Homework 2

Zachary Lazerick

2023-02-07

1) Write a function that computes the first 100 Fibonacci numbers (i.e. $X_n = X_{n-1} + X_{n-2}$, with $X_1 = 1$ and $X_2 = 1$).

```
Fibonacci <- function(x) {
  Fibo_list <- c(1, 1)
  while(length(Fibo_list) < x) {
    Fibo_list[length(Fibo_list)+1] <- Fibo_list[length(Fibo_list)-1] +
      Fibo_list[length(Fibo_list)]
  }
  return(Fibo_list)
}

## Test for 1
Fibonacci(100)
```

```
##   [1] 1.000000e+00 1.000000e+00 2.000000e+00 3.000000e+00 5.000000e+00
##   [6] 8.000000e+00 1.300000e+01 2.100000e+01 3.400000e+01 5.500000e+01
##  [11] 8.900000e+01 1.440000e+02 2.330000e+02 3.770000e+02 6.100000e+02
##  [16] 9.870000e+02 1.597000e+03 2.584000e+03 4.181000e+03 6.765000e+03
##  [21] 1.094600e+04 1.771100e+04 2.865700e+04 4.636800e+04 7.502500e+04
##  [26] 1.213930e+05 1.964180e+05 3.178110e+05 5.142290e+05 8.320400e+05
##  [31] 1.346269e+06 2.178309e+06 3.524578e+06 5.702887e+06 9.227465e+06
##  [36] 1.493035e+07 2.415782e+07 3.908817e+07 6.324599e+07 1.023342e+08
##  [41] 1.655801e+08 2.679143e+08 4.334944e+08 7.014087e+08 1.134903e+09
##  [46] 1.836312e+09 2.971215e+09 4.807527e+09 7.778742e+09 1.258627e+10
##  [51] 2.036501e+10 3.295128e+10 5.331629e+10 8.626757e+10 1.395839e+11
##  [56] 2.258514e+11 3.654353e+11 5.912867e+11 9.567220e+11 1.548009e+12
##  [61] 2.504731e+12 4.052740e+12 6.557470e+12 1.061021e+13 1.716768e+13
##  [66] 2.777789e+13 4.494557e+13 7.272346e+13 1.176690e+14 1.903925e+14
##  [71] 3.080615e+14 4.984540e+14 8.065155e+14 1.304970e+15 2.111485e+15
##  [76] 3.416455e+15 5.527940e+15 8.944394e+15 1.447233e+16 2.341673e+16
##  [81] 3.788906e+16 6.130579e+16 9.919485e+16 1.605006e+17 2.596955e+17
##  [86] 4.201961e+17 6.798916e+17 1.100088e+18 1.779979e+18 2.880067e+18
##  [91] 4.660047e+18 7.540114e+18 1.220016e+19 1.974027e+19 3.194043e+19
##  [96] 5.168071e+19 8.362114e+19 1.353019e+20 2.189230e+20 3.542248e+20
```

2) Write a program that prints out all prime numbers from m to n. Use $m = 1$ and $n = 100, 500$, and 1000, but feel free to choose other values as well.

```
## Problem 2

PrimeFinder <- function(x, y) {
  i <- x; prime_list <- c();
  while (i <= y) {
    j <- 1; factor_list <- c();
      while (j <= i) {
        if (i %% j == 0) {
          factor_list <- append(factor_list, j)
          j <- j + 1
        }
        else {
          j <- j + 1
        }
      }
    if (length(factor_list) == 2) {
      prime_list <- append(prime_list, i)
      i <- i + 1
    }
    else {
      i <- i + 1
    }
  }
  print(prime_list)
}

## Test for 2

PrimeFinder(1, 100)
```

```
##  [1]  2  3  5  7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

```
PrimeFinder(1, 500)
```

```
##  [1]   2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67
## [20]  71  73  79  83  89  97 101 103 107 109 113 127 131 137 139 149 151 157 163
## [39] 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269
## [58] 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383
## [77] 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499
```

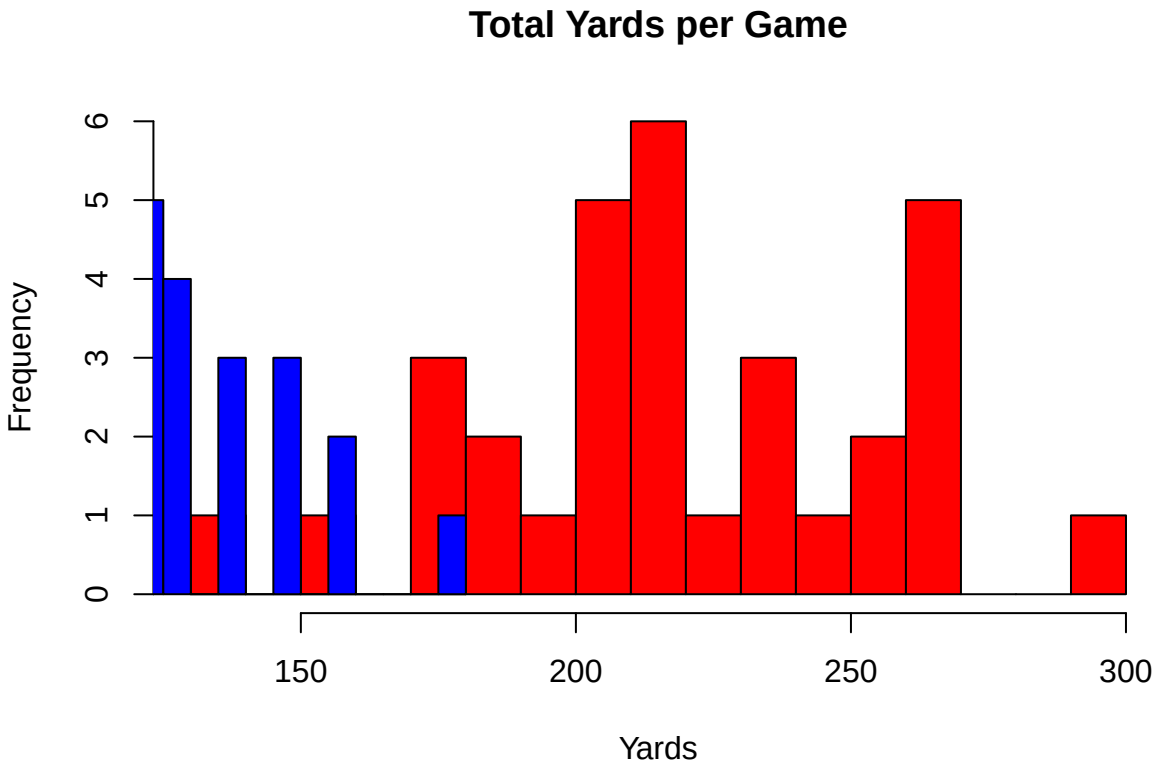
```
PrimeFinder(1, 1000)
```

```
##   [1]   2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61
##  [19]  67  71  73  79  83  89  97 101 103 107 109 113 127 131 137 139 149 151
##  [37] 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251
##  [55] 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359
##  [73] 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463
##  [91] 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593
## [109] 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701
## [127] 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827
## [145] 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953
## [163] 967 971 977 983 991 997
```

Graphing: 4) In honor of the Super Bowl, let's look at NFL team statistics from this past year. The .csv files that accompanies assignment contains offensive and defensive statistics for each of the 32 teams from this season. Load these two data sets in to R and create two data frames, one called 'offense' and the other 'defense'.

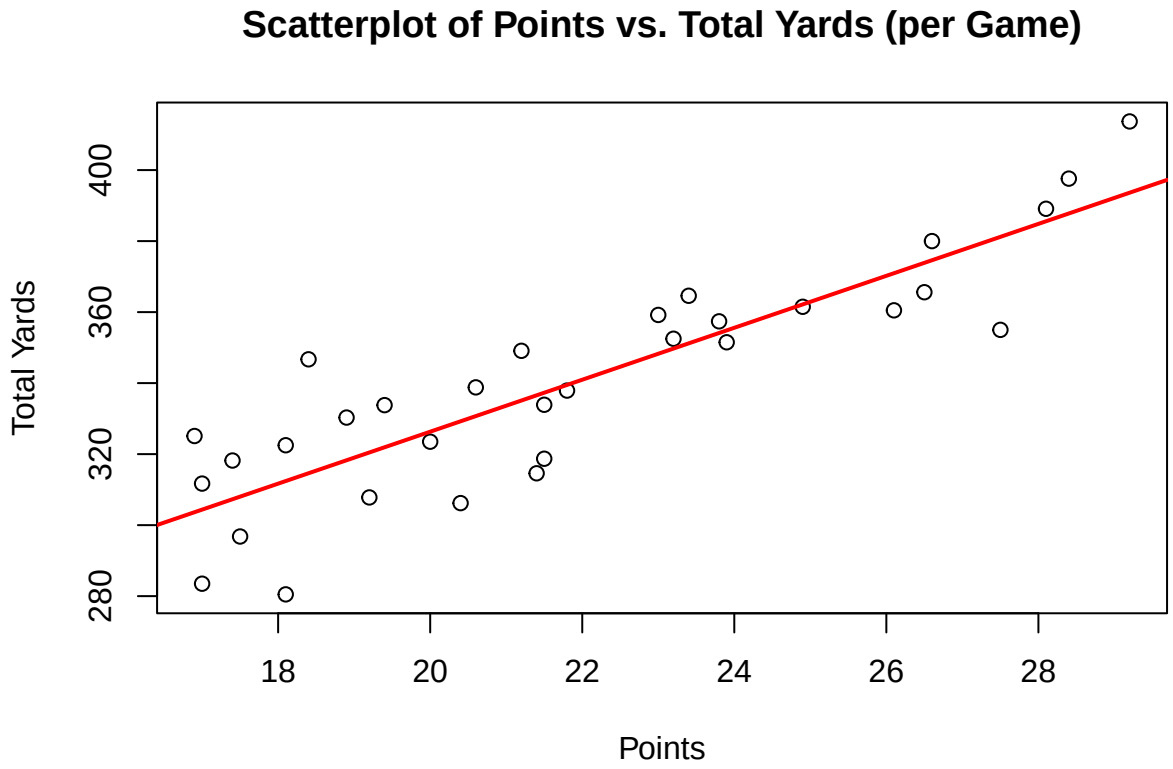
i) Construct a histogram that contains offensive rushing and passing yards per game on the same set of axes. The other option is to create a stacked histogram. Here, the idea is to make a histogram for the combined data set, and then "add" a second histogram to the sample graph for one of the individual data sets. It plots the second histogram on top of the first, making it look like a stacked histogram. For whichever option you choose, change the default bin size to be something more interesting to you. Label the x and y axis, and include a title for the graph.

```
## Build Hist. for Passing Yards
hist(NFL_Offense$PYds.G, col = 'red', breaks = 16,
     main = "Total Yards per Game",
     xlab = "Yards", ylab = "Frequency")

## Add Hist. for Rushing Yards to Hist. for Passing Yards
hist(NFL_Offense$RYds.G, add = T, col = 'blue', breaks = 16)
```



ii) Construct a scatterplot that compares offensive yards per game and points per game. Add a trendline to this graph and determine the R^2 value. Make sure to label your axes and give the graph a title.

```
## Scatterplot
plot(NFL_Offense$Pts.G, (NFL_Offense$PYds.G + NFL_Offense$RYds.G),
     main = "Scatterplot of Points vs. Total Yards (per Game)",
     xlab = "Points", ylab = "Total Yards")

## Build Regression Line
Pts_TotYds.G.lm <- lm((NFL_Offense$PYds.G + NFL_Offense$RYds.G)~
                        NFL_Offense$Pts.G)

## Add Regression Line to the Graph
abline(Pts_TotYds.G.lm, col = 'red', lwd = '2')
```

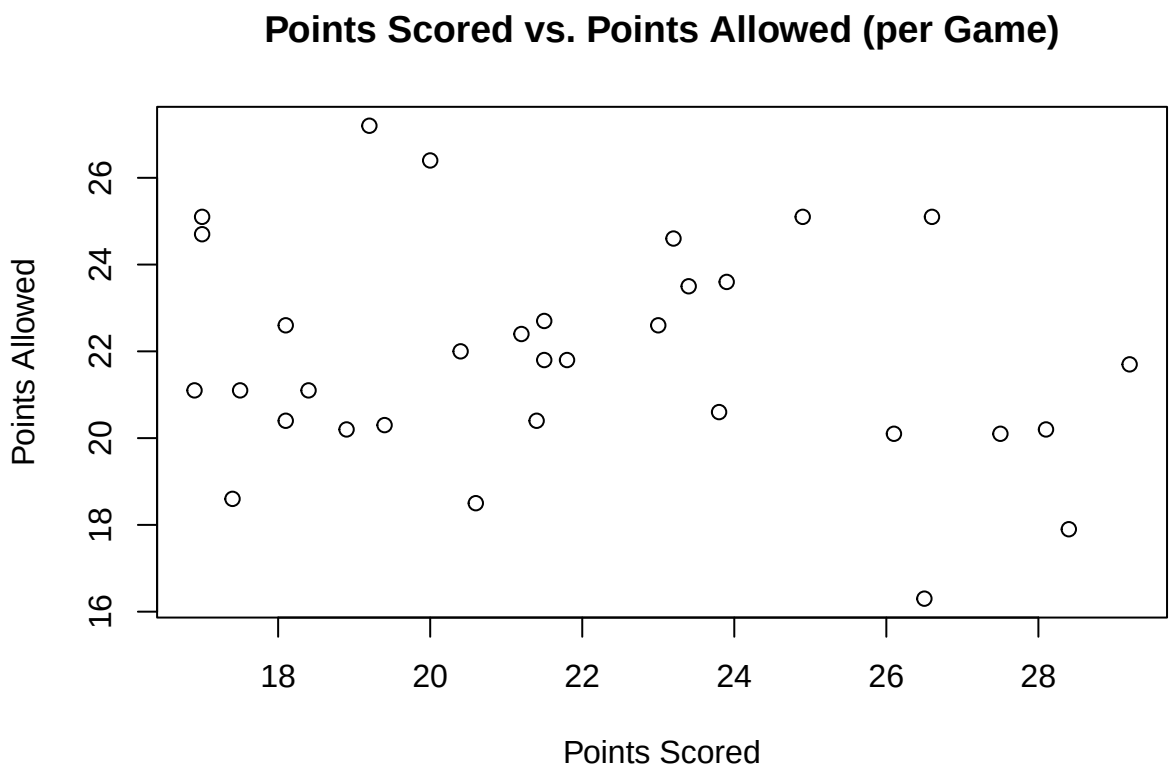



```
## Report the $R^2$ value
summary(Pts_TotYds.G.lm)$r.squared
```

```
## [1] 0.763792
```

iii) Construct a scatterplot which displays offensive points per game to defensive points per game. Do you notice any obvious patterns in the data set? What is the R^2 value here?

```
## Scatterplot
plot(NFL_Offense$Pts.G, NFL_Defense$Pts.G,
     main = "Points Scored vs. Points Allowed (per Game)",
     xlab = "Points Scored", ylab = "Points Allowed")
```



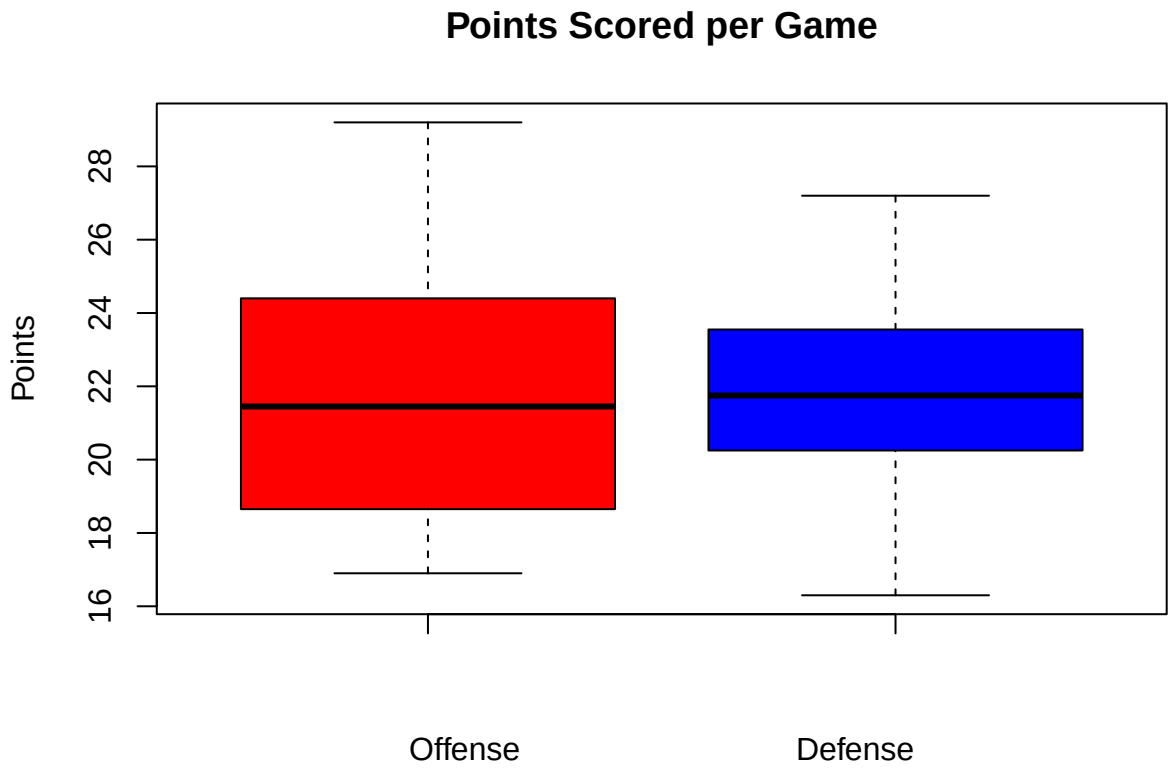

```
## Build Regression Model
Pts_Scored_Allowed.G.lm <- lm(NFL_Defense$Pts.G~NFL_Offense$Pts.G)

## Report $R^2$ Value
summary(Pts_Scored_Allowed.G.lm)$r.squared
```

```
## [1] 0.04236998
```

There are no obvious correlated patterns, however, on the extremes of the plot, the more points an offense can score, the less points the defense allows.

iv) Construct a boxplot which displays the total points scored/allowed for both offense and defense. Make sure to label your axes and give the graph a title. Based on this graph, are there any significant differences in the two data sets?

```
## Build Boxplot
## Red = Offense, Blue = Defense
boxplot(NFL_Offense$Pts.G, NFL_Defense$Pts.G, col = c('red','blue'),
        main = "Points Scored per Game",
        xlab = "Offense                                Defense",
        ylab = "Points")
```



The major difference between the two data sets is that the NFL_Defense Dataset has a tighter spread about the median compared to the NFL_Offense Dataset. Aside from that, the overall spread is not to much different.

5) From: An Introduction to Statistical Computing by Voss [Question E1.1 in textbook] Write a function to implement the LCG.

$$X_n = aX_{n-1} + b \pmod{m}$$

The function should take a length n, the parameters m, a, and b as well as the seed X_0 as input and should return a vector $X = (X_1, \ldots, X_n)$. Test your function by calling it with the parameters: (Note: Use $n = 10$ in all three cases.)

```
LCG <- function(n, m, a, b, x) {
  list_X <- c(x)
  while (length(list_X) <= n) {
    x <- list_X[length(list_X)]
    Next_X <- (a * x + b) %% m
    list_X <- append(list_X, Next_X)
  }
  print(list_X)
}

## Test 1 ($m = 8, a = 5$, and $b = 1$)
LCG(10, 8, 5, 1, 2)
```

```
##  [1] 2 3 0 1 6 7 4 5 2 3 0
```

```
## Test 2 ($m = 150, a = 3, b = 0, X_0 = 5$)
LCG(10, 150, 3, 0, 5)
```

```
##  [1]   5  15  45 135 105  15  45 135 105  15  45
```

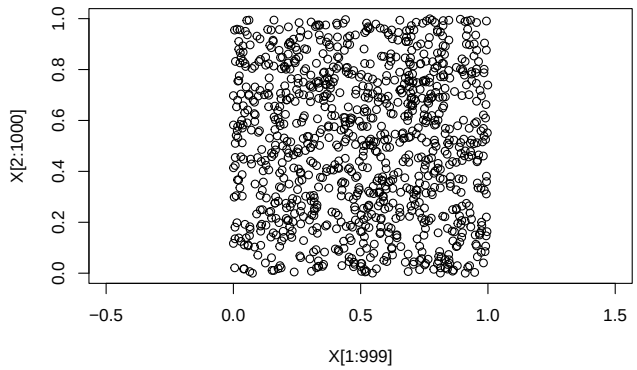
```
## Test 3 ($m = 200, a = 5, b = 7, X_0 = 3$)
LCG(10, 200, 5, 7, 3)
```

```
##  [1]   3  22 117 192 167  42  17  92  67 142 117
```

6) [Question E1.2 in textbook] Given a sequence $X_1, X_2, \ldots$ of Uniform (0,1) pseudo random numbers, we can use a scatterplot of $(X_i, X_i + 1)$ for $i = 1, \ldots, n - 1$ in order to try and assess whether the X_i are independent. [Note: R code for each part is given in textbook, E1.2]

a) Create such a plot with n = 1000 using the built-in random number generator for R. Can you explain the resulting plot?

```
X <- runif(1000)
plot(X[1:999], X[2:1000], asp=1)
```

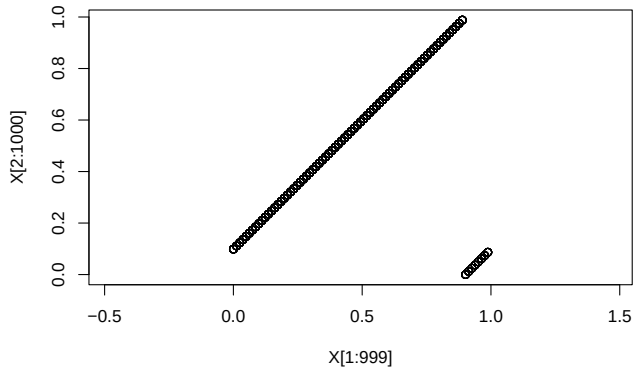


The resulting plot is plotting the pairs of points $(X_i, X_{i+1}$ for $i = 1, ..., 999$. What we want to see is a random scattering of points. This would mean that the base random number generator used by R generates independent random numbers. This is in fact what we see. This is an indicator that this is a 'good' random number generator.

b) Create a similar plot, using your function LCG from question 1, part (i). Discuss the resulting plot.

```
m <- 81; a <- 1; b <- 8; x <- 0
X <- LCG(1000, m, a, b, x)/m
```

```
plot(X[1:999], X[2:1000], asp=1)
```

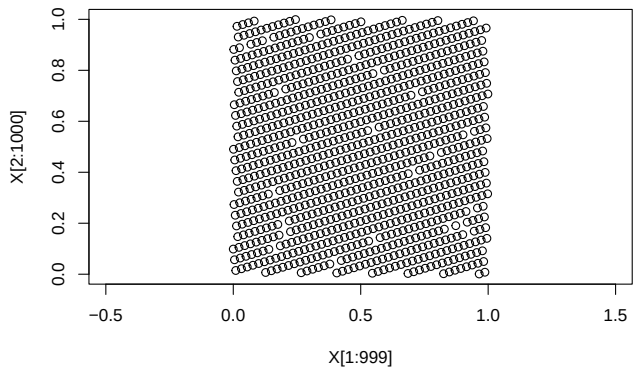


The resulting plot appears to be a linear line with a high degree of correlation. This indicates that the numbers generated using the LCG are not random enough because of the obvious pattern, thus we need to use different parameters to derive a 'better' random number generator using an LCG.

c) Repeat the experiment from (b) using the following parameters. Discuss the results.

```
## Test 1 ($m = 1024, a = 401, b = 101$)
m <- 1024; a <- 401; b <- 101; x <- 0
X <- LCG(1000, m, a, b, x)/m
```

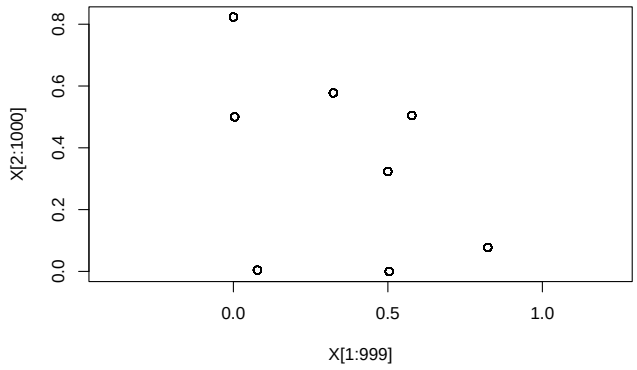
```
plot(X[1:999], X[2:1000], asp=1)
```



Although there appears to be an obvious structure in the construction of the plot, these parameters return the best random number generation using an LCG of the three sets of parameters tested. The pattern between the two points is not easily deducible, so these parameters would likely be random 'enough' for random number generation.

```
## Test 2 ($m = 232, a = 1664525, b = 1013904223$)
m <- 232; a <- 1664525; b <- 1013904223; x <- 0
X <- LCG(1000, m, a, b, x)/m
```

```
plot(X[1:999], X[2:1000], asp=1)
```



This plot however, is the worst. The period of this LCG is 8. Of a possible 232 unique values to hit, this combination of parameters only loops through 3.44% of them. This is not suitable for random number generation. These two plots highlight how effective LCGs can be used to simulate random numbers, and that not all combinations of parameters are built the same.

7) From Simulation by Sheldon Ross, Chapter 3, [Question 12 in book] For uniform (0,1) random variables $U_1, U_2, \ldots$ define:

$$N = minimum\{n : \Sigma_{i=1}^{n} U_i > 1\}$$

That is, N is equal to the number of random numbers that must be summed to exceed 1.

a) Estimate E(N) by generating 100 values of N.

```
N <- rep(0, 100); set.seed(3418)
for (i in 1:100) {
  Sum = 0; count = 0; X <- runif(1000)
  while (Sum < 1) {
    count = count + 1; Sum = Sum + X[count]
  }
  N[i] <- count
}
sum(N)/100
```

```
## [1] 2.77
```

b) Estimate E(N) by generating 1000 values of N.

```
N <- rep(0, 1000); set.seed(2218)
for (i in 1:1000) {
  Sum = 0; count = 0; X <- runif(1000)
  while (Sum < 1) {
    count = count + 1; Sum = Sum + X[count]
  }
  N[i] <- count
}
sum(N)/1000
```

```
## [1] 2.72
```

c) Estimate E(N) by generating 10000 values of N.

```
N <- rep(0, 10000); set.seed(1018)
for (i in 1:10000) {
  Sum = 0; count = 0; X <- runif(1000)
  while (Sum < 1) {
    count = count + 1; Sum = Sum + X[count]
  }
  N[i] <- count
}
sum(N)/10000
```

```
## [1] 2.7156
```

d) What do you think is the value of E(N)?

I think the value of E(N) is Euler's number, $e = 2.73$.

8) [Question 13 in book] Let U_i, for $i \geq 1$, be random numbers. Define N by

$$N = maximum\{n : \prod_{i=1}^{n} U_i \geq e^{-3}\}$$

where $\prod_{i=1}^{0} U_i = 1$.

a) Find E(N) by simulation

```
## Define Storage Vector
N <- c(); set.seed(1201)

for (i in 1:100) {
  n <- 0; prod <- 1; X <- runif(1000);
  while (prod >= exp(-3)) {
    n <- n + 1
    prod <- prod * X[n]
  }
  N <- append(N, n)
}

## Report E(N) for 100 Trials
sum(N)/100
```

```
## [1] 4.12
```

b) Find $P(N = i)$ for $i = 0, 1, 2, 3, 4, 5, 6$, by simulation

```
N <- c(); set.seed(1209)

## Expand to 1000 Trials
for (i in 1:1000) {
  n <- 0; prod <- 1; X <- runif(1000);
  while (prod >= exp(-3)) {
    n <- n + 1
    prod <- prod * X[n]
  }
  N <- append(N, n)
}

table(N)
```

```
## N
##   1   2   3   4   5   6   7   8   9  10  11  12
##  52 125 218 234 192  88  53  27   6   2   1   2
```

$P(X = 0) = 0$ because we need to generate at least one random uniform integer to compare our product against e^{-3}.

9) [Question 14 in textbook] With $X_1 = 23$ and $X_2 = 66$, define $X_n = 3X_{n-1} + 5X_{n-2} \pmod{100}$ for $n \geq 3$. Find the first 15 values of $U_n = X_n/100$.

```
## Initialize the First 2 Values of the List
X_list <- c(23, 66)

## Iterate List until Length = 15 (i.e. First 15 numbers)
while (length(X_list) < 15) {
  Next_X <- (3 * X_list[length(X_list) - 1] + 5 *
               X_list[length(X_list)]) %% 100
  X_list <- append(X_list, Next_X)
}

## Print List
X_list
```

```
##  [1] 23 66 99 93 62 89 31 22  3 81 14 13  7 74 91
```

```
## Convert List to the interval [0, 1)
U_list <- X_list / 100

## Print Converted List
U_list
```

```
##  [1] 0.23 0.66 0.99 0.93 0.62 0.89 0.31 0.22 0.03 0.81 0.14 0.13 0.07 0.74 0.91
```

10) From Simulation by Sheldon Ross, Chapter 4 [Question 3 in book] Write an efficient algorithm to simulate the value of a random variable X such that: $P(X = 1) = 0.3, P(X = 2) = 0.2, P(X = 3) = 0.35, P(X = 4) = 0.15$

Simulate 1000 values from this distribution and using the approach we talked about in class to simulate random variables from an arbitrary distribution. Check your results against the given probabilities by using the table command. Repeat this problem using the built-in R function, sample(). Do the two approaches give similar results?

```
## Build 1000 Uniform X on [0, 1)
set.seed(831); X <- runif(1000)
## Store Results in Y
Y <- rep(1, 1000)

for (i in 1:length(X)) {
  if ((X[i] >= 0) & (X[i] < .3)) {
    Y[i] <- 1
  }
  else if ((X[i] >= .3) & (X[i] < .5)) {
    Y[i] <- 2
  }
  else if ((X[i] >= .5) & (X[i] < .85)) {
    Y[i] <- 3
  }
  else if ((X[i] >= .85) & (X[i] < 1)) {
    Y[i] <- 4
  }
  else {
    Y[i] <- 0
  }
}

## Simulated Probabilities
table(Y)/1000
```

```
## Y
##     1     2     3     4
## 0.277 0.218 0.361 0.144
```

```
## Sampled Probabilities
set.seed(2031)
Sample <- sample(x = c(1, 2, 3, 4), size = 1000, replace = T,
                 prob = c(.3, .2, .35, .15))
table(Sample)/1000
```

```
## Sample
##     1     2     3     4
## 0.326 0.188 0.349 0.137
```

The two approaches do give roughly similar results. The simulated data is off the given probabilities by about 0.058 while the sampled data is off by about 0.052. So, the sampled data stays slightly truer to the specified probabilities.

11) (Non-Textbook Question) Suppose that balls are successively distributed among 8 urns with each ball being equally likely to be put in any of these urns. What is the probability that there will be exactly 3 empty urns after 9 balls have been distributed?

Steps:

i) Generate 9 random integers from 1 to 8, with repeats allowed. This represents the urn that each ball is placed into

ii) Count the number of unique integers

iii) Repeat a large number of times (1000+)

iv) Estimate the probability of 3 empty urns as the number of iterations where there were 6 unique integers in the sample (i.e. 6 urns with balls in them) divided by the total number of trials.

```
## Count how many iterations have 3 empty urns
count = 0; set.seed(2047)

## Simulate 1000 Times
for (i in 1:1000) {
  X <- sample(x = c(1, 2, 3, 4, 5, 6, 7, 8), size = 9, replace = T)
  Urn_list <- c()
  for (j in 1:length(X)) {
    ## Check if Urn_list already
    if (X[j] %in% Urn_list == F) {
      Urn_list <- append(Urn_list, X[j])
      j <- j + 1
    }
    else {
      j <- j + 1
    }
  }
  ## With 8 Urns, If length(Urn_list) == 5, then 3 urns are empty
  if (length(Urn_list) == 5) {
    count = count + 1
  }
}

##Report Estimated Probability of Exactly 3 Empty Urns
count/1000
```

```
## [1] 0.362
```