Overview

This assignment works through the details estimating an unknown mean, μ, and unknown noise, σ, of a Gaussian likelihood. You will practice visualizing the log-posterior, work through the mathematics of the estimation process, and ultimately use the Laplace Approximation to approximate the joint posterior distribution on μ and σ given observations. This assignment include programming and derivations.

IMPORTANT: The RMarkdown assumes you have downloaded the data set (CSV file) to the same directory you saved the template Rmarkdown file. If you do not have the CSV file in the correct location, the data will not be loaded correctly.

IMPORTANT!!!

Certain code chunks are created for you. Each code chunk has `eval=FALSE` set in the chunk options. You **MUST** change it to be `eval=TRUE` in order for the code chunks to be evaluated when rendering the document.

You are free to add more code chunks if you would like.

Load packages

You will use the `tidyverse` in this assignment, as you have done in the previous assignments.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

Problem 01

A large toy company recently completed a "digital transformation" and now collects, tracks, and records data from all areas involved in the production of their top selling toys. The company is interested in understanding the behavior of the plastic used in several toy lines and asks you to examine the data. After a few meetings with the company, you find out the Key Performance Indicator (KPI) they are interested in requires destructive tests. Thus, toys must be willingly destroyed in order to record the value of interest.

Destructive tests are expensive and tedious to perform and so only a small number of entries are available in the newly commissioned data warehouse that the company uses to store a majority of their data. You query the appropriate data tables in the data warehouse and return the following data set.

```
hw06_data_path <- "hw06_data.csv"

hw06_df <- readr::read_csv(hw06_data_path, col_names = TRUE)
```

```
## Rows: 8 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (2): obs_id, x
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

As you can see from the `glimpse()` below, `hw06_df` contains two columns. The `obs_id` column which is an observation index, and `x`, the performance metric of interest. There are just 8 observations to work with!

```
hw06_df %>% glimpse()
```

```
## Rows: 8
## Columns: 2
## $ obs_id <dbl> 1, 2, 3, 4, 5, 6, 7, 8
## $ x      <dbl> 8.233075, 13.262098, 13.631564, 13.710720, 8.078146, 6.887018, ~
```

It is believed that a Gaussian likelihood is appropriate for this performance metric. You feel it is appropriate to assume that the observations are conditionally independent given an unknown constant mean, μ, and unknown likelihood noise, σ. With the n-th observation denoted as x_n, the joint likelihood can be factored into the product of N likelihoods:

$$p(\mathbf{x} \mid \mu, \sigma) = \prod_{n=1}^{N} \left(\text{normal}\left(x_n \mid \mu, \sigma\right)\right)$$

Your goal is to infer the unknown mean of the performance metric, μ, as well as the unknown noise, σ, using the 8 measurements, $\mathbf{x}$.

1a)

Start out by calculating a few summary statistics about the measurements.

Calculate the sample average, the sample standard deviation, the min and max values of the `x` variable in the `hw06_df` data set.

```
avrg_x <- mean(hw06_df$x)
std_dev_x <- sd(hw06_df$x)
min_x <- min(hw06_df$x)
max_x <- max(hw06_df$x)
```

```
avrg_x
```

SOLUTION

```
## [1] 10.21831
```

```
std_dev_x
```

```
## [1] 3.204731
```

```
min_x
```

```
## [1] 6.138696
```

```
max_x
```

```
## [1] 13.71072
```

1b)

With such a small data set you decide to ask several Subject Matter Experts (SMEs) from the toy company their opinions about the performance metric. You find out the they have worked with this particular plastic for quite some time. However, while going through the "digital transformation" they also recently installed several new components to the machines that produce the plastic material. They are still getting used to working with the new equipment and software, but feel confident about the behavior of their material.

After a few more meetings, the SMEs believe a Gaussian prior on the unknown mean is appropriate. The prior distribution on the unknown mean, μ, will have prior mean, μ_0, and prior standard deviation, τ_0. The prior on μ is therefore:

$$\mu \mid \mu_0, \tau_0 \sim \text{normal}\left(\mu \mid \mu_0, \tau_0\right)$$

The SMEs feel there is approximately 95% probability the mean would be between values of 10 and 12. They believe that interval is a middle 95% prior uncertainty interval, and so the prior median is between 10 and 12.

Determine the values for the prior parameters, μ_0 and τ_0, based on the information provided by the SMEs.

SOLUTION We have the following equations:

$$\mu_0 + 2\tau_0 = 12$$
$$\mu_0 - 2\tau_0 = 10$$

When we solve the above system, we get $\mu_0 = 11$ and $\tau_0 = 0.5$.

1c)

You decide to treat the joint prior on μ and σ as independent, $p\left(\mu, \sigma\right) = p\left(\mu\right) \times p\left(\sigma\right)$. The prior on the noise is assumed to be an Exponential distribution with a prior rate of 0.5, $\lambda = 0.5$.

The un-normalized posterior on the two unknowns, μ and σ, is therefore:

$$p\left(\mu, \sigma \mid \mathbf{x}\right) \propto \prod_{n=1}^{N} \left(\text{normal}\left(x_n \mid \mu, \sigma\right)\right) \times \text{normal}\left(\mu \mid \mu_0, \tau_0\right) \times \text{Exp}\left(\sigma \mid \lambda = 0.5\right)$$

You can visualize the log-posterior surface to understand the joint posterior distribution on the unknowns, since there are only 2 unknowns. You must define a function which calculates the log-posterior at specific values of the unknown parameters. As you can see from the un-normalized posterior expression above, other pieces of information are required to calculate the log-posterior. The observations and prior parameters must also be provided to the same function as the unknown parameters.

Thus, before defining the log-posterior function, you must create an `R` list which stores the measurements, the prior parmaeterse associated with the prior on μ, μ_0 and τ_0, and the parameter associated with the prior on σ, λ.

The list of required information is started for you below. You must complete the code chunk below by assigning the correct values to each of the named elements in the list. The names of the variables and the comments specify what you should fill in.

```r
hw06_info <- list(
  xobs = hw06_df$x,### the meausrements
  mu_0 = 11,### mu_0 value
  tau_0 = 0.5,### tau_0 value
  sigma_rate = 0.5 ### rate (lambda) on sigma
)
```

SOLUTION

1d)

You must define a function which calculates the log-posterior on the unknown mean, μ, and unknown noise, σ. The `my_logpost()` function is started for you in the code chunk below. The first argument, `unknowns`, is a **vector** containing the unknown parameters we wish to learn. The second argument, `my_info`, is a list of required information. The unknowns are extracted from the `unknowns` vector for you with the unknown mean assigned to the `lik_mu` variable and the unknown noise assigned to the `lik_sigma` variable.

The `my_info` second argument is a generic name, but you will assume it is a list containing the fields (variables) in the `hw06_info` list you defined in the previous problem. You may use the `$` operator whenever you want to access a piece of information from the `my_info` list in the `my_logpost()` function. For example, to access the vector of observations within the `my_logpost()` function you should type `my_info$xobs`.

Complete the `my_logpost()` function. The variable names and comments describe what you are required to complete.

You ARE allowed to use built in `R` density functions in this problem.

Several test values for the `unknowns` input vector are provided for you to try out below.

```r
my_logpost <- function(unknowns, my_info)
{
  # unpack the unknowns into separate variables
  lik_mu <- unknowns[1]
  lik_sigma <- unknowns[2]

  # calculate the log-likelihood
  log_lik <- sum(dnorm(x=my_info$xobs,
                       mean=lik_mu,
                       sd=lik_sigma,
                       log=TRUE))

  # calculate the log-prior on mu
  log_prior_mu <- dnorm(x=lik_mu,
                        mean=my_info$mu_0,
                        sd=my_info$tau_0,
                        log=TRUE)

  # calculate the log-prior on sigma
  log_prior_sigma <- -my_info$sigma_rate * lik_sigma + log(my_info$sigma_rate)

  # return the (un-normalized) log-posterior
  log_lik+log_prior_mu+log_prior_sigma

}
```

SOLUTION Test out the function to check that it works as expected. Try a value of 13 for μ and a value of 5 for σ. If you programmed the `my_logpost()` function correctly, you should get a value of -34.32184 printed to the screen.

```r
unknowns <- c(13,5)
my_logpost(unknowns, hw06_info)
```

```
## [1] -34.32184
```

Test out the function to check that it works as expected. Try a value of 7 for μ and a value of 1.5 for σ. If you programmed the `my_logpost()` function correctly, you should get a value of -78.65353 printed to the screen.

```r
unknowns <- c(7,1.5)
my_logpost(unknowns, hw06_info)
```

```
## [1] -78.65353
```

1e)

You must define a grid of parameter values that will be applied to the `my_logpost()` function, in order to visualize the log-posterior surface. A simple way to create a **full-factorial** grid of combinations is with the `expand.grid()` function. The basic syntax of `expand.grid()` is shown in the example code chunk below for two variable `x1` and `x2`. The `x1` variable is a vector of just two values, `c(1, 2)`, and the variable `x2` is a vector of 3 values, `1:3`. As shown in the code chunk output printed to the screen, the `expand.grid()` function produces 6 combinations of these two variables. The variables are stored as columns. Their combinations correspond to a row within the generated object. The `expand.grid()` function takes care of the "book keeping" for us, to allow varying `x2` for all values of `x1`.

```r
expand.grid(x1 = c(1, 2),
            x2 = 1:3,
            # extra arguments I like to set
            KEEP.OUT.ATTRS = FALSE,
            stringsAsFactors = FALSE) %>%
  # convert to a tibble!
  as.data.frame() %>% tibble::as_tibble()
```

```
## # A tibble: 6 x 2
##      x1    x2
##   <dbl> <int>
## 1     1     1
## 2     2     1
## 3     1     2
## 4     2     2
## 5     1     3
## 6     2     3
```

You will use the `expand.grid()` function to create a grid of combinations of `mu` and `sigma`. You should create your `mu` and `sigma` variables in `expand.grid()` with the `seq()` function. The `from` (lower bound) and the `to` (upper bound) arguments that you should follow are:

- The lower bound on `mu` should equal 3 prior standard deviations away from the prior mean.
- The upper bound on `mu` should equal 3 prior standard deviations above the prior mean.
- The lower bound on `sigma` should equal 1.
- The upper bound on `sigma` should equal the 0.99 **prior** Quantile (99th **prior** percentile).

Complete the two code chunks below. In the first code chunk, define the lower and upper bounds on `mu` and `sigma` following the bulleted instructions. Use those bounds to create the grid of parameter combinations in the second code chunk below. Set the `length.out` argument in the `seq()` function to be 251 for both the `mu` and `sigma` variables.

SOLUTION Define the bounds on the two parameters:

```r
mu_grid_lwr <- avrg_x - 3 * std_dev_x
mu_grid_upr <- avrg_x + 3 * std_dev_x
```

```r
sigma_grid_lwr <- 1
sigma_grid_upr <- 3*(std_dev_x)
```

Define the grid of parameter combinations.

```r
param_grid <- expand.grid(mu = seq(mu_grid_lwr, mu_grid_upr, length.out = 251),
                          sigma = seq(sigma_grid_lwr, sigma_grid_upr, length.out = 251),
                          KEEP.OUT.ATTRS = FALSE, stringsAsFactors = FALSE) %>%
  as.data.frame() %>% tibble::as_tibble()
```

1f)

The `my_logpost()` function accepts a vector as the first input argument, `unknowns`. Thus, you cannot simply pass in the columns of the `param_grid` tibble into `my_logpost()`! To overcome this, you will define a "wrapper" function, which manages the call to the log-posterior function. The wrapper, `eval_logpost()`, is started for you in the first code chunk below. The arguments to `eval_logpost()` are setup to be rather general. The first and second arguments, `unknown_1` and `unknown_2`, are the first and second elements in the `unknowns` input vector to the `my_logpost()` function. In the current context, the first argument is `mu` and the second argument is `sigma`. The third argument is intended to be a function handle for a log-posterior function, thus `logpost_func` represents the `my_logpost` function. The fourth argument represents the required information to call that log-posterior function.

This problem tests that you understand how to call a function, and how to input the arguments to that function.

Complete the code chunk below, such that the user supplied `logpost_func` function is called. The `unknown_1` and `unknown_2` arguments must be combined together as the first argument to `logpost_func()`. Set the `logpost_info` variable as the second argument to `logpost_func()`.

HINT: If you are confused by this setup, think through how you called the `my_logpost()` function to test that it worked properly in Problem 1d).

Check that you setup `eval_logpost()` correctly by using the same first test in Problem 1d). Try a value of 13 for μ and a value of 5 for σ.

```r
eval_logpost <- function(unknown_1, unknown_2, logpost_func, logpost_info)
{
  unknowns <- c(unknown_1, unknown_2)
  ev_logpost_result <- logpost_func(unknowns, logpost_info)

  return(ev_logpost_result)
}
```

SOLUTION Test out `eval_logpost()`. You should get the same as result that you did in Problem 1d). Remember the third argument to `eval_logpost()` is the log-posterior function we want to call.

```r
eval_logpost(13,5,my_logpost,hw06_info)
```

```
## [1] -34.32184
```

```r
eval_logpost(7,1.5,my_logpost,hw06_info)
```

```
## [1] -78.65353
```

1g)

The code chunk below uses the `purrr::map2_dfr()` function to apply the `eval_logpost()` function to all combinations of `mu` and `sigma` within `param_grid`. Be sure to set the `eval` flag to `TRUE` after you run the code chunk, because by default `eval=FALSE`. The result is assigned to the variable `log_post_result`. You can check the RStudio Environment Panel to see that the length of `log_post_result` is equal to the number of rows in `param_grid`.

```
log_post_result <- purrr::map2_dbl(param_grid$mu, param_grid$sigma,
                                   eval_logpost,
                                   logpost_func = my_logpost,
                                   logpost_info = hw06_info)
```

The code chunk below visualizes the log-posterior surface for you. The log-posterior surface contours are plotted in the same style presented in lecture. You are required to interpret the log-posterior surface, and include the sample average and sample standard deviation with a `geom_point()` geom object. The sample average and sample standard deviation will be displayed as an orange square marker within the figure. You will discuss how the posterior mode compares to these estimates.

The code chunk below is almost complete. You must assign the sample average to the `xbar` variable and the sample standard deviation to the `xsd` variable in the `tibble` assigned as the `data` argument to the `geom_point()` geom. See the comments below for where you should make the changes.
You must describe how the sample average and standard deviation compare to the posterior mode. Are they similar? What can you say about the posterior uncertainty in μ and σ based on the visualization?

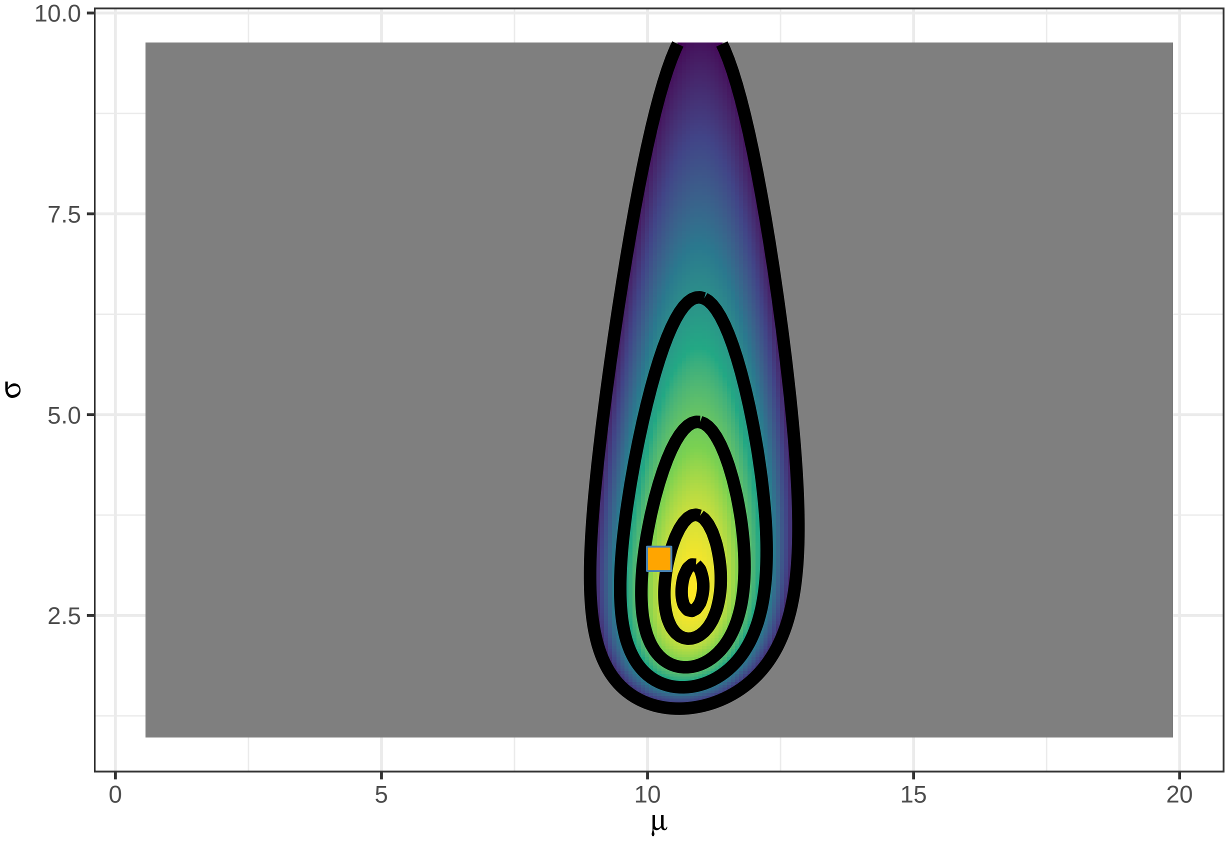
HINT: If you want to see what the log-posterior surface looks like before adding in the sample average and sample standard deviation point, just comment out all lines associated with the `geom_point()` call below.

SOLUTION What do you think?

```
param_grid %>%
  mutate(log_post = log_post_result,
         log_post_2 = log_post - max(log_post)) %>%
  ggplot(mapping = aes(x = mu, y = sigma)) +
  geom_raster(mapping = aes(fill = log_post_2)) +
  stat_contour(mapping = aes(z = log_post_2),
               breaks = log(c(0.01/100, 0.01, 0.1, 0.5, 0.9)),
               size = 2.2,
               color = "black") +
  # include the sample average (xbar) and the sample standard deviation (xsd)
  geom_point(data = tibble::tibble(xbar = avrg_x, xsd = std_dev_x),
             mapping = aes(x = xbar, y = xsd),
             shape = 22,
             size = 4.5, fill = "orange", color = "steelblue") +
  scale_fill_viridis_c(guide = FALSE, option = "viridis",
                       limits = log(c(0.01/100, 1.0))) +
  labs(x = expression(mu), y = expression(sigma)) +
  theme_bw()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: The `guide` argument in `scale_*()` cannot be `FALSE`. This was deprecated in
## ggplot2 3.3.4.
## i Please use "none" instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



1) When we look at the figure above, we can observe that the sample average values are looking symmeric (w.r.t. the vertical line that passes through log posterior mode), and the posterior uncertainty in the sample average has length close to 5 units.

2) When we look at the figure, we observe that the standard deviation values are non-symmetric (w.r.t. the horizontal line that passes through log posterior mode), and posterior uncertainty in standard deviation is larger (about 9 units). We would expect this to be smaller if the sample size N was larger.

Problem 02

We discussed in lecture how the visualization approach is useful, but is limited to just 1 or 2 unknowns. It does not scale well to more unknowns. We discussed that the Laplace or Normal Approximation allows us to approximate a distribution with a Multivariate Normal (MVN) distribution. The Laplace Approximation is convenient and useful for performing Bayesian inference in a wide variety of problems. You will ultimately perform the Laplace Approximation on the problem described in Problem 01.

The Laplace Approximation consists of three main steps. The first step finds the posterior mode via optimization, the second step evaluates the Hessian matrix at the posterior mode, and the third step calculates the approximate covariance matrix from the Hessian. You practiced the first step, finding the posterior mode, in the previous assignment with the one-parameter normal-normal model. Let's complete the Laplace Approximation for the one-parameter problem before executing the Laplace Approximation for the two parameter case. This gives you experience with each of the steps in the Laplace Approximation in a simplified setting, before applying the approximation to the more challenging two unknowns problem.

You will assume that the likelihood noise is equal to 3, $\sigma = 3$. All observations are still considered to be conditionally independent given the μ and σ parameters. The prior on the unknown mean is still a Gaussian with hyperparmeters μ_0 and τ_0. The un-normalized posterior on the unknown mean given N observations, $\mathbf{x}$, and likelihood noise, σ, is:

$$p\left(\mu \mid \mathbf{x}, \sigma\right) \propto \prod_{n=1}^{N} \left(\text{normal}\left(x_n \mid \mu, \sigma\right)\right) \times \text{normal}\left(\mu \mid \mu_0, \tau_0\right)$$

2a)

You wrote out the un-normalized log-posterior on μ, determined the first derivative with respect to μ, and derived the posterior mode (the MAP) in the previous assignment. Thus, you already performed the first step of the Laplace Approximation! You will work through the details of the second and third steps, starting with calculating the second derivative of the log-posterior with respect to the unknown mean, μ.

Determine the second derivative of the log-posterior with respect to the unknown mean, μ. Your solution should show at least several steps. You may reference your solution from the previous assignment, but you must write down the expression you are using as your starting point.

SOLUTION Firstly, we obtained the unnormalized log posterior on μ as:

$$\log\left[p(\mu \mid \mathbf{x}, \sigma)\right] \propto -\frac{1}{2\sigma^2}\sum_{n=1}^{N}\left\{(x_n-\mu)^2\right\} - \frac{1}{2\tau_0^2}(\mu-\mu_0)^2$$

Then, in HW5, we obtained its first derivatiove as:

$$\frac{d}{d\mu}\left\{-\frac{1}{2\sigma^2}\sum_{n=1}^{N}\left\{(x_n-\mu)^2\right\} - \frac{1}{2\tau_0^2}(\mu-\mu_0)^2\right\} = \frac{1}{\sigma^2}\sum_{n=1}^{N}(x_n-\mu) - \frac{1}{\tau_0^2}(\mu-\mu_0) = \frac{1}{\sigma^2}\left(\sum_{n=1}^{N}(x_n) - \sum_{n=1}^{N}(\mu)\right) - \frac{1}{\tau_0^2}(\mu-\mu_0)$$

$$= \frac{1}{\sigma^2}\left(\bar{x}N - N\mu\right) - \frac{1}{\tau_0^2}(\mu-\mu_0)$$

$$= \mu\left(-\frac{N}{\sigma^2} - \frac{1}{\tau_0^2}\right) + \frac{N}{\sigma^2}\bar{x} + \frac{\mu_0}{\tau_0^2}$$

Using the above expression, the second derivative of the log-posterior with respect to the unknown mean μ can be obtained as:

$$\frac{d^2}{d\mu^2}\left\{-\frac{1}{2\sigma^2}\sum_{n=1}^{N}\left\{(x_n-\mu)^2\right\} - \frac{1}{2\tau_0^2}(\mu-\mu_0)^2\right\} = -\frac{N}{\sigma^2} - \frac{1}{\tau_0^2}$$

2b)

You determined the expression for the posterior mode in Problem 2d) of Homework 04.

How can you confirm that the mode does in fact correspond to the μ value associated with the maximum log-posterior density and not the minimum log-posterior density?

SOLUTION We obtained that the 2nd derivative equals $-\frac{N}{\sigma^2} - \frac{1}{\tau_0^2}$. It is negative for all values of μ. So, it is negative at the critical point where the 1st derivative was 0. From Calculus, we know that if the 2nd derivative evaluated at a critical point gives a negative value, then the critical point is a maximum point of the function (this is called the 2nd derivative test). Thus, the result follows from the 2nd derivative test.

2c)

In this one parameter application, the Laplace Approximation approximates the posterior distribution as an univariate Gaussian.

$$p\left(\mu \mid \mathbf{x}, \sigma\right) \approx \text{normal}\left(\mu \mid \mathrm{m}_N, \mathrm{s}_N\right)$$

where m_N is the Laplace Approximation posterior mean and s_N is the Laplace Appoximation posterior standard deviation. Since this is a single parameter setting, the covariance matrix is just a scalar value (the variance). The square root of the variance is the standard deviation. You must determine the approximate posterior standard deviation using your result for the second derivative in Problem 2a).

Write out the expressions for the approximate posterior mean and posterior standard deviation. You may use the expression for the posterior mode from the previous assignment. You may write the posterior standard deviation in terms of precision.

SOLUTION The approximate posterior mean m_N is expressed as:

$$m_N = \frac{\frac{N}{\sigma^2}\bar{x} + \frac{\mu_0}{\tau_0^2}}{\frac{N}{\sigma^2} + \frac{1}{\tau_0^2}}$$

The posterior standard deviation s_N can be approximated using the result from Problem 2a), which is the negative inverse of the second derivative of the log-posterior evaluated at the mode:

$$s_N = \frac{1}{\sqrt{\frac{N}{\sigma^2} + \frac{1}{\tau_0^2}}}$$

2d)

We saw in lecture how the Laplace Approximation is just that, an approximation. However, for this specific application (one parameter normal-normal model with an unknown mean) the Laplace Approximation is **not** an approximation. In fact, the expressions for the posterior mean and posterior precision were discussed in lecture.

Why is the Laplace Approximation equal to the exact posterior distribution for this specific application?

SOLUTION In this specific case the likelihood is a normal distribution and the prior is also a normal distribution. When the likelihood and prior have conjugate forms, it means that the posterior distribution belongs to the same family of distributions as the prior. As a result, the Laplace Approximation, which approximates the posterior as a Gaussian distribution, is not an approximation but an exact representation of the posterior.

Problem 03

Let's now return to the two parameter application from Problem 01 with the goal of learning the unknown mean, μ, and unknown noise, σ. However, before applying the Laplace Approximation to this setting, you will perform a change-of-variables transformation to σ. The transformed variable, φ, is related to σ through the transformation or transformation function $g\left(\cdot\right)$:

$$\varphi = g\left(\sigma\right)$$

3a)

Why is it useful to transform σ to φ using a transformation like the natural log when we perform the inference with the Laplace Approximation?

SOLUTION

1) Transforming σ to φ via a logarithmic transformation makes the relationship between the parameters linear. This often simplify calculations.
2) The natural logarithm function is well-behaved, and it maps positive values of σ to the entire real number line. This transformation ensures that φ is not constrained to a specific range.
3) When finding the mode and calculating the Hessian matrix for the Laplace Approximation, taking derivatives of the log-posterior can be simplified when working with transformed variables like φ.

3b)

The generic inverse transformation function back-transforms from φ to the noise, σ:

$$\sigma = g^{-1}\left(\varphi\right)$$

Write out the un-normalized joint posterior between the unknown mean, μ, and the transformed noise, φ, via the probability change-of-variables formula.

You do NOT need to simplify the distributions in any way. You may write the "names" or labels of the distributions (such as normal() **and** Exp()**). You must correctly substitute in for the inverse transformation function into the log-posterior "based" on the original parameter σ. You must include all terms from the change-of-variables formula.**

SOLUTION The un-normalized joint posterior between the unknown mean, μ, and the transformed noise, φ, via the probability change-of-variables formula is:

$$p\left(\mu, \sigma \mid \mathbf{x}\right) \propto \prod_{n=1}^{N} \left(\text{normal}\left(x_n \mid \mu, g^{-1}\left(\varphi\right)\right)\right) \times \text{normal}\left(\mu \mid \mu_0, \tau_0\right) \times \text{Exp}\left(g^{-1}\left(\varphi\right) \mid \lambda\right) \cdot \left| \frac{d}{d\varphi}\left(g^{-1}\left(\varphi\right)\right) \right|$$

3c)

The weight example in lecture used the logit function as the transformation function. You will not use the logit function. Instead, you will use the natural log as the transformation function:

$$\varphi = g\left(\sigma\right) = \log\left(\sigma\right)$$

Write out the inverse transformation function and derive the natural log of the derivative adjustment.

SOLUTION The inverse transformation function is: $g^{-1}\left(\varphi\right) = \exp(\varphi)$.

The log of the derivative adjustment is: $\log\left|\frac{d}{d\varphi}\left(g^{-1}\left(\varphi\right)\right)\right| = \log(\exp(\varphi)) = \varphi$.

3d)

You must now define a function to calculate the log-posterior between μ and φ. The `my_cv_logpost()` is started for you in the code chunk below. It also uses two input arguments, with the same names as the `my_logpost()` function. The first argument is again the vector of unknowns and the second argument is the list of required information. However, the `unknowns` vector is intended to be different from that in `my_logpost()`. As shown in the code chunk below, the second element of `unknowns` corresponds to the transformed noise parameter, φ.

Note that you will use the same list of required information, `hw06_info`, that you defined in previously in Problem 01.

Complete the `my_cv_logpost()` function. The variable names and comments describe what you are required to complete.

You ARE allowed to use built in `R` functions for densities in this problem.

Several test values for the `unknowns` input vector are provided for you to try out below.

```r
my_cv_logpost <- function(unknowns, my_info)
{
  # unpack the unknowns into separate variables
  lik_mu <- unknowns[1]
  lik_varphi <- unknowns[2]

  # back transform to sigma
  lik_sigma <- exp(lik_varphi)

  # calculate the log-likelihood
  log_lik <- sum(dnorm(x=my_info$xobs,
                       mean=lik_mu,
                       sd=lik_sigma,
                       log=TRUE))

  # calculate the log-prior on mu
  log_prior_mu <- dnorm(x=lik_mu,
                        mean=my_info$mu_0,
                        sd=my_info$tau_0,
                        log=TRUE)

  # calculate the log-prior on sigma
  log_prior_sigma <- -my_info$sigma_rate * lik_sigma + log(my_info$sigma_rate)

  # calculate the log-derivative adjustment
  log_deriv_adjust <- lik_varphi

  # return the (un-normalized) log-posterior
  log_lik+log_prior_mu+log_prior_sigma+log_deriv_adjust
}
```

SOLUTION Test out the function to check that it works as expected. Try a value of 13 for μ and a value of 0 for φ. If you programmed the `my_logpost()` function correctly, you should get a value of -83.66777 printed to the screen.

```r
unknowns <- c(13,0)
my_cv_logpost(unknowns, hw06_info)
```

```
## [1] -83.66777
```

Test out the function to check that it works as expected. Try a value of 7 for μ and a value of -1 for φ. If you programmed the `my_logpost()` function correctly, you should get a value of -605.1904 printed to the screen.

```r
unknowns <- c(7,-1)
my_cv_logpost(unknowns, hw06_info)
```

```
## [1] -605.1904
```

3e)

Let's visualize what the the log-posterior surface looks like in the μ, φ space. You must define a grid of parameter combinations, similarly to what you did in Problem 01. However, this time you must define the grid in terms of combinations of μ and φ (instead of μ and σ).

You must define the from (lower bound) and to (upper bounds) on the φ parameter. You should apply the natural log transformation function to the bounds on σ defined in Problem 1e). After specifying the bounds, create the full-factorial combinations between `mu` and `varphi` using the `expand.grid()` function. Use the same bounds on `mu` that you used in Problem 01 and use `length.out=251` for both parameters. Assign the result to the `cv_param_grid` variable.

SOLUTION Define the bounds on φ for the grid.

```
varphi_grid_lwr <- log(sigma_grid_lwr)
varphi_grid_upr <- log(sigma_grid_upr)
```

Create the grid of full-factorial combinations with μ.

```
cv_param_grid <- expand.grid(mu = seq(mu_grid_lwr, mu_grid_upr, length.out = 251),
                             varphi = seq(varphi_grid_lwr, varphi_grid_upr, length.out = 251),
                             KEEP.OUT.ATTRS = FALSE, stringsAsFactors = FALSE) %>%
  as.data.frame() %>% tibble::as_tibble()
```

3f)

The `eval_logpost()` function was defined using generic variable names in order to be used for the original log-posterior evaluation **and** the change-of-variables log-posterior. Problem 1g) demonstrated how to apply `eval_logpost()` to every combination of `mu` and `sigma` using the `purrr::map2_dbl()` function. You should follow those steps but adapt the code provided to you in Problem 1g) in order to calculate the `my_cv_logpost()` function to every combination of `mu` and `varphi` contained in `cv_param_grid`.

Apply the `eval_logpost()` function to every combination of variables in `cv_param_grid`. You must use the `purrr::map2_dbl()` function to functionally loop over all combinations in `cv_param_grid`. You may follow the code example provided in Problem 1g). However, be careful to change the variable names! Assign the result to the `log_post_cv_results`.

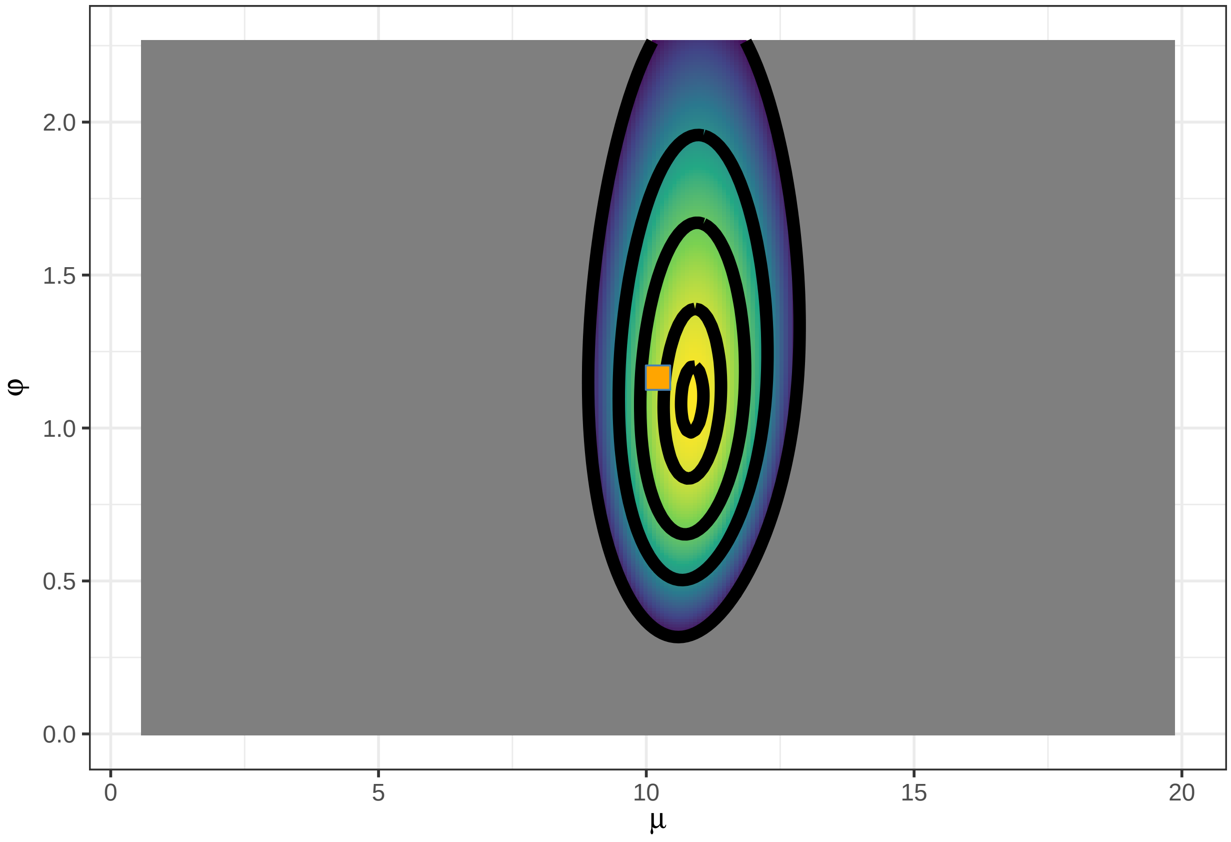
```
log_post_cv_result <- purrr::map2_dbl(cv_param_grid$mu, cv_param_grid$varphi,
                                      eval_logpost,
                                      logpost_func = my_cv_logpost,
                                      logpost_info = hw06_info)
```

3g)

The log-posterior surface between μ and φ is visualized for you in the code chunk below. As in Problem 1g), you must complete the `geom_point()` by including the sample average and the log-transformed sample standard deviation.

Complete the `geom_point()` call in the code chunk below. The comments specify where you should include the sample average and the log of the sample standard deviation. Describe the contour shapes of the log-posterior and how the overall shape compares to the log-posterior in the original parameter space between μ and σ.

```
cv_param_grid %>%
  mutate(log_post = log_post_cv_result,
         log_post_2 = log_post - max(log_post)) %>%
  ggplot(mapping = aes(x = mu, y = varphi)) +
  geom_raster(mapping = aes(fill = log_post_2)) +
  stat_contour(mapping = aes(z = log_post_2),
               breaks = log(c(0.01/100, 0.01, 0.1, 0.5, 0.9)),
               size = 2.2,
               color = "black") +
  # include the sample average (xbar) and the log-sample standard deviation (log_xsd)
  geom_point(data = tibble::tibble(xbar = avrg_x, log_xsd = log(std_dev_x)),
             mapping = aes(x = xbar, y = log_xsd),
             shape = 22,
             size = 4.5, fill = "orange", color = "steelblue") +
  scale_fill_viridis_c(guide = FALSE, option = "viridis",
                       limits = log(c(0.01/100, 1.0))) +
  labs(x = expression(mu), y = expression(varphi)) +
  theme_bw()
```



SOLUTION

Problem 04

It's now time to perform the Laplace Approximation on your transformed two parameter model. The first step is to find the posterior mode. You will not calculate the gradient vector and perform the optimization by hand in this question. Instead, you will use the `optim()` function to perform the optimization.

4a)

The code chunk below defines two different initial guesses for the unknown mean, μ, and unknown log-transformed noise, φ. You will try out both initial guesses and compare the optimization results.

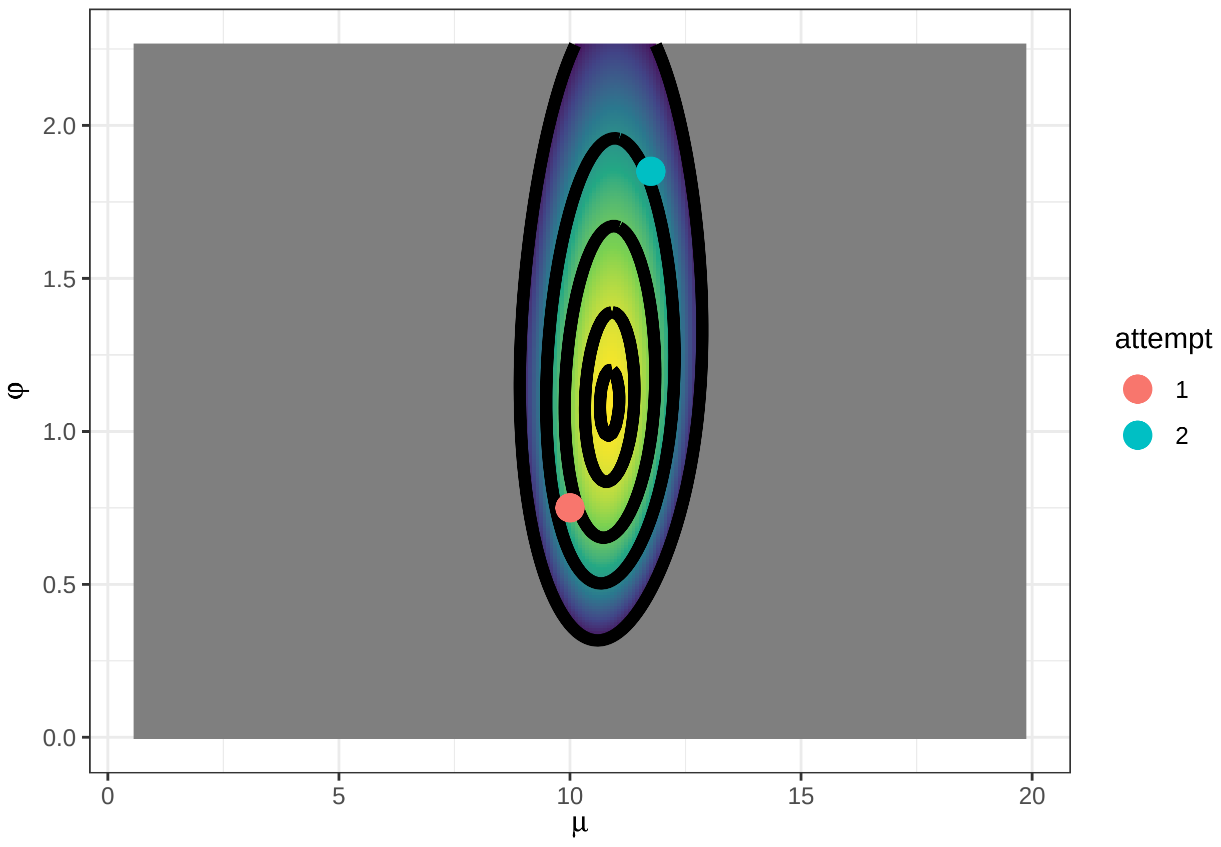
```r
init_guess_01 <- c(10, 0.75)

init_guess_02 <- c(11.75, 1.85)
```

Let's first visualize these two points relative to the posterior mode, since you know what the log-posterior surface looks like.

Complete the code chunk below by visualizing the two different initial guesses with a `geom_point()` geom on top of the log-posterior surface. Think through which element in the `init_guess_01` and `init_guess_02` vectors corresponds to which parameter. You do not need to change the `aes()` call within the `geom_point()` geom below. You must correctly specify the variables in the `tibble` of the `data` argument to `geom_point()`.

```r
cv_param_grid %>%
  mutate(log_post = log_post_cv_result,
         log_post_2 = log_post - max(log_post)) %>%
  ggplot(mapping = aes(x = mu, y = varphi)) +
  geom_raster(mapping = aes(fill = log_post_2)) +
  stat_contour(mapping = aes(z = log_post_2),
               breaks = log(c(0.01/100, 0.01, 0.1, 0.5, 0.9)),
               size = 2.2,
               color = "black") +
  # include the initial guess points
  geom_point(data = tibble::tibble(attempt = as.character(1:2),
                                   mu = c(init_guess_01[1], init_guess_02[1]),
                                   varphi = c(init_guess_01[2], init_guess_02[2])),
             mapping = aes(color = attempt),
             size = 4.5) +
  scale_fill_viridis_c(guide = FALSE, option = "viridis",
                       limits = log(c(0.01/100, 1.0))) +
  labs(x = expression(mu), y = expression(varphi)) +
  theme_bw()
```



SOLUTION

4b)

You will now find the posterior mode (the MAP) on the μ and φ parameters. You will repeat the optimization process twice. The first will use the `init_guess_01` starting guess, and the second will use the `init_guess_02` starting guess. Make sure you use the log-posterior function associated with the transformed noise.

Complete the two code chunks below. The first code chunk finds the posterior mode (MAP) based on the first initial guess `init_guess_01` and the second code chunk uses the second initial guess `init_guess_02`. You must fill in the arguments to the `optim()` call to find the μ and φ values which maximize the `my_cv_logpost()` function.

To receive full credit you must:
* **specify the initial guesses correctly**
* **specify the function to be optimized**
* **specify the gradient evaluation correctly**
* **correctly pass in the list of required information**
* **specify the "BFGS" algorithm to be used**
* **instruct `optim()` to return the Hessian matrix**
* **make sure `optim()` maximizes the log-posterior instead of trying to minimize it**
* **the max iterations (`maxit`) to be 1001**

SOLUTION Use the first initial guess.

```
map_res_01 <- optim(init_guess_01,
                    my_cv_logpost,
                    gr=NULL,
                    hw06_info,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
map_res_01
```

```
## $par
## [1] 10.855893  1.090389
##
## $value
## [1] -21.67633
##
## $counts
## function gradient
##       12        6
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##           [,1]       [,2]
## [1,] -4.903629   1.152284
## [2,]  1.152284 -18.463373
```

Use the second initial guess.

```
map_res_02 <- optim(init_guess_02,
                    my_cv_logpost,
                    gr=NULL,
                    hw06_info,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
map_res_02
```

```
## $par
## [1] 10.855942  1.090392
##
## $value
## [1] -21.67633
##
## $counts
## function gradient
##       21       11
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##           [,1]       [,2]
## [1,] -4.903623   1.152364
## [2,]  1.152364 -18.463377
```

4c)

You tried two different starting guesses… are the optimization results different?

Are the identified optimal parameter values the same? Are the Hessian matrices the same? Was anything different between the optimizations?

What about the log-posterior surface gave you a hint about how the two results would compare?

SOLUTION

1) The optimal parameter values and the Hessian matrices are the same in both cases.
2) The difference between the optimizations is the count of iterations. In the 1st initial guess the number of iterations is less than the one in the 2nd iteration guess.
3) The log-posterior surface gives us hint about why the first initial guess took less iterations. The location of the 1st iteration guess is closer to the posterior mode compared to the 2nd iteration guess. This resulted in getting convergence with less iterations.

4d)

Finding the posterior mode is the first step in the Laplace Approximation. The second step uses the negative inverse of the Hessian matrix as the approximate posterior covariance matrix. You wil use a function, `my_laplace()`, to perform the complete Laplace Approximation. This one function is all that is needed to perform all steps of the Laplace Approximation.

Complete the code chunk below. The `my_laplace()` function is adapted from the `laplace()` function from the `LearnBayes` package. Fill in the missing pieces to double check that you understand which portions of the optimization result correspond to the mode and which are used to approximate the posterior covariance matrix.

SOLUTION Complete the missing pieces of the code chunk below. The last portion of the `my_laplace()` function compiles the results into a list.

```r
my_laplace <- function(start_guess, logpost_func, ...)
{
  # code adapted from the `LearnBayes`` function `laplace()`
  fit <- optim(start_guess,
               logpost_func,
               gr = NULL,
               hw06_info,
               method = "BFGS",
               hessian =  TRUE,
               control = list(fnscale = -1, maxit = 5001))

  mode <- fit$par
  post_var_matrix <- -solve(fit$hessian)
  p <- length(mode)
  # we will discuss what int means in a few weeks...
  int <- p/2 * log(2 * pi) + 0.5 * log(det(post_var_matrix)) + logpost_func(mode, ...)
  # package all of the results into a list
  list(mode = mode,
       var_matrix = post_var_matrix,
       log_evidence = int,
       converge = ifelse(fit$convergence == 0,
                         "YES",
                         "NO"),
       iter_counts = as.numeric(fit$counts[1]))
}
```

4e)

You will now perform the Laplace Approximation to determine the approximate posterior on the μ and φ parameters given the measurements.

Call the `my_laplace()` function to approximate the posterior on μ and φ. Check that solution converged. Display the posterior means on each parameter. Display the posterior standard deviations on each parameter. What is the posterior correlation coefficient between μ and φ?

SOLUTION Execute the Laplace Approximation.

```
laplace_result <- my_laplace(init_guess_01,my_cv_logpost,hw06_info)
laplace_result
```

```
## $mode
## [1] 10.855893  1.090389
##
## $var_matrix
##             [,1]       [,2]
## [1,] 0.20696580 0.01291656
## [2,] 0.01291656 0.05496740
##
## $log_evidence
## [1] -22.08394
##
## $converge
## [1] "YES"
##
## $iter_counts
## [1] 12
```

1) Posterior means on μ is approximately 10.855893.
2) Posterior means on φ is approximately 1.090389.
3) Posterior standard deviation on μ is approximately $\sqrt{0.20696580}$=0.4549349.
4) Posterior standard deviation on φ is approximately $\sqrt{0.05496740}$=0.2344513.
5) Posterior correlation coefficient between μ and φ is $\frac{0.01291656}{0.4549349 * 0.2344513}$=0.1211002.
6) The solution is converged.

Problem 05

You will now use the Laplace Approximation to answer questions from the toy company described back in Problem 01. After all, we were learning the unknown parameters to describe behavior. It is now time to discuss what you learned!

5a)

Use the Laplace Approximation result to calculate the probability that the unknown mean, μ, is less than the sample average.

SOLUTION In problem 1a), we obtained that the sample average is 10.21831. In problem 4e), we obtained that posterior means on μ is 10.855893. Also, using c.o.v. we can calculate posterior means on σ as exp(1.090389)=2.9754313.

Then, using these values, the probability that the unknown mean, μ, is less than the sample average can be obtained as:

```r
prob <- pnorm(10.21831, mean = 10.855893, sd = 2.9754313)
prob
```

```
## [1] 0.4151634
```

5b)

The Laplace Approximation result in Problem 04 is associated the μ and φ parameters. However, the toy company described in Problem 01 is not interested in the φ parameter. They want to know about the noise in their process, and thus are interested in σ not φ. You will need to undo the change-of-variables transformation, while accounting for any potential posterior correlation with μ.

Rather than working through the math to accomplish this, let's just use random sampling. The Laplace Approximation is a known distribution type, specifically a MVN distribution. You will call a MVN random number generator, `MASS::mvrnom()`, to generate random observations from a MVN with a user specified mean, `mu`, and user specified covariance matrix, `Sigma`. You will then back-transform from φ to σ by simply calling the inverse transformation function.

The `generate_post_samples()` function is started for you in the code chunk below. The user provides the Laplace Approximation result as the first argument, `mvn_info`, and the number of samples to generate, `num_samples`. The `MASS::mvrnorm()` function is used to generate the posterior samples. A few data conversion steps are made before piping the result to a `mutate()` call. You **must** apply the correct inverse transformation function to calculate σ based on the randomly generated values of φ.

Complete the code chunk below. Assign the correct arguments to the `mu` and `Sigma` arguments to `MASS::mvrnorm()`. Use the correct inverse transformation function to back-transform from φ to σ.

NOTE: The `MASS` package is installed with base `R`, so you do **NOT** need to download it.

```r

generate_post_samples <- function(mvn_info, num_samples)
{
  MASS::mvrnorm(n = num_samples,
                mu = mvn_info$mode ,
                Sigma = mvn_info$var_matrix ) %>%
    as.data.frame() %>% tibble::as_tibble() %>%
    purrr::set_names(c("mu", "varphi")) %>%
    mutate(sigma = exp(varphi) )
}
```

SOLUTION

5c)

Generate 1e4 posterior samples from the Laplace Approximation posterior distribution and assign the result to the variable `post_samples`.

Use the `summary()` function to quickly summarize the posterior samples on each of the parameters.

Apply the correct inverse transformation function to the posterior mean on `varphi`. Does the result equal the posterior mean on `sigma`?

```r
set.seed(202004)
post_samples <- generate_post_samples(laplace_result, 10000)
summary(post_samples)
```

SOLUTION

```
##        mu             varphi           sigma      
##  Min.   : 9.056   Min.   :0.1896   Min.   :1.209  
##  1st Qu.:10.543   1st Qu.:0.9341   1st Qu.:2.545  
##  Median :10.855   Median :1.0927   Median :2.982  
##  Mean   :10.853   Mean   :1.0933   Mean   :3.068  
##  3rd Qu.:11.165   3rd Qu.:1.2477   3rd Qu.:3.482  
##  Max.   :12.553   Max.   :1.9432   Max.   :6.981  
```

Now lets apply the inverse transformation function (which is exp(x)) to the posterior mean on `varphi`.

```r
post_samples$sigma <- exp(post_samples$varphi)
mean(post_samples$sigma)
```

```
## [1] 3.068196
```

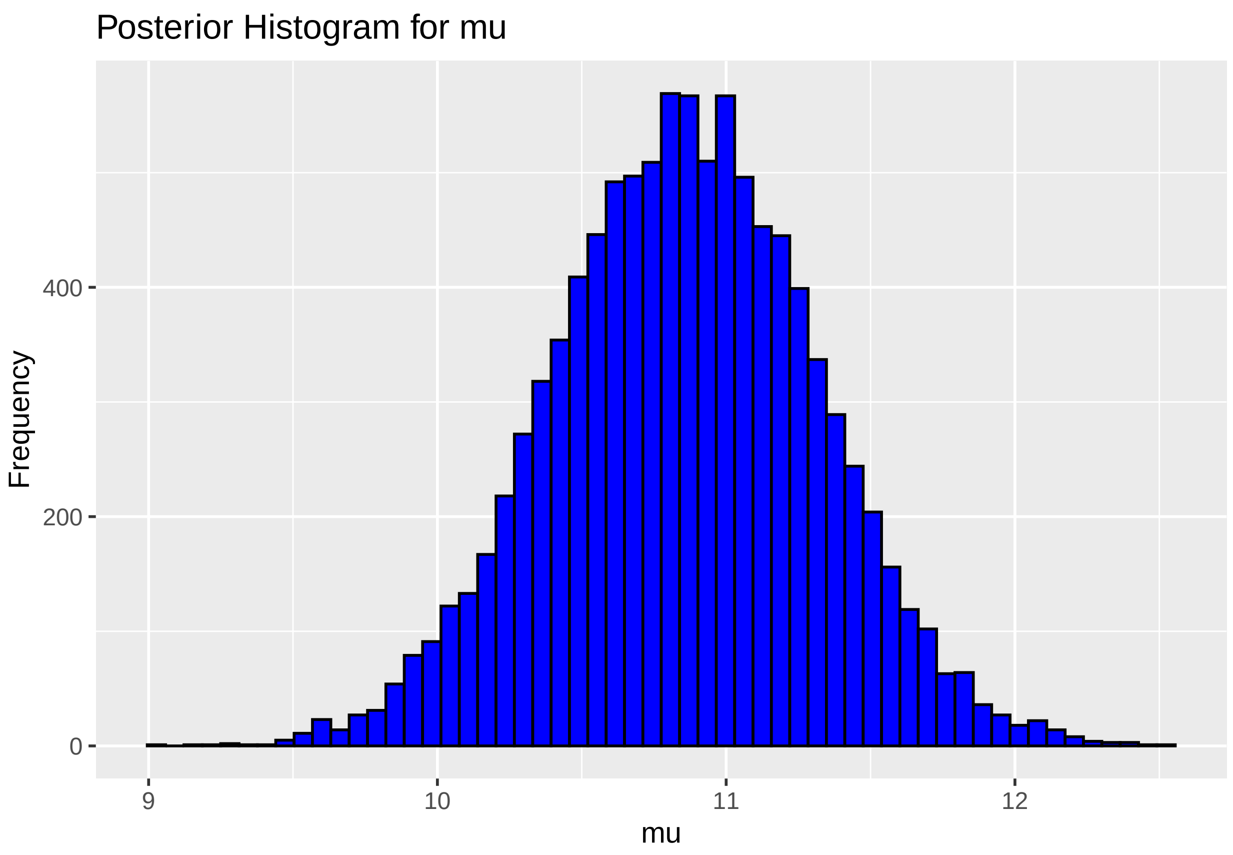
We can observe that the result equal the posterior mean on `sigma`.

5d)

Use `ggplot2` to visualize the posterior histograms on the μ and σ parameters. Set the number of bins to 55. You may use separate `ggplot()` calls for each histogram.

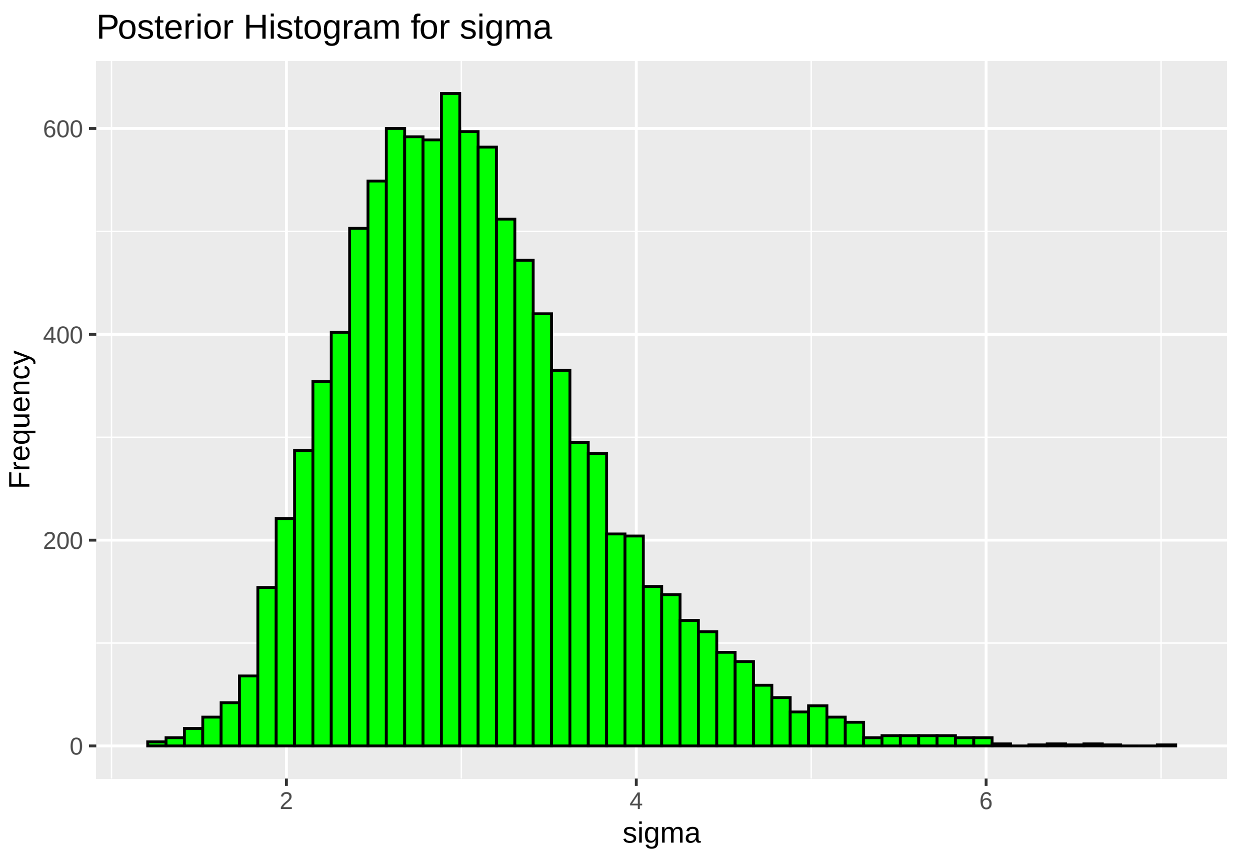
Does the posterior distribution on σ look Gaussian?

```r
ggplot(data = post_samples, aes(x = mu)) +
  geom_histogram(binwidth = (max(post_samples$mu)-min(post_samples$mu))/55, fill = "blue", color = "bla
  labs(title = "Posterior Histogram for mu", x= "mu", y= "Frequency")
```



SOLUTION

```
ggplot(data = post_samples, aes(x = sigma)) +
  geom_histogram(binwidth = (max(post_samples$sigma)-min(post_samples$sigma))/55, fill = "green", color
  labs(title = "Posterior Histogram for sigma", x ="sigma", y= "Frequency")
```



While the posterior on μ look Gaussian, we can say that the posterior on σ does not look Gaussian because it is not symmetric.

5e)

The toy company would like to know based on the limited data set the variation in their manufacturing process. Specifically, they want to know the probability that the noise is greater than 4 units.

Calculate the posterior probability that σ is greater than 4.

HINT: Remember the basic definition of probability!

```r
count_g_4 <- sum(post_samples$sigma > 4)
total_sampl <- nrow(post_samples)
prob_sigma_g_4 <- count_g_4 / total_sampl
prob_sigma_g_4
```

SOLUTION

```
## [1] 0.1084
```

In order to calculate the posterior probability that σ is greater than 4, first we need to count the number of samples where sigma is greater than 4. Then we divide this number by the total number of samples. The code above does this and calculates the probability as 0.1084.