ProblemB

Florian Beiser, Martin Lie

12.02.2021

Problem A: Stochastic Simulation by the Probability Integral Transform and Bivariate Techniques

1

For the exponential distribution $Exp(\lambda)$ with density $f(x) = \lambda \exp(-\lambda x)$, where λ is the so-called rate parameter, we use the inverse cumulative distribution function to sample from f. The cummulative is $F(x) = 1 - \exp(-\lambda x)$ and its inverse $F^{-1}(u) = -\frac{1}{\lambda}\log(u)$.

```
myExp <- function(lambda, n){
  # This function generates n indep samples
  # distributed according Exp(lambda)
  u = runif(n)
  samples = -1/lambda * log(u)
  return(samples)
}

# Testing
lambda = 1.0
sample_size = 10000
samples = myExp(lambda, sample_size)
hist(samples, breaks = 50, freq = F,
     xlab="x", main="Histogram for Exp(lambda) samples")
curve(dexp(x, rate=lambda), add=TRUE)
```

Figure 1 shows the empirical and theoretical distribution for our implementation of the exponential distribution with rate parameter $\lambda = 1.0$.

2

(a)

The cummulative distribution function for g is $G(x) = \begin{cases} 0 & x \leq 0 \\ \frac{c}{\alpha}x^{\alpha} & 0 < x < 1 \\ \frac{c}{\alpha} + c\exp(-1) - c\exp(x) & 1 \leq x \end{cases}$. Its inverse is given as $G^{-1}(u) = \begin{cases} (\frac{\alpha}{c}u)^{\frac{1}{\alpha}} & u \leq \frac{c}{\alpha} \\ 1 - \log((-y + c\exp(-1)/c) & u > \frac{c}{\alpha} \end{cases}$. This g is unnormalised we cannot sample u from $(0, 1)$ but have to adapt the interval. Note that $\int g dx = c/alpha + c * exp(-1)$ which is the normalising constant for plotting.

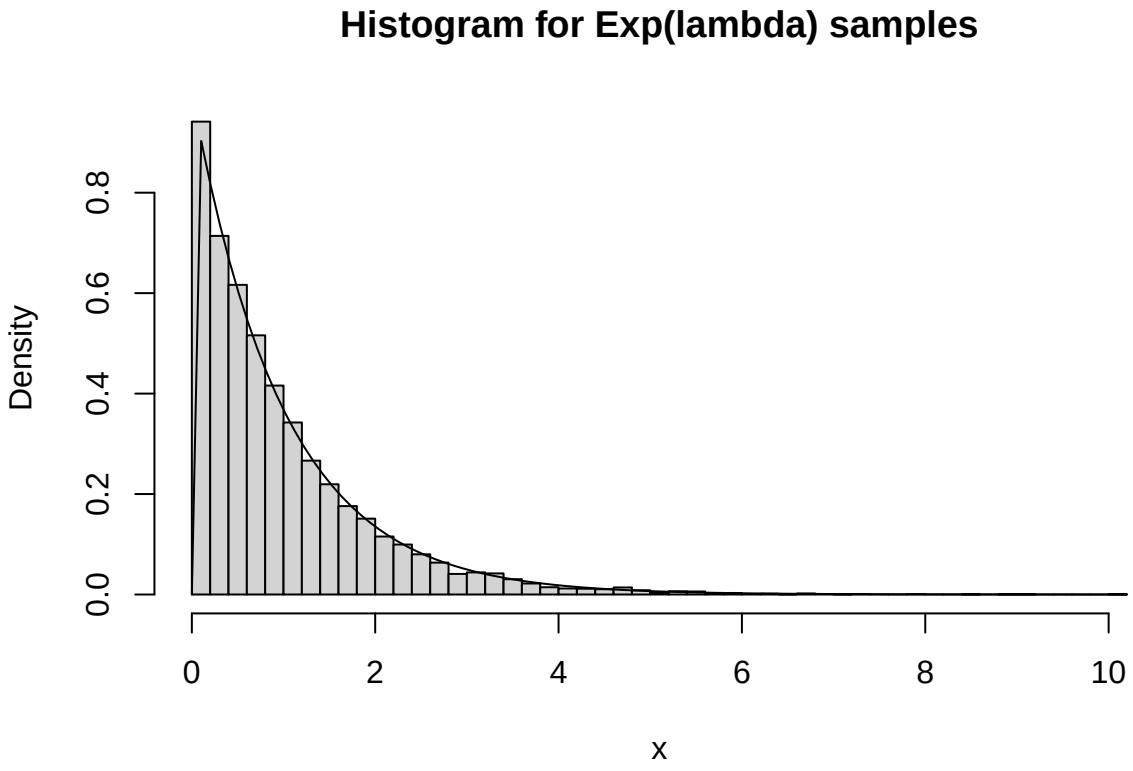
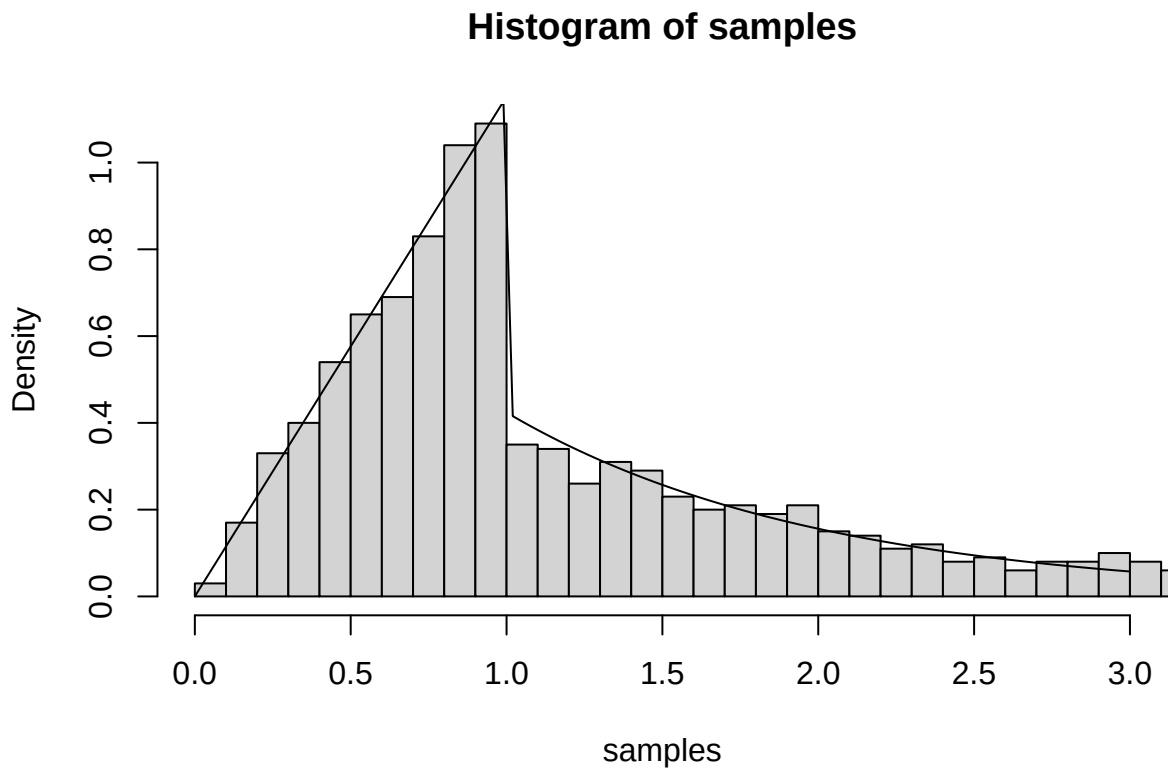


Figure 1: Histograms of Exp(lambda) distributed samples by inversion sampling compared to the theoretical density

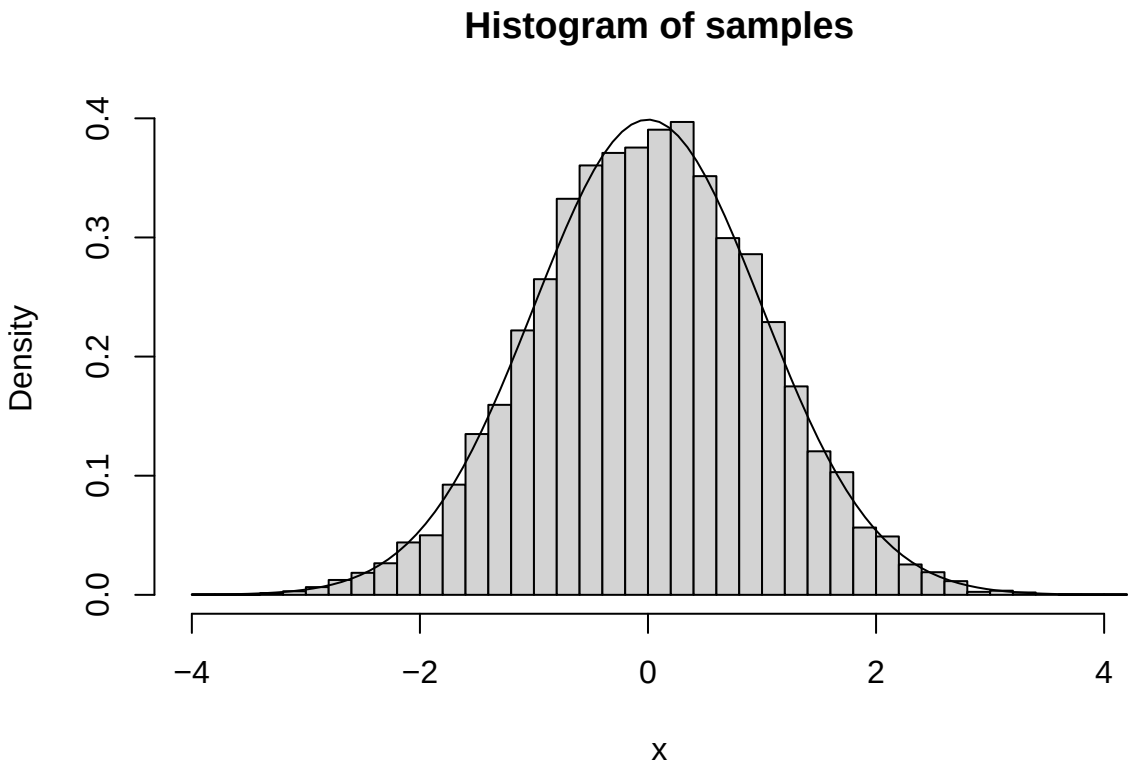
(b)

```
g <- function(c, alpha, x){
  # This function return the unnormalized density g
  g = ifelse( x<0, 0, ifelse(x<1, c*x^(alpha-1), c*exp(-x)) )
  return(g)
}

gSamples <- function(c, alpha, n){
  # This function returns samples from g
  u = runif(n, min=0, max=(c/alpha+c*exp(-1)))
  G = ifelse( u<c/alpha, (alpha/c*u)^(1/alpha), -log((c*exp(-1)+ c/alpha - u)/c) )
  return(G)
}

# Testing
c = 2.0
alpha = 2.0
sample_size = 1000

samples = gSamples(c, alpha, sample_size)
hist(samples, breaks=50, freq=F, xlim=c(0,3))
curve(g(c,alpha,x)/(c/alpha+c*exp(-1)), add=T, to=3)
```



4

```
BoxMuller <- function(n){
  # This function uses the Box Muller algorithm
  # to generate n N(0,1) samples

  m = ceiling(n/2)
  x1 = runif(m, min=0, max=2*pi)
  x2 = myExp(0.5, m)

  y1 = sqrt(x2)*cos(x1)
  y2 = sqrt(x2)*sin(x1)

  if ((n%%2) == 0){
    return(c(y1,y2))
  }
  else
    return(c(y1,head(y2,-1)))
}

# Testing
sample_size = 10001

samples = BoxMuller(sample_size)
hist(samples, breaks=50, freq=F, xlab="x")
curve(dnorm(x), add=T)
```



5

```
myMultivariateNormal <- function(d, mu, Sigma, n){
  # This function generates n samples
  # that are N_d(mu, Sigma) distributed

  # Sanity Check of Input
  stopifnot( d==length(mu) && d==dim(Sigma)[1] && d==dim(Sigma)[2] )

  # N(0,1) samples to N_d(0,1) samples
  samples = matrix(BoxMuller(d*n), nrow=d)

  # N_d(0,Sigma)
  samples = t(chol(Sigma)) %*% samples

  # N_d(mu,Sigma)
  samples = samples + mu

  return(samples)
}

# Testing
d = 3
mu = c(1,2,3)
Sigma = matrix(c(1,.8,.8,.8,1,.8,.8,.8,1), ncol=3)
sample_size = 10000

samples = myMultivariateNormal(d, mu, Sigma, sample_size)

# Validation (mean)
sample_mean = 1/sample_size * rowSums(samples)

print_mu = ""
for (i in 1:d)
  print_mu = paste(print_mu, mu[i])
print(paste("The analytical mean is:", print_mu))
```

```
## [1] "The analytical mean is:  1 2 3"
```

```
print_sample_mean = ""
for (i in 1:d)
  print_sample_mean = paste(print_sample_mean, sample_mean[i])
print(paste("The empirical mean is: ", print_sample_mean))
```

```
## [1] "The empirical mean is:   1.01472738511024 2.01918740499623 3.0100319270969"
```

```
# Validation (cov)
sample_cov = 1/(sample_size-1)*(samples-sample_mean)%*%t(samples-sample_mean)

print("The analytical covariance is:")
```

```
## [1] "The analytical covariance is:"
```

```
for (i in 1:d){
  print_cov=""
  for (j in 1:d)
```

```
    print_cov = paste(print_cov, Sigma[i,j])
  print(print_cov)
}
```

```
## [1] " 1 0.8 0.8"
## [1] " 0.8 1 0.8"
## [1] " 0.8 0.8 1"
```

```
print("The empirical covariance is:")
```

```
## [1] "The empirical covariance is:"
```

```
for (i in 1:d){
  print_cov=""
  for (j in 1:d)
    print_cov = paste(print_cov, sample_cov[i,j])
  print(print_cov)
}
```

```
## [1] " 0.986989210592696 0.793660340474732 0.798811063903769"
## [1] " 0.793660340474732 0.994999113305145 0.801578270829484"
## [1] " 0.798811063903769 0.801578270829484 1.01190391210547"
```

We see a great accordance of empirical and analytical quantities!

Problem B: The gamma distribution

1

We consider the gamma distribution with parameters $\alpha \in (0,1)$ and $\beta = 1$, i.e. $f(x) = \begin{cases} \frac{1}{\Gamma(\alpha)} x^{\alpha-1} \exp(-x) & x > 0 \\ 0 & \text{else} \end{cases}$.

(a)

To ensure $f(x) \leq Cg(x)$, what is required for rejection sampling, we choose the normalizing constant $c = 1$ in the definition of g and $C = \Gamma(\alpha)$. (Then $\alpha - 1 < 0$ yields $x^{\alpha-1} < 1$ and $\exp(-x) < 1$ anyways for positive x, such that $f(x) \leq Cg(x)$.)

(b)

```
f <- function(alpha, x){
  # This function evaluates the density of the gamma distribution
  # with parameters alpha in (0,1) and beta = 1 at the given x

  ifelse( x > 0, 1/gamma(alpha)*x^(alpha-1)*exp(-x), 0 )

}

# Visualising density for fix alpha
alpha = 0.5
x = seq(-1,5, length =101)
y = f(alpha, x)

plot(x,y, type="l", xlab="x", ylab="p(x)",
     main=paste("Density of Gamma distribution with alpha = ", toString(alpha), " and beta = 1"))
```

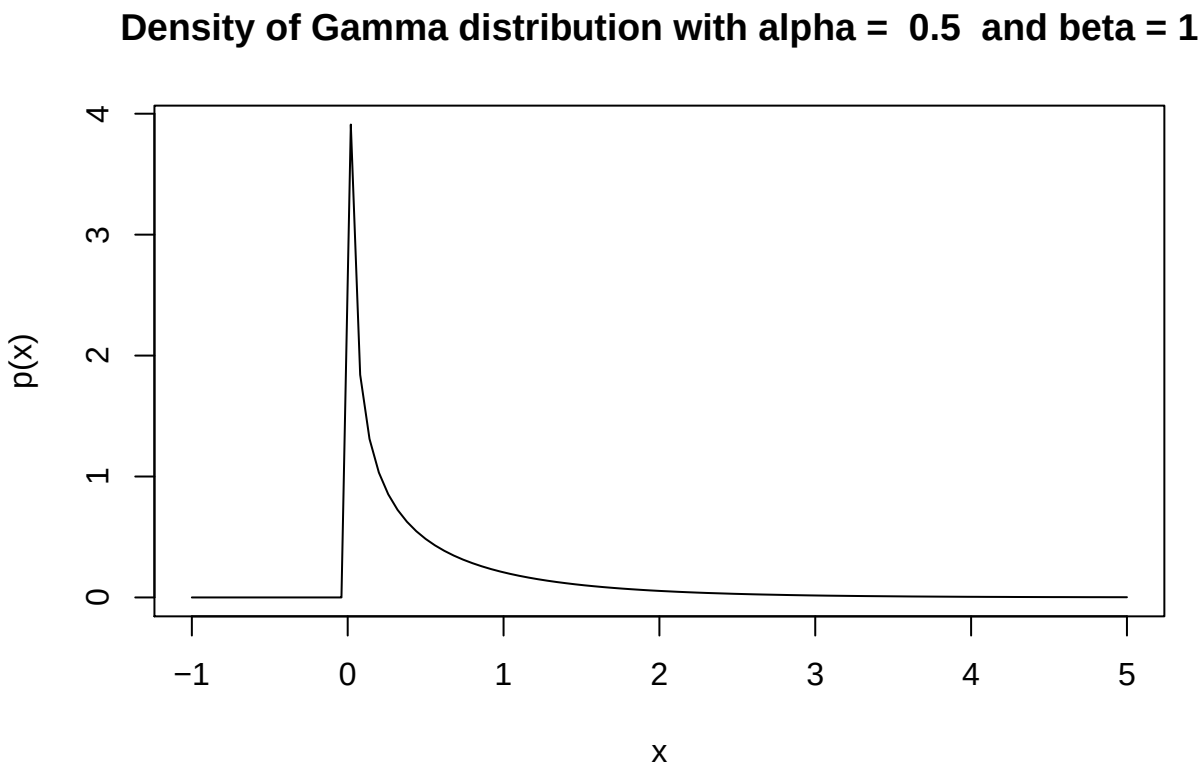
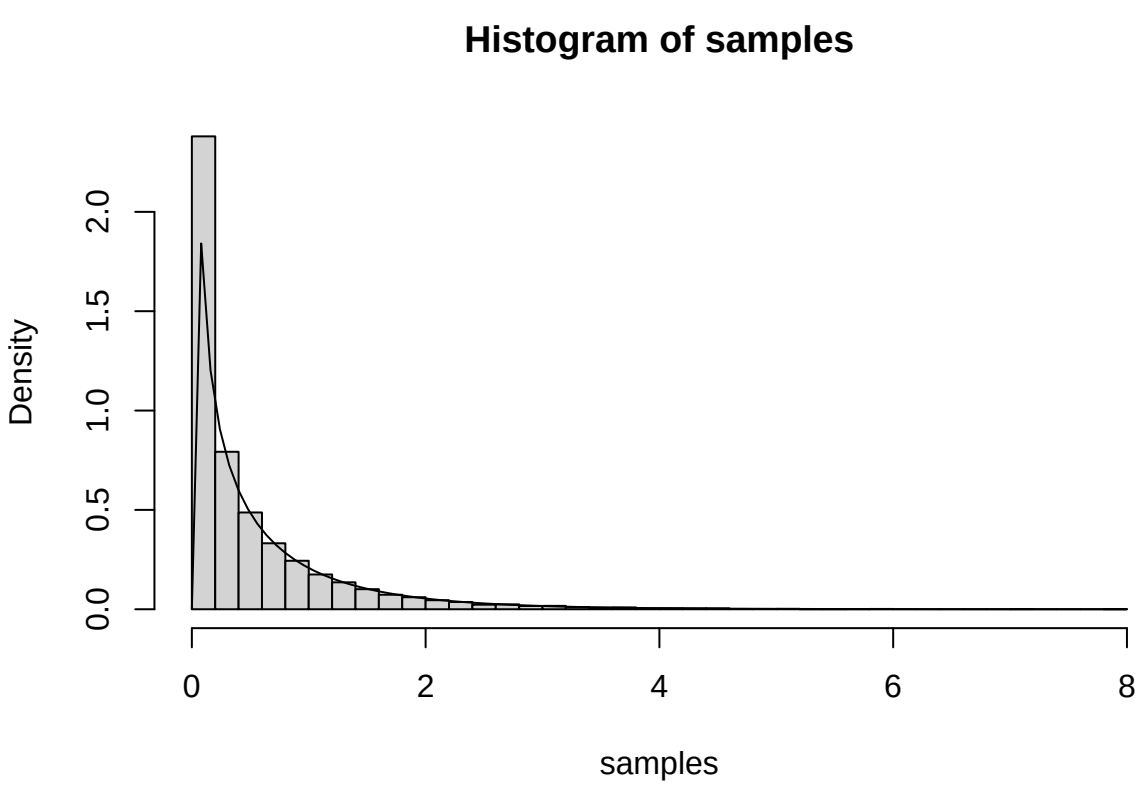


Figure 2: Plotting the density of the Gamma distribution for a fix alpha

Figure 2 show the target density f for $\alpha = 0.5$. The proposal density g is defined in the previous problem solution and not visualised here.

```
myGammaRejectionSampling <- function(alpha, n){
  # This function generates n samples from the gamma distribution
  # with parameter alpha in (0,1) and beta = 1
  # using rejection sampling the a proposal given in Problem A.

  # Check sanity of input
  stopifnot( alpha > 0 && alpha < 1 )

  # Bookkeeping
  num_accepted_samples = 0
  target = rep(0,n)

  C = gamma(alpha)
  # Trial implementation in for-loop
  while (num_accepted_samples<n){
    proposal = gSamples(1, alpha, 1)
    acceptance_rate = 1/C*f(alpha, proposal)/g(1, alpha, proposal)
    acception = runif(1)
    if (acception <= acceptance_rate){
      num_accepted_samples = num_accepted_samples + 1
      target[num_accepted_samples] = proposal
    }
  }
  return(target)
}

# Testing
alpha = 0.5
sample_size = 10000

samples = myGammaRejectionSampling(alpha, sample_size)
hist(samples, breaks=50, freq=F)
curve(f(alpha,x), add=T)
```



2

We now consider aforementioned Gamma distribution with $\alpha > 1$ and still $\beta = 1$. The domain C_f is defined as in the lecture.

Directly, we transform the density on log-scale: $\log f^*(x) = (\alpha - 1)log(x) - x$.

(a) The domain C_f is contained in the rectangle $[0, a] \times [b_-, b_+] = [0, \alpha - 1] \times [0, \alpha]$, where $a = \frac{1}{2} \sup \log f^*(x)$ and $b_\pm = \frac{1}{2} \sup_{x \leq \geq 0} \{2 \log x + \log f^*(x)\}$. Note that $f^*(x) = 0$ for $x < 0$ and hence $b_- = 0$ trivially. We obtain the values for a and b_- by 1) differentiation of the supremum argument $\frac{d}{dx} \log f^*(x) = \frac{\alpha - 1}{x} - 1$ and setting to 0. Since asymptotic analysis for continuous functions yields that the candidate is a unique maximum, we set $a = \frac{1}{2}(\alpha - 1)$. 2) differentiation of the supremum argument $\frac{d}{dx} 2 \log x + \log f^*(x) = \frac{\alpha + 1}{x} - 1$ and setting to 0. Since asymptotic analysis for continuous functions yields that the candidate is a unique maximum, we set $b_+ = \frac{1}{2}(\alpha + 1)$.

(b)

We start with the definition of the log-scaled core of f and the ratio-of-uniform sampling to generate n independent samples from the Gamma distribution.

```
fCoreLogScale <- function(alpha, x){
  # This function evaluates the density of the gamma distribution
  # with parameters alpha in (0,1) and beta = 1 at the given x

  # Transformation y = log(x+1)
  # Returning f(y)
```

```
  ifelse( x > 0, log(x)*(alpha-1)-x, 0 )

}

myGammaRatioOfUniformsLogScale <- function(alpha, n){
  # This function generates n samples from the gamma distribution
  # with parameter alpha in (0,1) and beta = 1
  # using the ratio of uniforms method

  # Check sanity of input
  stopifnot( alpha > 1 )

  # Rectangular domain for uniform sampling
  # (For stability size slightly increased
  # but no influence on distribution since rejection mechanism)
  x1_lower = 0
  x1_upper = 1/2*(alpha - 1) +0.1
  x2_lower = 0
  x2_upper = 1/2*(alpha + 1) +0.1

  # Bookkeeping
  num_accepted_samples = 0
  num_total_samples = 0

  # Generation of n samples with ratio of uniforms method
  target = rep(0,n)
  while (num_accepted_samples < n ){
    x1 = runif(1, x1_lower, x1_upper)
    x2 = runif(1, x2_lower, x2_upper)
    if (log(x1) <= 1/2*fCoreLogScale(alpha,x2/x1)){
      num_accepted_samples = num_accepted_samples + 1
      target[num_accepted_samples] = log(x2)-log(x1)
    }
    num_total_samples = num_total_samples + 1
  }
  return(list("num_total_samples" = num_total_samples, "samples" = target))
}

# Visualisation of samples for alpha = 2
alpha = 2.0
sample_size=10000

samples = myGammaRatioOfUniformsLogScale(alpha,sample_size)
hist(exp(samples$samples), breaks=50, freq=FALSE,
     xlab="x", main="Histogram for the Gamma distributed samples")
curve(f(alpha,x), from=0.0, to=10.0, add=TRUE)
```

In Figure 3 we see for $\alpha = 2$ that the ratio-of-uniform method approximates the target density pretty well. Bigger sample sizes would improve the approximation.

We continue with the actual analysis of the efficiency of the ratio-of-uniform method.

```
# Plotting the efficiency of the myGammaRatioOfUniformLogScale-function
# in dependence of alpha
```

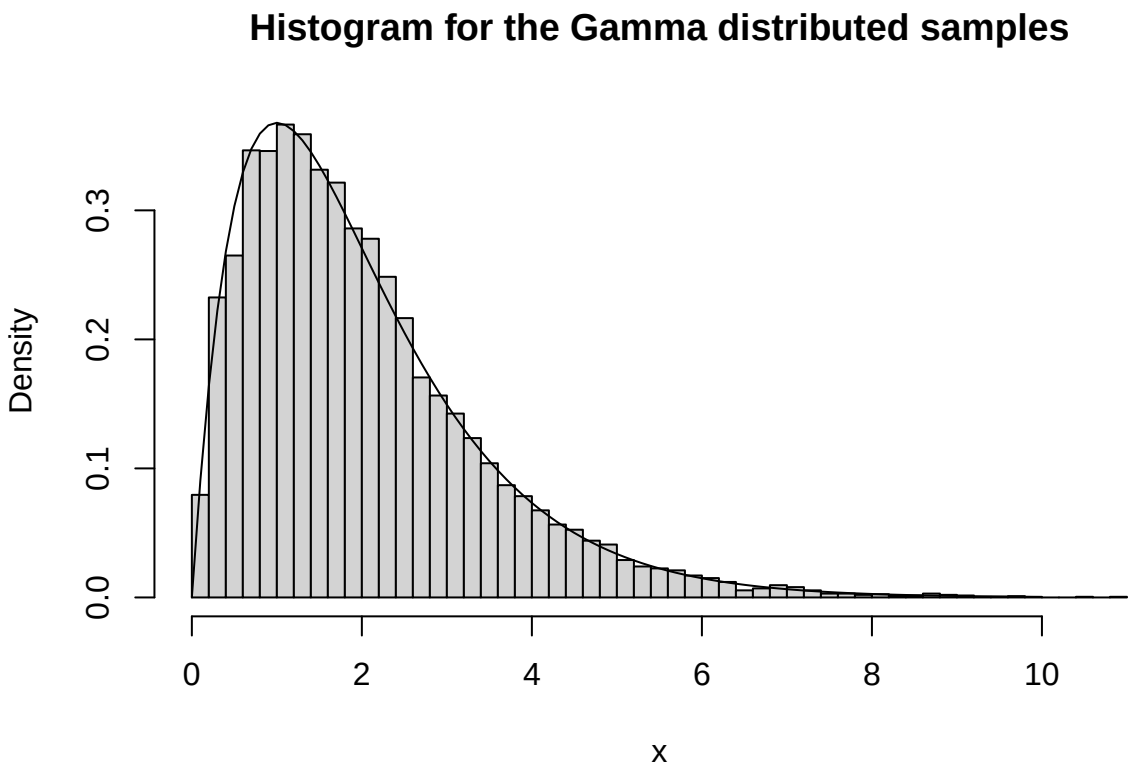


Figure 3: Histogram for the myGammaRatioOfUniformLogScale-function

```
alphas = c(1.01,1.1,1.25,1.5,2.0,3.0,5.0,10.0,20.0,50.0,100.0,200,300,400,500,600,700,800,900,1000,1100
efficiency = rep(0, length(alphas))

for (i in 1:length(alphas)){
  samples = myGammaRatioOfUniformsLogScale(alphas[i],1000)
  efficiency[i] = samples$num_total_samples
}

plot(alphas, efficiency, log="x", type="l",
     xlab="alpha", ylab="# total samples",
     main="Total number of used samples w.r.t. alpha")
```

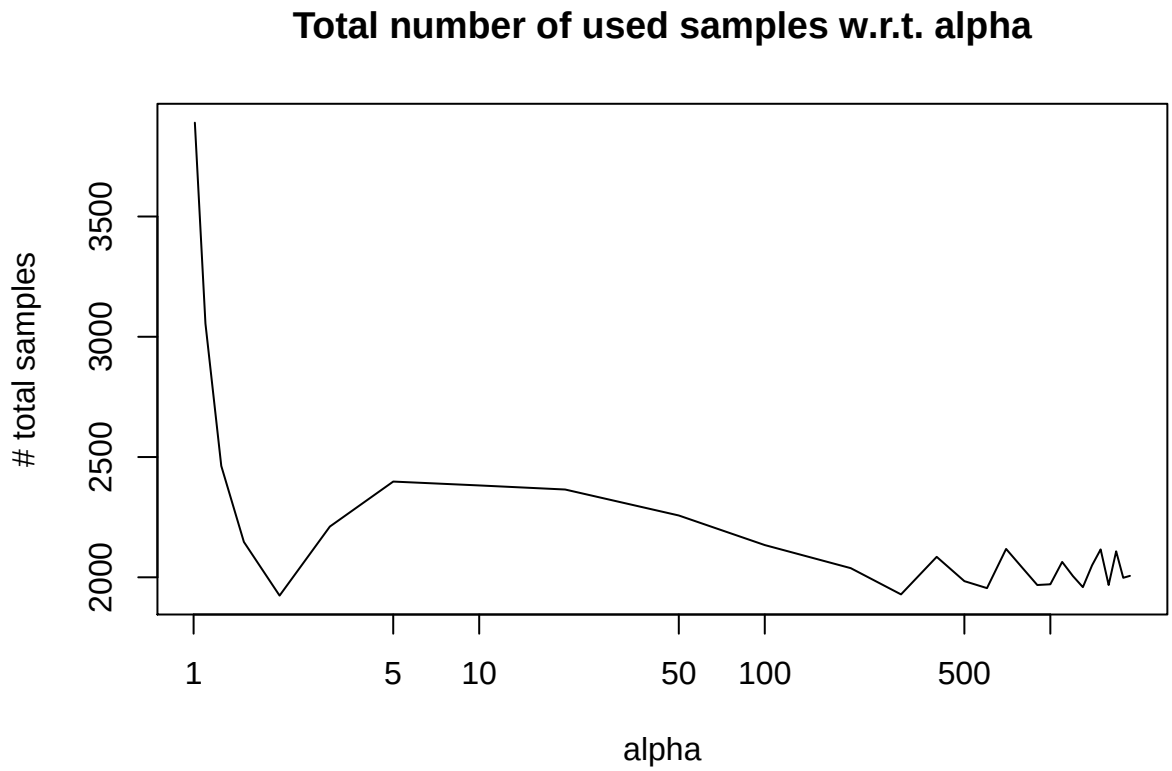


Figure 4: Number of total samples used to generate 1,000 Gamma(alpha,1)-distributed samples

Figure 3 shows the number of total samples which are used to generate 1,000 independent samples from the $G(\alpha, 1)$ distribution. Due to rejection sampling to generate uniform samples in C_f more samples are generated then accepted. From this figure we conclude that for $alpha \to 1$ the algorithm gets very inefficient where as it converges to a stable effectivity for $alpha > 5$

3

We now consider aforementioned Gamma distribution with generally $\alpha > 0$ and still $\beta = 1$.

(a)

Let $X_1 \sim G(\alpha_1, 1)$ and $X_2 \sim G(\alpha_2, 1)$ independent random variables.

Remember, that the moment generating function $M_X(t)$ for a $G(\alpha, 1)$ distributed random variable looks as $(1-t)^{-\alpha}$ and that a moment generating function uniquely determines the law of a random variable.

Hence, we consider the moment generating function of above's sum $M_{X_1+X_2}(t)$. Since both random variables are independent we can expand $M_{X_1+X_2}(t) = M_{X_1}(t)M_{X_2}(t) = (1-t)^{-\alpha_1}(1-t)^{-\alpha_2} = (1-t)^{-(\alpha_1+\alpha_2)}$. This is the moment generating function of $G(\alpha_1+\alpha_2, 1)$ which yields that $X_1 + X_2 \sim G(\alpha_1+\alpha_2, 1)$.

(b)

The gamma distribution $G(1,1)$ is equivalent to the Exponential distribution $Exp(1)$. To facilate above's result for $\alpha \in (1,2)$ we construct $X = Exp(1) + G(\alpha-1, 1) \sim G(\alpha, 1)$ where the first terms are generated inversion and rejection sampling instead of using the ratio-of-uniform method.

4

We now consider aforementioned Gamma distribution with generally $\alpha > 0$ and also generally $\beta > 0$.

For the rate we have the property if $X \sim Gamma(\alpha, 1)$ then $\frac{1}{\beta}X \sim Gamma(\alpha, \beta)$.

```
myGammaDensity <- function(alpha, beta, x){
  # This function calculate the density of a Gamma(alpha,beta) distr.
  # at given x

  f = beta^alpha/gamma(alpha) * x^(alpha-1) *exp(-beta*x)

  return(f)
}

myGammaGeneral <- function(alpha, beta, n){
  # This function uses all functions from above
  # to sample from a Gamma(alpha,beta) distribution

  # Sanity check of input
  stopifnot( alpha > 0 && beta > 0)

  # Three cases:
  # 1. alpha < 1 - myGammaRejectionSampling
  # 2. alpha = 1 - myExponentialInversionSampling
  # 3. alpha > 1 - myGammaRatioOfUniformsLogScale
  if (alpha < 1){
    samples = myGammaRejectionSampling(alpha, n)
  } else if (alpha == 1){
    samples = myExp(1.0, n)
  } else if (alpha > 1){
    samples = myGammaRatioOfUniformsLogScale(alpha, n)
    samples = exp(samples$samples)
  }

  # Scaling with beta
  samples = 1/beta*samples

  return(samples)
}

# Testing of myGammaGeneral
alpha = 1.0
```

```r
beta = 0.5
n = 1000

samples = myGammaGeneral(alpha, beta, n)
hist(samples, breaks=50, freq=FALSE,
     xlab="x", main="Histogram for myGammaGeneral")
curve(myGammaDensity(alpha,beta,x), from=0.0, to=15.0, add=TRUE)
```

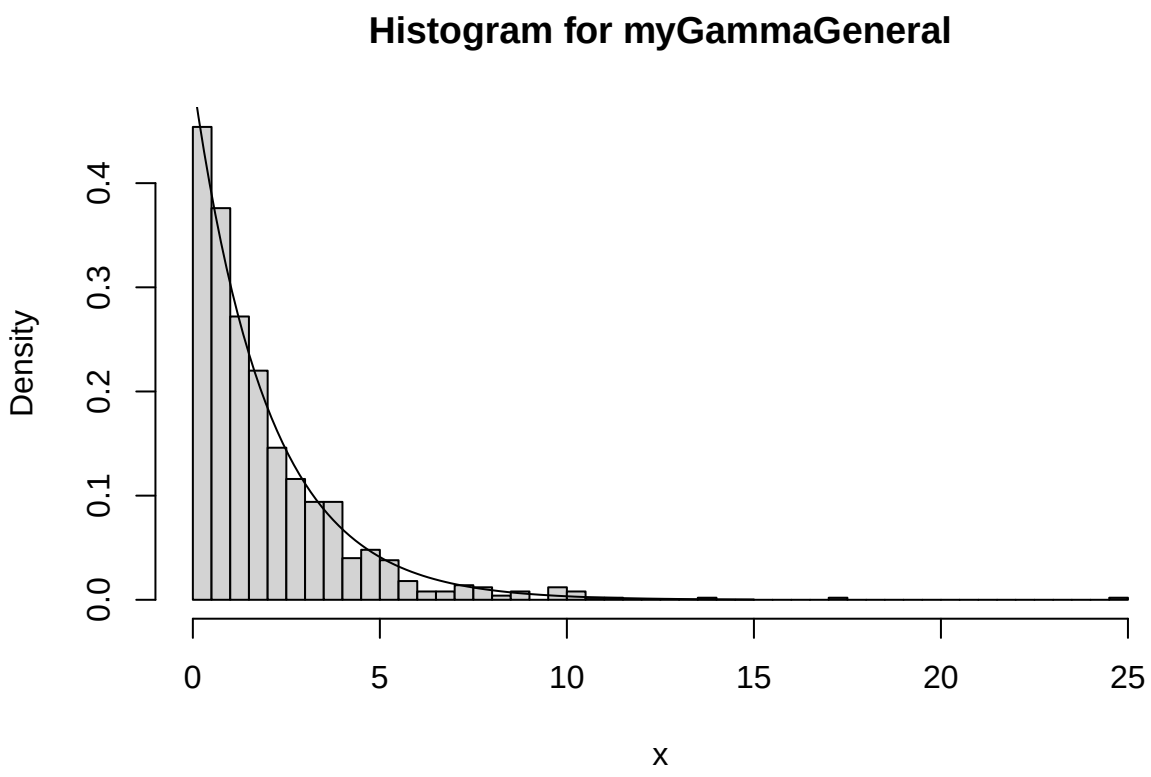


Figure 5: Histogram for the myGammaGeneral function

Figure 5 shows that the scaling for β works as expected and we can use the sampling routines from before to sample from gernal Gamma distributions.

5

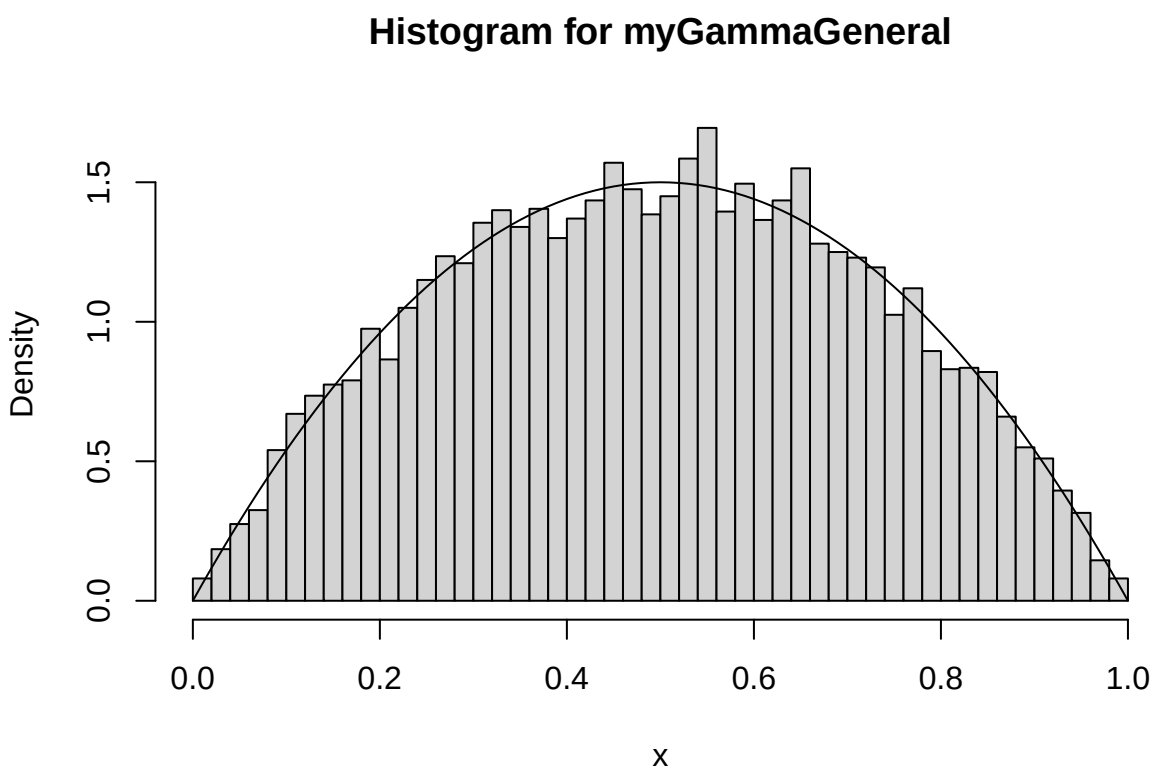
We consider $X \sim G(\alpha_1, 1)$ and $Y \sim G(\alpha_2, 1)$.

(a)

The joint probability density function of (X, Y) is $f_{(X,Y)}(x, y) = \frac{1}{\Gamma(\alpha_1)\Gamma(\alpha_2)} x^{\alpha_1 - 1} y^{\alpha_2 - 1} \exp(-(x + y))$ using a change of variables $u = \frac{x}{x+y}$ and $v = x + y$ we bring the terms of our interest into the equation as $f_{(U,V)}(u, v) = \frac{\Gamma(\alpha_1+\alpha_2)}{\Gamma(\alpha_1)\Gamma(\alpha_2)} u^{\alpha_1 - 1}(1 - u)^{\alpha_2 - 1} \cdot \frac{1}{\Gamma(\alpha_1+\alpha_2)} v^{\alpha_1+\alpha_2 - 1} \exp(-v)$, which clearly factorizes in distinct terms for u and v such that U and V are independent. This yields that $U = \frac{X}{X+Y} \sim Beta(\alpha_1, \alpha_2)$ by the form of the density. (We again see the result from B3(a).)

(b)

```
myBeta <- function(alpha, beta, n){
  # This function generates n samples from a Beta(alpha, beta) distribtion
  x = myGammaGeneral(alpha, 1, n)
  y = myGammaGeneral(beta , 1, n)
  z = x/(x+y)
  return(z)
}


# Testing implementation
alpha = 2.0
beta = 2.0
n = 10000

samples = myBeta(alpha, beta, n)
hist(samples, breaks=50, freq=FALSE,
     xlab="x", main="Histogram for myGammaGeneral")
curve(dbeta(x,alpha,beta), from=0.0, to=1.0, add=TRUE)
```



Problem D

We consider the experiment of Rao (1973) from genetics. It assumes a multinomial distribution with parameter θ for 4 distinct categories. The prior for θ is just a uniform distribution over $(0, 1)$. Further, 197 observations

are made with category counts for each of the 4 categories.H Herewith, the posterior looks as

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

where we are interested in the posterior mean.

For a Monte-Carlo approach, this requires that we can sample from the posterior distribution. In order to be able to do so, we will employ rejection sampling to the aforementioned posterior density in log-scale $\log f(\theta|y) = y_1 \log(2+\theta) + (y_2+y_3)\log(1-\theta) + y_4 \log\theta$.

1.

Using the uniform distribution over $(0,1)$ as proposal with density $g(\theta) = 1$, the constant $c : \log f(\theta|y) \leq c \log g(\theta) = c \cdot 1$ is found by differentiating $\frac{d}{d\theta}\log f(\theta|y)$ and equalizing $0 = -(y_1+y_2+y_3+y_4)\theta^2 + (y_1 - 2y_2 - 2y_3 - y_4)\theta + 2\theta_4 = -197\ \theta^2 + 15\ \theta + 68$ which has a root at 0.626821.... Since the density is very sharp in that area and we propagate numerical errors, we add a small number onto this value to ensure the bounding property.

```
# Cell counts from the experiment
y1 = 125
y2 = 18
y3 = 20
y4 = 34

posteriorLogScale <- function(t){
  # Posterior density of the Rao experiment

  f = y1*log(2+t)+ (y2+y3)*log(1-t) + y4*log(t)
  return(f)
}

# Setting the bound for rejection sampling
argmax_theta = 1/394*(15+sqrt(53809))
c = posteriorLogScale(argmax_theta) + 3.5

posteriorRejectionSamplingLogScale <- function(n){
  # This function generates n samples from the Rao posterior

  # Bookkeeping
  num_accepted_samples = 0
  num_total_samples = 0

  # Trial implementation in for-loop
  target = rep(0,n)
  while (num_accepted_samples < n ){
    x = runif(1)
    acceptance_rate = -c + posteriorLogScale(x) - log(x)
    u = runif(1)
    if (log(u) <= acceptance_rate){
      num_accepted_samples = num_accepted_samples + 1
      target[num_accepted_samples] = log(x)
    }
    num_total_samples = num_total_samples + 1
  }

  return(list("samples" = target, "num_total_samples" = num_total_samples))
```

```
}
```

2.

Monte-Carlo integration uses samples from the underlying distribution to estimates quantities of interest. In our case, we aim for $\mathbb{E}[h(\Theta)]$ where $h = 1$ and $\Theta \sim \mathcal{P}_{f(\theta|y)}$. Therefore, we estimate $E(\theta|y) = \frac{1}{N}\sum \theta_i$.

```
posteriorMonteCarloIntegration <- function(n, samples = NULL){
  # This function estimates the mean of the Rao posterior
  # using Monte Carlo integration with n samples

  if (is.null(samples)){
    samples = exp(posteriorRejectionSamplingLogScale(n)$samples)
  }

  posteriorMCMean = 1/n*sum(samples)

  print(paste("The MC estimate for the posterior mean is", posteriorMCMean))

  return(posteriorMCMean)

}


# Generating Histogram for posterior samples
sample_size = 10000

samples = exp(posteriorRejectionSamplingLogScale(sample_size)$samples)
hist(samples, xlim=c(0,1), breaks=50, freq=FALSE,
     main="Histogram for the posterior generated by rejection sampling",
     xlab="theta")

# Add marker for the mean value
posteriorMCMean = posteriorMonteCarloIntegration(sample_size, samples)
```

```
## [1] "The MC estimate for the posterior mean is 0.618560837253206"
```

```
abline(v=posteriorMCMean,col="red")

posterior <- function(t){
  # This functions defined the posterior density at x
  # in original scale

  return( (2+t)^y1 * (1-t)^(y2+y3) * t^y4)

}

# Add theoretical posterior distribution
curve(8.0/max(posterior(x))*posterior(x), from=0.0, to=1.0, add=TRUE)
```

In Figure 6 we depict the histogram for the posterior distribution generated by rejection sampling with a unifrom proposal. We see that the sampled density has a bit too little weight around the mode compared to the analytical distribution. The empirical mean, which is depicted as red line is located a bit left of the mode.

Finally, we compare the Monte-Carlo estimate to numerical integration. This is the numerical evaluation of

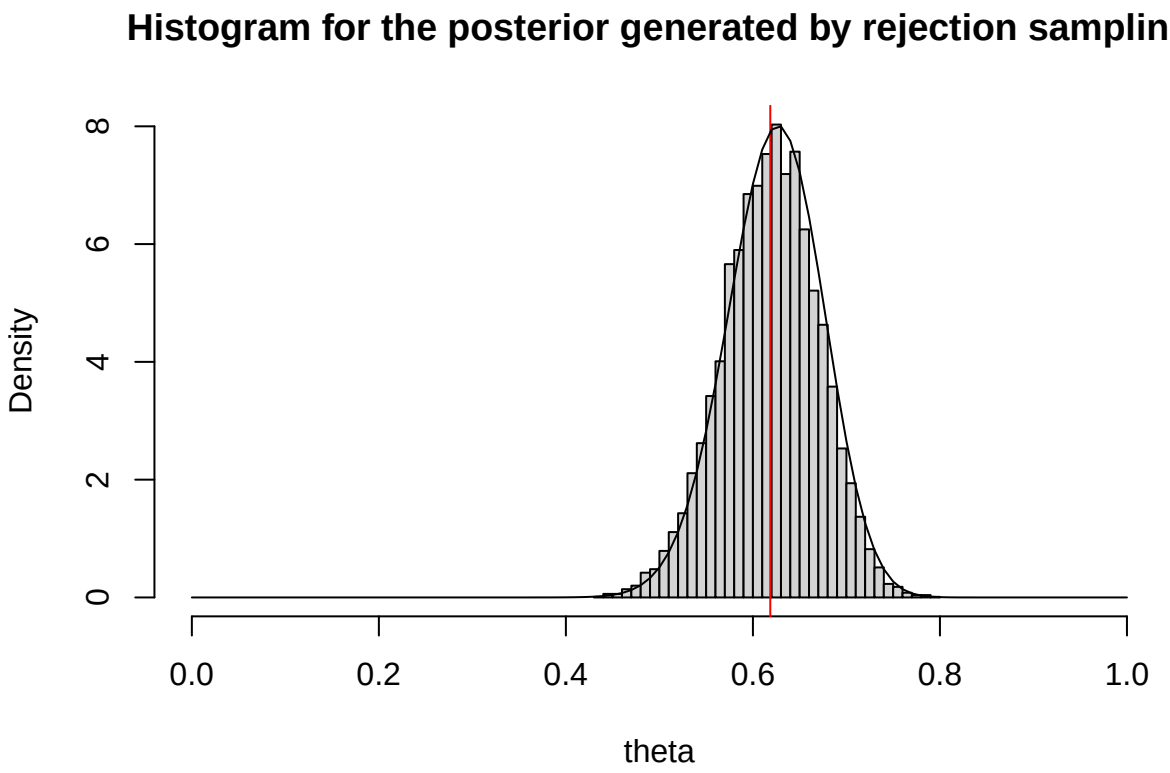


Figure 6: Sampled posterior density by rejection sampling

the integral $\int \theta \, d\mu(\theta) = \int \theta f(\theta|y) \, d\theta$ where μ is the probability measure and $f(\theta|y)$ the (here unnormalised) density - to obtain the probability density we normalise this function.

```
# Comparison to numerical integration
posteriorIntegrand <- function(t){
  # This function defines the integrand for the integral
  # which gives the mean of the posterior
  # t * f(t|y)

  return( t* (2+t)^y1 * (1-t)^(y2+y3) * t^y4)

}
normalisingConstant = integrate(posterior, 0, 1)$value
posteriorNIMean = integrate(posteriorIntegrand, 0, 1)$value
print(paste("The NI estimate for the posterior mean is", posteriorNIMean/normalisingConstant))
```

```
## [1] "The NI estimate for the posterior mean is 0.62280613191073"
```

The numerical integration yields a slighlty higher mean as the MC experiment. If the sample size and if the number of integration points tend to infitiy, those values should converge to each other!

3.

In theory, rejection sampling needs in average $c \cdot N$ total samples to generate N samples of the target distribution. We contrast this now, with some realisations of our practical implementation.

```
# Running m test to get an averaged number of total samples
m = 10

num_total_samples_average = 0
for (i in 1:m){
  num_total_samples_average = num_total_samples_average + 1/m * posteriorRejectionSamplingLogScale(sampl
}

print(paste("For the generation of", sample_size, "samples in average",
            round(c*sample_size,0), "random number generations would be needed in theory"))
```

```
## [1] "For the generation of 10000 samples in average 708841 random number generations would be needed
```

```
print(paste("For the generation of", sample_size, "samples in average",
            round(num_total_samples_average,0), "random number generations were needed in the practical
```

```
## [1] "For the generation of 10000 samples in average 1593654 random number generations were needed in
```

We observe that the practical implementation requires significantly more random number generations than the theory provides. We assume that the very huge value for c in original scale, leads to numerical instabilities and too many rejected random generations. Also around the mode of the distribution we get slightly too little samples where samples in that area would have a high acceptance rate and would decrease the unnecessary random number generations.

4.

For a prior $f(\theta)$, we get the posterior distribution as $f(\theta|y) \propto f(y|\theta)f(\theta)$. Above, we had $f(\theta) = f_{1,1}(\theta)$ and now we want to consider $f(\theta) = f_{1,5}(\theta)$ where $f_{\alpha,\beta}$ denotes the density of a $Beta(\alpha, \beta)$ distribution.

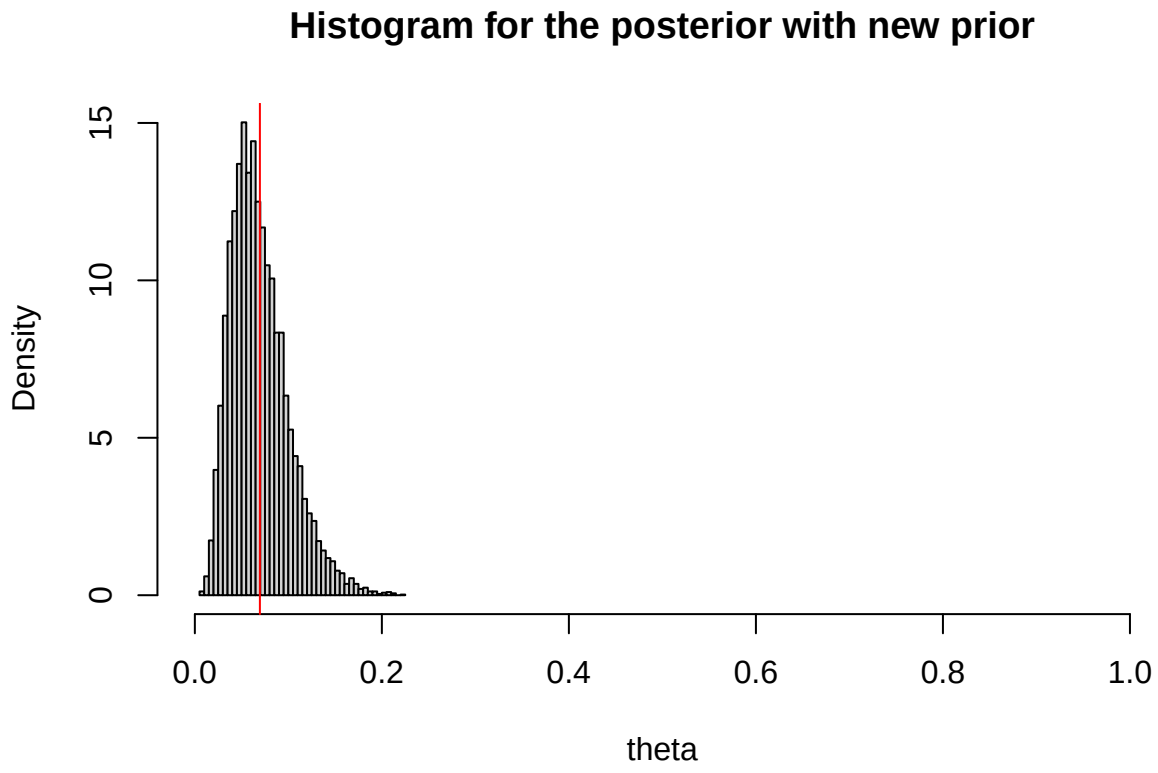
The classical trick of importance sampling $\int \theta f(y|\theta) f_{1,5}(\theta) d\theta = \int \theta f(y|\theta) f_{1,1}(\theta) \frac{f(y|\theta) f_{1,5}(\theta)}{f(y|\theta) f_{1,1}(\theta)} d\theta = \int \theta f(y|\theta) f_{1,1}(\theta) \frac{f_{1,5}(\theta)}{f_{1,1}(\theta)} d\theta$ allows to reuse the samples from before, when we introduce the weights

$w_i = \frac{f_{1,5}(\theta_i)}{f_{1,1}(\theta_i)} = f_{1,5}(\theta_i)$ into our Monte Carlo calculations: $E(\theta|y) = \frac{1}{N}\sum \theta_i w_i$ where the θ_i are distributed according to the posterior given the $f_{1,1}$ prior.

```
# Calculate weights
weights = dbeta(samples,1,5)
# Reweight samples
new_samples = samples*weights

hist(new_samples, breaks=50, xlim=c(0,1), freq=FALSE,
     xlab="theta", main="Histogram for the posterior with new prior")


# IS estimator for the mean
posteriorISMean = 1/sample_size*sum(new_samples)
print(paste("The IS estimate for the posterior mean is", posteriorISMean))
```

```
## [1] "The IS estimate for the posterior mean is 0.0695753909493488"
```

```
abline(v=posteriorISMean,col="red")
```



The $Beta(1,5)$ prior expresses a strong belief that the true θ would be close to 0, Bayes theorem corrects that model assumption into the direction of the observations. However, the posterior results still carries a lot of information from the prior. The uninformed prior leads to an posterior mean that is much closer to the observations.