# Introduction to Simulation with Variance Estimation

Exercise 2

Bosse Behrens, st.id 12347333

2024W

## Task 1

A Pseudo-random number generator is an algorithm that produces a sequence of numbers from an initial value (seed). This sequence behaves as if drawn from a random sample of a distribution, but actually follows certain recursive rules an dis therefore deterministic. Since the user normally doesn't have information about initial values or the recursive rules, it is near impossible to predict the next number of a sequence. By using the quantile function of a distribution on an uniform distributed random variable $F_X^{-1}(u)$, it is possible to get random samples of another distribution using only an uniform distributed sample due to the probability integral transform Lemma. As long as the quantile function from a distribution can be derived, we therefore only need to generate a (Pseudo-)random uniform sequence. One such algorithm is the linear congruential random number generator. It has 4 arguments: the desired sequence length $n$, a large integer the "modulus" $m$, best-case a prime, another integer the "multiplier" $a$, usually $a \approx \sqrt{m}$ and another integer the "increment" $c$ with $0 \leq c \leq m$. The algorithm then calculates in every step $x_{i+1} = (a \cdot x_i + c)(\text{mod } m)$ until $i + 1 = n$. Additionally in every step $u_i = \frac{x_i}{m}$ is created. Ther algorithm is initialized with the starting value or "seed" $x_1$ and the resulting sequence $u_1, u_2, ..., u_n$ are uniform pseudo random numbers.
We now implement the provided code for the linear congruential random number generator.

```
mc.gen<-function(n,m,a,c=0,x0){
  us<-numeric(n)
  for (i in 1:n){
    x0<-(a*x0+c) %% m
    us[i]<-x0/ m
  }
  return(us)
}
```

Now we test it with different values for $m, a$ and $c$.

```
round(mc.gen(8,16,4,1,1),4)
```

```
## [1] 0.3125 0.3125 0.3125 0.3125 0.3125 0.3125 0.3125 0.3125
```

In this case $a$ is the exact squareroot and also divisor of $m$ which leads to problems in the cycle since it stands that $(4 \cdot 1 + 1)(\text{mod } 16 = 5 = (4 \cdot 5 + 1)(\text{mod } 16$ and therefore having a cycle of 1 and a sequqnce of only one value.

```
round(mc.gen(8,7,3,1,1),4)
```

```
## [1] 0.5714 0.8571 0.7143 0.2857 0.0000 0.1429 0.5714 0.8571
```

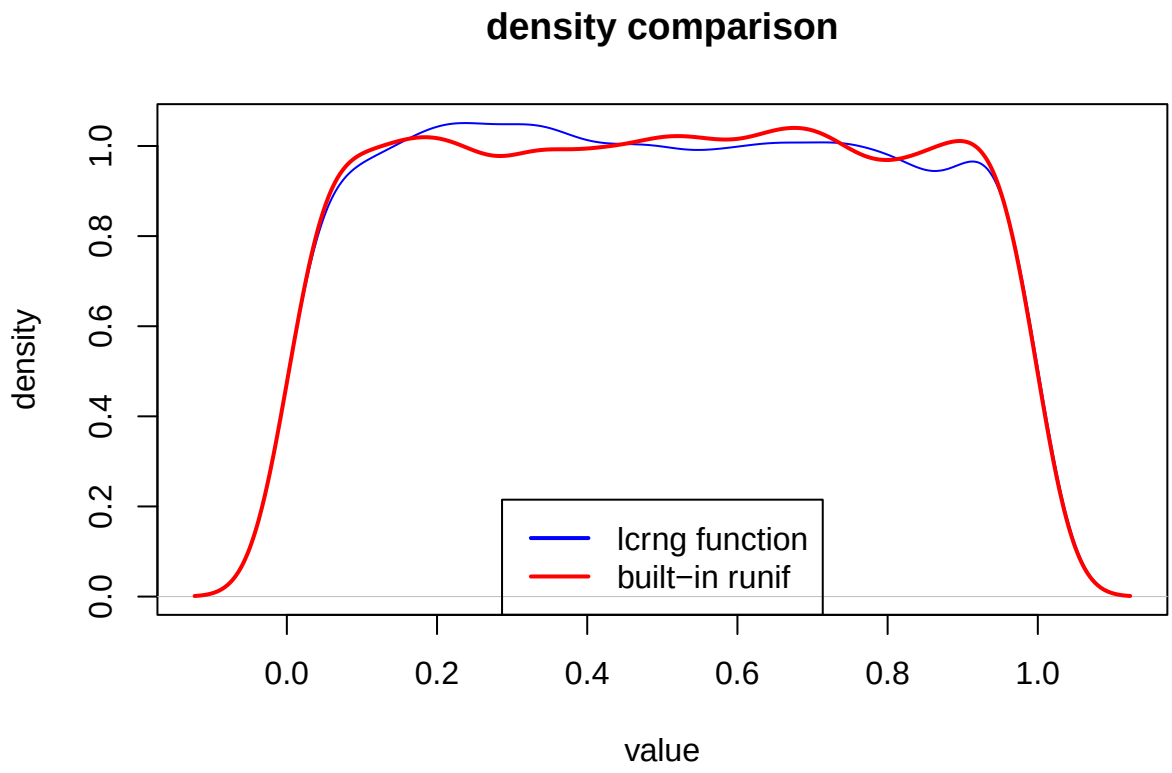
In this example both $a$ and $m$ are prime numbers. Now the first value is repeating 6 values later, since after obtaining $x_n$ it stands that $x_n = x_0$ with the largest possible cycle being $m$. Therefore $m$ also needs to be a verry large integer.

```r
round(mc.gen(8,100003,317,3,43),4)
```

```
## [1] 0.1363 0.2185 0.2688 0.2071 0.6424 0.6474 0.2260 0.6462
```

This sequence already looks far better with a large prime for $m$ and another prime as $a$ close to its squareroot. To compare this to R's built in runif function in generating random uniform distributed numbers, we will create one sequence using our lcrn generator function and one using runif with 10000 samples each. After that we plot the estimated density functions of both sequences against each other.

```r
test_seq <- round(mc.gen(10000,100003,317,3,43),8)
test_seq2 <- runif(10000)
d_test_seq <- density(test_seq)
d_test_seq2 <- density(test_seq2)
plot(d_test_seq, main = "density comparison",
     xlab = "value", ylab = "density", col = "blue")
lines(d_test_seq2, col = "red", lwd = 2)
legend("bottom", legend = c("lcrng function", "built-in runif"),
       col = c("blue", "red"), lwd = 2)
```




As we can observe, our own lcrng now performs really well and similar to the runif function.

### Task 2

If we can easily generate uniform random variables we can easily obtain a random sample from the exponential distribution using the inversion method. To do so, we first need to compute the quantile function of the

exponential distribution from its normal cdf $1 - e^{-\lambda x},\ \lambda > 0$:

$$\begin{aligned} F(x) &= 1 - e^{-\lambda x} \\ e^{-\lambda x} &= 1 - F(x) \\ -\lambda x &= ln(1 - F(X)) \\ x &= -\frac{1}{\lambda} ln(1 - F(x)) \end{aligned}$$

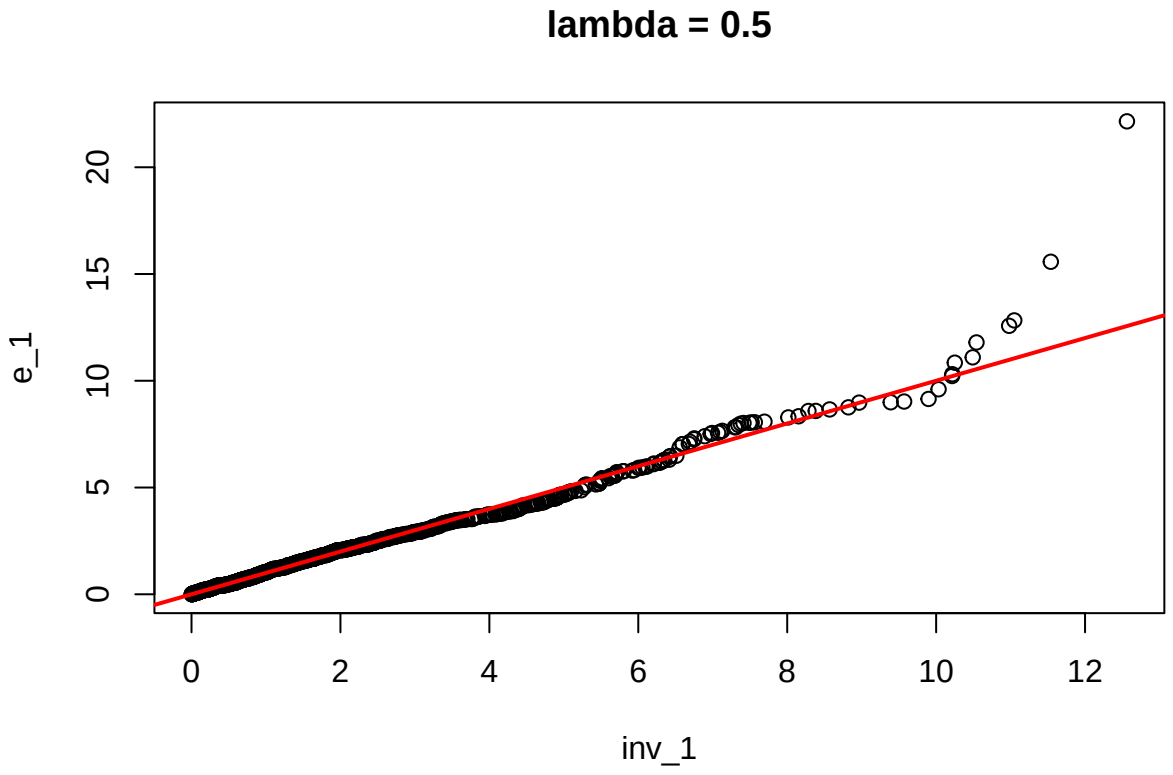
Therefore we are now able to get a random sample from the exponential distribution by transforming an uniform sample $u$ into $-\frac{1}{\lambda} ln(1-u)$. We will now implement a function that takes a number $n$ as the number of desired samples and the exponential distribution parameter $\lambda$.

```
inverse_exp <- function(n, l){
  un <- runif(n)
  expn <- (-1/l)*log(1-un)
  return(expn)
}
```

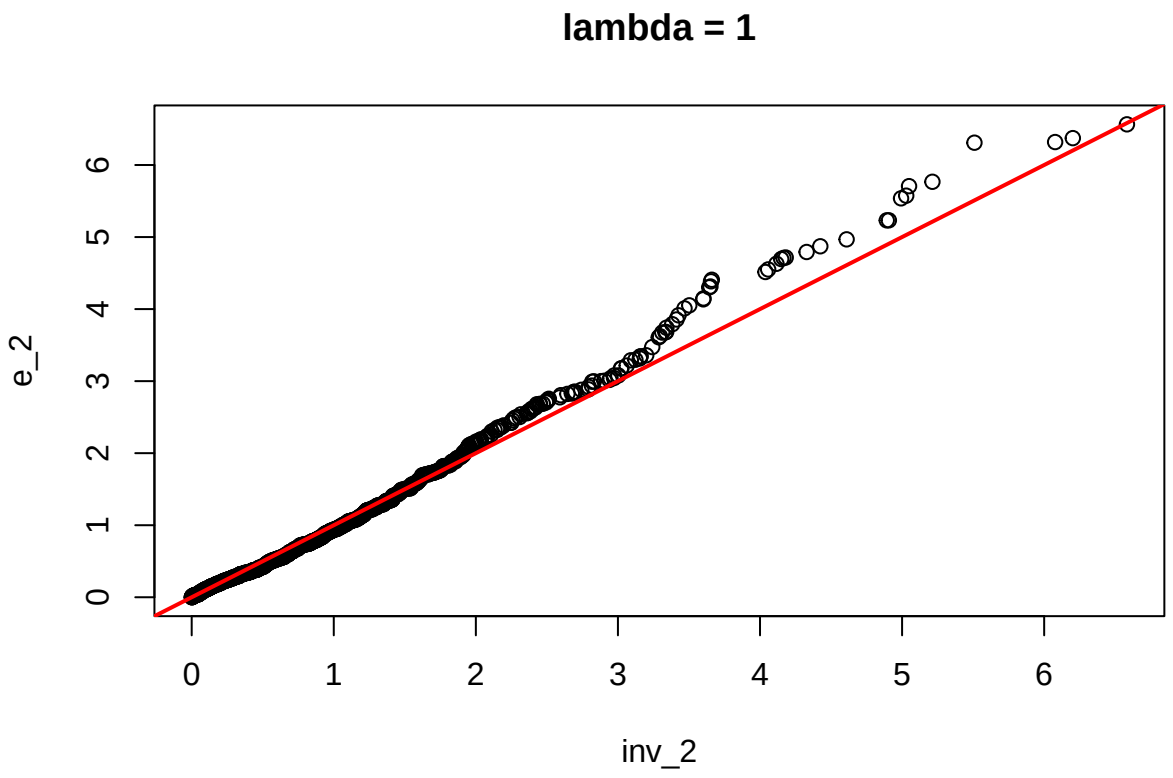
Now we use different values of $\lambda$, such as $\lambda = 0.5, 1, 5$.

```
set.seed(12347333)
inv_1 <- inverse_exp(1000, 0.5)
inv_2 <- inverse_exp(1000, 1)
inv_3 <- inverse_exp(1000, 5)
e_1 <- rexp(1000, 0.5)
e_2 <- rexp(1000, 1)
e_3 <- rexp(1000, 5)
```

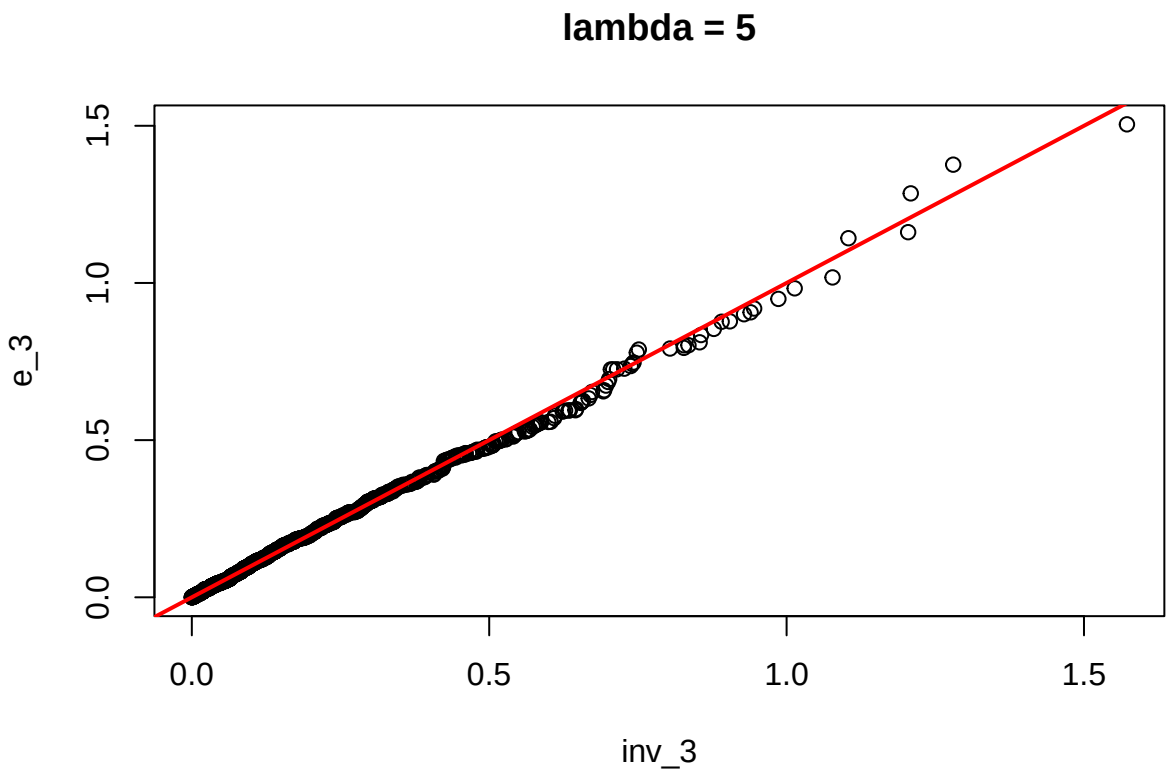
```
qqplot(inv_1, e_1, main = "lambda = 0.5")
abline(0, 1, col = "red", lwd = 2)
```




```
qqplot(inv_2, e_2, main = "lambda = 1")
abline(0, 1, col = "red", lwd = 2)
```




```
qqplot(inv_3, e_3, main = "lambda = 5")
abline(0, 1, col = "red", lwd = 2)
```




The qq-plots seem mostly fine, our transformed uniform sample and the real expoenntial example are very similar near the mode and mean of the exponential distribution. Only in its right tail the values start to differ more and a bit lower than they should be. This effect can be observed more heavily for the lower $\lambda$ values 0.5 and 1, while the more rare values in the right tail for $\lambda = 5$ seem to be better distributed towards an actual exponential distribution.

### Task 3

Since the pdf of the Beta is only defined on $[0, 1]$ the natural candidate for a proposal/instrumental distribution would be the uniform distribution $U_{[0,1]}$ on $[0, 1]$ as well. Furthermore, the Beta distribution stands in a direct relation to the uniform distribution since we know from the lecture that $B(1, 1) \sim U_{[0,1]}$. Also it is easy to use it for the acceptance-rejection approach since we only need to know the maximum value of the Beta pdf and can then easily set the constant $c$ to it. Another related distributions are the normal distribution with $\alpha, \beta$ converging to $\infty$ which does not really help in this approach, and the Gamma distribution, which can be transformed intothe Beta distribution. Using the Gamma distribution also doe not really make sense, since if we can generate random samples from a Gamma distribution we can already directly transform them into Beta distribution samples. Therefore we will use the uniform distribution as our approach.
First we calculate $\Gamma(2) = 1$ and $\Gamma(4) = 6$. Then we get for $\alpha = \beta = 2$ : $f(x; \alpha, \beta) = \frac{\Gamma(2+2)}{\Gamma(2)\Gamma(2)} x^{2-1}(1-x)^{2-1} = 6x(1-x)$. Now we calculate the max value in $[0, 1]$ using the optimize function:

```
f <- function(x){6*x*(1-x)}
max <- optimize(f, interval = c(0,1), maximum = TRUE)
print(max$objective)
```

```
## [1] 1.5
```

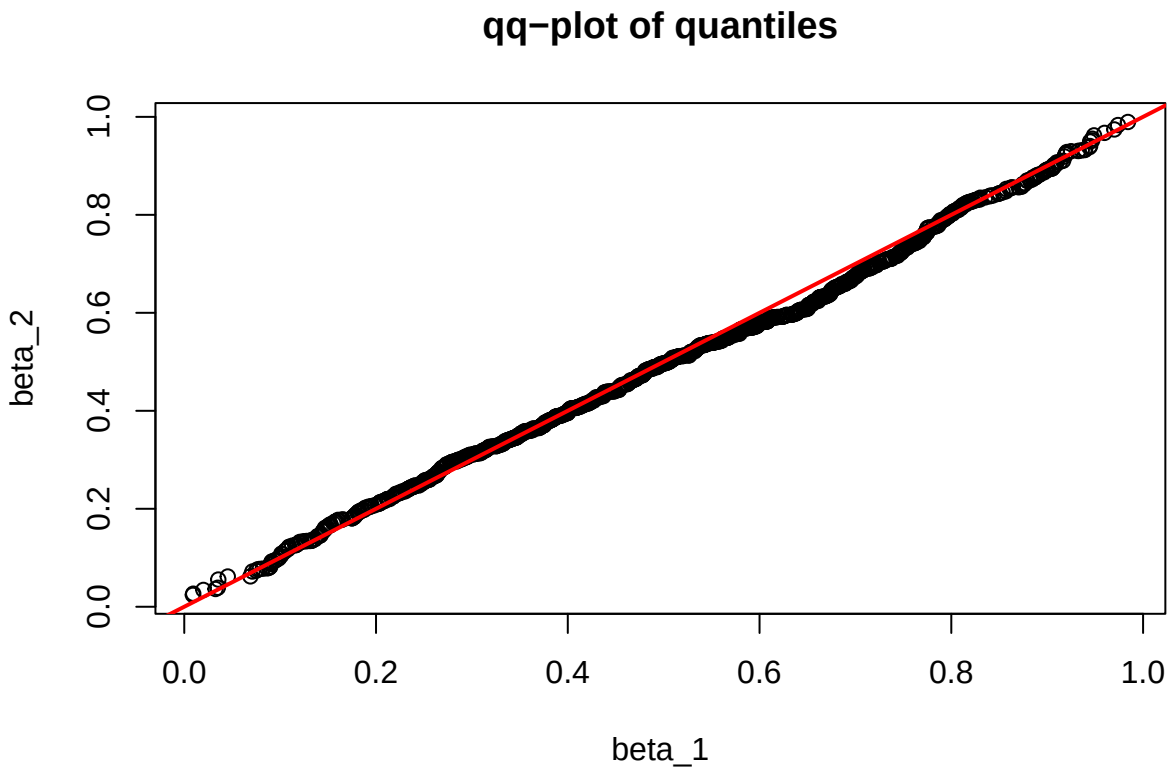
Now we know that the maximum value of the $B(2, 2)$ pdf is 1.5. Therefore it now holds that for $g(x)$ as the pdf of the uniform distribution and $f(x)$ from the Beta distribution that $f(x) \leq 1.5g(x)$. $c = 1.5$ is obviously

the lowest possible value or the $\leq$ would not hold and therefore also the best to keep the rejection proportion as low as possible.

```
beta_2_2_rej <- function(n){
  sample <- c()
  while (length(sample) < n) {
    y <- runif(1)
    u <- runif(1)
    if (u <= dbeta(y, 2, 2)  /(1.5)){ # g(x)=1, no need to multiply
      sample <- c(sample, y)
    }
  }
  return(sample)
}
```

Now we test this in a qq-plot by plotting the quantiles against those of samples from the built-in rbeta function.

```
set.seed(12347333)
beta_1 <- beta_2_2_rej(1000)
beta_2 <- rbeta(1000, 2, 2)
qqplot(beta_1, beta_2, main = "qq-plot of quantiles")
abline(0, 1, col = "red", lwd = 2)
```




As we can observe it has worked very well.

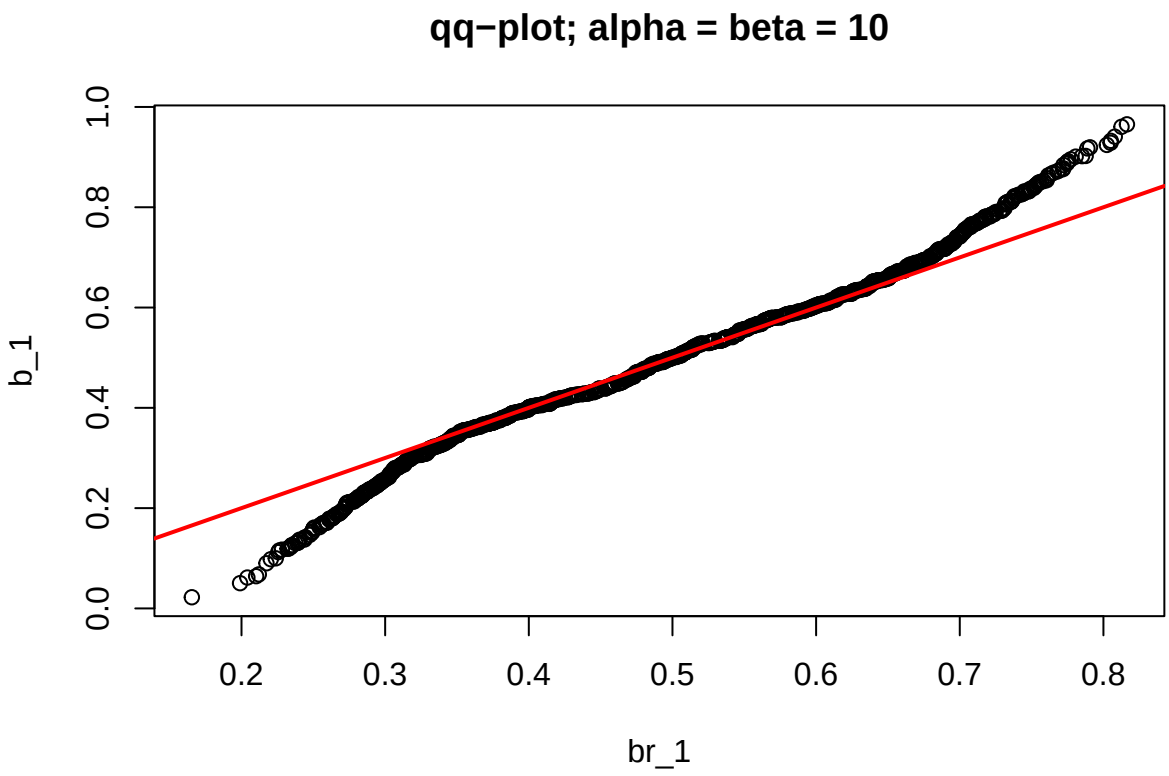
Now we will implement a function that samples from an arbitrary Beta distribution that adapts the constant to the specified parameters assuming both are larger than 1. To adapt the constant we use the same approach

by calculating the mode of the Beta distribution with its specified parameters. We know from our statistic classes that for $\alpha, \beta > 1$ the mode is $\frac{\alpha-1}{\alpha+\beta-2}$.

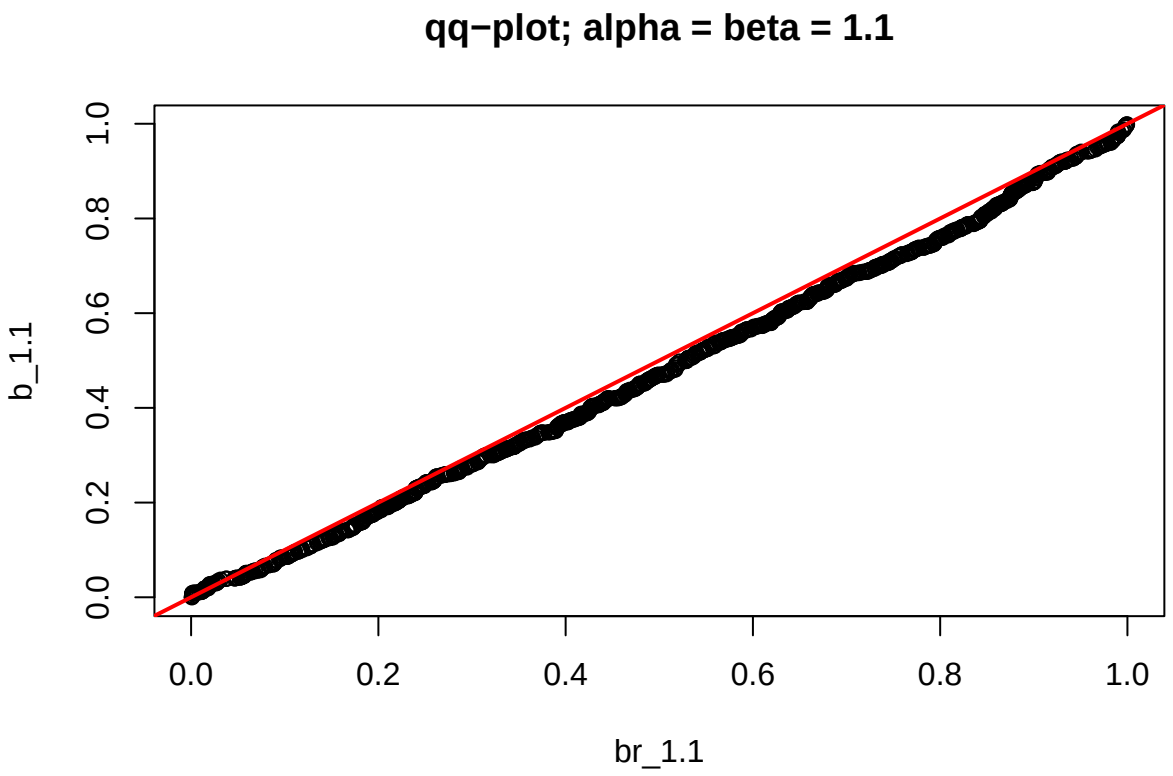
```
beta_rej <- function(n, a, b){
  c <- (a-1)/(a+b-2)
  sample <- c()
  while (length(sample) < n) {
    y <- runif(1)
    u <- runif(1)
    if (u <= dbeta(y, a, b)/c){  # g(x)=1, no need to multiply
      sample <- c(sample, y)
    }
  }
  return(sample)
}
```

We now test our new function with arguments for $\alpha$ and $\beta$ for the acceptance-rejection approach to draw random samples for Beta distributions with $\alpha, \beta > 1$. To do so we generate random samples from the built-in rbeta function and compare them in a qq-plot. We use as examples distributions with parameters $\alpha = \beta = 10; \alpha = \beta = 1.1; \alpha = 2, \beta = 5; \alpha = 5, \beta = 2$

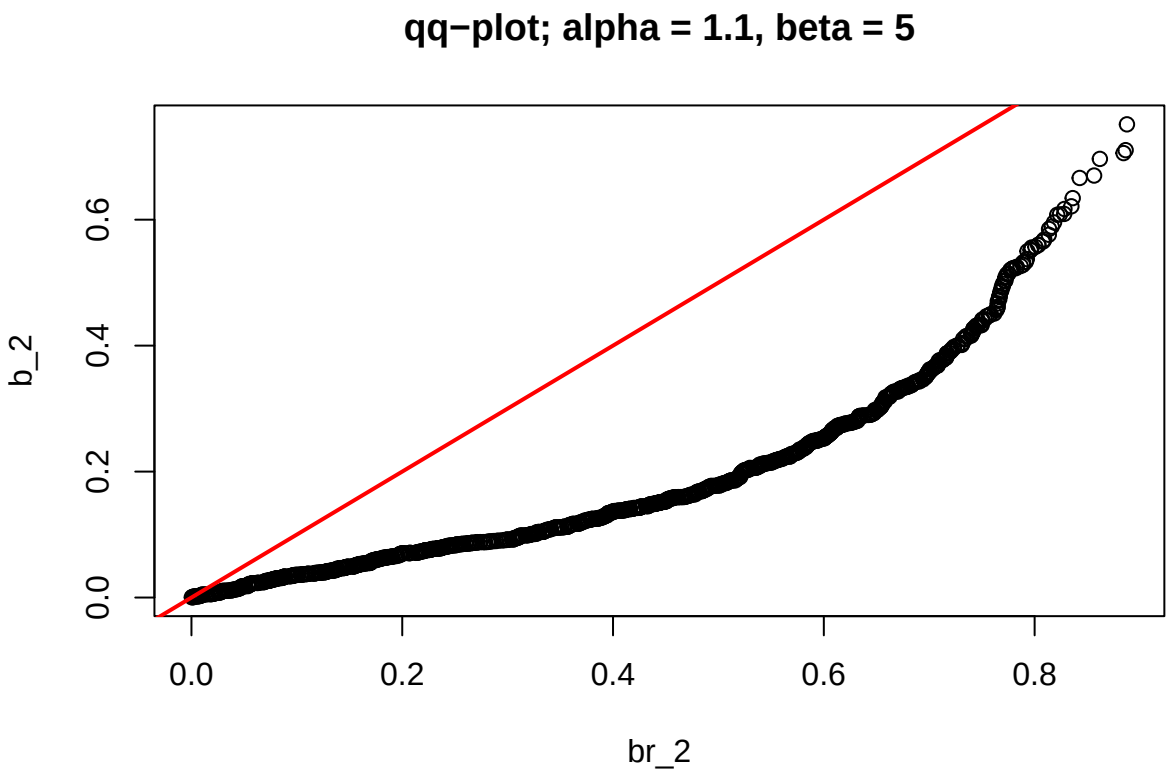
```
set.seed(12347333)
br_1 <- beta_rej(1000, 10, 10)
b_1 <- rbeta(1000, 3, 3)
br_1.1 <- beta_rej(1000, 1.1, 1.1)
b_1.1 <- rbeta(1000, 1.1, 1.1)
br_2 <- beta_rej(1000, 1.1, 5)
b_2 <- rbeta(1000, 1.1, 5)
br_3 <- beta_rej(1000, 5, 1.1)
b_3 <- rbeta(1000, 5, 1.1)
qqplot(br_1, b_1, main = "qq-plot; alpha = beta = 10")
abline(0, 1, col = "red", lwd = 2)
```



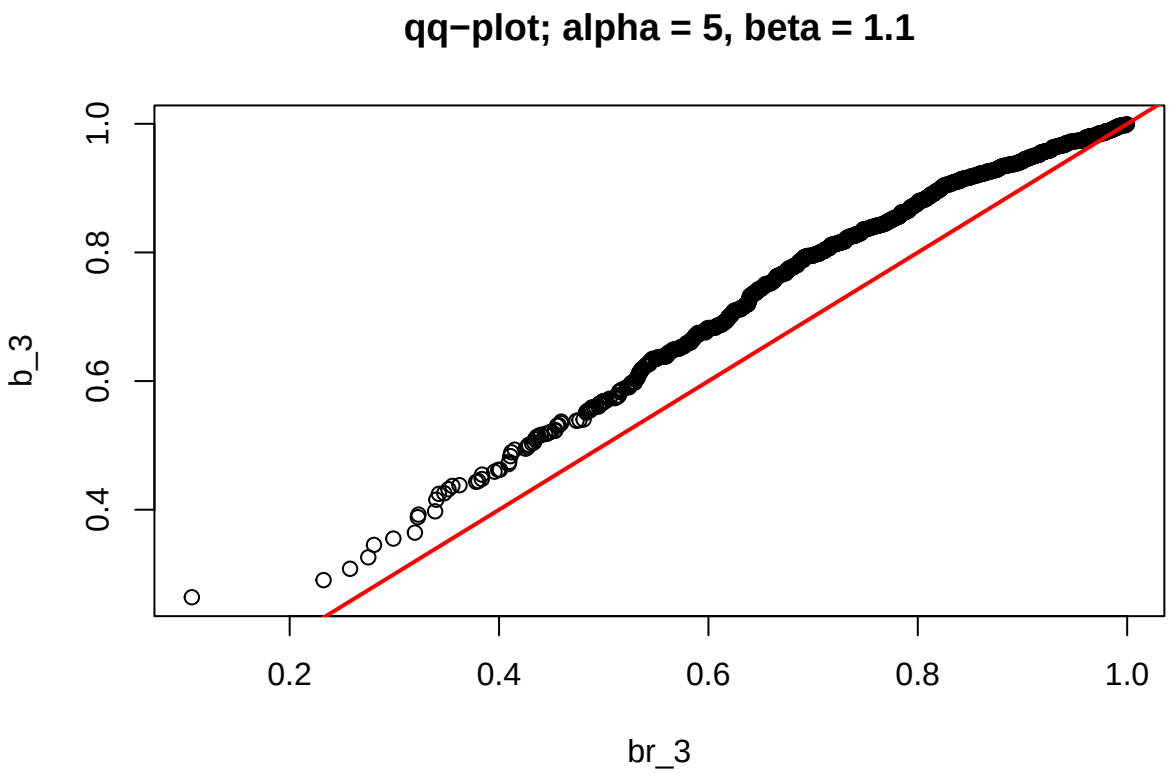

```
qqplot(br_1.1, b_1.1, main = "qq-plot; alpha = beta = 1.1")
abline(0, 1, col = "red", lwd = 2)
```




```r
qqplot(br_2, b_2, main = "qq-plot; alpha = 1.1, beta = 5")
abline(0, 1, col = "red", lwd = 2)
```




```
qqplot(br_3, b_3, main = "qq-plot; alpha = 5, beta = 1.1")
abline(0, 1, col = "red", lwd = 2)
```




As we can observe, not all of these plots perform as well as for the beta(2,2) distribution anymore. It seems that it works best for small values $> 1$ and also if both parameters have the same value. This is expected, for as closer the parameters are to one, the closer it is to an uniform distribution overall. Also the further the parameters are apart, the more asymmterical centered around $1/2$ the Beta distribution becomes. That explains the right/left skewedness in the two plots with different values for $\alpha$ and $\beta$.