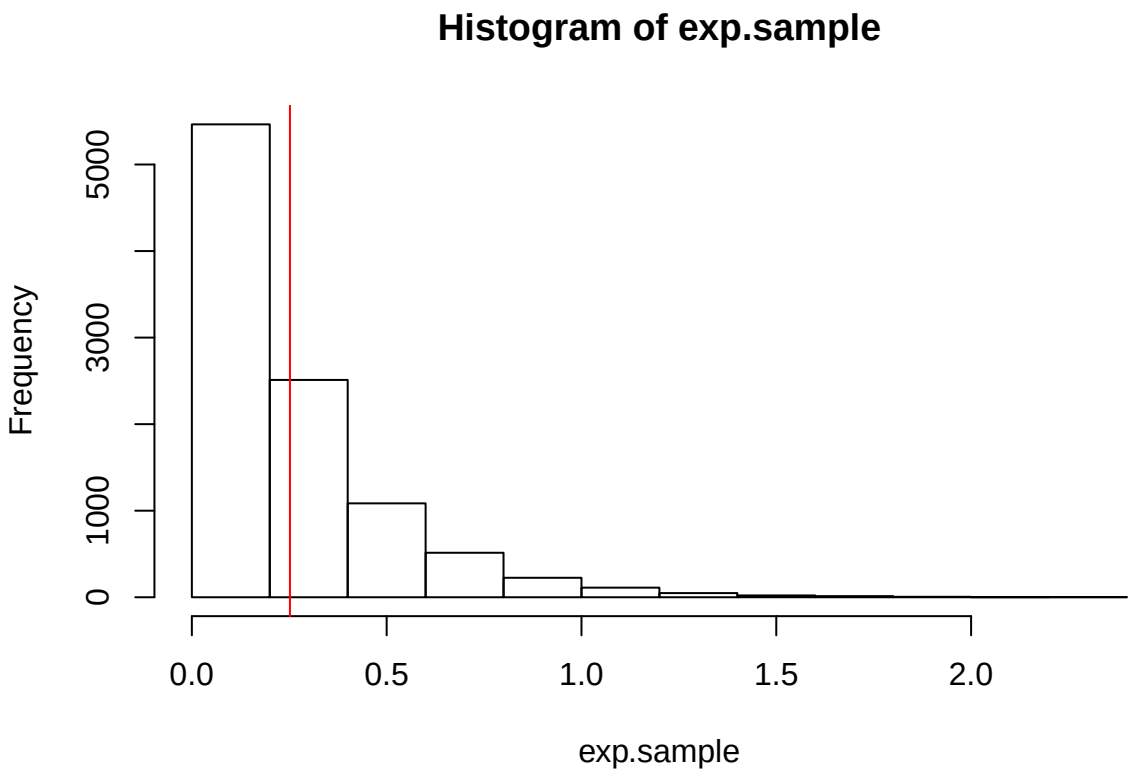
Exercise 1 - TMA4300

Helene Behrens and Martine Middelthon

Problem A

1.

```r
# Function to sample n times from exponential distribution
samples_exp = function(rate, n){
  u = runif(n)          # Sample from unif(0,1) distribution
  x = - log(u) / rate   # Compute values using inverse cdf
  return(x)             # Vector of n samples
}

# Function to print mean and variance
print.mean.var = function(n, m.emp, m.th, v.emp, v.th) {
  cat('Number of samples:', n)
  cat('\n\nEmpirical mean:', m.emp)
  cat('\nTheoretical mean:', m.th)
  cat('\n\nEmpirical variance:', v.emp)
  cat('\nTheoretical variance:', v.th)
}

# Check function
set.seed(0)
rate = 4                                        # Set rate parameter
n = 10000                                       # Number of samples
exp.sample = samples_exp(rate, n)               # Sample
hist(exp.sample)                                # Histogram of sample

mean.emp.exp = mean(exp.sample)                 # Empirical mean
abline(v = mean.emp.exp, col = "red")
```




```r
mean.th.exp = 1 / rate                                  # Theoretical mean

var.emp.exp = var(exp.sample)                           # Empirical variance
var.th.exp = 1 / rate^2                                 # Theoretical variance

print.mean.var(n, mean.emp.exp, mean.th.exp, var.emp.exp, var.th.exp)  # Print values
```

```
## Number of samples: 10000
##
## Empirical mean: 0.2517862
## Theoretical mean: 0.25
##
## Empirical variance: 0.06461034
## Theoretical variance: 0.0625
```

We generate 10000 samples from the exponential distribution with rate $\lambda = 4$ and make a histogram and compute the empirical and theoretical means and variances to test the algorithm. The histogram looks as expected and the empirical and theoretical values for the mean and variance coincides.

2.

a) We consider the probability density function

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

where $\alpha \in (0,1)$ and c is a normalizing constant. Given α, we find that the normalizing constant is $c = \frac{\alpha e}{\alpha + e}$. By integrating $g(x)$ from $-\infty$ to x, the cumulative distribution function is found to be

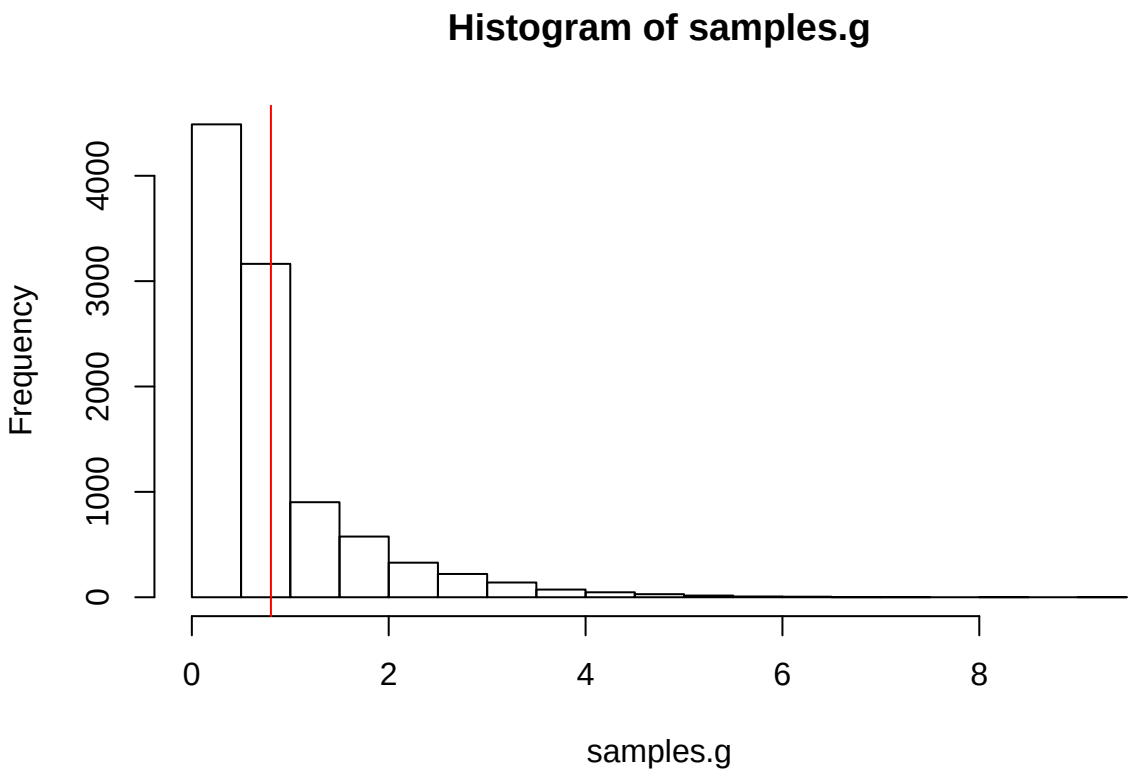
$$F(x) = P(X \leq x) = \begin{cases} 0, & x \leq 0 \\ \frac{c}{\alpha} x^{\alpha}, & 0 < x < 1 \\ c(\frac{1}{\alpha} + \frac{1}{e} - e^{-x}), & x \geq 1 \end{cases}.$$

In order to find the inverse cdf we let $F(x) = y$. The cdf maps $x \in (-\infty, \infty)$ to $y \in [0,1)$. By solving $F(x) = y$ with respect to x, we find that the inverse of the cdf is

$$F^{-1}(y) = \begin{cases} (\frac{\alpha y}{c})^{1/\alpha}, & 0 < y \leq \frac{c}{\alpha} \\ -\ln(\frac{1}{\alpha} + \frac{1}{e} - \frac{y}{c}), & y > \frac{c}{\alpha} \end{cases}.$$

b)

```
# Function to sample from g using inverse cdf
samples_g = function(alpha, n) {
  c = alpha * exp(1) / (alpha + exp(1))    # Normalizing constant
  u = runif(n)                             # Sample from unif(0,1) distribution
  vec = rep(0, n)                          # Vector to store the generated samples
  for(i in 1:length(u)){
    if(u[i]<= c/alpha) {vec[i] = (alpha * u[i] / c)^(1 / alpha)}  # Use case 1 in inverse cdf to genera
    else {vec[i] = -log(1/alpha + exp(-1) - u[i]/c)}              # Use case 2 in inverse cdf to genera
  }
  return(vec)
}

# Check function
set.seed(1)
alpha = 0.8                                # Set parameter
c = alpha * exp(1) / (alpha + exp(1))      # Normalizing constant
n = 10000                                  # Number of samples
samples.g = samples_g(alpha, n)            # Sampling from g
hist(samples.g)                            # Histogram of samples

mean.emp.g = mean(samples.g)                        # Empirical mean
abline(v = mean.emp.g, col = "red")
```




```r
mean.theo.g = c * (1/(alpha+1) + 2*exp(-1))       # Theoretical mean

var.emp.g = var(samples.g)                                  # Empirical variance
var.theo.g = c * (1/(alpha+2) + 5*exp(-1)) - mean.theo.g^2  # Theoretical variance

print.mean.var(n, mean.emp.g, mean.theo.g, var.emp.g, var.theo.g)   # Print values
```

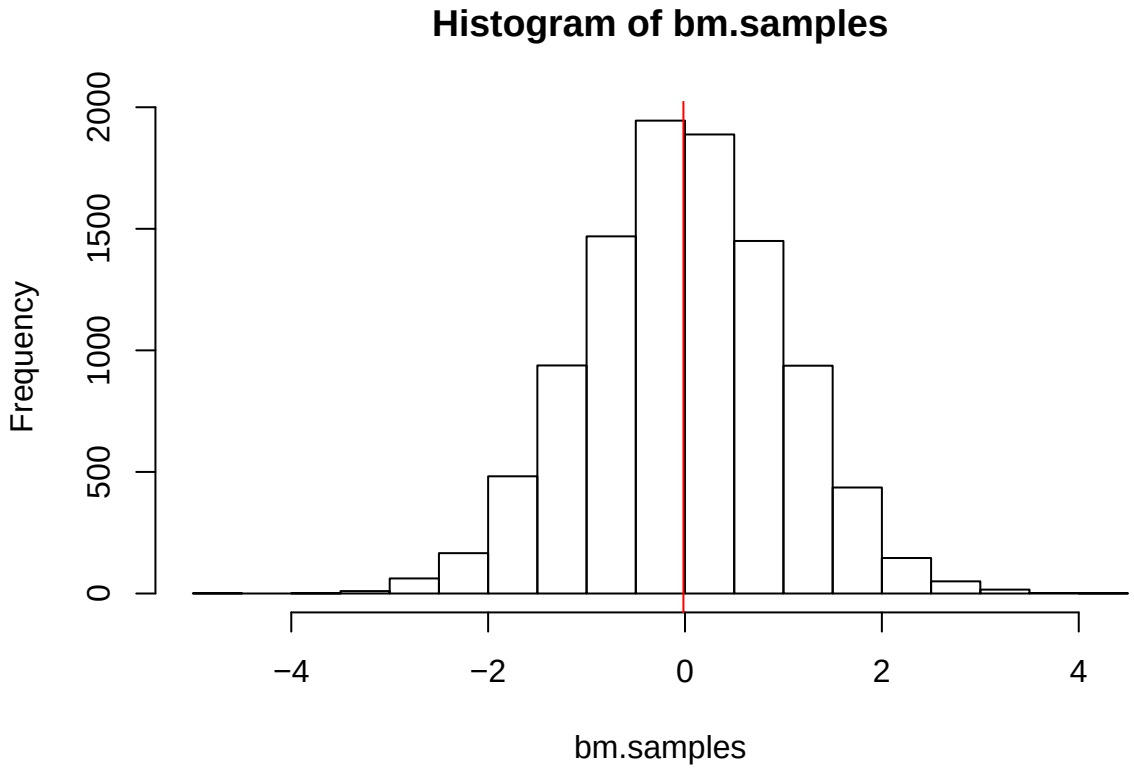
```
## Number of samples: 10000
##
## Empirical mean: 0.8034276
## Theoretical mean: 0.7981524
##
## Empirical variance: 0.7163302
## Theoretical variance: 0.7206188
```

For the 10000 samples from g we make a histogram and compute the empirical and theoretical means and variances. The histogram looks as expected and the empirical mean and variance seems reasonable.

3.

```r
# Function to generate n samples from N(0,1) using Box Muller
boxmuller = function(n) {
  # Sample from unif(0,1)
  u1 = runif(n)
  u2 = runif(n)
  # Polar coordinates
```

```
  r = sqrt(-2*log(u1))
  theta = 2*pi*u2
  #Cartesian coordinate(s)
  x = r*cos(theta)          # Vector of n independent samples
  #y = r*sin(theta)
  return(x)
}

# Check function
set.seed(10)
n = 10000                       # Number of samples
bm.samples = boxmuller(n)       # Sampling from N(0,1)
hist(bm.samples)                # Histogram of samples

mean_bm = mean(bm.samples)      # Empirical mean
abline(v = mean_bm, col="red")
```




```
var_bm = var(bm.samples)        # Empirical variance

print.mean.var(n, mean_bm, 0, var_bm, 1)  # Print values
```

```
## Number of samples: 10000
##
## Empirical mean: -0.01591201
## Theoretical mean: 0
##
```

```
## Empirical variance: 1.014185
## Theoretical variance: 1
```

For the 10000 samples from the standard normal distribution using the Box-Müller algorithm we make a histogram and compute the empirical and theoretical means and variances. The histogram looks as expected and the empirical mean and variance coincide with the theoretical values.

4.

```r
# Function to generate a realization from d-variate normal distribution
mult_normal = function(d, mu, sigma) {
  # mu is vector of means, must have length d
  # sigma is covariance matrix, must have dimension dxd
  x = boxmuller(d)    # d N(0,1) distributed variables
  A = t(chol(sigma))  # Cholesky decomposition of covariance matrix (lower triangular)
  y = mu + A %*% x    # d-variate N(mu, sigma)
  return(y)
}


# Check function

# vector of means and covariance matrix
mu = c(1,2)
sigma = cbind(c(1,0.5),c(0.5,1))

# Estimate mean and empirical variance
set.seed(50)
n = 10000                       # Number of realizations
mu_est = rep(0,length(mu))
multnor.sample = matrix(NA, n, length(mu))
for(i in 1:n) {
  multnor = mult_normal(length(mu),mu,sigma)
  mu_est = mu_est + multnor
  multnor.sample[i,] = multnor
  }
mu_est = mu_est / n               # Estimated mean vector

sigma_est = var(multnor.sample)   # Estimated covariance matrix

# Printing
cat('True mean:', mu)
cat('\nEstimated mean:', mu_est)
cat('\n\nTrue covariance matrix:\n')
sigma
cat('\nEstimated covariance matrix:\n')
sigma_est
```

```
## True mean: 1 2
## Estimated mean: 0.989272 2.003615
##
## True covariance matrix:
##      [,1] [,2]
## [1,]  1.0  0.5
```

```
## [2,]  0.5  1.0
##
## Estimated covariance matrix:
##           [,1]      [,2]
## [1,] 0.9931704 0.4905213
## [2,] 0.4905213 0.9935755
```

We compute the estimated mean vector and covariance matrix from 10000 samples for a 2-variate normal distribution. These seems to coincide with the true values.

Problem B

1.

a)

We consider a gamma distribution with parameters $\alpha \in (0,1)$ and $\beta = 1$. It has probability density function

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

We want to use rejection sampling to generate samples from f by proposing samples from g. Thus we need the acceptance probability $A(x) = \frac{1}{k}\frac{f(x)}{g(x)}$, so we have to find a $k > 1$ such that $\frac{f(x)}{g(x)} \leq k$ for all x where $f(x) > 0$. We evaluate this fraction when $0 < x < 1$ and find that

$$\frac{f(x)}{g(x)} = \frac{e^{-x}}{c\Gamma(\alpha)} \leq \frac{1}{c\Gamma(\alpha)},$$

on this interval. Here, c is the normalizing constant in g. When $x \geq 1$ the fraction becomes

$$\frac{f(x)}{g(x)} = \frac{x^{\alpha-1}}{c\Gamma(\alpha)} \leq \frac{1}{c\Gamma(\alpha)}.$$

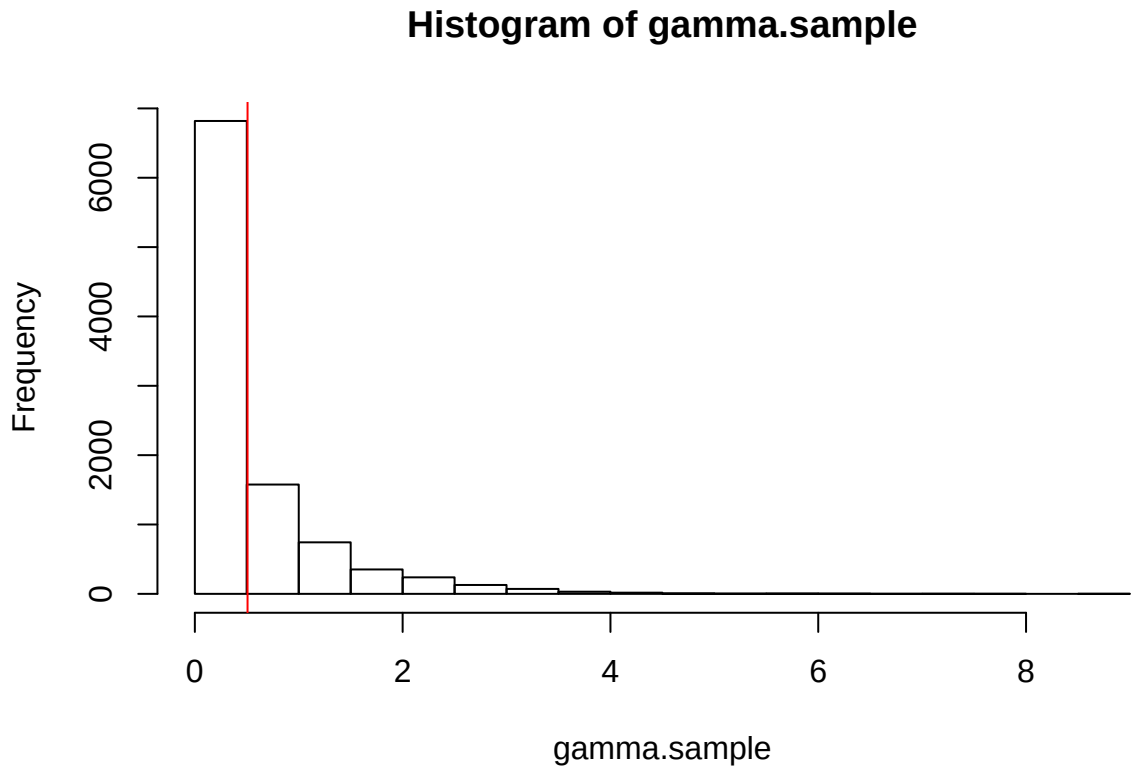
Thus, we set $k = \frac{1}{c\Gamma(\alpha)}$, and the acceptance probability becomes

$$A(x) = \begin{cases} e^{-x}, & 0 < x < 1 \\ x^{\alpha-1}, & x \geq 1 \end{cases}.$$

b)

```r
# Function to calculate the acceptance probability
accprob = function(x, alpha) {
  if(x<1) {return(exp(-x))}
  else if(x>=1) {return(x^(alpha-1))}
}

# Function to generate n samples from f
gamma_rejsamp = function(n, alpha) {
  vec = rep(0, n)              # Vector for storing the accepted samples
  i = 1
  while(i<=n) {
    x = samples_g(alpha, 1)    # Sample from proposal density
    u = runif(1)               # Sample from unif(0,1)
    a = accprob(x, alpha)      # Calculate acceptance probability
    if(u<=a) {                 # Accept x as a sample
```

```
      vec[i] = x              # Store x
      i = i + 1
      }
  }
  return(vec)
}

# Check function
set.seed(60)
alpha = 0.5     # Set parameter
n = 10000       # Number of samples
gamma.sample = gamma_rejsamp(n, alpha)    # Generate samples
hist(gamma.sample)


mean.emp = mean(gamma.sample)   # Empirical mean
abline(v = mean.emp, col = "red")
```




```
mean.theo = alpha               # Theoretical mean

var.emp = var(gamma.sample)     # Empirical variance
var.theo = alpha                # Theoretical variance

print.mean.var(n, mean.emp, mean.theo, var.emp, var.theo)   # Printing values
```

```
## Number of samples: 10000
```

```
##
## Empirical mean: 0.5072649
## Theoretical mean: 0.5
##
## Empirical variance: 0.5181357
## Theoretical variance: 0.5
```

For the 10000 samples from the $Gamma(0.5, 1)$ distribution we make a histogram and compute the empirical and theoretical means and variances. The histogram looks as expected and the empirical mean and variance coincide with the theoretical values.

2.

a)

We consider a gamma distribution with parameters $\alpha > 1$ and $\beta = 1$, and we want to use the ratio of uniforms method to generate samples from this distribution. We define $C_f = \{(x_1, x_2) : 0 \leq x_1 \leq \sqrt{f^*(\frac{x_1}{x_2})}\}$ and

$$f^*(x) = \begin{cases} x^{\alpha-1}e^{-x}, & x > 0 \\ 0, & \text{otherwise} \end{cases}.$$

We also define $a = \sqrt{\sup_x f^*(x)}$, $b_- = -\sqrt{\sup_{x\leq 0}(x^2 f^*(x))}$ and $b_+ = \sqrt{\sup_{x\geq 0}(x^2 f^*(x))}$, so that $C_f \subset [0, a] \times [b_-, b_+]$. Since $f^*(x)$ is bounded and positive for $x > 0$, we find its maximum by differentiating it on this interval and setting it equal to zero. This gives $x = 0$ and $x = \alpha - 1$, where the latter clearly gives the maximum. Thus we find that

$$a = \sqrt{f^*(\alpha - 1)} = \left(\frac{\alpha - 1}{e}\right)^{\frac{\alpha-1}{2}}$$

To find the maximum of $x^2 f^*(x)$ we follow the same procedure, and find that it attains its maximum for $x = \alpha + 1$. Thus,

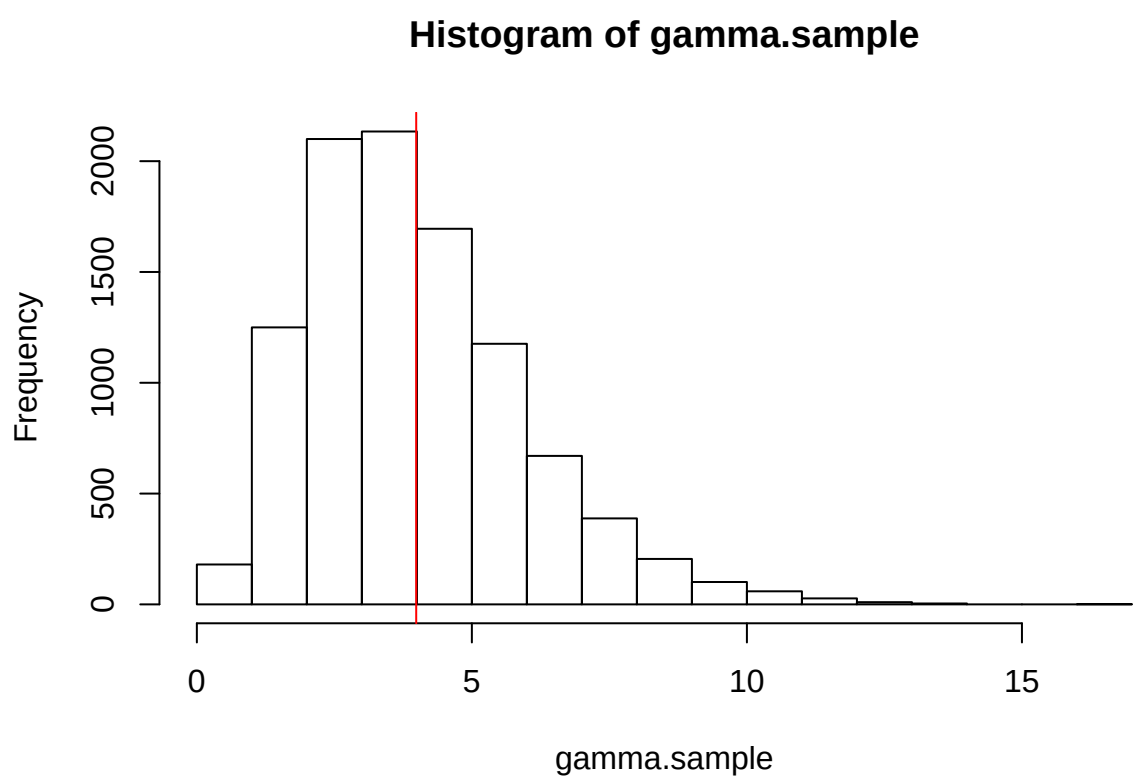
$$b_+ = \sqrt{(\alpha + 1)^2 f^*(\alpha + 1)} = \left(\frac{\alpha + 1}{e}\right)^{\frac{\alpha+1}{2}}$$

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

b)

```
# Function to generate n samples from ln(unif(0,ab))
inv.cdf.sampling = function(n, ln.ab) {
  u = runif(n)     # Sample from unif(0,1)
  x = ln.ab + log(u)   # Using inverse cdf
  return(x)
}

# Function to generate n samples from f
# alpha > 1
# Using log-scale to avoid numerical problems
samples.f = function(n, alpha) {
  vec = rep(0,n)                          # Vector for storing the samples
  i = 1                                   # Count index in vec
  lna = (alpha-1) / 2 * (log(alpha-1) -1) # Value of ln(a)
  lnb = (alpha+1) / 2 * (log(alpha+1) -1) # Value of ln(b_+)
  tries = 0                               # Count number of tries to generate n realizations
  while(i<=n) {
    z1 = inv.cdf.sampling(1,lna)          # Sample from ln(unif(0,a))
```

```r
    z2 = inv.cdf.sampling(1,lnb)          # Sample from ln(unif(b_-,b_+))
    diff = z2 - z1
    limit = ((alpha-1)*diff - exp(diff)) / 2  # Calculate upper limit for z1 in C_f
    if(z1 <= limit) {                     # Accept the sample
      vec[i] = exp(diff)                  # Store ratio of uniforms
      i = i + 1
    }
    tries = tries + 1
  }
  return(c(tries,vec))                    # Return number of tries and vector of samples
}

# Check function
set.seed(100)
alpha = 4       # Set parameter
n = 10000       # Number of samples
gamma.sample = samples.f(n,alpha)[-1]   # Generate samples
hist(gamma.sample)                      # Histogram of samples

mean.emp = mean(gamma.sample)           # Empirical mean
abline(v = mean.emp, col = "red")
```

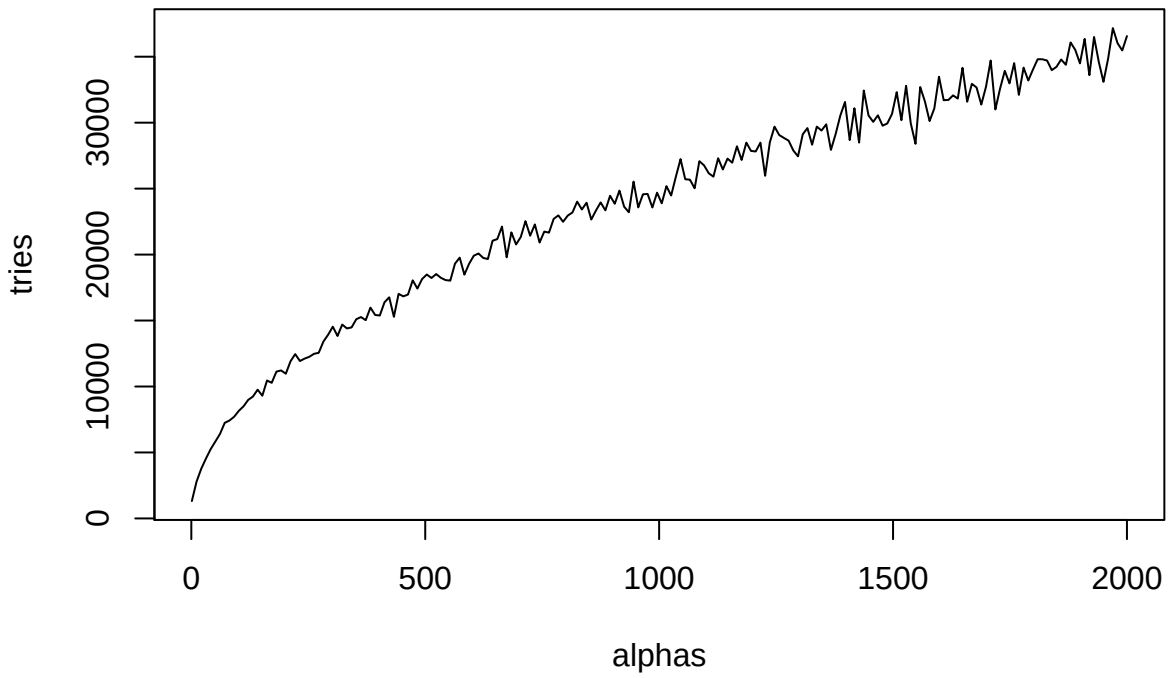



```r
mean.theo = alpha                       # Theoretical mean

var.emp = var(gamma.sample)             # Empirical variance
var.theo = alpha                        # Theoretical variance
```

```r
print.mean.var(n, mean.emp, mean.theo, var.emp, var.theo) # Printing values
```

```
## Number of samples: 10000
##
## Empirical mean: 3.988009
## Theoretical mean: 4
##
## Empirical variance: 3.880826
## Theoretical variance: 4
```

```r
# Number of tries for different alphas
n = 1000                                  # Number of samples to generate
alphas = seq(1.1,2000,length.out = 200)   # Values for alpha
tries = rep(0,length(alphas))             # Vector to store number of tries to generate n samples for dif
for(i in 1:length(alphas)) {
  s = samples.f(1000,alphas[i])
  tries[i] = s[1]
}
plot(alphas, tries, type = "l")                       # Plot of alpha vs number of tries
```



For the 10000 samples from the $Gamma(4,1)$ distribution we compute the empirical and theoretical means and variances. The empirical mean and variance coincide with the theoretical values.

From the figure we observe that the number of tries to generate 10000 samples as a function of α has an approximately logarithmic shape.

3.

```
# Function to generate n samples from gamma distribution
gamma = function(n, alpha, beta) {
  if(alpha<1) {return(gamma_rejsamp(n,alpha) / beta)}       # alpha<1, beta is an inverse scale paramet
  else if(alpha>1) {return(samples.f(n,alpha)[-1] / beta)}  # alpha>1, beta is an inverse scale paramet
  else {return(samples_exp(beta,n))}                        # alpha=1, exponential distribution with ra
}

# Test function

# Function to generate samples and print values
test.gam = function(n, alpha, beta) {
  gam.samp = gamma(n, alpha, beta)
  mean.emp = mean(gam.samp)
  mean.theo = alpha / beta
  var.emp = var(gam.samp)
  var.theo = alpha / beta^2
  cat('alpha=',alpha,'\nbeta=',beta,'\n')
  print.mean.var(n, mean.emp, mean.theo, var.emp, var.theo)
  cat('\n\n\n')
}

# Set parameters
set.seed(30)
n = 10000
alpha = c(0.5, 1, 10)
beta = 2

test.gam(n, alpha[1], beta)
```

```
## alpha= 0.5
## beta= 2
## Number of samples: 10000
##
## Empirical mean: 0.2508258
## Theoretical mean: 0.25
##
## Empirical variance: 0.1237791
## Theoretical variance: 0.125
```

```
test.gam(n, alpha[2], beta)
```

```
## alpha= 1
## beta= 2
## Number of samples: 10000
##
## Empirical mean: 0.4971266
## Theoretical mean: 0.5
##
## Empirical variance: 0.2349315
## Theoretical variance: 0.25
```

```
test.gam(n, alpha[3], beta)
```

```
## alpha= 10
```

```
## beta= 2
## Number of samples: 10000
##
## Empirical mean: 5.006441
## Theoretical mean: 5
##
## Empirical variance: 2.54346
## Theoretical variance: 2.5
```

We generate 10000 samples from the gamma distribution with $\beta = 2$ and different values for α and compute the empirical and theoretical means and variances. The empirical mean and variance seems to coincide pretty well with the theoretical values in all cases.

Problem C

We define

$$x_k = \frac{z_k}{z_1 + \ldots + z_K},$$

where z_k is gamma distributed with parameters $\alpha_k, 1$. Assuming the $z_k;\ k = 1, \ldots, K$ are independent, they have joint distribution:

$$f_Z(z_1, \ldots, z_K; \alpha_1, \ldots, \alpha_K, 1) = \prod_{k=1}^{K} \left(\frac{1}{\Gamma(\alpha_k)} z_k^{\alpha_k - 1} e^{-z_k} \right).$$

We start by defining a mapping $g : (z_1, \ldots, z_K) \to (x_1, \ldots, x_{K-1}, v)$, where $v = z_1 + \ldots + z_K$:

$$(x_1, \ldots, x_{K-1}, v) = g(z_1, \ldots, z_K) = \big(\frac{z_1}{\sum_{k=1}^{K} z_k}, \ldots, \frac{z_{K-1}}{\sum_{k=1}^{K} z_k}, \sum_{k=1}^{K} z_k \big).$$

Then $(z_1, \ldots, z_K) = g^{-1}(x_1, \ldots, x_{K-1}, v) = \big(x_1 \cdot v, \ldots, x_{K-1} \cdot v, (1 - \sum_{k=1}^{K-1} x_k) \cdot v \big)$. From the change of variables theorem, we have that, provided a relation $(x, \ldots, x_n) = g(z_1, \ldots, z_n)$,

$$f_X(x_1, \ldots, x_n) = f_Z\big(g^{-1}(x_1, \ldots, x_n)\big) \mid \mathbf{J} \mid,$$

where $\mid \mathbf{J} \mid$ is the Jacobian of g^{-1} with respect to $\mathbf{X}$. In our case, this is:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial g_1^{-1}}{\partial x_1} & \cdots & \frac{\partial g_1^{-1}}{\partial x_{K-1}} & \frac{\partial g_1^{-1}}{\partial v} \\ \vdots & \ddots & & \vdots \\ \frac{\partial g_K^{-1}}{\partial x_1} & \cdots & \frac{\partial g_K^{-1}}{\partial x_{K-1}} & \frac{\partial g_K^{-1}}{\partial v} \end{bmatrix},$$

Looking at the different elements of the Jacobian:

$$\frac{\partial g_i^{-1}}{\partial x_j} = \frac{\partial}{\partial x_j} x_i \cdot v = \{v \ : \ i = j, 0 \ : \ \text{otherwise}\},$$

$$\frac{\partial g_K^{-1}}{\partial x_i} = \frac{\partial}{\partial x_i} (1 - \sum_{k=1}^{K-1} x_k) \cdot v = -v,$$

$$\frac{\partial g_K^{-1}}{\partial v} = \frac{\partial}{\partial v} (1 - \sum_{k=1}^{K-1} x_k) \cdot v = (1 - \sum_{k=1}^{K-1} x_k),$$

$$\frac{\partial g_k^{-1}}{\partial v} = \frac{\partial}{\partial v} x_k \cdot v = x_k.$$

This gives us the Jacobi matrix

$$\mathbf{J} = \begin{bmatrix} v & 0 & \cdots & 0 & x_1 \\ \vdots & \ddots & & & \vdots \\ 0 & \cdots & \cdots & v & x_{K-1} \\ -v & \cdots & \cdots & -v & (1 - \sum_{k=1}^{K-1} x_k) \end{bmatrix},$$

and we need the determinant of this. We observe that this matrix is a block matrix on the form

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

where A is a $K-1 \times K-1$ diagonal matrix with v on the diagonals, B is $[x_1, \ldots, x_{K-1}]^{\mathrm{T}}$, C is the $1 \times K-1$ vector $[-v, \ldots, -v]$ and D is $(1 - \sum_{k=1}^{K-1} x_k)$. For this type of matrices, we have that

$$\det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix} \right) = (D - CA^{-1}B)\det(A).$$

Since A is diagonal with entries v, $\det(A) = v^{K-1}$. Furthermore, A^{-1} is a diagonal matric with $1/v$ as entries. This gives us

$$D - CA^{-1}B = (1 - \sum_{k=1}^{K-1} x_k) - [-v, \ldots, -v] \times \begin{bmatrix} \frac{1}{v} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{v} \end{bmatrix} \times [x_1, \ldots, x_{K-1}]^{\mathrm{T}}$$

$$= 1 - \sum_{k=1}^{K-1} x_k - (-\sum_{k=1}^{K-1} x_k) = 1.$$

We get

$$\mid \mathbf{J} \mid = v^{K-1}.$$

From the change of variable theorem, we now get

$$f_X(x_1, \ldots, x_{K-1}, v) = f_Z(x_1 \cdot v, \ldots, x_{K-1} \cdot v, (1 - \sum_{k=1}^{K-1} x_k) \cdot v) \cdot v^{K-1} = e^{-(1-\sum_{k=1}^{K-1} x_k)v} \frac{((1 - \sum_{k=1}^{K-1} x_k)v)^{\alpha_K - 1}}{\Gamma(\alpha_K)} \prod_{k=1}^{K-1} \left(\frac{(v \cdot x_k)^{\alpha_k -}}{\Gamma(\alpha_k)} \right.$$

We now have the distribution of $(x_1, \ldots, x_{K-1}, v)$ and integrate this over v to get the desired distribution of $(x_1, \ldots, x_{K-1})$. As only the first two factors are dependent on v, we look at

$$\int_0^\infty e^{-v} \cdot v^{\sum_{k=1}^{K} (\alpha_k) - 1} \mathrm{d}v = \Gamma(\sum_{k=1}^{K} \alpha_k).$$

Inserted into the distribution, we get

$$f_X(x_1, \ldots, x_{K-1}) = \Gamma(\sum_{k=1}^{K} \alpha_k) \cdot \frac{(1 - \sum_{k=1}^{K-1} x_k)^{\alpha_K - 1}}{\Gamma(\alpha_K)} \prod_{k=1}^{K-1} \frac{x_k^{\alpha_k - 1}}{\Gamma(\alpha_k)},$$

which is the Dirichlet distribution with parameters $(\alpha_1, \ldots, \alpha_K)$.

2.

```r
#Function for generating one Diriclet distributed variable with parameter alpha:
dirichlet <- function(alphas){
  zs = rep(0,length(alphas))
  for (k in 1:length(alphas)){
    zs[k] <- gamma(1,alphas[k],1)
  }
  return(zs/sum(zs))
}

#Testing
K = 3
alphas <- runif(K,0.1,5.0)
mean.theo <- alphas/sum(alphas)
dirichlet.samples <- rep(0,K)
for (i in 1:30){dirichlet.samples = dirichlet.samples + dirichlet(alphas)}
dirichlet.samples = dirichlet.samples/30
mean.theo
```

```
## [1] 0.04387331 0.23730507 0.71882161
```

```r
dirichlet.samples
```

```
## [1] 0.06353529 0.24229921 0.69416550
```

Problem D

1.

```r
# Function for ln(f(x))
lnf = function(x) {
  return(125*log(2+x) + 38*log(1-x) + 34*log(x))
}

# Function for ln of the acceptance probability
ln.accprob = function(x) {
  xmax = (15 + sqrt(53809)) / 394   # x value that gives the maximum of ln(f(x))
  lnc = lnf(xmax)                   # Maximum of ln(f(x))
  return(lnf(x) - lnc)
}

multbinom = function(n) {
  vec = rep(0,n)               # Vector to store the n samples
  i = 1
  tries = 0                    # Count number of tries
  while(i<=n) {
    theta = runif(1)           # Sample theta from proposal density
    lnu = log(runif(1))        # ln of sample from unif(0,1)
    lna = ln.accprob(theta)    # ln of acceptance probability
    if(lnu<=lna) {             # Accept sample
      vec[i] = theta
      i = i + 1
    }
    tries = tries + 1
```

```
  }
  return(c(tries,vec))
}
```

2.

The expected value of θ, conditioned on $\mathbf{y}$ is

$$\mathbf{E}[\theta \mid \mathbf{y}] = \int_0^1 \theta f(\theta \mid \mathbf{y}) \mathrm{d}\theta,$$

which gives the Monte Carlo estimator for the expectation

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} \theta_i,$$

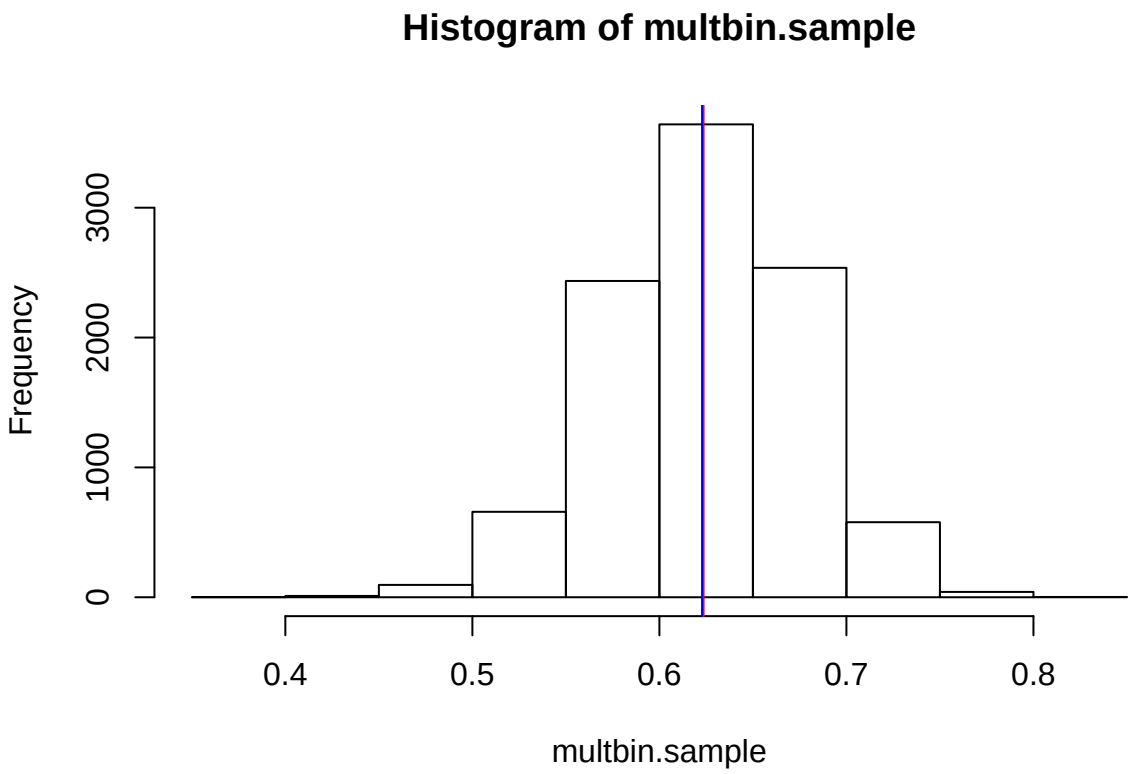
where θ_i are sampeled from $f(\theta \mid \mathbf{y})$ N repetitions.

To test the accuracy of this estimator, we find the solution of the integral numerically. We have that $f(\theta \mid \mathbf{y}) \propto f(\mathbf{y} \mid \theta) f(\theta)$, with $f(\mathbf{y})^{-1}$ as normalizing constant. We use $f(\mathbf{y})^{-1} = \int_0^1 f(\mathbf{y} \mid \theta) f(\theta)$ with the uniform prior of θ and get

```
multbin = multbinom(10000)            #sample from multinomial
multbin.tries = multbin[1]            #no of attempts
multbin.sample = multbin[-1]          #samples
multbin.mean.MC <- mean(multbin.sample)   #Monte Carlo estimator of exepctation

#Function for integrand to get prior of y
multbin.prior <- function(x){
  return((2+x)^125*(1-x)^38*x^34)
}

#function for integrand to get expectation of theta
multbin.integrand <- function(x){
  c <- integrate(multbin.prior,0,1)                 #gives f(y)
  return((1/(c$value))*x*(2+x)^125*(1-x)^38*x^34)   #gives theta*f(theta|y)
}

multbin.mean.exact <- integrate(multbin.integrand,0,1)  #numerical integral, theta from 0 to 1
multbin.mean.exact
```

```
## 0.6228061 with absolute error < 1.7e-07
```

```
#Histogram of sampled values of theta
hist(multbin.sample)
abline(v = multbin.mean.MC, col="red")        #Monte Carlo integration mean
abline(v = multbin.mean.exact$value, col="blue")    #Expectation from numerical integration
```



3.

The theoretical overall acceptance rate in importance sampling is given by

$$\mathbf{P}(\text{acceptance}) = \frac{1}{c}.$$

```
#Average number of attempts before getting an accepted sample
tries.per.sample = multbin.tries / length(multbin.sample)
tries.per.sample
```

```
## [1] 7.8257
```

```
#The theoretical value of the average number of samples needed to get acceptance:
norm.const <- integrate(multbin.prior,0,1)     #the normalizing constant, f(y)
lg.norm.const <- log(norm.const$value)         #log-scaling normalizing constant
xmax = (15 + sqrt(53809)) / 394                #x-val wchich gives maximum f(theta|y)
lg.overall.acceptance <- lnf(xmax)             #overall acceptance c, f(xmax)
lgc <- lg.overall.acceptance - lg.norm.const   #normalized c, log-scale
tries.per.sample.theo <- exp(lgc)              #taking the exponential of the log-scale c
tries.per.sample.theo
```

```
## [1] 7.799308
```

The actual average number of attempts before accepting a value as a sample conicides with the theoretical overall acceptance rate 1/c, i.e. the theoretical number of attemps before a sample is accepted.

4.

We want to use a new prior distribution of θ:

$$\text{Beta}(\theta; 1, 5) = 5(1-\theta)^4.$$

We have N samples and $\theta_i \sim f(\theta \mid \mathbf{y})|_{f(\theta)=\text{Beta}(1,1)}$, and we want to find the expectation

$$\mathbf{E}[\theta \mid \mathbf{y}]|_{f(\theta)=\text{Beta}(1,5)}.$$

The self-normalizing importance sampling estimator gives us

$$\mathbf{E}[\theta \mid \mathbf{y}]|_{f(\theta)=\text{Beta}(1,5)} \approx \hat{\mu}_{IS} = \frac{\sum_{i=1}^{N} \theta_i \cdot w(\theta_i)}{\sum_{i=1}^{N} w(\theta_i)}$$

```
set.seed(50)
#Monte Carlo estimator of theta given y, with Beta(1,5) as prior of theta
multbin.beta15.IS <- sum(multbin.sample*(1-multbin.sample)^4)/sum((1-multbin.sample)^4)
multbin.beta15.IS
```

```
## [1] 0.5963192
```

The $\hat{\mu}_{IS}$ is approximately 0.6, which is also quite close to the expectation of θ given with a Beta(1,1) prior distribution. This is expected, as the prior distribution should have a decreasing influence on the prosterior distribution, with an increasing amount of observations.