TMA4300 Project 1

Celine Olsson, Jakob Heide

feb 11, 2024

Part A - Stochastic simulation by the probability integral transform and bivariate techniques

The inversion method

We begin by implementing an R function to generate samples from a exponential distribution with rate λ. This will be done using the inversion method, which needs the inverse of the cumulative distribution function (cdf). Both the cdf and its inverse are calculated using the probability density function (pdf). The pdf of the exponential distribution is given by

$$f_X(x) = \lambda e^{-\lambda x}$$

The cumulative distribution function (cdf) is determined by calculating the following integral

$$F_X(x) = \int_{-\infty}^{x} f_X(t)\, dt \tag{1}$$

Given the exponential density function above, the cdf becomes $F_X(x) = 1 - e^{-\lambda x}$.
Next the inverse of the cumulative distribution function is found by defining $u = F_X(x)$ and solving for x. This gives the following inverse

$$u = F_X(x) = 1 - e^{-\lambda x} \quad \Rightarrow \quad x = \frac{1}{\lambda}\ln(1-u) = F_X^{-1}(u) \tag{2}$$

Since the cdf is the probability that $X \leq x$, it can only have a value between 0 and 1. By using the `runif()` function, n random numbers can be generated that will be uniformly distributed between 0 and 1. This vector of random numbers simulates samples from the cdf, $u = F_X(x)$. Then the inverse of the cdf is calculated to find the x-values. We call this implemented function `sample_exponential`, and it takes in the rate (λ) and the number of samples to be generated.

```r
#Sampling from an exponential distribution - using the inversion method
sample_exponential <- function(n,rate){
  u <- runif(n) #sample n numbers from unif[0,1]
  x <- -(1/rate)*log(u) #inverse cdf
  return(x)
}
```

Figure 1 shows $n = 10000$ generated numbers with $\lambda = 2$, along with the theoretical distribution.

```r
rate <- 2
n <- 10000

samples <- sample_exponential(n,rate)
hist(samples, freq=F, breaks = 100, main=NULL, xlab="x")
x <- seq(0,3.5,0.1)
lines(x, rate*exp(-rate*x), col="red")
legend("topright", legend=c("f(x)"), col=c("red"), lty = 1, cex = 0.8, lw=2, inset=0.1, box.lty=0)
```

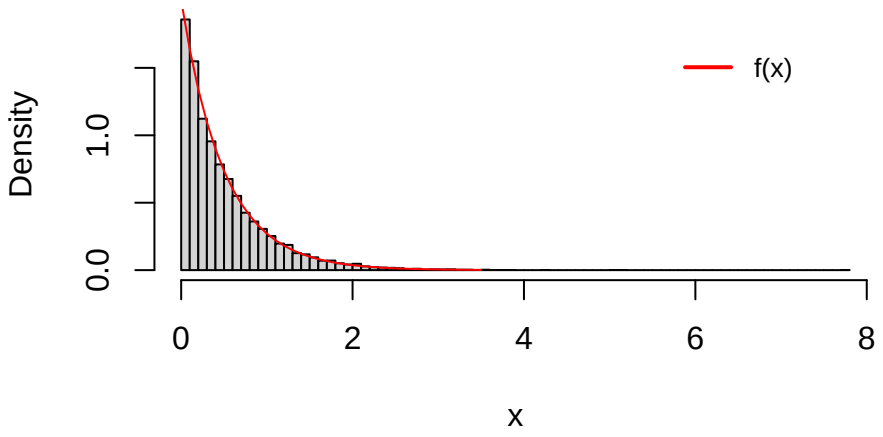

Figure 1: Samples from exponential distribution

The samples appear to agree well with the actual distribution. This can be checked by comparing the sample mean and variance to the theoretical mean and variance, where the theoretical values are calculated with $E(X) = 1/\lambda$ and $Var(X) = 1/\lambda^2$.

```
## Sample mean:  0.5052661
## Sample variance:  0.2643871
## 1/rate:  0.5
## 1/rate^2:  0.25
```

The mean and variance of the generated numbers match that of the exponential distribution, with only a minor difference.

Now a new probability density function is considered, namely

$$g(x) = \begin{cases} cx^{\alpha-1}, & 0 < x < 1 \\ ce^{-x} & x \geq 1 \\ 0, & \text{else} \end{cases} \tag{3}$$

where c is a normalizing constant and $\alpha \in (0, 1)$. A new R function is to be implemented to generate samples from this distribution. Again, the inversion method is used. The cdf is once more calculated by using equation (1), but in this case there are three different domains to consider. If $0 < x < 1$, then the integral becomes:

$$G_X(x) = \int_{-\infty}^{0} 0\, dt + \int_{0}^{x} ct^{\alpha-1}\, dt = c[t^\alpha/\alpha]_0^x = cx^\alpha/\alpha$$

When $x \geq 1$, the integral is:

$$G_X(x) = \int_{-\infty}^{0} 0\, dt + \int_{0}^{1} ct^{\alpha-1}\, dt + \int_{1}^{x} ce^{-t}\, dt = c[t^\alpha/\alpha]_0^1 + c[-e^{-t}]_1^x = c(1/\alpha + e^{-1} - e^{-x})$$

And with other values of x we just get zero as the cdf. All this gives the following cdf for the density function g

$$G_X(x) = \begin{cases} cx^\alpha/\alpha & ,0 < x < 1 \\ c(1/\alpha + e^{-1} - e^{-x}) & ,x \geq 1 \\ 0 & ,\text{else} \end{cases}$$

The inverse of the cdf is found the same way as before, by defining $u = G_X(x)$ and solving for x. Again, we get three different cases, where the first two are the following

$$u = \frac{cx^\alpha}{\alpha} \quad \Rightarrow \quad x = \left(\frac{\alpha u}{c}\right)^{1/\alpha} \tag{4}$$

$$u = c(1/\alpha + e^{-1} - e^{-x}) \quad \Rightarrow \quad x = -\ln(1/\alpha + 1/e - u/c), \tag{5}$$

and the last case where x is below zero, the inverse of the cdf is also zero. Now we need to find the domain for u. In the first case where $0 < x < 1$ the x is replaced with the one found in Equation (4), which gives

$$0 < \left(\frac{\alpha u}{c}\right)^{1/\alpha} < 1 \quad \Rightarrow \quad 0 < u < c/\alpha$$

For the second case where $x \geq 1$ Equation (5) is used instead:

$$-ln(1/\alpha + 1/e - u/c) \geq 1 \quad \Rightarrow \quad u \geq c/\alpha$$

If u has any other value than the first two cases, the inverse becomes $G_X^{-1}(u) = 0$. These inequalities only hold if $c > 0$, which we will prove is the case. Since c is a normalizing constant it makes the total probability of the probability density function equal to one. I.e. one should get

$$\int_{-\infty}^{\infty} g(x)\,dx = 1 \tag{6}$$

This gives

$$\begin{aligned} \int_{-\infty}^{\infty} g(x)\,dx &= \lim_{a\to\infty} \int_{-\infty}^{0} 0\,dx + \int_0^1 cx^{\alpha-1}\,dx + \int_1^a ce^{-x}\,dx \\ &= \lim_{a\to\infty} c(1/\alpha + 1/e - 1/e^a) = c(1/\alpha + 1/e) = 1, \end{aligned}$$

and for this to hold, the normalizing constant needs to be $c = \frac{\alpha e}{\alpha + e}$. Since α is non-negative, c must also be non-negative.

This gives the following inverse of the cdf:

$$G_X^{-1}(u) = \begin{cases} (\alpha u/\alpha)^{1/\alpha} & 0 < u < c/\alpha \\ -ln(1/\alpha + 1/e - u/c) & u \geq c/\alpha \\ 0 & \text{else} \end{cases}$$

Below is the R-function `sample_g` which generates samples from our density function g. The function takes in the number of samples to be generated and the constant α.

```
sample_g <- function(n,a){
  u <- runif(n)
  c <- a*exp(1) / (a + exp(1))
  ca <- c/a

  u1 <- u[u < ca]
  u2 <- u[u >= ca]

  x1 <- (a*u1/c)^(1/a)
  x2 <- -log(exp(-1) - u2/(c) + 1/(a))
  return (c(x1,x2))
}
```

To check the accuracy of the samples generated, a function for calculating the actual distribution of g is needed. This function is called g and can be seen below.

```
g <- function(x,a){
  x1 <- x[0 < x]
  x1 <- x1[x1 < 1]
  x2 <- x[1 <= x]
  c <- a*exp(1) / (a + exp(1))

  return (c(c*x1^(a-1),c*exp(-x2)))
}
```

In Figure 2, $n = 10000$ generated samples are then plotted with the actual distribution of g, using $\alpha = 0.5$.

```
set.seed(420)
samples <- sample_g(10000,0.5)
x <- seq(0,6,0.01)
x <- x[-1]

hist(samples,freq = FALSE, breaks = 100,main = NULL, xlab = "x")
lines(x,g(x,0.5),col = "red", lwd = 2)
legend("topright", legend=c("g(x)"), col=c("red"), lty = 1, cex = 0.8, lw=2, inset=0.1, box.lty=0)
```

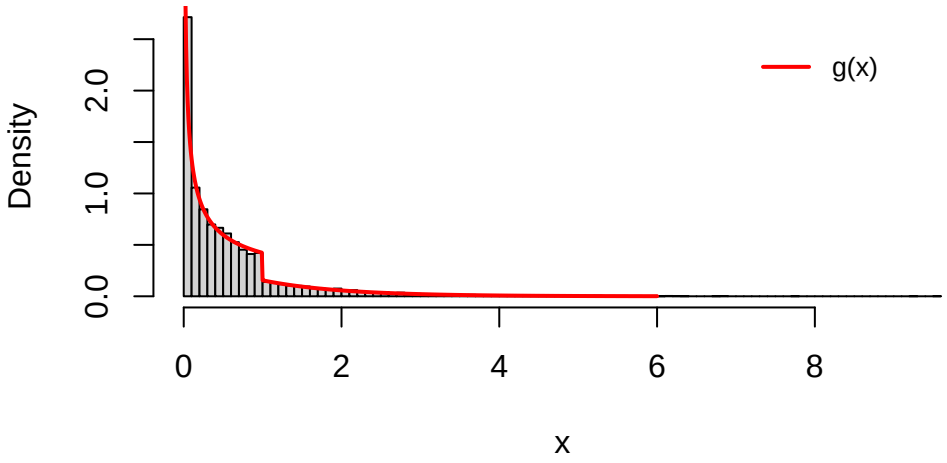

Figure 2: Samples from $g(x)$ distribution, using $\alpha = 0.5$

We see that the samples seem to match the distribution. This can be verified by looking at the sample mean and variance compared to the theoretical mean and variance. The theoretical mean and variance are determined by using

$$E[X] = \int_{-\infty}^{\infty} xg(x)\,dx \quad \text{and} \quad Var[X] = \int_{-\infty}^{\infty} (x-\mu)^2 g(x)\,dx \tag{7}$$

where $\mu = E[X]$. These are calculated by using the `integrate` function as shown below.

```r
g_mean <- function(x) {x*g(x,0.5)}
mean <- integrate(g_mean,0,Inf)$val

g_var <- function(x){(x-mean)^2 * g(x,0.5)}
var <- integrate(g_var,0,Inf)$val
```

We get the following values.

```
## Theoretical mean:  0.5922707
## Sample mean:  0.6048482

## Theoretical variance:  0.5949549
## Sample variance:  0.6487042
```

The difference between the theoretical and sample variance is 0.0537, while the difference between the means is just 0.0126.

We now consider a new probability density function

$$f(x) = \frac{ce^{\alpha x}}{(1+e^{\alpha x})^2}, \quad -\infty < x < \infty, \alpha > 0$$

where c is the normalizing constant. To find the value of c Equation (6) is used the same way as before.

$$\int_{x=-\infty}^{x=\infty} \frac{ce^{\alpha x}}{(1+e^{\alpha x})^2}\,dx = \frac{c}{\alpha}\int_{u=1}^{u=\infty} \frac{1}{u^2}\,du = \frac{c}{\alpha}\left[-\frac{1}{u}\right]_{u=1}^{u=\infty} = \frac{c}{\alpha} = 1$$

The substitution $u = 1 + e^{\alpha x}$ is used in the integral, making it much easier to solve. With this substitution the lower limit goes from ∞ to 1, while the upper limit stays the same. For the last part to be true the normalizing constant needs to be $c = \alpha$.

Once again, we will need the formulas for the cumulative distribution function and its inverse. Equation (1) is used to calculate the cdf F.

$$\begin{aligned} F_X(x) &= \int_{-\infty}^{x} f(x)\,dt = \int_{-\infty}^{x} \frac{\alpha e^{\alpha x}}{(1+e^{\alpha x})^2}\,dt = \int_{t=-\infty}^{t=x} \frac{1}{u^2}\,du \\ &= \left[-\frac{1}{u}\right]_{t=-\infty}^{t=x} = \left[-\frac{1}{1+e^{\alpha t}}\right]_{-\infty}^{x} = \frac{e^{\alpha x}}{1+e^{\alpha x}} \end{aligned}$$

Here $c = \alpha$ is used, as well as the same substitution as before, $u = 1 + e^{\alpha x}$.

By defining $u = F_X(x)$ and solving for x, the inverse of the cdf is found.

$$u = \frac{e^{\alpha x}}{1+e^{\alpha x}} \quad \Rightarrow \quad x = \frac{\ln(u/(1-u))}{\alpha} = F_X^{-1}(u) \tag{8}$$

Next the domain for u must also be defined. Using the definition of x in Equation (8) in the original domain $-\infty < x < \infty$ gives:

$$\begin{aligned} \lim_{a\to\infty} -a < \frac{\ln(u/(1-u))}{\alpha} \quad &\Rightarrow \quad \lim_{a\to\infty} \frac{e^{-a\alpha}}{1+e^{-a\alpha}} < u \quad \Rightarrow \quad 0 < u \\ \lim_{a\to\infty} a > \frac{\ln(u/(1-u))}{\alpha} \quad &\Rightarrow \quad \lim_{a\to\infty} \frac{1}{\frac{1}{e^{a\alpha}}+1} > u \quad \Rightarrow \quad 1 > u \end{aligned}$$

The inverse of the cdf is then defined by:

$$F_X^{-1}(u) = \frac{\ln(u/(1-u))}{\alpha}, \quad \text{for } 0 < u < 1$$

The inversion method is then used to make a R-function which generates samples from the pdf f. The function takes is the number of samples to be generated and the constant α.

```r
# Sampling from f(x) - using the inversion method
sample_f <- function(a, n){
  u <- runif(n)
  x <- log(u/(1-u)) / a
  return(x)
}

# Pdf
f <- function(x,a){
  return( a*exp(a*x) / (1+exp(a*x))^2 )
}
```

To check if the function works correctly, $n = 1000000$ generated samples is plotted with the actual distribution, as seen in Figure 3. Here $\alpha = 0.5$ is used.

```r
set.seed(42)
a <-0.5
n <- 1000000
samples <- sample_f(a,n)
x <- seq(-15,15,0.1)

hist(samples, freq=F, breaks=100,main=NULL, xlim=c(-12,12), xaxp=c(-12,12,8),xlab = "x")
lines(x,f(x,a),col = "red", lwd = 2)
legend("topright", legend=c("f(x)"), col=c("red"), lty = 1, cex = 0.8, lw=2, inset=0.1, box.lty=0)
```

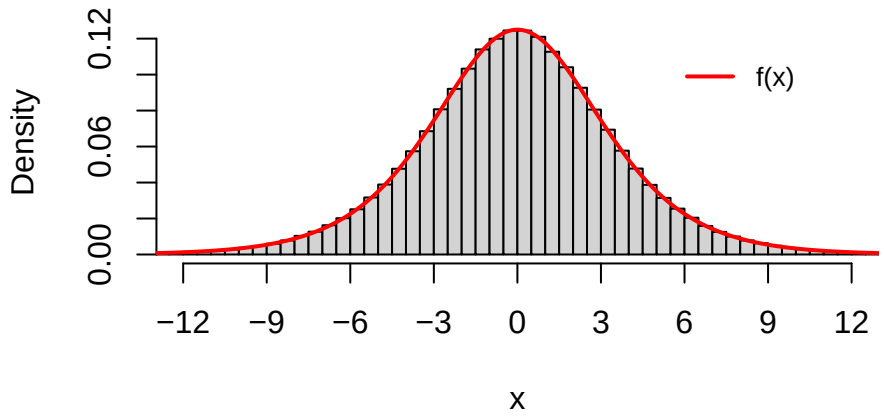

Figure 3: Samples from $f(x)$ distribution, using $\alpha = 0.5$.

It looks to be a good fit.

We note that $f(x)$ is actually a logistic distribution with location parameter $\mu = 0$ and scale parameter $\sigma = 1/\alpha$. The expected value and variance of a logistic distribution are μ and $\frac{\pi^2\sigma^2}{3}$ respectively, so we can compare these to the sample mean and variance.

```r
mean <- 0
var <- pi^2/(a^2*3)
```

```
## Theoretical mean:  0
## Sample mean:  -0.0005581292
```

```
## Theoretical variance:  13.15947
## Sample variance:  13.13792
```

The sample properties matches that of the theoretical values well.

The Box-Muller method

Now we want to make a R function which uses the Box-Muller algorithm to generate n samples from the standard normal distribution. Having $X_1, X_2 \overset{iid}{\sim} \mathcal{N}(0,1)$, the joint pdf of X_1 and X_2 is given by:

$$f_{X_1,X_2}(X_1,X_2) = f_{X_1}(X_1) \cdot f_{X_2}(X_2) = \frac{1}{2\pi} \exp\left(-\frac{X_1^2 + X_2^2}{2}\right)$$

Using r and θ as the polar coordinates for the point (X_1, X_2), gives a new way of calculating X_1 and X_2:

$$X_1 = r\cos(\theta) \quad \text{and} \quad X_2 = r\sin(\theta)$$

It is well known that $\theta \sim Unif(0, 2\pi)$, since it represents the degrees of a circle. To find a value for r however, finding the pdf in polar form is the first step.

$$f_{R,\Theta}(r,\theta) = f_{X_1,X_2}(X_1,X_2) \cdot \begin{vmatrix} \partial X_1/\partial r & \partial X_2/\partial r \\ \partial X_1/\partial \theta & \partial X_2/\partial \theta \end{vmatrix} = \frac{1}{2\pi} r e^{-r^2/2} = f_\Theta(\theta) \cdot f_R(r)$$

where $f_R(r) = re^{-r^2/2}$. The cdf of f_R is uniformly distributed between 0 and 1, and by finding the inverse of the cdf we have a easy way of generating r. The cdf is:

$$F_R(r) = \int_{-\infty}^{r} xe^{-x^2/2}\,dx = 1 - e^{-r^2/2}$$

The cdf is then defined as $u = F_R(r)$, and by solving for r the inverse is found.

$$u = 1 - e^{-r^2/2} \quad \Rightarrow \quad r = \sqrt{-2\ln(1-u)} = F_R^{-1}(u)$$

where u is uniformly distributed between 0 and 1. In the function both u and θ is generated by using `runif()`. These are then used to calculate r, and further calculate X_1 and X_2. The function only takes in the number of samples to be generated.

```r
# add normal_BoxMuller()
normal_BoxMuller <- function(n){
  u <- runif(n)  # generates n random numbers between 0 and 1
  theta <- runif(n, max=(2*pi)) # generates n random numbers between 0 and 2*pi
  r <- sqrt(-2*log(u))
  X1 <- r * cos(theta)
  X2 <- r * sin(theta)
  return(list(X1 = X1,X2 = X2))
}
```

To check if these two are standard normally distributed, we plot them in a histogram with the theoretical distribution in addition to checking the sample mean and variance. Here $n = 10000$ samples are generated, and the plot can be seen in Figure 4.

```r
# standard normal distribution used in plotting
standard_distribution <- function(x){
  return(1/(sqrt(2*pi)) * exp(-0.5 * x^2) )
}
```

```r
# plotting histograms
set.seed(420)
n <- 10000
X <- normal_BoxMuller(n)
X1 <- X$X1
X2 <- X$X2

x <- seq(-4,4,0.1)
par(mfrow=c(1,2))
hist(X1, freq=F, main=NULL,xlab = "x")
lines(x, standard_distribution(x), col="red")
legend("topright", legend=c("N(0,1)"), col=c("red"), lty = 1, cex = 0.6, lw=2, box.lty=0)
hist(X2, freq=F, main=NULL,xlab = "x")
lines(x, standard_distribution(x), col="red")
legend("topright", legend=c("N(0,1)"), col=c("red"), lty = 1, cex = 0.6, lw=2, box.lty=0)
```

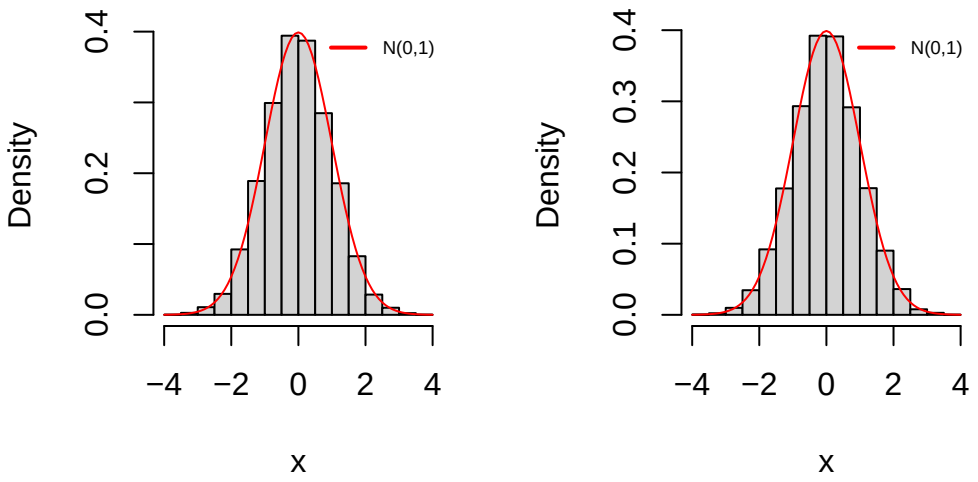

Figure 4: Left: X_1 samples generated from Box-Muller. Right: X_2 samples generated from Box-Muller.

Both the samples looks to match the standard normal distribution. The sample mean and variance should be as close to 0 and 1 as possible.

```
## X1:
## Sample mean:  -0.02123665
## Sample variance:  0.9884373

## X2:
## Sample mean:  -0.001598324
## Sample variance:  1.002433
```

We see that both the mean and variance is close to the theoretical values.

Sampling from d-variate normal distributions

We now want to generate realizations from a d-variate normal distribution, with given mean vector μ and covariance matrix Σ.

Letting $\mathbf{x} \sim \mathcal{N}_d(0, I_d)$, it is known that $\mathbf{Y} = \mu + A\mathbf{x} \sim \mathcal{N}_d(\mu, AA^T)$. Then $\mathbf{Y} \sim \mathcal{N}_d(\mu, \Sigma)$, as long as $AA^T = \Sigma$. To find the A matrix, the function `chol()` is used to perform a Cholesky decomposition of the given covariance matrix. The function `normal_BoxMuller()` is used from the previous task to generate the standard normal random variable $\mathbf{x}$. The R-function `multinormal` takes in the mean vector μ, the covariance matrix Σ and the number of samples to be generated.

```
multinormal <- function(n, mu, sigma){
  d <- length(mu)
  A <- chol(sigma) #returns L^T
  Y <- c()

  for (i in 1:n) {
    # Generate x which is standard normal, using Box-Muller
    X_BM <- normal_BoxMuller(d)
    x <- unlist(X_BM[1])

    y <- mu + t(A) %*% x
    Y <- rbind(Y, as.vector(y))
  }
  return(Y)
}
```

To check if this function works one can compare the estimated mean and covariance with the known μ and Σ. In this case $\mu = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ and $\Sigma = \begin{bmatrix} 0.01 & 0.016 \\ 0.016 & 0.04 \end{bmatrix}$ is used as input for the function. $n = 10000$ samples are generated, and can be seen in Figure 5.

```
# calculate mean and covariance
mu = c(1,2)
sigma = rbind(c(0.01,0.016), c(0.016, 0.04))
n <- 10000

y <- multinormal(n, mu, sigma) # n samples of Y-vector

hist(y[,1], main=NULL, freq=F, col="blue", xlim = c(0.5,3), xlab="y")
hist(y[,2], freq=F, col="red", add=T)
legend("topright", legend=c("y1", "y2"), col=c("blue", "red"), lty = 1, cex = 0.8, lw=2, inset=0.1, box.lty=0)
```

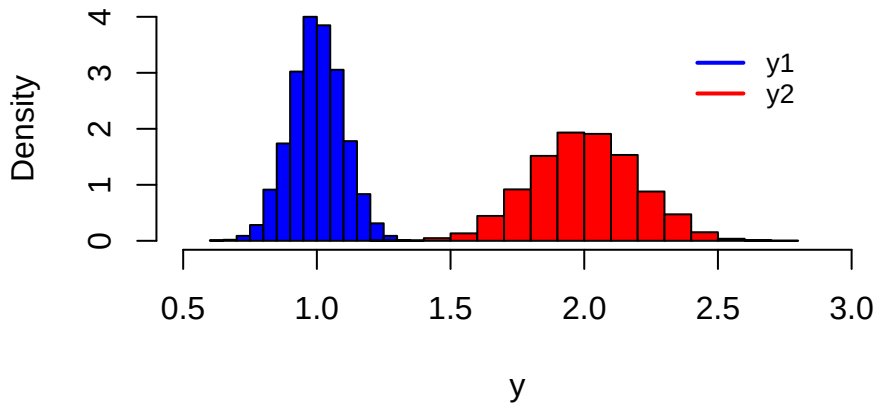

Figure 5: Samples from d-variate normal distribution function

We see that the blue histogram seems to have a mean matching to $\mu_1 = 1$ and the red histogram has a mean matching to $\mu_2 = 2$. The mean and covariance of the samples are printed below.

```
## Mean of y:
```

```
## [1] 0.9992626 2.0000999
```

```
## Covariance of y:
```

```
##              [,1]       [,2]
## [1,] 0.009570763 0.01526494
## [2,] 0.015264939 0.03884717
```

Both of them matches the original μ vector and Σ matrix.

Part B - The gamma distribution

We consider a gamma distribution with $\alpha \in (0, 1)$ and $\beta = 1$, which has the probability distribution function

$$f(x) = \begin{cases} \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}, & 0 < x \\ 0, & \text{else} \end{cases}$$

Rejection sampling accepts a sample in the case that $u < \frac{f^*(x)}{Mg(x)}$, where $u \sim \text{unif}(0, 1)$, x is drawn from the proposal distribution $g(x)$, and M is a constant such that $Mg(x) \geq f^*(x)$ for all $x \in \mathbb{R}$. The acceptance probability is thus

$$P\left(U < \frac{f^*(X)}{Mg(X)}\right) = \int_{-\infty}^{\infty} P\left(U < \frac{f^*(x)}{Mg(x)} \middle| X = x\right) g(x)dx$$

where we have used the law of total probability. Note that since $\frac{f^*(X)}{Mg(X)} \in [0, 1]$, $P\left(U < \frac{f^*(x)}{Mg(x)} \middle| X = x\right)$ is Bernoulli distributed with probability $\frac{f^*(X)}{Mg(X)}$, and so we have

$$P\left(U < \frac{f^*(X)}{Mg(X)}\right) = \int_{-\infty}^{\infty} \frac{f^*(x)}{Mg(x)} g(x)dx = \int_{-\infty}^{\infty} \frac{f^*(x)}{M} dx = \frac{1}{M}$$

where the last equality holds only if $f^*(x) = f(x)$.

We implement a function that samples from the gamma distribution defined above. The function uses rejection sampling by sampling from the distribution (3).

```
#B1
#Sampling from a gamma distribution by using the distribution in A2 for rejection sampling
gamma_distribution <- function(x,alpha,beta = 1){
  #Pdf of gamma distribution. Implemented with beta = 1 for later use
  x1 <- x[x > 0]
  x2 <- x[x <= 0]
  return ((1/(gamma(alpha)*beta^alpha)*c(0*x2,x1^(alpha-1)*exp(-x1/beta))))
}

sample_gamma_1 <- function(n,alpha){
  #Sampling from the gamma distribution using rejection sampling. Works only for
  #0 < alpha < 1
  #Returns: samples, a vector containing n draws from gamma
```

```
  samples <- numeric(length(n))
  d <- (alpha + exp(1)) /(alpha*exp(1))
  count <- 0

  while (count < n){
      x <- sample_g(1,alpha) #Sample one x from g(x)
      u <- runif(1) #Sample one unif(0,1)
      a <- gamma_distribution(x,alpha)/(d*g(x,alpha))

      if (u < a){ #Success!
        count <- count + 1
        samples[count] <- x
      }
  }
  return (samples)
}
```

To test the implementation, we sample $n = 10000$ numbers with $\alpha = 0.9$, and plot these in a histogram, along with the actual distribution. This is shown i Figure 6. We also calculate the sample mean and variance and compare these to the theoretical mean and variance, which we know to be $\alpha\beta = \alpha$ and $\alpha\beta^2$, respectively.

```
set.seed(42)
a <- 0.9 #alpha
b <- 1 #beta
samples <- sample_gamma_1(10000,a)
x <- seq(0.1,6,0.01)

hist(samples,freq = FALSE, breaks = 100,main = NULL, xlab = "x")
lines(x,gamma_distribution(x,a),col = "red", lwd = 2)
legend("topright", legend=c("Gamma(0.9,1)"), col=c("red"), lty = 1, cex = 0.8, lw=2, inset=0.1, box.lty=0)
```

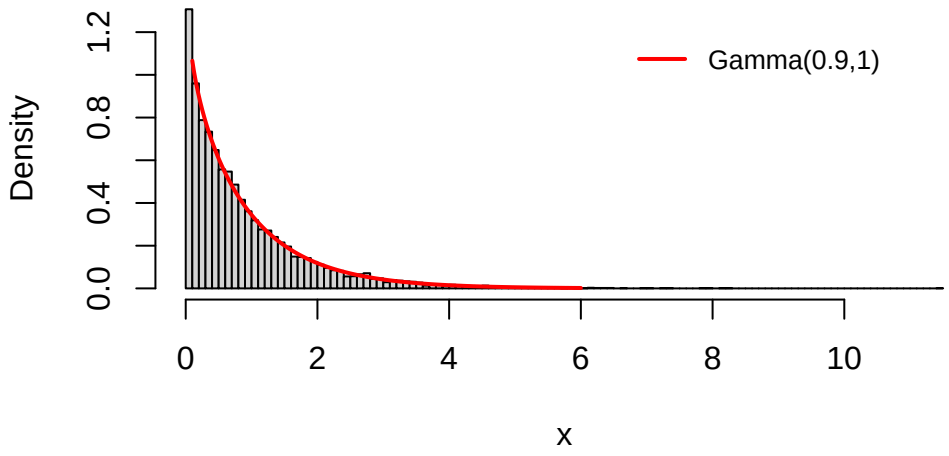

Figure 6: Gamma samples, rejection sampling

```
## Theoretical mean:  0.9 , Sample mean:  0.8931921
```

```
## Theoretical variance:  0.9 , Sample variance:  0.8932746
```

Ratio-of-uniforms

In the ratio-of-uniforms method, we accept only the samples $\frac{x_2}{x_1}$ that satisfy $0 \leq x_1 \leq \sqrt{f^*\left(\frac{x_2}{x_1}\right)}$, where x_1 and x_2 are uniformly sampled and independent, and $f^*(x)$ is the target distribution. We sample x_1 and x_2 from the rectangle $[0, a] \times [b_-, b_+]$, where $a = \sqrt{\sup_x f^*(x)}$, $b_- = \sqrt{\sup_{x \geq 0} x^2 f^*(x)}$ and $b_- = \sqrt{\inf_{x \leq 0} x^2 f^*(x)}$.

Consider the gamma distribution with $\alpha > 1$ and $\beta = 1$, which has the unscaled distribution

$$f^*(x) = \begin{cases} x^{\alpha-1}e^{-x}, & 0 < x \\ 0, & \text{else} \end{cases} \tag{9}$$

We can calculate the rectangle bounds as follows. For a,

$$\begin{aligned} (f^*(x))' = 0 \implies (\alpha-1)e^{-x}x^{\alpha-2} - e^{-x}x^{\alpha-1} = 0 \implies x = (\alpha-1) \\ \implies \sup_x f^*(x) = \left(\frac{\alpha-1}{e}\right)^{\alpha-1} \\ \implies a = \left(\frac{\alpha-1}{e}\right)^{\frac{\alpha-1}{2}} \end{aligned}$$

Similarily, for b_+, we find

$$\begin{aligned} (x^2 f^*(x))' = 0 \implies (\alpha+1)e^{-x}x^{\alpha} - e^{-x}x^{\alpha+1} = 0 \implies x = (\alpha+1) \\ \implies \sup_{x \geq 0} x^2 f^*(x) = \left(\frac{\alpha+1}{e}\right)^{\alpha+1} \\ \implies b = \left(\frac{\alpha+1}{e}\right)^{\frac{\alpha+1}{2}} \end{aligned}$$

Since $f^*(x) = 0$ for $x \leq 0$, $b_- = 0$.

Below, an implementation of the ratio-of-uniforms method is shown. The algorithm is implemented on log-scale to avoid overflow errors, which occur for sufficiently large values of α. The rectangle bounds on log scale are

$$\ln(a) = \frac{\alpha-1}{2}(\ln(\alpha-1) - 1) \ln(b_+) = \frac{\alpha+1}{2}(\ln(\alpha+1) - 1)$$

which means that in practice, the samples become

$$\ln(x_1) = \ln(a) + \ln(u_1),\ u_1 \sim \text{unif}(0,1) \ln(x_2) = \ln(b_+) + \ln(u_2),\ u_2 \sim \text{unif}(0,1)$$

In addition, the acceptance condition for a sample becomes

$$\ln(x_1) \leq \frac{\alpha-1}{2}(\ln(x_2) - \ln(x_1)) - \exp(\ln(x_2) - \ln(x_1) - \ln(2))$$

where the right hand side is the logarithm of (9) evaluated in $\frac{x_2}{x_1}$.

```
#B2
#Sampling from gamma using ratio-of-uniforms method
gamma_unscaled <- function(x,alpha,beta = 1){
  x1 <- x[x > 0]
  x2 <- x[x <= 0]
  return (c(0*x2,(x1^(alpha-1))*exp(-x1/beta)))
}

sample_gamma_2 <- function(n,alpha,report_attempts = F){
```

```
  #Sampling from the gamma distribution using ratio-of-uniforms.
  #Input: alpha, shape parameter > 1
  #       n, number of draws
  #Returns: samples, a vector containing n draws from gamma
  #         attempts, integer

  samples <- NULL
  attempts <- 0

  a <- ((alpha-1)/2) * (log((alpha-1)) - 1)
  bp <- ((alpha+1)/2) * (log((alpha+1)) - 1)
  bn <- 0

  while (length(samples) < n){
    i <- n - length(samples) #How many samples do we need still?
    attempts <- attempts + i #We try i new samples

    x1 <- a + log(runif(i,min = 0, max = 1))
    x2 <- bp + log(runif(i,min = bn, max = 1))

    accept <- (x1 <= ((alpha-1)*(x2-x1)/2 - exp(x2-x1-log(2)))) #What samples are accepted?

    samples <- c(samples, (exp(x2-x1))[accept]) #Add the new samples
  }
  if (report_attempts){
    return (list(attempts = attempts, samples = samples))
  }
  else{
    return (samples)
  }
}
```

To test the implementation, we draw $n = 10000$ samples with $\alpha = 2$ and plot these in a histogram, along with the function (9) in red, as seen in Figure 7.

```
set.seed(420)
alpha <- 2
x <- seq(0.1,2100,0.1)
hist(sample_gamma_2(10000,alpha), freq = FALSE,breaks = 100,main = NULL, xlab = "x")
lines(x,gamma_unscaled(x,alpha),col ="red", lw=2)
legend("topright", legend=c("Gamma(2,1)"), col=c("red"), lty = 1, cex = 0.8, lw=2, inset=0.1, box.lty=0)
```

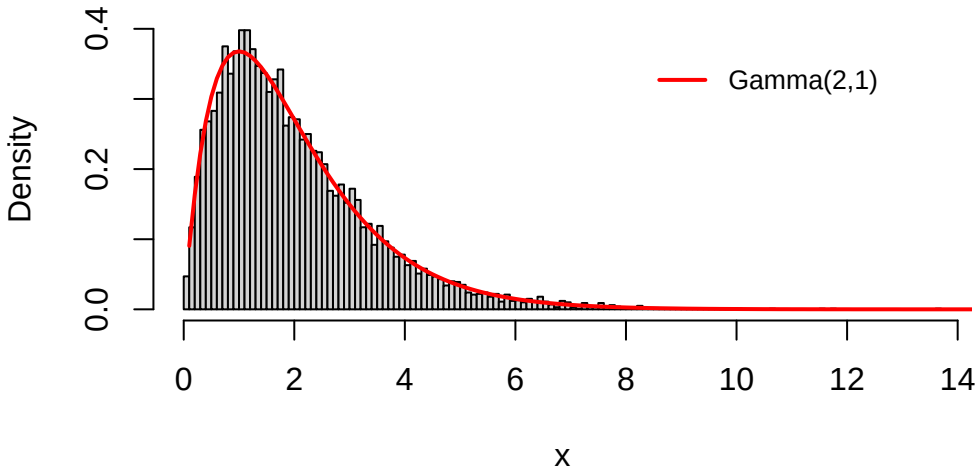

Figure 7: Gamma samples, ratio-of-uniforms

In addition, we sample $n = 1000$ for $\alpha \in (1, 2000)$ and record how many samples (including rejected ones) in total the algorithm used. A plot of the attempts vs the values of α is shown In Figure 8 below.

```
alpha_vals <- seq(2,2000,3)
attempt_vals <- numeric(length = length(alpha_vals))
for (i in 1:length(alpha_vals)){
  attempt_vals[i] <- sample_gamma_2(1000,alpha_vals[i],report_attempts = T)$attempts
}
plot(alpha_vals,attempt_vals,main = NULL, xlab = "Alpha",ylab = "Attempts")
lines(alpha_vals,800*sqrt(alpha_vals),col = "red",lwd = 2)
legend("topleft", legend=c("800*sqrt(alpha)"), col=c("red"), lty = 1, cex = 0.8, lw=2, inset=0.1, box.lty=0)
```

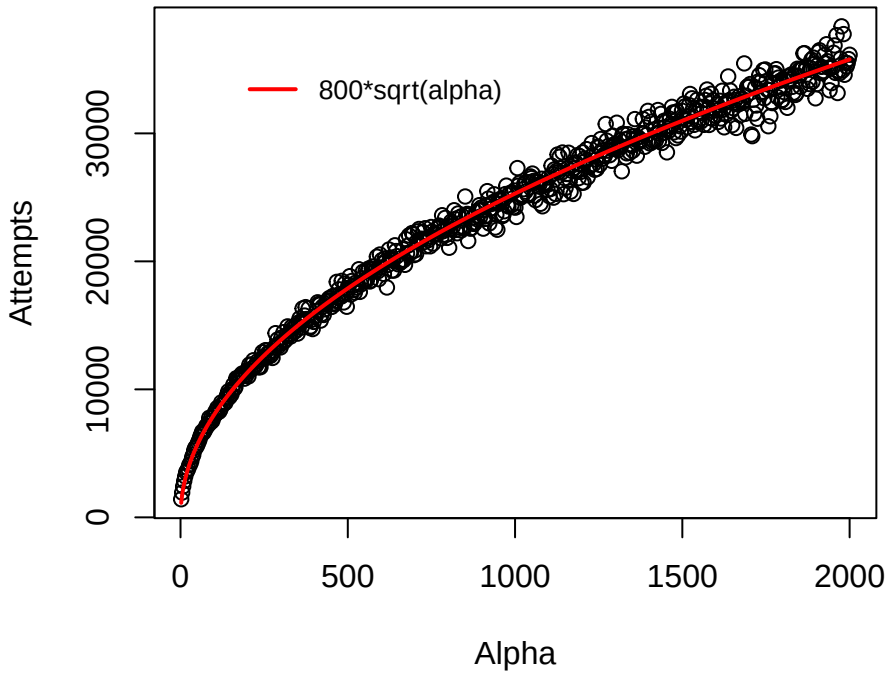

Figure 8: Number of attempts as a function of alpha, n = 1000

We see that the number of attempts increases with the value of α. The plot can be seen as an estimate of the complexity of the algorithm, as a function of α. The red line is $800\sqrt{\alpha}$, which indicates a complexity of $O(\alpha^{1/2})$.

We can now sample from the gamma distribution for any parameters $\alpha, \beta > 0$. For $\alpha \in (0, 1)$, we use rejection sampling, for $\alpha > 1$, we use ratio-of-uniforms, and for the special case of $\alpha = 1$, the gamma distribution becomes the exponential distribution with rate parameter $\frac{1}{\beta}$, from which we can sample using the inversion method. To handle the parameter β, we note the following property of the gamma distribution:

$$X \sim \text{Gamma}(\alpha, \theta) \implies cX \sim \text{Gamma}(\alpha, c\theta)$$

which means that if we want to sample from $\text{Gamma}(\alpha, \beta)$, we can first simulate from $\text{Gamma}(\alpha, 1)$ and then multiply the samples by β.

Below, the implementation of the final gamma sampling function is shown, which works for any parameters $\alpha, \beta > 0$.

```
sample_gamma <- function(n,alpha,beta){
  if (alpha > 1){ #Use ratio-of-uniforms
    return (beta*sample_gamma_2(n,alpha))
  }
  else if (alpha == 1){ #Use exponential sampling
    return (sample_exponential(n,1/beta))
  }
  else if (alpha > 0){ #Use rejection sampling
    return (beta*sample_gamma_1(n,alpha))
  }
```

```
  else {
    print("Error: negative parameters not allowed.")
  }
}

set.seed(420)
alpha <- 10
beta <- 2
x <- seq(0.1,50,0.1)
samples <- sample_gamma(10000,alpha,beta)
hist(samples, freq = FALSE,breaks = 100,main = NULL, xlab = "x",ylim = c(0,0.15),col = "pink")
lines(x,gamma_distribution(x,alpha,beta = beta),col ="red", lw=2)
legend("topright", legend=c("Gamma(10,2)","Gamma(0.6, 3)"), col=c("red","blue"), lty = 1, cex = 0.8, lw=2, inset=0.1, box.lty=0)

alpha <- 0.6
beta <- 3
x <- seq(0.1,50,0.1)
hist(sample_gamma(10000,alpha,beta), freq = FALSE,breaks = 100,main = NULL, xlab = "x",add= T,col = "lightblue")
lines(x,gamma_distribution(x,alpha,beta = beta),col ="blue", lw=2)
```

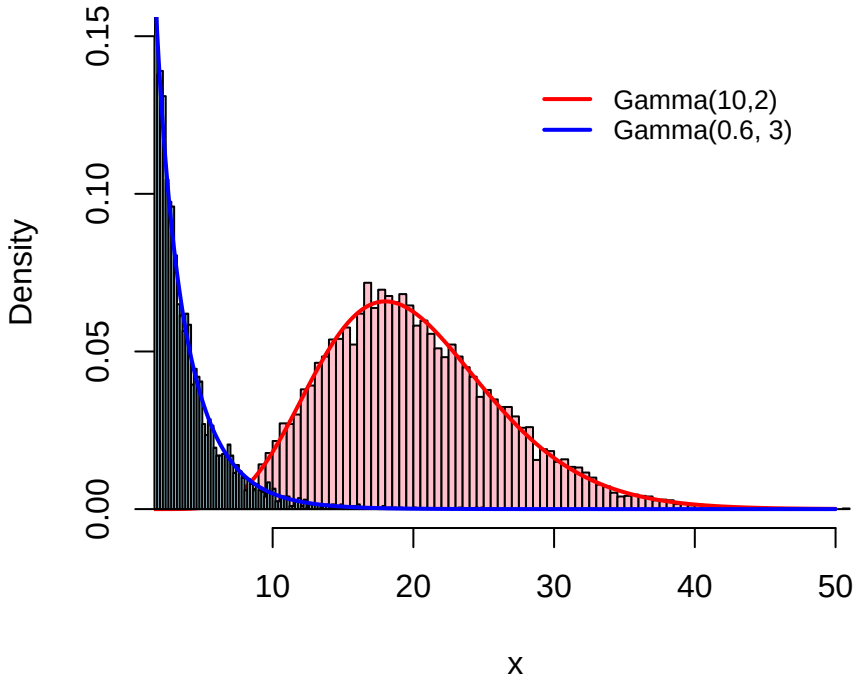

Figure 9: Samples from Gamma(10,2) (pink) and Gamma(0.6,3) (light blue).

In Figure (9), we test the implementation for two gamma distributions and plot the theoretical pdfs for comparison. In addition, we compare the theoretical mean and variance with the sample mean and variance for the Gamma$(10, 2)$ distribution.

```
## Gamma(10,2) - theoretical mean:  20  ,sample mean:  19.96817

## Gamma(10,2) - theoretical variance:  40  , sample variance:  39.19013
```

Beta distribution from the gamma distribution

Let $x \sim \text{Gamma}(\alpha, 1)$ and $y \sim \text{Gamma}(\beta, 1)$ be independent, and define $z = \frac{x}{x+y}$. Then $z \sim \text{beta}(\alpha, \beta)$, which we will prove in the following:

Consider the joint pdf of x, y,

$$f_{X,Y}(x,y) = f_X(x)f_Y(y) = \frac{1}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} y^{\beta-1} e^{-(x+y)}$$

since x, y are independent. We consider the transformations $U = X/(X+Y)$ and $V = X+Y$, or the inverse transformations $X = UV$ and $Y = V(1-U)$. The transformation formula gives us the joint distribution of U, V:

$$f_{U,V}(u,v) = f_{X,Y}(x,y) \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} \\ \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} \end{vmatrix} = f_{X,Y}(x,y) \begin{vmatrix} v & -v \\ u & 1-u \end{vmatrix} = f_{X,Y}(x,y) \cdot v$$

so the joint distribution becomes

$$f_{U,V}(u,v) = \frac{v}{\Gamma(\alpha)\Gamma(\beta)} (vu)^{\alpha-1} (v(1-u)))^{\beta-1} e^{-(vu+v(1-u))} = \frac{1}{\Gamma(\alpha)\Gamma(\beta)} v^{\alpha+\beta-1} u^{\alpha-1} (1-u)^{\beta-1} e^{-v}$$

Rearranging the terms, we find that

$$f_{U,V}(u,v) = \frac{1}{\Gamma(\alpha+\beta)} v^{\alpha+\beta-1} e^{-v} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} u^{\alpha-1} (1-u)^{\beta-1} = f_V(v) f_U(u)$$

Hence, V and U are independent, with $f_V(v)$ being the pdf of the Gamma$(\alpha+\beta, 1)$ distribution, and $f_U(u)$ being the pdf of the Beta(α, β) distribution, which gives us the desired result, namely that $z = U = \frac{x}{x+y} \sim$ beta(α, β).

Using the gamma sampler from before, we can now sample from the beta(α, β) distribution, by first sampling $x \sim \text{Gamma}(\alpha, 1)$ and then $y \sim \text{Gamma}(\beta, 1)$. The implementation is shown below.

```
beta_distribution <- function(x,alpha,beta){
  #pdf of beta distribution, only defined for 0 <= x <= 1
  x <- x[x >= 0]
  x <- x[x <= 1]
  return ((gamma(alpha + beta)/(gamma(alpha)*gamma(beta)))*x^(alpha-1)*(1-x)^(beta-1))
}

sample_beta <- function(n,alpha,beta){
  x <- sample_gamma(n,alpha,1)
  y <- sample_gamma(n,beta,1)
  z <- x/(x+y)
  return (z)
}
```

To test the function, we sample $n = 10000$ numbers from the beta distribution, and plot these along with the pdf in Figure 10. We also calculate the sample mean and variance and compare these to the theoretical mean and variance.

```
#Testing the function
alpha <- 3
beta <- 3
n <- 10000
x <- seq(0,1,0.01)
samples <- sample_beta(10000,alpha,beta)
hist(samples,freq=F,breaks = 100,main = NULL, xlab = "x")
lines(x,beta_distribution(x,alpha,beta),col = "red")
legend("topright", legend=c("Beta(3, 3)"), col=c("red"), lty = 1, cex = 0.8, lw=2, inset=0.05, box.lty=0)
```

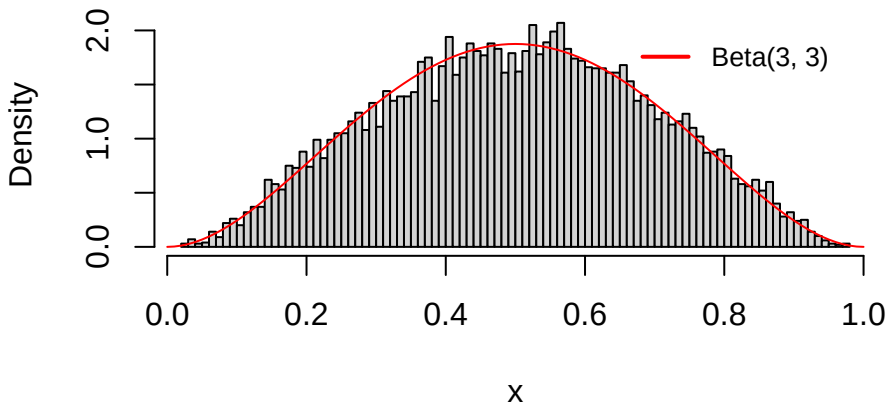

Figure 10: Samples from beta distribution, $\alpha = \beta = 3$

```
## Theoretical mean:  0.5 , sample mean:  0.5024896
```

```
## Theoretical variance:  0.03571429 , sample variance:  0.03668608
```

We see that the sample and theoretical moments coincide.

Part C - Monte Carlo integration and variance reduction

Monte Carlo integration is a method for calculating a definite integral. More specifically, we will use Monte Carlo integration to find the expectation $\mathrm{E}(h(X))$, where $h(X) = \theta = P(X > 4)$, with $X \sim N(0, 1)$.

The estimate is then given by $\widehat{\mathrm{E}(h(x))} = \frac{1}{n}\sum_{i=1}^{n} h(x_i)$, $x_i \sim N(0, 1)$. We can also calculate confidence intervals, by noting that

$$\frac{\mathrm{E}(h(x)) - \widehat{\mathrm{E}(h(x))}}{\sqrt{\mathrm{Var}(\widehat{\mathrm{E}(h(x))})}} \sim t_n$$

which is asymptotically a standard normal distribution as $n \to \infty$. Below, an implementation of Monte Carlo integration is shown. The estimate $\widehat{\mathrm{E}(h(x))}$ for $n = 100000$ samples and a 95% confidence interval are shown in the output.

```
MC_integration <- function(n){
  normal_samples <- normal_BoxMuller(n)$X1
  est <- (1/n)*sum((normal_samples > 4))
  ci <- c(est - 1.96*sd(normal_samples > 4)/sqrt(n), est + 1.96*sd(normal_samples > 4)/sqrt(n))

  return (list(est = est,ci = ci))
}
set.seed(420)
mc <- MC_integration(100000)
mc
```

```
## $est
```

```
## [1] 5e-05
##
## $ci
## [1] 6.173944e-06 9.382606e-05
```

We see that the Monte Carlo integration gives an estimated probability of $5 \cdot 10^{-5}$.

Consider the same case as above - however, this time we can only sample from the distribution $g(x)$. The importance sampling algorithm provides a solution by weighting each term in the sum by $f(x_i)/g(x_i)$, so that the estimate becomes

$$\widehat{\mathrm{E}(h(x))} = \frac{1}{n}\sum_{i=1}^{n} h(x_i)\frac{f(x_i)}{g(x_i)}, \; x_i \sim g(x)$$

We can calculate 95% confidence intervals in the same way as for regular Monte Carlo integration. Here, we use the function

$$g(x) = \begin{cases} cxe^{-\frac{1}{2}x^2}, & x > 4 \\ 0, & \text{else} \end{cases} \tag{10}$$

where c is a normalising constant we set to be e^8. To sample from $g(x)$, we employ inversion sampling - we sample $u \sim \text{unif}(0,1)$, which gives $x = \sqrt{-2\ln(-u/c + e^{-8})} \implies x \sim g(x)$. The implementation of the importance sampling is shown below, along with the produced estimate for $n = 100000$ samples and a 95% confidence interval.

```
sample_g2 <- function(n,antithetic = F){
  c <- exp(8)
  u <- runif(n)
  x <- sqrt(-2*log(-u/c + exp(-8)))
  if (antithetic){
    x1 <- sqrt(-2*log(-(1-u)/c + exp(-8)))
    return (c(x,x1))
  }
  return (x)
}

g2 <- function(x){
  x1 <- x[x <= 4]
  x2 <- x[x > 4]
  c <- exp(8)

  return (c(0*x1, c*x2*exp(-0.5*x2^2)))
}

importance_sampling <- function(n,antithetic = F){

  g_samples <- sample_g2(n,antithetic = antithetic)
  n <- length(g_samples)
  fg <- dnorm(g_samples)/g2(g_samples)

  prod <- (g_samples > 4) %*% fg
  est <- (1/n)*(prod)

  sum_elements <- (g_samples > 4) * fg

  stdev <- sqrt(var(sum_elements)/n)
  if (antithetic){
    stdev <- sqrt(var(sum_elements)/n + cov(sum_elements[1:(n/2)],sum_elements[(n/2+1):n])/n)
  }

  ci <- c(est - 1.96*stdev, est + 1.96*stdev)

  return (list(est = est, ci = ci))
}

set.seed(420)
```

```r
im <- importance_sampling(100000)
im
```

```
## $est
##              [,1]
## [1,] 3.166944e-05
##
## $ci
## [1] 3.165979e-05 3.167910e-05
```

We see that the estimate produced by the importance sampling is similar to that of the Monte Carlo integration. We can compare the the precision of the two methods by for example looking at the width of the confidence intervals:

```r
#Compare precision of MC integration and importance sampling
diff(mc$ci)
```

```
## [1] 8.765211e-05
```

```r
diff(im$ci)
```

```
## [1] 1.930921e-08
```

The confidence interval of the importance sampling method is much more narrow with a width of $1.931 \cdot 10^{-8}$, whilst the Monte Carlo integration method produced a confidence interval with width $8.765 \cdot 10^{-5}$. This indicates that importance sampling produces estimates with much less variance.

To achieve the same precision in both methods, one would need to increase the number of samples n in the Monte Carlo integration. We can calculate the required n if we make the assumption that the standard error of the estimate in the MC integration is constant as n becomes large enough. That is, we set $n = 100\ 000$ (or any large enough number), calculate the standard error of the estimate produced in the MC integration, and set

$$\frac{SD_{MC}}{\sqrt{n_{MC}}} = \frac{SD_{IM}}{\sqrt{n_{IM}}}$$

If the above equality holds, both estimates have the same precision. Here, we calculate SD_{MC} by setting $n = 100\ 000$, and we set $n_{IM} = 100\ 000$ to find $SD_{IM}/\sqrt{n_{IM}}$. We can then solve for n_{MC}:

$$n_{MC} = \left(\frac{SD_{MC}\sqrt{n_{IM}}}{SD_{IM}} \right)^2$$

The following code calculates and reports the estimated n_{MC} required to achieve the same precision in Monte Carlo integration as in importance sampling.

```r
SD_n_IM <- (im$est - im$ci[1])/1.96
SD_MC <- (mc$est - mc$ci[1])*sqrt(100000)/1.96
(SD_MC / SD_n_IM)^2
```

```
##             [,1]
## [1,] 2.06061e+12
```

The number of samples required is on the order of 10^{12}, which is infeasible.

Antithetic sampling

Antithetic sampling provides another option to reduce the variance in the importance sampling. By sampling n pairs of uniformly distributed numbers, namely the pairs $(u, 1-u)$, we can reduce the variance by making terms negatively correlated. The estimate becomes

$$\widehat{\mathrm{E}(h(x))} = \frac{1}{2n}\sum_{i=1}^{n}\left(h(F^{-1}(u_i))\frac{f(F^{-1}(u_i))}{g(F^{-1}(u_i))} + h(F^{-1}(1-u_i))\frac{f(F^{-1}(1-u_i))}{g(F^{-1}(1-u_i))}\right)$$

where $F^{-1}(\cdot)$ is the inverse cdf of $g(x)$ defined in (10).

The implementation of importance sampling from before includes a default boolean argument `antithetic` which can be set to `True` in order for the negative correlation to be included in the calculation of the standard error of the estimate. To compare the antithetic sampling to the importance sampling, we again look at the width of the 95% confidence interval. To get a fair comparison, we only use $n = 50000$ pairs of samples in the antithetic sampling, since this still gives a total of 100 000 terms in the estimate (but some of the terms are correlated!). The width of the confidence intervals are shown below:

```
#Compare precision of importance sampling and antithetic sampling
set.seed(420)
diff(importance_sampling(100000,antithetic = F)$ci)
```

```
## [1] 1.930921e-08
```

```
diff(importance_sampling(50000,antithetic = T)$ci)
```

```
## [1] 9.450126e-09
```

We see that the antithetic sampling produces an estimate which has lower variance.

Part D - Bayesian inference

We consider the cell count data with four categories, $\{y_1 = 125, y_2 = 18, y_3 = 20, y_4 = 34\}$, with respective probabilities $\{\frac{1}{2} + \frac{\theta}{4}, \frac{1-\theta}{4}, \frac{1-\theta}{4}, \frac{\theta}{4}\}$. The data is assumed to be multinomially distributed, which gives the multinomial mass function $f(\mathbf{y}|\theta) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}$. If we use unif$(0,1)$ as a prior, we get the posterior density

$$f(\theta, \mathbf{y}) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4} \quad \text{for} \quad \theta \in (0,1) \tag{11}$$

First, a rejection sampling algorithm is implemented to simulate from $f(\theta, \mathbf{y})$ using a $\mathcal{U}(0,1)$ as the proposal density.

```
# Function for calculating the posterior mean
posterior <- function(theta){
  y1 <- 125
  y2 <- 18
  y3 <- 20
  y4 <- 34

  f <- (2+theta)^y1 * (1-theta)^(y2+y3) * theta^y4

  return(f)
}


# Sampling from f(theta|y) - using rejection sampling
rejection_sampling <- function(n,func){
```

```
  samples <- c()
  c <- max(func(seq(0,1,length.out=n)))  # since g is u(0,1), then c>=f

  count <- 0  # how many random numbers must be generated to get n samples

  repeat{
    m <- n - length(samples)  # m = samples we need
    if(m == 0){
      break()  # we have got n samples
    }
    x <- runif(m)  # proposal density
    u <- runif(m)
    f <- func(x)  # f*
    index <- u <= f/c     # all x where u<alpha, alpha=f/c
    samples <- c(samples, x[index])

    count <- count + m
  }

  return(list(samples = samples, avg_count = count/n))
}
```

Using samples from $f(\theta, \mathbf{y})$ the posterior mean can be estimated. This is done using the Monte-Carlo integration, which gives the estimate

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

In our case $h(x_i)$ is the generated samples from the $f(\theta, \mathbf{y})$ density. Generating $n = 10000$ samples from the `rejection_sampling` function gives the the following posterior mean:

```
n <- 10000
sample <- rejection_sampling(n,posterior)$samples
posterior_mean <- (1/n)*sum(sample)
```

```
## Posterior mean:  0.6229478
```

To check if the function works, we would like to plot the samples along with the theoretical distribution, as well as comparing the posterior mean to the theoretical mean. To find both the theoretical mean and density function, the normalizing constant needs to be calculated. This is done by integrating the posterior function, Equation (11), over its domain. The normalizing constant d, is shown below.

```
d <- integrate(posterior,0,1)$val  # normalizing constant
```

```
## d:  2.357695e+28
```

The now properly scaled density function along with the generated samples from `rejection_sampling` is plotted in Figure 11:

```
hist(sample, freq = F,xlim=c(0,1), breaks=50, xlab="x", main=NULL)
abline(v = posterior_mean, col="blue", lty=2, lw=3)

x <- seq(0,1,0.01)
lines(x,posterior(x)/d, col="red", lw=2)
legend("topleft", legend=c("f(theta|y)/d", "posterior mean"), col=c("red", "blue"), lty = c(1,2), cex = 0.8, lw=2, inset=0.1, bo
```

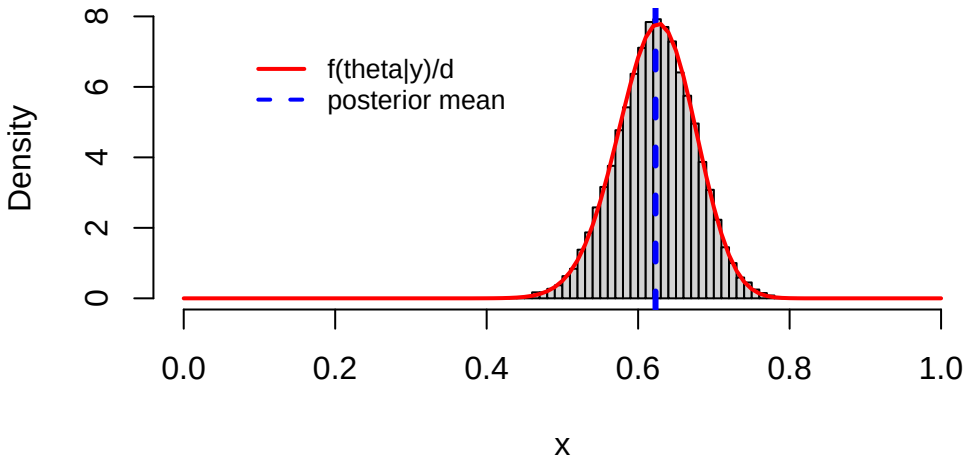

Figure 11: Samples from $f(\theta|y)$

We see that the samples fit the theoretical distribution well. Now to check the theoretical mean, which is calculated using Equation (7). This is done by using the `integrate` function, as seen below.

```
fx <- function(x) {x*posterior(x)/d}
theoretical_mean <- integrate(fx,0,1)$val
```

```
## Theoretical mean:  0.6228061
```

There is a difference of only 0.0001 between the posterior and theoretical mean, indicating that our model works.

To see how many random numbers the algorithm needs to generate on average in order to get one sample from $f(\theta|\mathbf{y})$, we recall the acceptance probability of the rejection sampling algorithm, which we have previously found to be

$$P(\text{sample accepted}) = \int_{-\infty}^{\infty} \frac{f^*(x)}{M} dx$$

where M is a constant such that $Mg(x) \geq f^*(x)$ for all x. Here, $g(x) = 1$ for all x and so $M = \max_x f^*(x)$, which we can compute numerically. The acceptance probability becomes

$$p = \frac{1}{M} \int_0^1 f^*(x) dx$$

The number of samples required until one is accepted is geometrically distributed and so the expected number of samples required is $1/p$. We modify the algorithm to report number of attempts, and compare this with $1/p$ where p is calculated numerically:

```
avg_count <- rejection_sampling(n,posterior)$avg_count

M <- max(posterior(seq(0,1,length.out=n)))
p <- integrate(posterior, 0, 1)$val/M  # probability of success
avg <- 1/p  # number of tries for 1 success

avg_count
```

```
## [1] 7.8458
```

```
avg
```

```
## [1] 7.799305
```

We see that the empirical and theoretical result coincide rather well - to obtain one sample from $f(\theta|\mathbf{y})$ the algorithm needs approximately 8 attempts.

Using another prior

Instead of using a Beta$(1,1)$ prior (which is the same as unif$(0,1)$), we now use a Beta$(1,5)$ prior. The new posterior takes the form

$$f_{new}(\theta|\mathbf{y}) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}\frac{\Gamma(6)}{\Gamma(1)\Gamma(5)}(1-\theta)^4$$

We can use importance sampling to estimate the posterior mean by using the samples we found previously by rejection sampling. We use $f(\theta|\mathbf{y})$ as the proposal density, so that the weights in the importance sampling become

$$w_i = \frac{f_{new}(\theta_i)}{g(\theta_i)} = \frac{\Gamma(6)}{\Gamma(1)\Gamma(5)}(1-\theta)^4$$

Since we only know the unscaled density, we use the estimate

$$\widetilde{E(h(\theta))} = \frac{\sum_{i=1}^n h(\theta_i)w_i}{\sum_{i=1}^n w_i} \tag{12}$$

which is a biased but consistent estimate of the posterior mean. The code and new estimated posterior mean is shown below.

```
new_posterior <- function(theta){
  return (posterior(theta)*beta_distribution(theta,1,5))
}

IM_mean <- (sample %*% (new_posterior(sample)/posterior(sample)))  / sum(new_posterior(sample)/posterior(sample))

#Compare the posterior mean with Beta(1,1) as prior with analytic posterior mean and
#the posterior mean with Beta(1,5) as prior:
```

```
## posterior_mean:  0.6229478
## theoretical_mean:  0.6228061
## IM_mean:  0.5956358
```

We see that the estimated posterior mean from the importance sampling lies further away from the theoretical mean than the estimate from Monte Carlo integration. This is due to the estimator (12) being biased.