FinalAssignment

Niko Itänen

2024-12-09

Initialization

```
# Import all libraries to be used
library(StanHeaders)
library(rstan)
```

```
##
## rstan version 2.32.6 (Stan version 2.32.2)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using ‘reduce_sum()‘ or ‘map_rect()‘ Stan functions,
## change ‘threads_per_chain‘ option:
## rstan_options(threads_per_chain = 1)

## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
```

```
library(ggplot2)
library(lintr)
library(gridExtra)

# For some reason, no data files could be found, so I had to put the working directory here.
setwd("C:/Users/itane/Documents/GitHub/ProbabilisticProgramming/final_assignment")

# Rstan noticed to set options, so I set them here.
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)
rstan_options(threads_per_chain = 1)
```

1. Explanations

1.1 What is the difference between an informative and a non-informative prior?

Informative prior and non-informative prior are two different kind of terms that are used in Bayesian statistics. They are both used to describe the prior distribution before the data itself is observed, but the main difference here is how much data prior contains itself. [1, p.37-56]

Informative prior is a prior which contains already known information about the parameter, such as previous studies, historical data or expert knowledge. The weather forecast is a good example of informative prior, where we can use the previous weather data to predict the future weather by using the previously observed information about the weather patterns.

Informative prior can direct the posterior distribution towards a specific direction, which can be a good thing especially when we have limited amount of data. [1, p.37-50], [3]

Non-informative prior, on the other hand, is a prior which does not contain significant information about the parameter, while it try to stay neutral as possible. It can be used in cases where we have no clear prior knowledge of the parameter, or when we want to give more weight to the information in the analysis. A good example of a non-informative prior could be the estimation of the probability of a coin flip, where the distribution is somewhere between 0 and 1. In this case, the prior distribution cannot favor of these values over the others. [1, p.51-56], [4]

1.2 Given N posterior samples (α_i, β_i), where i=1,. . . ,N is the sample index, how can you compute marginal posterior samples for square roor of alpha?

Because $\alpha_1, \alpha_2, ..., \alpha_n$ are posterior samples for α, we can calculate the square root for each of these samples to get the marginal posterior samples.

This will result in the following transformation:

$$(\sqrt{\alpha_i} | i = 1, ..., N)$$

Now this result of $\sqrt{\alpha_1}, \sqrt{\alpha_2}, ..., \sqrt{\alpha_n}$ represents the samples from the marginal posterior distribution of $\sqrt{\alpha}$.

This can happen like this because Bayesian principles allow us to transform the posterior samples straight from the posterior samples of the original parameter. [1, p. 23]

1.3 Given the samples $\sqrt{\alpha}_i$ in 1.2, how do you compute the posterior probability that -1 < $\sqrt{\alpha}$ < 1?

Firstly, we need need to determine how many samples are between -1 and 1. Then we need to compute the posterior probability by dividing the count of samples between -1 and 1 by the total number of samples.

Now we need to calculate the count of samples between -1 and 1, which is marked as $Count$.

$$Count = \sum_{i=1}^{N} 1_B(-1 < \sqrt{\alpha}_i < 1)$$

Where 1_B is the indicator function, which is 1 if the condition is true and 0 otherwise.

Finally, we can calculate the posterior probability between -1 < $\sqrt{\alpha}$ < 1, by using formula of:

$$P(-1 < \sqrt{\alpha} < 1) \approx \frac{Count}{N}$$

Where N is the total number of samples and $Count$ is the amount of samples between -1 and 1. [1, p. 9], [5]

1.4 A parameter in a statistical model, θ, is given a prior $Beta(\alpha, \beta)$. After fitting, there are 1000 samples for the hyperparameters α, β. Using these samples, how can you generate samples from the population distribution of θ?

The population distribution of θ can be generated by using the samples of the hyperparameters of the Beta distribution. These hyperparameters are α and β, which are the parameters of the Beta distribution.

First we want to represent posterios samples α and β N times, where N is the number of the samples. Here we need to use the N value of 1000.

Then we want to generate sample θ_i from the each posterior samples of α and β.

This can be done by using the following formula:

$$\theta \sim Beta(\alpha_i, \beta_i)$$

Now we want to iterate this process for N times, which will give us N samples of θ from the population distribution. [1, p. 32-36], [6]

1.5 Consider the Stan program below. What is computed in the generated quantities block?

```
data {
  int<lower=0> N;
  vector[N] x;
  int<lower=0,upper=1> y[N];

  real x_pred;
}
parameters {
  real alpha;
  real beta;
}
transformed parameters {
  vector[N] theta = 1 ./ (1 + exp(-(alpha + beta*x)));
}
model {
  y ~ bernoulli(theta);
  alpha ~ normal(0, 1);
  beta ~ normal(0, 1);
}
generated quantities {
  real theta_pred = 1 / (1 + exp(-(alpha + beta*x_pred)));
}
```

If we look at Stan's guide, we can see that the quantities produced in Stan are used for the different calculations that take place after the sampling of parameters has already been completed.

In our code, the generated quantities block calculates the predicted probability of success (y = 1) for a new input value x_pred based on a logistic regression model. For this calculation, we use the alpha and beta posterior distributions of the parameters, resulting in theta_pred, which is the posterior distribution of the predictions of the input x_pred.

As Stan's guide says, we can use the quantities produced in the block:

- To generate predictions for new data
- To calculate posterior expectations and event probabilities
- To specify quantities for reporting or comparison that do not affect the model itself.

In a nutshell, this block allows us to calculate the prediction of posterior distribution for x_pred, leveraging the sampled values of alpha and beta. [1, p.141-145], [7]

1.6 Consider the Stan program below. If you try using it on some data, it will not work. How can you fix it without changing anything in the model block?

```
data{
  int<lower=0> N;
  vector[N] x;
  vector[N] y;
}
parameters{
  real a;
}

model{
  y ~ normal(x, a);
  a ~ normal(0, 1);
}
```

The probability calculation is happening in the model block, where we use the vector x and the standard deviation parameter "a". When calculating a normal distribution, the standard deviation parameter "a" must have a positive value. However, in this case we have specified that "a" is a real value without any restrictions, which means that it can have both negative and zero values, which can lead to error during sampling. Since this y-model is about computing a normal distribution, we need to ensure that the parameter a only takes positive values. [1, p.41-42]

To fix this, we just need to add a constraint on a in the parameter block to ensure that it is positive.

```
parameters {
    real<lower=0> a;
}
```

1.7 What is the point in running multiple MCMC chains when sampling from a probability distribution?

The purpose of multiple MCMC chain runs is to ensure that the posterior distribution is taken correctly, but also so that we can improve our convergence diagnostics, and reduce the risk of bias or incomplete exploration.

Multiple chain runs can be used to assess whether the MCMC process has converged to the target distribution. If all our chains start from different initial values and converge to the same region in the parameter space, it is an indication that these chains have explored the distribution appropriately.

We can also reduce sampling bias by running these chains multiple times, because different chains can explore different parts of the parameter space, increasing the diversity of the samples.

Running MCMC chains multiple times also provides more robustness to initialization, because multiple chains of different initial values ensure that the results are not dependent on the choice of initial values the sample itself has.

In summary, multiple chains improve the reliability of the results. There is no need to run an excessive number of chains, as this can be computationally expensive and lead to overfitting, but running a few chains is useful for robust inference. [1, p.275-284], [8]

1.8 When using the random walk Metropolis-Hastings algorithm, which generates proposals from a multivariate normal distribution, why does the acceptance probability reduce to the form $r = min(1, \frac{P(\theta^*)}{P(\theta^{i-1})})$?

Above, P is the target distribution, and θ^* , θ^{i-1} the proposal and current chain values, respectively.

The acceptance probability for the random walk Metropolis-Hasting (RWMH) algorithm reduces the given form because of the symmetry in the proposal distribution. In this algoririthm, the acceptance probability is derived to ensure detailed balance, a condition necessary for the Markov chain to converge to the target distribution $P(\theta)$

For the general case, the acceptance ratio is following:

$$r = (\frac{P(\theta^*|y)/J_t(\theta^*|\theta^{t-1})}{P(\theta^{t-1}|y)/J_t(\theta^{t-1}|\theta^*)})$$

where J_t is the proposal distribution.

The symmetric proposals, where $J_t(\theta^*)|(\theta^{i-1}) = J_t(\theta^{i-1})|(\theta^*)$, causes that the proposal terms cancels out, which can be simplified as:

$$r = min(1, \frac{P(\theta^*)}{P(\theta^{i-1})})$$

[1, p.279-280]

1.9 In Bayesian leave-one-out cross-validation, how is the effective number of parameters computed and what do the components of the formula represent?

The effective number of parameters are calculated by using formula of:

$$lppd_{loo-cv} = \sum_{i=1}^{N} \log p_{post}(y_i|\theta)$$

where:

n is the number of data points in the dataset.

$p_{post}(y_i|\theta)$ is the likelihood of the data point i, evaluated for parameter values of θ from the posterior distribution.

And lastly the whole $\log p_{post}(y_i|\theta)$ is the posterior variance of the likelihood of the data point i. [1, p.176-177]

1.10

No answer here.

2. Grid approximation: Exponential

Consider the following statistical model:

$$y \sim Exp(e^{a+bx})$$
$$a, b \sim N(0, 1),$$

where Exp is the exponential distribution.

```
# 2.1
# Generate data from this model.
# Use sample size N = 50, generate values for a and b from N(0,1).
# Generate the x-values from Uniform(0,2).

# Here we want to generate 50 sampels for and b variables by using normal distribution with mean 0 and
# Then we want to generate x-values from uniform distribution between 0 and 2.

# Generate data
set.seed(111) # I set the seed for make the results reproducible.
N <- 50
a <- rnorm(1, 0, 1) # Here we generate "a" from N(0,1).
b <- rnorm(1, 0, 1) # Same for "b" here.
x <- runif(N, 0, 2) # Here we generate x-values from uniform distribution between 0 and 2

# After the values are generated,
# we want to calculate the y-values for each x-values as the model suggests us.

# Now generate the y-values for each x-value.
y <- rexp(N, exp(a + b * x))

# Print the values to see the results
print(a)
```

```
## [1] 0.2352207
```

```
print(b)
```

```
## [1] -0.3307359
```

```
print(head(x))
```

```
## [1] 0.75532643 0.83667465 0.02131569 1.06459048 0.86432123 0.18736304
```

```
print(head(y))
```

```
## [1] 0.06792721 0.63697223 5.61296458 0.82054879 0.03832842 0.76584509
```

```
print(length(y))
```

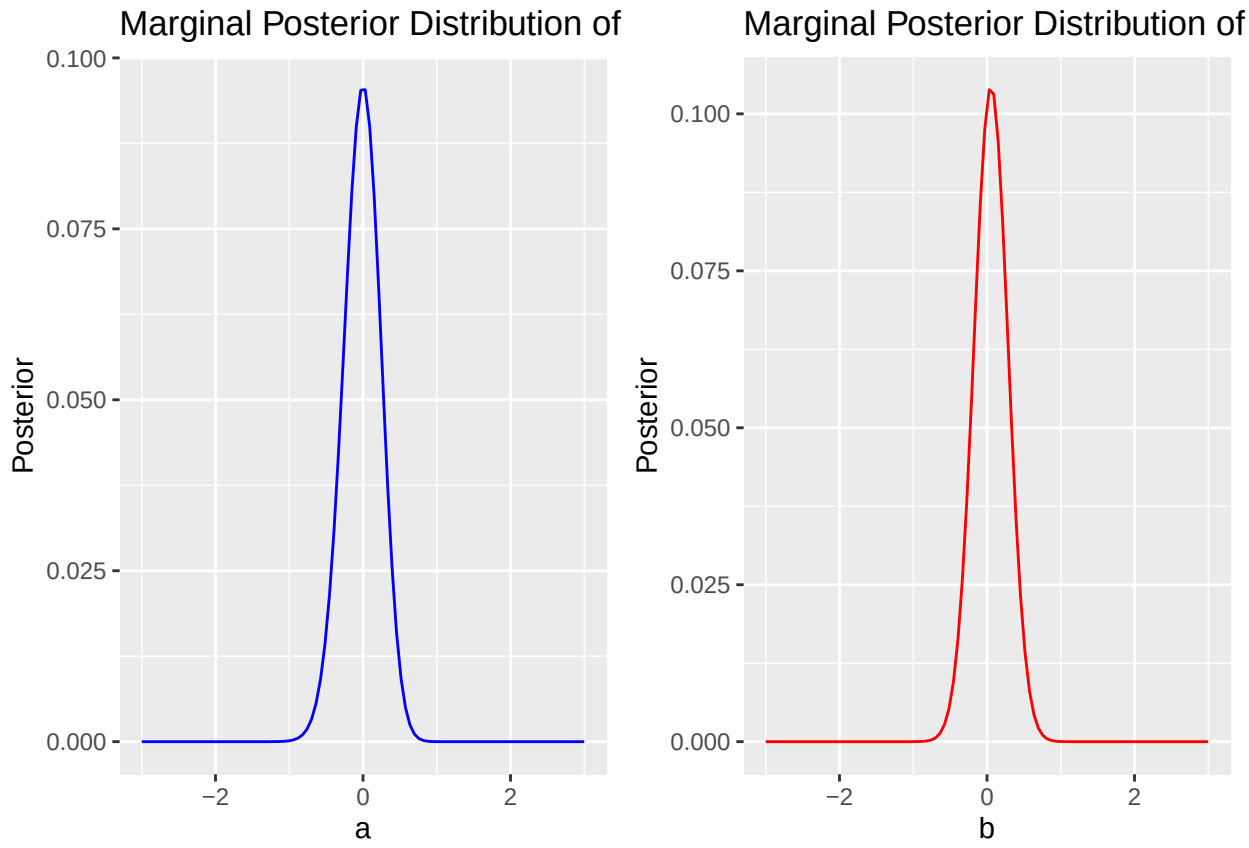
```
## [1] 50
```

```r
# 2.2
# Implement the grid approximation for this model.

# Here we want to do the grid approximation,
# which is used to approximate the posterior distribution of a and b.

# Define the grid
a_grid <- seq(-3, 3, length.out = 100) # Here we create a grid for "a"
b_grid <- seq(-3, 3, length.out = 100) # Here for b.

# Not let's calculate the likelihood for each a and b values. Here we want to create a matrix for all t
likelihood <- matrix(0, nrow = length(a_grid), ncol = length(b_grid))

for (i in 1:length(a_grid)) {
  for (j in 1:length(b_grid)) {
    likelihood[i, j] <- prod(dexp(y, exp(a_grid[i] + b_grid[j] * x)))
  }
}

# [1, p.6-8], [9]

# Because the priors for a and b are N(0,1),
# we can calculate the prior for a and b by using dnorm function where we use the values of a and b.

# Compute the prior for a and b
a_prior <- dnorm(a_grid, 0, 1)
b_prior <- dnorm(b_grid, 0, 1)
prior <- a_prior * b_prior

# Bayes theorem tells us that the p(a,b|y) = p(y|a,b)p(a,b) / p(y).
# Since the p(y) is constant, we can just drop it from the posterior calculation,
# where we can multiply the likelihood and prior.

# Compute the posterior
posterior <- likelihood * prior

# Normalize the posterior
posterior <- posterior / sum(posterior)

print(dim(posterior)) # We can see there is 100x100 matrix for the posterior values now.
```

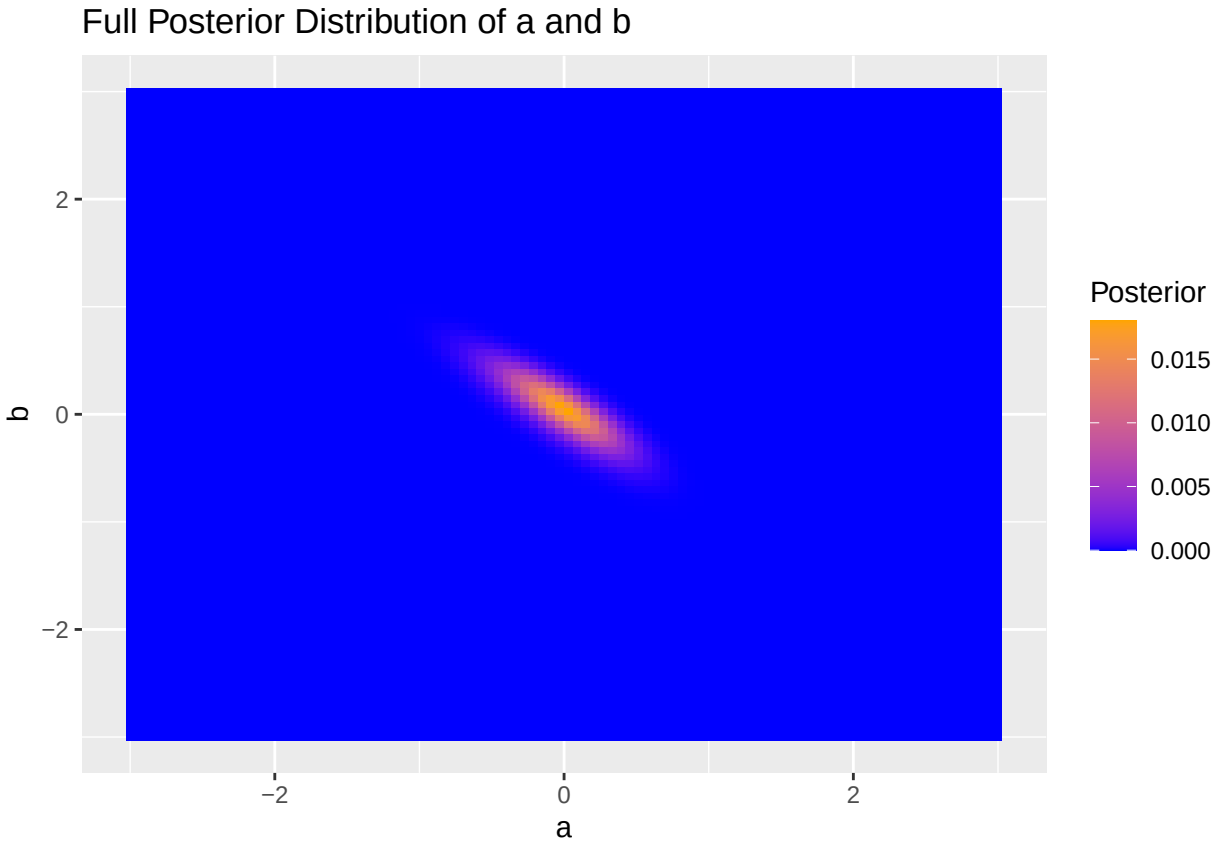
```
## [1] 100 100
```

```r
# [1, p.6-8]

# 2.3
# Plot the full posterior distribution.
# What information is there in the full posterior that is lost in the marginal posteriors of a and b?

# Do the marginal posteriors for a and b
marginal_a <- rowSums(posterior)
marginal_a <- marginal_a / sum(marginal_a)
```

```r
marginal_b <- colSums(posterior)
marginal_b <- marginal_b / sum(marginal_b)

# Plot the marginal posterior distribution of a
plot_a <- ggplot() +
  geom_line(aes(x = a_grid, y = marginal_a), color = "blue") +
  labs(title = "Marginal Posterior Distribution of a", x = "a", y = "Posterior")

# Plot the marginal posterior distribution of b
plot_b <- ggplot() +
  geom_line(aes(x = b_grid, y = marginal_b), color = "red") +
  labs(title = "Marginal Posterior Distribution of b", x = "b", y = "Posterior")

# Combine these two plots for clearer comparison
grid.arrange(plot_a, plot_b, ncol = 2) # [10]
```




```r
# Here we want to convert earlier calculated posterior matrix to a data frame.
posterior_df <- expand.grid(a = a_grid, b = b_grid)
posterior_df$posterior <- as.vector(posterior)

# Do the full posterior plotting now here
ggplot(posterior_df, aes(x = a, y = b, fill = posterior)) +
  geom_tile() +
  scale_fill_gradient(low = "blue", high = "orange") +
  labs(title = "Full Posterior Distribution of a and b", x = "a", y = "b", fill = "Posterior")
```




```
# [11] ggplot2

# When we take a look at the marginal posteriors of a and b,
# we can see that these plots only shows the individual distributions of a and b,
# but their dependencies are not shown at all.

# On the other hand, the full posterior distribution plot here shows their dependencies,
# which may be their correlation for one another.

# The heatmap also shows more richer and complete representation of the whole posterior distribution.

# 2.4
# Compute the marginal posterior mode of a.

# Now at the end, let's calculate the marginal posterior mode of a here.

# Here we can use the earlier calculated marginal_a vector to find the mode of a.
mode_a <- a_grid[which.max(marginal_a)]
sprintf("Marginal Postrior Mode of a is: %f", mode_a)
```

```
## [1] "Marginal Postrior Mode of a is: 0.030303"
```

This value looks correct to me if we compare it to the earlier plotted marginal posterior distribution of a. [1, p.311]

3. Stan: E.Coli

In microbial ecology, the evolution of a population's size is often described using logistic growth, governed by the differential equation:

$$\frac{dP}{dt} = rP(1 - \frac{P}{K})$$

Here, P represents the population size, t denotes time, and is r the growth rate. The carrying capacity K determines the maximum size of the population, preventing unbounded growth. The solution to this differential equation is

$$P(t) = \frac{K}{1 + (\frac{K}{P_0} - 1)e^{-rt}}$$

In other words, this solution provides the population size at time t, given the initial size P_0 and the model parameters.

The data set "ecoli3.txt" contains measurements of an E. coli bacterial strain culture over a 24-hour period. Assume the measurements P_meas are noisy and normally distributed: $P_{meas}(t) \sim N(P(t), \sigma^2)$. The initial population size is $P_0 = 0.005$.

3.1 Build a Stan program to estimate the parameters K,r and the measurement error σ^2. Use non-uniform priors.

The Stan program is created for models directory and named as "ecoli_model.stan".

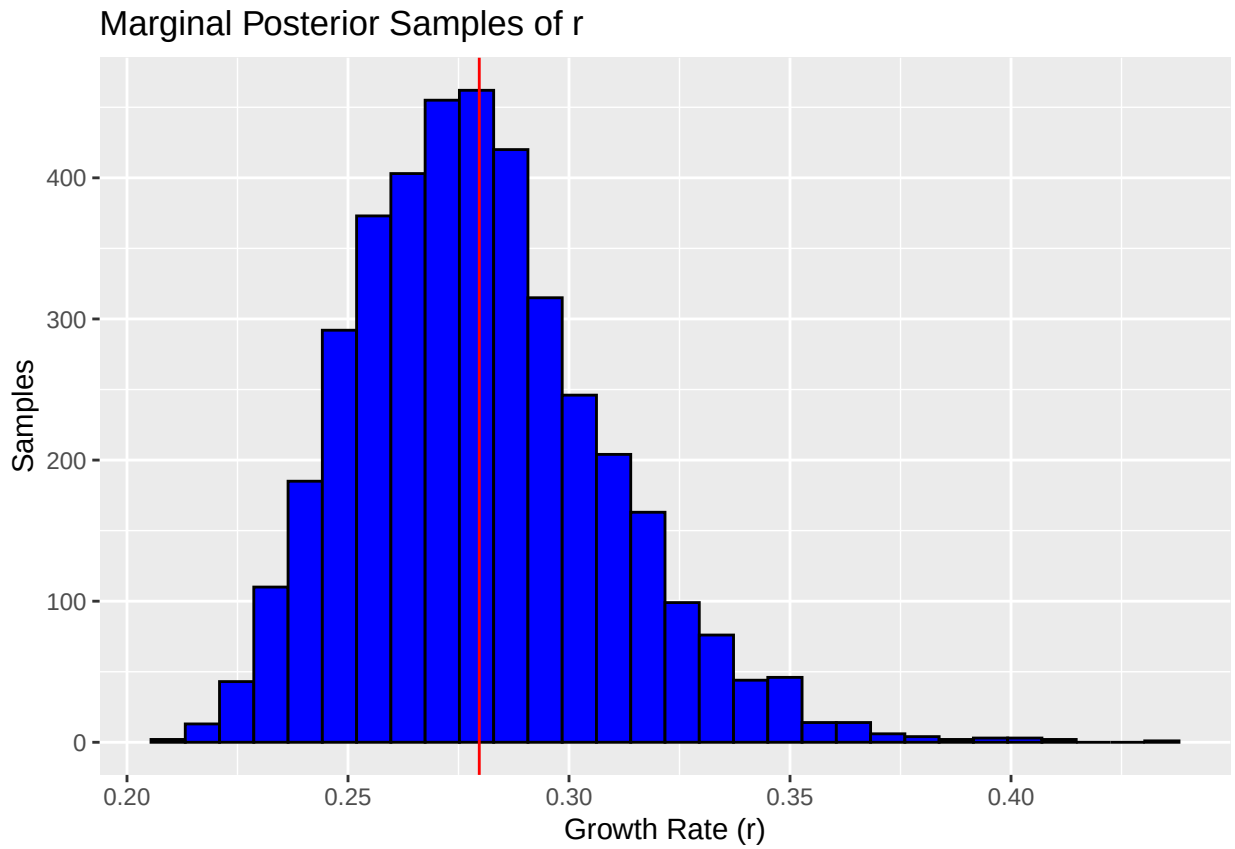
I write it here to show the structure of the model for the exercise.

```
data {
  int<lower=1> N; // Number of data points (1 or more points)
  array[N] real<lower=0> t; // Array of time points
  array[N] real<lower=0> P_meas; // Array of measured population sizes
  real<lower=0> P0; // Initial population size
}

parameters {
  real<lower=0> K; // Carrying capacity
  real<lower=0> r; // Growth rate
  real<lower=0> sigma; // Measurement error
}

model {
  array[N] real P_pred; // Array of predicted population sizes

  // Now define the priors for K, r and sigma
  K ~ normal(0.5, 0.25); // When we look at the dataset column E4,
                         // we have values somewhere between -0.25 and 0.5.
                         // But we are looking at the positive values here, why we use 0.5 and 0.25.

  r ~ normal(0.2, 0.1); // The growth rate per hour is somewhere between 0.1 and 0.3,
                        // so we use 0.2 and 0.1.
```

```
  sigma ~ exponential(1);

  // Growth model and likelihood
  for (n in 1:N) {
    P_pred[n] = K / (1 + (K / P0 - 1) * exp(-r * t[n]));
    P_meas[n] ~ normal(P_pred[n], sigma);
  }
}
```

[12]

```r
# 3.2 Plot histogram of the marginal posterior samples of r and include the marginal posterior mean of
# Import needed libraries
library(rstan)
library(ggplot2)

# Load the data
ecoli_data <- read.csv("data/ecoli3.txt")

# Change negative values to zero in the data, because the population size cannot be negative.
ecoli_data$E4[ecoli_data$E4 < 0] <- 0

# Define the data variables for Stan model
stan_data <- list(
  N = nrow(ecoli_data), # Number of data points
  t = ecoli_data$time, # Time points from data
  P_meas = ecoli_data$E4, # Measured population sizes
  P0 = 0.005 # Initial population size from the exercise description.
)

# Fit the data to our Stan model
fit <- stan(file = "models/ecoli_model.stan", data = stan_data, chains = 4, iter = 2000, seed = 111)

# Here we extract the samples from the fit, which will be used for the plotting.
posterior_samples <- extract(fit)
r_samples <- posterior_samples$r

# Plot the histogram and the vertical line for the mean value of r
ggplot(data.frame(r = r_samples), aes(x = r)) +
  geom_histogram(fill = "blue", color = "black", bins = 30) +
  geom_vline(xintercept = mean(r_samples), color = "red") +
  labs(title = "Marginal Posterior Samples of r", x = "Growth Rate (r)", y = "Samples")
```




```
# 3.3 Determine the posterior probability that r < 0.25.

# Now just calculate the posterior probability that r < 0.25.
r_prob <- mean(r_samples < 0.25)
sprintf("The posterior probability that r < 0.25 is: %f", r_prob)
```

```
## [1] "The posterior probability that r < 0.25 is: 0.141750"
```

[12], [13]

4. Stan: Indomethacin

Indomethacin is a nonsteroidal anti-inflammatory drug used e.g. to relieve symptoms of arthritis. The data in indometh.txt contains measurements of indomethacin concentration in blood samples in 6 subjects measured over 8 hours

Model the concentraction using exponential decay, defined as:

$$C(t) = C_0 e^{-\lambda t}$$

where $C(t)$ is the concentration at time t, C_0 is the unknown initial concentration (identical across all subjects), and $\lambda > 0$ is the rate of decay. Assume that the measurements contain some error, which is distributed as $N(0, \sigma^2)$.

Assume a hierarchical structure for the decay rate λ. In other words, estimate partially pooled λ estimates for each subject. Exclusively use gamma distributions as the (hyper)priors.

4.1 Implement a Stan program for the model. In the generated quantities block, generate the population distribution for λ.

I have created the Stan program again for this model to the models directory and named it as "indometh_model.stan".

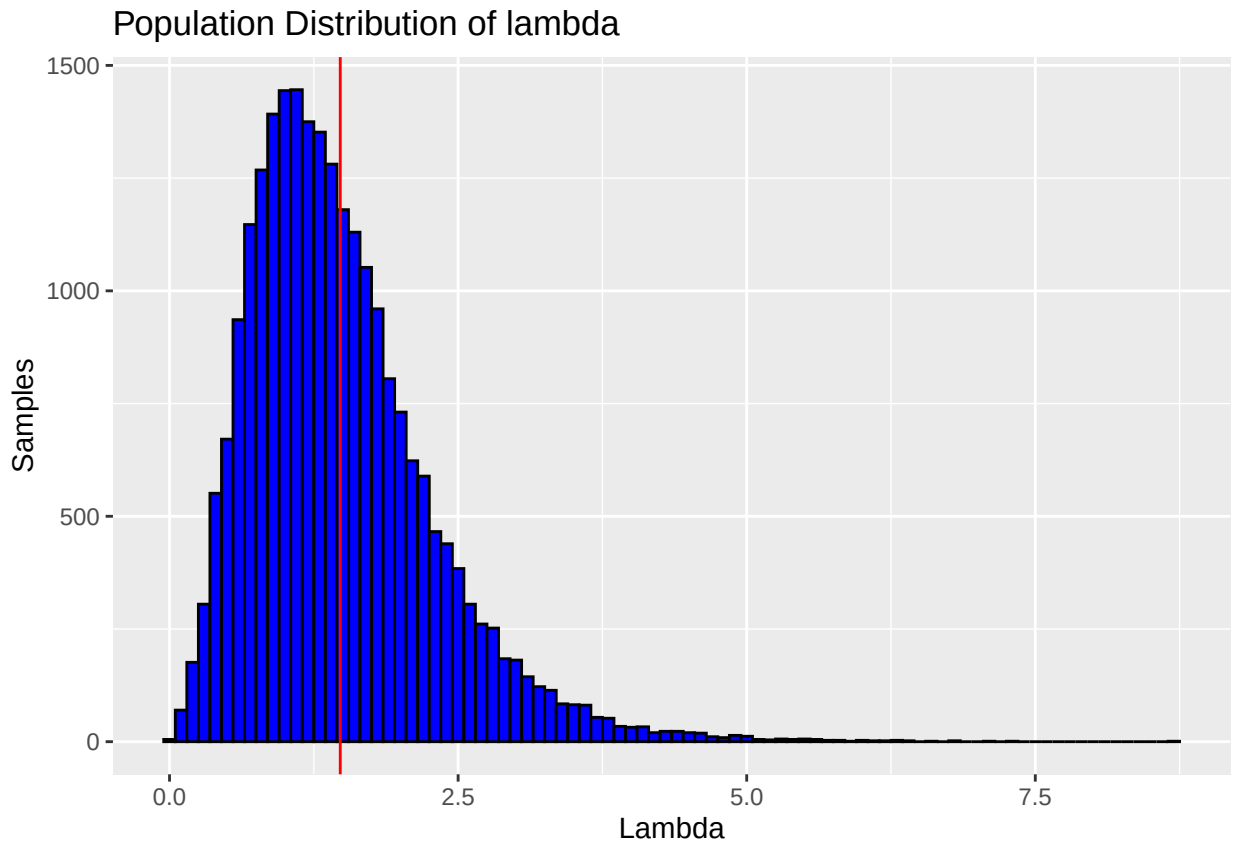
I write it here to show the structure of the model for the exercise.

```
data {
    int<lower=1> N; // Number of data points
    int<lower=0> J; // Number of subjects
    array[N] int<lower=1,upper=J> subject; // Array of subject indices
    vector[N] y; // Array of response values
    vector[N] t; // Array of time values
}

parameters {
    real<lower=0> C0; // Unknown initiual concentration
    vector<lower=0>[J] lambda; // Unknown decay rate for each subject
    real<lower=0> alpha; // Hyperparameter alpha
    real<lower=0> beta; // Hyperparameter beta
    real<lower=0> sigma; // Measurement error
}

model {
    alpha ~ gamma(3, 1); // Prior for alpha
    beta ~ gamma(3, 1); // Prior for beta
    lambda ~ gamma(alpha, beta); // Prior for lambda
    y ~ normal(C0 * exp(-lambda[subject] .* t), sigma); // Likelihood
}

generated quantities {
    vector[N] lambda_pop;
    for (n in 1:N) {
        lambda_pop[n] = gamma_rng(alpha, beta);
    }
}
```

4.2 Plot a histogram of the population distribution samples. Include marginal posterior means of the subject-specific λ_i in the same figure as vertical lines.

```r
# Load the data
indometh <- read.csv("data/indometh.txt")

# Define the variables from the data for Stan model
stan_data <- list(
  N = nrow(indometh), # Number of data points
  J = length(unique(indometh$Subject)), # Number of subjects
  subject = indometh$Subject, # Array of subject indices
  y = indometh$conc, # Vector of response values
  t = indometh$time # Vector of time values
)
```

```r
# Fit the model
fit <- stan(file = "models/indometh_model.stan", data = stan_data, chains = 4, iter = 2000, seed = 111)

# Do the extraction of the posterior samples for the plotting.
posterior <- extract(fit)
lambda_pop <- posterior$lambda_pop

# Convert the lambda_pop to a data frame
lambda_pop_df <- data.frame(lambda_pop = as.vector(lambda_pop))


# Plot the histogram with marginals
ggplot(lambda_pop_df, aes(x = lambda_pop)) +
  geom_histogram(binwidth = 0.1, fill = "blue", color = "black") +
  geom_vline(aes(xintercept = mean(lambda_pop)), color = "red") +
  labs(title = "Population Distribution of lambda", x = "Lambda", y = "Samples")
```

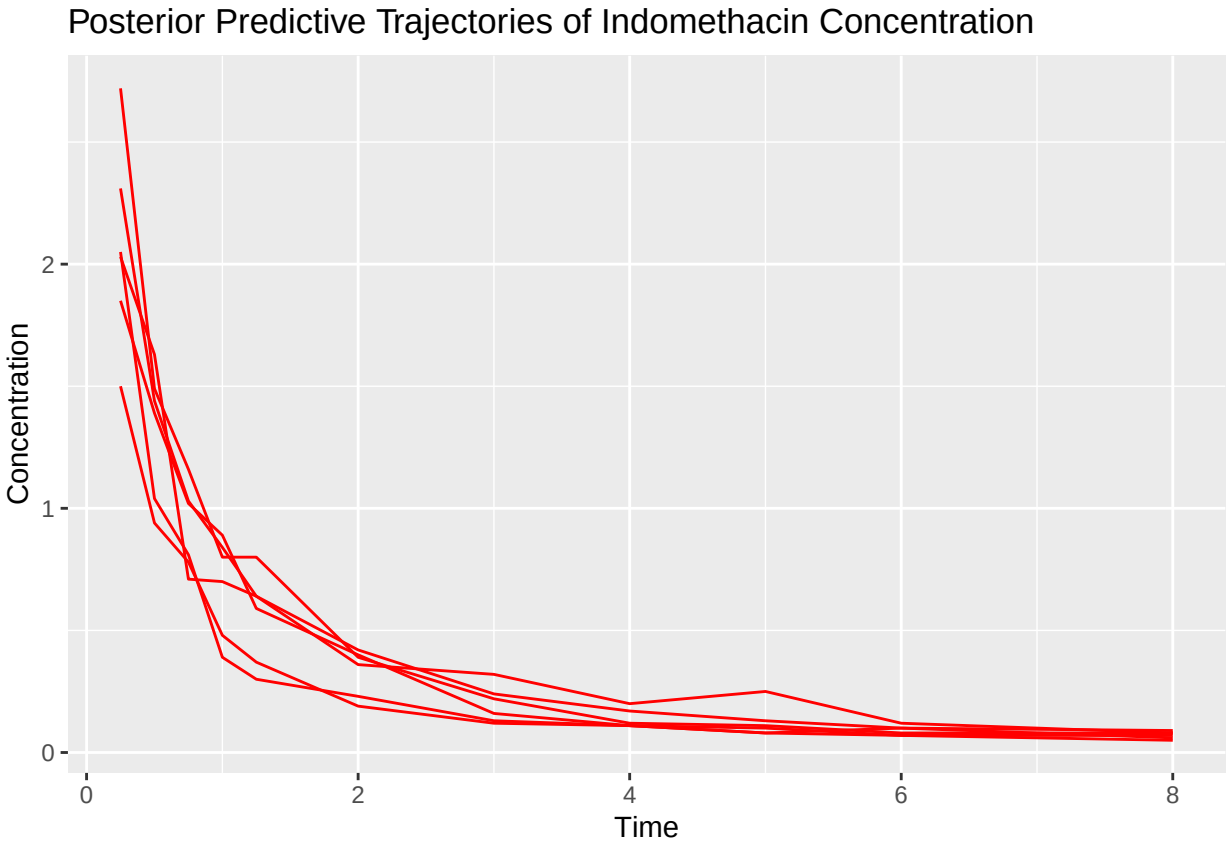


4.3 Assume a new subject enrolls in the study. Each sample from the posterior predictive distribution corresponds to a trajectory of Indomethacin concentration. Generate a plot of these trajectories.

```
# Extract the posterior samples for the prediction
alpha_samples <- posterior$alpha
beta_samples <- posterior$beta
C0_samples <- posterior$C0
sigma_samples <- posterior$sigma

# Simulate a new subject's decays rate
num_samples <- length(C0_samples)
print(num_samples)
```

```
## [1] 4000
```

```
new_lambda <- rgamma(num_samples, shape = alpha_samples, rate = beta_samples)

# Define a time range for the prediction
time_range <- seq(0, 8, by = 2)

# Generate the posterior predictive trajectories
set.seed(111)
predictions <- matrix(0, nrow = num_samples, ncol = length(time_range))
for(i in 1:num_samples) {
  predictions[i, ] <- C0_samples[i] * exp(-new_lambda[i] * time_range) + rnorm(length(time_range), mean
}

predictions_df <- data.frame(time = rep(time_range, num_samples), concentration = as.vector(predictions)

# Plot the trajectories
ggplot(predictions_df, aes(x = time, y = concentration, group = sample)) +
  geom_line(data = indometh, aes(x = time, y = conc, group = Subject), color = "red") +
  labs(title = "Posterior Predictive Trajectories of Indomethacin Concentration", x = "Time", y = "Conc
```



4.4 There is a time point t^* after which, at each $t > t^*$, the predicted concentration for the new subject is less than 0.1 with >90% probability. Find t^* to a precision of 0.01.

```
# Define the time range for the prediction
fine_time_range <- seq(0, 8, by = 0.01)

# Initialize an prorability vector
probabilities <- numeric(length = length(fine_time_range))

# Calculate the proportons of samples where concentration is less than 0.1
for (i in seq_along(fine_time_range)) {
  t <- fine_time_range[i]
  concentration <- C0_samples * exp(-new_lambda * t)
  probabilities[i] <- mean(concentration < 0.1)
}

# Find the first time point where the probability is greater than 0.9
t_star_index <- which(probabilities > 0.9)[1]
t_star <- fine_time_range[t_star_index]

# Print the result
sprintf("The time point t* is at: %f", t_star)
```

```
## [1] "The time point t* is at: 5.360000"
```

[1, p. 145-152], [7], [14] [15]

6. Bayes factors

One relative popular method for comparing competing models is by using Bayes' factors. The Bayes factor BF is defined as the ratio of marginal likelihoods for two models $M1$ and $M2$:

$$BF = \frac{p(X)_{M_1}}{p(X)_{M_2}} = \frac{\int p(X|\theta_1)p(\theta_1)d\theta_1}{\int p(X|\theta_2)p(\theta_2)d\theta_2},$$

where θ_1,$theta_2$ are the parameter in the models 1 and 2, respectively. The higher this ratio is above 1, the better the support for the suitability of model 1.

Using the data X provided below, compute the Bayes factor for the normal and Cauchy models:

$$X \sim N(\mu, \sigma^2)$$
$$\mu \sim N(0, 1)$$
$$\sigma \sim Gamma(2, 1)$$

and

$$X \sim Cauchy(x_0, \gamma)$$
$$x_0 \sim N(0, 1)$$
$$\gamma \sim Gamma(2, 1)$$

```
X=c(-0.48, -0.17, 0, 0.74, 0.9, -3.72, -0.39, -3.62,
-2.54, 2.17, -2.02, -0.15, 0.53, -1.1, 0.16, -0.91,
0.88, -0.03, 1.66, -12.49, -2.66, 0.53, -0.68, -3.8,
4.07, 3.49, -0.46, 3.71, -1.12, 0.06, 1.81, 3.84,
0.83, -0.23, 0.31, -0.21, -1.52, 0.44, 0.56, 0.32,
1.91, 0.93, -0.36, -0.57, -2.96, -6.35, 1.28, -1.13,
0.33, 0.8, 7.83, -1.78, -0.64, -2.6, 0.29, 3.13,
-1.36, 1.2, -8.12, 0.75, 0.21, 1.44, 0, 48.37, -0.81,
-5.03, 2.51, 0.04, -2.65, 0.37, -1.85, 6.98, 0.32,
-0.76, -0.34, -0.26, 0.54, 1.09, 1.17, 1.62, 3.7, 16.54,
-0.03, -0.79, 0.75, 0.33, -2.74, 1.08)
```

Refer to the table about the interpretation of the value of the Bayes' factor. Which is the superior model for the model and what is the strength of evidence for it?

How will I solve this exercise?

At the first, let's take a look at the model and its components. We have two models here, which are the normal and cauchy models. The normal model is defined as $X \sim N(\mu, \sigma^2)$, where we have priors $\mu \sim N(0, 1)$ and $\sigma \sim Gamma(2, 1)$

The Cauchy model is defined as $X \sim Cauchy(x_0, \gamma)$, where we have priors $x_0 \sim N(0, 1)$ and $\gamma \sim Gamma(2, 1)$.

For these models, we want to define their likelihood functions and priors. I will create separate Stan programs for these models.

Once the likelihood functions are calculated, I fit the data to the models, from which I extract the logarithm of the likelihoods of both models. I then calculate the marginal likelihoods for each model using the samples we create.

Finally, we calculate the Bayes factors by using the formula given in the exercise.

Let's start with the models

Normal model

```
data {
    int<lower=1> N; // Number of datapoints
    array[N] real X; // Data
}

parameters {
    real mu; // Mean value of the normal distribution.
    real<lower=0> sigma; // Standard deviation of the normal distribution. We want to make sure that th
}

model {
    // Define priors
    mu ~ normal(0, 1); // Prior for the mean value of the normal distribution.
    sigma ~ gamma(2,1); // Prior for the standard deviation of the normal distribution.

    X ~ normal(mu, sigma); // Likelihood
}

generated quantities {
    real log_likelihood = 0; // Log likelihood
    for (n in 1:N) {
      log_likelihood += normal_lpdf(X[n] | mu, sigma); // Calculate the log likelihood
    }
}
```

Cauchy model

```
data {
    int <lower=1> N; // Number of datapoints
    array [N] real X; // Data
}

parameters {
```

```
    real x0; // Location parameter of the Cauchy distribution.
    real<lower=0> gamma; // Scale parameter of the cauchy distribution. Let's make sure this is also pos
}

model {
    // Define priors
    x0 ~ normal(0,1); // Prior for the location parameter of the cauchy distribution.
    gamma ~ gamma(2,2); // Prior for the scale parameter of the cauchy distribution.

    X ~ cauchy(x0, gamma); // Likelihood
}

generated quantities {
    real log_likelihood = 0; // Log likelihood
    for(n in 1:N) {
        log_likelihood += cauchy_lpdf(X[n] | x0, gamma); // Calculate the log likelihood
    }
}
```

[16]

Now we have created our models, so lets move forward.

Now we want to fit the both models with the data and extract the samples from the models. Then we have the samples, which we can use to calculate the margianal likelihoods for each model by using Monte Carlo Approximation.

Finally, we can compute the Bayes factor by using the marginal likelihoods.

Fit the models and calculate the Bayes factor

```r
# Create the data list for the models
stan_data <- list(
  N = length(X), # Number of data points
  X = X # List of datapoints
)

# Now we fit the model by using the data
normal_fit <- stan(file = "models/bayes_normal.stan", data = stan_data, chains = 4, iter = 2000, seed =
cauchy_fit <- stan(file = "models/bayes_cauchy.stan", data = stan_data, chains = 4, iter = 2000, seed =

# Extract the samples from the both models.
log_lik_normal = extract(normal_fit)$log_lik
log_lik_cauchy = extract(cauchy_fit)$log_lik

# Now lets use these samples to calculate the marginal likelihoods for each model.
marginal_likelihood_normal = log(mean(exp(log_lik_normal)))
marginal_likelihood_cauchy = log(mean(exp(log_lik_cauchy)))

# Now we can calculate the Bayes factor by using the marginal likelihoods.
bayes_factor = marginal_likelihood_normal / marginal_likelihood_cauchy

# Print the results
sprintf("Bayes Factor: %f", bayes_factor)
```

```
## [1] "Bayes Factor: 1.354313"
```

[1, p.182-184], [7]

Which model is superior?

We got the bayes factor as 1.354..., which means that the normal model is 1.35 times more likely to explain the data than the cauchy model.

The magnitudes of bayes factors suggests minimal evidence for the normal model, because the table shows following:

- $1 < \text{BF} < 3.2$: Not worth more than a bare mention
- $3.2 < \text{BF} < 10$: Substantial
- $10 < \text{BF} < 100$: Strong
- $100 < \text{BF}$: Decisive

Even if we can say the normal model is slightly superior here, the BF value of 1.35 is not overwhelmingly strong, and table of Bayes factor values states it is not worth more than mention here. [17]

References:

[1] Gelman and others: Bayesian Data Analysis (3rd edition)

[3] Informative Priors and Bayesian Computation, https://www.sfu.ca/~sgolchi/DSAA_unblinded.pdf

[4] https://www.statlect.com/fundamentals-of-statistics/uninformative-prior

[5] https://www.statlect.com/glossary/posterior-probability

[6] https://www.statlect.com/probability-distributions/beta-distribution

[7] https://mc-stan.org/docs/reference-manual/blocks.html#program-block-generated-quantities

[8] https://hschuett.github.io/BayesForAccountingResearch/mcmc.html

[9] https://stats.stackexchange.com/questions/609238/what-is-the-grid-in-bayesian-grid-approximations

[10] https://cran.r-project.org/web/packages/gridExtra/vignettes/arrangeGrob.html

[11] https://ggplot2-book.org/

[12] Gangan Manasi S. and Athale Chaitanya A. 2017, Threshold effect of growth rate on population variability of Escherichia coli cell length, sR. Soc. Open Sci.4160417, http://doi.org/10.1098/rsos.160417

[13] https://mc-stan.org/docs/reference-manual/execution.html

[14] https://en.wikipedia.org/wiki/Indometacin

[15] https://www.sciencedirect.com/topics/mathematics/marginal-posterior

[16] https://mc-stan.org/docs/stan-users-guide/proportionality-constants.html

[17] https://en.wikipedia.org/wiki/Bayes_factor#Interpretation