Exercise 1 - TMA4300 Computer Intensive Statistical Methods

Marion Helen Røed Stephanie Jebsen Fagerås

Problem A: Stochastic simulation by the probability integral transform and bivariatetechniques

A1.

This section contains a function for generating n samples from an exponential distribuion with rate parameter λ. The probability density function is

$$f(x) = \lambda \cdot e^{-\lambda \cdot x}, x > 0.$$

In code this is given as

```
exponential_distribution_pdf <- function(x,lambda){
  #x = an vector of x-values
  #lambda = the rate
  #returns: a vector of values from the exponential probability density function
  ################################################################################

  #check which values of x are bigger than 0, as the function is 0, when this is not true
  bigger0=(x>0)

  #make the returned vector
  u=numeric(length(x))
  u[which(bigger0)]=lambda*exp(-lambda*x[bigger0])
  return(u)
}
```

The method used for generating random numbers from the exponential distribution is the inversion method. First the cummulative density function was found to be

$$F(x) = \int_{-\infty}^{x} f(x)dx = \begin{cases} \int_0^x \lambda \cdot e^{-\lambda \cdot x} dx = 1 - e^{-\lambda \cdot x} & x \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The inverse of this function is

$$F^{-1}(u) = \frac{-ln(1-u)}{\lambda} \quad , \quad 0 \leq u \leq 1.$$

Thus if one samples from the inverse cummulative function with uniformly distributed random variables as input one gets random variables distributed after the exponential distirbution.

```
exponential_distribution_samples <- function(lambda, n){
  #lambda = the rate
  #n=number of samples to generate
```

```
  #returns: a vector with generated random numbers
  ################################################

  #make a vector of uniformly distributed random numbers, the vector has length n
  u=runif(n)

  #The samples are found by the inversion method
  x=-1/lambda*log(1-u)
  return(x)
}
```

```
#testing for lambda=2 and n=10000
lambda=2
n=100000

#the samples:
exp_samples=exponential_distribution_samples(lambda,n)

#the mean should be 1/lambda
our_mean=mean(exp_samples)
exact_mean=1/lambda

#the variance should be 1/lambda^2
our_var=var(exp_samples)
exact_var=1/lambda^2
```

The values of the exact mean and variance for the exponential distribution are 0.5 and 0.25. The values of the calculated mean and variance from our samples are 0.4996678 and 0.2503544. One can see that the expected mean and variance are very close to the ones found by using the sampling algorithm.

```
#plotting the histogram and the probability density function in the same plot
xmin=-1
xmax=7
x <- seq(xmin,xmax,0.1)
exp_data <- data.frame("exp_samples"=exp_samples)
ggplot(data=exp_data, aes(exp_data$exp_samples)) +
  geom_histogram(aes(y=..density..), breaks = seq(xmin, xmax, by = 0.25)) +
  labs(title="Exponential Distribution", x="x", y="Density") +
  xlim(c(xmin,xmax)) +
  stat_function(fun = exponential_distribution_pdf, args = list(lambda))
```

```
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```

In Figure 1 one can see that the probability density function is coherent with the samples found by the sampling algorithm.

A2.

In this section we are considering the following probability density function

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

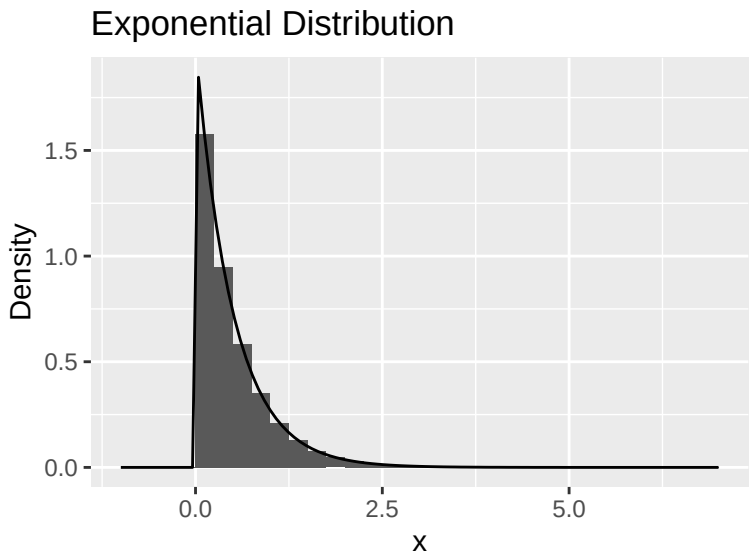

Figure 1: A histogram of the generated random numbers compared to the known exponential probability density function. Lambda had value 2 and 100000 samples were generated.

where c is the normalising constant and $\alpha \in (0, 1)$

A2a)

The cummulative distribution function is given as

$$G_X(x) = \begin{cases} 0 & x < 0 \\ \int_0^x cx^{\alpha-1}dx & 0 < x < 1 \\ \int_0^1 cx^{\alpha-1}dx + \int_1^x ce^{-x}dx & 1 \leq x \end{cases} \tag{2}$$

$$= \begin{cases} 0 & x < 0 \\ [\frac{c}{\alpha}x^{\alpha}]_0^x & 0 < x < 1 \\ \frac{c}{\alpha} + [-ce^{-x}]_1^x & 1 \leq x \end{cases} \tag{3}$$

$$= \begin{cases} 0 & x < 0 \\ \frac{c}{\alpha}x^{\alpha} & 0 < x < 1 \\ \frac{c}{\alpha} + \frac{c}{e} - \frac{c}{e^x} & 1 \leq x \end{cases} \tag{4}$$

An the inverse is

$$G^{-1}(u) = \begin{cases} \frac{\alpha}{c}u^{\frac{1}{\alpha}} & 0 \leq u < \frac{c}{\alpha} \\ -\ln\left(\frac{u-\frac{c}{\alpha}-ce^{-1}}{-c}\right) & u \geq \frac{c}{\alpha} \end{cases} \tag{5}$$

$$= \begin{cases} \frac{\alpha}{c}u^{\frac{1}{\alpha}} & 0 \leq u < \frac{c}{\alpha} \\ -\ln\left(\frac{1}{\alpha} + e^{-1} - \frac{u}{c}\right) & u \geq \frac{c}{\alpha} \end{cases} \tag{6}$$

$$\tag{7}$$

We have that

$$c = \int_{-\infty}^{\infty} g(x)dx = \int_{-\infty}^{0} 0dx + \int_{0}^{1} cx^{\alpha-1}dx + \int_{1}^{\infty} ce^{-x}dx = \frac{c}{\alpha} + ce^{-1} = 1,$$

giving us the normalising constant

$$c = \frac{1}{\frac{1}{\alpha} + e^{-1}}.$$

A2b)

In this section we want to generate n samples from the distribution $g(x)$. The functions found in section A-2a are implemented below.

```
g <-function(x,alpha){
  #the probability density function g(x)
  #alpha= parameter
  #x= input vector
  ######################################

  #const=variable for samples_from_g
  const=1/(1/alpha+exp(-1))

  u <- c(1:length(x))*0
  for (i in 1:length(x)){
    if (x[i]<0){
      u[i]=0
    }
    else if (x[i]<1){
      u[i]=const*x[i]^(alpha-1)
    }
    else {
      u[i]=const*exp(-x[i])
    }
  }
  return(u)
}

Ginverse <-function(u,alpha){
  #the inverse of the cummulative distribution function
  #alpha=parameter
  #u=input vector
  ######################################################

  #const=variable for samples_from_g
  const=1/(1/alpha+exp(-1))

  x=c(1:length(u))*0
  for (i in 1:length(u)){
    if (u[i]<0){
      print(FALSE)
    }
    else if (u[i]<=const/alpha){
      x[i]=(alpha/const*u[i])^(1/alpha)
    }
```

```
    else {
      x[i]=-log((-const/alpha+u[i]-const*exp(-1))/(-const))
    }
  }
  return(x)
}
```

The inversion method was used to sample from the probability distribution.

```
samples_from_g <- function(alpha,n){
  #lambda = the rate
  #n=number of samples to generate
  #returns: a vector with generated random numbers
  ##################################################

  #make a vector of uniformly distributed random numbers, the vector has length n
  u=runif(n)

  #the values are found by the inversion method
  x=Ginverse(u,alpha)
  return(x)
}
```

```
#testing the function
alpha=0.5
n=100000
samples_g=samples_from_g(alpha,n)
```

```
#plotting the histogram and the probability density function in the same plot
xmin=-1
xmax=15
x <- seq(xmin,xmax,0.1)

g_data <- data.frame("g_samples"=samples_g)
ggplot(data=g_data, aes(g_data$g_samples)) +
  geom_histogram(aes(y=..density..), breaks = seq(xmin, xmax, by = 0.25)) +
  labs(title="Probabiliy density function g(x)", x="x", y="Density") +
  xlim(c(xmin,xmax)) +
  stat_function(fun = g, args = list(alpha))+
  theme(plot.title = element_text(hjust = 0.5))
```

In Figure 2 one can see that the probability density function $g(x)$ and the found samples are coherent.

A3.

For this assignment we want to sample from the normal distribution using the Box-Muller algorithm. The implementation is shown below.

```
Box_Muller <- function(n){
  #n=number of samples to generate
  #returns: a vector with generated random numbers
  ########################################
  u1=runif(n)
  u2=runif(n)
  return (sqrt(-2*log(u1))*sin(2*pi*u2))
}
```

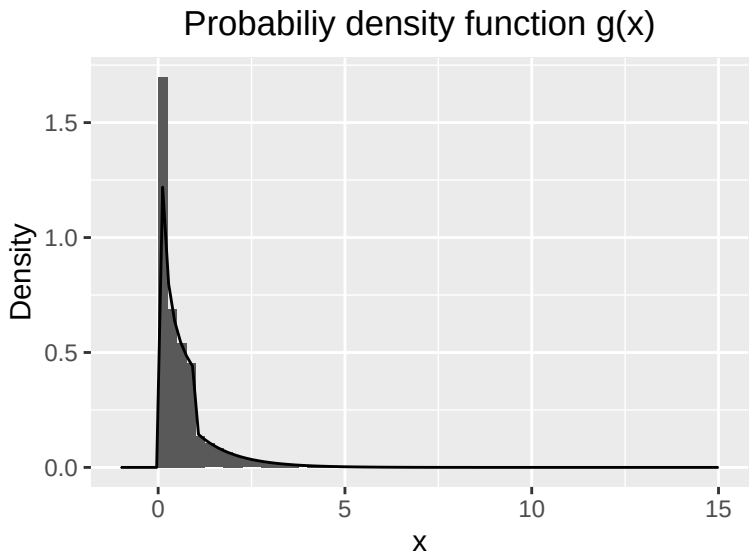

Figure 2: A histogram of the generated random numbers compared to the known probability density function. alpha had value 0.9 and 100000 samples were generated.

```
#running the function
std_normal_samples <- Box_Muller(10000)

#calculating the mean and variance
normalmean=mean(std_normal_samples)
normalvar=var(std_normal_samples)
```

The algorithm gives the mean -0.0118801 and variance 0.9969603 which is close to the theoretical consecutive values 0 and 1.

```
#plotting the histogram and the probability density function in the same plot
xmin=-5
xmax=5
x <- seq(xmin,xmax,0.1)

mu=0
sigma=1

#Normal distribution for comparison
NormalPDF <- function(x,mu,sigma){
  return(1/(sigma*sqrt(2*pi))*exp((-1/2*((x-mu)/sigma)^2)))
}

std_normal_data <- data.frame("std_normal_samples"=std_normal_samples)
ggplot(data=std_normal_data, aes(std_normal_data$std_normal_samples)) +
  geom_histogram(aes(y=..density..), breaks = seq(xmin, xmax, by = 0.25)) +
  labs(title="Standard normal distribution", x="x", y="Density") +
  xlim(c(xmin,xmax)) +
  stat_function(fun = NormalPDF, args = list(mu,sigma))+
  theme(plot.title = element_text(hjust = 0.5))
```

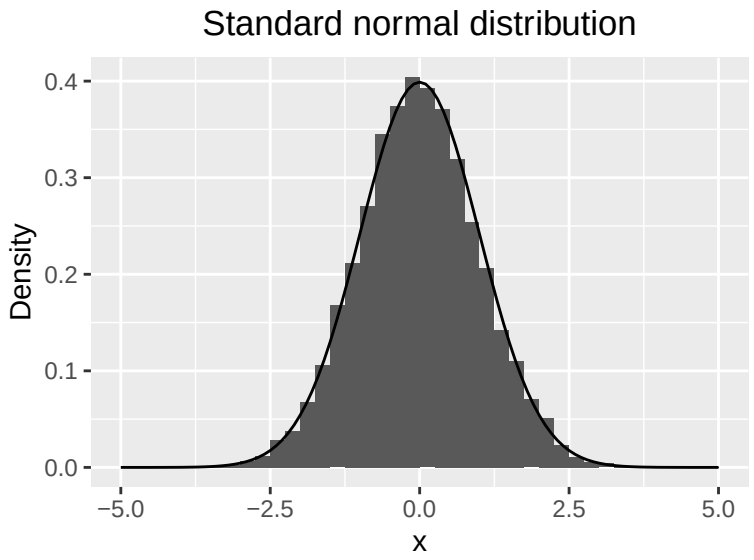

Figure 3: A histogram of the generated random numbers compared to the known standard normal probability density function.

In Figure 3 one can see that the probability density function $g(x)$ and the found samples are coherent.

A4.

For this assignment we want to simulate from a d-variate normal distribution with a given mean vector μ and covariance matrix Σ. As we don't have given values, we start by constructing μ by making a vector of length d with a normal distribution using the Box-Muller function from above. $\Sigma = MM^T$ is constructed from d×d matrix M with a normal distribution, making sure that Σ is positive definite.

```
dim = 3
#Constructing a mean vector
mu <- Box_Muller(dim)
#Constructing a positive definite covariance matrix
M=matrix(Box_Muller(dim*dim),dim,dim)
Sigma <- M%*%t(M)
```

To simulate from $y \sim N_d(\mu, \Sigma)$ we use that $y = \mu + Mx \Rightarrow y \sim N_d(\mu, MM^T)$ with $\Sigma = MM^T$, where M is already given from the construction of Σ. For a previously defined Σ, M could be found by Cholesky decompostion. The implementation of the simulation is shown below.

```
#running the function
n=100000
y=matrix(0, dim, n)
for (i in 1:n){
  y[,i]=mu + M%*%Box_Muller(dim)
}

#Calculating mean for comparison
means=numeric(dim)
for (i in 1:dim){
  means[i]=mean(y[i,])
}
#Calculating covariance for comparison
```

```
covmatrix=cov(t(y))
```

The chosen μ and the mean calculated from the multivariate distribution are -0.9653624, -0.3933774, 1.2372224 and -0.96639, -0.391763, 1.2353844, while the chosen Σ and the calculated covariance matrix are

```
##            [,1]       [,2]       [,3]
## [1,]  2.9473097 -0.4220189 -1.2122199
## [2,] -0.4220189  0.7625978  0.2288073
## [3,] -1.2122199  0.2288073  1.8469863
```

and

```
##            [,1]       [,2]       [,3]
## [1,]  2.9462696 -0.4246598 -1.2087131
## [2,] -0.4246598  0.7658903  0.2273673
## [3,] -1.2087131  0.2273673  1.8544439
```

implying by their closeness that the algorithm produces a multivariate normal distribution of samples.

Problem B: The gamma distribution

We are considering the gamma distribution

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \quad , \quad x \in (0, \infty).$$

B1.

In the first section we're considering the gamma distribution for $\alpha \in (0, 1)$ and $\beta = 1$. Rejection sampling will be used to generate samples from the distribution. The distribution considered in A-2, $g(x)$, will be used as a proposal density.

B1a)

The acceptance probability is called β here, as α is already taken. The acceptance probability is given by $f(x) \leq d * g(x)$, where d has replaced c as this value is also taken. Thus we want $\frac{f(x)}{g(x)} \leq d$. So we want to find $\beta = \frac{f(x)}{d \cdot g(x)}$. We have

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

and

$$\frac{f(x)}{g(x)} \leq \frac{1}{c * \Gamma(\alpha)} = d,$$

so the acceptance probability is given as

$$\beta = \begin{cases} e^{-x} & 0 < x < 1 \\ x^{\alpha-1} & 1 \leq x. \end{cases}$$

B1b)

We want to generate n samples from the gamma distribution, for $\alpha \in (0,1)$ and $\beta = 1$. The implementation is shown below.

```
gamma_distribution_small_alpha <- function(alpha,n){
  #rejection sampling
  #alpha=parameter in the gamma distribution
  #n= the number of samples
  #returns: a n-dimensional vector with samples
  ##############################################


  #making the return vector
  ret=numeric(n)

  #for each sample
  for (i in 1:n){
    #rejection sampling
    finished = 0
    while (finished == 0){
      #the proposal distribution
      x=samples_from_g(alpha,1)

      #the acceptance probability
      beta=acceptance_prob(x,alpha)

      #generate a uniformly distributed number
      u=runif(1)

      #if the uniformly distributed number is smaller than the acceptance
      #probability, then it is accepted.
      if (u<=beta){
        finished = 1
        ret[i]=x
      }
    }
  }
  return(ret)
}

acceptance_prob <- function(x,alpha){
  #the acceptance probability
  #x=a number
  #alpha=parameter in the gamma distribution
  #returns: the acceptance probability
  ############################################

  if (x<0){
    return(0)
  }
  else if (x<1){
    return(exp(-x))
  }
  else{
```

```
    return(x^(alpha-1))
  }
}

gamma_distribution_pdf<- function(x,alpha,beta=1){
  #the gamma distribution probability density function
  #x = vector of x values
  #alpha=parameter
  #beta=parameter
  #returns: gamma pdf
  ##################################################

  #valid for x>0
  bigger0=(x>0)

  #make the gamma pdf
  u=numeric(length(x))
  #u[which(bigger0)]=1/factorial(alpha-1)*beta^alpha*x[bigger0]^(alpha-1)*exp(-beta*x[bigger0])
  u[which(bigger0)]=exp(alpha*log(beta)+(alpha-1)*log(x[bigger0])+(-beta*x[bigger0])-lfactorial(alpha-1)

  return(u)
}
```

```
#testing the algorithm
alpha=0.5
n=10000
samples_gamma_small_alpha=gamma_distribution_small_alpha(alpha,n)

beta=1
#check if theoretical mean and sample mean are coherent
our_mean=mean(samples_gamma_small_alpha)
ex_mean=alpha/beta

#check if theoretical variance and sample variance are coherent
our_var=var(samples_gamma_small_alpha)
ex_var=alpha/beta^2
```

The values of the exact mean and variance are $\mu = \frac{\alpha}{\beta} = 0.5$ and $\sigma^2 = \frac{\alpha}{\beta^2} = 0.5$. The values found for the mean and variance for our samples are 0.4938145 and 0.500877. One can observe that the expected values and variances are similar to the ones for the samples.

```
#plotting the histogram and the probability density function in the same plot
xmin=-1
xmax=10
x <- seq(xmin,xmax,0.1)

gamma_small_alpha_data <- data.frame("gamma_small_alpha_samples"=samples_gamma_small_alpha)
ggplot(data=gamma_small_alpha_data, aes(gamma_small_alpha_data$gamma_small_alpha_samples)) +
  geom_histogram(aes(y=..density..), breaks = seq(xmin, xmax, by = 0.25)) +
  labs(title="Gamma distribution for small alpha", x="x", y="Density") +
  xlim(c(xmin,xmax)) +
  stat_function(fun = gamma_distribution_pdf, args = list(alpha))+
  theme(plot.title = element_text(hjust = 0.5))
```

One can observe that the probability density function and the samples are fairly similar in Figure 4. From

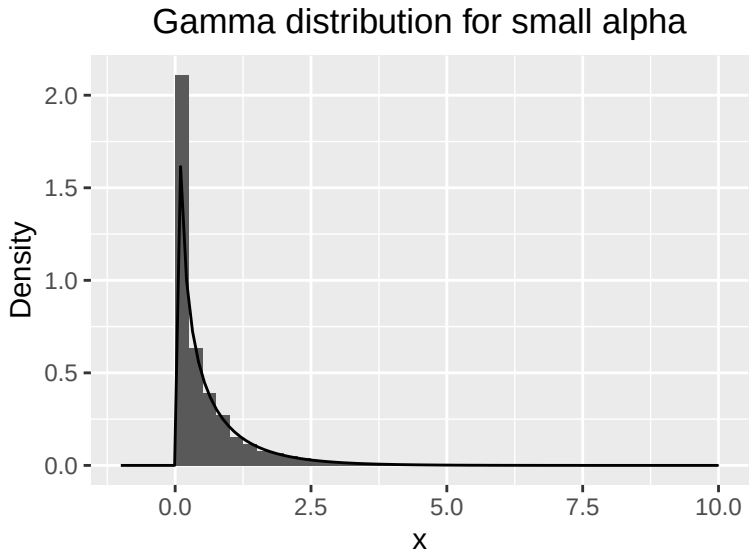

Figure 4: A histogram of the generated random numbers compared to the known gamma probability density function. For alpha in (0,1) and beta equal 1.

this and the mean and variance result we conclude that the algorithm seems correct.

B2.

The ratio of uniforms method will be used to generate samples from the gamma distribution for $\alpha > 1$ and $\beta = 1$.

B2a)

For the the ratio of uniforms method we define

$$C_f = \left\{ (x_1, x_2) : 0 \leq x_1 \leq \sqrt{f^\star\left(\frac{x_2}{x_1}\right)} \right\} \quad \text{where} \quad f^\star(x) = \begin{cases} x^{\alpha-1}e^{-x} & 0 < x \\ 0 & \text{otherwise} \end{cases}$$

and

$$\begin{aligned} a &= \sqrt{\sup_x f^\star(x)} = \sqrt{(\alpha-1)^{\alpha-1}e^{-(\alpha-1)}} \\ b_+ &= \sqrt{\sup_{x\geq 0}(x^2 f^\star(x))} = \sqrt{(\alpha+1)^{\alpha+1}e^{-(\alpha+1)}} \\ b_- &= -\sqrt{\sup_{x\leq 0}(x^2 f^\star(x))} = -\sqrt{\sup_{x\leq 0} 0} = 0 \end{aligned}$$

where the supremum with regards to x for a is found by

$$\begin{aligned} \frac{\partial(x^{\alpha-1}e^{-x})}{\partial x} = (\alpha-1)x^{\alpha-2}e^{-x} + x^{\alpha-1}\cdot(-e^{-x}) = ((\alpha-1)x^{\alpha-2} - x^{\alpha-1})e^{-x} &= 0 \\ (\alpha-1)x^{\alpha-2} &= x^{\alpha-1} \\ x &= \alpha - 1, \end{aligned}$$

and the supremum with regards to x for b_+ is found by

$$\frac{\partial(x^2x^{\alpha-1}e^{-x})}{\partial x} = (\alpha+1)x^\alpha e^{-x} + x^{\alpha+1}\cdot(-e^{-x}) = ((\alpha+1)x^\alpha - x^{\alpha+1})e^{-x} = 0$$
$$(\alpha+1)x^\alpha = x^{\alpha+1}$$
$$x = \alpha+1.$$

This means that $C_f \subset [0,a]\text{x}[b_-,b_+]$.

B2b)

The ratio of uniforms method is used to generate random numbers. However, the α's are very big, we want them to be in $\alpha \in (1,2000]$, thus we need to do the calculations in logarithmic scale. To do this we make a change of variables so that $x = ln(y)$. Where y is uniformly distributed on [0,a]. Using the formula

$$f_x(x) = f_y(g^{-1}(y))\left|\frac{dg^{-1}(y)}{dy}\right|,$$

we get that

$$f_x(x) = \begin{cases} \frac{1}{a}|e^x| & x \in (-\infty, log(a)) \\ 0 & \text{otherwise.} \end{cases}$$

We calculate the values of the cummulative distribution function and the inverse cummulative distribution function to use the inversion method.

$$F_x(x) = \int_{-\infty}^{x} \frac{1}{a}e^y dy = \begin{cases} [\frac{1}{a}e^y]_{-\infty}^{\ln a} & x > \ln a \\ [\frac{1}{a}e^y]_{-\infty}^{x} & x \le \ln a \end{cases}$$
$$= \begin{cases} 1 & x > \ln a \\ \frac{1}{a}e^x & x \le \ln a \end{cases}$$
$$F^{-1}(u) = ln(u) + ln(a) \quad , \quad u \le 1$$

We also change the formula for $f^*(x)$ to return its values in logarithmic scale, and we have the values of a and b_+ in logarithmic scale. All the calculations are done in logarithmic scale and the answer is transformed back at the end of the algorithm.

```
#check how many tries the algorithm needs to generate n=1000 realisations depending
#on the value of alpha in (0,2000)
#generate a plot with values of alpha on the x-axis and number of tries on the y-axis.
#interperet the result.
#caution: need log scale og will get NAs


f_star_log<- function(logx,alpha){
  #logx: log(x), vector of size K
  #alpha: parameter for probability density distribution
  #returns: a vector in log scale og the probability density function of length K
  ##############################################################################

  #the returned vector
  logu=numeric(length(logx))

  #log scale of u:
```

```
  #u=x^(alpha-1)*exp(-x)
  logu=(alpha-1)*logx+(-exp(logx))

  return(logu)
}

gamma_distribution_big_alpha<-function(n,alpha){
  #n= number of samples returned
  #alpha= parameter for gamma distribution
  #returns: a n-size vector of samples from the gamma distribution
  #the calculations have been done in log scale to avoind NA's
  #the method used is ratio of uniforms method
  ######################################################################

  #C_f subset of [0,a]x[b-,b+]

  #a=sqrt((alpha-1)^(alpha-1)*exp(-(alpha-1)))
  #loga=log(sqrt((alpha-1)^(alpha-1)*exp(-(alpha-1))))
  loga=1/2*((alpha-1)*log(alpha-1)-(alpha-1))

  #bminus=0
  logbminus=-Inf

  #bplus=sqrt((alpha+1)^(alpha+1)*exp(-(alpha+1)))
  #logbplus=log(sqrt((alpha+1)^(alpha+1)*exp(-(alpha+1))))
  logbplus=1/2*((alpha+1)*log(alpha+1)-(alpha+1))

  #x=log(y), where y is the vector we want to return
  x=numeric()

  #count the iterations
  itcount=0

  #number of samples generated per loop of the while loop
  n_it=n

  while (length(x)<n){
    itcount=itcount+n_it

    #used change of variables on x=log(y),
    #found that f_x1(x1)=1/a*exp(x1) on [-inf,loga],
    #and f_x2(x_2)=1/bplus*exp(x2) on [-inf,logb]
    #used inversion method got:
    x1=log(runif(n_it))+loga
    x2=log(runif(n_it))+logbplus

    #check if the values can be accepted
    inside = 2*x1<=f_star_log(x2-x1,alpha)

    #a vector of the values that can be accepted
    xbef=x2[inside]-x1[inside]

    #add vector to returned vector
```

```
    x = c(x,xbef)
  }
  #only want n first values, the iteration count also too big, but only problem for small alpha.
  x=x[1:n]
  return(c(exp(x),itcount))
}

n=1000
alpha=2000
beta=1
n_gamma=gamma_distribution_big_alpha(n,alpha)
itcount=n_gamma[length(n_gamma)]
samples_gamma_big_alpha=n_gamma[1:length(n_gamma)-1]


#check mean
our_mean_big_alpha=mean(samples_gamma_big_alpha)
exact_mean_big_alpha=alpha/beta

#check variance
our_var_big_alpha=var(samples_gamma_big_alpha)
exact_var_big_alpha=alpha/beta^2
```

First we test the algorithm for $\alpha = 2000$. The exact mean is given as $\mu = \frac{\alpha}{\beta}$, the value here is 2000, whereas the exact variance is given as $\sigma^2 = \frac{\alpha}{\beta^2}$, which has the value 2000. The values found numerically for our samples are 1999.3827785 for the mean and 1939.0101622 for the variance. These values are fairly similar.

```
#plotting the histogram and the probability density function in the same plot
xmin=1800
xmax=2300
x <- seq(xmin,xmax,0.01)

gamma_big_alpha_data <- data.frame("gamma_big_alpha_samples"=samples_gamma_big_alpha)
ggplot(data=gamma_big_alpha_data, aes(gamma_big_alpha_data$gamma_big_alpha_samples)) +
  geom_histogram(aes(y=..density..), breaks = seq(xmin, xmax, by = 15)) +
  labs(title="Gamma distribution for big alpha", x="x", y="Density") +
  xlim(c(xmin,xmax)) +
  stat_function(fun = gamma_distribution_pdf, args = list(alpha))+
  theme(plot.title = element_text(hjust = 0.5))
```

A histogram of the generated random numbers compared to the known gamma probability density function for $\alpha = 2000$ and $\beta = 1$, is shown in Figure 5. One can observe that the line corresponds with the histogram.

```
#plot alpha vs. iteration count
plotmatrix=matrix(0,200,2)

#finding number of samples per alpha
for (i in 1:200){
  alpha=10*i
  samples=gamma_distribution_big_alpha(n,alpha)
  plotmatrix[i,1]=alpha
  plotmatrix[i,2]=samples[length(n_gamma)]
}
```

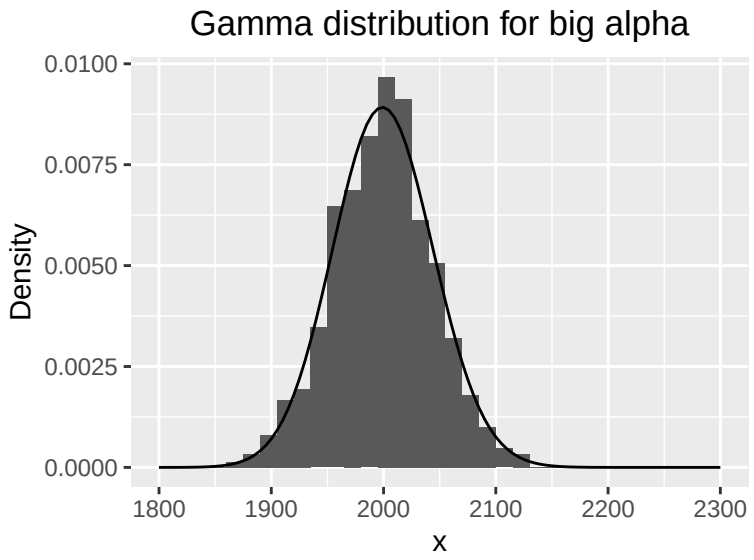

Figure 5: A histogram of the generated random numbers compared to the known gamma probability density function. For alpha bigger than 1 and beta equal 1.

```
d <- data.frame(plotmatrix)
p2 <- ggplot(d, aes(x = X1, y = X2)) +
  geom_point(alpha = .5) +
  labs(title="Number of Iterations vs Size of Alpha",x="Alpha",y="Iteration count") +
  theme(plot.title = element_text(hjust = 0.5))
p2
```

In Figure 6 the number of tries the algorithm used is plotted against the value of α. One can see that the value of tries increases with increasing α, but that the growth is slower for bigger α than for smaller α. The growth seems to be almost linear when the α's are big enough.

B3.

In this section we make a function that generates samples from the gamma distribution for any α and β. For $\alpha \in (0,1)$ and $\alpha > 1$ we made samplers in part B-1 and B-2. For $\alpha = 1$ one can observe that the probability density function is equivalent to the exponential probability density function with $\lambda = 1$. For the parameter β one has to observe that it is an inverse scale parameter, thus the only thing needed is to divide by β.

```
gamma_distribution <- function(n,alpha,beta){
  #n=number of samples
  #alpha=shape parameter
  #beta=scale parameter
  #returns: vector of n samples from the gamma distribution
  ############################################################

  if (alpha==1){
    #for alpha = 1, the gamma distribution is equal to the exponential distribution with lambda=1
    return(exponential_distribution_samples(1,n)/beta)
  }
  else if (alpha<1){
    return(gamma_distribution_small_alpha(alpha,n)/beta)
```

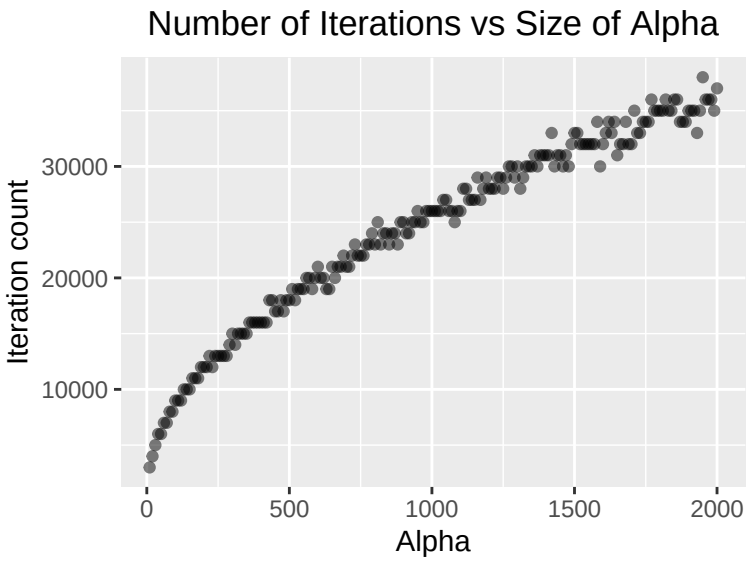

Figure 6: The alpha's plotted on the x-axis and the number of tries plotted on the y-axis.

```
  }
  else{
    u=gamma_distribution_big_alpha(n,alpha)
    return(u[1:length(u)-1]/beta)
  }
}
```

```
n=1000
alpha=1
beta=2
gamma_samples=gamma_distribution(n,alpha,beta)


our_mean_gamma=mean(gamma_samples)
ex_mean_gamma=alpha/beta

our_var_gamma=var(gamma_samples)
ex_var_gamma=alpha/beta^2
```

The expected mean and variance are 0.5 and 0.25, while the mean and variance found from our samples are 0.5189265 and 0.2708285. One can observe that the expected value and variance are similar to the expected value and variance of the samples.

```
#plotting the histogram and the probability density function in the same plot
xmin=-1
xmax=10
x <- seq(xmin,xmax,0.1)

gamma_data <- data.frame("gamma_samples"=gamma_samples)
ggplot(data=gamma_data, aes(gamma_data$gamma_samples)) +
  geom_histogram(aes(y=..density..), breaks = seq(xmin, xmax, by = 0.25)) +
  labs(title="Gamma distribution", x="x", y="Density") +
  xlim(c(xmin,xmax)) +
```

```
  stat_function(fun = gamma_distribution_pdf, args = list(alpha,beta))+
  theme(plot.title = element_text(hjust = 0.5))
```

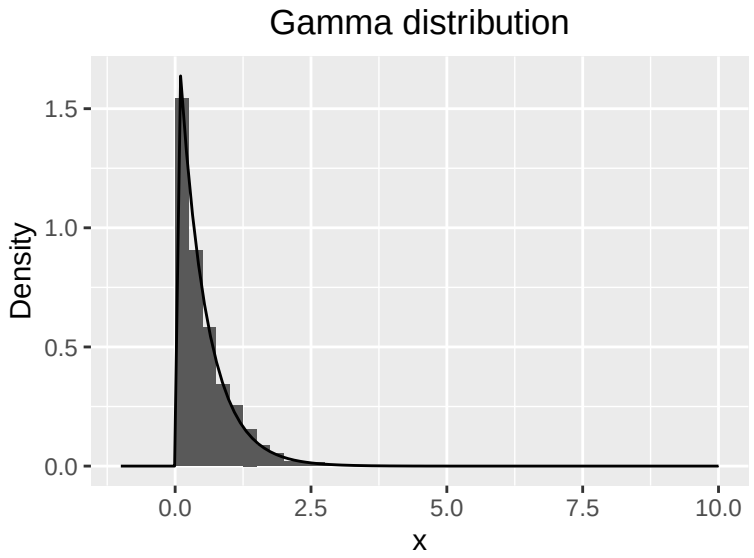

Figure 7: A histogram of the generated random numbers compared to the known gamma probability density function with alpha=1 and beta!=1.

In Figure 7 one can see that the algorithm works also for $\alpha = 1$ and $\beta \neq 1$. The other cases have been shown for $\beta = 1$ in B-1 and B-2, and the only difference here is scaling with β.

Problem C: The Dirichlet distribution: simulation using known relations

C1.

We assume that $z_k \sim$gamma$(\alpha_k, 1)$ and define that $x_k = \frac{z_k}{v}$ with $v = z_1 + \dots z_K$ for $k = 1, \dots, K$ and $\sum_{k=1}^{K} x_k = 1$. We wish to prove that $x = (x_1, \dots, x_K)$ has a Dirichlet distribution with parameter vector $\alpha = (\alpha_1, \dots, \alpha_K)$.

Using $z_1 = x_1 v$, $z_2 = x_2 v$, $\dots$, $z_K = (1 - x_1 - \dots - x_{K-1})v$ and defining that $x_K = v$ for convenience, the Jacobi matrix becomes

$$
\begin{aligned}
|J| &= \begin{vmatrix} \frac{\partial z_1}{\partial x_1} & \frac{\partial z_1}{\partial x_2} & \dots & \frac{\partial z_1}{\partial x_K} \\ \vdots & \ddots & & \\ \frac{\partial z_K}{\partial x_1} & \dots & & \frac{\partial z_K}{\partial x_K} \end{vmatrix} = \begin{vmatrix} \frac{\partial x_1 v}{\partial x_1} & \frac{\partial x_1 v}{\partial x_2} & \dots & \frac{\partial x_1 v}{\partial v} \\ \vdots & \ddots & & \\ \frac{\partial (1-x_1-\dots-x_{K-1})v}{\partial x_1} & \dots & & \frac{\partial (1-x_1-\dots-x_{K-1})v}{\partial v} \end{vmatrix} \\
&= \begin{vmatrix} v & 0 & \cdots & x1 \\ 0 & v & \ddots & x2 \\ \vdots & \ddots & \ddots & \vdots \\ -v & -v & \cdots & 1 - x_1 - \dots - x_{K-1} \end{vmatrix} \sim \begin{vmatrix} v & 0 & \cdots & x1 \\ 0 & v & \ddots & x2 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{vmatrix} = v^{K-1}
\end{aligned}
$$

The last operation on the Jacobi matrix is a row operation where the first $K - 1$ rows are added to the K-th row, making the matrix upper diagonal with determinant given by the product of the diagonal.

The gamma distribution is given by

$$f_Z(z) = \begin{cases} \frac{1}{\Gamma(\alpha)} z^{\alpha-1} e^{-z} & 0 < z \\ 0 & \text{otherwise.} \end{cases}$$

Using the change-of-variables technique and that $f(z_1, \ldots, z_K) = \Pi f(z_i)$ because the z_is are independently distributed, we find that

$$\begin{aligned} f_X(x_1, \ldots, x_K) &= f_Z(x_i v, \ldots, x_K v) \cdot |J| \\ &= \prod_{i=1}^{K} \frac{1}{\Gamma(\alpha_i)} (x_i v)^{\alpha_i - 1} e^{-x_i v} \cdot v^{K-1} \\ &= \frac{1}{\Gamma(\alpha_1) \cdot \ldots \cdot \Gamma(\alpha_K)} (x_1 v)^{\alpha_1 - 1} \cdot \ldots \cdot (x_K v)^{\alpha_K - 1} e^{-v(x_1 + \ldots + x_K)} \cdot v^{K-1} \\ &= \frac{1}{\Gamma(\alpha_1) \cdot \ldots \cdot \Gamma(\alpha_K)} (x_1)^{\alpha_1 - 1} \cdot \ldots \cdot (1 - x_1 - \ldots - x_{K-1})^{\alpha_K - 1} e^{-v} \cdot v^{\sum_{i=1}^{K} (\alpha_i - 1) + K - 1}. \end{aligned}$$

The function is now independent of x_K, as we have used that the xs add up to 1. We remove v by integration and get

$$\begin{aligned} \int_0^\infty f_X(x_1, \ldots, x_{K-1}) dv &= \int_0^\infty \frac{(x_1)^{\alpha_1 - 1} \cdot \ldots \cdot (1 - x_1 - \ldots - x_{K-1})^{\alpha_K - 1}}{\Gamma(\alpha_1) \cdot \ldots \cdot \Gamma(\alpha_K)} \cdot e^{-v} \cdot v^{\sum_{i=1}^{K} \alpha_i - 1} dv \\ &= \frac{(x_1)^{\alpha_1 - 1} \cdot \ldots \cdot (1 - x_1 - \ldots - x_{K-1})^{\alpha_K - 1}}{\Gamma(\alpha_1) \cdot \ldots \cdot \Gamma(\alpha_K)} \int_0^\infty e^{-v} \cdot v^{\sum_{i=1}^{K} \alpha_i - 1} dv \\ &= \frac{(x_1)^{\alpha_1 - 1} \cdot \ldots \cdot (1 - x_1 - \ldots - x_{K-1})^{\alpha_K - 1}}{\Gamma(\alpha_1) \cdot \ldots \cdot \Gamma(\alpha_K)} \cdot \Gamma(\sum_{i=1}^{K} \alpha_i) \\ &= \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\Gamma(\alpha_1) \cdot \ldots \cdot \Gamma(\alpha_K)} \cdot (x_1)^{\alpha_1 - 1} \cdot \ldots \cdot (x_{K-1})^{\alpha_{K-1} - 1} \cdot (1 - \sum_{i=1}^{K-1} x_i)^{\alpha_K - 1} \end{aligned}$$

The integral on the second line is the probability density function of the gamma distribution, without the normalizing constant, with parameter $\alpha = \sum_{i=1}^{K} \alpha_i$ and $\beta = 1$, which has to integrate to the normalizing constant, thus the solution of the integral is $\Gamma(\sum_{i=1}^{K} \alpha_i)$. The final answer is the density of a vector $x = (x_1, \ldots, x_{K-1})$ with parameter $\alpha = (\alpha_1, \ldots, \alpha_K)$ when x has a Dirichlet distribution, which is what we wanted to show.

C2.

To sample from the dirichlet distribution, one needs to sample from the gamma distribution for $\beta = 1$ for each α_i, and divide by the sum of the generated samples for each realistation of random numbers $j = 1, ..., n$.

```
sample_from_dirichlet_distribution<-function(n,alpha){
  #n=number of samples
  #alpha=parameter vector for the dirichlet distribution, of size K
  #returns: a matrix of size Kxn, with samples from the dirichlet distribution
  ##############################################################################

  #the return matrix
  ret=matrix(ncol=length(alpha),nrow=n)

  #for each alpha we need to generate n samples from the gamma distribution
```

```
  for (i in 1:length(alpha)){
    ret[,i]=gamma_distribution(n,alpha[i],1)
  }

  #Divide by the sum of the samples for each iteration j=1,...,n
  for (j in 1:n){
    ret[j,]=ret[j,]/sum(ret[j,])
  }
  return(ret)
}
```

```
#testing the algorithm
n=1000
alpha=c(0.5,0.9,1,2,3)
#alpha=c(1,2,3)
samples_dirichelt=sample_from_dirichlet_distribution(n,alpha)

#expected value:
alpha0=sum(alpha)
expectedvalue=alpha/alpha0

numeric_exp_value=numeric(length(alpha))
for (i in 1:length(alpha)){
  numeric_exp_value[i]=mean(samples_dirichelt[,i])
}

#variance:
variance=expectedvalue*(1-expectedvalue)/(alpha0+1)

numeric_variance=numeric(length(alpha))
for (i in 1:length(alpha)){
  numeric_variance[i]=var(samples_dirichelt[,i])
}

#covariance:
numeric_covariance=matrix(nrow=length(alpha),ncol=length(alpha))
for (i in 1:length(alpha)){
  for (j in 1:length(alpha)){
    numeric_covariance[i,j]=cov(samples_dirichelt[,i],samples_dirichelt[,j])
  }
}

covariance=matrix(nrow=length(alpha),ncol=length(alpha))
for (i in 1:length(alpha)){
  for (j in 1:length(alpha)){
    covariance[i,j]=-expectedvalue[i]*expectedvalue[j]/(alpha0+1)
  }
}
for (i in 1:length(alpha)){
  covariance[i,i]=variance[i]
}
```

The values for the exact mean, variance and covariance are $E[X_i] = \frac{\alpha_i}{\sum_{k=1}^{K} \alpha_k}$ which gives the expected values

```
## [1] 0.06756757 0.12162162 0.13513514 0.27027027 0.40540541
```

and $Var[X_i] = \frac{E(X_i)(1-E(X_i))}{\sum_{k=1}^{K}\alpha_k+1}$ and $Cov[X_i, X_j] = \frac{-E[i]E[j]}{\sum_{k=1}^{K}\alpha_k+1}$, the resulting covariance matrix is

```
##               [,1]          [,2]         [,3]         [,4]         [,5]
## [1,]  0.0075002609 -0.0009782949 -0.001086994 -0.002173989 -0.003260983
## [2,] -0.0009782949  0.0127178337 -0.001956590 -0.003913180 -0.005869769
## [3,] -0.0010869943 -0.0019565898  0.013913527 -0.004347977 -0.006521966
## [4,] -0.0021739887 -0.0039131796 -0.004347977  0.023479078 -0.013043932
## [5,] -0.0032609830 -0.0058697694 -0.006521966 -0.013043932  0.028696650
```

The mean and variance found for our samples are

```
## [1] 0.06771149 0.12249962 0.13417566 0.27666564 0.39894758
```

and

```
##               [,1]          [,2]         [,3]         [,4]         [,5]
## [1,]  0.0073603401 -0.0004149856 -0.001285121 -0.002065976 -0.003594258
## [2,] -0.0004149856  0.0121968892 -0.001689381 -0.004617351 -0.005475172
## [3,] -0.0012851205 -0.0016893809  0.013257574 -0.004231437 -0.006051635
## [4,] -0.0020659763 -0.0046173511 -0.004231437  0.024543932 -0.013629167
## [5,] -0.0035942577 -0.0054751715 -0.006051635 -0.013629167  0.028750232
```

One can observe that the exact mean and variance are close to the numerical mean and variance, thus it seems like the algorithm worked.

Problem D: Rejection sampling and importance sampling

In this exercise we are looking at the multinomial distribution for a set of data $\boldsymbol{y} = (y_1, y_2, y_3, y_4)$. The multinomial mass function is $f(\boldsymbol{y}|\theta) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}$. The bayes theorem can be used to compute the posterior distribution, the theorem is

$$f(\theta|x) = \frac{f(x|\theta)f(\theta)}{f(x)}.$$

The denominator does not depend on θ, thus the posterior distribution can be written as

$$f(\theta|x) \propto f(x|\theta)f(\theta)$$

Using a uniform prior we can now write the posterior distribution as

$$f(\boldsymbol{y}|\theta) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}$$

since the probability density function of the uniform distribution is a constant.

D1.

In this section we use rejection sampling from $f(\theta|y)$ with uniform proposal density.

```
f_theta_given_y<-function(theta,y){
  #theta in (0,1)
  #y=parameter
  #returns: pdf for posterior density function
  ###########################################

  return((2+theta)^y[1]*(1-theta)^(y[2]+y[3])*theta^y[4])
}
```

```r
f_theta_given_y_normalized<-function(theta,y){
  #normalize the posterior probability density function with the built in function integrate()

  #normalizing constant
  b=integrate(f_theta_given_y,0,1,y)
  a=b$value
  return(f_theta_given_y(theta,y)/a)
}



sample_from_f_theta_given_y<-function(n,y){
  #n= number of samples
  #y= parameter vector
  #returns: a vector of n samples from the posterior distribution,
  #         and a vector of n iteration count values
  ##################################################################

  #find the maximum theta
  x=seq(0,1,0.01)
  theta_index=which.max(f_theta_given_y(x,y))
  theta_max=x[theta_index]

  #return vector
  ret=numeric(n)

  #iteration counter
  itcount=numeric(n)
  for (i in 1:n){
    finished=0
    while(finished==0){
      theta=runif(1)

      #acceptance probability
      #alpha=1/c*f_theta_given_y(theta,y)
      alpha=f_theta_given_y(theta,y)/(f_theta_given_y(theta_max,y))

      #check if a random number in (0,1) is smaller than acceptance probability
      u=runif(1)
      if (u<=alpha){
        finished=1
        ret[i]=theta
        itcount[i]=itcount[i]+1
      }
      else{
        itcount[i]=itcount[i]+1
      }
    }
  }
  return(c(ret,itcount))
}
```

```r
#test the algorithm
n=1000
y=c(125,18,20,34)

posterior_samples=sample_from_f_theta_given_y(n,y)
posterior_samples=posterior_samples[1:n]
```

```r
#plotting the histogram and the probability density function in the same plot
xmin=0
xmax=1
x <- seq(xmin,xmax,0.001)

posterior_data <- data.frame("posterior_samples"=posterior_samples)
ggplot(data=posterior_data, aes(posterior_data$posterior_samples)) +
  geom_histogram(aes(y=..density..), breaks = seq(xmin, xmax, by = 0.05)) +
  labs(title="Posterior distribution", x="x", y="Density") +
  xlim(c(xmin,xmax)) +
  stat_function(fun = f_theta_given_y_normalized, args = list(y))+
  theme(plot.title = element_text(hjust = 0.5))
```

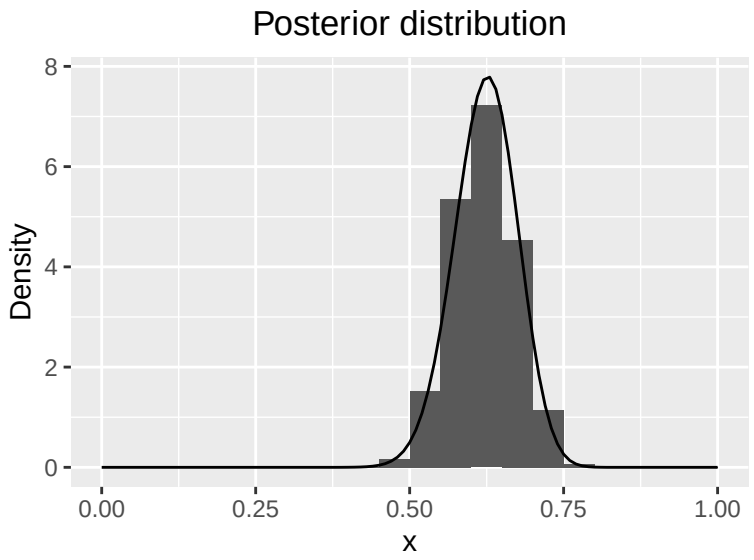

Figure 8: A histogram of the generated random numbers compared to the known posterior probability density function.

It can be observed in Figure 8 that the theoretical probability density is coherent with the samples from the algorithm.

D2.

In this section we want to estimate the posterior mean of θ by Monte-Carlo integration using $M = 10000$ samples from $f(\theta|y)$. The formula for the Monte Carlo estimate of the mean $\mu = E[h(x)]$ is

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} h(x_i)$$

In our case we get $h(\theta) = \theta$.

```r
#sample
M=10000
theta=sample_from_f_theta_given_y(M,y)
theta=theta[1:M] #since the last elements are the iteration counter

#the Monte-Carlo estimate
muhat=1/M*sum(theta)

#the exact solution
int_func<-function(theta,y){
  return(theta*f_theta_given_y_normalized(theta,y))
}
exact_mu=integrate(int_func,0,1,y)
```

The found value for the posterior mean is 0.6227843, while the exact value was found with the function integrate() to be 0.6228061. These values are very similar.

```r
#plotting the histogram and the probability density function in the same plot
xmin=0
xmax=1
x <- seq(xmin,xmax,0.001)

posterior_data <- data.frame("posterior_samples"=posterior_samples)
ggplot(data=posterior_data, aes(posterior_data$posterior_samples)) +
  geom_histogram(aes(y=..density..), breaks = seq(xmin, xmax, by = 0.05)) +
  labs(title="Posterior distribution", x="x", y="Density") +
  xlim(c(xmin,xmax)) +
  stat_function(fun = f_theta_given_y_normalized, args = list(y)) +
  geom_vline(xintercept = muhat,color="red")+
  theme(plot.title = element_text(hjust = 0.5))
```

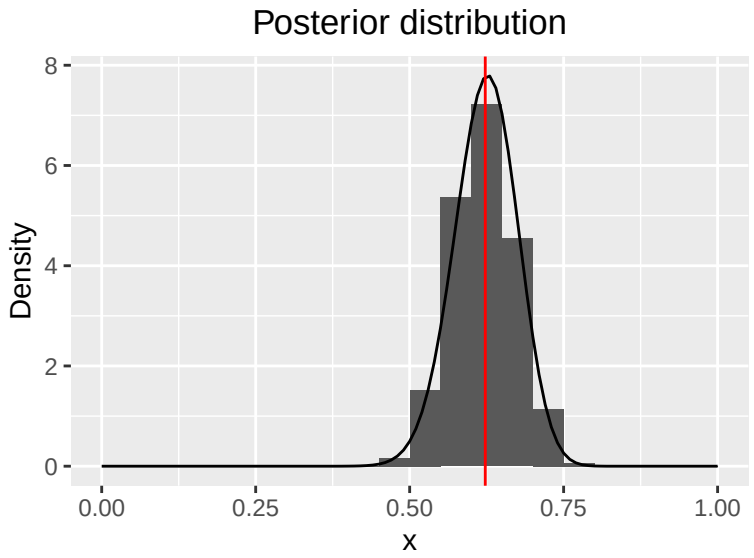

Figure 9: A histogram of the generated random numbers compared to the known posterior probability density function. The estimated posterior mean is marked by a line.

In Figure 9 the line for the estimated posterior mean is added. One can see that it is fairly centered in the

plot.

D3.

In this section we want to find the number of random numbers generated by the algorithm on average to generate one sample.

The theoratical value of the expected number of trials is found by noticing that the trials are independent and that the probability of success is given by

$$P(U \leq \frac{1}{c}\frac{f(X)}{g(X)}) = \int_{-\infty}^{\infty} \frac{f(x)}{cg(x)} g(x)dx = \int_{-\infty}^{\infty} \frac{f(x)}{c} dx = c^{-1},$$

thus one can conclude that the expected number of trials is c.

```
n=10000
count_samples=sample_from_f_theta_given_y(n,y)
itcount=tail(count_samples,length(count_samples)-n)
sampler_average_count=sum(itcount)/length(itcount)

#expected number is c:

#find the maximum theta
x=seq(0,1,0.0001)
theta_index=which.max(f_theta_given_y_normalized(x,y))
theta_max=x[theta_index]

#find the value of c
c_value=f_theta_given_y_normalized(theta_max,y)
```

The theoretical mean number of generated samples per accepted sample is c, as the overall acceptance probability is $\frac{1}{c}$, the value found is 7.799307. While the average number of generated samples from the samples is 7.752. These values are fairly similar.

D4.

Now we want to assume a Beta(1,5) prior and use importance sampling weights to estimate the posterior mean based on the previous samples. The probability density function for the Beta distribution is

$$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, x \in (0,1)$$

So the probability density function for Beta(1,5) is

$$f(x) = \frac{\Gamma(6)}{\Gamma(5)}(1-x)^4 = 5(1-x)^4$$

Thus the posterior distribution is given as

$$f(\theta|y) \propto f(\theta)f(y|\theta) = 5(1-\theta)^4(2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}$$

The importance sample method uses re-weighting of an ausilirary distribution $g(x)$. The weights are given as $w(x_i) = \frac{f(x_i)}{g(x_i)}$. The estimate of the mean is

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

This estimate is biased for finite n, it is consistent and self-normalizing.

```r
posterior_given_Beta_prior <- function(theta,y){
  #theta in (0,1)
  #y=parameter
  #returns: pdf for new posterior density function with Beta(1,5) prior
  ######################################################################

  return(5*(1-theta)^4*f_theta_given_y(theta,y))
}


weights <- function(theta,y){
  #calculate the weights for the importance sampling

  return(posterior_given_Beta_prior(theta,y)/f_theta_given_y(theta,y))
}
```

```r
#test the importance sampling
M=10000

#the samples
theta=sample_from_f_theta_given_y(M,y)
theta=theta[1:M]

#calculate the mean
importance_sample_mean=sum(theta*weights(theta,y))/sum(weights(theta,y))
```

```r
#plotting the probability density function in the same plot as the estimated mean
xmin=0
xmax=1
x <- seq(xmin,xmax,0.001)

posterior_data <- data.frame("posterior_samples"=posterior_samples)
ggplot(data=posterior_data, aes(posterior_data$posterior_samples)) +
  labs(title="Posterior distribution", x="x", y="Density") +
  xlim(c(xmin,xmax)) +
  stat_function(fun = posterior_given_Beta_prior, args = list(y)) +
  geom_vline(xintercept = importance_sample_mean,color="red")+
  theme(plot.title = element_text(hjust = 0.5))
```

The value of the estimated posterior mean is found to be 0.596144, this value is plotted in Figure 10 together with the posterior probability distribution without the normalizing constant. One can observe that the value lies in the expected position, which is at the middle point of the distribution.

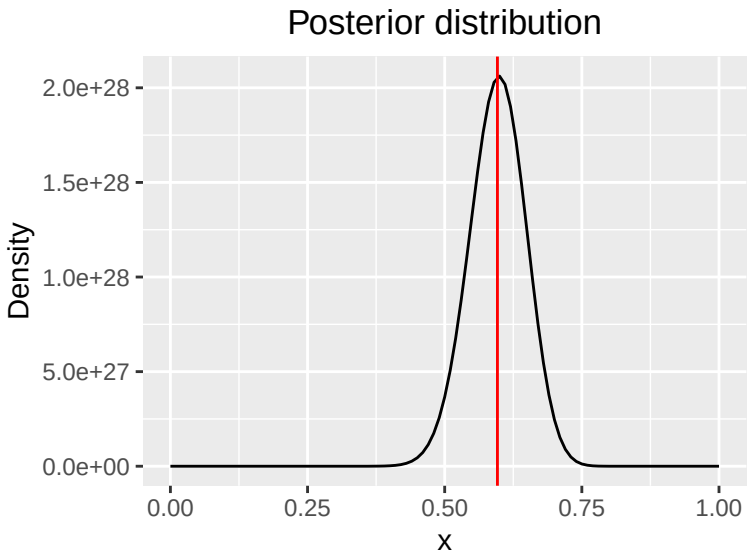

Figure 10: The known posterior probability density function with the estimated posterior mean is marked by a line.