Generation of Random Numbers

Code by Claudio P.

I created this report using R Markdown.

Exercise 1

Functions to Generate random numbers

First I created two functions for x and y

```
lcg.x <- function(N=10, a=40014, b=0, m=2147483563, uniform=TRUE){
  x<-numeric(N) #initializing
  x0 <- 10 #seed
  x[1] <- (a*x0+b) %% m # first generated value
  for ( n in 1:(N-1) ){
    x[n+1] <- ((a*x[n]+b) %% m )
  }
  if (uniform) x<-x/m
  return(x)
}

lcg.y <- function(N=10, a=40692, b=0, m=2147483399, uniform=TRUE){
  y<-numeric(N)
  y0 <- 10
  y[1] <- (a*y0+b) %% m # first generated value
  for ( n in 1:(N-1) ){
    y[n+1] <- ((a*y[n]+b) %% m )
  }
  if (uniform) y<-y/m
  return(y)
}
```

Then I combined them into the final one:

```
lcg.w <- function(N=10, m=c(2147483563, 2147483399, 2147483562), uniform=TRUE){
  #initializing values
  a <- c(40014, 40692, 1)
  b <- c(0,0,0)
  x0 <- 10 #seed
  y0 <- 10 #seed
  x<-numeric(N);#initialing
  y<-numeric(N);
  w<-numeric(N);
```

```
  #initial values
  x[1] <- (a[1]*x0+b[1]) %% m[1]
  y[1] <- (a[2]*y0+b[2]) %% m[2]
  w[1] <- (a[3]*(x[1]-y[1])+b[3]) %% m[3]

  for ( n in 1:(N-1) ){
    x[n+1] <- ((a[1]*x[n]+b[1]) %% m[1] )
    y[n+1] <- ((a[2]*y[n]+b[2]) %% m[2] )
    w[n+1] <- ((a[3]*(x[n+1]-y[n+1])+b[3]) %% m[3])
  }
  if (uniform) w<-w/m[3]
  return(w)
}
```

Several random sequences, with different size

I run the function and stored them into variables. I'll show the smallest one:

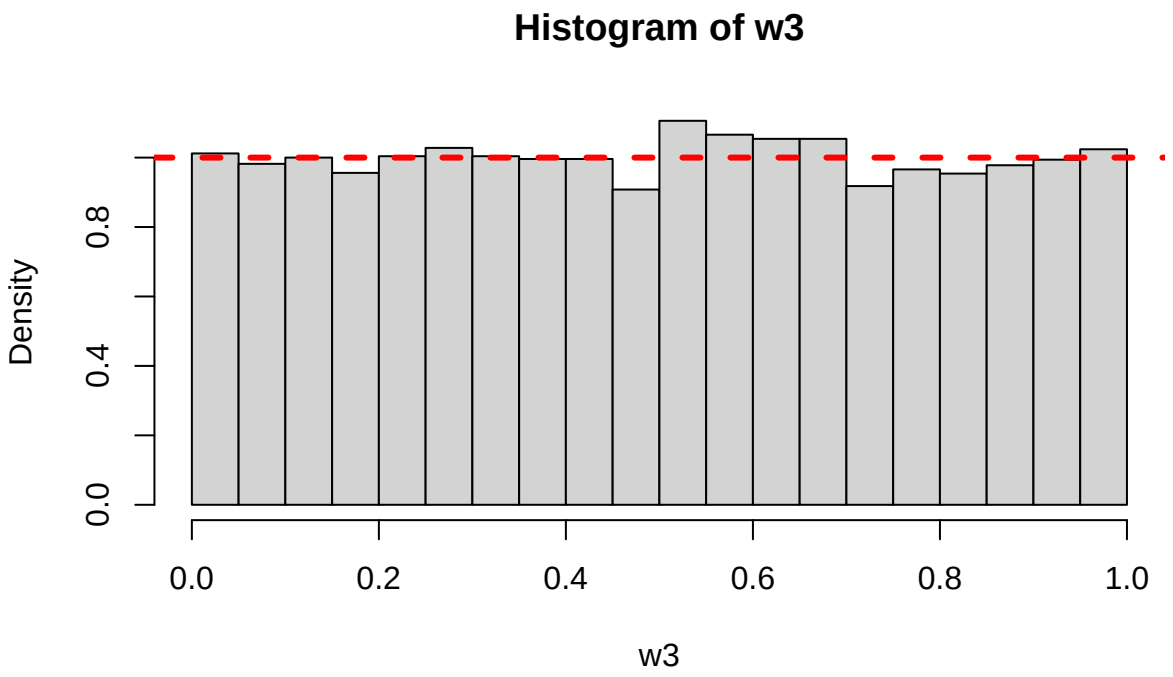
```
N <- c(100, 1000, 10000)
w1<- lcg.w(N=100)
w2<- lcg.w(N=1000)
w3<- lcg.w(N=10000)

w1
```

```
##   [1] 0.99999684 0.74519580 0.47483846 0.33085534 0.36944521 0.61711811
##   [7] 0.64645017 0.84672857 0.04136353 0.67306886 0.54957368 0.28816083
##  [13] 0.10081651 0.11716168 0.99021105 0.65447439 0.99824220 0.50282338
##  [19] 0.91810683 0.02341471 0.59686680 0.14975266 0.41179065 0.73548282
##  [25] 0.24359287 0.88210468 0.08177839 0.62233630 0.47707930 0.78922316
##  [31] 0.05425573 0.21749666 0.80334993 0.30136363 0.05256697 0.46307537
##  [37] 0.82026274 0.40707494 0.60981674 0.19936691 0.60773598 0.74579841
##  [43] 0.10617091 0.14229373 0.32932125 0.55183188 0.90494980 0.05473967
##  [49] 0.61298659 0.91246735 0.68887847 0.34269015 0.77925837 0.88529071
##  [55] 0.91748471 0.37011162 0.47704096 0.50205522 0.63276080 0.68989702
##  [61] 0.05215908 0.29304990 0.77485055 0.65103307 0.05842928 0.32094676
##  [67] 0.10769872 0.82376741 0.85657327 0.46697018 0.23716637 0.83116759
##  [73] 0.34344117 0.39119962 0.62550043 0.71047019 0.76484515 0.54543344
##  [79] 0.76777537 0.90316061 0.68215021 0.08072001 0.97184842 0.53967395
##  [85] 0.40927918 0.28700119 0.45025709 0.64555031 0.28742293 0.99047767
##  [91] 0.70199269 0.82884432 0.96315791 0.31291554 0.73544275 0.64434657
##  [97] 0.62209114 0.35129244 0.30283369 0.11908684
```

Graphical Checks

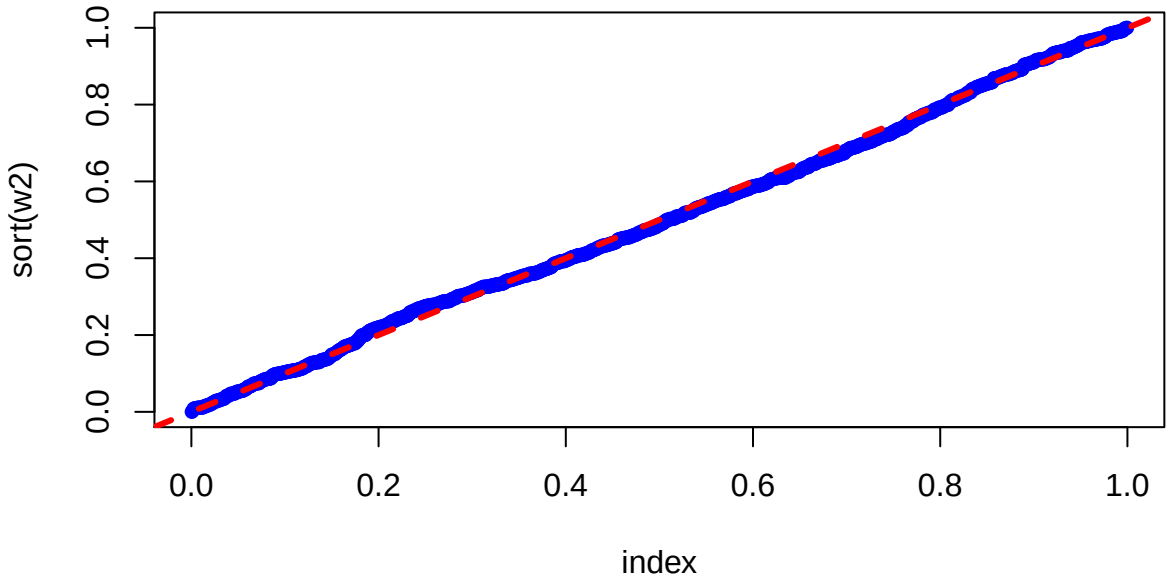
I didn't show every plot for every N I computed, but only the one I thought it was the most relevant.

```
hist(w3, probability=TRUE)
abline(h=1, col="red", lwd=3, lty=2)
```



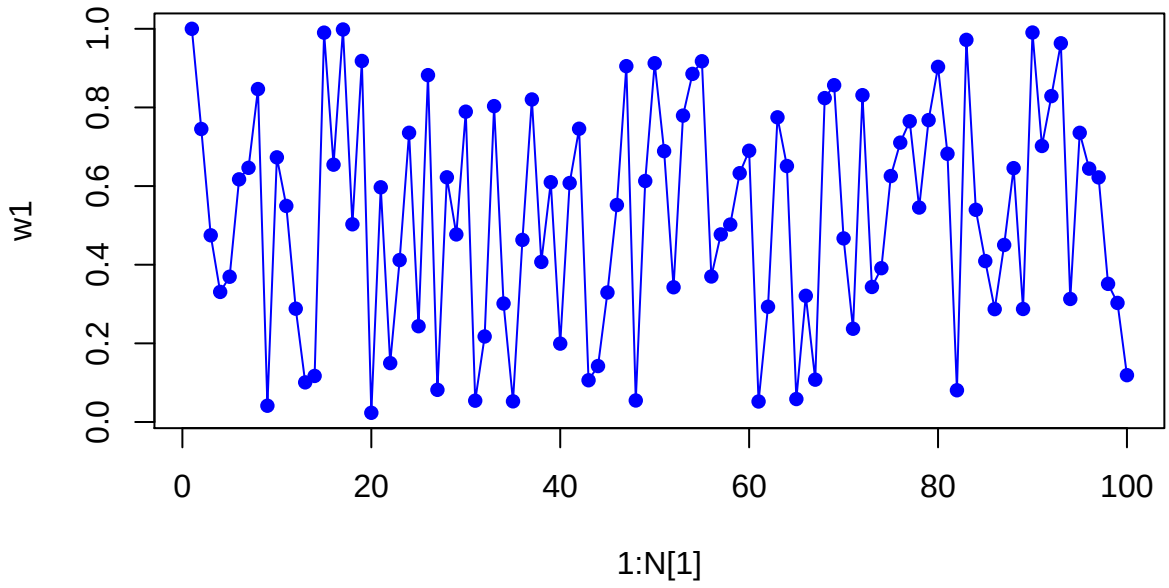
Histogram, this one says that for N=10000 the density of the produced data is very similar to the theoretical Uniform density.

```
index<-1:N[2]/N[2]-1/(2*N[2])
plot( index, sort(w2), cex=1.0, pch=16, col="blue")
abline(b=1,a=0,col="red", lwd=3, lty=2)
```



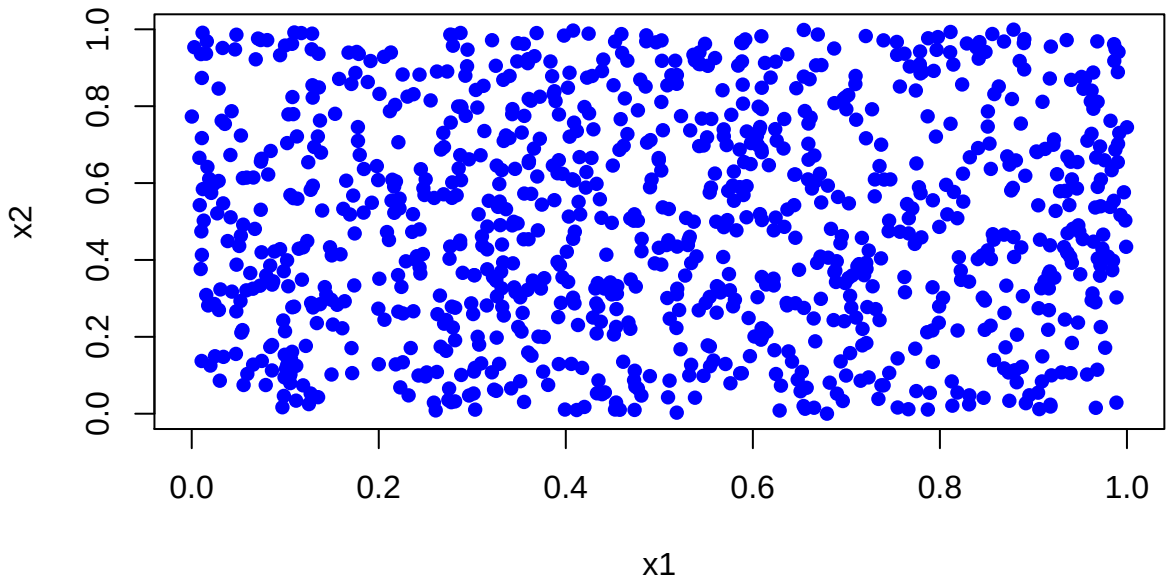
QQ-Plot for N=100, the theoretical quantiles are very similar to the computed ones. the Match is almost perfect.

```
plot( 1:N[1], w1, cex=1.0, pch=16, col="blue",type='o')
```



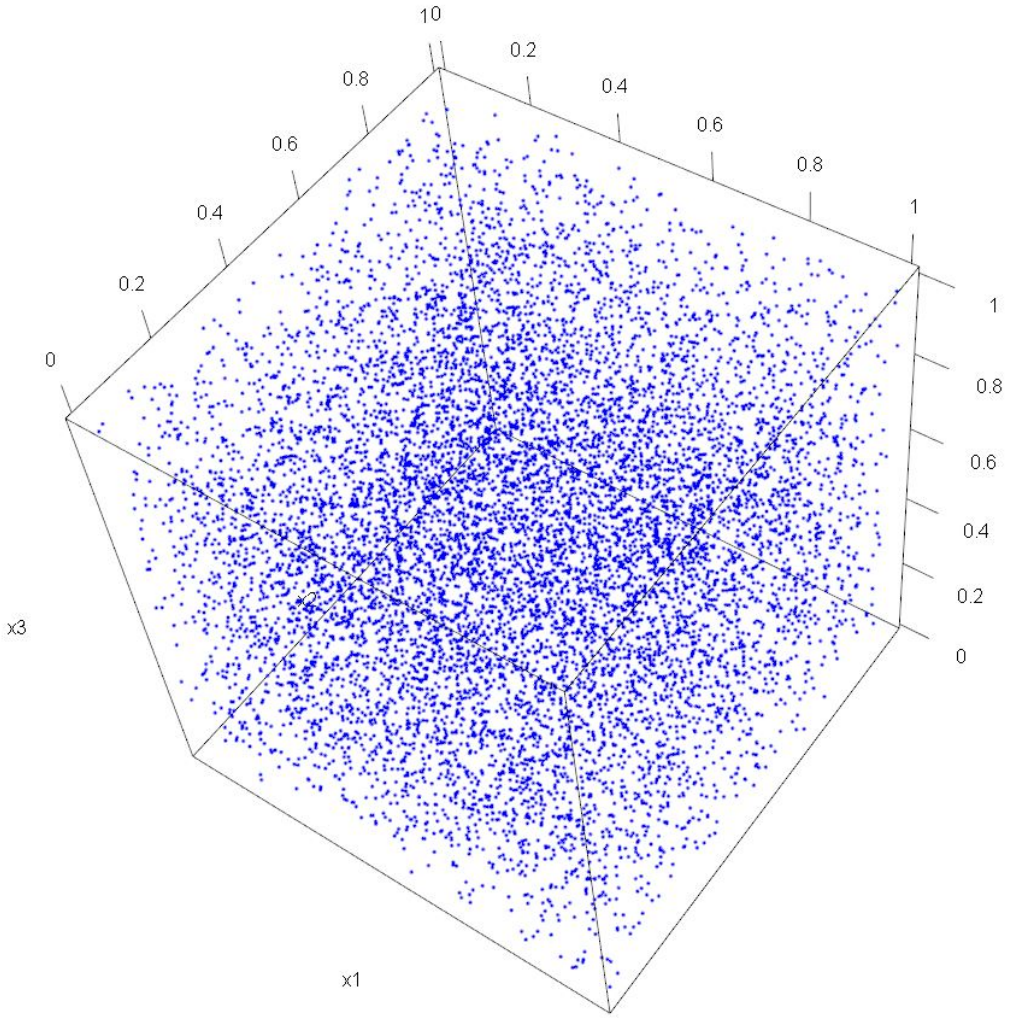
Time plot for N=100 (with N= 1000+ it was unreadable), it shows no pattern in the data.

```
i<-1:(N[2]-1)
x1 <- w2[i]
x2 <- w2[i+1]
plot( x1, x2, cex=1.0, pch=16, col="blue")
```



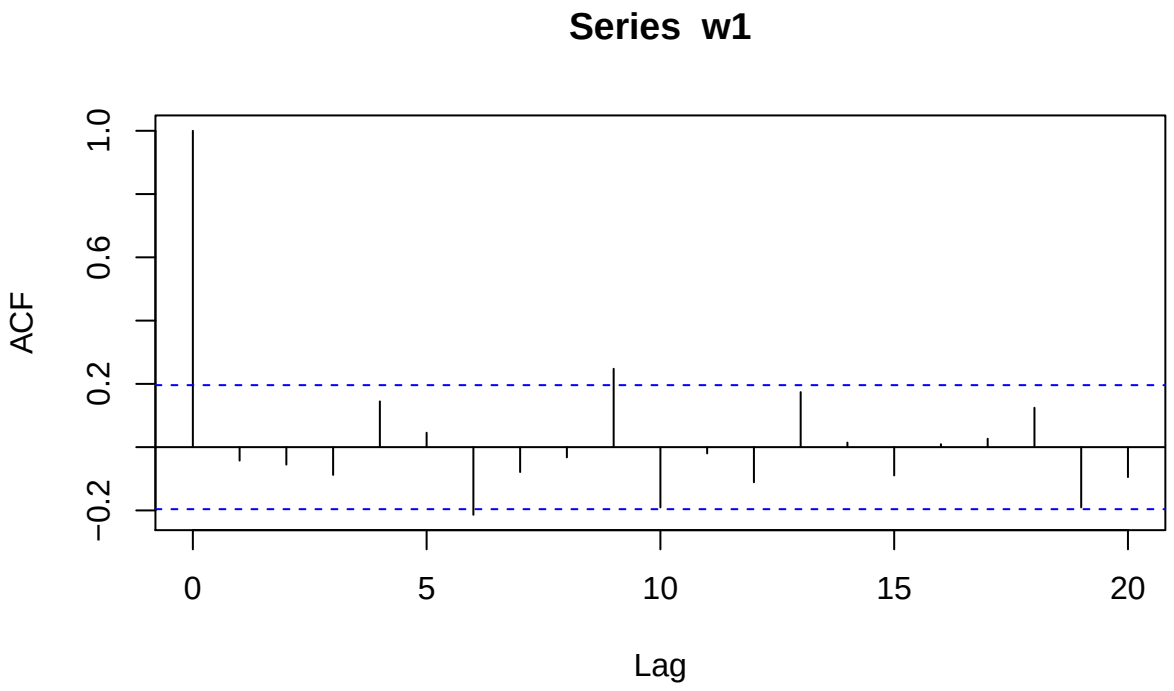
Scatterplot for N=1000, the points seem correctly to be random.

```
library(rgl)
i<-1:(N[3]-2)
x1 <- w3[i]
x2 <- w3[i+1]
x3 <- w3[i+2]
plot3d( x1, x2, x3, cex=1.0, pch=16, col="blue")
```



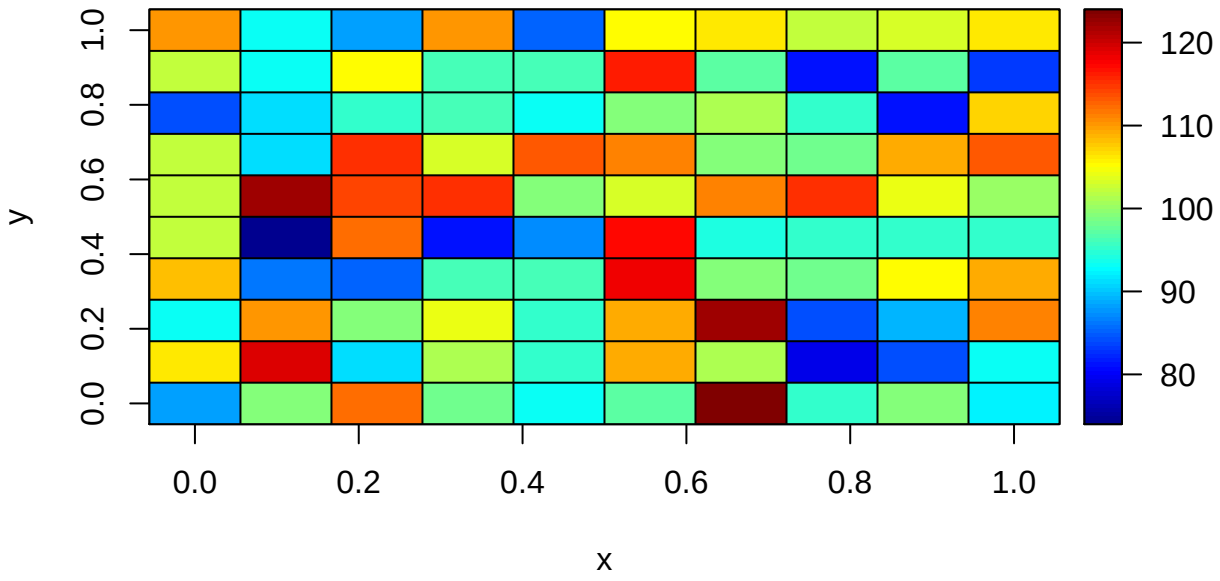
The cube showed no empty areas and no hyperplanes inside.

```
acf(w1)
```



Autocorrelation plot for N=10000. It doesn't show any particular auto-correlation

```
library(plot3D)
ucat <- cut(w3, seq(0,1,0.1))
tabu <- table( ucat[i], ucat[i+1] )
image2D(z=tabu, border="black")
```



3D histogram: points go from 80 to 120 (it's ok, since they're supposed to be around 100).

Tests of randomness

First, I'll use the functions we used during lectures to calculate serial pairs and triads.

```
library(randtoolbox)
library(randtests)
serial.pairs.test<-function(x,K=5){
  index <- seq(0,1,length.out=K+1)
  xcat<-cut( x, index )
  x1 <- xcat[1:(N-1)]
  x2 <- xcat[2:N]
  f <- as.vector(table(x1,x2))
  p <- rep(1/K^2,K^2)
  return(chisq.test(f,p=p))
}

serial.triads.test<-function(x,K=5){
  index <- seq(0,1,length.out=K+1)
  xcat<-cut( x, index )
  x1 <- xcat[1:(N-2)]
  x2 <- xcat[2:(N-1)]
  x3 <- xcat[3:N]
  f <- as.vector(table(x1,x2,x3))
  p <- rep(1/K^3,K^3)
  chisq.test(f,p=p)
}
```

Then I'll run the tests and store them into variables:

```
Freq <- freq.test(w3, 0:10)
Serial <- serial.test(w3)
Gap <- gap.test(w3)
Poker <- poker.test(w3)
Runs <- runs.test(w3)
Pairs <- serial.pairs.test(w3,10)
Triads <- serial.triads.test(w3,10)
```

Now I'll show their results and their names in a vector. The result is the p-value for each test.

```
p_names <- c("Frequency Test","Serial Test","Gap Test","Poker Test","Runs Test",
           "Permutation Test","Serial Correlation Test")
p_values <- c(Freq$p.value, Serial$p.value, Gap$p.value, Poker$p.value,
              Runs$p.value, Pairs$p.value, Triads$p.value)
names(p_values) <- p_names
p_values
```

```
##          Frequency Test             Serial Test                 Gap Test
##                    0.73                    0.32                    0.99
##              Poker Test               Runs Test        Permutation Test
##                    0.53                    0.47                    0.89
## Serial Correlation Test
##                    0.28
```

So, since every p-value is greatly above the standard 0.05 threshold, we cannot reject the null hypothesis of randomness of the data.

Conclusion

It seems to be a good generator. Graphical tests didn't show any pattern in the data, so we can assume it is random. This is confirmed by the tests, due to the p-values that are greatly above 5%.

Exercise 2

Logistic(0,1) from Uniform - Transformation method

$$x = log\frac{u}{1-u} := g(u)$$

then

$$-x = log\frac{1-u}{u}$$

so we can write

$$e^{-x} = \frac{1-u}{u}$$

which is

$$u(1+e^{-x})) = 1$$

then

$$u = \frac{1}{1+e^{-x}} := g^{-1}(x)$$

Therefore, we can compute the derivative of $g^{-1}(x)$: $(g^{-1}(x))' = (\frac{1}{1+exp(-x)})' = \frac{e^{-x}}{(1+e^{-x})^2}$ Finally, since U~Uniform(0,1) and $f_u(g^{-1}(x)) = 1$ (because f is the density of a uniform), we can say that: $f_x(x) = 1 * \frac{e^{-x}}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2}$ which is the Logistic(0,1)

Logistic(0,1) from Uniform - Inversion method

The Inversion Theorem says that if X ~ F(x), then U=F(X) ~ U(0,1). So: $x = log\frac{u}{1-u}$ as before

$$e^{-x} = \frac{1-u}{u}$$

Then, if

$$u = \frac{1}{1+e^{-x}}$$

we can observe that can write

$$u = \frac{1}{1+e^{-\frac{x-\mu}{\sigma}}}$$

where μ=0 and σ=1. But this is the CDF of a logistic distribution, so thanks to the Inversion theorem:

$$\frac{1}{1+e^{-x}} = u = F(x)$$

therefore x ~ F(x), so X is a Logistic(0,1)

General Logistic from Uniform - Transformation method

I'll use the Transformation method.

$$\frac{x-\mu}{\sigma} = log\frac{u}{1-u} := g(u)$$

then

$$-\frac{x-\mu}{\sigma} = log\frac{1-u}{u}$$

so we can write

$$e^{-x} = \frac{1-u}{u}$$

which is

$$u(1+e^{-\frac{x-\mu}{\sigma}})) = 1$$

then

$$u = \frac{1}{1+e^{-\frac{x-\mu}{\sigma}}} := g^{-1}(x)$$

since μ and σ are parameters. Therefore, we can compute the derivative of $g^{-1}(x)$:

$$(g^{-1}(x))' = (\frac{1}{1+exp(-\frac{x-\mu}{\sigma})})' = \frac{e^{-\frac{x-\mu}{\sigma}}}{(1+e^{-\frac{x-\mu}{\sigma}})^2}$$

Finally, since U~Uniform(0,1) and $f_u(g^{-1}(x)) = 1$ (because f is the density of a uniform), we can say that:

$$f_x(x) = 1 * \frac{e^{-\frac{x-\mu}{\sigma}}}{(1+e^{-\frac{x-\mu}{\sigma}})^2} = \frac{e^{-\frac{x-\mu}{\sigma}}}{(1+e^{-\frac{x-\mu}{\sigma}})^2}$$

which is the density of the Logistic(μ,σ)

Function to simulate a Logistic distribution

To simulate the Uniform distribution, I used a random generator there the seed (if not specified) is equal to the system time. For simplicity I provided some default values for the parameters:

```
logi.rand <- function(N=10, seed=NA, mu=0, sig=1) {
  u <- numeric(N) #initializing
  m <- 2 ** 34 #component of the random generator
  a <- 1103515245
  b <- 12345

  if (is.na(seed)) {
    seed <- as.numeric(Sys.time()) * 1000
  }
  x0 <- seed
  for (i in 1:N) {
    x0 <- (a * x0 + b) %% m
    u[i] <- x0 / m
  }
  x<- mu +sig*log( u/(1-u) )
  return(x)
}
```

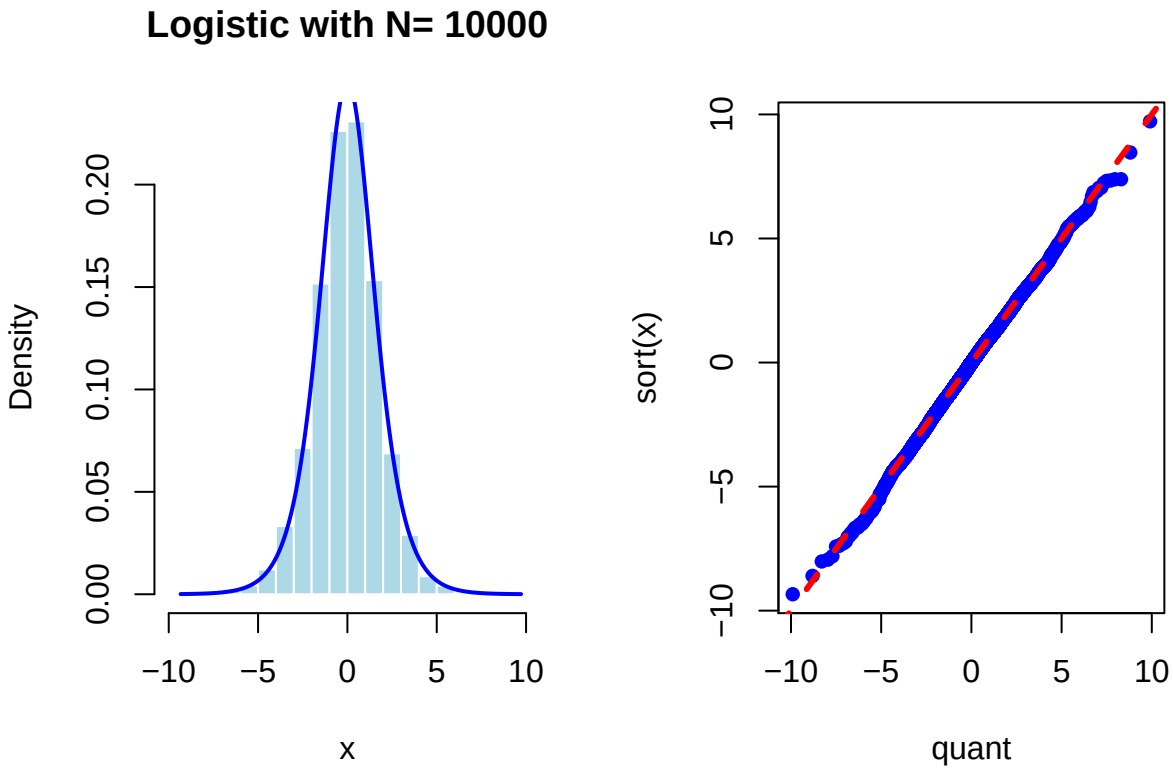
Graphical comparison

First, I used this parameters: default seed (system time), size of the sample 10000, mu and sigma parameters for the logistic(0,1).

```
mu<-0
sig<-1
N<-10000
```

Then the plots, where for the QQplot I computed the theoretical quantiles for the logistic:

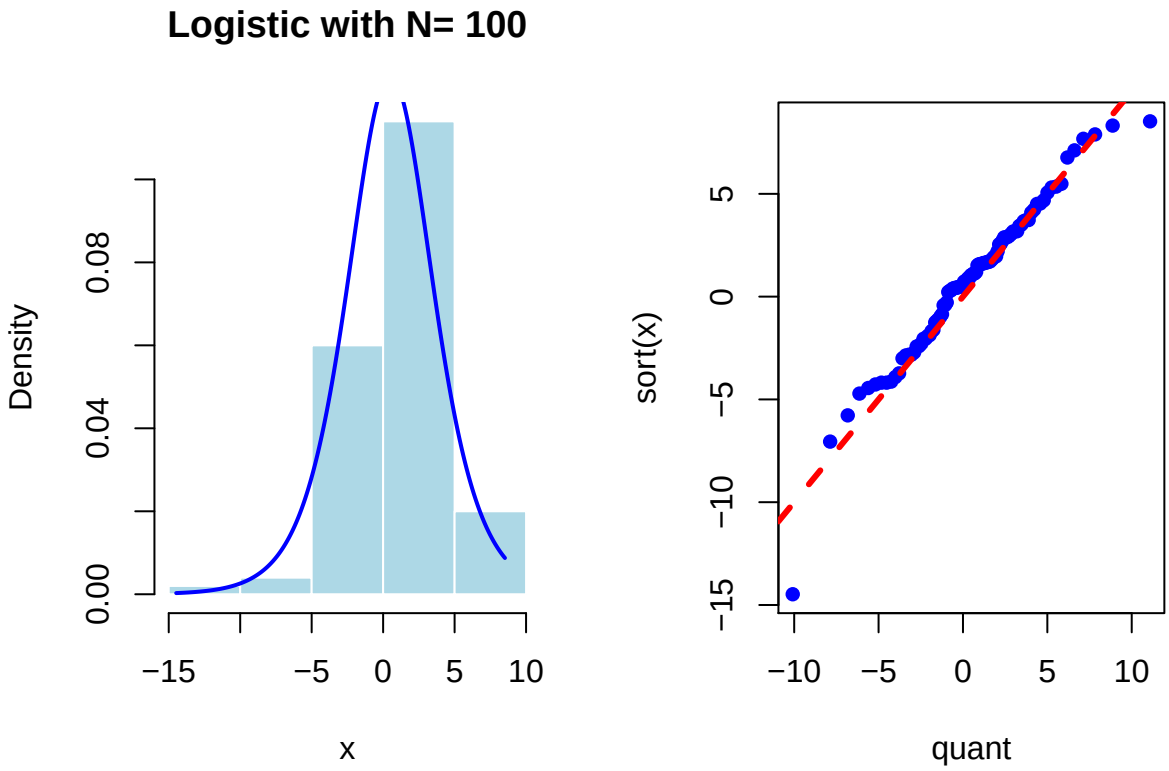
```
x <- logi.rand(10000,mu=0,sig=1)

par(mfrow=c(1,2))
x0 <- seq( min(x), max(x), 0.01)
hist(x, probability=T, col = "lightblue", border = "white", main=paste("Logistic with N=",N)  )
lines(x0, dlogis(x0, mu,sig), col="blue", lwd=2)

index<- index<- 1:N/N-1/(2*N)
quant<-qlogis(index, mu, sig) #theoretical quantiles of the logistic
plot( quant, sort(x), cex=1.0, pch=16, col="blue")
abline(b=1,a=0,col="red", lwd=3, lty=2)
```



The generator has something unusual: the tails of the generated values are bigger than the theoretical ones. We can see that in the histogram and especially in the QQ plot, where the extremes values diverge from the actual logistic.

The second plot used different parameters, with a sample size of 100, seed=2 and logistic(0.5,2)

```
N<- 100
mu<-0.5
sig<-2
seed<-10

x <- logi.rand(N, seed, mu, sig)

par(mfrow=c(1,2))
x0 <- seq( min(x), max(x), 0.01)
hist(x, probability=T, col = "lightblue", border = "white", main=paste("Logistic with N=",N)  )
lines(x0, dlogis(x0, mu,sig), col="blue", lwd=2)

index<- index<- 1:N/N-1/(2*N)
quant<-qlogis(index, mu, sig) #theoretical quantiles of the logistic
plot( quant, sort(x), cex=1.0, pch=16, col="blue")
abline(b=1,a=0,col="red", lwd=3, lty=2)
```



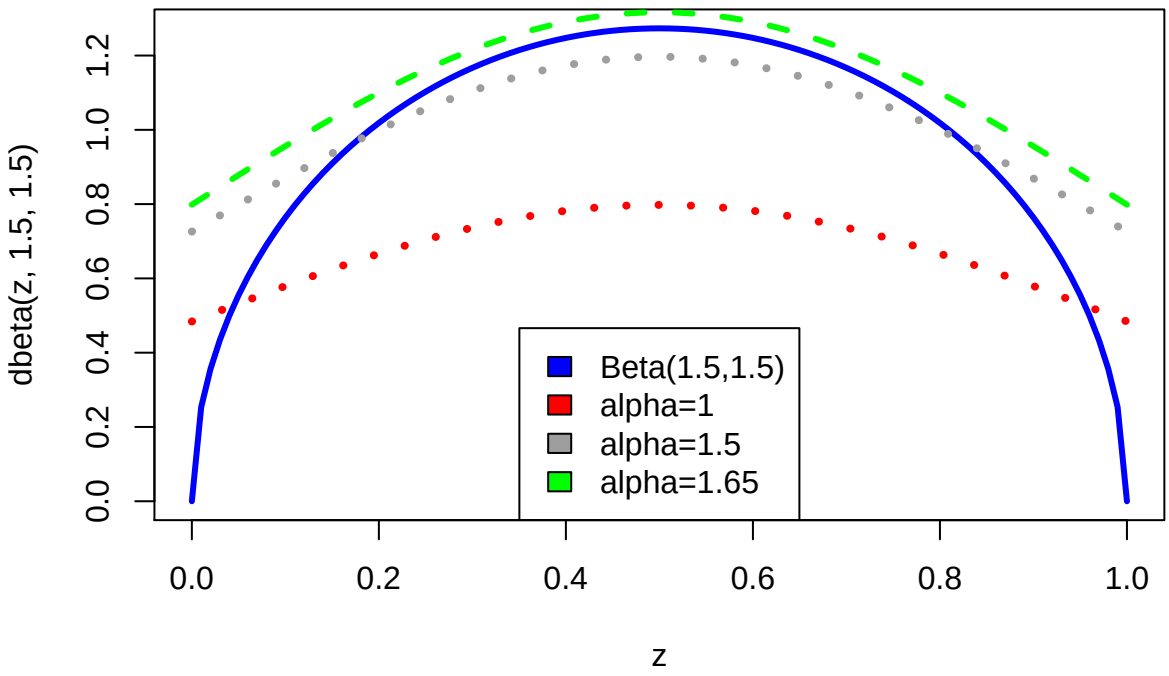
Same as before, the tails appear to be different from the theoretical values.

Exercise 3

Rejection Method

The blue line represents the Beta distribution. The other lines are normal distribution with several values of α.

```
z<-seq(0,1, 0.01)
plot(z, dbeta(z,1.5,1.5), type='l', lwd=3, col='blue')
lines(z, dnorm(z, 0.5, 0.5), col="red", lwd=4, lty=3)
lines(z, dnorm(z, 0.5, 0.5)*1.5, col=8, lwd=4, lty=3)
lines(z, dnorm(z, 0.5, 0.5)*1.65, col="green", lwd=3, lty=2)
legend("bottom", c("Beta(1.5,1.5)","alpha=1","alpha=1.5","alpha=1.65"),
       fill=c("blue","red", 8, "green"))
```



So, it seems that 1.65 is an acceptable values for α, since with it the normal distribution is strictly greater than the Beta.

```
alpha<- 1.65
```

Function

Here's the function. It accepts the sample size and the value of α (with provided default values):

```
beta.rand <- function(N=1000, alpha=1.65){
  y <- runif(N)
  u <- runif(N)
  index<- alpha*u < dbeta(y,1.5,1.5)
  xfinal <- y[index]
  return(xfinal)
}
```

Theory vs Results

According to the theory, the accepted probability is equal to $1/\alpha$, which is around 60%:

```
p<-1/alpha
p
```

```
## [1] 0.61
```

To verify if it is comparable to my results, I'll run my function **beta.rand** fir three times, with 3 different N (100, 1000, 10000). Every time, I'll compute the proportion of accepted values and all generated ones. For example, if I generate 100 values from the normal distribution, I expect that 60 will be accepted. I'll store the portion of accepted values in a variable called *count*

```
count<- numeric(3)
i<-1
for(T in c(100,1000,10000)){
  accepted.values<-beta.rand(N=T)
  count[i]<- length(accepted.values)/T
  i<- i+1
}
count
```

```
## [1] 0.68 0.63 0.60
```

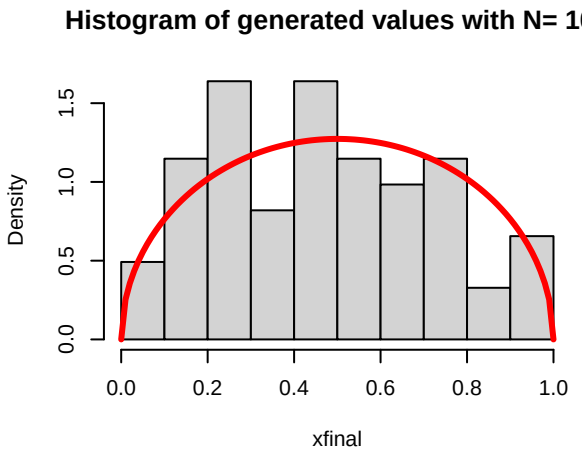
So, for every different N the proportion of accepted values is around 60%, which is exactly like the theory anticipated.

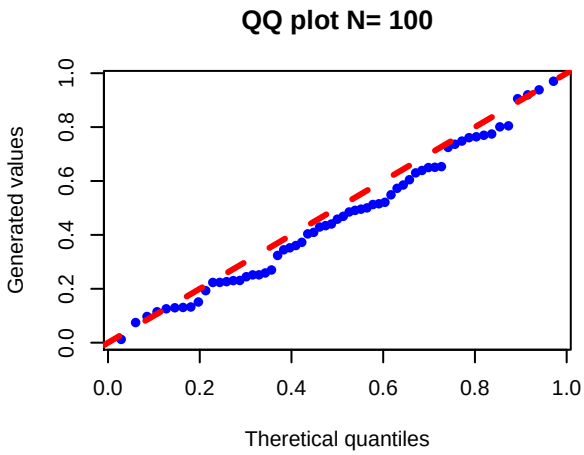
Graphical Comparison

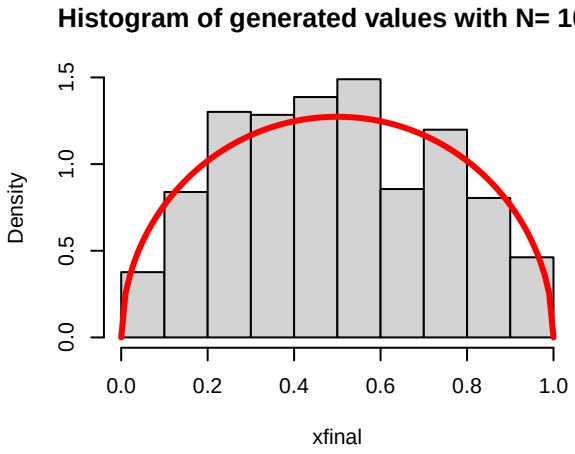
I'll plot the histograms and QQplots (computed with quantiles from the real beta distribution vs the generated values). In the first column, red line represents the theoretical values of the distribution.

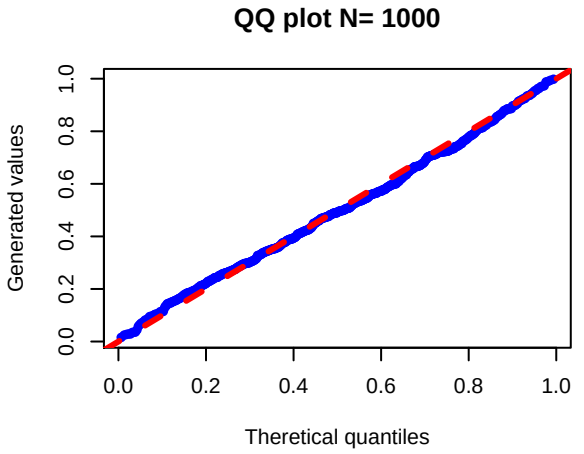
As we can see, the generator seems to be good: the bigger N is, the closer is the generated data to the theoretical values. With N=10000, the QQ plot is basically a perfect match and the histogram is very close to the real distribution.

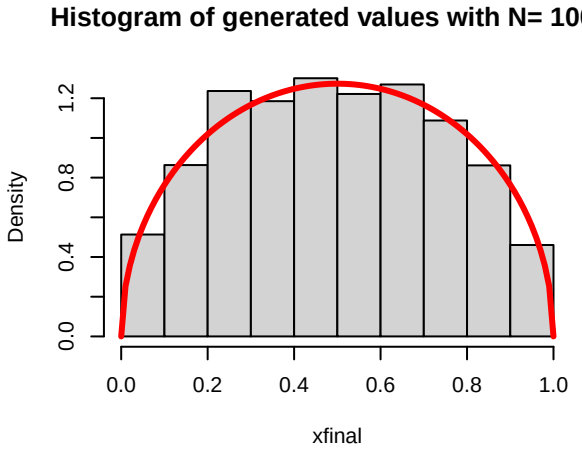
```
par(mfrow=c(3,2))
for(T in c(100,1000,10000)){
  xfinal<-beta.rand(N=T)
  hist(xfinal, col="lightgrey", probability=TRUE,
       main=paste("Histogram of generated values with N=",T))
  lines(z, dbeta(z,1.5,1.5), lwd=3, col='red')

  temp<-length(xfinal)
  index<- 1:temp/temp-1/(2*temp)
  quant<-qbeta(index,1.5,1.5)
  plot( quant, sort(xfinal), cex=1.0, pch=16, col="blue",
        main=paste("QQ plot N=",T), xlab="Theretical quantiles",
        ylab="Generated values")
  abline(b=1,a=0,col="red", lwd=3, lty=2)
}
```

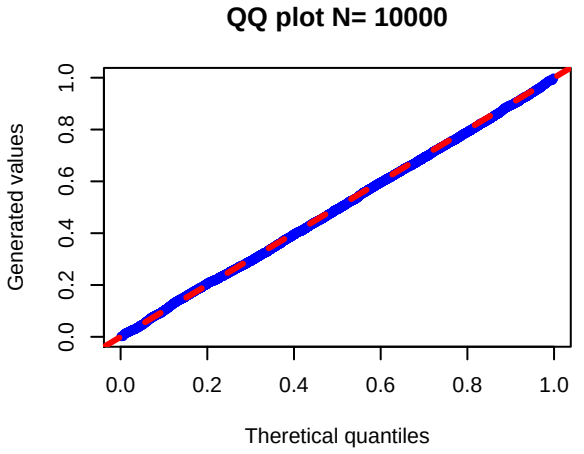

Histogram of generated values with N= 100
Density
xfinal
0.0
0.5
1.0
1.5
0.0
0.2
0.4
0.6
0.8
1.0


QQ plot N= 100
Generated values
Theretical quantiles
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of generated values with N= 1000
Density
xfinal
0.0
0.5
1.0
1.5
0.0
0.2
0.4
0.6
0.8
1.0


QQ plot N= 1000
Generated values
Theretical quantiles
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of generated values with N= 10000
Density
xfinal
0.0
0.4
0.8
1.2
0.0
0.2
0.4
0.6
0.8
1.0


QQ plot N= 10000
Generated values
Theretical quantiles
0.0
0.2
0.4
0.6
0.8
1.0