ESS 575: Markov Chain Monte Carlo Lab - Gibbs Sampling

Team England

29 September, 2022

Team England:

- Caroline Blommel
- Carolyn Coyle
- Bryn Crosby
- George Woolsey

cblommel@mail.colostate.edu, carolynm@mail.colostate.edu, brcrosby@rams.colostate.edu, george.woolsey@colostate.edu

Motivation

This problem challenges you to understand how the Markov chain Monte Carlo algorithm takes a multivariate joint distribution and breaks it into a series of univariate, marginal distributions that can be approximated one at a time. There are only two unknowns in this problem, but the same principles and approach would apply if there were two hundred. The accompanying document, MCMCMath1.pdf, describes the math that stands behind the coding that you will do here. You should study this thoroughly before proceeding.

Problem

You will write code using conjugate relationships, also known as Gibbs updates, to draw samples from marginal posterior distributions of a mean and variance.

1.

Set the seed for random numbers = 10 in R with `set.seed(10)`.

```
# set seed
set.seed(10)
```

2.

Load the `actuar` library, which contains functions for inverse gamma distributions needed in step 4.

```
library(actuar)
```

3.

Simulate `100` data points from a normal distribution with mean $\theta = 100$ and variance $\varsigma^2 = 25$ (where). Call the data set `y`. Be careful here. R requires the standard deviation, not the variance, as a parameter. You will use these "fake" data to verify the Gibbs sampler you will write below. Simulating data is always a good way to test methods. Your method should be able to recover the generating parameters given a sufficiently large number of simulated observations.

The variance in the likelihood (ς^2) is the variance of the distribution of the observations (y_i), not the variance of the distribution of the mean of the observations (θ).

```
# set parameters
n <- 100
varsigma_sq <- 25
theta <- 100
# draw random sample
y <- rnorm(n = n, mean = theta, sd = sqrt(varsigma_sq))
```

4.

I have saved you some time by writing a function called `draw_mean` that makes draws from the marginal posterior distributions for θ using a normal-normal conjugate relationship where the variance is assumed to be known. It is vital that you study the MCMCMath1.pdf notes relative to this function.

```
# normal likelihood with normal prior conjugate for mean, assuming variance is known
# mu_0 is prior mean
# sigma.sq_0 is prior variance of mean
# varsigma.sq is known variance of data

draw_mean = function(mu_0, sigma.sq_0, varsigma.sq, y){
    mu_1 =((mu_0 / sigma.sq_0 + sum(y)/varsigma.sq)) / (1/sigma.sq_0 + length(y) / varsigma.sq)
    sigma.sq_1 = 1/(1 / sigma.sq_0 + length(y) / varsigma.sq)
    z = rnorm(1, mu_1, sqrt(sigma.sq_1))
    param = list(z = z, mu_1 = mu_1, sigma.sq_1 = sigma.sq_1)
    return(param)
}
```

5.

I have also provided a function called `draw_var` that makes draws from the marginal posterior distribution for σ^2 using a inverse gamma-normal conjugate relationship where the mean is assumed to be known. Study this function relative to the MCMCMath1.pdf handout.

```
# normal likelihood with gamma prior conjugate relationship for variance, assuming mean is known
# alpha_0 is parameter of prior for variance
# beta_0 is parameter of prior for variance
# Note that this uses scale parameterization for inverse gamma

draw_var = function(alpha_0, beta_0, theta, y){
```

```
    alpha_1 = alpha_0 + length(y) / 2
    beta_1 = beta_0 + sum((y - theta)^2) / 2
    z = actuar::rinvgamma(1, alpha_1, scale = beta_1)
    param = list(z = z, alpha_1 = alpha_1, beta_1 = beta_1)
    return(param)
}
```

6.

Check the functions by simulating a large number of data points from a normal distribution using a mean and variance of your own choosing. Store the data points in a vector called `y_check`. Assume flat priors for the mean and the variance. A vague prior for the inverse gamma has parameters $\alpha_0 = 0.001$ and $\beta_0 = 0.001$.

```
# set parameters
n_check <- 10000
varsigma_sq_check <- 11
theta_check <- 23
alpha_0 <- 0.001
beta_0 <- 0.001
# draw random sample
y_check <- rnorm(n = n_check, mean = theta_check, sd = sqrt(varsigma_sq_check))
# set up vector to hold simulation results
variance_check <- numeric(n_check)
mean_check <- numeric(n_check)
# check draw functions
for(i in 1:n_check){
  # draw_mean
  mean_check[i] <- draw_mean(
      mu_0 = 0 # assume flat prior
      , sigma.sq_0 = 10000 # assume flat prior (the variance is a very large number to be uniformative)
      , varsigma.sq = varsigma_sq_check
      , y = y_check
    )$z # this returns only the random variable from the draw
  # draw_var
  variance_check[i] <- draw_var(
      alpha_0 = alpha_0
      , beta_0 = beta_0
      , theta = theta_check
      , y = y_check
    )$z # this returns only the random variable from the draw
}
```

The mean of the draws from the marginal posterior distributions for θ using a normal-normal conjugate relationship where the variance is assumed to be known based on a simulated normal distribution with mean = 23 and a variance = 11 is: **23.0**

The mean of the draws from the marginal posterior distribution for σ^2 using a inverse gamma-normal conjugate relationship where the mean is assumed to be known based on a simulated normal distribution with mean = 23 and a variance = 11 is: **11.1**

Writing a sampler

Now execute these steps:

1.

Set up a matrix for storing samples from the posterior distribution of the mean. The number of rows should equal the number of chains (3) and number of columns should equal the number of iterations (10,000). Do the same thing for storing samples from the posterior distribution of the variance.

```
# set parameters
n_iterations <- 10000
n_chains <- 3
# set up matrix for storing samples
posterior_mean <- matrix(nrow = n_chains, ncol = n_iterations)
posterior_variance <- matrix(nrow = n_chains, ncol = n_iterations)
```

2.

Assign initial values to the first column of each matrix, a different value for each of the chains. These can be virtually any value within the support of the random variable, but it would be fine for this exercise to use values not terribly far away from to those you used to simulate the data, reflecting some prior knowledge. You might try varying these later to show that you will get the same results.

```
# set initial values in first column
posterior_mean[1:3, 1] <- c(123, 111, 85)
posterior_variance[1:3, 1] <- c(33, 21, 3)
# check
posterior_mean[,1:6]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  123   NA   NA   NA   NA   NA
## [2,]  111   NA   NA   NA   NA   NA
## [3,]   85   NA   NA   NA   NA   NA
```

```
posterior_variance[,1:6]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   33   NA   NA   NA   NA   NA
## [2,]   21   NA   NA   NA   NA   NA
## [3,]    3   NA   NA   NA   NA   NA
```

3.

Set up nested for loops to iterate from one to the total number of iterations for each of the three chains for each parameter. Use the conjugate functions `draw_mean` and `draw_var` to draw a sample from the distribution of the mean using the value of the variance at the current iteration. Then make a draw from the variance using the current value of the mean. Repeat. Assume vague priors for the mean and variance:

```r
# A vague prior for the inverse gamma has parameters:
alpha_0 <- 0.001
beta_0 <- 0.001
# Set up nested for loops
for(c in 1:n_chains){
  for(i in 2:n_iterations){
    # draw_mean
    posterior_mean[c, i] <- draw_mean(
        mu_0 = 0 # assume flat prior
        , sigma.sq_0 = 10000 # assume flat prior (the variance is a very large number to be uniformativ
        , varsigma.sq = posterior_variance[c, i-1]
        , y = y
      )$z # this returns only the random variable from the draw
    # draw_var
    posterior_variance[c, i] <- draw_var(
        alpha_0 = alpha_0
        , beta_0 = beta_0
        , theta = posterior_mean[c, i-1]
        , y = y
      )$z # this returns only the random variable from the draw
  }
}
# check
posterior_mean[,1:6]
```

```
##      [,1]     [,2]      [,3]      [,4]      [,5]      [,6]
## [1,]  123 99.67418 100.75187  99.37638  99.08503  99.60730
## [2,]  111 98.58493  97.75623  98.87565  99.41820 100.41588
## [3,]   85 99.20689 103.62134 100.01897 100.34251  98.10434
```

```r
posterior_variance[,1:6]
```

```
##      [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## [1,]   33 529.1705 18.08878 24.86689 21.79517 28.47206
## [2,]   21 144.5952 19.66484 21.91610 24.34829 22.19251
## [3,]    3 244.7083 22.90137 42.00224 18.80265 21.75202
```

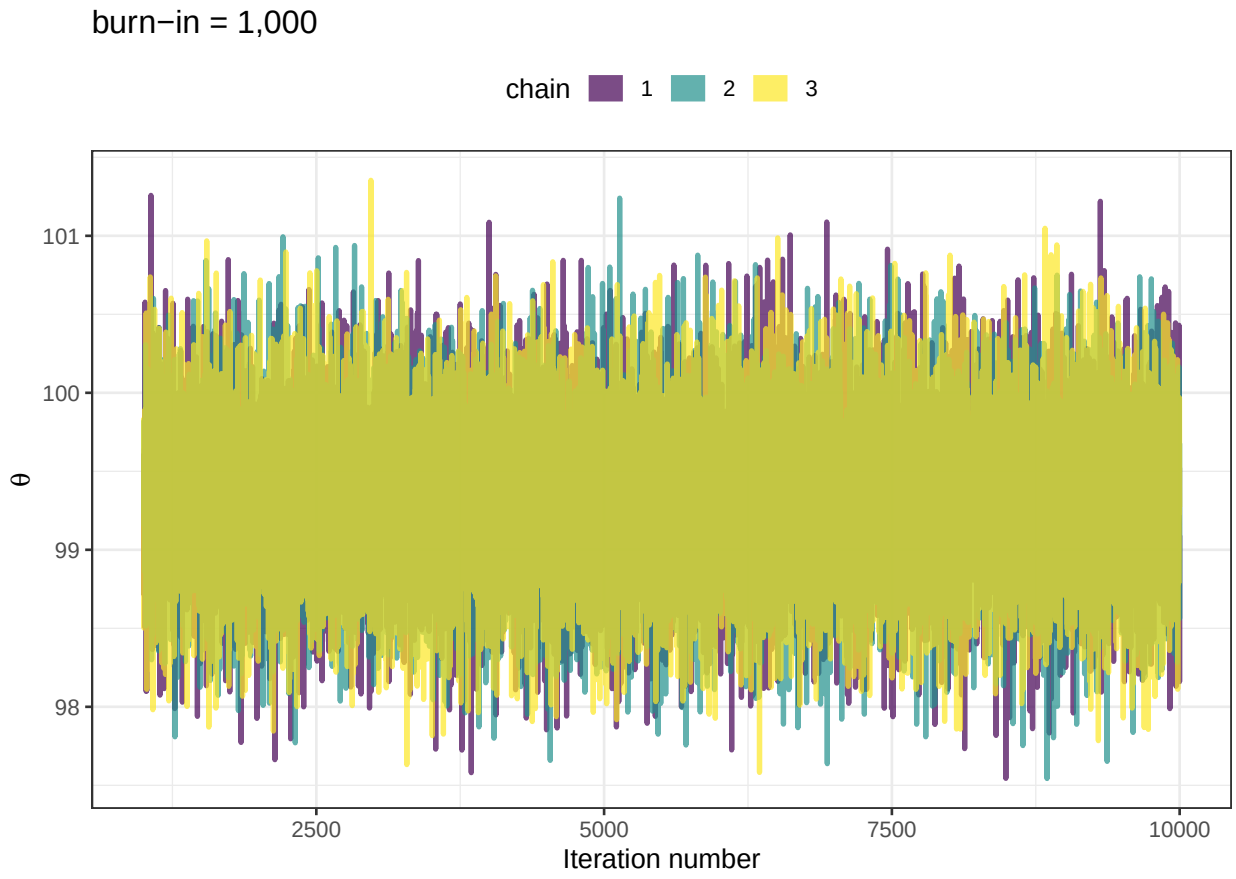
Trace plots and plots of marginal posteriors

1.

Discard the first 1000 iterations as burn-in. Plot the value of the mean as a function of iteration number for each chain. This is called a trace plot.

```r
# Discard the first 1000 iterations as burn-in.
burnin <- 1000
# make burnin data frames
# mean
posterior_mean_burnin <- posterior_mean[,(burnin+1):n_iterations] %>%
  as.data.frame() %>%
```

```
  dplyr::mutate(chain = as.character(dplyr::row_number())) %>%
  tidyr::pivot_longer(
    cols = tidyr::starts_with("V")
    , names_to = "iteration"
    , names_prefix = "V"
    , values_to = "value"
    , values_drop_na = FALSE
  ) %>%
  dplyr::mutate(iteration = as.numeric(iteration)+burnin)
# variance
posterior_variance_burnin <- posterior_variance[,(burnin+1):n_iterations] %>%
  as.data.frame() %>%
  dplyr::mutate(chain = as.character(dplyr::row_number())) %>%
  tidyr::pivot_longer(
    cols = tidyr::starts_with("V")
    , names_to = "iteration"
    , names_prefix = "V"
    , values_to = "value"
    , values_drop_na = FALSE
  ) %>%
  dplyr::mutate(iteration = as.numeric(iteration)+burnin)
# Plot the value of the mean as a function of iteration number for each chain
ggplot(
    data = posterior_mean_burnin
    , mapping = aes(x = iteration, y = value, color = chain)
  ) +
  geom_line(lwd = 1) +
  scale_color_viridis_d(alpha = 0.7, option = "viridis") +
  xlab("Iteration number") +
  ylab(latex2exp::TeX("$\\theta$")) +
  labs(title = paste0("burn-in = ", scales::comma(burnin, accuracy = 1))) +
  theme_bw() +
  theme(
    legend.position = "top"
    , legend.direction = "horizontal"
  ) +
  guides(color = guide_legend(override.aes = list(size = 5)))
```

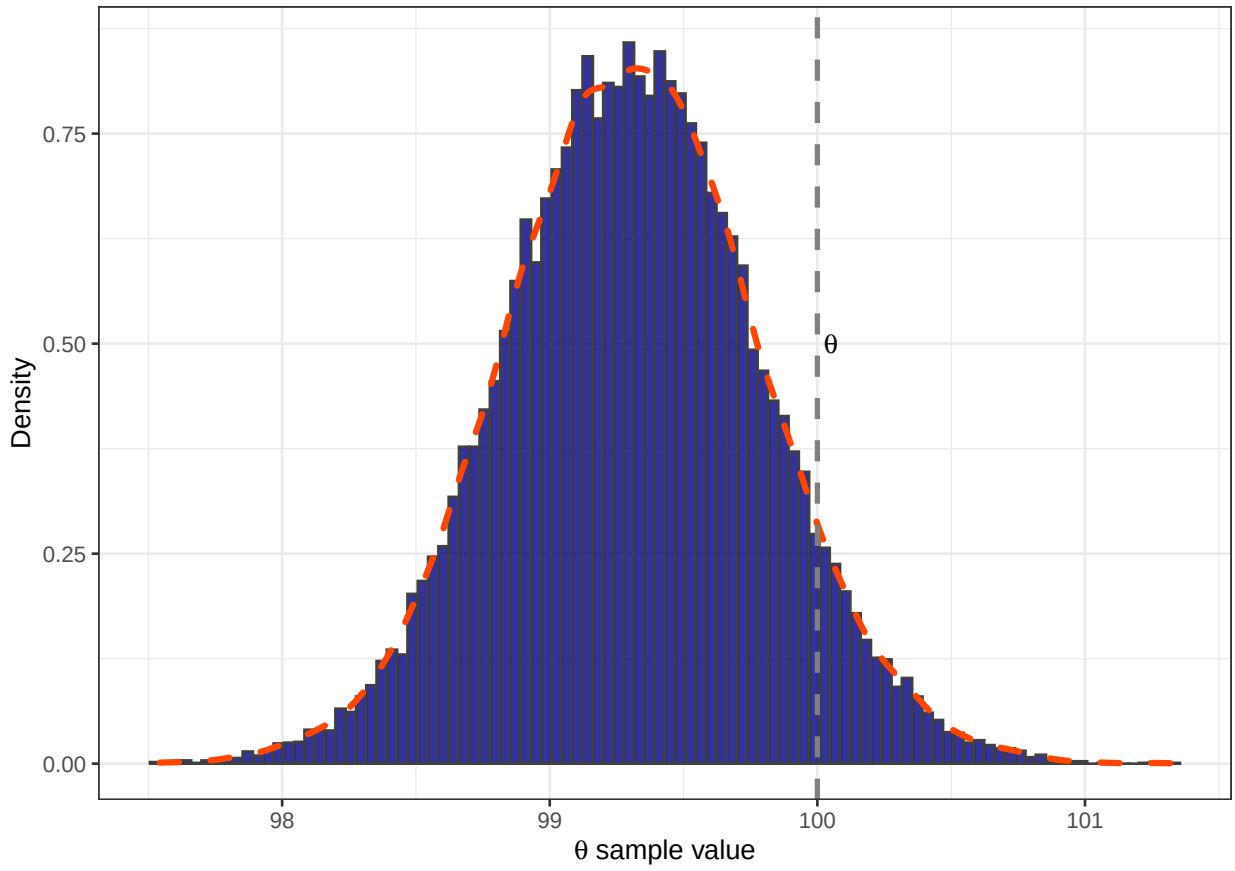


2.

Make a histogram of the samples of the mean retained after burn-in including all chains. Put a vertical line on the plot showing the generating value.

```
# Make a histogram of the samples of the mean retained after burn-in
ggplot(
    data = posterior_mean_burnin
    , mapping = aes(x = value)
  ) +
  geom_histogram(
    aes(y = ..density..)
    , bins = 100
    , fill = "navy"
    , alpha = 0.8
    , color = "gray25"
  ) +
  geom_density(
    aes(y = ..density..)
    , linetype = 2
    , lwd = 1.2
    , color = "orangered"
  ) +
  geom_vline(xintercept = theta, color = "gray50", linetype = 2, lwd = 1) +
```

```
  annotate("text", x = theta*1.0005, y = 0.5, parse = TRUE, label = "theta", color = "black") +
  xlab(latex2exp::TeX("$\\theta$ sample value")) +
  ylab("Density") +
  theme_bw()
```

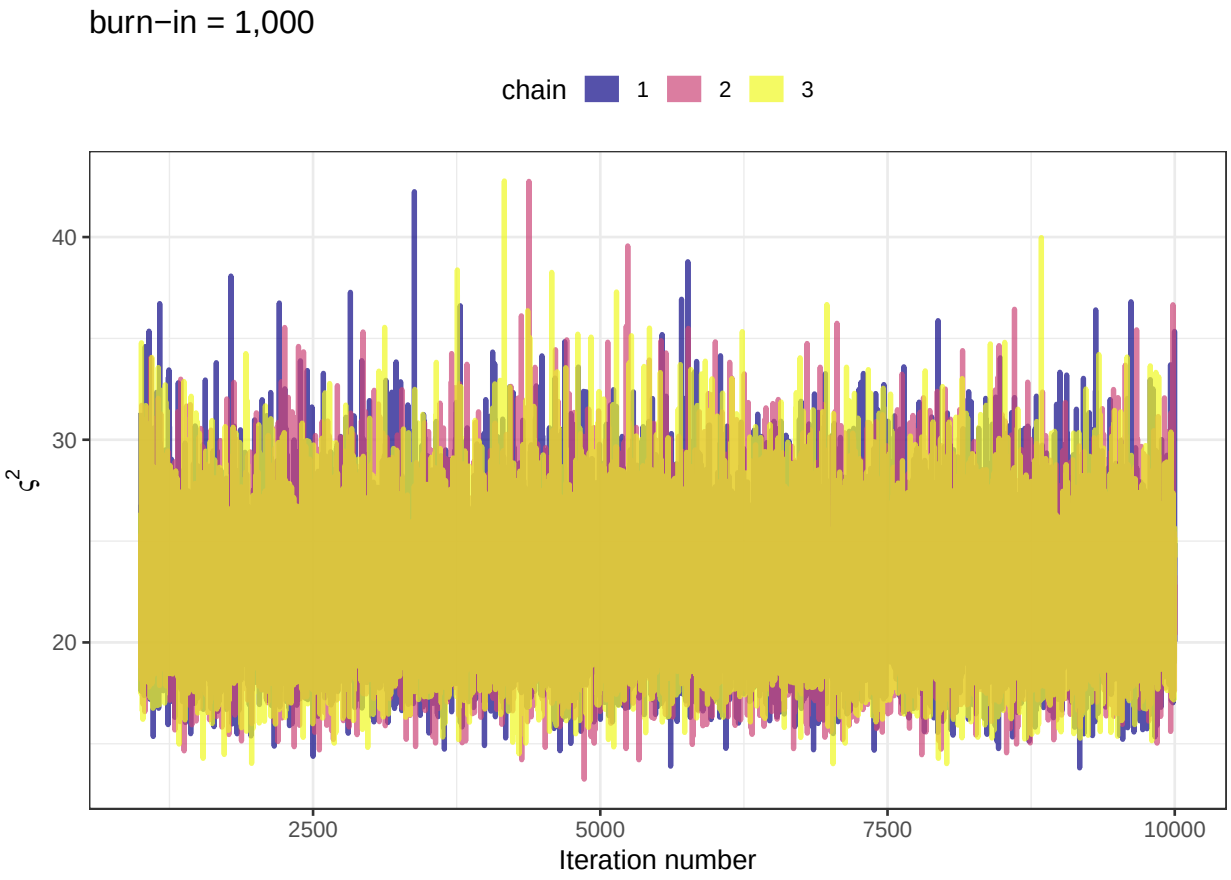


3.

Repeat steps 1-2 for the variance.

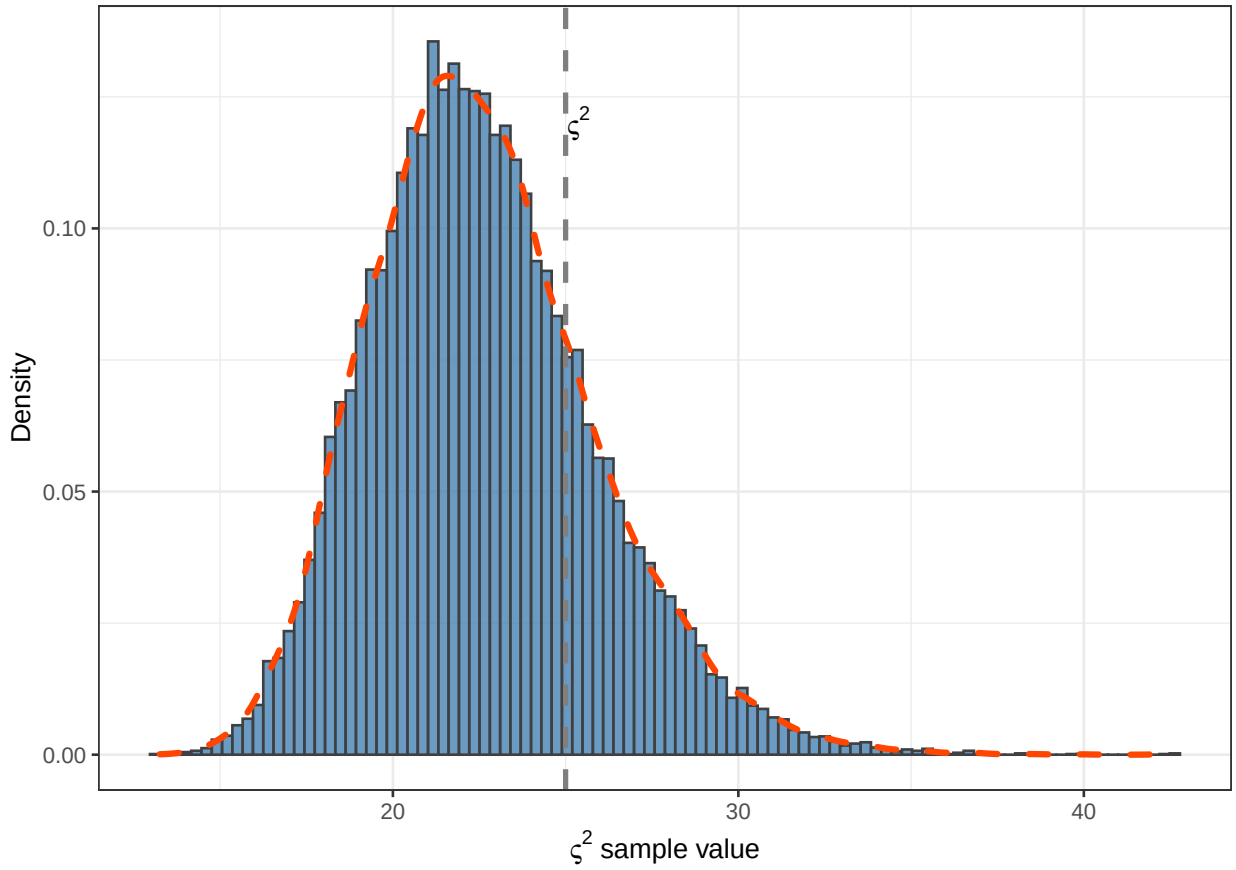
```
# Plot the value of the variance as a function of iteration number for each chain
ggplot(
    data = posterior_variance_burnin
    , mapping = aes(x = iteration, y = value, color = chain)
  ) +
  geom_line(lwd = 1) +
  scale_color_viridis_d(alpha = 0.7, option = "plasma") +
  xlab("Iteration number") +
  ylab(latex2exp::TeX("$\\varsigma^2$")) +
  labs(title = paste0("burn-in = ", scales::comma(burnin, accuracy = 1))) +
  theme_bw() +
  theme(
    legend.position = "top"
    , legend.direction = "horizontal"
```

```
  ) +
  guides(color = guide_legend(override.aes = list(size = 5)))
```




```
# Make a histogram of the samples of the variance retained after burn-in
ggplot(
    data = posterior_variance_burnin
    , mapping = aes(x = value)
  ) +
  geom_histogram(
    aes(y = ..density..)
    , bins = 100
    , fill = "steelblue"
    , alpha = 0.8
    , color = "gray25"
  ) +
  geom_density(
    aes(y = ..density..)
    , linetype = 2
    , lwd = 1.2
    , color = "orangered"
  ) +
  geom_vline(xintercept = varsigma_sq, color = "gray50", linetype = 2, lwd = 1) +
  annotate("text", x = varsigma_sq*1.015, y = 0.12, parse = TRUE, label = "varsigma^2", color = "black"
  xlab(latex2exp::TeX("$\\varsigma^2$ sample value")) +
```

```
  ylab("Density") +
  theme_bw()
```



4.

For both θ and ς^2, calculate the mean of all the chains combined and its standard deviation. Interpret these quantities.

For θ:

- The mean of the draws for all chains combined from the marginal posterior distribution for θ discarding the first 1000 iterations as burn-in is: **99.31**
- The standard deviation of the draws for all chains combined from the marginal posterior distribution for θ discarding the first 1000 iterations as burn-in is: **0.48**

For ς^2:

- The mean of the draws for all chains combined from the marginal posterior distribution for ς^2 discarding the first 1000 iterations as burn-in is: **22.60**
- The standard deviation of the draws for all chains combined from the marginal posterior distribution for ς^2 discarding the first 1000 iterations as burn-in is: **3.26**

The generating value of $\theta = 100.0$ and the generating value of $\varsigma^2 = 25.0$. The simulated values shown above are close to the generating value for both θ and ς^2 but they are not exact. This is due to the random process of generating the data and the relatively small sample size of 100. As we increase our sample size for the random collection of data, the marginal posteriors of θ and ς^2 should approach the generating value.

5.

Compare the standard deviation of the posterior distribution of θ with an approximation using the standard deviation of the data divided by the square root of the sample size. What is this approximation called in the frequentist world?

- The standard deviation of the draws for all chains combined from the marginal posterior distribution for θ discarding the first 1000 iterations as burn-in is: **0.48**
- The standard error of the data is calculated as: $SE = \frac{\sigma}{\sqrt{n}}$; where σ is the standard deviation of the data and n is the sample size of the data. The standard error of the data 'y' is: **0.47**

The standard deviation of marginal posterior distribution for θ closely matches the standard error of the data 'y'.

6.

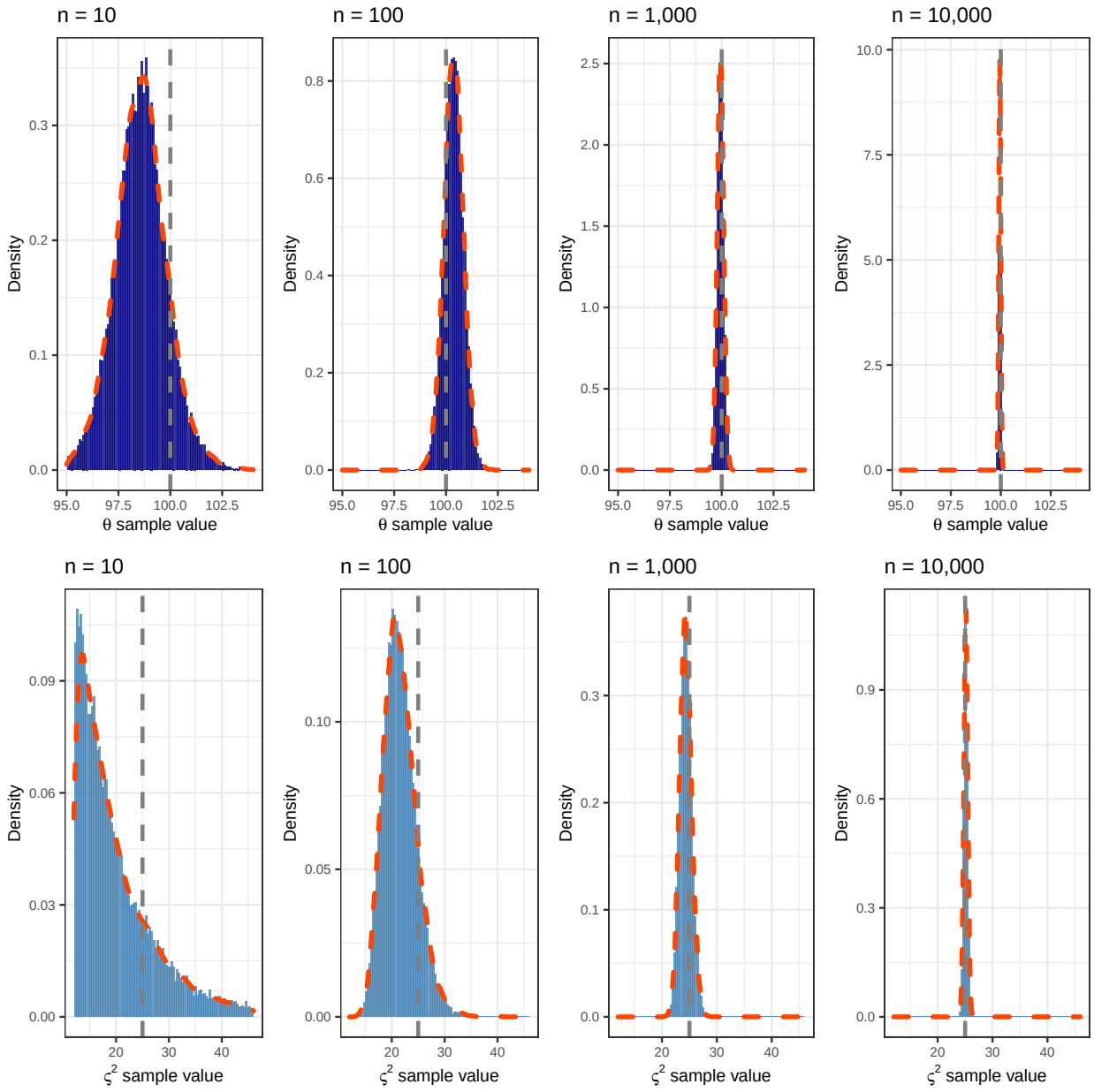
Vary the number of values in the simulated data set, e.g., n = 10, 100, 1,000, 10,000. We do not exactly recover the generating values of θ and ς^2 when n is small. Why? The mean of the marginal posterior distribution of the variance is further away from its generating value than the mean is. Why? Try different values for set seed with n = 100 and interpret the effect of changing the random number sequence.

```
# set parameters
n <- 100
varsigma_sq <- 25
theta <- 100
n_iterations <- 10000
n_chains <- 3
alpha_0 <- 0.001
beta_0 <- 0.001
burnin <- 1000
# set up plot grid
plts <- list()
mean_limt <- c(round(min(posterior_mean_burnin$value)*.97), round(max(posterior_mean_burnin$value)*1.03
var_limt <- c(round(min(posterior_variance_burnin$value)*.93), round(max(posterior_variance_burnin$valu
these_ns <- c(10, 100, 1000, 10000)
for(this_n in 1:length(these_ns)) {
  # draw random sample
  temp_y <- rnorm(n = these_ns[this_n], mean = theta, sd = sqrt(varsigma_sq))
  # set up matrix for storing samples
  temp_mean <- matrix(nrow = n_chains, ncol = n_iterations)
  temp_variance <- matrix(nrow = n_chains, ncol = n_iterations)
  # set initial values in first column
  temp_mean[1:3, 1] <- c(123, 111, 85)
  temp_variance[1:3, 1] <- c(33, 21, 3)
  # A vague prior for the inverse gamma has parameters:
  #  nested for loops to simulate posterior dist
```

```r
for(c in 1:n_chains){
  for(i in 2:n_iterations){
    # draw_mean
    temp_mean[c, i] <- draw_mean(
        mu_0 = 0 # assume flat prior
        , sigma.sq_0 = 10000 # assume flat prior (the variance is a very large number to be uniformat
        , varsigma.sq = temp_variance[c, i-1]
        , y = temp_y
      )$z # this returns only the random variable from the draw
    # draw_var
    temp_variance[c, i] <- draw_var(
        alpha_0 = alpha_0
        , beta_0 = beta_0
        , theta = temp_mean[c, i-1]
        , y = temp_y
      )$z # this returns only the random variable from the draw
  }
}
# Discard the first 1000 iterations as burn-in.
# make burnin data frames
# mean
plts[[this_n]] <-
  temp_mean[,(burnin+1):n_iterations] %>%
    as.data.frame() %>%
    dplyr::mutate(chain = as.character(dplyr::row_number())) %>%
    tidyr::pivot_longer(
      cols = tidyr::starts_with("V")
      , names_to = "iteration"
      , names_prefix = "V"
      , values_to = "value"
      , values_drop_na = FALSE
    ) %>%
    dplyr::mutate(iteration = as.numeric(iteration)+burnin) %>%
    # Make a histogram of the samples of the mean retained after burn-in
    ggplot(
        data = .
        , mapping = aes(x = value)
      ) +
      geom_histogram(
        aes(y = ..density..)
        , bins = 100
        , fill = "navy"
        , alpha = 0.8
        # , color = "gray25"
      ) +
      geom_density(
        aes(y = ..density..)
        , linetype = 2
        , lwd = 1.2
        , color = "orangered"
      ) +
      geom_vline(xintercept = theta, color = "gray50", linetype = 2, lwd = 1) +
      # annotate("text", x = theta*1.0005, y = -0.01, parse = TRUE, label = "theta", color = "black")
```

```
      xlab(latex2exp::TeX("$\\theta$ sample value")) +
      ylab("Density") +
      scale_x_continuous(limits = mean_limt) +
      labs(title = paste0("n = ", scales::comma(these_ns[this_n], accuracy = 1))) +
      theme_bw() +
    theme(
      axis.text = element_text(size = 7)
      , title = element_text(size = 9)
    )

# variance
plts[[this_n+length(these_ns)]] <-
temp_variance[,(burnin+1):n_iterations] %>%
  as.data.frame() %>%
  dplyr::mutate(chain = as.character(dplyr::row_number())) %>%
  tidyr::pivot_longer(
    cols = tidyr::starts_with("V")
    , names_to = "iteration"
    , names_prefix = "V"
    , values_to = "value"
    , values_drop_na = FALSE
  ) %>%
  dplyr::mutate(iteration = as.numeric(iteration)+burnin) %>%
  # Make a histogram of the samples of the variance retained after burn-in
  ggplot(
      data = .
      , mapping = aes(x = value)
    ) +
    geom_histogram(
      aes(y = ..density..)
      , bins = 100
      , fill = "steelblue"
      , alpha = 0.8
      # , color = "gray25"
    ) +
    geom_density(
      aes(y = ..density..)
      , linetype = 2
      , lwd = 1.2
      , color = "orangered"
    ) +
    geom_vline(xintercept = varsigma_sq, color = "gray50", linetype = 2, lwd = 1) +
    # annotate("text", x = varsigma_sq*1.015, y = -0.01, parse = TRUE, label = "varsigma^2", color =
    xlab(latex2exp::TeX("$\\varsigma^2$ sample value")) +
    ylab("Density") +
    scale_x_continuous(limits = var_limt) +
    labs(title = paste0("n = ", scales::comma(these_ns[this_n], accuracy = 1))) +
    theme_bw() +
    theme(
      axis.text = element_text(size = 7)
      , title = element_text(size = 9)
    )
```

```
}
remove(list = ls()[grep("temp_",ls())])
# combine plots
cowplot::plot_grid(plotlist =  plts, nrow = 2, ncol = length(these_ns))
```



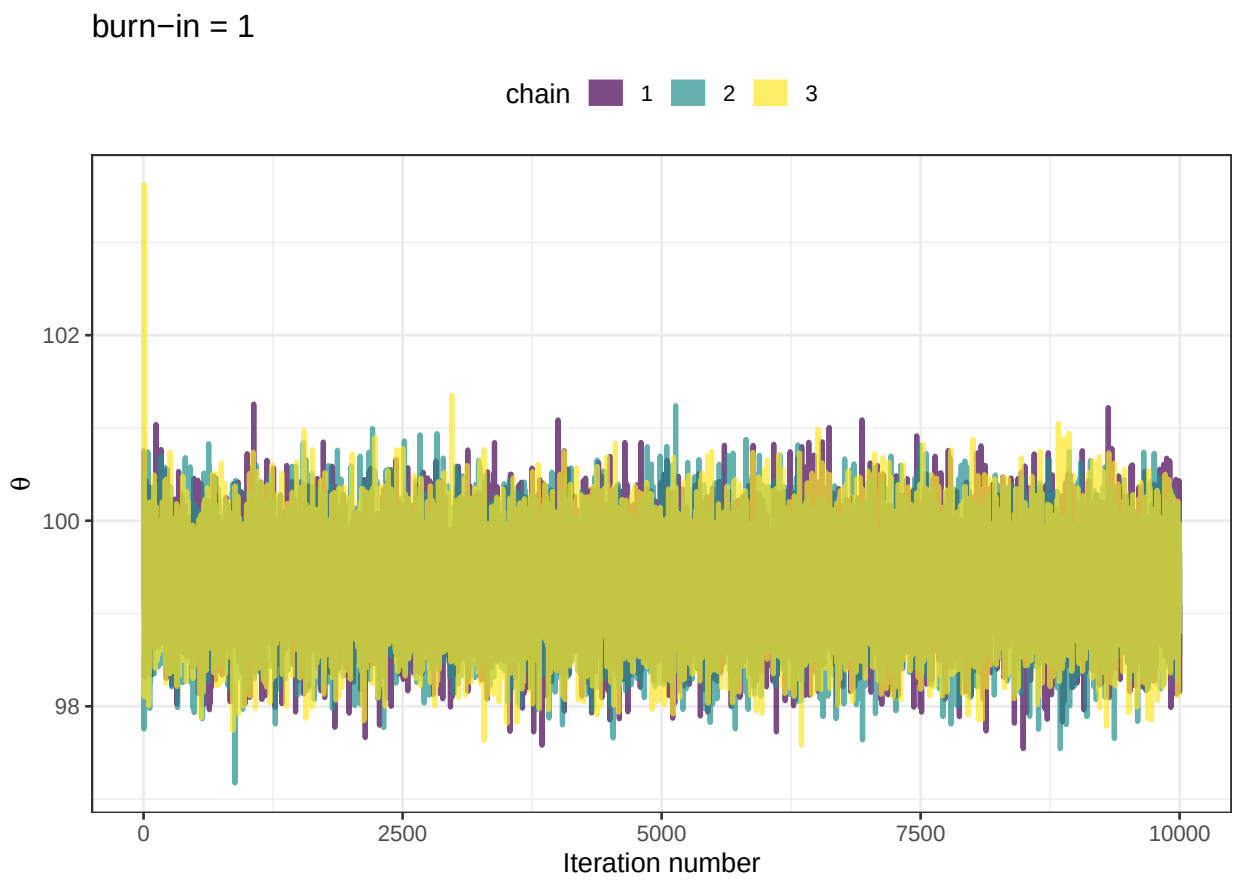
As the number samples in the simulated data increases, the variance of the posterior distribution decreases and the mean of the posterior distribution moves closer to the generating value. The mean of the posterior distribution of θ and ς^2 deviate slightly from the generating value with small sample sizes because the simulated data is just one representation of a random process. The marginal posterior distribution of the variance is further away from its generating value than the mean is because the variance is more sensitive to values in the tails of the data distribution.

7.

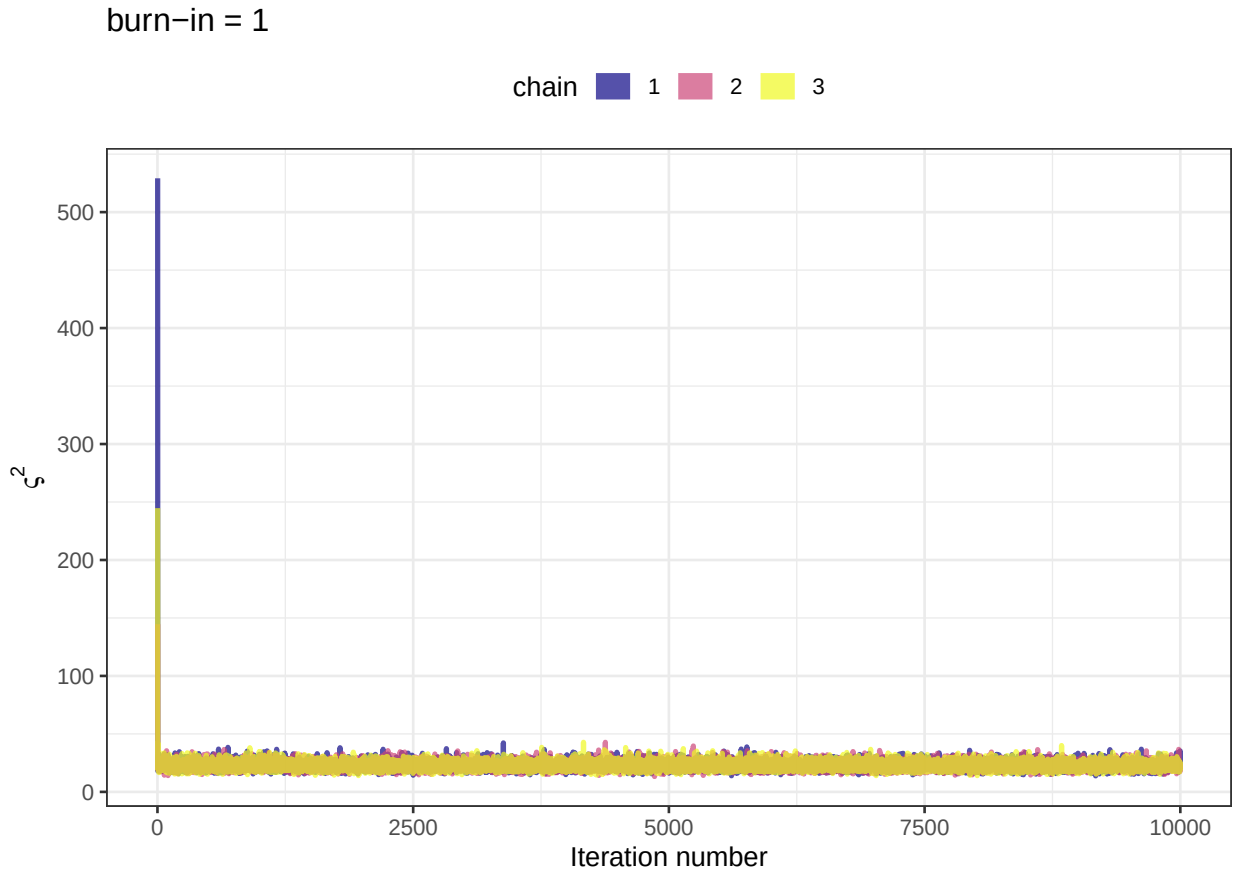
Make the burnin = 1 instead of 1000. Does this change your results? Why or why not?

```
# Discard the first 1000 iterations as burn-in.
burnin <- 1
# Plot the value of the mean as a function of iteration number for each chain
plt_mean <-
posterior_mean[,(burnin+1):n_iterations] %>%
  as.data.frame() %>%
  dplyr::mutate(chain = as.character(dplyr::row_number())) %>%
  tidyr::pivot_longer(
    cols = tidyr::starts_with("V")
    , names_to = "iteration"
    , names_prefix = "V"
    , values_to = "value"
    , values_drop_na = FALSE
  ) %>%
  dplyr::mutate(iteration = as.numeric(iteration)+burnin) %>%
ggplot(
    data = .
    , mapping = aes(x = iteration, y = value, color = chain)
  ) +
  geom_line(lwd = 1) +
  scale_color_viridis_d(alpha = 0.7, option = "viridis") +
  xlab("Iteration number") +
  ylab(latex2exp::TeX("$\\theta$")) +
  labs(title = paste0("burn-in = ", scales::comma(burnin, accuracy = 1))) +
  theme_bw() +
  theme(
    legend.position = "top"
    , legend.direction = "horizontal"
  ) +
  guides(color = guide_legend(override.aes = list(size = 5)))


# Plot the value of the variance as a function of iteration number for each chain
plt_var <-
posterior_variance[,(burnin+1):n_iterations] %>%
  as.data.frame() %>%
  dplyr::mutate(chain = as.character(dplyr::row_number())) %>%
  tidyr::pivot_longer(
    cols = tidyr::starts_with("V")
    , names_to = "iteration"
    , names_prefix = "V"
    , values_to = "value"
    , values_drop_na = FALSE
  ) %>%
  dplyr::mutate(iteration = as.numeric(iteration)+burnin) %>%
ggplot(
    data = .
    , mapping = aes(x = iteration, y = value, color = chain)
  ) +
  geom_line(lwd = 1) +
```

```
  scale_color_viridis_d(alpha = 0.7, option = "plasma") +
  xlab("Iteration number") +
  ylab(latex2exp::TeX("$\\varsigma^2$")) +
  labs(title = paste0("burn-in = ", scales::comma(burnin, accuracy = 1))) +
  theme_bw() +
  theme(
    legend.position = "top"
    , legend.direction = "horizontal"
  ) +
  guides(color = guide_legend(override.aes = list(size = 5)))

plt_mean
```




```
plt_var
```



Changing the burn-in from 1,000 to 1 does not significantly change the final mean of the draws for all chains combined from the marginal posterior distribution for θ and ς^2. As shown in the trace plots above, there is convergence almost immediatly because Gibbs sampling makes "smart" parameter proposals so we retain all of the sampled values. Accept-reject sampling methods like Metropolis and Metropolis-Hastings require lots of computation time because of the several steps required to calculate likelihoods, compute ratios and choose proposals versus current values. These multiple steps result in slower chain convergence.

8.

Reverse the order of the conjugate functions in step 3 of the Writing a Sampler section so that the variance is drawn first followed by the mean. (Be careful, this involves a bit more than simply reversing the order of the functions in the loop). Does this reordering have an effect on the posteriors? Why or why not?

```
# set parameters
n_iterations <- 10000
n_chains <- 3
# set up matrix for storing samples
posterior_mean_rev <- matrix(nrow = n_chains, ncol = n_iterations)
posterior_variance_rev <- matrix(nrow = n_chains, ncol = n_iterations)
# set initial values in first column
posterior_mean_rev[1:3, 1] <- c(123, 111, 85)
posterior_variance_rev[1:3, 1] <- c(33, 21, 3)
# A vague prior for the inverse gamma has parameters:
alpha_0 <- 0.001
beta_0 <- 0.001
```

```r
# Set up nested for loops
for(c in 1:n_chains){
  for(i in 2:n_iterations){
    # draw_var
    posterior_variance_rev[c, i] <- draw_var(
        alpha_0 = alpha_0
        , beta_0 = beta_0
        , theta = posterior_mean_rev[c, i-1]
        , y = y
      )$z # this returns only the random variable from the draw
    # draw_mean
    posterior_mean_rev[c, i] <- draw_mean(
        mu_0 = 0 # assume flat prior
        , sigma.sq_0 = 10000 # assume flat prior (the variance is a very large number to be uniformativ
        , varsigma.sq = posterior_variance_rev[c, i-1]
        , y = y
      )$z # this returns only the random variable from the draw
  }
}
```

For θ:

- Sampling the conjugate function for the mean first, the mean of the draws for all chains combined from the marginal posterior distribution for θ discarding the first 1000 iterations as burn-in is: **99.31**
- Sampling the conjugate function for the variance first, the mean of the draws for all chains combined from the marginal posterior distribution for θ discarding the first 1000 iterations as burn-in is: **99.32**

For ς^2:

- Sampling the conjugate function for the mean first, the mean of the draws for all chains combined from the marginal posterior distribution for ς^2 discarding the first 1000 iterations as burn-in is: **22.64**
- Sampling the conjugate function for the variance first, the mean of the draws for all chains combined from the marginal posterior distribution for ς^2 discarding the first 1000 iterations as burn-in is: **22.66**

This reordering does not seem to have an effect on the posterior distributions of θ and ς^2. The reason that the ordering does not seem to matter is because Gibbs sampling makes "smart" parameter proposals so we retain all of the sampled values and convergence happens quickly.