Overview

This assignment provides practice working with and manipulating Gaussian distributions. You will calculate derivatives, perform change-of-variable transformations, and also start working with the `optim()` function! You will therefore get practice numerically solving optimization problems even though lecture and readings focused on the derivations.

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

Consider a situation where a manufacturing company wants to get a better idea about the **mean** performance of a part they produce. Unfortunately, measuring the performance of that part requires a destructive experiment. Therefore, they would like to test as few parts as possible. Engineers within the company have performed experiments in the past, and so they hope that experience can be incorporated.

To help out this company, we will assume that the part performance can be represented by a Gaussian likelihood, given the mean performance μ and the noise σ. We will assume the observations, x_n, are conditionally independent given those parameters. The joint distribution between N observations can therefore be factored into the product of N likelihoods:

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

1a)

In order to perform a Bayesian analysis, we need to specify our prior belief on the unknown parameters. The parameter the engineers are most concerned about is μ. Let's represent our prior belief on μ with a Gaussian prior parameterized by μ_0 and τ_0. After asking multiple engineers, it seems that the prior ± 2 standard deviation interval or roughly 95% uncertainty interval around the prior mean, μ_0, is between 115 units and 135 units.

Based on the interval suggested by the engineers, solve for the μ_0 and τ_0 parameters for the Gaussian prior on the mean, μ.

SOLUTION We have the following equations:

$$\mu_0 + 2\tau_0 = 135$$
$$\mu_0 - 2\tau_0 = 115$$

When we solve the above system, we get $\mu_0 = 125$ and $\tau_0 = 5$.

1b)

Let's examine the prior uncertainty another way besides the 95% prior uncertainty interval.

Based on your result in Problem 1a), calculate the 25th and 75th quantiles associated with the prior Gaussian on μ.

The interval between these two quantiles contains what fraction of the probabilty mass?

SOLUTION We can use the `qnorm` function in `R`. We can find the 25th and 75th quantiles associated with the prior Gaussian on μ as follows.

```
quantile_25 <- qnorm(0.25,mean =125,sd = 5)
quantile_75 <- qnorm(0.75,mean =125,sd = 5)
```

```
quantile_25
```

```
## [1] 121.6276
```

```
quantile_75
```

```
## [1] 128.3724
```

Also, the interval between these two quantiles contains 1/2 of the probabilty mass.

1c)

Besides wanting to produce parts with as high as performance as possible, the manufacturing company has several customer requirements they must satisfy. One of their customers has stated they are concerned if large numbers of parts have performance values of 117 units or less.

Calculate the prior probability that the mean performance, μ is less than 117, based on your solution to Problem 1a).

```
z_score <- (117 - 125) / 5
prior_probability <- pnorm(z_score)
prior_probability
```

SOLUTION

```
## [1] 0.05479929
```

1d)

Most of the engineers feel confident that the process standard deviation, σ, is known and is $\sigma = 5$ units. We will now use this value of σ to estimate the required number of observations to achieve a target posterior standard deviation on the mean performance, μ.

Based on $\sigma = 5$ and your result in Problem 1a), how many observations are required to have a posterior standard deviation on μ to be $\tau_N = 2$? How many observations are required to have $\tau_N = 0.5$?

You should write out the expression for the number of required observations to achieve this goal and then calculate the values associated with two different τ_N values.

SOLUTION Using the formula $\frac{1}{\tau_N^2} = \frac{1}{\tau_0^2} + \frac{N}{\sigma^2}$, and the values $\tau_0 = 5,\ \sigma = 5$ and $\tau_N = 2$ we get the required number of observations as N=5.25.

Similarly, using the formula $\frac{1}{\tau_N^2} = \frac{1}{\tau_0^2} + \frac{N}{\sigma^2}$, and the values $\tau_0 = 5,\ \sigma = 5$ and $\tau_N = 0.5$ we get the required number of observations as N=99.

1e)

A small contingent of engineers feel that the others are overly optimistic. This group of engineers feel that the process standard deviation is $\sigma = 15$ units, and they do not feel comfortable specifying such a tight prior on μ. They are worried that a large number of tests will be needed to achieve $\tau_N = 3$ units.

Based on this small group of engineers, estimate the number of observations required to achieve $\tau_N = 3$ assuming $\sigma = 15$ and an infinitely diffuse or infinitely uncertain prior with $\tau_0 \to \infty$.

You should first write out the expression for the number of required observations and then calculate the value.

SOLUTION Using the formula $\frac{1}{\tau_N^2} = \frac{1}{\tau_0^2} + \frac{N}{\sigma^2}$, and the values $\tau_0 = 5,\ \sigma = 15$ and $\tau_N = 3$ we get the required number of observations as N=16.

As $\tau_0 \to \infty$ the above formula becomes $\frac{1}{\tau_N^2} = \frac{N}{\sigma^2}$, and using the values $\tau_N = 3,\ \sigma = 15$ we get the required number of observations as N=25.

Problem 02

In lecture, we introduced the normal-normal model with an unknown mean, μ, and known likelihood standard deviation (noise), σ. Although we discussed the interpretation of the posterior distribution when using the conjugate normal prior on μ, we did not fully derive the posterior. In this problem, you will practice working with Gaussians by deriving the expression for the *posterior mode* or Max A-Posteriori (MAP) estimate. You do **not** need to *complete the square* to solve this problem. Instead, you will find the MAP by optimizing the un-normalized log-posterior with respect to μ. The recording and notes provided on Canvas which show how to derive the Maximum Likelihood Estimate (MLE) may be helpful with this problem.

This problem uses the same nomenclature as lecture. The prior on μ is a Gaussian with prior mean μ_0 and prior standard deviation τ_0.

2a)

The posterior on μ given N observations $\mathbf{x}$ and known likelihood standard deviation (noise) σ is proportional to:

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

The denominator in Bayes' rule is ignored in the above expression and thus is referred to as the *un-normalized* posterior.

Write out the expression for the un-normalized log-posterior on μ. You may drop all terms that do not involve μ. You should show several steps and not simply write down the final answer.

SOLUTION We can express the following using the definition of Normal distribution:

$$p\left(\mu \mid \mathbf{x}, \sigma\right) \propto \prod_{n=1}^{N} \left\{ \exp\left(-\frac{1}{2\sigma^2}\left(x_n - \mu\right)^2 \right) \right\} \times \exp\left(-\frac{1}{2\tau_0^2}\left(\mu - \mu_0\right)^2 \right)$$

When we take log of both sides in the above expression and use properties of logartihms, we get the following,

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

which is the desired result.

2b)

Derive the first derivative of the un-normalized log-posterior with respect to μ.

Your final answer MUST be written in terms of the sufficient statistic, the sample average $\bar{x}$.

SOLUTION When we take the derivative of the last expression in 2a) with respect to μ, we get the following:

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

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

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

which gives the desired expression.

2c)

Determine the posterior mode or MAP by setting the derivative expression you determined in Problem 2b) to 0. Denote the posterior mode on μ as μ_{MAP}.

SOLUTION When we set the fnal equation to 0 and solve for μ, we get,

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

which is the desired expression.

2d)

Your expression for the posterior mode should look familiar. In fact, we worked with the expression for the MAP in lecture, but we referred to that expression by a different name.

Your answer in Problem 2c) should be equivalent to an expression provided in lecture. Which expression is it the same as? Why is your result for the posterior mode the same as that expression?

HINT: What type of distribution is the posterior in this case?

SOLUTION The result is the same as the expression given for "posterior mean" in the lecture slides. The results are the same because the posterior on μ is a normal distribution in this case and thus the mean is expected to give the MLE.

Problem 03

In lecture, we discussed learning an unknown mean of a normal likelihood, given a series of observations. We compared the MLE to the Bayesian result with a conjugate normal prior, assuming the measurement standard deviation (the likelihood noise) was known. We introduced the more complicated situation of an unknown mean and unknown noise. We will continue that discussion in the next series of lectures, where we will see how to approximate the joint posterior with the Laplace Approximation.

As a stepping stone to the two unknown parameter model, you will work with a normal likelihood but switch which parameters are known and unknown. Thus, you will work with a normal likelihood with a *known* mean, μ, while the likelihood standard deviation (noise), σ, is unknown. The N observations (or measurements) are still denoted as a vector, $\mathbf{x}$.

The posterior distribution on the unknown σ given the observations, $\mathbf{x}$, and the known mean, μ, is proportional to:

$$p\left(\sigma \mid \mathbf{x}, \mu\right) \propto p\left(\mathbf{x} \mid \mu, \sigma\right) p\left(\sigma\right)$$

We will continue to assume all observations are conditionally independent given the parameters. The likelihood therefore factors into the product of N normal likelihoods, and the posterior is proportional to:

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

Notice that the prior distribution must be placed on σ instead of μ. The example discussed in lecture used a Uniform distribution as the prior on σ. However, in this assignment you will work with a different distribution, one that we will work with more when we start fitting regression models. You will use an **Exponential distribution** as the prior on σ. The shorthand notation for the Exponential distribution is usually Exp, and I will always use a capital E to denote the distribution and not the exponential function exp(). The Exponential distribution has a single hyperparameter, known as the **rate parameter** which is usually denoted as λ. The Exponential probability density function with rate λ is:

$$\mathrm{Exp}\left(\sigma \mid \lambda\right) = \lambda \exp\left(-\lambda \sigma\right)$$

The un-normalized posterior on the unknown likelihood noise, σ, is therefore:

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

3a)

You will now start working with the un-normalized log-posterior for the unknown σ.

You must derive the expression for the un-normalized log-posterior on σ. You may drop the constant terms and terms that do not directly involve σ. You do not need to simplify the quadratic portion within the likelihood.

SOLUTION We can express the following using the definition of Normal distribution:

$$p(\sigma \mid \mathbf{x}, \mu) \propto \prod_{n=1}^{N} \left\{ \frac{1}{\sigma} \exp\left(-\frac{1}{2\sigma^2} (x_n - \mu)^2 \right) \right\} \times \lambda \exp(-\lambda\sigma)$$

When we take log of both sides in the above expression and use properties of logartihms, we get the following,

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

which is the desired result.

3b)

When we discussed the normal-normal model for the unknown mean, we talked about the sufficient statistic being the sample average. In this situation, an unknown standard deviation with a known mean, we also have a sufficient statistic, but it is **not** the sample average. The sufficient statistic is defined for you in the equation block below:

$$v = \frac{1}{N} \sum_{n=1}^{N} \left((x_n - \mu)^2 \right)$$

Simplify your result from Problem 3a) by making use of the sufficient statistic, v. Why do you think the quantity v is referred to as being "sufficient"? Does your expression for the un-normalized log-posterior depend on the individual observations after making use of v?

SOLUTION The sample standard deviation is considered a sufficient statistic for estimating the standard deviation when you assume a normal distribution. It captures the relevant information about the spread of the data.

When we plug in the expression of v in 3a), we get

$$\log\left[p(\sigma \mid \mathbf{x}, \mu)\right] \propto -N \log \sigma - \frac{1}{2\sigma^2} N v + \log \lambda - \lambda\sigma$$

As it can be seen in the above equation, the expression for the un-normalized log-posterior does not depend on the individual observations after making use of v?

3c)

Let's see how the un-normalized log-posterior behaves under a particular set of assumptions. The code chunk below defines a sample size of $N = 21$ and a sufficient statistic $v = 4.2$.

```r
N <- 21
v <- 4.2
```

In this problem, you will define a function `log_post_sigma_unnorm()` which has as input arguments, `sigma`, `N`, `v`, and `lambda`. The argument `sigma` is the σ value, `N` is the sample size, `v` is the sufficient statistic, and `lambda` is the prior rate parameter, λ.

Define the `log_post_sigma_unnorm()` function using the arguments and order of the arguments given in the problem statement.

```r
### set input arguments !!!
log_post_sigma_unnorm <- function(sigma, N, v, lambda)
{
  log_posterior <- -N*log(sigma)- (N * v) / (2 * (sigma^2)) + log(lambda)-lambda*sigma
  return(log_posterior)
}
```

SOLUTION

3d)

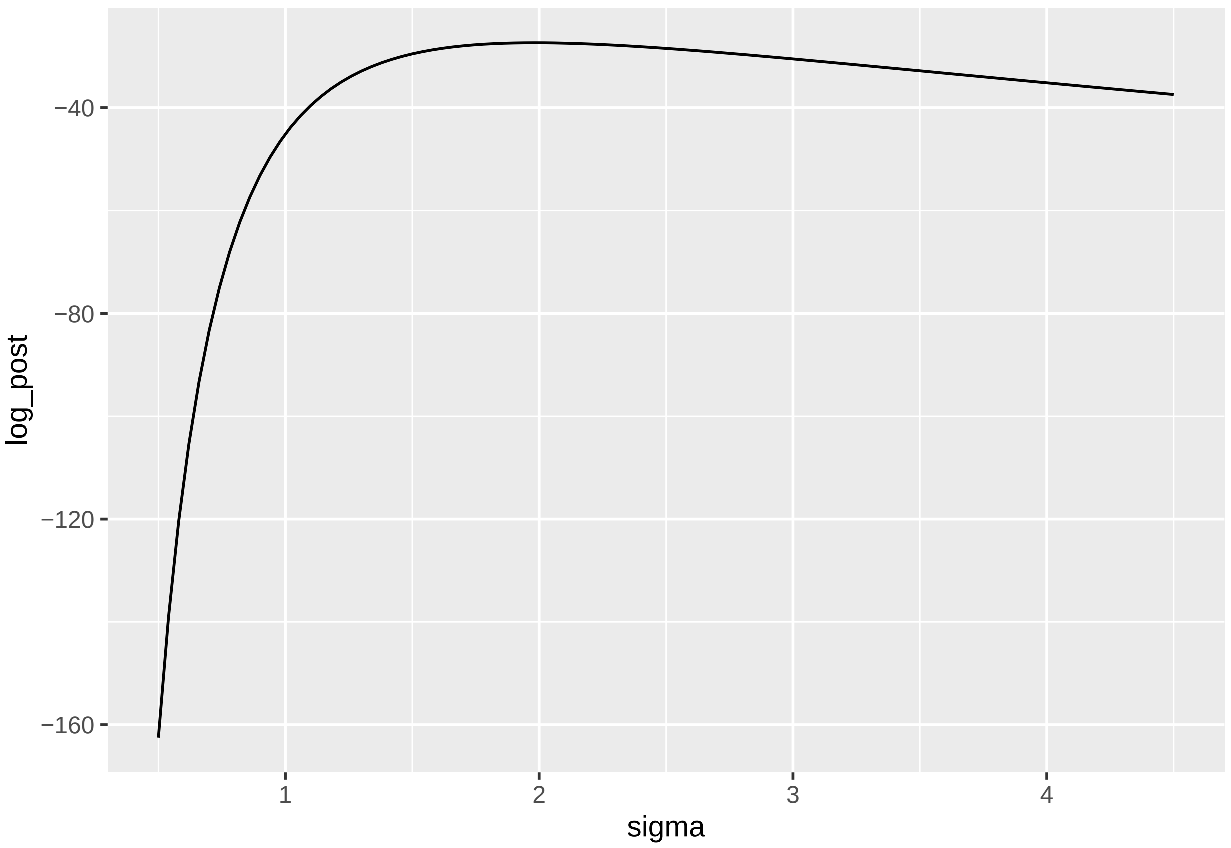
Let's visualize the log-posterior with respect to σ. You will use the assigned `N` and `v` variables, and assume that the prior rate parameter is equal to 0.75, $\lambda = 0.75$.

Complete the code chunk below. Assign a vector of 101 evenly spaced values between 0.5 and 4.5 to the variable `sigma` within the `tibble`. Pipe the result to a `mutate()` call and calculate the log-posterior using the `log_post_sigma_unnorm()` function. The log-posterior should be saved to the variable `log_post`. Pipe the result to `ggplot()` and plot the `sigma` and `log_post` variables as the `x` and `y` aesthetics, respectively. Use a `geom_line()` geometric object.

Does the log-posterior have one or several modes? Does the shape of the log-posterior look like a parabola?

```r

lambda=0.75
tibble::tibble(
  sigma = seq(0.5, 4.5, length.out = 101)
) %>%
  mutate(log_post = log_post_sigma_unnorm(sigma, N, v, lambda)) %>%
  ggplot(aes(x = sigma, y = log_post)) +
  geom_line()
```



SOLUTION

The log-posterior has one mode (close to 2), and the shape of the log-posterior does not look like a parabola.

3e)

One of the primary tools we will use throughout the semester is the quadratic or Laplace Approximation to approximate a generic posterior distribution as a Multivariate Normal. Although this can be a useful approximation, it is not appropriate for this type of situation. The likelihood noise, σ, has a natural lower bound of 0. A Gaussian distribution has no such constraints, and so allows all values from negative infinity to positive infinity! Therefore, even though the prior Exponential distribution respects the bound on σ, the lower bound would be violated if we apply the Laplace Approximation to approximate the posterior!

We need to make use of a transformation in order to overcome this limitation. Since the likelihood noise is lower bounded at 0, we will apply a log-transformation. The transformed variable, φ, will be equal to the log of σ. Thus, the transformation or **link** function, $g\left(\right)$, is equal to the log function:

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

The change-of-variables formula, written in terms of the log-density, is given below:

$$\log\left(p_{\varphi}\left(\varphi \mid \mathbf{x}, \mu\right)\right) = \log\left(p_{\sigma}\left(g^{-1}\left(\varphi\right) \mid \mathbf{x}, \mu\right)\right) + \log\left(\left|\frac{d}{d\varphi}\left(g^{-1}\left(\varphi\right)\right)\right|\right)$$

This may seem like an intimidating expression, but it simply says plug in the expression for σ in terms of φ (the inverse of the link function) into the log-posterior expression on σ. Then add in the log of the derivative of the inverse link function with respect to φ. The recording and notes provided on Canvas about performing change-of-variables with the simple linear z-score transformation can help with further discussion and practice working with the change-of-variables procedure.

Let's tackle this problem by first determining the log of the derivative of the inverse link function.

The link function is the log. What is the inverse link function? Write down the expression for calculating σ as a function of φ. Then calculate the derivative of the inverse link function with respect to φ. Finally, write down the log of the derivative of the inverse link function.

SOLUTION The inverse of the link function is the exp function. We can obtain that $\sigma = g^{-1}(\varphi) = \exp(\varphi)$. Then we can write:

$$\frac{d}{d\varphi}\left(g^{-1}\left(\varphi\right)\right) = \exp(\varphi)$$

Finally, we obtain that:

$$\log\left(\left|\frac{d}{d\varphi}\left(g^{-1}\left(\varphi\right)\right)\right|\right) = \varphi$$

which is the desired result.

3f)

To handle the substitution portion, you will define a new function `log_post_varphi_unnorm()` which accepts as input arguments, `varphi`, `N`, `v`, and `lambda`. The last three arguments are identical to those in `log_post_sigma_unnorm()`, while the first argument is the value of the transformed variable φ. Within the function, you must calculate `sigma` based on the supplied value of `varphi` using the correct inverse link function. With the value of `sigma` calculated, you can evaluate the log-posterior just as you did in previously. You must then account for the log-derivative adjustment by correctly adding in the log of the derivative you calculated in Problem 3e).

Complete the code chunk below by calculating the quantities specified in the comments.

```r
### set the input arguments !!!
log_post_varphi_unnorm <- function(varphi, N, v, lambda)
{
  # back-calculate sigma given varphi
  sigma <- exp(varphi)

  # calculate the unnormalized log-posterior on sigma
  log_post_sigma <- -N*log(sigma)- (N * v) / (2 * (sigma^2)) + log(lambda)-lambda*sigma

  # account for the derivative adjustment
  log_post <- log_post_sigma + varphi

  return(log_post)
}
```

SOLUTION

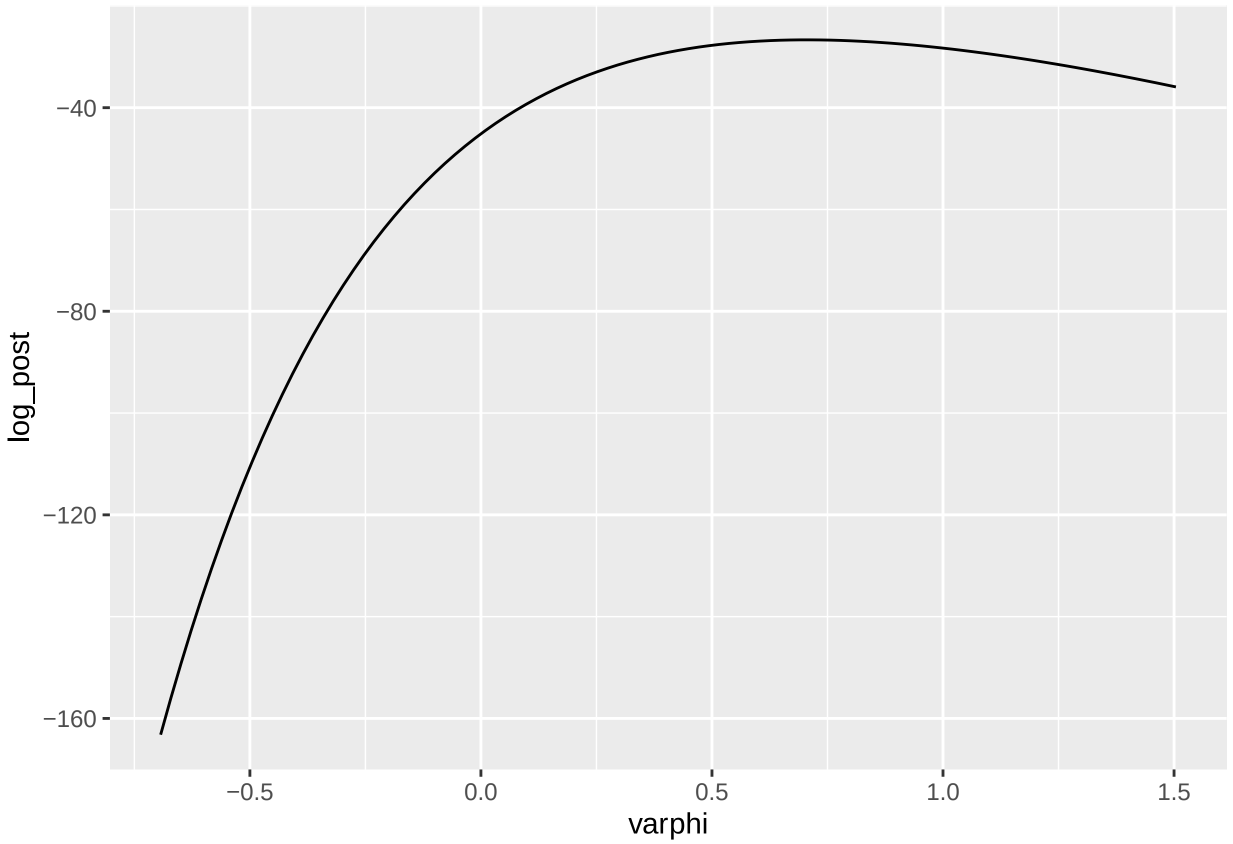
3g)

Now visualize the un-normalized log-posterior on the transformed variable φ. The code chunk below is similar to that used Problem 3d), except now you must define a vector of φ values instead of σ values. In Problem 3d), you calculated an evenly spaced vector of points between 0.5 and 4.5, in the σ space. Transform those bounds to the appropriate values in φ space and define a vector of 101 evenly spaced points between the transformed bounds. Pipe the result into `mutate()` and calculate the un-normalized log-posterior using the `log_post_varphi_unnorm()` function and save the result to the `log_post` variable. Pipe the result into `ggplot()` and visualize `varphi` and `log_post` as the `x` and `y` aesthetics with a `geom_line()`.

You will continue to use a prior rate parameter value of 0.75, $\lambda = 0.75$.

Visualize the un-normalized log-posterior on the transformed φ variable, following the instructions in the problem statement. Does the log-posterior appear more parabolic now?

```r
lambda=0.75
tibble::tibble(
  varphi = seq(log(0.5), log(4.5), length.out = 101)
) %>%
  mutate(log_post = log_post_varphi_unnorm(varphi, N, v, lambda)) %>%
  ggplot(aes(x = varphi, y = log_post)) +
  geom_line()
```



SOLUTION

Even though it is not a parabola, the log-posterior appear more parabolic compared to the result in 3d) above.

Problem 04

As you have seen in both lecture and the homework assignments, we will work with optimizing likelihoods and posteriors a lot in this course. Although we will work through the derivations for most problems, you will be allowed to use existing general purpose optimization packages to manage the optimization. There are many different optimization frameworks and packages we could try, but we will use `optim()` in base `R` as our optimization engine. This problem introduces you to the `optim()` function so that you may get used to its arguments and syntax.

4a)

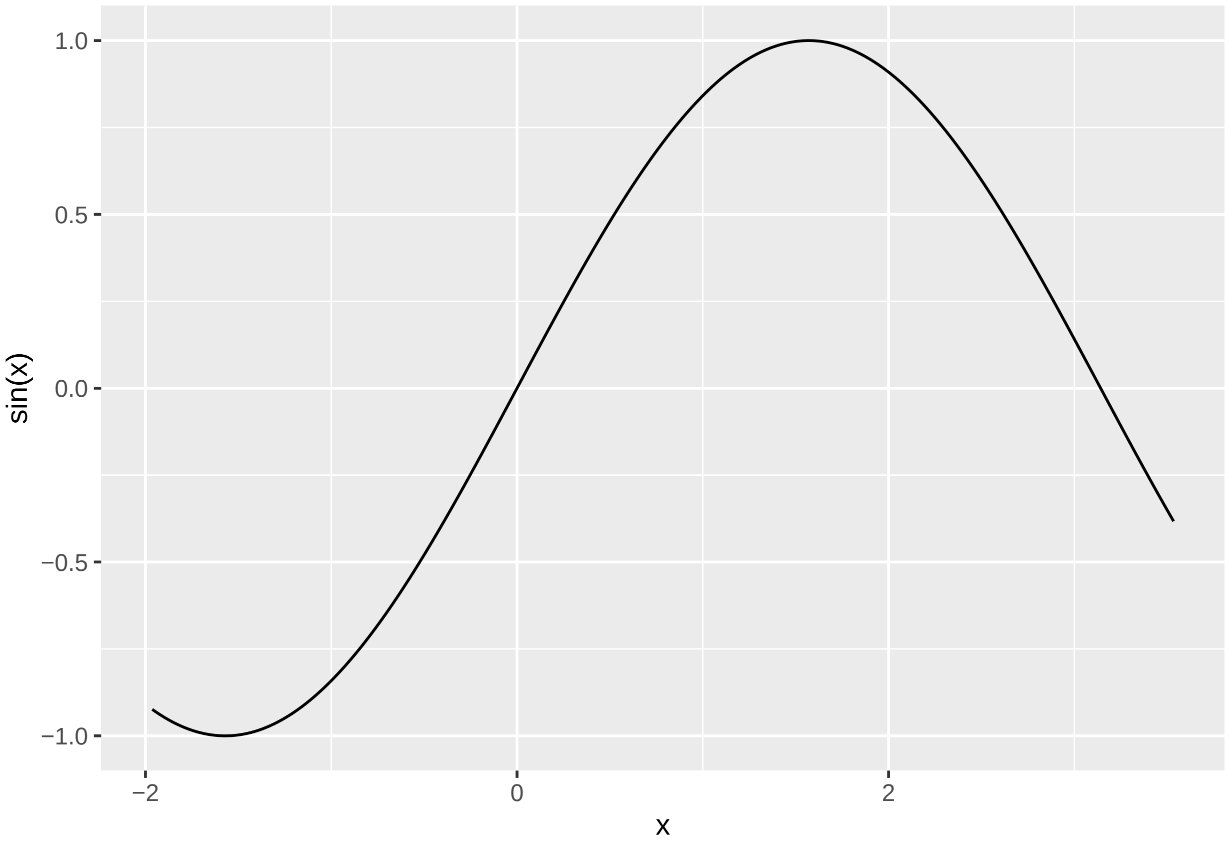
You will use `optim()` to find the value of `x` which maximizes the function `sin(x)` over the interval $-5\pi/8$ to $9\pi/8$. First, plot the `sin()` function over this interval. There are multiple ways to accomplish plotting a function within `ggplot2`, the method outlined below is a simple approach to get you further practice working with basic `ggplot2` commands.

Use the `seq()` function to create a vector `x` within the `tibble` started in the code chunk below. Set the arguments to `seq()` such that the vector contains 1001 elements from $-5\pi/8$ to $9\pi/8$. Pipe the `tibble` into `ggplot()` and plot the `sin(x)` with `geom_line()`.

```
x_values <- seq(-5*pi/8, 9*pi/8, length.out = 1001)

sin_plot_data <- tibble::tibble(x = x_values)

ggplot(sin_plot_data, aes(x = x)) +
  geom_line(aes(y = sin(x))) +
  labs(x = "x", y = "sin(x)")
```



SOLUTION

4b)

Let's now optimize the function with `optim()`. Type `?optim` into the R console to bring up the help documentation for the function. The documentation describes the arguments we need to pass into the function. The `par` argument is the first argument to `optim()` and is the initial guess. The second argument, `fn`, is the function we wish to optimize.

Use `optim()` to find the `x` value which maximizes `sin()` within the interval of interest. Set the initial guess to 0. Set the `method` in `optim()` to be `"Brent"` and set the `hessian` argument equal to

`TRUE`. Set the `lower` and `upper` arguments in the `optim()` to be `-5*pi/8` and `9*pi/8`, respectively. Store the result to `opt_sine_01` and print the result to the screen. Which `control` argument in `optim()` tells `optim()` to maximize instead of minimize? We will not pass in a gradient function, what should `gr` be set to?

```
obj_funct <- function(x) sin(x)

opt_sine_01 <- optim(par = 0, fn = obj_funct, method = "Brent",
                     lower = -5*pi/8, upper = 9*pi/8, hessian = TRUE,
                     control = list(fnscale = -1))

opt_sine_01
```

SOLUTION

```
## $par
## [1] 1.570796
##
## $value
## [1] 1
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##            [,1]
## [1,] -0.9999997
```

4c)

Reperform the `optim()` call from Problem 3b), but this time with an initial guess of 3 instead of 0. Store the result to `opt_sine_02`. Print `opt_sine_02` to the screen.

```
obj_funct <- function(x) sin(x)

opt_sine_02 <- optim(par = 3, fn = obj_funct, gr=NULL, method = "Brent",
                     lower = -5*pi/8, upper = 9*pi/8, hessian = TRUE,
                     control = list(fnscale = -1))

opt_sine_02
```

```
## $par
## [1] 1.570796
##
## $value
## [1] 1
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##            [,1]
## [1,] -0.9999997
```

4d)

We will now practice passing in extra arguments to the function we wish to optimize. We will first create our own custom function, for the following relationship:

$$f(x) = -\exp\left((x - a)^m\right)$$

You must create a custom function and define a list which stores the necessary or required information needed to evaluate the function.

Create the function, `my_func()` for the functional relationship shown in the above equation. The first argument to `my_func()` must be the variable `x`. The second argument will be a list named `constants`. The parameters a and m will be contained within this list as elements `a` and `m`, respectively. After defining `my_func()` create the list `my_constants` with elements `a` set to 0.7 and `m` equal to 2.

```
my_func <- function(x, constants)
{
  a <- constants$a
  m <- constants$m
  result <- -exp((x - a)^m)
  return(result)
}

my_constants <- list(a = 0.7, m = 2)
```

SOLUTION

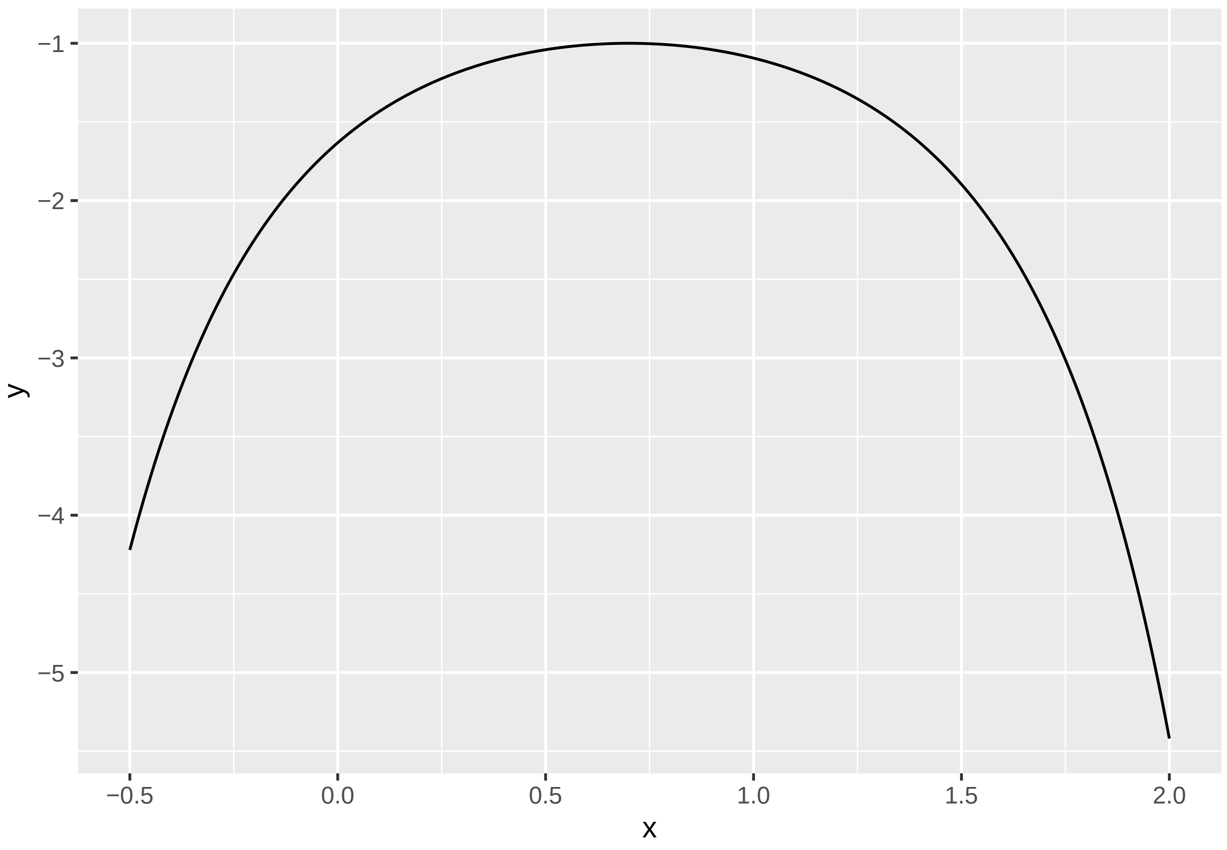
4e)

Let's visualize our custom function over the interval -0.5 to 2.

Within the `tibble` started below, set the `x` variable equal to the result of the `seq()` function consisting of 501 elements between -0.5 and 2. Pipe the `tibble` into `ggplot()` and plot the result of `my_func()` by passing in the `my_constants` list. Use `geom_line()` as the geometric object.

```
tibble::tibble(
  x = seq(-0.5, 2, length.out = 501),
  y = my_func(x, my_constants)
) %>%
```

```r
ggplot(mapping = aes(x = x, y = y)) +
geom_line()
```



SOLUTION

4f)

We will now call `optim()` to optimize our custom function over the interval -0.5 to 2.

Use `optim()` to find the `x` value which maximizes `my_func()`. Set the initial guess to 1.75. Set the `method` in `optim()` to be `"Brent"` and set the `hessian` argument equal to `TRUE`. Set the `lower` and `upper` arguments in the `optim()` to be -0.5 and 2, respectively. Which `control` argument in `optim()` tells `optim()` to maximize instead of minimize? We will not pass in a gradient function, what should `gr` be set to? You need to pass the `my_constants` list in order for `my_func()` to execute properly! Store the result to `opt_result` and print the result to the screen.

```r
obj_funct <- function(x, constants) {
  a <- constants$a
  m <- constants$m
  -exp((x - a)^m)
}

opt_result <- optim(par = 1.75, fn = obj_funct,constants = my_constants, gr=NULL, method = "Brent",
                     lower = -0.5, upper = 2, hessian = TRUE,
                     control = list(fnscale = -1))

opt_result
```

SOLUTION

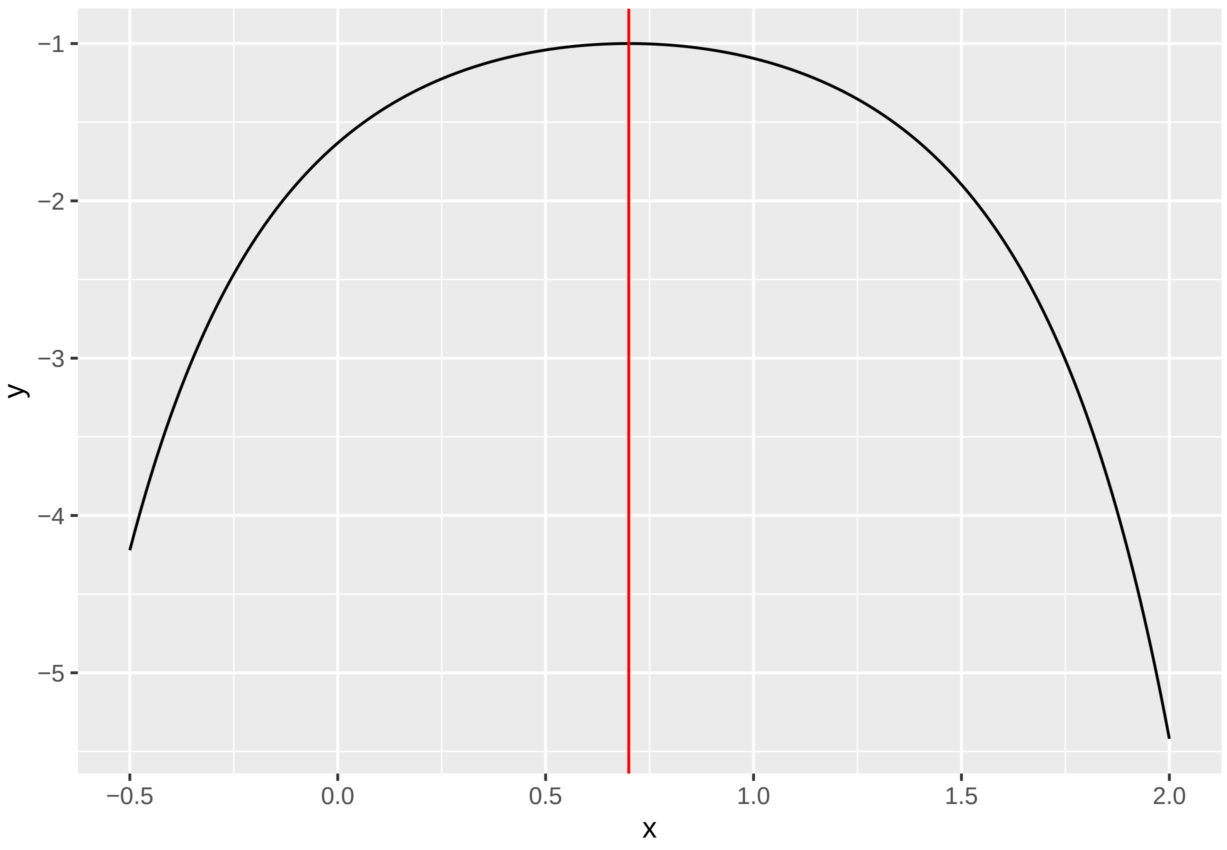
```
## $par
## [1] 0.7
##
## $value
## [1] -1
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##           [,1]
## [1,] -2.000004
```

4g)

We will now plot the identified optimal `x` value as a vertical line on top of the function over the interval of interest.

Add a vertical line with `geom_vline()` to your plot from Problem 4e). Set the `xintercept` within the `geom_vline()` function equal to the optimal `x` value by accessing the result stored in `opt_result` correctly. Set the color of the vertical line to red.

```r
tibble::tibble(
  x = seq(-0.5, 2, length.out = 501),
  y = my_func(x, my_constants)
) %>%
  ggplot(mapping = aes(x = x, y = y)) +
  geom_line() +
  geom_vline(xintercept = opt_result$par, color = "red")
```



SOLUTION

Problem 05

Now that you have practiced working with the `optim()` function, it's time to use `optim()` to find the posterior mode or MAP of the unknown φ parameter from Problem 03. Because φ is unbounded you do not have to include the `lower` and `upper` arguments to the `optim()` function call. Also, specify `method='BFGS'` instead of `method='Brent'` as used in Problem 04. Otherwise the `optim()` function arguments are consistent with the previous questions from Problem 04.

5a)

Use `optim()` to find the MAP of the log-transformed likelihood noise, φ, using an initial guess equal to -0.5. Assign all arguments to `optim()` that you feel are required to find the value that maximizes the log-posterior. However, assign the `method='BFGS'` instead of `method='Brent'` as used in Problem 04.

Assign the result to the `varphi_opt_a` object and display the object to the screen.

HINT: The additional parameters to the function are supplied after `gr`. Those additional arguments should be supplied in the order consistent with the function you are optimizing.

Continue to use `lambda = 0.75`.

```
obj_funct <- function(varphi, N, v, lambda) {
  sigma <- exp(varphi)
  -N*log(sigma)- (N * v) / (2 * (sigma^2)) + log(lambda)-lambda*sigma + varphi
}
```

```r
varphi_opt_a <- optim(par = -0.5, fn = obj_funct,gr=NULL, lambda = 0.75,N=21,v=4.2, method = "BFGS",
                      lower = log(0.5), upper = log(4.5), hessian = TRUE,
                      control = list(fnscale = -1))
```

SOLUTION

```
## Warning in optim(par = -0.5, fn = obj_funct, gr = NULL, lambda = 0.75, N = 21,
## : bounds can only be used with method L-BFGS-B (or Brent)
```

```r
varphi_opt_a
```

```
## $par
## [1] 0.7053493
##
## $value
## [1] -26.67228
##
## $counts
## function gradient
##       10       10
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
##
## $hessian
##           [,1]
## [1,] -44.55528
```

5b)

Repeat the `optim()` call from 5a), but this time use an initial guess of 1.35. Assign the result to the `varphi_opt_b` object and display the result to the screen.

Is the posterior mode from the initial guess of 1.35 different from the initial guess of -0.5?

```r
obj_funct <- function(varphi, N, v, lambda) {
  sigma <- exp(varphi)
  -N*log(sigma)- (N * v) / (2 * (sigma^2)) + log(lambda)-lambda*sigma + varphi
}

varphi_opt_b <- optim(par = 1.35, fn = obj_funct,gr=NULL, lambda = 0.75,N=21,v=4.2, method = "BFGS",
                      lower = log(0.5), upper = log(4.5), hessian = TRUE,
                      control = list(fnscale = -1))
```

SOLUTION

```
## Warning in optim(par = 1.35, fn = obj_funct, gr = NULL, lambda = 0.75, N = 21,
## : bounds can only be used with method L-BFGS-B (or Brent)
```

```r
varphi_opt_b
```

```
## $par
## [1] 0.7053493
```

```
##
## $value
## [1] -26.67228
##
## $counts
## function gradient
##        8        8
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
##
## $hessian
##           [,1]
## [1,] -44.55528
```

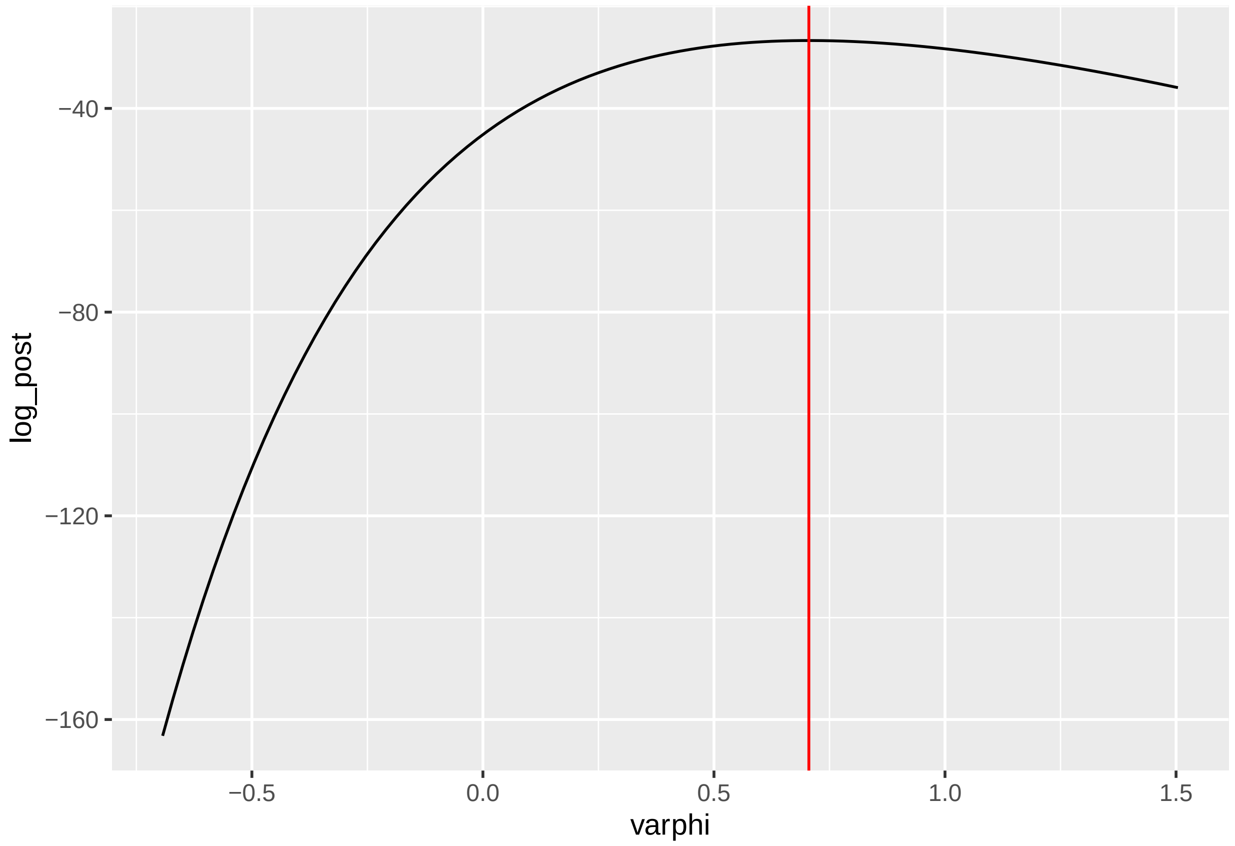
The posterior mode from the initial guess of 1.35 is the same as the initial guess of -0.5. Both have the value of 0.7053493.

5c)

Let's confirm the `optim()` identified parameter value is the mode by visualizing the optimal value relative to the log-posterior of the log-transformed noise, φ.

Visualize the un-normalized log-posterior on the transformed φ variable, just as you did in Problem 3g). However, this time include the optimal value as a vertical red line with `geom_vline()`.

```
lambda=0.75
tibble::tibble(
  varphi = seq(log(0.5), log(4.5), length.out = 101)
) %>%
  mutate(log_post = log_post_varphi_unnorm(varphi, N, v, lambda)) %>%
  ggplot(aes(x = varphi, y = log_post)) +
  geom_line()+
  geom_vline(xintercept = varphi_opt_a$par, color = "red")
```



SOLUTION