Group 2

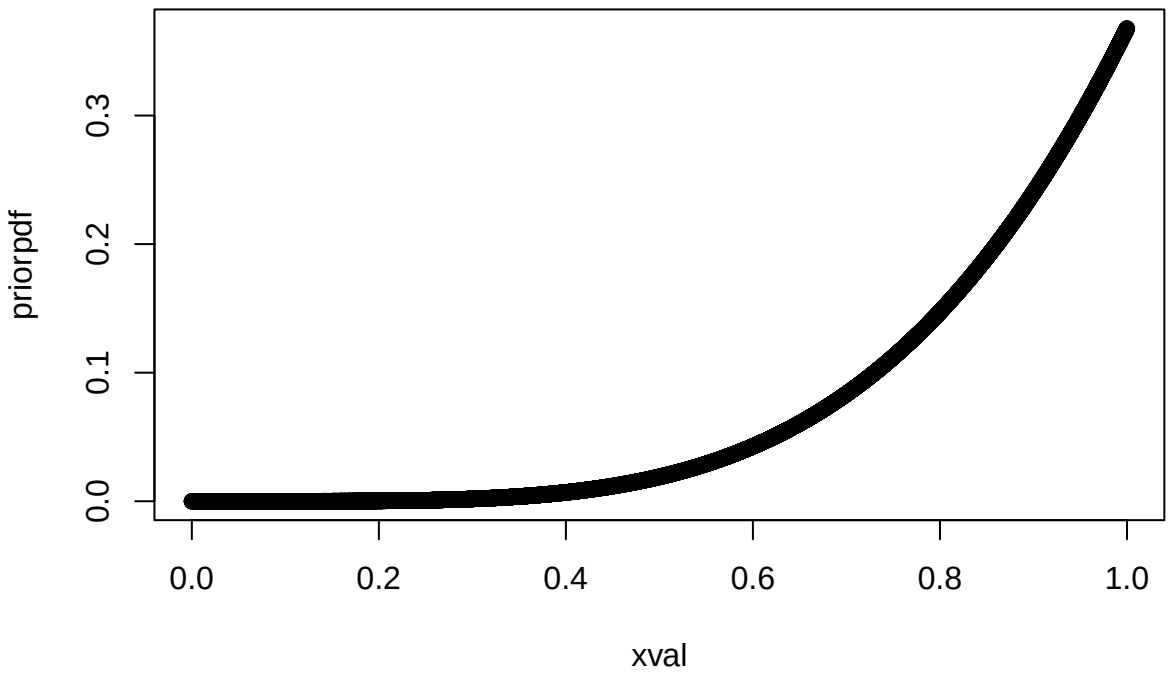
Greeshma Jeev Koothuparambil(greko370), Sangeeth Sankunny Menon(sansa237)

2023-11-08

Question 1: Computations with Metropolis–Hastings

Consider a random variabel X with the following probability density function:
$f(x) = x_5 e_{-x}, x > 0$
The distribution is known up to some constant of proportionality. If you are interested (NOT part of the Lab) this constant can be found by applying integration by parts multiple times and equals 120 The prior function looks like this:

```
library(ggplot2)


#prior function
posterior <- function(x){
  value <- (x^5) * exp(-x)
  return(value)
}
xval <- runif(10000)
priorpdf <- c()
for (i in 1:10000) {
  priorpdf[i] <- posterior(xval[i])
}

plot(x= xval, y= priorpdf)
```



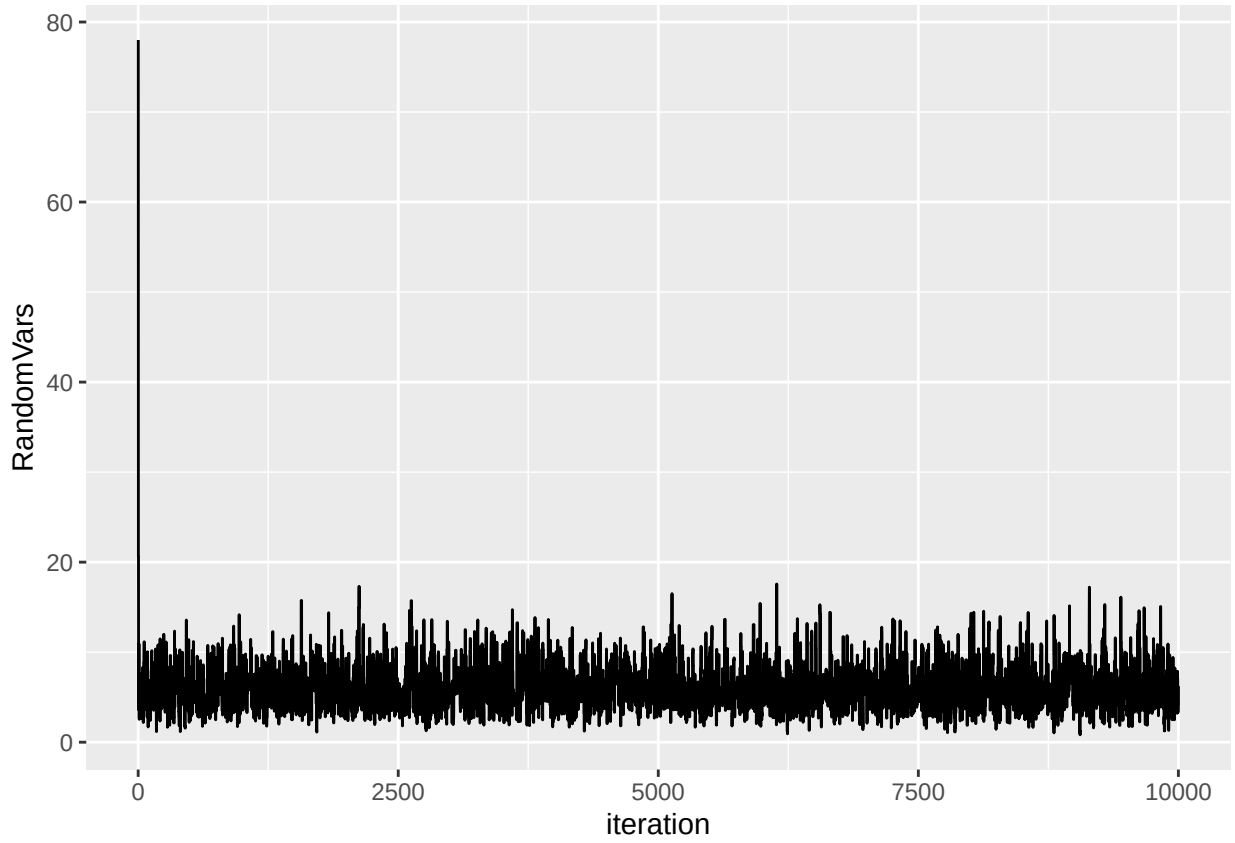
a. Use the Metropolis–Hastings algorithm to generate 10000 samples from this distribution by using a log–normal $LN(X_t, 1)$ proposal distribution; take some starting point. Plot the chain you obtained with iterations on the horizontal axis. What can you guess about the convergence of the chain? If there is a burn–in period, what can be the size of this period? What is the acceptance rate? Plot a histogram of the sample.

```r
# Metropolis-Hastings Method:
MHMethodlog <- function(posterior, n, x0){
  random <- c()
  accept <- 0

    for (i in 1:n) {

      randomx <- rlnorm(1, meanlog = log(x0), sdlog = 1)
      post_random <- posterior(randomx)
      post_chosen <- posterior(x0)
      prop_numerator <- dlnorm(x0, meanlog = log(randomx),sdlog = 1)
      prop_denominator <- dlnorm(randomx, meanlog = log(x0), sdlog = 1)

      MHRatio <- (post_random* prop_numerator)/(post_chosen* prop_denominator)
      test <- min(1, MHRatio)
      ap    <- runif(1)
      if (ap<test){

        x0 <- randomx
        accept <- accept+1
      }
```

```
      random[i] <- x0
    }
  acceptrate <- accept/n
  retvals <-list("RandomVars" = random, "AcceptanceRate" = acceptrate)
  return(retvals )
}
set.seed(13)
LNpost <- MHMethodlog(posterior = posterior, 10000,78)
LNData <- as.data.frame(LNpost[1])
LNData$iteration <- 1:10000
```

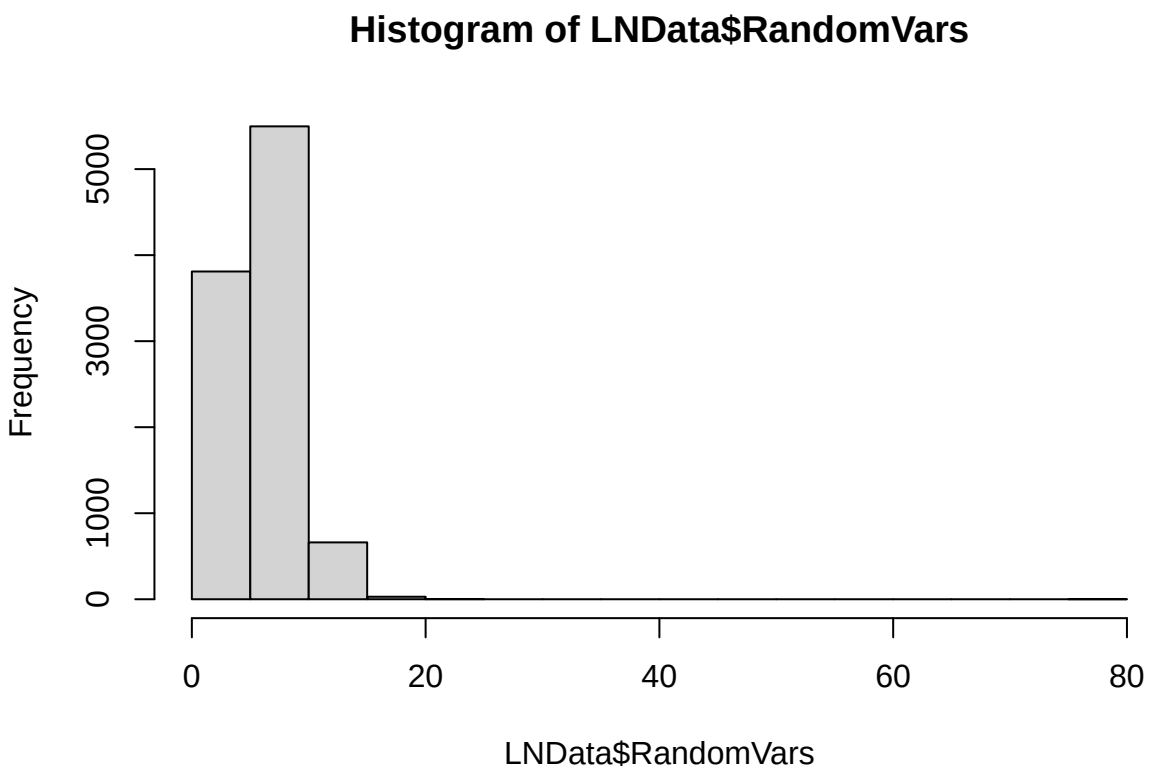
The chain obtained is as follows:

```
ggplot(LNData,mapping = aes(x= iteration, y=RandomVars))+ geom_line()
```



The histogram on Log Normal Proposal function looks lilke this:

```
hist(LNData$RandomVars)
```



The convergence of the plot was attained pretty soon. In first 3 iterations the values of random variable generated reached 3.61.

The average acceptance rate is 0.4425.

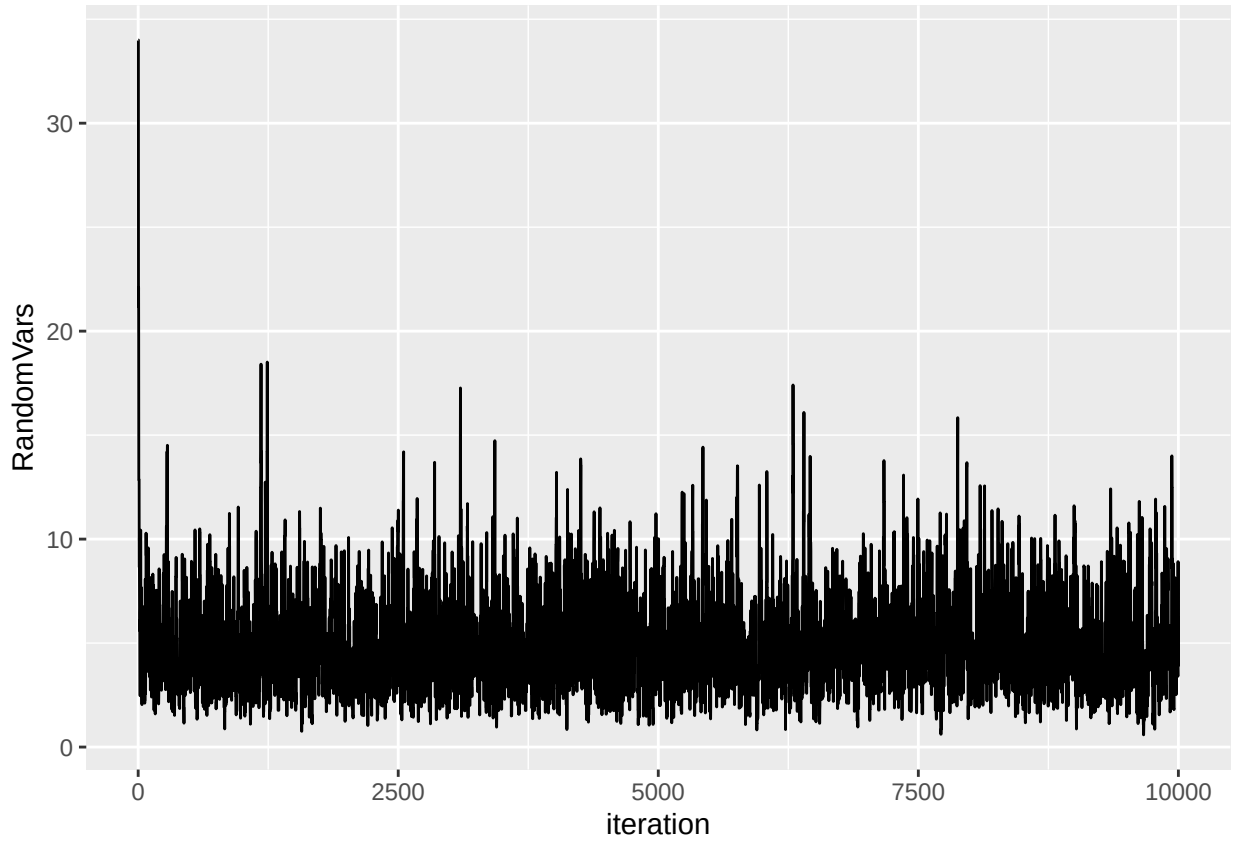
b. Perform Part a by using the chi–square distribution $\chi^2(\lfloor Xt + 1 \rfloor)$ as a proposal distribution, where x is the floor function, meaning the integer part of x for positive x, i.e. $2.95 = 2$

```
# Metropolis-Hastings Method Chi squared:
MHMethodchi <- function(posterior, n, x0){
  random <- c()
  accept <- 0

  for (i in 1:n) {

    randomx <- rchisq(1, df = floor(x0+1))
    post_random <- posterior(randomx)
    post_chosen <- posterior(x0)
    prop_numerator <- dchisq(randomx,df = floor(randomx+1))
    prop_denominator <- dchisq(x0, df= floor(x0+1))

    MHRatio <- (post_random* prop_numerator)/(post_chosen* prop_denominator)
    ap    <- runif(1)
    if (ap<MHRatio){

      x0 <- randomx
      accept <- accept+1
    }
```

```
    random[i] <- x0
  }
  acceptrate <- accept/n
  retvals <-list("RandomVars" = random, "AcceptanceRate" = acceptrate)
  return(retvals )
}
set.seed(13)
chipost <- MHMethodchi(posterior = posterior, 10000,34)
chiData <- as.data.frame(chipost[1])
chiData$iteration <- 1:10000
```

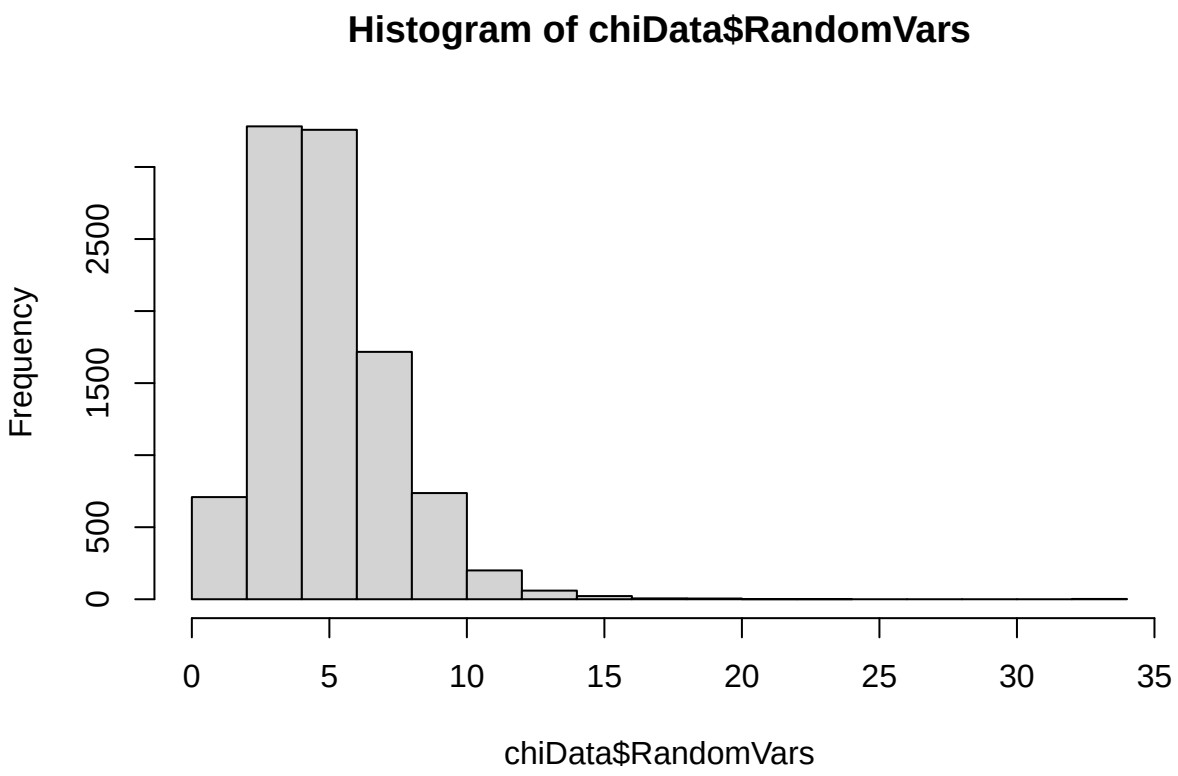
The chain obtained is as follows:

```
ggplot(chiData,mapping = aes(x= iteration, y=RandomVars))+ geom_line()
```



The histogram on Log Normal Proposal function looks lilke this:

```
hist(chiData$RandomVars)
```



The convergence of the plot was attained. From 10th iteration onwards the data is in a reasonable range. So the burn-in period can be taken as 10.

The average acceptance rate is 0.6015.

c. 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.

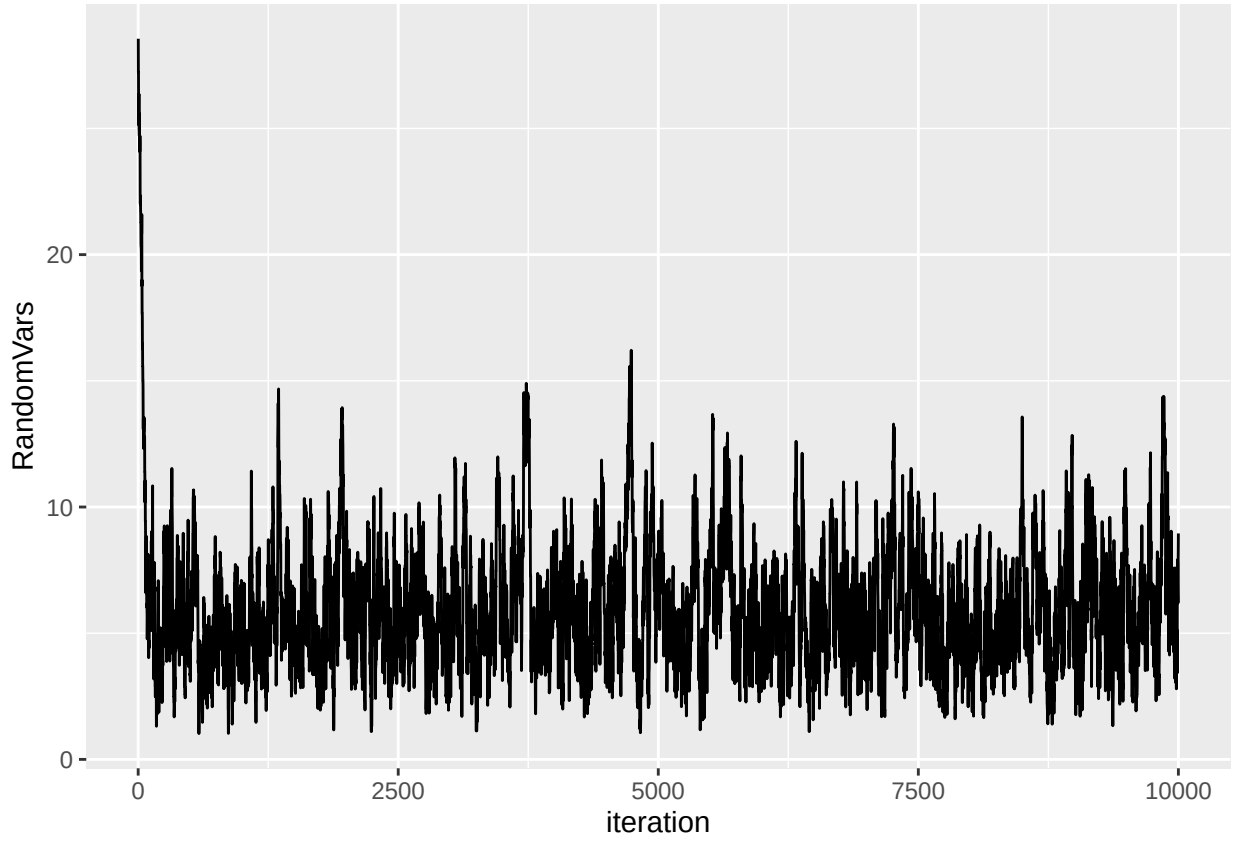
The proposal distribution chosen is a normal distribution with standard deviation 1.

```
# Metropolis-Hastings Method normal:
MHMethodnor <- function(posterior, n, x0){
  random <- c()
  accept <- 0

  for (i in 1:n) {

    randomx <- rnorm(1, mean=x0, sd = 1)
    post_random <- posterior(randomx)
    post_chosen <- posterior(x0)
    prop_numerator <- dnorm(x0, mean = randomx, sd = 1)
    prop_denominator <- dnorm(randomx, mean = x0, sd = 1)

    MHRatio <- (post_random* prop_numerator)/(post_chosen* prop_denominator)
    test <- min(1, MHRatio)
    ap    <- runif(1)
    if (ap<test){
```

```
      x0 <- randomx
      accept <- accept+1
    }
    random[i] <- x0
  }
  acceptrate <- accept/n
  retvals <-list("RandomVars" = random, "AcceptanceRate" = acceptrate)
  return(retvals )
}
set.seed(13)
Npost <- MHMethodnor(posterior = posterior, 10000,28)
NData <- as.data.frame(Npost[1])
NData$iteration <- 1:10000
```

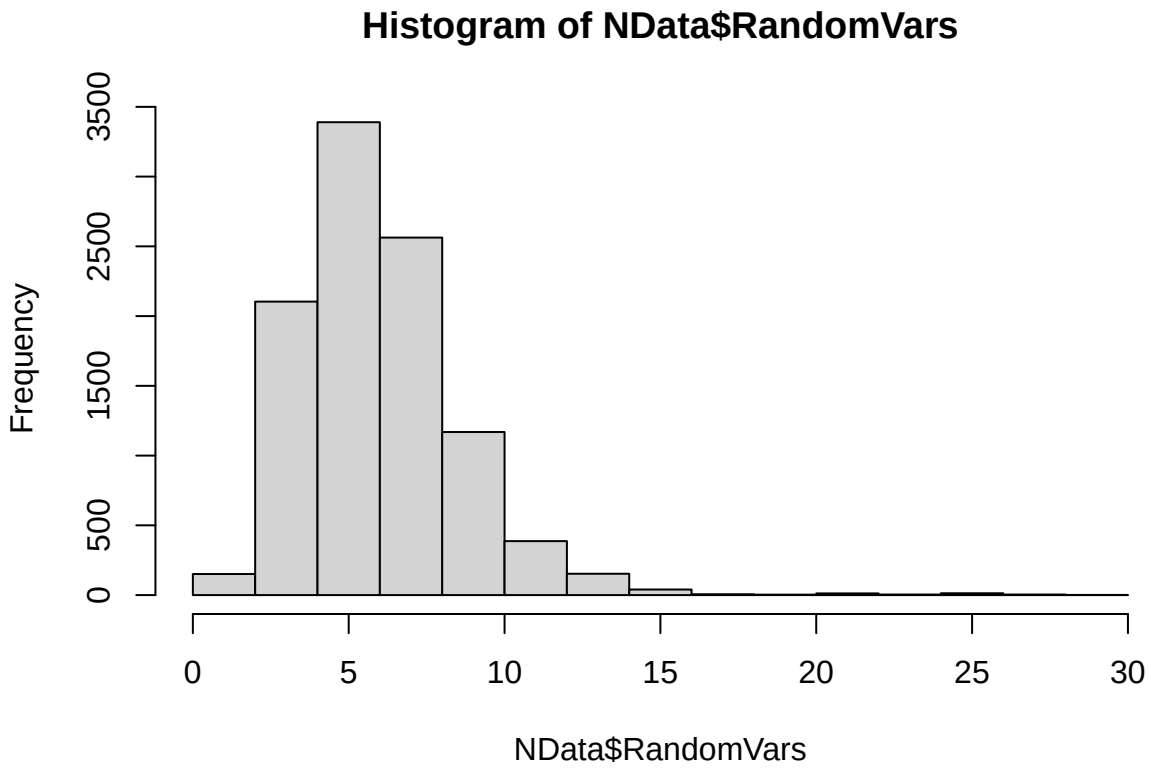
The chain obtained is as follows:

```
ggplot(NData,mapping = aes(x= iteration, y=RandomVars))+ geom_line()
```



The histogram on Log Normal Proposal function looks like this:

```
hist(NData$RandomVars)
```



The convergence of the plot was attained. The convergence see slower. From 71st iteration onwards the data is in a reasonable range. So the burn-in period can be taken as 71.

The average acceptance rate is 0.8608.

d. Compare the results of Parts a, b, and c and make conclusions. The acceptance rate of all the sampling methods seems good. But all of the sampled random variables are concentrated in the range of 0 to 8.
Summary of Log Normal Random Variables :
0.8197428, 4.2414977, 5.7049063, 6.0081149, 7.3317977, 78

Summary of Chi Squared Random Variables :
0.5881459, 3.1547069, 4.5417697, 4.8992847, 6.1567859, 34

Summary of Normal Random Variables :
1.0241771, 4.1353739, 5.6347278, 5.966913, 7.3050579, 28.5543269

e. Estimate

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

using the samples from Parts a, b, and c. Integration of the x*f(x) results in the mean value of a distribution.

Mean of Log Normal Random variables is : 6.0081149

Mean of Chi Squared Random variables is : 4.8992847

Mean of Normal Random variables is : 5.966913

f. 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.

The gamma distribution generated is as follows : G(6,1) where $\alpha = 6$ and $\beta = 1$.

Therefore the Mean Value of the generated gamma is $\alpha^* \beta = 6$.
The mean values obtained from all the above Metropolis Hastings methods falls around the value 6.

Question 2: Gibbs sampling

Let X = (X1,X2) be a bivariate distribution with density

$$f(x_1, x_2) = 1\{x_1^2 + wx_1x_2 + x_2^2 < 1\}$$

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).

a. 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.

```
#2.1

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

b. What is the conditional distribution of X_1 given X_2 and that of X_2 given X_1 ?

Conditional Distribution for X_1 given X_2 is on the interval:

$$[-w/2 * x2 - sqrt(1 - (1 - w^2/4) * x2^2) \ , \ w/2 * x2 + sqrt(1 - (1 - w^2/4) * x2^2)]$$

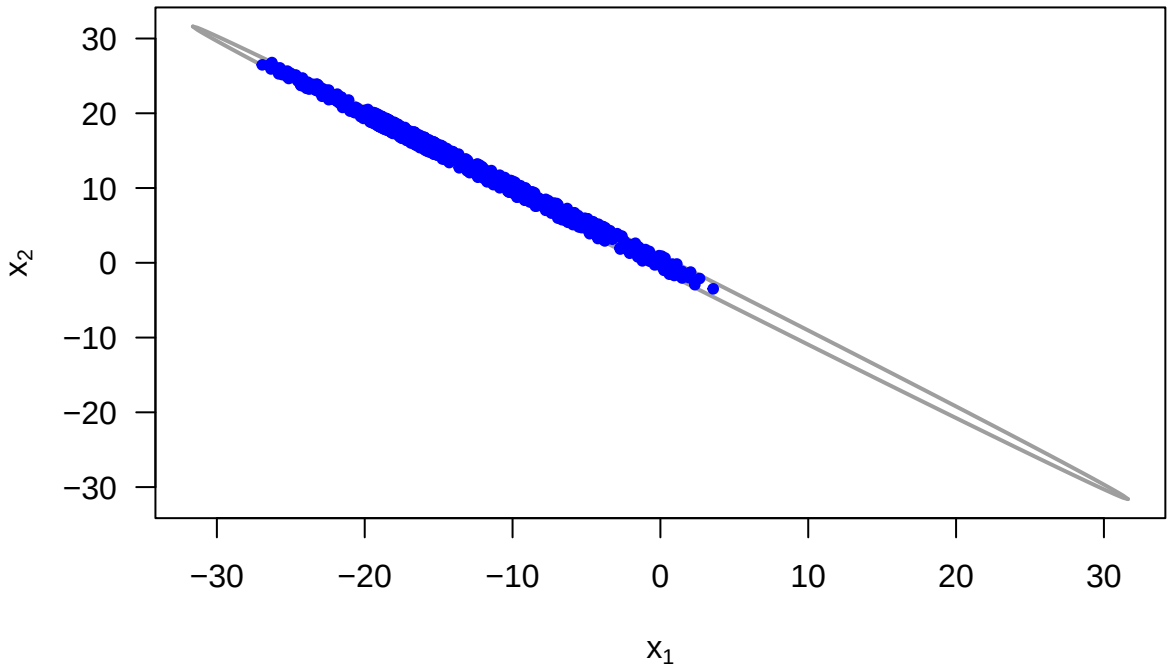
Conditional Distribution for X_2 given X_1 is on the interval:

$$[-w/2 * x1 - sqrt(1 - (1 - w^2/4) * x1^2) \ , \ w/2 * x1 + sqrt(1 - (1 - w^2/4) * x1^2)]$$

```
# Function to sample from conditional distribution of x2 given x1
cd_x2x1 <- function(x1) {
  lower_boundary <- -w/2 * x1 - sqrt(1 - (1 - w^2 / 4) * x1^2)
  upper_boundary <- -w/2 * x1 + sqrt(1 - (1 - w^2 / 4) * x1^2)
  return(runif(1, min = lower_boundary, max = upper_boundary))
}

# Function to sample from conditional distribution of x1 given x2
cd_x1x2 <- function(x2) {
  lower_boundary <- -w/2 * x2 - sqrt(1 - (1 - w^2 / 4) * x2^2)
  upper_boundary <- -w/2 * x2 + sqrt(1 - (1 - w^2 / 4) * x2^2)
  return(runif(1, min = lower_boundary, max = upper_boundary))
}
```

c. 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?

```r
#2.3

# Gibbs sampling function
gibbs_sampling <- function( n) {

  x1_vals <- c()
  x2_vals <- c()

  # Initial values for X1 and X2
  x1 <- 0
  x2 <- 0


  for (i in 1:n) {
    x1 <- cd_x1x2(x2)
    x2 <- cd_x2x1(x1)

    x1_vals[i] <- x1
    x2_vals[i] <- x2
  }
  samples <- list("x1" = x1_vals,"x2"= x2_vals)
  return(samples)
}


n <- 1000  # Number of iterations

# Run Gibbs sampling
generated_samples <- gibbs_sampling(n)
plot(xv, xv, type = "n", xlab = expression(x[1]), ylab = expression(x[2]), las = 1)

# Plot the boundary ellipse
lines(xv, -(w / 2) * xv - sqrt(1 - (1 - w^2 / 4) * xv^2), lwd = 2, col = 8)
lines(xv, -(w / 2) * xv + sqrt(1 - (1 - w^2 / 4) * xv^2), lwd = 2, col = 8)

# Plot the generated samples
points(generated_samples$x1, generated_samples$x2, col = "blue", pch = 20)
```




```
# Estimate P(X1 > 0) based on the sample
prob_x1_greater_than_0 <- c()
for (i in 1:15) {

  generated_samples <- gibbs_sampling(n)
  prob_x1_greater_than_0[i] <- mean(generated_samples$x1 > 0)
}

print(paste("Average ofEstimated P(X1 > 0) for 15 iterations:", mean(prob_x1_greater_than_0)))
```

```
## [1] "Average ofEstimated P(X1 > 0) for 15 iterations: 0.655933333333333"
```

The ellipse is perfectly divided into half at x1=0. The probability can be determined by taking half of the area of the ellipse and divide it by total area. There fore the probability will be 0.5 * Total areal of the ellipse divided by Total area which equals *0.5*.

d. 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. The gibbs sampling for w= 1.999 has not covered the entire area of the ellipse.

The shape of the ellipse has become extremely elongated. Thus the gibbs algorithm is restricted to cover entire area as increased eccentricity develops highly correlated conditional distributions. This causes convergence time or iterations to increase.

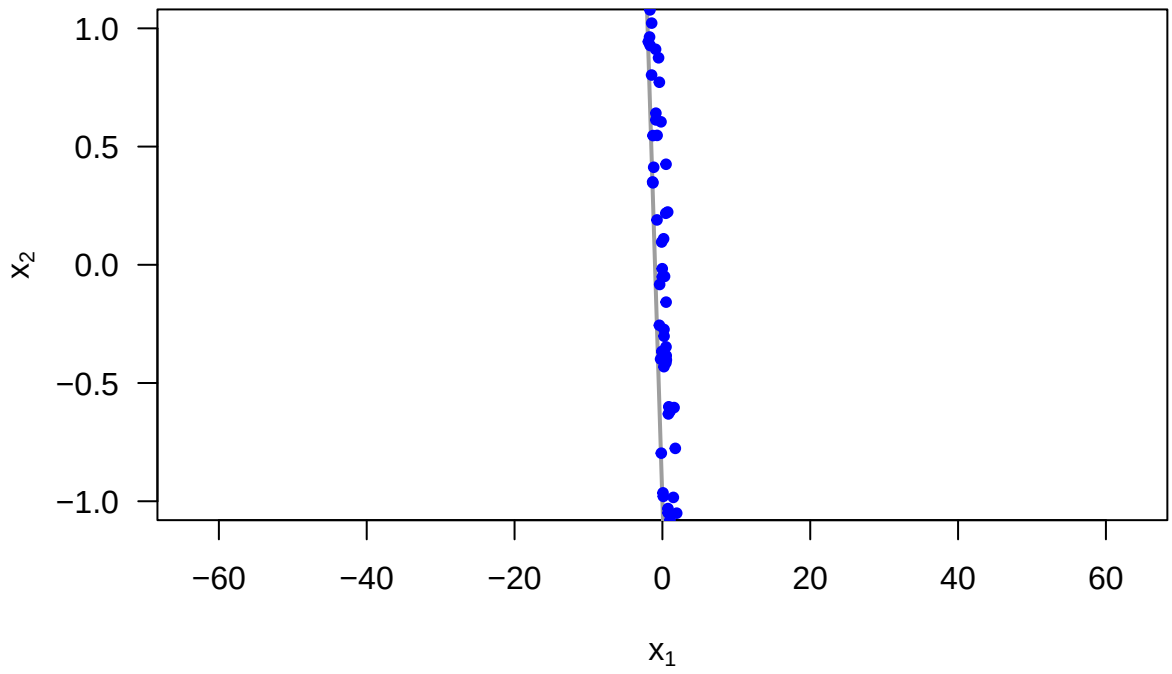
e. We might transform the variable X and generate $U = (U_1, U_2) = (X_1 - X_2, X_1 + X_2)$ instead. In this case, the density of the transformed variable $U = (U1, U2)$ is again a uniform distribution on a transformed region (no proof necessary for this claim). Determine the boundaries of the transformed 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 Xi. Plot the boundaries for* (U_1, U_2)*.Generate* $n = 1000$ *random vectors with Gibbs sampling for U and plot them. Determine* $P(X_1 > 0) = P((U_2 + U_1)/2 > 0)$*. Compare the results with Part c.*

```
# 2.5 Transforming Variables

# Generate sequences for x1 and x2
x1<-seq(-1,1,0.01)
x2<-seq(-1,1,0.01)
# Transform variables to U1 and U2
U1<- x1+x2
U2<- x1-x2

xv_U1 <- U1 * 1/sqrt(1-w^2/4)
xv_U2 <- U2 * 1/sqrt(1-w^2/4)
#Plot
plot(xv_U1,xv_U2, type="n", xlab=expression(x[1]), ylab=expression(x[2]), las=1)

lines(xv_U1, -(w/2)*xv_U1-sqrt(1-(1-w^2/4)*xv_U1^2), lwd=2, col=8)
lines(xv_U2, -(w/2)*xv_U2+sqrt(1-(1-w^2/4)*xv_U2^2), lwd=2, col=8)
# Define conditional distribution functions for U
cd_u2u1 <- function(u1) {
  lower_boundary <- -w/2 * u1 - sqrt(1 - (1 - w^2 / 4) * u1^2)
  upper_boundary <- -w/2 * u1 + sqrt(1 - (1 - w^2 / 4) * u1^2)
  return(runif(1, min = lower_boundary, max = upper_boundary))
}
cd_u1u2 <- function(u2) {
  lower_boundary <- -w/2 * u2 - sqrt(1 - (1 - w^2 / 4) * u2^2)
  upper_boundary <- -w/2 * u2 + sqrt(1 - (1 - w^2 / 4) * u2^2)
  return(runif(1, min = lower_boundary, max = upper_boundary))
}
n <- 1000
u1 <- numeric(n)
u2 <- numeric(n)
u1[1] <- runif(1, min = -1, max = 1)
u2[1] <- runif(1, min = -1, max = 1)
# Gibbs sampling
for (i in 2:n) {
  u1[i] <- cd_u1u2(u2[i - 1])
  u2[i] <- cd_u2u1(u1[i])
}
#Probability calculation
total <- (u1 + u2) / 2
probabilty <- mean(total > 0)
cat("Estimated P(X1 > 0) for U:", probabilty, "\n")
```

```
## Estimated P(X1 > 0) for U: 0.492
```

```
points(u1, u2, col = "blue", pch = 20)
```



***

##APPENDIX

##Code for Question 1:

```
library(ggplot2)


#prior function
posterior <- function(x){
  value <- (x^5) * exp(-x)
  return(value)
}
xval <- runif(10000)
priorpdf <- c()
for (i in 1:10000) {
  priorpdf[i] <- posterior(xval[i])
}

plot(x= xval, y= priorpdf)

#log- normal function

postpdf  <- dlnorm(xval,0,1)
plot(x= xval, y= postpdf)

# Metropolis-Hastings Method:
```

```r
MHMethodlog <- function(posterior, n, x0){
  random <- c()
  accept <- 0
  random[1] <- x0

    for (i in 1:n) {

      randomx <- rlnorm(1, meanlog = log(x0), sdlog = 1)
      post_random <- posterior(randomx)
      post_chosen <- posterior(x0)
      prop_numerator <- dlnorm(x0, meanlog = log(randomx),sdlog = 1)
      prop_denominator <- dlnorm(randomx, meanlog = log(x0), sdlog = 1)

      MHRatio <- (post_random* prop_numerator)/(post_chosen* prop_denominator)
      test <- min(1, MHRatio)
      ap    <- runif(1)
      if (ap<=test){

        x0 <- randomx
        accept <- accept+1
      }
      random[i] <- x0
    }
  acceptrate <- accept/n
  retvals <-list("RandomVars" = random, "AcceptanceRate" = acceptrate)
  return(retvals )
}
set.seed(13)
LNpost <- MHMethodlog(posterior = posterior, 10000,28)
LNData <- as.data.frame(LNpost[1])
LNData$iteration <- 1:10000

ggplot(LNData,mapping = aes(x= iteration, y=RandomVars))+ geom_line()
hist(LNData$RandomVars)

#Acceptance Rate
LNpost$AcceptanceRate


## b

# Metropolis-Hastings Method Chi squared:
MHMethodchi <- function(posterior, n, x0){
  random <- c()
  accept <- 0

  for (i in 1:n) {

    randomx <- rchisq(1, df = floor(x0+1))
    post_random <- posterior(randomx)
    post_chosen <- posterior(x0)
    prop_numerator <- dchisq(randomx,df = floor(randomx+1))
    prop_denominator <- dchisq(x0, df= floor(x0+1))
```

```
    MHRatio <- (post_random* prop_numerator)/(post_chosen* prop_denominator)
    ap    <- runif(1)
    if (ap<MHRatio){

      x0 <- randomx
      accept <- accept+1
    }
    random[i] <- x0
  }
  acceptrate <- accept/n
  retvals <-list("RandomVars" = random, "AcceptanceRate" = acceptrate)
  return(retvals )
}
set.seed(13)
chipost <- MHMethodchi(posterior = posterior, 10000,0.01)
chiData <- as.data.frame(chipost[1])
chiData$iteration <- 1:10000

ggplot(chiData,mapping = aes(x= iteration, y=RandomVars))+ geom_line()
hist(chiData$RandomVars, 40)

#Acceptance Rate
chipost$AcceptanceRate


##c

# Metropolis-Hastings Method normal:
MHMethodnor <- function(posterior, n, x0){
  random <- c()
  accept <- 0

  for (i in 1:n) {

    randomx <- rnorm(1, mean=x0, sd = 1)
    post_random <- posterior(randomx)
    post_chosen <- posterior(x0)
    prop_numerator <- dnorm(x0, mean = randomx, sd = 1)
    prop_denominator <- dnorm(randomx, mean = x0, sd = 1)

    MHRatio <- (post_random* prop_numerator)/(post_chosen* prop_denominator)
    test <- min(1, MHRatio)
    ap    <- runif(1)
    if (ap<test){

      x0 <- randomx
      accept <- accept+1
    }
    random[i] <- x0
  }
  acceptrate <- accept/n
  retvals <-list("RandomVars" = random, "AcceptanceRate" = acceptrate)
  return(retvals )
```

```
}
set.seed(13)
Npost <- MHMethodnor(posterior = posterior, 10000,28)
NData <- as.data.frame(Npost[1])
NData$iteration <- 1:10000

ggplot(NData,mapping = aes(x= iteration, y=RandomVars))+ geom_line()
hist(NData$RandomVars)

#Acceptance Rate
Npost$AcceptanceRate



##e

mean(LNData$RandomVars)
mean(chiData$RandomVars)
mean(NData$RandomVars)
```

##Code for Question 2:

```
#2.1

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

plot(xv, xv, type = "n", xlab = expression(x[1]), ylab = expression(x[2]), las = 1)

# Plot the boundary ellipse
lines(xv, -(w / 2) * xv - sqrt(1 - (1 - w^2 / 4) * xv^2), lwd = 2, col = 8)
lines(xv, -(w / 2) * xv + sqrt(1 - (1 - w^2 / 4) * xv^2), lwd = 2, col = 8)


#2.2

# Function to sample from conditional distribution of x2 given x1
cd_x2x1 <- function(x1) {
  lower_boundary <- -w/2 * x1 - sqrt(1 - (1 - w^2 / 4) * x1^2)
  upper_boundary <- -w/2 * x1 + sqrt(1 - (1 - w^2 / 4) * x1^2)
  return(runif(1, min = lower_boundary, max = upper_boundary))
}

# Function to sample from conditional distribution of x1 given x2
cd_x1x2 <- function(x2) {
  lower_boundary <- -w/2 * x2 - sqrt(1 - (1 - w^2 / 4) * x2^2)
  upper_boundary <- -w/2 * x2 + sqrt(1 - (1 - w^2 / 4) * x2^2)
  return(runif(1, min = lower_boundary, max = upper_boundary))
}
#2.3

# Gibbs sampling function
gibbs_sampling <- function( n) {
```

```r
  x1_vals <- c()
  x2_vals <- c()

  # Initial values for X1 and X2
  x1 <- 0
  x2 <- 0


  for (i in 1:n) {
    x1 <- cd_x1x2(x2)
    x2 <- cd_x2x1(x1)

    x1_vals[i] <- x1
    x2_vals[i] <- x2
  }
  samples <- list("x1" = x1_vals,"x2"= x2_vals)
  return(samples)
}


n <- 1000  # Number of iterations

# Run Gibbs sampling
generated_samples <- gibbs_sampling(n)

# Plot the generated samples
points(generated_samples$x1, generated_samples$x2, col = "blue", pch = 20)

# Estimate P(X1 > 0) based on the sample
prob_x1_greater_than_0 <- c()
for (i in 1:15) {

  generated_samples <- gibbs_sampling(n)
  prob_x1_greater_than_0[i] <- mean(generated_samples$x1 > 0)
}

print(paste("Average ofEstimated P(X1 > 0) for 15 iterations:", mean(prob_x1_greater_than_0)))

#The ellipse is perfectly divided into half at x1=0. The probability can be determined
# by taking half of the area of the ellipse and divide it by total area:
#There fore the probability will be 0.5*Total  areal of the ellipse divided by Total
#area which equals 0.5.


#2.4
#The gibbs sampling for w= 1.999 has not covered the entire area of the ellipse

#The shape of the ellipse has become extremely elongated. Thus the gibbs algorithm
#is restricted to cover entire area as increased eccentricity develops highly correlated
#conditional distributions. This causes convergence time or iterations to increase.

# 2.5 Transforming Variables
```

```r
# Generate sequences for x1 and x2
x1<-seq(-1,1,0.01)
x2<-seq(-1,1,0.01)
# Transform variables to U1 and U2
U1<- x1+x2
U2<- x1-x2

xv_U1 <- U1 * 1/sqrt(1-w^2/4)
xv_U2 <- U2 * 1/sqrt(1-w^2/4)
#Plot
plot(xv_U1,xv_U2, type="n", xlab=expression(x[1]), ylab=expression(x[2]), las=1)

lines(xv_U1, -(w/2)*xv_U1-sqrt(1-(1-w^2/4)*xv_U1^2), lwd=2, col=8)
lines(xv_U2, -(w/2)*xv_U2+sqrt(1-(1-w^2/4)*xv_U2^2), lwd=2, col=8)
# Define conditional distribution functions for U
cd_u2u1 <- function(u1) {
  lower_boundary <- -w/2 * u1 - sqrt(1 - (1 - w^2 / 4) * u1^2)
  upper_boundary <- -w/2 * u1 + sqrt(1 - (1 - w^2 / 4) * u1^2)
  return(runif(1, min = lower_boundary, max = upper_boundary))
}
cd_u1u2 <- function(u2) {
  lower_boundary <- -w/2 * u2 - sqrt(1 - (1 - w^2 / 4) * u2^2)
  upper_boundary <- -w/2 * u2 + sqrt(1 - (1 - w^2 / 4) * u2^2)
  return(runif(1, min = lower_boundary, max = upper_boundary))
}
n <- 1000
u1 <- numeric(n)
u2 <- numeric(n)
u1[1] <- runif(1, min = -1, max = 1)
u2[1] <- runif(1, min = -1, max = 1)
# Gibbs sampling
for (i in 2:n) {
  u1[i] <- cd_u1u2(u2[i - 1])
  u2[i] <- cd_u2u1(u1[i])
}
#Probability calculation
total <- (u1 + u2) / 2
probabilty <- mean(total > 0)
cat("Estimated P(X1 > 0) for U:", probabilty, "\n")

points(u1, u2, col = "blue", pch = 20)
```
