TMA4300 Computer Intensive Statistical Methods Exercise 1, Spring 2019

Group members: Henrik Syversveen Lie, Mikal Solberg Stapnes

30.01.2023

Problem A: Stochastic simulation by the probability integral transform and bivariate techniques

1.

We make a function that draws n random numbers from an exponential distribution with parameter λ. If u is uniformly distributed between 0 and 1, $u \sim U(0,1)$, then $x = -\log(u)/\lambda$ will be exponentially distributed with parameter λ, or $x \sim \text{Exp}(\lambda)$.

```
library(ggplot2)  # Load library for plotting
exponential <- function(lambda, n = 1) {
    # Make default n = 1
    u = runif(n)  # Draw uniformly distributed variables
    x = -1/lambda * log(u)  # Do transformation to get exp. distr. variables
    return(x)
}
```

Then we want to check if this function is correct. To do this, we check a few things. Firstly we compare the mean and variance of a sample drawn from our function with the theoretical mean and variance. The theoretical mean and variance of a exponential distribution are $\mu = 1/\lambda$ and $\sigma^2 = 1/\lambda^2$. We draw a sample with $\lambda = 4.300$ and compare:

```
n = 10000  # Number of samples
lambda = 4.3
x = exponential(lambda, n)  # Draw samples from created function
cat("Theoretical mean: ", 1/lambda, " Computed mean: ", mean(x), "\n")
```

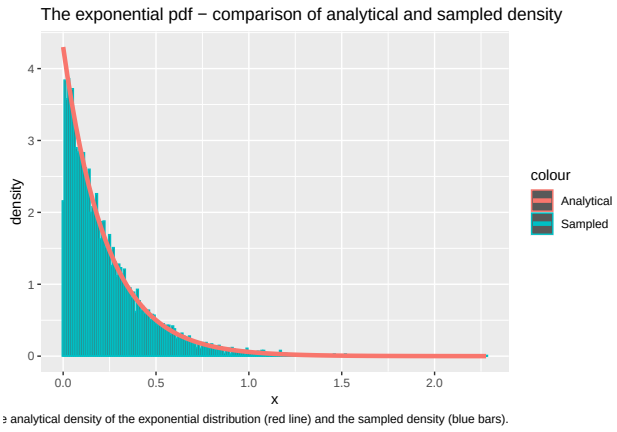
```
## Theoretical mean:  0.2325581  Computed mean:  0.2342689
```

```
cat("Theoretical variance: ", 1/lambda^2, " Computed variance: ", var(x))
```

```
## Theoretical variance:  0.05408329  Computed variance:  0.0550151
```

We see that they are almost the same. Finally we plot a histogram of our random sample compared with the analytical exponential distribution.

```r
# Prepare generated samples for plotting by putting them in a
# dataframe
df2 <- data.frame(theo = seq(0, max(x), length.out = n), x = x)
# Plot the data
ggplot(df2, aes(x = x)) + geom_histogram(aes(y = ..density.., color = "Sampled"),
    binwidth = 0.01) + stat_function(fun = dexp, geom = "line", size = 1.6,
    args = (mean = lambda), aes(color = "Analytical")) + ggtitle("The exponential pdf - comparison of a
    labs(x = "x", y = "density", caption = "Comparison of the analytical density of the exponential dis
```



From the figure we see that the density of our samples overlaps with the analytical density and can thus conclude that our algorithm is correct.

2.

We wish to sample from the pdf

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

We first note that the normalizing constant will be $c = \frac{e\alpha}{e+\alpha}$, which can be found by solving

$$\int_0^\infty g(x)dx = 1.$$

We find the cumulative distribution by noting that $G_X(x) = P(X \leq x)$, which gives

$$G_X(x) = \int_0^x g(\xi)d\xi = \begin{cases} \int_0^x c\xi^{\alpha-1}d\xi = \frac{cx^\alpha}{\alpha}, & 0 \leq x \leq 1, \\ \frac{c}{\alpha} + \int_1^x c\exp(-\xi)d\xi = \frac{c}{\alpha} + \frac{c}{e} - c\exp(-x), & 1 \leq x, \\ 0, & \text{otherwise.} \end{cases}$$

Because $c = \frac{e\alpha}{e+\alpha}$, we have that $c/\alpha + c/e = 1$, giving

$$G_X(x) = \begin{cases} \frac{cx^\alpha}{\alpha}, & 0 \leq x \leq 1, \\ 1 - c\exp(-x), & 1 \leq x, \\ 0, & \text{otherwise.} \end{cases}$$

Then we take the inverse of this function to obtain

$$G_X^{-1}(u) = \begin{cases} (\frac{\alpha}{c}u)^{1/\alpha}, & 0 \leq u < \frac{e}{\alpha+e}, \\ \ln(\frac{c}{1-u}), & \frac{e}{\alpha+e} \leq u \leq 1. \end{cases}$$

This can be used to sample from $g(x)$. To confirm this, we sample from $u \sim U(0,1)$ and compute the inverse $X = G_X^{-1}(u)$. Then our claim is that $X \sim g(x)$. We start by computing the analytical mean of our distribution and compare this with the mean of the drawn sample. We find the mean by

$$\mathrm{E}(X) = \int_0^\infty x g(x) dx = \frac{c}{\alpha+1} + \frac{2c}{e}.$$

We compare results in R.

```
# The function g(x) from problem A.2
g = function(x, alpha) {
    c = alpha * exp(1)/(alpha + exp(1))
    result = vector(length = length(x))
    result[x < 1] = c * x[x < 1]^(alpha - 1)
    result[x >= 1] = c * exp(-x[x >= 1])
    return(as.double(result))
}

# The following function returns n from the density function g(x)
# by doing inversion sampling
G_inv = function(n, alpha) {
    c = alpha * exp(1)/(alpha + exp(1))
    u = runif(n)
    result = vector(length = n)
    result[u < c/alpha] = (alpha/c * u[u < c/alpha])^(1/alpha)
    result[u >= c/alpha] = -log(1/alpha + exp(-1) - u[u >= c/alpha]/c)
    return(result)
}

alpha = 0.43
n = 1e+05
xsamples = G_inv(n, alpha)  # Sample from the density function g(x) by inversion sampling
c = alpha * exp(1)/(alpha + exp(1))
mean = c/(alpha + 1) + 2 * c/exp(1)
cat("Theoretical mean: ", mean, " Sampled mean: ", mean(xsamples))
```

```
## Theoretical mean:  0.5327939  Sampled mean:  0.5329663
```

Then we compute the theoretical variance by $\mathrm{Var}(X) = \mathrm{E}(X^2) - \mathrm{E}(X)^2$ and compare it with the sampled variance. The theoretical variance will be

$$\mathrm{Var}(X) = \mathrm{E}(X^2) - \mathrm{E}(X)^2 = \frac{c}{\alpha+2} + \frac{5c}{e} - \left(\frac{c}{\alpha+1} + \frac{2c}{e}\right)^2.$$

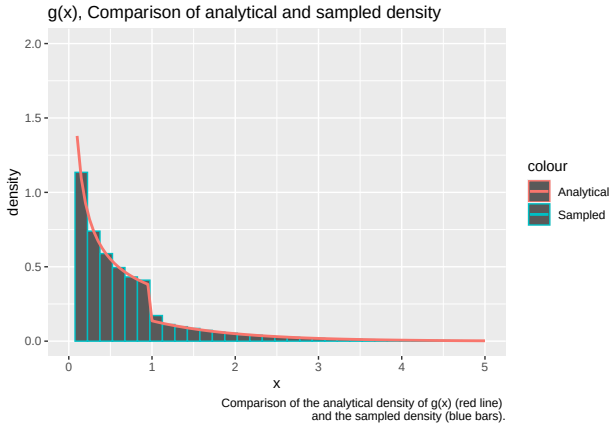
We compare results in R

```
variance = c/(alpha + 2) + 5 * c/exp(1) - mean^2
cat("Theoretical variance: ", variance, " Sampled variance: ", var(xsamples))
```

```
## Theoretical variance:  0.5518286  Sampled variance:  0.5544993
```

We see that the sampled mean and variance coincide with the theoretical mean and variance. To conclude on the correctness of our algorithm, we also make a histogram of the sample and compare it to the density function $g(x)$.

```
# Prepare generated samples for plotting by putting them in a
# dataframe
df2 <- data.frame(theo = seq(0, max(xsamples), length.out = n), value = xsamples)
# Plot the data
ggplot(df2, aes(x = value)) + geom_histogram(aes(y = ..density.., color = "Sampled"),
    binwidth = 0.15) + stat_function(fun = g, geom = "line", size = 1,
    args = (mean = alpha), alpha = 1, aes(color = "Analytical")) + ggtitle("g(x), Comparison of analyti
    ylim(0, 2) + xlim(0, 5) + labs(x = "x", y = "density", caption = paste("Comparison of the analytical
```



From the figure we see that the analytical density overlaps with the sampled density, confirming that our algorithm is correct.

3.

To simulate from the standard normal distribution, we implement a function that performs the Box-Müller transformation. Given $x_1 \sim U(0,1)$ and $x_2 \sim \text{Exp}(1/2)$, we have that $y_1 = \sqrt{x_2}\cos(2\pi x_1)$ and $y_2 = \sqrt{x_2}\sin(2\pi x_1)$ will be i.i.d. standard normal. To sample from the exponential distribution, we use the `exponential` function implemented earlier. The following function implements the Box-Müller transformation to generate n independent samples from the standard normal distribution. To reduce runtime of the algorithm, we assume n to be an even number, so that we can generate only $n/2$ samples from the uniform and exponential distributions and use both y_1 and y_2.

```
# Function to draw n i.i.d. Normal(0,1) variables based on
# Box-Muller transform
normal <- function(n) {
    x_1 <- runif(n/2)
    x_2 <- exponential(1/2, n/2)
    y_1 <- sqrt(x_2) * cos(2 * pi * x_1)
    y_2 <- sqrt(x_2) * sin(2 * pi * x_1)
    y <- c(y_1, y_2)
    return(y)
}
```

Then we want to verify the algorithm, so we compute the mean and variance of the sample.

```
n = 1e+05
x <- normal(n)  # Draw n i.i.d. Normal(0,1) variables
cat("Theoretical mean: ", 0, " Computed mean: ", mean(x), "\n")
```

```
## Theoretical mean:  0  Computed mean:  -0.005619136
```

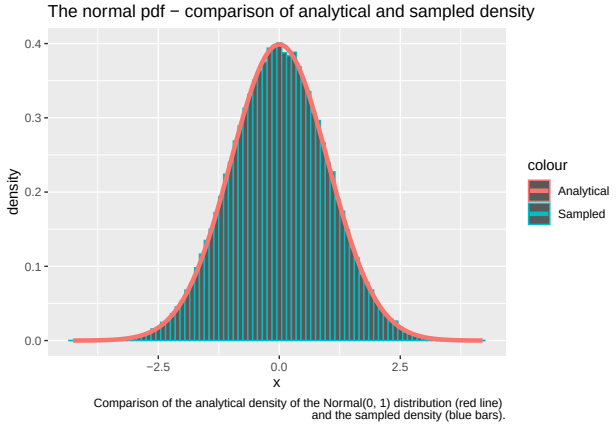
```
cat("Theoretical variance: ", 1, " Computed variance: ", var(x))
```

```
## Theoretical variance:  1  Computed variance:  1.000177
```

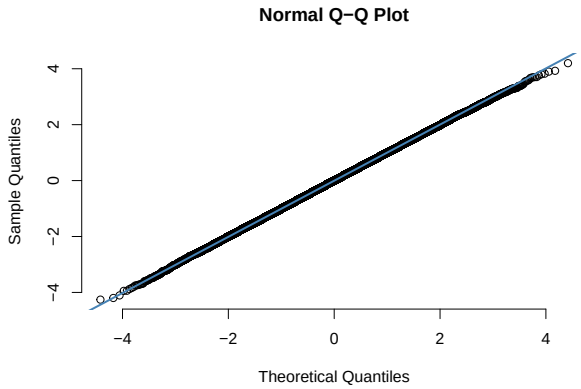
We see that the computed mean and variance coincide with the theoretical mean and variance of 0 and 1.

Finally we make a histogram of our sample and compare it with the analytical density of the standard normal distribution. In addition, we provide a Q-Q plot to check normality of the data.

```
# Prepare generated samples for plotting by putting them in a
# dataframe
df2 <- data.frame(theo = seq(0, max(x), length.out = n), value = x)
# Plot the data
ggplot(df2, aes(x = value)) + geom_histogram(aes(y = ..density.., col = "Sampled"),
    binwidth = 0.1) + stat_function(fun = dnorm, aes(col = "Analytical"),
    geom = "line", size = 1.6, args = list(mean = 0, sd = 1)) + ggtitle("The normal pdf - comparison of
    labs(x = "x", y = "density", caption = paste("Comparison of the analytical density of the Normal(0,
```




```
# Make a qq-plot of the generated samples
qqnorm(x, pch = 1, frame = FALSE)
qqline(x, col = "steelblue", lwd = 2)
```



Also here, the histogram coincides well with the standard normal density function. Lastly, the sampled points in the Q-Q plot coincide with the analytical line, so we conclude that our algorithm is correct.

4.

We now implement a function that generates samples from a d-variate normal distribution with mean vector μ and covariance matrix Σ. We use the function `normal`, which we created in task A.3. to generate d i.i.d. standard normal samples. Then we use the transformation $y = \mu + Ax \sim N(\mu, AA^T)$, with $AA^T = \Sigma$ and x being a vector with the independent standard normal samples. We use the Cholesky decomposition for A.

```
# Function to draw one sample of a d-variate Normal(mu, sigma)
# variable The function uses a transformation of d i.i.d.
# Normal(0,1) variables
normal_d <- function(d, mu, sigma) {
    x <- normal(d)
    y <- mu + t(chol(sigma)) %*% x
    return(y)
}
```

Then, we draw samples from a d-variate distribution to check our results. We use

$$\mu = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 2.5 & 3.0 \\ 3.0 & 10.0 \end{pmatrix}.$$

We check if the generated sample has the correct mean vector and covariance matrix to see if the generated sample truly is d-variate normal.

```
d = 2
n = 1e+05
mu = c(1, 2)  # Mean vector
sigma = matrix(c(2.5, 3, 3, 10), ncol = 2, nrow = 2)  # Covariance matrix
sample = matrix(NA, ncol = d, nrow = n)  # Prepare to draw samples
for (i in 1:n) {
    sample[i, ] <- normal_d(d, mu, sigma)  # Draw n samples
}
average = apply(sample, 2, mean)
cat("Mean vector: \n")
```

```
## Mean vector:
```

```
average
```

```
## [1] 0.9873293 1.9749158
```

```
covmat = cov(sample)
cat("Covariance matrix: \n")
```

```
## Covariance matrix:
```

```
covmat
```

```
##          [,1]      [,2]
## [1,] 2.500115  3.006545
## [2,] 3.006545 10.034632
```

Both the mean vector and the covariance matrix coincide well with the theoretical mean and covariance.

Finally we implement an Anderson-Darling normality test to check the normality of the data. We check that each of the two elements are normal.

```
library(nortest)
# Conduct Anderson-Daling test to check normality of the data
ad.test(sample[, 1])
```

```
##
##  Anderson-Darling normality test
##
## data:  sample[, 1]
## A = 0.47661, p-value = 0.2381
```

```
ad.test(sample[, 2])
```

```
##
##  Anderson-Darling normality test
##
## data:  sample[, 2]
## A = 0.28075, p-value = 0.642
```

Both p-values are above any reasonable significance level, which does not reject the hypothesis that the data is normal distributed, and we conclude that our algorithm is correct.

Problem B: The gamma distribution

1.

We can sample from the Gamma distribution with $\alpha \in (0, 1)$ and $\beta = 1$,

$$f_X(x) = \begin{cases} \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}, & 0 < x, \\ 0, & \text{otherwise}, \end{cases}$$

using rejection sampling with the distribution $g(x)$ found in A.2 as a proposal distribution. We can do this because $g(x)$ is nonzero at all points where $f_X(x)$ is nonzero. As a result, we can find c s.t. $f_X(x) \leq cg(x)$ for all x.

To generate n samples from $f_X(x)$ we sample from $X \sim g(x)$ and $U \sim u[0,1]$, and accept the sample x if $u \leq \frac{f_X(x)}{cg(x)}$. We note that the overall acceptance probabiltiy, P_{accept}, is

$$P_{\text{accept}} = P\left(U \leq \frac{1}{c}\frac{f_X(x)}{g(x)}\right) = \int_0^\infty \frac{f_X(x)}{cg(x)} g(x)dx = c^{-1}.$$

To maximize this acceptance probability, we minimize c by choosing

$$c = \sup_x \frac{f_X(x)}{g(x)} = \sup_x \begin{cases} \frac{e^{-x}(\frac{1}{\alpha}+\frac{1}{e})}{\Gamma(\alpha)}, & 0 \leq x \leq 1 \\ \frac{x^{\alpha-1}(\frac{1}{\alpha}+\frac{1}{e})}{\Gamma(\alpha)}, & 1 \leq x \end{cases},$$

which gives $c = \frac{(\frac{1}{\alpha}+\frac{1}{e})}{\Gamma(\alpha)}$.

We implement a function in R that generates n samples from $f_X(x)$ by the rejection algorithm.

```
small_gamma = function(alpha, n = 1) {
    result = vector()
    iter = 0
    sampled = 0
    c = (1/alpha + exp(-1))/gamma(alpha)

    while (sampled < n) {

        # Sample n realizations from g(x)
        x = G_inv(n, alpha)
        # Compute the acceptance probabilites for the n
        # realizations
        probs = dgamma(x, alpha, rate = 1)/(c * as.double(g(x, alpha)))
        # Generate n samples from U(0,1)
        u = runif(n)
        # Append samples where u < f(x) / c g(x)
        result = append(result, x[u <= probs])
        # Increase counters
        sampled = sampled + sum(u <= probs)
        iter = iter + n
    }
    # If redundant samples were generated, we do not want to count
    # these as iterations
    iter = iter - length(result) + n
    # If redundant samples were generated, return only n samples
    return(list(samples = result[seq(1:n)], iter = iter))
}
```

We compare the sample mean and variance with the theoretical mean and variance and compare the sampled density with the analytical density.

```
alpha = 0.95
G1 <- small_gamma(alpha, 10000)  # Generate samples
df = data.frame(G1)  # Put samples in dataframe to prepare for plotting
```

```
cat("Theoretical mean: ", alpha, " Computed mean: ", mean(G1$samples),
    "\n")
```

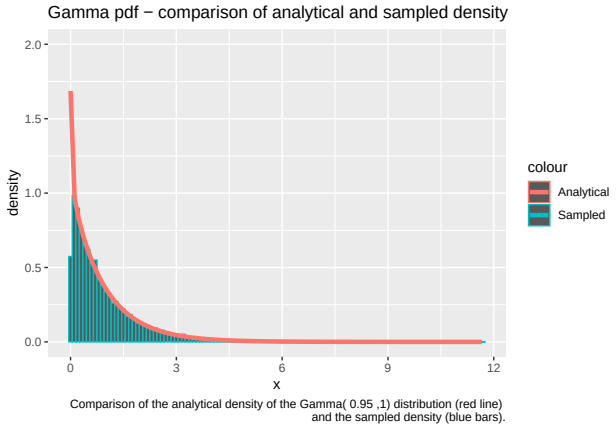
```
## Theoretical mean:  0.95  Computed mean:  0.9505148
```

```
cat("Theoretical variance: ", alpha, " Computed variance: ", var(G1$samples))
```

```
## Theoretical variance:  0.95  Computed variance:  0.9442347
```

We see that the computed mean and variance coincide with the theoretical mean and variance.

```
ggplot(data = df, aes(x = samples)) + geom_histogram(data = df, aes(x = samples,
    y = ..density.., color = "Sampled"), binwidth = 0.1) + stat_function(fun = dgamma,
    geom = "line", size = 1.6, args = list(shape = alpha, rate = 1),
    aes(x = samples, color = "Analytical")) + ggtitle("Gamma pdf - comparison of analytical and sampled
    ylim(0, 2) + labs(x = "x", y = "density", caption = paste("Comparison of the analytical density of
    alpha, ",1) distribution (red line) \n and the sampled density (blue bars)."))
```



From the figure we see that we get the correct distribution when using rejection sampling.

2.

As we remove the limitation on α, we no longer get a closed form expression for the c giving the highest acceptance probability. Instead, we may use the ratio of uniforms method to sample from the Gamma distribution with parameters $\alpha > 1$, $\beta = 1$. We define the area C_f,

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

Then we note that $y = x_2/x_1$ will be Gamma-distributed if sampled uniformly inside C_f. By finding

$$a = \sqrt{\sup_x f^*(x)}, \quad b_+ = \sqrt{\sup_{x \geq 0} x^2 f^*(x)}, \quad b_- = -\sqrt{\sup_{x \leq 0} x^2 f^*(x)},$$

$$(f^*(x))' = 0 \Leftrightarrow x^{\alpha-2}e^{-x}(\alpha - 1 - x) = 0 \Rightarrow x = \alpha - 1$$
$$a = \sqrt{(\alpha-1)^{\alpha-1}e^{1-\alpha}},$$

$$(x^2 f^*(x))' = 0 \Leftrightarrow x^{\alpha}e^{-x}(\alpha + 1 - x) = 0 \Rightarrow x = \alpha + 1$$
$$b_+ = \sqrt{(\alpha+1)^{\alpha+1}e^{-\alpha-1}},$$

$$b_- = 0,$$

we can sample uniformly from $[0, a] \times [b_-, b_+]$ and select the set of samples that are also in C_f. Then $y = \frac{x_2}{x_1}$ will be Gamma$(\alpha, 1)$ distributed. Note that in the code below, we check if $(x_1, x_2) \in C_f$ on a log-scale as $\log a, \log b$ are more easily stored than a, b when α becomes large. The equivalent condition becomes

$$\log x_1 = \log a + \log u_1, \quad u_1 \sim U[0, 1]$$
$$\log x_2 = \log b + \log u_2, \quad u_2 \sim U[0, 1]$$
$$(x_1, x_2) \in C_f \Leftrightarrow 2\log x_1 \leq (\alpha - 1)(\log x_2 - \log x_1) - \exp\{\log x_2 - \log x_1\}.$$

```
large_gamma <- function(alpha, n = 1) {
    # Log transformations of a and b+
    loga = 1/2 * (alpha - 1) * (log(alpha - 1) - 1)
    logb = 1/2 * (alpha + 1) * (log(alpha + 1) - 1)
    result = vector(mode = "numeric")

    accepted_samples = 0
    iter = 0
    while (accepted_samples < n) {
        iter = iter + 1
        log_x_1 = loga + log(runif(1))
        log_x_2 = logb + log(runif(1))
        # Accept sample if equivalent condition of (x_1, x_2) in
        # C_f is met
        if (2 * log_x_1 <= (alpha - 1) * (log_x_2 - log_x_1) - exp(log_x_2 -
            log_x_1)) {
            result = append(result, exp(log_x_2 - log_x_1))
            accepted_samples = accepted_samples + 1
        }
    }
    list(samples = result, iter = iter)
}
```

We then generate samples with our algorithm and conduct tests to check its correctness.

```
alpha = 7
n = 10000
G2 = large_gamma(alpha, n)  # Generate samples
cat("Theoretical mean: ", alpha, " Computed mean: ", mean(G2$samples),
    "\n")
```

```
## Theoretical mean:  7  Computed mean:  6.993202
```

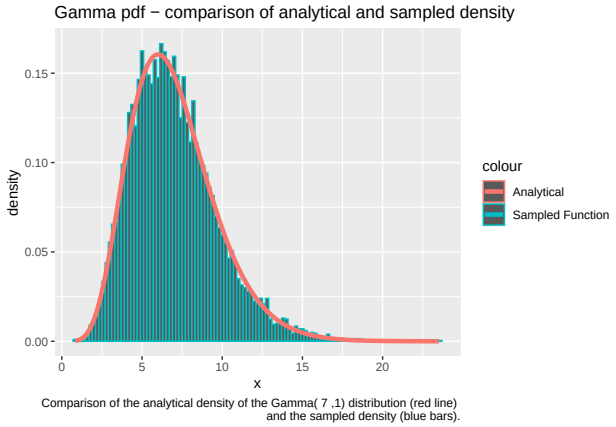
```
cat("Theoretical variance: ", alpha, " Computed variance: ", var(G2$samples))
```

```
## Theoretical variance:  7  Computed variance:  7.130239
```

We see that the computed mean and variance coincide with the theoretical mean and variance.

```
df = data.frame(data = G2)

ggplot(df, aes(x = data.samples)) + geom_histogram(data = df, aes(x = data.samples,
    y = ..density.., color = "Sampled Function"), binwidth = 0.2) + stat_function(fun = dgamma,
    geom = "line", size = 1.6, aes(color = "Analytical"), args = list(shape = alpha,
        rate = 1)) + ggtitle("Gamma pdf - comparison of analytical and sampled density") +
    labs(x = "x", y = "density", caption = paste("Comparison of the analytical density of the Gamma(",
        alpha, ",1) distribution (red line) \n and the sampled density (blue bars).",))
```



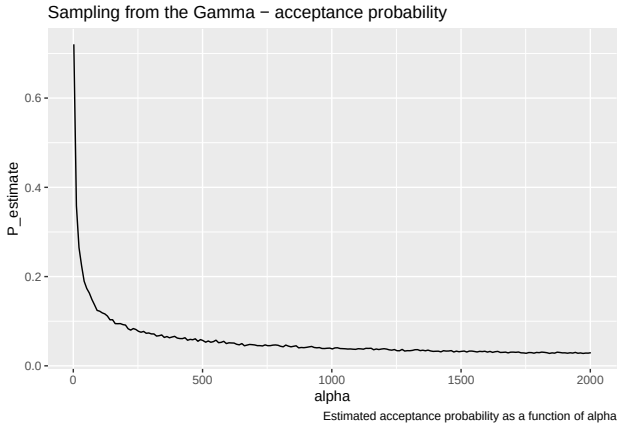
From the figure we see that we get the correct distribution when sampling using the ratio of uniforms method.

We wish to investigate in what manner α affects the acceptance probability P of the ratio of uniforms method. We draw $n = 1000$ samples for $\alpha \in (2, 2002]$ and count, for each α, the average number of iterations of the rejection sampling algorithm required to sample one gamma-distributed variable. The inverse of this will then be an estimate of the acceptance probability.

```
# The following function returns number of iterations required to
# generate 1000 samples from the large_gamma function
get_gamma_iterations <- function(alpha) {
    return(large_gamma(alpha, 1000)$iter)
}
alpha = seq(2, 2002, 10)  # Specify the alphas
iterations <- sapply(alpha, get_gamma_iterations)  # Compute number of iterations for each alpha

df = data.frame(alpha = seq(2, 2002, 10), P_estimate = 1000/iterations)  # Put data in dataframe to pre

ggplot(df, aes(x = alpha, y = P_estimate)) + geom_line() + ggtitle("Sampling from the Gamma - acceptanc
    labs(caption = "Estimated acceptance probability as a function of alpha")
```



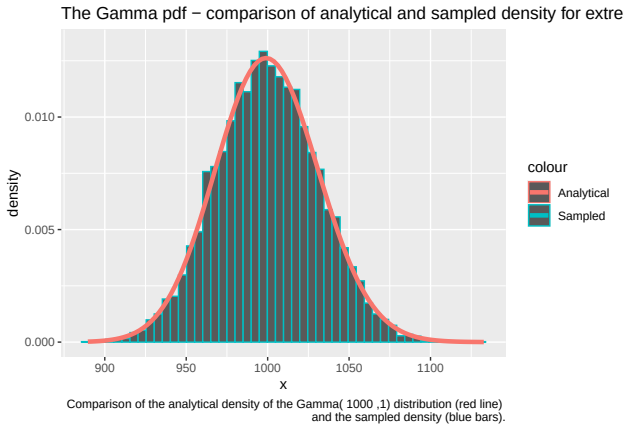
From the plot we see that P_accept decreases with larger α. This is as expected; with increasing α, the pdf flattens out and the proportion of the area in $[0, a] \times [b_-, b_+]$ covered by C_f decreases.

We now have reasonably efficient algorithms for sampling from the Gamma with $\alpha \in (0, 1), \beta = 1$ and $\alpha \in (1, \infty), \beta = 1$. We note that for $\alpha = 1, \beta = 1$, the pdf of the Gamma coincides with the unscaled exponential distribution. We also note that β is an inverse scale parameter, and $\frac{1}{\beta}X \sim Gamma(\alpha, \beta)$ if $X \sim Gamma(\alpha, 1)$. We use this and our two algorithms to develop a general algorithm for sampling from the Gamma distribution.

```
# As specified above, gamma2 samples using rejection sampling for
# alpha < 1, and the ratio of uniforms method for alpha > 1. If
# alpha = 1 we sample from the exponential
gamma2 = function(alpha, beta = 1, n = 1) {
    if (alpha == 1) {
        return(list(samples = exponential(beta, n), iter = n))
    } else if (alpha <= 1) {
        G = small_gamma(alpha, n)
        # Insert rate / scale parameters
        G$samples = G$samples/beta
        return(G)
    } else if (alpha > 1) {
        G = large_gamma(alpha, n)
        # Insert rate / scale parameters
        G$samples = G$samples/beta
        return(G)
    }
}

n = 10000
alpha = 1000
beta = 1
G = gamma2(alpha, beta, n)  # Draw samples
df <- data.frame(G)  # Prepare samples for plotting
histo = hist(G$samples, plot = F)  # Compute the sampled density
# Plot results
ggplot(df, aes(x = samples)) + geom_histogram(aes(y = ..density.., color = "Sampled"),
    bins = 50) + stat_function(fun = dgamma, geom = "line", size = 1.6,
    aes(color = "Analytical"), args = list(shape = alpha, rate = beta)) +
    ggtitle("The Gamma pdf - comparison of analytical and sampled density for extreme alpha") +
    ylim(0, 1.1 * max(histo$density)) + labs(x = "x", y = "density",
```

```
    caption = paste("Comparison of the analytical density of the Gamma(",
        alpha, ",1) distribution (red line) \n and the sampled density (blue bars)."))
```



We note that as $\alpha \to \infty$, as in the figure above, the Gamma pdf approaches a normal distribution. Taking advantage of this, sophisticated sampling techniques modify samples from the normal when sampling from the Gamma with very large α.

Problem C: The Dirichlet distribution: simulation using known relations

1.

We assume $z_k \sim Gamma(\alpha_k, 1)$ for $k = 1, \ldots, K$ independently, and define $x_k = z_k/(z_1 + \cdots + z_K)$ for $k = 1, \ldots, K$. Then we want to find the distribution of $x = (x_1, \ldots, x_K)$. We use a transformation of the joint distribution of $z = (z_1, \ldots, z_K)$ to $(x_1, \ldots, x_{K-1}, v), v = z_1 + \cdots + z_K$. The inverse transformation will be $z_k = vx_k, k = 1, \ldots, K$, which has a Jacobian

$$|J| = \begin{vmatrix} \frac{\partial z_1}{\partial x_1} & \cdots & \frac{\partial z_1}{\partial x_{K-1}} & \frac{\partial z_1}{\partial v} \\ \vdots & \ddots & \vdots & \vdots \\ \frac{\partial z_K}{\partial x_1} & \cdots & \frac{\partial z_K}{\partial x_{K-1}} & \frac{\partial z_K}{\partial v} \end{vmatrix} = \begin{vmatrix} v & 0 & \ldots & 0 & x_1 \\ 0 & v & \ldots & 0 & x_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & v & x_{K-1} \\ -v & -v & \ldots & -v & 1 - \sum_{k=1}^{K-1} x_k \end{vmatrix} = v^{K-1}.$$

Then, we note that the joint distribution of $z_k, k = 1, \ldots, K$ will be

$$F_{z_1,\ldots,z_K}(z_1, \ldots, z_K) = \prod_{k=1}^{K} \frac{1}{\Gamma(\alpha_k)} z_k^{\alpha_k - 1} \exp(-z_k) = \prod_{k=1}^{K} \left(\frac{1}{\Gamma(\alpha_k)} z_k^{\alpha_k - 1} \right) \exp(-\sum_{k=1}^{K} z_k).$$

Now, we do the transformation with $z_k = vx_k, k = 1, \ldots, K-1$ and $v = z_1 + \cdots + z_K$ to get

$$\begin{aligned} F_{x_1,\ldots,x_{K-1},v}(x_1, \ldots, x_{K-1}, v) &= \prod_{k=1}^{K-1} \left(\frac{1}{\Gamma(\alpha_k)} (vx_k)^{\alpha_k - 1} \right) \cdot \frac{1}{\Gamma(\alpha_K)} (v(1 - \sum_{k=1}^{K-1} x_k))^{\alpha_K - 1} \exp(-v) v^{K-1} \\ &= \prod_{k=1}^{K} \left(\frac{1}{\Gamma(\alpha_k)} \right) v^{\left(\sum_{k=1}^{K} \alpha_k\right) - 1} \exp(-v) \left(\prod_{k=1}^{K-1} x_k^{\alpha_k - 1} \right) \left(1 - \sum_{k=1}^{K-1} x_k \right)^{\alpha_K - 1}. \end{aligned}$$

Finally, we want to find the marginal distribution of $F_{x_1,\dots,x_{K-1}}(x_1,\dots,x_{K-1})$, so we integrate out v. The variable $v = z_1 + \cdots + z_k$ is a sum of Gamma distributed variables, so it can vary between 0 and ∞. We note that the Gamma function is defined by

$$\Gamma(z) = \int_0^\infty \exp(-x)x^{z-1}dx,$$

which gives

$$\int_0^\infty \exp(-v)v^{(\sum_{k=1}^K \alpha_k)-1}dv = \Gamma(\sum_{k=1}^K \alpha_k),$$

and in total we have

$$F_{x_1,\dots,x_{K-1}}(x_1,\dots,x_{K-1}) = \frac{\Gamma(\sum_{k=1}^K \alpha_k)}{\prod_{k=1}^K \Gamma(\alpha_k)}\left(\prod_{k=1}^{K-1} x_k^{\alpha_k-1}\right)\left(1-\sum_{k=1}^{K-1} x_k\right)^{\alpha_K-1}.$$

We also know that $\sum_{k=1}^K x_k = 1$, $x_1,\dots,x_{K-1} > 0$ and $\sum_{k=1}^{K-1} x_k < 1$, which means that $x = (x_1,\dots,x_K)$ will be Dirichlet distributed.

2.

Now we want to implement a function that draws one realization from the Dirichlet distribution with parameter vector $(\alpha_1,\dots,\alpha_K)$. We do so in the following code.

```
library(MCMCpack)
# A function for fetching only the samples, not number of
# iterations from our implemented gamma distribution function
get_gamma_samples <- function(alpha) {
    return(gamma2(alpha)$samples)
}

# Dirichlet sampler, as specified in the text above
dirichlet <- function(alpha) {
    z <- sapply(alpha, get_gamma_samples)
    x <- z/sum(z)
    return(x)
}
```

We want to test if the function draws from the correct distribution. To do so, we first check the mean and variance of a generated sample. From theory, it is known that if $(x_1,\dots,x_K) \sim \text{Dirichlet}(\alpha_1,\dots,\alpha_K)$, then

$$\text{E}(X_k) = \frac{\alpha_k}{\alpha_0}, \quad \text{Var}(X_i) = \frac{\alpha_i(\alpha_0-\alpha_i)}{\alpha_0^2(\alpha_0+1)}, \quad \alpha_0 = \sum_{i=1}^K \alpha_i.$$

To determine $(\alpha_1,\dots,\alpha_k)$, we simulate them from an exponential distribution with $\lambda = 1$ to ensure that all α are positive. We then compare sample mean and variance with theoretical mean and variance for a sample with $k = 5$.

```
k = 5
n = 20000
x <- matrix(NA, ncol = k, nrow = n)
# Sample the alphas from the exponential distribution
alpha <- exponential(1, k)
```

```r
# Sample n realizations from the Dirichlet distribution with the
# given alpha
for (i in 1:n) {
    x[i, ] <- dirichlet(alpha)
}

cat("Theoretical mean: ", alpha/sum(alpha), "\n", "Computed mean: ",
    apply(x, 2, mean), "\n")
```

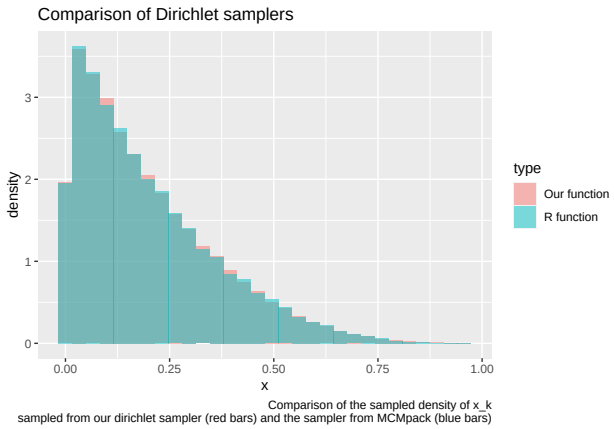
```
## Theoretical mean:  0.3797607 0.04331501 0.07724619 0.01589592 0.4837822
##  Computed mean:  0.3816152 0.04191956 0.07692338 0.01652642 0.4830155
```

```r
cat("Theoretical variance: ", alpha * (sum(alpha) - alpha)/(sum(alpha)^2 *
    (sum(alpha) + 1)), "\n", "Computed variance: ", apply(x, 2, var))
```

```
## Theoretical variance:  0.06299073 0.01108191 0.01906208 0.004183445 0.06678674
##  Computed variance:  0.06262008 0.01051496 0.01899109 0.004366568 0.06635292
```

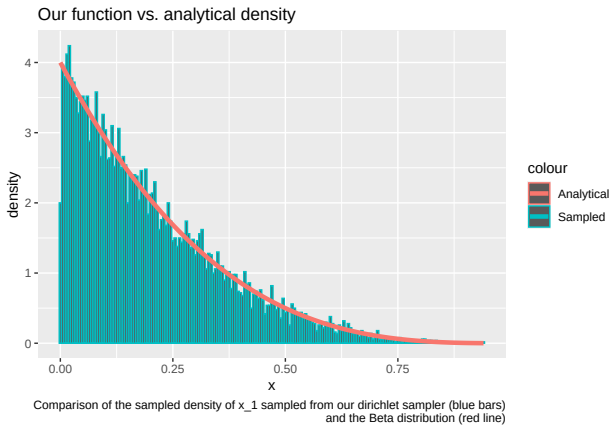
We see that the mean and variance coincide well for all x_k. To further confirm that our implementation is correct, we compare a sample drawn from our function with the already implemented R function `rdirichlet` from the `MCMCpack` library. For simplicity, we let all $\alpha_k = 1$, so each x_k has the same marginal distribution. We then make a histogram of all x_k from the two samples.

```r
n <- 10000
alpha <- rep(1, k)
x <- matrix(NA, ncol = k, nrow = n)
for (i in 1:n) {
    x[i, ] <- dirichlet(alpha)
}
y <- as.vector(rdirichlet(n, alpha))
rfunc = data.frame(value = y)
ourfunc = data.frame(value = as.vector(x))
rfunc$type = "R function"
ourfunc$type = "Our function"
df = rbind(rfunc, ourfunc)  # Prepare samples for plotting
ggplot(df, aes(value, fill = type)) + geom_histogram(alpha = 0.5, aes(y = ..density..),
    position = "identity") + ggtitle("Comparison of Dirichlet samplers") +
    labs(x = "x", y = "density", caption = paste("Comparison of the sampled density of x_k\n sampled fr
```



Also here, the histograms coincide well. As a final test, we plot a histogram of our generated sample and compare it with the analytical density function. Because the Dirichlet distribution is a multivariate distribution, we only investigate the marginal distribution of x_1, and it can be shown that the marginal distribution of any x_k is the Beta distribution with parameters α_k and $\sum_{i \neq k} \alpha_i$. This means that we can plot the density of x_1 from our generated sample and compare it with a Beta distribution density. Again, for simplicity we assume all $\alpha_k = 1$.

```
df2 <- data.frame(theo = seq(0, max(x[, 1]), length.out = n), value = x[,
    1])
ggplot(df2, aes(x = value)) + geom_histogram(aes(y = ..density.., colour = "Sampled"),
    binwidth = 0.005) + stat_function(fun = dbeta, geom = "line", size = 1.6,
    aes(col = "Analytical"), args = list(shape1 = 1, shape2 = k - 1)) +
    ggtitle("Our function vs. analytical density") + labs(x = "x", y = "density",
    caption = paste("Comparison of the sampled density of x_1 sampled from our dirichlet sampler (blue l
```



The histogram fits the analytical density well, and we conclude that we have implemented the function to generate Dirichlet samples correctly.

Problem D: Rejection sampling and importance sampling

1.

We wish to sample from

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

We construct a rejection sampling algorithm to sample from $f(\theta|\mathbf{y})$ using $U(0,1)$ as the proposal density. The uniform density is nonzero for all points where $f(\theta|\mathbf{y})$ is nonzero. As a result, we can use it as a proposal density. We get the following algorithm.

```
# The unscaled density. Note that in the text we denote f as the
# actual density.
f <- function(x) {
    return((2 + x)^125 * (1 - x)^38 * x^34)
}
# Find c in f(x) / c g(x) by finding the maximum value of f(x).
```

```r
# optimum$objective will then equal sup{f(x)}
optimum = optimize(f, c(0, 1), maximum = TRUE)

N_C = integrate(f, 0, 1)$value  # Find the normalizing constant of f

draw_samples <- function(n) {
    samples <- rep(NA, n)
    iter = 0
    for (i in 1:n) {
        finish = 0
        while (finish == 0) {
            iter = iter + 1
            x_sample <- runif(1)  # Sample from proposal density
            # Compute acceptance probability alpha
            alpha <- f(x_sample)/optimum$objective
            # Draw uniform variable to determine acceptance
            u = runif(1)
            # Accept sample if u<alpha
            if (u <= alpha) {
                samples[i] <- x_sample
                finish = 1
            }
        }
    }
    return(list(samples = samples, iterations = iter))
}
```

2.

Now we want to estimate the posterior mean θ by Monte-Carlo integration using $M = 10000$ samples from $f(\theta|\mathbf{y})$. We also draw a histogram of the samples and compare it with the analytical density, in addition to marking the estimated posterior mean.

```r
n = 10000
# The following function is integrated from 0 to 1 to find the
# theoretical mean
g <- function(x) {
    return(x * f(x))
}

# f_density is f / NC, where NC is the normalizing constant of f.
f_density <- function(x) {
    return(f(x)/N_C)
}

y <- draw_samples(n)  # Draw samples from the distribution f(theta|y)

cat("Computed mean: ", sum(y$samples)/n, "\n")
```

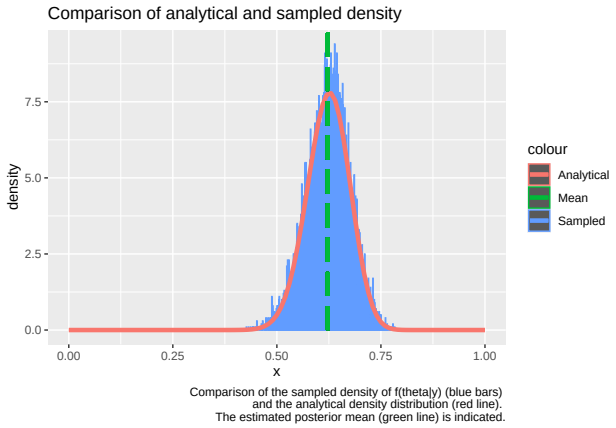
```
## Computed mean:  0.6219552
```

```r
cat("Theoretical mean: ", integrate(g, 0, 1)$value/N_C, "\n")
```

```
## Theoretical mean:  0.6228061
```

```r
df <- data.frame(theo = seq(0, 1, length.out = n), value = y$samples)

ggplot(df, aes(x = value)) + geom_histogram(aes(y = ..density.., colour = "Sampled"),
    binwidth = 0.001) + stat_function(fun = f_density, geom = "line",
    size = 1.6, aes(colour = "Analytical")) + ggtitle("Comparison of analytical and sampled density") +
    xlim(0, 1) + geom_segment(x = sum(y$samples)/n, xend = sum(y$samples)/n,
    y = 0, yend = 100, aes(colour = "Mean"), size = 1.2, linetype = "longdash") +
    labs(x = "x", y = "density", caption = paste("Comparison of the sampled density of f(theta|y) (blue
```



The theoretical mean is found by numerically integrating $\mathrm{E}(X) = \int_0^1 xf(x)dx$, and we see that it coincides well with the estimated posterior mean. From the figure we see that the density of the sample in the histogram also coincides well with the analytical density distribution, so we conclude that the sampling algorithm is correct. Also, the indicated estimated posterior mean looks reasonable, considering the density distribution.

3.

In theory, we would expect the rejection sampling algorithm to generate $c \geq f(x)/g(x)$ random numbers on average to obtain one sample of $f(\theta|y)$, where $g(x)$ is the proposal density. Because we want our algorithm to be as efficient as possible, we want c as small as possible. The proposal density $g(x)$ is $U(0,1)$, so $g(x) = 1$. Then, we can set $c = \max f(x)/g(x) = \max f(x)$. We now compare the theoretical expected number of generated numbers with the sample average.

```r
cat("Average random numbers generated: ", y$iterations/n, "\n")
```

```
## Average random numbers generated:  7.7275
```

```r
cat("Theoretical expected number: ", optimum$objective/N_C)
```

```
## Theoretical expected number:  7.799308
```

Again, we see that the average number from the sample coincide well with the theoretical expected number.

4.

We now wish to estimate the posterior mean of θ when we assume the new prior $\pi_{new} = Beta(1, 5)$. We then get the new posterior pdf

$$f_{new}(\theta|\mathbf{y}) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}\frac{\Gamma(1+5)}{\Gamma(1)\Gamma(5)}\theta^0(1-\theta)^4.$$

We want to estimate the mean of θ, $E[\theta]$. To be computationally efficient, we may wish to use importance sampling on the batch of samples already generated under the uniform prior. Using f_{new} as the target density and f_{old},

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

as the proposal density, we get the weights

$$w_i = f_{new}(\theta_i|y)/f_{old}(\theta_i|y) \propto \pi_n(\theta_i)/\pi_o(\theta_i) = \frac{\Gamma(1+5)}{\Gamma(1)\Gamma(5)}(1-\theta_i)^4,$$

The importance sampling posterior mean becomes

$$\hat{\mu}_{IS} = \frac{1}{n}\sum \theta_i w_i.$$

```
# The new unscaled posterior target pdf, NC * f_new
unscaled_posterior_target = function(x) {
    return(f_density(x) * dbeta(x, 1, 5))
}

# Compute the normalizing constant of f_new
NC_posterior = integrate(unscaled_posterior_target, 0, 1)$value

# Normalized new posterior target pdf, f_new
posterior_target = function(x) {
    return(unscaled_posterior_target(x)/NC_posterior)
}

# The density of x, f_new(x) * x
posterior_target_mean_density = function(x) {
    return(x * posterior_target(x))
}

theoretical_mean = integrate(posterior_target_mean_density, 0, 1)$value

weights = posterior_target(y$samples)/f_density(y$samples)
posterior_mean = sum(y$samples * weights)/n
cat("IS posterior mean estimate:", posterior_mean, "\n")
```

```
## IS posterior mean estimate: 0.6004672
```

```
cat("Analytical posterior mean:", theoretical_mean)
```

```
## Analytical posterior mean: 0.5959316
```

We see that the IS posterior mean estimate is close to the analytical mean. We suspect that the target and proposal densities are sufficiently close such that the bias introduced by estimating the new mean based on the already generated samples is sufficiently small.

We have here computed the normalizing constant of the posterior. If this was not numerically feasible, we could use the self-normalizing importance sampling estimate of the posterior mean

$$\tilde{\mu}_{IS} = \frac{\sum \theta_i w_i}{\sum w_i}.$$

```
unscaled_weights = unscaled_posterior_target(y$samples)/f(y$samples)
posterior_mean_SN = sum(y$samples * unscaled_weights)/(sum(unscaled_weights))

cat("self-normalizing IS posterior mean estimate:", posterior_mean_SN,
    "\n")
```

```
## self-normalizing IS posterior mean estimate: 0.5948415
```

```
cat("Analytical posterior mean:", theoretical_mean)
```

```
## Analytical posterior mean: 0.5959316
```

Again we observe the two values to be very close.

Under a new prior, we observe a slight shift in the estimate of the posterior mean. This is as expected, as such an estimate will fall between the Monte Carlo estimate under a uniform prior (0.6219552) and the expectation of the new prior $B \sim Beta(1, 5) \Rightarrow E(B) = 1/(1 + 5) = 0.167$.

We investigate what proportion of its mass the $Beta(1, 5)$ prior has on the left side of 0.6219552.

```
dbeta2 = function(x) {
    return(dbeta(x, 1, 5))
}
est = sum(y$samples)/n
cat(paste("Mass left of x =", est, ":", integrate(dbeta2, 0, est)$value))
```

```
## Mass left of x = 0.621955226750183 : 0.992278241941839
```

We observe that $Beta(1, 5)$ has the majority of its mass for small θ. This means that assuming the $Beta(1, 5)$ prior is reasonable if we have a priori knowledge that θ will be very small. Here, however, this does not seem reasonable to assume.

Simply by assuming a prior, we have induced a non-negligible change in the estimate of the posterior mean. In this case, it is possible that the $Beta(1, 5)$ prior brings us away from the true distribution of θ. This is an example of the power and dangers of the Bayesian approach; while assuming a reasonable prior may give increased accuracy in the results, wrong assumptions may induce a bias. Some might call this another facet of the bias-variance trade-off.