8, 4, 6, 10, 8, 9)
increase.2 <- c(6, 7, 5, 9, 5, 5, 3, 7, 5, 10, 8, 5, 8, 4, 4, 8, 6, 11, 7, 5, 5, 5, 7, 4,
```

How do I track non-random variables in Stan?

One nice thing about BUGS is you can easily track both deterministic and non-deterministic variables while sampling. For Stan, you add the variable to the `generated quantities` block.

```r
mod <- cmdstan_model(equivalence_stan)
```

Stress, Diet, and Plasma Acids Model

```stan
data {
  int<lower=0> N;
  int<lower=0> M;
  vector[N] y_type1;
  vector[M] y_type2;
}

parameters {
  real mu_type1;
  real mu_type2;
  real prec;
}

transformed parameters {
  real mudiff = mu_type1 - mu_type2;
  real sigma = 1 / sqrt(prec);
  real probint = ((mudiff + 2) >= 0 ? 1 : 0) * ((2 - mudiff) >= 0 ? 1 : 0);
}

model {
  mu_type1 ~ normal(10, 1 / sqrt(1e-5));
  mu_type2 ~ normal(10, 1 / sqrt(1e-5));
  prec ~ gamma(0.001, 0.001);

  y_type1 ~ normal(mu_type1, sigma);
  y_type2 ~ normal(mu_type2, sigma);
}
```

Sampling

Let us prepare the data to be passed into sample

```r
input_data <- list(N=length(increase.1),
                   M=length(increase.2),
                   y_type1=increase.1,
                   y_type2=increase.2
)
```

Now that we have the data to pass into our sampling, let's proceed.

```r
fit <- mod$sample(
  data = input_data,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh = 500,   # print update every 500 iterations
  iter_warmup = 1000,
  iter_sampling = 5000
)
```

```r
fit$summary()
```

```
# A tibble: 7 x 10
  variable    mean  median     sd    mad      q5     q95  rhat ess_bulk ess_tail
  <chr>      <num>   <num>  <num>  <num>   <num>   <num> <num>    <num>    <num>
1 lp__     -49.4   -49.1   1.22   0.978  -51.8   -48.1    1.00    9921.   12770.
2 mu_type1   7.33    7.34  0.467  0.461    6.57    8.09   1.00   16880.   13439.
3 mu_type2   6.20    6.21  0.400  0.393    5.55    6.86   1.00   16065.   13287.
4 prec       0.263   0.259 0.0581 0.0575   0.176   0.365  1.00   15209.   11437.
5 mudiff     1.13    1.13  0.617  0.616    0.122   2.14   1.00   16797.   13707.
6 sigma      1.99    1.96  0.226  0.218    1.66    2.39   1.00   15209.   11437.
7 probint    0.922   1     0.268  0        0       1      1.00   14222.      NA
```

These results are very similar to PyMC results and BUGS results.

Psoriasis: Two Sample Problem - paired data

```r
library(cmdstanr)
```

This is our first example of hypothesis testing.

It is adapted from Unit 6: psoriasis.odc.

Associated lecture video: Unit 6 lesson 7

Problem Statement

Woo and McKenna (2003) investigated the effect of broadband ultraviolet B (UVB) therapy and topical calcipotriol cream used together on areas of psoriasis. One of the outcome variables is the Psoriasis Area and Severity Index (PASI), where a lower score is better.

The PASI scores for 20 subjects are measured at baseline and after 8 treatments. Do these data provide sufficient evidence to indicate that the combination therapy reduces PASI scores?

Classical Analysis:

```
d = baseline - after;
n = length(d);
dbar = mean(d);   dbar = 6.3550
sdd = sqrt(var(d)); sdd = 4.9309
tstat = dbar / (sdd / sqrt(n));  tstat = 5.7637

Reject H_0 at the level alpha = 0.05 since the p_value = 0.00000744 < 0.05

95% CI is [4.0472, 8.6628]
```

See Unit 6: Stress, Diet and Plasma Acids to find out more about recreating the BUGS step function.

```r
baseline <- c(5.9, 7.6, 12.8, 16.5, 6.1, 14.4, 6.6, 5.4, 9.6, 11.6 ,11.1, 15.6, 9.6, 15.2,
after <- c(5.2, 12.2, 4.6, 4, 0.4 , 3.8, 1.2, 3.1, 3.5, 4.9, 11.1, 8.4, 5.8, 5, 6.4, 0, 2.
mod <- cmdstan_model(psoriasis_stan)
```

We do get a decent amount of warnings, but the model compiles and runs.

Model

```
// Psoriasis: Two Sample Problem - Paired Data
data {
  int<lower=0> N;
  vector[N] baseline;
  vector[N] after;
}

transformed data {
  vector[N] diff = baseline - after;
}

parameters {
  real mu;
  real prec;
}

transformed parameters {
  real sigma = 1 / sqrt(prec);
  real ph1;
  ph1 = (mu >= 0 ? 1 : 0);
}

model {
  mu ~ normal(0, 316);
  prec ~ gamma(0.001, 0.001);

  diff ~ normal(mu, sigma);
}
```

Sampling

Let us prepare the data to be passed into sample

```
input_data <- list(N=length(baseline),
                   baseline=baseline,
                   after=after
)
```

Now that we have the data to pass into our sampling, let's proceed.

```
fit <- mod$sample(
  data = input_data,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh = 500,    # print update every 500 iterations
  iter_warmup = 1000,
  iter_sampling = 5000
)
```

```
fit$summary()
```

```
# A tibble: 5 x 10
  variable     mean   median     sd    mad       q5      q95  rhat ess_bulk
  <chr>       <num>    <num>  <num>  <num>    <num>    <num> <num>    <num>
1 lp__     -39.2    -38.9    1.02   0.714  -41.2    -38.3     1.00    8495.
2 mu         6.37     6.37   1.18   1.12     4.45     8.27    1.00    9440.
3 prec       0.0411   0.0398 0.0133 0.0129   0.0218   0.0651  1.00    9661.
4 sigma      5.14     5.01   0.893  0.814    3.92     6.78    1.00    9661.
5 ph1        1        1      0      0        1        1      NA         NA
# i 1 more variable: ess_tail <num>
```

These results are very similar to BUGS and PyMC results.

Taste of Cheese

```
library(cmdstanr)
```

Adapted from Unit 6:

The data was downloaded from here and there is a copy here. If you wish to download the data, then right click and save the link as a csv file.

Problem Statement

As cheddar cheese matures, a variety of chemical processes take place. The taste of matured cheese is related to the concentration of several chemicals in the final product. In a study of cheddar cheese from the LaTrobe Valley of Victoria, Australia, samples of cheese were analyzed for their chemical composition and were subjected to taste tests. Overall taste scores were obtained by combining the scores from several tasters.

Can the score be predicted well by the predictors: `Acetic`, `H2S`, and `Lactic`?

```
df <- read.csv(file.path('..', 'data', 'cheese.csv'), header = T, colClasses = c("NULL", N

# This data list will be passed into the stan model
data_list <- list(
  N = dim(df)[1],
  acetic = df$Acetic,
  h2s = df$H2S,
  lactic = df$Lactic,
  y = df$taste
)
```

Model

We will compile/build the stan model by running `cmdstan_model`. The `cheese_program` is the file path for the stan file, i.e. `../cheese.stan`.

```r
mod <- cmdstan_model(cheese_stan)
```

Now that we have compiled our stan file, we can print out the model:

```r
mod$print()
```

```
// The data that is input into the model
data {
  int<lower=0> N;
  vector[N] acetic;
  vector[N] h2s;
  vector[N] lactic;
  vector[N] y;
}

parameters {
  real b0;                // Intercept coefficient
  real b1;                // Slope coefficient
  real b2;
  real b3;
  real tau;
}

model {
  b0 ~ normal(0, 1 / sqrt(1e-5));
  b1 ~ normal(0, 1 / sqrt(1e-5));
  b2 ~ normal(0, 1 / sqrt(1e-5));
  b3 ~ normal(0, 1 / sqrt(1e-5));
  tau ~ gamma(0.001, 0.001);

  y ~ normal(b0 + b1 * acetic + b2 * h2s + b3 * lactic, 1 / sqrt(tau));
}

generated quantities {
  array[N] real y_hat;
  y_hat = normal_rng(b0 + b1 * acetic + b2 * h2s + b3 * lactic, 1 / sqrt(tau));
}
```

Sampling

Now that we have created the model above and compiled the stan program, we can start sampling.

```r
fit <- mod$sample(
  data = data_list,
  seed = 1,
  chains = 4,
  parallel_chains = 4,
  refresh = 500,   # print update every 500 iterations
  iter_warmup = 1000,
  iter_sampling = 5000   # iterate 5,000 times
)
```

```r
fit$summary(variables = c('b0', 'b1', 'b2', 'b3', 'tau'))
```

```
# A tibble: 5 x 10
  variable      mean    median       sd       mad       q5      q95  rhat ess_bulk
  <chr>        <num>     <num>    <num>     <num>    <num>    <num> <num>    <num>
1 b0       -28.6     -28.6     20.6      20.3     -6.26e+1  4.76    1.00    7508.
2 b1         0.262     0.232    4.68      4.59    -7.32e+0  7.89    1.00    7056.
3 b2         3.92      3.91     1.30      1.29     1.78e+0  6.04    1.00    8871.
4 b3        19.7      19.6      9.02      8.70     4.87e+0 34.4     1.00    9316.
5 tau        0.00973   0.00949  0.00270   0.00267  5.75e-3  0.0145  1.00   11940.
# i 1 more variable: ess_tail <num>
```

The results are pretty close to OpenBUGS.

Predictions

In order to get the predictions, we will just need to pass in a new list of parameters with the predicted value

```r
pred_data <- list(
  N = 1,
  acetic = 5.0,
  h2s = 7.1,
  lactic = 1.5,
  y = 0 # We don't care what the value is for this because we are just trying to predict
)

pred <- mod$generate_quantities(fitted_params = fit, data = pred_data)
```

```r
print(pred)
```

```
variable  mean median    sd   mad    q5   q95
y_hat[1] 30.29  30.27 11.28 10.95 11.61 48.63
```

Part VII

UNIT 7

5 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
1 + 1
```

```
[1] 2
```

```
install.packages("rstan", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))
install.packages("cmdstanr", repos = c("https://mc-stan.org/r-packages/", getOption("repos"))
```

Part VIII

UNIT 8

6 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
1 + 1
```

```
[1] 2
```

```
install.packages("rstan", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))
install.packages("cmdstanr", repos = c("https://mc-stan.org/r-packages/", getOption("repos"))
```

Part IX

UNIT 9

7 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
1 + 1
```

```
[1] 2
```

```
install.packages("rstan", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))
install.packages("cmdstanr", repos = c("https://mc-stan.org/r-packages/", getOption("repos"))
```

Part X

UNIT 10

8 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
1 + 1
```

```
[1] 2
```

```
install.packages("rstan", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))
install.packages("cmdstanr", repos = c("https://mc-stan.org/r-packages/", getOption("repos")
```

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.