

CompStat 1

Problem A

1)

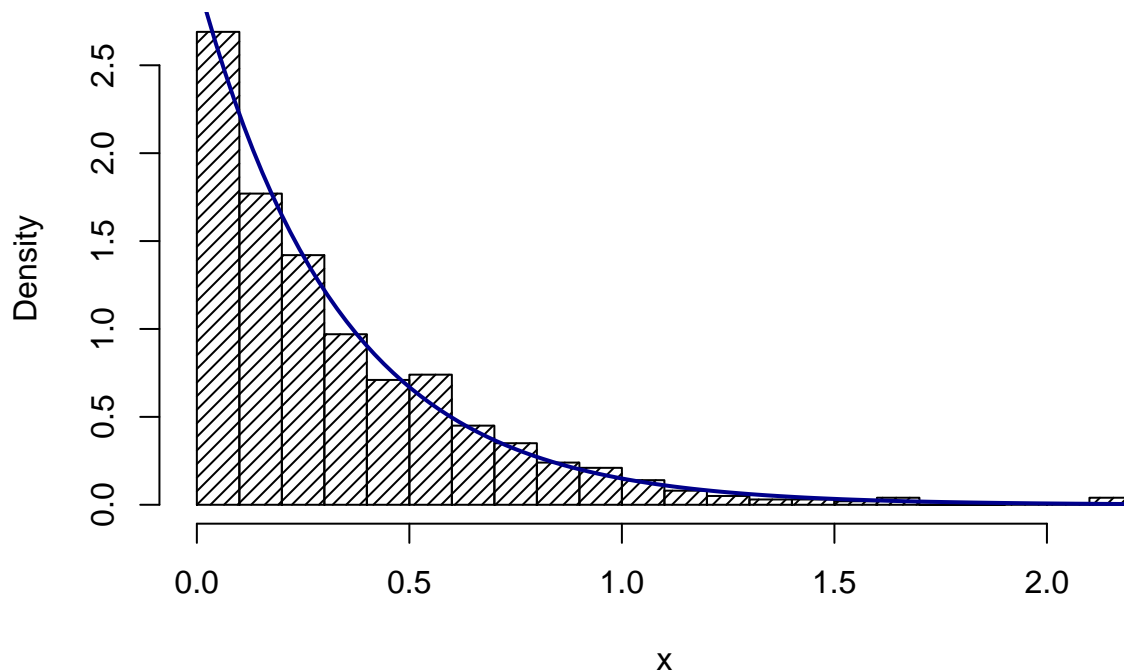
Here using the inversion method, we get:

$$x = F^{-1}(u) = -\frac{1}{\lambda} \log(u), \quad u \sim U(0,1).$$

Thus we get exponential samples, by sampling from the uniform distribution, and use the inversion method. Under there is a function that generates a vector of n samples from an exponentially distributed random variable with rate λ .

```
# Generate samples from exponential distribution
rate <- 3 #Rate parameter
n <- 1000 #Number of samples
generate.exponential <- function(rate, n){
  u <- runif(n) # Uniform distributed random variable
  x <- - 1/rate * log(u) # Inversion method
  return(x)
}
x <- generate.exponential(rate, n)
hist(x, density=20, breaks = 30, prob=TRUE)
curve(dexp(x, rate=rate), add=TRUE, col="darkblue", lwd=2)
```

Histogram of x



2)

The cdf $G(x)$ of $g(x)$ and its inverse G^{-1} are

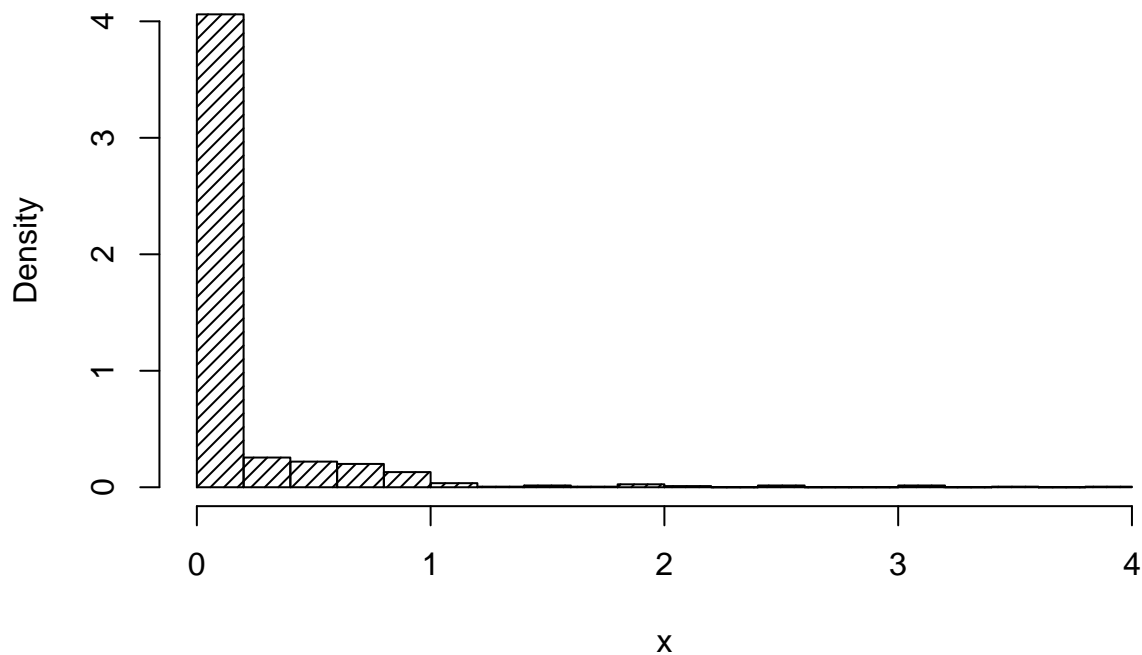
$$G(x) = \begin{cases} \int_0^1 cx^{\alpha-1}dx & , 0 < x \leq 1 \\ \int_1^\infty ce^x dx & , 1 \leq x \\ 0 & , x \leq 0 \end{cases} = \begin{cases} c(\alpha-1)x^\alpha & , 0 < x \leq 1 \\ c(e^{-1} - e^{-x}) & , 1 \leq x \\ 0 & , x \leq 0 \end{cases}$$

$$G^{-1}(u) = \begin{cases} \left(\frac{u}{c(\alpha-1)}\right)^{\frac{1}{\alpha}} & , \text{SJEKK GRENSER } 0 < \left(\frac{u}{c(\alpha-1)}\right)^{\frac{1}{\alpha}} \leq 1 \\ -\log(e^{-1} - \frac{u}{c}) & , 1 \leq -\log(e^{-1} - \frac{u}{c}) \\ 0 & , u \leq 0 \end{cases}$$

Writing a function that returns g -samples.

```
# Generate samples from g(X)
rate <- 3 #Rate parameter
n <- 1000 #Number of samples
alpha <- 0.1 # Alpha parameter
generate.samples.g<- function(alpha, n){
  c <- alpha*exp(1)/(alpha+exp(1)) # Analytical expression for c
  u <- runif(n) # Uniform distributed random variable
  x <- c()
  # For loop as we have to check which expression to apply
  for (i in 1:n){
    if (u[i] < c/(alpha)){
      x <- c(x, (u[i]*(alpha)/c)^(1/alpha))
    }
    else{
      x <- c(x, 1-log(exp(1)/c*(1-u[i])))
    }
  }
  return(x)
}
x <- generate.samples.g(alpha, n)
hist(x, density=20, breaks = 20, prob=TRUE)
```

Histogram of x

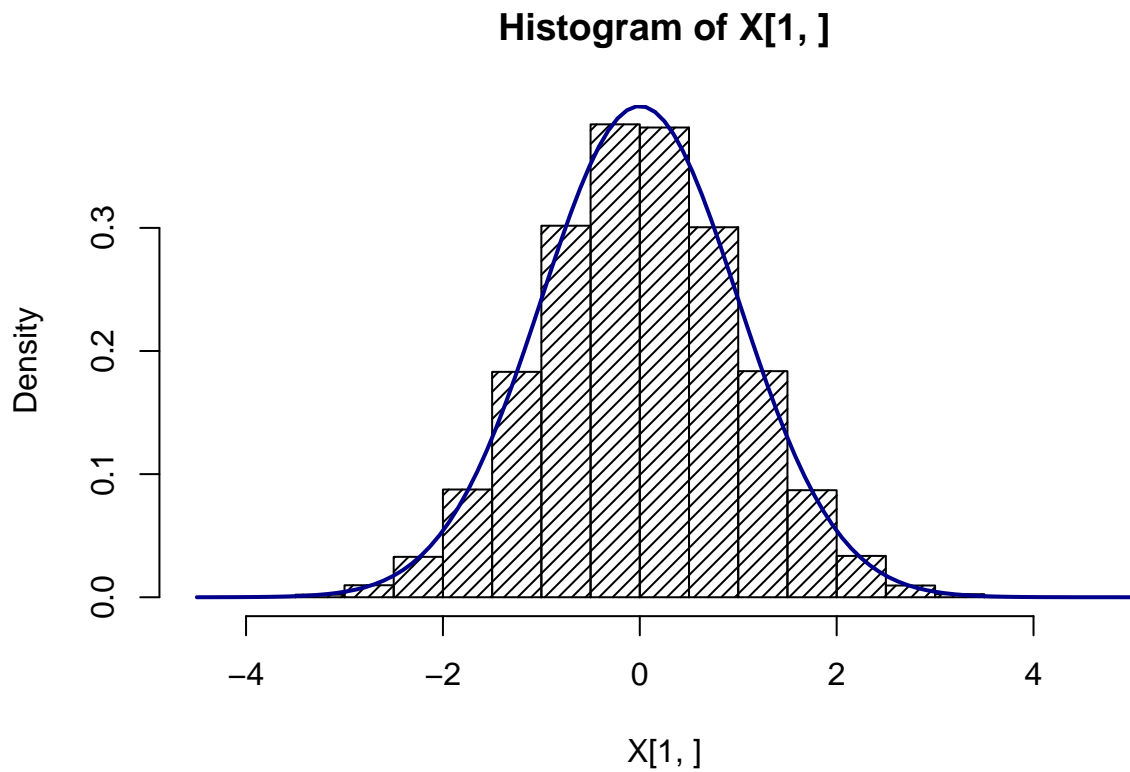


The histogram is used to check that `generate.samples.g` returns the correct value. As we see here, ...

3)

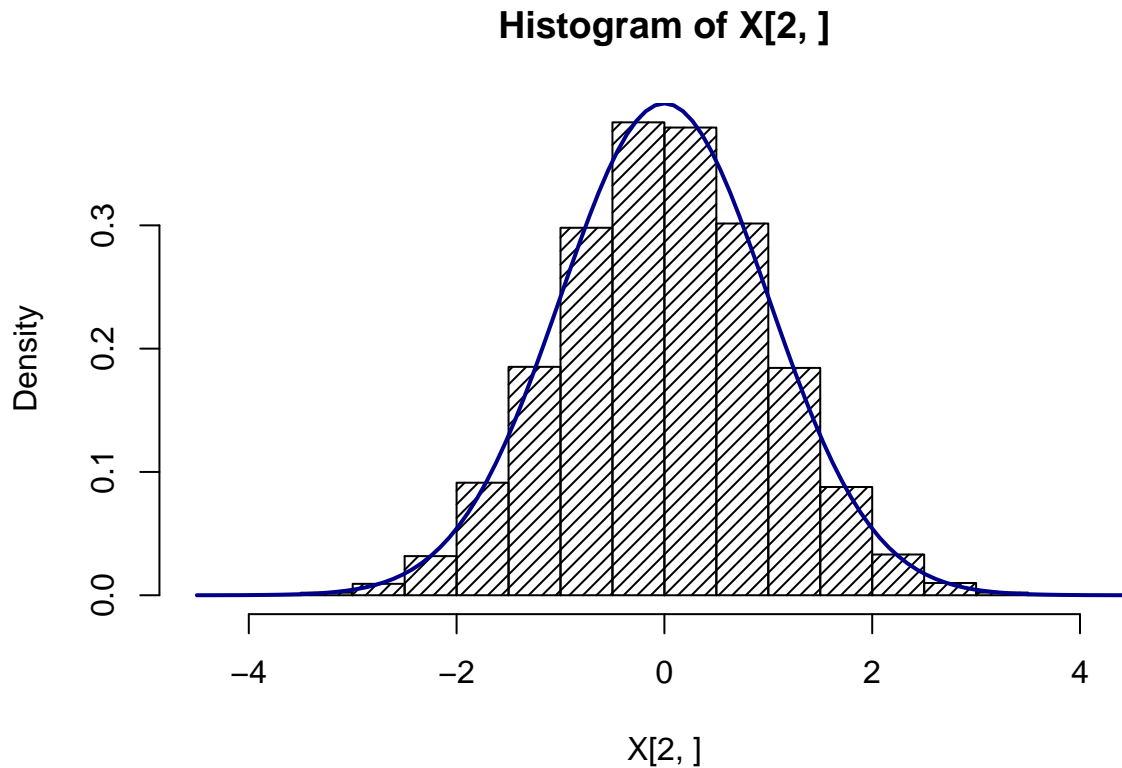
The function `generate.normal` uses Box-Muller to generate an n -vector of independent samples from the standard normal distribution.

```
# Generate samples from normal distribution with Box-Muller
n <- 100000 #Number of samples
generate.normal <- function(n){
  u1 <- runif(n) # Uniform distributed random variable
  u2 <- runif(n) # Uniform distributed random variable
  x1 <- sqrt(-2*log(u1))*cos(2*pi*u2) # Normal distributed random variable
  x2 <- sqrt(-2*log(u1))*sin(2*pi*u2) # Normal distributed random variable
  X <- rbind(x1, x2)
  return (X)
}
X <- generate.normal(n)
hist(X[1,], density=20, breaks = 30, prob=TRUE)
curve(dnorm, add=TRUE, col="darkblue", lwd=2)
title(main = NULL, sub = "Histogram showing the distribution of x1, as well as a normal density plot.",
```



Histogram showing the distribution of x1, as well as a normal density plot.

```
hist(X[2,], density=20, breaks = 30, prob=TRUE)
curve(dnorm, add=TRUE, col="darkblue", lwd=2)
title(main = NULL, sub = "Histogram showing the distribution of x2, as well as a normal density plot.",
```



Histogram showing the distribution of x_2 , as well as a normal density plot.

```
## Expected mean:  0 0
## Expected variance:
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
## Sample mean:  0.0003806055 0.0005302173
## Sample variance:
##           x1           x2
## x1  0.9970464462 -0.0006501524
## x2 -0.0006501524  1.0005229163
```

As we see, the observed mean and variance is fairly close to the moments of the standard normal distribution, which are zero and one. The empirical covariance $Cov(x_1, x_2)$ is also close to zero, as expected.

4)

Writing function `generate.d.normal` that generates samples from a d -variate normal distribution with mean vector μ and covariance matrix Σ .

```
# Generate n samples from d variate normal distribution
n <- 10000 #Number of samples
mu <- c(0,0)
sigma <- cbind(c(1,0.5),c(0.5,1))
generate.d.normal <- function(n, mu, sigma){
  A <- chol(sigma)
```

```

d <- length(mu)
X <- generate.normal(n*d)
x <- matrix(X[1,], nrow=d, ncol=n)
Z <- mu+t(A)%*%x
}

## Expected mean:  0 0
## Expected variance:
##      [,1] [,2]
## [1,]  1.0  0.5
## [2,]  0.5  1.0

## Sample mean:  0.004539636 0.000553604
## Sample variance:
##      [,1]      [,2]
## [1,] 0.9878056 0.4834432
## [2,] 0.4834432 0.9857431

```

As we see, $\tilde{\mu}$ is close to ER MU KUN 2-VARIAT?

Problem B

```

#Function f(x)
function.f <- function(x, alpha){
  if (x>0){
    return (x^(alpha-1)*exp(-x)/gamma(alpha))
  }
  else{
    return(0)
  }
}

#Function g(x)
function.g <- function(x, alpha){
  c = alpha*exp(1)/(alpha+exp(1))
  if (x>0 && x<1){
    return (c*x^(alpha-1))
  }
  else if(x >= 1){
    return(c*exp(-x))
  }
  else{
    return(0)
  }
}

# Generate n samples from d variate normal distribution
n <- 10000 #Number of samples
alpha <- 0.3
rejection.sampling <- function(n, alpha){
  c <- (alpha+exp(1))/(gamma(alpha)*alpha*exp(1))
  samples <- c()
}

```

```

for (i in 1:n){
  repeat{
    x <- generate.samples.g(alpha, 1)
    acceptance.prob <- function.f(x, alpha)/(c*function.g(x, alpha))
    if(runif(1) <= acceptance.prob){
      samples <- c(samples, x)
      break
    }
  }
}
return (samples)
}

```

Expected mean: 0.3

Expected variance: 0.3

Sample mean: 0.3017258

Sample variance: 0.2883756

```

# An analytic log(gamma(x, alpha))-function
log.gamma <- function(x, alpha){
  return((alpha-1)*log(x)-x)
}
# Ratio-of-uniforms method
n <- 10000
alpha <- 30
ratio.of.uniforms <- function(n, alpha){
  sample.counter <- 0
  b.minus <- 0
  log.b.plus <- (alpha+1)/2*(log(alpha+1)-1)
  log.a <- (alpha-1)/2*(log(alpha-1)-1)
  samples <- c()
  for(i in 1:n){
    repeat{
      sample.counter <- sample.counter + 1
      log.x1 <- -generate.exponential(1, rate=1) + log.a
      log.x2 <- -generate.exponential(1, rate=1) + log.b.plus
      log.y <- log.x2 - log.x1
      y <- exp(log.y)
      if (2*log.x1 <= log.gamma(y, alpha)){
        samples <- c(samples, y)
        break
      }
    }
  }
}
cat("Rejection ratio: ", (sample.counter-n)/n , "\n")
return (samples)
}

```

Rejection ratio: 3.4087

[1] "Expected: "

Mean: 30

Variance: 30

```
## [1] "Observed: "
## Mean:  29.99028
## Variance:  30.08334
n = 10000
alpha = 0.1
beta = 2
sample.gamma <- function(n, alpha, beta){
  if(alpha < 1){
    return(rejection.sampling(n, alpha)/beta)
  }
  else if (alpha == 1){
    return(generate.exponential(1, n)/beta)
  }
  else if (alpha > 1){
    return(ratio.of.uniforms(n, alpha)/beta)
  }
  else{
    return(0)
  }
}

## [1] "Expected: "
## Mean:  0.05
## Variance:  0.025
## [1] "Observed: "
## Mean:  0.05170244
## Variance:  0.02695578
```

Problem C

1)

Claim: Given independent variables $z_k \sim \text{gamma}(\alpha_k, \beta = 1)$ for $k = 1, \dots, K$, define $x_k = z_k \cdot (\sum_{k=1}^K z_k)^{-1}$ for $k = 1, \dots, K$. Then vector (x_1, \dots, x_K) has the Dirichlet distribution with parameter vector $\alpha = (\alpha_1, \dots, \alpha_K)$.
 Demonstration: Consider vector $x = (x_1, \dots, x_{K-1})$ with x_k as before and variable $v = \sum_{k=1}^K z_k$. Express the transformation

$$Z_k = X_k \cdot V \text{ for } k = 1, \dots, K-1 ; Z_K = V \cdot X_K = V(1 - X_1 - \dots - X_{K-1})$$

X, V now has joint distribution $f_{X,V} = |J| \cdot f_Z(z) = |J| \cdot \prod_{k=1}^K e^{-z_k} z_k^{\alpha_k-1} \cdot \frac{1}{\Gamma(\alpha_k)}$

$$f_{X,V} = |J| \cdot f_Z(z) = |J| \cdot \prod_{k=1}^K e^{-z_k} z_k^{\alpha_k-1} \cdot \frac{1}{\Gamma(\alpha_k)}$$

with

$$J = \begin{bmatrix} V & 0 & \dots & X_1 \\ 0 & V & \dots & \vdots \\ \vdots & \vdots & \ddots & X_{K-1} \\ -V & \dots & -V & X_K \end{bmatrix} \sim \begin{bmatrix} V & 0 & \dots & X_1 \\ 0 & V & \dots & \vdots \\ \vdots & \vdots & \ddots & X_{K-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \implies |J| = V^{K-1}$$

Here we have used row reduction on the bottom row, along with the fact that $X_K = \sum_{k=1}^{K-1} X_k$ and determinant rule for triangular matrices.

Isolate the v -part of $f_{X,V}$ and integrate with respect to v , yielding the marginal distribution f_X .

$$f_{X,V} = v^{K-1} \prod_{k=1}^K e^{-x_k v} (x_k v)^{\alpha_k - 1} \cdot \frac{1}{\Gamma(\alpha_k)} = v^{\sum \alpha_k - 1} e^{-v} \prod_{k=1}^K x_k^{\alpha_k - 1} \cdot \frac{1}{\Gamma(\alpha_k)}$$

$$f_X(x) = \prod_{k=1}^K x_k^{\alpha_k - 1} \cdot \frac{1}{\Gamma(\alpha_k)} \cdot \int_0^\infty v^{\sum \alpha_k - 1} e^{-v} dv$$

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

where the integral resolves from the fact that the Gamma distribution with shape $\sum_{k=1}^K \alpha_k$ and rate one integrates to one. We again used that $X_K = 1 - \sum_{k=1}^{K-1} X_k$. ## 2)

```
# Here, the user can manually change the alpha vector
# in order to draw a sample from any chosen Dirichlet distribution.
alphas.from.user=c(1.34534,0.9345,0.6356,20,14,5,2)

gammas.in.dirichlet <- function(alpha){
  n=length(alpha)
  gammas <- c()
  for(i in 1:n){
    gammas <- c(gammas, sample.gamma(1,alpha[i],1))
  }
  return(gammas)
}

generate.dirichlet <- function(alpha){ # Creates Dirichlet distributed vector
  g<-gammas.in.dirichlet(alpha)       # with the K chosen alphas
  print(g)                           # as K-variate parameter vector.
  v<-sum(g)
  z=g/v
  return(z)
}

dirich.var<-generate.dirichlet(alphas.from.user)

## Rejection ratio: 1
## Rejection ratio: 0
## Rejection ratio: 3
## Rejection ratio: 0
## Rejection ratio: 0
## [1] 0.5878408 0.6012833 2.5481486 16.9595079 10.4941100 6.4033915
## [7] 1.9318253
```

```

dirich.var          # The elements in the Dirichlet distribution

## [1] 0.01487222 0.01521231 0.06446748 0.42907104 0.26549819 0.16200410
## [7] 0.04887467

print(sum(dirich.var)) # The sum of the elements in the Dirichlet distribution

## [1] 1

```

In the code, the normalising of the Dirichlet distributed vector x occurs after the gammas z have been sampled, so $\|x\|_1$ is always one, which is equal to the joint expected value of x 's elements.

Problem D

1)

Given a prior Beta(1, 1), which is equal to the uniform distribution on (0, 1), and a multinomial density function:

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

we have that the posterior is distributed with:

$$f(\theta, \mathbf{y}) \propto (2 + \theta)^{y_1} (1 - \theta)^{y_2 + y_3} \theta^{y_4}, \quad \theta \in (0, 1).$$

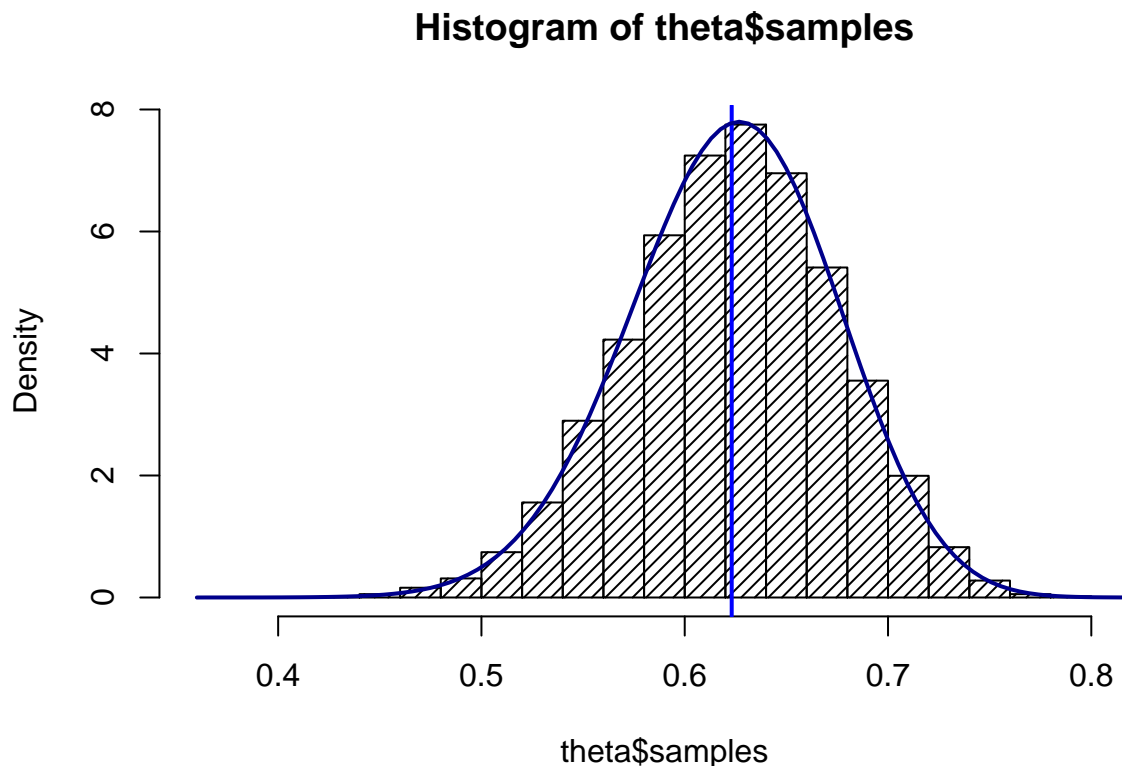
We also have the values for \mathbf{y} : $y_1 = 125$, $y_2 = 18$, $y_3 = 20$, $y_4 = 34$. Thus we can use a rejection sample method, where we sample thetas from the prior, compute the acceptance probability, and then reject/include a random number into sample. To find the constant c , we have optimized over the posterior distribution. This will not be the real c value, but c times the proportional constant of the posterior distribution and the multinomial distribution. There is however no need to find this constant as when dividing the posterior by c , it would cancel.

```

# Rejection sampling
n <- 20000 #Number of samples
posterior.f <- function(theta){
  y <- c(125, 18, 20, 34)
  return ((2+theta)^y[1]*(1-theta)^(y[2]+y[3]))*theta^y[4])
}
rejection.sampling.multinomial <- function(n){
  sample.counter <- 0
  c <- optimize(posterior.f, c(0, 1), maximum=TRUE)$objective
  samples <- c()
  for(i in 1:n){
    repeat{
      sample.counter <- sample.counter + 1
      theta <- runif(1)
      acceptance.prob <- posterior.f(theta)/c
      u <- runif(1)
      if (u <= acceptance.prob){
        samples <- c(samples, theta)
        break
      }
    }
  }
  li <- list("samples" = samples, "number.samples" = sample.counter)
  return (li)
}

```

```
}
theta <- rejection.sampling.multinomial(n)
```



am showing the distribution of theta, as well as a multinormal density plot and the mean v

```
## Theta sample mean 0.623103
```

Here is a plot over the density of the sampled θ 's as well as the sampled mean. As we can see the sample mean of θ is circa 0.623, which may suggest that the prior was not right (as we would expect mean close to 0.5 from a uniform (0, 1) distribution).

2)

Here we have sampled from the rejection sampling to obtain a set of samples such that we can use monte carlo integration to estimate the mean:

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

As well we have computed the expected mean by numerical integration of the expression:

$$E[\theta] = \int_{-\infty}^{\infty} h(\theta)f(\theta)d\theta = \int_0^1 \theta f(\theta)d\theta.$$

We also had to divide by the integral over $f(\theta)$ to get the expression normalized.

```
# Using monte carlo integration to estimate the mean
monte.carlo.integration <- function(){
  n <- 10000 # Number of samples
  samples <- rejection.sampling.multinomial(n)$samples # Sample n samples from the posterior distribution
  mu <- sum(samples)/n # Monte carlo estimate of the mean
```

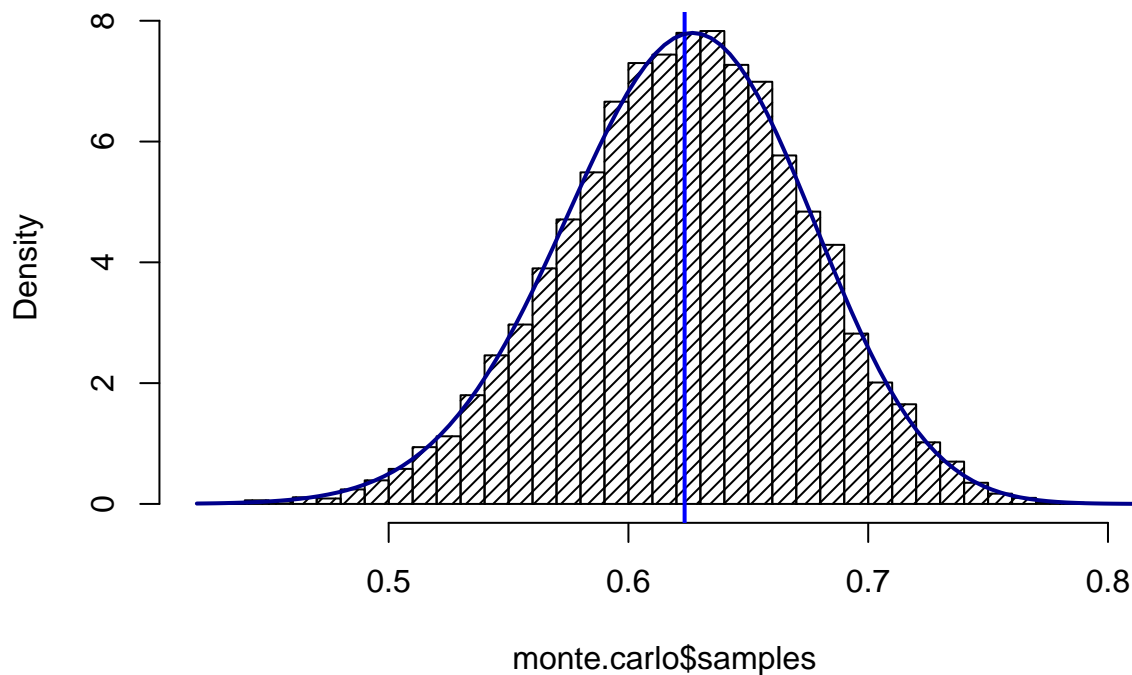
```

    return (list("observed_mean" = mu, "samples" = samples))
  }
monte.carlo <- monte.carlo.integration()

# Show histogram, density and mean of the samples
hist(monte.carlo$samples, density=20, breaks = 30, prob=TRUE)
abline(v = mean(monte.carlo$samples), col = "blue", lwd = 2)
curve(posterior.f(x)/integrate(posterior.f, lower=0, upper=1)$value, add=TRUE, col="darkblue", lwd=2)

```

Histogram of monte.carlo\$samples



```

# Function that returns theta*f(theta) for some value of theta
numerical.integration <- function(theta){
  return (theta*posterior.f(theta))
}
# Numerical integration of theta*f(theta) to obtain the expected mean
Analytic_mean <- integrate(numerical.integration, lower=0,upper=1)$value/integrate(posterior.f, lower=0
## Monte Carlo mean:  0.62343
## Analytic mean:  0.6228061

```

As we can see from the print out, the mean obtained by monte carlo integration is almost equal to the expected θ obtained by numerical integration.

3)

Sampling using the rejection algorithm leads to a number of rejected samples. In this case we have counted the ratio as we sampled, marked as observed, as well as found the analytic solution which will be:

$$c^{-1} = \int_0^1 \frac{f(x)}{c} dx.$$

Using the built in optimize and integrate function, we get the integral, which will be the ratio $\frac{\text{Total samples}}{\text{Accepted samples}}$:

```
## Observed: 7.74085
```

```
## Analytic: 7.799308
```

As we see we have a observed ratio close to the analytic ratio.

4)

We have the samples from earlier which are sampled with the prior Beta(1, 1): $x_1, \dots, x_n \sim g(x)$. However we can change this by using importance sampling:

$$w_i = \frac{f(x_i)}{g(x_i)}$$

Here $f(x) \sim \text{Beta}(1, 5)$. Since the density function will stay the same, all the elements will cancel, and we will get weights:

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

Thus the importance sampling algorithm will be for this case:

$$\tilde{\mu}_{IS} = \frac{\sum h(\theta_i) w(\theta_i)}{\sum w(\theta_i)} = \frac{\sum \theta_i w(\theta_i)}{\sum w(\theta_i)}, \quad w(\theta_i) = (1 - \theta_i)^4$$

Running this algorithm as code gives a estimator $\tilde{\mu}_{IS}$ for the new mean.

```
#
beta.distribution <- function(x, alpha, beta){
  return(x^(alpha-1)*(1-x)^(beta-1)/beta(alpha, beta))
}
importance.sampling <- function(n){
  samples <- rejection.sampling.multinomial(n)$samples
  w = (1-samples)^4
  mu.is <- sum(samples*w)/sum(w)
  return(mu.is)
}

importance.sampling(20000)
```

```
## [1] 0.5959495
```

Here we see from the print out, that the importance sampling shifts the observed mean from 0.623 to 0.596.