Tarea 6

Juan M Karawcki, Bruno Pintos

2025-09-22

Ejercicio 6.7 (Aproximación por grilla Normal-Normal)

Consideremos el modelo Normal–Normal para μ con

$$Y_i \mid \mu \sim N(\mu,\, 1.3^2)$$

y

$$\mu \sim N(10,\, 1.2^2)$$

Supongamos que en $n = 4$ observaciones independientes se obtienen los datos:

$$(Y_1, Y_2, Y_3, Y_4) = (7.1,\, 8.9,\, 8.4,\, 8.6)$$

1. Utiliza la aproximación por grilla con valores de

$$\mu \in \{5, 6, 7, \ldots, 15\}$$

para aproximar el modelo posterior de μ.

```
# PASO 1: Definir una grilla de 11 valores de mu
grid_data <- data.frame(mu_grid = seq(from = 5, to = 15, by = 1))
y <- c(7.1, 8.9, 8.4, 8.6)
```

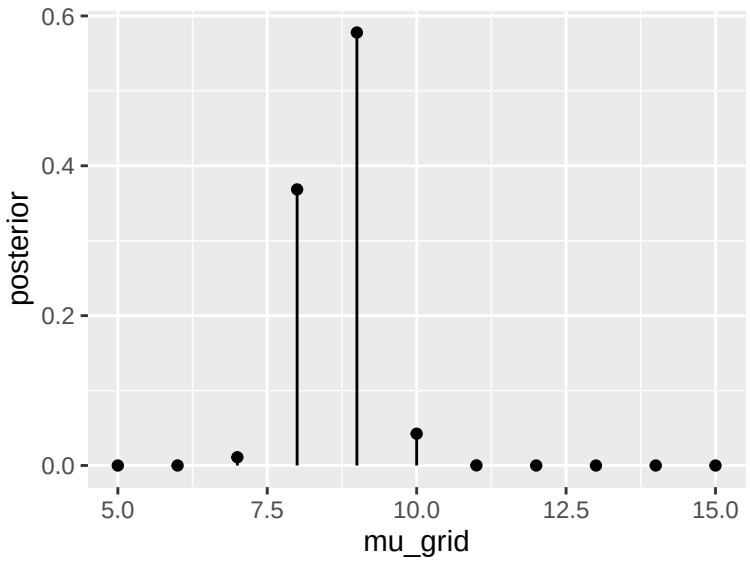
```
# PASO 2: Evaluar la priori y la verosimilitud para cada valor de mu

grid_data <- grid_data %>%
  mutate(
    prior = dnorm(mu_grid, 10, 1.2),
    # Priori: distribución normal(10,1.2**2)
    likelihood = sapply(mu_grid, function(m) prod(dnorm(y, mean = m, sd = 1.3)))
    # Verosimilitud:
  )
```

```
# PASO 3: Aproximación de la distribución a posteriori (no normalizada y normalizada)
grid_data <- grid_data %>%
  mutate(
    unnormalized = likelihood * prior,
```

```
    # Producto prior * verosimilitud
    posterior = unnormalized / sum(unnormalized)
    # Normalización para que sume 1
  )
```

```
# Verificar que la suma de las probabilidades que
# definen la aproximación computacional de la
# distribución a posteriori normalizada es 1
grid_data %>%
  summarize(sum(unnormalized), sum(posterior))
```

```
##   sum(unnormalized) sum(posterior)
## 1        0.00105926              1
```



2. Repite la parte (a) usando una grilla de 201 valores igualmente espaciados entre 5 y 15.

```
# PASO 1: Definir una grilla de 11 valores de mu
grid_data <- data.frame(mu_grid = seq(from = 5, to = 15, length.out = 201))
y <- c(7.1, 8.9, 8.4, 8.6)
```

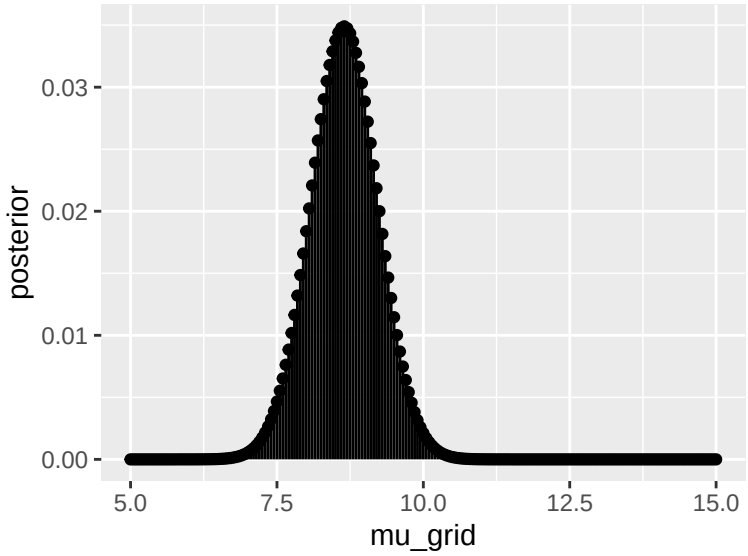
```
# PASO 2: Evaluar la priori y la verosimilitud para cada valor de mu

grid_data <- grid_data %>%
  mutate(
    prior = dnorm(mu_grid, 10, 1.2),
    # Priori: distribución normal(10,1.2**2)
    likelihood = sapply(mu_grid, function(m) prod(dnorm(y, mean = m, sd = 1.3)))
    # Verosimilitud:
  )
```

```
# PASO 3: Aproximación de la distribución a posteriori (no normalizada y normalizada)
grid_data <- grid_data %>%
```

```
  mutate(
    unnormalized = likelihood * prior,
    # Producto prior * verosimilitud
    posterior = unnormalized / sum(unnormalized)
    # Normalización para que sume 1
  )
```

```
# Verificar que la suma de las probabilidades que
# definen la aproximación computacional de la
# distribución a posteriori normalizada es 1
grid_data %>%
  summarize(sum(unnormalized), sum(posterior))
```

```
##   sum(unnormalized) sum(posterior)
## 1        0.02122574              1
```



Ejercicio 6.15 (MCMC con RStan: Gamma–Poisson)

Consideremos el modelo Gamma–Poisson para λ con

$$Y_i \mid \lambda \sim \text{Pois}(\lambda) \qquad \text{y} \qquad \lambda \sim \text{Gamma}(20,\, 5),$$

donde usamos la parametrización **Gamma(shape, rate)**. Observamos $n = 3$ datos independientes

$$(Y_1, Y_2, Y_3) = (0,\, 1,\, 0).$$

Objetivo

- Simular el modelo a posteriori de λ con **RStan** usando **4 cadenas** y **10 000 iteraciones por cadena**.
- Producir *trace plots* y densidades para las cuatro cadenas.
- A partir de las densidades, indicar el valor de λ más plausible a posteriori.

- Especificar el modelo a posteriori de λ por conjugación y comparar con la aproximación MCMC.

```
# Técnica de aproximación (MCMC) de la posteriori
# Produce cadenas dependientes de lambda a partir del modelo Gamma-Poisson.

# Ejemplo Gamma-Poisson
#
# PASO 1: Definición de la estructura del modelo bayesiano
gp_model <- "
  data {
    int<lower=0> N;            // número de observaciones
    int<lower=0> y[N];         // datos Poisson
    real<lower=0> alpha;       // shape de la Gamma prior
    real<lower=0> beta;        // rate  de la Gamma prior
  }
  parameters {
    real<lower=0> lambda;      // parámetro Poisson
  }
  model {
    lambda ~ gamma(alpha, beta);  // prior Gamma(shape, rate)
    y ~ poisson(lambda);          // verosimilitud
  }
"
```

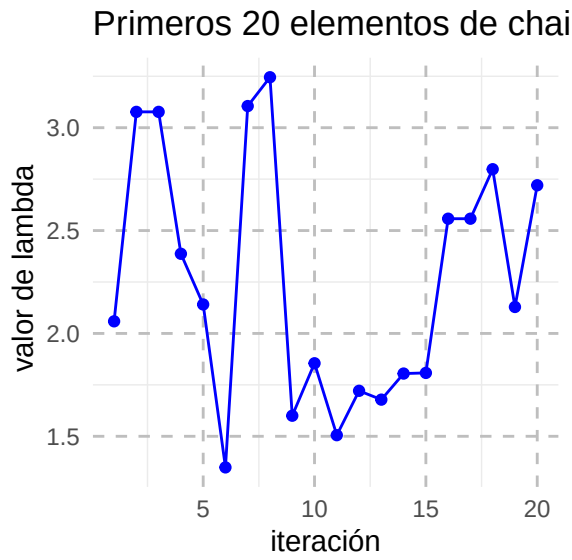
```
# Datos y prior
y <- c(0L, 1L, 0L)
N <- length(y)
alpha <- 20
beta  <- 5
```

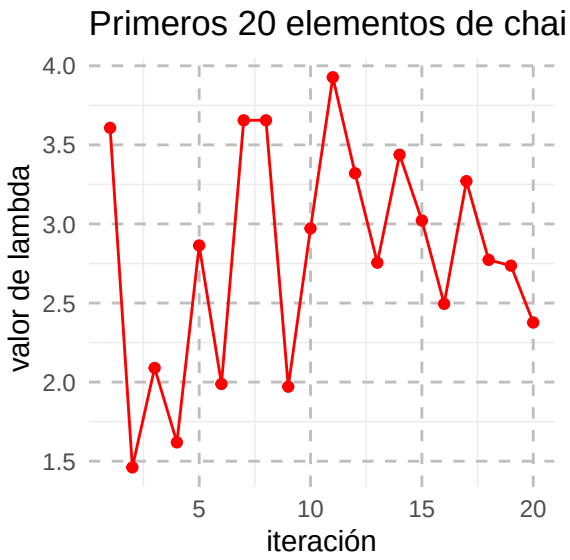
```
# PASO 2: Simulación de la distribución a posteriori con rstan::stan()
gp_sim <- stan(
  model_code = gp_model,
  data  = list(N = N, y = y, alpha = alpha, beta = beta),
  chains = 4, iter = 10000, warmup = 1000, seed = 84735
)
```

```
# Visualización de los primeros cuatro valores de las realizaciones de cada cadena
as.array(gp_sim, pars = "lambda") %>% head(4)
```

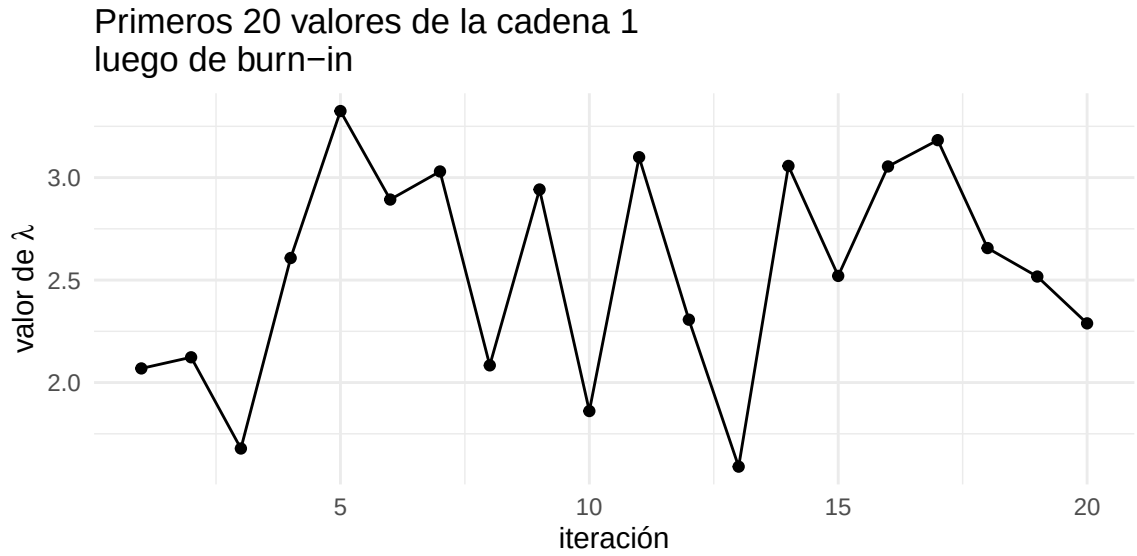
```
## , , parameters = lambda
##
##           chains
## iterations  chain:1  chain:2  chain:3  chain:4
##       [1,] 2.058805 2.953002 3.607447 3.039642
##       [2,] 3.076931 2.764217 1.460747 2.048070
##       [3,] 3.076931 2.764217 2.089549 1.884896
##       [4,] 2.387108 3.041081 1.618826 1.943340
```

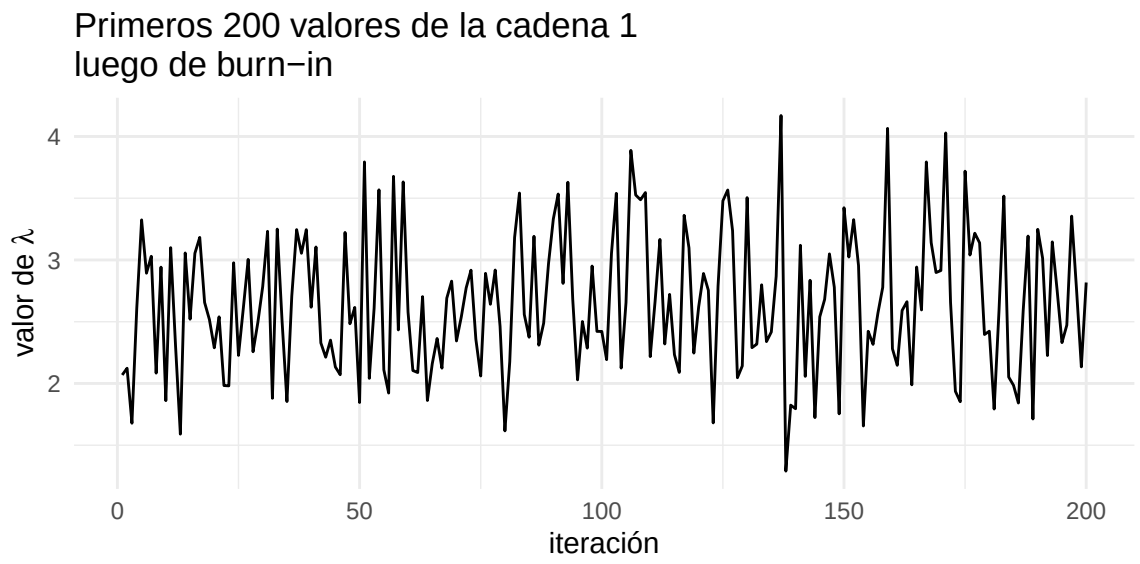
```
# Extraer los primeros 20 elementos de chain:1 y chain:3
samples_array <- as.array(gp_sim, pars = "lambda")
chain_1 <- samples_array[1:20, 1, 1]
chain_3 <- samples_array[1:20, 3, 1]
```



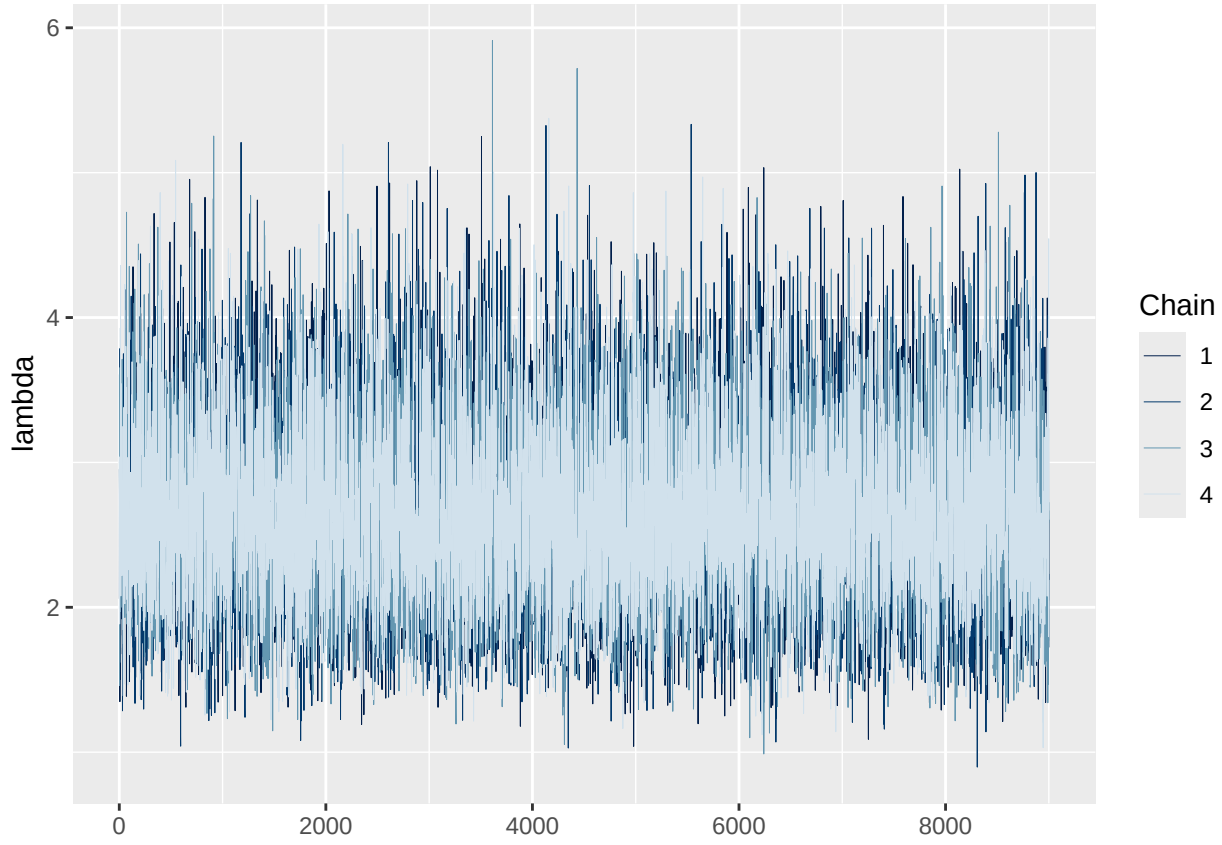



```
##  num [1:36000(1d)] 2.07 2.12 1.68 2.61 3.32 ...
##  - attr(*, "dimnames")=List of 1
##   ..$ iterations: NULL
```





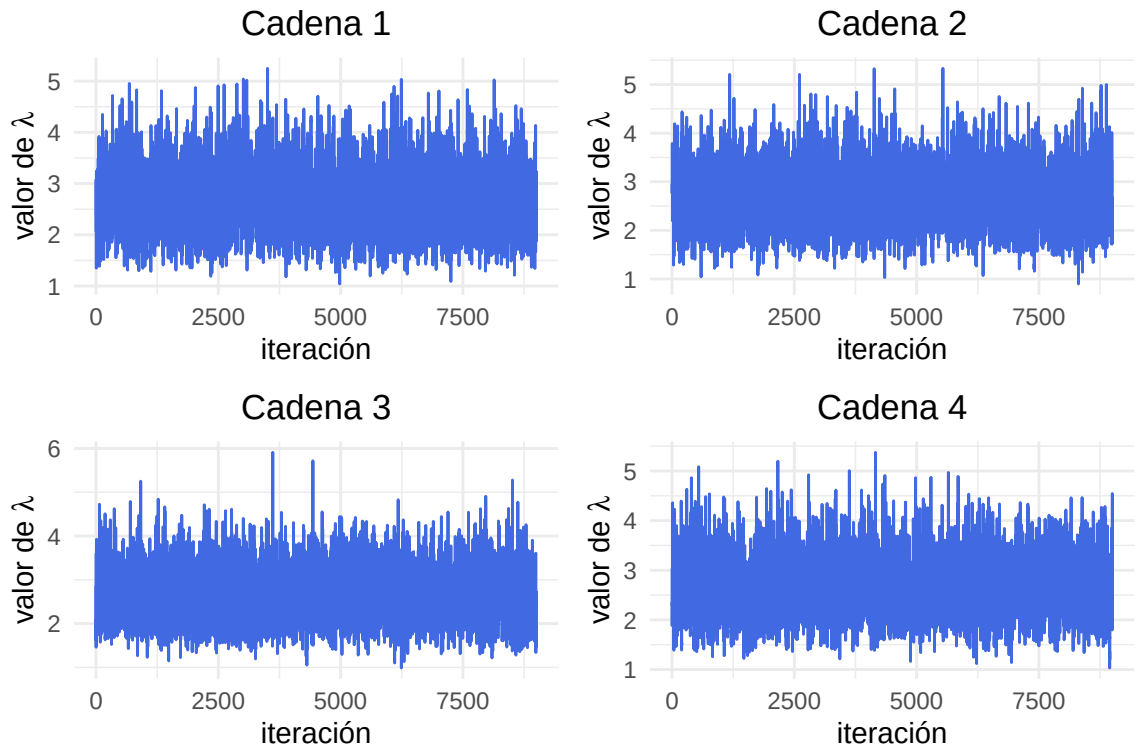

```
# Trazas de las cuatro cadenas de Markov para lambda
mcmc_trace(gp_sim, pars = "lambda", size = 0.1)
```



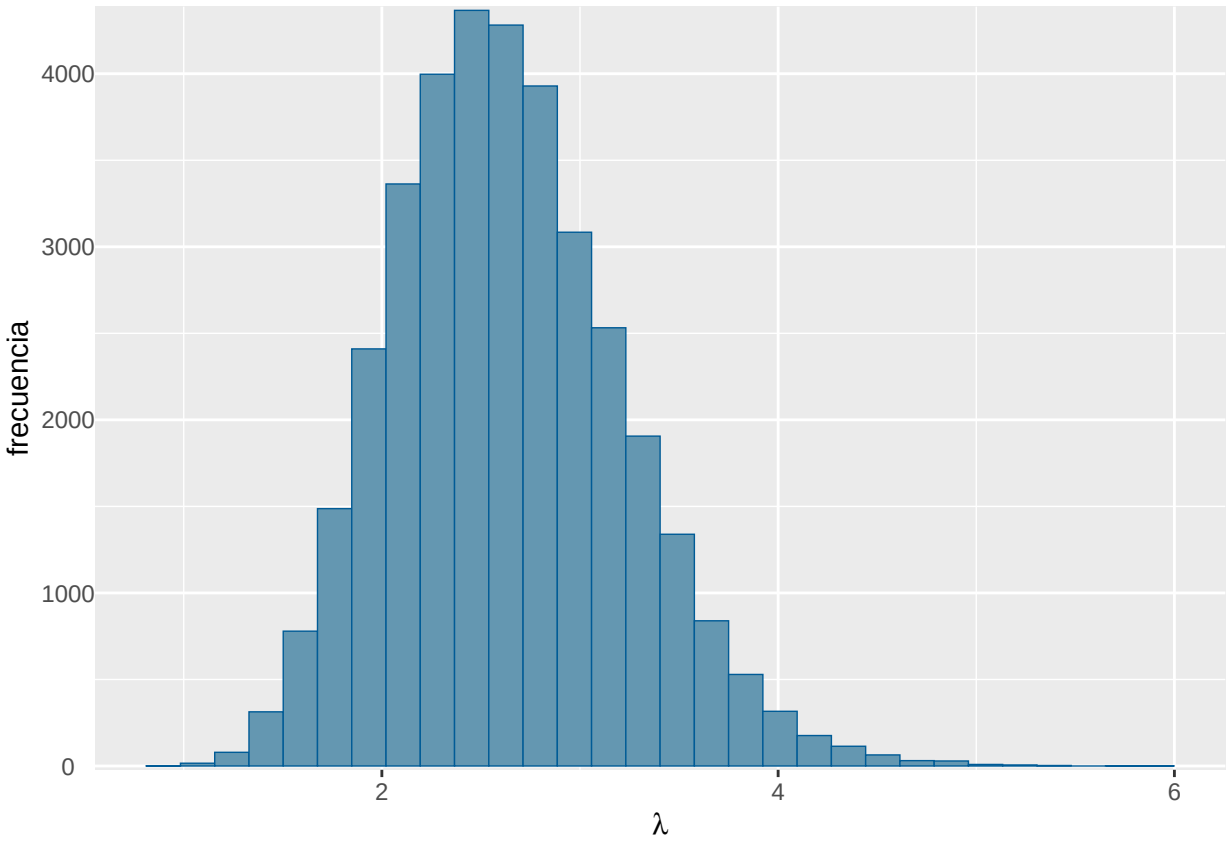

```
# Extraer todas las cadenas completas (array)
samples_array <- as.array(gp_sim, pars = "lambda")
chain_1 <- samples_array[, 1, 1]
```

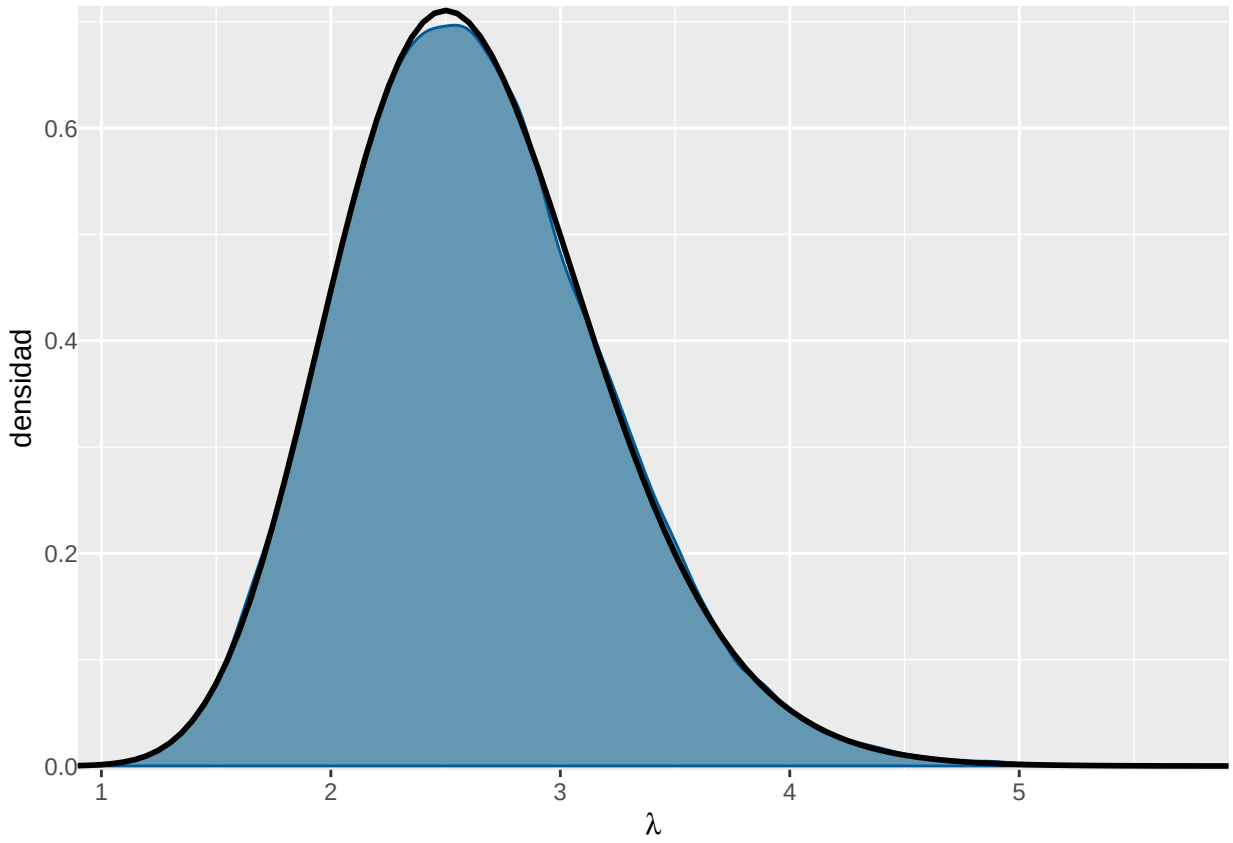
```r
chain_2 <- samples_array[, 2, 1]
chain_3 <- samples_array[, 3, 1]
chain_4 <- samples_array[, 4, 1]
```

```r
# Función para graficar una cadena
grafico_cadena <- function(valores, titulo) {
  ggplot(data.frame(iteracion = 1:length(valores), lambda = valores),
         aes(x = iteracion, y = lambda)) +
    geom_line(color = "royalblue") +
    labs(title = titulo, x = "iteración", y = expression("valor de " * lambda)) +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5))
}
```

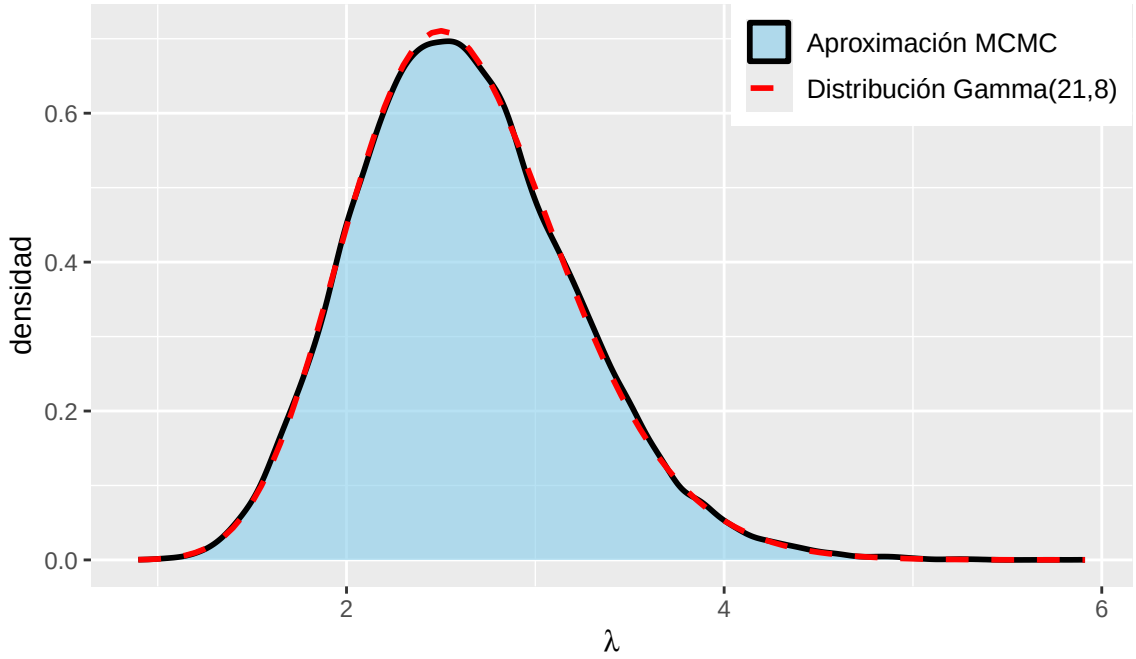



```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```

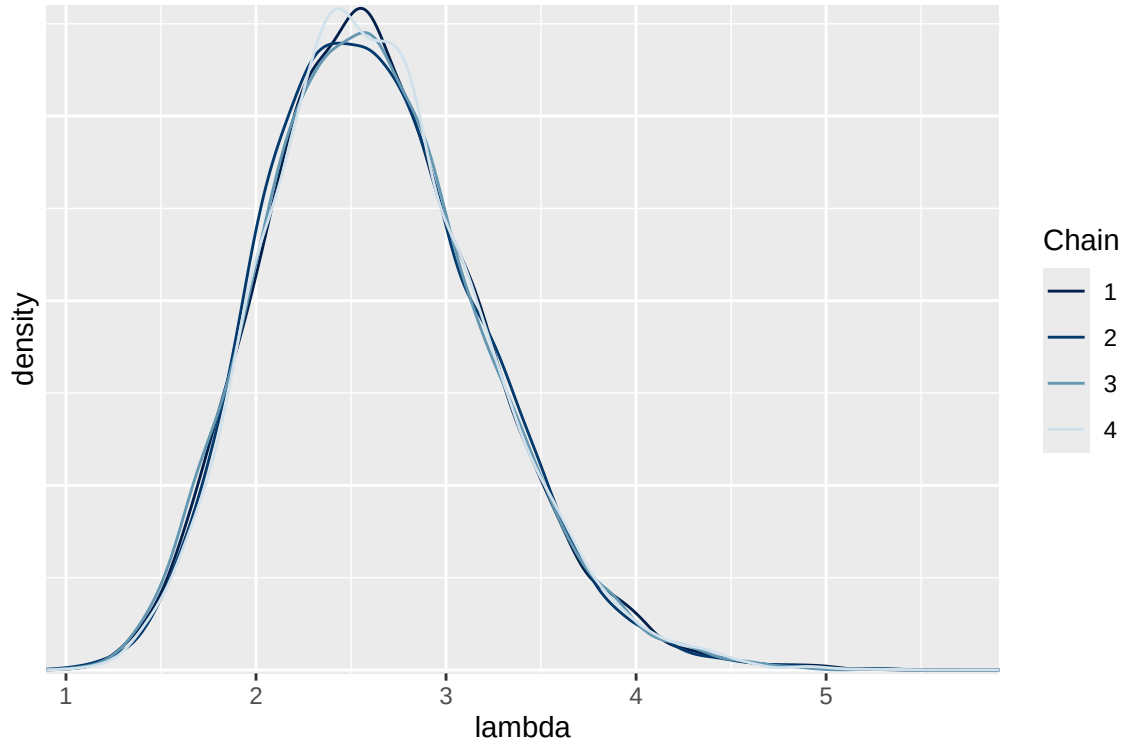

4000
3000
2000
1000
0
frecuencia
2
4
6
λ



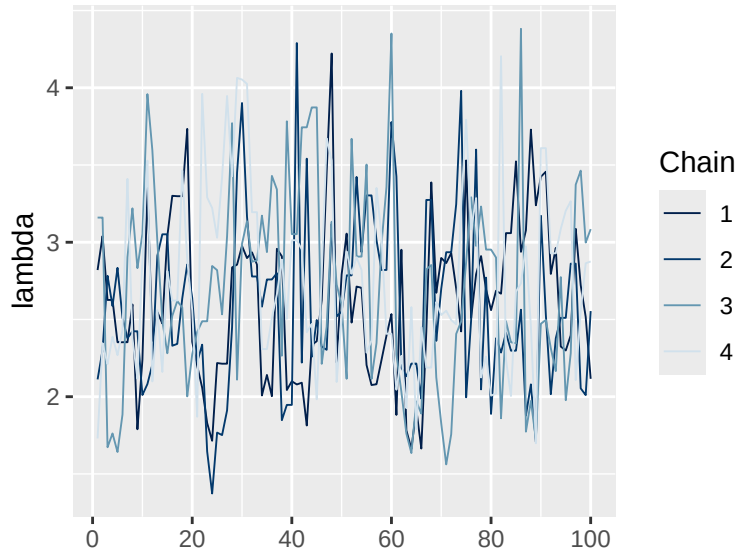
Comparación entre la aproximación MCMC de la posterior de .
y la distribución Gamma(21,8)



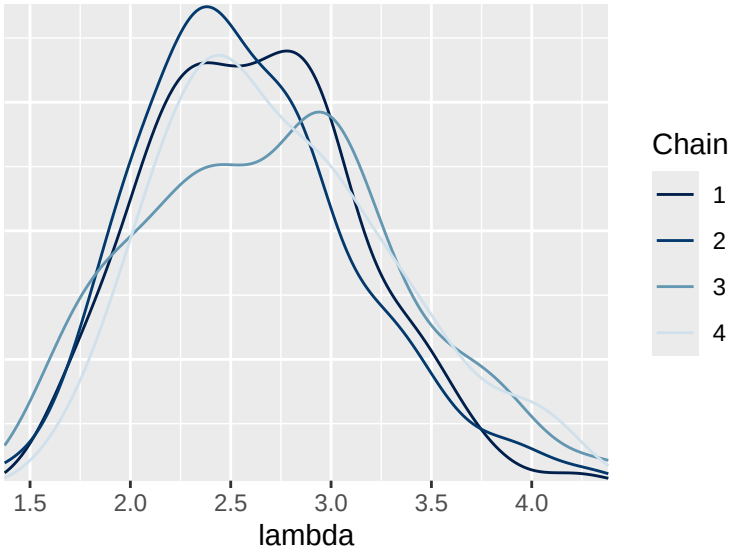
```r
# Densidades por cadena superpuestas
mcmc_dens_overlay(gp_sim, pars = "lambda") + ylab("density")
```




```r
# Simulación corta para ilustrar convergencia pobre
gp_sim_short <- stan(model_code = gp_model,
                     data = list(N = N, y = y, alpha = alpha, beta = beta),
                     chains = 4, iter = 100 * 2, seed = 84735)

mcmc_trace(gp_sim_short, pars = "lambda")
```



```r
mcmc_dens_overlay(gp_sim_short, pars = "lambda")
```



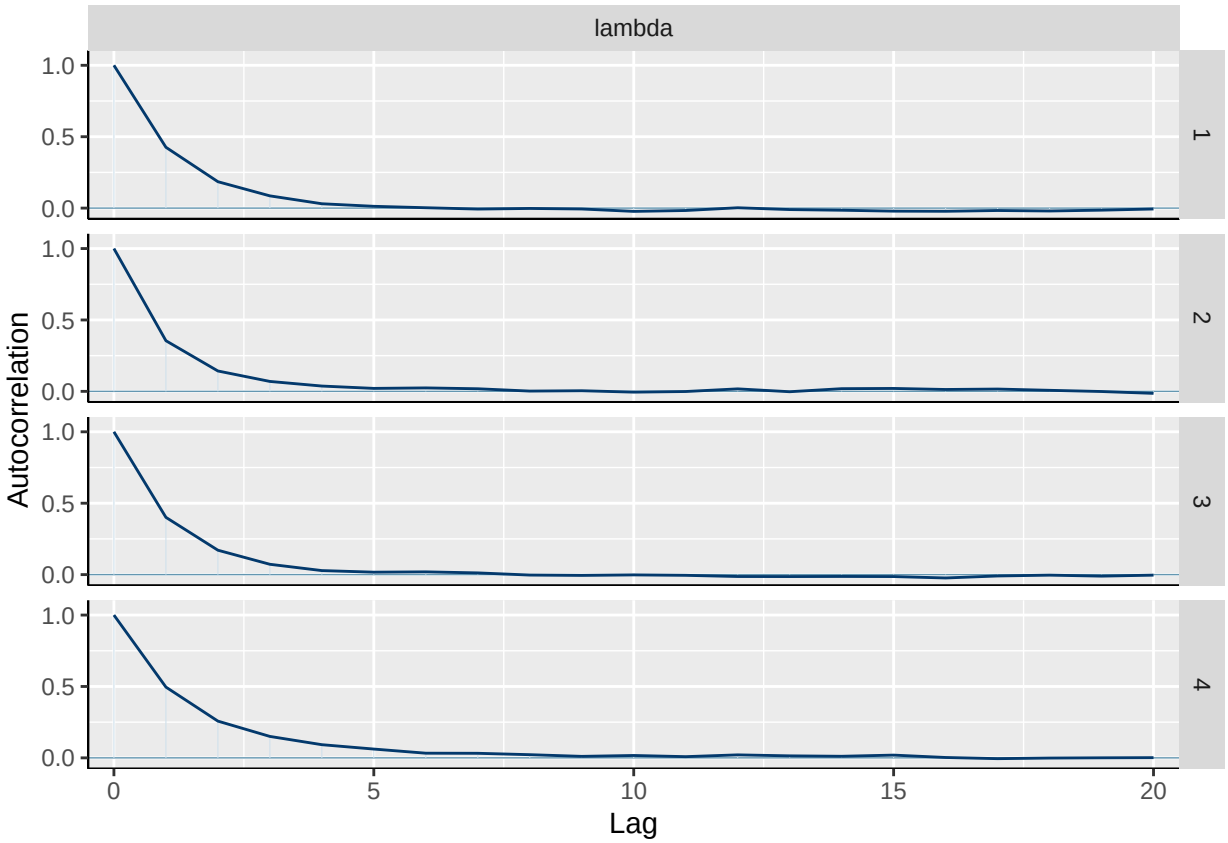

```r
# Diagnósticos: tamaño efectivo, R-hat y autocorrelación
neff_ratio(gp_sim, pars = c("lambda"))
```

```
## [1] 0.3802014
```

```r
rhat(gp_sim, pars = "lambda")
```

```
## [1] 1.000029
```

```r
mcmc_acf(gp_sim, pars = "lambda")
```


lambda
Autocorrelation
1.0
0.5
0.0
1
2
3
4
0
5
10
15
20
Lag

Condiciones iniciales para el capitulo 7

7.9.2 Práctica: Simulación Normal-Normal

En el siguiente conjunto de ejercicios, volvemos al modelo bayesiano del apartado (7.1):

$$Y \mid \mu \sim N(\mu, 0.75^2), \quad \mu \sim N(0, 1^2)$$

Supongamos que observamos Y = 6.25 y deseamos construir una simulación de Metropolis–Hastings del modelo posterior correspondiente para λ

Ejercicio 7.9 (Una iteración con un modelo de propuesta Uniforme)

La función `one_mh_iteration()` del texto utiliza un modelo de propuesta Uniforme:

$$\mu' \mid \mu \sim \text{Unif}(\mu - w,\ \mu + w)$$

con ancho w=1.

Partiendo de un valor actual de $\mu = 3$ y utilizando `set.seed(1)`, ejecuta el siguiente código y comenta sobre los valores retornados de `proposal`, `alpha`, y `next_stop`:

```
one_mh_iteration(w = 0.01, current = 3)
one_mh_iteration(w = 0.5, current = 3)
one_mh_iteration(w = 1, current = 3)
one_mh_iteration(w = 3, current = 3)
```

Primero seteamos la semilla en 1 y aplicamos la función `one_mh_iteration()` al valor inicial $\mu = 3$ con un ancho de $w = 0.01$.

proposal	alpha	next_stop
3.0026	1	3.0026

Con nuestros primeros valores, el algoritmo partió de $\mu = 3$ y, dado que el ancho era muy pequeño ($w = 0.01$), la propuesta resultó ser apenas distinta: $\mu' = 3.0026$. Al comparar la plausibilidad del valor actual y de la propuesta bajo la posterior, ambas resultaron prácticamente iguales, de modo que la razón de aceptación alcanzó el valor máximo posible, $\alpha = 1$. Eso implica que la propuesta se acepta sin ninguna duda, y por lo tanto la cadena se mueve al nuevo valor, quedando en $\mu = 3.0026$. Esto tiene lógica porque, al ser la propuesta muy cercana al valor actual, tanto la prior como la verosimilitud cambian mínimamente, lo que hace que la posterior apenas varíe y, por ende, la aceptación sea prácticamente segura.

Ahora aplicamos la función `one_mh_iteration()` al mismo valor inicial $\mu = 3$, pero aumentando el ancho a $w = 0.5$.

proposal	alpha	next_stop
2.98	0.95	2.98

A diferencia de la iteración anterior con $w = 0.01$, donde la propuesta estaba prácticamente pegada al valor actual, esta vez la propuesta $\mu' = 2.98$ se aleja un poco más de $\mu = 3$. Como resultado, la plausibilidad bajo la posterior cambia un poco más, y la razón de aceptación $\alpha = 0.95$ ya no es 1, aunque sigue siendo alta. Esto significa que la propuesta se acepta con una probabilidad cercana a 1, lo que refleja que la cadena puede

moverse pasos más grandes cuando el ancho es mayor. La lógica detrás de esto es permitirnos explorar un rango más amplio alrededor del valor actual, la cadena puede realizar saltos más amplios, manteniendo el equilibrio entre explorar nuevas regiones y quedarse en zonas de alta probabilidad, lo que nos asegura una buena exploración del espacio posterior.

A continuación, usamos `one_mh_iteration()` con el mismo valor inicial $\mu = 3$, pero aumentando aún más el ancho a $w = 1$.

proposal	alpha	next_stop
2.87	0.67	3

En comparación con las iteraciones anteriores, la propuesta $\mu' = 2.87$ se aleja bastante del valor actual. Esto provoca un cambio más marcado en la plausibilidad bajo la posterior, y la razón de aceptación cae a $\alpha = 0.67$. Como resultado, la propuesta no se acepta en esta ocasión, y la cadena permanece en $\mu = 3$. La lógica de este comportamiento es consistente con el algoritmo, al ampliar el rango de propuestas, se permiten saltos más grandes que exploran más el espacio, pero también aumenta la probabilidad de proponer valores menos compatibles con los datos y la prior. Así, el algoritmo equilibra exploración y aceptación, evitando moverse automáticamente a cualquier propuesta que no sea suficientemente plausible.

Finalmente, aplicamos `one_mh_iteration()` con el valor inicial $\mu = 3$ y un ancho aún mayor, $w = 3$.

proposal	alpha	next_stop
2.63	0.29	3

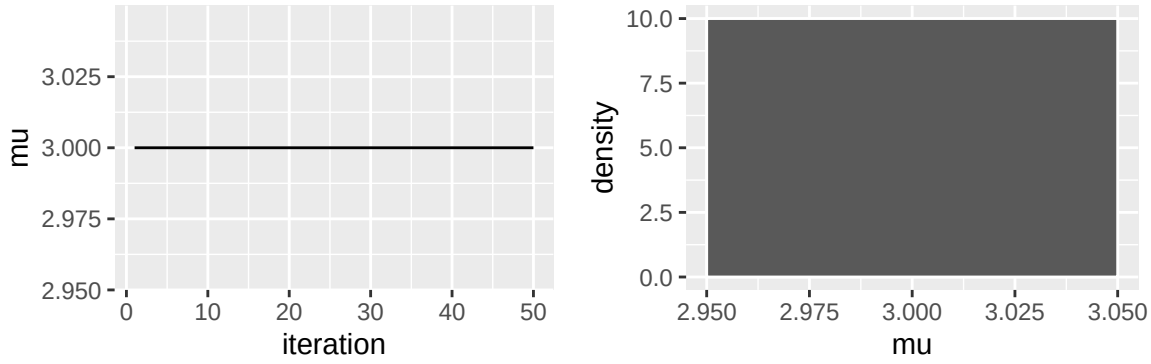
En esta iteración, la propuesta $\mu' = 2.63$ se aleja significativamente del valor actual, mucho más que en las pruebas anteriores. Esto provoca un cambio considerable en la plausibilidad de la posterior, y la razón de aceptación disminuye notablemente a $\alpha = 0.29$. Como resultado, la propuesta no se acepta, y la cadena permanece en $\mu = 3$. La lógica detrás de este comportamiento es coherente con el diseño del algoritmo: al generar propuestas que se alejan demasiado del valor actual, la probabilidad de aceptar un valor menos compatible con los datos y la prior disminuye, lo que evita saltos grandes que podrían llevar a regiones de baja probabilidad.

En pocas palabras, con w pequeño la cadena avanza despacio y casi siempre acepta, mientras que con w más grande las propuestas se alejan más y se aceptan menos. Elegir w es un equilibrio entre pasos seguros y explorar más el espacio.

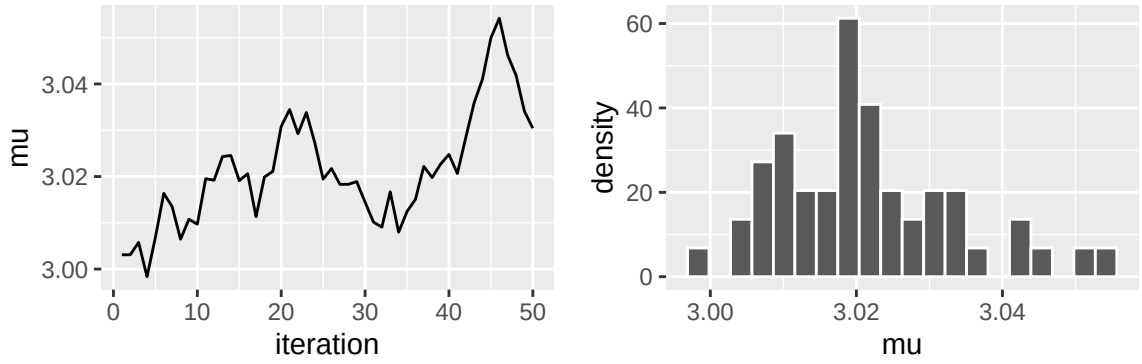
Ejercicio 7.10 (Un recorrido completo con un modelo de propuesta Uniforme)

Implementa la función de Metropolis-Hastings `mh_tour()` definida en la Sección 7.3 para construir recorridos de μ bajo cada uno de los siguientes escenarios. Construye gráficos de trazas e histogramas para cada recorrido:

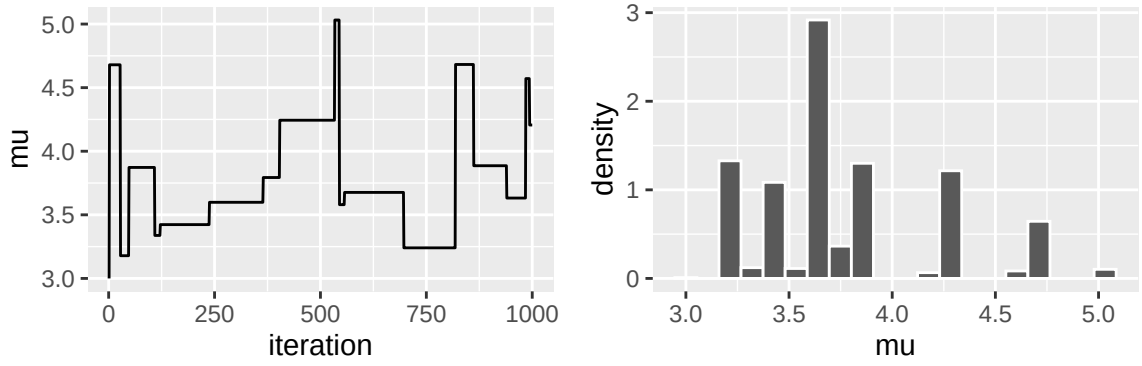
1. 50 iteraciones, $w = 50$



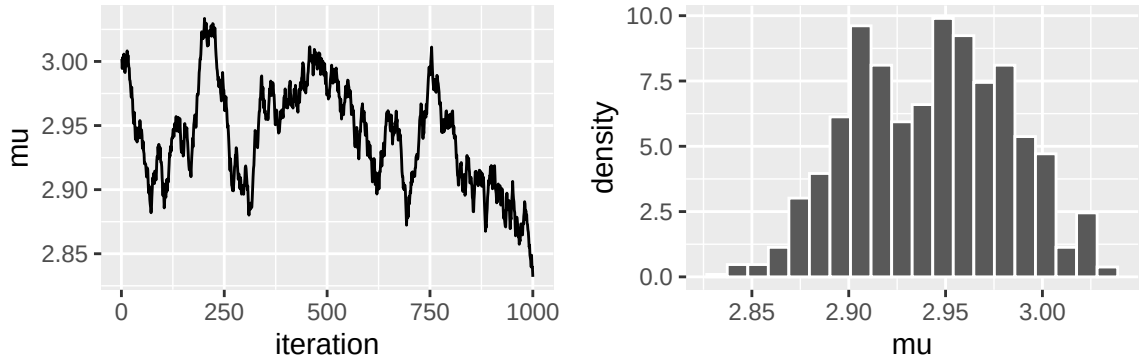
2. 50 iteraciones, $w = 0.01$



3. 1000 iteraciones, $w = 50$



4. 1000 iteraciones, $w = 0.01$



5. Contrasta los gráficos de trazas de los apartados a y b. Explica por qué cambiar w produce este efecto.

Cuando el valor de w es muy grande (como $w = 50$), las propuestas se generan lejos del centro de la distribución objetivo y la mayoría son rechazadas, lo que provoca que la cadena se quede estancada largos periodos en un mismo valor y se observen trazas en forma de escalones. En cambio, cuando w es muy pequeño (por ejemplo, $w = 0.01$), casi todas las propuestas son aceptadas, pero los pasos son diminutos, por lo que la cadena avanza lentamente y apenas logra explorar la distribución en pocas iteraciones. En pocas palabras, w define qué tan grandes son los pasos que da la cadena y marca un balance entre moverse y aceptar propuestas, si w es muy grande, casi nunca se aceptan los saltos y la cadena se queda trabada, si es muy chico, se aceptan casi todos, pero los movimientos son tan mínimos que no se explora bien la distribución. Para que el algoritmo funcione de forma eficiente conviene elegir un w intermedio

6. Considera los resultados de los apartados c y d. ¿Es el valor de w igualmente importante cuando el número de iteraciones es mucho mayor? Explica.

Cuando aumentamos mucho el número de iteraciones, el valor de w sigue siendo importante, pero su impacto inmediato se atenúa. Con pocas iteraciones, un w inadecuado (muy grande o muy pequeño) impide que la cadena explore correctamente la distribución, y el resultado es un muestreo muy pobre. En cambio, con muchas iteraciones, incluso si w no es óptimo, la cadena eventualmente logrará recorrer más espacio y aproximarse a la distribución objetivo. Sin embargo, un w mal elegido hace que la cadena sea mucho menos eficiente: tardará mucho más en mezclarse y en producir muestras útiles (alta autocorrelación). Por lo tanto, aunque con muchas iteraciones la cadena puede compensar en parte una mala elección de w, sigue siendo preferible ajustar w a la escala de la distribución para lograr convergencia y mezcla más rápida.