Tarea 6: Ejercicios 6.7, 6.15, 7.9 y 7.12

Santiago Robatto, Sofía Terra

Exercise 6.7 (Normal-Normal grid approximation)

Exercise 6.7 (Normal-Normal grid approximation) Consider the Normal-Normal model for μ with $Y_i|\mu \sim N(\mu, 1.3^2)$ and $\mu \sim N(10, 1.2^2)$. Suppose that on $n = 4$ independent observations, you observe data $(Y_1, Y_2, Y_3, Y_4) = (7.1, 8.9, 8.4, 8.6)$.

a. Utilize grid approximation with grid values $\mu \in \{5, 6, 7, \ldots, 15\}$ to approximate the posterior model of μ.

b. Repeat part a using a grid of 201 equally spaced values between 5 and 15.

Respuestas:

Este ejercicio busca que realicemos una aproximacion por grilla para una distribución a posteriori. Lo que tenemos que hacer es convertir un problema del mundo continuo a uno discreto, más simple. Para lograr esto, realizaremos los siguientes pasos: 1. Definir el rango de parámetros en una grilla de puntos. 2. Calcular la plausibilidad posteriori en cada punto. 3. Normalizar para convertir esas plausibilidades en probabilidades. 4. Hacer un muestreo de esos puntos según sus probabilidades.

Parte A:

Como primer paso definimos la grilla de valores posibles de μ, entre 5 y 15 by 1, dado que es el recorrido indicado en la letra.

Table 1: Valores de mu en la grilla

5	6	7	8	9	10	11	12	13	14	15

Para cada valor de μ evaluaremos el prior y la verosimilitud.

Para el prior utilizaremos el dato de letra que $\mu \sim \mathcal{N}(10, 1.2^2)$ y evaluaremos eso.

A su vez, creamos un vector con el valor de las observaciones $(Y_1, Y_2, Y_3, Y_4) = (7.1,\ 8.9,\ 8.4,\ 8.6)$ para poder hallar la verosimilitud.

Table 2: Primeros 6 datos de Grid Data

mu_grid	prior	likelihood
5	5.646918e-05	1.889500e-08
6	1.285232e-03	1.267971e-05
7	1.460692e-02	7.979256e-04
8	8.289762e-02	4.708682e-03
9	2.349266e-01	2.605676e-03
10	3.324519e-01	1.352152e-04

Finalmente ya podemos aproximar el posterior, haciendo el producto de la verosimilitud con la priori y dividiendo entre su suma. Hacer el cociente lo que hace es asegurarnos de obtener una probabilidad, dado que estamos normalizando la función.

$$p(\mu_j \mid y) = \frac{p(y \mid \mu_j)\, p(\mu_j)}{\sum_k p(y \mid \mu_k)\, p(\mu_k)}$$

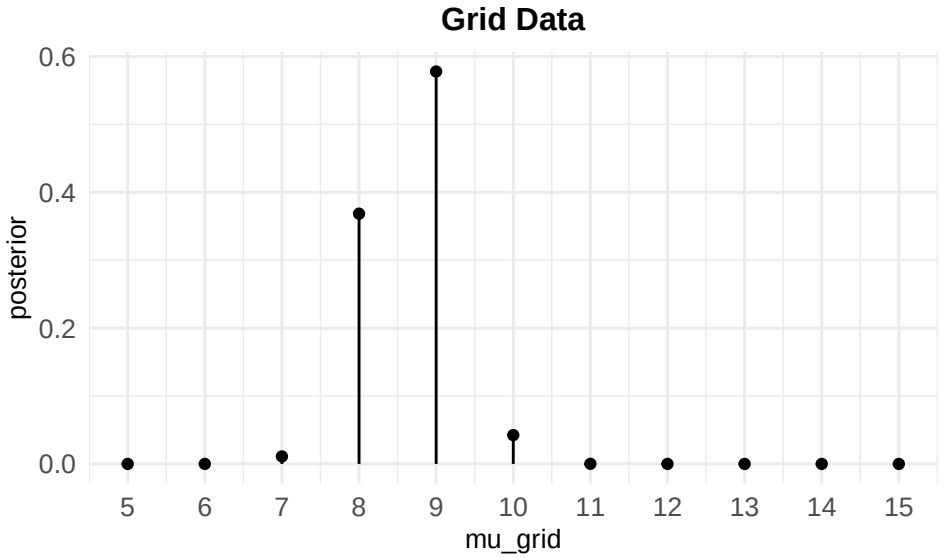
Table 3: Primeros 6 datos de Grid Data Actualizada (con posteriori)

mu_grid	prior	likelihood	unnormalized	posterior
5	5.646918e-05	1.889500e-08	1.000000e-12	1.007000e-09
6	1.285232e-03	1.267971e-05	1.629600e-08	1.538469e-05
7	1.460692e-02	7.979256e-04	1.165523e-05	1.100319e-02
8	8.289762e-02	4.708682e-03	3.903385e-04	3.685013e-01
9	2.349266e-01	2.605676e-03	6.121424e-04	5.778965e-01
10	3.324519e-01	1.352152e-04	4.495254e-05	4.243770e-02

El siguiente paso se realiza sólo para verificar que todo esté funcionando bien. El hecho de que la suma de la variable posterior sume 1 nos asegura que esta efectivamente sea una probabilidad.

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

Graficamos el posterior obtenido en grid data, el cual nos va a devolver los valores más probables de μ. De esta manera, a partir de un priori normal (continuo) y 4 observaciones, generamos una distribución a posteriori discreta para μ. Los valores mas probables son 8 y 9, pero al usar solo 10 valores para modelar μ perdimos mucha precisión. Es por ello que repetiremos el experimento en la parte b, pero usando 201 valores posibles para μ.



Además, podemos realizar un muestreo a partir de lo obtenido anteriormente y compararlo contra el posterior real:

Table 4: Medidas de resumen Normal-Normal

model	mean	mode	var	sd
prior	10.000	10.000	1.4400	1.2000
posterior	8.647	8.647	0.3267	0.5715

Por ende sabemos que el posterior teórico del ejemplo distribuye $\mu \sim \mathcal{N}(8.64698, 0.571^2)$

Para el muestreo:

```
mu_grid     n percent
      7   112  0.0112
      8  3687  0.3687
      9  5770  0.5770
     10   430  0.0430
```

```
   11     1  0.0001
Total 10000  1.0000
```

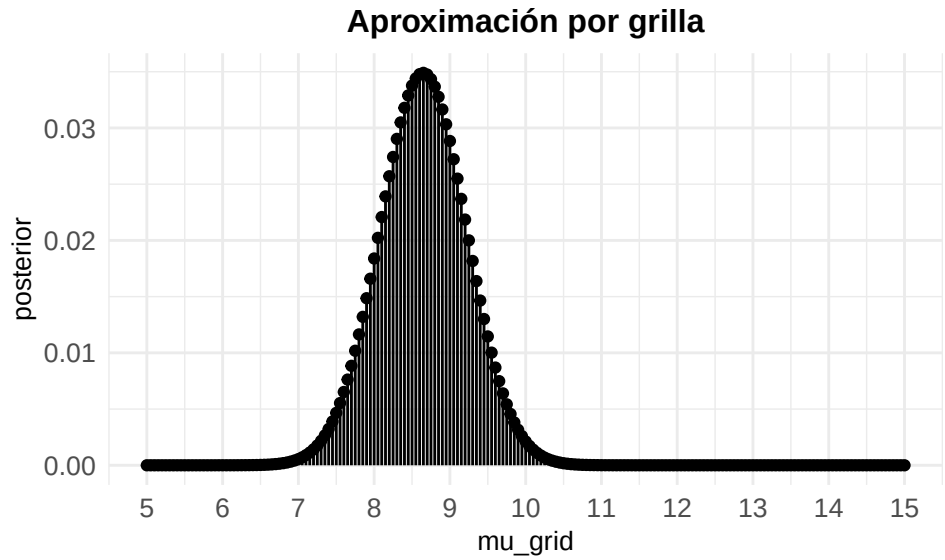
De esta manera vemos que el histograma de aproximación por grilla es sumamente similar al posterior real.

Sin embargo, como ya vimos antes, al haber elegido pocos valores posibles para μ la aproximación no es precisa.

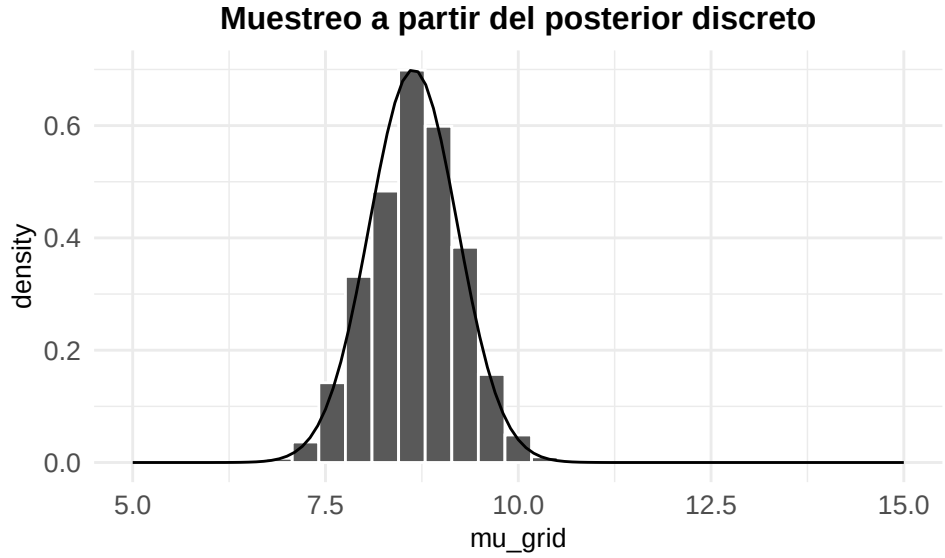
Parte B:

Replicaremos el código realizado en la parte anterior, con la única diferencia que la secuencia para valores de μ será de a 0.05 en vez de a 1.

Ahora, al haber graficado para 201 casos, la aproximación por grilla se asemeja de manera casi perfecta a la normal del posteriori que calculamos teóricamente. Al tener la variable discreta, pero ser la diferencia entre los valores muy pequeña, se aproxima en mayor proporción a una normal.



El muestreo a partir del posterior discreto produce datos cuya distribución prácticamente coincide con la densidad normal teórica del posterior, lo que sugiere que nuestra aproximación por grilla es adecuada para modelar μ



Exercise 6.15 (MCMC with RStan: Gamma-Poisson)

Exercise 6.15 (MCMC with RStan: Gamma-Poisson) Consider the Gamma-Poisson model for λ with $Y_i|\lambda \sim \text{Pois}(\lambda)$ and $\lambda \sim \text{Gamma}(20, 5)$. Suppose that you observe $n = 3$ independent data points $(Y_1, Y_2, Y_3) = (0, 1, 0)$.

a. Simulate the posterior model of λ with RStan using 4 chains and 10000 iterations per chain.

b. Produce trace and density plots for all four chains.

c. From the density plots, what seems to be the most posterior plausible value of λ?

d. Hearkening back to Chapter 5, specify the posterior model of λ. How does your MCMC approximation compare?

Respuestas:

Ahora trabajamos con un modelo más complejo, donde sería difícil calcular la distribución a posteriori de forma directa. Por eso recurrimos a las Caddenas de Markov Monte Carlo (MCMC). Esta técnica computacional permite aproximar distribuciones mediante la generación de una cadena de muestras aleatorias, que, una vez estabilizada, converge a la distribución posteriori que estamos buscando. Para realizar esto utilizamos RStan, que nos permite utilizar herramientas de Stan en R.

El primer paso que debemos hacer es definir el modelo. Para ello utilizaremos la estructura base de Stan: definir el recorrido de los datos, los parámetros y por último el modelo. Para definir los datos utilizaremos $(Y_1, Y_2, Y_3) = (0, 1, 0)$, que representa el vector de observaciones Y. En los parametros usaremos λ y lo definiremos en $\mathbb{R}^+$, dado que es el recorrido de cualquier distribución poisson. Finalmente definiremos la distribución de Y y λ, con el modelo gamma-poisson. Sabemos que los datos siguen una distribución Poisson (λ) y que $\lambda \sim \text{Gamma}(20, 5)$. La definición del modelo por ende es la siguiente:

```
y <- c(0,1,0)
gp_model <- "
  data {
   int<lower = 0> N;
   array[N] int<lower=0> Y;
  }
  parameters {
    real<lower = 0 > lambda;
  }
  model {
    lambda ~ gamma (20,5);
    Y ~ poisson (lambda);
```

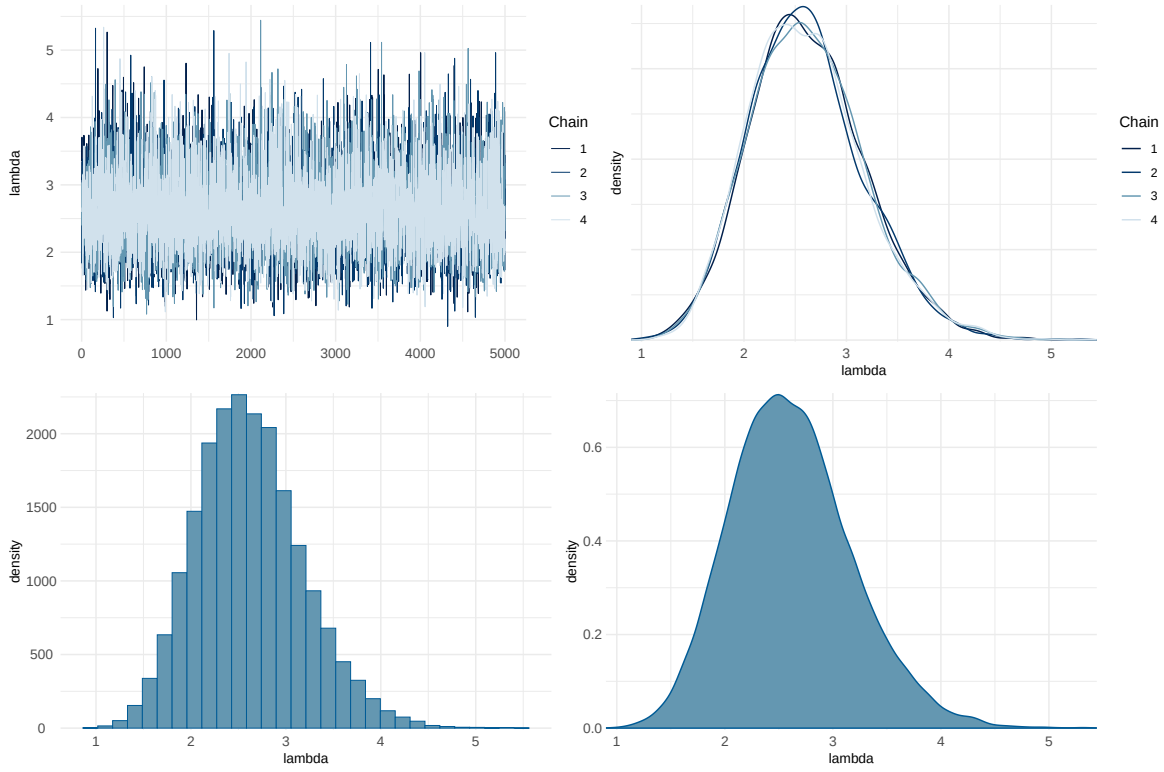
```
  }
"
```

El siguiente paso es utilizar la función stan para simular las 4 cadenas:

Se observa que se va haciendo la simulación de cada una, indicando el total de iteraciones realizadas y el tiempo que llevó completar la simulación. Cabe destacar que cuantas más cadenas simulemos, más tiempo va a tardar. Es un algoritmo "costoso" en términos de recursos y tiempo.

Utilizamos las funciones mcmc_trace para graficar la traza, densoverlay para ver todas las densidades superpuestas, mcmc_hist para calcular el histograma y mcmc_dens para calcular la función de densidad.

A continuación observamos los resultados de la simulación:

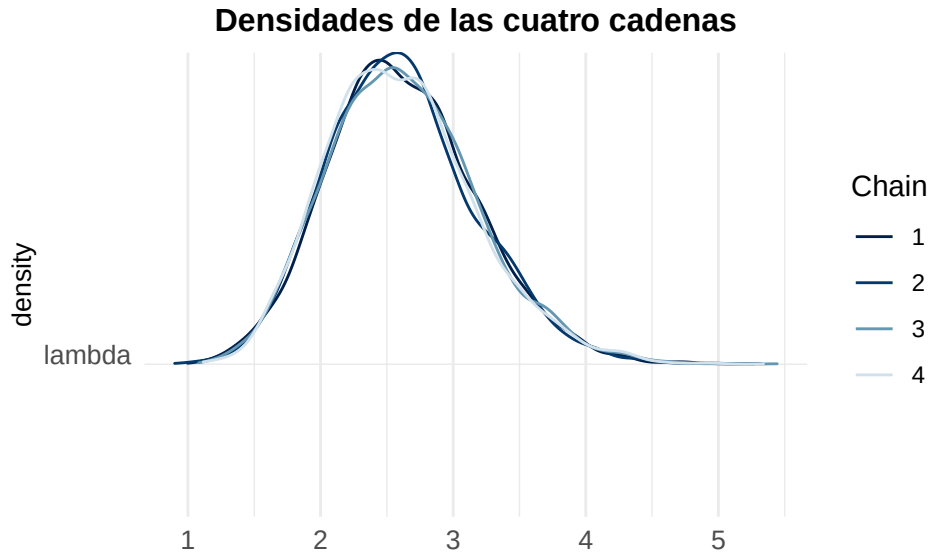


Obtuvimos una gráfica que muestra las distintas trazas de nuestras cadenas, otra que muestra sus densidades, un histograma y una función de densidad de todo lo simulado.

Observamos que la simulación convergió correctamente. Las trazas oscilan de forma estable. Asimismo, es casi imposible distinguir las cadenas, ya que están todas superpuestas. Esto es lo

que buscamos, es ideal, ya que todas las cadenas independientes nos dan resultados parecidos, a pesar de haber empezado de puntos diferentes, esto es naturaleza bayesiana en su escencia.

Ahora observemos más detalladamente las densidades de las distintas cadenas:



Las cuatro gráficas de densidad estan casi perfectamente superpuestas. Esto confirma lo que vimos en el gráfico de las trazas. La distribución final es casi igual para las cuatro cadenas. Analizando el gráfico concluimos que el valor más plausible de λ es aproximadamente 2.5.

Finalmente, vamos a comparar la densidad simulada con la teórica, vamos a analizar una distribución Gamma.

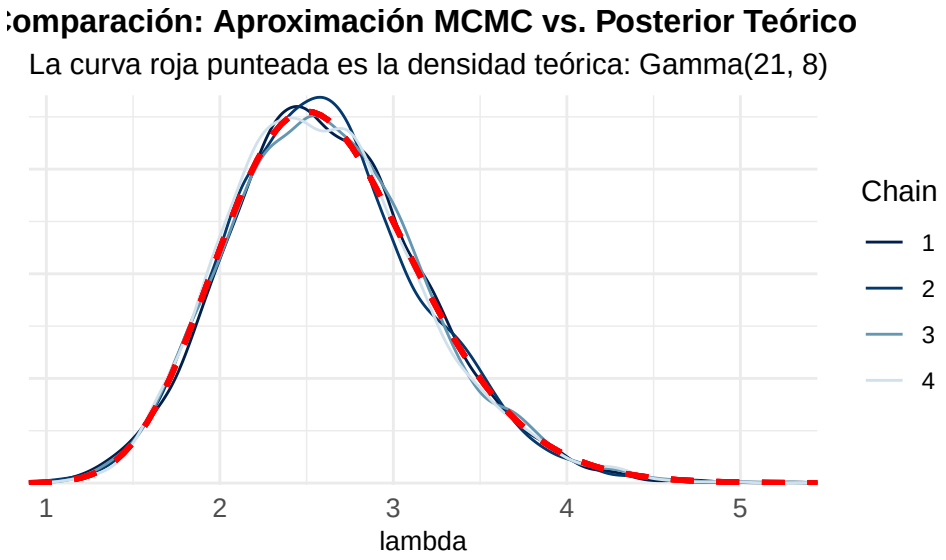
Nuestro modelo inicial es Gamma(α, β). Al observar n datos con una suma de $\sum_{i=1}^{n} Y_i$, obtenemos el posterior:

$$\lambda \mid Y \sim \text{Gamma}\left(\alpha + \sum_{i=1}^{n} Y_i, \beta + n\right)$$

Aplicando esto a nuestros datos obtenemos: Gamma$(20, 5) \implies \alpha = 20, \beta = 5$ $0, 1, 0) \implies n = 3$ y $\sum Y_i = 0 + 1 + 0 = 1$

Ahora sustituimos en la fórmula: $\alpha' = 20 + 1 = 21$ $\beta' = 5 + 3 = 8$

Obtenemos que la distribución teórica de λ es Gamma$(21, 8)$. Ahora la graficamos junto con lo que obtuvimos antes:



Observamos que el valor más plausible de λ también se encuentra en torno al 2.5. Esto significa que nuestra simulación MCMC funcionó correctamente, se acercó extremadamente bien a la distribución teórica. Esta aproximación es muy potente y muy acertada.

Exercise 7.9: One iteration with a Uniform proposal model

Exercise 7.9 (One iteration with a Uniform proposal model) The function `one_mh_iteration()` from the text utilizes a Uniform proposal model, $\mu'|\mu \sim \text{Unif}(\mu - w, \mu + w)$ with half-width $w = 1$. Starting from a current value of $\mu = 3$ and using `set.seed(1)`, run the code below and comment on the returned `proposal`, `alpha`, and `next_stop` values.

a. `one_mh_iteration(w = 0.01, current = 3)`

b. `one_mh_iteration(w = 0.5, current = 3)`

c. `one_mh_iteration(w = 1, current = 3)`

d. `one_mh_iteration(w = 3, current = 3)`

Utilizaremos un algoritmo de Metropolis Hastings para realizar la iteración. En este caso particular, dado que tomaremos un proposal simétrico, el algoritmo tiene otro nombre y se denomina Metropolis algorithm.

Este algorito tiene propiedades que simplifican las reglas de aceptación de moverse o no a la nueva ubicación (location) propuesta de μ.

Tras analizar tal como se hace en el libro (reescribiendo y desestimando la constante de normalización), la regla de decisión se simplifica a comparar si el posterior evaluado en μ es mayor que en μ'.

$$\alpha = \min\left\{1, \frac{f(\mu' \mid y)}{f(\mu \mid y)}\right\}.$$

Cuando $f(\mu' \mid y) > f(\mu \mid y)$ entonces el mínimo es 1 y nos movemos a la nueva ubicación.

Cuando $f(\mu' \mid y) < f(\mu \mid y)$ podríamos aceptarlo o no, y eso lo desarrollaremos en el algoritmo con un sorteo aleatorio con probabilidad α de aceptación y su opuesto de rechazo.

```
set.seed(1)
one_mh_iteration <- function(w, current){
  #Propomemos la nueva ubicación en base a la ubicaciónn actual y w.
  #W funciona como el ancho del intervalo
  proposal <- runif(1, min = current - w, max = current + w)

  #Definimos la regla de decisión para evaluar si movernos o no.
  #Primero calculamos la plausibilidad del proposal y del actual.

  proposal_plaus <- dnorm(proposal, 0, 1) * dnorm(6.25, proposal, 0.75)
  current_plaus  <- dnorm(current, 0, 1) * dnorm(6.25, current, 0.75)
  #Calculamos el alhpa definido anteriormente
  alpha <- min(1, proposal_plaus / current_plaus)
```

```
  #Calculamos la siguiente parada  haciendo un sorteo aleatorio,
  #utilizando las  probablidades anteriormente.
  #Podriamos pensar las probabilidades como las caras de una moneda cargada.
  #Con probablidad alpha nos movemos.
  #con probabilidad 1-alpha nos quedamos donde estabamos.
  next_stop <- sample(c(proposal, current),
                      size = 1, prob = c(alpha, 1-alpha))

  # Return the results
  return(data.frame(proposal, alpha, next_stop))
}
```

Parte a: W=0.01

```
  proposal    alpha next_stop
1  2.99531 0.987027   2.99531
```

En primer lugar, como w es muy pequeño (0.01), los valores del proposal estan entre 2.99 y 3.01. El nuevo proposal pertenece al intervalo, lo que es correcto.

α, la probabilidad de aceptación del nuevo proposal, es casi 1. Por ende muy probablemente aceptaremos el valor propuesto. Es la probabilidad que le cargaremos al sorteo aleatorio.

Como era muy probable, aceptamos la nueva ubicación (location) para el proposal dado. Por ende next_stop es igual al proposal.

Con un w tan bajo casi siempre aceptaremos el proposal, pero a cambio daremos pasos muy pequeños, recorreremos los valores del recorrido de manera muy lenta.

Parte B: W=0.5

```
  proposal alpha next_stop
1 3.072853     1  3.072853
```

El proposal pertenece al intervalo determinado por $current = 3 \pm w = 0.5$

En este caso, se aceptó automaticamente, por lo que sabemos que $f(\mu' \mid y) > f(\mu \mid y)$. Como α es exactamente igual a 1 no hubo sorteo. Esto significa que el proposal mejora la plausibilidad respecto al valor anterior. Por ende la cadena se movió automaticamente al nuevo valor.

Parte c: w=1

```
  proposal     alpha next_stop
1 2.403364 0.1162826  2.403364
```

En este caso el valor propuesto salta a 2.897 y fue obtenido dentro del intervalo $[2, 4]$ En este caso, el valor es menos plausible que el actual, por lo que α es menor a 1 y se debe sortear. Se realiza dicho sorteo con probabilidad $\alpha = 0.7402203$ y fue aceptado, por lo que el valor de la cadena se actualiza a $\mu' = 2.897$. En las pruebas que realizamos en clase, vimos que para este current, $w = 1$ es un buen valor, que tiene un equilibrio dentro de estabilidad y avance en el recorrido. Lo utilizaremos para comparar en el siguiente ejercicio.

Parte D: w=3.

```
  proposal      alpha next_stop
1 5.668052 0.08411669         3
```

En este caso el valor del proposal es 4.0474 y pertenece al intervalo 3 ± 3, es decir $[0, 6]$. En este caso $\alpha = 1$, lo que significa que la plausibilidad (posterior en el nuevo punto) fue mayor o igual a la del punto actual(3). En este caso, la probabilidad de aceptación es 1, por lo tanto, siempre aceptamos la propuesta.

Ejercicio 7.10: (An entire tour with a Uniform proposal model)

Exercise 7.10 (An entire tour with a Uniform proposal model) Implement the Metropolis-Hastings function `mh_tour()` defined in Section 7.3 to construct tours of μ under each of the following scenarios. Construct trace plots and histograms for each tour.

a. 50 iterations, $w = 50$
b. 50 iterations, $w = 0.01$
c. 1000 iterations, $w = 50$
d. 1000 iterations, $w = 0.01$
e. Contrast the trace plots in parts a and b. Explain why changing w has this effect.
f. Consider the results in parts c and d. Is the w value as important when the number of iterations is much larger? Explain.

Modificaremos la función mh_tour de manera de hacerla más universal. Para ello le pasaremos el valor de current al llamar la función y no dentro de ella, como se realizó en el ejercico en clase. De este modo, cada vez que llamemos a current debemos pasarle 3 datos: N = cantidad de iteraciones, w = ancho del intervalo para la uniforme y current = valor inicial.

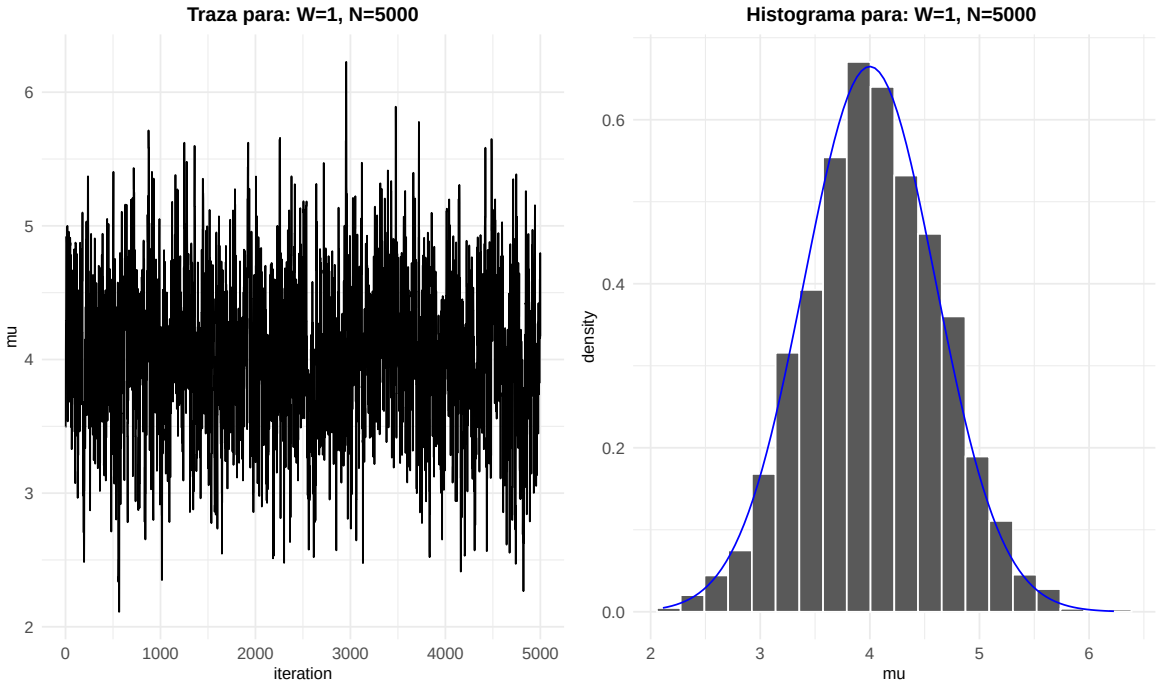
```
mh_tour <- function(N, w, current){
  # Inicializamos la simulacion, creando un vector de ceros de tamaño N.
  mu <- rep(0, N)

#Simulamos N veces en la cadena.

  for(i in 1:N){
    # Simulamos una iteración con la función que definimos antes.
    sim <- one_mh_iteration(w = w, current = current)

    #Avanzar a la siguiente ubicación.
    mu[i] <- sim$next_stop

    #Reseteamos el valor de current para volver a iterar.
    current <- sim$next_stop
  }

  # Retornamos los valores iterados.
  return(data.frame(iteration = c(1:N), mu))
}
```

Por ejemplo, tal como vimos en clase, un buen valor para realizar las iteraciones es w=1 y un n suficientemente grande, por ejemplo n=5000, dado que la traza recorre los distintos valores

plausibles para μ, pero con mayor probabilidad los cercanos a 3, y al graficar el histograma nos aproximamos mucho a una normal.

```
set.seed(84735) #Decidimos utilizar la misma semilla del libro.
mh_simulation_1 <- mh_tour(N = 5000, w = 1, current=3)
```

De esta manera, el gráfico de traza mantendrá su mayor concentración de ruido alrededor del μ teórico del posterior (4), oscilando y recorriendo distintos valores. Se observa que en ningún momento se estanca (no genera líneas constantes) y tanto para arriba como para abajo recorre el recorrido de μs posibles. Además aparenta ser estable, no tiene patrón a crecer ni a achicarse. Tampoco se observan valores demasiado alejados ni saltos descontrolados.



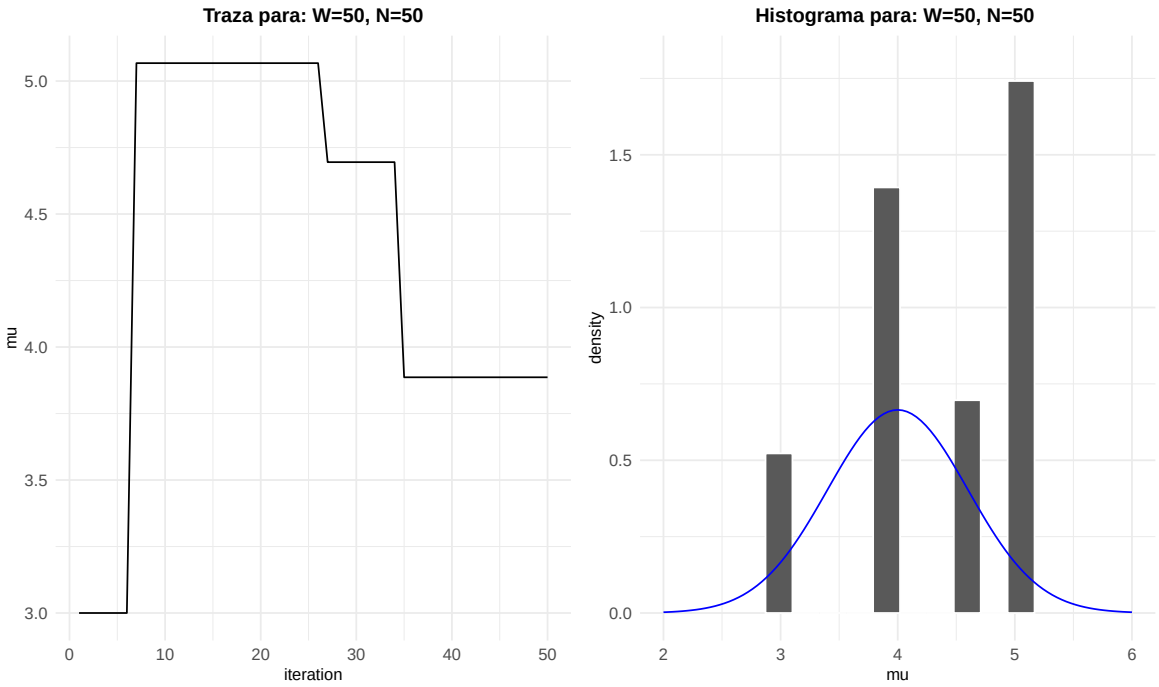
Esta simulación generara una muy similar a la normal calculada de manera teórica, por lo qjue decimos que el algoritmo ajusta bien a la posterior que queriamos calcular. Por lo tanto, la simulación MCMC reproduce correctamente la forma de la distribución teórica.

Parte A: 50 iterations, W=50.

Al iterar 50 veces con un w tan grande, los valores posibles para la uniforme perteneceran a $[-47, 53]$. Este intevalo es sumamente amplio si lo comparamos con la normal que estamos

intentando muestrear. Esto generara que los valores de proposal esten muy dispersos respecto a 3, por ende al evaluar la función en proposal tenderá a cero, haciendo muy probable que los α de cada iteración sean muy pequeños y por ende la probabilidad de hacer el salto en el sorteo sea muy baja. Por lo tanto, sería esperable ver pocos saltos de valor, es decir que nos va a costar salir del current inicial y a su vez nos costará salir del nuevo valor que tomemos.

Respecto al histograma, es de esperar tener pocos valores con muchas observaciones cada uno.



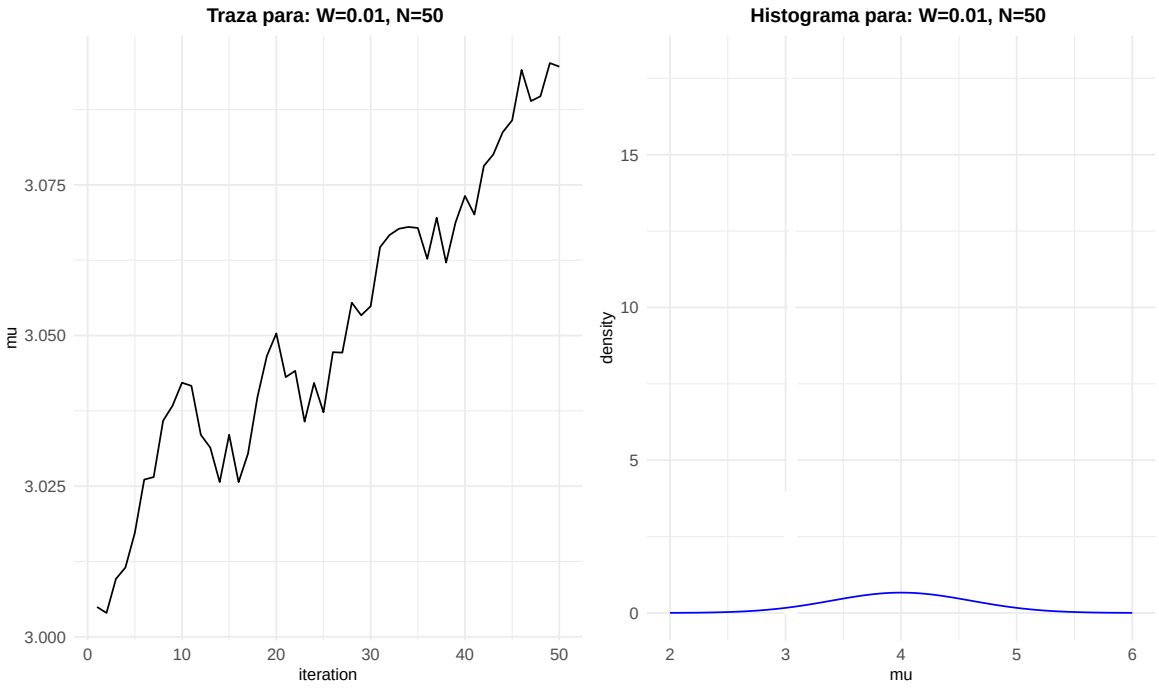
El gráfico de la traza muestra que se rechaza para los 8 primeros valores, es decir que los valores fueron menos plausibles y el sorteo fue negativo. En el caso del valor 9 lo acepta. Finalmente sigue rechazando durante toda la iteración, habiendo quedado fijo en aproximadamente 3.3.

Esto generara un histograma con dos barras, dado que la cadena tuvo solo dos valores: 3 (inicial) y 3.3, que fue el aceptado en el caso numero 9.

Se observa correctamente graficada la normal que representa el posterior teórico. A su vez, se generaron las dos columnas mencionadas antereiormente en el historgrama, las cuales son sumamente altas dado que se repitió mucas veces el 3 y el 3.3. Es sumamente evidente que con n=50 y w=50, no se ajusta bien.

Parte B: 50 iterations, W=0.01.

En este caso tenemos 50 iteraciones, las mismas que antes, pero cambia drásticamente nuestro W, ahora estamos dando "saltos" muy pequeños.



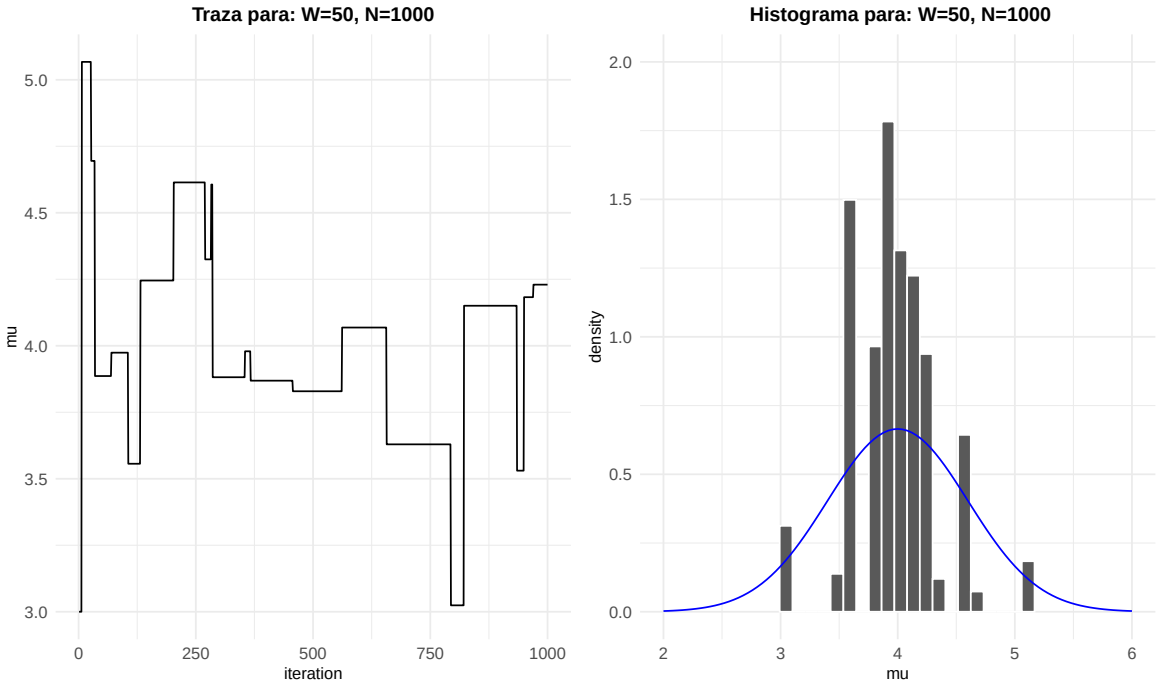
Al hacer la gráfica de la traza, se observa que pasa lo contrario que en el caso A. Aquí la cadena acepta casi todos los pasos, pero como son tan pequeños avanza muy lentamente. En 50 iteraciones ni siquiera llegó a un valor de $\mu = 4$.

El histograma es coherente con el gráfico de traza anterior. Tenemos unas barras muy "delgadas" concentradas cerca del valor inicial. La cadena no logra ver el panorama completo.

Comparación entre a y b:

Concluimos que en un número muy bajo de iteraciones, 50 en nuestro caso, la elección de W es muy importante ya que cambios en él alteran significativamente los resultados que obtenemos, es un parámetro sensible. Se hizo la prueba con dos valores de W extremadamente diferentes, uno muy grande y otro muy chico, sin embargo, para esta cantidad de iteraciones, la simulación falla en ambos casos al inentar aproximar la posteriori. El primero porque la cadena se "estanca" y el segundo porque no "explora" lo suficientemente rápido, va muy lento.

Parte c: Que pasa si agrandamos a n=1000?



En el histograma se observa que, a pesar de no usar el w ideal, logramos mejorar bastante la aproximación a la normal. Y si bien le falta mucho para ser buena, empieza a verse reflejada la forma. En el centro se observa una serie de valores entorno a 4 que no lograron tener buena densidad.

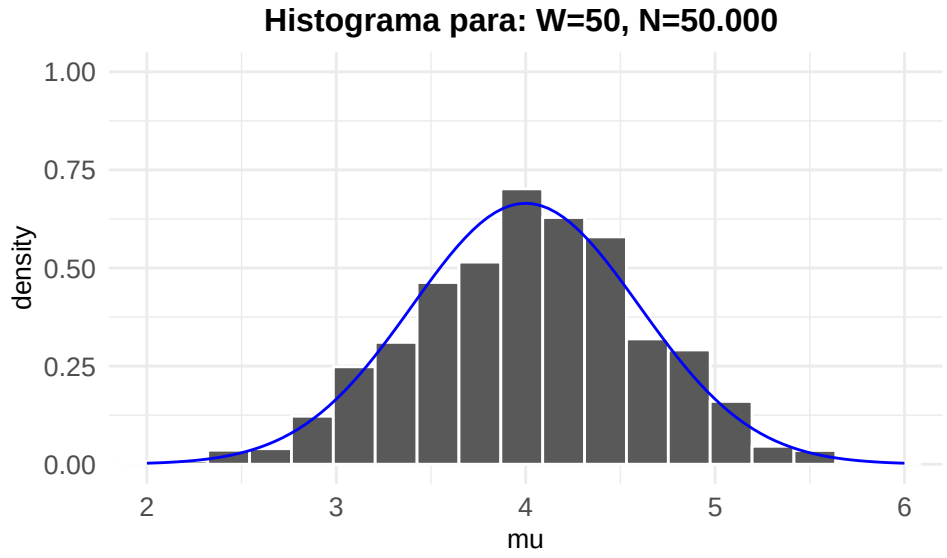
El gráfico de traza en esta oportunidad muestra muchas mesestas, lo que nos hace pensar en que se estancó varias veces en muchos valores. Un w tan grande genera que se tranque mucho la cadena, por el tema del sorteo, que ya mencionamos anteriormente. Por otro lado, muestra varios saltos de valores bajos a altos y viceversa, lo que nos hace pensar que en alguno de los sorteos tuvimos éxito a pesar de tener baja probabilidad de que esto ocurra.vPor lo tanto, a pesar de tener un w tan proporcionalmente grande, con $(n = 1000)$ logramos compensar parte del efecto de los estancamientos para poder ver todo el recorrido. Probablemente genere un histograma sesgado, o con columnas mas altas de lo que deberían en algunos valores.

Entonces, podemos llegar a pensar que agrandando suficientemente la N, logramos compensar gran parte del efecto de haber elegido un w tan grande.

Si volvemos a repetir la simulación, pero esta vez con n=50000, logramos aproximar de mucha mejor manera la normal teorica del posterior.

Aun no llega a ser perfecta, pero cada vez es mejor.

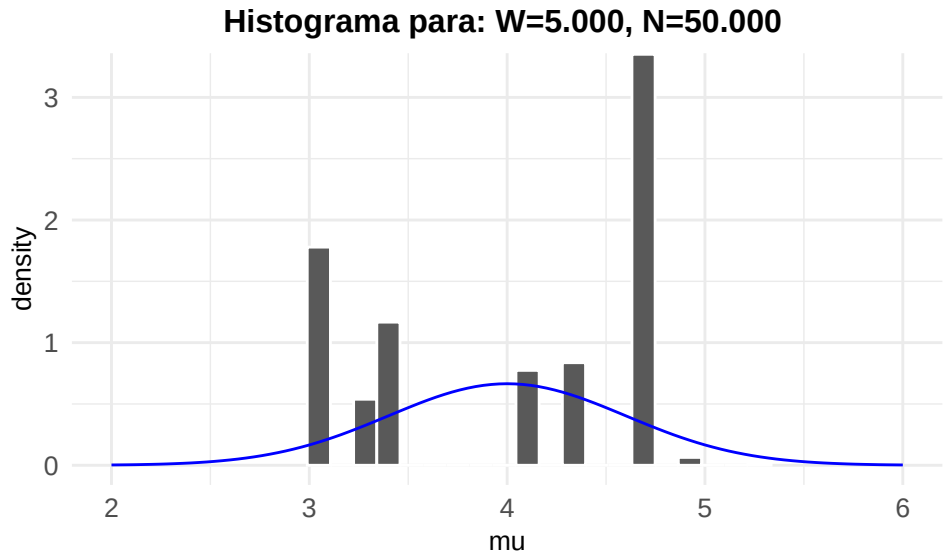
Esto nos da lugar a pensar que era cierta nuestra teoria de que con un n suficientemente grande, compensamos el efecto de un mal w.



Computacionalmente podemos verlo como que estamos sacrificando eficiencia a cambio de haber elegido mal los parametros. Se muestra que con un tamaño de muestra suficientemente grande podemos compensar un tuning ineficiente, aunque a costa de un uso mucho mayor de recursos computacionales. Se intento volver a reproducir con n=50.000, pero la computadora no tuvo capacidad de soportarlo.

Por lo tanto, no todo tuning ineficiente sera compensable.

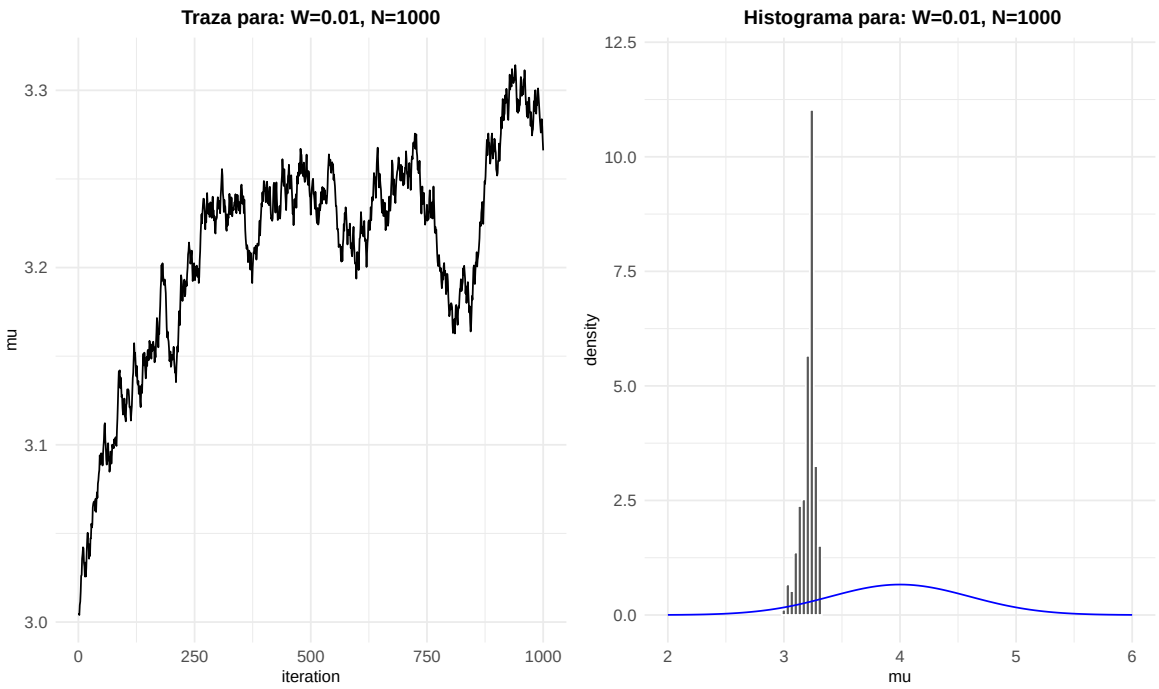
Como prueba de esto, si hubieramos elegido un w considerablemente peor que 50, por ejemplo 5000, el efecto de n no alcanza para arreglarlo. La gráfica se parece al ejemplo inicial de w=50 y n=50, donde le cuesta muchísimo salir de cada valor. Necesitaríamos un n inmensamente superior para compensarlo, lo que computacionalmente no siemrpe será viable. Visualizamos esto en el siguiente histograma:



De esta manera, concluimos que es mas eficaz mejorar el w antes que seguir agrandando la simulación.

Parte D: 1000 iterations, w=0.01

Finalmente, trabajaremos con la misma cantidad de iteraciones que en el caso anterior, solo que en este caso vamos a disminuir w considerablemente, pasa de ser 50 a 0.01.



En cuanto a la traza, observamos que se va moviendo de forma más rápida que en los casos anteriores, sin embargo, sigue sin llegar al centro de la distribución, que es aproximadamente $\mu = 4$.

En el histograma, se muestra la distribución de los valores visitados, pero sólo es una pequeña porción de la verdadera ditribución a posteriori.

Si bien el valor de W es más adecuado en este caso, vemos que todavía se podría mejorar nuestro modelo.

Como fue explicado anteriormente, es mejor buscar un mejor valor de W, un mejor "tuning", que simplemente aumentar la cantidad de iteraciones. Sin embargo, surge la pregunta de cómo se "encuentra" este valor de W adecuado. Si bien no existe una fórmula para esto, en general lo más conveniente es elegir uno que no sea ni muy alto ni muy bajo, se busca un balance, que no se estanque la cadena, pero que tampoco vaya recorriendo los valores de forma demasiado lenta. Como vimos con estos ejemplos, si bien un valor de N alto puede compensar hasta cierto punto el efecto de un tuning malo, de todas formas lo mejor es primer encontrar un valor de W adecuado. Para los disntintos gráficos de traza e histogramas que realizamos, con sus correspondientes valores de W diferentes, se observa que el más adecuado fue el primero, el que se realizó al presentar el ejercicio, el cual tenía un valor de 1. Esto es razonable con lo que planteamos antes, no es un valor extremo. En cambio, 50 y 0.01 sí lo son.

Anexo

Simulación de las cadenas de markov monte carlo del ejercicio 6.15:

```
y <- c(0,1,0)

gp_sim <-stan(
  model_code = gp_model,
  data = list(N = length(y), Y=y),
  chains = 4,
  iter = 5000*2,
  seed = 84735)
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 4e-06 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.028 seconds (Warm-up)
Chain 1:                0.045 seconds (Sampling)
Chain 1:                0.073 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 4e-06 seconds
```

```
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:    1 / 10000 [  0%]  (Warmup)
Chain 2: Iteration: 1000 / 10000 [ 10%]  (Warmup)
Chain 2: Iteration: 2000 / 10000 [ 20%]  (Warmup)
Chain 2: Iteration: 3000 / 10000 [ 30%]  (Warmup)
Chain 2: Iteration: 4000 / 10000 [ 40%]  (Warmup)
Chain 2: Iteration: 5000 / 10000 [ 50%]  (Warmup)
Chain 2: Iteration: 5001 / 10000 [ 50%]  (Sampling)
Chain 2: Iteration: 6000 / 10000 [ 60%]  (Sampling)
Chain 2: Iteration: 7000 / 10000 [ 70%]  (Sampling)
Chain 2: Iteration: 8000 / 10000 [ 80%]  (Sampling)
Chain 2: Iteration: 9000 / 10000 [ 90%]  (Sampling)
Chain 2: Iteration: 10000 / 10000 [100%]  (Sampling)
Chain 2:
Chain 2:  Elapsed Time: 0.026 seconds (Warm-up)
Chain 2:                0.022 seconds (Sampling)
Chain 2:                0.048 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 3e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:    1 / 10000 [  0%]  (Warmup)
Chain 3: Iteration: 1000 / 10000 [ 10%]  (Warmup)
Chain 3: Iteration: 2000 / 10000 [ 20%]  (Warmup)
Chain 3: Iteration: 3000 / 10000 [ 30%]  (Warmup)
Chain 3: Iteration: 4000 / 10000 [ 40%]  (Warmup)
Chain 3: Iteration: 5000 / 10000 [ 50%]  (Warmup)
Chain 3: Iteration: 5001 / 10000 [ 50%]  (Sampling)
Chain 3: Iteration: 6000 / 10000 [ 60%]  (Sampling)
Chain 3: Iteration: 7000 / 10000 [ 70%]  (Sampling)
Chain 3: Iteration: 8000 / 10000 [ 80%]  (Sampling)
Chain 3: Iteration: 9000 / 10000 [ 90%]  (Sampling)
Chain 3: Iteration: 10000 / 10000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.028 seconds (Warm-up)
```

```
Chain 3:                0.022 seconds (Sampling)
Chain 3:                0.05 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 3e-06 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 10000 [  0%]  (Warmup)
Chain 4: Iteration: 1000 / 10000 [ 10%]  (Warmup)
Chain 4: Iteration: 2000 / 10000 [ 20%]  (Warmup)
Chain 4: Iteration: 3000 / 10000 [ 30%]  (Warmup)
Chain 4: Iteration: 4000 / 10000 [ 40%]  (Warmup)
Chain 4: Iteration: 5000 / 10000 [ 50%]  (Warmup)
Chain 4: Iteration: 5001 / 10000 [ 50%]  (Sampling)
Chain 4: Iteration: 6000 / 10000 [ 60%]  (Sampling)
Chain 4: Iteration: 7000 / 10000 [ 70%]  (Sampling)
Chain 4: Iteration: 8000 / 10000 [ 80%]  (Sampling)
Chain 4: Iteration: 9000 / 10000 [ 90%]  (Sampling)
Chain 4: Iteration: 10000 / 10000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.037 seconds (Warm-up)
Chain 4:                0.022 seconds (Sampling)
Chain 4:                0.059 seconds (Total)
Chain 4:
```