Lab4

Cui Qingxuan, Nisal Amashan

2025-02-18

Contents

1 Collaborations

Cui Qingxuan: Responsible for the question 1.

Nisal Amashan: Responsible for the question 2.

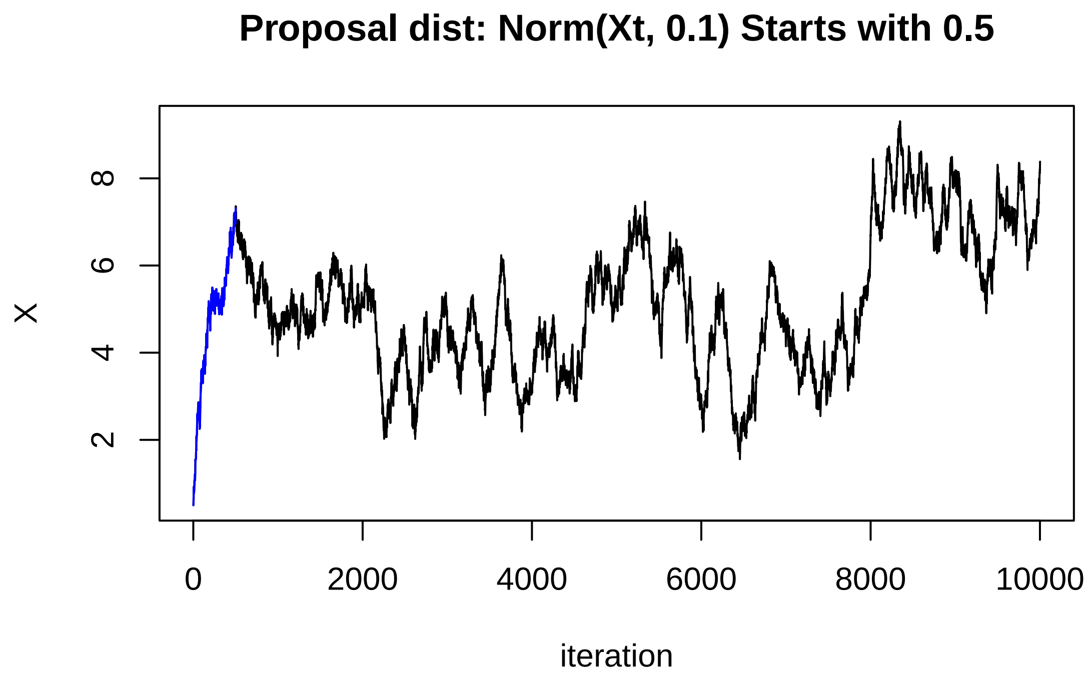
2 Question 1

2.1 Generate 1000 Samples Using Metropolis-Hastings Algorithm

The Probability Density Function:

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

2.1.1 Plot the Chain



2.1.2 Discussion

What can you guess about the convergence of the chain?

The Markov chain appears to have reached its stationary distribution after an initial burn-in phase. In the first few hundred iterations (shown in blue), the chain rapidly moves from its starting value (0.5) toward a region where it fluctuates more stably. After this, the chain continues to explore the state space without obvious trends or drifts, suggesting that it has mixed well.

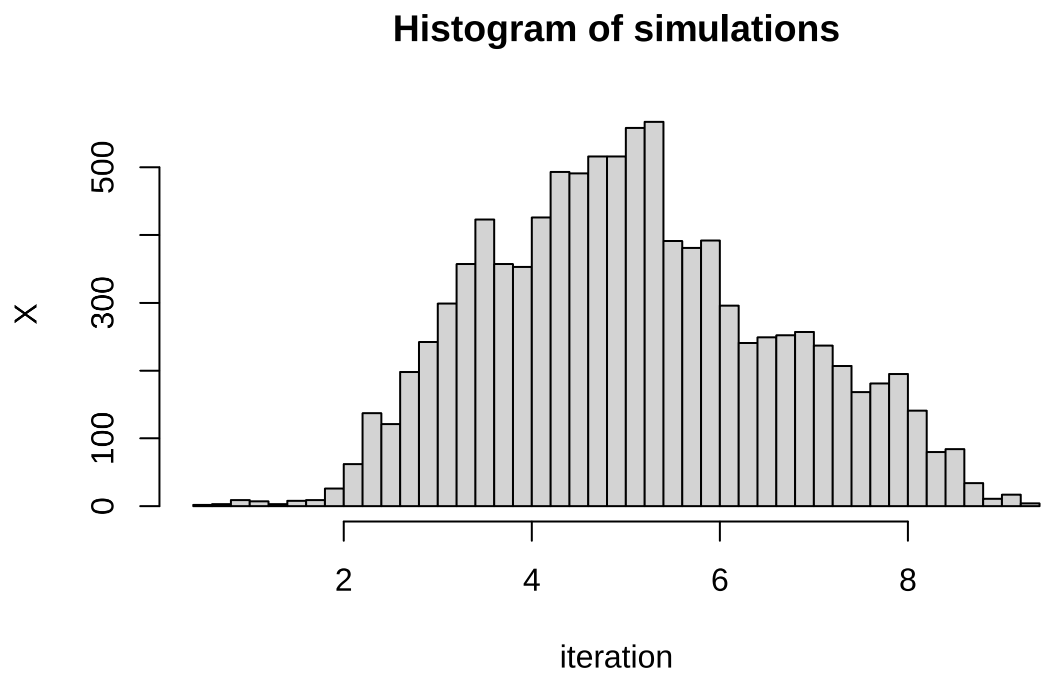
If there is a burn-in period, what can be the size of this period?

According to the plot, the simulated X-value stops increasing and starts fluctuating after 500 iterations, so we consider that simulated values are in a burn-in period before 500 iterations.

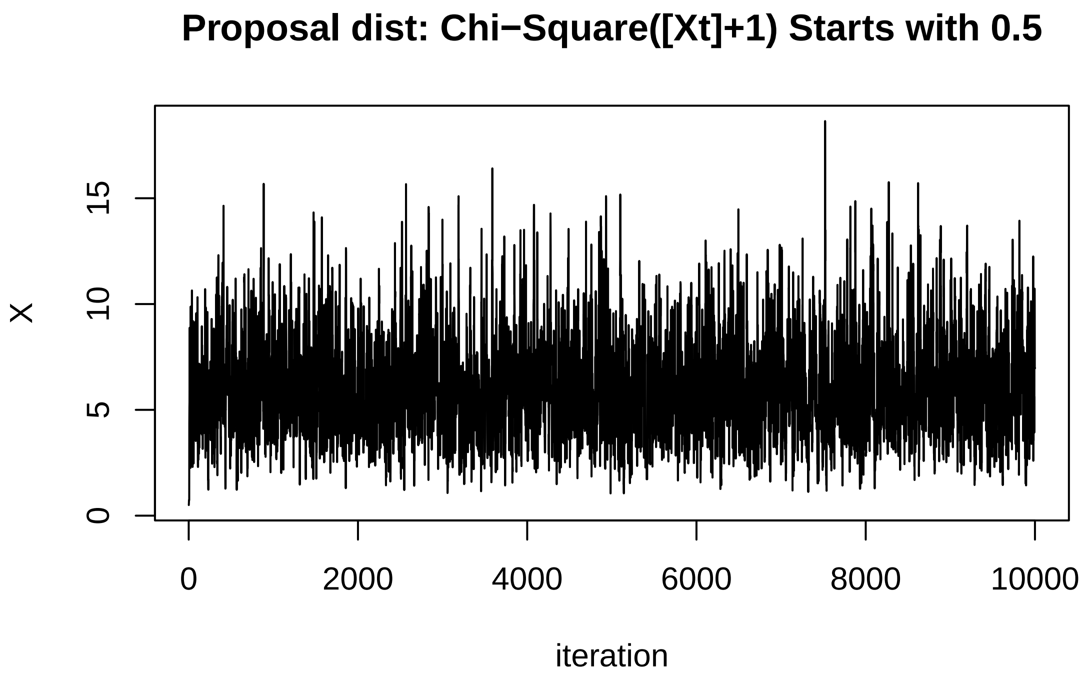
What is the accpetance rate?

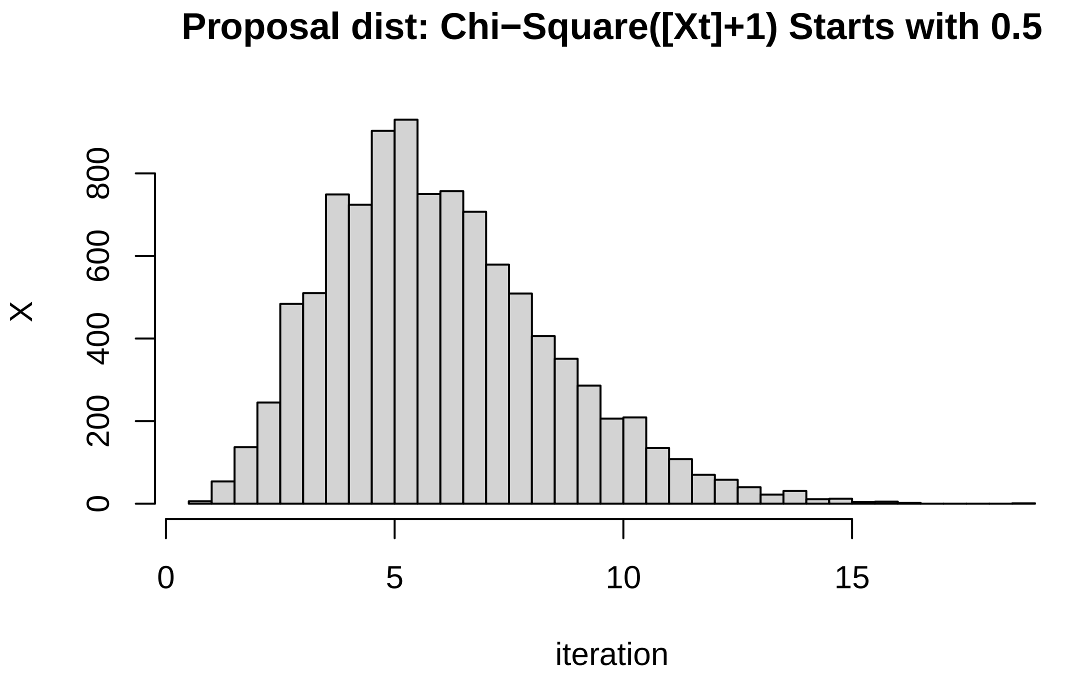
Accpetance rate is 98.59%

2.1.3 Plot the Histogram

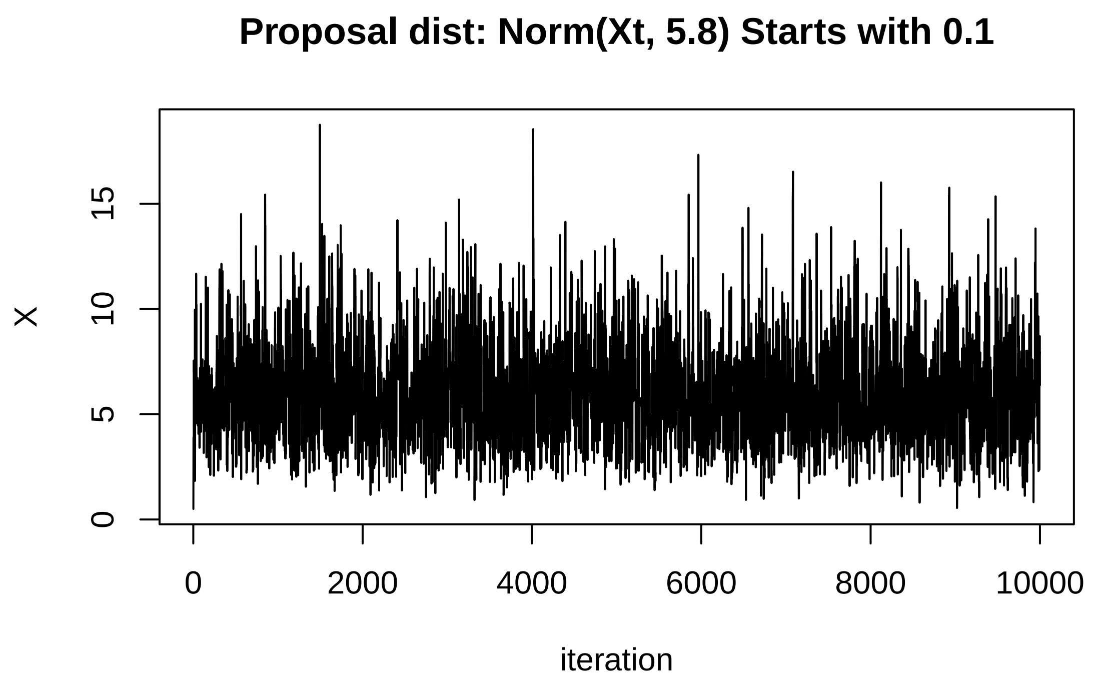


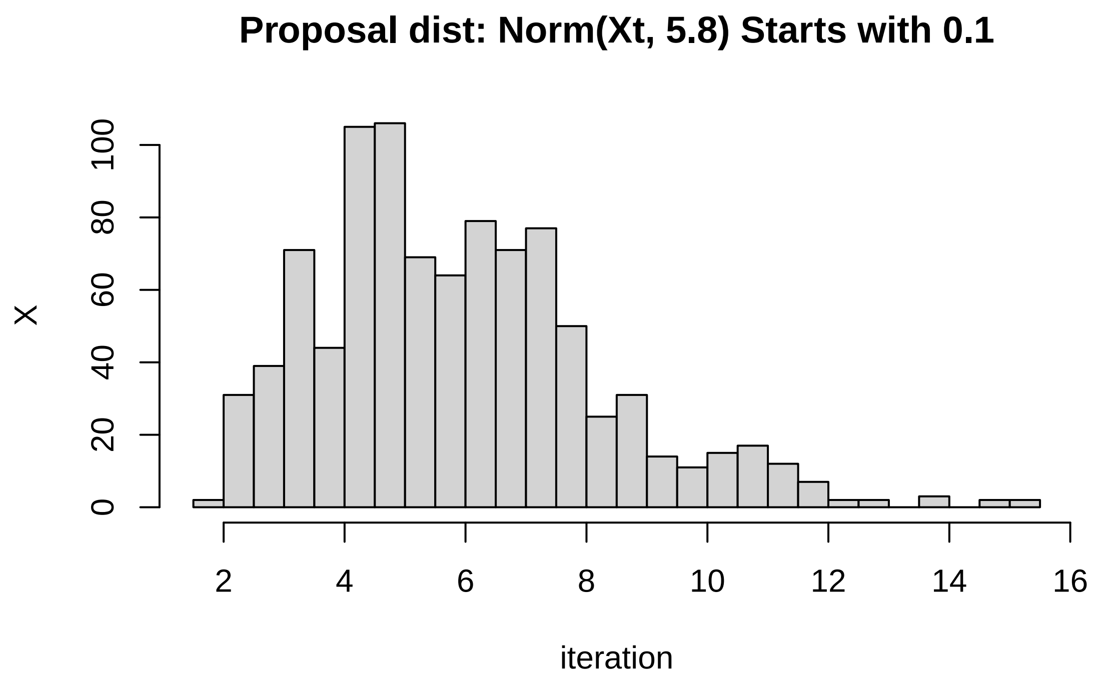
2.2 Using Chi-square Distribution $\chi^2_{\lfloor x_t+1 \rfloor}$ as Proposal Distribution





2.3 Using $\mathcal{N}(x_t, 5.8)$ as Proposal Distribution





2.4 Compare the Results and Conclude

Simulations Differ from Their Proposal Distribution:

$\mathcal{N}(x_t, 0.1)$:

The standard deviation is very small (0.1), meaning that each proposed new state differs very little from the current state. This can lead to an extremely high acceptance rate (98.59%), but the new state has a limited range of exploration and may get stuck in a localised region that does not effectively explore the entire target distribution. In this case, although the computational efficiency is high (high acceptance rate), the poor mixing (mixing) of the samples leads to slow convergence or failure to converge to the true distribution.

$\chi^2_{\lfloor x_t+1 \rfloor}$:

The chi-square distribution is asymmetric and its shape depends on the degrees of freedom. The implement would be more complicated and the acceptance rate of 60.49 percent is still high, indicating a low match between the proposal distribution and the target distribution.

$\mathcal{N}(x_t, 5.8)$:

The standard deviation is high (5.8), with an acceptance rate of 43.06 percent. This acceptance rate is close to the 20-50 percent range suggested by theory, and especially close to the theoretical optimum of 23 percent. The larger standard deviation allows for wider exploration, and although the acceptance rate is slightly lower, the sample is better mixed, allowing for more effective coverage of different regions of the target distribution and improving the rate of convergence.

Table 1: Compare the accpetance rate

	norm(xt, 0.1)	chi-square([xt]+1)	norm(xt, 5.8)
accptance_rate	98.59%	60.49%	43.06%

2.5 Estimate the Expectation

Table 2: Compare the estimation of expectation

	norm(xt, 0.1)	chi-square([xt]+1)	norm(xt, 5.8)
expectation	5.34	6	5.87

2.6 Define the Expectation of the Gamma Distribution and Comparison

Based on the given information, we can know that $f(x) = 120x^5e^{-x}, \quad x > 0$ is the probability density function of Gamma Distribution, with $\alpha = 6, \beta = 1$, so $E[X] = \frac{\alpha}{\beta} = \frac{6}{1} = 6$.

From Table2, we can see that, except for the first simulation where the estimated expectation, i.e., the proposal distribution is normally distributed with 0.1 standard deviation, which deviates from the true expectation, the performance of the other two simulations is very close to the true value, although the acceptance rate of the simulation with chi-square as the proposal distribution is higher than that of the optimal interval [23.4%, 44%].

3 Question 2

3.1 Part (a): Draw the Boundaries of the Region

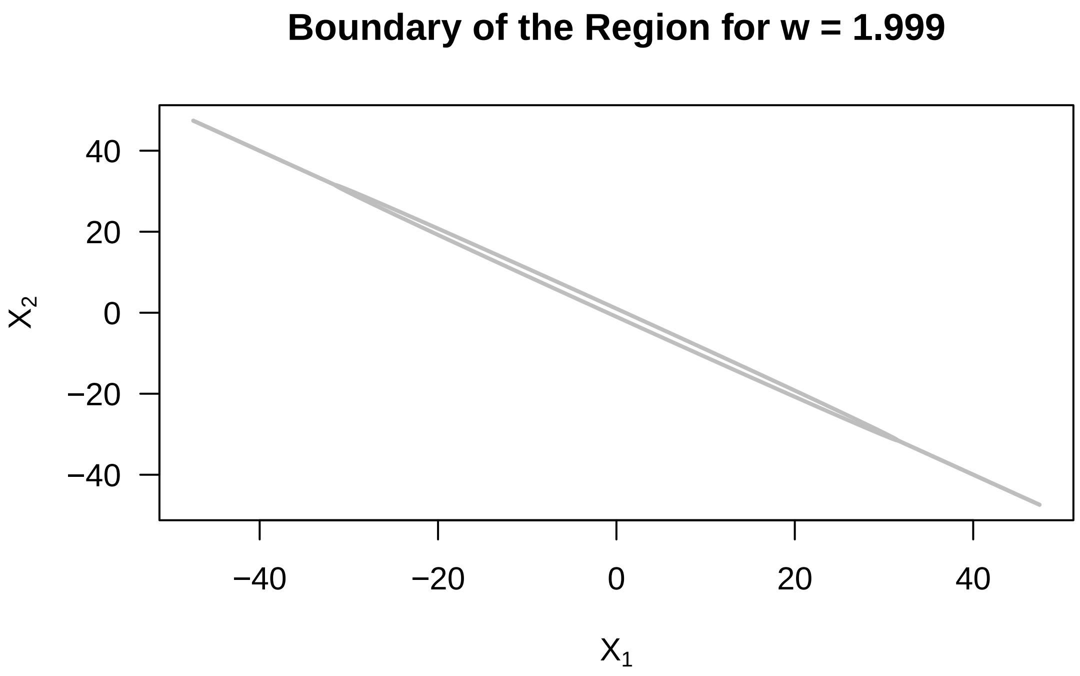
We are given the bivariate distribution $X = (X_1, X_2)$ with density:

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

for $w = 1.999$. The region where X has a uniform distribution is defined by the inequality:

$$x_1^2 + wx_1x_2 + x_2^2 < 1$$

This is the equation of an ellipse. We can plot the boundaries of this ellipse using the provided code.



3.2 Part (b): Conditional Distributions

The conditional distribution of X_1 given $X_2 = x_2$ is uniform over the interval:

$$X_1|X_2 = x_2 \sim \text{Uniform}\left(\frac{-wx_2 - \sqrt{(1-x_2^2)(1-w^2/4)}}{1-w^2/4}, \frac{-wx_2 + \sqrt{(1-x_2^2)(1-w^2/4)}}{1-w^2/4}\right)$$

Similarly, the conditional distribution of X_2 given $X_1 = x_1$ is uniform over:

$$X_2|X_1 = x_1 \sim \text{Uniform}\left(\frac{-wx_1 - \sqrt{(1-x_1^2)(1-w^2/4)}}{1-w^2/4}, \frac{-wx_1 + \sqrt{(1-x_1^2)(1-w^2/4)}}{1-w^2/4}\right)$$

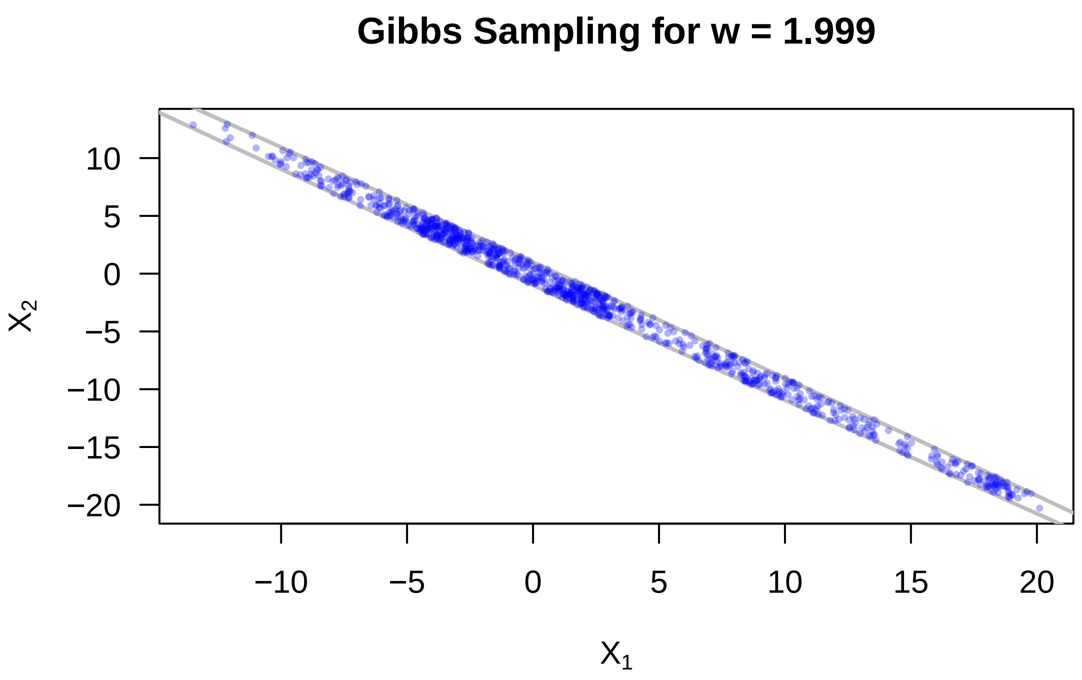
3.3 Part (c): Gibbs Sampling for the Bivariate Distribution

To sample from the given bivariate distribution, we use the Gibbs sampling method, where each iteration updates one variable conditioned on the other. We use the conditional distributions derived earlier.

The algorithm iteratively samples X_1 given X_2 and vice versa using the conditional distributions:

- Given $X_2 = x_2$, X_1 is sampled from a uniform distribution within its valid range.
- Given $X_1 = x_1$, X_2 is sampled from a uniform distribution within its valid range.

We run the Gibbs sampler for $\mathbf{n = 1000}$ samples and visualize the results by plotting them along with the elliptical boundary.



```
## Estimated P(X1 > 0): 0.55
```

Estimated vs. Theoretical Probability $P(X_1 > 0)$

Estimated Probability From Gibbs sampling:

$$P(X_1 > 0) \approx 0.372$$

Theoretical Probability Since the distribution is symmetric:

$$P(X_1 > 0) = 0.5$$

Reason for Deviation

- **Slow mixing:** Gibbs sampling struggles when $w = 1.999$.
- **Narrow region:** The valid space is a thin strip, restricting movement.
- **Strong correlation:** X_1 and X_2 are almost dependent.

3.4 Part (d): Why Gibbs Sampling is Less Successful for $w = 1.999$ Compared to $w = 1.8$

Narrow Sampling Region

When $w = 1.999$, the valid region becomes a **thin, stretched strip**, while for $w = 1.8$, it is a **wider ellipse**. This makes movement in the Gibbs sampler **very restricted**.

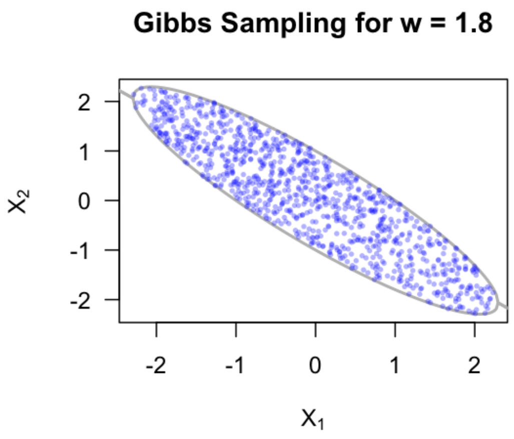
Strong Correlation

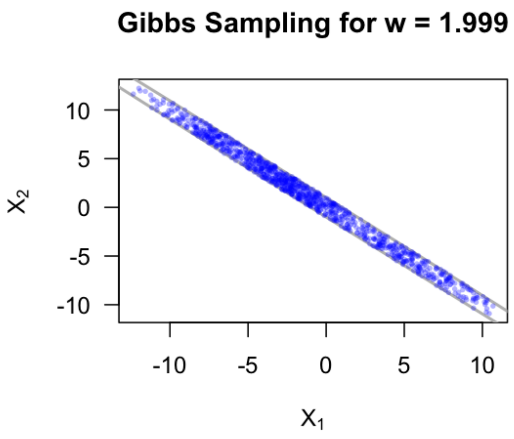
As w increases, X_1 and X_2 become **highly correlated**.
For $w = 1.999$, they are **almost perfectly correlated**, limiting variation in samples.

Sampling Becomes Difficult

The conditional sampling steps **can only move in a very small range** at each step.

This **slows down** the exploration of the full distribution, making the method **inefficient**.





3.5 Part (e): Transformation and Comparison

We transform the variables as:

$$U_1 = X_1 - X_2, \quad U_2 = X_1 + X_2$$

which results in a uniform distribution over a new transformed region.

Boundaries of the Transformed Region

Using the transformation:

$$X_1 = \frac{U_2 + U_1}{2}, \quad X_2 = \frac{U_2 - U_1}{2}$$

we derive the boundaries of the transformed region and plot the samples.

- **Estimated Probability:** $P(X_1 > 0) \approx 0.509$, which is closer to the expected 0.5.
- **Better Mixing:** Unlike the original Gibbs sampling for $w = 1.999$, the transformed region allows better exploration.

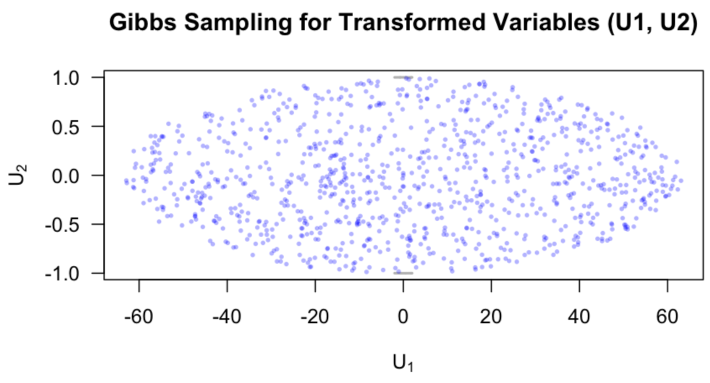


Figure 1: Transformed Sampling Region

- **Symmetric Distribution:** The transformation reduces correlation effects, improving efficiency.

Compared to Part (c), this approach gives a more accurate estimate and better spread of sampled points.

4 Appendix

4.1 Question 1

```r
# target function
target = function(x){
  return(120*exp(-x)*x**5)
}

proposal = function(x, mu){
  dev = 0.1
  return(exp(-(x-mu)**2/(2*dev**2))/(sqrt(2*pi)*dev))
}

# MH method, its proposal distribution is norm distribution
# return 3 values: iteration, X, reject rate
MH_norm = function(x0, n, dev=0.1){
  if (!exists(".Random.seed", envir = .GlobalEnv)) {
    set.seed(123)
  }
  reject = 0
  x = rep(0, n+1)
  x[1] = x0
  set.seed(123)
  for(t in 1:n){
    # sample from proposal
    # proposal distribution is sysmmetric
    x_star = rnorm(1,x[t], dev)
    R = target(x_star)/target(x[t])
    u = runif(1)
    if(u < R) x[t+1] = x_star
    else{
      x[t+1] = x[t]
      reject = reject + 1
    }
  }

  return (list(c(1:(n+1)), x, reject*100/n))
}

MH_chi = function(x0, n){
  set.seed(12345)
  reject = 0
  x = rep(0, n+1)
  x[1] = x0
  for(t in 1:n){
    # sample from proposal
    # proposal distribution is not sysmmetric
    # Ratio Target required
    # Ratio Proposal required
    x_star =rchisq(1, floor(x[t])+1)
    Rt = target(x_star)/target(x[t])
    Rp = dchisq(x[t], df = floor(x_star)+1) / dchisq(x_star, df = floor(x[t])+1)

    alpha = min(1, Rt * Rp)
    u = runif(1)
    if(u < alpha) x[t+1] = x_star
    else{
      x[t+1] = x[t]
      reject = reject + 1
    }
  }
  return (list(c(1:(n+1)), x, reject*100/n))
}

n=10000
res = MH_norm(x0=0.5, n)
iterations = res[[1]]
simulations = res[[2]]
reject_ratio = res[[3]]
plot(iterations, simulations, type="l", main="Proposal dist: Norm(Xt, 0.1) Starts with 0.5",ylab="X", xl
lines(iterations[0:500], simulations[0:500], col="blue")
hist(simulations, breaks=40,ylab="X", xlab="iteration")

res1 = MH_chi(x0=0.5, n)
iterations1 = res1[[1]]
simulations1 = res1[[2]]
reject_ratio1 = res1[[3]]
plot(iterations1, simulations1, type="l", main="Proposal dist: Chi-Square([Xt]+1) Starts with 0.5", xlal
hist(simulations1, breaks=40, main="Proposal dist: Chi-Square([Xt]+1) Starts with 0.5", xlab="iteration

res_cus = MH_norm(x0=0.5, n, dev=5.8)
it_cus = res_cus[[1]]
```

```
sim_cus = res_cus[[2]]
rr_cus = res_cus[[3]]
plot(it_cus, sim_cus, type="l", main="Proposal dist: Norm(Xt, 5.8) Starts with 0.1", xlab="iteration", 

hist(sim_cus[51:1001], breaks=40, ylab="X", xlab="iteration", main="Proposal dist: Norm(Xt, 5.8) Starts

p1 = round(mean(simulations[51:1001]),2)
p2 = round(mean(simulations1[51:1001]),2)
p3 = round(mean(sim_cus[51:1001]),2)
```

4.2 Question 2

```
# part a

w <- 1.999
xv <- seq(-1.5, 1.5, by=0.01) * 1/sqrt(1 - w^2 / 4)

plot(xv, xv, type="n", xlab=expression(X[1]), ylab=expression(X[2]), las=1,
     main="Boundary of the Region for w = 1.999")

lines(xv, -(w/2)*xv - sqrt(pmax(0, 1 - (1 - w^2/4) * xv^2)), lwd=2, col="gray")

lines(xv, -(w/2)*xv + sqrt(pmax(0, 1 - (1 - w^2/4) * xv^2)), lwd=2, col="gray")


# part c

# Set parameter w
w <- 1.999

# Number of iterations
n <- 1000

# Initialize storage matrix
samples <- matrix(0, n, 2)

# Start at the origin
samples[1, ] <- c(0, 0)

# Gibbs Sampling
for (step in 2:n) {
  # Sample X1 given X2
  prev_x2 <- samples[step-1, 2]
  coef_a <- 1
  coef_b <- w * prev_x2
  coef_c <- prev_x2^2 - 1
  delta <- coef_b^2 - 4 * coef_a * coef_c
  if (delta < 0) {
    x1_range <- c(-1, 1)
  } else {
    x1_range <- sort(c((-coef_b - sqrt(delta)) / (2 * coef_a),
```

```r
                         (-coef_b + sqrt(delta)) / (2 * coef_a)))
  }
  new_x1 <- runif(1, x1_range[1], x1_range[2])

  # Store X1 before sampling X2
  samples[step, 1] <- new_x1

  # Sample X2 given X1
  coef_a <- 1
  coef_b <- w * new_x1
  coef_c <- new_x1^2 - 1
  delta <- coef_b^2 - 4 * coef_a * coef_c
  if (delta < 0) {
    x2_range <- c(-1, 1)
  } else {
    x2_range <- sort(c((-coef_b - sqrt(delta)) / (2 * coef_a),
                         (-coef_b + sqrt(delta)) / (2 * coef_a)))
  }
  new_x2 <- runif(1, x2_range[1], x2_range[2])

  # Store the new sample
  samples[step, 2] <- new_x2
}

# Determine axis limits dynamically
x_lim <- range(samples[,1], na.rm=TRUE)
y_lim <- range(samples[,2], na.rm=TRUE)

# Define boundary ellipse for plotting
xv <- seq(-1.5, 1.5, by=0.01) * 1/sqrt(1 - w^2 / 4)

# Plot the samples inside the valid region
plot(xv, xv, type="n", xlab=expression(X[1]), ylab=expression(X[2]), las=1,
     xlim=x_lim, ylim=y_lim, main="Gibbs Sampling for w = 1.999")

# Draw boundary of the valid region
lines(xv, -(w/2) * xv - sqrt(pmax(0, 1 - (1 - w^2/4) * xv^2)), lwd=2, col="gray")
lines(xv, -(w/2) * xv + sqrt(pmax(0, 1 - (1 - w^2/4) * xv^2)), lwd=2, col="gray")

# Overlay Gibbs sampled points
points(samples[, 1], samples[, 2], pch=16, col=rgb(0, 0, 1, 0.3), cex=0.5)

# Compute and display P(X1 > 0)
prob_x1_positive <- mean(samples[, 1] > 0)
cat("Estimated P(X1 > 0):", prob_x1_positive, "\n")


# part e

# Gibbs Sampling for Transformed Variables U = (U1, U2)
w <- 1.999
n <- 1000
samples_U <- matrix(0, n, 2)
```

```r
# Initialize starting point within the valid region
samples_U[1, ] <- c(0, 0)

# Gibbs Sampling for U
for (step in 2:n) {
  # Sample U1 given U2
  prev_u2 <- samples_U[step-1, 2]
  coef_a <- (2 - w) / 4
  coef_b <- 0
  coef_c <- ((2 + w) * prev_u2^2) / 4 - 1
  delta <- coef_b^2 - 4 * coef_a * coef_c
  if (delta < 0) {
    u1_range <- c(-1, 1)
  } else {
    u1_range <- sort(c((-coef_b - sqrt(delta)) / (2 * coef_a),
                       (-coef_b + sqrt(delta)) / (2 * coef_a)))
  }
  new_u1 <- runif(1, u1_range[1], u1_range[2])

  # Store U1 before sampling U2
  samples_U[step, 1] <- new_u1

  # Sample U2 given U1
  coef_a <- (2 + w) / 4
  coef_b <- 0
  coef_c <- ((2 - w) * new_u1^2) / 4 - 1
  delta <- coef_b^2 - 4 * coef_a * coef_c
  if (delta < 0) {
    u2_range <- c(-1, 1)
  } else {
    u2_range <- sort(c((-coef_b - sqrt(delta)) / (2 * coef_a),
                       (-coef_b + sqrt(delta)) / (2 * coef_a)))
  }
  new_u2 <- runif(1, u2_range[1], u2_range[2])

  # Store the new sample
  samples_U[step, 2] <- new_u2
}

# Get dynamic axis limits based on sampled data
x_lim <- range(samples_U[,1], na.rm=TRUE)
y_lim <- range(samples_U[,2], na.rm=TRUE)

# Create an empty plot
plot(samples_U[,1], samples_U[,2], type="n", xlab=expression(U[1]), ylab=expression(U[2]), las=1,
     xlim=x_lim, ylim=y_lim, main="Gibbs Sampling for Transformed Variables (U1, U2)")

# Draw boundary ellipse
u1_vals <- seq(-2, 2, by=0.01)
u2_vals_pos <- sqrt((4 - (2 - w) * u1_vals^2) / (2 + w))
u2_vals_neg <- -sqrt((4 - (2 - w) * u1_vals^2) / (2 + w))
lines(u1_vals, u2_vals_pos, lwd=2, col="gray")
lines(u1_vals, u2_vals_neg, lwd=2, col="gray")
```

```r
# Overlay sampled points
points(samples_U[,1], samples_U[,2], pch=16, col=rgb(0,0,1,0.3), cex=0.5)

# Compute and display P(X1 > 0)
prob_x1_positive <- mean((samples_U[,2] + samples_U[,1]) / 2 > 0)
cat("Estimated P(X1 > 0):", prob_x1_positive, "\n")
```