INFSCI 2595

Fall 2019

Information Sciences Building: Room 403

Lecture 05

Last week…

- We discussed two continuous probability distributions
  - Beta distribution
  - Gaussian distribution

- Please see the Probability Density Function Review on the Course Github page to see how to work with pdf’s in R.

Last week…

- We discussed the normal model with unknown mean, μ, and known standard deviation, σ.

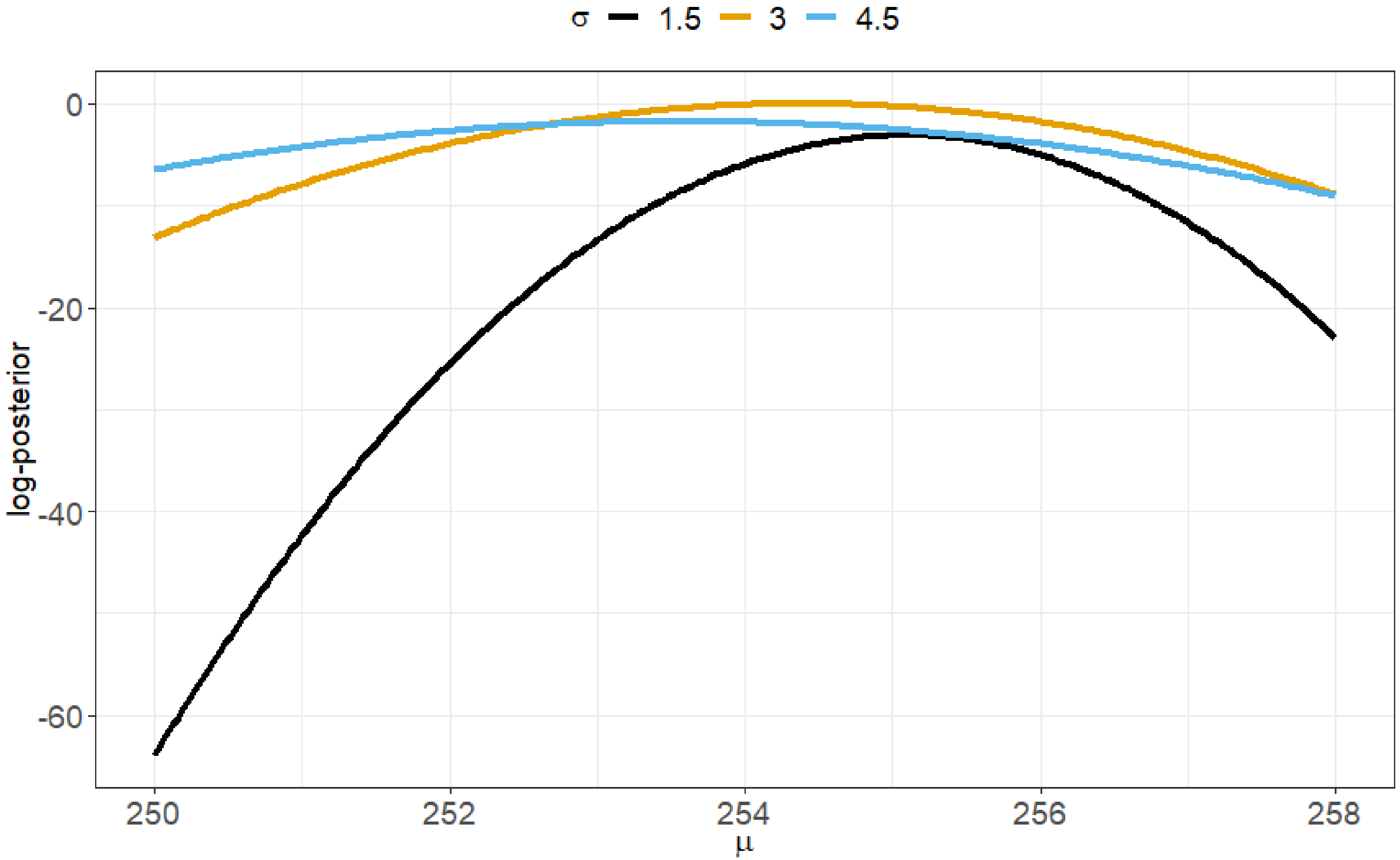
- Please see the normal-normal model supplemental material for more details and example code for evaluating log-posterior densities.

We concluded last week by introducing the grid approximation

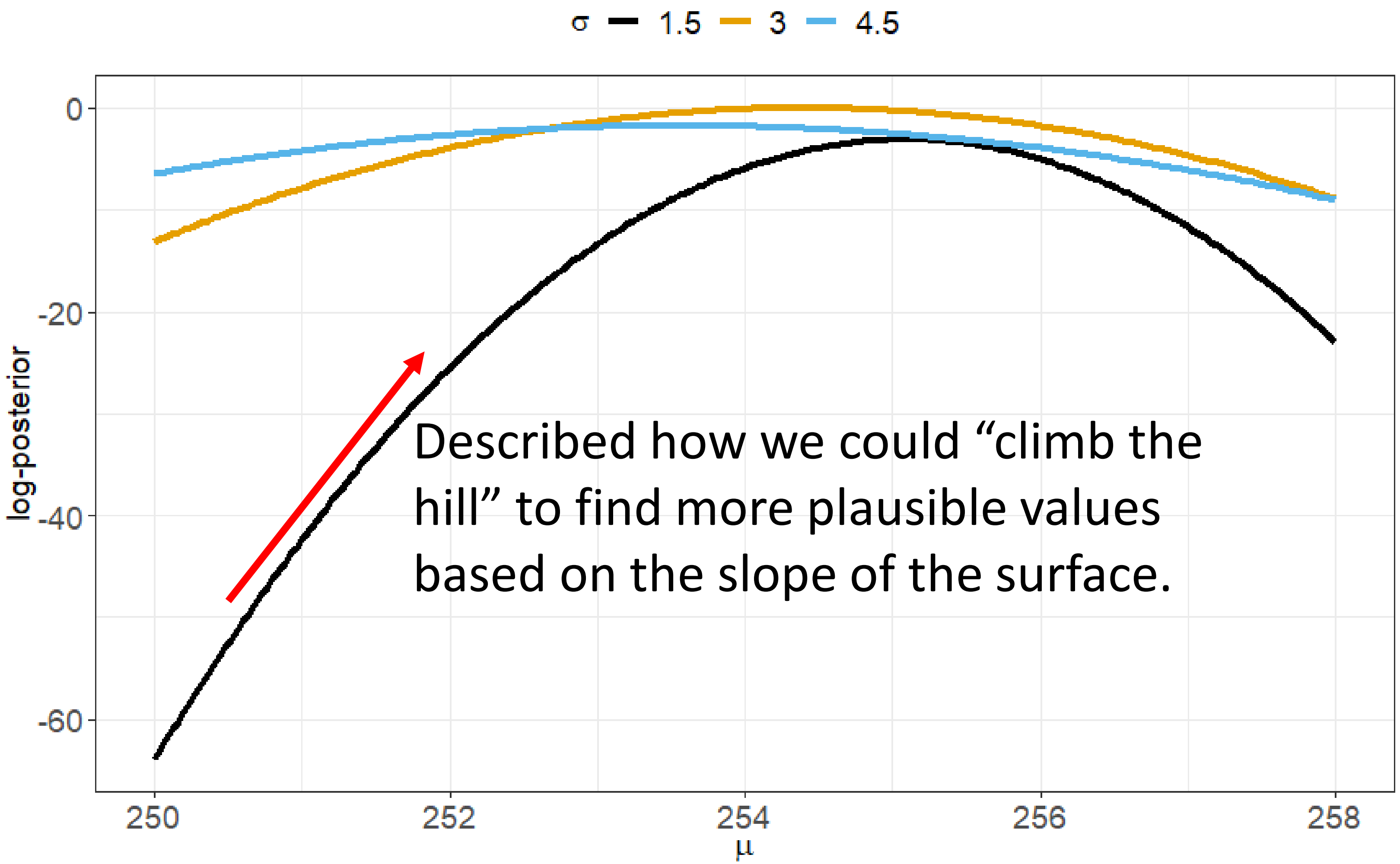
- We used the **grid approximation**, or *direct sampling*, to estimate the joint posterior distribution for an unknown mean **<u>and</u>** unknown standard deviation.

- Allowed us to estimate the probability that I weigh less than 255 pounds based on 10 observations.

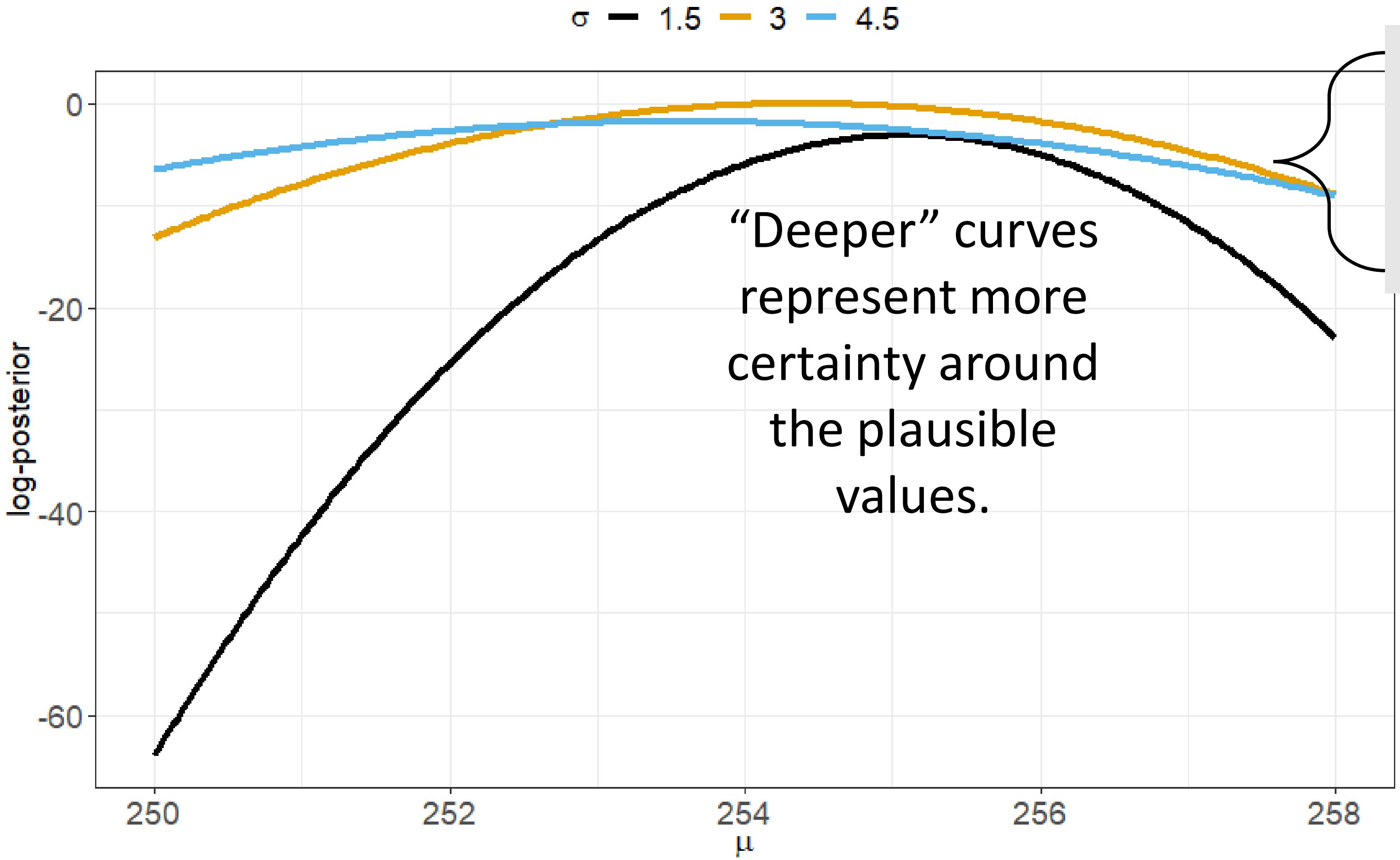
In our example, we visualized the log-posterior density with respect to the parameters



We discussed how **<u>higher</u>** log-posterior values correspond to ***more probable*** parameter values



We discussed how the shape or **<u>curvature</u>** of the surface represented the *certainty of plausible* parameter values



“Shallower” curves represent more parameter values are plausible, and thus are less certain.

Today, we will introduce a method to quantify these aspects.

- This method will allow us to approximate the posterior distribution centered around the most probable value or *Max A Posteriori* (MAP) estimate.

- This method will approximate our uncertainty in the parameter around the MAP based on the log-posterior surface **<u>curvature</u>**.

Laplace, Quadratic, or Normal approximation

- Approximate the joint posterior distribution with a **Multivariate Normal** (MVN) centered on the **MAP**.

- Benefits:
  - Straightforward to implement.
  - Relatively fast to execute.
  - Scales to a moderate number of variables.
- Cons:
  - Let’s see with an example later…

First things first…what's a MVN?

- Generalization of the Gaussian distribution to more than 1 dimension.

- Each dimension (variable) is a Gaussian and each subset of variables are MVN.

MVN density function

- There are D elements to the vector of variables:

$$\mathbf{x} = \{x_1, x_2, \ldots x_d, \ldots, x_D\}$$

- IMPORTANT: D refers to the number of variables, NOT the number of observations!

$$p(x_1, x_2, \ldots, x_d | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right\}$$

Break down the terms in the density

$$p(x_1, x_2, \ldots, x_d \,|\, \boxed{\boldsymbol{\mu}}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right\}$$

Vector of means associated with each element in the x-vector:
$\boldsymbol{\mu} = \{\mu_1, \mu_2, \ldots, \mu_d, \ldots, \mu_D\}$

Break down the terms in the density

$$p(x_1, x_2, \ldots, x_d | \boldsymbol{\mu}, \boxed{\boldsymbol{\Sigma}}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right\}$$

$D \times D$ (variance-) covariance matrix between all elements of the x-vector.

Off-diagonal elements store the covariance between the variables.
Main-diagonal elements store the variance of each element in the x-vector

Break down the terms in the density

$$p(x_1, x_2, \ldots, x_d | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D \boxed{|\boldsymbol{\Sigma}|}}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right\}$$

Determinant of the covariance matrix.

Break down the terms in the density

$$p(x_1, x_2, \ldots, x_d | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boxed{\boldsymbol{\Sigma}^{-1}} (\mathbf{x} - \boldsymbol{\mu})\right\}$$

Inverse of the covariance matrix.

Break down the terms in the density

$$p(x_1, x_2, \ldots, x_d | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left\{ -\frac{1}{2} \boxed{(\mathbf{x} - \boldsymbol{\mu})^T} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$

Transpose of the $(\mathbf{x} - \boldsymbol{\mu})$ vector

Break down the terms in the density

$$p(x_1, x_2, \dots, x_d | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}\boxed{(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}\right\}$$

What's this?

Break down the terms in the density

$$p(x_1, x_2, \ldots, x_d | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2} \boxed{(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})}\right\}$$

Multidimensional generalization of the 1-D Gaussian term:

$$\left(\frac{x - \mu}{\sigma}\right)^2$$

Break down the terms in the density

$$p(x_1, x_2, \ldots, x_d | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2} \boxed{(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})}\right\}$$

$\sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})}$ is a generalized distance known as the Mahalanobis distance

Bivariate Gaussian – 2D case

- $D = 2$ the vector of elements becomes: $\mathbf{x} = \{x_1, x_2\}$
- Define the correlation coefficient between the two variables as, ρ.
- The mean vector and covariance matrix are:

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$

Bivariate Gaussian – marginal distributions

- Each variable has a marginal Gaussian distribution:

$$x_1|\mu_1,\sigma_1 \sim \mathrm{normal}(x_1|\mu_1,\sigma_1)$$

$$x_2|\mu_2,\sigma_2 \sim \mathrm{normal}(x_2|\mu_2,\sigma_2)$$

Holds for higher dimensions!

Bivariate Gaussian – conditional distribution

- The conditional distribution of one variable given the other…is also a Gaussian!

$$x_1|x_2, \boldsymbol{\mu}, \boldsymbol{\Sigma} \sim \mathcal{N}\left(\mu_1 + \frac{\sigma_1}{\sigma_2}\rho(x_2 - \mu_2), (1 - \rho^2)\sigma_1^2\right)$$

Holds for higher dimensions!

How can we use a MVN to help us?

- Denote the parameters of interest as $\boldsymbol{\theta}$.

- For the weight example from last week, $\boldsymbol{\theta} = \{\mu, \sigma\}$.

- Assume we can find the posterior **<u>mode</u>**, or MAP, denote as $\widehat{\boldsymbol{\theta}}$.

How can we use a MVN to help us?

- A *second-order Taylor series expansion* of the log-posterior around the posterior mode.

$$\log[p(\boldsymbol{\theta}|\mathbf{x})] \approx \log\left[p\left(\widehat{\boldsymbol{\theta}}|x\right)\right] + \left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)^T \mathbf{g}\Big|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}} + \frac{1}{2}\left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)^T \mathbf{H}\Big|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}} \left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)$$

How can we use a MVN to help us?

- A *second-order Taylor series expansion* of the log-posterior around the posterior mode.

$$\log[p(\boldsymbol{\theta}|\mathbf{x})] \approx \log\left[p\left(\widehat{\boldsymbol{\theta}}|x\right)\right] + \left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)^T \boxed{\mathbf{g}\Big|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}}} + \frac{1}{2}\left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)^T \mathbf{H}\Big|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}} \left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)$$

Gradient vector of the log-posterior with respect to each parameter of interest, evaluated at the posterior mode.

For the weight example: $\mathbf{g} = \left\{\frac{\partial}{\partial\mu}(\log[p(\boldsymbol{\theta}|\mathbf{x})]), \frac{\partial}{\partial\sigma}(\log[p(\boldsymbol{\theta}|\mathbf{x})])\right\}$

How can we use a MVN to help us?

- A *second-order Taylor series expansion* of the log-posterior around the posterior mode.

$$\log[p(\boldsymbol{\theta}|\mathbf{x})] \approx \log\left[p\left(\hat{\boldsymbol{\theta}}|x\right)\right] + \left(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\right)^T \boxed{\mathbf{g}\Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}} + \frac{1}{2}\left(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\right)^T \mathbf{H}\Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \left(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\right)$$

Gradient vector of the log-posterior with respect to each parameter of interest, evaluated at the posterior mode.

At $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$ the gradient, $\mathbf{g}$, by definition, is equal to…

How can we use a MVN to help us?

- A *second-order Taylor series expansion* of the log-posterior around the posterior mode.

$$\log[p(\boldsymbol{\theta}|\mathbf{x})] \approx \log\left[p\left(\widehat{\boldsymbol{\theta}}|x\right)\right] + \left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)^T \boxed{\mathbf{g}\big|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}}} + \frac{1}{2}\left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)^T \mathbf{H}\big|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}} \left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)$$

Gradient vector of the log-posterior with respect to each parameter of interest, evaluated at the posterior mode.

At $\boldsymbol{\theta} = \widehat{\boldsymbol{\theta}}$ the gradient, $\mathbf{g}$, by definition, is equal to…$\mathbf{0}$**!!!!!!!**

The second-order Taylor series expansion simplifies to:

$$\log[p(\boldsymbol{\theta}|\mathbf{x})] \approx \log\left[p\left(\widehat{\boldsymbol{\theta}}|x\right)\right] + \frac{1}{2}\left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)^T \mathbf{H}\Big|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}} \left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)$$

The second-order Taylor series expansion simplifies to:

$$\log[p(\boldsymbol{\theta}|\mathbf{x})] \approx \log\left[p\left(\hat{\boldsymbol{\theta}}|x\right)\right] + \frac{1}{2}\left(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\right)^T \boxed{\mathbf{H}\Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}} \left(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\right)$$

The **Hessian matrix** of the log-posterior with respect to the pair-wise parameter combinations evaluated at the posterior mode.

The Hessian matrix is a matrix of second derivatives and represents the **local curvature** of the surface.

The second-order Taylor series expansion simplifies to:

$$\log[p(\boldsymbol{\theta}|\mathbf{x})] \approx \log\left[p\left(\hat{\boldsymbol{\theta}}|x\right)\right] + \frac{1}{2}\left(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\right)^T \boxed{\mathbf{H}\Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}} \left(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\right)$$

For the weight example, the Hessian matrix is a 2×2 matrix of second derivatives:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2}{\partial \mu^2}(\log[p(\boldsymbol{\theta}|\mathbf{x})]) & \frac{\partial^2}{\partial \mu \partial \sigma}(\log[p(\boldsymbol{\theta}|\mathbf{x})]) \\ \frac{\partial^2}{\partial \sigma \partial \mu}(\log[p(\boldsymbol{\theta}|\mathbf{x})]) & \frac{\partial^2}{\partial \sigma^2}(\log[p(\boldsymbol{\theta}|\mathbf{x})]) \end{bmatrix}$$

What does this expression remind us of…?

$$\log[p(\boldsymbol{\theta}|\mathbf{x})] \approx \log\left[p\left(\widehat{\boldsymbol{\theta}}|x\right)\right] + \frac{1}{2}\left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)^T \mathbf{H}\Big|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}} \left(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\right)$$

What does this expression remind us of…?

$$\log[p(\boldsymbol{\theta}|\mathbf{x})] \approx \log\left[p\left(\hat{\boldsymbol{\theta}}|x\right)\right] + \frac{1}{2}\left(\boldsymbol{\theta} - \boxed{\hat{\boldsymbol{\theta}}}\right)^T \mathbf{H}\Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \left(\boldsymbol{\theta} - \boxed{\hat{\boldsymbol{\theta}}}\right)$$

A vector of mean values!

What does this expression remind us of…?

$$\log[p(\boldsymbol{\theta}|\mathbf{x})] \approx \log\left[p\left(\hat{\boldsymbol{\theta}}|x\right)\right] + \frac{1}{2}\left(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\right)^T \boxed{\mathbf{H}\big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}} \left(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\right)$$

Negative inverse of the covariance matrix!

The negative of the Hessian matrix has a special name…the **observed information matrix**.

The Laplace approximation

$$\boxed{p(\boldsymbol{\theta}|\mathbf{x}) \approx \mathcal{N}\left(\hat{\boldsymbol{\theta}}, \left[-\mathbf{H}\Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}\right]^{-1}\right)}$$

Let's apply the Laplace approximation to our weight example from last week.

- The posterior distribution on μ, σ was proportional to:

$$p(\mu, \sigma | \mathbf{x}) \propto \prod_{n=1}^{N} \{\mathrm{normal}(x_n | \mu, \sigma)\} \cdot \mathrm{normal}(\mu | \mu_0, \tau_0) \cdot \mathrm{uniform}(\sigma | l, u)$$

- Last week we used the following hyperparameters:

$$\mu_0 = 250, \tau_0 = 2$$
$$l = 1, u = 5$$

Let's apply the Laplace approximation to our weight example from last week.

- The posterior distribution on μ, σ was proportional to:

$$p(\mu, \sigma|\mathbf{x}) \propto \prod_{n=1}^{N} \{\mathrm{normal}(x_n|\mu, \sigma)\} \cdot \mathrm{normal}(\mu|\mu_0, \tau_0) \cdot \mathrm{uniform}(\sigma|l, u)$$

- However, this week, increase the upper bound on σ:

$$\mu_0 = 250, \tau_0 = 2$$
$$l = 1, u = 20$$

Last week, we used 10 observations…

- However, this week we will use just the **first observation**, $N = 1$:

$$x_1 = 260.30$$

- Given this single observation, and our prior specification, **what is the posterior joint distribution on $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$**?

Let’s first visualize the log-posterior surface over a fine grid of μ and σ

- Thus, we will repeat the **grid approximation** from last week using our new assumptions and just the single observation.

- Define a function for evaluating the log-posterior.

- Define the following grid: $\mu \in [240, 260]$, $\sigma \in [1, 20]$

The log-posterior:

$$\log[p(\mu, \sigma | \mathbf{x})] \propto \sum_{n=1}^{N} \{\log[\text{normal}(x_n | \mu, \sigma)]\} + \log[\text{normal}(\mu | \mu_0, \tau_0)] + \log[\text{uniform}(\sigma | l, u)]$$

In R, define the function `my_logpost()`

```r
my_logpost <- function(theta, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- theta[1]
  # the unknown standard deviation is the second
  lik_sigma <- theta[2]

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior
  log_lik + log_prior
}
```

In R, define the function `my_logpost()`

The parameters, μ and σ, are defined as the first and second elements of the `theta` vector.

The second argument, `my_info`, is a list which stores all information required to evaluate the log-posterior.

- Contains the vector `xobs` which stores the observations.
- Contains of the hyperparameters to the prior on μ.
- Contains the hyperparameters to the prior on σ.

```r
my_logpost <- function(theta, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- theta[1]
  # the unknown standard deviation is the second
  lik_sigma <- theta[2]

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior
  log_lik + log_prior
}
```

In R, define the function `my_logpost()`

$$\sum_{n=1}^{N} \{\log[\mathrm{normal}(x_n | \mu, \sigma)]\}$$

```r
my_logpost <- function(theta, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- theta[1]
  # the unknown standard deviation is the second
  lik_sigma <- theta[2]

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior
  log_lik + log_prior
}
```

In R, define the function `my_logpost()`

$$\log[\text{normal}(\mu|\mu_0, \tau_0)]$$

```r
my_logpost <- function(theta, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- theta[1]
  # the unknown standard deviation is the second
  lik_sigma <- theta[2]

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior
  log_lik + log_prior
}
```

In R, define the function `my_logpost()`

```r
my_logpost <- function(theta, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- theta[1]
  # the unknown standard deviation is the second
  lik_sigma <- theta[2]

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior
  log_lik + log_prior
}
```

$$\log[\text{uniform}(\sigma|l,u)]$$

Wrap `my_logpost()` in a function to help manage it’s execution.

```r
### create a wrapper function which will allow evaluating the log-posterior
### over a defined grid of parameter values
eval_logpost <- function(mu_val, sigma_val, my_info)
{
  my_logpost(c(mu_val, sigma_val), my_info)
}
```

Create the full-factorial grid of parameter values with the `expand.grid()` function

```
### define a grid of points to use
param_grid <- expand.grid(mu = seq(240, 260, length.out = 201),
                          sigma = seq(info_use$sigma_lwr,
                                      info_use$sigma_upr,
                                      length.out = 201),
                          KEEP.OUT.ATTRS = FALSE,
                          stringsAsFactors = FALSE) %>%
  as.data.frame() %>% tbl_df()

```

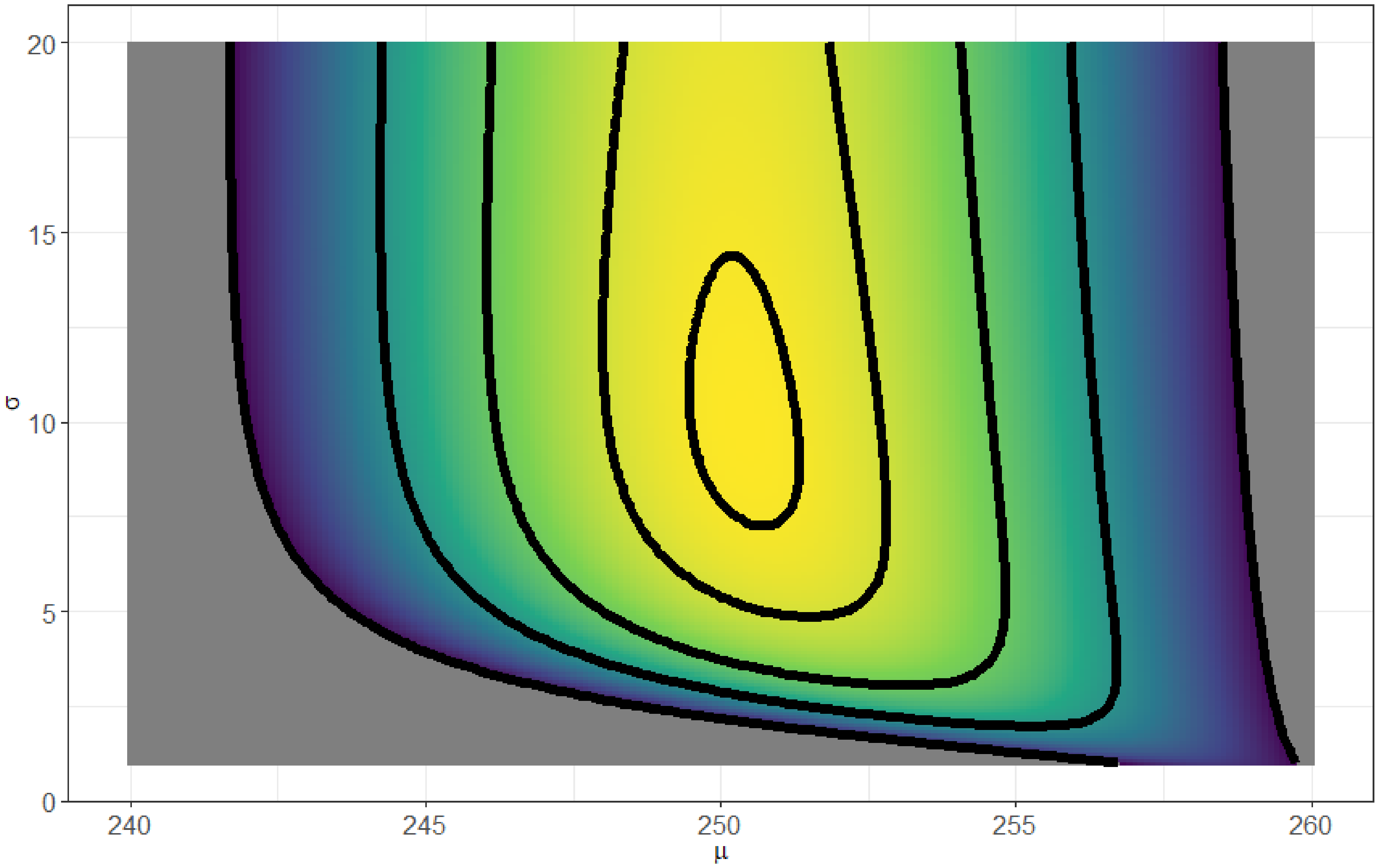
Define the hyperparameters and set the single observation appropriately

```
Nuse <- 1
xuse <- x[1:Nuse]

info_use <- list(
  xobs = xuse,
  mu_0 = 250,
  tau_0 = 2,
  sigma_lwr = 1,
  sigma_upr = 20
)
```

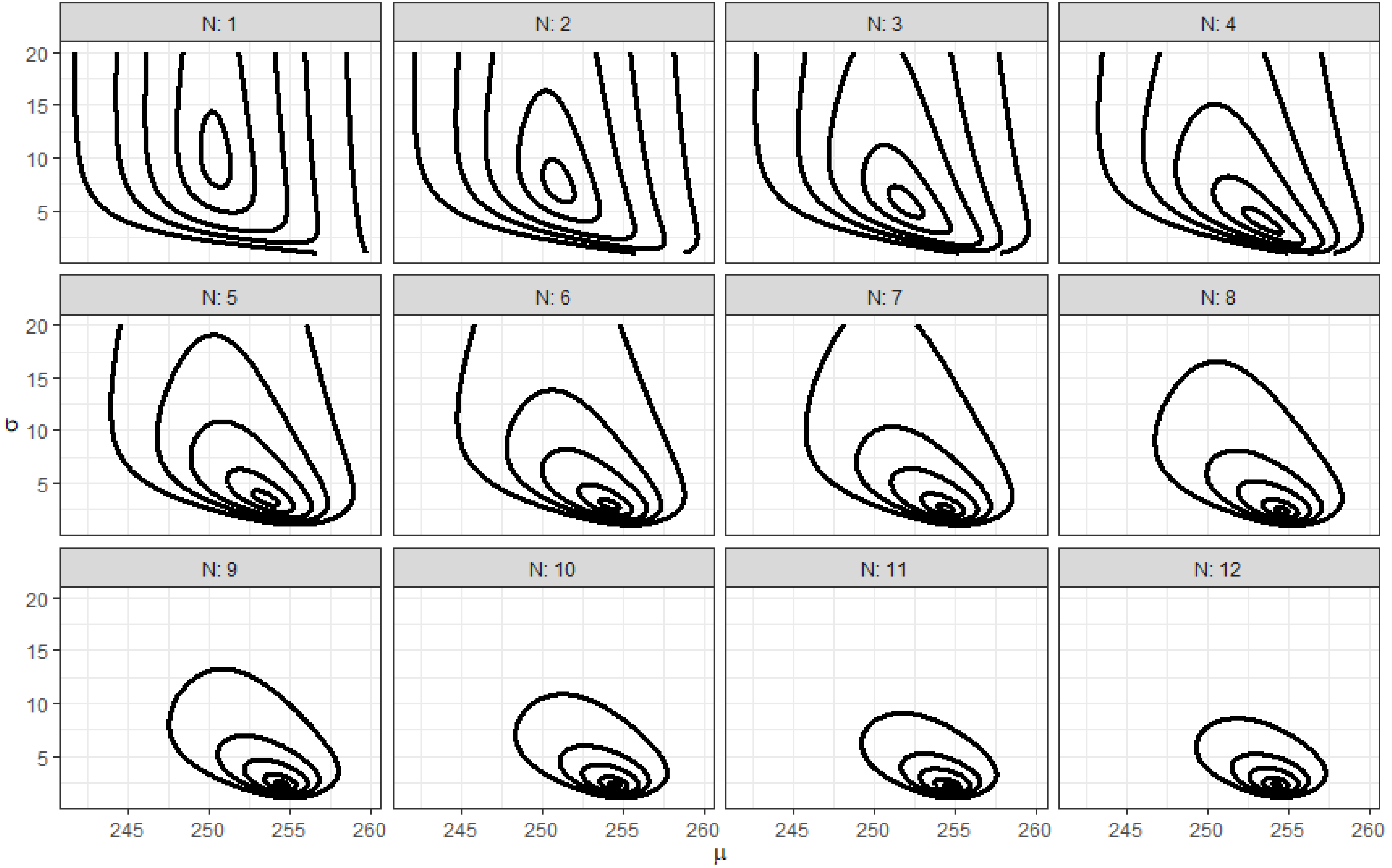
Loop over all parameter pairs with `purrr`

```r
### evaluate the log-posterior over the grid
log_post_result <- purrr::map2_dbl(param_grid$mu,
                                   param_grid$sigma,
                                   eval_logpost,
                                   my_info = info_use)
```

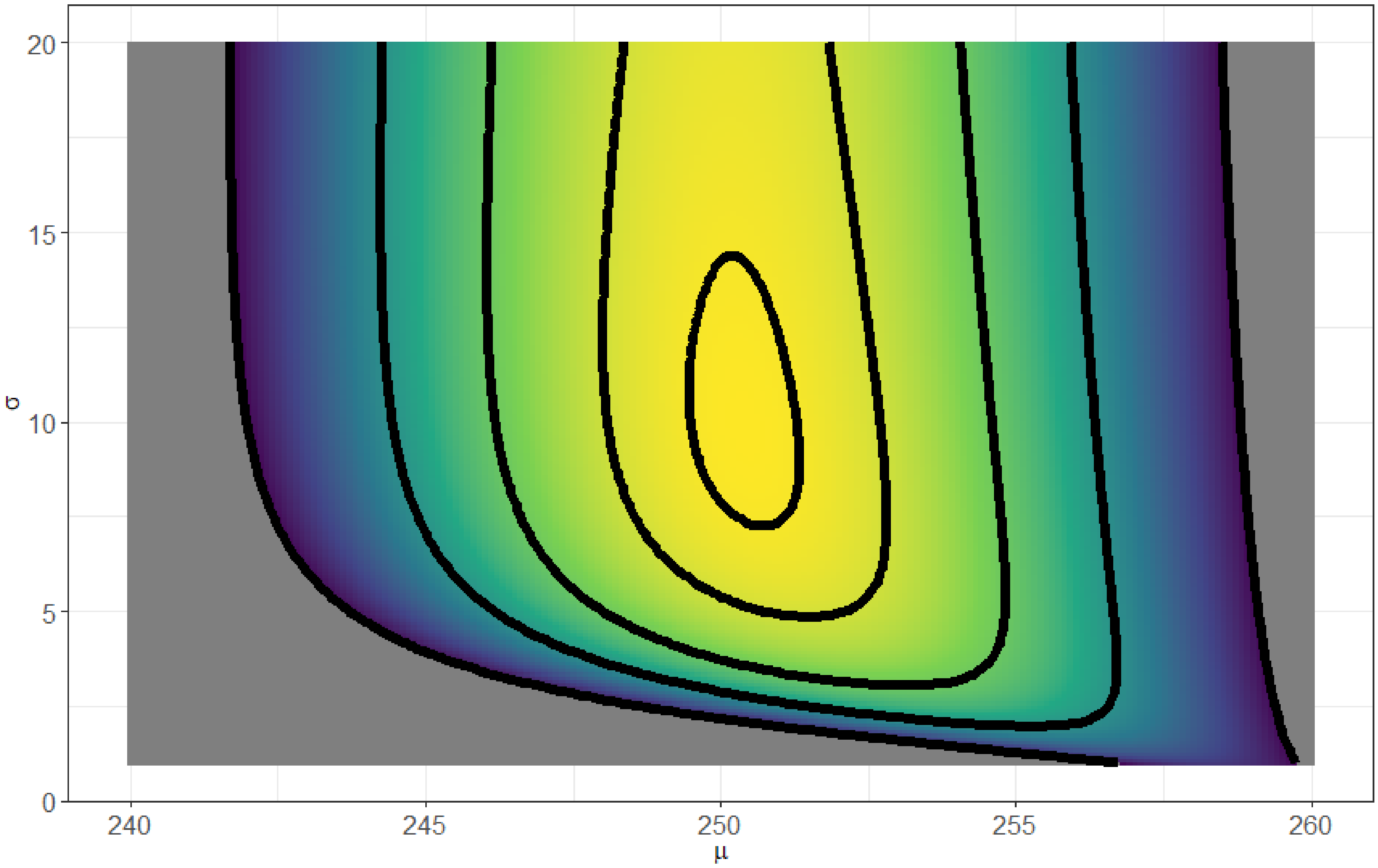
Log-posterior surface contour. Grey areas are μ, σ values with posterior probability of less than 0.01%



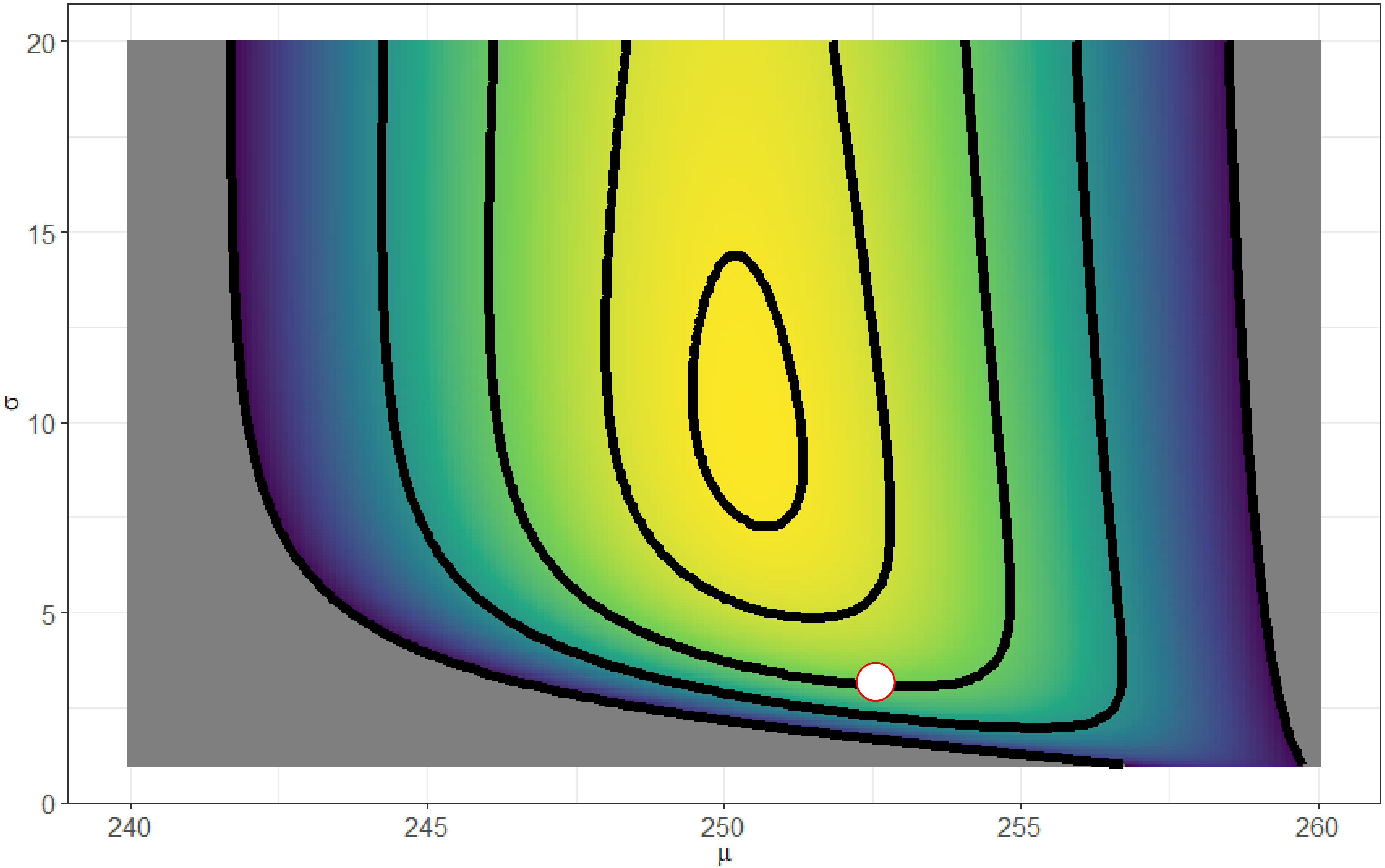
As a check, the log-posterior surface becomes much more concentrated as the sample size increases.



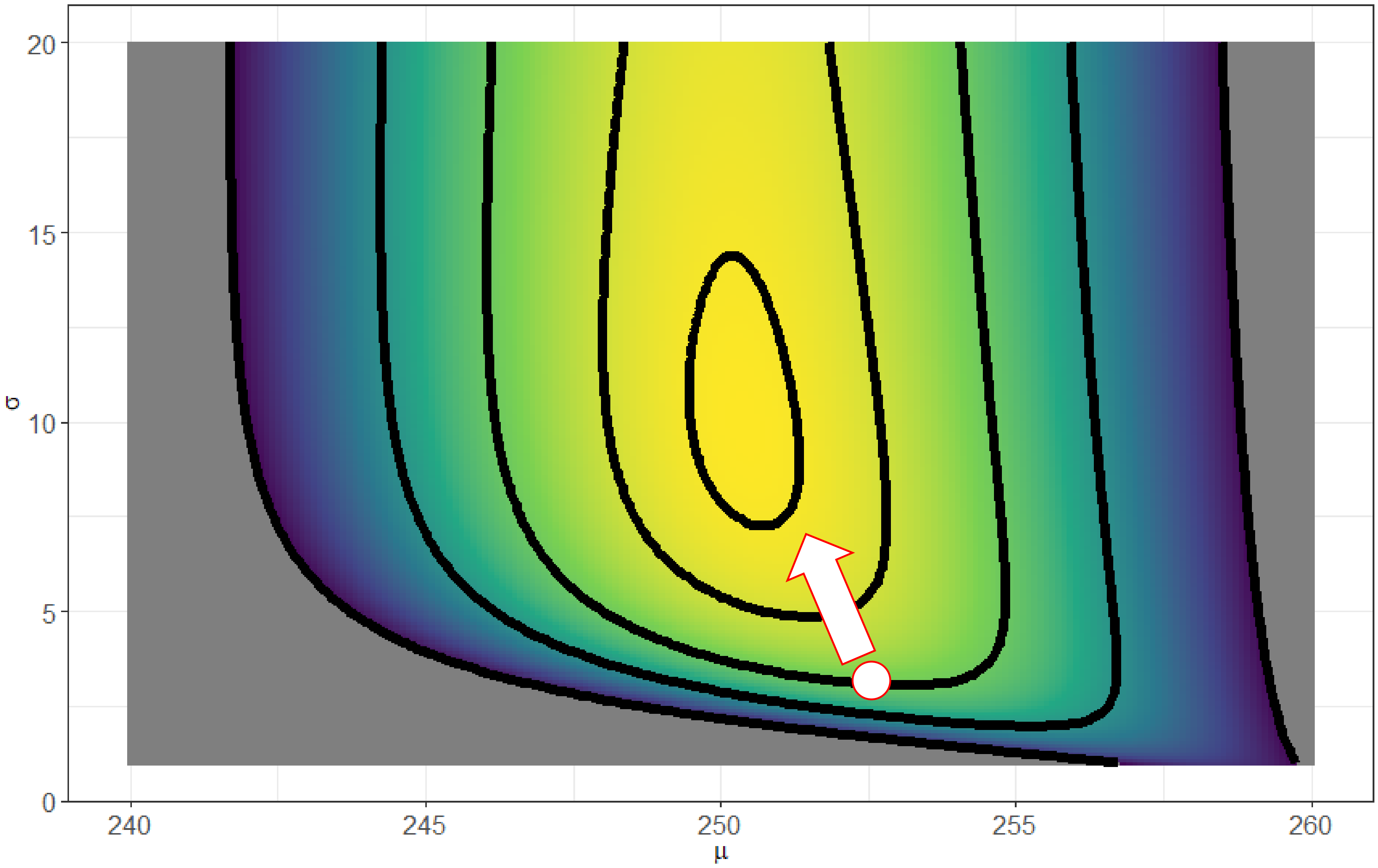
For the $N = 1$ case, how do we find the posterior mode?



Define an initial guess. How should we update our guess?



Steepest ascent! $\Longrightarrow$ Find the direction to the peak!$\Longrightarrow$ Need to calculate the gradient!



How far should we move in the direction defined by the gradient?

- Numerous algorithms exist for selecting the path or search length.
- Newton-Raphson method sets the search length based on the Hessian!
- For the k-th iteration or step, the $k+1$ (new) value is:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \gamma \left[\mathbf{H} \Big|_k \right]^{-1} \mathbf{g} \Big|_k$$

$\gamma \in (0,1)$ is a multiplier
which reduces the step size.

You do **NOT** need to program the optimization routine…

- We will use the `optim()` function in `R` to find the posterior mode.

- There are many different optimization algorithms to choose from in `R`.

- `optim()` is a great starting place for learning about optimization!

`optim()` manages the bookkeeping of the optimization algorithm for us

- Requires that the function we wish to optimize has its **FIRST** argument as a **numeric vector**.

- `optim()` takes care of modifying the elements of that numeric vector in order to optimize the function of interest.

```r
my_logpost <- function(theta, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- theta[1]
  # the unknown standard deviation is the second
  lik_sigma <- theta[2]

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior
  log_lik + log_prior
}
```

`optim()` manages the bookkeeping of the optimization algorithm for us

That is why `theta` is the first argument to `my_logpost()` !!!

- Requires that the function we wish to optimize has its **FIRST** argument as a **numeric vector**.

- `optim()` takes care of modifying the elements of that numeric vector in order to optimize the function of interest.

```r
my_logpost <- function(theta, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- theta[1]
  # the unknown standard deviation is the second
  lik_sigma <- theta[2]

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior
  log_lik + log_prior
}
```

Basic syntax of an `optim()` call.

```r
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using the optimization scheme
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
```

Basic syntax of an `optim()` call.

```r
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using the optimization scheme
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
```

We select an initial guess.

That initial guess is the first argument to `optim()`.

Basic syntax of an `optim()` call.

```r
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using the optimization scheme
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
```

Second argument is the function we wish to optimize.

Basic syntax of an `optim()` call.

```r
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using the optimization scheme
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFGS
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
```

Third argument is a **function** which returns the gradient vector. If we require the gradient to be evaluated **numerically**, set to `NULL`

Basic syntax of an `optim()` call.

```r
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using the optimization scheme
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFG
                    hessian = TRU
                    control = list(fnscale = -1, maxit = 1001))
```

After the `gr` argument, we can pass in **ALL additional inputs** required to evaluate the function we wish to optimize.

Basic syntax of an `optim()` call.

```r
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale =
```

```r
my_logpost <- function(theta, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- theta[1]
  # the unknown standard deviation is the second
  lik_sigma <- theta[2]

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior
  log_lik + log_prior
}
```

Basic syntax of an `optim()` call.

After setting the `gr` argument, all remaining arguments <u>**MUST**</u> be named.

```
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using the optimization scheme
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
```

Basic syntax of an `optim()` call.

After setting the `gr` argument, all remaining arguments **MUST** be named.

```
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using the optimization scheme
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
```

Set which optimization method to use. By default `optim()` uses Nelder-Mead. I like to use the Quasi-Newton BFGS algorithm. Try out different methods, see which one you prefer.

Basic syntax of an `optim()` call.

After setting the `gr` argument, all remaining arguments <u>**MUST**</u> be named.

```r
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using the optimization scheme
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
```

Tell `optim()` to compute and return the hessian matrix by setting the `hessian` flag to TRUE.

Basic syntax of an `optim()` call.

After setting the `gr` argument, all remaining arguments **MUST** be named.

```r
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using the optimization scheme
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
```

`control` is a list of parameters which dictate important operating behavior of the algorithm.
See `?optim` for a complete list of all available control parameters.

Basic syntax of an `optim()` call.

After setting the `gr` argument, all remaining arguments **MUST** be named.

```r
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using the optimization scheme
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
```

`control` is a list of parameters which dictate important operating behavior of the algorithm.
See `?optim` for a complete list of all available control parameters.

By default `optim()` seeks to **MINIMIZE** a function, so to tell it to **MAXIMIZE** set the `fnscale` control parameter to `-1`.

Basic syntax of an `optim()` call.

After setting the `gr` argument, all remaining arguments **MUST** be named.

```r
### define a starting guess to try
init_param <- c(info_use$mu_0, 5)

### find the posterior mode (the MAP) using the optimization scheme
map_result <- optim(init_param,
                    my_logpost,
                    gr = NULL,
                    info_use,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
```

`control` is a list of parameters which dictate important operating behavior of the algorithm.
See `?optim` for a complete list of all available control parameters.

`maxit` controls the maximum number of iterations, default for **derivative** based method is 100.

`optim()` result

```
> map_result
$par
[1] 250.404541   9.892691

$value
[1] -8.288217

$counts
function gradient
      20       18

$convergence
[1] 0

$message
NULL

$hessian
            [,1]        [,2]
[1,] -0.26021812 -0.02044647
[2,] -0.02044647 -0.02046694
```

To complete the Laplace approximation…

- Define a wrapper which executes the `optim()` call and then calculates the remaining pieces of the Laplace approximation.

- The following code is adapted from the `laplace()` function from the `LearnBayes` package by Jim Albert.

- Great book! Bayesian Computation with R

Can you decipher what's happening in this function?

```r
### define a function for performing the laplace approximation
my_laplace <- function(start_guess, logpost_func, ...)
{
  # code adapted from the `LearnBayes`` function `laplace()`
  fit <- optim(start_guess,
               logpost_func,
               gr = NULL,
               ...,
               method = "BFGS",
               hessian = TRUE,
               control = list(fnscale = -1, maxit = 1001))

  mode <- fit$par
  h <- -solve(fit$hessian)
  p <- length(mode)
  int <- p/2 * log(2 * pi) + 0.5 * log(det(h)) + logpost_func(mode, ...)
  list(mode = mode,
       var_matrix = h,
       log_evidence = int,
       converge = ifelse(fit$convergence == 0,
                         "YES",
                         "NO"),
       iter_counts = fit$counts[1])
}
```

Can you decipher what's happening in this function?

```r
### define a function for performing the laplace approximation
my_laplace <- function(start_guess, logpost_func, ...)
{
  # code adapted from the `LearnBayes`` function `laplace()`
  fit <- optim(start_guess,
               logpost_func,
               gr = NULL,
               ...,
               method = "BFGS",
               hessian = TRUE,


  mode <- f
  h <- -sol
  p <- length(mode)
  int <- p/2 * log(2 * pi) + 0.5 * log(det(h)) + logpost_func(mode, ...)
  list(mode = mode,
       var_matrix = h,
       log_evidence = int,
       converge = ifelse(fit$convergence == 0,
                         "YES",
                         "NO"),
       iter_counts = fit$counts[1])
}
```

We will return to this aspect of the Laplace approximation in a future lecture!

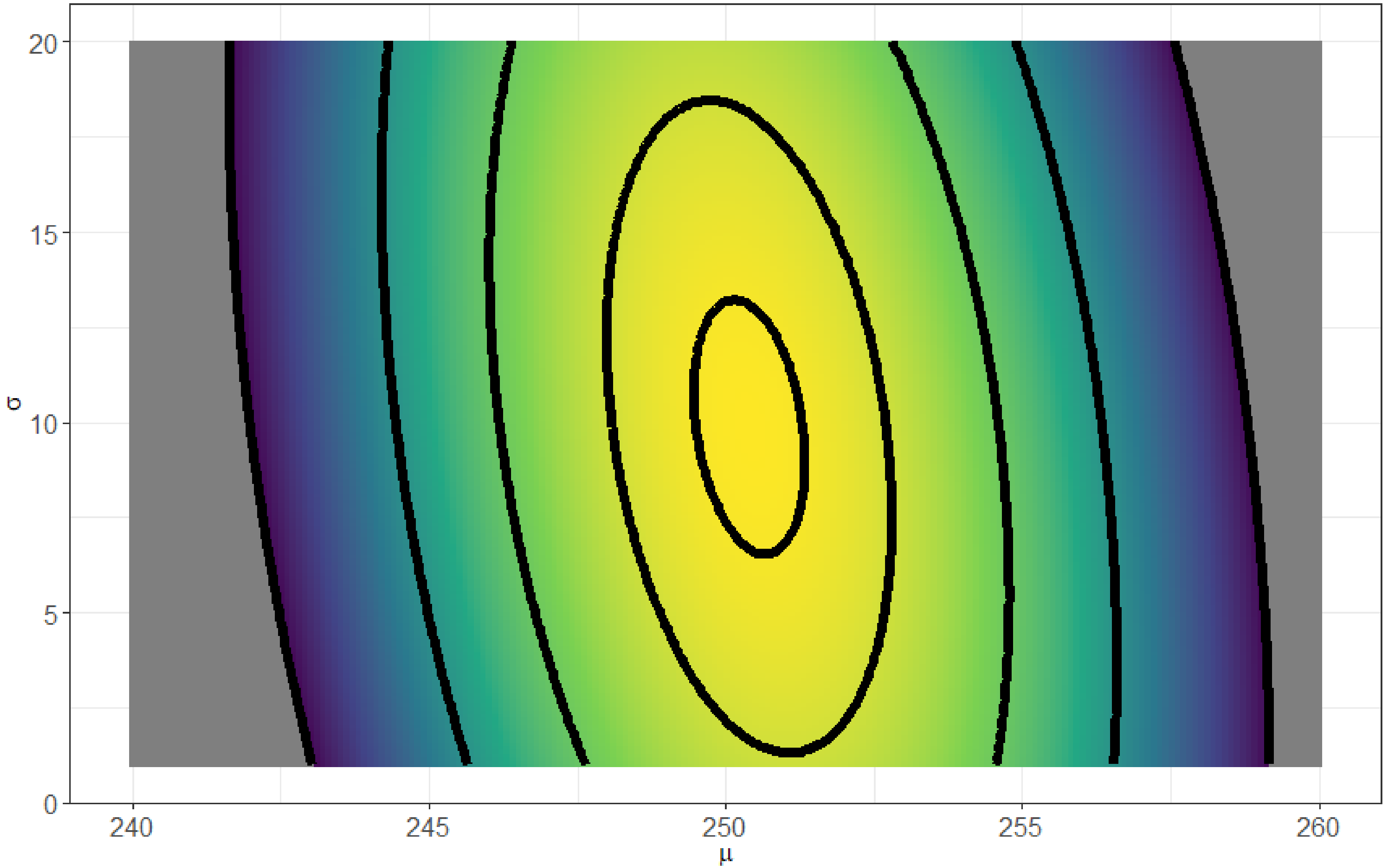
Perform the Laplace approximation

```
> laplace_result_N01 <- my_laplace(init_param, my_logpost, info_use)
> laplace_result_N01
$mode
[1] 250.404541   9.892691

$var_matrix
          [,1]      [,2]
[1,]  4.170279 -4.166109
[2,] -4.166109 53.021223

$log_evidence
[1] -3.791876

$converge
[1] "YES"

$iter_counts
function
      20
```

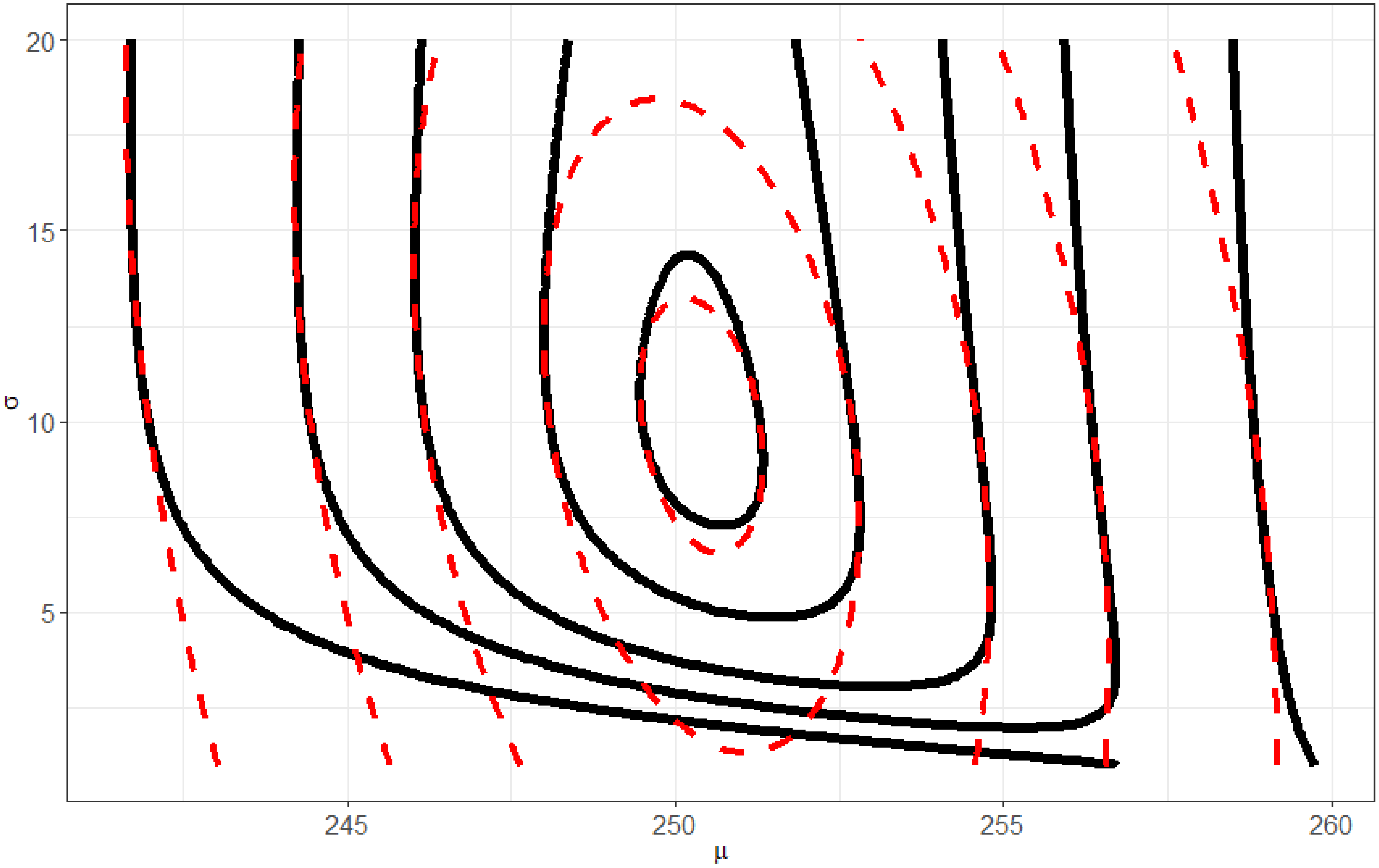
WHAT IS THIS????????

```
> laplace_result_N01 <- my_laplace(init_param, my_logpost, info_use)
> laplace_result_N01
$mode
[1] 250.404541   9.892691

$var_matrix
          [,1]      [,2]
[1,]  4.170279 -4.166109
[2,] -4.166109 53.021223

$log_evidence
[1] -3.791876

$converge
[1] "YES"

$iter_counts
function
      20
```

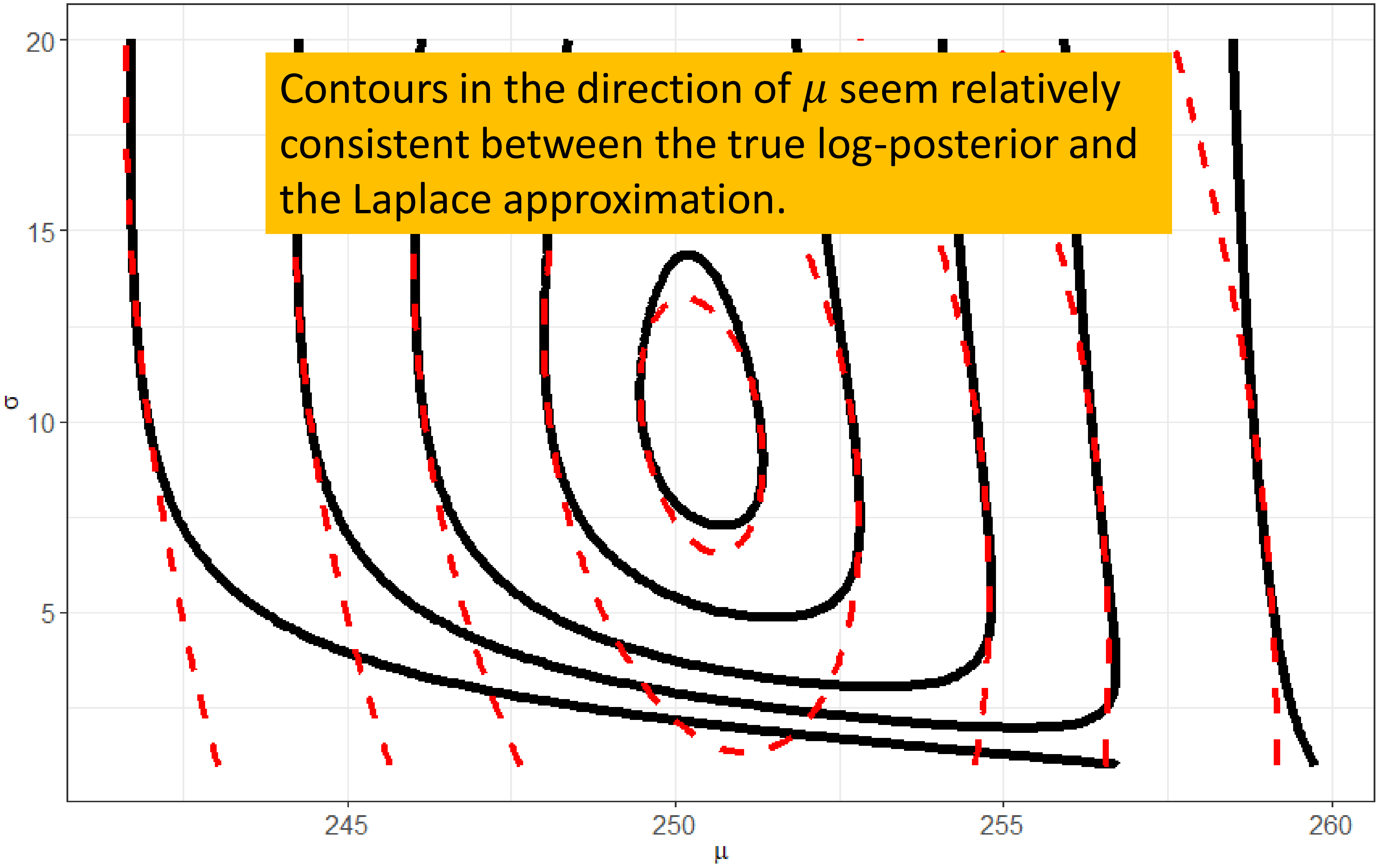
Approximate log-posterior based on the MVN Laplace approximation.



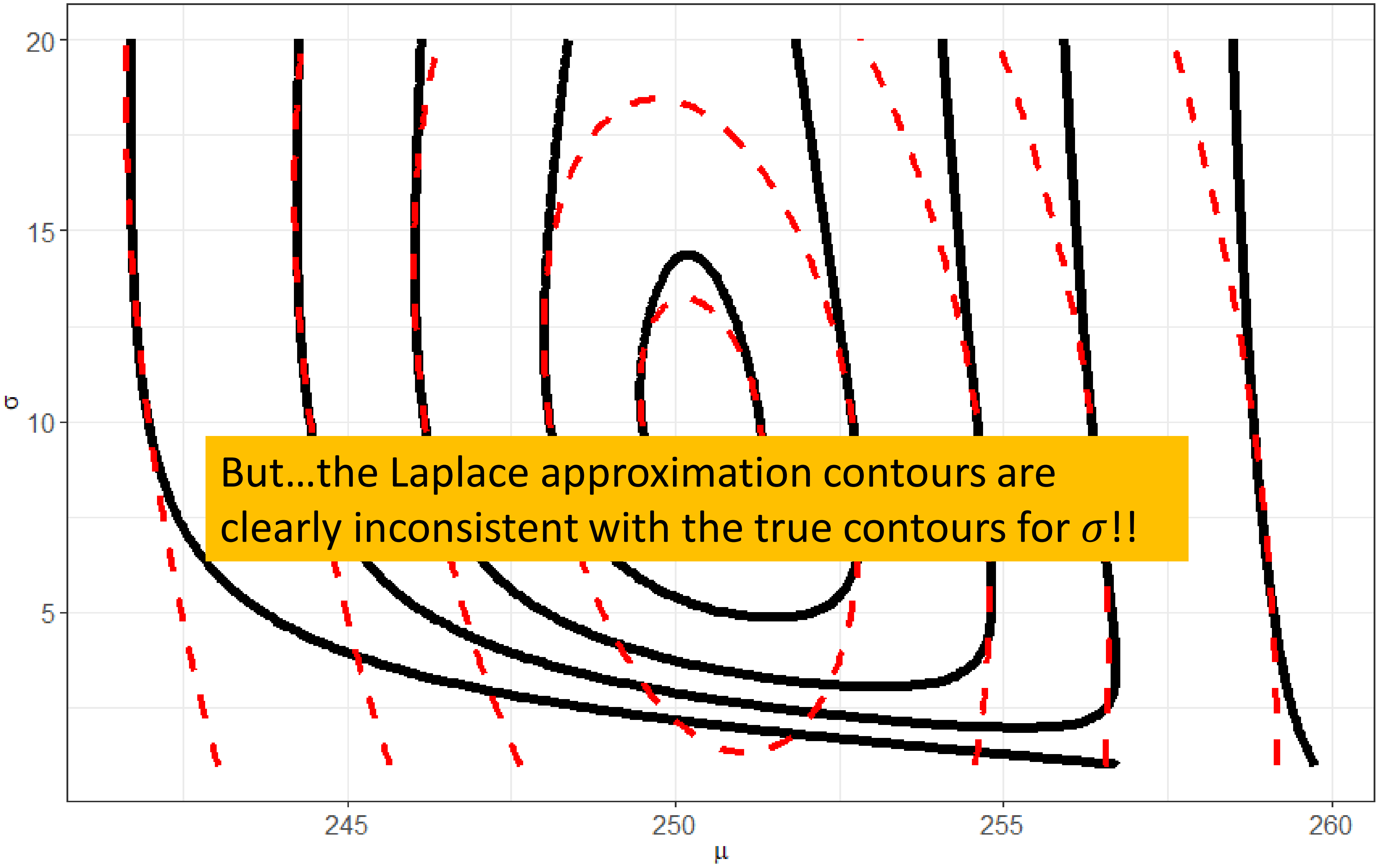
Compare true log-posterior surface (black) with the MVN approximate log-posterior (dashed red)



Compare true log-posterior surface (black) with the MVN approximate log-posterior (dashed red)



Compare true log-posterior surface (black) with the MVN approximate log-posterior (dashed red)



Confirm similarities and differences by drawing samples and visualizing results

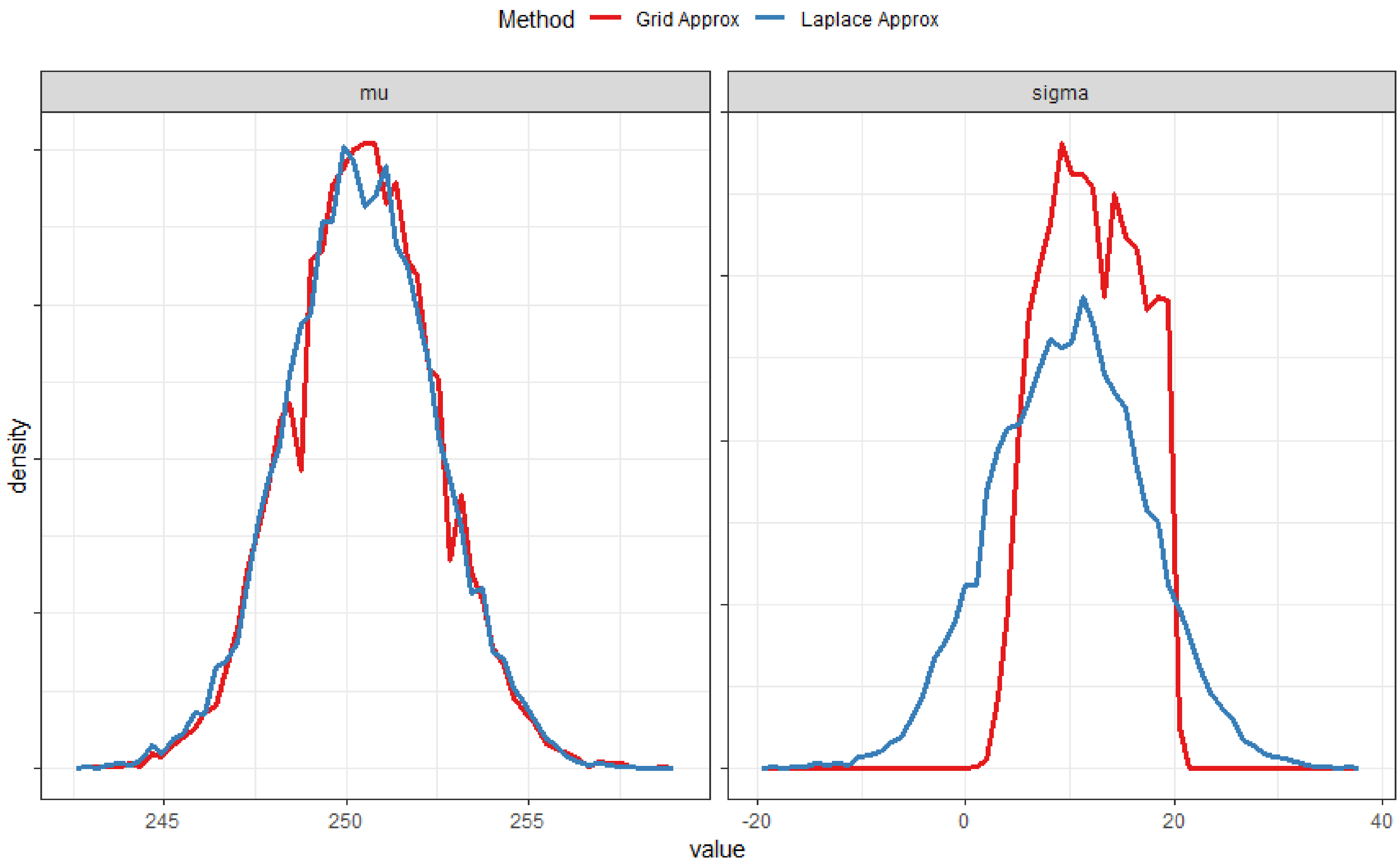
```r
### random draws from the approximate MVN posterior
set.seed(5002)
post_mvn_samples <- MASS::mvrnorm(n = 1e4,
                                  mu = laplace_result_N01$mode,
                                  Sigma = laplace_result_N01$var_matrix) %>%
  as.data.frame() %>% tbl_df() %>%
  purrr::set_names(c("mu", "sigma"))

post_mvn_samples

### draw samples from the grid approximation directly to compare to the
### approximate MVN posterior samples

grid_approx_result <- param_grid %>%
  mutate(log_post = log_post_result) %>%
  mutate(log_post_2 = log_post - max(log_post))

set.seed(5003)
direct_sample_id <- sample(1:nrow(param_grid),
                           size = 1e4,
                           replace = TRUE,
                           prob = exp(grid_approx_result$log_post_2))

grid_approx_samples <- grid_approx_result %>%
  slice(direct_sample_id)
```

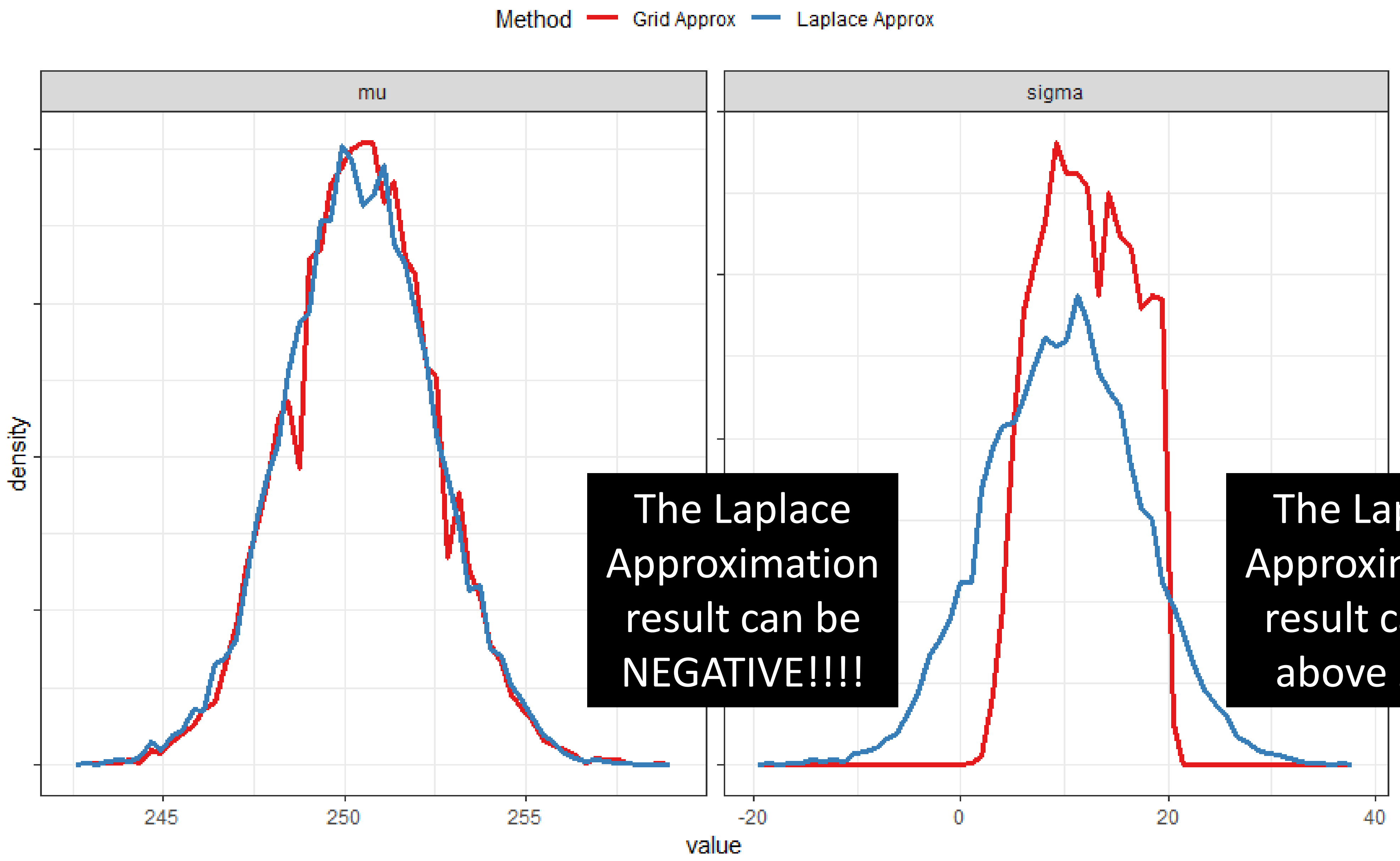
Confirm similarities and differences by drawing samples and visualizing results

```r
### random draws from the approximate MVN posterior
set.seed(5002)
post_mvn_samples <- MASS::mvrnorm(n = 1e4,
                                  mu = laplace_result_N01$mode,
                                  Sigma = laplace_result_N01$var_matrix) %>%
  as.data.frame() %>% tbl_df() %>%
  purrr::set_names(c("mu", "sigma"))

post_mvn_samples

### draw samples from the grid approximation directly to compare to the
### approximate MVN posterior samples

grid_approx_result <- param_grid %>%
  mutate(log_post = log_post_result) %>%
  mutate(log_post_2 = log_post - max(log_post))

set.seed(5003)
direct_sample_id <- sample(1:nrow(param_grid),
                           size = 1e4,
                           replace = TRUE,
                           prob = exp(grid_approx_result$log_post_2))

grid_approx_samples <- grid_approx_result %>%
  slice(direct_sample_id)
```

Samples based on the approximate MVN distribution directly.

Sample with replacement with row weights equal to the posterior probability of that row.


Method
Grid Approx
Laplace Approx
mu
sigma
density
245
250
255
-20
0
20
40
value



The Laplace Approximation result can be NEGATIVE!!!!

The Laplace Approximation result can be above 20!!!

What's going on?

- Each variable in a Multivariate Normal distribution has a Gaussian distribution.

- Gaussian variables are unbounded: $-\infty \rightarrow +\infty$ are allowed!

- The natural lower 0 bound on σ is therefore ignored, as is the upper bound imposed by the UNIFORM prior.

How can we have the imposed UNIFORM constraint satisfied?

- Apply a transformation!

- Use the change-of-variables procedure to transform from the bounded σ to a new unbounded variable ϕ.

$$\sigma = g^{-1}(\phi), \phi = g(\sigma)$$

- Apply the Laplace approximation to the joint posterior distribution on μ, ϕ since both are unbounded.

With the change-of-variables procedure we can still use our original prior on $\boldsymbol{\sigma}$

The joint-posterior on μ, ϕ can therefore be written based on the joint posterior μ, σ:

$$p(\mu, \phi|\mathbf{x}) = \prod_{n=1}^{N} \left\{ p\left(x_n|\mu, g^{-1}(\phi)\right) \right\} p(\mu|\mu_0, \tau_0) p(g^{-1}(\phi)|l, u) \cdot \left| \frac{d}{d\phi}\left(g^{-1}(\phi)\right) \right|$$

With the change-of-variables procedure we can still use our original prior on $\boldsymbol{\sigma}$

The joint-posterior on μ, ϕ can therefore be written based on the joint posterior μ, σ:

$$p(\mu, \phi | \mathbf{x}) = \underbrace{\prod_{n=1}^{N} \left\{ p\left(x_n | \mu, g^{-1}(\phi)\right)\right\} p(\mu | \mu_0, \tau_0) p(g^{-1}(\phi) | l, u)}_{\text{“Back-transform” } \phi \text{ to } \sigma \text{ and substitute into the joint-posterior just as before when we were working with } \sigma \text{ directly.}} \cdot \left| \frac{d}{d\phi} \left(g^{-1}(\phi)\right) \right|$$

“Back-transform” ϕ to σ and substitute into the joint-posterior just as before when we were working with σ directly.

With the change-of-variables procedure we can still use our original prior on $\boldsymbol{\sigma}$

The joint-posterior on μ, ϕ can therefore be written based on the joint posterior μ, σ:

$$p(\mu, \phi|\mathbf{x}) = \prod_{n=1}^{N} \{p\left(x_n|\mu, g^{-1}(\phi)\right)\} p(\mu|\mu_0, \tau_0) p(g^{-1}(\phi)|l, u) \cdot \left|\frac{d}{d\phi}\left(g^{-1}(\phi)\right)\right|$$

Derivative of the transformation with respect to ϕ

Use a logit-transformation to satisfy the lower and upper bounds on σ

- The logit, or log-odds transformation maps a lower and upper bounded variable to an unbounded variable.

$$\phi = \text{logit}(\psi) = \log\left[\frac{\psi}{1-\psi}\right]$$

$$\psi = \text{logit}^{-1}(\phi) = \text{logistic}(\phi) = \frac{1}{1+\exp(-\phi)}$$

- **We will discuss the logit-transformation in more detail when we talk about classification.**

Use a logit-transformation to satisfy the lower and upper bounds on σ

Transformation

$$\phi = \mathrm{logit}\left(\frac{\sigma - l}{u - l}\right) = g(\sigma)$$

Inverse transformation

$$\sigma = l + (u - l) \cdot \mathrm{logit}^{-1}(\phi) = g^{-1}(\phi)$$

Derivative

$$\frac{d\sigma}{d\phi} = (u - l) \cdot \mathrm{logit}^{-1}(\phi) \cdot \left(1 - \mathrm{logit}^{-1}(\phi)\right)$$

Define a new function `my_logpost_cv()` which calculates the log-posterior on μ, ϕ

```r
my_logpost_cv <- function(phi, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- phi[1]
  # the unknown logit-transformed standard deviation
  # is the second, back transform to sigma
  lik_sigma <- my_info$sigma_lwr +
    (my_info$sigma_upr - my_info$sigma_lwr) * boot::inv.logit(phi[2])

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior and account
  # for the derivative adjustment
  deriv_adjust <- log(my_info$sigma_upr - my_info$sigma_lwr) +
    log(boot::inv.logit(phi[2])) +
    log(1 - boot::inv.logit(phi[2]))

  log_lik + log_prior + deriv_adjust
}
```

The log-likelihood and log-priors are evaluated just as they were in the original function!

```r
my_logpost <- function(theta, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- theta[1]
  # the unknown standard deviation is the second
  lik_sigma <- theta[2]

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior
  log_lik + log_prior
}
```

```r
my_logpost_cv <- function(phi, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- phi[1]
  # the unknown logit-transformed standard deviation
  # is the second, back transform to sigma
  lik_sigma <- my_info$sigma_lwr +
    (my_info$sigma_upr - my_info$sigma_lwr) * boot::inv.logit(phi[2])

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior and account
  # for the derivative adjustment
  deriv_adjust <- log(my_info$sigma_upr - my_info$sigma_lwr) +
    log(boot::inv.logit(phi[2])) +
    log(1 - boot::inv.logit(phi[2]))

  log_lik + log_prior + deriv_adjust
}
```

The new function “back-transforms” ϕ to σ

```r
my_logpost <- function(theta, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- theta[1]
  # the unknown standard deviation is the second
  lik_sigma <- theta[2]

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior
  log_lik + log_prior
}
```

```r
my_logpost_cv <- function(phi, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- phi[1]
  # the unknown logit-transformed standard deviation
  # is the second, back transform to sigma
  lik_sigma <- my_info$sigma_lwr +
    (my_info$sigma_upr - my_info$sigma_lwr) * boot::inv.logit(phi[2])

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior and account
  # for the derivative adjustment
  deriv_adjust <- log(my_info$sigma_upr - my_info$sigma_lwr) +
    log(boot::inv.logit(phi[2])) +
    log(1 - boot::inv.logit(phi[2]))

  log_lik + log_prior + deriv_adjust
}
```

The new function calculates the log-derivative of σ with respect to ϕ

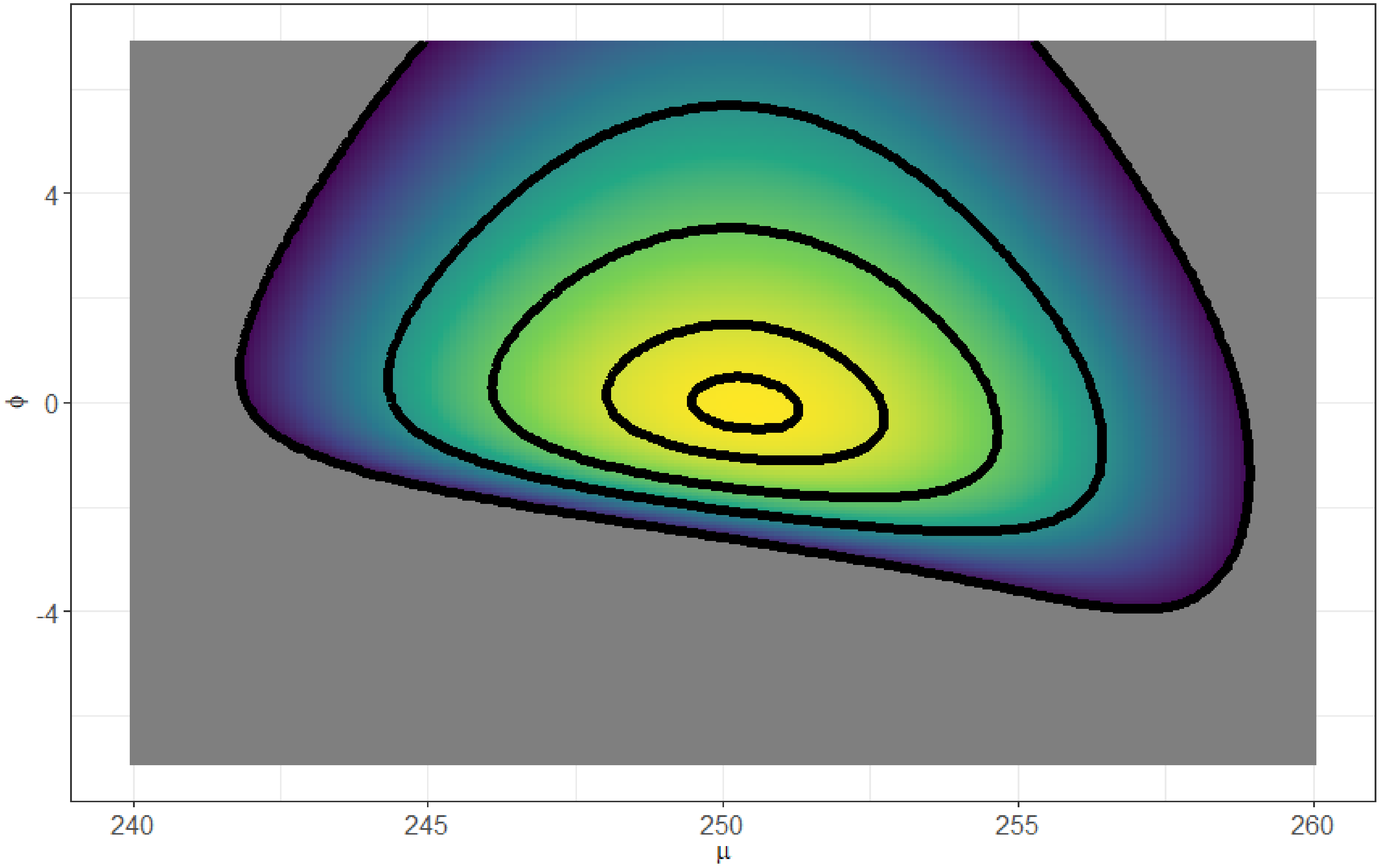
```r
my_logpost <- function(theta, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- theta[1]
  # the unknown standard deviation is the second
  lik_sigma <- theta[2]

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior
  log_lik + log_prior
}
```

```r
my_logpost_cv <- function(phi, my_info)
{
  # the unknown mean is the first parameter
  lik_mu <- phi[1]
  # the unknown logit-transformed standard deviation
  # is the second, back transform to sigma
  lik_sigma <- my_info$sigma_lwr +
    (my_info$sigma_upr - my_info$sigma_lwr) * boot::inv.logit(phi[2])

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

  # the log-prior -> sum up the independent priors
  log_prior <- dnorm(x = lik_mu,
                     mean = my_info$mu_0,
                     sd = my_info$tau_0,
                     log = TRUE) +
    dunif(x = lik_sigma,
          min = my_info$sigma_lwr,
          max = my_info$sigma_upr,
          log = TRUE)

  # add the log-likelihood and log-prior and account
  # for the derivative adjustment
  deriv_adjust <- log(my_info$sigma_upr - my_info$sigma_lwr) +
    log(boot::inv.logit(phi[2])) +
    log(1 - boot::inv.logit(phi[2]))

  log_lik + log_prior + deriv_adjust
}
```

Let's first visualize the log-posterior for the unbounded parameters μ, ϕ

```r
### define a new grid within the unbounded parameter
### space, and evaluate the (mu,phi) log-posterior on this
### new grid
phi_grid <- expand.grid(mu = seq(240, 260, length.out = 201),
                        phi = seq(boot::logit(1e-3),
                                  boot::logit(0.999),
                                  length.out = 201),
                        KEEP.OUT.ATTRS = FALSE,
                        stringsAsFactors = FALSE) %>%
  as.data.frame() %>% tbl_df()
```

Log-posterior for the unbounded parameters μ, ϕ



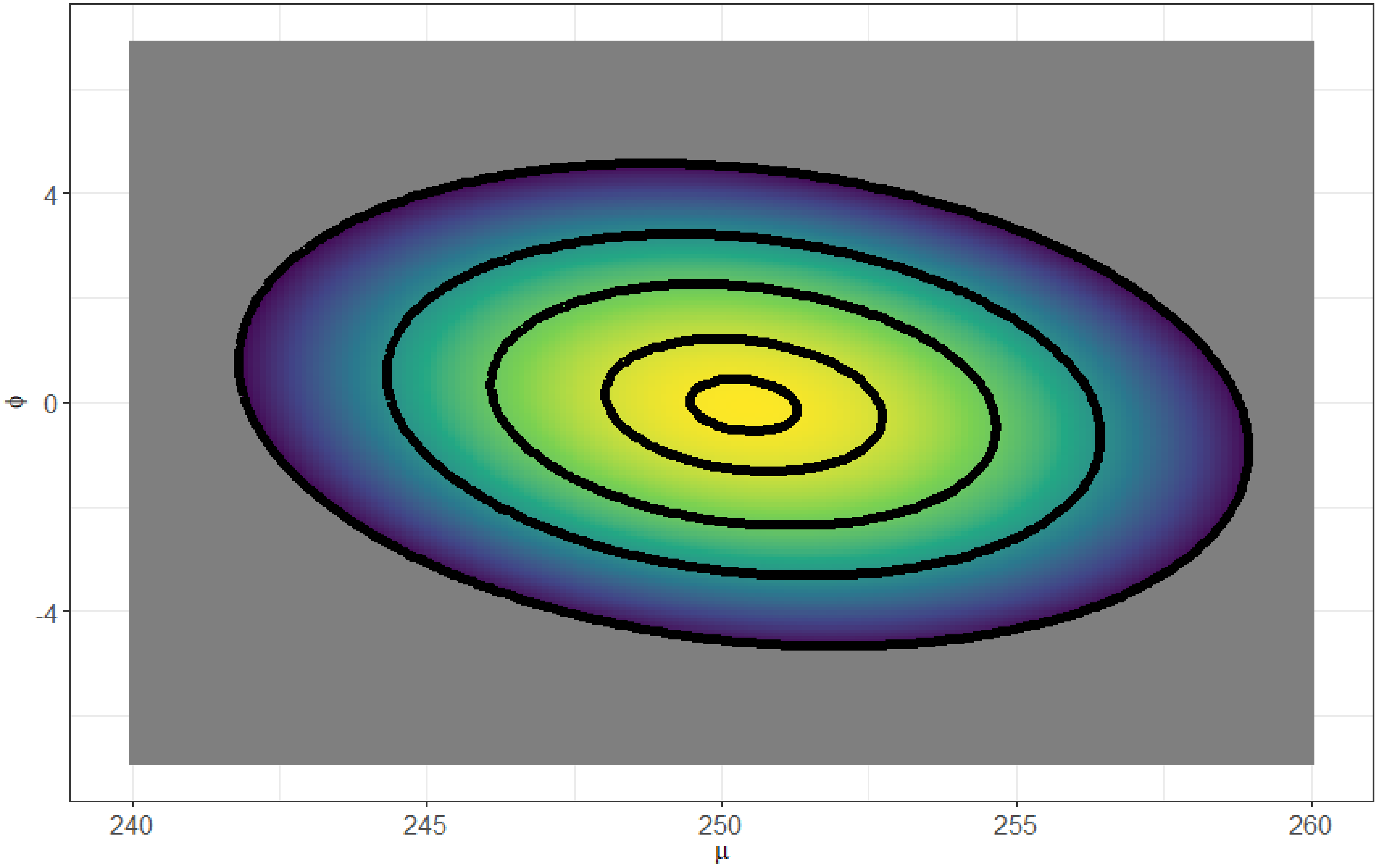
Perform the Laplace approximation within the unbounded domain

```
> cv_laplace_result_N01 <- my_laplace(c(250, log(5)), my_logpost_cv, info_use)
> cv_laplace_result_N01
$mode
[1] 250.37888274  -0.05571467

$var_matrix
           [,1]       [,2]
[1,]  3.9857940 -0.3926613
[2,] -0.3926613  1.1601275

$log_evidence
[1] -4.14531

$converge
[1] "YES"

$iter_counts
function
      18
```

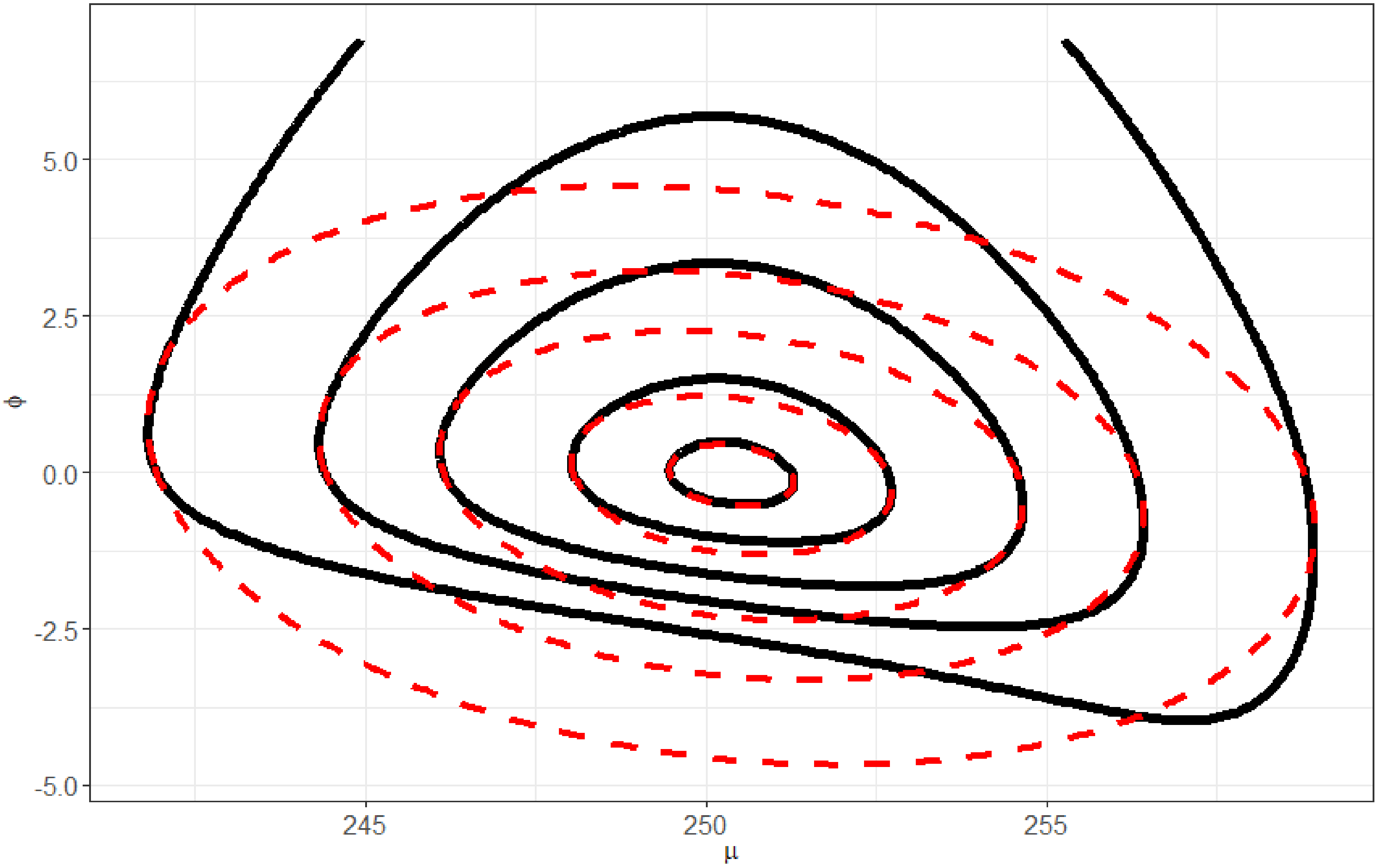
Perform the Laplace approximation within the unbounded domain

```
> cv_laplace_result_N01 <- my_laplace(c(250, log(5)), my_logpost_cv, info_use)
> cv_laplace_result_N01
$mode
[1] 250.37888274  -0.05571467

$var_matrix
           [,1]       [,2]
[1,]  3.9857940 -0.3926613
[2,] -0.3926613  1.1601275

$log_evidence
[1] -4.14531

$converge
[1] "YES"

$iter_counts
function
      18
```

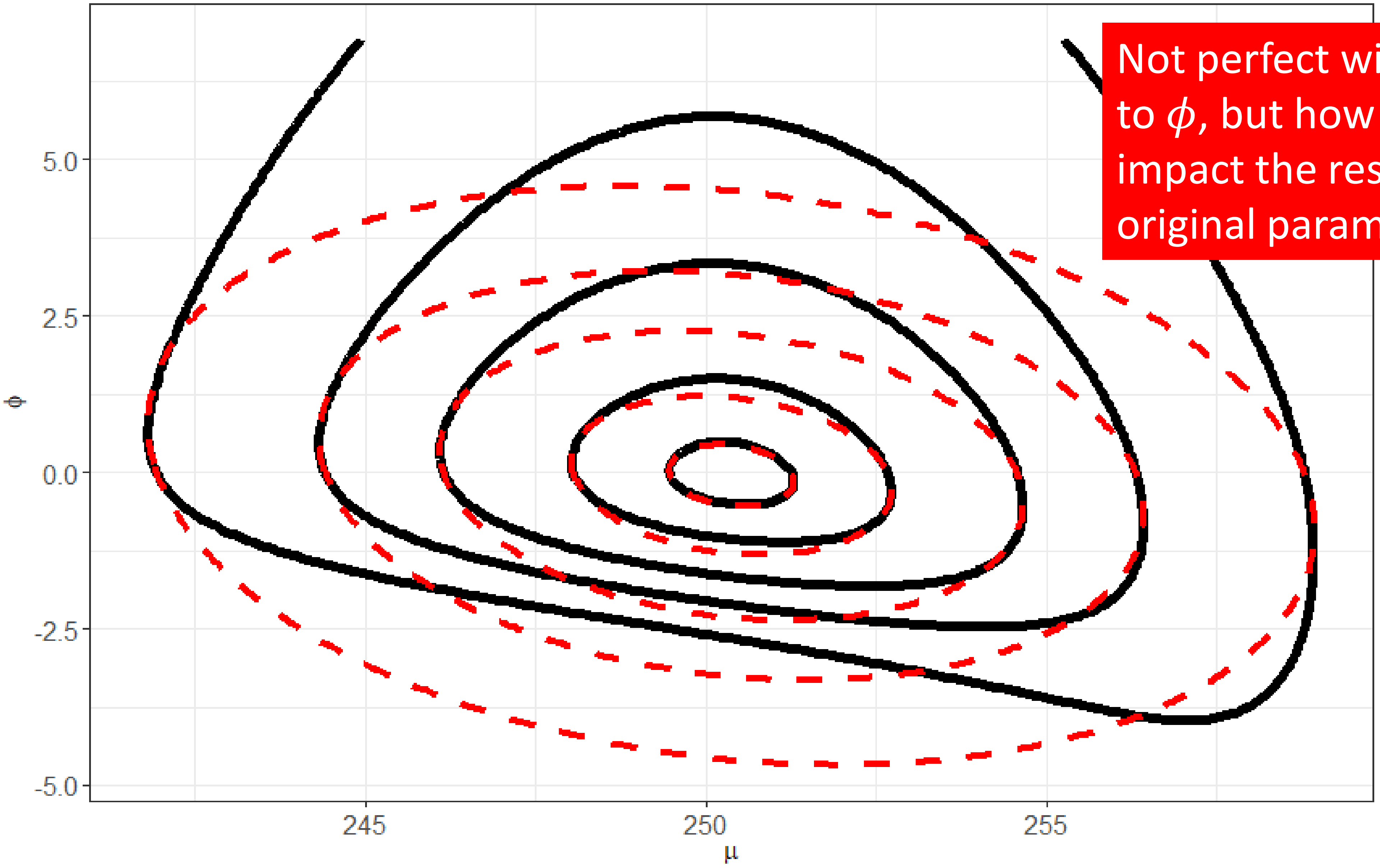
Approximate log-posterior based on the MVN Laplace approximation in the unbounded parameter space.



Compare true log-posterior surface (black) with the MVN approximate log-posterior (dashed red)



Compare true log-posterior surface (black) with the MVN approximate log-posterior (dashed red)



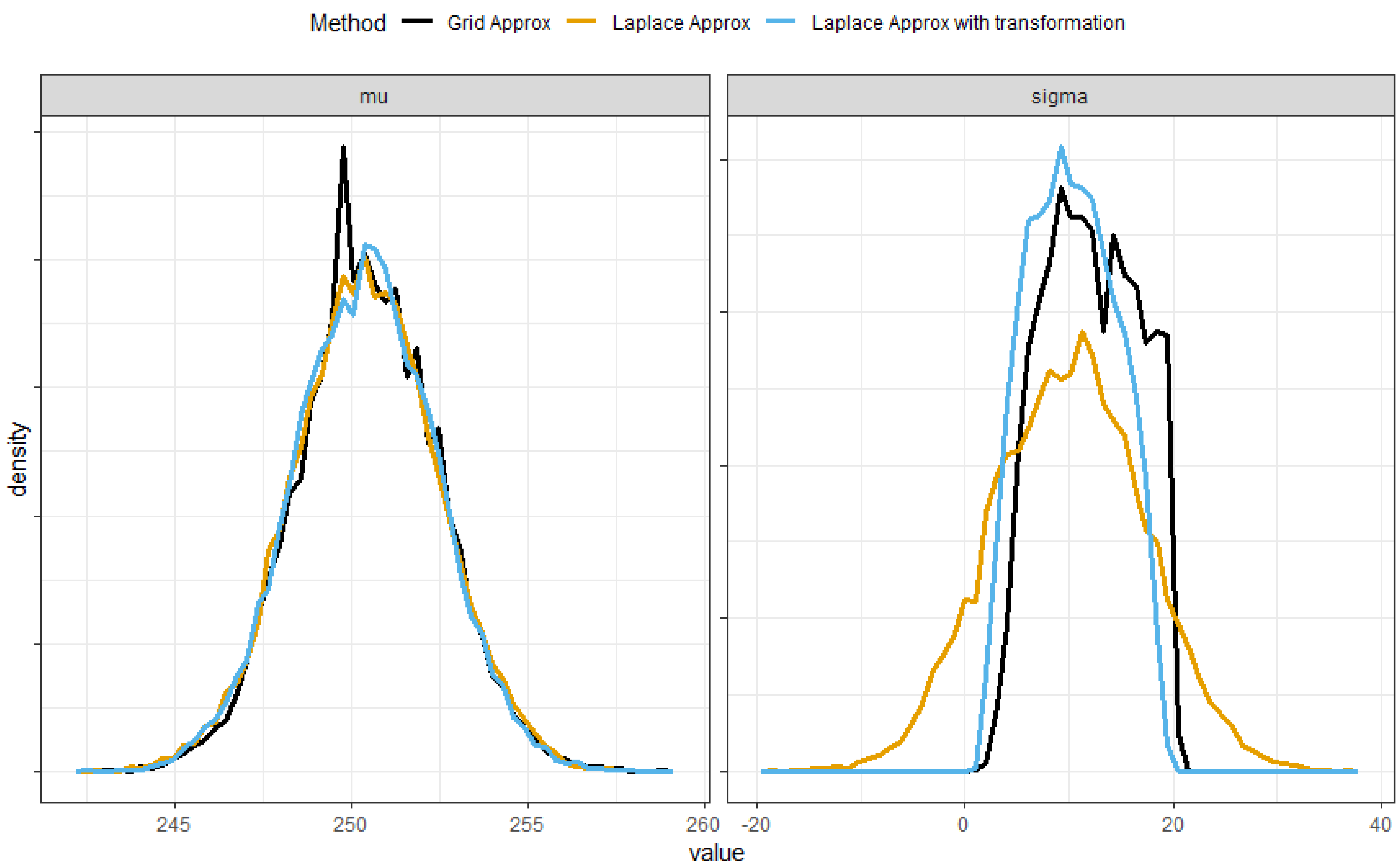
Not perfect with respect to ϕ, but how does this impact the result on the original parameter σ?

Generate random samples from the approximate MVN posterior on μ, ϕ

```r
### draw posterior samples from the MVN approximate posterior
### on the unbounded variables
set.seed(5004)

post_cv_mvn_samples <- MASS::mvrnorm(n = 1e4,
                                     mu = cv_laplace_result_N01$mode,
                                     Sigma = cv_laplace_result_N01$var_matrix) %>%
  as.data.frame() %>% tbl_df() %>%
  purrr::set_names(c("mu", "logit_sigma"))
```

“Back-transform” the posterior ϕ samples to σ and compare with the previous Laplace approximation and “true” grid approximation results

```r
grid_approx_samples %>%
  select(mu, sigma) %>%
  mutate(type = "Grid Approx") %>%
  bind_rows(post_mvn_samples %>%
              mutate(type = "Laplace Approx")) %>%
  bind_rows(post_cv_mvn_samples %>%
              mutate(sigma = info_use$sigma_lwr +
                       (info_use$sigma_upr-info_use$sigma_lwr)*boot::inv.logit(logit_sigma)) %>%
              select(mu, sigma) %>%
              mutate(type = "Laplace Approx with transformation")) %>%
  tibble::rowid_to_column("post_id") %>%
  tidyr::gather(key = "key", value = "value", -post_id, -type) %>%
  ggplot(mapping = aes(x = value)) +
  geom_freqpoly(mapping = aes(group = interaction(key, type),
                              color = type,
                              y = stat(density)),
                bins = 55, size = 1.15) +
  facet_wrap(~ key, scales = "free") +
  ggthemes::scale_color_colorblind("Method") +
  theme_bw() +
  theme(legend.position = "top",
        axis.text.y = element_blank())
```


Method
Grid Approx
Laplace Approx
Laplace Approx with transformation
mu
sigma
density
245
250
255
260
-20
0
20
40
value