

Project 1, TMA4300

(mihan@stud.ntnu.no), (alessatc@stud.ntnu.no)

February 9, 2026

Contents

Introduction	2
Problem 1	2
Point a)	2
Point b)	3
Point c)	3
Problem 2	4
Point a)	4
Point b)	4
Point c)	5
Problem 3	7
Point a)	7
Point b)	8
Problem 4	9
Point a)	9
Point b)	10
Problem 5	10
Point a)	10
Point b)	12
Problem 6	14
Point a)	14
Point b)	15
Problem 7	17
Point a)	17
Point b)	18
Point c)	18
Point d)	20
Point e)	21
Problem 8	21
Point a)	21
Point b)	22
Point c)	23
Point d)	25
Plot Functions	25

Introduction

This report presents Project 1 for the course TMA4300 – Computer Intensive Statistical Methods, taught by Professor Jarle Tufto at NTNU.

Throughout the project, we make extensive use of the random number generators provided by the R programming language. In particular, we adopt the Mersenne–Twister generator due to its very long period, good equidistribution properties, and widespread use in statistical computing. To ensure full reproducibility of the results, the random seed is fixed.

```
RNGkind(kind = "Mersenne-Twister"); # Use Mersenne-Twister generator
set.seed(9);                         # set seed for reproducibility
```

All plots are generated through specific functions and collected in the section "Plot Functions".

Problem 1

Point a)

Consider the random variable X with density

$$f(x) = \frac{1}{\pi(1+x^2)} \quad \text{for } -\infty < x < \infty \quad (1)$$

The CDF of X can be computed as

$$F_X(k) = \int_{-\infty}^k f(x) \, dx = \int_{-\infty}^k \frac{1}{\pi(1+x^2)} \, dx = \frac{1}{\pi} \arctan(k) + \frac{1}{2} \quad (2)$$

Using the inversion method, a sample from X can be generated by solving

$$u = \frac{1}{\pi} \arctan(k) + \frac{1}{2} \quad (3)$$

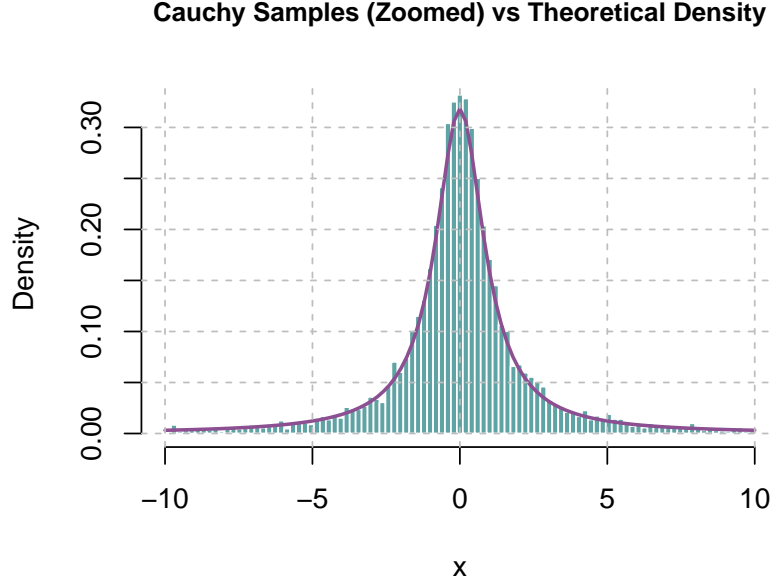
$$\implies k = \tan \left[\pi \left(u - \frac{1}{2} \right) \right] \quad (4)$$

where $U \sim \text{Unif}(0, 1)$

```
u <- runif(10000); # generate the uniform distributed samples
x <- tan((u-1/2)*pi); # apply transformation
```

We can now proceed with the sanity check. Since the distribution is Cauchy, we can visualize the generated samples using a histogram and overlay the theoretical density:

```
plotcauchyzoom(x, primary, secondary)
```



Point b)

Let X be a continuous random variable with probability density function $f_X(x)$. If

$$\mathbb{E}[|X|] = \int_{-\infty}^{\infty} |x| f_X(x) dx < \infty,$$

then the expected value of X is defined as

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x) dx.$$

Since

$$\mathbb{E}[|X|] = \int_{-\infty}^{\infty} |x| f_X(x) dx = \int_{-\infty}^{\infty} \frac{|x|}{\pi(1+x^2)} dx = 2 \int_0^{\infty} \frac{x}{\pi(1+x^2)} dx \sim \int_0^{\infty} \frac{1}{x} dx \rightarrow \infty, \quad (5)$$

the mean $\mathbb{E}[X]$ does not exist.

Point c)

Since we proved that the expected value does not exist (Point b), the strong law of large numbers can't be applied. Assuming $X_i \stackrel{\text{iid}}{\sim} \text{Cauchy}(0, 1)$. Since $\varphi_{X_i}(t) = e^{-|t|}$, we have

$$\varphi_{X_1 + \dots + X_n}(t) = (e^{-|t|})^n \quad (6)$$

$$\implies X_1 + \dots + X_n = Y \sim \text{Cauchy}(0, n) \quad (7)$$

$$\varphi_{\frac{1}{n}Y}(t) = e^{-n|\frac{1}{n}t|} = e^{-|t|} \implies \frac{1}{n}Y \sim \text{Cauchy}(0, 1) \quad \forall n \quad (8)$$

We can see that the mean continues to be a Cauchy-distributed for all n . Thus,

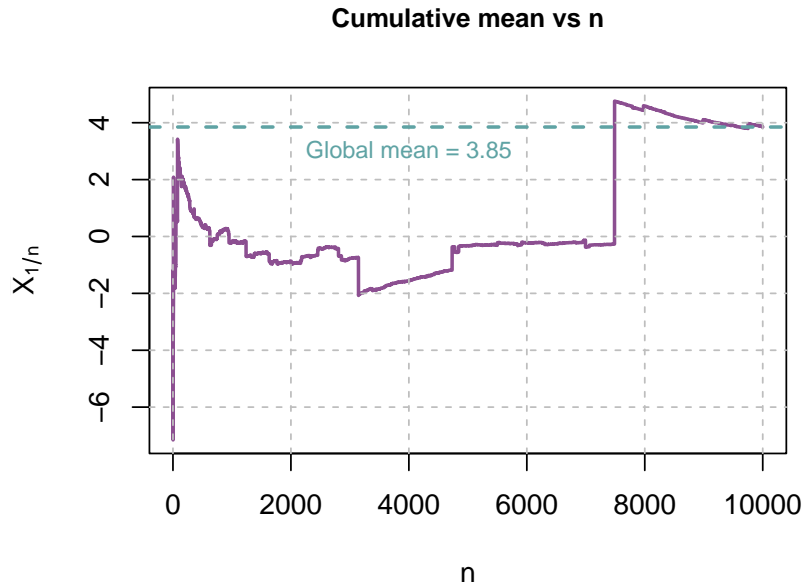
$$\mathbb{P}\left(\left|\frac{1}{n}Y - c\right| \geq \varepsilon\right) = 1 - \int_{c-\varepsilon}^{c+\varepsilon} \frac{1}{\pi(1+x^2)} dx \xrightarrow{n \rightarrow \infty} 0 \quad (9)$$

This implies that the cumulative mean does not converge in probability to any constant (thus not in law, nor almost surely). This explains the behaviour we see in the graph below:

```

cummean <- function(x) {
  cumsum(x) / (1:length(x))
}
plotcumulativemean(cummean)

```



As n increases, the mean doesn't converge to any constant value.

Problem 2

Point a)

Algorithm 1 Left Right Truncated Random Variable Generator

Require: F Require: F^{-1} Require: a Require: b Require: n Ensure: $X_1, \dots, X_n \sim X \mid a \leq X \leq b$	\triangleright Cumulative Distribution Function of X \triangleright Quantile function of X \triangleright Left truncation point (default $-\infty$) \triangleright Right truncation point (default ∞) \triangleright Number of samples \triangleright Generated samples
--	---

```

1: Compute lb  $\leftarrow F(a)$ 
2: Compute ub  $\leftarrow F(b)$ 
3: for  $i = 1$  to  $n$  do
4:   Draw  $U_i \sim \mathcal{U}(0, 1)$ 
5:   Set  $\tilde{U}_i \leftarrow U_i \cdot \text{lb} + (1 - U_i) \cdot \text{ub}$ 
6:   Set  $X_i \leftarrow F^{-1}(\tilde{U}_i)$ 
7: end for
8: return  $(X_1, \dots, X_n)$ 

```

Point b)

Function implementation:

```

rtrunc <- function(n, pfn, qfn, a = -Inf, b = Inf, ...) {
  # calculate lower and upper bounds
  bound.lower <- do.call(pfn, c(list(a), list(...)))

```

```

bound.upper <- do.call(pfn, c(list(b), list(...)))

# generate uniformly randomly distributed samples in the provided interval
u <- runif(n)
u_trunc <- bound.lower*u + (1-u)*bound.upper

# calculate and return the transformed x
return(do.call(qfn, c(list(u_trunc), list(...))))
}

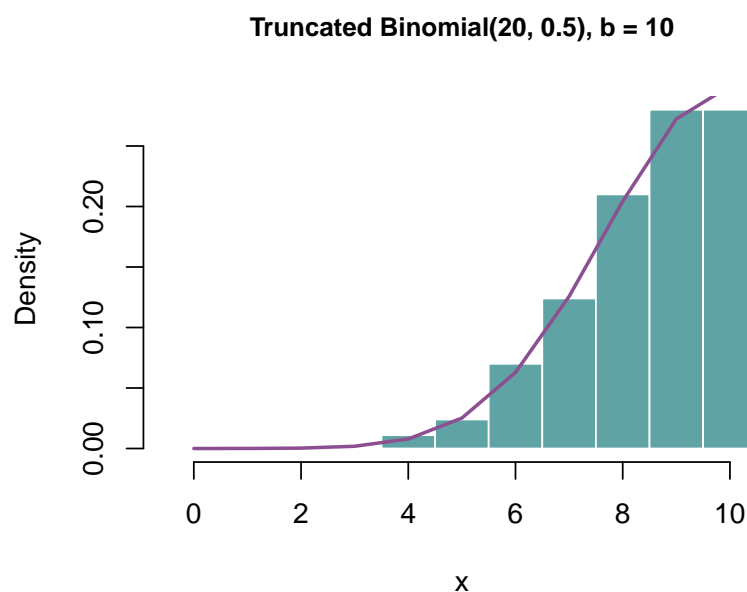
```

Testing with the Binomial distribution truncated for $X \leq 10$

```

x <- rtrunc(1e+3, pbinom, qbinom, b=10, size=20, prob=.5)
rtrunchistbinom(x)

```



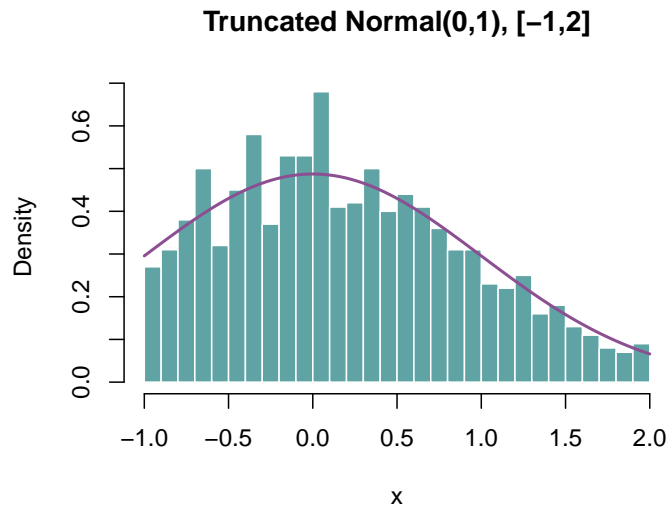
Point c)

We can try applying the same function to the Normal distribution truncating it from -1 to 2.

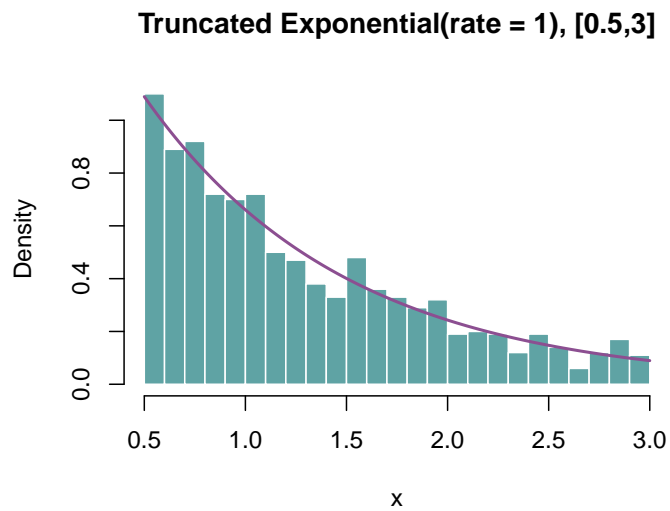
```

x <- rtrunc(1e3, pfn = "pnorm", qfn = "qnorm", a = -1, b = 2, mean = 0, sd = 1)
rtrunchistnorm(x)

```



```
x <- rtrunc(1e3, pfn = "pexp", qfn = "qexp", a = 0.5, b = 3, rate = 1)
rtrunchistexp(x)
```



As shown in the graph above, the function works correctly: the histogram of the simulated samples matches the theoretical distribution within the selected interval.

Problem 3

Point a)

Algorithm 2 Bivariate Normal Sampler Truncated to a Disk

Require: n	▷ Number of samples
Require: ρ	▷ Radius of the truncation disk
Require: δ	▷ Shift along the x -axis
Ensure: $\text{samples} \in \mathbb{R}^{n \times 2}$	▷ Generated samples

- 1: Draw $\theta_1, \dots, \theta_n \sim \mathcal{U}(0, 2\pi)$
- 2: Compute $u_{\max} \leftarrow \exp(-\rho^2/2)$
- 3: Draw $u_1, \dots, u_n \sim \mathcal{U}(u_{\max}, 1)$
- 4: Compute $r_i \leftarrow \sqrt{-2 \log(u_i)}$ for $i = 1, \dots, n$
- 5: Compute $x_{1,i} \leftarrow r_i \cos(\theta_i) + \delta$
- 6: Compute $x_{2,i} \leftarrow r_i \sin(\theta_i)$
- 7: **return** $\text{samples} \leftarrow \{(x_{1,i}, x_{2,i})\}_{i=1}^n$

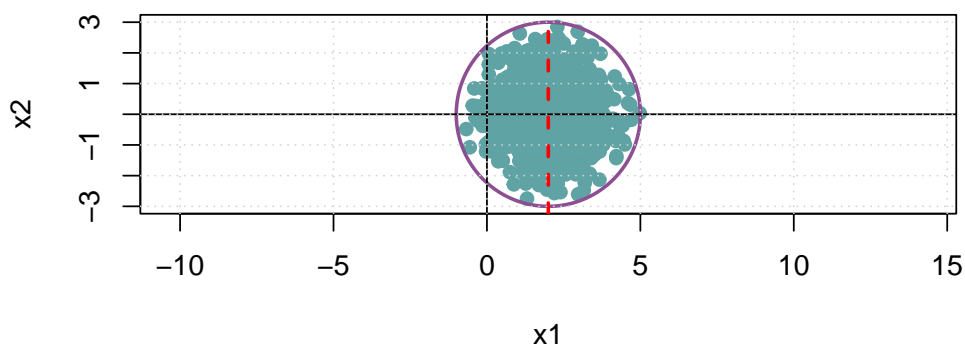
Implementation:

```
# Truncated bivariate normal sampler
bnormsampler <- function(n, rho, delta) {
  theta <- runif(n, 0, 2 * pi)
  u_max <- exp(-rho^2 / 2)
  u <- runif(n, u_max, 1)
  r <- sqrt(-2 * log(u))
  x1 <- r * cos(theta) + delta
  x2 <- r * sin(theta)
  cbind(x1, x2)
}

# Parameters
n <- 1000
rho <- 3
delta <- 2

samples <- bnormsampler(n, rho, delta)
scatterplotdisk(samples, rho, delta)
```

Scatter plot of bnormsampler samples

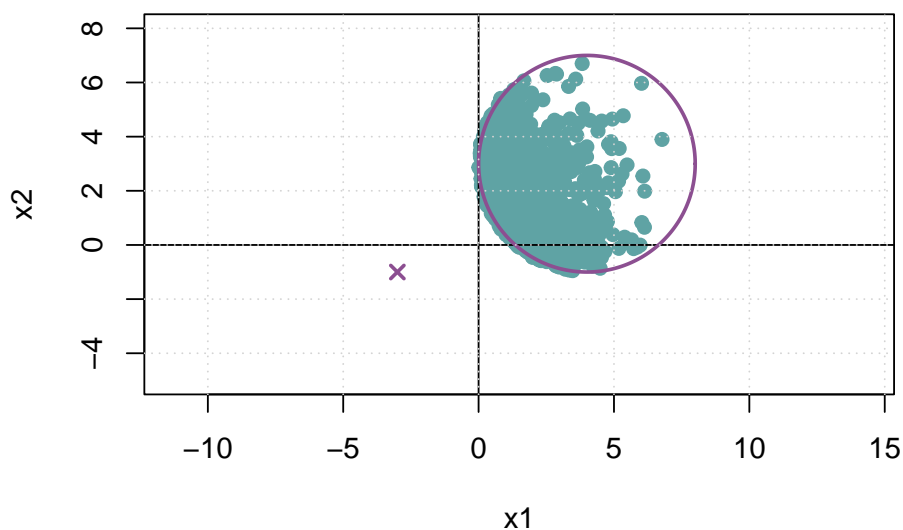


No samples are rejected, so the acceptance rate is exactly 100%, independent of the disk radius ρ or the shift δ .

Point b)

```
gbnormsampler <- function(n, mean1, mean2, std, rho, c1, c2) {  
  
  samples <- matrix(NA, nrow = n, ncol = 2)  
  count <- 0  
  
  while(count < n) {  
    # Generate candidate points  
    u <- runif(1,0,1)  
    theta <- runif(1, 0, 2*pi)  
    r <- sqrt(-2*log(u))  
    x <- r*cos(theta)*std + mean1  
    y <- r*sin(theta)*std + mean2  
  
    # Check if inside the truncation disk  
    if((x - c1)^2 + (y - c2)^2 <= rho^2) {  
      count <- count + 1  
      samples[count, ] <- c(x, y)  
    }  
  }  
  return(samples)  
}  
  
# Parameters  
n <- 1000  
mean1 <- -3  
mean2 <- -1  
std <- 2.9  
rho <- 4  
c1 <- 4  
c2 <- 3  
  
samples <- gbnormsampler(n, mean1, mean2, std, rho, c1, c2)  
gbnormamplerplotter(samples, mean1, mean2, rho, c1, c2)
```

Truncated bivariate normal, mean= $(-3, -1)$, $(4, 3)$



Problem 4

Point a)

In this problem, we have n -uniformly distributed random variables $U_1, \dots, U_n \sim \text{unif}(0, 1)$ from which we can get the joint density of minimum and maximum order statistics $U_{(1)}$ and $U_{(n)}$ using formula:

$$f_{U_{(i)}, U_{(j)}}(u, v) = n! \cdot \frac{u^{i-1}}{(i-1)!} \cdot \frac{(v-u)^{j-i-1}}{(j-i-1)!} \cdot \frac{(1-v)^{n-j}}{(n-j)!} \quad \text{for } 0 < u < v < 1.$$

In our case we have $i = 1$ and $j = n$ so the joint density is:

$$f_{U_{(1)}, U_{(n)}}(u, v) = n(n-1)(v-u)^{n-2}.$$

Our goal is to get the density $f_X(x)$ of random variable

$$X = \frac{\frac{1}{2}(U_{(1)} + U_{(n)}) - \frac{1}{2}}{U_{(n)} - U_{(1)}}.$$

First, we set

$$Y = h(U_{(1)}, U_{(n)}) = U_{(n)} - U_{(1)}$$

and from expression for X we get that

$$\frac{1}{2}(U_{(1)} + U_{(n)}) = X \cdot Y + \frac{1}{2}.$$

We can now express $U_{(1)}$ and $U_{(n)}$ using expressions for X and Y :

$$U_{(1)} = \frac{1}{2} + XY - \frac{Y}{2}$$

and

$$U_{(n)} = \frac{1}{2} + XY + \frac{Y}{2}.$$

Now we want to compute Jacobian determinant based on the inverse transformation, therefore we first compute partial derivatives: $\frac{\partial U_{(1)}}{\partial X} = Y$, $\frac{\partial U_{(1)}}{\partial Y} = X - \frac{1}{2}$, $\frac{\partial U_{(n)}}{\partial X} = Y$ and $\frac{\partial U_{(n)}}{\partial Y} = X + \frac{1}{2}$. We compute Jacobian determinant as:

$$|J| = \begin{vmatrix} Y & X - 1/2 \\ Y & X + 1/2 \end{vmatrix} = Y.$$

Joint density is there calculated using transformation formula and definition of X and Y . We can use tranformation formula sicer we have a one-to-one differentiable function with inverse, as shown above.

$$\begin{aligned} f_{X,Y}(x, y) &= f_{U_{(1)}, U_{(n)}}(u, v) \cdot |J| \\ &= n(n-1)(v-u)^{n-2} \cdot |Y| \\ &= n(n-1)y^{n-2} \cdot y \\ &= n(n-1)y^{n-1}. \end{aligned}$$

We now get density of X by integrating over Y . We get the boundaries for integration from constraint $0 < U_1 < U_n < 1$ from which we get that $y < \frac{1}{1-2x}$ and $y < \frac{1}{1+2x}$. We therefore integrate over y from 0 to $\frac{1}{1+2|x|}$. Hence,

$$\begin{aligned} f_X(x) &= \int_{y=0}^{\frac{1}{1+2|x|}} n(n-1)y^{n-1} dy \\ &= (n-1) \left(\frac{1}{1+2|x|} \right)^n. \end{aligned}$$

Point b)

We now integrate our proposed algorithm into R as seen below:

```
#Problem 4
# We decide to take 100 random uniformly distributed random variables, so we take  $U_{\{1\}}$  and  $U_{\{100\}}$ 
n <- 100
num_simulations <- 1000 #number of samples

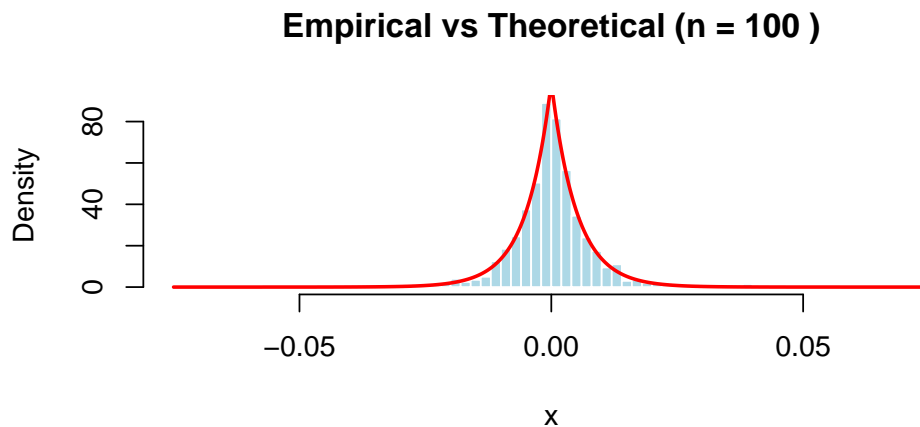
theoretical_fX <- function(x, n) {
  (n - 1) * (1 / (1 + 2 * abs(x))^n)
}

set.seed(42)
x_samples <- replicate(num_simulations, {
  u <- runif(n)
  u_min <- min(u)
  u_max <- max(u)
  x_val <- (0.5 * (u_min + u_max) - 0.5) / (u_max - u_min)
  return(x_val)
})

x_limit <- 0.075

hist(x_samples, breaks = 50, probability = TRUE,
     main = paste("Empirical vs Theoretical (n =", n, ")"),
     xlab = "x", xlim = c(-x_limit, x_limit),
     col = "lightblue", border = "white")

# Add theoretical curve
curve(theoretical_fX(x, n), add = TRUE, col = "red", lwd = 2, n = 1000)
```



Based on graph we conclude that the calculated density function $f_X(x)$ is correct as the curve fits the simulated samples.

Problem 5

Point a)

The aim of this problem is to show how we would simulate data from joint distribution of random variables X and Y . We are given joint distribution:

$$F_{X,Y}(x, y) = \exp\left(-\frac{1}{x} - \frac{1}{y} - \frac{1}{xy}\right) \quad \text{for } x > 0, y > 0.$$

First, we get the marginal cumulative distribution function of X :

$$F_X(x) = \lim_{y \rightarrow \infty} F_{X,Y}(x, y) = \exp\left(-\frac{1}{x}\right) \quad \text{for } x > 0.$$

We want to get cdf of $Y|X = x$ from formula

$$F_{Y|X}(y|x) = \frac{1}{f_X(x)} \frac{\partial}{\partial x} F_{X,Y}(x, y).$$

First, we show theoretically that the formula holds so we can use it later. We can write righthand side as

$$\begin{aligned} RS &= \frac{1}{f_X(x)} \frac{\partial}{\partial x} F_{X,Y}(x, y) \\ &= \frac{1}{f_X(x)} \frac{\partial}{\partial x} \left(\int_{-\infty}^x \int_{-\infty}^y f_{X,Y}(u, v) dv du \right) \\ &= \frac{1}{f_X(x)} \left(\int_{-\infty}^y f_{X,Y}(x, v) dv \right), \end{aligned}$$

Which follows from fundamental theorem of calculus. We rewrite lefthand side using formula $f_{Y|X}(y|x) = \frac{f_{X,Y}(x, y)}{f_X(x)}$;

$$\begin{aligned} F_{Y|X}(v|x) &= \int_{-\infty}^v f_{Y|X}(v|x) dv \\ &= \int_{-\infty}^v \frac{f_{X,Y}(x, v)}{f_X(x)} dv \\ &= \frac{1}{f_X(x)} \left(\int_{-\infty}^v f_{X,Y}(x, v) dv \right), \end{aligned}$$

and we see that both sides are the same. The same obviously holds for our case of $x > 0$ and $y > 0$. We calculate what is needed:

$$\frac{\partial}{\partial x} F_{X,Y}(x, y) = \exp\left(-\frac{1}{x} - \frac{1}{y} - \frac{1}{xy}\right) \cdot \left(\frac{1}{x^2} + \frac{1}{x^2 y}\right) \quad \text{for } x > 0, y > 0$$

$$f_X(x) = \frac{\partial F_X(x)}{\partial x} = \frac{1}{x^2} \cdot \exp\left(-\frac{1}{x}\right) \quad \text{for } x > 0.$$

Using previous formula conditional cdf is

$$\begin{aligned} F_{Y|X}(y|x) &= \frac{1}{f_X(x)} \cdot \frac{\partial}{\partial x} F_{X,Y}(x, y) = \\ &= \frac{1}{\frac{1}{x^2} \exp\left(-\frac{1}{x}\right)} \cdot \exp\left(-\frac{1}{x} - \frac{1}{y} - \frac{1}{xy}\right) \cdot \left(\frac{1}{x^2} + \frac{1}{x^2 y}\right) \\ &= \exp\left(-\frac{1}{y} - \frac{1}{xy}\right) \cdot \left(1 + \frac{1}{y}\right). \end{aligned}$$

From this, after another derivation it quickly follows that

$$\begin{aligned} f_{Y|X}(y|x) &= \frac{\partial}{\partial y} F_{Y|X}(y|x) \\ &= \exp\left(-\frac{1}{y} - \frac{1}{xy}\right) \cdot \left(\frac{1}{xy^2} + \frac{1}{y^3} + \frac{1}{xy^3}\right) \end{aligned}$$

We also compute density function $f_{X,Y}(x, y)$ as:

$$\begin{aligned} f_{X,Y}(x, y) &= \frac{\partial^2}{\partial x \partial y} F_{X,Y}(x, y) \\ &= \exp\left(-\frac{1}{x} - \frac{1}{y} - \frac{1}{xy}\right) \cdot \left(\frac{1}{x^2 y^3} + \frac{1}{x^3 y^2} + \frac{1}{x^3 y^3}\right). \end{aligned}$$

Recall that density of inverse Gamma distributed random variable is defined as $f(y, \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} y^{-\alpha-1} \exp\left(-\frac{\beta}{y}\right)$ for $x > 0, \alpha > 0, \beta > 0$. From density $f_{Y|X}(y|x)$ we recognize that $Y|X = x$ is a mixture of two independent random variables that have inverse Gamma distribution. By rewriting the expression, we recognize that $\beta = \frac{x+1}{x}$, $w_1 = \frac{1}{1+x}$, $w_2 = \frac{x}{1+x}$, $\alpha_1 = 1$ and $\alpha_2 = 2$ and the density is then written as:

$$\begin{aligned} f_{Y|X}(y|x) &= \exp\left(-\frac{1}{y} - \frac{1}{xy}\right) \cdot \left(\frac{1}{xy^2} + \frac{1}{y^3} + \frac{1}{xy^3}\right) \\ &= \frac{1}{x+1} \cdot \frac{\left(\frac{x+1}{x}\right)^1}{\Gamma(1)} y^{-2} \cdot \exp\left(-\frac{1+1/x}{y}\right) \\ &\quad + \frac{x}{x+1} \cdot \frac{\left(\frac{x+1}{x}\right)^2}{\Gamma(2)} y^{-3} \cdot \exp\left(-\frac{1+1/x}{y}\right), \end{aligned}$$

Therefore, we can write conditional cdf as

$$Y|X = x \sim \frac{1}{x+1} \cdot IG\left(1, \frac{x+1}{x}\right) + \frac{x}{1+x} \cdot IG\left(2, \frac{x+1}{x}\right).$$

We can now simulate data from joint distribution of X and Y by first simulating X. This is done by inverse method via random variable $U \sim \text{unif}(0, 1)$ and as $U = F_X(x) = 1 - \exp(-1/x)$, we have $X = -\frac{1}{\ln(U)}$. Next we simulate $Y|X = x$ as a mixture of two inverse Gamma distributions. First, we decide to define a new Bernoulli random variable Z with parameter $p = \frac{x}{x+1}$; if $Z = 1$ we simulate Y from $IG\left(1, \frac{x+1}{x}\right)$ and if $Z = 0$ we simulate Y from $IG\left(2, \frac{x+1}{x}\right)$. Implemented algorithm is shown in R code below.

Point b)

We therefore simulate our empirical cdf using the algorithm below, written in R:

```
n_samples <- 50000
set.seed(123)

given_fXY <- function(x, y) {
  exp(-1/x - 1/y - 1/(x*y)) * (1/(x^2 * y^3) + 1/(x^3 * y^2) + 1/(x^3 * y^3))
}

u <- runif(n_samples)
x_sim <- -1 / log(u)

y_sim <- numeric(n_samples)
for (i in 1:n_samples) {
  x <- x_sim[i]
  beta <- (x + 1) / x
  if (runif(1) < (x / (x + 1))) {
    y_sim[i] <- 1/rgamma(1, shape = 2, rate = beta)
  } else {
    y_sim[i] <- 1/rgamma(1, shape = 1, rate = beta)
  }
}
```

We provide sanity check by plotting joint density $f_{X,Y}(x, y)$ and 100000 samples generated by a previous algorithm.

```
install.packages("plot3D")
library(plot3D)
grid_limit <- 5

grid_seq <- seq(0.1, grid_limit, length.out = 50)
z_theory <- outer(grid_seq, grid_seq, given_fXY)
```

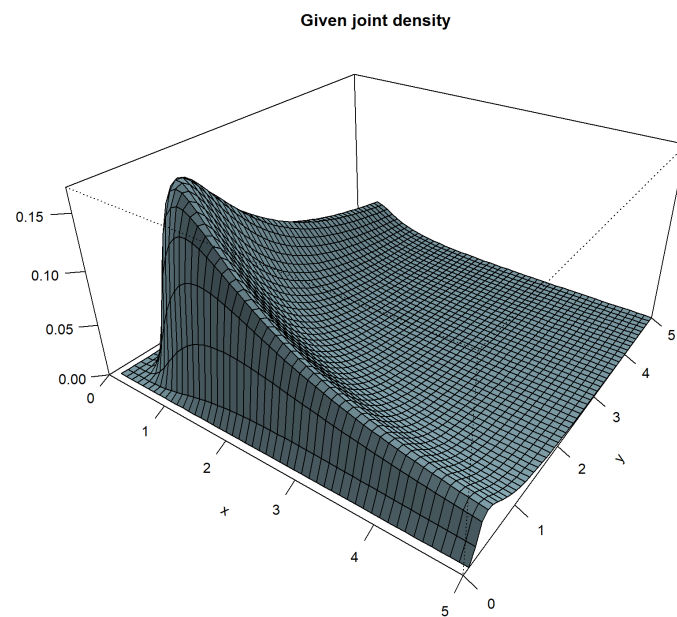
```

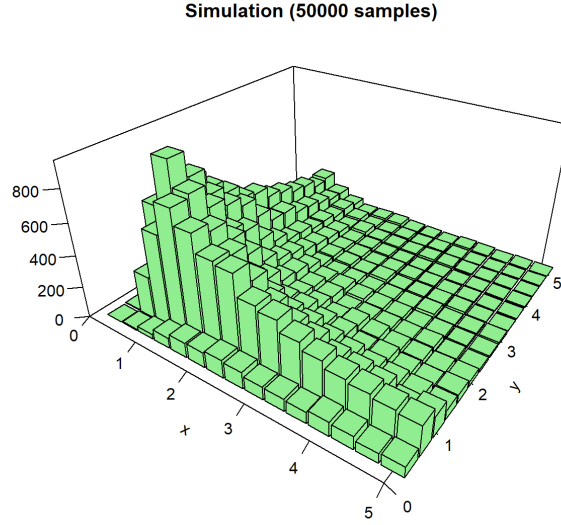
persp(grid_seq, grid_seq, z_theory,
      theta = 35, phi = 30, expand = 0.5,
      col = "lightblue", shade = 0.5,
      main = "Given joint density",
      xlab = "x", ylab = "y", zlab = "",
      xlim = c(0,5), ylim = c(0,5),
      ticktype = "detailed", nticks = 5)

num_bins <- 15
x_cuts <- seq(0, grid_limit, length.out = num_bins + 1)
y_cuts <- seq(0, grid_limit, length.out = num_bins + 1)
z_counts <- table(cut(x_sim, x_cuts), cut(y_sim, y_cuts))

hist3D(z = as.matrix(z_counts), x = x_cuts[-1], y = y_cuts[-1],
      theta = 35, phi = 30, expand = 0.5,
      col = "lightgreen", border = "black",
      space = 0.1,
      main = paste("Simulation (50000 samples)"),
      xlab = "x", ylab = "y", zlab = "",
      xlim = c(0,5), ylim = c(0,5),
      ticktype = "detailed", nticks = 5)

```





On first graph, z axis represent probability and on second one number of samples from which probability can be computed as well. We are satisfied with the proposed algorithm as the graphs look similar.

Problem 6

Point a)

In this problem, we want simulate data from distribution of random variable Y with density which omits the normalizing constant:

$$f^*(y) = \begin{cases} \left(\sqrt{y^2 + 4} - y\right)^2 & y \geq 0, \\ 0 & y < 0. \end{cases}$$

We decide to use a ratio - of - uniforms method. We first show that both $f^*(y)$ and $y^2 f^*(y)$ are bounded. We first rewrite

$$\sqrt{y^2 + 4} - y = \frac{4}{\sqrt{y^2 + 4} + y}$$

and therefore

$$f^*(y) = \frac{16}{(\sqrt{y^2 + 4} + y)^2} \quad \text{for } y \geq 0.$$

We see that the function $f^*(y)$ is bounded with value 4 in $y = 0$ and in limit, function behaves as $f^*(y) \sim \frac{4}{y^2}$ and $\lim_{y \rightarrow \infty} f^*(y) = 0$. We also analyze the function $y^2 f^*(y)$:

$$y^2 f^*(y) = y^2 \cdot \left(\frac{4}{\sqrt{y^2 + 4} + y} \right)^2 = \frac{16y^2}{y^2 + 4 + 2y\sqrt{y^2 + 4}} \quad \text{for } y \geq 0.$$

From behaviour of previous expression in the limit we quickly see that

$$\lim_{y \rightarrow \infty} y^2 f^*(y) = 4.$$

As it is also bounded in $y = 0$ with value 0, we conclude that $y^2 f^*(y)$ is also bounded. The conditions are therefore satisfied.

We continue with simulating $(X_1, X_2) \sim Unif(C)$, where

$$C = \{(x_1, x_2) \in \mathbb{R}^2 : 0 < x_1 \leq \sqrt{f^*(x_2/x_1)}, \quad 0 < x_2 \leq x_1 \sqrt{f^*(x_2/x_1)}\}.$$

We have to emphasize that we accept if $X_1 \leq f^*(X_2/x_1)$ which also implies that there will be some rejections and rejection probability to calculate.

We calculate bounds of set $C \subset [0, a] \times [b^-, b^+]$, where

$$\begin{aligned} a &= \sup_{y \geq 0} \sqrt{f^*(y)} = \sqrt{4} = 2, \\ b^- &= -\sqrt{\sup_{y \leq 0} y^2 f^*(y)} = 0 \\ b^+ &= \sqrt{\sup_{y \geq 0} y^2 f^*(y)} = \sqrt{4} = 2. \end{aligned}$$

Proposed algorithm goes as follows. We first sample $X_1 \sim Unif(0, 2)$ and $X_2 \sim Unif(0, 2)$, accept if $X_1^2 \leq f^*(X_2/X_1)$ and for accepted (X_1, X_2) set $Y = X_2/X_1$ which is our simulated data from desired distribution. We plot 10000 samples of normalized $f(\frac{x_2}{x_1})$.

Before moving onto the R code we have to compute normalization constant for $f^*(y)$ so it integrates to 1. By integration we get that

$$\int_0^\infty \left(\sqrt{y^2 + 4} - y \right)^2 dy = \frac{16}{3},$$

so the constant is $\frac{3}{8}$ and our density function we want to plot is:

$$f(x) = \frac{3}{16} \left(\sqrt{y^2 + 4} - y \right)^2.$$

Point b)

We implement algorithm described above in R:

```
set.seed(42)
n_sim <- 10000
total_proposals <- 0

f_star <- function(y) {
  ifelse(y >= 0, (sqrt(y^2 + 4) - y)^2, 0)
}

simulate_y <- function(n) {
  y <- numeric(0)
  while (length(y) < n) {
    m <- max(1000, n - length(y))
    total_proposals <- total_proposals + m
    u <- runif(m, 0, 2)
    v <- runif(m, 0, 2)
    y_prop <- v / u
    accept <- (u^2 <= f_star(y_prop))
    y <- c(y, y_prop[accept])
  }
  return(y[1:n])
}

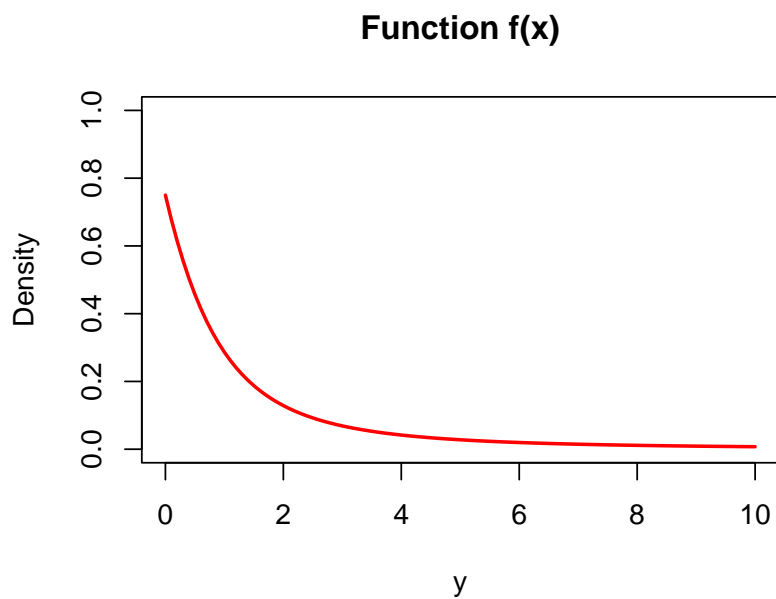
y_samples <- simulate_y(n_sim)
f_normalized <- function(y) (3/16) * f_star(y)

curve(f_normalized(x),
```

```

from = 0,
to = 10,
col = "red",
lwd = 2,
main = "Function f(x)",
xlab = "y",
ylab = "Density",
ylim = c(0, 1))

```

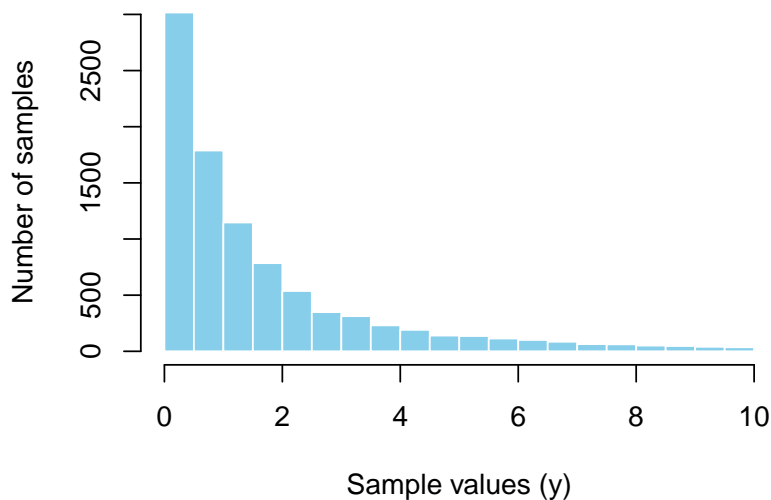


```

hist(y_samples[y_samples <= 10],
     breaks = seq(0, 10, by = 0.5),
     probability = FALSE,
     col = "skyblue",
     border = "white",
     main = "Simulation (10000 samples)",
     xlab = "Sample values (y)",
     ylab = "Number of samples",
     xlim = c(0, 10))

```

Simulation (10000 samples)



```
emp_prob <- n_sim / total_proposals
cat("Empirical Acceptance Probability:", emp_prob, "\n")

## Empirical Acceptance Probability: 0.6496459
```

We are satisfied with our result as it seems like simulation fits the density function. We limited ourselves to values from 0 to 10 as the probability of samples with higher numbers than that is quite low and the graph is more readable that way. We have calculated acceptance probability empirically since we counted number of total tries of sampling and number of accepted samples and got the probability of 0.65 percent.

Problem 7

We have a normal distributed random variable $X \sim N(\mu, \sigma^2)$ and logit-normal distributed random variable $Y = \frac{1}{1+e^{-X}}$.

Point a)

We write $Y = h(X) = \frac{1}{1+e^{-X}}$ and compute a Monte-Carlo estimate of $E(h(X))$ based on $n = 1000$ realizations x_1, \dots, x_n of X by i.i.d. simulating $x_1, \dots, x_n \sim f(x)$, where $f(x)$ is density of our normal distribution, and then using formula:

$$\widehat{E(h(X))} = \frac{1}{n} \sum_{i=1}^n h(x_i).$$

We use R code:

```
set.seed(123)
n <- 1000
mu <- 1
sigma <- 1
h <- function(x) 1 / (1 + exp(-x))
x <- rnorm(n, mean = mu, sd = sigma)
h_x <- h(x)
mc_estimate <- mean(h_x)
se_mc <- sd(h_x) / sqrt(n)
cat("MC Estimate:", mc_estimate, "\nSE:", se_mc)
```

```
## MC Estimate: 0.6996887
## SE: 0.005696
```

Our estimated $E(h(X))$ is 0.6996 with standard error of 0.005696.

Point b)

We produce additional n samples x_1^*, \dots, x_n^* by antithetic sampling and computed the wanted estimate as proposed by R code:

```
z <- rnorm(n)
x <- mu + sigma * z
x_star <- mu - sigma * z
h_combined <- (h(x) + h(x_star)) / 2
antithetic_estimate <- mean(h_combined)
se_anti <- sd(h_combined) / sqrt(n)
correlation <- cor(h(x), h(x_star))
cat("Antithetic Estimate:", antithetic_estimate, "\nSE:", se_anti, "\nCorr:", correlation)

## Antithetic Estimate: 0.6960051
## SE: 0.00121636
## Corr: -0.9129796
```

Our estimated $E(h(X))$ is again 0.6960 but with smaller standard error of 0.001216. We compute the standard error by treating each pair as a single independent observation \hat{Y}_i , so $\hat{Y}_i = \frac{h(x_i) + h(x_i^*)}{2}$. Since the pairs are independent, we can apply the standard se formula to these n averages: $se = \frac{sd(\hat{Y})}{\sqrt{n}}$. We also know that both $h(x_i)$ and $h(x_i^*)$ have same variance and all samples are i.i.d. We therefore compute variance of estimator as

$$\begin{aligned} Var(\hat{\mu}_b) &= \frac{Var(h(X) + h(X^*))}{4n} \\ &= \frac{1}{2n} (Var(h(X)) + Cov(h(X), h(X^*))) \\ &= \frac{Var(h(X))}{2n} (1 + Corr(h(X), h(X^*))). \end{aligned}$$

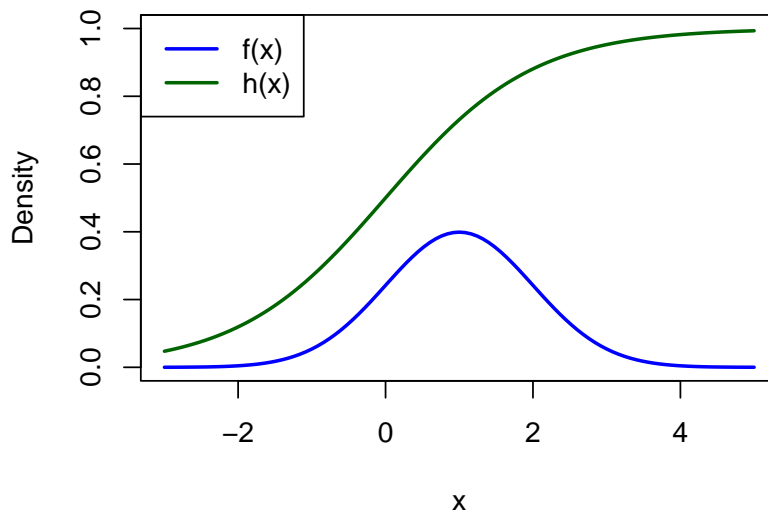
We get standard deviation as a square root of the variance. We computed that $Corr(h(X), h(X^*)) = -0.91$, therefore less than zero which explains why the standard error is lower than in first part of the problem.

Point c)

We first need to set the parameters of normal distribution that will help us with importance sampling. First, we plot $f(x)$ and $h(x)$ on graph below:

```
f <- function(x) dnorm(x, 1, 1)
h <- function(x) 1 / (1 + exp(-x))
curve(f(x), -3, 5, col="blue", lwd=2, ylab="Density", main="Functions f(x) and h(x)", ylim = c(0, 1))
curve(h(x), -3, 5, col="darkgreen", lwd=2, add=TRUE)
legend("topleft", legend=c("f(x)", "h(x)"),
      col=c("blue", "darkgreen"), lwd=2)
```

Functions $f(x)$ and $h(x)$

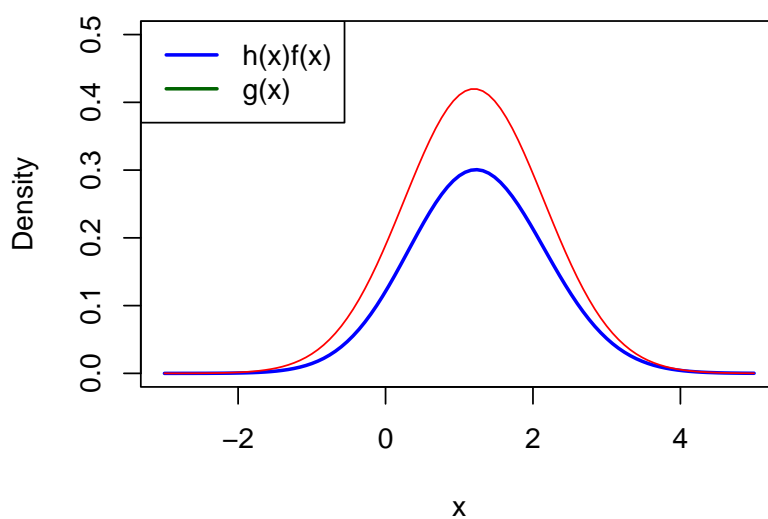


Now, we will plot function $f(x) * h(x)$ and then try different parameters for normally distributed $g(x)$ to find one that suits it the most.

```
target_prod <- function(x) f(x) * h(x)
curve(target_prod, -3, 5, col="blue", lwd=2, ylab="Density", main="Choosing g(x)", ylim = c(0, 0.5))

g_mu <- 1.2
g_sigma <- 0.95
curve(dnorm(x, g_mu, g_sigma), -3, 5, col="red", add=TRUE)
legend("topleft", legend=c("h(x)f(x)", "g(x)"),
      col=c("blue", "darkgreen"), lwd=2)
```

Choosing $g(x)$



On graph, we see the proposed $g(x)$ we are going to use for importance sampling as it seemed to give the best results, also in comparison to results in part e. We implement in R importance sampling algorithm

as we did in lectures, simulating i.i.d. $x_1, x_2, \dots, x_n \sim g(x)$ and computing

$$E(\widehat{h(X)}) = \frac{1}{n} \sum_{i=1}^n h(x_i) \frac{f(x_i)}{g(x)_i}.$$

```
set.seed(123)
n <- 1000
x_c <- rnorm(n, g_mu, g_sigma)
w_c <- (h(x_c) * f(x_c)) / dnorm(x_c, g_mu, g_sigma)
is_estimate <- mean(w_c)
se_is <- sd(w_c)/sqrt(n)
cat("Importance Sampling Estimate:", is_estimate, "\nSE:", se_is, "\n")

## Importance Sampling Estimate: 0.6978581
## SE: 0.001442259
```

We see that the estimation of mean stayed accurate and standard error got just a bit worse compared to anthetic sampling method.

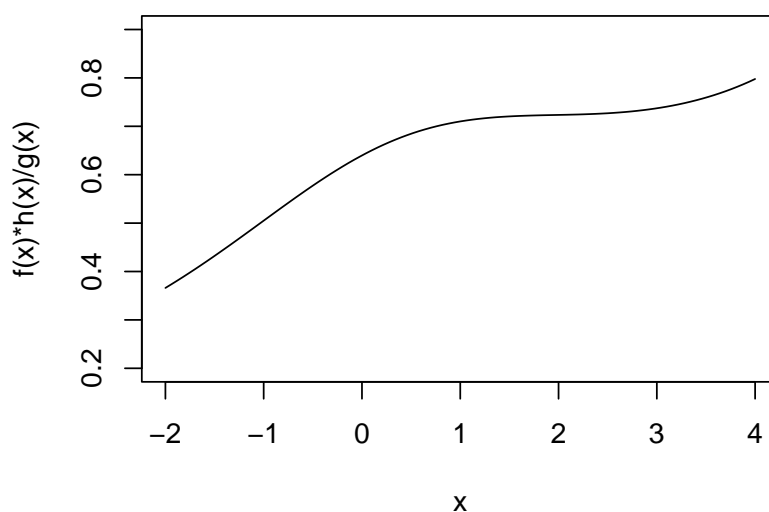
Point d)

Now we combine both previously used method, same importance sampling as in part c but just with $2n$ samples as in part b:

```
z <- rnorm(n/2)
y1 <- g_mu + g_sigma * z
y2 <- g_mu - g_sigma * z
w1 <- target_prod(y1) / dnorm(y1, g_mu, g_sigma)
w2 <- target_prod(y2) / dnorm(y2, g_mu, g_sigma)
combined_samples <- (w1 + w2) / 2
combined_estimate <- mean(combined_samples)
se_combined <- sd(combined_samples) / sqrt(n/2)

combined_function <- function(x) target_prod(x) / dnorm(x, g_mu, g_sigma)
curve(combined_function, -2, 4, main="Combined function", ylab="f(x)*h(x)/g(x)", xlab="x", ylim = c(0.2, 0.8))
```

Combined function



```
cat("Combined Estimate:", combined_estimate, "\nSE:", se_combined, "\n")

## Combined Estimate: 0.6970523
## SE: 0.0009958441
```

On graph, we see the function $\frac{f(x)h(x)}{g(x)}$. It is monotone which is wanted because then the samples of antithetic pair will be negatively correlated which will lower overall variance. This showed up to be correct as we see that while estimation of mean stayed the same, standard error got lower.

Point e)

```
true_val <- integrate(target_prod, lower = -Inf, upper = Inf)$value

results <- data.frame(
  Method = c("Monte Carlo (a)", "Antithetic (b)", "Importance (c)", "Combined (d)", "Numerical value"),
  Estimate = c(mc_estimate, antithetic_estimate, is_estimate, combined_estimate, true_val),
  Std_Error = c(se_mc, se_anti, se_is, se_combined, 0)
)
print(results)
```

##	Method	Estimate	Std_Error
## 1	Monte Carlo (a)	0.6996887	0.0056959998
## 2	Antithetic (b)	0.6960051	0.0012163598
## 3	Importance (c)	0.6978581	0.0014422595
## 4	Combined (d)	0.6970523	0.0009958441
## 5	Numerical value (e)	0.6967347	0.0000000000

In the table above, we see that all methods returned similar and accurate estimates for mean. The estimator with smallest standard error seems to be the one, derived using both antithetic and importance sampling.

Problem 8

Point a)

We used Cholesky factor decomposition of the variance matrix.

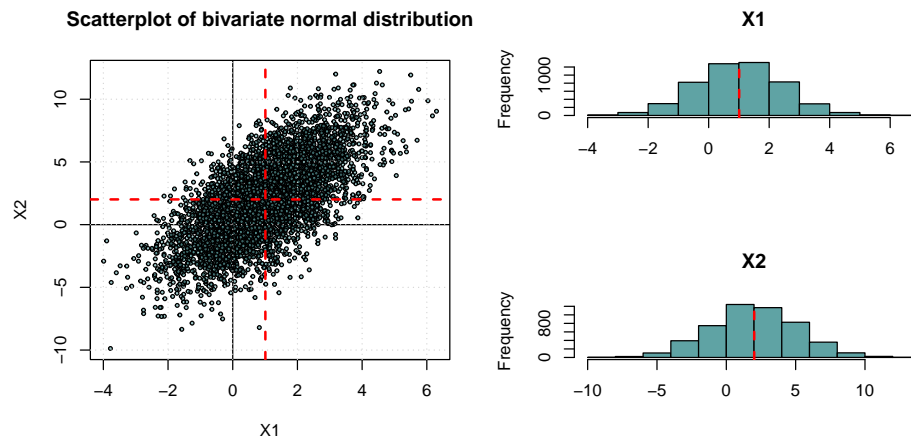
```
mvnchol <- function(size, n, mu, S) {
  L <- t(chol(S)) # compute Cholesky lower triangular
  u <- runif(size*n,0,1)
  theta <- runif(size*n,0,2*pi)
  r <- sqrt(-2*log(u))
  Z <- matrix(r*cos(theta), nrow = size, ncol = n)

  X <- L %*% Z + matrix(mu, nrow = size, ncol = n)

  return(X)
}

# Parameters
size <- 2
n <- 5000
mu <- c(1, 2)
S <- matrix(
  c(2, 3,
    3, 10),
  nrow = 2, byrow = TRUE)
```

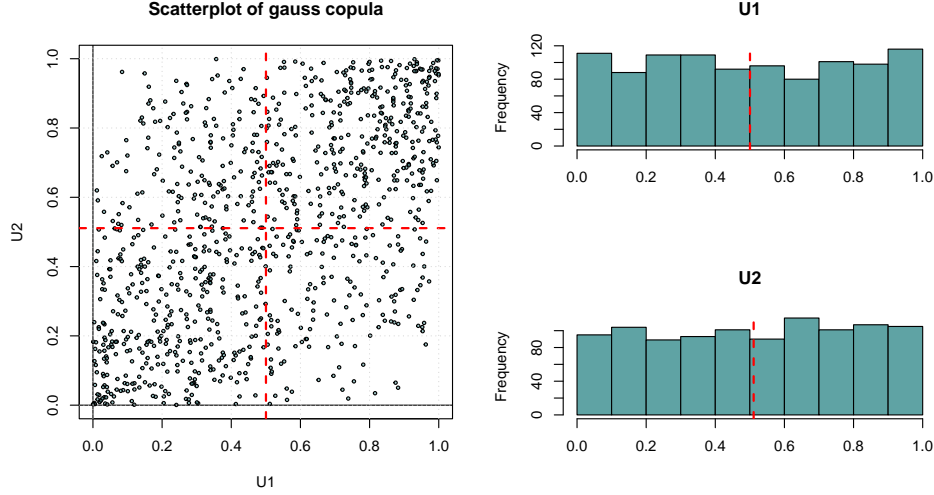
```
X <- mvnchol(size, n, mu, S)
plotMarginal2D(X, scattermain = "Scatterplot of bivariate normal distribution",
  scatterxlab = "X1",
  scatterylab = "X2")
```



Point b)

```
gausscopula <- function(n, std = c(1,1), mu = c(0,0), corr){
  cov <- corr*std[1]*std[2]
  S <- matrix(
    c(std[1]^2, cov,
      cov, std[2]^2),
    nrow = 2,
    byrow = TRUE
  )
  X <- mvnchol(2, n, mu, S)
  U <- pnorm(X)
  return(U)
}

truecorr <- 0.5 # correlation
n <- 1000
U <- gausscopula(n = n, corr = truecorr)
plotMarginal2D(U, scattermain = "Scatterplot of gauss copula",
  scatterxlab = "U1",
  scatterylab = "U2")
```



Since $X_1, X_2 \sim \mathcal{N}(\mu, \mathbf{S})$, $U_1 = \Phi(X_1)$ and $U_2 = \Phi(X_2)$ are uniformly distributed, as we can see in the marginal histograms above. Moreover, they are dependent, which is evident from the scatterplot: the points tend to accumulate along the diagonal from (0,0) to (1,1), reflecting the positive correlation of the original Gaussian variables.

Monte Carlo estimation of correlation:

```
gammamc <- function(U){
  n <- ncol(U)
  E1 <- mean(U[1,])
  E2 <- mean(U[2,])
  E12 <- mean(U[1,]*U[2,])
  E11 <- mean(U[1,]^2)
  E22 <- mean(U[2,]^2)
  cov12 <- E12 - E1*E2 # covariance
  var1 <- E11 - E1^2
  var2 <- E22 - E2^2
  gamma <- cov12 / sqrt(var1*var2)
  return(gamma)
}

print(gammamc(U))

## [1] 0.5141008
```

Point c)

Given

$$(X_1, X_2) \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right) \quad \rho \in [-1, 1] \quad (10)$$

define the trasnformed variables $Y_1 = \Phi(X_1)$, $Y_2 = \Phi(X_2)$.

$$Y_1, Y_2 \sim \text{Uniform}(0, 1) \implies \mathbb{E}[Y_i] = \frac{1}{2}, \quad \text{Var}(Y_i) = \frac{1}{12} \quad (11)$$

Considering that $\Phi(x) = \mathbb{P}(Z \leq x)$, $Z \sim \mathcal{N}(0, 1)$, then

$$\mathbb{E}[\Phi(X_1)\Phi(X_2)] = \mathbb{P}(Z_1 \leq X_2, Z_2 \leq X_2) \quad (12)$$

Defining

$$K_1 = X_1 - Z_1, \quad K_2 = X_2 - Z_2 \quad (13)$$

Then

$$\mathbb{P}(Z_1 \leq X_2, Z_2 \leq X_2) = \mathbb{P}(K_1 \geq 0, K_2 \geq 0) \quad (14)$$

Since (X_1, X_2) is gaussian and (Z_1, Z_2) is independent gaussian,

$$(K_1, K_2) \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 & \rho \\ \rho & 2 \end{pmatrix}\right) \quad (15)$$

The correlation of (K_1, K_2) is therefore

$$\text{Corr}(K_1, K_2) = \frac{\rho}{2} \quad (16)$$

For a centered bivariate normal (U, V) with correlation r ,

$$\mathbb{P}(U \geq 0, V \geq 0) = \frac{1}{4} + \frac{1}{2\pi} \arcsin(r) \quad (17)$$

Setting $r = \frac{\rho}{2}$

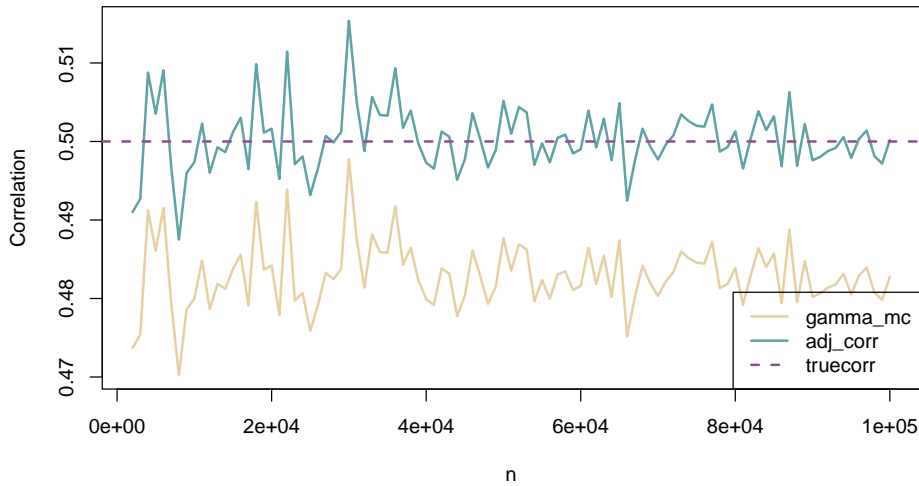
$$\mathbb{E}[\Phi(X_1)\Phi(X_2)] = \mathbb{E}[Y_1Y_2] = \frac{1}{4} + \frac{1}{2\pi} \arcsin\left(\frac{\rho}{2}\right) \quad (18)$$

$$\text{Cov}(Y_1, Y_2) = \mathbb{E}[Y_1Y_2] - \mathbb{E}[Y_1]\mathbb{E}[Y_2] = \frac{1}{2\pi} \arcsin\left(\frac{\rho}{2}\right) \quad (19)$$

$$\gamma = \text{Corr}(Y_1, Y_2) = \frac{\text{Cov}(Y_1, Y_2)}{1/12} = \frac{6}{\pi} \arcsin\left(\frac{\rho}{2}\right) \quad (20)$$

$$\implies \rho = 2 \sin\left(\frac{\pi}{6}\gamma\right) \quad (21)$$

```
n_seq <- seq(2000, 100000, by = 1000)
gamma_mc <- numeric(length(n_seq))
adj_corr <- numeric(length(n_seq))
for(i in seq_along(n_seq)){
  n <- n_seq[i]
  U <- gausscopula(n = n, corr = truecorr)
  gamma_mc[i] <- gammamc(U)
  adj_corr[i] <- 2*sin(pi*gamma_mc[i]/6) # relation (21) in the text
}
adjustedcorrplot(n_seq, gamma_mc, adj_corr)
```

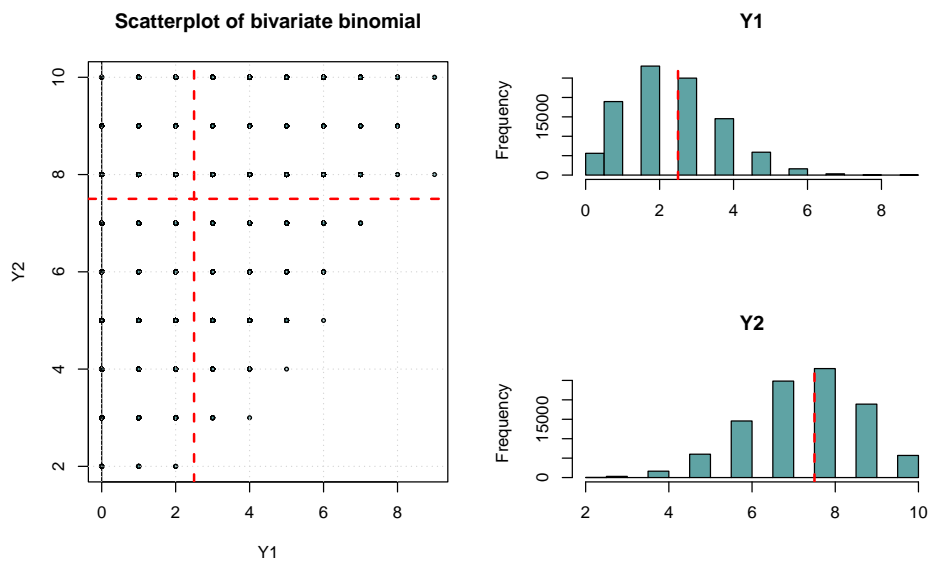


The plot above confirms the correctness of our results: as n increases, the adjusted correlation ρ approaches the true value more closely than the Monte Carlo estimate γ .

Point d)

```
U <- gausscopula(n=100000,corr=truecorr)
Y <- rbind(
  qbinom(U[1, ], 10, 0.25),
  qbinom(U[2, ], 10, 0.75)
)

plotMarginal2D(Y, scattermain = "Scatterplot of bivariate binomial",
  scatterxlab = "Y1",
  scatterylab = "Y2")
```



```
print(gammamc(Y))
## [1] 0.4739484
```

The marginal distributions are just the binomial distributions.
Since U_1 and U_2 are not independent, also Y_1 and Y_2 are not independent. That is also confirmed by the estimated correlation $\sim 0.4704 > 0$.

Plot Functions

```
primary <- "#88498f"
secondary <- "#779fa1"
tertiary <- "#e0cba8"
quaternary <- "#ff6542"
quinary <- "#564154"
```

```
plotcauchyzoom <- function(x, primary = "blue", secondary = "lightgray") {
  # Exclude extreme outliers for visualization
  hist(
    x[x > -10 & x < 10],
    breaks = seq(-10, 10, length.out = 100),
    probability = TRUE,
    main = "Cauchy Samples (Zoomed) vs Theoretical Density",
```

```

    xlab = "x",
    col = secondary,
    border = "white"
)

# Overlay theoretical density
curve(
  dcauchy(x),
  from = -10, to = 10,
  col = primary, lwd = 2, add = TRUE
)

# Add grid
grid(lty = 2, col = "gray", lwd = 1)
}

plotcumulativemean <- function(cummean){
  plot(1:length(x), cummean(x), type="l",
       main="Cumulative mean vs n",
       col = primary,
       lwd = 2,
       xlab="n",
       ylab=expression(bar(X)[1/n]))
  grid(lty = 2, col = "gray", lwd = 1)
  x.mean <- mean(x)
  abline(h = x.mean, col = secondary, lwd = 2, lty = 2)
  text(x = length(x)*0.4,
       y = x.mean,
       labels = paste("Global mean =", round(x.mean, 2)),
       pos = 1,
       col = secondary,
       cex = 0.8)
}

rtrunchistbinom <- function(x){
  hist(
    x,
    breaks = seq(-0.5, 10.5, by = 1),
    prob = TRUE,
    col = secondary,
    border = "white",
    main = "Truncated Binomial(20, 0.5), b = 10",
    xlab = "x"
  )
  k <- 0:10

  pmf.trunc <- dbinom(k, size = 20, prob = 0.5) /
    pbinom(10, size = 20, prob = 0.5)
  lines(k, pmf.trunc, col = primary, lwd = 2)
}

rtrunchistnorm <- function(x){
  hist(x, breaks = 30, prob = TRUE, col = secondary, border = "white",
       main = "Truncated Normal(0,1), [-1,2]", xlab = "x")
  x.grid <- seq(-1, 2, length.out = 200)
  dens.trunc <- dnorm(x.grid, mean = 0, sd = 1) /
    (pnorm(2, mean = 0, sd = 1) - pnorm(-1, mean = 0, sd = 1))

```

```

    lines(x.grid, dens.trunc, col = primary, lwd = 2)
}

rtrunchistexp <- function(x) {
  hist(x, breaks = 30, prob = TRUE, col = secondary, border = "white",
    main = "Truncated Exponential(rate = 1), [0.5,3]", xlab = "x")

  x.grid <- seq(0.5, 3, length.out = 200)
  dens.trunc <- dexp(x.grid, rate = 1) /
    (pexp(3, rate = 1) - pexp(0.5, rate = 1))
  lines(x.grid, dens.trunc, col = primary, lwd = 2)
}

scatterplotdisk <- function(samples, rho, delta){
  # Disk (support) boundary
  t <- seq(0, 2 * pi, length.out = 400)
  x.circle <- delta + rho * cos(t)
  y.circle <- rho * sin(t)

  plot(samples[, 1], samples[, 2], xlab = "x1", ylab = "x2",
    main = "Scatter plot of bnormsampler samples", pch = 19, col = secondary,
    xlim = c(delta - rho, delta + rho), ylim = c(-rho, rho), asp = 1)
  lines(x.circle, y.circle, col = primary, lwd = 2)
  abline(h = 0, v = 0, col = "black", lwd = 1)
  abline(v = delta, col = "red", lwd = 2, lty = 2)

  grid()
}

gbnormamplerploth <- function(samples, mean1, mean2, rho, c1, c2){
  t <- seq(0, 2 * pi, length.out = 400)
  x.circle <- c1 + rho * cos(t)
  y.circle <- c2 + rho * sin(t)
  plot(samples[, 1], samples[, 2], xlab = "x1", ylab = "x2",
    main = paste0("Truncated bivariate normal",
    mean="(",mean1,",",mean2,")", c="(",c1,",",c2,")"),
    pch = 19, col = secondary, xlim = c(-5,8), ylim = c(-5,8),asp = 1)
  abline(h = 0, v = 0, col = "black", lwd = 1)
  lines(x.circle, y.circle, col = primary, lwd = 2)
  points(mean1, mean2, pch = 4, col = primary, lwd = 2)
  grid()
}

plotMarginal2D <- function(X,
  scattermain = "Scatter plot of gauss copula",
  scatterxlab = "U1",
  scatterylab = "U2",
  histcol = secondary,
  pointcol = "black",
  bgcol = adjustcolor(secondary, alpha.f = 0.6),
  cex = 0.5) {

  # Compute means
  meanX1 <- mean(X[,1])
  meanX2 <- mean(X[,2])

  # Layout: scatter plot left, histograms right

```

```

layout(matrix(c(1,2,
                1,3),
                nrow = 2, byrow = TRUE))

# Scatter plot
plot(X[1,], X[2,],
     pch = 21,
     bg = bgcol,
     col = pointcol,
     cex = cex,
     xlab = scatterxlab,
     ylab = scatterylab,
     main = scattermain)

abline(h = 0, v = 0, col = "black", lwd = 1)           # origin lines
abline(h = meanX2, v = meanX1, col = "red", lwd = 2, lty = 2) # dotted mean lines
grid()

# Histograms with mean indication
hist(X[1,], main = scatterxlab, xlab = "", col = histcol)
abline(v = meanX1, col = "red", lwd = 2, lty = 2) # dotted

hist(X[2,], main = scatterylab, xlab = "", col = histcol)
abline(v = meanX2, col = "red", lwd = 2, lty = 2) # dotted
}

adjustedcorrplot <- function(n_seq, gamma_mc, adj_corr){
  plot(n_seq, gamma_mc, type = "l", col = tertiary, lwd = 2,
       xlab = "n", ylab = "Correlation", ylim = range(c(gamma_mc, adj_corr, truecorr)))
  lines(n_seq, adj_corr, col = secondary, lwd = 2)
  abline(h = truecorr, col = primary, lty = 2, lwd = 2)

  legend("bottomright", legend = c("gamma_mc", "adj_corr", "truecorr"),
        col = c(tertiary, secondary, primary), lty = c(1,1,2), lwd = 2)
}

```