CompStat 1

Adrian Bruland & Mathias Opland

Problem A

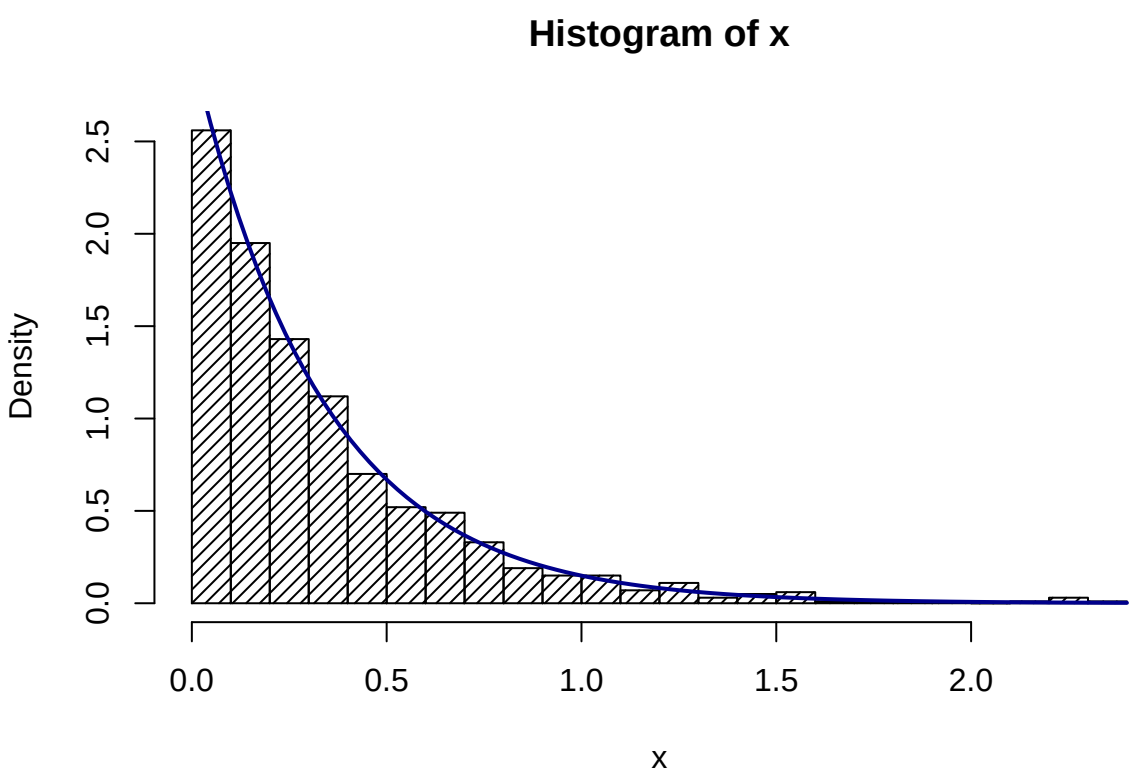
1)

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 xponential 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)
```



2)

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

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

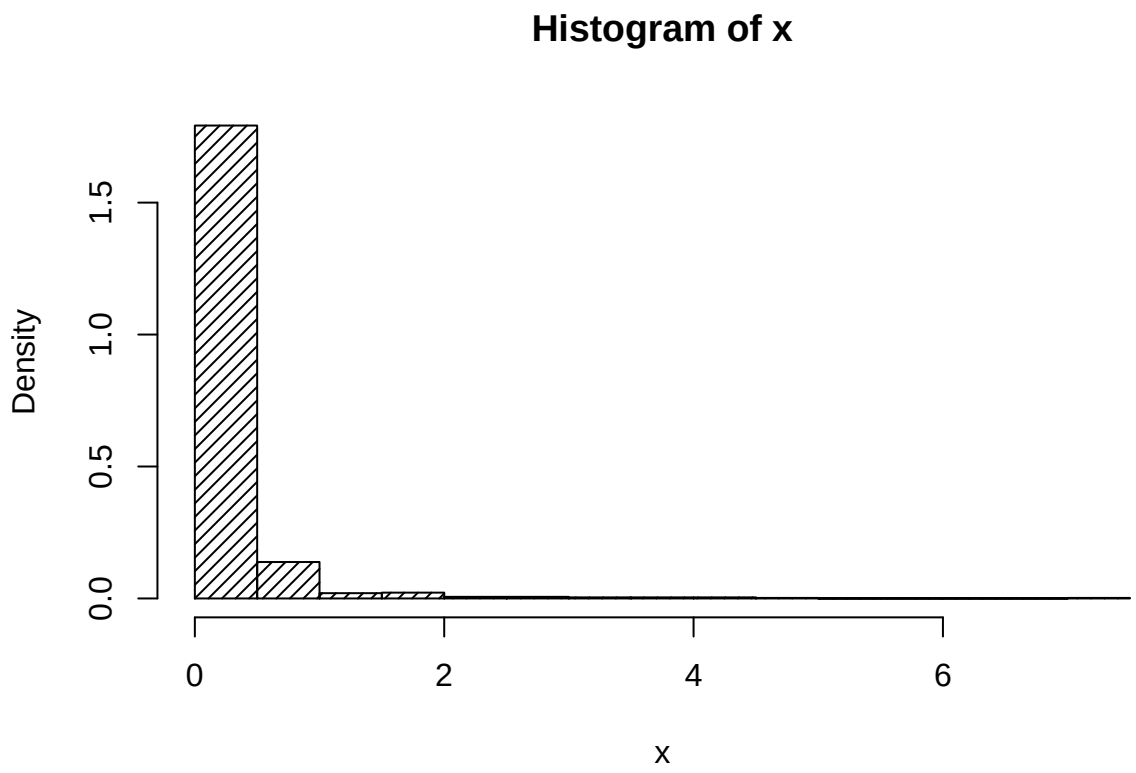
$$G^{-1}(u) = \begin{cases} \left(\frac{u}{c(\alpha-1)}\right)^{\frac{1}{\alpha}} & , \text{ SJEKK GRENSER } 0 < (\frac{u}{\alpha-1})^{\frac{1}{\alpha}} \leq 1 \\ -\log(e^{-1} - \frac{u}{c}) & , 1 \leq -\log(e^{-1} - \frac{u}{c}) \\ 0 & , u \leq 0 \end{cases}$$

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)) # Analyticaly 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)
```



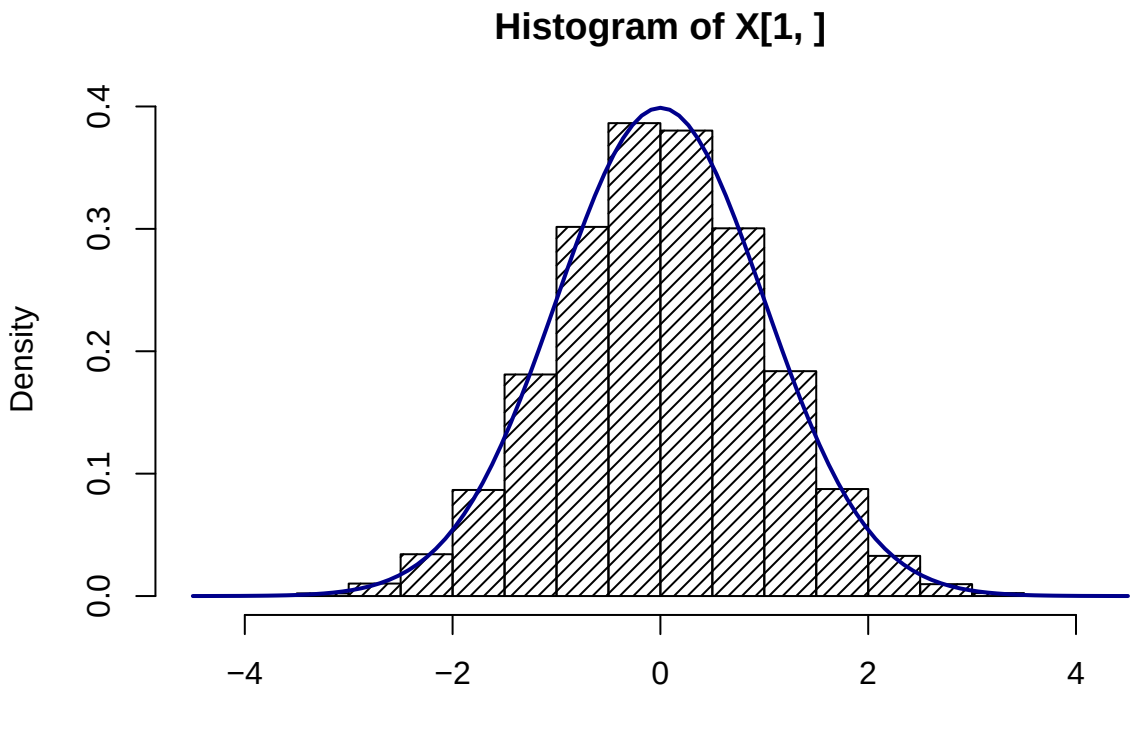
The histogram is used to check that generate.samples.g returns the correct value. As we see here, ...

3)

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

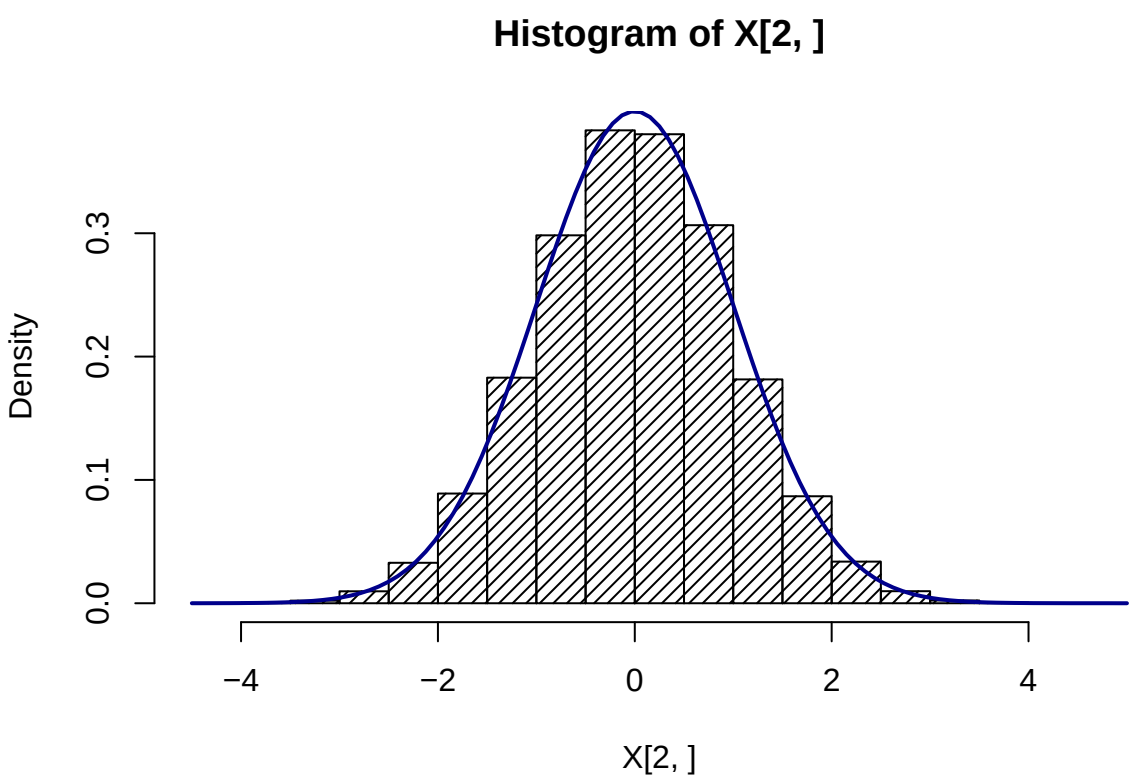
```
# Generate samples from normal distribution with Box-Muller
n <- 100000 #Number of samples
generate.normal <- function(n){
  u1 <- runif(n) # Uniform distributed random variable
  u2 <- runif(n) # Uniform distributed random variable
  x1 <- sqrt(-2*log(u1))*cos(2*pi*u2) # Normal distributed random variable
  x2 <- sqrt(-2*log(u1))*sin(2*pi*u2) # Normal distributed random variable
```

```
  X <- rbind(x1, x2) # Format the variates to be a matrix
  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 x2, as well as a normal density plot.

```
## Expected mean:  0 0
```

```
## Expected covariance:
```

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

```
## Sample mean:  -0.0005620139 0.001123503
```

```
## Sample covariance:
```

```
##              x1           x2
## x1  0.999182448 -0.001038513
## x2 -0.001038513  1.000293912
```

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) #Expected values. d is the length of the mu vector
sigma <- cbind(c(1,0.5),c(0.5,1)) # Covariance between the different distributions
generate.d.normal <- function(n, mu, sigma){
  A <- chol(sigma) # Cholesky decompostion of the covariance matrix
```

```
  d <- length(mu) # Number of variates
  X <- generate.normal(n*d) # n*d normal distributed random variables
  x <- matrix(X[1,], nrow=d, ncol=n) # Format the random variables as a n x d matrix
  Z <- mu+t(A)%*%x # Scales the random variables to be d variate
  return(Z)
}
Z <- generate.d.normal(n, mu, sigma)
```

```
## Expected mean:  0 0
```

```
## Expected covariance:
```

```
##      [,1] [,2]
## [1,]  1.0  0.5
## [2,]  0.5  1.0
```

```
## Sample mean:  0.01417172 0.009114146
```

```
## Sample covariance:
```

```
##           [,1]      [,2]
## [1,] 1.0109381 0.4987294
## [2,] 0.4987294 1.0177987
```

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

Problem B

1)

(a) c is found in A.2 to be $c = \frac{\alpha e}{\alpha + e}$. Thus we have the acceptance probability is given by:

$$P\left(U \leq \frac{1}{c} \cdot \frac{f(x)}{g(x)}\right) = \int_{-\infty}^{\infty} \frac{f(x)}{c \cdot g(x)} g(x) dx = \int_{-\infty}^{\infty} \frac{f(x)}{c} dx = c^{-1} = \frac{\alpha + e}{\alpha e}.$$

(b) Using the result in (a), we write a rejection sampling alogrithm that samples from g, check if they are accepted in f, and then return n samples from f.

```
#Function f(x) - gamma distributed returns a value given x and alpha
function.f <- function(x, alpha){
  if (x>0){
    return (x^(alpha-1)*exp(-x)/gamma(alpha))
  }
  else{
    return(0)
  }
}

#Function g(x) from A returns a value given x and alpha
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)) # Expression for c
  samples <- c()
  for (i in 1:n){
    repeat{
      x <- generate.samples.g(alpha, 1) # Generate a sample from g
      acceptance.prob <- function.f(x, alpha)/(c*function.g(x, alpha)) # Compute acceptance probability
      if(runif(1) <= acceptance.prob){ # Check if random number is accepted
        samples <- c(samples, x) # Save accepted samples
        break
      }
    }
  }
  return (samples)
}
```

```
## Expected mean:  0.3
```

```
## Expected variance:  0.3
```

```
## Sample mean:  0.2887137
```

```
## Sample variance:  0.2765553
```

As the print out shows, our sampled mean and variance is close to the expected values for gamma distribution:

$$E[X] = \frac{\alpha}{\beta}, \quad Var[X] = \frac{\alpha}{\beta^2}.$$

2)

(a) Finding expressions for a, b_+ and b_-, and taking the logarithm to avoid high values for the boundaries.

$$a = \sqrt{\sup_x(f^*(x))} = \sqrt{\sup_x(x^{(\alpha-1)}e^{-x})}$$

Finding the maximum value of $x^2 f^*(x)$:

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

Further:

$$(\alpha-1) - x = 0 \implies x = (\alpha-1)$$

Thus we have an expression for x at the maximum point:

$$a = \sqrt{(\alpha-1)^{(\alpha-1)}e^{-(\alpha-1)})} \implies log(a) = \frac{(\alpha-1)}{2}log(\alpha-1) - \frac{(\alpha-1)}{2}$$

And find the expression for log(a). Doing the same for b_+:

$$b_+ = \sqrt{\sup_{x\geq 0}(x^2 f^*(x))} = \sqrt{\sup_x(x^2 x^{(\alpha-1)}e^{-x})} = \sqrt{\sup_x(x^{(\alpha+1)}e^{-x})}$$

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

$$b_+ = \sqrt{(\alpha+1)^{(\alpha+1)}e^{-(\alpha+1)}} \implies log(b_+) = \frac{(\alpha+1)}{2}log(\alpha+1) - \frac{(\alpha+1)}{2}$$

$$b_- = -\sqrt{\sup_{x\leq 0}(x^2 f^*(x))} = 0$$

We want to find an expression for the logarithm of a uniform distributed random variable, without drawing from the unifrom variable (since the upper limits are so large numbers). Thus we use:

$$u \sim U(0,1) \implies -log(u) \sim Exp(1).$$

Using linear transformation, we get:

$$u \sim U(0,1) \implies x = \beta u \sim U(0,\beta).$$

Thus we get:

$$-log(x/\beta) = -log(u) \sim Exp(1).$$

Using this we can sample log (x), where $x \sim U(0,\beta)$:

$$log(x) = -exp(1) + log(\beta)$$

(b) Under is a code sampling from the gamma distribution using ratio-of-uniforms method.

```
# 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 # Expression for b-
  log.b.plus <- (alpha+1)/2*(log(alpha+1)-1) # Expression for log(b+)
  log.a <- (alpha-1)/2*(log(alpha-1)-1) # Expression for log(a)
  samples <- c()
  for(i in 1:n){
    repeat{
      sample.counter <- sample.counter + 1
      log.x1 <- -generate.exponential(1, rate=1) + log.a # Draw from log(x1)
      log.x2 <- -generate.exponential(1, rate=1) + log.b.plus # Draw from log(x2)
      log.y <- log.x2 - log.x1 # Compute log(y) = log(x2/x1)
      y <- exp(log.y) # Get y-value
      if (2*log.x1 <= log.gamma(y, alpha)){ # Check if value are accepted
        samples <- c(samples, y)
        break
```

```
      }
    }
  }
  cat("Rejection ratio: ", (sample.counter)/n , "\n")
  return (samples)
}
y <- ratio.of.uniforms(n, alpha)
```

```
## Rejection ratio:  4.4563
```

```
## Expected mean:  30
```

```
## Expected variance:  30
```

```
## Sample mean:  30.0898
```

```
## Sample variance:  30.4631
```

As we can see, the expected mean and variance is close to our sampled mean and variance. This is a good indicator that the algorithm works. We can also see that the ratio $\frac{Total}{Accepted}$ is over 4, which means that a lot of the sample is rejected (the sample area is much larger than the function area).

3)

Now using thath the rejection sampling works for $\alpha < 1$, ratio-of-uniforms method works for $\alpha > 1$ and Gam(1, 1) = Exp(1). Now, using linear transformations:

$$X \sim Gam(\alpha, 1) \implies \frac{X}{\beta} \sim Gam(\alpha, \beta),$$

we get the following code for sampling from gamma distribution:

```
n = 10000
alpha = 0.1 # alpha parameter
beta = 2 # beta parameter
sample.gamma <- function(n, alpha, beta){
  if(alpha < 1){ #If alpha is less than 1, draw from the rejection sampling method
    return(rejection.sampling(n, alpha)/beta)
  }
  else if (alpha == 1){
    return(generate.exponential(1, n)/beta) # If alpha is equal to 1, draw from Exp(1)
  }
  else if (alpha > 1){
    return(ratio.of.uniforms(n, alpha)/beta) # If alpha is larger than 1, draw from ratio-of-uniforms m
  }
  else{
    return(0)
  }
}
```

```
## Expected mean:  0.05
```

```
## Expected variance:  0.025
```

```
## Sample mean:  0.05164078
```

```
## Expected variance:  0.02729617
```

As in B2(b), we see that the sampled mean and variance is close to the expected values. # Problem C ## 1) Claim: Given independent variables $z_k \sim \text{gamma}(\alpha_k, \beta = 1)$ for $k = 1, ..., K$, define $x_k = z_k \cdot (\sum_{k=1}^{K} z_k)^{-1}$

for $k = 1, ..., K$. Then vector $(x_1, ..., x_K)$ has the Dirichlet distrution with parameter vector $\alpha = (\alpha_1, ..., \alpha_K)$. Demonstration: Consider vector $x = (x_1, ..., x_{K-1})$ with x_k as before and vairable $v = \sum_{k=1}^{K} z_k$. Express the transformation

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

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.

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

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

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

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 $X_K = 1 - \sum_{k=1}^{K-1}$. ## 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:  2
## Rejection ratio:  3
## Rejection ratio:  1
## Rejection ratio:  2
## [1]  2.5858957  0.9579504  0.6889005 16.9023156  7.6513967  1.7446824
## [7]  1.8041887
```

```
dirich.var              # The elements in the Dirichlet distribution
```

```
## [1] 0.07997122 0.02962550 0.02130488 0.52271975 0.23662652 0.05395592
## [7] 0.05579620
```

```
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 mulitnomial density function:

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

we have that the posterior is distributed with:

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

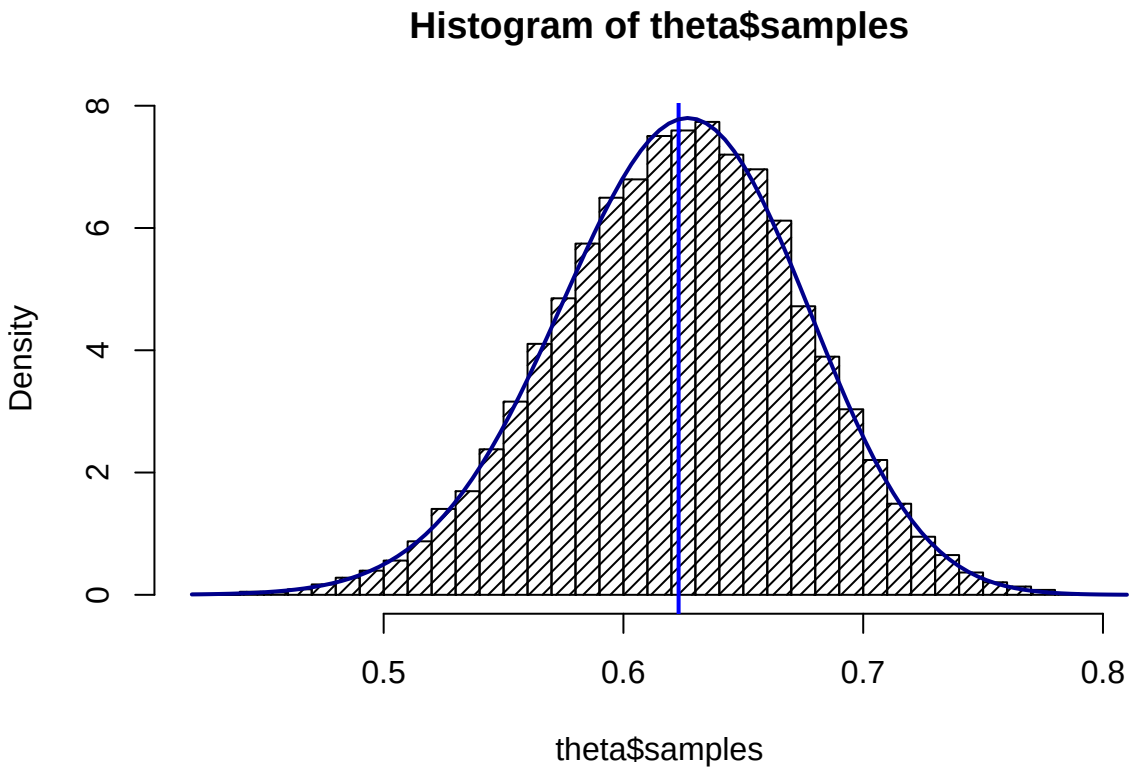
We also have the values for $\boldsymbol{y}$: $y_1 = 125$, $y_2 = 18$, $y_3 = 20$, $y_4 = 34$. Thus we can use a rejection sample method, where we sample thetas from the prior, computes the acceptance probability, and then reject/include a random number into sample. To find the constant c, we have optimized 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

# The posterior function, returns value based on input theta
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 # Finds c times the normalizing constant
```

```
  samples <- c()
  for(i in 1:n){
    repeat{
      sample.counter <- sample.counter + 1
      theta <- runif(1) # Draw theta from uniform distribution
      acceptance.prob <- posterior.f(theta)/c # Computes alpha
      u <- runif(1) # Draw random value for uniform distribution
      if (u <= acceptance.prob){ # Check if the theta is accepted
        samples <- c(samples, theta) # If accepted, store theta in samples
        break
      }
    }
  }
  return (list("samples" = samples, "number.samples" = sample.counter))
}
theta <- rejection.sampling.multinomial(n)
```



Histogram showing the distribution, density and mean value of theta.

```
## Theta sample mean 0.6230071
```

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:

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

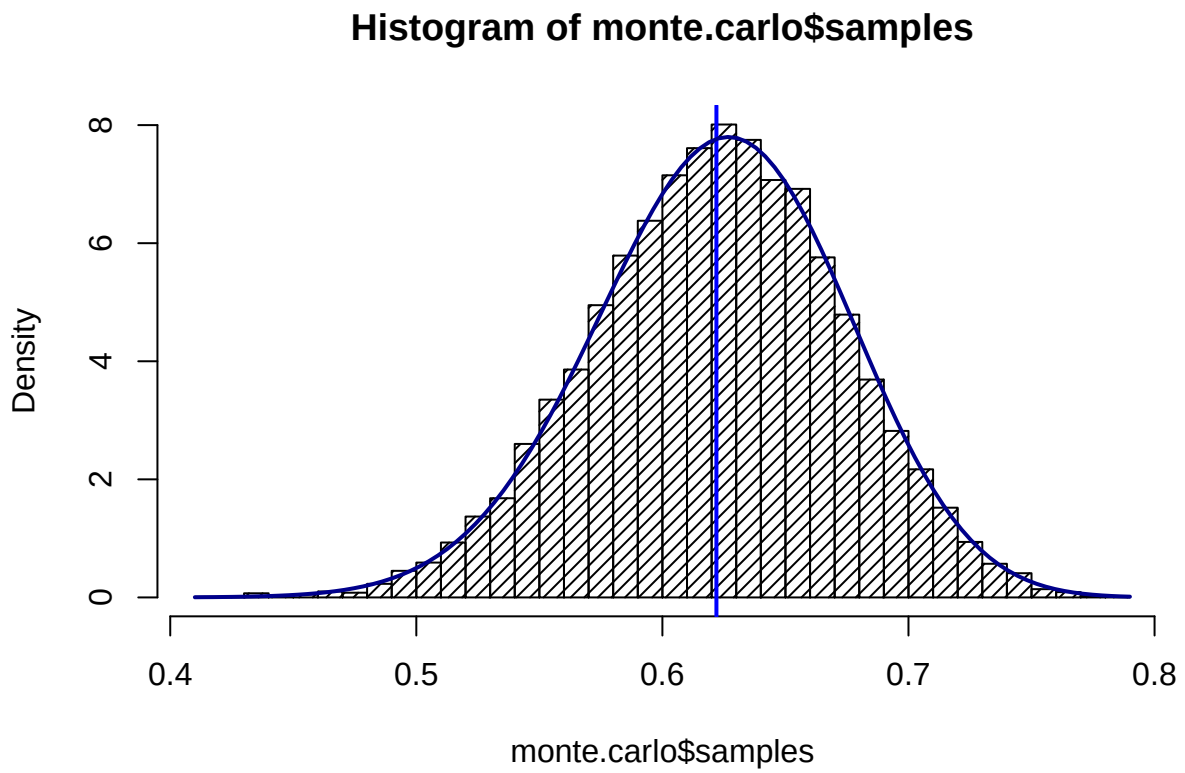
As well we have computed the expected mean by numerical integration of the expression:

$$E[\theta] = \int_{-\infty}^{\infty} h(\theta)f(\theta)d\theta = \int_{0}^{1} \theta f(\theta)d\theta.$$

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 distributi
  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)
```



```r
# Function that returns theta*f(theta) for some value of theta
numerical.integration <- function(theta){
  return (theta*posterior.f(theta))
}
# Numerival 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:  0.6219562
```

```
## 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 $\frac{\text{Total samples}}{\text{Accepted samples}}$:

```r
n = 20000 #number of samples
cat("Observed: ", rejection.sampling.multinomial(n)$number.samples/n)
```

```
## Observed:  7.7648
```

```r
cat("Analytic: ", optimize(posterior.f, c(0, 1), maximum=TRUE)$objective/integrate(posterior.f, lower=0
```

```
## 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): $x_1, ..., x_n \sim g(x)$. However we can change this by using importance sampling:

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

Here $f(x) \sim Beta(1, 5)$. 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.

```r
# Function that returns value from Beta(alpha, beta) distribution
beta.distribution <- function(x, alpha, beta){
  return(x^(alpha-1)*(1-x)^(beta-1)/beta(alpha, beta))
}
```

```r
importance.sampling <- function(n){
  samples <- rejection.sampling.multinomial(n)$samples # Draw samples from the posterior with Beta(1, 1
  w = (1-samples)^4 # Compute the weights
  mu.is <- sum(samples*w)/sum(w) # Compute the importance sampling
  return(mu.is)
}

importance.sampling(20000)
```

```
## [1] 0.5959572
```

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