Lab4

Ayesha Gamage-(ayega981)/Muditha Cherangani(mudch175)

2023-11-24

Question 1

Random variable X with the probability density function:

$$f(x) = cx^5 e^{-x}, x > 0$$

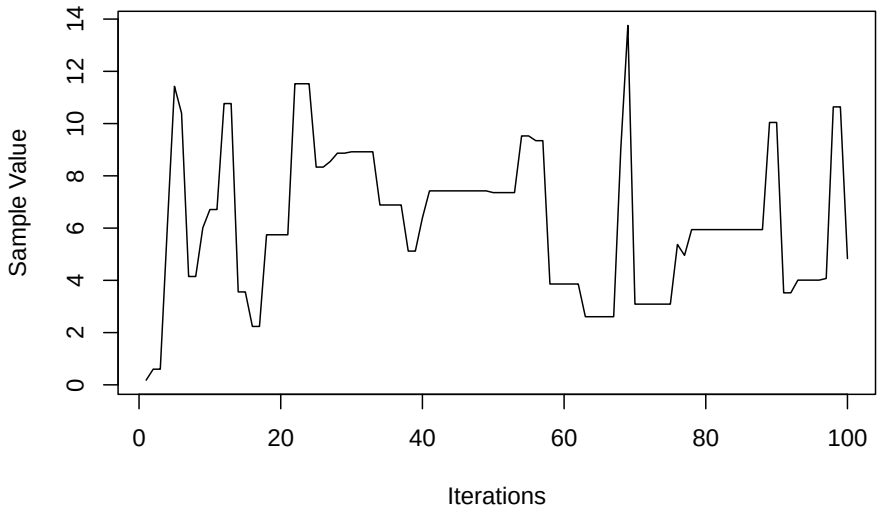
In here c is a constant. Used the Metropolis–Hastings algorithm to generate 10000 samples, Proposal distribution is "log–normal $LN(X_t, 1)$".In Metropolis–Hastings algorithm the acceptance ratio is given by (R):

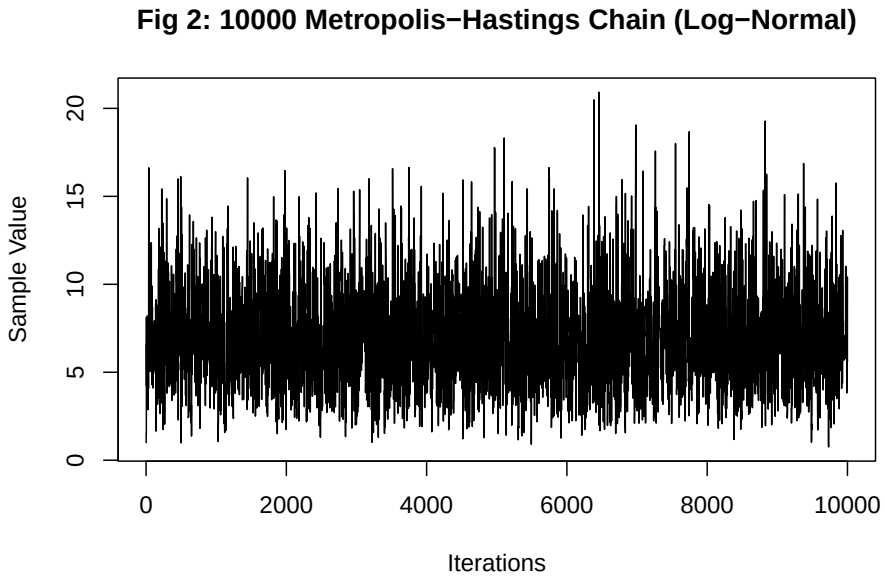
$$R = \min\left(1, \frac{f(Y)}{f(X)} \cdot \frac{g(X \mid Y)}{g(Y \mid X)}\right)$$

In given formula,
$f(\cdot)$ is the target distribution.
$q(\cdot \mid \cdot)$ is the proposal distribution.In my function log-normal distribution.(In r code *proposal_dist* function).
X is the current state.
Y is the proposed state.

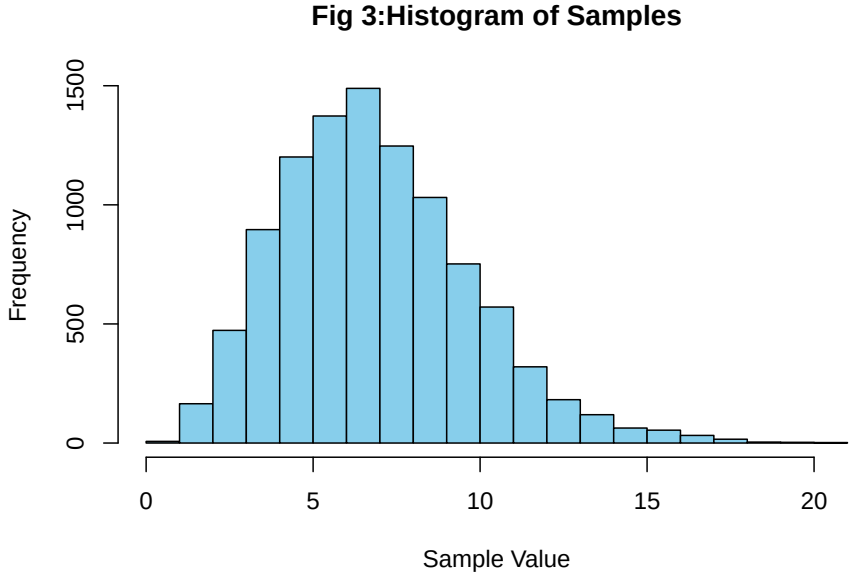
Proposal destribution is log-normal LN(X_t,1)

Fig 1: 100 Samples,Metropolis−Hastings Chain (Log−Normal)





The obtained samples values varied between 1 and 20.When considering following histogram(Fig 3), during this period sample converges to target distribution. Burn-in period is discard initial samples which highly depend on starting value. In here to determine the burn-in period, Examine the Fig 1 visually, the trace plot first 15 values from the chain are can discarded as a burn-in period.When consider Fig 2, difficult to identify a burn-in period.According to plots acceptance race can be taken from 15 to 10000.



Proposal distribution is chi–square $\chi^2(X_t + 1)$

Fig 4: 100 Metropolis−Hastings Chain

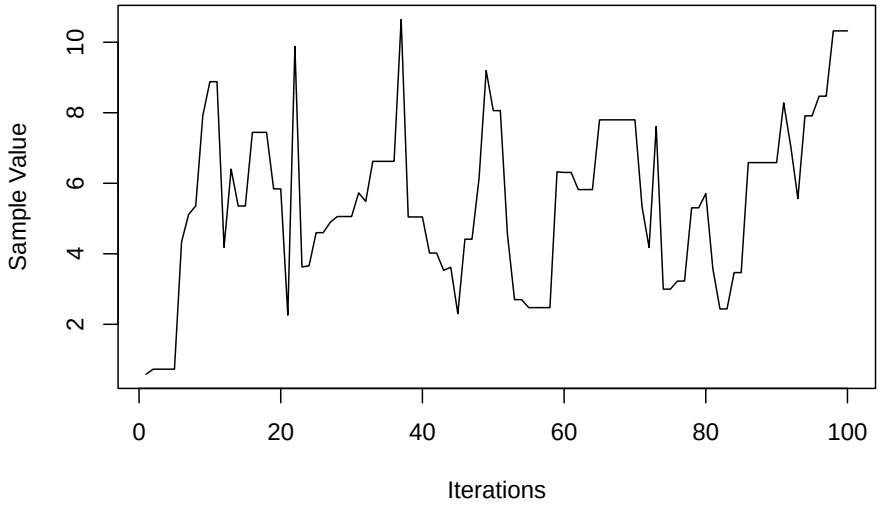


Fig 5: 10000 Metropolis−Hastings Chain

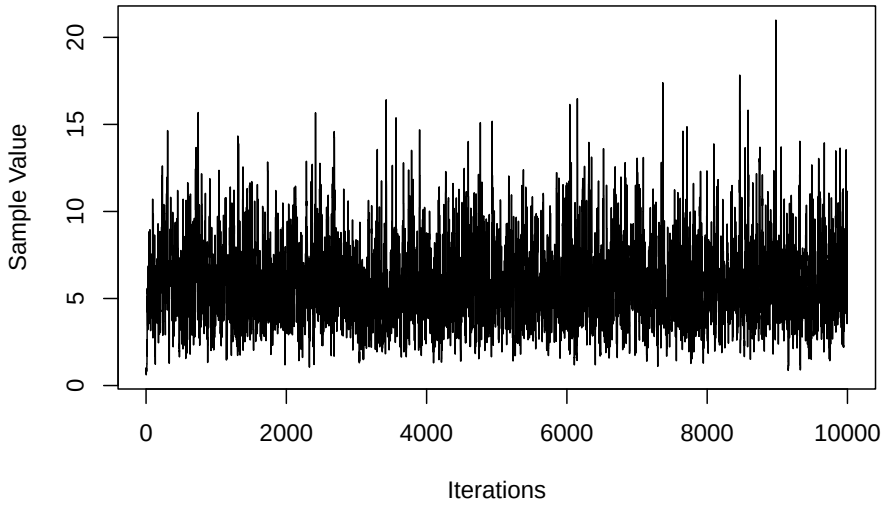
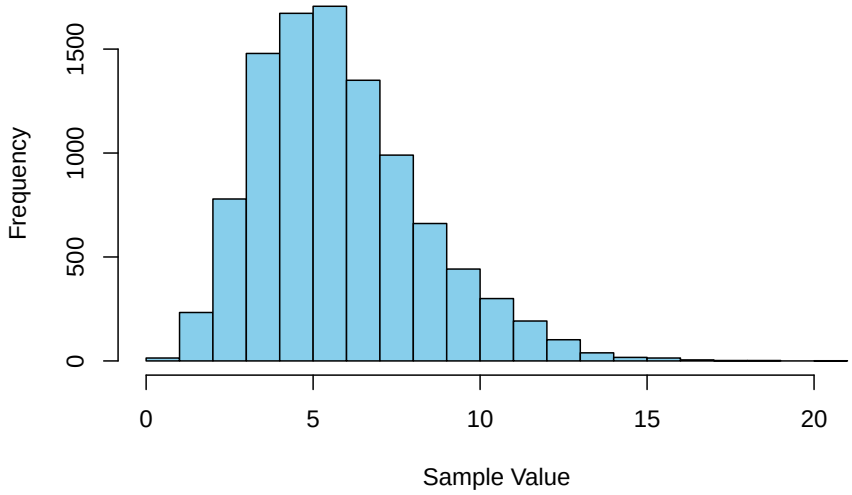


Fig 6:Histogram of Samples



c.

Suggestion distribution is log-normal distribution with sd value is 1.5.In here sample is converges to target distribution. It is clearly figure in following histogram.

Fig 2: 10000 Metropolis−Hastings Chain (Log−Normal)

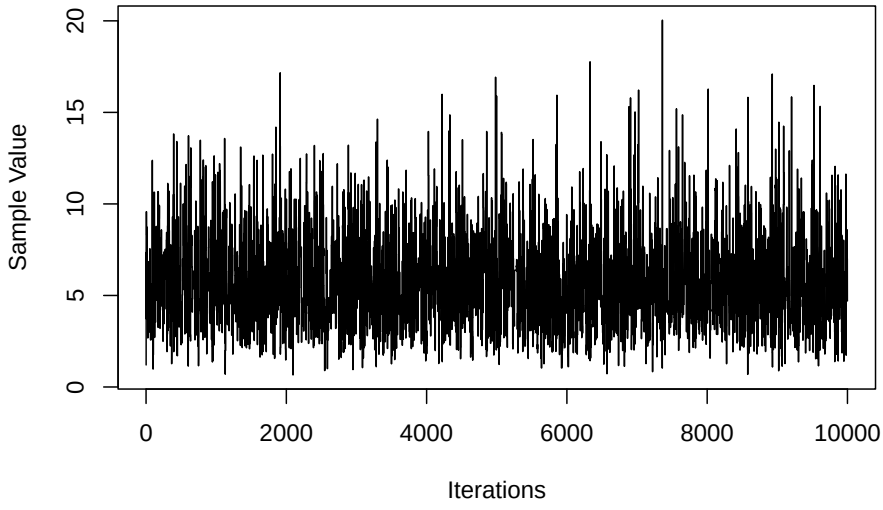
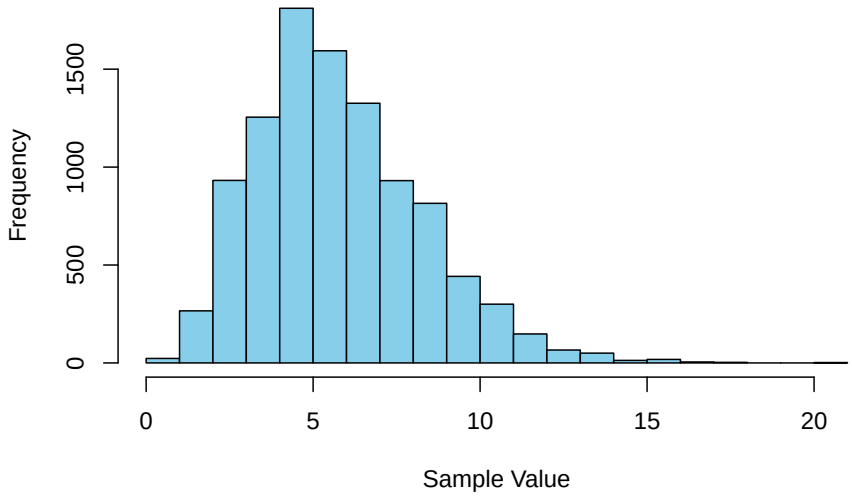


Fig 6:Histogram of Samples



d.

Compare the results of Parts a, b and c given figures,samples are approximately equal and convergence to proposed distribution.In all parts, Counted "acceptance rate" and take the percentage value of it.According to calculation following rates got for the acceptance , A. Acceptance rate = 45.55 B. Acceptance rate = 60.6 C. Acceptance rate = 34.42 Here we can get a conclusion using those values.According to reference "(GH) Givens-Hoeting - Computational Statistics", best choice being about 44% for one-dimensional problems. There for,Based on sample path plots,can conclude log-normal $LN(X_t, 1)$ as good proposal distribution. As well as we can consider about the variance of all samples.Following values are got for the variance in part a,b and c.

```
## Variance of sample generated by part a,  2.831131

## Variance of sample generated by part b,  2.460278

## Variance of sample generated by part c,  2.484158
```

e. Expected values

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

Using r mean function in R can compute the expected values of each samples.

```
## Expected value of sample generated by part a,  6.872181

## Expected value of sample generated by part b,  5.830452

## Expected value of sample generated by part c,  5.776559
```

Expected values for part a is high. Part b,and c mean values are approximately equal.From previous calculations variances also approximately equal.Here we can conclude each steps calculated samples are equally distributed.

f .

The distribution generated is in fact a gamma distribution.PDF of gamma distribution is,

$$f(x \mid k, \theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-\frac{x}{\theta}}$$

$$f(x) = cx^5 e^{-x}$$
$$k - 1 = 5, k = 6$$

$$\theta = 1$$

$$C = \frac{1}{\Gamma(k)\theta^k} = \frac{1}{\Gamma(6) * (1)^5} = \frac{1}{\Gamma(6) * (1)^5} = \frac{1}{(6-1)! * (1)^5} = \frac{1}{120}$$

By using $c * \int_0^\infty f(x)dx = 1$ we can compute actual value of the integral as 120 because $C = \frac{1}{\int_0^\infty f(x)dx}$. Using the following function integrate the $f(x)$ to find actual value of the integral using following function.

```
# Find the normalization constant c
integral <- (integrate(f, lower = 0, upper = Inf)$value)
cat("C =",integral)
```

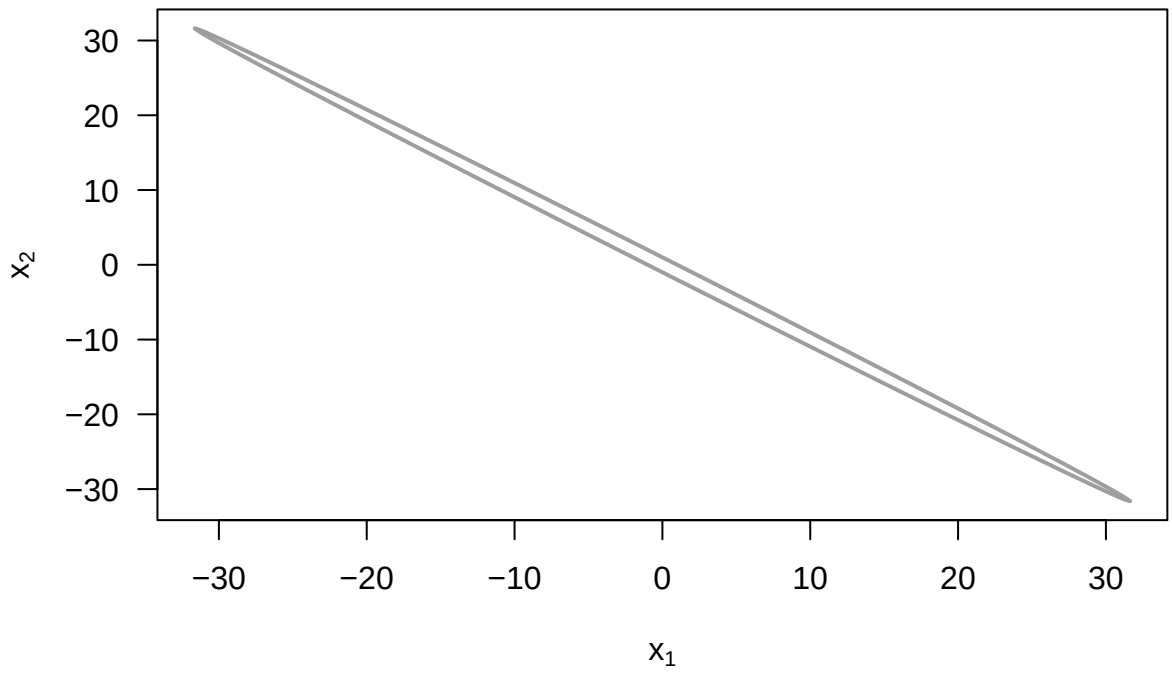
```
## C = 120
```

This function given the 120 for integral.

Question 2

a.

```
w  <- 1.999
xv <- seq(-1, 1, by=0.01) * 1/sqrt(1-w^2/4)  # a range of x1-values, where the term below the root is n
plot(xv, xv, type="n", xlab=expression(x[1]), ylab=expression(x[2]), las=1)
# 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)
```



b. Density function,

$$f(x_1, x_2) \propto 1\{x_1^2 + 1.999 x_1 x_2 + x_2^2 < 1\}$$

Conditional Distribution of x_1 given x_2 :

The conditional distribution of x_1 given x_2 is determined by the inverse relationship. Since the original distribution has a symmetric form, the conditional distribution of x_1 given x_2 given should be a uniform distribution on the interval $(-0.9995 * x_2 - \sqrt{1 - 0.001 x_2^2}),(-0.9995 * x_2 + \sqrt{1 - 0.001 x_2^2})$

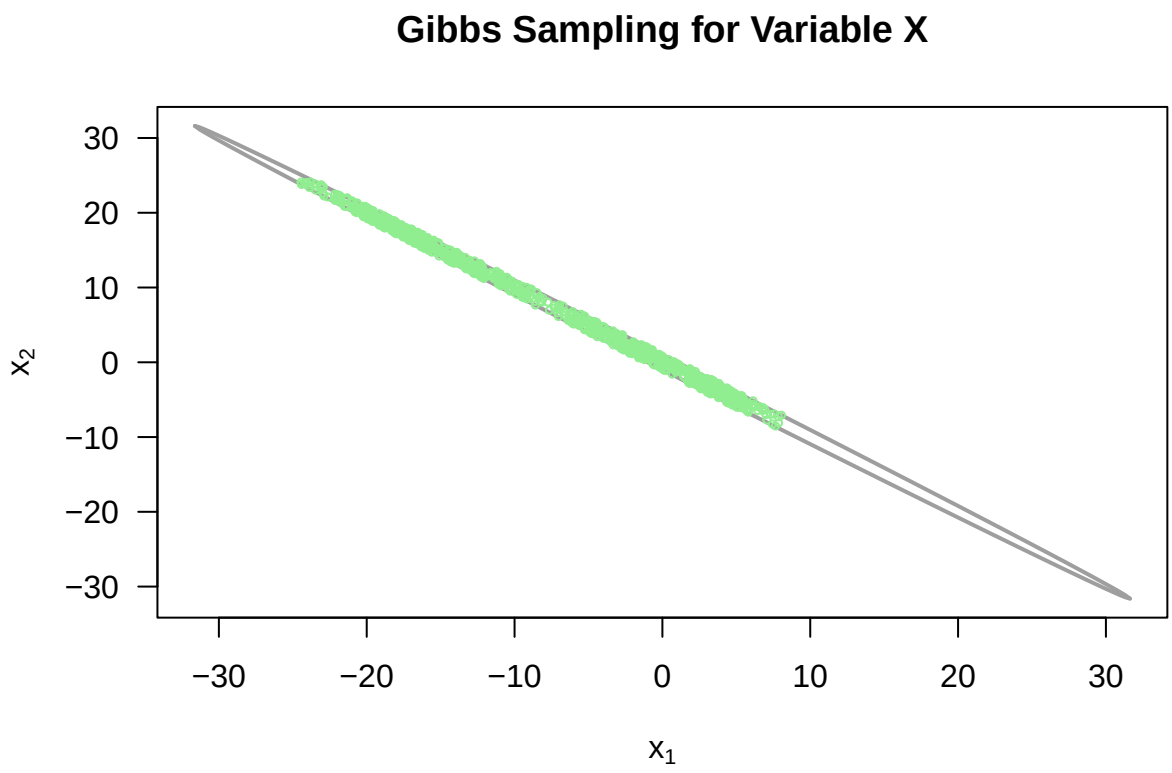
Conditional Distribution of x_2 given x_1 :

$(-0.9995 * x_1 - \sqrt{1 - 0.001 x_1^2}),(-0.9995 * x_1 + \sqrt{1 - 0.001 x_1^2})$

c.

```
# Gibbs Sampling
set.seed(123)  # for reproducibility
#rm(.Random.seed, envir=globalenv())
n <- 1000
generate_gibbs_sampling <- function(n){
  # Initialize vectors to store samples
  x1_samples <- numeric(n)
  x2_samples <- numeric(n)

  # Initial values
```

```
  x1 <- 0
  x2 <- 0

  # Gibbs Sampling
  for (i in 1:n) {
    # Sample from the conditional distribution of x2 given x1
    x2 <- runif(1, -0.9995*x1-sqrt(1-(0.001*x1^2 )),-0.9995*x1+sqrt(1 -(0.001*x1^2)))
    x2_samples[i] <- x2

    # Sample from the conditional distribution of x1 given x2
    x1 <- runif(1, -0.9995*x2-sqrt(1-(0.001*x2^2)), -0.9995*x2+sqrt(1 -(0.001*x2^2)))
    x1_samples[i] <- x1
  }
  return (list("x1"=x1_samples,"x2"=x2_samples))
}
samples <- generate_gibbs_sampling(n)

# Plot the ellipse
plot(xv, xv, type = "n", xlab = expression(x[1]), ylab = expression(x[2]), las = 1,main = "Gibbs Sampli
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 Gibbs samples
points(samples$x1, samples$x2, pch = 1, col = "lightgreen", cex = 0.5)
```



```r
# Determine P(X1 > 0) based on the sample
prob_X1_positive <- sum(samples$x1 > 0) / n
print(paste("P(X1 > 0):", prob_X1_positive))
```

```
## [1] "P(X1 > 0): 0.24"
```

```r
for(i in 1:10){
  samples <- generate_gibbs_sampling(n)
  prob_X1_positive <- sum(samples$x1 > 0) / n
  print(paste("P(X1 > 0):", prob_X1_positive))
}
```

```
## [1] "P(X1 > 0): 0.004"
## [1] "P(X1 > 0): 0.942"
## [1] "P(X1 > 0): 0.686"
## [1] "P(X1 > 0): 0.576"
## [1] "P(X1 > 0): 0.241"
## [1] "P(X1 > 0): 0.756"
## [1] "P(X1 > 0): 0.184"
## [1] "P(X1 > 0): 0.266"
## [1] "P(X1 > 0): 0.21"
## [1] "P(X1 > 0): 0.025"
```

True result for this probability ,$P(X1 > 0) = 0.24$

d.

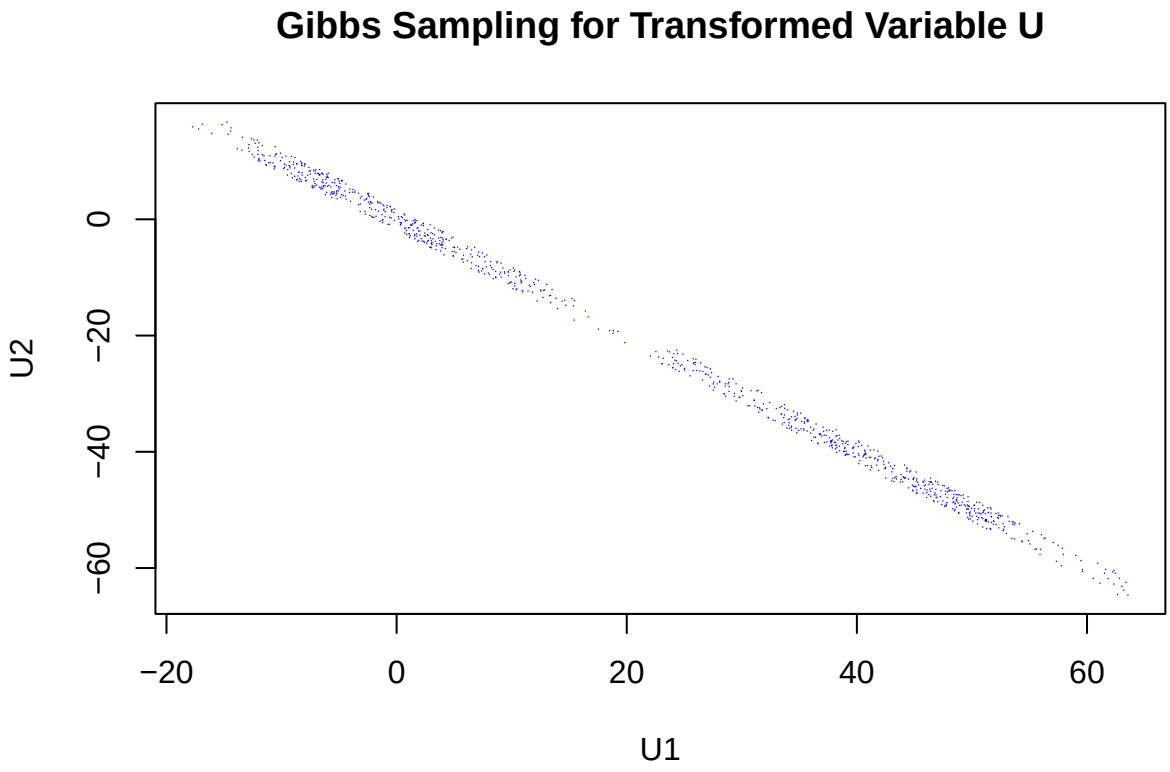
Convergence Issues: Gibbs sampling relies on the Markov chain to converge to the true distribution. If the distribution has complex geometry, and high correlations, or if the chain gets stuck in certain regions, convergence may be slower or less effective.

Tuning Parameters: The choice of tuning parameters, such as step sizes in the sampling process, might need adjustment for different parameter values. Values that worked well for w=1.8 might not be optimal for w=1.999.

e.

```r
# Gibbs Sampling for Transformed Variable U = (U1, U2)
set.seed(123)  # for reproducibility
n <- 1000


  # Initialize vectors to store samples
  u1_samples <- numeric(n)
  u2_samples <- numeric(n)

  # Initial values
  u1 <- 0
  u2 <- 0
```

```r
  # Gibbs Sampling for U
  for (i in 1:n) {
    # Sample from the conditional distribution of u1 given u2
    u1 <- runif(1, -2 - u2, 2 - u2)
    u1_samples[i] <- u1

    # Sample from the conditional distribution of u2 given u1
    u2 <- runif(1, -2 - u1, 2 - u1)
    u2_samples[i] <- u2
  }

# Plot the transformed region
plot(u1_samples, u2_samples, pch = 16, col = "blue", cex = 0.1, main = "Gibbs Sampling for Transformed V
     xlab = "U1", ylab = "U2")
```




```r
# Determine P(X1 > 0) = P((U2 + U1)/2 > 0)
prob_X1_positive = sum((u1_samples + u2_samples) / 2 > 0) / n
print(paste("P(X1 > 0):", prob_X1_positive))
```

```
## [1] "P(X1 > 0): 0.485"
```

Both codes implement Gibbs sampling for their respective variables (X and U).

The bounds for sampling in the U code are adjusted based on the transformation and the properties of the conditional distributions.

The shape of the transformed space is influenced by the conditional distributions and the specific form of the transformation equations.

probability of “c” - “P(X1 > 0): 0.24”.probability of “d” - “P(X1 > 0): 0.485”.

Appendix: All r code for this report

```
knitr::opts_chunk$set(echo = TRUE)
###density function ####
f = function(x){
  return(x^5*exp(-x))
}

##metropolis_hastings algorithm####
metropolis_hastings = function(x,n_iteration,prop_distribution,proposed_density){
  chain  <- c()
  accepted_proposals <- 0
  # generate first five values (change 5 to higher number for longer chain)
  for (i in 1:n_iteration)
  {
    u <- prop_distribution(x)
    R     <- min(1, f(u) * proposed_density(x)/(f(x) * proposed_density(u))) # MH ratio

    ap    <- runif(1)
    if (ap<R){
      x <- u
      accepted_proposals <- accepted_proposals + 1
    }

    chain <- rbind(chain, x)
  }
  acceptance_rate <- (accepted_proposals / n_iteration)*100
  #cat("Acceptant rate = ",acceptance_rate,"\n")
   chain  # each row is one two-dimensional observation
}

# Proposal distribution(log-normal) get random sample
proposal_dist = function(x, sigma = 1) {
  return(rlnorm(1, meanlog=log(x), sdlog = sigma))
}

# Proposal density function(Density log-normal)
g = function(x, sigma = 1) {
  return(dlnorm(1, meanlog=log(x), sdlog = sigma))
}

set.seed(12345)
#100 samples
samples1 <- metropolis_hastings(0.1,100,proposal_dist,g)

plot(samples1, type = "l", main = "Fig 1: 100 Samples,Metropolis-Hastings Chain (Log-Normal)", xlab = "
  ##10000 samples
```

```r
  samples <- metropolis_hastings(1,10000,proposal_dist,g)
  plot(samples, type = "l", main = "Fig 2: 10000 Metropolis-Hastings Chain (Log-Normal)", xlab = "Itera
 # mean(samples)
###Histogram sample function ####
 hist(samples, main = "Fig 3:Histogram of Samples", xlab = "Sample Value", col = "skyblue", border = "bl
# Proposal distribution(chi-squired) get random sample
proposal_dist1 <- function(x) {
  df <- floor(x)  # floor function
  return(rchisq(1, df+1))
}

# Proposal density function(Density chi-squired)
g1 = function(x, sigma = 1) {
  df <- floor(x)  # floor function
  return(dchisq(x, df+1))
}

set.seed(12345)
#100 samples
samples_chi <- metropolis_hastings(0.1,100,proposal_dist1,g1)

plot(samples_chi, type = "l", main = "Fig 4: 100 Metropolis-Hastings Chain", xlab = "Iterations", ylab =
##10000 samples
  samples_chi2 <- metropolis_hastings(1,10000,proposal_dist1,g1)
  plot(samples_chi2, type = "l", main = "Fig 5: 10000 Metropolis-Hastings Chain", xlab = "Iterations", y
 # mean(samples)


###Histogram sample function ####
 hist(samples_chi2, main = "Fig 6:Histogram of Samples", xlab = "Sample Value", col = "skyblue", border
# Proposal distribution(log-normal) get random sample
proposal_dist = function(x, sigma = 1.5) {
  return(rlnorm(1, meanlog=log(x), sdlog = sigma))
}

# Proposal density function(Density log-normal)
g = function(x, sigma = 1.5) {
  return(dlnorm(1, meanlog=log(x), sdlog = sigma))
}

set.seed(12345)

##10000 samples
  samples3 <- metropolis_hastings(0.5,10000,proposal_dist,g)
  plot(samples3, type = "l", main = "Fig 2: 10000 Metropolis-Hastings Chain (Log-Normal)", xlab = "Itera
 # mean(samples)
  ###Histrogram sample function ####
 hist(samples3, main = "Fig 6:Histogram of Samples", xlab = "Sample Value", col = "skyblue", border = "
sd1=sd(samples)
cat("Variance of sample generated by part a, ",sd1,"\n")
sd2=sd(samples_chi2)
cat("Variance of sample generated by part b, ",sd2,"\n")
sd3=sd(samples3)
```

```r
cat("Variance of sample generated by part c, ",sd3,"\n")
mean1=mean(samples)
cat("Expected value of sample generated by part a, ",mean1,"\n")
mean2=mean(samples_chi2)
cat("Expected value of sample generated by part b, ",mean2,"\n")
mean3=mean(samples3)
cat("Expected value of sample generated by part c, ",mean3,"\n")

# Find the normalization constant c
integral <- (integrate(f, lower = 0, upper = Inf)$value)
cat("C =",integral)

w  <- 1.999
xv <- seq(-1, 1, by=0.01) * 1/sqrt(1-w^2/4)  # a range of x1-values, where the term below the root is n
plot(xv, xv, type="n", xlab=expression(x[1]), ylab=expression(x[2]), las=1)
# 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)
# Gibbs Sampling
set.seed(123)  # for reproducibility
#rm(.Random.seed, envir=globalenv())
n <- 1000
generate_gibbs_sampling <- function(n){
  # Initialize vectors to store samples
  x1_samples <- numeric(n)
  x2_samples <- numeric(n)

  # Initial values
  x1 <- 0
  x2 <- 0

  # Gibbs Sampling
  for (i in 1:n) {
    # Sample from the conditional distribution of x2 given x1
    x2 <- runif(1, -0.9995*x1-sqrt(1-(0.001*x1^2 )),-0.9995*x1+sqrt(1 -(0.001*x1^2)))
    x2_samples[i] <- x2

    # Sample from the conditional distribution of x1 given x2
    x1 <- runif(1, -0.9995*x2-sqrt(1-(0.001*x2^2)), -0.9995*x2+sqrt(1 -(0.001*x2^2)))
    x1_samples[i] <- x1
  }
  return (list("x1"=x1_samples,"x2"=x2_samples))
}
samples <- generate_gibbs_sampling(n)

# Plot the ellipse
plot(xv, xv, type = "n", xlab = expression(x[1]), ylab = expression(x[2]), las = 1,main = "Gibbs Samplin
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 Gibbs samples
points(samples$x1, samples$x2, pch = 1, col = "lightgreen", cex = 0.5)
```

```r
# Determine P(X1 > 0) based on the sample
prob_X1_positive <- sum(samples$x1 > 0) / n
print(paste("P(X1 > 0):", prob_X1_positive))
for(i in 1:10){
  samples <- generate_gibbs_sampling(n)
  prob_X1_positive <- sum(samples$x1 > 0) / n
  print(paste("P(X1 > 0):", prob_X1_positive))
}

# Gibbs Sampling for Transformed Variable U = (U1, U2)
set.seed(123)  # for reproducibility
n <- 1000


  # Initialize vectors to store samples
  u1_samples <- numeric(n)
  u2_samples <- numeric(n)

  # Initial values
  u1 <- 0
  u2 <- 0

  # Gibbs Sampling for U
  for (i in 1:n) {
    # Sample from the conditional distribution of u1 given u2
    u1 <- runif(1, -2 - u2, 2 - u2)
    u1_samples[i] <- u1

    # Sample from the conditional distribution of u2 given u1
    u2 <- runif(1, -2 - u1, 2 - u1)
    u2_samples[i] <- u2
  }

# Plot the transformed region
plot(u1_samples, u2_samples, pch = 16, col = "blue", cex = 0.1, main = "Gibbs Sampling for Transformed
     xlab = "U1", ylab = "U2")

# Determine P(X1 > 0) = P((U2 + U1)/2 > 0)
prob_X1_positive = sum((u1_samples + u2_samples) / 2 > 0) / n
print(paste("P(X1 > 0):", prob_X1_positive))
```