Homework 2

Santiago Ruiz, Nikita Karetnikov, Felix Ubl

2025-05-14

Exercise 5

We want to generate pseudo-random numbers from a Beta(3, 1)-distribution which has the density f(x) = 3x^2 using only runif implemented for random number generation in R.

Write a function implementing the inversion method.

```
library(GoFKernel)
```

```
## Loading required package: KernSmooth

## KernSmooth 2.23 loaded
## Copyright M. P. Wand 1997-2009
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.2.3

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Using the fact that a the cdf of the Beta(3, 1)
# distribution can be approximated by the binomial distribution with parameters n = 3, k = 0,  and p =
# The cumulative distribution function:

cdf_binomial_alpha_3_beta_1 <- function(x) {
  ifelse(x < 0, 0, ifelse(x > 1, 1, pbinom(0, 3, 1 - x)))
}

cdf_inverse_cdf_binomial_alpha_3_beta_1 <- inverse(cdf_binomial_alpha_3_beta_1, lower = 0, upper = 1)

# Checking that it works:
cdf_inverse_cdf_binomial_alpha_3_beta_1(0.2)
```

```
## [1] 0.584804
```

```
cdf_binomial_alpha_3_beta_1(cdf_inverse_cdf_binomial_alpha_3_beta_1(0.2))
```

```
## [1] 0.2000005
```

```r
# Generating random numbers using the inverse method:

gen_inv_beta_3_1 <- function(n) {
sample <- runif(n)
beta_3_1_sample <- numeric(n)
for (i in 1:n) {
beta_3_1_sample[i] <- cdf_inverse_cdf_binomial_alpha_3_beta_1(sample[i])
}
return(beta_3_1_sample)
}

# Measure execution time and save in time_1
time_1 <- system.time({
    gen_inv_beta_3_1(10000)
})
```

Write a function which implements a transformation method.

```r
library(lmomco)
# From the class slides, we know that the random variable X = U / (U + V)
# where U ~ Gamma(r, lambda) and V ~ Gamma(s, lambda) is distributed as Beta(r, s)
# so that
n = 1000
sample <- runif(n)
lambda <- 1

cdf_gamma_3_lambda <- function(x) {
    pgamma(x,3,lambda)
}

cdf_inverse_gamma_3 <- function(p, shape = 3, rate = 1, lower = 0, upper = 100000) {
    uniroot(function(x) pgamma(x, shape = shape, rate = rate) - p, lower = lower, upper = upper)$root
}

cdf_inverse_gamma_1 <- function(p, shape = 1, rate = 1, lower = 0, upper = 100000) {
    uniroot(function(x) pgamma(x, shape = shape, rate = rate) - p, lower = lower, upper = upper)$root
}

# Checking that it works: (There is a small error in the last digit)
cdf_inverse_gamma_3(cdf_gamma_3_lambda(20))
```

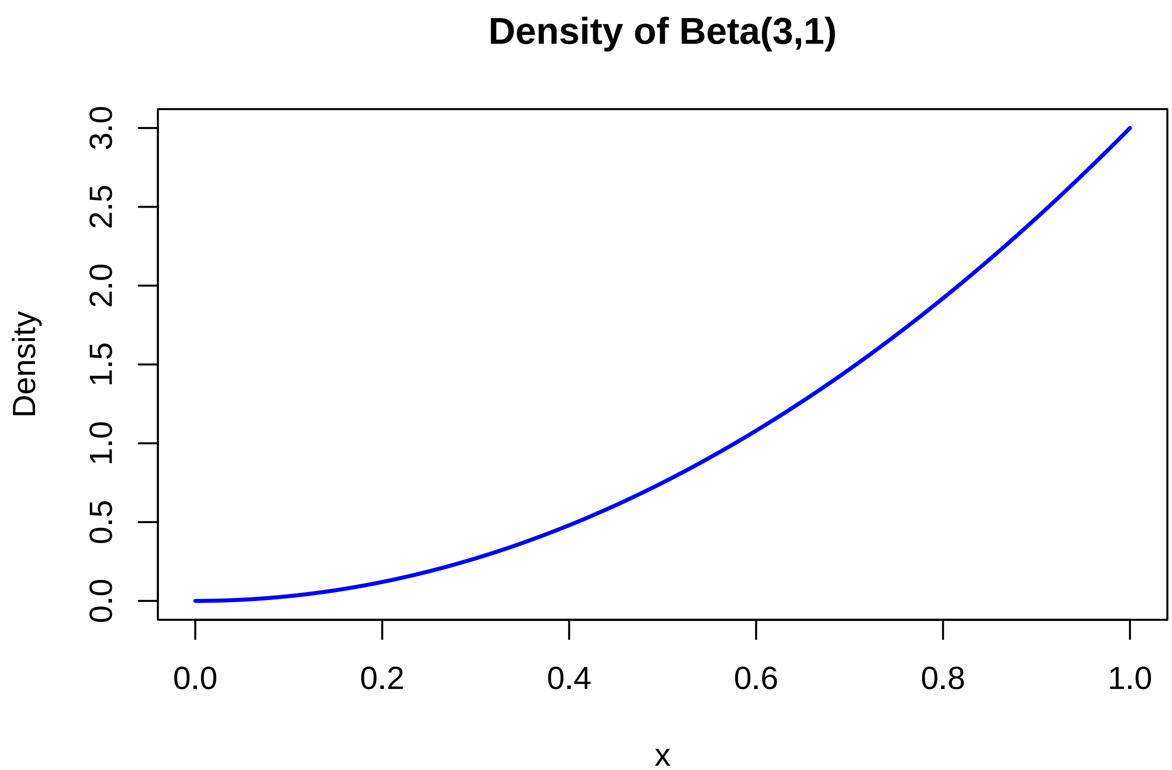
```
## [1] 20.00001
```

```r
# Function to generate samples from the inverse CDF
gen_transf_beta_3_1 <- function(n) {
    sample <- runif(n)
    sample_2 <- runif(n)
    U <- numeric(n)
    V <- numeric(n)
    B <- numeric(n)

    for (i in 1:n) {
        U[i] <- cdf_inverse_gamma_3(sample[i])
        V[i] <- cdf_inverse_gamma_1(sample_2[i])
```

```r
        B[i] <- U[i] / (U[i] + V[i])
    }

    return(B)
}

# Measure execution time and save in time_2
time_2 <- system.time({
    Beta_random <- gen_transf_beta_3_1(10000)
})
```

Write a function which implements rejection sampling using a suitable proposal density.

```r
# Follwing the example in the class, we know that the highest
# value of the density function provided is at


# Plot the density function of Beta(3,1)
curve(dbeta(x, 3, 1), from = 0, to = 1, lwd = 2, col = "blue",
   ylab = "Density", xlab = "x", main = "Density of Beta(3,1)")
```




```r
# The density funciotn of the proposalt destribution is such
# that it is monotonically increasing and has a maximum at 1 with
# a value of 3.

# We establish the minimal possible M equal to 3
# And choose as distribution for Y the uniform distribution in 0,1
```

```r
M <- 3

rbeta31_rs <- function(n) {
   x <- numeric(n)
   i <- 1
   while (i <= n) {
       y <- runif(1)
       u <- runif(1)
       if (u <= y^2 ) {
           x[i] <- y
           i <- i + 1
       }
   }
   return(x)
}

time_3 <- system.time({
  rbeta31_rs(10000)
})

print(time_1)
```

```
##    user  system elapsed
##    0.28    0.00    0.72
```

```r
print(time_2)
```

```
##    user  system elapsed
##    0.95    0.00    1.55
```

```r
print(time_3)
```

```
##    user  system elapsed
##    0.03    0.01    0.11
```

```r
# The 'user time' is the CPU time charged for the execution of user instructions of the calling process
```

- Compare the computational performance of these implementations.

The Rejection Sampling Method seems to be the fastest, followed by the Inversion Method and then the Transformation Method.