Exercise 1

Øyvind Klåpbakken
Jakob Gerhard Martinussen

January 18, 2019

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

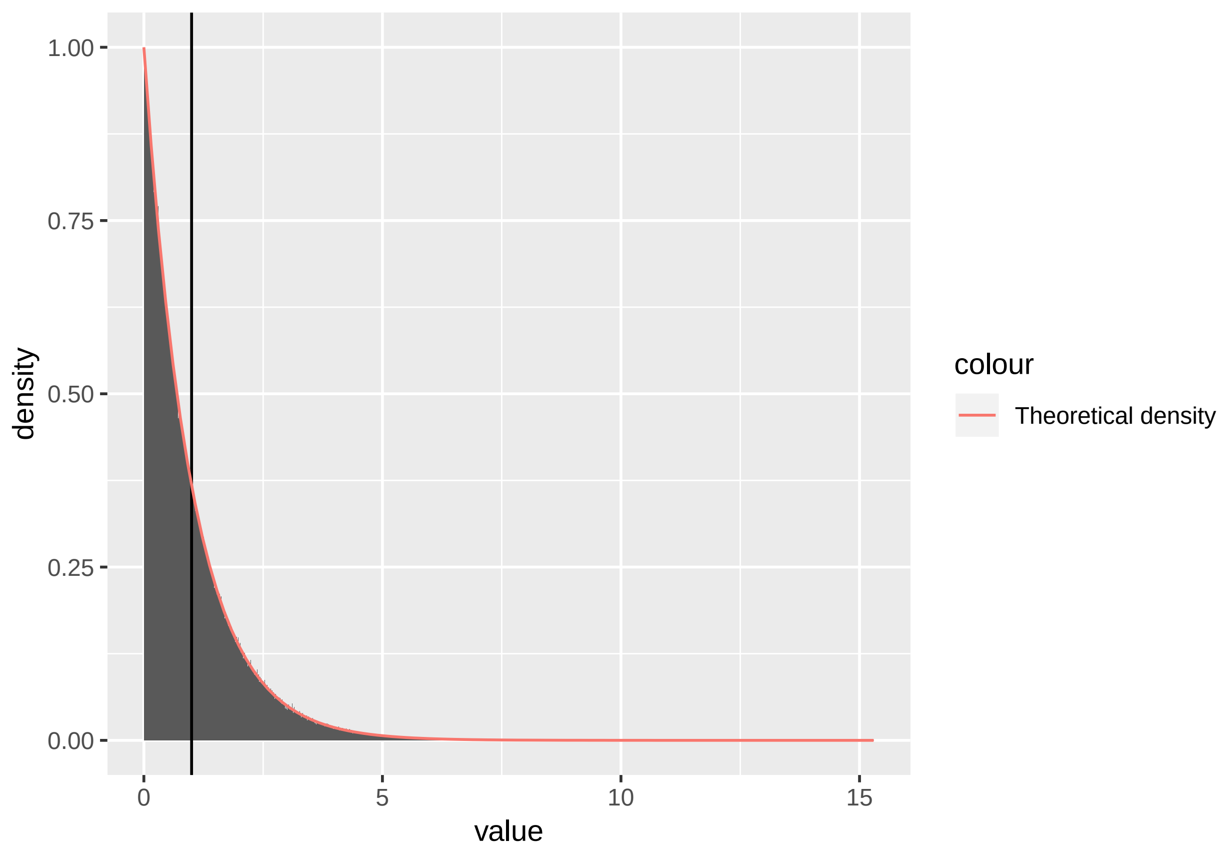
1. Sampling from the exponential distribution

We want to generate n samples from the exponential distribution with rate parameter λ.

```r
library(tidyverse)
library(magrittr)
library(numbers)
```

```r
rexp <- function(n, rate = 1) {
  uniformly_distributed <- runif(n=n)
  exponentially_distributed <- -log(uniformly_distributed) / rate
  return(enframe(exponentially_distributed))
}
```

```r
samples <- 1000000
rate <- 1
exponential_samples <- rexp(n = samples, rate = rate)
ggplot() +
  geom_histogram(
    data = exponential_samples,
    mapping = aes(x=value, y=..density..),
    binwidth = 0.01,
    boundary = 0
  ) +
  geom_vline(
    aes(xintercept = mean(exponential_samples$value))
  ) +
  stat_function(
    fun = dexp,
    args=list(rate = rate),
    aes(col='Theoretical density')
  )
```



```
results <- list(mean=1/rate,
                sample_mean = mean(exponential_samples$value),
                variance = 1/rate^2,
                sample_variance = var(exponential_samples$value))
print(results)
```

```
## $mean
## [1] 1
##
## $sample_mean
## [1] 1.000579
##
## $variance
## [1] 1
##
## $sample_variance
## [1] 0.9993657
```

2. Probability density function

We now consider the probability density function

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

where c is the normalising constant and $\alpha \in (0, 1)$. First we determine the normalising constant by integration,

$$1 = \int_{\mathbb{R}} g(x)\mathrm{d}x = \int_0^1 cx^{\alpha-1}\mathrm{d}x + \int_1^\infty ce^{-x}\mathrm{d}x = c\frac{\alpha+e}{\alpha e} \implies c = \frac{\alpha e}{\alpha+e}$$

Inserting c into the density function yields

$$g(x) = \begin{cases} \frac{\alpha e}{\alpha+e}x^{\alpha-1}, & 0 < x < 1, \\ \frac{\alpha e}{\alpha+e}e^{-x}, & 1 \leq x, \\ 0, & \text{otherwise.} \end{cases}$$

(a) Cumulative distribution

The cumulative distribution can now be found

$$G(x) = \int_0^x g(y)\mathrm{d}y = \begin{cases} 0, & x \leq 0 \\ \frac{e}{\alpha+e}x^\alpha, & 0 < x < 1, \\ 1 - \frac{\alpha e}{\alpha+e}e^{-x}, & 1 \leq x. \end{cases}$$

Now we find the inverse of the cumulative distribution function. First for the case when $G(x) < \frac{e}{\alpha+e}$

$$G(x) = \frac{e}{\alpha+e}x^\alpha \implies x = \sqrt[\alpha]{\frac{\alpha+e}{e}G(x)}$$

And for $G(x) > \frac{e}{\alpha+e}$ we have

$$G(x) = 1 - \frac{\alpha e}{\alpha+e}e^{-x} \implies x = -\ln\left(\frac{\alpha+e}{\alpha e}(1-G(x))\right)$$

The inverse cumulative function thus becomes

$$G^{-1}(x) = \begin{cases} \sqrt[\alpha]{\frac{\alpha+e}{e}x}, & 0 \leq x < \frac{e}{\alpha+e}, \\ -\ln\left(\frac{\alpha+e}{\alpha e}(1-x)\right), & \frac{e}{\alpha+e} \leq x \leq 1. \end{cases}$$

The expectation is given by $E(X) = \int_{-\infty}^\infty xg(x)dx = c\int_0^1 x^\alpha dx + c\int_1^\infty xe^{-x} = c\left(\frac{1}{\alpha+1} + \frac{2}{e}\right)$.

The variance is given $Var(X) = E(X^2) - E(X)^2$, where $E(X^2) = \int_{-\infty}^\infty x^2 g(x)dx = c\int_0^1 x^{\alpha+1}dx + c\int_1^\infty x^2 e^{-x} = c\left(\frac{1}{\alpha+2} + \frac{5}{e}\right)$.

The expression for the variance becomes $Var(X) = c\left(\frac{1}{\alpha+2} + \frac{5}{e}\right) - c^2\left(\frac{1}{\alpha+1} + \frac{2}{e}\right)^2$

(b) Sampling from $g(x)$

We now want to generate random samples from $g(x)$. Since we know the inverse of the cumulative distribution, we can use the inverse transform technique.

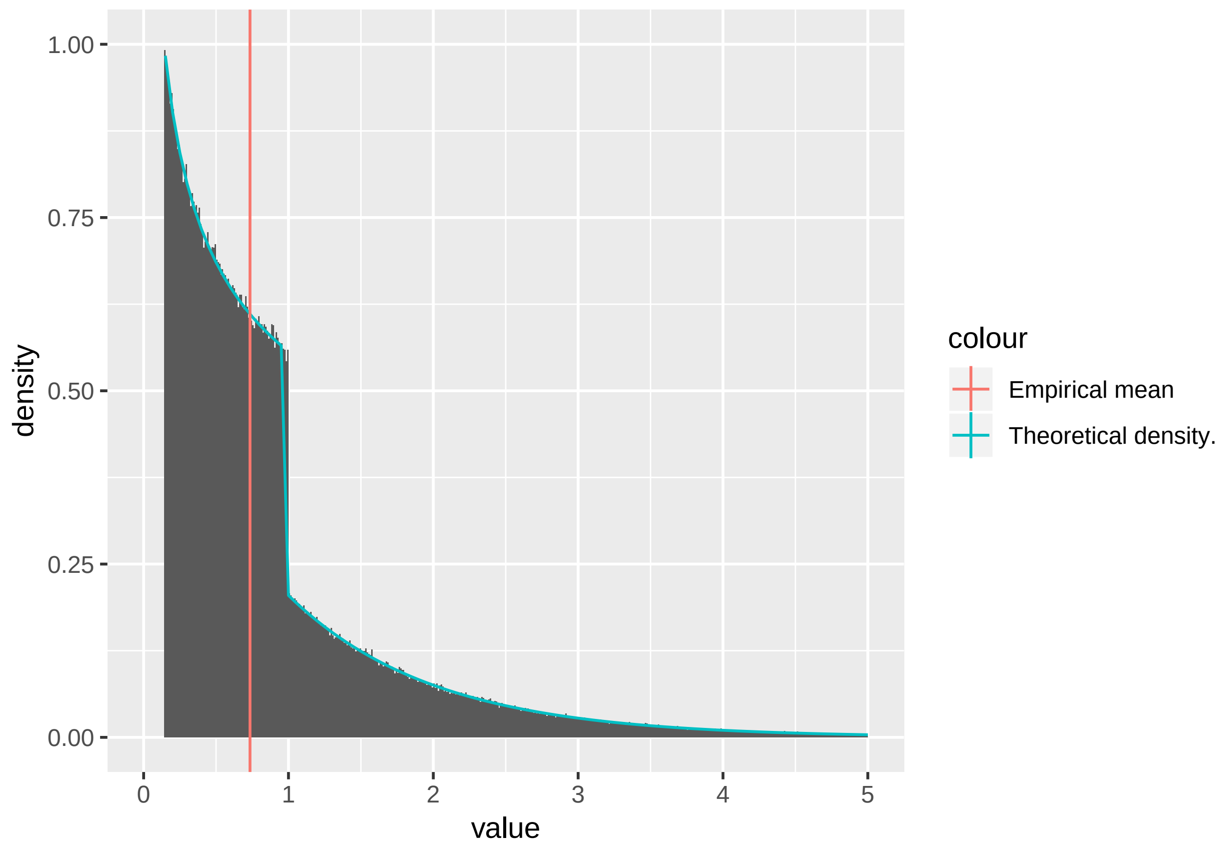
```r
rg <- function(n, alpha = 1) {
  u <- runif(n = n)
  boundary <- exp(1) / (alpha + exp(1))
  left <- u < boundary
  right <- !left
  u[left] <- (u[left] / boundary) ** (1 / alpha)
  u[right] <- -log((1 - u[right]) / (boundary * alpha))
  return(enframe(u))
}
```

We also implement the density function for the purpose of comparison

```r
dg <- function(x, alpha = 1) {
  normalizing_constant <- alpha * exp(1) / (alpha + exp(1))
  d <- rep(0, length(x))
  left_indices <- 0 < x & x < 1
  right_indices <- 1 <= x
  d[left_indices] <- normalizing_constant * (x[left_indices] ** (alpha - 1))
  d[right_indices] <- normalizing_constant * exp(-x[right_indices])
  return(d)
}
```

We now compare one million random samples generated with this sampling technique and compare it with the theoretical density

```r
samples <- 1000000
alpha <- 0.7
g_samples <- rg(n = samples, alpha = alpha)
ggplot() +
  geom_histogram(
    data = g_samples,
    mapping = aes(x=value, y=..density..),
    binwidth = 0.01,
    boundary = 0
  ) + stat_function(
    fun = dg,
    args = list(alpha = alpha),
    aes(col = 'Theoretical density.')
  ) +
  geom_vline(
    aes(
      xintercept = mean(g_samples$value),
      col = 'Empirical mean'
    )
  ) +
  ylim(0, 1) +
  xlim(0, 5)
```

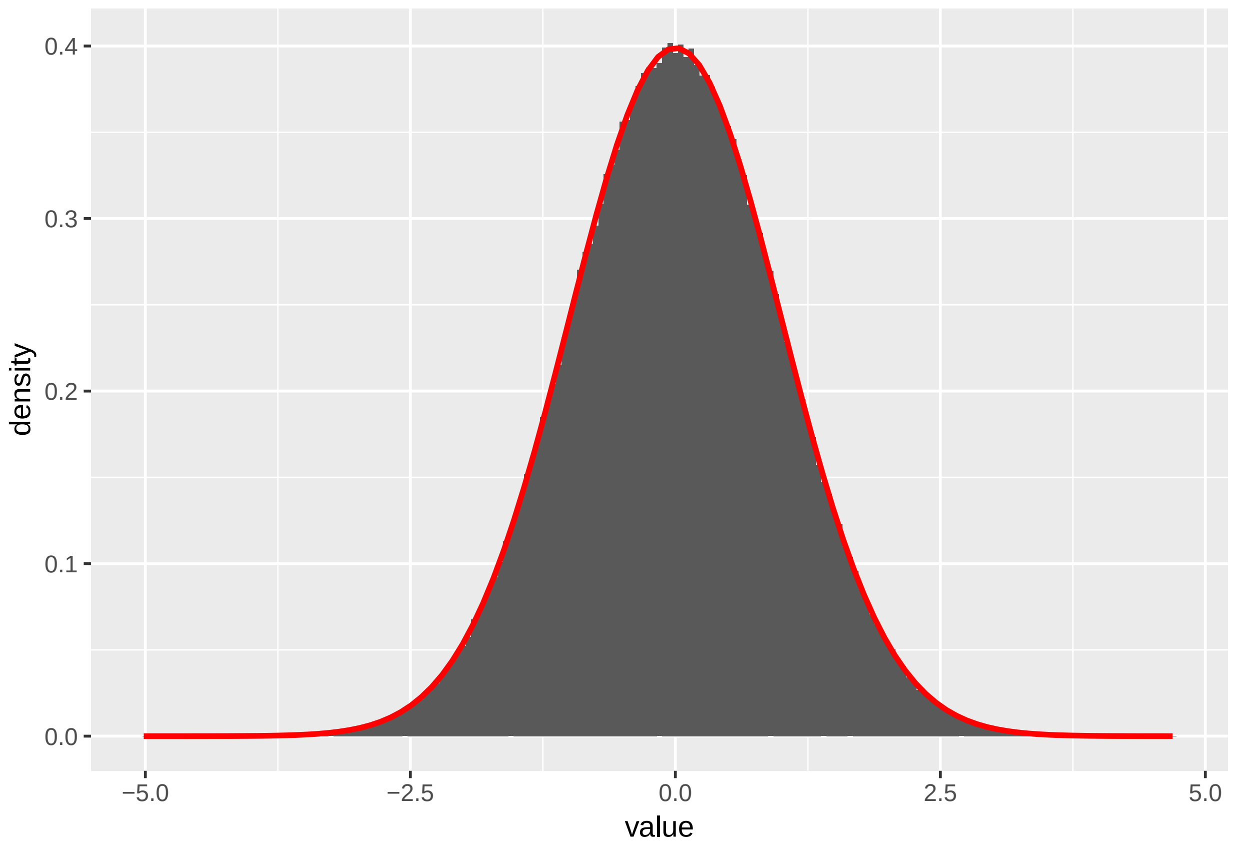


```
c <- alpha*exp(1)/(alpha + exp(1))
mean <- c*(1/(alpha + 1) + 2/exp(1))
second_moment <- c*(1/(alpha+2) + 5/exp(1))
variance <- second_moment - mean^2

results <- list(
  mean=mean,
  sample_mean = mean(g_samples %>% use_series(value)),
  variance = variance,
  sample_variance = var(g_samples %>% use_series(value)))

print(results)
```

```
## $mean
## [1] 0.7370055
##
## $sample_mean
## [1] 0.7345135
##
## $variance
## [1] 0.6868969
##
## $sample_variance
## [1] 0.6779079
```

3. Box-Muller algorithm for standard normal distribution

```
box_muller <- function(n){
  x1 <- runif((n+1)/2)*2*pi
  x2 <- rexp((n+1)/2, rate=1/2) %>% use_series(value)
  y1 <- map2_dbl(x2, x1, ~sqrt(.x)*cos(.y))
  y2 <- map2_dbl(x2, x1, ~sqrt(.x)*sin(.y))
  return(c(y1, y2)[1:n])
}

dnorm_std <- partial(dnorm, mean = 0, sd = 1)
n <- 1000000
x <- box_muller(n)

sample <- tibble(value = x)
results <- list(mean = 0,
                sample_mean = mean(x),
                variance = 1,
                sample_variance = var(x))

box_muller_plot <- sample %>%
  ggplot() +
  geom_histogram(aes(x=value, y=..density..), binwidth=0.05) +
  stat_function(fun=dnorm_std, color="red", size=1)

print(box_muller_plot)
```



```
print(results)
```

```
## $mean
## [1] 0
##
## $sample_mean
## [1] 0.0004133461
##
## $variance
## [1] 1
##
## $sample_variance
## [1] 0.9994034
```

4. Arbitrary normal distribution

```
multivariate_normal <- function(mean, covariance, n){
  d <- length(mean)
  A <- chol(covariance)

  xs <- rerun(n, box_muller(d))
  y <- map(xs, ~A%*%.x + mean)

  return(y)
}
```

The implementation is tested with $d = 3$, $\mu = [1, 4, 2]^T$ and $\Sigma = 3\mathbb{I}_3$. The empirical mean and covariance matrix is shown to agree with the specified theoretical mean and covariance matrix.

```
mean <- c(1,4,2)
covariance <- diag(3)*3
n <- 10000

sample <- multivariate_normal(mean, covariance, n)

sample_tbl <- sample %>%
  map(t) %>%
  map(as_tibble) %>%
  bind_rows()

sample_mean <- sample_tbl %>%
  colMeans()

sample_covariance <- sample_tbl %>%
  cov()

results <- list(mean=mean,
                sample_mean=as.numeric(sample_mean),
                covariance=covariance,
                sample_covariance=as.matrix(sample_covariance))
colnames(results$sample_covariance) <- NULL
rownames(results$sample_covariance) <- NULL
print(results)
```

```
## $mean
## [1] 1 4 2
##
## $sample_mean
## [1] 1.019935 4.007895 1.994632
##
## $covariance
##      [,1] [,2] [,3]
## [1,]    3    0    0
## [2,]    0    3    0
## [3,]    0    0    3
##
## $sample_covariance
##            [,1]        [,2]        [,3]
## [1,]  3.0107516 -0.01888260 -0.04014330
## [2,] -0.0188826  3.01148359 -0.04489787
## [3,] -0.0401433 -0.04489787  2.94197416
```

Problem B: The gamma distribution

1. Rejection sampling

(a) Acceptance probability

The acceptance probability is the inverse of the constant c used in the rejection-sampling algorithm. The constant c is chosen to be the smallest value that satisfies $c \geq \frac{f(x)}{g(x)}$.

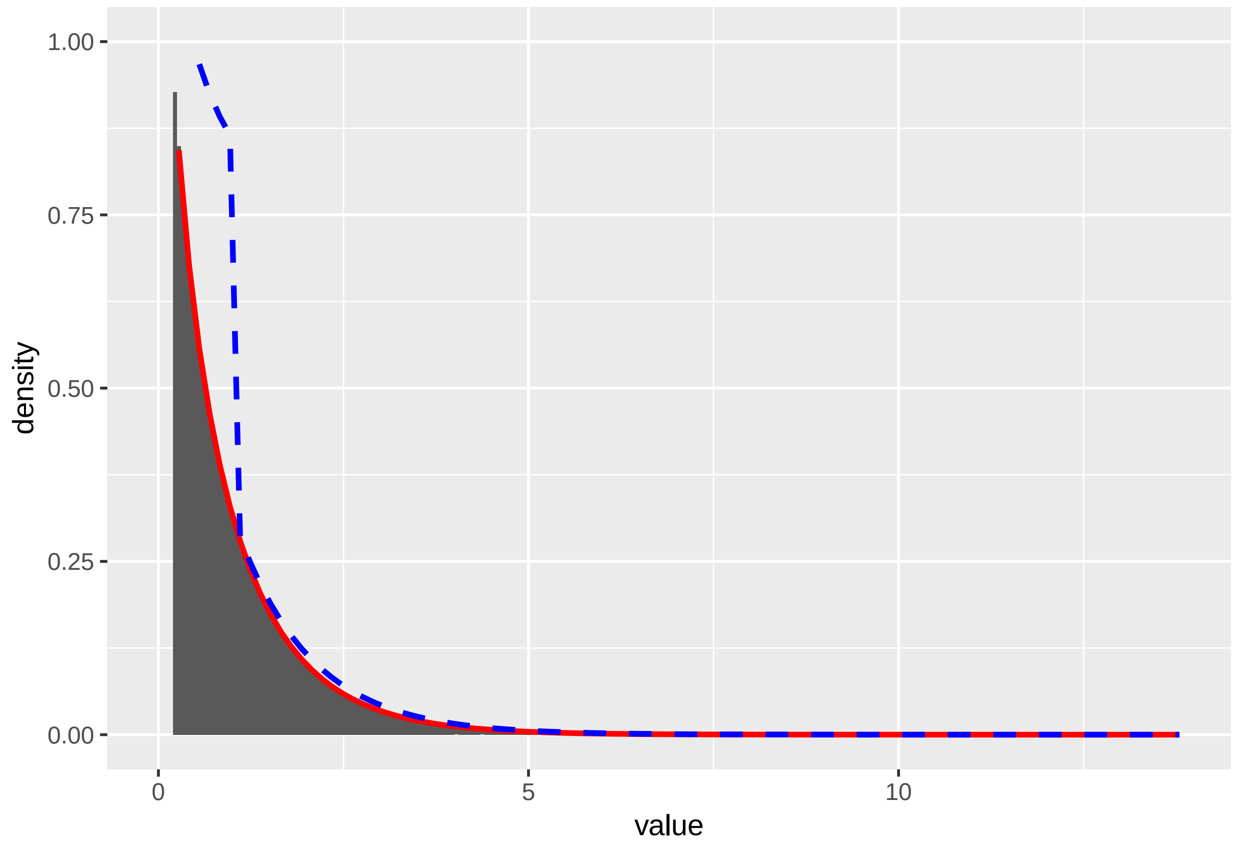
With $f(x) = \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}$ and $g(x)$ as specified in A.2, we must choose c such that

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

This functions attains it's maximum at $x = 1$, where we find $\frac{f(x)}{g(x)} \mid_{x=1} = \frac{1}{\Gamma(\alpha)} \frac{\alpha+e}{e\alpha}$ and by choosing $c = \frac{f(x)}{g(x)} \mid_{x=1}$ we satisfy the criterion above. This leads to the acceptance probability $\Gamma(\alpha)\frac{e\alpha}{e+\alpha}$.

(b) Rejection-sampling implementation

```r
n <- 1000000
alpha <- 0.8

rejection_sampling_f <- function(n, alpha){
  dg_fixed <- partial(dg, alpha=alpha)
  df <- as_mapper(~1/gamma(alpha)*.x^(alpha - 1)*exp(-.x))
  c <- (alpha + exp(1))/(alpha*exp(1))/gamma(alpha)

  n_remaining <- n
  samples <- vector()

  while (n_remaining > 0){
    acceptance_threshold_mapper <- as_mapper(~1/c*df(.x)/dg_fixed(.x))

    xs <- rg(n_remaining, alpha=alpha) %>% use_series(value)
    acceptance_threshold <- xs %>% map_dbl(acceptance_threshold_mapper)

    accepted_xs <- xs %>%
      keep(runif(n_remaining) < acceptance_threshold) %>%
      enframe()
    samples <- c(samples, accepted_xs$value)

    n_remaining <- n_remaining - nrow(accepted_xs)
  }
  return(samples)
}

samples <- rejection_sampling_f(n, alpha)

df <- as_mapper(~1/gamma(alpha)*.x^(alpha - 1)*exp(-.x))
dg_fixed <- partial(dg, alpha=alpha)
c <- (alpha + exp(1))/(alpha*exp(1))/gamma(alpha)
dg_scaled <- as_mapper(~dg_fixed(.x)*c)

ggplot(samples %>% enframe()) +
  geom_histogram(aes(x=value, y=..density..), binwidth=0.05, boundary=0) +
  stat_function(fun=df, color='red', size=1) +
  stat_function(fun=dg_scaled, color='blue', size=1, linetype="dashed") +
  ylim(0, 1) +
  xlim(0, NA)
```



```
results = list(mean=alpha,
               sample_mean=mean(samples),
               variance=alpha,
               sample_variance=var(samples))
print(paste("Acceptance probability:", 1/c))
```

```
## [1] "Acceptance probability: 0.719602267077657"
```

```
print(results)
```

```
## $mean
## [1] 0.8
##
## $sample_mean
## [1] 0.7992622
##
## $variance
## [1] 0.8
##
## $sample_variance
## [1] 0.7990012
```

2. Ratio of uniforms method

(a)

To find the values of a and b_+ we take the of derivative of $f^\star(x)$ and $x^2 f^\star(x)$and set it equal to zero. This gives us the values for which the two functions are maximized. By evaluating the functions at these points, $x = \sqrt{\alpha - 1}$ and $x = \sqrt{\alpha + 1}$ respectively, one obtains that $a = \sqrt{\left(\frac{\alpha-1}{e}\right)^{\alpha-1}}$ and $b_+ = \sqrt{\left(\frac{\alpha+1}{e}\right)^{\alpha+1}}$. The function $f^\star(x)$ is zero for $x \leq 0$, so it's supremum is 0 in this region, giving us $b_- = 0$.

Neither the values of a and b_+ nor the function $f^\star(x)$ is easy to evaluate numerically for high values of α. This problem is solved by sampling directly from the distribution of $Y_1 = \log(X_1)$ and $Y_2 = \log(X_2)$, and proceeding by checking whether the values of Y_1 and Y_2 is such that (X_1, X_2) falls within the region C_f. X_1 and X_2 are uniformly distributed on the intervals with $[0, \alpha]$ and $[b_-, b_+]$ respectively.

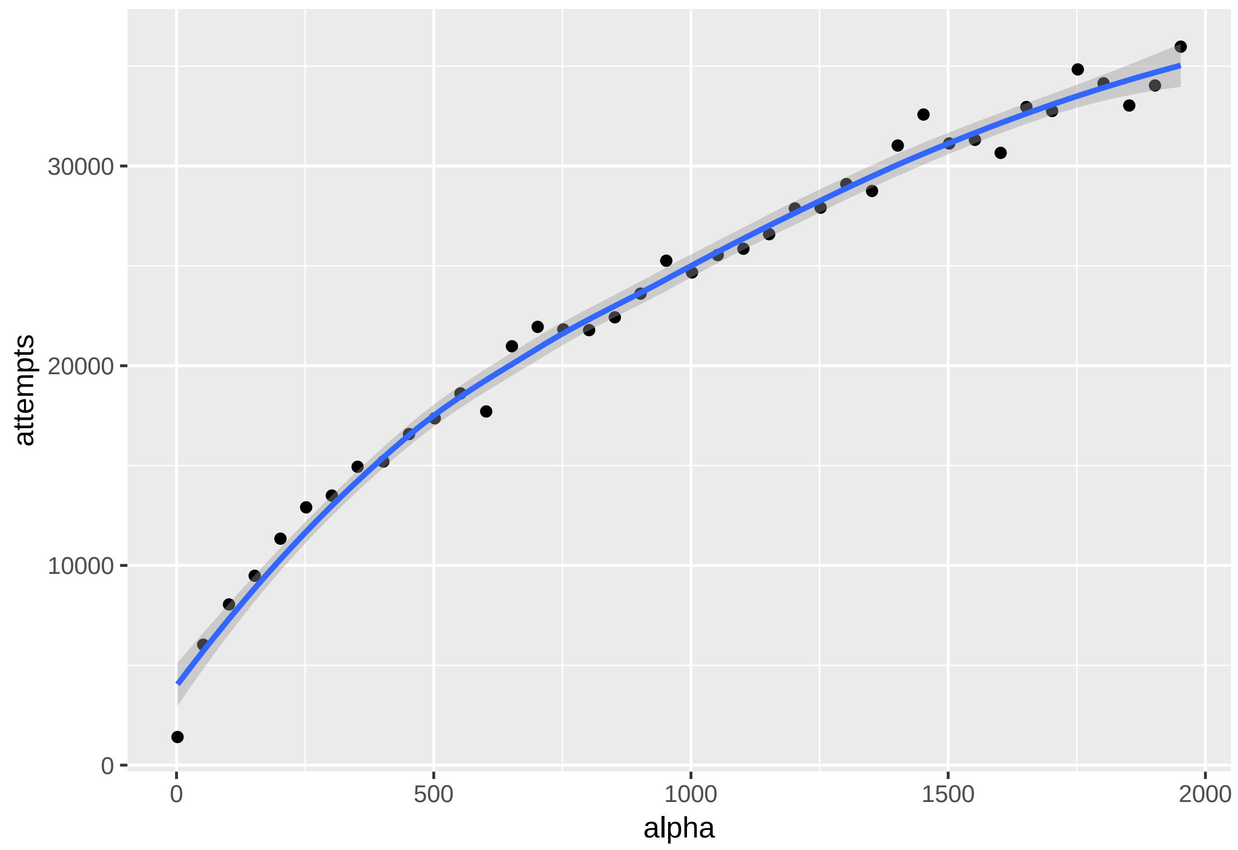
The condition on a realization of Y for checking whether X is in C_f becomes

$$y_1 = \log(x_1) \leq \frac{1}{2}\left[(\alpha - 1)(\log(x_2) - \log(x_1)) - e^{\log(x_2) - \log(x_1)}\right]$$

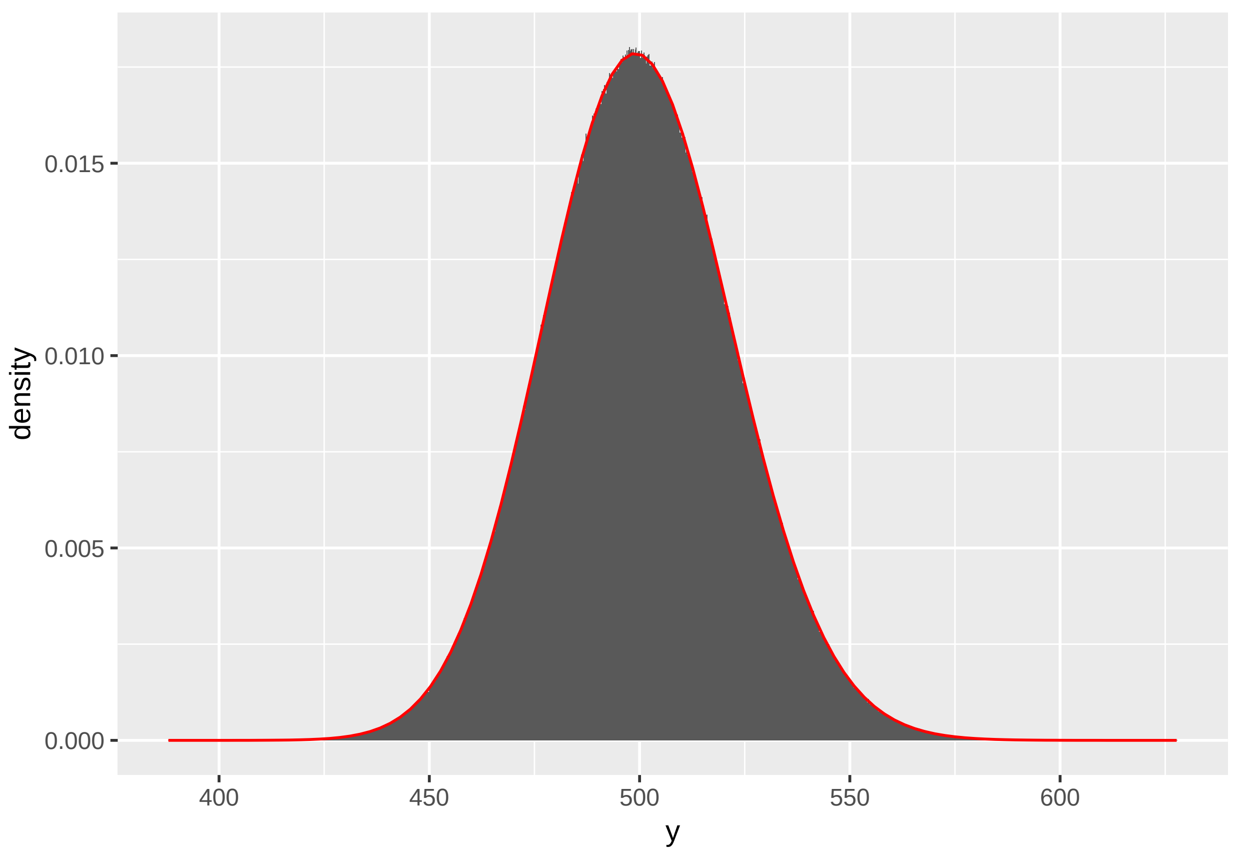
To be able to sample from the distribution of Y we need to find the inverse cumulative distribution function. The distribution function of Y_1 is given by $f_{Y_1}(x) = e^{x - \log(a)} 1_{\{x \leq \log(a)\}}$. The cumulative distribution is given by $F_{Y_1}(x) = e^{x - \log(a)}$. Setting $u = F_{Y_1}(x)$ yields $x = \log(u) + \log(a)$. In an equivalent fashion, we obtain $x = \log(u) + \log(b_+)$ for sampling from the distribution of Y_2.

```r
alpha <- 500
n <- 10000000

ratio_of_uniforms_f <- function(n, alpha){
  n_remaining <- n
  attempts <- 0
  samples <- vector()

  log_a <- (alpha - 1)/2*log((alpha - 1)/exp(1))
  log_b_plus <- (alpha + 1)/2*log((alpha + 1)/exp(1))

  while (n_remaining > 0) {
    log_x1 <- runif(n_remaining) %>% log() %>% add(log_a)
    log_x2 <- runif(n_remaining) %>% log() %>% add(log_b_plus)
    random_samples <- tibble(log_x1, log_x2)

    random_samples <- random_samples %>%
      mutate(upperBound = ((alpha - 1)*(log_x2 - log_x1)
                           - exp(log_x2 - log_x1))/2) %>%
      mutate(inRegion = log_x1 <= upperBound)

    accepted_samples <- random_samples %>%
      filter(inRegion == TRUE) %>%
      mutate(y = exp(log_x2 - log_x1))

    samples <- c(samples, accepted_samples$y)
    attempts <- attempts + n_remaining
    n_remaining <- n_remaining - nrow(accepted_samples)
  }
  return(list(samples=samples, attempts=attempts))
}
```

```r
alphas <- seq(2,2000, 50)
rf_n <- partial(ratio_of_uniforms_f, n=1000)
attempts_df <- tibble(alpha=alphas, attempts = alphas %>% map_dbl(~rf_n(.x)$attempts))
attempts_df %>%
  ggplot(aes(x=alpha, y=attempts)) +
  geom_point() +
  geom_smooth()
```



```r
accepted_samples <- ratio_of_uniforms_f(n, alpha)$samples %>%
  enframe() %>%
  set_names(c('idx', 'y'))

accepted_samples %>%
  ggplot() +
  geom_histogram(aes(x=y, y=..density..), binwidth=0.2) +
  stat_function(aes(x=y), fun = dgamma, args = list(shape = alpha, scale = 1), color='red')
```



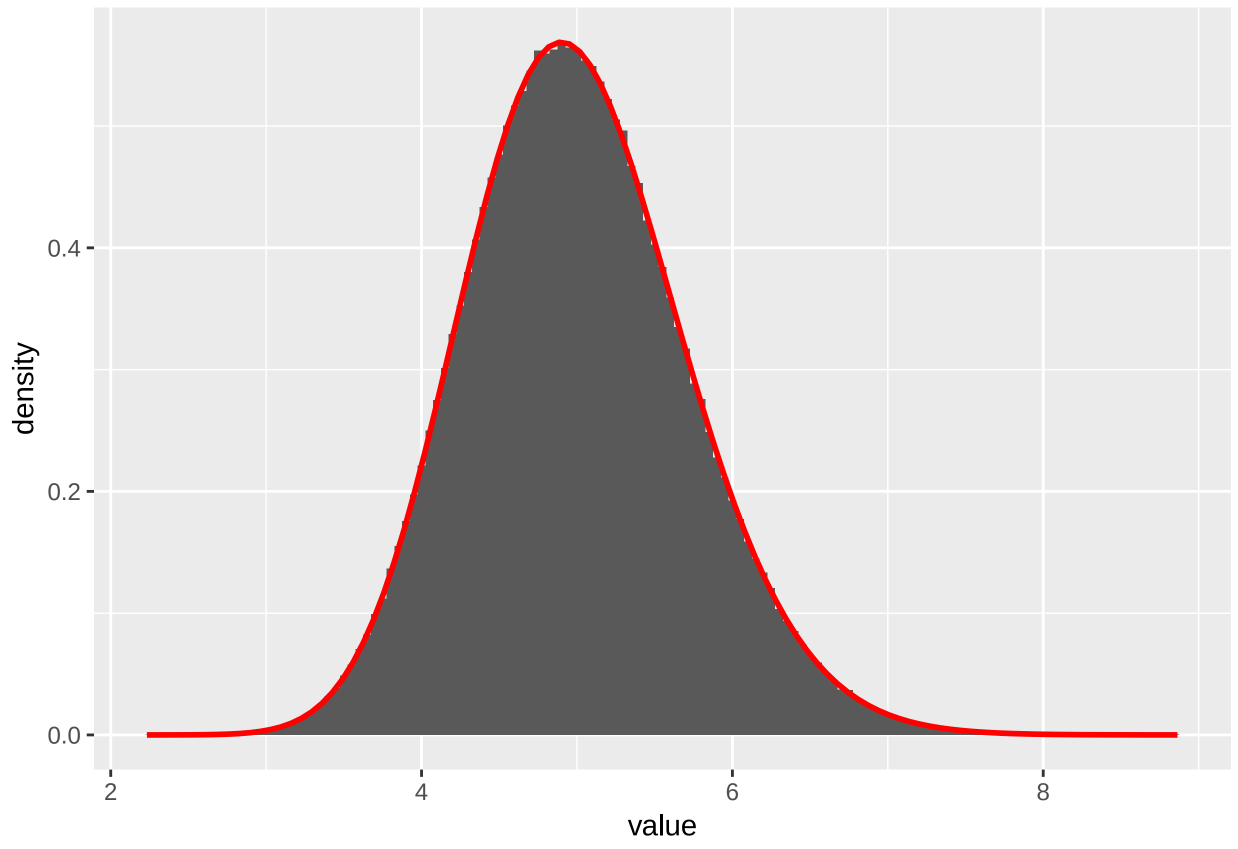
```r
results <- list(mean=alpha,
                sample_mean=mean(accepted_samples$y),
                variance=alpha,
                sample_variance=var(accepted_samples$y))

print(results)
```

```
## $mean
## [1] 500
##
## $sample_mean
## [1] 500.0028
##
## $variance
## [1] 500
##
## $sample_variance
## [1] 499.8097
```

The plot above shows how the acceptance probability decreases as the value of α increases. This happens because the proportion of the total area $[0, a] \times [b_-, b_+]$ covered by C_f shrinks as α increases. It can be seen from the plot that the proportion shrinks more rapidly when the value of α is still comparatively small, but tapers off as α approaches $\alpha = 2000$.

3. Arbitrary gamma function

```r
alpha <- 50
beta <- 10

random_sample_from_f <- function(n, alpha, beta){
  if (alpha < 1){
    result <- rejection_sampling_f(n, alpha)
  }
  else if (alpha == 1){
    result <- rexp(n, rate=1)$value
  }
  else{
    result <- ratio_of_uniforms_f(n, alpha)$samples
  }
  return(result/beta)
}

sample <- random_sample_from_f(1000000, alpha, beta)
sample %>%
  enframe() %>%
  ggplot() +
  geom_histogram(aes(x=value, y=..density..), binwidth=0.05) +
  stat_function(fun = partial(dgamma, shape=alpha, rate=beta), color='red', size=1)
```



```r
results <- list(mean = alpha/beta,
                sample_mean = mean(sample),
                variance = alpha/beta^2,
                sample_variance = var(sample))
print(results)
```

```
## $mean
## [1] 5
##
## $sample_mean
## [1] 4.998608
##
## $variance
## [1] 0.5
##
## $sample_variance
## [1] 0.5000519
```

Problem C: The Dirichlet distribution: simulating using known relations

Define the K dimensional stochastic vector $\vec{x} := (x_1, ..., x_K)$ where $x_k \in [0,1]$ and $\sum_{x=1}^{K} x_k = 1$. Since x_K is determined by $(x_1, ..., x_{K-1})$, define $\tilde{x} := (x_1, ..., x_{K-1})$. Also, define the the parameter vector $\vec{\alpha} = (\alpha_1, ..., \alpha_K)$.

The vector $\vec{x}$ is said to have a Dirichlet distribution with parameter $\vec{\alpha}$ when the probability density for $\tilde{x}$ is given by

$$f(\tilde{x}; \vec{\alpha}) = \frac{\Gamma(\sum_{k=1}^{K} \alpha_k)}{\prod_{k=1}^{K} \alpha_k} \cdot \left(\prod_{k=1}^{K-1} x_k^{\alpha_k - 1} \right) \cdot \left(1 - \sum_{k=1}^{K-1} x_k \right)^{\alpha_K - 1}$$

for $x_1, ..., x_{K-1} > 0$ and $\sum_{k=1}^{K-1} x_k < 1$.

1. Theory

Define a new K-dimensional stochastic vector $\vec{z} := (z_1, ...z_K)$ where $z_k \sim \text{gamma}(\alpha_k, 1)$, where z_k are independently distributed. We will now show that the transformation

$$x_k = \frac{z_k}{\sum_{k=1}^{K} z_k}$$

results in $\vec{x}$ being Dirichlet distributed with parameter vector $\vec{\alpha}$.

Since the z_k's are identically and independently distributed, the joint distribution can be found by

$$f_{z_k}(z_k; \alpha_k) = \frac{z_k^{\alpha_k - 1} e^{-z_k}}{\Gamma(\alpha_k)} \implies f_z(\vec{z}; \vec{\alpha}) = \prod_{k=1}^{K} f_{z_k}(z_k; \alpha_k) = \prod_{k=1}^{K} \left(\frac{z_k^{\alpha_k - 1}}{\Gamma(\alpha_k)} \right) e^{-v}$$

$$v := \sum_{k=1}^{K} z_k.$$

Now, perform a transformation, h, of the variable $\vec{z}$ to $(x_1, ..., x_{K-1}, v)$, with v as defined above. This yields

$$z_k = h_k(x_k, v) = v x_k, \ k \in \{0 \ldots, K-1\}$$

$$z_K = h_K(\tilde{x}, v) = v(1 - \sum_{k=1}^{K-1} x_k)$$

The change-of-variables formula gives the new joint distribution as

$$f_x(\tilde{x}, v; \vec{\alpha}) = f_z(\vec{z}; \vec{\alpha}) \cdot |J|,$$

where $|J|$ is the determinant of the Jacobian. It is defined by

$$J_{j,k} = \frac{\mathrm{d}z_k}{\mathrm{d}x_j},$$

for row j and column k. For brevity's sake we've defined $x_K := v$ in order to make the above expression hold for all j, k.

The elements of the Jacobian can be found by calculating $\frac{dh_k}{dx_j}$ in four distinct cases.

$$\frac{dh_k}{dx_j} = \frac{dvx_j}{dx_j} = \begin{cases} v, & k = j \\ 0, & k \neq j \end{cases} \quad , \; k, \; j \in \{1, \dots, K-1\}$$

$$\frac{dh_K}{dx_j} = \frac{d}{dx_j} v \left(1 - \sum_{k=1}^{K-1} x_k\right) = -v, \; j \in \{1, \dots, K-1\}$$

$$\frac{dh_k}{dv} = \frac{d}{dv} v x_k = x_k, \; k \in \{1, \dots, K-1\}$$

$$\frac{dh_K}{dv} = \frac{d}{dv} v \left(1 - \sum_{k=1}^{K-1} x_k\right) = 1 - \sum_{k=1}^{K-1} x_k$$

The Jacobian matrix can be reduced to upper triangular form by $K-1$ row additions without changing its determinant. The determinant of an upper triangular matrix is the product of its diagonal entries, which results in

$$\det \begin{bmatrix} v & 0 & \dots & 0 & x_1 \\ 0 & v & \dots & 0 & x_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & v & x_{K-1} \\ -v & -v & \dots & -v & 1 - \sum_{k=1}^{K-1} x_k \end{bmatrix} = \det \begin{bmatrix} v & 0 & \dots & 0 & x_1 \\ 0 & v & \dots & 0 & x_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & v & x_{K-1} \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} = v^{k-1}$$

Inserting the expressions $z_k, \; k = 0, \dots, K$ into $f_z(\vec{z}; \vec{\alpha})$ yields

$$f_z(\tilde{x}, v; \vec{\alpha}) = \frac{\prod_{k=1}^{K-1} x_k^{\alpha_k - 1}}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \left(1 - \sum_{k=1}^{K-1} x_k\right)^{\alpha_K - 1} \frac{v^{\sum_{k=1}^{K} \alpha_k}}{v^K} e^{-v}$$

Multiplying by this by the Jacobian leaves us with

$$f_z(\tilde{x}, v; \vec{\alpha}) = \frac{\prod_{k=1}^{K-1} x_k^{\alpha_k - 1}}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \left(1 - \sum_{k=1}^{K-1} x_k\right)^{\alpha_K - 1} v^{\sum_{k=1}^{K} \alpha_k - 1} e^{-v}$$

We recognize the two latter terms as the kernel of the gamma function, and by integrating out v we obtain

$$f(\tilde{x}; \vec{\alpha}) = \int_0^\infty f_z(\tilde{x}, v; \vec{\alpha}) dv = \frac{\prod_{k=1}^{K-1} x_k^{\alpha_k - 1}}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \left(1 - \sum_{k=1}^{K-1} x_k\right)^{\alpha_K - 1} \int_0^\infty v^{\sum_{k=1}^{K} \alpha_k - 1} e^{-v} dv$$

$$\implies f(\tilde{x}; \vec{\alpha}) = \frac{\prod_{k=1}^{K-1} x_k^{\alpha_k - 1}}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \left(1 - \sum_{k=1}^{K-1} x_k\right)^{\alpha_K - 1} \Gamma\left(\sum_{k=1}^{K} \alpha_k\right)$$

This is the Dirichlet distribution, as we wanted to show.

2. Implementation

We will now implement a random sampling algorithm for the Dirichlet distribution, as explained above.

```r
rdirichlet <- function(n, alpha) {
  # Dimension of Dirichlet distribution
  K <- length(alpha)

  # A tibble which will contain samples from Gamma(alpha_k, 1)
  zValues <- tibble(n = seq(1, n))
  for (k in seq_along(alpha)) {
    # Generate n samples from Gamma(alpha_k, 1)
    z <- random_sample_from_f(
      n = n,
      alpha = alpha[k],
      beta = 1
    )

    # Add these values to column named x_k,
    # since they will be transformed z_k -> x_k later
    zValues <- add_column(
      zValues,
      !!(paste("x_", toString(k), sep = "")) := z
    )
  }

  # Delete unnecessary column named "n"
  zValues <- zValues[-1]

  # Find v, the sum of the z_k's
  v <- rowSums(zValues)

  # Perform variable transformation z_k -> x_k
  xValues <- zValues / v

  # Return these values, they are now Dirichlet distributed
  return(xValues)
}
```

In order to validate the algorithm, we will draw 10 million samples with $\vec{\alpha} = (1, 3, 7, 10)$. The sample mean and variance can be compared to the theoretical mean and variance, which are given without proof

$$\mathrm{E}[X_k] = \frac{\alpha_k}{\alpha_0} \mathrm{Var}(X_k) = \frac{\alpha_k(\alpha_0 - \alpha_k)}{\alpha_0^2(\alpha_0 + 1)} \alpha_0 := \sum_{k=1}^{K} \alpha_k$$

```r
N <- 10000000
alpha <- c(1, 3, 7, 10)
alphaSum <- sum(alpha)
expectedMean <- alpha / alphaSum
expectedVariance <- alpha * (alphaSum - alpha) / (alphaSum ** 2 * (alphaSum + 1))

xValues <- rdirichlet(n = N, alpha = alpha)
sampleMean <- colMeans(xValues)
sampleVariance <- sapply(xValues, var)

comparison <- tibble(
  x = c("$x_1$", "$x_2$", "$x_3$", "$x_4$"),
  sampleMean = sampleMean,
  expectedMean = expectedMean,
  sampleVariance = sampleVariance,
  expectedVariance = expectedVariance
)
```

Table 1: Dirichlet sample statistics comparison

x	Sample mean	Theoretical mean	Sample variance	Theoretical variance
x_1	0.0476013	0.0476190	0.0020606	0.0020614
x_2	0.1428174	0.1428571	0.0055636	0.0055659
x_3	0.3332926	0.3333333	0.0101027	0.0101010
x_4	0.4762887	0.4761905	0.0113395	0.0113379

```r
comparison %>%
  kable(
    caption = "Dirichlet sample statistics comparison",
    col.names = c(
      "$x$",
      "Sample mean",
      "Theoretical mean",
      "Sample variance",
      "Theoretical variance"
    ),
    escape = FALSE
  )
```

Both the sample mean and variance is well within acceptable bounds from the theoretical values. We therefore conclude that the implementation is correct.

Table 2: Table 1: Genetic linkage data

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

Problem D: Rejection sampling and importance sampling

We now consider a specific recombination rate in genetics given by the data of Rao, C. R. (1973). 197 counts are classified into four categories, $\vec{y} = (y_1, y_2, y_3, y_4)$. These are assumed to be multinomially distributed, and the data is given in Table 1.

The multinomial mass function is given, proportionally, by

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

Using Beta(1, 1) as a prior for θ, i.e. a uniform prior, yields the following posterior density

$$f(\theta|\vec{y}) = d \cdot h(\theta|\vec{y})h(\theta) := (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4} d := \int_0^1 h(\theta|\vec{y})\mathrm{d}\theta > 0$$

Here we have introduced the unknown normalizing constant d for $h(\theta|\vec{y})$.

1. Rejection sampling algorithm

We will now implement the rejection sampling algorithm for $f(\theta|\vec{y})$. The proposal density, $g(\theta|\vec{y})$, is chosen to be the uniform distribution $\mathcal{U}(0,1)$. Thus,

$$g(\theta|\vec{y}) \equiv 1.$$

The acceptance probability thus becomes

$$\alpha = \frac{1}{c} \cdot \frac{f(\theta)}{g(\theta)} = \frac{d}{c} \cdot h(\theta|\vec{y}) \in [0,1].$$

Notice that neither c nor d are known to us, but we can numerically approximate their proportion as

$$\frac{1}{\beta} := \frac{c}{d} = \max_{\theta} h(\theta|\vec{y})$$

Such that the acceptance probability can be written as

$$\alpha = \beta \cdot h(\theta|\vec{y})$$

We now implement a function constructor, which given $\vec{y}$, returns $\hat{f}(\theta)$ such that $\max_\theta \hat{f}(\theta) = 1$.

```r
construct_f <- function(y) {
  #' Return proportional distribution function for f which satisfies max(f) = 1

  # Proportional function of f in log-space
  unscaled_log_f <- function(theta) {
    y[1] * log(2 + theta) + (y[2] + y[3]) * log(1 - theta) + y[4] * log(theta)
  }

  # Find the maxima of this function in log-space
  log_maxima <- optimize(
    f = unscaled_log_f,
    maximum = TRUE,
    interval = c(0, 1)
  )$objective

  # Scale f to have max value 1
  # NB! This does not integrate to 1, so it is not a proper density function!
  # You can use make_density for that purpose.
  scaled_f <- function(theta) {
    return(exp(unscaled_log_f(theta) - log_maxima))
  }
  return(scaled_f)
}
```

Remember that this is not a proper density function, as it does not integrate to 1. Another function constructor can be implemented to normalize $\hat{f}$. We will use this function later for plotting comparisons.

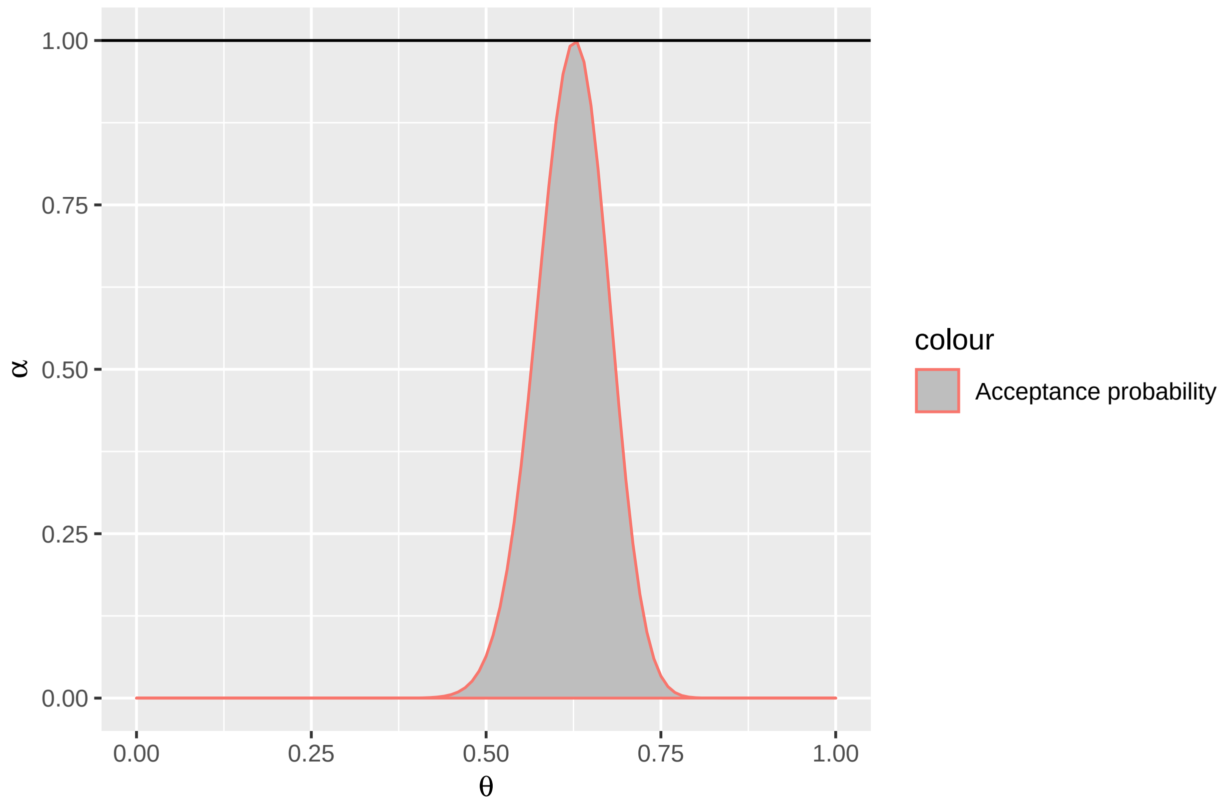
```r
make_density <- function(f, lower = 0, upper = 1) {
  #' Given f, returns a new function which integrates to 1 over the given interval
  normalizer = integrate(f, lower = lower, upper = upper)$value
  normalized_function <- function(...) {
    return(f(...) / normalizer)
  }
  return(normalized_function)
}
```

We can now implement the rejection sampling algorithm, using the constructor function. The function will also return the total number of generated proposals, which will be used in subtask c).

```r
sample_theta <- function(n, y) {
  #' Generate n theta samples from f function, given 4-vector y
  #' Returns a list with thetas key to $theta,
  #' and the number of tries keyed to $tries.
  f_density <- construct_f(y = y)
  found <- vector()
  tries <- 0
  while(length(found) < n) {
    # Number of samples that remain to be found
    remaining <- n - length(found)
    tries <- tries + remaining

    # These are proposed values that might be accepted...
    x <- runif(remaining)

    # ... with acceptance probability
    alpha <- f_density(x)

    # Append the values that get accepted
    u <- runif(remaining)
    success <- u <= alpha
    found <- c(found, x[success])
  }
  return(list(theta = found, tries = tries))
}
```

We can now plot the acceptance probability, α, as a function of θ. We will here use $\vec{y} = (125, 18, 20, 34)$, as in the provided dataset.

```r
y <- c(125, 18, 20, 34)
f_scaled_density <- construct_f(y = y)
df <- enframe(rnorm(1))
ggplot(data = df) + aes(x = value) +
  stat_function(
    fun = f_scaled_density,
    xlim = c(0, 1),
    mapping = aes(col = 'Acceptance probability'),
    geom = "area",
    fill = "gray"
  ) +
  geom_hline (
    yintercept = 1,
    mapping = aes(col = 'Uniform distribution')
  ) +
  scale_y_continuous(
    name = expression(alpha)
  ) +
  scale_x_continuous(
    name = expression(theta)
  ) +
  labs(
    caption = "Acceptance probability as a function of proposed parameter theta."
  )
```



Acceptance probability as a function of proposed parameter theta.

We can now observe that the rejection algorithm will sample values mostly within the interval $[0.5, 0.75]$. Realized samples will be investigated in the following section.

2. Posterior mean by Monte-Carlo integration

Now we sample ten million samples, θ_i, from the implemented rejection algorithm.

```r
M <- 10000000
theta_sampling_result <- sample_theta(n = M, y = y)
theta_samples <- enframe(theta_sampling_result$theta)
```

The posterior mean can be practically derived from the samples as

$$E\big[\theta \mid p(\theta) \sim \mathcal{U}(0,1)\big] \approx \frac{1}{M}\sum_{i=1}^{M} \theta_i$$

Or equivalently, in R

```r
thetaSampleMean <- mean(theta_samples$value)
```

In order to check the correctness of the rejection sampling algorithm, we can approximate the posterior mean by numerically solving the integral

$$E\big[\theta \mid p(\theta) \sim \mathcal{U}(0,1)\big] = \int_0^1 \theta \cdot f(\theta|\vec{y})\ \mathrm{d}\theta.$$

This is done by using the `integrate` R function

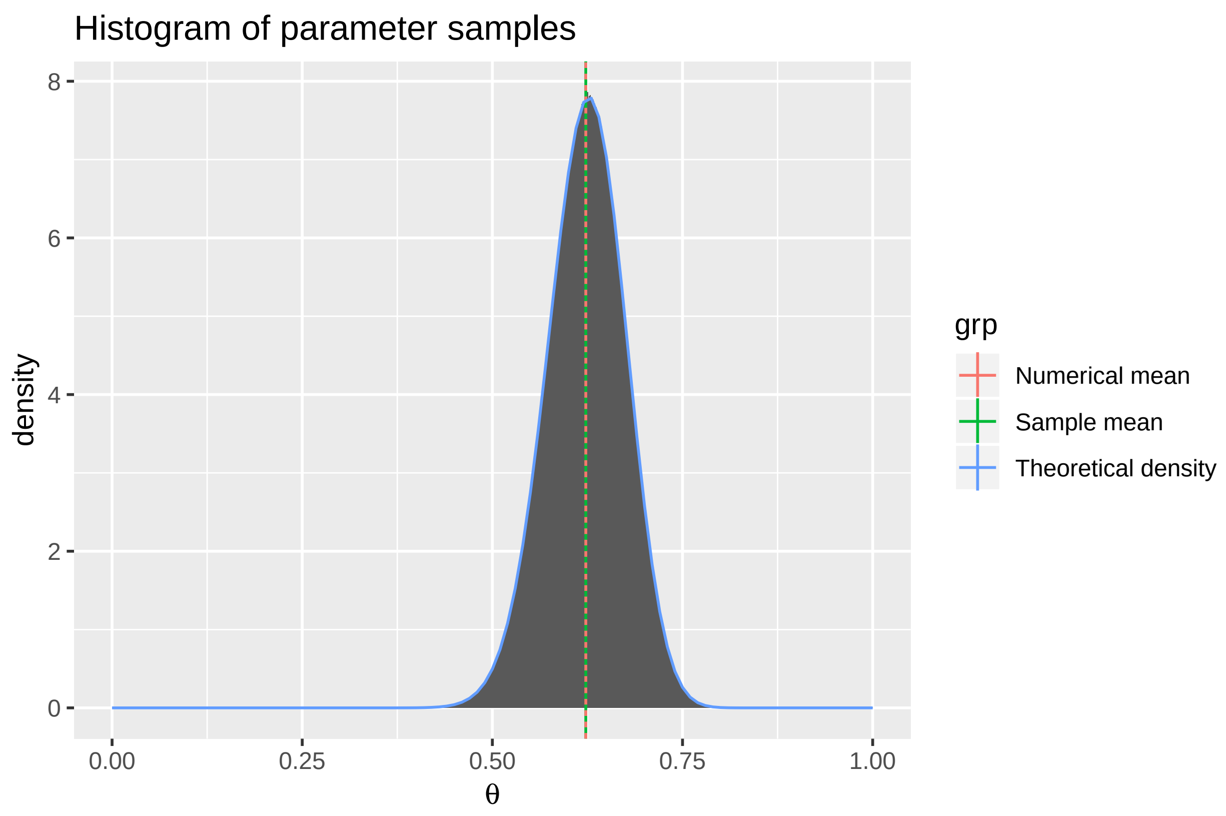
```r
f_density <- make_density(f_scaled_density)
thetaExpectedMean <- integrate(
  f = function(theta) theta * f_density(theta),
  lower = 0,
  upper = 1
)$value
```

All these results can now be shown in a comparison plot.

```r
means <- tibble(
  xint = c(thetaSampleMean, thetaExpectedMean),
  grp = c("Sample mean", "Numerical mean")
)
binWidth = 0.001
theta_samples %>%
  ggplot() +
  geom_histogram(
    mapping = aes(x = value, y = ..density..),
    binwidth = binWidth,
    boundary = 0,
    size = 0
  ) +
  geom_vline(
    data = means,
    aes(
      xintercept = xint,
      col = grp,
      linetype = c("dashed", "dotted")
    )
  ) +
  stat_function(
    fun = f_density,
    xlim = c(0, 1),
    aes(col = 'Theoretical density')
  ) +
  guides(linetype = FALSE) +
  scale_x_continuous(
    name = expression(theta),
    limits = c(0, 1)
  ) +
  labs(
    title = "Histogram of parameter samples",
    caption = "The histogram of the samples is colored in grey."
  )
```

Table 3: Table 2: Posterior mean calculation comparison

Posterior mean	Method
0.6228080	Sample mean
0.6228061	Numerical mean



The histogram of the samples is colored in grey.

The sample histogram, which is colored in grey, perfectly coincides with the theoretical density. This is what we would expect with ten million samples. The same can be said of the sample mean and numerical mean. We can therefore conclude that the rejection sampling algorithm has been correctly implemented.

The calculated posterior means are shown in the following table.

```r
means %>%
  kable(
    col.names = c("Posterior mean", "Method"),
    caption = "Table 2: Posterior mean calculation comparison"
  )
```

These values are close enough to conclude that the implementation is correct.

3. Required iterations for one sample

The overall acceptance rate for the rejection sampling algorithm is c^{-1}. We would therefore expect, on average, to generate c samples before one is accepted. Since $\mathcal{U}(0, 1)$ is used as the proposal density, we can numerically calculate c as

Table 4: Table 3: Required proposals for each accepted sample

Method	Required proposals
Sampling	7.800810
Theoretical	7.799308

$$c = \max_{\theta \in [0,1]} f(\theta|\vec{y})$$

We can compare this theoretical result with the numerical one, calculated earlier

```r
average_theta_tries <- theta_sampling_result$tries / M
cNumeric = optimize(
  f = f_density,
  interval = c(0, 1),
  maximum = TRUE
)$objective
attempts <- tibble(
  method = c("Sampling", "Theoretical"),
  attempts = c(average_theta_tries, cNumeric)
)
attempts %>%
  kable(
    caption = "Table 3: Required proposals for each accepted sample",
    col.names = c("Method", "Required proposals")
  )
```

The acceptance rate is close to the theoretical optimal. We have now confirmed both the validity and optimality of the implemented algorithm within the bounds of the assigned problem.

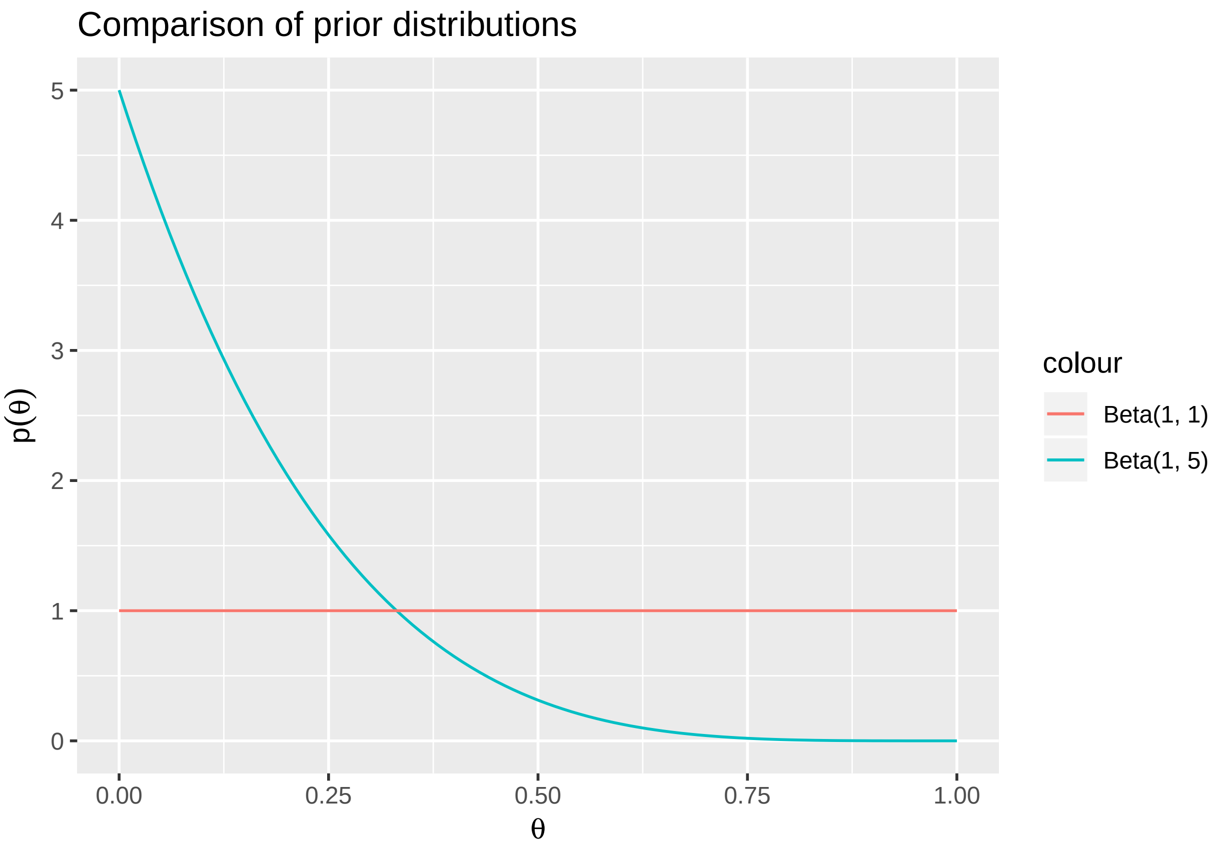
4. New prior

Previously, we used the prior of theta to be $p(\theta) \sim Beta(1,1) \equiv 1$. Now denote the posterior distribution for this prior as

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

Now we want to investigate the posterior mean under a new prior for θ, Beta(1, 5). We can compare these prior distributions as follows

```r
alpha <- 1
beta <- 5
tibble(x = c(0, 1)) %>%
  ggplot(aes(x)) +
  stat_function(
    fun = partial(dbeta, shape1 = alpha, shape2 = beta),
    aes(col = "Beta(1, 5)")
  ) +
  stat_function(
    fun = partial(dbeta, shape1 = 1, shape2 = 1),
    aes(col = "Beta(1, 1)")
  ) +
  scale_x_continuous(
    name = expression(theta)
  ) +
  scale_y_continuous(
    name = expression(p(theta)),
    limits = c(0, NA)
  ) +
  labs(
    title = "Comparison of prior distributions"
  )
```



As you can see, this new Beta(1, 5) prior heavily favours lower values for θ, compared to the uniform prior, which does not favour any domain in particular. We can therefore expect a lower posterior mean under the new prior.

The new posterior distribution is denoted as

$$f_{1,5}(\vec{y}|\theta) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3+\mathbf{4}}\theta^{y_4}$$

We can now use importance sampling weights in order to "resample" our existing samples from the old posterior distribution to the new one. $f_{1,1}$ is therefore our proposal density, and $f_{1,5}$ our target density. The importance sampling weight becomes

$$w(\theta_i) = \frac{f_{1,5}(\theta_i)}{f_{1,1}(\theta_i)} \propto (1-\theta)^4$$

In order to calculate the posterior mean under the new prior, we use the identity function as the objective function $h(\theta_i) = \theta_i$.

Since neither $f_{1,1}$ nor $f_{1,5}$ are normalized, we use self-normalizing importance sampling in order to calculate the new posterior mean

$$E\big[\theta \mid p(\theta) \sim Beta(1,5)\big] = \frac{\sum_{i=1}^{M} h(\theta_i) w(\theta_i)}{\sum_{i=1}^{M} w(\theta_i)}$$

We implement this in R, and "resample" the M previously generated samples.

```r
posteriorMean <- function(theta_samples, beta = 5) {
  weights <- (1 - theta_samples) ** (beta - 1)
  importanceThetaMean <- sum(theta_samples * weights) / sum(weights)
  return(importanceThetaMean)
}
importanceThetaMean <- posteriorMean(theta_samples = theta_samples$value)
```

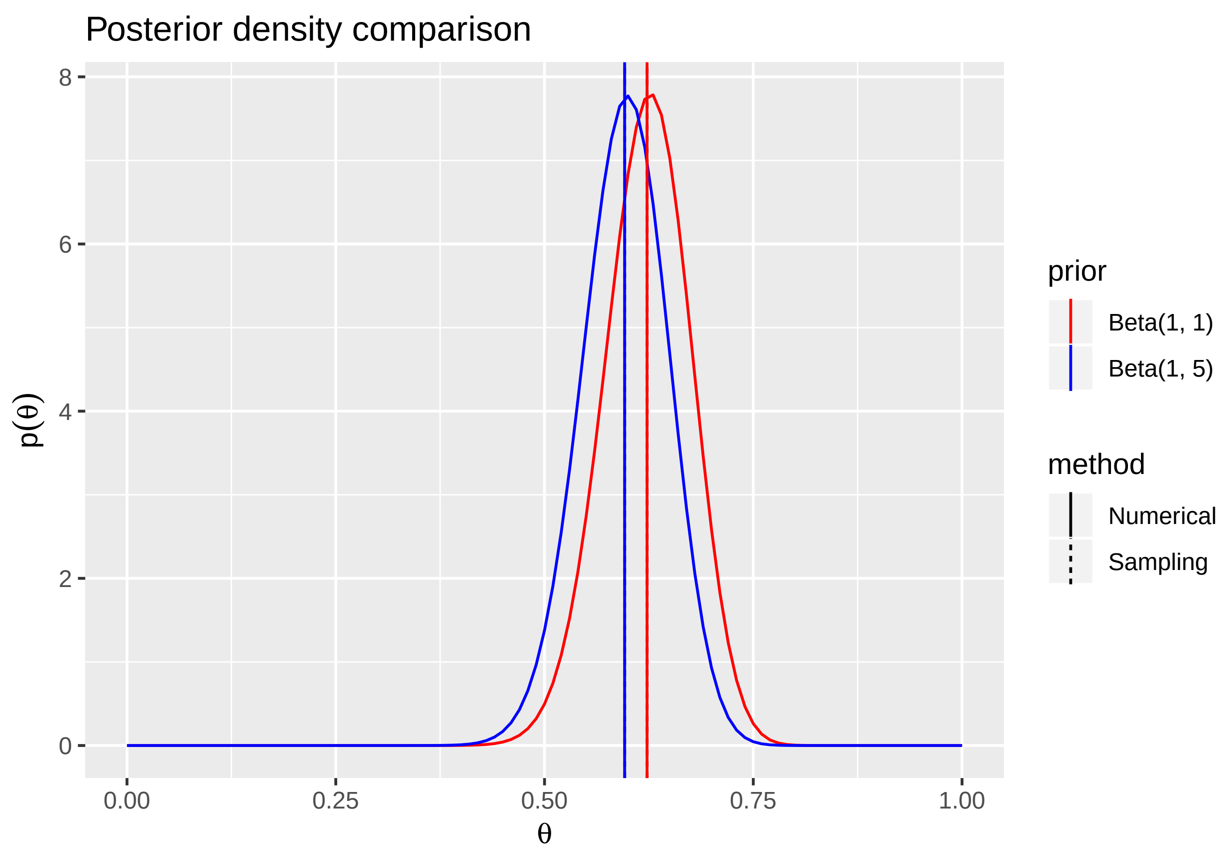
Again, for comparison, we implement the posterior densities for the old and new prior, and calculate the expected posterior mean by numerical integration.

$$E\big[\theta \mid p(\theta) \sim Beta(1,5)\big] = \int_0^1 \theta \cdot f(\theta|\vec{y})(1-\theta)^4 \, \mathrm{d}\theta.$$

```r
generatePosterior <- function(y, alpha = 1, beta = 5) {
  log_unscaled = function(theta) {
    y[1] * log(2 + theta) + (y[2] + y[3] + beta - 1) * log(1 - theta) + (y[4] + alpha - 1) * log(theta)
  }
  normalizingConstant <- integrate(
    f = function(theta) exp(log_unscaled(theta)),
    lower = 0,
    upper = 1
  )$value
  scaledPosterior <- function(theta) {
    return(exp(log_unscaled(theta)) / normalizingConstant)
  }
  return(scaledPosterior)
}
newPosterior <- generatePosterior(y = y, beta = 5)
newPosteriorIntegralMean <- integrate(
  f = function(theta) theta * newPosterior(theta),
  lower = 0,
  upper = 1
)$value

oldPosterior <- generatePosterior(y = y, beta = 1)
oldPosteriorIntegralMean <- integrate(
  f = function(theta) theta * oldPosterior(theta),
  lower = 0,
  upper = 1
)$value
```

We now have two different posterior densities, which are compared in the following plot

```r
meanResults <- tibble(
  posteriorMeans=c(
    mean(theta_samples$value),
    importanceThetaMean,
    oldPosteriorIntegralMean,
    newPosteriorIntegralMean
  ),
  prior = c(
    "Beta(1, 1)",
    "Beta(1, 5)",
    "Beta(1, 1)",
    "Beta(1, 5)"
  ),
  method = c(
    "Sampling",
    "Sampling",
    "Numerical",
    "Numerical"
  ),
  grp=c(
    "Old posterior asymptotic",
    "New posterior asymptotic",
    "Old posterior numerical integration",
    "New posterior numerical integration"
  )
)

ggplot(data = data.frame(x = c(0, 1))) +
  aes(x) +
  stat_function(
    fun = oldPosterior,
    xlim = c(0, 1),
    col = "red"
  ) +
  stat_function(
    fun = newPosterior,
    xlim = c(0, 1),
    col = "blue"
  ) +
  geom_vline(
    data = meanResults,
    mapping = aes(
      xintercept = posteriorMeans,
      col = prior,
      linetype = method
    )
  ) +
  scale_x_continuous(
    name = expression(theta)
  ) +
  scale_y_continuous(
    name = expression(p(theta))
  ) +
  scale_colour_manual(
    values = c("red", "blue")
  ) +
  labs(
    title = "Posterior density comparison"
  )
```

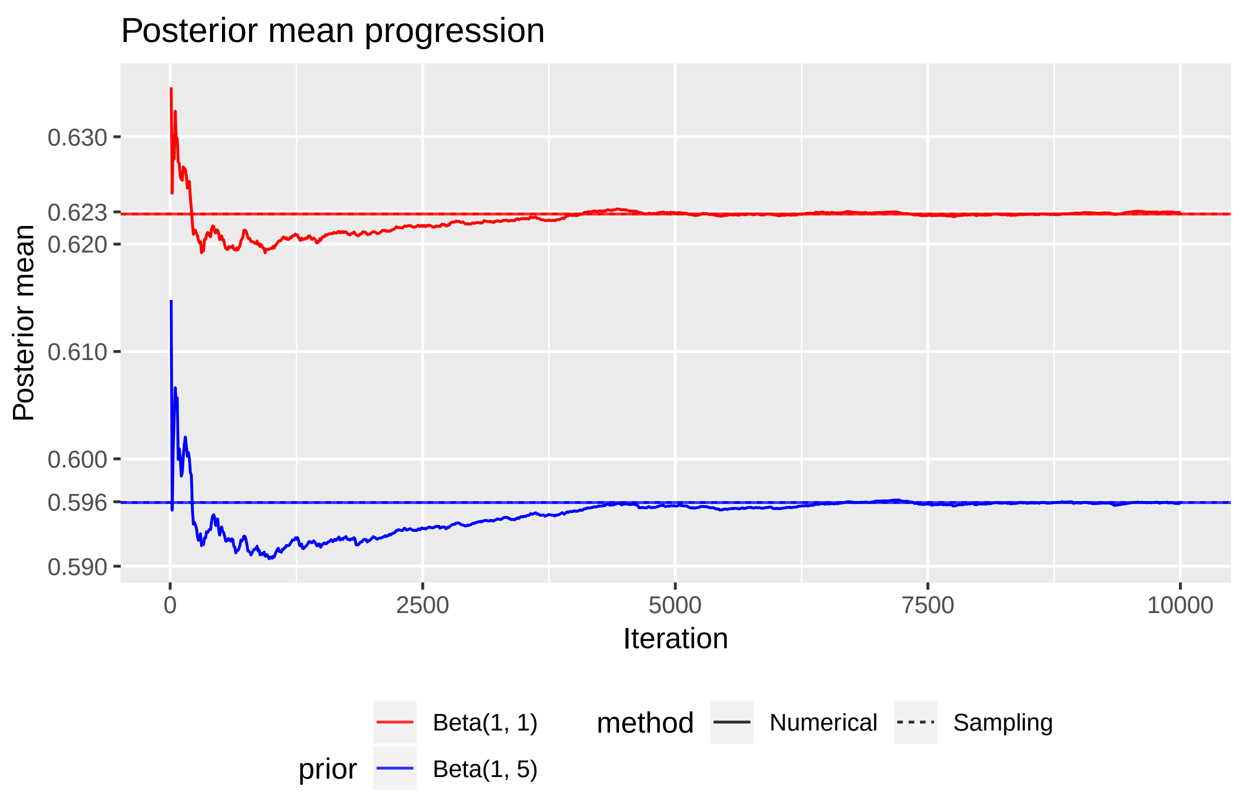


Observe that for both posterior distributions, the sample mean perfectly coincides with the theoretical numerical mean. The distribution under the new prior is also shifted to the left, as previously postulated.

The sampling mean converges towards the theoretical mean. The rate of convergence is visualized in the following plot.

```
# Take the first 10 thousand iterations with step size = 10
row_seq <- seq(10, 10000, 10)

# Subset the first k samples of theta under the uniform prior
thetaSubsets <- lapply(row_seq, function(k) theta_samples$value[1:k])

# Calculate the posterior mean under the old and new prior
meanProgression <- tibble(
  iteration = row_seq,
  oldPosterior = sapply(thetaSubsets, mean),
  newPosterior = sapply(thetaSubsets, posteriorMean)
)

# Plot the posterior mean progressions
meanProgression %>%
  ggplot(aes(x = iteration, posteriorMean)) +
  geom_line(
    aes(y = oldPosterior),
```

```r
    col = "red"
  ) +
  geom_line(
    aes(y = newPosterior),
    col = "blue"
  ) +
  geom_hline(
    data = meanResults,
    aes(
      yintercept = posteriorMeans,
      col = prior,
      linetype = method
    ),
    alpha = 0.8
  ) +
  theme(legend.position = "bottom") +
  guides(col=guide_legend(nrow=3, byrow = FALSE)) +
  ylab("Posterior mean") +
  xlab("Iteration") +
  scale_y_continuous(
    breaks = c(
      seq(0.59, 0.64, 0.01),
      round(oldPosteriorIntegralMean, digits = 3),
      round(newPosteriorIntegralMean, digits = 3)
    ),
    limits = c(NA, NA)
  ) +
  scale_colour_manual(
    values = c("red", "blue")
  ) +
  theme(
    panel.grid.minor.y = element_blank()
  ) + labs(
    title = "Posterior mean progression"
  )
```



In both cases, it seems to be enough with approximately 5 thousand samples in order to calculate a relatively accurate posterior mean for the θ parameter.

Rao, C. Radhakrishna. 1973. "Linear Statistical Inference and Its Applications." doi:10.1002/9780470316436.