Laboration report in Computational Statistics

Laboration 4

732A90

Duc Tran
William Wiik

Division of Statistics and Machine Learning
Department of Computer Science
Linköping University

28 November 2023

Contents

1 Question 1: Computations with Metropolis–Hastings

Consider a random variabel X with the following probability density function:

$$f(x) \propto x^5 e^{-x}, x > 0$$

The distribution is known up to some constant of proportionality.

1.1 1 a)

Question

Use the Metropolis–Hastings algorithm to generate 10000 samples from this distribution by using a log–normal $LN(X_t, 1)$ proposal distribution; take some starting point. Plot the chain you obtained with iterations on the horizontal axis.

Answer

Metropolis–Hastings algorithm:

- A starting value $x^{(0)}$ is generated from some starting distribution
- Given observation $x^{(t)}$, generate $x^{(t+1)}$ as follows:
- Sample a candidate x^* from a proposal distribution $g(\cdot|x^{(t)})$
- Compute the MH ratio $R(x^{(t)}, x^*) = \frac{f(x^*)g(x^{(t)}|x^*)}{f(x^{(t)})g(x^*|x^{(t)})}$
- Sample $x^{(t+1)}$ according to

$$x^{(t+1)} = \begin{cases} x^*, & \text{with probabilty } \min\{R(x^{(t)}, x^*), 1\} \\ x^{(t)} & \text{otherwise} \end{cases} \tag{1}$$

- If more observations needed set, t <- t + 1; go to 1

```
library(ggplot2)

f <- function(x){
  x^5 * exp(-x)
}

numbers_a <- c()
accept_a <- 0
set.seed(13)

# Randomising starting value
current_x <- runif(1, 0, 1)
```

```r
for(iter in 1:10000){
  # Next proposed number
  next_x <- rlnorm(1, meanlog = log(current_x), sdlog = 1)

  numerator <- f(next_x) * dlnorm(current_x, log(next_x))
  denominator <- f(current_x) * dlnorm(next_x, log(current_x))
  mh_ratio <- numerator / denominator

  prob <- min(mh_ratio, 1)
  threshold <- runif(1, 0, 1)

  if(prob >= threshold){

    numbers_a[iter] <- next_x
    current_x <- next_x
    accept_a <- accept_a + 1

  } else {

    numbers_a[iter] <- current_x

  }

}
```

```r
library(cowplot)


plot_a_1 <-ggplot(as.data.frame(numbers_a), aes(x = 1:10000, y = numbers_a)) + geom_line() +
           theme_bw() + labs(x = "Iteration", y = "x")


numbers_a_burn <- numbers_a[1:40]

plot_a_1_burn <- ggplot(as.data.frame(numbers_a_burn), aes(x = 1:40, y = numbers_a_burn)) + geom_line()
           theme_bw() + labs(x = "Iteration", y = "x")


plot_grid(plot_a_1,plot_a_1_burn, labels = c("10 000 iteration", "The first 40 iterations"),
          label_size = 10, nrow = 2)
```

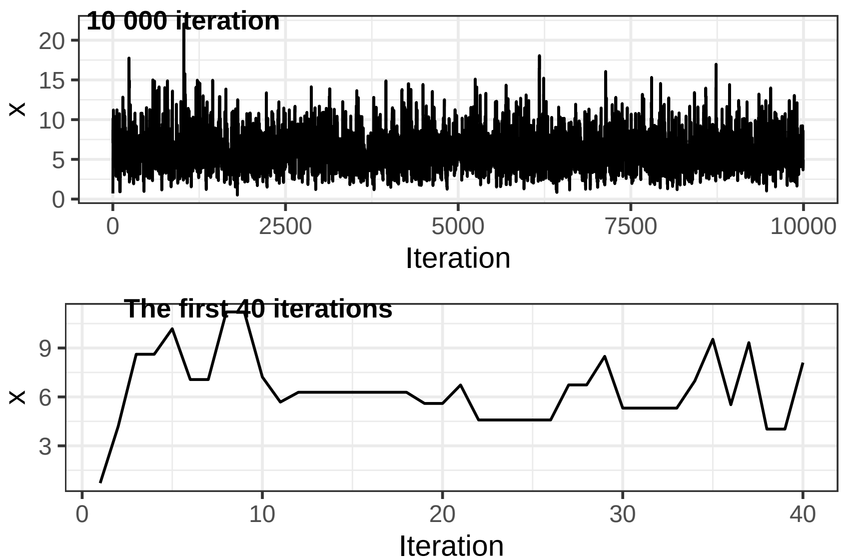

Figure 1: The chain from the Metropolis–Hastings algorithm .

Question

What can you guess about the convergence of the chain? If there is a burn–in period, what can be the size of this period? What is the acceptance rate? Plot a histogram of the sample.

Answer

We can guess that the chain converge, because it appears to sample values from a consistent range and does not exhibit spikes. We think the size of the burn-in period should be around 10, because it converge very fast.

```
print(round(accept_a / 10000, 3))
```

```
## [1] 0.444
```

The acceptance rate is approximately 44 %.

```
plot_a_2 <- ggplot(as.data.frame(numbers_a), aes(x = numbers_a)) +
  geom_histogram(colour="black",fill="indianred", bins=40) +
  theme_bw() +
  labs(y="Frequency", x="x") + theme_bw()
plot_a_2
```

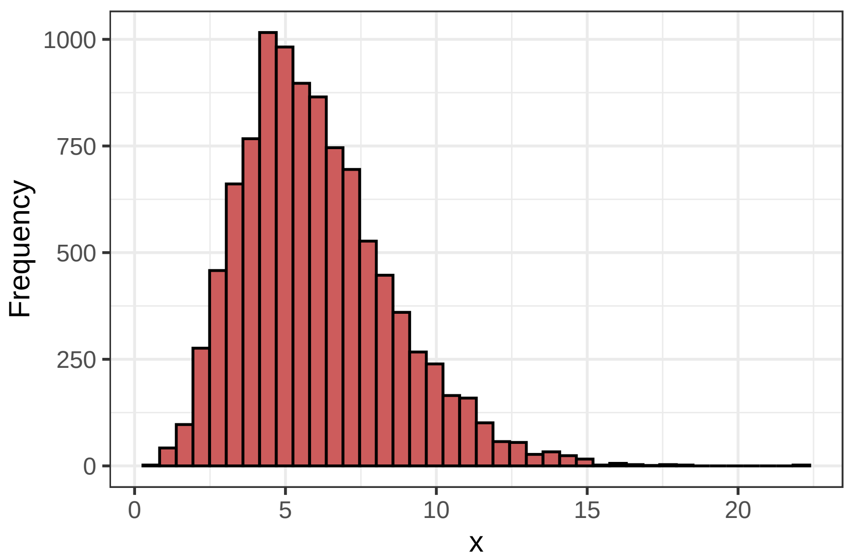

Figure 2: Histogram of the sample.

1.2 1 b)

Question

Perform Part a by using the chi–square distribution $\chi^2(\lfloor X_t + 1 \rfloor)$ as a proposal distribution, where $\lfloor x \rfloor$ is the floor function, meaning the integer part of x for positive x.

Answer

```r
set.seed(13)

f <- function(x){
  x^5 * exp(-x)
}

numbers_b <- c()
accept_b <- 0
set.seed(13)
# Randomising starting value
current_x <- floor(runif(1, 1, 5))


for(iter in 1:10000){
  # Next proposed number
  next_x <- rchisq(1, floor(current_x+1))

  numerator <- f(next_x) * dchisq(current_x, next_x)
  denominator <- f(current_x) * dchisq(next_x, current_x)
  mh_ratio <- numerator / denominator

  prob <- min(mh_ratio, 1)
  threshold <- runif(1, 0, 1)

  if(prob >= threshold){

    numbers_b[iter] <- next_x
    current_x <- next_x
    accept_b <- accept_b + 1

  } else {

    numbers_b[iter] <- current_x

  }
}
```

```r
plot_b_1 <-ggplot(as.data.frame(numbers_b), aes(x = 1:10000, y = numbers_b)) + geom_line() +
  theme_bw() + labs(x = "Iteration", y = "x")
plot_b_1
```

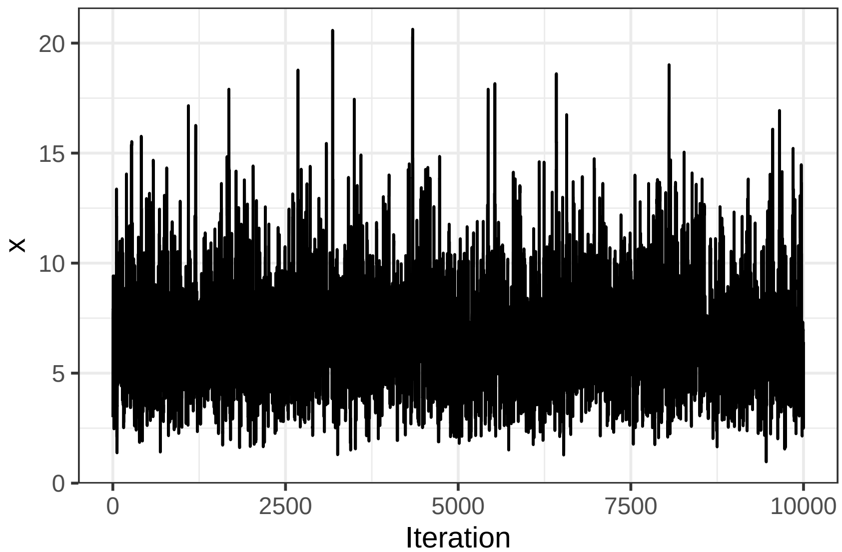

Figure 3: The chain from the Metropolis–Hastings algorithm with Chi2 distribution as a proposal distribution.

```r
print(round(accept_b / 10000, 3))
```

```
## [1] 0.605
```

The acceptance rate is approximately 61 %.

```
plot_b_2 <- ggplot(as.data.frame(numbers_b), aes(x = numbers_b)) +
  geom_histogram(colour="black",fill="indianred", bins=40) +
  theme_bw() +
  labs(y="Frequency", x="x") + theme_bw()
plot_b_2
```

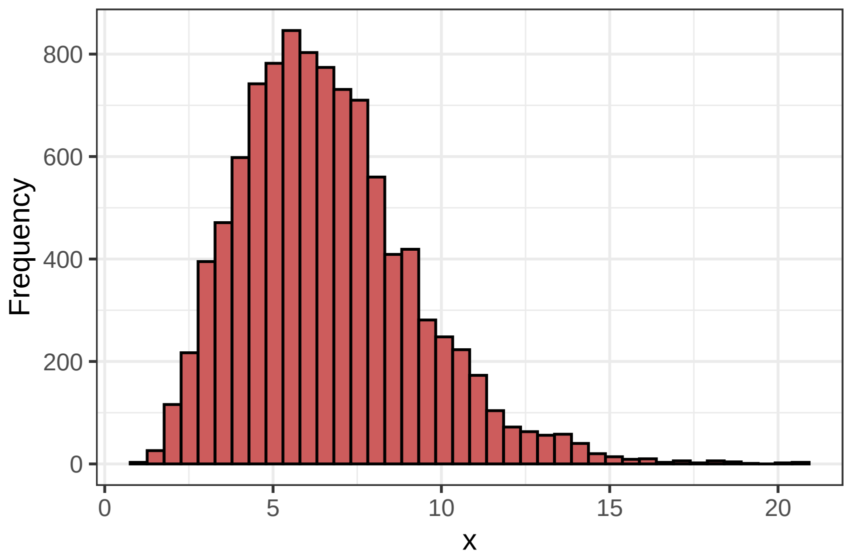

Figure 4: Histogram of the sample from 1 b).

1.3 1 c)

Question

Suggest another proposal distribution (can be a log normal or chi–square distribution with other parameters or another distribution) with the potential to generate a good sample. Perform part a with this distribution.

Answer

We have choosen log–normal $LN(X_t, 0.5)$ as proposal distribution.

```
set.seed(13)
f <- function(x){
  x^5 * exp(-x)
}

numbers_c <- c()
accept_c <- 0
set.seed(13)
# Randomising starting value
current_x <- runif(1, 0, 1)
```

```r
for(iter in 1:10000){
  # Next proposed number
  next_x <- rlnorm(1, meanlog = log(current_x), sdlog = 0.5)

  numerator <- f(next_x) * dlnorm(current_x, log(next_x), sdlog = 0.5)
  denominator <- f(current_x) * dlnorm(next_x, log(current_x), sdlog = 0.5)
  ratio <- numerator / denominator

  prob <- min(ratio, 1)
  threshold <- runif(1, 0, 1)

  if(prob >= threshold){
    numbers_c[iter] <- next_x
    current_x <- next_x
    accept_c <- accept_c + 1
  } else {
    numbers_c[iter] <- current_x
  }
}
```

```r
plot_c_1 <-ggplot(as.data.frame(numbers_c), aes(x = 1:10000, y = numbers_c)) + geom_line() +
  theme_bw() + labs(x = "Iteration", y = "x")
plot_c_1
```

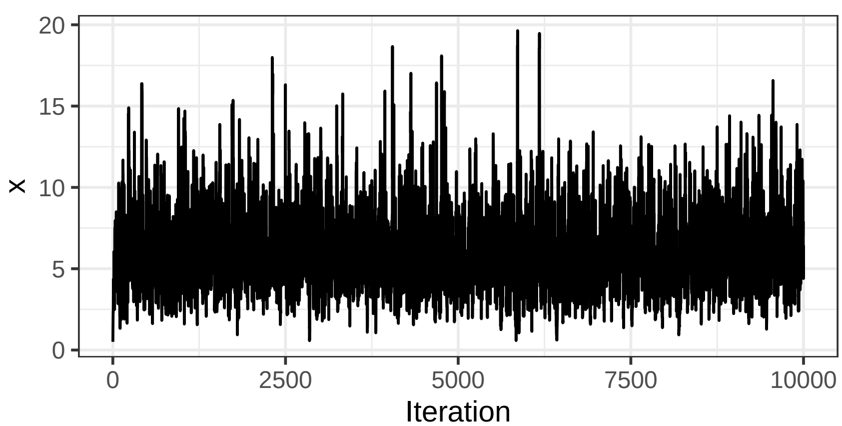

Figure 5: The chain from the Metropolis–Hastings algorithm with LN(Xt,0.5) as a proposal distribution.

```r
print(round(accept_c / 10000, 3))
```

```
## [1] 0.66
```

The acceptance rate is approximately 66 %.

```r
plot_c_2 <- ggplot(as.data.frame(numbers_c), aes(x = numbers_c)) +
  geom_histogram(colour="black",fill="indianred", bins=40) +
  theme_bw() +
  labs(y="Frequency", x="x") + theme_bw()

plot_c_2
```

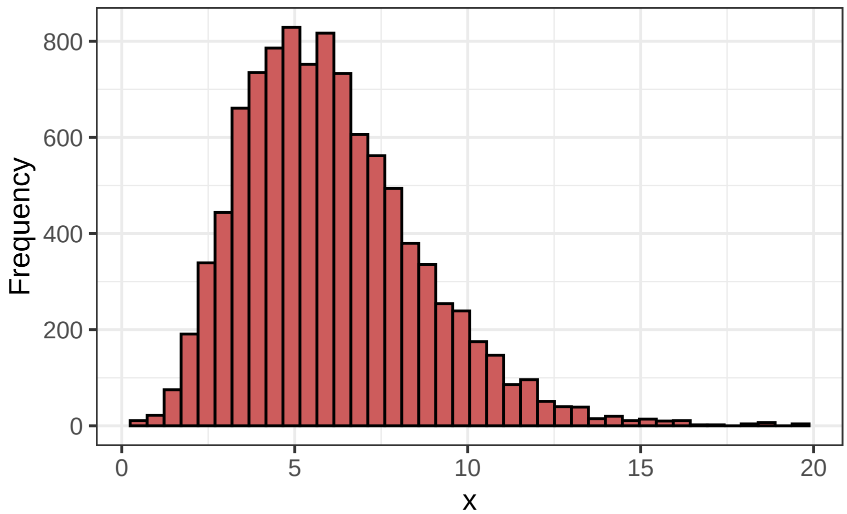

Figure 6: Histogram of the sample from 1 c).

1.4 1 d)

Question

Compare the results of Parts a, b, and c and make conclusions.

Answer

Table 1: Different Proposal distributions and Acceptance rate

Proposal.distribution	Acceptance.rate
$LN(X_t, 1)$	0.4436
$\chi^2(\lfloor X_t + 1 \rfloor)$	0.6048
$LN(X_t, 0.5)$	0.6599

The proposal distribution with $LN(X_t, 0.5)$ has the highest acceptance rate, 66 % and $LN(X_t, 1)$ has the lowest acceptance rate, 44 %.

According to Givens and Hoeting[1] the optimal acceptance rate for uni-dimension should be 44 %, which indicates that $LN(X_t, 1)$ is the best proposal distribution.

[1] G. Givens & J. Hoeting, *Computational Statistics* (2012)

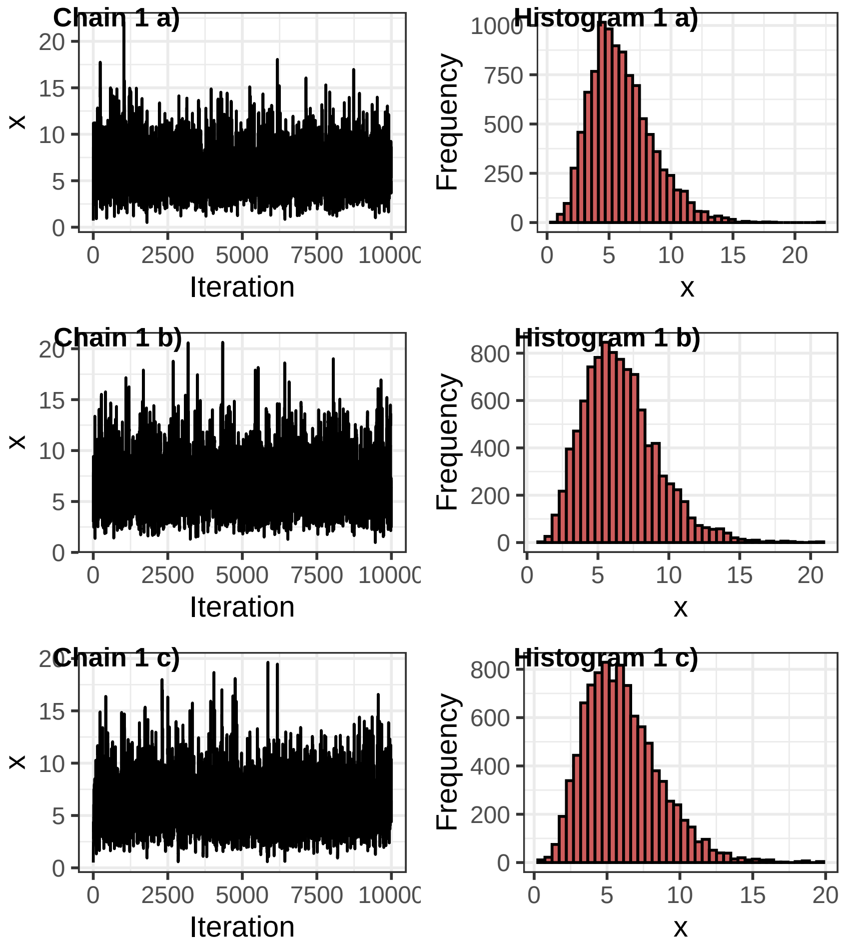

Figure 7: Comparing chains and histogram from a), b) and c).

The chains and histogram looks very similar between the different proposal distribution.

1.5 1 e)

Question

Estimate

$$E(x) = \int_0^\infty x f(x) dx$$

using the samples from Parts a, b, and c

Answer

```r
mean(numbers_a)
```

```
## [1] 6.039928
```

```r
mean(numbers_b)
```

```
## [1] 6.606114
```

```r
mean(numbers_c)
```

```
## [1] 6.060726
```

```r
(mean(numbers_a) + mean(numbers_b) + mean(numbers_c))/ 3
```

```
## [1] 6.235589
```

$E(x)$ is estimated to be around 6.24.

1.6 1 f)

Question

The distribution generated is in fact a gamma distribution. Look in the literature and define the actual value of the integral. Compare it with the one you obtained.

Answer

The gamma PDF is defined as

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} dx$$

and the $E(x)$ is defined as

$$E(x) = \frac{\alpha}{\beta}$$

Compared with the following probability density function:

$$f(x) \propto x^5 e^{-x}, x > 0$$

We can see that α is 6 and β is 1 which give us

$$E(x) = \frac{6}{1} = 6$$

If we compare this value with ones we got from the samples we can see that the sample from 1 a) (6.04) and 1 c) (6.06) is almost equal to 6. However the $E(x)$ from sample from 1 b) (6.61) is not as close to 6 as 1 a) and 1 c).

2 Question 2: Gibbs sampling

Let $X = (X_1, X_2)$ be a bivariate distribution with density $f(x_1, x_2) \propto \mathbf{1}\{x_1^2 + wx_1x_2 + x_2^2 < 1\}$ for some specific w with $|w| < 2$. X has a uniform distribution on some two-dimensional region. We consider here the case $w = 1.999$ (in Lecture 4, the case w = 1.8 was shown).

2.1 2.a

Question: Draw the boundaries of the region where X has a uniform distribution. You can use the code provided on the course homepage and adjust it.

Answer: The boundaries of the region were drawn by modifying part of the code provided on the course homepage and changing the value of w from 1.8 to 1.999. The boundaries are presented in figure 8.

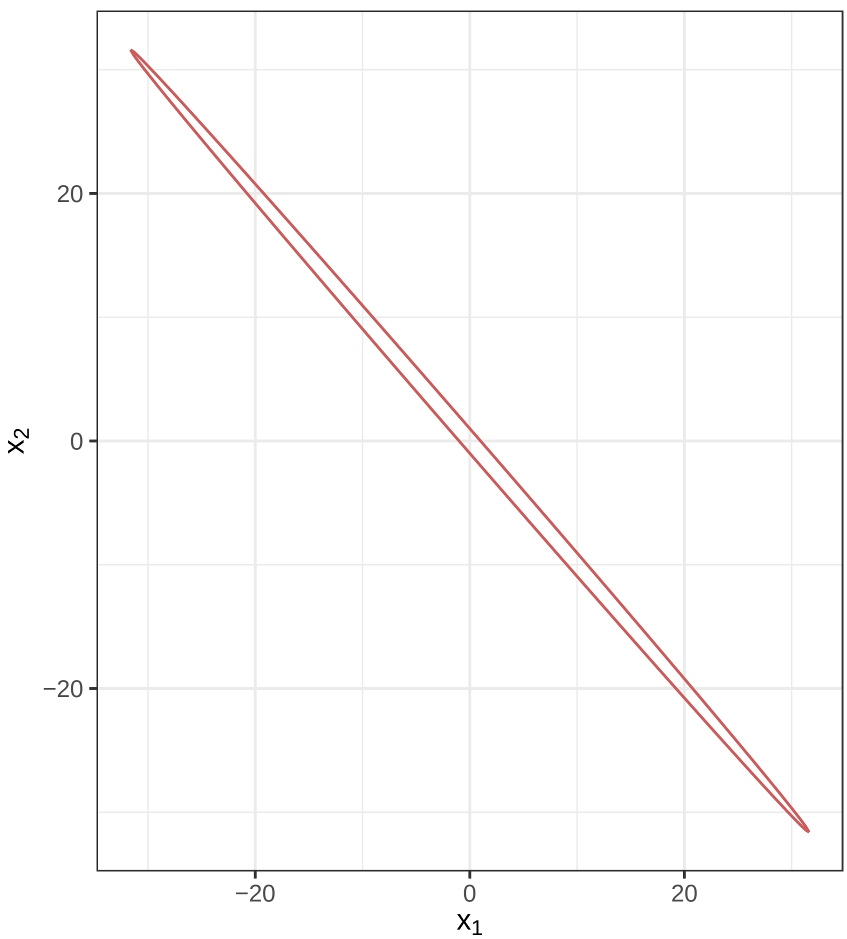

Figure 8: Boundaries for the ellipse $x_1^2 + 1.999x_1x_2 + x_2^2 < 1$

2.2 2.b

Question: What is the conditional distribution of X_1 given X_2 and that of X_2 given X_1?

Answer: To find the condition distributions the equation as follows needs to be solved for X_1 and X_2 one by one.

$$x_1^2 + a \cdot x_1 x_2 + x_2^2 - 1 = 0 \tag{2}$$

where a is a constant equal to 1.999 in this question. It can be seen that solving for X_1 in equation (2) will lead to similar expression when solving for X_2. The idea to solve this equation is to rewrite the left expression so that we get the expression:

$$(x_1 + b \cdot x_2)^2 + c \cdot x_2^2 - 1 = 0 \tag{3}$$

where b and c are constants so that the left-hand side of equation (2) is equivalent to (3). Evaluating the expression $(x_1 + b \cdot x_2)^2$ gives us:

$$(x_1 + b \cdot x_2)^2 = x_1^2 + 2b \cdot x_1 x_2 + b^2 x_2^2 \tag{4}$$

Comparing the coefficients for $x_1 x_2$ in (4) and (2) the constant b can be found. Inserting the evaluated expression from (4) in (3) and then comparing the coefficient for x_2^2 with (2) the constant c can be found. The comparison were as follows:

$$\begin{cases} a = 2b \\ c + b^2 = 1 \end{cases} \iff \begin{cases} b = \frac{a}{2} \\ c = 1 - \left(\frac{a}{2}\right)^2 \end{cases} \tag{5}$$

This gives us that solving equation (6):

$$\left(x_1 + \frac{a}{2} \cdot x_2\right)^2 + \left(1 - \left(\frac{a}{2}\right)^2\right)^2 \cdot x_2^2 - 1 = 0 \tag{6}$$

will give the same solution as solving equation (2). The solution for equation (6) for X_1 is as follows:

$$\left(x_1 + \frac{a}{2} \cdot x_2\right)^2 + \left(1 - \left(\frac{a}{2}\right)^2\right) \cdot x_2^2 - 1 = 0 \tag{7}$$

$$\iff \left(x_1 + \frac{a}{2} \cdot x_2\right)^2 = 1 - \left(1 - \left(\frac{a}{2}\right)^2\right) \cdot x_2^2 \tag{8}$$

$$\iff x_1 + \frac{a}{2} \cdot x_2 = \pm\sqrt{1 - \left(1 - \left(\frac{a}{2}\right)^2\right) \cdot x_2^2} \tag{9}$$

$$\iff x_1 = -\frac{a}{2} \cdot x_2 \pm \sqrt{1 - \left(1 - \left(\frac{a}{2}\right)^2\right) \cdot x_2^2} \tag{10}$$

The solution gives us the lower and upper values of the boundary of the ellipse. Inserting the value $a = 1.999$ in (10), the conditional distribution of X_1 given X_2 will be uniformly distributed on the interval

$$\Big(- 0.9995x_2 - \sqrt{1 - 0.00099975x_2^2}, -0.9995x_2 + \sqrt{1 - 0.00099975x_2^2}\Big) \tag{11}$$

The conditional distribution of X_2 given X_1 will be uniformly distributed on the interval

$$\Big(- 0.9995x_1 - \sqrt{1 - 0.00099975x_1^2}, -0.9995x_1 + \sqrt{1 - 0.00099975x_1^2}\Big) \tag{12}$$

2.3 2.c

Question: Write your own code for Gibbs sampling the distribution. Run it to generate n = 1000 random vectors and plot them into the picture from Part a. Determine $P(X_1 > 0)$ based on the sample and repeat this a few times (you need not to plot the repetitions). What should be the true result for this probability?

Answer: We generated 1000 random vectors with our implementation of Gibbs sampling. In figure 9 the result is visualised and the code for our implementation of the algorithm is as follows:

```r
# Take as a starting value x1=x2=0
set.seed(13)
x1 <- 0
x2 <- 0
# Lower boundary of the ellipse
lower_bound <- function(x){
  -0.9995*x-sqrt(1-0.00099975*x^2)
}
# Upper boundary of the ellipse
upper_bound <- function(x){
  -0.9995*x+sqrt(1-0.00099975*x^2)
}

sample_gibbs <- function(x1=0, x2=0, n=1000){
  # Empty vectors to store sampled points
  sample_x1 <- c()
  sample_x2 <- c()

  for(iter in 1:n){
    # Lower and upper values of the ellipse conditioned on x2
    lower <- lower_bound(x2)
    upper <- upper_bound(x2)

    # Sample new value for x1* = f(x1|x2)
    new_x1 <- runif(1, lower, upper)

    # Sample new value for x2* = f(x2|x1*)
    lower <- lower_bound(new_x1)
    upper <- upper_bound(new_x1)
    new_x2 <- runif(1, lower, upper)

    # Accept the new sampled point
    x1 <- new_x1
    x2 <- new_x2
    # Saves the sampled point
    sample_x1[iter] <- x1
    sample_x2[iter] <- x2
  }
  sample <- data.frame(x1=sample_x1, x2=sample_x2)
  return(sample)
}
```

```r
sample1 <- sample_gibbs()
```

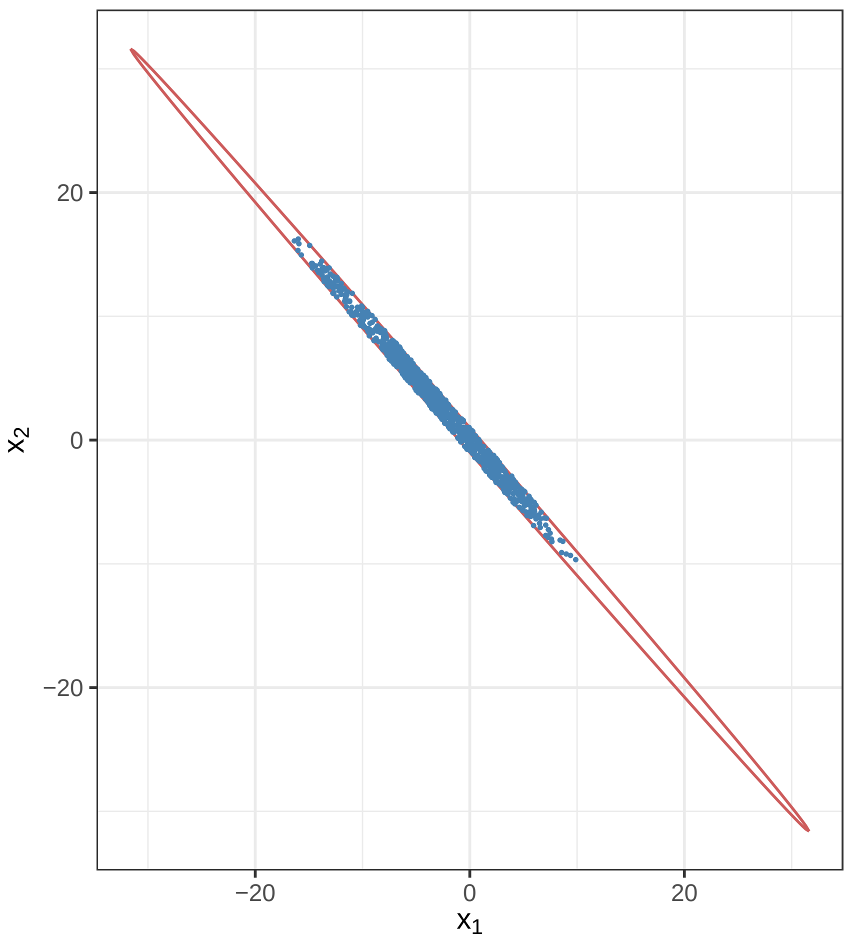

Figure 9: 1000 randomly sampled points with Gibbs sampling for the ellipse: $x_1^2 + 1.999x_1x_2 + x_2^2 < 1$, with starting values $x_1 = x_2 = 0$

For the sampled values in figure 9 the $P(X_1 > 0)$ is 29.8%:

```r
sample1_prob <- sum(sample1$x1>0) / 1000
sample1_prob
```

```
## [1] 0.298
```

With the Gibbs sampling, three other samples of 1000 random vectors were drawn to calculate $P(X_1 > 0)$, the result is presented in table 2.

```r
sample2 <- sample_gibbs()
sample3 <- sample_gibbs()
sample4 <- sample_gibbs()

sample2_prob <- sum(sample2$x1>0) / 1000
sample3_prob <- sum(sample3$x1>0) / 1000
sample4_prob <- sum(sample4$x1>0) / 1000
```

Table 2: $P(X_1 > 0)$ for three different samples of 1000 with Gibbs.

	Probability
Sample 2	0.185
Sample 3	0.461
Sample 4	0.003

The results from table 2 varies a lot, which could indicate that we need much larger samples to properly sample the distribution.

Since the distribution symmetric and uniformly distributed around the origin, the true probability is 50%.

2.4 2.d

Question: Discuss, why the Gibbs sampling for this situation seems to be less successful for w = 1.999 compared to the case w = 1.8 from the lecture.

Answer: The boundaries for the ellipse from the lecture with $w = 1.8$ is presented in figure 10.

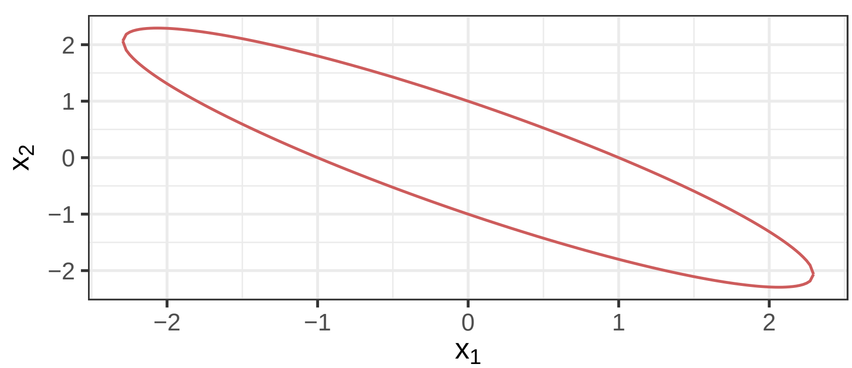

Figure 10: Boundaries for the ellipse from the lecture with w=1.8

In figure 10 the range for X_1, X_2 with $w = 1.8$ is approximately ± 2.3. In figure 8 the range for $w = 1.999$ is approximately ± 31.63. By increasing w, the ellipse gets more stretched out in a $-45°$ angle. This can be interpreted that X_1 and X_2 are more correlated for larger values of w. This affect the Gibbs sampling for $w = 1.999$ that the algorithm can only take smaller steps since a new point is generated by "walking" first in the direction of x_2 (vertical) and then in the direction x_1(horizontal). Which means that for $w = 1.999$ it takes a lot longer for the algorithm to sample evenly from the whole distribution since it takes smaller steps on an ellips with larger range of values.

2.5 2.e

Question: We might transform the variable X and generate $U = (U_1, U_2) = (X_1 - X_2, X_1 + X_2)$ instead. In this case, the density of the transformed variable $U = (U_1, U_2)$ is again a uniform distribution on a transformed region (no proof necessary for this claim). Determine the boundaries of the transformed region where U has a uniform distribution on. You can use that the transformation corresponds to $X_1 = (U_2 + U_1)/2$ and $X_2 = (U_2 - U_1)/2$ and set this into the boundaries in terms of X_i. Plot the boundaries for (U_1, U_2). Generate n = 1000 random vectors with Gibbs sampling for U and plot them. Determine $P(X_1 > 0) = P((U2+U1)/2 > 0)$. Compare the results with Part c.

Answer: First we substitute $X_1 = (U_2 + U_1)/2$ and $X_2 = (U_2 - U_1)/2$ in equation (2). The equation for the U is as follows:

$$\left(\frac{U_2 + U_1}{2}\right)^2 + w\left(\frac{U_2 + U_1}{2} \cdot \frac{U_2 + -U_1}{2}\right) + \left(\frac{U_2 - U_1}{2}\right)^2 - 1 = 0 \tag{13}$$

$$\iff \frac{U_2^2 + 2U_1U_2 + U_1^2}{4} + w\left(\frac{U_2^2 - U_1^2}{4}\right) + \frac{U_2^2 - 2U_1U_2 + U_1^2}{4} - 1 = 0 \tag{14}$$

$$\iff \frac{2U_2^2 + 2U_1^2 + w(U_2^2 - U_1^2)}{4} - 1 = 0 \tag{15}$$

$$\iff (2 + w)U_2^2 + (2 - w)U_1^2 = 4 \tag{16}$$

By solving equation (16) for U_1, the conditional distribution of U_1 given U_2 is as follows:

$$U_1 = \pm\sqrt{\frac{4 + (2 + w)U_2^2}{2 - w}} \tag{17}$$

By solving equation(16) for U_2, the conditional distribution of U_2 given U_1 is as follows:

$$U_2 = \pm\sqrt{\frac{4 + (2 - w)U_2^2}{2 + w}} \tag{18}$$

In equation (17) and (18) the lower boundary is by taking the negative value and upper boundary the positive value.

To plot the boundaries, the semi-minor axis and semi-major axis were calculated[2]. This was done by setting $U_1^2 = 0$ for (16) and then solving for U_2, the same was done for U_1.

The semi-major axis is $\sqrt{4/0.001}$ and the semi-minor axis is $\sqrt{4/3.999}$, the ellipse is plotted in figure 11.

[2]https://en.wikipedia.org/wiki/Semi-major_and_semi-minor_axes

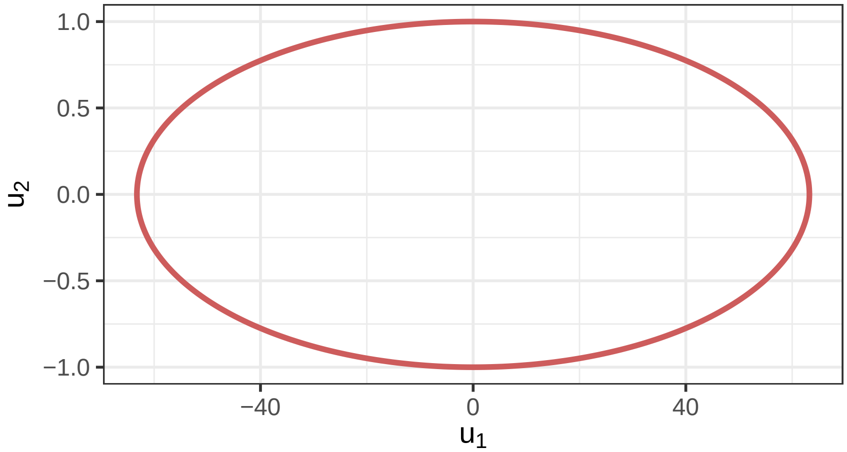

Figure 11: Boundary for the ellipse U.

In figure 11, the x- and y-axis are on different scales. The ellipse is very narrow (more than the ellipse in figure 8) but the ellipse is no longer on an angle.

For U, 1000 random vectors were generated and the sampled values are presented in figure 12

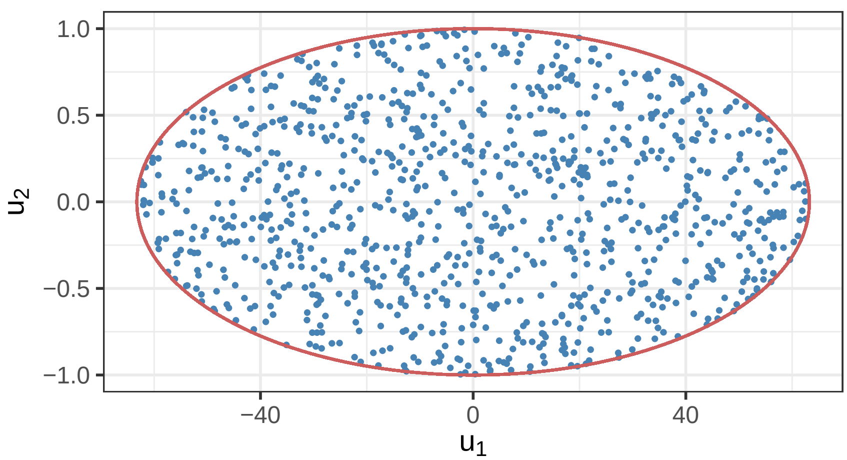

Figure 12: 1000 randomly sampled points with Gibbs sampling for U.

In figure 12 the $P((U_2 + U_1)/2 > 0)$ is 49.10%.

```
sum((sample_u$u2 + sample_u$u1)/2 > 0) / 1000
```

```
## [1] 0.491
```

In comparison to 2.c, the sampled values are closer to the expected value. The Gibbs sampling "walks" in horizontal and vertical direction and the method is suited for "non-correlated" random variables. When variables are "correlated" a lot more random variables needs to be drawn to have a representative sample from the distribution.

3 Statement of Contribution

We solved the tasks together and then we divided the task of writing the laboration report.

3.1 Question 1

Text written by William.

3.2 Question 2

Text written by Duc.

4 Appendix

The code used in this laboration report are summarised in the code as follows:

```
library(knitr)
library(dplyr)
library(ggforce)
library(kableExtra)
knitr::opts_chunk$set(
  echo = TRUE,
  fig.width = 4.5,
  fig.height = 3)
library(ggplot2)

f <- function(x){
  x^5 * exp(-x)
}

numbers_a <- c()
accept_a <- 0
set.seed(13)

# Randomising starting value
current_x <- runif(1, 0, 1)

for(iter in 1:10000){
  # Next proposed number
  next_x <- rlnorm(1, meanlog = log(current_x), sdlog = 1)

  numerator <- f(next_x) * dlnorm(current_x, log(next_x))
  denominator <- f(current_x) * dlnorm(next_x, log(current_x))
  mh_ratio <- numerator / denominator

  prob <- min(mh_ratio, 1)
```

```r
  threshold <- runif(1, 0, 1)

  if(prob >= threshold){

    numbers_a[iter] <- next_x
    current_x <- next_x
    accept_a <- accept_a + 1

  } else {

    numbers_a[iter] <- current_x

  }

}




library(cowplot)


plot_a_1 <-ggplot(as.data.frame(numbers_a), aes(x = 1:10000, y = numbers_a)) + geom_line() +
           theme_bw() + labs(x = "Iteration", y = "x")


numbers_a_burn <- numbers_a[1:40]

plot_a_1_burn <- ggplot(as.data.frame(numbers_a_burn), aes(x = 1:40, y = numbers_a_burn)) + geom_line()
           theme_bw() + labs(x = "Iteration", y = "x")


plot_grid(plot_a_1,plot_a_1_burn, labels = c("10 000 iteration", "The first 40 iterations"),
          label_size = 10, nrow = 2)


print(round(accept_a / 10000, 3))



plot_a_2 <- ggplot(as.data.frame(numbers_a), aes(x = numbers_a)) +
  geom_histogram(colour="black",fill="indianred", bins=40) +
  theme_bw() +
  labs(y="Frequency", x="x") + theme_bw()
plot_a_2


set.seed(13)
```

```r
f <- function(x){
  x^5 * exp(-x)
}

numbers_b <- c()
accept_b <- 0
set.seed(13)
# Randomising starting value
current_x <- floor(runif(1, 1, 5))


for(iter in 1:10000){
  # Next proposed number
  next_x <- rchisq(1, floor(current_x+1))

  numerator <- f(next_x) * dchisq(current_x, next_x)
  denominator <- f(current_x) * dchisq(next_x, current_x)
  mh_ratio <- numerator / denominator

  prob <- min(mh_ratio, 1)
  threshold <- runif(1, 0, 1)

  if(prob >= threshold){

    numbers_b[iter] <- next_x
    current_x <- next_x
    accept_b <- accept_b + 1

  } else {

    numbers_b[iter] <- current_x

  }
}



plot_b_1 <-ggplot(as.data.frame(numbers_b), aes(x = 1:10000, y = numbers_b)) + geom_line() +
  theme_bw() + labs(x = "Iteration", y = "x")
plot_b_1


print(round(accept_b / 10000, 3))



plot_b_2 <- ggplot(as.data.frame(numbers_b), aes(x = numbers_b)) +
  geom_histogram(colour="black",fill="indianred", bins=40) +
```

```
  theme_bw() +
  labs(y="Frequency", x="x") + theme_bw()
plot_b_2

set.seed(13)
f <- function(x){
  x^5 * exp(-x)
}

numbers_c <- c()
accept_c <- 0
set.seed(13)
# Randomising starting value
current_x <- runif(1, 0, 1)

for(iter in 1:10000){
  # Next proposed number
  next_x <- rlnorm(1, meanlog = log(current_x), sdlog = 0.5)

  numerator <- f(next_x) * dlnorm(current_x, log(next_x), sdlog = 0.5)
  denominator <- f(current_x) * dlnorm(next_x, log(current_x), sdlog = 0.5)
  ratio <- numerator / denominator

  prob <- min(ratio, 1)
  threshold <- runif(1, 0, 1)

  if(prob >= threshold){
    numbers_c[iter] <- next_x
    current_x <- next_x
    accept_c <- accept_c + 1
  } else {
    numbers_c[iter] <- current_x
  }
}

plot_c_1 <-ggplot(as.data.frame(numbers_c), aes(x = 1:10000, y = numbers_c)) + geom_line() +
  theme_bw() + labs(x = "Iteration", y = "x")
plot_c_1
print(round(accept_c / 10000, 3))

plot_c_2 <- ggplot(as.data.frame(numbers_c), aes(x = numbers_c)) +
  geom_histogram(colour="black",fill="indianred", bins=40) +
  theme_bw() +
  labs(y="Frequency", x="x") + theme_bw()

plot_c_2

df <- data.frame("Proposal distribution" = c("$LN(X_t,1)$",
                                             "$\\chi^2(\\lfloor X_t + 1\\rfloor)$",
```

```r
                                              "$LN(X_t,0.5)$"),
                 "Acceptance rate" = c((accept_a / 10000),(accept_b / 10000),(accept_c / 10000)))

kable(df, booktabs=T, escape=FALSE,
      caption = "Different  Proposal distributions and Acceptance rate") %>%
  kable_classic(latex_options = "hold_position")



library(cowplot)


plot_grid(plot_a_1, plot_a_2,
          plot_b_1, plot_b_2,
          plot_c_1, plot_c_2, labels=c("Chain 1 a)", "Histogram 1 a)",
                                       "Chain 1 b)", "Histogram 1 b)",
                                       "Chain 1 c)", "Histogram 1 c)"), ncol = 2, nrow = 3,
          label_size = 10)


mean(numbers_a)
mean(numbers_b)
mean(numbers_c)

(mean(numbers_a) + mean(numbers_b) + mean(numbers_c))/ 3

# Modified code from:
###############################
### Boundary ellipse
### for Gibbs sampling example
### from Lecture 4
### Fall 2023, by Frank Miller
###############################

w <- 1.999
xv <- seq(-1, 1, by=0.01) * 1/sqrt(1-w^2/4)
lower <- -(w/2)*xv-sqrt(1-(1-w^2/4)*xv^2)
upper <- -(w/2)*xv+sqrt(1-(1-w^2/4)*xv^2)

ggplot() +
  geom_line(aes(x=xv, y=lower), colour="indianred") +
  geom_line(aes(x=xv, y=upper), colour="indianred") +
  xlab(bquote(x[1])) +
  ylab(bquote(x[2])) +
  theme_bw()

# Take as a starting value x1=x2=0
set.seed(13)
x1 <- 0
```

```r
x2 <- 0
# Lower boundary of the ellipse
lower_bound <- function(x){
  -0.9995*x-sqrt(1-0.00099975*x^2)
}
# Upper boundary of the ellipse
upper_bound <- function(x){
  -0.9995*x+sqrt(1-0.00099975*x^2)
}

sample_gibbs <- function(x1=0, x2=0, n=1000){
  # Empty vectors to store sampled points
  sample_x1 <- c()
  sample_x2 <- c()

  for(iter in 1:n){
    # Lower and upper values of the ellipse conditioned on x2
    lower <- lower_bound(x2)
    upper <- upper_bound(x2)

    # Sample new value for x1* = f(x1|x2)
    new_x1 <- runif(1, lower, upper)

    # Sample new value for x2* = f(x2|x1*)
    lower <- lower_bound(new_x1)
    upper <- upper_bound(new_x1)
    new_x2 <- runif(1, lower, upper)

    # Accept the new sampled point
    x1 <- new_x1
    x2 <- new_x2
    # Saves the sampled point
    sample_x1[iter] <- x1
    sample_x2[iter] <- x2
  }
  sample <- data.frame(x1=sample_x1, x2=sample_x2)
  return(sample)
}

sample1 <- sample_gibbs()
# Plotting the result
ggplot() +
  geom_line(aes(x=xv, y=lower), colour="indianred") +
  geom_line(aes(x=xv, y=upper), colour="indianred") +
  geom_point(aes(x=sample1$x1, y=sample1$x2), colour="steelblue", size=0.3) +
  xlab(bquote(x[1])) +
  ylab(bquote(x[2])) +
  theme_bw()
sample1_prob <- sum(sample1$x1>0) / 1000
```

```r
sample1_prob
sample2 <- sample_gibbs()
sample3 <- sample_gibbs()
sample4 <- sample_gibbs()

sample2_prob <- sum(sample2$x1>0) / 1000
sample3_prob <- sum(sample3$x1>0) / 1000
sample4_prob <- sum(sample4$x1>0) / 1000
# Sampled values to estimate P(X1>0)
table_samples <- data.frame(Probability = c(sample2_prob, sample3_prob, sample4_prob))
rownames(table_samples) <- c("Sample 2", "Sample 3", "Sample 4")
kable(table_samples, booktabs=T, caption = "$P(X_1>0)$ for three different samples of 1000 with Gibbs.")
  kable_classic(latex_options = "hold_position")
# Ellips from the lecture.
w <- 1.8
xv <- seq(-1, 1, by=0.01) * 1/sqrt(1-w^2/4)
lower <- -(w/2)*xv-sqrt(1-(1-w^2/4)*xv^2)
upper <- -(w/2)*xv+sqrt(1-(1-w^2/4)*xv^2)

  ggplot() +
  geom_line(aes(x=xv, y=lower), colour="indianred") +
  geom_line(aes(x=xv, y=upper), colour="indianred") +
  xlab(bquote(x[1])) +
  ylab(bquote(x[2])) +
  theme_bw()

ggplot() +
  geom_ellipse(aes(x0=0, y0=0, a=sqrt(4/0.001), b=sqrt(4/3.999), angle=0),
               colour="indianred", linewidth=1) +
  xlab(bquote(u[1])) +
  ylab(bquote(u[2])) +
  theme_bw()

#######################################
# Code to sample u ###################
#######################################
# Boundaries for f(u1|u2)
lower_bound_u1 <- function(x, w=1.999){
  -sqrt( (4-(2+w)*x^2) / (2-w) )
}
upper_bound_u1 <- function(x, w=1.999){
  sqrt( (4-(2+w)*x^2) / (2-w) )
}


# Boundaries for f(u2|u1)
lower_bound_u2 <- function(x, w=1.999){
  -sqrt( (4-(2-w)*x^2) / (2+w) )
}
```

```r
upper_bound_u2 <- function(x, w=1.999){
  sqrt( (4-(2-w)*x^2) / (2+w) )
}

sample_gibbs_u <- function(u1=0, u2=0, n=1000){
  # Empty vectors to store sampled points
  sample_u1 <- c()
  sample_u2 <- c()

  for(iter in 1:1000){
    # Lower and upper values of the ellipse conditioned on u2
    lower <- lower_bound_u1(u2)
    upper <- upper_bound_u1(u2)

    # Sample new value for u1* = y(u1|u2)
    new_u1 <- runif(1, lower, upper)

    # Sample new value for u2* = y(u2|u1*)
    lower <- lower_bound_u2(new_u1)
    upper <- upper_bound_u2(new_u1)
    new_u2 <- runif(1, lower, upper)

    # Accept the new sampled point
    u1 <- new_u1
    u2 <- new_u2
    # Saves the sampled point
    sample_u1[iter] <- u1
    sample_u2[iter] <- u2
  }
  sample <- data.frame(u1=sample_u1, u2=sample_u2)
  return(sample)
}

sample_u <- sample_gibbs_u()
ggplot(sample_u, aes(x=u1, y=u2)) +
  geom_point(size=0.5, colour="steelblue") +
  geom_ellipse(aes(x0=0, y0=0, a=sqrt(4/0.001), b=sqrt(4/3.999), angle=0),
               colour="indianred", linewidth=0.5) +
  xlab(bquote(u[1])) +
  ylab(bquote(u[2])) +
  theme_bw()

sum((sample_u$u2 + sample_u$u1)/2 > 0) / 1000
```