Computer Intensive Statistical Methods - Exercise 1

Martin Outzen Berild, Sindre Henriksen

January 2018

Problem A

1. We consider the exponential distribution with parameter $\lambda > 0$, and wish to generate samples from this. The probability density function and cumulative density function are

$$f(x) = \lambda e^{-\lambda x} \quad \text{and} \quad F(x) = 1 - e^{-\lambda x}\,, \tag{1}$$

respectively. Let $F(x) = u$. The inverse F^{-1} of F is given by

$$x = F^{-1}(u) = -\frac{1}{\lambda}\log(1-u)\,. \tag{2}$$

Using the inverse and letting $u \sim \mathcal{U}[0,1]$ (and so also $1-u \sim \mathcal{U}[0,1]$), we generate a vector of n samples and compare it with the exact exponential distribution in the code below.

```
# Simulate n values from Exp(lambda)
r_exp <- function(n, lambda) {
  exp_dist = - log(1-runif(n))/ lambda
  return(exp_dist)
}
```

Next we test `r_exp()`.

```
set.seed(123)

# Params
lambda = 0.5
n = 10000

# Simulate from r_exp and create a dataframe
samples = enframe(r_exp(n, lambda))

# Plot histogram of distribution together with PDF
ggplot(data = samples) +
  geom_histogram(
    aes(x = value, y = ..density..),
    bins = 50,
    colour = "white",
    fill = "cornflowerblue"
  ) +
  stat_function(
```

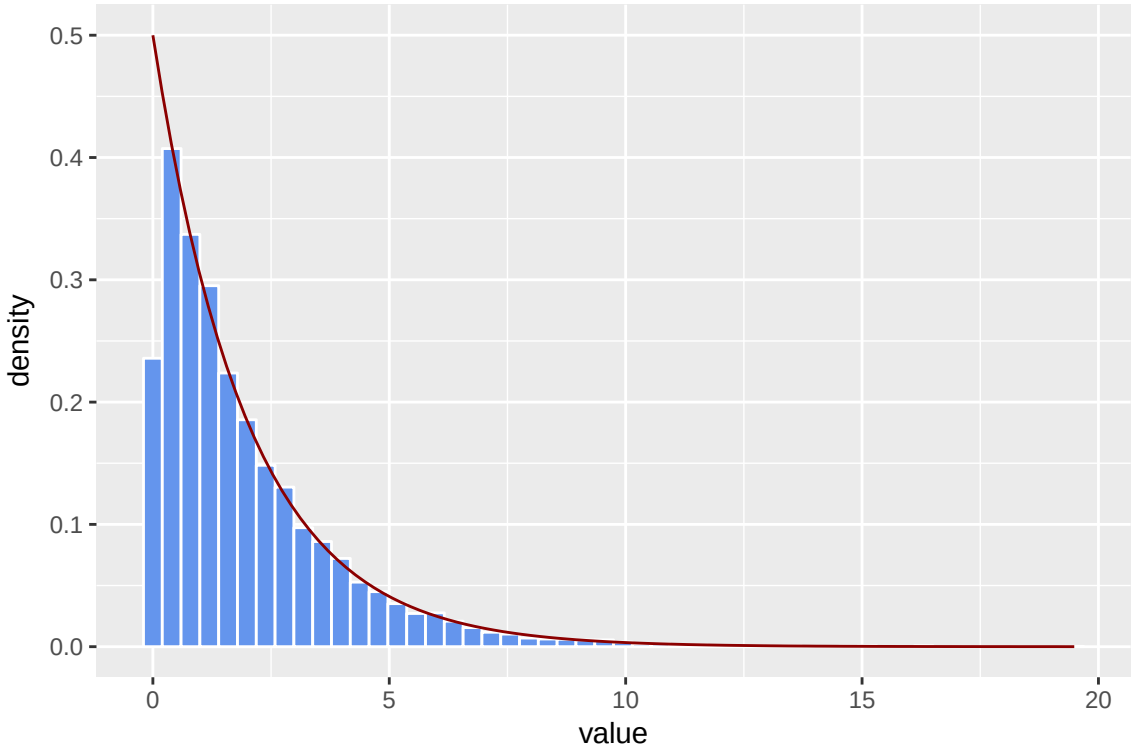

Figure 1: Theoretical density (red line) from a exponential distribution with parameter $\lambda = 0.5$ and frequency histogram of simulations.

```
    fun = function(x)
      dexp(x, lambda),
    color = "darkred"
  )
```

```
# Compare empirical means and variances with true values for different alphas
n_lambdas = 9
lambdas_start = 1
lambdas_stop = 50
empirical_means = empirical_vars = numeric(n_lambdas)
true_means = true_vars = numeric(n_lambdas)
lambdas = seq(lambdas_start,
              lambdas_stop,
              (lambdas_stop - lambdas_start) / (n_lambdas - 1))
for (i in 1:n_lambdas) {
  samples_i = enframe(x = r_exp(n, lambdas[i]))
  empirical_means[i] = mean(samples_i$value)
  empirical_vars[i] = var(samples_i$value)
  true_means[i] = 1 / lambdas[i]
  true_vars[i] = 1 / lambdas[i] ^ 2
}
max((empirical_means - true_means) / true_means)

## [1] 0.01167621
```

```r
max((empirical_vars - true_vars) / true_vars)
```

```
## [1] 0.03510261
```

In Figure 1 we can see that the histogram of the samples follow the exponential distribution using the same λ. The code also outputs the maximum relative error of the mean and variance compared to the true values λ^{-1} and λ^{-2}, respectively. As we can see, the errors are small.

2.

(a) We consider the density function $g(x)$ given in 2. The cumulative density function $G(x) = \int_{-\infty}^{x} g(x)\,\mathrm{d}x$ is

$$G(x) = \begin{cases} 0, & x \leq 0 \\ c\alpha^{-1}x^{\alpha}, & 0 < x < 1 \\ c(\alpha^{-1} + e^{-1} - e^{-x}), & 1 \leq x \end{cases}, \tag{3}$$

where $\alpha \in (0, 1)$. This gives the inverse of G(x)

$$G^{-1}(u) = \begin{cases} \left(\frac{u\alpha}{c}\right)^{\frac{1}{\alpha}}, & 0 < u \leq c\alpha^{-1} \\ \log\left[c(1-u)^{-1}\right], & c\alpha^{-1} < u \leq 1 \end{cases}. \tag{4}$$

We find the expected value $\mathbf{E}(x) = \int_0^{\infty} xg(x)\,\mathrm{d}x$:

$$\mathbf{E}(x) = c\left[\frac{1}{\alpha+1} + 2e^{-1}\right]. \tag{5}$$

The variance one can find from the equation $\mathrm{Var}(x) = \mathbf{E}(x^2) - \left[\mathbf{E}(x)\right]^2$. Then by calculating $\mathbf{E}(x^2)$ the same way as the expected value in (5), we get

$$\mathrm{Var}(x) = c\left[\frac{1}{\alpha+2} + 5e^{-1}\right] - c^2\left[\frac{1}{\alpha+1} + 2e^{-1}\right]. \tag{6}$$

The normalizing constant c is given by solving $\int_0^{\infty} g(x)\,\mathrm{d}x = 1$ for c, which yields $c = \frac{e\alpha}{e+\alpha}$.

(b) We wish to generate samples from g and compare it to the theoretical distribution. The function `rg()` below generates a vector of n independent samples from g with $\alpha \in (0, 1)$.

```r
# PDF g
g = function(x, alpha) {
  stopifnot(0 < alpha && alpha < 1)
  c = c_func(alpha)
  x = ifelse(0 < x,
             ifelse(x < 1, c * x ^ (alpha - 1), c * exp(-x)),
             0)
  return(x)
}
```

```r
# Simulate n values from g with 0 < alpha < 1
rg = function(n, alpha) {
  stopifnot(0 < alpha && alpha < 1)
  c = c_func(alpha)
  u = runif(n)
  x = numeric(n)
  x = ifelse((u < c / alpha), (u * alpha / c) ^ (1 / alpha),
             log(c / (1- u)))
  return(x)
}
```

Next we test `rg()`.

```r
set.seed(123)

# Params
alpha = 0.5
n = 100000

# Simulate from rg() and create a dataframe
samples = tibble(x = rg(n, alpha))

# Plot histogram of distribution together with PDF
ggplot(data = samples) +
  geom_histogram(
    aes(x = x, y = ..density..),
    bins = 100,
    colour = "white",
    fill = "cornflowerblue"
  ) +
  stat_function(
    fun = function(x)
      (g(x, alpha)),
    color="darkred",
    xlim=c(0.05, max(samples$x)))
```

```r
# Compare empirical means and variances with theoretical for different alphas
empirical_means = empirical_vars = numeric(9)
true_means = true_vars = numeric(9)
alphas = seq(0.1, 0.9, 0.1)
for (i in 1:9) {
  samples_i = tibble(x = rg(n, alphas[i]))
  empirical_means[i] = mean(samples_i$x)
  empirical_vars[i] = var(samples_i$x)
  true_means[i] = c_func(alphas[i]) * (1 / (alphas[i] + 1) +
                                         2 * exp(-1))
  true_vars[i] = c_func(alphas[i]) * (1 / (alphas[i] + 2) +
                                        5 * exp(-1)) - true_means[i] ^2
}
max((empirical_means - true_means) / true_means)
```

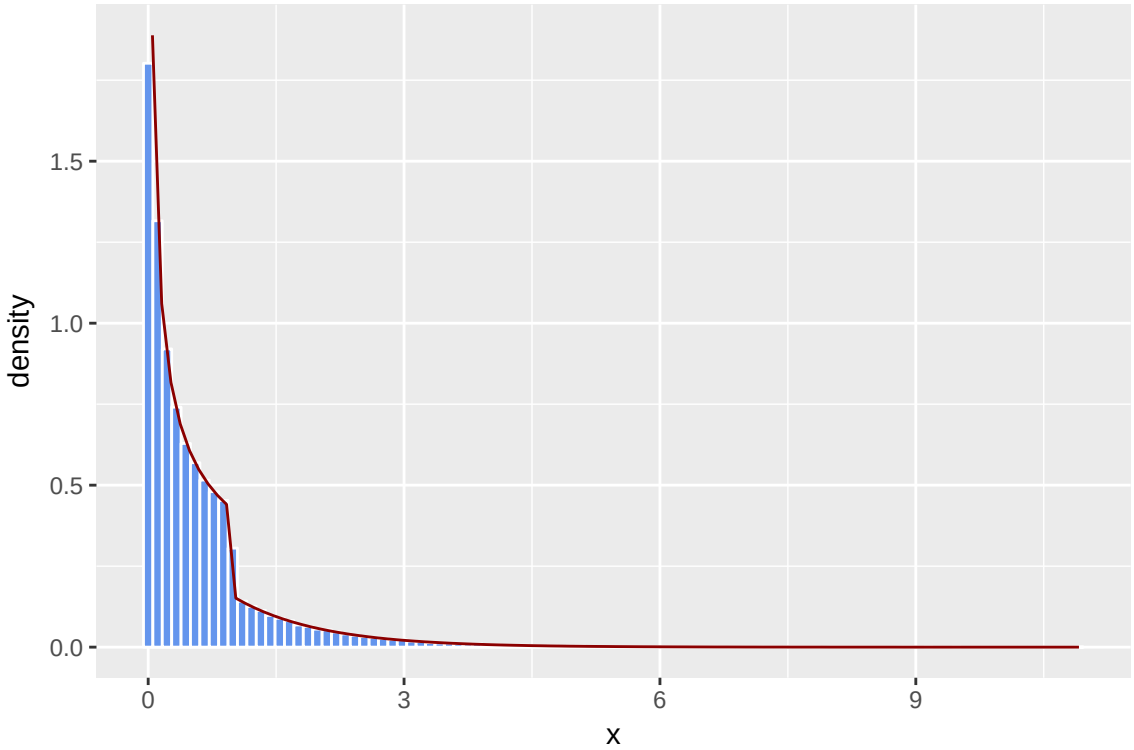

Figure 2: Theoretical density (red line) from the $g(x)$ distribution with $\alpha = 0.5$ and frequency histogram of simulations.

```
## [1] 0.010662

max((empirical_vars - true_vars) / true_vars)

## [1] 0.02929972
```

In Figure 2 we see that the histogram height follows the theoretical distribution of g(x) using $\alpha = 0.5$. Also the code test of `rg()` outputs the maximum relative error and variance using different values of α. The maximum error is less than 3%, which indicates that the samples are indeed from the distribution $g(x)$.

3. We now wish to generate n samples of two standard normal distribution using the Box-Muller algorithm, which consists of generating two uniformly distributed variables in polar coordinates and then transforming them into Cartesian coordinates. We will check if these are indeed then normal distributed.

```
# Simulate n values from the standard normal distribution using the box muller
# algorithm
r_boxmuller = function(n) {
  x1 = 2 * pi * runif(n)
  x2 = -2 * log(runif(n))
  sample_box = data.frame(
    y1 = sqrt(x2) * cos(x1),
    y2 = sqrt(x2) * cos(x1),
    x1 = x1,
```

```
    x2 = x2
  )
  return(sample_box)
}
```

Next we test `r_boxmuller()`.

```
set.seed(123)

# Params
n <- 50000

# Simulate and calculate empirical and true means and variances
samples = r_boxmuller(n)
empirical_mean = mean(samples$y1)
empirical_var = var(samples$y1)

# Creating ggplot variable of a histrogram of one of the samles and a normal
# distribution N(0,1)
norm1 = ggplot(data = samples) +
  geom_histogram(
    aes(x = y1, y = ..density..),
    bins = 40,
    colour = "white",
    fill = "cornflowerblue"
  ) +
  stat_function(
    fun = function(x)
      dnorm(x, 0, 1),
    color = "darkred"
  )

# Creating ggplot variable of a histrogram of the other samle and a normal
# distribution N(0,1)
norm2 = ggplot(data = samples) +
  geom_histogram(
    aes(x = y2, y = ..density..),
    bins = 40,
    colour = "white",
    fill = "cornflowerblue"
  ) +
  stat_function(
    fun = function(x)
      dnorm(x, 0, 1),
    color = "darkred"
  )

# Plotting the ggplot variables next to each other in columns
multiplot(norm1, norm2, cols = 2)
```

In Figure 3 we see that the frequency plot of the two sample follow the theoretical standard nor-

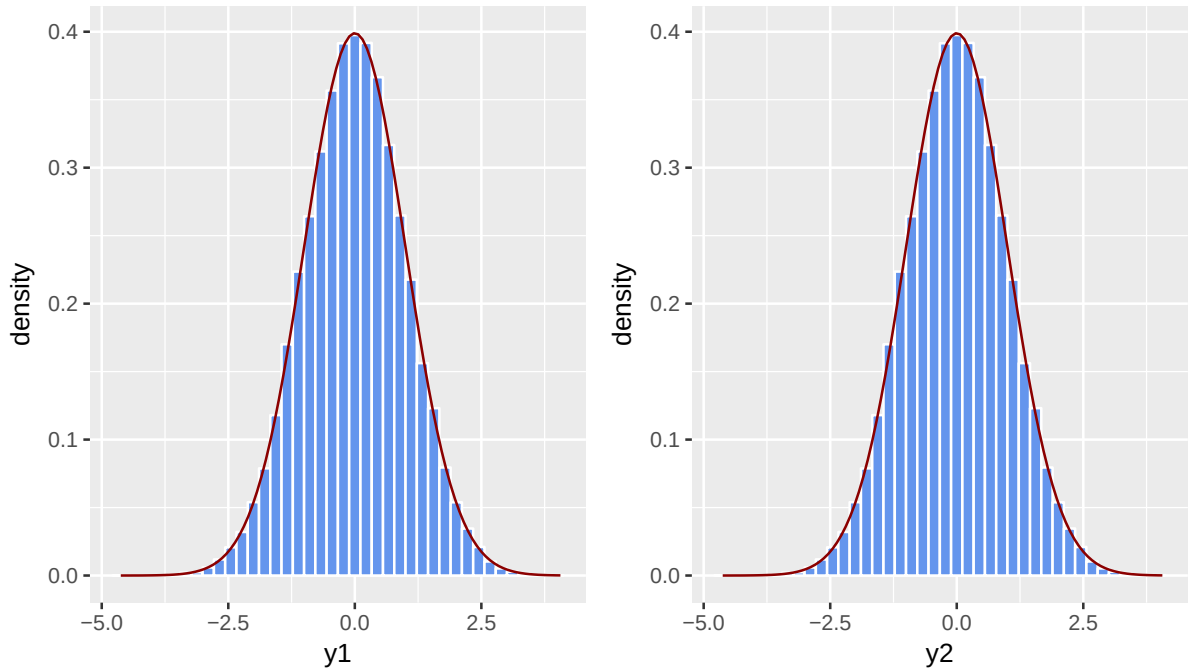

Figure 3: Theoretical density (red lines) from the standard normal distribution with an frequency histogram of a simulation using the Box-Muller algorithm.

mal distribution $\mathcal{N}[0,1]$. The empirical mean and variance are -0.0018 and 0.9966, respectively. We conclude that the Box-Muller algorithm successfully generates samples from a standard normal distribution.

4. We wish to generate n samples from a d-variate normal distribution with a given mean μ and covariance Σ. We use the Box-Muller algorithm to generate n samples of d standard normal distribution ($\mathbf{x} \sim \mathcal{N}_d[0,1]$). Then we use that, if $AA^T = \Sigma$, $\mathbf{y} = \boldsymbol{\mu} + A\mathbf{x} \sim \mathcal{N}_d[\boldsymbol{\mu}, \Sigma]$.

We set $\mu \sim \mathcal{U}_d[0,10]$ and A to be a $(d \times d)$ matrix of uniformly distributed elements ($a_{ij} = \mathcal{U}[0,1]$). Then we choose as covariance matrix $\Sigma = AA^T$, which ensures it is symmetric and positive definite.

```
# Simulate d x n values from a d-variate normal distribution
r_multinorm = function(n, d) {
  x = matrix(0, d, n)
  # Generating from r_boxmuller() in problem A3
  # Using both samples if necessary
  for (i in 1:d) {
    df = r_boxmuller(n)
    x[i, ] = t(df[, 1])
    if ((d - i) > 0) {
      i = i + 1
      x[i, ] = t(df[, 2])
    }
  }
  A = matrix(0, d, d)
  # Generating a matrix A
  for (i in 1:d) {
```

```
    A[, i] = runif(d)
  }
  mu = runif(d) * 10
  multinorm = list(y = mu + A %*% x,
                   true_mean = mu,
                   # Creating a positive definite covariance
                   true_cov = A %*% t(A))
  return(multinorm)
}
```

Next we test `r_multinorm()`.

```
set.seed(123)

# Params
d = 5
n = 10000

# Simulate and calculate empirical and true means and variances
samples = r_multinorm(n, d)
empirical_mean = rowMeans(samples$y)
empirical_cov = cov(t(samples$y))
max((empirical_mean - samples$true_mean) / samples$true_mean)

## [1] 0.006095029

max((empirical_cov - samples$true_cov) / samples$true_cov)

## [1] -0.0004305788
```

We see from the outputs of the maximum relative error of the mean and covariances, that the algorithm works as expected; the errors are very small.

Problem B

1. We consider a Gamma distribution with parameters $\alpha \in (0, 1)$ and $\beta = 1$, i.e.

$$f(x) = \begin{cases} \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}, & 0 < x, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

(a) We wish to generate samples from the distribution given in equation (7) using rejection sampling with $g(x)$ (from problem A) as proposal density. We need to find $k \geq 1$ s.t. $f(x) \leq k \cdot g(x)$. In addition to the trivial case $x \leq 0$ (where $f(x) = g(x) = 0$, there are two cases:

$$\begin{aligned} 0 < x < 1: \quad & \frac{e^{-x}}{\Gamma(\alpha)c} \leq \frac{1}{\Gamma(\alpha)c} = \frac{e+\alpha}{\Gamma(\alpha+1)e} \leq k \\ x \geq 1: \quad & \frac{x^{\alpha-1}}{\Gamma(\alpha)c} \leq \frac{1}{\Gamma(\alpha)c} = \frac{e+\alpha}{\Gamma(\alpha+1)e} \leq k\,, \end{aligned}$$

where $c = e\alpha/(e+\alpha)$, as before. We set k as small as possible to maximize the overall acceptance probability k^{-1}, i.e. $k = [\Gamma(\alpha)c]^{-1}$.

(b) The function `r_gamma1()` below generates a vector of n independent samples from f with $\alpha < 1$ and $\beta = 1$.

```
# k is 1/overall acceptance probability
k = function(alpha) {
  (exp(1) + alpha) / (gamma(alpha + 1) * exp(1))
}
```

```
# PDF f
f_gamma1 = function(x, alpha) {
  ifelse(x <= 0, 0, x ^ (alpha - 1) * exp(-x) / gamma(alpha))
}
```

```
# Simulate n values from f with 0 < alpha < 1
r_gamma1 = function(n, alpha) {
  stopifnot(0 < alpha && alpha < 1)
  k = k(alpha)
  xs = numeric(n)
  n_accepted = 0
  while (n_accepted < n) {
    x = rg(1, alpha)
    acceptance_level = f_gamma1(x, alpha) / (g(x, alpha) * k)
    u = runif(1)
    if (u <= acceptance_level) {
      n_accepted = n_accepted + 1
      xs[n_accepted] = x
    }
  }
  return(xs)
}
```

Next we test `r_gamma1()`.

```
set.seed(123)

# Params
alpha = 0.5
n = 10000

# Simulate and calculate empirical and true means and variances
samples = tibble(x = r_gamma1(n, alpha))
empirical_mean = mean(samples$x)
empirical_var = var(samples$x)
true_mean = true_var = alpha

# Plot histogram of distribution together with PDF
ggplot(data = samples) +
  geom_histogram(
    aes(x = x, y = ..density.., col = 0),
    bins = 50,
    colour = "white",
```

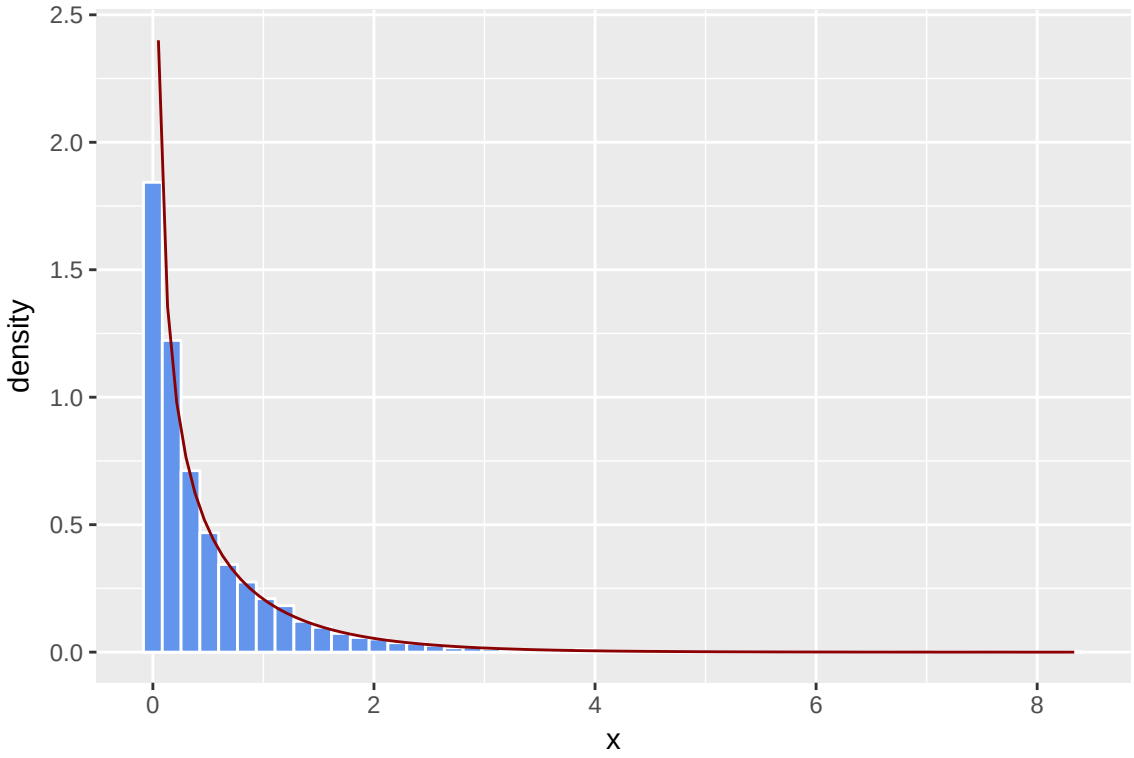

Figure 4: Theoretical density (red line) from a Gamma distribution with parameters $\alpha = 0.5$ and $\beta = 1$ and frequency histogram of simulations.

```
    fill = "cornflowerblue"
  ) +
  stat_function(
    fun = function(x)
      f_gamma1(x, alpha),
    colour = "darkred",
    xlim = c(0.05, max(samples$x))
  )
```

```
# Compare empirical means and variances with true values for different alphas
empirical_means = empirical_vars = numeric(9)
alphas = seq(0.1, 0.9, 0.1)
for (i in 1:9) {
  x = r_gamma1(n, alphas[i])
  empirical_means[i] = mean(x)
  empirical_vars[i] = var(x)
}
true_means = true_vars = alphas
max((empirical_means - true_means) / true_means)

## [1] 0.02904358

max((empirical_vars - true_vars) / true_vars)

## [1] 0.04186997
```

We see in figure 4 that the density from the simulations matches f almost perfectly. Also, with $\alpha = 0.5$, the empirical mean and variance are 0.4997 and 0.4979, respectively, which is close to the theoretical value α. Simulating with 9 different values of α the greatest relative error of the empirical means and variances lie within a few percent.

2. We now consider a Gamma distribution with parameters $\alpha > 0$ and $\beta = 1$ from (7). We wish to generate samples from the distribution using the ratio of uniforms method. That is, we sample uniformly from

$$C_f = \left\{ (x_1, x_2) : 0 \leq x_1 \leq \sqrt{f^*\left(\frac{x1}{x2}\right)} \right\} \quad \text{where} \quad f^*(x) = \begin{cases} x^{\alpha-1}e^{-x}, & 0 < x \\ 0, & \text{otherwise.} \end{cases}$$

We know that $C_f \subset [0, a] \times [b_-, b_+]$, where

$$a = \sqrt{\sup_x f^*(x)} = \left(\frac{\alpha - 1}{e}\right)^{\frac{\alpha-1}{2}}, \quad b_- = -\sqrt{\sup_{x \leq 0} x^2 f^*(x)} = 0 \text{ and}$$
$$b_+ = \sqrt{\sup_{x \geq 0} x^2 f^*(x)} = \left(\frac{\alpha + 1}{e}\right)^{\frac{\alpha+1}{2}}.$$

This can be utilized by sampling uniformly on $[0, a] \times [b_-, b_+]$ and accepting only values $(x_1, x_2) \in C_f$. Then $y = x_1/x_2$ will be a sample from the desired distribution f.

The problem is that we will get very big numerical values in intermediate steps for large α, so we have to be careful and implement a logarithmic version. We let $(y_1, y_2) = (\log x_1, \log x_2)$ and sample from y_1 and y_2 by letting u_1 and u_2 be uniform on $(0, 1)$, so that

$$y_1 = \log a + \log u_1 = \frac{\alpha - 1}{2} \log \frac{\alpha - 1}{e} + \log u_1 \quad \text{and}$$
$$y_1 = \log b_+ + \log u_2 = \frac{\alpha + 1}{2} \log \frac{\alpha + 1}{e} + \log u_2$$

are distributed as desired, i.e. such that $(x_1, x_2) = (\exp y_1, \exp y_2)$ are distributed uniformly on $[0, a] \times [b_-, b_+]$. Now we accept (x_1, x_2) for which

$$\log x_1 \leq 1/2 \cdot \log f^*(x_2/x_1) = \frac{1}{2}\left[(\alpha - 1)(\log x_2 - \log x_1) - \frac{x_2}{x_1}\right].$$

The algorithm is implemented in the following functions.

```
# Function sqrt(f*(x2/x1)), logarithmic implementation
log_sqrt_f_star = function(log_x1, log_x2, alpha) {
  0.5 * ((alpha - 1) * (log_x2 - log_x1) - exp(log_x2 - log_x1))
}
```

```
# Simulate n values from f with alpha > 1
r_gamma2 = function(n, alpha) {
  stopifnot(alpha > 1)
  xs = numeric(n)
  n_accepted = 0
  n_tries = 0
  while (n_accepted < n) {
    n_missing = n - n_accepted
```

```
    u1 = runif(n_missing)
    u2 = runif(n_missing)
    log_x1 = (alpha - 1) / 2 * log((alpha - 1) / exp(1)) + log(u1)
    log_x2 = (alpha + 1) / 2 * log((alpha + 1) / exp(1)) + log(u2)
    inside = log_x1 <= log_sqrt_f_star(log_x1, log_x2, alpha)
    n_inside = sum(inside)
    if (n_inside > 0) {
      xs[(n_accepted + 1):(n_accepted + n_inside)] = exp(log_x2[inside] -
                                                         log_x1[inside])
      n_accepted = n_accepted + n_inside
    }
    n_tries = n_tries + n_missing
  }
  return(list("x" = xs, "n_tries" = n_tries))
}
```

Next we test `r_gamma2()`.

```
set.seed(123)

# Params
alpha = 5
n = 10000

# Simulate and calculate empirical and true means and variances
sim = r_gamma2(n, alpha)
samples = tibble(x = sim$x)
empirical_mean = mean(samples$x)
empirical_var = var(samples$x)
true_mean = true_var = alpha

# Plot histogram of distribution together with PDF
ggplot(data = samples) +
  geom_histogram(
    aes(x = x, y = ..density..),
    bins = 50,
    colour = "white",
    fill = "cornflowerblue"
  ) +
  stat_function(
    fun = function(x)
      dgamma(x, shape = alpha),
    colour = "darkred"
  )
```

```
# Compare empirical means and variances with true values for different alphas
n = 1000
alphas = c(1.1, 2, 5, 10, 20, 35, 50, 75, 100, 200, 350, 500, 750, 1000, 1250,
           1500, 1750, 2000)
empirical_means = empirical_vars = n_tries = numeric(length(alphas))
```

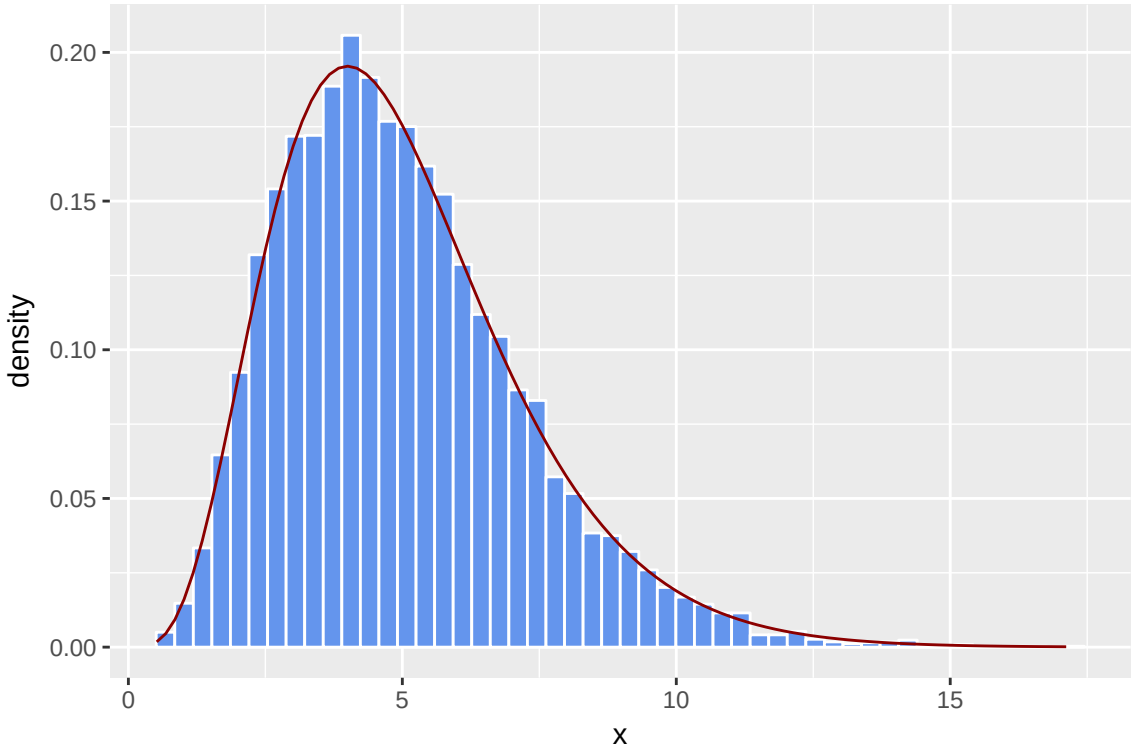

Figure 5: Theoretical density (red line) from a Gamma distribution with parameters $\alpha = 5$ and $\beta = 1$ and frequency histogram of simulations.

```
for (i in 1:length(alphas)) {
  sim_i = r_gamma2(n, alphas[i])
  n_tries[i] = sim_i$n_tries
  x = sim_i$x
  empirical_means[i] = mean(x)
  empirical_vars[i] = var(x)
}
true_means = true_vars = alphas
max((empirical_means - true_means) / true_means)

## [1] 0.0289285

max((empirical_vars - true_vars) / true_vars)

## [1] 0.07685994

# Plot number of tries needed to generate 1000 realizations versus alpha
ggplot(data = tibble(x = alphas, y = n_tries)) +
  geom_point(aes(x = x, y = y)) + xlab("alpha") + ylab("tries")
```

```
# Log-log plot of number of tries versus alpha
ggplot(data = tibble(x = alphas, y = n_tries), aes(x = x, y = y)) +
  geom_point() +
  stat_function(
```

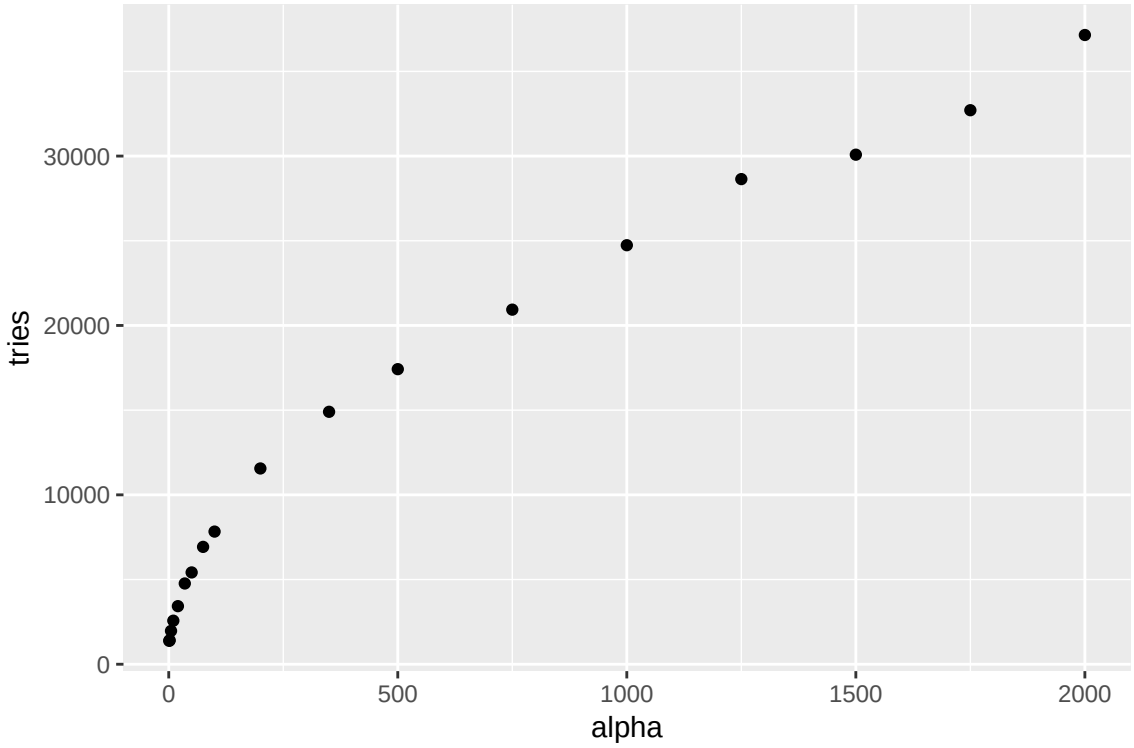

Figure 6: Number of tries needed to generate 1000 realizations from a Gamma distribution with $\beta = 1$ as a function of alpha.

```
    fun = function(x)
      (log10(500 * sqrt(x))),
    colour = "darkred",
    geom = "line"
  ) +
  scale_x_log10(
    breaks = scales::trans_breaks("log10", function(x)
      (10 ^ x)),
    labels = scales::trans_format("log10", scales::math_format(10 ^ .x))
  ) +
  scale_y_log10(
    breaks = scales::trans_breaks("log10", function(x)
      (10 ^ x)),
    labels = scales::trans_format("log10", scales::math_format(10 ^ .x))
  ) +
  annotation_logticks() +
  xlab("alpha") + ylab("tries")
```

We see in figure 5 that the density from the simulations matches f almost perfectly. Also, with $\alpha = 5$, the empirical mean and variance are 4.9804 and 4.8742, respectively, which is close to the theoretical value α. Simulating with different values of α, including large ones to test for numerical stability, the greatest relative error of the empirical means and variances lie within a few percent.

Figure 6 shows a plot of the number of tries needed to simulate 1000 realizations for different $\alpha \in (1, 2000]$. The number of tries increases rapidly for small increasing α. This indicates the

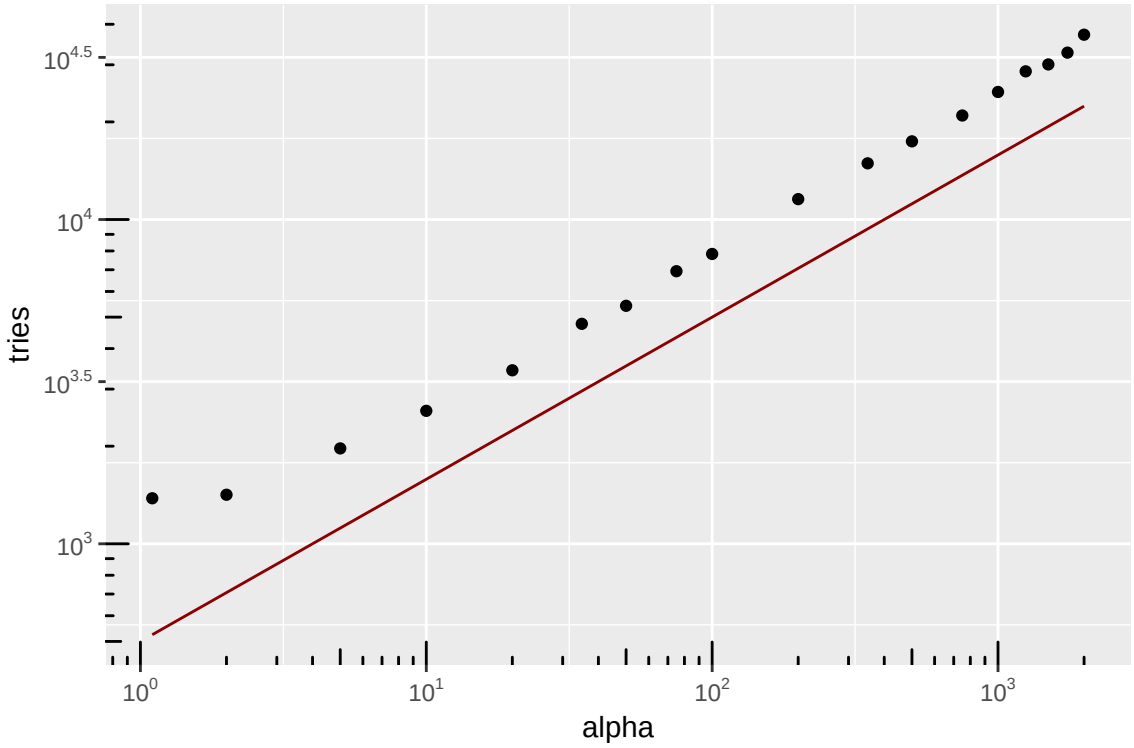

Figure 7: Log-log plot of the number of tries needed to generate 1000 realizations from a Gamma distribution with $\beta = 1$ as a function of alpha. The red line is a reference line with $y = constant \cdot x^{1/2}$.

relative size of C_f compared to $[0, a] \times [b_-, b_+]$ decreases rapidly. The second derivative of the number of tries needed is negative though, i.e. the number flattens out while α increases. In fact it seems like for large α the number of tries increases with the square root of α, which is illustrated by the log-log plot in figure 7.

3. The parameter β in a Gamma distribution

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

$\alpha, \beta > 1$, is an inverse scale parameter. Thus, if $X \sim \text{Gamma}(\alpha, 1)$, then $X/\beta \sim \text{Gamma}(\alpha, \beta)$. Furthermore $\text{Gamma}(1, 1) = \text{Exp}(1)$. So we can simulate from an arbitrary Gamma distribution using the following function.

```
# Simulate n values from a gamma distribution
r_gamma = function(n, alpha, beta) {
  stopifnot(alpha > 0 && beta > 0)
  if (alpha < 1) {
    x = r_gamma1(n, alpha)
  }
  else if (alpha == 1) {
    x = r_exp(n, lambda = 1)
  }
  else {
```

```
    x = r_gamma2(n, alpha)$x
  }
  return(x / beta)
}
```

The following code illustrates that it works as expected.

```
set.seed(123)

# Params
alpha = beta = 10
n = 10000

# Simulate and calculate empirical and true means and variances
samples = tibble(x = r_gamma(n, alpha, beta))
empirical_mean = mean(samples$x)
empirical_var = var(samples$x)
true_mean = alpha / beta
true_var = alpha / beta ^ 2

# Plot histogram of distribution together with PDF
ggplot(data = samples) +
  geom_histogram(
    aes(x = x, y = ..density..),
    bins = 50,
    colour = "white",
    fill = "cornflowerblue"
  ) +
  stat_function(
    fun = function(x)
      dgamma(x, shape = alpha, rate = beta),
    colour = "darkred"
  )
```

```
# Compare empirical means and variances with true values for different alphas
empirical_means = empirical_vars = numeric(25)
true_means = true_vars = numeric(25)
alphas = c(0.5, 1, 10, 100, 2000)
betas = c(0.5, 2, 5, 20, 50)
for (i in 1:25) {
  a = ((i - 1) %% 5) + 1
  b = ceiling(i / 5)
  x = r_gamma(n, alphas[a], betas[b])
  empirical_means[i] = mean(x)
  empirical_vars[i] = var(x)
  true_means[i] = alphas[a] / betas[b]
  true_vars[i] = alphas[a] / betas[b] ^ 2
}
max((empirical_means - true_means) / true_means)

## [1] 0.01635585
```

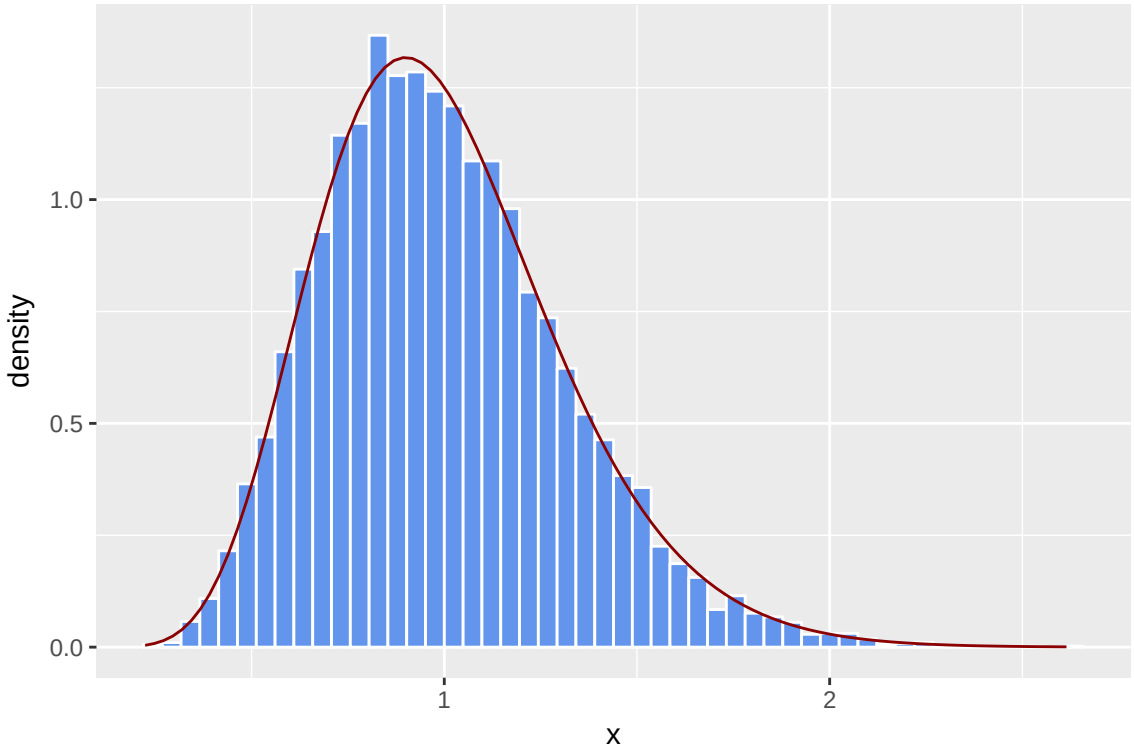

Figure 8: Theoretical density (red line) from a Gamma distribution with parameters $\alpha = 10$ and $\beta = 10$ and frequency histogram of simulations.

```
max((empirical_vars - true_vars) / true_vars)

## [1] 0.07215996
```

Figure 8 shows the density from the simulations matches f almost perfectly. With $\alpha = 10$ and $\beta = 10$, the empirical mean and variance are 0.9969 and 0.0985, respectively, which is close to the theoretical values $\alpha/\beta = 1$ and $\alpha/\beta^2 = 0.1$, respectively. Simulating with 25 different combinations of values of α and β, the greatest relative error of the empirical means and variances lie within a few percent.

Problem C

1. We want to show that x given in 1. has a Dirichlet distribution with parameter vector $\alpha = (\alpha_1, \alpha_2, ..., \alpha_k)$. We have that $z_k \sim \Gamma(\alpha_k, 1)$ for k = 1, 2, ..., K , and that $x_k = z_k/(z_1 + z_2 + ... + z_k)$. We want to do a change of variables from $(z_1, z_2, ..., z_k)$ to $(x_1, x_2, ..., x_{K-1}, v)$, with $v = z_1 + z_2 + ... + z_k$. After which we want to integrate out v, to get a distribution of x_k for $k = 1, 2, ..., K - 1$.

Using v we get $x_k = z_k/v$ or

$$z_k = v x_k \,. \tag{8}$$

Since we are doing the a change of variables of the form

$$f_{\mathbf{x}}(x_1, x_2, ..., x_{K-1}, v) = f_{\mathbf{z}}(z_1, z_2, ..., z_k)\,|J|\,, \tag{9}$$

we need to find $|J|$, the determinant of the Jacobian. We are given that $\sum_{k=1}^{K} x_k = 1$, so we

can write (8) as

$$[z_1, z_2, ..., z_K] = \left[vx_1, vx_2, ..., vx_{K-1}, v\left(1 - \sum_{k=1}^{K-1} x_k\right) \right] .$$

From this we calculate the determinant of the Jacobian

$$\begin{aligned} |J| &= \begin{vmatrix} dz_1/dx_1 & dz_1/dx_2 & \cdots & dz_1/dx_{K-1} & dz_1/dv \\ dz_2/dx_1 & dz_2/dx_2 & \cdots & dz_2/dx_{K-1} & dz_2/dv \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ dz_{K-1}/dx_1 & dz_{K-1}/dx_2 & \cdots & dz_{K-1}/dx_{K-1} & dz_{K-1}/dv \\ dz_k/dx_1 & dz_k/dx_2 & \cdots & dz_k/dx_{K-1} & dz_k/dv \end{vmatrix} \\ &= \begin{vmatrix} v & 0 & \cdots & 0 & x_1 \\ 0 & v & \cdots & 0 & x_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & v & X_{K-1} \\ -v & -v & \cdots & -v & 1 - \sum_{k=1}^{K-1} x_k \end{vmatrix} \\ &= v^{K-1} . \end{aligned}$$

To derive this determinant we used Laplace expansion on the first column, $K-2$ times, dealing with the products, and then calculating the determinant of 2x2 matrices. Since the variables are independent, the change of variables formula becomes

$$f_{\mathbf{x}}(x_1, x_2, ..., x_{K-1}, v) = \frac{\left(\prod_{k=1}^{K-1} x_k\right)\left(1 - \sum_{k=1}^{K-1} x_k\right)^{\alpha_K - 1}}{\prod_{k=1}^{K} \Gamma(\alpha_k)} v^{\left(\sum_{k=1}^{K} \alpha_k\right) - 1} e^{-v} .$$

Then we integrate out v to get the marginal of $(x_1, x_2, ..., x_{K-1})$, and we get

$$f_{\mathbf{x}}(x_1, x_2, ..., x_{K-1}) = \frac{\left(\prod_{k=1}^{K-1} x_k^{\alpha_k - 1}\right)\left(1 - \sum_{k=1}^{K-1} x_k\right)^{\alpha_K - 1}}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \int_0^\infty v^{\left(\sum_{k=1}^{K} \alpha_k\right) - 1} e^{-v} \, \mathrm{d}v .$$

The integral part is $\Gamma\left(\sum_{k=1}^{K} \alpha_k\right)$, so

$$f_{\mathbf{x}}(x_1, x_2, ..., x_{K-1}) = \frac{\Gamma\left(\sum_{k=1}^{K} \alpha_k\right)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \left(\prod_{k=1}^{K-1} x_k^{\alpha_k - 1}\right)\left(1 - \sum_{k=1}^{K-1} x_k\right)^{\alpha_K - 1} ,$$

which is the Dirichlet distribution we wanted to find.

2. We wish to generate one realization of the Dirichlet distribution. To do this we use `r_gamma()` K times to generate one sample from each of the Gamma distributions with parameters α_k. Using (8) and the samples we can simulate from the Dirichlet distribution. One can also calculate the theoretical mean and covariance, which is (with $A = \sum_{i=1}^{K} \alpha_k$)

$$E(x_k) = \frac{\alpha_k}{A},$$

and

$$\Sigma_{ij} = \begin{cases} \frac{-\alpha_i \alpha_j}{A^2(A+1)}, & i \neq j \\ \frac{\alpha_i}{A(A+1)} - \frac{\alpha_i^2}{A^2(A+1)}, & i = j \end{cases}.$$

```r
# Simulate n points in the dirichlet distribution of dimension K-1
r_dirichlet <- function(K, n, alpha, beta) {
  x = matrix(0, n, K - 1)
  z = matrix(0, n, K)
  # Samling from gamma distribution Gamma(alpha, beta = 1) in problem B
  for (k in 1:K) {
    z[, k] = r_gamma(n, alpha[k], beta)
  }
  x = z[, 1:(K - 1)] / rowSums(z)
  A = sum(alpha)
  true_cov = matrix(0, K - 1, K - 1)
  # Calulcating theoretical covariance
  for (i in 1:(K - 1)) {
    for (j in i:(K - 1)) {
      if (i!=j) {
        true_cov[i, j] = -alpha[i] * alpha[j] / (A ^ 2 * (A + 1))
        true_cov[j, i] = true_cov[i, j]
      } else{
        true_cov[i, j] = alpha[i] * (A - alpha[i]) / (A ^ 2* (A + 1))
      }
    }
  }
  return(
    # Returning a list of samples, empirical and theoretical mean/covariance
    list(
      x = x,
      empirical_mean = colMeans(x),
      empirical_cov = cov(x),
      # Calculating theoretical mean
      true_mean = alpha[1:(K - 1)] / A,
      true_cov = true_cov
    )
  )
}
```

Next we test `r_dirichlet()` by simulating from a Dirichlet distribution with five parameters.

```r
set.seed(123)

# Params
K = 5
n = 100000
# Generating alphas from a uniform distribution U[0,20]
alpha = runif(K)*20
# z_k ~ Gamma(alpha_k, 1) so setting beta = 1
beta = 1

# Simulate from the dirichlet distribution
samples = r_dirichlet(K,n,alpha,beta)
# Calculate the maximum relative error in mean and covariance.
max((samples$empirical_mean - samples$true_mean) / samples$true_mean)

## [1] 0.0004573807

max((samples$empirical_cov - samples$true_cov) / samples$true_cov)

## [1] 0.04224961
```

The code above the outputs the maximum relative error of the empirical mean and covariance. The errors are small, indicating that the samples are in fact Dirichlet distributed.

Problem D

We consider a vector $\mathbf{y} = (y_1, y_2, y_3, y_4)$ of multinomially distributed counts with probabilities $\mathbf{p} = (\frac{1}{2} + \frac{\theta}{4}, \frac{1-\theta}{4}, \frac{1-\theta}{4}, \frac{\theta}{4})$. The multinomial mass function is given by $f(y|\theta) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}$. The observed data is $y = (125, 18, 20, 34)$. First we assume a uniform prior on $(0, 1)$, resulting in a posterior density for θ

$$f(\theta|\mathbf{y}) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4} =: f^*(\theta|y)\,, \quad \theta \in (0,1)\,. \tag{10}$$

In this task we are mainly interested in the posterior mean $\mathbf{E}(\theta|\mathbf{y})$.

1. We wish to construct a rejection sampling algorithm to simulate from $f(\theta|\mathbf{y})$ using $\mathcal{U}(0,1)$ as the proposal density. This can be done exactly or approximately. To do it exactly we need to find $\max_\theta f^*(\theta)$. For the observed $\mathbf{y}$ we get

$$\frac{\mathrm{d}}{\mathrm{d}\theta} f^*(\theta|y) \propto -197\theta^2 + 15\theta + 68 = 0 \quad \Rightarrow \quad \theta_m \approx 0.62682150\,,$$

and $f^*(\theta_m) \approx 1.83884 \cdot 10^{29}$. In the implementation the maximum is found numerically. We also need $c = \int_0^1 f^*(\theta|y)\,\mathrm{d}\theta \approx 2.3577 \cdot 10^{28}$, which is found numerically. This gives the ideal inverse of the overall acceptance probability $k = \max_\theta f(\theta|\mathbf{y}) = f^*(\theta_m)/c \approx 7.780$ since the proposal density is $g(x) = 1$.

To implement the approximate algorithm described in lecture 4, k and c are not needed, since

$$w_i = \frac{f(x_i)}{g(x_i)} \Big/ \sum_{j=1}^n \frac{f(x_j)}{g(x_j)} = \frac{f^*(x_i)}{g(x_i)} \Big/ \sum_{j=1}^n \frac{f^*(x_j)}{g(x_j)}\,.$$

Both methods are implemented in the code shown below.

```r
# Posterior density f(theta|y) (up to a normalising constant)
f_posterior_star = function(theta, y) {
  stopifnot(all(0 < theta) && all(theta < 1))
  stopifnot(length(y) == 4)
  return((2 + theta) ^ y[1] * (1 - theta) ^ (y[2] + y[3]) * theta ^ y[4])
}
```

```r
# Simulate n values from f with 0 < alpha < 1
r_posterior = function(n, y) {
  f_star_max = -optim(0.5, function(x)
    (-f_posterior_star(x, y)),
    method = "L-BFGS-B", lower = 1e-10, upper = 1 - 1e-10)$value
  c = integrate(function(x)
    (f_posterior_star(x, y)),
    lower = 0,
    upper = 1)$value
  f_posterior = function(x)
    (f_posterior_star(x, y) / c)
  k = f_star_max / c  # 1 / overall acceptance probability
  xs = numeric(n)
  n_accepted = 0
  n_random_numbers = 0
  while (n_accepted < n) {
    n_missing = n - n_accepted
    x = runif(n_missing)
    acceptance_level = f_posterior(x) / k
    u = runif(n_missing)
    inside = u <= acceptance_level
    n_inside = sum(inside)
    if (n_inside > 0) {
      xs[(n_accepted + 1):(n_accepted + n_inside)] = x[inside]
      n_accepted = n_accepted + n_inside
    }
    n_random_numbers = n_random_numbers + n_missing
  }
  return(list("x" = xs, "n_random_numbers" = n_random_numbers))
}

r_posterior_approx = function(n, y) {
  m = 20 * n
  u = runif(m)
  f_over_g = f_posterior_star(u, y) / 1
  weights = f_over_g / sum(f_over_g)
  x = sample(u, size = n, prob = weights)
  return(list(x = x, n_random_numbers = m))
}
```

2. In the following code we sample 10000 times from $f(\theta|\mathbf{y})$ using both the exact and the approximate sampling method, estimate the mean and compare with the theoretical distribution and mean.

```r
set.seed(123)

# Params
n = 10000

# Simulate and calculate empirical and true means and variances
y = c(125, 18, 20, 34)
sim = r_posterior(n, y)
sim_approx = r_posterior_approx(n, y)
samples = tibble(x = sim$x, x_approx = sim_approx$x)
empirical_mean = mean(samples$x)
empirical_mean_approx = mean(samples$x_approx)

# Calculate integration constant and true mean numerically
c = integrate(function(x)(f_posterior_star(x, y)),
              lower = 0, upper = 1)$value
f_posterior = function(x)(f_posterior_star(x, y) / c)
true_mean = integrate(function(x)(x * f_posterior(x)),
                      lower = 0, upper = 1)$value

# Plot histogram of distribution together with PDF
ggplot(data = samples) +
  geom_histogram(
    aes(x = x, y = ..density.., col = 0),
    bins = 50,
    colour = "white",
    fill = "cornflowerblue",
    alpha = 0.5
  ) +
  geom_histogram(
    aes(x = x_approx, y = ..density.., col = 0),
    bins = 50,
    colour = "white",
    fill = "grey",
    alpha = 0.5
  ) +
  stat_function(
    fun = function(x)
      f_posterior(x),
    colour = "darkred",
    xlim = c(0.05, max(samples$x))
  ) +
  geom_vline(xintercept = empirical_mean,
             col = "red",
             size = 1) +
  geom_vline(xintercept = true_mean,
             col = "darkgreen",
             size = 0.5)
```

The results are illustrated in figure 9. The estimated means from the exact method and the approximate method are 0.6231 and 0.6226, respectively, close to the theoretical (numerically

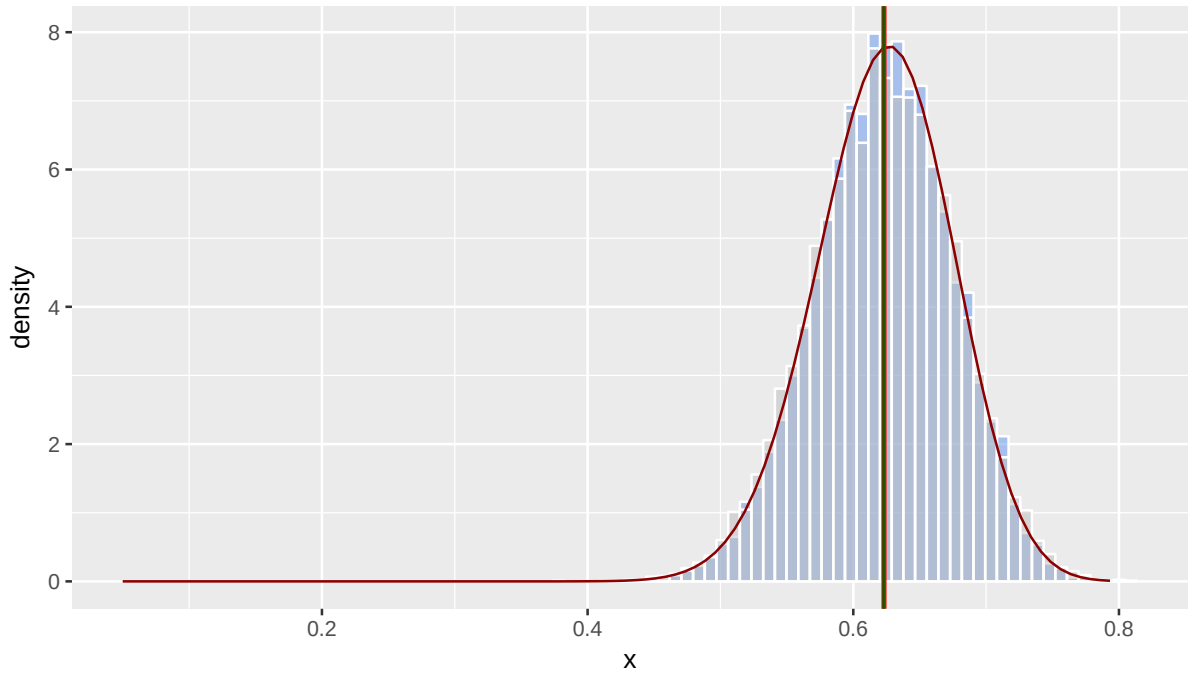

Figure 9: The red line is the theoretical posterior density $f(\theta|y)$ (see (10)). The bins are frequency histograms of simulations from an exact simulation algorithm (light blue) and an approximate algorithm (light grey). The colors are mixed where they overlap. The vertical red line is the theoretical mean, and the dark green line is the empirical mean from the exact algorithm.

computed) mean $\mu = 0.6228$. The exact algorithm gives samples matching the theoretical density especially well. The approximate density has a larger spread than the theoretical density (this becomes evident when running more simulations, e.g. 100000). This is not surprising given the uniform prior. The estimates are more accurate when increasing m (the number of samples we resample from), which comes at a high computational cost.

3. In the code below we check how many random numbers the algorithms have generated.

```
# Number of random numbers generated to obtain n samples from the posterior
n_random_numbers = sim$n_random_numbers
n_random_numbers_approx = sim_approx$n_random_numbers
```

The exact algorithm generates on average 7.7077×10^4 / $10^4 = 7.7077$ random numbers to obtain one sample of $f(\theta|\boldsymbol{y})$, closely matching the inverse of the acceptance probability calculated approximately before ($k \approx 7.780$). The approximate algorithm generates m, set to 20 in the code, random numbers per sample. With this m the approximation is easily distinguished from the true distribution. The comparison might have had a different conclusion for a different proposal density, but here the exact algorithm is both more efficient and notably more precise.

4. We now assume a Beta$(1, 5)$ prior

$$f_5(\theta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha,\beta)} = \frac{(1-\theta)^4}{B(1,5)} \propto (1-\theta)^4 .$$

The new posterior is given by

$$f_5(\theta|\mathbf{y}) \propto f(\boldsymbol{y}|\theta) f_5(\theta) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3+4}\theta^{y_4} =: f_5^*(\theta|\mathbf{y}) , \quad \theta \in (0,1) .$$

We wish to use importance sampling to estimate the new posterior mean μ. The algorithm is as following: Sample n times from a distribution $g(x)$ where $g(x) > 0$ where $f_5(\theta|\mathbf{y}) > 0$. We use $n = 10000$ and $g(x) = f(\theta|\mathbf{y})$. Let $w(x_i) = f_5(x_i|\mathbf{y})/f(x_i|\mathbf{y})$. Then $\hat{\mu}_{IS} = \frac{\sum_i x_i w(x_i)}{n}$ is an unbiased estimate of μ. This estimator, however, requires knowledge of the integration constant $c = \int_0^1 f_5^*(x|\mathbf{y})\,\mathrm{d}x$. We could calculate this numerically, but we choose to proceed with a self-normalizing (but in general not unbiased) estimate of μ

$$\tilde{\mu}_{IS} = \frac{\sum_{i=1}^n x_i w(x_i)}{\sum_{i=1}^n w(x_i)} = \frac{\sum_{i=1}^n x_i w^*(x_i)}{\sum_{i=1}^n w^*(x_i)}, \quad \text{where} \quad w^*(x_i) = \frac{f_5^*(x_i|\boldsymbol{y})}{f^*(x_i)} = (1 - x_i)^4 .$$

The method is implemented and tested in the code below.

```
# Posterior density f(theta|y) with prior Beta(1,5)
# (up to a normalising constant)
f_posterior_5_star = function(theta, y) {
  stopifnot(all(0 < theta) && all(theta < 1))
  stopifnot(length(y) == 4)
  return((2 + theta) ^ y[1] * (1 - theta) ^ (y[2] + y[3] + 4) * theta ^
           y[4])
}
```

```
# Use importance sampling to estimate the posterior mean with prior Beta(1,5)
posterior_mean_is = function(n, y) {
  x = r_posterior(n, y)$x
  weights = (1-x)^4  # f_posterior_5_star(x, y) / f_posterior_star(x, y)
  mean_is = sum(x * weights) / sum(weights)
  return(mean_is)
}
```

```
# Use importance sampling to estimate the posterior mean with prior Beta(1,5)
mean_is = posterior_mean_is(n, y)

# Calculate integration constant and true mean numerically
c_5 = integrate(function(x)(f_posterior_5_star(x, y)),
                lower = 0, upper = 1)$value
true_mean_5 = integrate(function(x)(x * f_posterior_5_star(x, y) / c_5),
                        lower = 0, upper = 1)$value
```

The estimated posterior mean is $\tilde{\mu}_{IS} = 0.5960$, while the true mean (computed numerically) is $\mu = 0.5959$. The new prior has changed the posterior and the mean, but importance sampling using the posterior from a uniform prior still results in an accurate estimate of the new posterior mean.