Lab 4 - Computational Statistics (732A89)

Helena Llorens Lluís (hllor282), Yi Yang (yiyan338)

QUESTION 1: Computations with Metropolis–Hastings

First, the target function is defined as

$$f(x) = 120x^5e^{-x}, \quad x > 0$$

We are going to generate a Markov chain with the Metropolis-Hastings (MH) algorithm, which generates a sequence of dependent observations which follow the target distribution approximately. The next observation of the chain is generated based on a proposal distribution which depends on the current observation.

For this assignment, we are asked to try three different proposal distributions.

1. Normal: $g_1(X) \sim N(\mu = X_t, \sigma = 0.1)$
2. Chi squared: $g_2(X) \sim \chi^2(\lfloor X_t + 1 \rfloor)$
3. Uniform: $g_3(X) \sim U(X_t - 0.5, X_t + 0.5)$

The Uniform is proposed because it is a symmetric distribution, and it makes the computation straightforward. Compared to the normal proposal, which can generate very small steps, the uniform proposal ensures a consistent range of exploration. Unlike the Chi-square proposal, which can introduce very large jumps, the uniform step size prevents excessive variation.

For the first proposed distribution, the chain looks as follows.

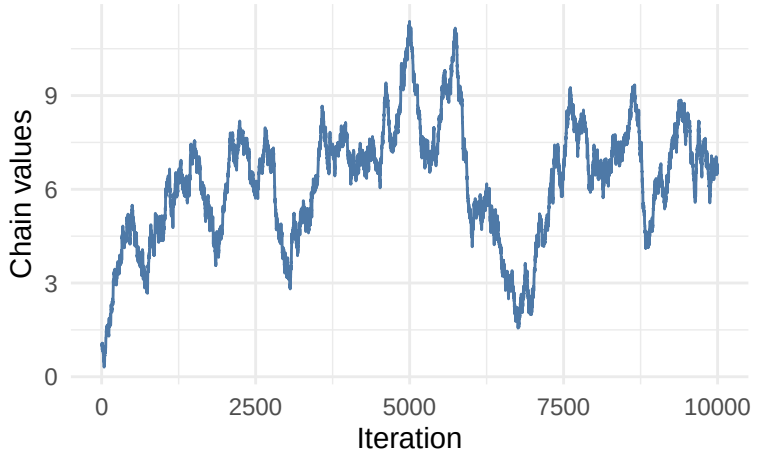

Figure 1: Chain with proposed Normal distribution

The chain appears to mix well and explore the distribution efficiently. The acceptance rate is 0.985, indicating that most proposals are accepted. However, the small step size may slow convergence. The plot shows a

clear upward trend in the first 2000-2500 iterations, meaning the chain is still moving toward the stationary distribution during that phase. For this chain, a burn-in of 2500 iterations is recommended to remove the initial transient phase.

In the plot below, we compare the sample distribution (histogram) with the density of the target distribution (orange line).

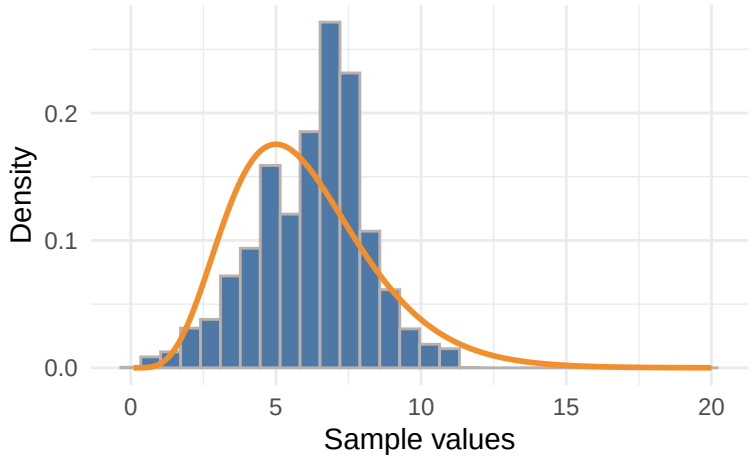

Figure 2: Histogram of sample with Normal proposed distribution

For the second proposed distribution, the chain looks as follows.

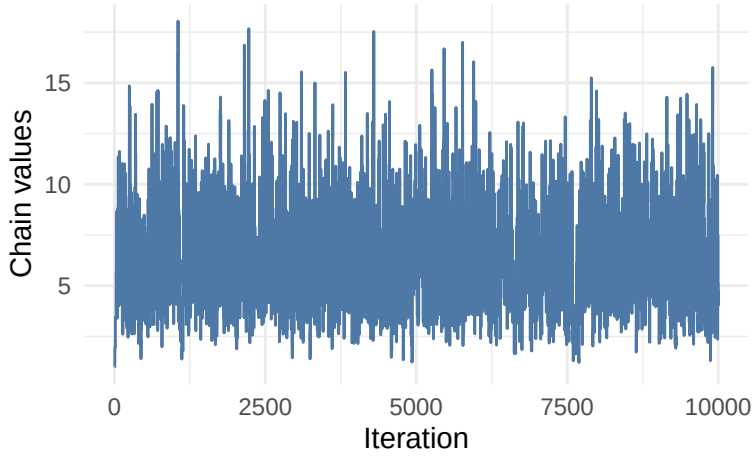

Figure 3: Chain with proposed Chi distribution

The chain exhibits larger jumps, suggesting a more exploratory behavior. The acceptance rate is 0.5288, lower than the normal case, meaning more proposals are rejected. This could lead to a more robust exploration of the target distribution. The chain seems to fluctuate around a stationary distribution almost immediately, which suggest a sort burn-in, about the first 500 iterations.

In the plot below, we compare the sample distribution with the density of the target distribution.

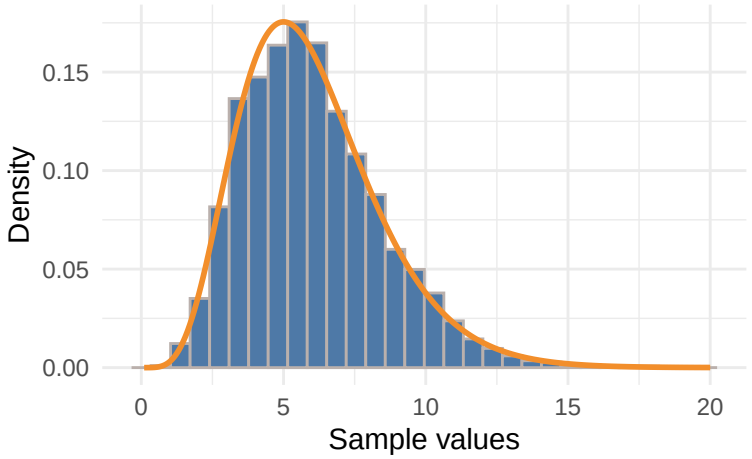

Figure 4: Histogram of sample with Chi proposed distribution

For the third proposed distribution, the chain looks as follows.

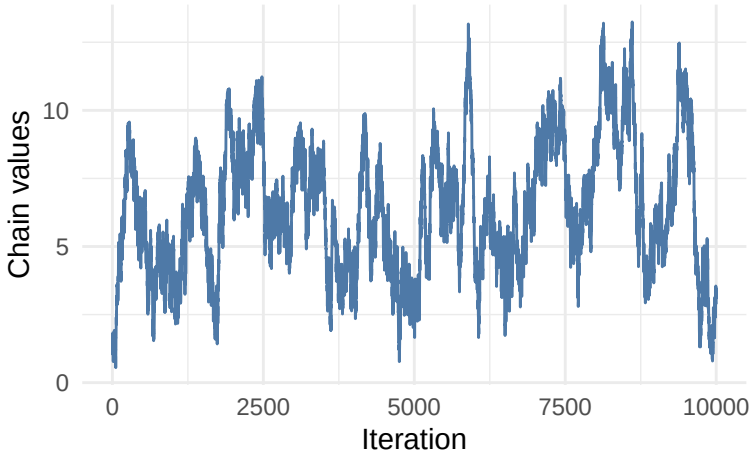

Figure 5: Chain with proposed Uniform distribution

The chain moves moderately between states with an acceptance rate of 0.956 The Uniform proposal offers a balanced exploration but may still be less efficient than the normal proposal. the chain appears to start at lower values and gradually reach a region of more stable fluctuation. The first few iterations (likely the first 1000 to 2000 iterations) show a clear upward trend before the chain appears to mix well.

In the plot below, we compare the sample distribution with the density of the target distribution.

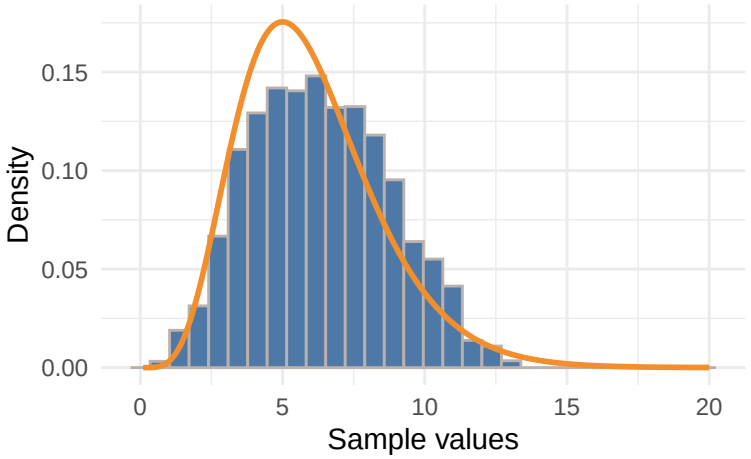

Figure 6: Histogram of sample with Uniform proposed distribution

We can estimate $E(X)$ from the samples as

$$E(X) \approx \frac{1}{N} \sum_{t=1}^{N} X_t$$

The results are shown in the following table.

Table 1: Sample mean values

Proposal.distribution	Mean.values
Normal	6.238768
Chi	6.031685
Uniform	6.357880

Our target distribution follows a $Gamma(\alpha = 6, \beta = 1)$, since the $Gamma$ distribution has following probability density function:

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

Then, we can calculate rge theoretical expected value as

$$E(X) = \alpha\beta = 6 \cdot 1 = 6$$

As observed, the MH algorithm with the three proposal distributions provides a reliable estimation of $E(X)$.

QUESTION 2: Gibbs sampling

The boundaries of X with density $f(x_1, x_2) \propto \mathbf{1}\{x_1^2 + w x_1 x_2 + x_2^2 < 1\}$ when $W = 1.999$ are shown as follows:

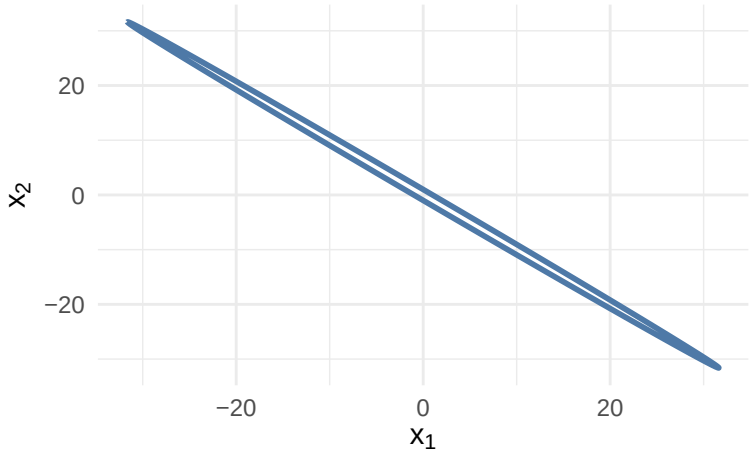

Figure 7: Boundaries of X

The conditional distribution of `X1` given `X2` is a uniform distribution on the interval:

$$\frac{-1.999X_2 - \sqrt{1.999^2X_2^2 + 4(1 - X_2^2)}}{2} < X_1 < \frac{-1.999X_2 + \sqrt{1.999^2X_2^2 + 4(1 - X_2^2)}}{2}$$

Since X has a uniform distribution, the density of

$$f(x_1|x_2) = \frac{1}{\text{interval length}}$$

where the interval length is given by:

$$\text{Interval Length} = \sqrt{1.999^2x_2^2 + 4(1 - x_2^2)}$$

Thus:

$$f(x_1 \mid x_2) = \frac{1}{\sqrt{1.999^2x_2^2 + 4(1 - x_2^2)}}$$

The conditional distribution of `X1` given `X2` has a similiar distribution :

$$f(x_2 \mid x_1) = \frac{1}{\sqrt{1.999^2x_1^2 + 4(1 - x_1^2)}}$$

We are going to generate 1000 random vectors with `Gibbs sampling` method, the resulting plot is shown as follows:

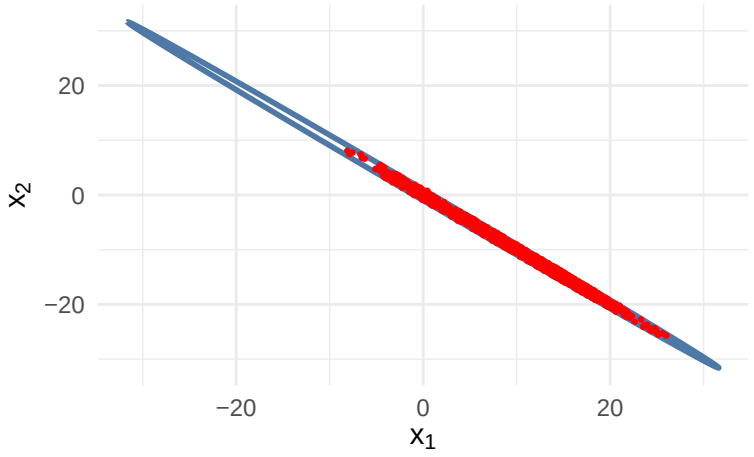

Figure 8: Gibbs Sampling Distribution

Then we repeat the algorithm 10 times and use the samples to calculate $P(X_1 > 0)$, the results are shown in the following table.

Table 2: Probability of X1>0

x
0.864
0.417
0.220
0.151
0.653
0.398
0.433
0.493
0.411
0.769

From the table we can observe that the probability of $X_1 > 0$ varies across different generations.However,the true probability should be **0.5**, as the boundaries of the density distribution are symmetric about the y-axis.

The reason why `Gibbs sampling` is less successful for $W = 1.999$ compared to $W = 1.8$ is that the distribution's boundaries become much narrower when $w = 1.999$.This leads to slower convergence in `Gibbs sampling`.Since `Gibbs sampling` updates one variable at a time while keeping the other variable fixed in this case.The narrow shape restricts the movement of $X1$ and $X2$.The sample values change less in each iteration,causing samples to get stuck in a limited region,which makes it difficult to cover the entire distribution efficiently.

Then, we transform the variable X and generate $U = (U_1, U_2) = (X_1 - X_2, X_1 + X_2)$ instead.By calculating $U_1 = X_1 - X_2$, $U_2 = X_1 + X_2$,we could determine the boundaries of the tranformed region. The plot is shown as follows:

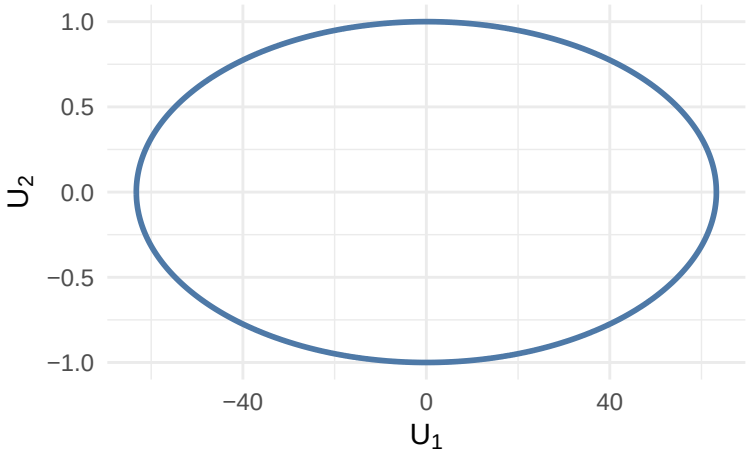

Figure 9: Boundaries of U

Then we use `Gibbs sampling` method to generate 1000 random variables and plot them.

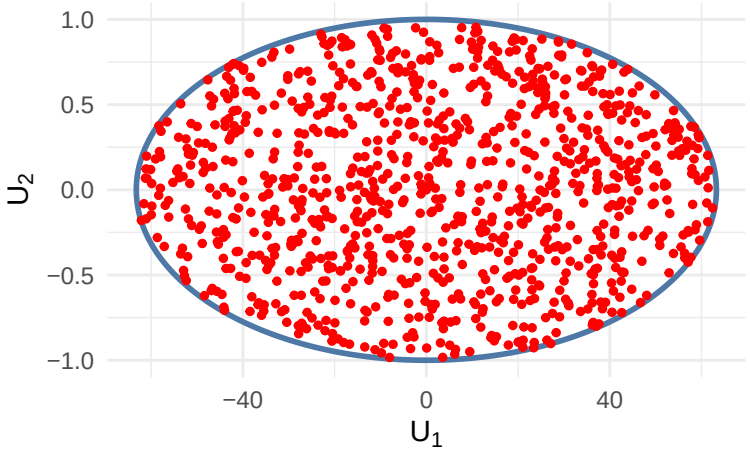

Figure 10: Gibbs Sampling Distribution of U

From the plot, we observe that the samples are approximately uniformly distributed within the boundaries. To verify this, we compute $P(X_1 > 0) = P((U_2 + U_1)/2 > 0)$ using the generated samples.

```
## The probability of X1 > 0: 0.519
```

Then we repeat the algorithm 10 times and the results are shown in the following table.

Table 3: Probability of X1>0

x
0.519
0.491
0.490

x
0.491
0.516
0.494
0.493
0.501
0.494
0.507

Compared to the result from part c, we observe that the probabilities approach 0.5 after the transformation, indicating improved generation ability.

APPENDIX

Question 1

```
library(ggplot2)
library(knitr)

# define target function
f <- function(x){
  ifelse(x > 0, 120*(x^5)*exp(-x), 0)
}

# Normalize f(x) so it integrates to 1
normalizing_constant <- integrate(f, lower = 0, upper = Inf)$value
f_normalized <- function(x) f(x) / normalizing_constant

# Generate points for the normalized density function
x_f <- seq(0.1, 20, length.out = 1000)
y_f <- sapply(x_f, f_normalized)
data_f <- data.frame(x = x_f, y = y_f)

# MH algorithm function
MetropolisHastings <- function(x_init, n_iter, prop_dist, prop_dist_rand, target){
  set.seed(1234)

  # initial values
  x <- numeric(length = n_iter)
  x[1] <- x_init
  acc <- 0

  for(t in 2:(n_iter)){

    # get candidate for x from the proposal distribution
    x_candidate <- prop_dist_rand(x[t - 1])

    # calculate acceptance rate
    r <- (target(x_candidate) * prop_dist(x[t-1], x_candidate)) / (target(x[t - 1]) * prop_dist(x_candi

    if (runif(1) < min(r, 1)) {
```

```r
      # Accept new candidate
      x[t] <- x_candidate
      acc <- acc + 1

    } else {
      # Stay at previous value
      x[t] <- x[t - 1]
    }
  }

  return(list(chain = x, acceptance_rate = sum(acc)/n_iter))
}

prop_normal <- function(x1, x2){dnorm(x1, mean = x2, sd = 0.1)}
prop_normal_rand <- function(x){rnorm(1, mean = x, sd = 0.1)}
res1 <- MetropolisHastings(1, 10000, prop_normal, prop_normal_rand, f)

prop_chi <- function(x1, x2){dchisq(x1, df = floor(x2))}
prop_chi_rand <- function(x){rchisq(1, df = floor(x))}
res2 <- MetropolisHastings(1, 10000, prop_chi, prop_chi_rand, f)

prop_unif <- function(x1, x2){dunif(x1, min = x2 - 0.5, max = x2 + 0.5)}
prop_unif_rand <- function(x){runif(1, x - 0.5, x + 0.5)}
res3 <- MetropolisHastings(1, 10000, prop_unif, prop_unif_rand, f)

data <- data.frame(it = 1:10000, x1 = res1$chain, x2 = res2$chain, x3 = res3$chain)

ggplot(data, aes(x = it, y = x1)) +
  geom_line(col = "#4E79A7") +
  theme_minimal() +
  labs(x = "Iteration", y = "Chain values")

ggplot(data, aes(x = x1)) +
  geom_histogram(aes(y = ..density..), color = "#BAB0AC", fill = "#4E79A7", bins = 30) +
  geom_line(data = data_f, aes(x = x, y = y), color = "#F28E2B", linewidth = 1) +
  theme_minimal() +
  labs(x = "Sample values", y = "Density")

ggplot(data, aes(x = it, y = x2)) +
  geom_line(col = "#4E79A7", size = 0.5) +
  theme_minimal() +
  labs(x = "Iteration", y = "Chain values")

ggplot(data, aes(x = x2)) +
  geom_histogram(aes(y = ..density..), color = "#BAB0AC", fill = "#4E79A7", bins = 30) +
  geom_line(data = data_f, aes(x = x, y = y), color = "#F28E2B", linewidth = 1) +
  theme_minimal() +
  labs(x = "Sample values", y = "Density")

ggplot(data, aes(x = it, y = x3)) +
  geom_line(col = "#4E79A7") +
  theme_minimal() +
  labs(x = "Iteration", y = "Chain values")
```

```r
ggplot(data, aes(x = x3)) +
  geom_histogram(aes(y = ..density..), color = "#BABOAC", fill = "#4E79A7", bins = 30) +
  geom_line(data = data_f, aes(x = x, y = y), color = "#F28E2B", linewidth = 1) +
  theme_minimal() +
  labs(x = "Sample values", y = "Density")

mean_values <- c(mean(data$x1), mean(data$x2), mean(data$x3))
distribution <- c("Normal", "Chi", "Uniform")
mean_data <- data.frame("Proposal distribution" = distribution, "Mean values" = mean_values)
kable(mean_data, caption = "Sample mean values")
```

Question 2

```r
library(ggplot2)
#a draw the boundary
w  <- 1.999
x_max <- sqrt(4 / (4 - w^2))  # maximum of x1
xv <- seq(-x_max, x_max, by=0.01)  # x1


x2_pos <- -(w/2)*xv + sqrt(1 - (1 - w^2/4) * xv^2)
x2_neg <- -(w/2)*xv - sqrt(1 - (1 - w^2/4) * xv^2)


ellipse_df <- data.frame(
  x1=c(xv,rev(xv)),
  x2=c(x2_pos,rev(x2_neg))
)

ggplot(ellipse_df,aes(x=x1,y=x2))+
  geom_path(color="#4E79A7",size=1)+
  labs(x=expression(x[1]),y=expression(x[2]))+
  theme_minimal()


#b

#c Gibbs sampling

gibbsSampling <- function(n,W){

  #initialize x1 and x2
  X1 <- numeric(n)
  X2 <- numeric(n)

  #initialize the first iteration
  X1[1] <- runif(1,-1,1)
  X2[1] <- runif(1,-1,1)

  for(i in 2:n){
   #generating x1 given x2
   X1_range <- c(-0.5*W*X2[i-1]-sqrt(1-(1-1/4*W^2)*X2[i-1]^2),-0.5*W*X2[i-1]+sqrt(1-(1-1/4*W^2)*X2[i-1]
   X1[i] <- runif(1,min = X1_range[1],max = X1_range[2])
```

```
   #generating x2 given x1
   X2_range <- c(-0.5*W*X1[i]-sqrt(1-(1-1/4*W^2)*X1[i]^2),-0.5*W*X1[i]+sqrt(1-(1-1/4*W^2)*X1[i]^2))
   X2[i] <- runif(1,min = X2_range[1],max = X2_range[2])

  }
   result <- data.frame(X1,X2)
   return(result)
}
set.seed(12345)
samples <- gibbsSampling(1000,1.999)
ggplot(ellipse_df,aes(x=x1,y=x2))+
  geom_path(color="#4E79A7",size=1)+
  labs(x=expression(x[1]),y=expression(x[2]))+
  geom_point(data=samples,aes(x = X1, y = X2), color = "red", size = 0.5) +
  theme_minimal()


#repeat the sampling
prob_x1 <- function(sample,n){
  prob <- sum(sample$X1>0)/n
  return(prob)

}

probs <- numeric(10)
sample_result <- list()
for (i in 1:10) {
  sample_result[[i]] <- gibbsSampling(1000,1.999)
  probs[i] <- prob_x1(sample_result[[i]],1000)
}
cat("The probabilities of X1 >0 in 10 times :",probs,"\n")


#d

#e generate U
#calculate U1 = X1 - X2, U2 = X1 + X2
U1<- c(xv - x2_pos, rev(xv - x2_neg))
U2 <- c(xv + x2_pos, rev(xv + x2_neg))

#plot the boundaries of U
ellipse_U_df <- data.frame(U1 = U1, U2 = U2)
ggplot(ellipse_U_df, aes(x = U1, y = U2)) +
  geom_path(color = "#4E79A7", size = 1) +
  labs(x = expression(U[1]), y = expression(U[2])) +
  theme_minimal()

#generate 1000 random variables using Gibbs Sampling
gibbsSampling_U <- function(n, W) {
  # initialize U1 and U2
  U1 <- numeric(n)
  U2 <- numeric(n)
```

```r
  # initialize the first iteration
  U1[1] <- 0
  U2[1] <- 0

  for (i in 2:n) {
    # given U2[i-1] generate U1[i]
    numerator_U1 <- 4 - (2 + W) * U2[i-1]^2
    U1_max <- sqrt(numerator_U1 / (2 - W))
    U1_range <- c(-U1_max, U1_max)

    U1[i] <- runif(1, min = U1_range[1], max = U1_range[2])

    # given U1[i] generate U2[i]
    numerator_U2 <- 4 - (2 - W) * U1[i]^2
    U2_max <- sqrt(numerator_U2 / (2 + W))
    U2_range <- c(-U2_max, U2_max)
    U2[i] <- runif(1, min = U2_range[1], max = U2_range[2])
  }
  data.frame(U1, U2)
}
set.seed(12345)
sample_U <- gibbsSampling_U(1000,1.999)

#plot the samples of U
ggplot(ellipse_U_df, aes(x = U1, y = U2)) +
  geom_path(color = "#4E79A7", size = 1) +
  labs(x = expression(U[1]), y = expression(U[2])) +
  geom_point(data=sample_U,aes(x = U1, y = U2), color = "red", size = 1) +
  theme_minimal()

#calculate P(x1>0)
samples_U_transformed_df <- data.frame(
  X1=(sample_U$U1+sample_U$U2)/2,
  X2=(sample_U$U2-sample_U$U1)/2
)

prob_X1_gibbs <- sum(samples_U_transformed_df$X1 > 0) /1000

cat("The probability of X1 > 0:",prob_X1_gibbs,"\n")


#repeat the sampling
prob_U <- function(sample,n){
  prob <- sum((sample$U2 + sample$U1) / 2 > 0) / 1000
  return(prob)

}
set.seed(12345)
probs_u <- numeric(10)
sample_U_result <- list()
for (i in 1:10) {
  sample_U_result[[i]] <- gibbsSampling_U(1000,1.999)
  probs_u[i] <- prob_U (sample_U_result[[i]],1000)
```

```
}
kable(probs_u,caption = "Probability of X1>0")
```