Assignment 1, TMA4300 Computer intenstive statistical methods

Pål Vegard Johnsen and Kristin Benedicte Bakka.

February 20, 2017

A Stochastic simulation by the probability integral transform

A.1 Exponential distribution

The first task is to generate samples from the exponential distribution. One may use the the Probability Integral Transform method (PIT) to generate random samples from an exponential distribution. Given the cumulative distribution function $F(x) = 1 - exp(-\lambda x)$ with rate λ, the inverse function is easily computed as $F^{-1}(F(x)) = -\frac{1}{\lambda}\log(1 - F(x))$. Using PIT an algorithm producing n samples is then given by:

```
expgenerator = function (lambda, n){

  u = runif(n);

  x = (-1/lambda)*log(1-u)

return(x)

}
```

The intuition behind the method is that we draw the cumulative distribution function (cdf), sample a random variable $u \sim U(0, 1)$, and look at what value on the x-axis has $F(x) = u$.

In order to check if the implementation is done correctly, we apply the following code and obtain figure 1. The estimate for the rate is given by the maximum likelihood estimator, and the resulting plot is displaed in figure **??**. Both of these behaves as expected, and we move on to the next task.

```
x = expgenerator(lambda = 2, n = 1000)
hist(x, prob = TRUE,main='')
library(MASS)
x.est <- fitdistr(x, "exponential")$estimate
print(x.est)
```

```
##     rate
## 2.058858
```

```
curve(dexp(x, rate = x.est), add = TRUE, col = "red", lwd = 2)
```

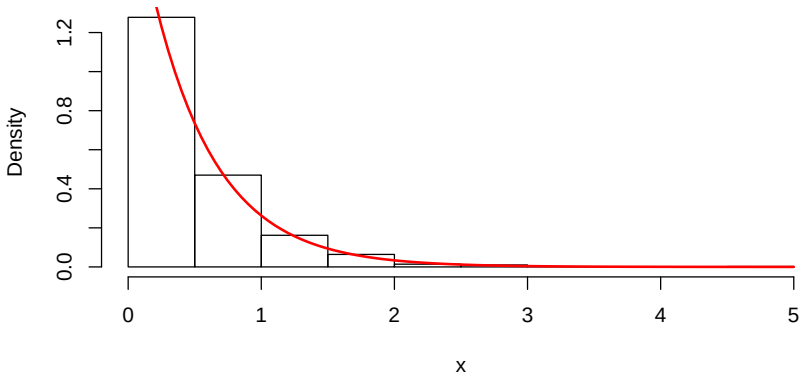


Figure 1: Histogram of 1000 pseudorandom draws from an exponential distribution together with the probability density function of the exponential distribution.

A.2 Logistic distribution

The next task is to sample from the logistic distribution with mean 0 and scale parameter $\frac{1}{\alpha}$, which has pdf

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

where c is a normalizing constant. We will use PIT to do this, just as we did in the previous exercise.

A.2 (a) Normalizing constant

First c needs to be found. Integrating over the whole x-axis, c must satisfy:

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

which implies that $c = \alpha$.

A.2 (b) Cumulative distribution and inverse function

To use PIT the inverse cdf, $g(y)$, must be found. From the definition the cdf becomes

$$F(x) = \alpha\int_{-\infty}^{x} \frac{e^{\alpha x'}}{(1+e^{\alpha x'})^2}dx' = \left[\frac{e^{\alpha x'}}{1+e^{\alpha x'}}\right]_{-\infty}^{x} = \frac{e^{\alpha x}}{1+e^{\alpha x}}.$$

Now inverse function can be found by direct computation

$$g(y) = \frac{1}{\alpha}\log\frac{y}{1-y}.$$

A.2 (c) Generate samples from logistic distribution

Now all that is left is to write a code that generates the desired amount of samples, and to run the code to check for implementation errors. In order to generate samples from this distribution, the same method is used as for the exponential, and the algorithm is given by:

```
logisticgenerator = function (alpha, n){
  #Generate random samples
  u = runif(n);
  #Find corresponding x values
  x = (1/alpha)*log(u/(1-u))
  return(x)
}
```

As before we check how proper this generator is. The sample is displayed in figure 2. The estimated parameters comes out fairly close to the target sample parameters, and we are satisfied that the implementation might be correct. Note that R uses scale parameter while we use rate, so that scale should equal $\frac{1}{\alpha}$, and that location should equal 0.

```
x = logisticgenerator(alpha = 2, n = 1000)
hist(x, prob = TRUE,main='',ylim=c(0,0.5))

library(MASS)
x.est <- fitdistr(x, "logistic", lower = c(-Inf, 0) )$estimate
print(x.est)
```

```
##   location      scale
## 0.02381618 0.49801368
```

```
curve(dlogis(x, location = 0, scale = 0.5 ), add = TRUE, col = "red", lwd = 2)
```

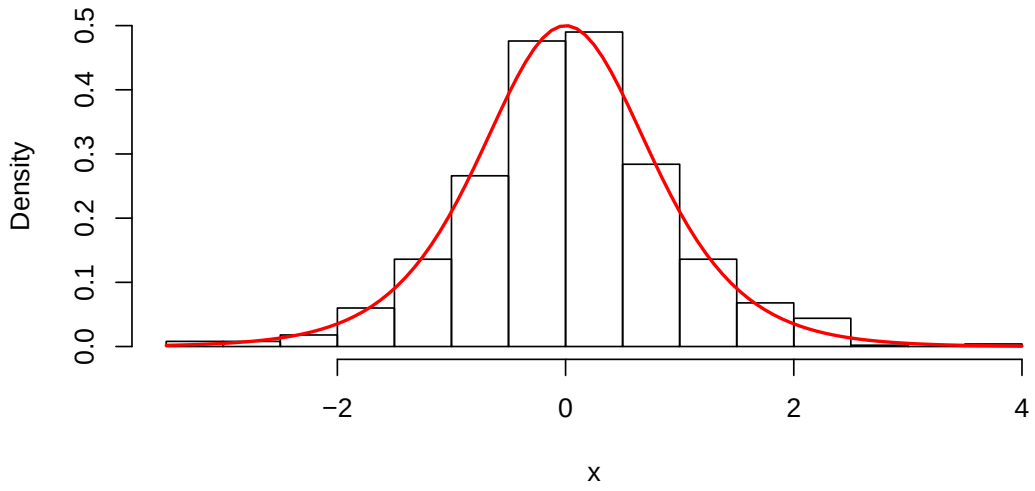


Figure 2: Histogram of 1000 samples from logistic distribution, together with fitted pdf.

A.3 Distribution with piecewise continuous probability density function

The distribution we want to draw from now has pdf

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

To find the cumulative distribution function we integrate g(x) from minus infinity to the value we are looking for. When $x \geq 1$

$$\begin{aligned} G(x) &= \int_{w=0}^{1} cw^{\alpha-1}dw + \int_{w=1}^{x} ce^{-w}dw \\ &= c(\frac{1}{\alpha} - [e^{-w}]_{w=1}^{x}) \\ &= c(\frac{1}{\alpha} + e^{-1} - e^{-x}) \end{aligned}$$

and

$$1 = \lim_{x\to\infty} G(x) = c(\alpha^{-1} + e^{-1}) \, ,$$

such that

$$c = (\alpha^{-1} + e^{-1})^{-1} \, .$$

Thus the cumulative distribution can be expressed as

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

Since $G(1) = (1 + ce^{-1})^{-1}$ the inverse function is expressed as

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

such that the following algorithm generates pseudorandom samples from the distribution.

```
a3generator = function (alpha,beta=1,n=1000){
    u = runif(n)
    c= 1/(alpha^-1+exp(-1))
    u=ifelse(u < (1+exp(-1)*alpha)^-1,((alpha/c)*u)^(alpha^-1),-log(1-u)+log(c))
    return(u)
}
```

The following plot, that is the plot in figure 3, shows the distribution of a sample of 10000 values, along with the pdf. Note that there were observations bigger than 2 as well, but we only chose to show the distribution where x takes values between zero and two. The figure verifies that the correctness of the impelentation.

```
lower = 0
upper = 2
a=0.7
x = a3generator(alpha = a, n = 10000)
hist(x, prob = TRUE,xlim=c(lower,upper),main='')
a3pdf(alpha = a,L=lower,U=upper)
```

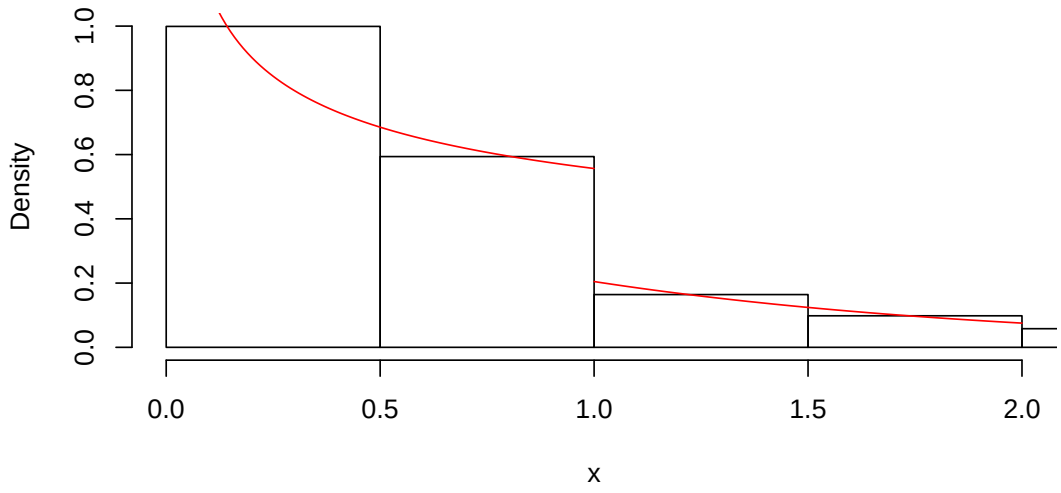


Figure 3: 1000 samples and pdf of distribution in task C.2.

B Rejection sampling and importance sampling

B.1 Rejection sampling

Consider the multinomial distribution $f(\mathbf{y}|\theta)$ with probability given by:

Variable	Probability
$y_1 = 125$	$\frac{1}{2} + \frac{\theta}{4}$
$y_2 = 18$	$\frac{1-\theta}{4}$
$y_3 = 20$	$\frac{1-\theta}{4}$
$y_4 = 34$	$\frac{\theta}{4}$

Therefore $f(\mathbf{y}|\theta) \propto (2+\theta)^{y_1} \frac{1-\theta}{4}^{y_2+y_3} \theta^{y_4}$. Assume θ is uniformly distributed, $\theta \sim Uniform(0,1)$. Then the posterior distribution is then given by:

$$f_1(\theta|\mathbf{y}) \propto (2+\theta)^{y_1} \frac{1-\theta}{4}^{y_2+y_3} \theta^{y_4}.$$

We now want to compute samples from this distribution by rejection sampling using a $Uniform(0,1)$ density as the proposal density, $g(\theta) = 1 \cdot I_{\theta \in (0,1)}(\theta)$. The normalizing constant for $f(\mathbf{y}|\theta)$ is then usefull. The normalizing constant is very big due to the great number of combinations. Using the integral-function in R this can be found. In order to find a bound for $\frac{f(\theta|\mathbf{y}}{g(\theta)}$, the optim function in R may be used. The algorithm for the rejection sampling can then be given by:

```
# Rejection sampling method. f(x) are known.

rejection = function(y1,y2,y3,y4,density, lo, up){

  max = optim(par = 0.5,density, method = "L-BFGS-B", control = list(fnscale = -1),
  lower = lo,  upper = up) # Find for which theta the maximum appears
  N = integrate(density, lo,up)
  c = as.numeric(max[2])/as.numeric(N[1]) # Upper bound for f(x)/g(x)
  a = (1/(as.numeric(N[1]))) # Normalizing constant
  finished = 0 # End while loop when sample is legal
  counter = 0 # Counter for how many iterations needed before legal sample
  while(finished == 0){
    x = runif(1)
    alpha = (1/c)*a*((2+x)^y1)*((1-x)^(y2+y3))*x^y4
    u = runif(1) # Proposal density is uniformly distributed
    if(u <= alpha){
      finished = 1

    }
    else{
      counter = counter + 1
    }

  }
counter = counter + 1
l = list(x,counter)
return(l)


}
```

B.2 Monte Carlo method in order to estimate the posterior mean of θ

In order to estimate the posterior mean of θ, we use the Law of large numbers such that the estimator of the mean $\bar{Y} = \sum_{i=1}^{N} Y_1$ converges to the mean as $N \to \infty$. This is known as the Monte Carlo Method and an algorithm to estimate the posterior mean with $N = 10000$ samples is given by:

```
density = function(x){
  ((2+x)^125)*((1-x)^(38))*x^34
}

# Monte Carlo method in order to estimate the mean:
it = 10000 # Number of iterations/samples
lo = 0 # lower bound for density
up = 1 # upper bound for density
sample = vector(mode = "numeric", it) #Save samples in a vector
iterations = vector(mode = "numeric",it)
for(i in 1:it){
s = rejection(125,18,20,34,density,lo,up)
sample[i] = as.numeric(s[1])
iterations[i] = as.numeric(s[2])
}
```

```r
# Estimated posterior mean
EsPoMean = mean(sample)
EsPoMean
```

```
## [1] 0.6232134
```

```r
# Average number of iterations needed to generate a sample from f(x)
meanIterations = mean(iterations)
meanIterations
```

```
## [1] 7.8406
```

B.3 Estimated number of iterations needed

In order to compute the efficiency of the algorithm, we notice that the distribution for number of iterations needed before successfull sample is geometrically distributed. The probability to get a successfull sample is given by

$$\begin{aligned} P(U \leq \frac{1}{c}\frac{f(\Theta|\mathbf{y})}{g(\Theta)}) &= \int_0^1 P(U \leq \frac{1}{c}\frac{f(\Theta|\mathbf{y})}{g(\Theta)}|\Theta = \theta)g(\theta)d\theta \\ &= \frac{1}{c}\int_0^1 f(\Theta|\mathbf{y})d\theta = \frac{1}{c} \end{aligned}$$

The mean, μ of a geometric density with probability, $P(X = 1) = p$, is given by $\mu = \frac{1}{p}$. So the efficiency of the rejection algorithm, μ_r, can be computed to be $\mu_r = 7.8$. The estimation by counting the number of iterations and taking the mean after 10 000 iterations is given by $\bar{\mu_r} = 7.8406$, which is quite close to the theoretical mean.

B.4 Importance sampling

Suppose now that the prior distribution is $\pi(\theta) \sim Beta(\alpha = 1, \beta = 5)$. The new posterior distribution, $f(\theta\|\mathbf{y})$, can be found by the regular Bayes' rule:

$$f_2(\theta|\mathbf{y}) = \frac{f(\mathbf{y}|\theta)\pi(\theta)}{\int(f(\mathbf{y}|\theta)\pi(\theta)d\theta} \propto (2+\theta)^{y_1}\Big(\frac{1-\theta}{4}\Big)^{y_2+y_3+4}\theta^{y_4}$$

Importance sampling may now be used to find the posterior mean, $E_{f_2}[\theta|\mathbf{y}]$, using the samples produced when the prior distribution where uniform as above. Notice that:

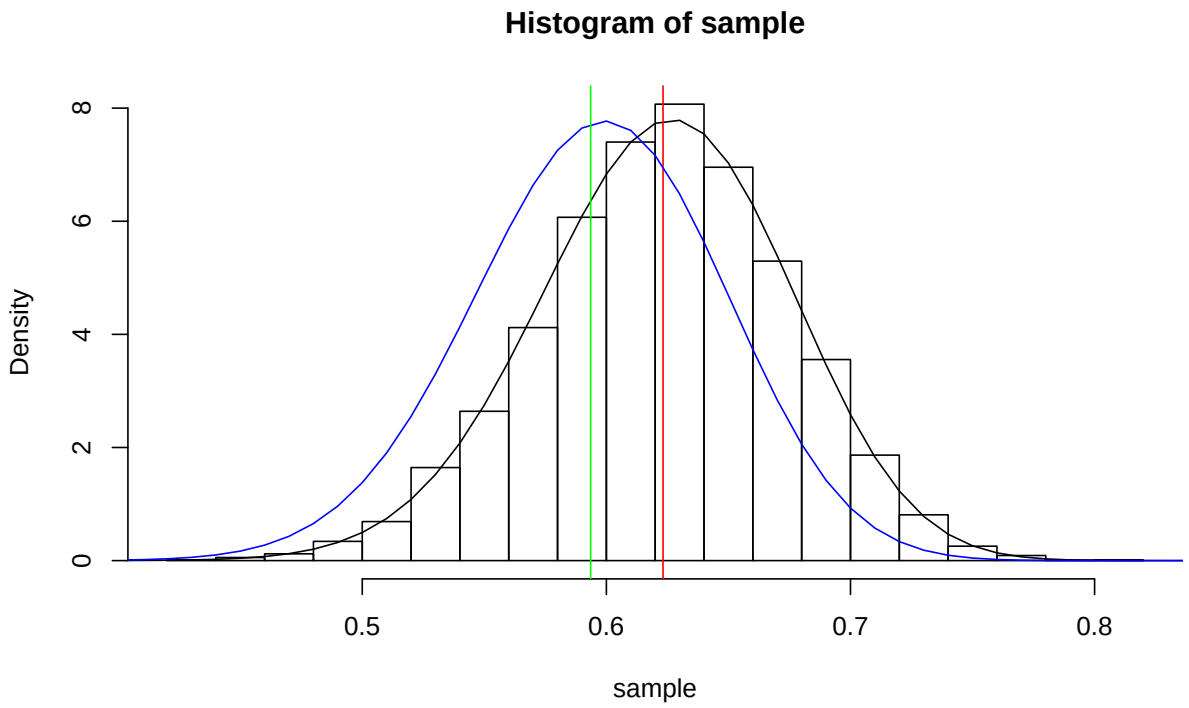
$$\begin{aligned} E_{f_2}[\theta|\mathbf{y}] &= \int_0^1 \theta f_2(\theta|\mathbf{y})d\theta = \int_0^1 \theta\frac{f_2(\theta|\mathbf{y})}{f_1(\theta|\mathbf{y})}f_1(\theta|\mathbf{y})d\theta \\ &= \frac{c_1}{c_2}\int_0^1 (1-\theta)^4\theta f_1(\theta|\mathbf{y})d\theta = \frac{c_1}{c_2}E_{f_1}[(1-\theta)^4\theta|\mathbf{y}], \end{aligned}$$

where c_1 and c_2 are normalizing constants. Using the samples from above, we can now estimate the posterior mean using the Monte Carlo method. The algorithm can then be:

```r
# Importance sampling. Given prior density equal to beta(1,5), new posterior density is f(x) = (2+x)^y1
densityposterior = function(x){
  ((2+x)^(125))*((1-x)^(42))*x^34
}
# Compute normalizing constants
N = integrate(density, lo,up) # Normalizing constant for f_1(x)
N2 = integrate(densityposterior,0,1)
a2 = 1/as.numeric(N2[1])

# IMPORTANCE SAMPLING to estimate posterior mean:

NewEsPoMean = 0
for(i in 1:it){

NewEsPoMean = NewEsPoMean + (as.numeric(N[1])/as.numeric(N2[1]))*sample[i]*(1-sample[i])^4

}

NewEsPoMean = NewEsPoMean/it
```

A plot can now be given to summarize what we have done. The posterior density with Uniform prior is plotted with dark lines with a corresponding histogram of the sample generated by the rejection method. The red vertical line represents the estimated posterior mean. The blue line represents the posterior density with prior $Beta(1,5)$, and the corresponding vertical green line represents the estimated posterior mean by the importance sample method.

```r
# Generate histogram from f_1

hist(sample, freq = FALSE)
# Plot density og f_1 in same graph
a = (1/(as.numeric(N[1]))) # Normalizing constant
curve(a*((2+x)^(125))*((1-x)^(38))*x^34,from = 0, to = 1, add = TRUE )
# Add vertical line for Estimated posterior mean for f_1
abline(v = EsPoMean, col = "red")
# Add plot of distribution of f_2
curve(a2*((2+x)^(125))*((1-x)^(42))*x^34, from = 0, to = 1, add = TRUE, col = "blue" )
# Add vertical line for the estimated posterior mean of f_2
abline(v = NewEsPoMean, col = "green")
```



C Stochastic simulation by bivariate techniques and rejection sampling

C.1 Box-Muller algorithm for iid standard normal.

This task is to generate samples from the standard normal distribution. Generateing n independent samples from the standard normal distribution is accomplished with the Box-Muller algorithm implemented below.

```
stdnormalgenerator = function (n=1000){
  #generate random numbers
  u = 2*pi*runif(n/2)
  x = expgenerator(1/2,n/2)

  #calculate normal distributed ones
  return(c(sqrt(x)*sin(u), sqrt(x)*cos(u)))
}
```

In order for the function to be able to generate an odd number of samples this is revised to

```
stdnormalgenerator = function (n){
    u = 2*pi*runif(floor(n/2))
    x = expgenerator(1/2,floor(n/2))

    #even number
    if(n/2 == floor(n/2)){
    return(c(sqrt(x)*sin(u), sqrt(x)*cos(u)))
```

```
    }

    #odd number, need 1 more
    v = 2*pi*runif(1)
    y = expgenerator(1/2,1)
    return(c(sqrt(x)*sin(u), sqrt(x)*cos(u), sqrt(y)*cos(v)))
}
```

which does the same, but don't look as pretty. To verify that the code works, the sample in figure 4 is drawn. The histogram and the pdf look like they come from the same distribution, and so it is likely that the code is correct.

```
x = stdnormalgenerator(n=1000)
hist(x, prob = TRUE,xlim=c(-3,3),ylim=c(0,0.5),main=' ')
x1=seq(-3,3,length=1000)
f1=dnorm(x1,0,1)
lines(x=x1,y=f1, col = "red",lwd='2')
```

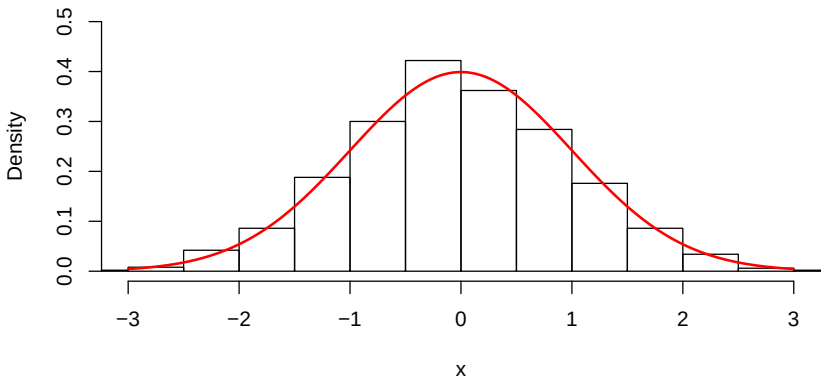


Figure 4: Histogram of 1000 pseudorandom draws from a standard normal distribution together with the probability density function of the standard normal distribution.

C.2 Rejection sampling for Gamma distribution, $0 < \alpha < 1$ and $\beta = 1$

The pdf of the Gamma distribution is

$$f(x) = \begin{cases} 0 & x \leq 0 \\ \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} & 0 < x \end{cases} \tag{2}$$

and for the moment we will let $0 < \alpha \leq 1$ and $\beta = 1$. We want to sample from this distribution, and will use rejection sampling to accomplish it. The principle of rejection sampling is to first draw x from a simple distribution, and then with acceptance probability a decide whether to keep the value, or to try again. Denoting by $f(x)$ the distribution we want to sample from and by g(x) the simpler distribution, the acceptance probability is $a = k\frac{f(x)}{g(x)}$, so it is required that $k\frac{f(x)}{g(x)} \leq 1$ for all x, where k is some constant. Letting g(x) be the distribution with pdf in equation (1) and noting that only parts of $g(x)$ and $f(x)$ that

contain x are of interest, the requirement becomes

$$k_1 \frac{x^{\alpha-1}e^{-\beta x}}{x^{\alpha-1}} \leq 1 \, ,$$

when $0 < x < 1$, and

$$k_2 \frac{x^{\alpha-1}e^{-\beta x}}{e^{-\beta x}} \leq 1 \, ,$$

when $1 \leq x$. Since the first expression is maximal for $x = 0$, and the second is maximal for $x = 1$, the expressions become $k_i \leq 1$, and the maximal value for k is $k = 1$, such that the acceptance probability becomes

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

The following algorithm samples from the distribution.

```
rejectionGamma = function (alpha=0.5,beta,n=1000){
  i=0;
  sample = rep(NaN,n)
  while (i<n){
    x = a3generator(alpha,1,1)
    accept = ifelse (x<1,exp(-x), x^(alpha-1))
    u = runif(1)
    if(u <= accept){
      i = i+1 #R is 1-indexed
      sample[i] = x/beta
    }
  }
  return(sample)
}
```

To check the code for bugs a sample is made with the following code, and the result is displayed in figure 5.

```
#Draw sample
x = rejectionGamma(alpha=0.5,beta=3,n=1000)
hist(x, prob = TRUE,xlim = c(lower,upper),main=' ')
library(MASS)
m <- fitdistr(x[x!=0], "gamma" )$estimate
#Test generated alpha and beta to see if approx 0.5 and 3
print(m)
```

```
##    shape     rate
## 0.489354 2.784107
```

```
#Draw figure
curve(dgamma(x, m[1],m[2]), add = TRUE, col = "red", lwd = 2)
```

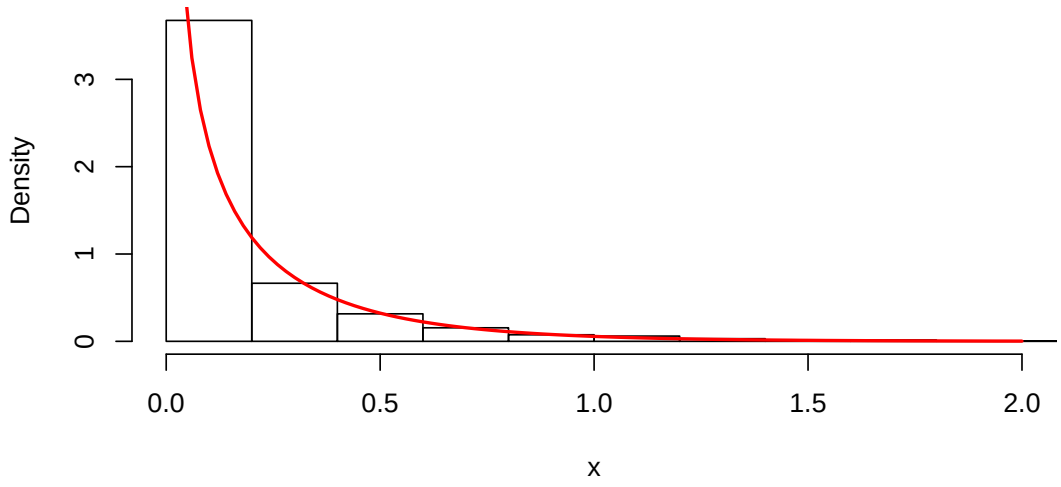


Figure 5: Histogram of 1000 draws from gamma distribution. The red line is the pdf of the fitted gamma distribution.

The plot of the fitted gamma distribution in figure 5 and the histogram of the sample look like they represent the same distribution. The fitted parameters are approximately equal to the sample parameters, and multiple runs show that the fitted parameters are sometimes too large and sometimes too small, with no apparent pattern. Together this means that it is likely that the code is free of bugs.

C.3 Ratio of uniforms method for Gamma distribution, $1 < \alpha$

The idea in the rate of uniforms method is to construct a representation C_f of the area under the pdf, and then sample a point from that area such that the probability to draw each point is uniform. The corresponding value on the x-axis comes from the intended distribution.

In practice we might do much the same as in rejection sampling. First we draw u_1 and u_2 uniformely distributed on $C_{max} = [0, a] \times [b_-, b_+]$, if the resulting point is on C_f we keep the corresponding x-value, else we try again. Note that the process of rejecting a value is not pseudorandom in this method. To draw from the Gamma distribution in equation (2), but now with $1 < \alpha$ we define the function

$$f^\star(x) = \begin{cases} 0, & x \leq 0 \\ x^{\alpha-1}e^{-\beta x} & 0 < x \end{cases}$$

and the entities

$$\begin{aligned} a &= \sqrt{\sup_x(f^\star(x))}\,, \\ b_+ &= \sqrt{\sup_{0\leq x}(x^2 f^\star(x))}\,, \\ b_- &= \sqrt{\sup_{x\leq 0}(x^2 f^\star(x))}\,, \\ t(p) &= \sqrt{f^\star(p)}\,, \end{aligned}$$

and define $C_f = \{(x_1, x_2) : 0 < x_2 < b_+, 0 < x_1 < t(p) \text{ with } p = \frac{x_2}{x_1}$. Noting that $\sup_x(x^{k_1}e^{-k_2 x})$ is for $x = \frac{k_1}{k_2}$, we get $\sup_x(x^{k_1}e^{-k_2 x}) = ((\frac{k_1}{k_2})^{k_1}e^{-k_1}$ and thus

$$a = \sqrt{(\frac{\alpha-1}{\beta})^{\alpha-1}e^{-(\alpha-1)}}\,, b_+ = \sqrt{(\frac{\alpha+1}{\beta})^{\alpha+1}e^{-(\alpha+1)}}\,, b_- = 0\,,$$

and the resulting algorithm that draws from the Gamma distribution when $1 < \alpha$ becomes:

```
ratioGamma = function (alpha,beta=1,n=1000,tries=FALSE){
  i=0;
  tries=0;
  sample = rep(NaN,n)
  #construct limits of C_max
  a=((alpha-1)/beta * exp(-1))^((alpha-1)/2)
  bmax=((alpha+1)/beta * exp(-1))^((alpha+1)/2)
  while (i<n){
    #Draw a point in C_max
    u1 = runif(1)*a
    u2 = runif(1)*bmax
    temp= u2/u1
    #Is the point in C_f?
    t=sqrt(temp^(alpha-1)*exp(-beta*temp))
    if(u1 <= t){
      i = i+1
      sample(i)=temp
    }
    tries=tries+1
  }
  if(tries){
    return(tries)
  }
  return(sample)
}
```

Except these numbers become very large, so instead the calculations should be made in logscale. The entities needed are

$$\begin{aligned} log(a) &= \frac{\alpha-1}{2}(log(\alpha-1) - log(\beta) - 1)\,, \\ log(b_+) &= \frac{\alpha+1}{2}(log(\alpha+1) - log(\beta) - 1)\,, \\ log(t(p)) &= \frac{\alpha-1}{2}log(x) - \frac{1}{2}\beta x\,, \end{aligned}$$

such that the algorithm in log scale is as follows:

```
logRatioGamma = function (alpha,beta=1,n=1000,tries=FALSE){
  i=0;
  attempts=0;
  sample = rep(NaN,n)
  #construct limits of C_max
  la=(alpha-1)*0.5*(log(alpha-1)-log(beta)-1)
  lbmax=(alpha+1)*0.5*(log(alpha+1)-log(beta)-1)
  while (i<n){
    #Draw a point in C_max
    u1 = log(runif(1)) + la
```

```
    u2 = log(runif(1)) + lbmax
    ltemp= u2-u1
    #Is the point in C_f?
    lt=((alpha-1)*ltemp-beta*exp(ltemp))/2
    if(u1 <= lt){
      i = i+1
      sample[i]=exp(ltemp)
    }
    attempts=attempts+1
  }
  if(tries){
    return(attempts)
  }
  return(sample)
}
```

First we draw from the distribution to test the code for bugs using the following code.

```
a=2
b=3
x = logRatioGamma(alpha=a,beta=b,n=1000)
hist(x, prob = TRUE,xlim = c(0,2.2),ylim = c(0,1.4), main='')
library(MASS)
m <- fitdistr(x[x!=0], "gamma" )$estimate
## Warning in densfun(x, parm[1], parm[2], ...):  NaNs produced
## Warning in densfun(x, parm[1], parm[2], ...):  NaNs produced
print(m)
##    shape     rate
## 1.969780 2.975946
curve(dgamma(x, m[1],m[2]), add = TRUE, col = "red", lwd = 2)
```

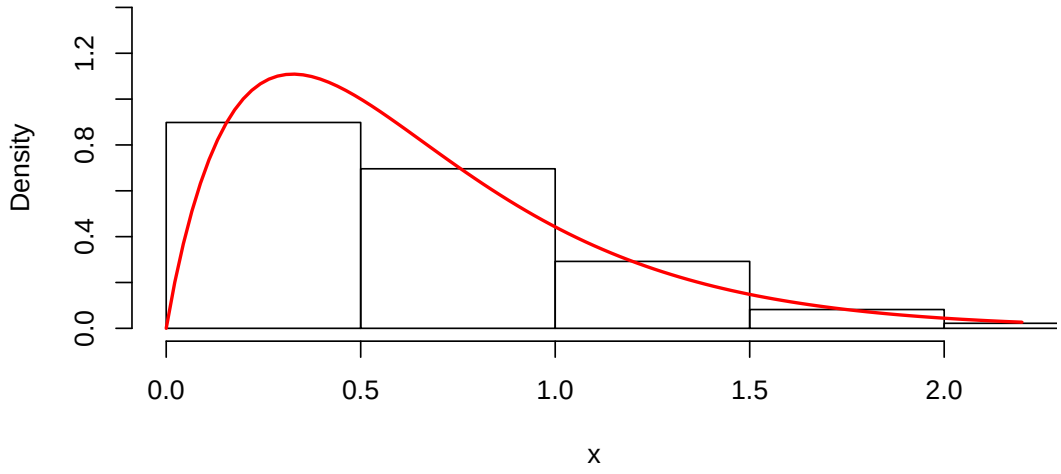


Figure 6: 1000 samples from Gamma distribution.

The resulting plot is displayed in figure 6. The histogram follows the curve nicely, and the fitted parameters are approximately what they should be. Note that the first bar in the histogram looks like it is too small in most of the area it covers. This shape is reasonable, as it covers a part of the x-axis with large change in the probability density. Unfortunately the code that plots the distribution yields several a warnings, but this does not seem to affect the result. Concluding that the implementation works as intended, we move on to find number of attempts needed to find 1000 samples for different α.

```
numberOfTries = function(beta=1, n=1000, xmin=1.01, xmax=2000,l=200){
  x=seq(xmin, xmax, length.out = l)
  y=rep(NaN,l)
  for (i in 1:l){
    y[i]=logRatioGamma(alpha=x[i],beta=1,n=1000,tries=TRUE)
  }
  plot(x,y,'l',xlab = '\alpha', ylab = 'Tries', main=' ')
}
numberOfTries()
```

```
## Warning in title(...):  font width unknown for character 0x7
```

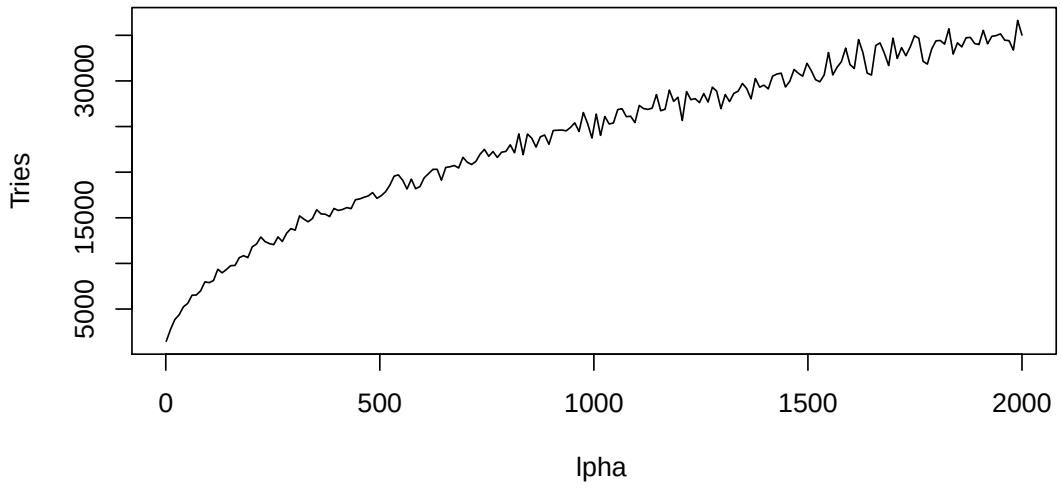


Figure 7: Number of tries needed to draw 1000 samples from gamma distribution with different shape parameters.

Figure 7 displays that the number of tries increase almost linearly with *alpha*. Number of supurflous tries is the number of tries where a point in C_{max} which is not in C_f is drawn, which is proportional to $r = \frac{C_{max}-C_f}{C_{max}}$, that is the area of C_{max} not covered by C_f. This expression becomes

$$r = \frac{1}{ab_+}(ab_+ - \int_0^{b_+} x^{\frac{1}{2}(\alpha-1)} e^{\frac{1}{2}\beta x})$$

which for $\beta = 1$, and defining $k = \frac{C_f}{C_{max}}$ we get

$$k = 2^{\frac{1}{b}(\alpha+1)} a^{-1} b^{-\frac{1}{2}(\alpha+1)} \beta^{-\frac{1}{2}(\alpha+1)} (\Gamma(\frac{1}{2}(\alpha+1)) - \Gamma(, \frac{b}{2})) \ ,$$

which is quite a messy expression. Ideally we would plot $1000r(\alpha)$ together with the actual number of tries in figure 7, but we did not think of that in time, and the incomplete gamma function is not so straightforward to compute. For small $\alpha s \mathrm{C}_{max}$ is very small, and C_f increase with α. Intuitively C_f is not expected to increase as much with α, as the distribution with higher shape parameter have heavier tails. In the limit when $\alpha = 1$ the distribution is the exponential distribution, and will only have tail on one side. This would make r an increasing function, like the plot in figure 7 indicates.

C.4 Gamma for any $0 < \alpha$ and $0 < \beta$

Finally we want to be able to draw from any Gamma distribution with parameters $0 < \alpha$ and $0 < \beta$. When $0 < \alpha < 1$ we can use the rejection sampling method, and when $1 < \alpha$ we can use the ratio of uniform algorithm. When $\alpha = 1$ the pdf becomes $f(x) = \frac{1}{\Gamma(1)} e^{-\beta x}$ and the distribution is exponential. Following is a simple general algorithm for generating a sample from the Gamma distribution.

```
gammagenerator = function(alpha,beta=1,n=1000){
  if (alpha<1){
    return(a3generator(alpha=alpha,beta=beta,n=n))
  }
```

```
  if(alpha>1){
    return(logRatioGamma(alpha=alpha,beta=beta,n=n))
  }
  return(expgenerator(lambda=beta,n=n))
}
```

D Multivariate distributions

D.1 Multinormal Distribution

In order to generate samples from a d-variate multinormal distribution with mean, μ, and covariance matrix, $\Sigma_{d\times d}$, the trick is to make use of the Box-muller algorithm to generate standard normal samples in order to make a d-variate standard mulitnormal distribution, $N_d(\mathbf{0}, I_{d\times d})$ with mean, $\mu_0 = \mathbf{0}$, and covariance, $\Sigma_0 = I_{d\times d}$.

The next step is to use the well-known fact that if $x \sim N_d(\mathbf{0}, I_{d\times d})$, then $y = \mu + Ax$ is also a d-variate multinormal distribution with mean, μ, and covariance matrix AA^T, and so $y \sim N_d(\mu, AA^T)$. Therefore we need to find a matrix A such $AA^T = \Sigma$. This can be computed for instance by the Cholesky decomposition. An algorithm to generate a d-variate multinormal distribution is then given by:

```
multinormalgenerator = function(mean, covariance,d){
# Inputs are the mean, the covariance matrix and the dimension.

# Generate d univariate standard normal samples
stdnormalsamples = stdnormalgenerator(d)

# Find a matrix a such t(A)*A = \Sigma
A = chol(covariance)

# Generate a d-variate multinormal distribution.
return(mean + t(A)%*%stdnormalsamples)

}
```

The estimated mean and covariance can now be computed and therefore compared with the true mean and covariance. For the d-variate multinormal distribution the Maximum Likelihood-estimator for the mean is given by:

$$\hat{\mu} = \begin{bmatrix} \bar{x_1} \\ \bar{x_2} \\ \vdots \\ \bar{x_d} \end{bmatrix}$$

, where $\bar{x_i} = \frac{1}{n}\sum_{j=1}^{n} x_{ji}$, where x_{ji} is the ith element in sample vector number j.
The estimated Covariance matrix may also be found by the Maximum Likelihood theory, but it turns out that this estimate is slightly unbiased. An unbiased estimator is instead:

$$\hat{\Sigma} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{\mu})(x_i - \hat{\mu})^T$$

D.2 Dirichlet Distribution

A vector $\mathbf{x}$ of K elements with Dirichlet distribution and parameter vector α has probability density function

$$f_D(x_1,\ldots,x_{K-1}) = x_i^{\alpha_i-1}\cdots x_{K-1}^{\alpha_{K-1}-1}\cdot(1-\sum_{i=1}^{K-1}x_i)^{\alpha_K-1}\frac{\Gamma(\sum_{i=1}^{K}\alpha_i)}{\Gamma(\alpha_1)\cdots\Gamma(\alpha_K)}\ , \tag{3}$$

and in addition $\Gamma(\sum_{i=1}^{K})x_i = 1$, $\Gamma(\sum_{i=1}^{K-1})x_i < 1$ and $0 < x_i < 1, \forall i$.

D.2 (a) Relationship between Dirichlet and Gamma distributions

In this section the relationship between Dirichlet and Gamma distributions is shown. In principle this is quite straight forward, and can be found by direct computation. In practise it is easy to mess up these calculations. Therefore we have chosen to include a lot of steps in the computation

We want to prove that using a vector z with independent entries $z_i \sim \text{gamma}(\alpha_i, 1)$, the vector $x = g(z)$ with entries $x_i = z_i v^{-1}$ and $v = \sum_{i=1}^{K} z_i$ is Dirichlet distributed. This is true if and only if

$$f_X(x_1,\ldots,x_{K-1},v) = f_G(g^{-1}(x_1,\ldots,x_{K-1},v))|J|\ . \tag{4}$$

Starting with the left hand side

$$f_X(x_1,\ldots,x_{K-1},v) = f_D(x_1,\ldots,x_{K-1})f_V(v)\ ,$$

as v and x_i are independent. Using that $v \sim \text{gamma}(\sum_{i=1}^{K}\alpha_i, 1)$ and the distribution in (2) we have

$$f_V(v) = \frac{1}{\Gamma(\sum_{i=1}^{K}\alpha_i)}v^{(\sum_{i=1}^{K}\alpha_i)-1}e^{-v}\ . \tag{5}$$

Multiplying (3) and (5) the it becomes

$$f_X(x_1,\ldots,x_{K-1},v) = e^{-v}\,v^{(\sum_{j=1}^{K}\alpha_j)-1}\,(1-\sum_{j=1}^{K-1}x_j)^{\alpha_K-1}\prod_{i=1}^{K-1}x_i^{\alpha_i-1}\prod_{j=1}^{K}\frac{1}{\Gamma(\alpha_j)}\ .$$

Now the expression for the left side of equation (4) is found, consequently the right hand side is considered. Each of the variables z are gamma distributed such that

$$f_Z(z_i) = \frac{1}{\Gamma(\alpha_i)}z_i^{\alpha_i-1}e^{-z_i}$$

is the pdf for $z_i > 0$. Expressing

$$z_i = \begin{cases} v\ x_i & i = 1,2,\ldots,K-1 \\ v(1-\sum_{i=1}^{K-1}x_i) & i = K\ , \end{cases}$$

the pdf of each z is

$$f_Z(z_i) = \begin{cases} \frac{1}{\Gamma(\alpha_i)}(vx_i)^{\alpha_i-1}e^{-vx_i} & i = 1,2,\ldots,K-1 \\ \frac{1}{\Gamma(\alpha_K)}(v(1-\sum_{i=1}^{K-1}x_i))^{\alpha_K-1}e^{-(v(1-\sum_{i=1}^{K-1}x_i))} & i = K\ , \end{cases}$$

and which can be written

$$f_Z(z_i) = \begin{cases} \frac{v^{\alpha_i-1}}{\Gamma(\alpha_i)}x_i^{\alpha_i-1}e^{-vx_i} & i = 1,2,\ldots,K-1 \\ \frac{v^{\alpha_K-1}}{\Gamma(\alpha_K)}(1-\sum_{i=1}^{K-1}x_i)^{\alpha_K-1}e^{-(v(1-\sum_{i=1}^{K-1}x_i))} & i = K\ , \end{cases}$$

such that the joint pdf is

$$f_G(g^{-1}(x_1,\ldots,x_{K-1},v)) = \prod_{j=1}^{K} \frac{v^{\alpha_j-1}}{\Gamma(\alpha_j)} (1-\sum_{i=1}^{K-1} x_i)^{\alpha_K-1} \prod_{i=1}^{K-1} x_i^{\alpha_i-1} e^{-v\sum_{l=1}^{l=K} x_l} .$$

Noticing that $\sum_{l=1}^{l=K} x_l = 1$ per definition of the Dirichlet distribution this can be rewritten in the form

$$f_G(x_1,\ldots,x_{k-1},v) = e^{-v}\, v^{(\sum_{j=1}^{K}\alpha_j)-K} (1-\sum_{j=1}^{K-1} x_j)^{\alpha_K-1} \prod_{i=1}^{K-1} x_i^{\alpha_i-1} \prod_{j=1}^{K} \frac{1}{\Gamma(\alpha_j)} .$$

Now turning to the Jacobian in expression (4)

$$J = \begin{vmatrix} \frac{\partial z_1}{\partial x_1} & \cdots & \frac{\partial z_K}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_1}{\partial x_{K-1}} & & \\ \frac{\partial z_1}{\partial v} & \cdots & \frac{\partial z_K}{\partial v} \end{vmatrix} = \begin{vmatrix} v & & & -v \\ & \ddots & & \vdots \\ & & v & -v \\ x_1 & \cdots & x_{K-1} & 1-\sum_{i=1}^{K-1} x_i \end{vmatrix},$$

such that

$$J = v^{K-1}.$$

The condition in equation (4) is satisfied as (**??**) and the jacobian above multiplied equals the pdf in equation (**??**). Thus the vector with entries $x_i = z_i v^{-1}$ is Diriclet distributed and the proof is concluded.

D.2 (b) Generate sample from Dirichlet distribution

Although the computation of the proof is somewhat cumbersome the algorithm is quite simple. This takes in a vector of the α_is, draws one value from each of the gamma distributions, and return a vector containing of the corresponding dirichlet sample.

```
dirichletgenerator = function(alphas){
  z = rep(NaN,length(alphas))
  for (i in 1:length(alphas)){
    z[i]=gammagenerator(alpha=alphas[i])
  }
  return(z/sum(z))
}
```