Tutorial 1 – Giovanni Filomeno - 12315325

1. Exercise 1

a) Compute by hand the resulting sequence of pseudo-random numbers

Since the part (b) asks for the period of the sequence, I will evaluate by hand sequence of pseudo-random numbers until I have again the same number. The general formula is $x_{n+1} = 7x_n(\text{mod } 31)$ with $x_0 = 19$.

- $x_1 = 7 \times 19 \text{ mod } 31 = 133 \text{ mod } 31 = 9$
- $x_2 = 7 \times 9 \text{ mod } 31 = 63 \text{ mod } 31 = 1$
- $x_3 = 7 \times 1 \text{ mod } 31 = 7 \text{ mod } 31 = 7$
- $x_4 = 7 \times 7 \text{ mod } 31 = 49 \text{ mod } 31 = 18$
- $x_5 = 7 \times 18 \text{ mod } 31 = 126 \text{ mod } 31 = 2$
- $x_6 = 7 \times 2 \text{ mod } 31 = 14 \text{ mod } 31 = 14$
- $x_7 = 7 \times 14 \text{ mod } 31 = 98 \text{ mod } 31 = 5$
- $x_8 = 7 \times 5 \text{ mod } 31 = 35 \text{ mod } 31 = 4$
- $x_9 = 7 \times 4 \text{ mod } 31 = 28 \text{ mod } 31 = 28$
- $x_{10} = 7 \times 28 \text{ mod } 31 = 196 \text{ mod } 31 = 10$
- $x_{11} = 7 \times 10 \text{ mod } 31 = 70 \text{ mod } 31 = 8$
- $x_{12} = 7 \times 8 \text{ mod } 31 = 56 \text{ mod } 31 = 25$
- $x_{13} = 7 \times 25 \text{ mod } 31 = 175 \text{ mod } 31 = 20$
- $x_{14} = 7 \times 20 \text{ mod } 31 = 140 \text{ mod } 31 = 16$
- $x_{15} = 7 \times 16 \text{ mod } 31 = 112 \text{ mod } 31 = 19$

b) What is the period of the sequence?

Since I obtain again 19 after 15 iterations, the period is 15.

c) Why does it not have maximum period $m - 1 = 30$?

Because the period $31 - 1 = 30$ is the theoretical maximum period given $m = 31$, however not every multiplier a is a primitive root of mod 31. Finding a period of 15 leads to the following congruence $7^{15} \equiv 1 \ (\text{mod } 31)$ which is true if I compute it by hand $(4747561509943 - 1)/31 = 153147145482$.

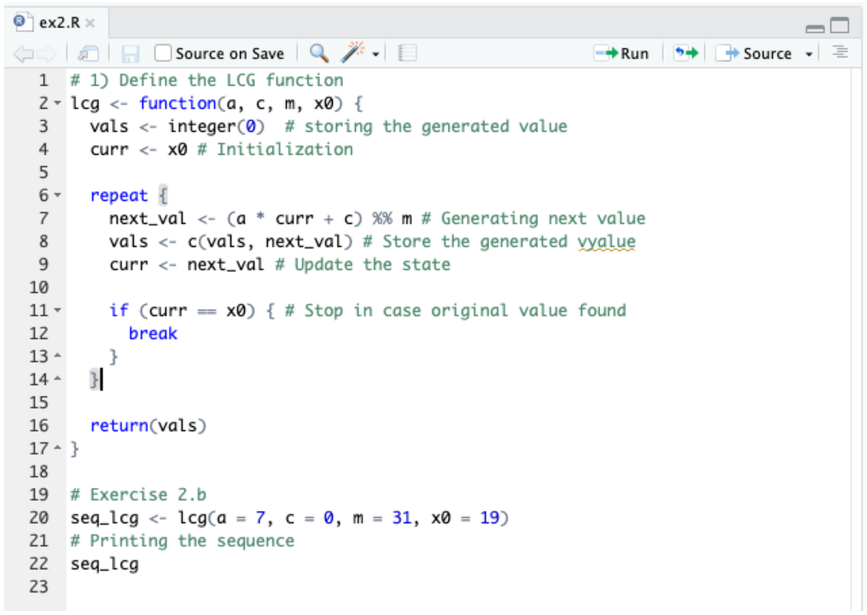
d) Find a value for a such that the generator has maximum period

I can use the formula $a^k \equiv 1 (\text{mod } 31)$ where $k = 30$ and I try some proper divisor of 30 for example $d = \{3, 5, 6, \ldots\}$. Doing this, I found that $a = 3$, which leads to the following formula $x_{n+1} = 3x_n (\text{mod } 31)$.

2. Exercise 2

a) Implement the linear congruential generator

The screenshot of the R-code is provided in Figure 1.

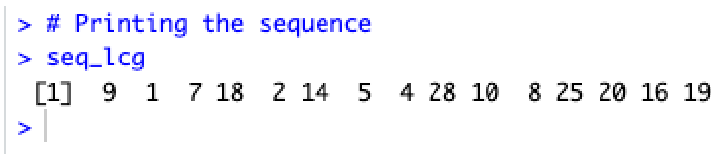


```r
# 1) Define the LCG function
lcg <- function(a, c, m, x0) {
  vals <- integer(0)  # storing the generated value
  curr <- x0 # Initialization

  repeat {
    next_val <- (a * curr + c) %% m # Generating next value
    vals <- c(vals, next_val) # Store the generated vyalue
    curr <- next_val # Update the state

    if (curr == x0) { # Stop in case original value found
      break
    }
  }

  return(vals)
}

# Exercise 2.b
seq_lcg <- lcg(a = 7, c = 0, m = 31, x0 = 19)
# Printing the sequence
seq_lcg
```

Figure 1: Linear Congruential Generator

b) Run the function using values from Exercise 1

The code in exercise 2.a gives the following sequence:



```
> # Printing the sequence
> seq_lcg
 [1]  9  1  7 18  2 14  5  4 28 10  8 25 20 16 19
> 
```

Figure 2: Sequence

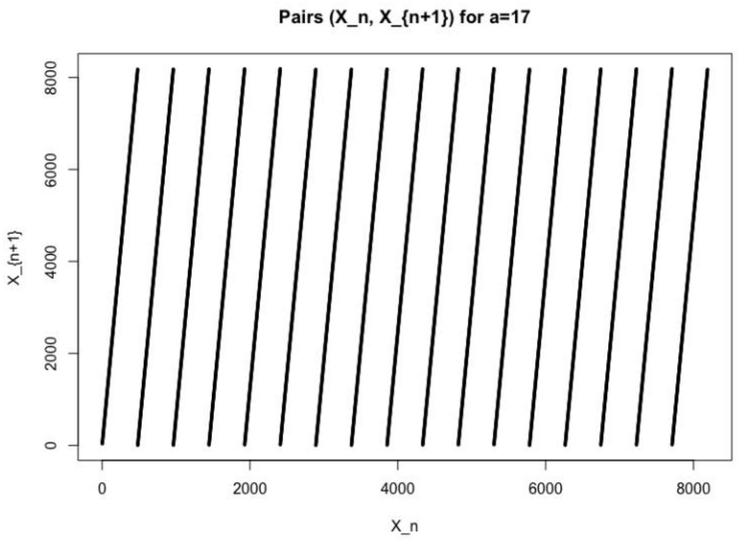
3. Exercise 3

a) What is the period of each generated sequence?

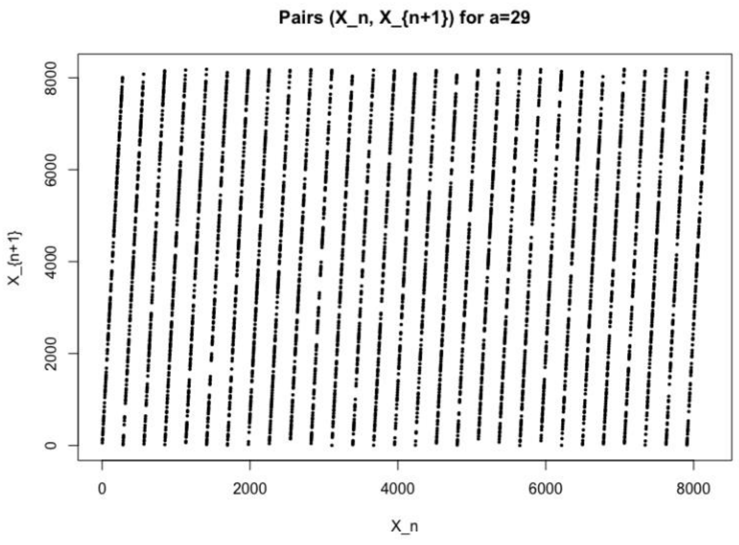
I used the same code of Exercise 2 three times, using the command *length()*. The results are: 8190 (A), 4095 (B), 4095 (C).

b) Plot all pairs of consecutive numbers and consider the auto-correlation for each of the three cases.

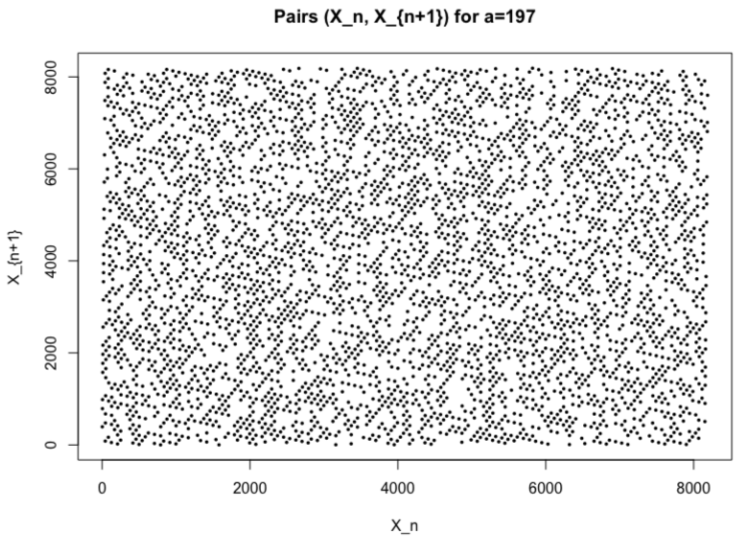
The Figure 3 shows the plot of all pairs of consecutive numbers for a=17, a=29, and a=197.



(a)



(b)



(c)

Figure 3: Plots of consecutive numbers

Additionally, as requested from the exercises, I also plot the auto-correlation, however, since the correlation in each sequence is pretty small, the three graphs look the same even though their periods differ (8190 vs 4095). Therefore, they do not add any meaningful information. However, just for sake of completeness, I also added in Figure 4 the plot coming from the function *acf()* for a=17.

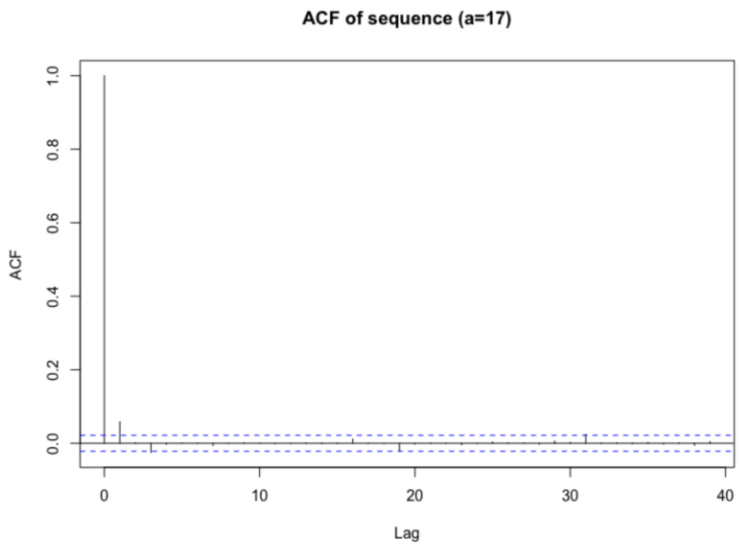


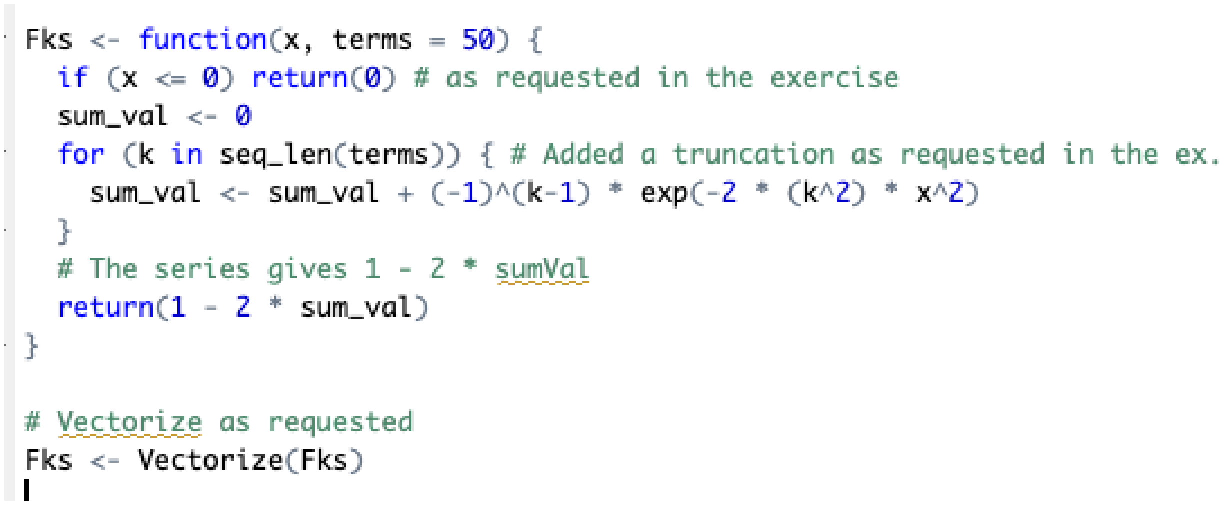
Figure 4: ACF plot for a=17

c) What is the best choice of multiplier a between (A), (B) and (C)?

I would take the multiplier of (A), a=17, since the period is larger, so it means a repetition comes later, even if visually it tends to form straight lines.

4. Exercise 4

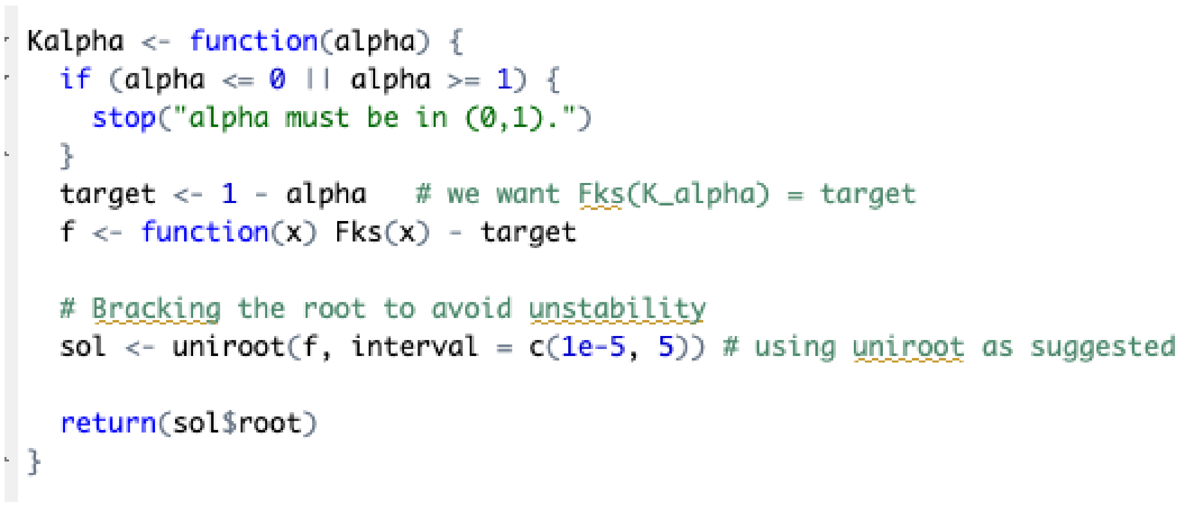
a) Write function *Fks(x)*



```
Fks <- function(x, terms = 50) {
  if (x <= 0) return(0) # as requested in the exercise
  sum_val <- 0
  for (k in seq_len(terms)) { # Added a truncation as requested in the ex.
    sum_val <- sum_val + (-1)^(k-1) * exp(-2 * (k^2) * x^2)
  }
  # The series gives 1 - 2 * sumVal
  return(1 - 2 * sum_val)
}

# Vectorize as requested
Fks <- Vectorize(Fks)
```

Figure 5: Fks(x) function

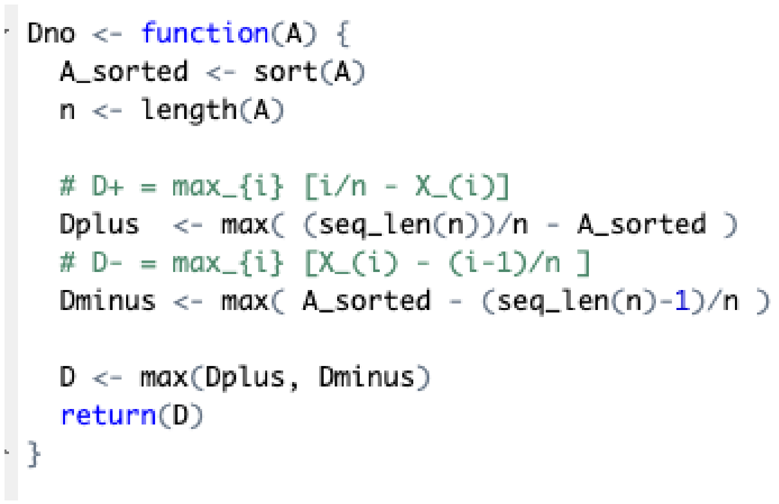
b) Define function *Kalpha(alpha)*



```r
Kalpha <- function(alpha) {
  if (alpha <= 0 || alpha >= 1) {
    stop("alpha must be in (0,1).")
  }
  target <- 1 - alpha   # we want Fks(K_alpha) = target
  f <- function(x) Fks(x) - target

  # Bracking the root to avoid unstability
  sol <- uniroot(f, interval = c(1e-5, 5)) # using uniroot as suggested

  return(sol$root)
}
```

Figure 6: Kalpha(alpha) function

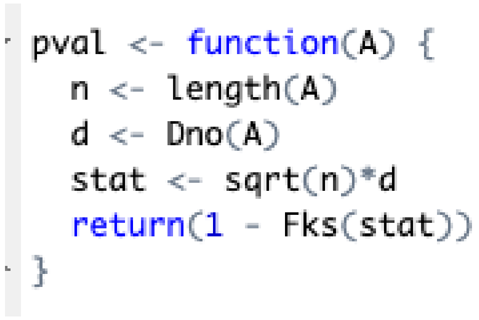
c) Write function *Dno(A)*



```r
Dno <- function(A) {
  A_sorted <- sort(A)
  n <- length(A)

  # D+ = max_{i} [i/n - X_(i)]
  Dplus  <- max( (seq_len(n))/n - A_sorted )
  # D- = max_{i} [X_(i) - (i-1)/n ]
  Dminus <- max( A_sorted - (seq_len(n)-1)/n )

  D <- max(Dplus, Dminus)
  return(D)
}
```

Figure 7: Dno(A) function

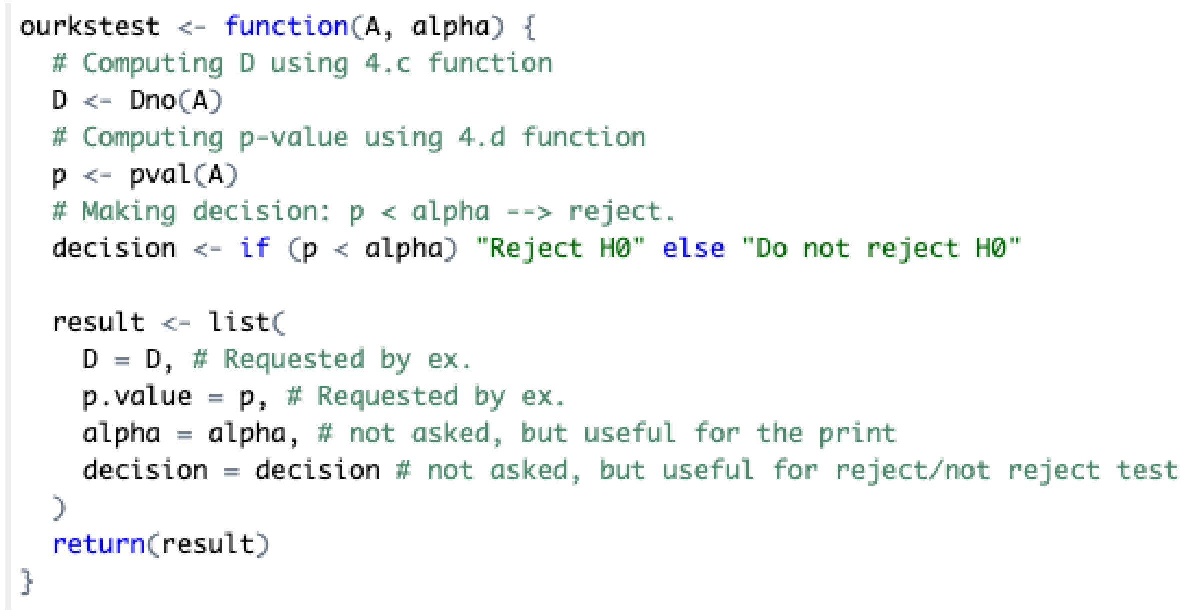
d) Write function *pval(A)*



```r
pval <- function(A) {
  n <- length(A)
  d <- Dno(A)
  stat <- sqrt(n)*d
  return(1 - Fks(stat))
}
```

Figure 8: Dno(A) function

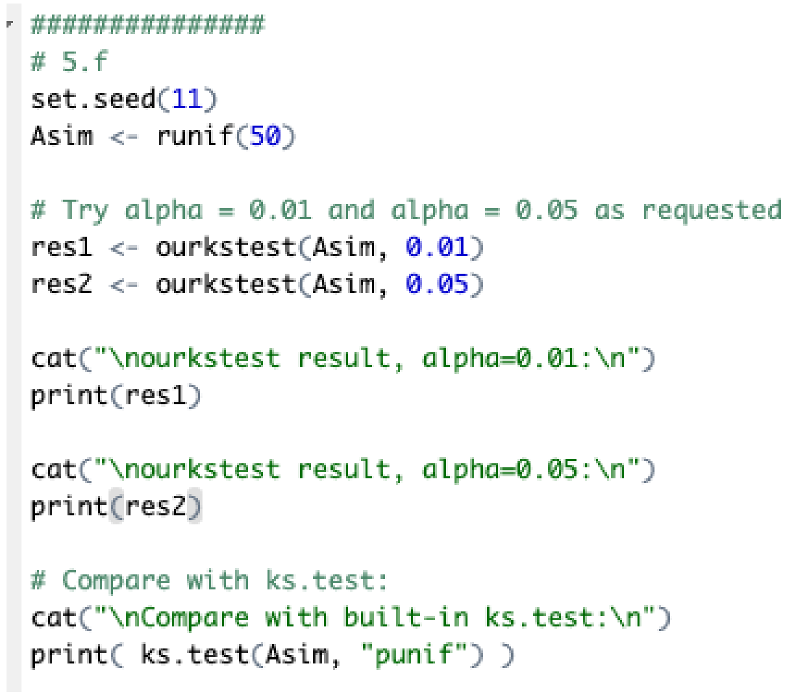
e) Write function *ourkstest(A,alpha)*



```r
ourkstest <- function(A, alpha) {
  # Computing D using 4.c function
  D <- Dno(A)
  # Computing p-value using 4.d function
  p <- pval(A)
  # Making decision: p < alpha --> reject.
  decision <- if (p < alpha) "Reject H0" else "Do not reject H0"

  result <- list(
    D = D, # Requested by ex.
    p.value = p, # Requested by ex.
    alpha = alpha, # not asked, but useful for the print
    decision = decision # not asked, but useful for reject/not reject test
  )
  return(result)
}
```

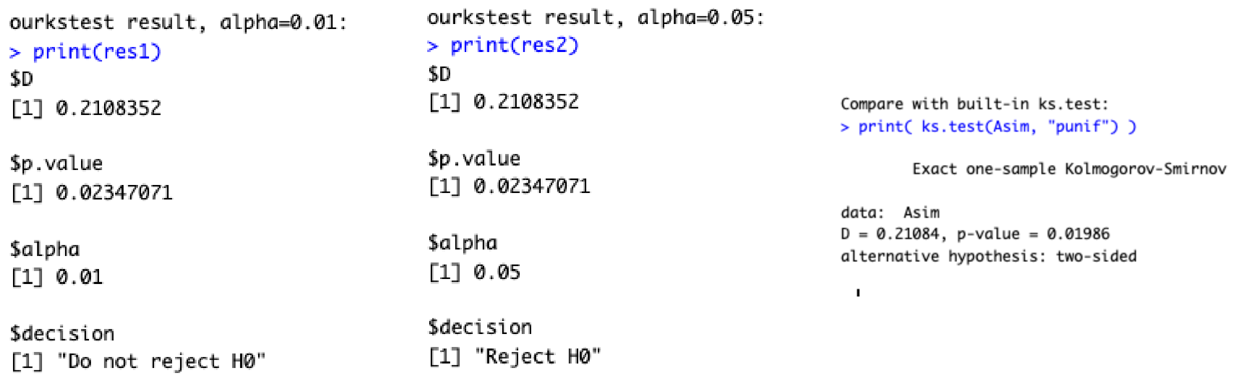
Figure 9: ourkstest(A,alpha) function

f) Run the code



```r
################
# 5.f
set.seed(11)
Asim <- runif(50)

# Try alpha = 0.01 and alpha = 0.05 as requested
res1 <- ourkstest(Asim, 0.01)
res2 <- ourkstest(Asim, 0.05)

cat("\nourkstest result, alpha=0.01:\n")
print(res1)

cat("\nourkstest result, alpha=0.05:\n")
print(res2)

# Compare with ks.test:
cat("\nCompare with built-in ks.test:\n")
print( ks.test(Asim, "punif") )
```

Figure 10: Run the two examples



```
ourkstest result, alpha=0.01:
> print(res1)
$D
[1] 0.2108352

$p.value
[1] 0.02347071

$alpha
[1] 0.01

$decision
[1] "Do not reject H0"
```

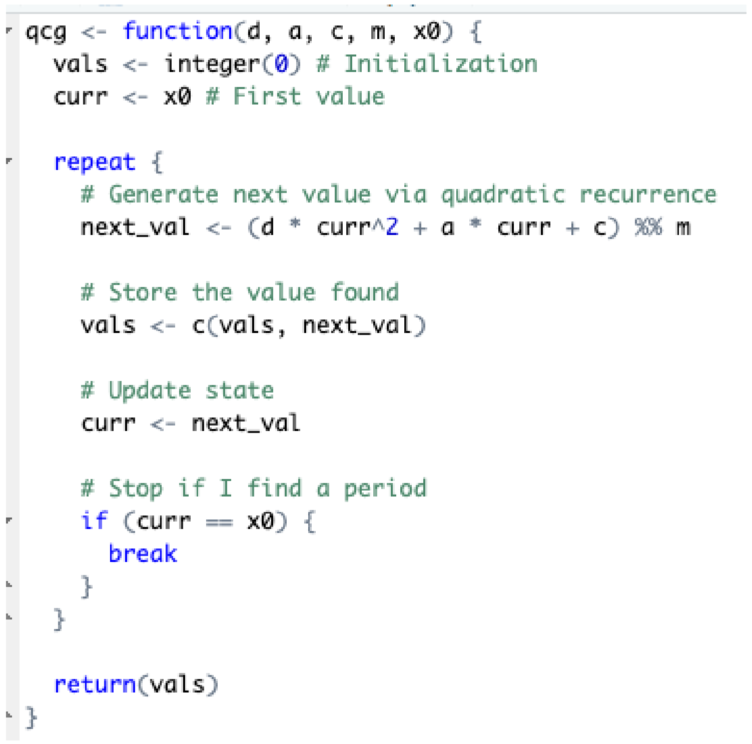
```
ourkstest result, alpha=0.05:
> print(res2)
$D
[1] 0.2108352

$p.value
[1] 0.02347071

$alpha
[1] 0.05

$decision
[1] "Reject H0"
```

```
Compare with built-in ks.test:
> print( ks.test(Asim, "punif") )

        Exact one-sample Kolmogorov-Smirnov

data:  Asim
D = 0.21084, p-value = 0.01986
alternative hypothesis: two-sided
```

Figure 11: Output of the previous code

My functions give in both tests the same final decision w.r.t. to buil-in function. At $\alpha = 0.01$ I do not reject H0 ($p \approx 0.0235$ for my functions, $p \approx 0.0199$ for the build-in. Both > 0.01). At $\alpha = 0.05$ I reject H0 ($p \approx 0.0235$ for my functions, $p \approx 0.0199$ for the build-in. Both < 0.05). The numerical difference comes from the truncation ($n = 50$) I have in the function.

5. Exercise 5



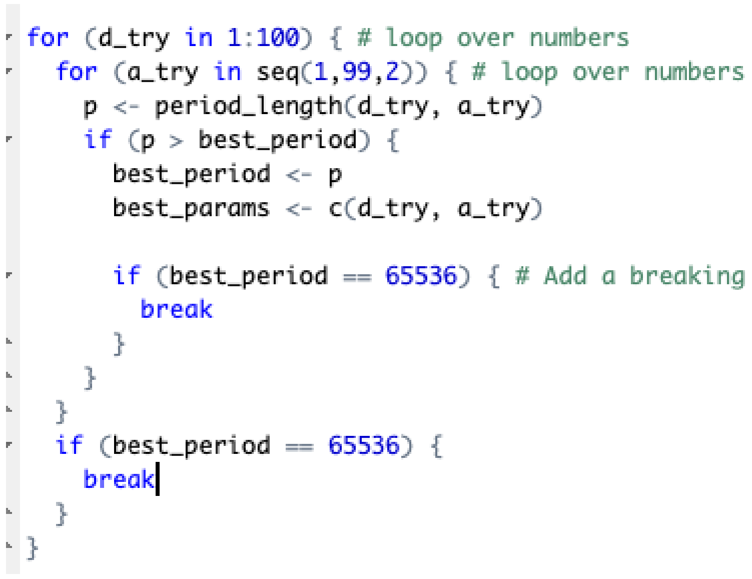
```r
qcg <- function(d, a, c, m, x0) {
  vals <- integer(0) # Initialization
  curr <- x0 # First value

  repeat {
    # Generate next value via quadratic recurrence
    next_val <- (d * curr^2 + a * curr + c) %% m

    # Store the value found
    vals <- c(vals, next_val)

    # Update state
    curr <- next_val

    # Stop if I find a period
    if (curr == x0) {
      break
    }
  }

  return(vals)
}
```

Figure 12: Quadratic Congruential Generator function

6. Exercise 6

a) Find *d* and *a*

To find the optimal parameters, I simply hard-coded a double for-loop (see Figure 13), which gives me $d = 2$ and $a = 3$.



```
for (d_try in 1:100) { # loop over numbers
  for (a_try in seq(1,99,2)) { # loop over numbers
    p <- period_length(d_try, a_try)
    if (p > best_period) {
      best_period <- p
      best_params <- c(d_try, a_try)

      if (best_period == 65536) { # Add a breaking
        break
      }
    }
  }
  if (best_period == 65536) {
    break
  }
}
```

Figure 13: Double for to find the parameters

b) Perform a test

After finding the requested *a* and *d*, I generated the sequence, and I normalized the sequence over *m* and then I applied the build-in Kolmogorov Smirnov test which gives me $D = 1.5259e - 05$ and $p - value = 1$.

c) Plot 1000 pairs

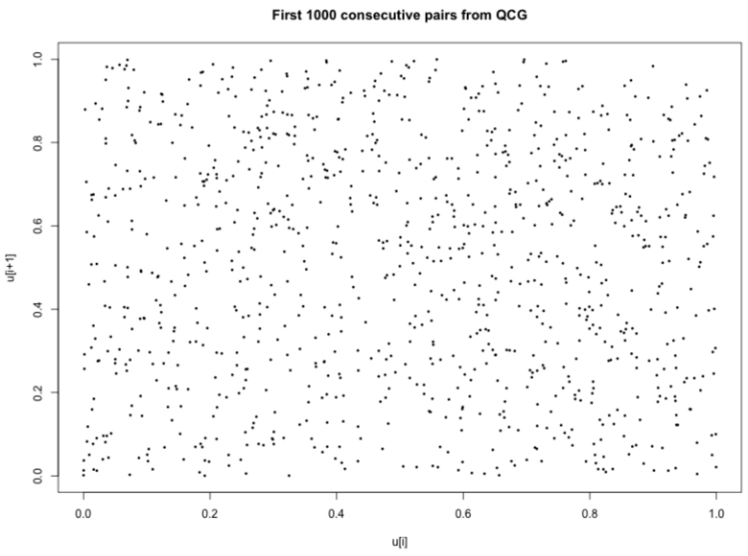


Figure 14: 1000 pairs of corresponding consecutive numbers