CompStatLab4

Xuan Wang & Priyarani Patil

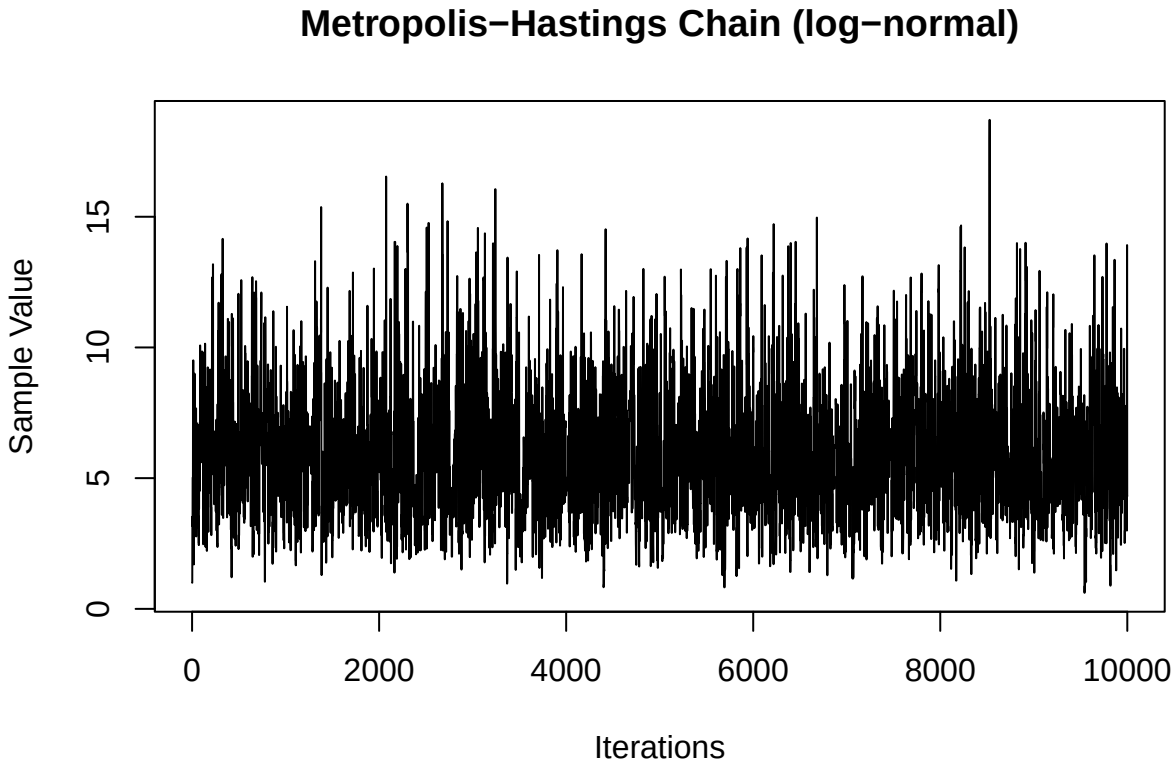
2023-11-24

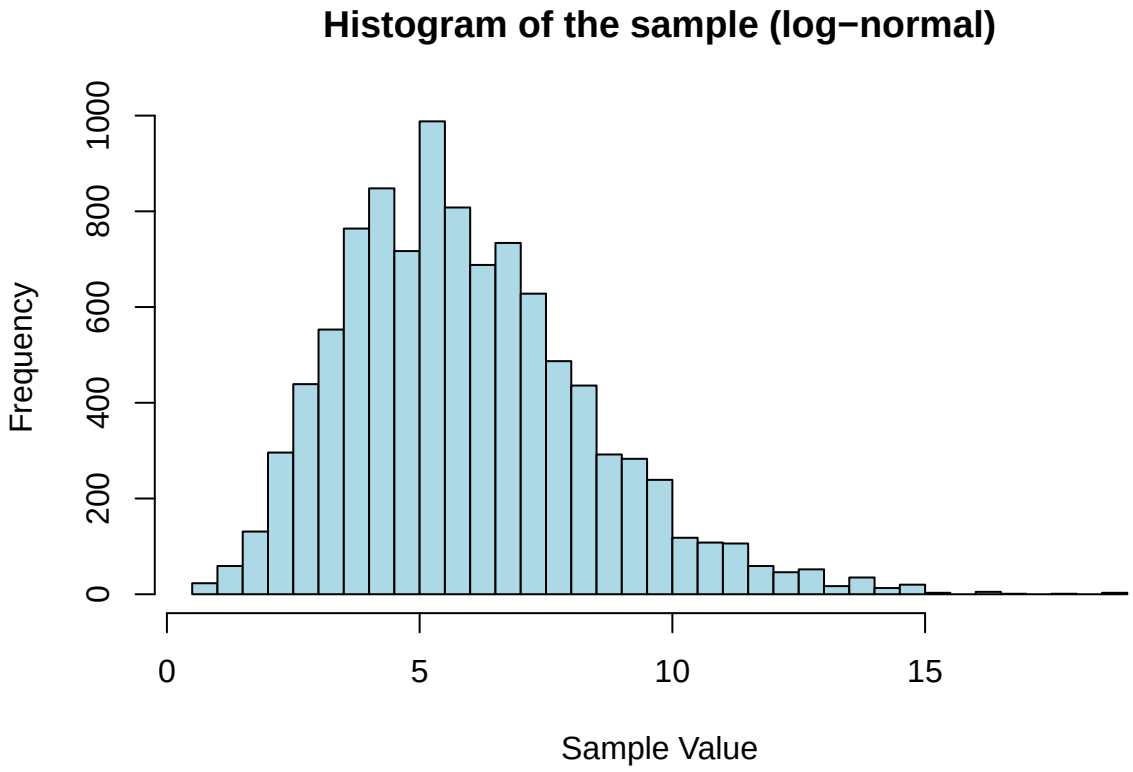
Computational Statistics 732A90 | Computer Lab 4

Question 1: Computations with Metropolis–Hastings

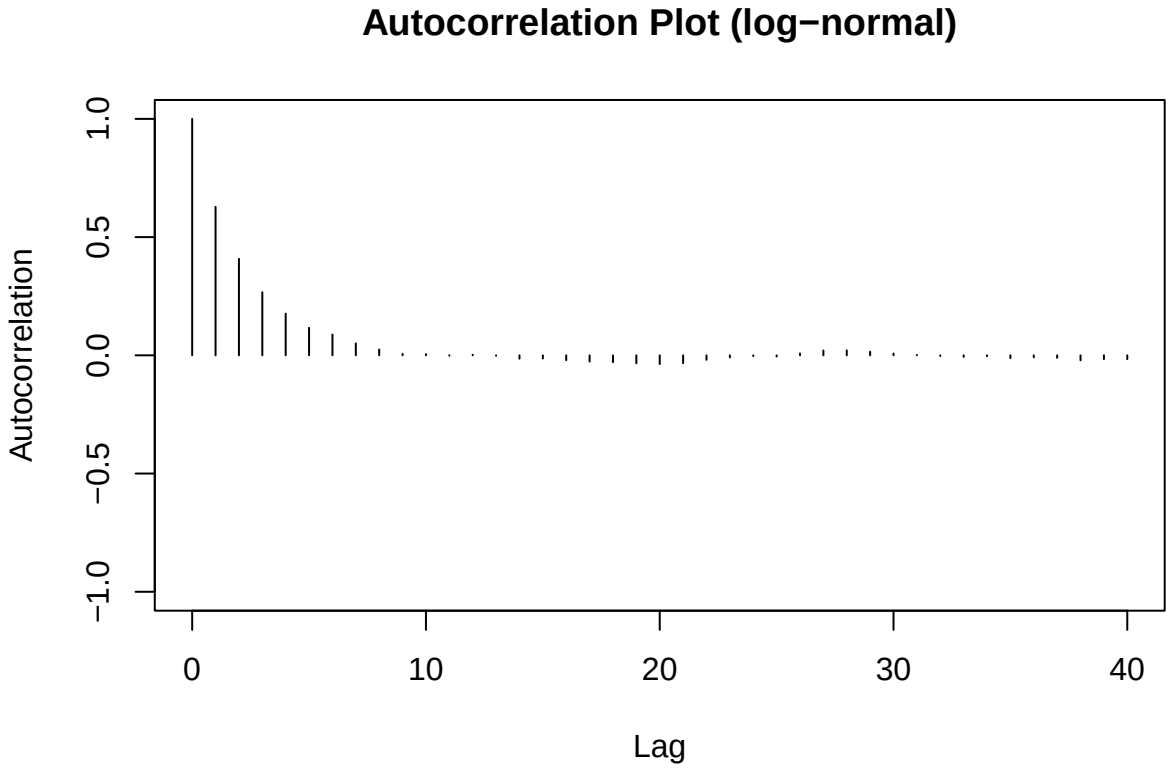
a

```
## When proposal function is log-normal function, the acceptance rate is : 0.4438
```






```
## The first 20 chain values as below:
##  [1] 1.000000 1.000000 3.287760 3.527943 3.163722 5.016163 5.016163 5.016163
##  [9] 5.016163 7.489011 9.503400 9.503400 9.503400 9.503400 4.563758 3.669926
## [17] 3.669926 1.964293 1.698488 1.980030
```



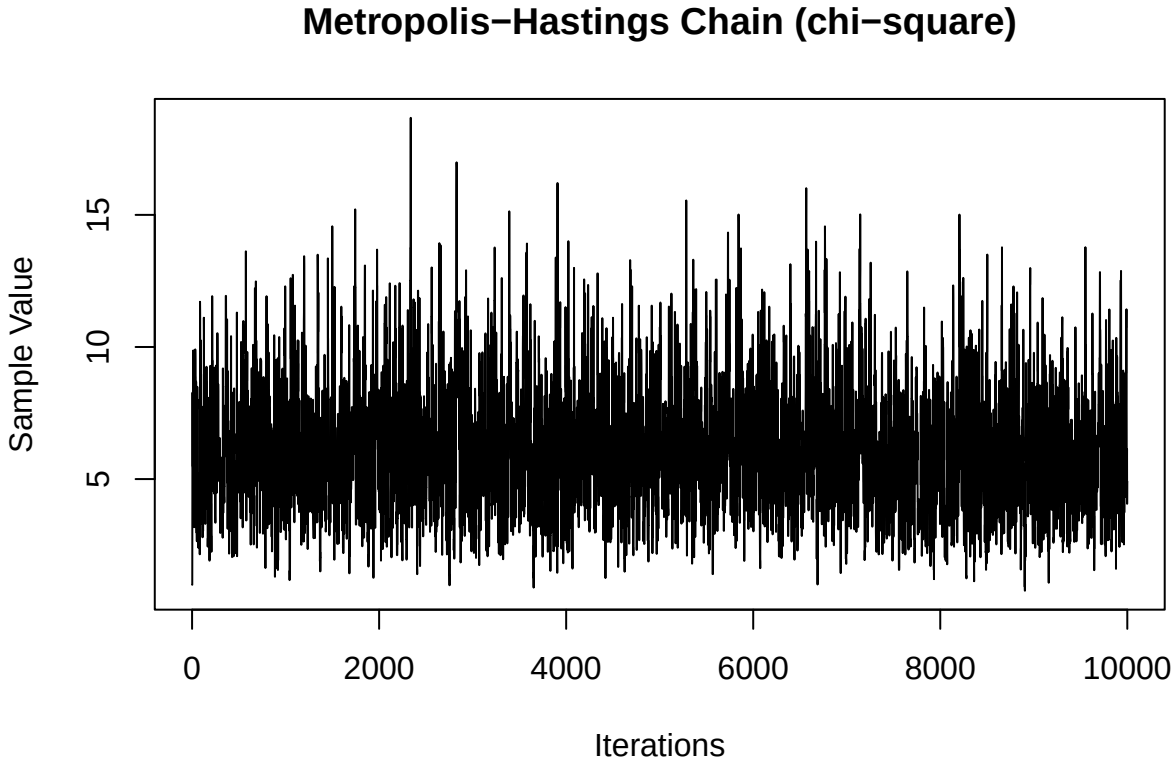
From above plots and histogram, it can seen that when take log-normal as proposal distribution, the chain trace plot shows a stable "hairy caterpillar" pattern in the later iterations and autocorrelation plot shows a decrease and drop to nearly 0 with the increase of lag. Both indicate that the convergence well.

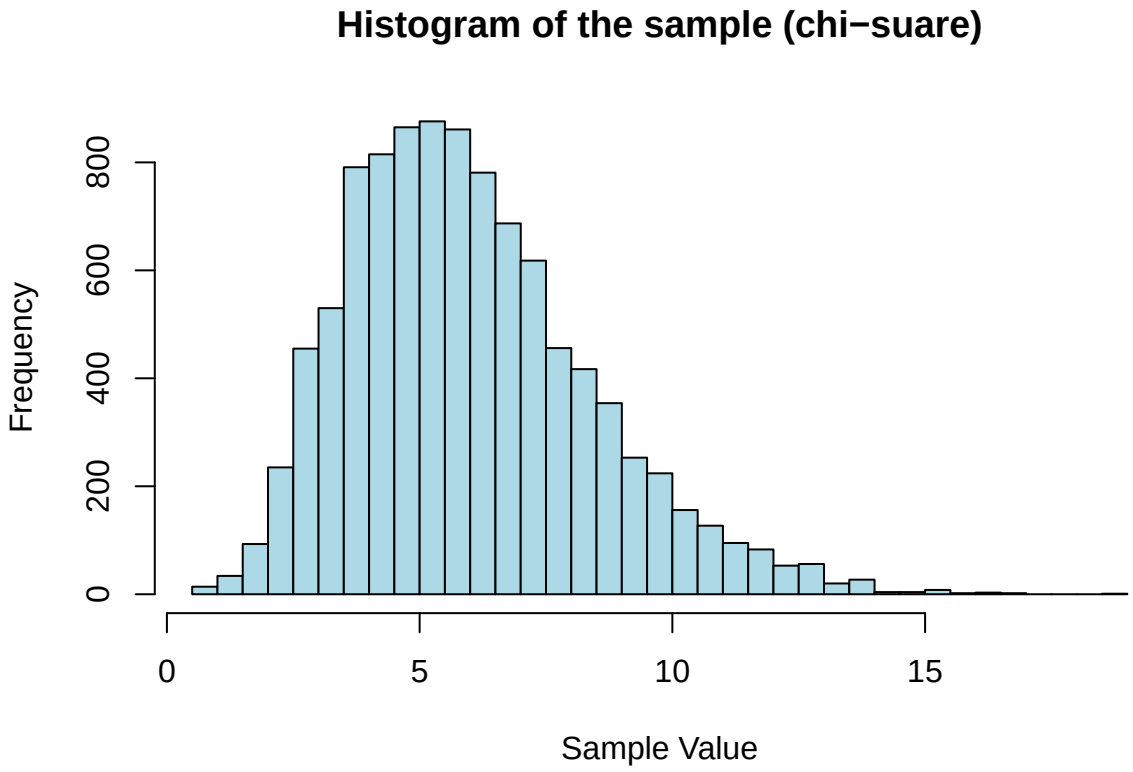
When proposal function is log-normal function, the acceptance rate is : 0.4438

According to the first 20 chain values, it can see there is a very small burn-in period, after the 11 iteration, the chain reach a stationary distribution.

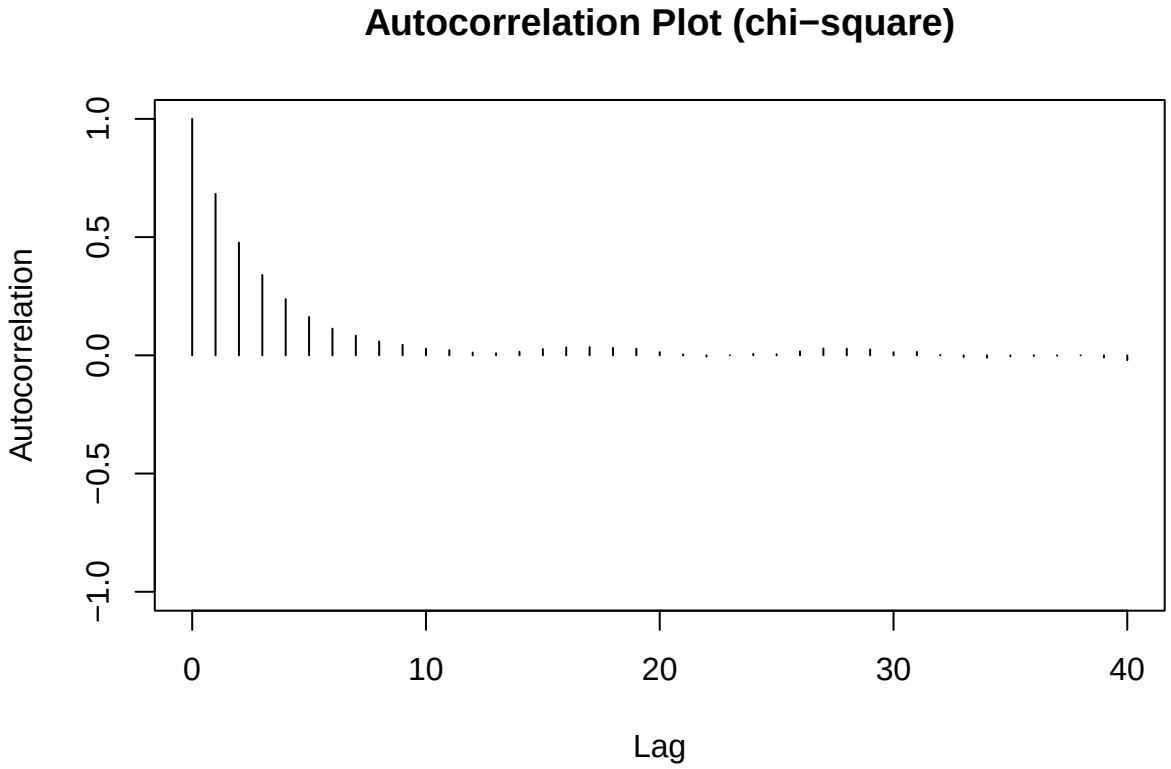
b

```
## WHen the proposal function is chi-squre function, the acceptance rate is : 0.6062
```






```
## The first 20 chain values as below:
##  [1] 1.000000 1.000000 4.419132 8.273999 8.273999 8.273999 5.488472 5.488472
##  [9] 5.488472 5.356135 5.356135 7.951344 9.870184 6.157023 4.895911 3.762035
## [17] 3.762035 3.762035 3.762035 3.762035
```



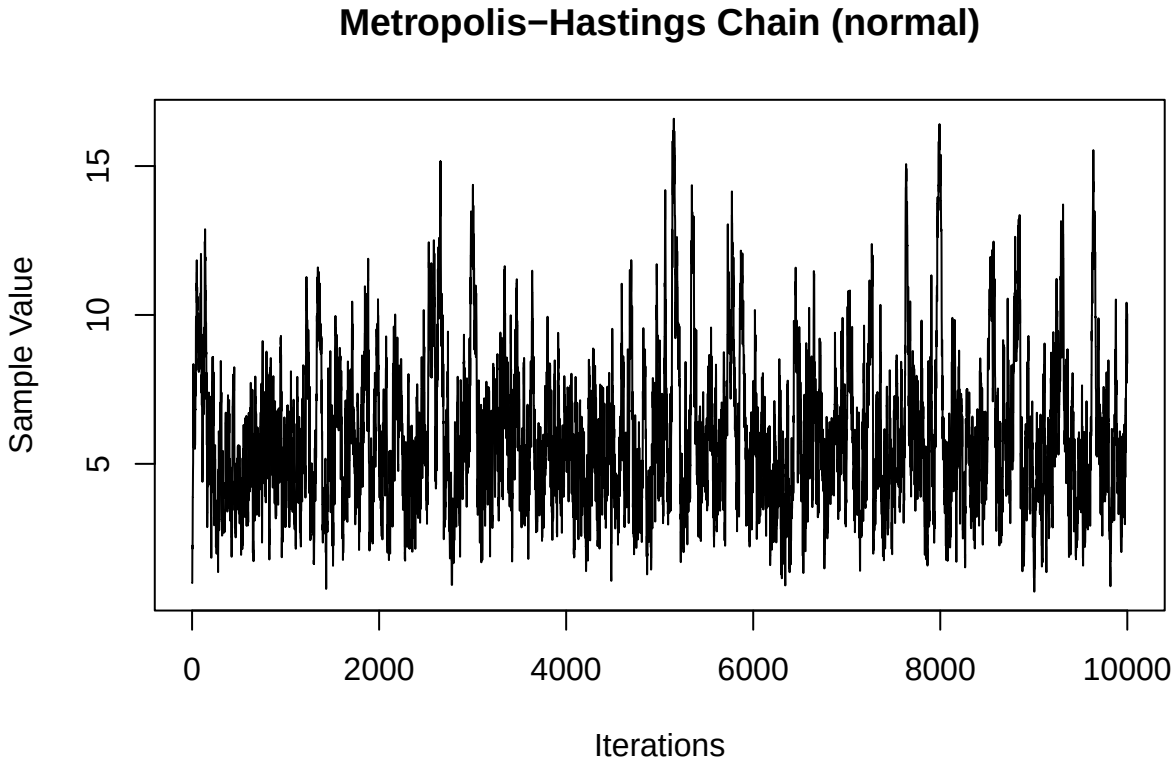
From above plots and histogram, it can seen that when take chi-square as proposal distribution, the chain trace plot also shows a stable "hairy caterpillar" pattern in the later iterations and autocorrelation plot shows a decrease and drop to nearly 0 with the increase of lag. Both indicate that the convergence well. The histogram is also similar to that when proposal distribution is log-normal.

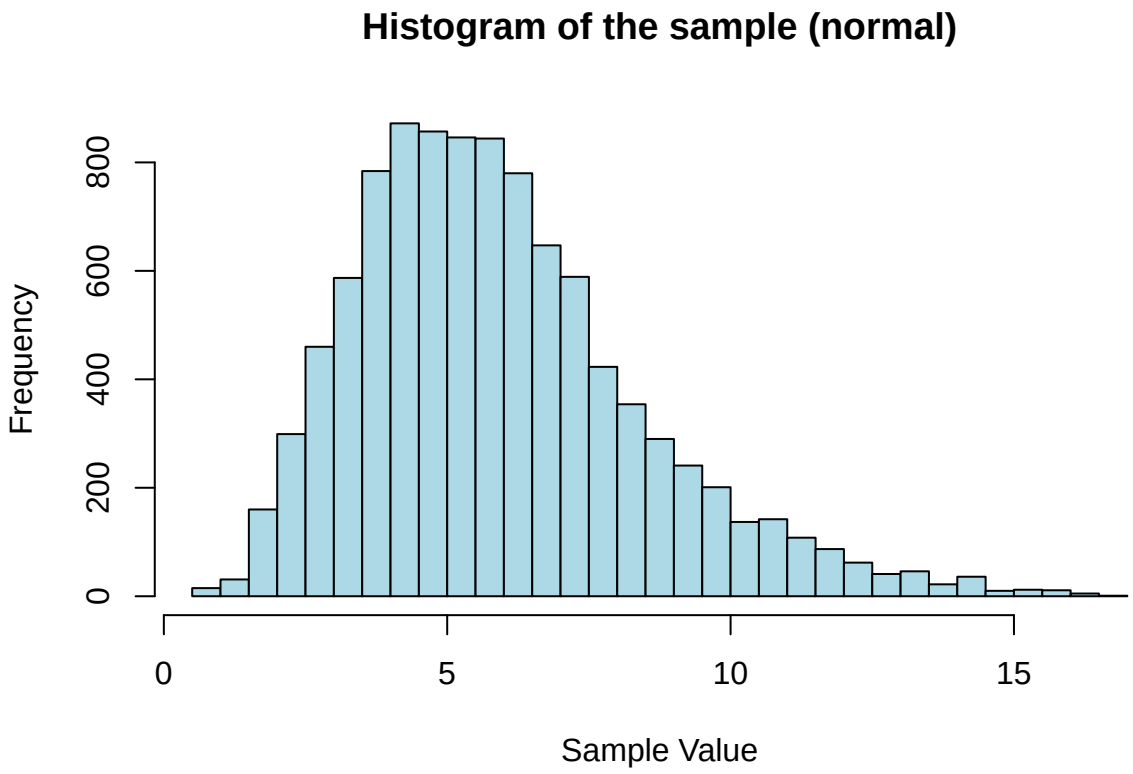
When proposal function is chi-square function, the acceptance rate is : 0.6062

According to the first 20 chain values, it can see there is also a very small burn-in period, after 4 iteration, the chain reach a stationary distribution.

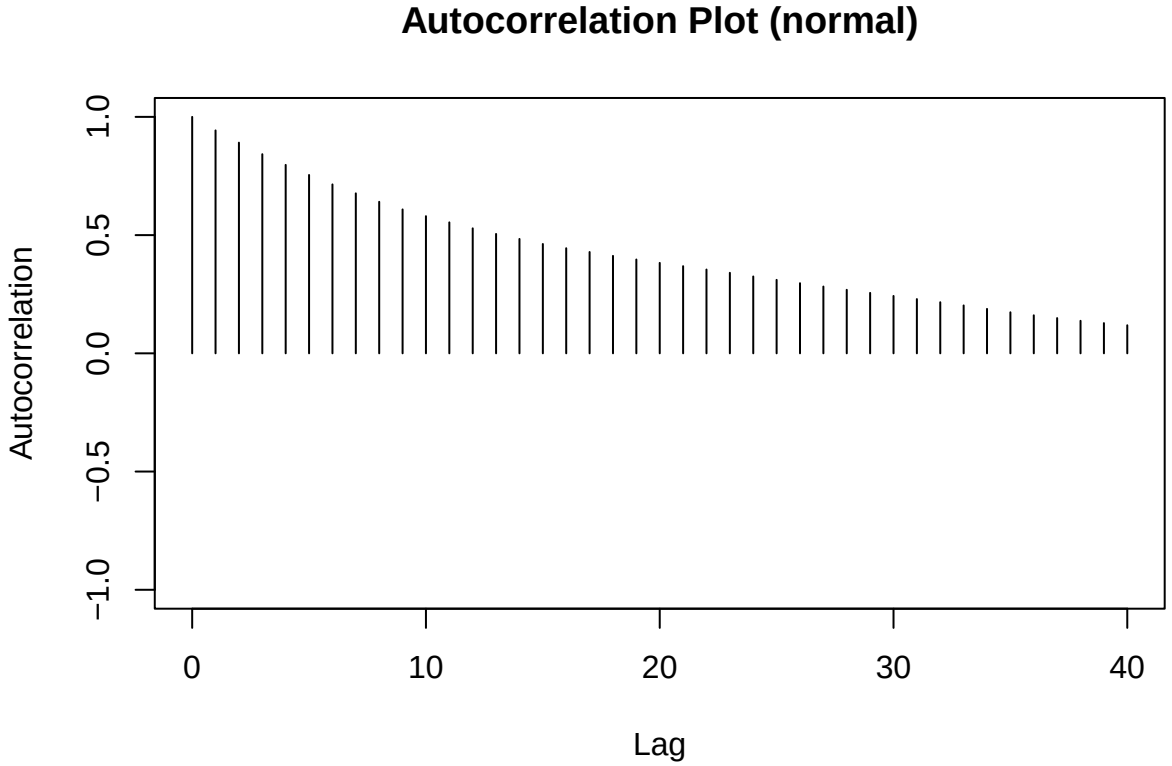
c

```
## When the proposal function is normal function, the acceptance rate is : 0.857
```






```
## The first 20 chain values as below:
##  [1] 1.000000 1.000000 2.190207 2.260715 2.151749 2.612665 3.893220 3.893220
##  [9] 4.397033 4.797804 5.036017 6.822930 7.648470 8.349826 7.616323 7.398348
## [17] 6.312649 5.687609 5.542216 5.695589
```



From above plots and histogram, it can be seen that when take normal as proposal distribution, the chain trace plot also shows a stable "hairy caterpillar" pattern in the later iterations and autocorrelation plot shows a decrease, but it didn't drop to 0 with the increase of lag.It indicates that samples in the chain are still influenced by previous samples, and the chain might not have explored the parameter space sufficiently.

When proposal function is normal function, the acceptance rate is : 0.857

According to the first 20 chain values, it can see there is also a very small burn-in period, after the 15 iteration, the chain reach a stationary distribution.

d

```
##   proposal_function acceptance_rate     mean
## 1        log-normal          0.4438 5.924427
## 2        chi-square          0.6062 5.976130
## 3            normal          0.8570 5.910895
```

From above comparison among three different proposal functions, it can be seen that when proposal function is normal function, its acceptance rate is highest. It means that normal function as proposal function could compute more efficient. However it shows high autocorrelation in later lags, it indicates that there is still some level correlation between the values in the Markov chain at those lags. While Chi-square's acceptance is higher than log-normal, and the expectation value computed by chi-square function is most close to 6(real value). Therefore, it seems chi-square would likely to be a better choice among three options.

e

Please see below R compute results.

```
##   proposal_function     mean
## 1        log-normal 5.924427
## 2        chi-square 5.976130
## 3            normal 5.910895
```

f

The expected value for a gamma distribution is $E(X) = k * \theta$, where k is the shape parameter and θ is the scale parameter.

In this case, the shape parameter k is 6 (since the power of x is 5+1) and the scale parameter θ is 1 (since the rate parameter, which is the reciprocal of the scale parameter, is 1). Therefore, the expected value $E(X) = 6 * 1 = 6$.

This actual integral value is quite close to the mean values when using three different proposal distributions.

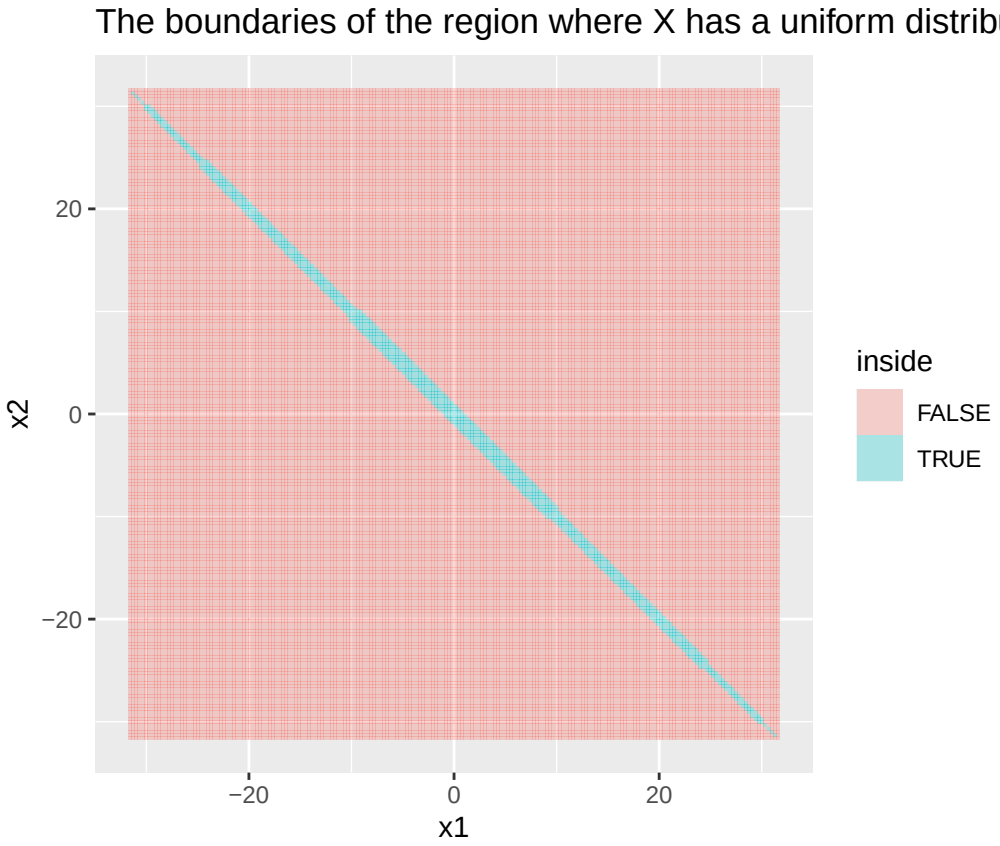
See Appendix.

```
## The actual integral value of gamma distribution is:  6
```

Question 2: Gibbs sampling

a

See Appendix R code and below plots:



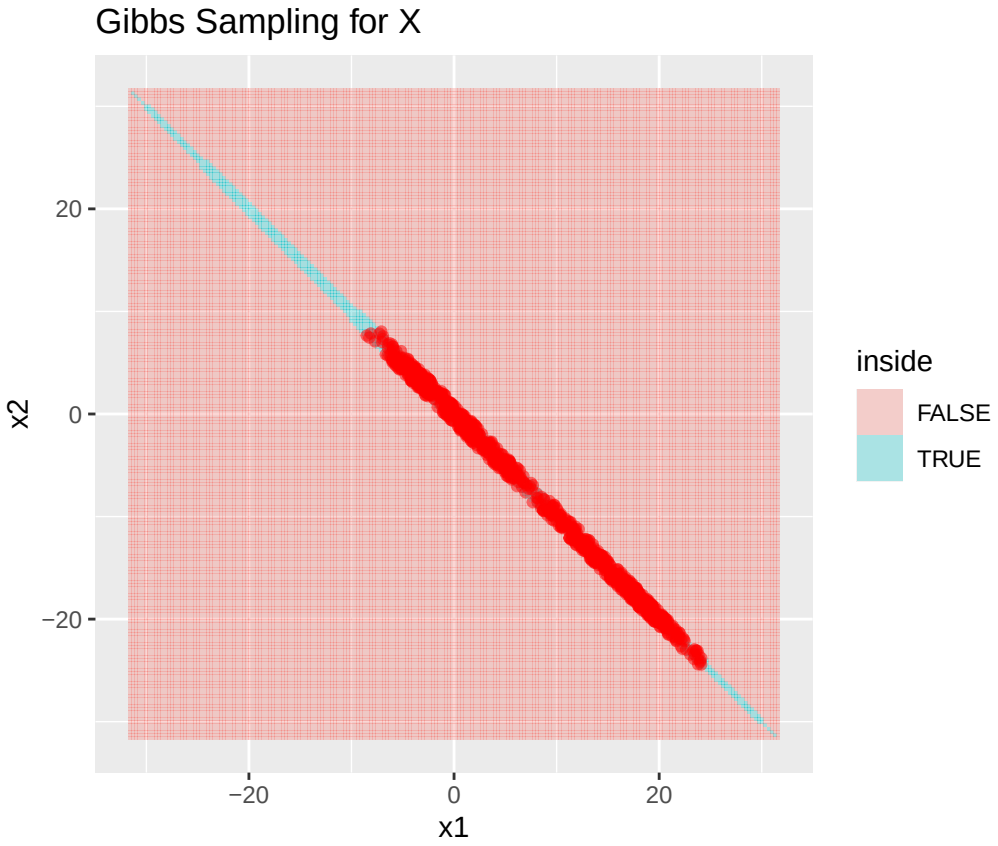
b

The conditional distribution of X1 given X2 is a uniform distribution on the interval $(-(w/2)*x2-sqrt(1-(1-w^2/4)*x2^2), -(w/2)*x2+sqrt(1-(1-w^2/4)*x2^2))$

this is same as that of X2 give X1, whose the interval is also: $(-(w/2)*x1-sqrt(1-(1-w^2/4)*x1^2), -(w/2)*x1+sqrt(1-(1-w^2/4)*x1^2))$

c

See Appendix R code.




```
## P(X1 > 0): 0.755
```

The calculated probability P(X1 $>$ 0) = 0.755 suggests that in the generated sample, approximately 75.5% of the X1 values are greater than 0. The true result for this probability depends on the exact shape of the distribution. However, given that X1 and X2 are from a bivariate uniform distribution with a specific boundary condition, and considering the symmetry of the distribution, the true probability P(X1 $>$ 0) should theoretically be 0.5.

The discrepancy between estimated (0.755) and the theoretical true probability (0.5) could be due to random variation in the Gibbs sampling process, or it could indicate that the Gibbs sampler has not yet fully converged to the target distribution. It's also possible that the specific boundary condition (w = 1.999) is causing X1 to be greater than 0 more often than it is less than 0.

d

The success of Gibbs sampling depends on the conditional distributions. For w = 1.999, it approaches 2, the shape of the region where X has a uniform distribution becomes narrower, making it harder for the Gibbs sampler to explore the whole region, therefore leading to slower convergence and less effective sampling.

e

As the transformation from X to U is given by U1 = X1 - X2 and U2 = X1 + X2. The inverse transformation from U to X is given by X1 = (U2 + U1)/2 and X2 = (U2 - U1)/2.

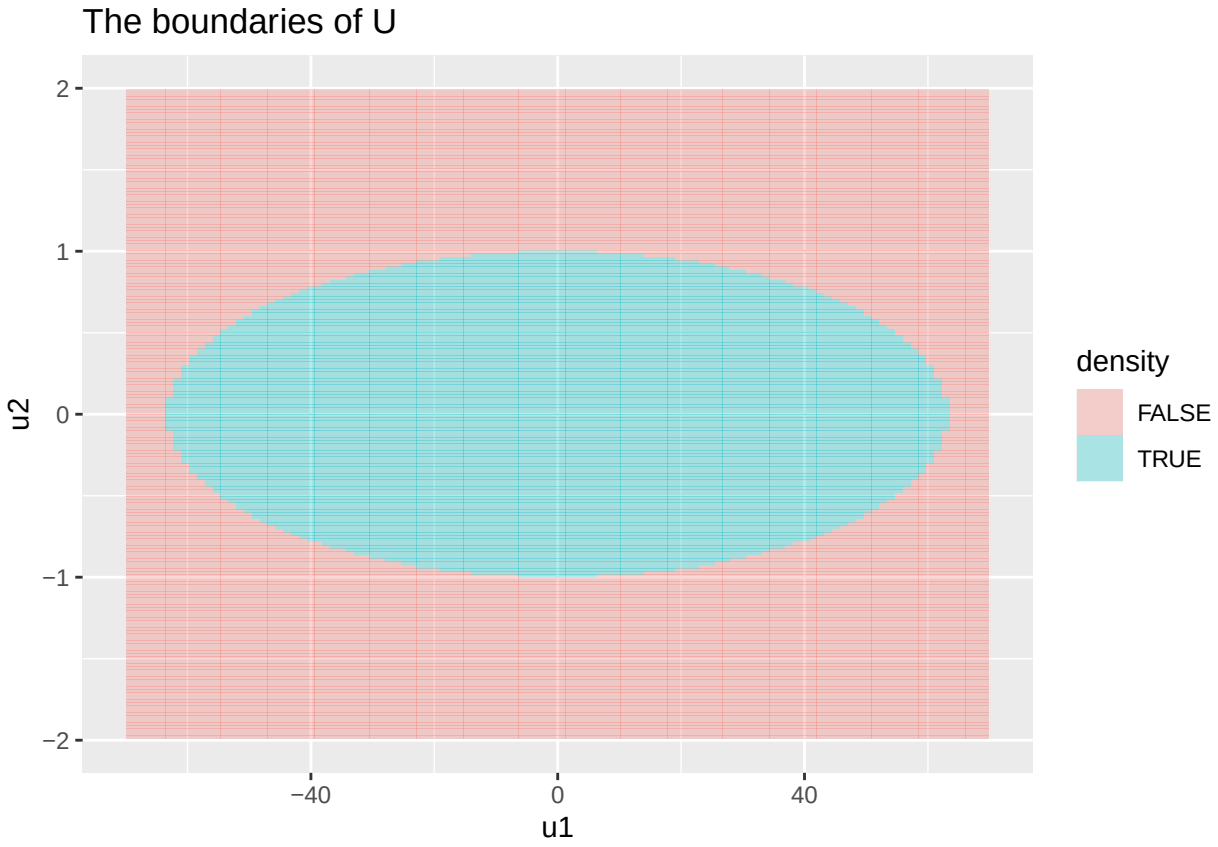
Then substitute these into the boundary equation for X, which is $x1^2 + wx1x2 + x2^2 < 1$, we get:

$$((U2 + U1)/2)^2 + w * ((U2 + U1)/2) * ((U2 - U1)/2) + ((U2 - U1)/2)^2 < 1$$

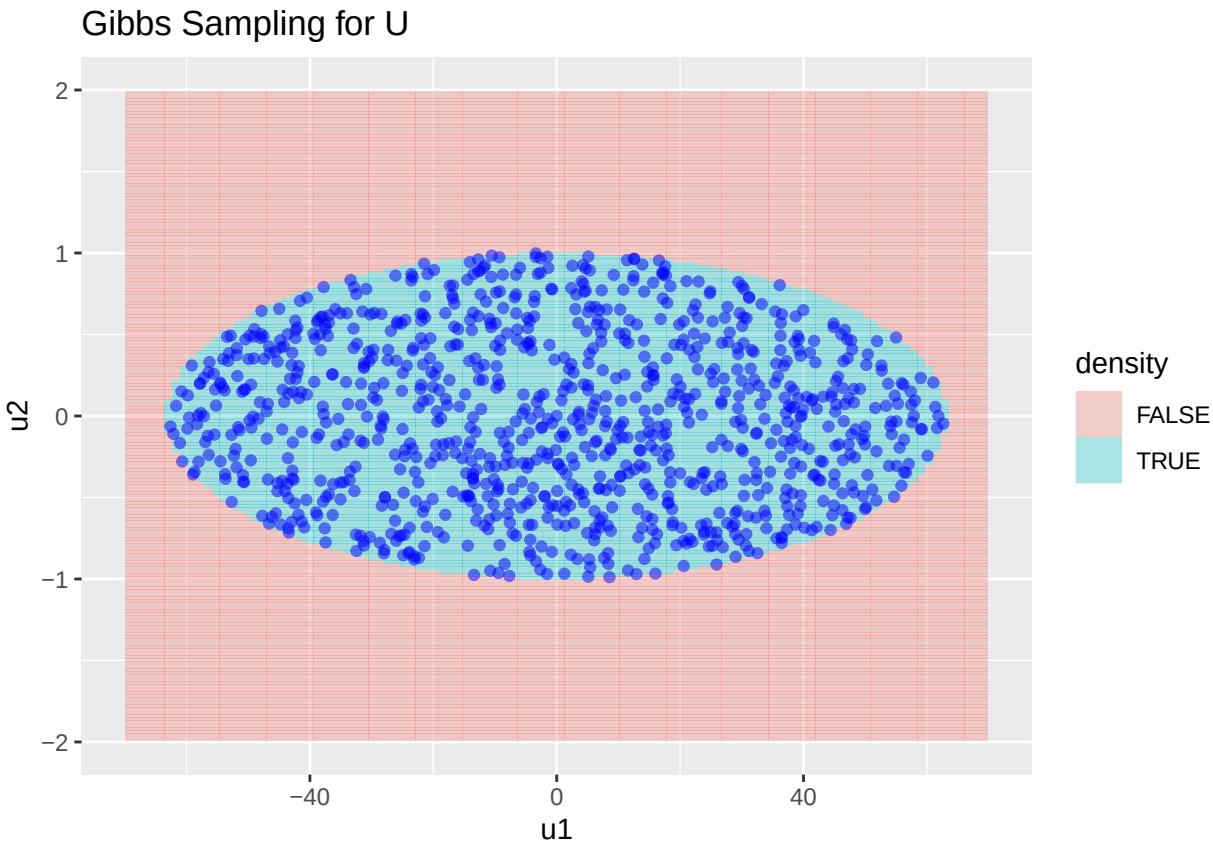
Simplifying this equation gives us the boundaries for U:

$$(2 + w)U2^2 + (2 - w)U1^2 < 4$$

```
## Warning: Removed 18220 rows containing missing values (`geom_tile()`).
```




```
## Warning: Removed 18220 rows containing missing values (`geom_tile()`).
```




```
## P(X1 > 0) = P((U2+U1)/2 > 0): 0.501
```

For part c, when the parameter w is close to 2, the region where X has a uniform distribution becomes very narrow. This makes it difficult for the Gibbs sampler to explore the whole region, especially if the initial point is not well-chosen. As a result, the Gibbs sampler might get "stuck" in certain areas of the distribution, leading to a biased sample.

On the other hand, when transforming the variable X to U = (U1, U2) = (X1 - X2, X1 + X2), the region where U has a uniform distribution is a square, which is much easier for the Gibbs sampler to explore. The calculated probability $P(X1 > 0) = P((U2+U1)/2 > 0) = 0.501$ which is approximately 0.5, it suggests that in the distribution is symmetric around X1=0, which is also consistent with what plot shows. Therefore, Gibbs sampling for U tends to be more successful in generating a representative sample from the distribution.

Appendix:

Question 1

a

```
# Load the 'coda' library for traceplot and autocorr.plot
#install.packages("coda")
library(coda)

# Function for the target distribution
target_distribution <- function(x) {
  return(x^5 * exp(-x))
}
```

```r
# Metropolis hastings method when proposal distribution is log-normal function
metropolis_hastings_log_normal <- function(n) {
  set.seed(123)
  chain <- rep(0,n)
  chain[1] <- 1
  acceptance_rate <- 0

  for (i in 2:n) {
    proposal <- rlnorm(1, meanlog = log(chain[i-1]), sdlog = 1)
    ratio <- (target_distribution(proposal) * dlnorm(chain[i-1], meanlog = log(proposal), sdlog = 1)) /
    if (runif(1) < min(1,ratio)) {
      chain[i] <- proposal
      acceptance_rate <- acceptance_rate + 1
    } else {
      chain[i] <- chain[i-1]
    }
  }
  acceptance_rate <- acceptance_rate / n
  return(list(chain=chain,acceptance_rate=acceptance_rate))
}

samples1 <- metropolis_hastings_log_normal(n=10000)
cat("When proposal function is log-normal function, the acceptance rate is :", samples1$acceptance_rate

plot(samples1$chain, type = "l", main = "Metropolis-Hastings Chain (log-normal)", xlab = "Iterations", y
hist(samples1$chain, main = "Histogram of the sample (log-normal)", xlab = "Sample Value", breaks = 50,

# Burn-in period analysis
cat("The first 20 chain values as below:\n")
samples1$chain[1:20]

# Autocorrelation plot
autocorr.plot(as.mcmc(samples1$chain), main = "Autocorrelation Plot (log-normal)")
```

b

```r
# Metropolis hastings method when proposal distribution is chi-square function
metropolis_hastings_chi_square <- function(n) {
  set.seed(123)
  chain <- rep(0,n)
  chain[1] <- 1
  acceptance_rate <- 0

  for (i in 2:n) {
    proposal <- rchisq(1, df = floor(chain[i-1]) + 1)
    ratio <- (target_distribution(proposal) * dchisq(chain[i-1], df = floor(proposal) + 1)) / (target_di
    if (runif(1) < min(1,ratio)) {
      chain[i] <- proposal
      acceptance_rate <- acceptance_rate + 1
    } else {
      chain[i] <- chain[i-1]
    }
  }
```

```
  acceptance_rate <- acceptance_rate / n
  return(list(chain=chain,acceptance_rate=acceptance_rate))
}

samples2 <- metropolis_hastings_chi_square(n=10000)
cat("WHen the proposal function is chi-squre function, the acceptance rate is :", samples2$acceptance_ra

plot(samples2$chain, type = "l", main = "Metropolis-Hastings Chain (chi-square)", xlab = "Iterations", y
hist(samples2$chain, main = "Histogram of the sample (chi-suare)", xlab = "Sample Value", breaks = 50,

# Burn-in period analysis
cat("The first 20 chain values as below:\n")
samples2$chain[1:20]

# Autocorrelation plot
autocorr.plot(as.mcmc(samples2$chain), main = "Autocorrelation Plot (chi-square)")
```

c

```
# Metropolis hastings method when proposal distribution is normal function
metropolis_hastings_normal <- function(n) {
  set.seed(123)
  chain <- rep(0,n)
  chain[1] <- 1
  acceptance_rate <- 0

  for (i in 2:n) {
    proposal <- rnorm(1, mean = chain[i-1], sd = 1)
    # proposal normal function is symmetric, therefore no need to calculate g(x(t)|x(*)) and g(x(*)|x(t
    ratio <- target_distribution(proposal) / target_distribution(chain[i-1])
    if (runif(1) < min(1,ratio)) {
      chain[i] <- proposal
      acceptance_rate <- acceptance_rate + 1
    } else {
      chain[i] <- chain[i-1]
    }
  }
  acceptance_rate <- acceptance_rate / n
  return(list(chain=chain,acceptance_rate=acceptance_rate))
}

samples3 <- metropolis_hastings_normal(n=10000)
cat("When the proposal function is normal function, the acceptance rate is :", samples3$acceptance_rate

plot(samples3$chain, type = "l", main = "Metropolis-Hastings Chain (normal)", xlab = "Iterations", ylab
hist(samples3$chain, main = "Histogram of the sample (normal)", xlab = "Sample Value", breaks = 50,  co

# Burn-in period analysis
cat("The first 20 chain values as below:\n")
samples3$chain[1:20]

# Autocorrelation plot
autocorr.plot(as.mcmc(samples3$chain), main = "Autocorrelation Plot (normal)")
```

d

```
results_df <- data.frame(proposal_function=c("log-normal", "chi-square", "normal"),
                         acceptance_rate=c(samples1$acceptance_rate, samples2$acceptance_rate, samples3$
                         mean=c(mean(samples1$chain), mean(samples2$chain), mean(samples3$chain)))

print(results_df)
```

e

```
mean_df <- data.frame(proposal_function=c("log-normal", "chi-square", "normal"),
                         mean=c(mean(samples1$chain), mean(samples2$chain), mean(samples3$chain)))

print(mean_df)
```

f

```
# Define the function with scaling factor
# integral of function from 0 to infinity should be 6
# c*120=6 ==> c=0.05
f <- function(x) {0.05*x^5 * exp(-x)}

# Compute the integral from 0 to Inf
result <- integrate(f, lower = 0, upper = Inf)

# Print the result
cat("The actual integral value of gamma distribution is: ",result$value, "\n")
```

Question 2

a

```
# load necessary library
library(ggplot2)

# Function for the target bivariate distribution
target_bivariate_function <- function(x1, x2, w) {
  indicator <- (x2^2 + w*x1*x2 + x1^2 < 1)
  return(indicator)
}

w <- 1.999
x1 <- seq(-1, 1, length.out = 200) * 1/sqrt(1-w^2/4) # a range of x1-values, where the term below the r
x2 <- seq(-1, 1, length.out = 200) * 1/sqrt(1-w^2/4) # a range of x1-values, where the term below the r
df <- expand.grid(x1 = x1, x2 = x2)
df$inside <- target_bivariate_function(df$x1, df$x2, w)
ggplot(df, aes(x1, x2)) +
  geom_tile(aes(fill = inside), alpha = 0.3) +
  ggtitle("The boundaries of the region where X has a uniform distribution") + coord_fixed()
```

c

```
# Gibbs Sampling Function
gibbs_sampling <- function(n, w) {
  samples <- matrix(0, n, 2)
  x1 <- 0
  x2 <- 0
  set.seed(123)

  for (i in 1:n) {
    # Sample x1 given x2
    x1 <- runif(1, -(w/2)*x2-sqrt(1-(1-w^2/4)*x2^2), -(w/2)*x2+sqrt(1-(1-w^2/4)*x2^2))
    # Sample x2 given x1
    x2 <- runif(1, -(w/2)*x1-sqrt(1-(1-w^2/4)*x1^2), -(w/2)*x1+sqrt(1-(1-w^2/4)*x1^2))
    samples[i,] <- c(x1, x2)
  }

  return(samples)
}
# Set parameters
n <- 1000
w <- 1.999

# Run Gibbs sampling
samples <- gibbs_sampling(n, w)

# Plot the boundaries
ggplot(df, aes(x1, x2)) +
  geom_tile(aes(fill = inside), alpha = 0.3) +
  geom_point(data = as.data.frame(samples), aes(x = V1, y = V2), color = "red", alpha = 0.5) +
  ggtitle("Gibbs Sampling for X") + coord_fixed()

# Determine P(X1 > 0)
prob_x1_positive <- mean(samples[,1] > 0)
cat("P(X1 > 0):", prob_x1_positive, "\n")
```

d

e

```
# U Boundaries
u_boundary_function <- function(u1, u2, w) {
  u_indicator <- ((2+w)*u2^2 + (2-w)*u1^2 < 4)
  return(u_indicator)
}

w <- 1.999
u1 <- seq(-2, 2, length.out = 200) * sqrt(4/(2-w))  # a range of u1-values
u2 <- seq(-2, 2, length.out = 200) * sqrt(4/(2+w))  # a range of 21-values
u_df <- expand.grid(u1 = u1, u2 = u2)
u_df$density <- u_boundary_function(u_df$u1, u_df$u2, w)
ggplot(u_df, aes(u1, u2)) +
  geom_tile(aes(fill = density), alpha = 0.3) + xlim(-70,70) + ylim(-2,2) +
  ggtitle("The boundaries of U")
```

```r
# Gibbs Sampling function for U
u_function <- function(n, w) {
  set.seed(123)
  u <- matrix(nrow=n, ncol=2)
  u[1,] <- c(0, 0)  # starting value

  for (i in 2:n) {
    u[i,1] <- runif(1, -sqrt((4-(2+w)*u[i-1,2]^2)/(2-w)), sqrt((4-(2+w)*u[i-1,2]^2)/(2-w)))
    u[i,2] <- runif(1, -sqrt((4-(2-w)*u[i,1]^2)/(2+w)), sqrt((4-(2-w)*u[i,1]^2)/(2+w)))
  }
  return(u)
}

# Set parameters
n <- 1000
w <- 1.999

# Run u function
u_samples <- u_function(n, w)

# Plot gibbs sampling for U
ggplot(u_df, aes(u1, u2)) +
  geom_tile(aes(fill = density), alpha = 0.3) + xlim(-70,70) + ylim(-2,2) +
  geom_point(data = as.data.frame(u_samples), aes(x = V1, y = V2), color = "blue", alpha = 0.5) +
  ggtitle("Gibbs Sampling for U")

# Determine P(X1 > 0) = P((U2+U1)/2 > 0)
prob_x1_positive_u <- mean((u_samples[,2] + u_samples[,1])/2 > 0)
cat("P(X1 > 0) = P((U2+U1)/2 > 0):", prob_x1_positive_u, "\n")
```