Computer Intensive Statistical Methods-Exercise 1

Vebjørn Rekkebo, Camilla Karlsen

04.02.2020

Problem A: Stochastic simulation by the probability integral transform and bivariate techniques

The inverse of the exponential cumulative density function with rate parameter λ is given by

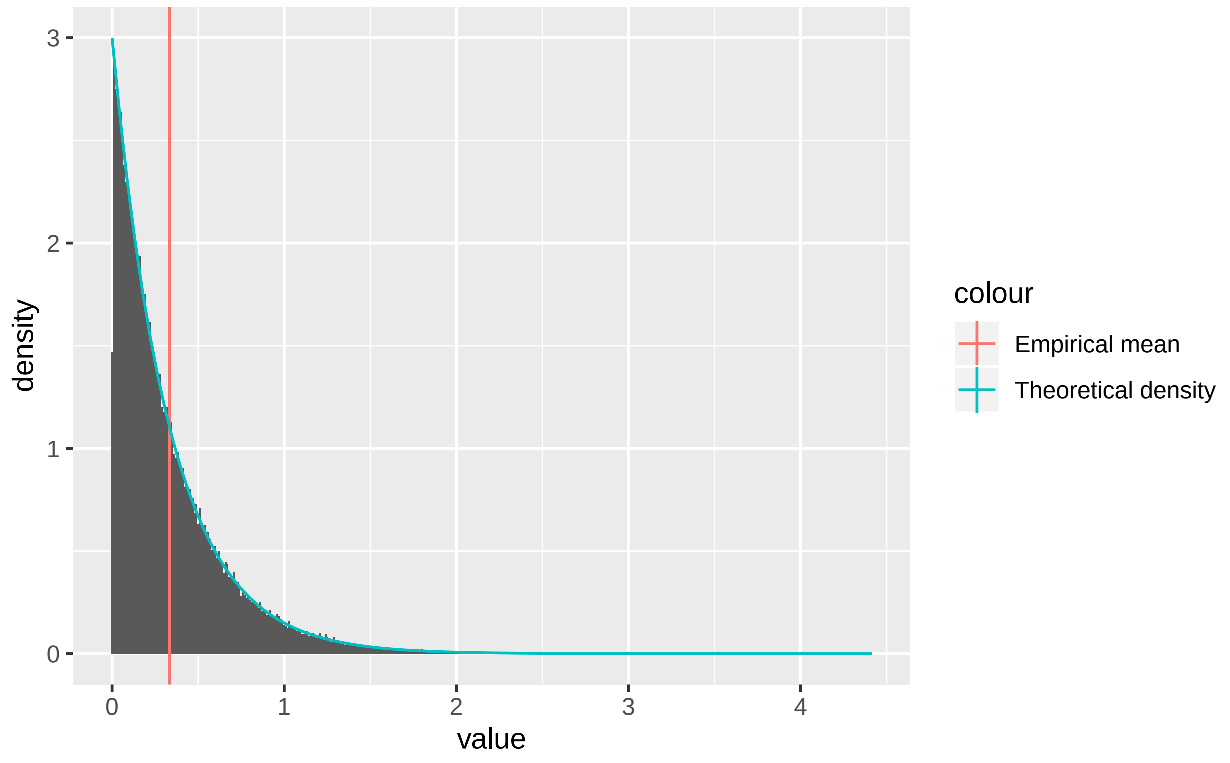
$$F^{-1}(u) = -\frac{1}{\lambda}\log(u).$$

Hence we can sample from the exponential distribution by generating $u \sim \mathcal{U}[0,1]$ and returning $-\frac{1}{\lambda}\log(u)$.

```r
#This function returns a vector of n independent samples from the exponential distribution
#with rate parameter lam, generated from the inverse cumulative distribution.
random.exp <- function(n,lam){
  return(-1/lam * log(runif(n)))
}
```

We know that $\text{E}(X) = \frac{1}{\lambda}$ and $\text{Var}(X) = \frac{1}{\lambda^2}$. The empirical mean is given by the average of the sample and the empirical variance is $\widehat{\text{Var}(X)} = \frac{\sum_{i=1}^n x_i}{n-1}$.

```r
#Function for comparing empirical mean and variance with theoretical values
#n=number of samples, func=sampling function, arg=arguments to func
#amean and avar=analytical mean and variance
test.function <- function(n,func,arg,amean,avar){
  exs.sample=func(n,arg)
  cat("Empirical mean:", mean(exs.sample), "Analytical mean:",amean, "\n")
  cat("Empirical variance:", var(exs.sample), "Analytical variance:",avar,"\n")
}

#Make histogram of sample
lam <- 3
sample.exp <- enframe(random.exp(100000,lam))

ggplot() +
  geom_histogram(
    data = data.frame(sample.exp),
    mapping = aes(x=value, y=..density..),
    binwidth = 0.01
  ) +
  geom_vline(
    aes(xintercept=mean(sample.exp$value),
        col='Empirical mean')
  ) +
  stat_function(
    fun = dexp,
    args = list(rate=lam),
    aes(col='Theoretical density')
  ) +
  labs(
    caption = expression(atop("Histogram of sample from exponential distribution with ",
                        paste(lambda*"=3 compared to theoretical density.")))
  )
```



Histogram of sample from exponential distribution with

λ=3 compared to theoretical density.

The histogram fits very well with the theoretical density function and the mean is close to $\frac{1}{\lambda} = \frac{1}{3}$ as it should be.

```
test.function(10000,random.exp,lam,1/lam,1/lam^2)
```

```
## Empirical mean: 0.3331241 Analytical mean: 0.3333333
## Empirical variance: 0.1144668 Analytical variance: 0.1111111
```

The printout shows that the empirical mean and variance are close to the analytical values so the implementation seems correct.

We now consider the density function

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

where c is a normalising constant and $\alpha \in (0,1)$. Integration over the different intervals yields the cumulative distribution

$$G(x) = \begin{cases} \frac{c}{\alpha}x^{\alpha}, & 0 < x < 1, \\ c(\alpha^{-1} + e^{-1} - e^{-x}) & 1 \leq x, \\ 0, & \text{otherwise}. \end{cases}$$

We can now set $u = G(x)$ and solve for x

$$\begin{cases} u = \frac{c}{\alpha}x^{\alpha} & \iff x = (\frac{\alpha}{c}u)^{\frac{1}{\alpha}} \\ u = c(\alpha^{-1} + e^{-1} - e^{-x}) & \iff x = -\log(\alpha^{-1} + e^{-1} - \frac{u}{c}) \end{cases}$$

and the boundary $x = 1$ is equivalent to $u = \frac{c}{\alpha}$. Hence the inverse of the cumulative distribution is

$$G^{-1}(u) = \begin{cases} (\frac{\alpha}{c}u)^{\frac{1}{\alpha}}, & 0 < u < \frac{c}{\alpha}, \\ -\log(\alpha^{-1} + e^{-1} - e^{-x}), & \frac{c}{\alpha} \leq u, \\ 0, & \text{otherwise.} \end{cases}$$

The normalising constant c is found by integrating $g(x)$ over its domain and let the integral equal 1.

$$\int_0^{\infty} g(x)dx = c(\alpha^{-1} + e^{-1}) = 1 \iff c = \frac{\alpha e}{\alpha + e}.$$

Now we can sample from $g(x)$ by generating $u \sim \mathcal{U}[0, 1]$ and returning $G^{-1}(u)$.

```r
#Function for normalising constant in g distribution given parameter a (=alpha).
c.g <- function(a){
  return (a*exp(1)/(a+exp(1)))
}

#Normalised density function for g
#Used only to compare with histogram
g <- function(x, a) {
  c <- c.g(a)
  out <- rep(0, length(x))
  left <- 0 < x & x < 1
  right <- 1 <= x
  out[left] <- c * (x[left] ^ (a - 1))
  out[right] <- c * exp(-x[right])
return(out)
}

#This function returns a vector of n independent samples from the g distribution with
#parameter a (=alpha), generated from the inverse cumulative distribution.
random.g <- function(n,a){
  c <- c.g(a) #Normalising constant
  u <- runif(n)
  x <- rep(0,n)
  for (i in 1:n){
    if (u[i]<c/a){
      x[i] <- (a*u[i]/c)^(1/a)
    }
    else{
      x[i] <- (-log(1/a + exp(-1) - u[i]/c))
    }
  }
  return(x)
}
```

The analytical mean and variance of $X \sim g(x)$ are computed in the usual way

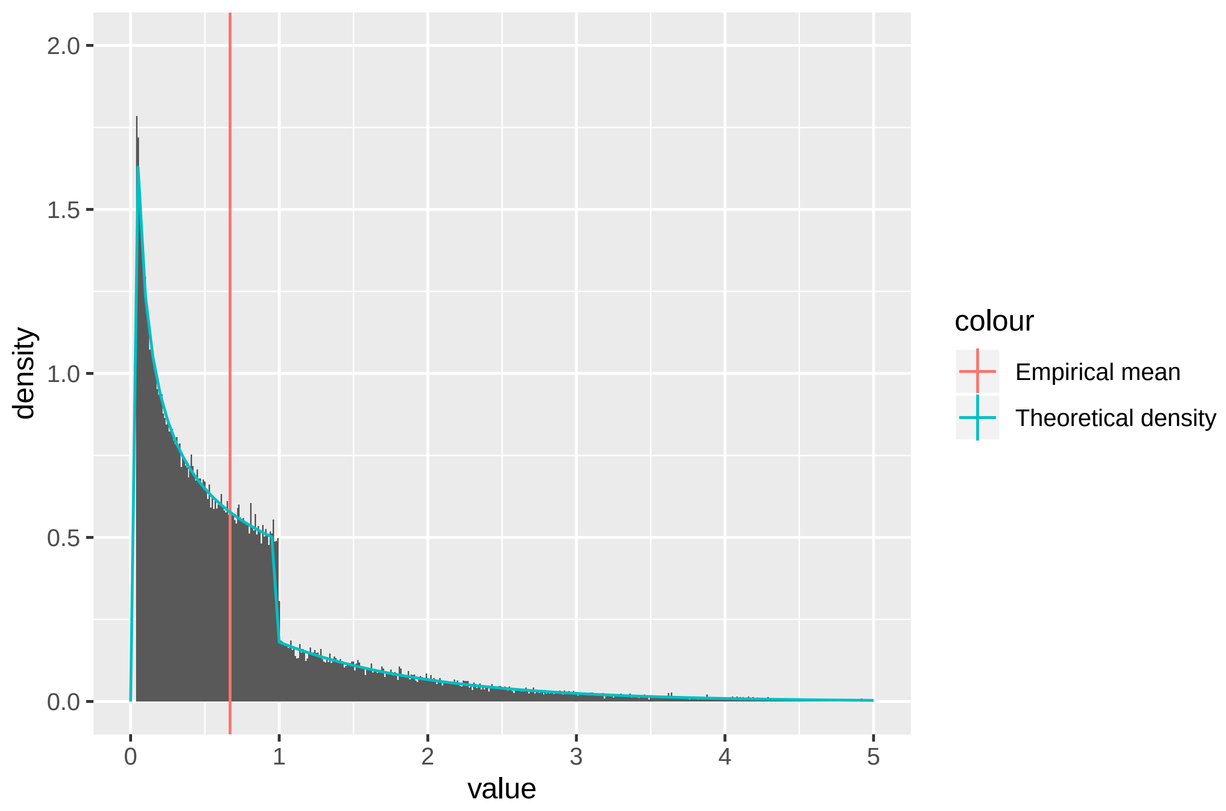
$$\begin{aligned}
\mathrm{E}(X) &= \int_0^\infty x g(x) dx = \frac{c}{\alpha+1} + \frac{2c}{e}, \\
\mathrm{E}(X^2) &= \int_0^\infty x^2 g(x) dx = \frac{c}{\alpha+2} + \frac{5c}{e}, \\
\mathrm{Var}(X) &= \mathrm{E}(X^2) - (\mathrm{E}(X))^2 \\
&= c(\frac{1}{\alpha+2} + \frac{5}{e}) - (\frac{c}{\alpha+1} + \frac{2c}{e})^2.
\end{aligned}$$

```
#Testing sampling function with n=10000 samples
a <- .6 #parameter
c <- c.g(a) #normalising constant
g.mean <- c/(a+1)+2*c*exp(-1) #analytical mean
g.var <- c*(1/(a+2)+ 5*exp(-1)) - g.mean^2 #analytical variance
test.function(10000,random.g,a,g.mean,g.var)
```

```
## Empirical mean: 0.6800451 Analytical mean: 0.6688268
## Empirical variance: 0.6691641 Analytical variance: 0.6457955
```

The mean and variance of the sample are close to the theoretical values.

```
#Making histogram of sample from g
sample.g <- enframe(random.g(100000,a))

ggplot() +
  geom_histogram(
    data = data.frame(sample.g),
    mapping = aes(x=value, y=..density..),
    binwidth = 0.01,
    na.rm = TRUE
  ) +
  geom_vline(
    aes(xintercept=mean(sample.g$value),
        col='Empirical mean')
  ) +
  stat_function(
    fun = g,
    args = list(a=a),
    aes(col='Theoretical density')
  ) +
  ylim(0,2) +
  xlim(0,5) +
    labs(
      caption = bquote("Histogram of sample from g-distribution with "
                       ~alpha*"=0.6 compared to theoretical density.")
    )
```



Histogram of sample from g−distribution with α=0.6 compared to theoretical density.

The histogram and density function coincide and that further indicates that the implementation is correct. Generating even more samples would smoothen the histogram even more.

The Box-Müller algorithm samples from the bivariate standard normal distribution. We first generate $x_1 \sim \mathcal{U}(0, 2\pi)$ and $x_2 \sim \exp(\frac{1}{2})$ and then return $y_1 = \sqrt{x_2}\cos(x_1)$ and $y_2 = \sqrt{x_2}\sin(x_1)$. We want to generate a vector of n independent samples from the standard normal distribution. Thus, for simplicity, we only keep y_1 from each bivariate sample.

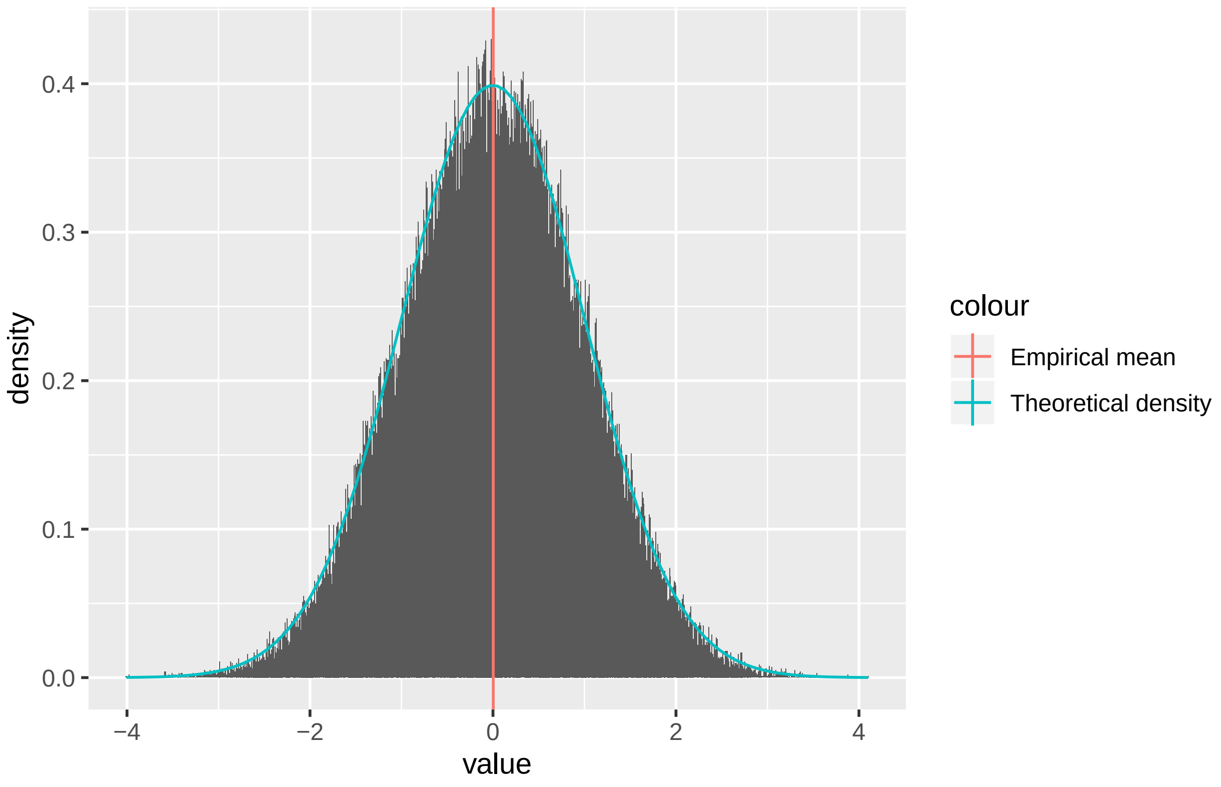
```r
#This function returns a vector of n independent samples from the standard normal
#distribution, generated by the Box-Müller algorithm
random.norm <- function(n){
  x1 <- runif(n)*2*pi
  x2 <- random.exp(n,.5)
  return(sqrt(x2)*cos(x1))
}

#Testing sampling function for standard normal distribution
sample.normal=enframe(random.norm(100000))
cat("Empirical mean:", mean(sample.normal$value), "Analytical mean:",0.0, "\n")
```

```
## Empirical mean: 0.002900696 Analytical mean: 0
```

```r
cat("Empirical variance:", var(sample.normal$value), "Analytical variance:",1.0,"\n")
```

```
## Empirical variance: 0.994302 Analytical variance: 1
```

```r
ggplot() +
  geom_histogram(
    data = sample.normal,
    mapping = aes(x=value, y=..density..),
    binwidth = 0.01
  ) +
  geom_vline(
    aes(xintercept=mean(sample.normal$value),
        col='Empirical mean')
  ) +
  stat_function(
    fun = dnorm,
    aes(col='Theoretical density')
  ) +
  labs(
    caption = "Histogram of sample from standard normal distribution
              compared to theoretical density."
  )
```



Histogram of sample from standard normal distribution compared to theoretical density.

The empirical mean and variance are close to the analytical values so the implementation seems correct. The histogram of the samples is also just like expected.

Now we want to sample from a d-variate normal distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. This is done by generating a vector $\boldsymbol{x}$ of d standard normal samples and doing the transform $\boldsymbol{y} = \mathbf{A}\boldsymbol{x} + \boldsymbol{\mu}$ where $\mathbf{A}\mathbf{A}^{\mathbf{T}} = \boldsymbol{\Sigma}$. We find $\mathbf{A}$ by the Cholesky decomposition of $\boldsymbol{\Sigma}$.

```
#This function returns a sample from the d-variate normal distribution with mean mu
#and covariance S, transformed from standard normal samples.
random.multinorm <- function(d,mu,S){
  x <- random.norm(d)
  A <- t(chol(S)) #transpose to get lower triangular cholesky matrix
  return(mu + A%*%x)
}

#Testing d-variate normal distribution sampler with 10000 samples
d = 3
N = 10000
sigma <- cbind(c(3,1,0), c(1,3,1), c(0,1,3)) #if d=3
mu <- seq(1,d)
multinormal.sample <- matrix(NA,N,d)
for (i in 1:N){
  multinormal.sample[i,] <- random.multinorm(d, mu, sigma)
}
#Theoretical mean vector:
mu
```

```
## [1] 1 2 3
```

```
#Empirical mean vector:
colMeans(multinormal.sample)
```

```
## [1] 1.011259 2.007344 2.996438
```

```
#Empirical covariance matrix:
sigma
```

```
##      [,1] [,2] [,3]
## [1,]    3    1    0
## [2,]    1    3    1
## [3,]    0    1    3
```

```
#Theoretical covariance matrix:
var(multinormal.sample)
```

```
##           [,1]     [,2]      [,3]
## [1,]  2.996461 1.006559 -0.013109
## [2,]  1.006559 3.078220  1.027384
## [3,] -0.013109 1.027384  3.008295
```

We see that the empirical moments are very close to the real values, as expected.

Problem B: The gamma distribution

First, we consider the gamma distribution with parameters $\alpha \in (0,1)$ and $\beta = 1$. To generate samples from $f(x)$, we use rejection sampling with $g(x)$ as proposal density. We need to find a $k > 1$ such that $k \geq \frac{f(x)}{g(x)}$.

$$\frac{f(x)}{g(x)} = \begin{cases} \frac{1}{c\Gamma(\alpha)} e^{-x}, & 0 < x < 1 \\ \frac{1}{c\Gamma(\alpha)} x^{\alpha-1}, & 1 \leq x \end{cases} \leq \frac{1}{c\Gamma(\alpha)} = k$$

Hence, the acceptance probability in the rejection sampling is

$$\alpha = \frac{1}{k}\frac{f(x)}{g(x)} = c\Gamma(\alpha)\frac{f(x)}{g(x)} = \begin{cases} e^{-x}, & 0 < x < 1 \\ x^{\alpha-1}, & 1 \leq x \end{cases}.$$

```r
#Acceptance probability
accept_prob <- function(x,a){
  if (x<1)
      return (exp(-x))
    else
      return (x^(a-1))
}
```

```r
#This function returns a vector of n independent samples from the gamma distribution
#with 0<a<1, beta=1, generated by the Rejection sampling
random.gamma1 <- function(n,a){
  x=rep(0,n)
  for (i in 1:n){
    finish = 0
    while(finish==0){
      # Sample from the proposal distribution, g(x) (x>=0)
      sample.x = random.g(1,a)
      # Compute the acceptance probability alpha
      alpha=accept_prob(sample.x,a)
      # Generate a helping variable U from a uniform(0,1)
      u = runif(1, 0, 1)
      # Check if we accept it
      if(u <= alpha){
        x[i]=sample.x
        finish=1
      }
    }
  }
  return (x)
}

#Testing function for gamma sampling with 0<alpha<1 and 10000 samples
alpha=0.5
test.function(10000,random.gamma1,alpha,alpha,alpha)
```

```
## Empirical mean: 0.5073222 Analytical mean: 0.5
## Empirical variance: 0.5363278 Analytical variance: 0.5
```

The printout shows that the empirical mean and variance are close to the analytical values so the implementation seems correct.

Now we consider the gamma distribution with parameters $\alpha > 1$ and $\beta = 1$. To use ratio of uniforms method to simulate from this distrubution, we need to find $C_f \subset [0, a] \times [b_-, b_+]$. First we find the value of $a = \sqrt{sup_x f^*(x)}$.

$$\frac{d}{dx} f^*(x) = (\alpha-1)x^{\alpha-2}e^{-x} - e^{-x}x^{\alpha-1} = e^{-x}x^{\alpha-2}(\alpha-1-x) = 0 \Rightarrow x = \alpha - 1 \vee x = 0$$

$f^*(x)$ has its maximum when $x = \alpha - 1$, hence $a = \sqrt{f^*(\alpha-1)} = \sqrt{(\alpha-1)^{\alpha-1}e^{-\alpha+1}}$.

Then, we have to find the values of $b_+ = \sqrt{sup_{x\geq 0} x^2 f^*(x)}$ and $b_- = -\sqrt{sup_{x\leq 0} x^2 f^*(x)}$. Since $f^*(x) = 0$ for $x \leq 0$, we have $b_- = 0$.

$$\frac{d}{dx} x^2 f^*(x) = e^{-x}x^{\alpha}(\alpha+1-x) = 0 \Rightarrow x = \alpha + 1 \vee x = 0$$

$x^2 f^*(x)$ has its maximum when $x = \alpha + 1$, hence $b_+ = \sqrt{(\alpha+1)f^*(\alpha+1)} = \sqrt{(\alpha+1)^{\alpha+1}e^{-\alpha-1}}$.

To avoid overflow, we implement the algorithm on log-scale, $u = log(x)$. Hence, we define $C_f = \{(u_1, u_2) : 0 \leq u_1 \leq 0.5 f^*(e^{u_2 - u_1})\}$ where

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

```r
logf <- function(u,alpha){ #log of f* with u=log(x)
  return (u*(alpha-1)-exp(u))
}

#This function sample from the distribution of log(x), using
#the inverse of the cumulative distribution.
sample <- function(ab){
  u=log(runif(1,0,1))
  return (ab+u)
}
```

```r
#This function returns a vector of n independent samples from the gamma distribution
#with a>1, beta=1, generated by the ratio of uniforms method on log-scale.
random.gamma2 <- function(n,alpha){
  count=0 #count how many tries the algorithm needs to generate n realisations
  x=rep(0,n)
  a=(alpha-1)/2*(log(alpha-1)-1) #log(a)
  b=(alpha+1)/2*(log(alpha+1)-1) #log(b+)
  for (i in 1:n){
    inside=FALSE
    while (inside==FALSE){
      u1=sample(a)
      u2=sample(b)
      u=u2-u1
      test=0.5*logf(u,alpha)
      inside=test>=u1
      count=count+1
    }
    x[i] = u
  }
  vec=c(count,exp(x)) #count + n realisations
}

#Testing gamma sampler with alpha>1 and 1000 samples
alpha=2000
ex.gamma2=random.gamma2(1000,alpha)
cat("Empirical mean:",mean(ex.gamma2[-1]), "Analytical mean:",alpha, "\n")
```

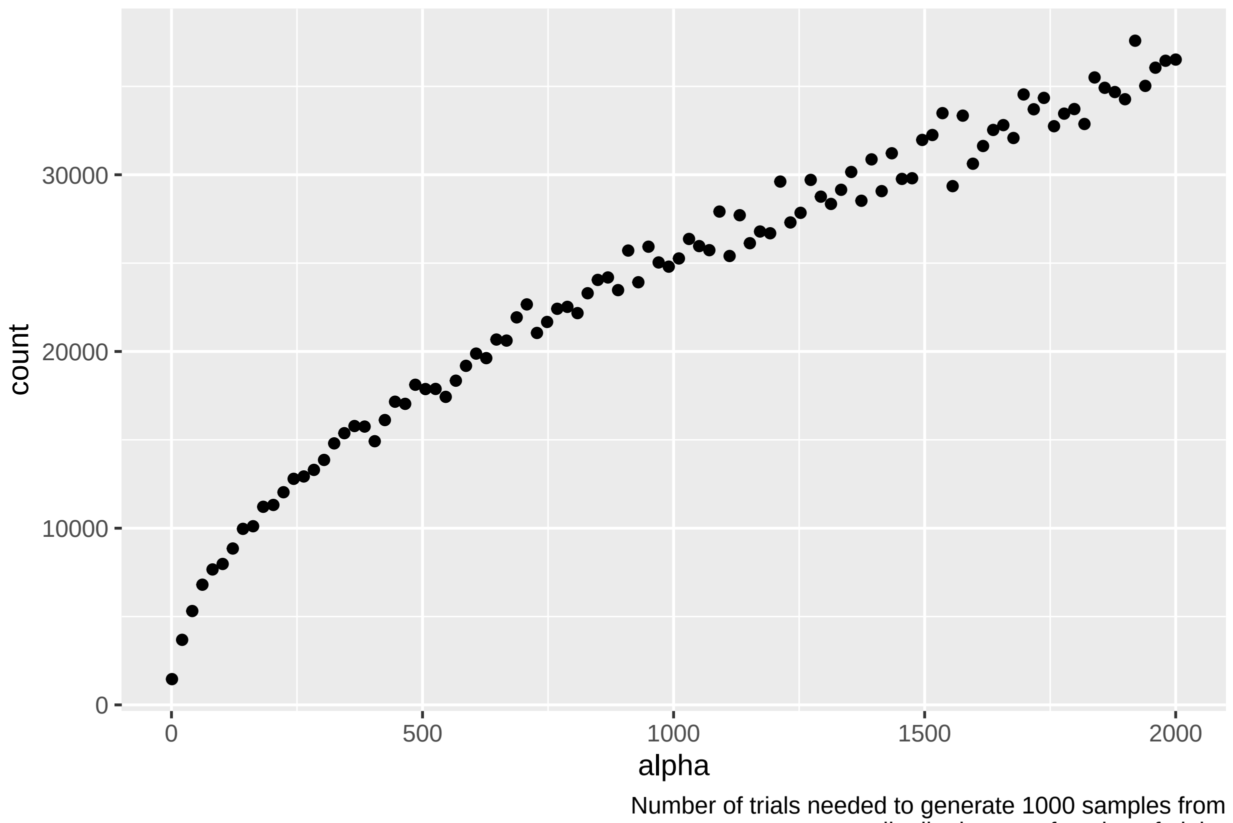
```
## Empirical mean: 2000.009 Analytical mean: 2000
```

```r
cat("Empirical variance:", var(ex.gamma2[-1]), "Analytical variance:",alpha,"\n")
```

```
## Empirical variance: 2002.635 Analytical variance: 2000
```

The empirical mean and variance are close to the analytical values so the implementation seems correct.

```r
#Generate a plot with values of alpha on the x-axis and the number of tries used
#on the y-axis.
plot.count <- function(nsample, n){
  count=rep(0,n)
  alpha=seq(1.01,2000,length.out=n)
  for (i in 1:n){
    a=alpha[i]
    c=random.gamma2(nsample,a)[1]
    count[i]=c
  }
  return(cbind(alpha,count))
}
num.trials <- plot.count(1000, 100)
num.trials.frame <- data.frame(num.trials)
ggplot() +
  geom_point(
    data=num.trials.frame,
    mapping = aes(x=alpha, y=count)
  ) +
  labs(
    caption = "Number of trials needed to generate 1000 samples from
    gamma distribution as a function of alpha"
  )
```



Number of trials needed to generate 1000 samples from gamma distribution as a function of alpha

From the plot we can see that the number of tries the algorithm needs to generate $n = 1000$ realisations increases almost logarithmic with the value of α. The acceptance probability decreases as the value of α increases, since the proportion of the total area $[0, a] \times [0, b_+]$ covered by C_f shrinks as α increases. The proportion shrinks fastest in the beginning and tapers off when alpha becomes larger.

By using the above functions and $X \sim Ga(\alpha, 1) \Leftrightarrow X/\beta \sim Ga(\alpha, \beta)$, we write a function that generates a vector of n independent samples from the gamma distribution with parameters α and β.

```r
#This function returns a vector of n independent samples from the gamma distribution
#with parameters alpha and beta.
random.gamma <- function(n, alpha, beta){
  if (alpha==1){
    x=random.exp(n,alpha) #sample from exponential distribution
  }
  else if (alpha<1){
    x=random.gamma1(n,alpha)
  }
  else {
    x=random.gamma2(n,alpha)[-1] #ignoring the first element, which is the number of tries
  }
  return (x/beta) #Transform from x~Ga(alpha,1) to x/beta~Ga(alpha,beta)
}

#Generic function for testing complete gamma sampling function
test.gamma <- function(n,alpha, beta){
  ex.gamma=random.gamma(n,alpha,beta)
  amean=alpha/beta #analytical mean
  avar=alpha/beta^2 #analytical variance
  cat("Empirical mean:", mean(ex.gamma), "Analytical mean:",amean, "\n")
  cat("Empirical variance:", var(ex.gamma), "Analytical variance:",avar,"\n")
}

#Genereric function for plotting histogram and density of gamma distribution
plot.gamma <- function(n,alphav,betav){
  sample.gamma <- enframe(random.gamma(n,alpha=alphav,beta=betav))
  ggplot() +
    geom_histogram(
      data = data.frame(sample.gamma),
      mapping = aes(x=value, y=..density..),
      binwidth = 0.01,
      boundary = 0,
      na.rm = TRUE
    ) +
    geom_vline(
      aes(xintercept=mean(sample.gamma$value),
          col='Empirical mean')
    ) +
    stat_function(
      fun = partial(dgamma, shape=alphav, rate=betav),
      aes(col='Theoretical density')
    ) +
    ylim(0,3) +
    xlim(0,5) +
    labs(
      caption = bquote("Histogram of sample from gamma distribution with"
                       ~ alpha *"="* .(alphav) ~ " and "
                       ~ beta *"="* .(betav))
    )
}
#Testing for each of the three cases
test.gamma(100000,0.5,0.7)
```

```
## Empirical mean: 0.7143184 Analytical mean: 0.7142857
## Empirical variance: 1.014287 Analytical variance: 1.020408
```

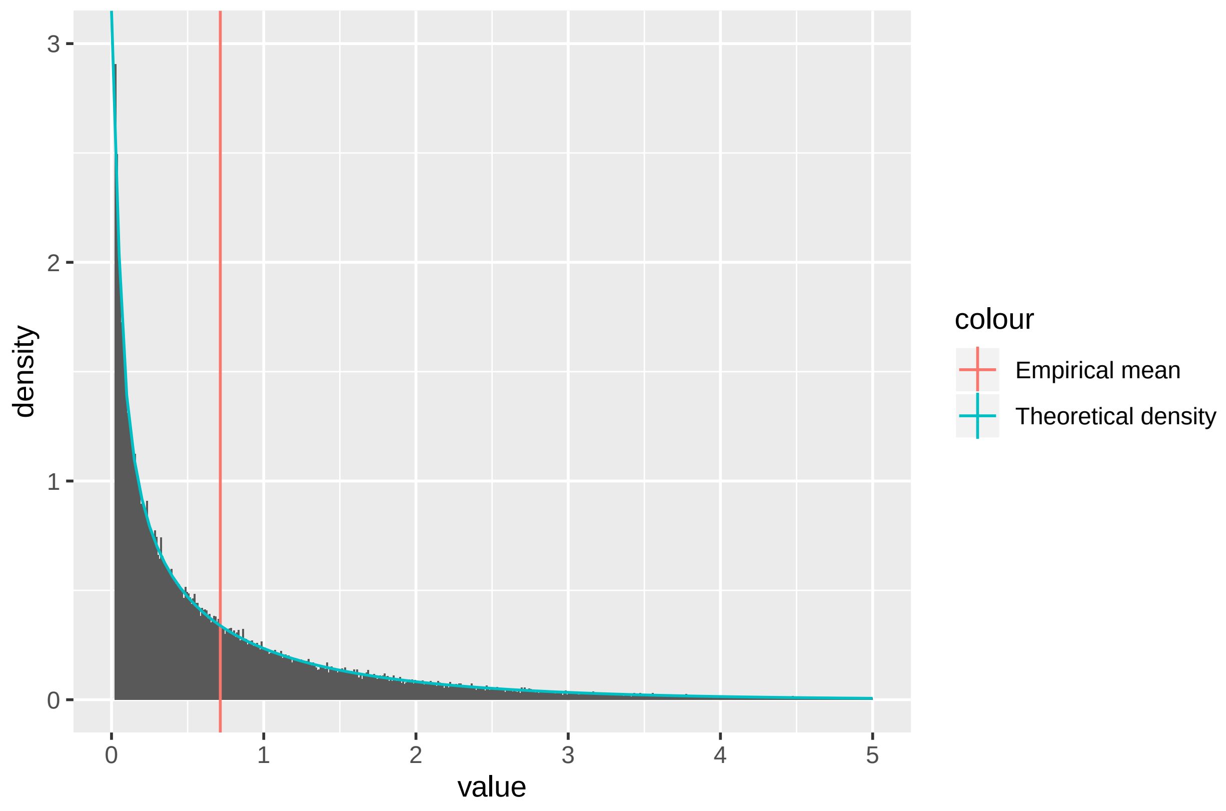
```
test.gamma(100000,1,3)
```

```
## Empirical mean: 0.3324531 Analytical mean: 0.3333333
## Empirical variance: 0.1106543 Analytical variance: 0.1111111
```

```
test.gamma(100000,4,10)
```

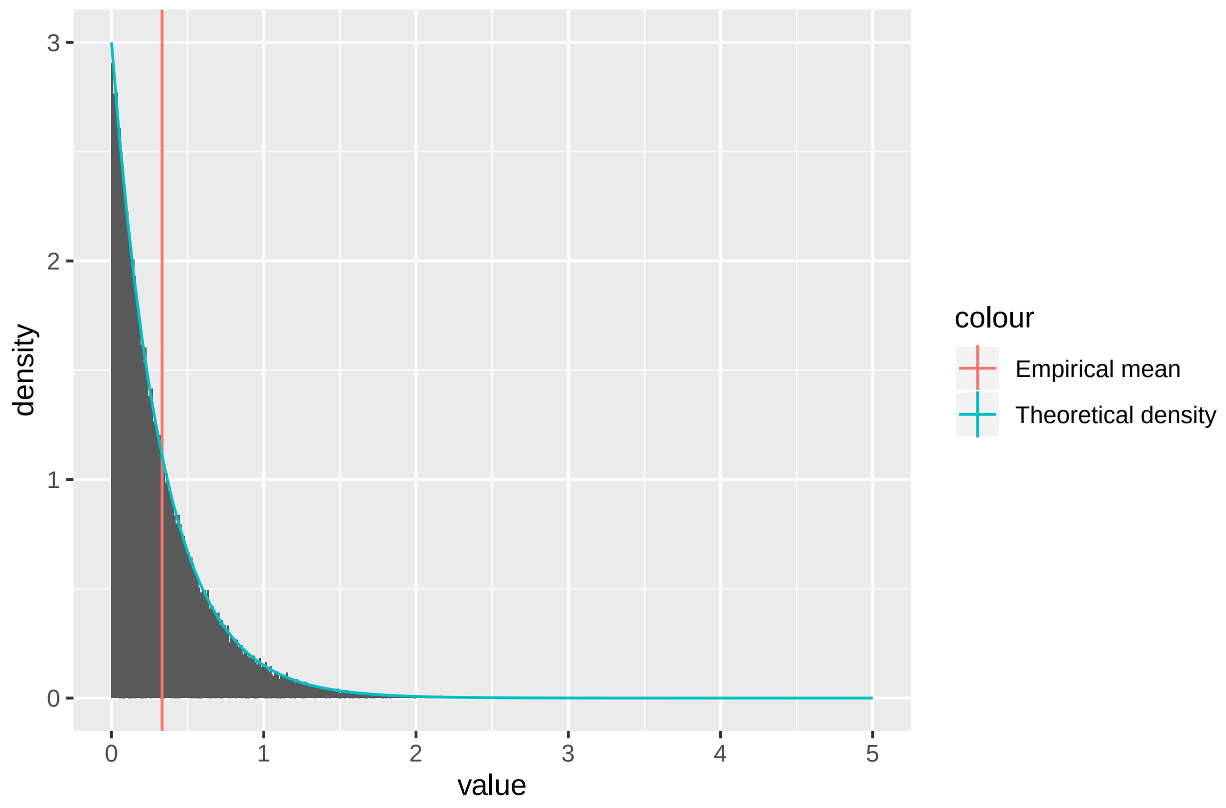
```
## Empirical mean: 0.4004707 Analytical mean: 0.4
## Empirical variance: 0.03991571 Analytical variance: 0.04
```

```
plot.gamma(100000,0.5,0.7)
```



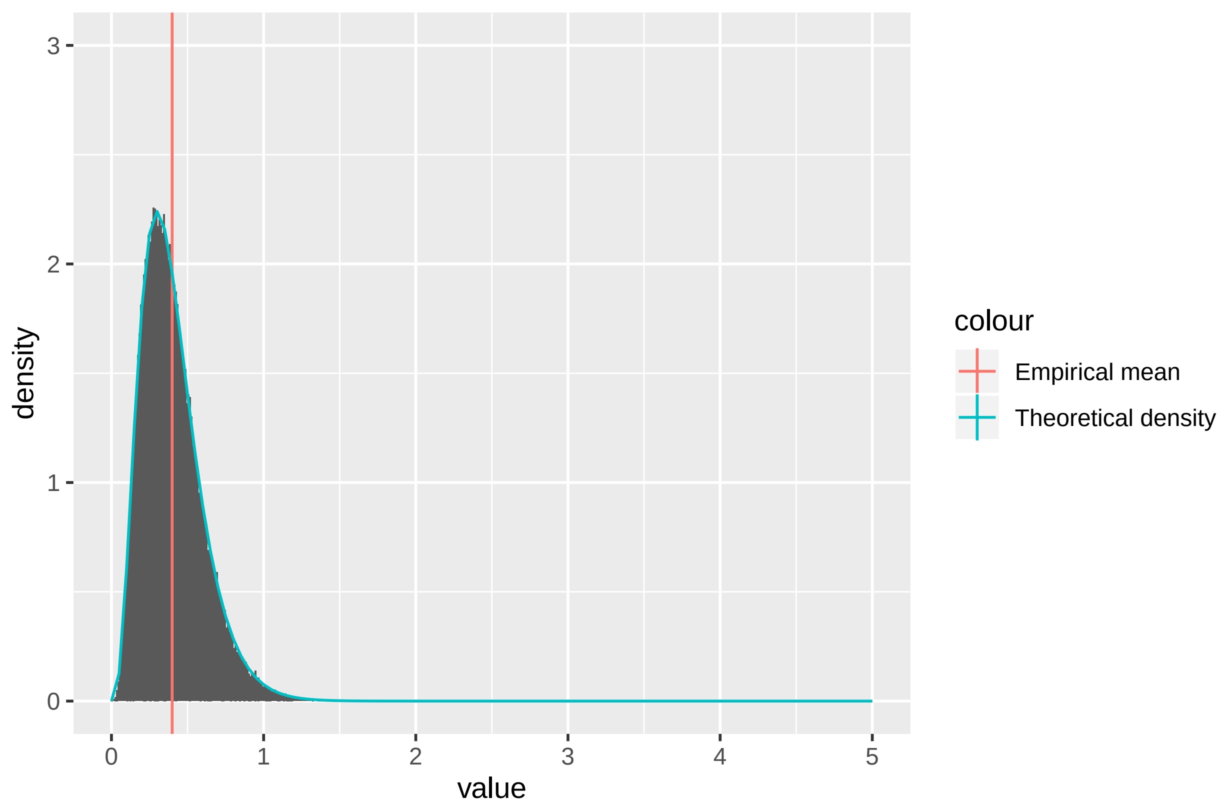
Histogram of sample from gamma distribution with α=0.5 and β=0.7

```
plot.gamma(100000,1,3)
```



Histogram of sample from gamma distribution with α=1 and β=3

```
plot.gamma(100000,4,10)
```



Histogram of sample from gamma distribution with α=4 and β=10

We made three plots to test the different cases of the gamma-distribution, one for each of $\alpha < 1$, $\alpha = 1$ and $\alpha > 1$. All three histograms fit nicely to the theoretical gamma-distribution.

Problem C: The Dirichlet distribution: simulation using known relations

Let $x = (x_1, \ldots, x_K)$ be a vector of stochastic random variables where $x_k \in [0, 1]$ for $k = 1, \ldots, K$ and $\sum_{k=1}^{K} x_k = 1$. We assume $z_k \sim \text{gamma}(\alpha_k, 1)$ independently for $k = 1, \ldots, K$. The joint distribution is then given by

$$f_Z(z_1, \ldots, z_K) = \prod_{k=1}^{K} f_Z(z_k) = \frac{1}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \cdot \prod_{k=1}^{K} z_k^{\alpha_k - 1} \cdot e^{-\sum_{k=1}^{K} z_k}.$$

Let $v = \sum_{i=1}^{K} z_i$ and $x_k = \frac{z_k}{z_1, \ldots, z_K} = \frac{z_k}{v}$. Then $z_k = g^{-1}(x_k, v) = v x_k$ and because the x_i sum to 1, the K-th variable can be written as $z_K = (1 - \sum_{k=1}^{K-1} x_i) v$. We can now find the Jacobian

$$\left|\frac{\partial(z_1,\dots,z_K)}{\partial(x_1,\dots,x_{K-1},v)}\right| = \left|\frac{\partial(vx_1,\dots,vx_{K-1},v(1-\sum_{k=1}^{K-1}x_k))}{\partial(x_1,\dots,x_{K-1},v)}\right|$$

$$= \begin{vmatrix} v & 0 & \cdots & 0 & x_1 \\ 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & v & x_{K-1} \\ -v & \cdots & \cdots & -v & 1-\sum_{k=1}^{K-1}x_k \end{vmatrix} = \begin{vmatrix} v & 0 & \cdots & 0 & x_1 \\ 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & v & x_{K-1} \\ 0 & \cdots & \cdots & 0 & 1 \end{vmatrix} = v^{K-1}.$$

Thus, by the transformation formula we get the joint distribution

$$f_{X,V}(x_1,\dots,x_{K-1},v) = f_Z\left(vx_1,\dots,vx_{K-1},v\left(1-\sum_{k=1}^{K-1}x_k\right)\right)\cdot\left|\frac{\partial\left(vx_1,\dots,vx_{K-1},v\left(1-\prod_{k=1}^{K-1}x_k\right)\right)}{\partial(x_1,\dots,x_{K-1},v)}\right|$$

$$= \frac{1}{\prod_{k=1}^{K}\Gamma(\alpha_k)}\cdot\prod_{k=1}^{K-1}(vx_k)^{\alpha_k-1}\cdot\left(v\left(1-\prod_{k=1}^{K-1}x_k\right)\right)^{\alpha_K-1}\cdot e^{-\sum_{k=1}^{K}vx_k}\cdot v^{K-1}$$

$$= \frac{v^{\sum_{k=1}^{K-1}(\alpha_k-1)+(\alpha_K-1)+(K-1)}e^{-v}}{\prod_{k=1}^{K}\Gamma(\alpha_k)}\cdot\prod_{k=1}^{K-1}x_k^{\alpha_k-1}\cdot\left(1-\prod_{k=1}^{K-1}x_k\right)^{\alpha_K-1}$$

$$= \frac{v^{(\sum_{k=1}^{K}\alpha_k)-1}e^{-v}}{\prod_{k=1}^{K}\Gamma(\alpha_k)}\cdot\prod_{k=1}^{K-1}x_k^{\alpha_k-1}\cdot\left(1-\prod_{k=1}^{K-1}x_k\right)^{\alpha_K-1}$$

And by integration the marginal distribution of $(x_1,\dots,x_{K-1})$ is

$$f_X(x_1,\dots,x_{K-1}) = \frac{\int_0^\infty v^{(\sum_{k=1}^{K}\alpha_k)-1}e^{-v}dv}{\prod_{k=1}^{K}\Gamma(\alpha_k)}\cdot\prod_{k=1}^{K-1}x_k^{\alpha_k-1}\cdot\left(1-\prod_{k=1}^{K-1}x_k\right)^{\alpha_K-1}$$

$$= \frac{\Gamma(\sum_{k=1}^{K}\alpha_i)}{\prod_{k=1}^{K}\Gamma(\alpha_k)}\cdot\prod_{k=1}^{K-1}x_k^{\alpha_k-1}\cdot\left(1-\prod_{k=1}^{K-1}x_k\right)^{\alpha_K-1}$$

which means $(x_1,\dots,x_K)$ has a Dirichlet distribution.

```
#This function generates one realisation from Dirichlet distribution with
#parameter vector alpha.
random.dirichlet <- function(alpha){
  K <- length(alpha)
  z <- rep(0,K)
  for (i in 1:K)
    z[i] <- random.gamma(1,alpha[i],1) #n,shape,rate
  v <- sum(z)
  return (z/v)
}

#Generating Dirichlet samples with K=5
N_dir <- 10000
alpha <- c(.1, .5, .1, .3, .8)
K <- length(alpha)
```

```
dirichlet.sample <- matrix(1,N_dir,K)
for (i in 1:N_dir){
  dirichlet.sample[i,] <- random.dirichlet(alpha)
}
```

The mean and variance of a random variable X_i from a Dirichlet distribution are

$$\mathrm{E}(X_i) = \frac{\alpha_i}{\sum_{k=1}^{K} \alpha_k} := \tilde{\alpha}_i$$

and

$$\mathrm{Var}(X_i) = \frac{\tilde{\alpha}_i(1-\tilde{\alpha}_i)}{\sum_{k=1}^{K} \alpha_k + 1}.$$

```
alphasum <- sum(alpha)
#Empirical mean
colMeans(dirichlet.sample)
```

```
## [1] 0.05656376 0.27729560 0.05544981 0.16596621 0.44472461
```

```
#Analytical mean
alpha1 <- alpha/alphasum
alpha1
```

```
## [1] 0.05555556 0.27777778 0.05555556 0.16666667 0.44444444
```

```
#Empirical variance
diag(var(dirichlet.sample))
```

```
## [1] 0.01841008 0.07180625 0.01845017 0.04959944 0.08859133
```

```
#Analytical variance
alpha1*(1-alpha1)/(1+alphasum)
```

```
## [1] 0.01873898 0.07164903 0.01873898 0.04960317 0.08818342
```

The empirical mean and variance are very close to the analytical values and thus the implementation seems good.

Problem D: Rejection sampling and importance sampling

In this exercise we are interested in the posterior mean $E\big[\theta|y\big]$. For $\theta \in (0,1)$ the observed posterior density is $f(\theta|y) \propto (2+\theta)^{125}(1-\theta)^{38}\theta^{34}$.

First we implement a rejection sampling algorithm to simulate from $f(\theta|y)$ using the uniform distribution $\mathcal{U}(0,1)$ as the proposal density. Hence, the acceptance probability is $\alpha = \frac{1}{c}f(x)$

```
f <- function(theta){ #The proportional function
  return ((2+theta)^125*(1-theta)^(18+20)*theta^34)
}

#Find the maxima of the proportional function
res = optimise(f, interval = c(0,1),maximum=TRUE)
maxima=res$objective

new_f <- function(theta){ #Normalised density function
  return (f(theta)/as.numeric(integrate(f,0,1)[1]))
}
```

```r
theta_new_f <- function(theta){ #theta*normalised function
  return (theta*new_f(theta))
}

posterior <- function(theta){ #proportional function with prior beta(1,5)
  return(f(theta)*(1-theta)^4)
}

posterior.normalised <- function(theta){ #normalised function with prior beta(1,5)
  posterior(theta)/as.numeric(integrate(posterior,0,1)[1])
}

theta_new_f_g <- function(theta){ #theta*normalised function with prior beta(1,5)
  return (theta*posterior.normalised(theta))
}
```

```r
#This algorithm simulate from f(theta|y) using a uniform density as the proposal
#density, rejection sampling.
random.multinomial <- function(n){
  x.out <- rep(NA,n)
  tries <- 0
  for (i in 1:n){
    finished <- FALSE
    c <- maxima #res$objective
    while(!finished){
      x <- runif(1) #sample from the proposal distribution, U(0,1)
      alpha <- f(x) / c #compute the acceptance probability alpha
      u <- runif(1) #generate a helping variable U from a uniform(0,1)
      if (u <= alpha){ #Check if we accept it
        finished <- TRUE
      }
      tries=tries+1 #count how many random numbers the algorithm generates
    }
    x.out[i] <- x #appending the values that get accepted (u<=alpha)
  }
  return(list(theta = x.out, tries = tries))
}
```

To estimate the posterior mean of θ we use Monte-Carlo integration with M=10000 samples from $f(\theta|y)$.

```r
M=10000
sample.multinomial <- random.multinomial(M)
tries <- sample.multinomial$tries
cat("Estimated posterior mean:", mean(sample.multinomial$theta), "\n")
```

```
## Estimated posterior mean: 0.6233696
```

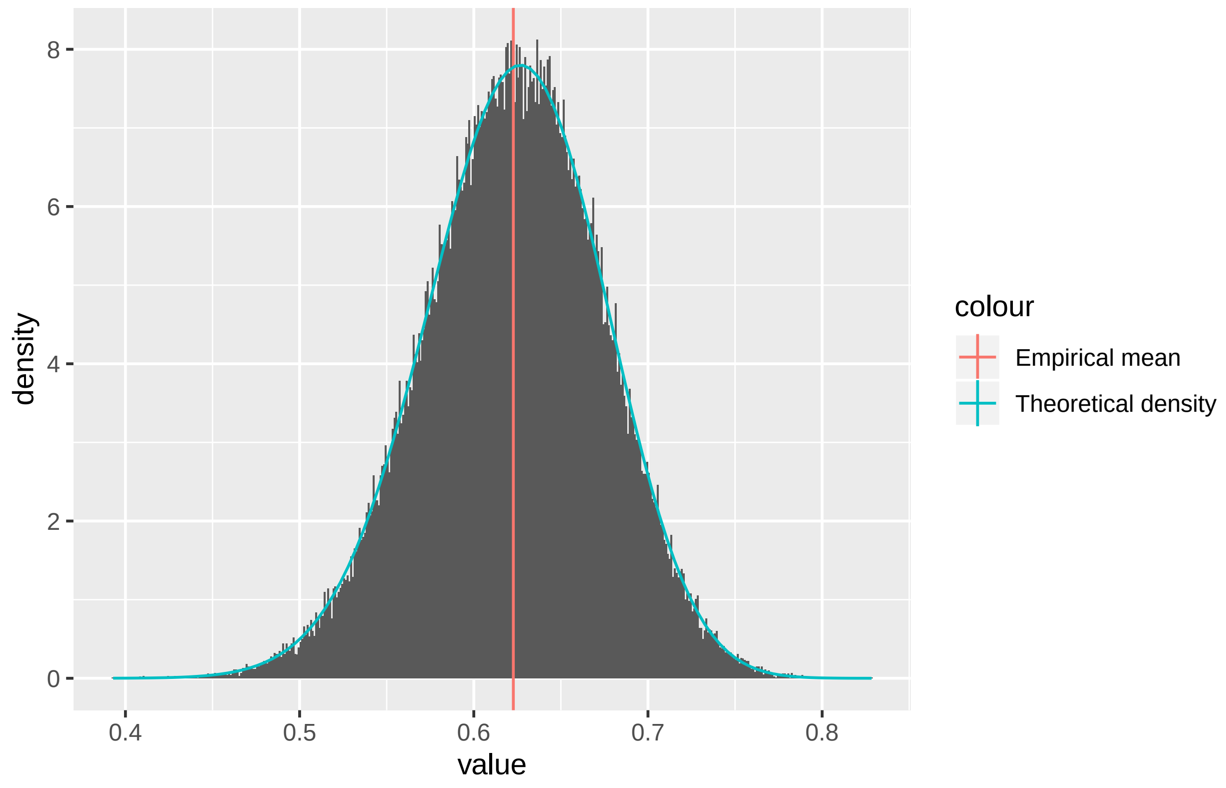
To check the value of the estimated posterior mean, we approximate it by numerically solving the integral

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

```r
expected_mean=integrate(theta_new_f,0,1)
cat("Expected posterior mean:", expected_mean$value, "\n")
```

```
## Expected posterior mean: 0.6228061
```

```r
sample.multinomial <- enframe(random.multinomial(100000)$theta)

ggplot() +
    geom_histogram(
      data = data.frame(sample.multinomial),
      mapping = aes(x=value, y=..density..),
      binwidth = 0.001,
      boundary = 0,
      na.rm = TRUE
    ) +
    geom_vline(
      aes(xintercept=mean(sample.multinomial$value),
          col='Empirical mean')
    ) +
    stat_function(
      fun = new_f,
      aes(col='Theoretical density')
    ) +
    labs(
      caption = "Histogram of sample from the posterior distribution
                     compared to its theoretical density."
    )
```



Histogram of sample from the posterior distribution compared to its theoretical density.

The histogram indicates that the sampler works well since it has similar shape as the density function.

To check how many random numbers our sampling algorithm need to generate on average in order to obtain one sample of $f(\theta|\mathbf{y})$ we divide the number of trials on M.

```r
average_theta_tries <- tries /M
cat("Average random numbers generated to obtain one sample:", average_theta_tries,"\n")
```

```
## Average random numbers generated to obtain one sample: 7.6801
```

We would expect, on average, to generate c samples before one i accepted, since the overall acceptance rate is c^{-1}. Since the proposal density is $\mathcal{U}(0,1)$, we have $c \geq f(x)$. Hence, we can numerically calculate $c = \max_{\theta\in[0,1]} f(\theta|\mathbf{y})$.

```r
c=optimize(new_f,interval = c(0,1),maximum=TRUE)$objective
cat("Expected random numbers generated to obtain one sample:", c, "\n")
```

```
## Expected random numbers generated to obtain one sample: 7.799308
```

The average of generated random numbers are close to the theoretical result, so the implementation of the algorithm seems correct.

To estimate the posterior mean under the new prior $g(\theta) \sim Beta(1,5)$ we use importance sampling with samples from $f(\theta|y)$ with $f(\theta) \sim Beta(1,1)$. Since neither $f(\theta) \propto 1$ and $g(\theta) \propto (1-\theta)^4$ are normalize, we use self-normalizing importance sampling to calculate the new posterior mean.

$$\tilde{\mu} = \frac{\sum_{i=1}^n h(\theta_i)w(\theta_i)}{\sum_{i=1}^n w(\theta_i)},$$

where $h(\theta_i) = \theta_i$ is the samples from $f(\theta|y)$ and

$$w(\theta_i) = \frac{g(\theta_i)}{f(\theta_i)} \propto (1-\theta)^4.$$

```r
importance_sampling <- function(n){
  theta <- random.multinomial(n)$theta #samples from f(theta|y) with rejection sampling
  w=(1-theta)^4 #importance weight
  mu <- sum(theta*w)/sum(w) #self-normalizing
  return(mu)
}

mu.estimate <- importance_sampling(10000)
mu.estimate
```

```
## [1] 0.5969128
```

```r
expected_mean=integrate(theta_new_f_g,0,1)
expected_mean
```

```
## 0.5959316 with absolute error < 9.7e-07
```

The estimated mean is the same as the expected mean, which somewhat validates the implementation. We observe that this mean is smaller than with the uniform prior. This can be explained by the shape of the priors. While the $Beta(1,1)$ is uniform on $[0,1]$, the $Beta(1,5)$-distribution is much larger for small θ and close to 0 for $\theta > 0.75$. This necessarily leads to fewer samples for θ close to 1, which again skewes the mean downwards.