Statistical Analysis, MSCA 31007, Lecture 2

Hope Foster-Reyes

October 7, 2016

Contents

Simulation of Random Variables

Notes

- *Style Guide:* https://google.github.io/styleguide/Rguide.xml
- *Packages required: random, compositions, randtests, randtoolbox.*
- *Files required: ScratchOffMonteCarlo.rda; store in RStudio project directory*

Part 1. Generate uniformly distributed random numbers

1.1. Use runif(), short for random (r) unified distribution (unif).

```r
# Simulate n pseudo-random numbers uniformly distributed on [a,b]
set.seed(15)
sample.runif <- runif(n = 1000, min = 0, max = 1)
str(sample.runif)
```

```
##  num [1:1000] 0.602 0.195 0.966 0.651 0.367 ...
```

```r
head(sample.runif)
```

```
## [1] 0.6021140 0.1950439 0.9664587 0.6509055 0.3670719 0.9888592
```

```r
quantile(sample.runif)
```

```
##            0%           25%           50%           75%          100%
## 0.00001273188 0.26783758850 0.52099344938 0.76198713941 0.99802551256
```

1.2. Simulate Uniform Random Sample on [0,1] Using Random.org.

```r
# Load random package to interface with random.org
library(random)
```

```r
# Download binary sequence from random.org
num.flips <- 1000
data.random.org <- randomNumbers(n = num.flips, min = 0, max = 1,
                                 col = 1, base = 2, check = TRUE)
head(data.random.org)
```

```
##      V1
## [1,]  0
## [2,]  1
## [3,]  0
## [4,]  0
## [5,]  1
## [6,]  0
```

Let us turn our sequence of {0,1} into uniform random numbers on [0,1].

```r
# Load compositions package
suppressMessages(library(compositions))
```

```r
# Create function that interprets a sequence of zeros and ones of length n as a binary
#   number and converts that binary into decimal
BinaryToDec <- function(binary.seq){
  # Argument binary.seq: vector containing a binary sequence of 0's and 1's
  unbinary(paste(binary.seq, collapse = ""))
}

# Demonstrate function
BinaryToDec(c(1,1,1,1,1,0))
```

```
## [1] 62
```

```r
# Turn the sequence of 0's and 1's from random.org into a 100x10 matrix
matrix.binary <- matrix(data.random.org, ncol = 10)
head(matrix.binary)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    1    0    1    1    1    1    1    1     0
## [2,]    1    1    1    0    0    0    0    1    1     0
## [3,]    0    1    1    0    1    1    0    1    0     1
## [4,]    0    1    0    0    1    1    0    0    0     0
## [5,]    1    1    1    1    0    0    1    1    0     1
## [6,]    0    1    0    0    1    1    1    0    1     0
```

```r
str(matrix.binary)
```

```
##  int [1:100, 1:10] 0 1 0 0 1 0 0 0 0 0 ...
```

```r
# Transform each row into decimal format
vector.decimal <- apply(matrix.binary, 1, BinaryToDec)
str(vector.decimal)
```

```
##  num [1:100] 382 902 437 304 973 314 490 391 177 82 ...
```

```r
# Divide the numbers by 2^10 to make real numbers in [0,1]
sample.random.org <- vector.decimal / 2^10
str(sample.random.org)
```

```
##  num [1:100] 0.373 0.881 0.427 0.297 0.95 ...
```

```r
sample.random.org
```

```
##   [1] 0.373046875 0.880859375 0.426757812 0.296875000 0.950195312
##   [6] 0.306640625 0.478515625 0.381835938 0.172851562 0.080078125
##  [11] 0.488281250 0.178710938 0.733398438 0.704101562 0.295898438
##  [16] 0.215820312 0.863281250 0.022460938 0.079101562 0.913085938
##  [21] 0.040039062 0.969726562 0.653320312 0.313476562 0.227539062
##  [26] 0.728515625 0.631835938 0.160156250 0.373046875 0.777343750
##  [31] 0.294921875 0.545898438 0.062500000 0.452148438 0.742187500
##  [36] 0.322265625 0.822265625 0.428710938 0.188476562 0.302734375
##  [41] 0.743164062 0.607421875 0.606445312 0.284179688 0.773437500
##  [46] 0.615234375 0.369140625 0.251953125 0.875000000 0.691406250
##  [51] 0.082031250 0.773437500 0.763671875 0.533203125 0.108398438
##  [56] 0.404296875 0.673828125 0.481445312 0.240234375 0.545898438
##  [61] 0.195312500 0.506835938 0.759765625 0.655273438 0.844726562
##  [66] 0.741210938 0.642578125 0.169921875 0.632812500 0.063476562
##  [71] 0.390625000 0.743164062 0.234375000 0.977539062 0.358398438
##  [76] 0.027343750 0.334960938 0.034179688 0.826171875 0.572265625
##  [81] 0.457031250 0.387695312 0.282226562 0.247070312 0.541015625
##  [86] 0.432617188 0.799804688 0.177734375 0.519531250 0.837890625
##  [91] 0.692382812 0.099609375 0.002929688 0.343750000 0.890625000
##  [96] 0.651367188 0.326171875 0.586914062 0.801757812 0.354492188
```

```
quantile(sample.random.org)
```

```
##          0%         25%         50%         75%        100%
## 0.002929688 0.274658203 0.454589844 0.710205078 0.977539062
```

All numbers in sample.random.org are between 0 and 1. This is our equivalent of the sample obtained by runif().

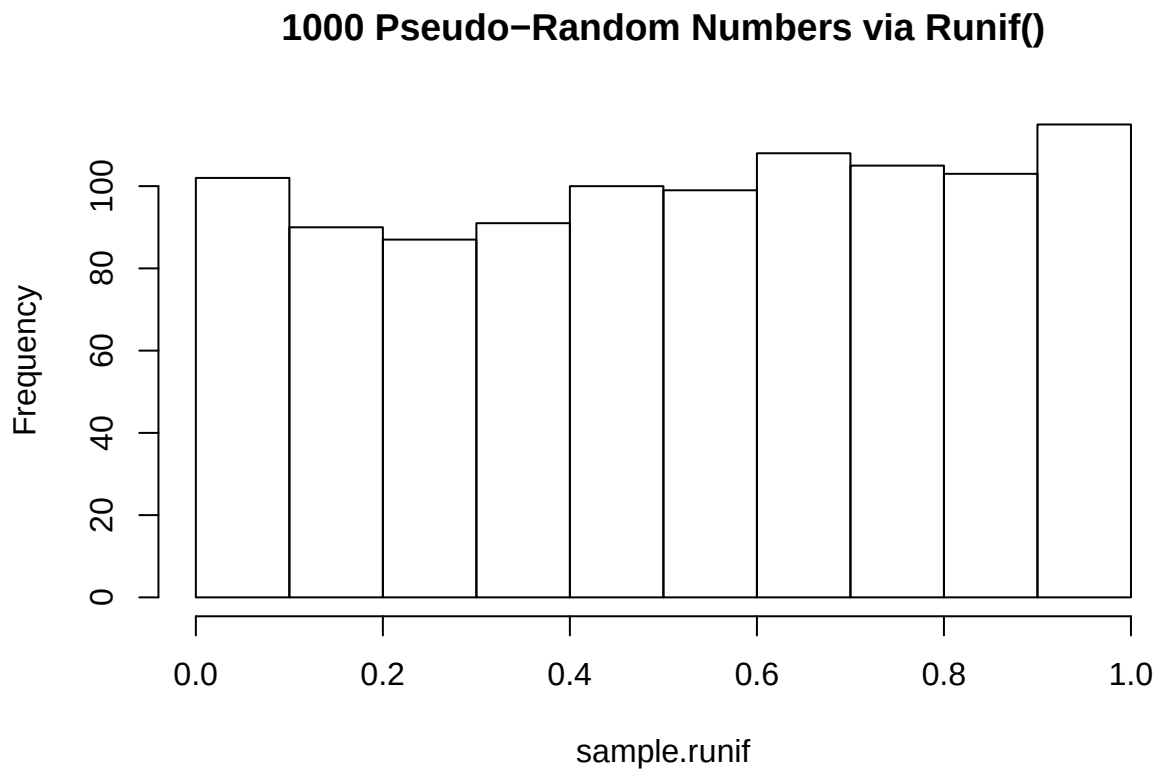
Part 2. Test random number generator

2.1. Test uniformity of distribution of both random number generators

2.1.1. Sample obtained by runif()

Analyze what was simulated.

```
title.runif <- "1000 Pseudo-Random Numbers via Runif()"
hist.runif <- hist(sample.runif, main = title.runif)
```




```
hist.runif
```

```
## $breaks
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
##
## $counts
##  [1] 102  90  87  91 100  99 108 105 103 115
##
```

```
## $density
##  [1] 1.02 0.90 0.87 0.91 1.00 0.99 1.08 1.05 1.03 1.15
##
## $mids
##  [1] 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85 0.95
##
## $xname
## [1] "sample.runif"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

What does the histogram tell you about the distribution? Is it consistent with the goal of simulation?

Our goal was to simulate 1000 random numbers having a continuous uniform distribution between [0,1]. We have stored these 1000 numbers in a sample vector and graphed a histogram of this vector to examine whether we were successful.

As seen in the histogram, when we group our 1000 numbers into bins of width one tenth (0.1), each of these bins holds approximately 100 numbers, with some variation. This is consistent with our goal (i.e. $1000/10 = 100$ and $1/10 = 0.1$). Our sample of continuous numbers appears to be evenly distributed between the bounds of 0 and 1, as expected for a distribution of $X \sim \text{Unif}(0,1)$, and appears consistent with the goal of our simulation.

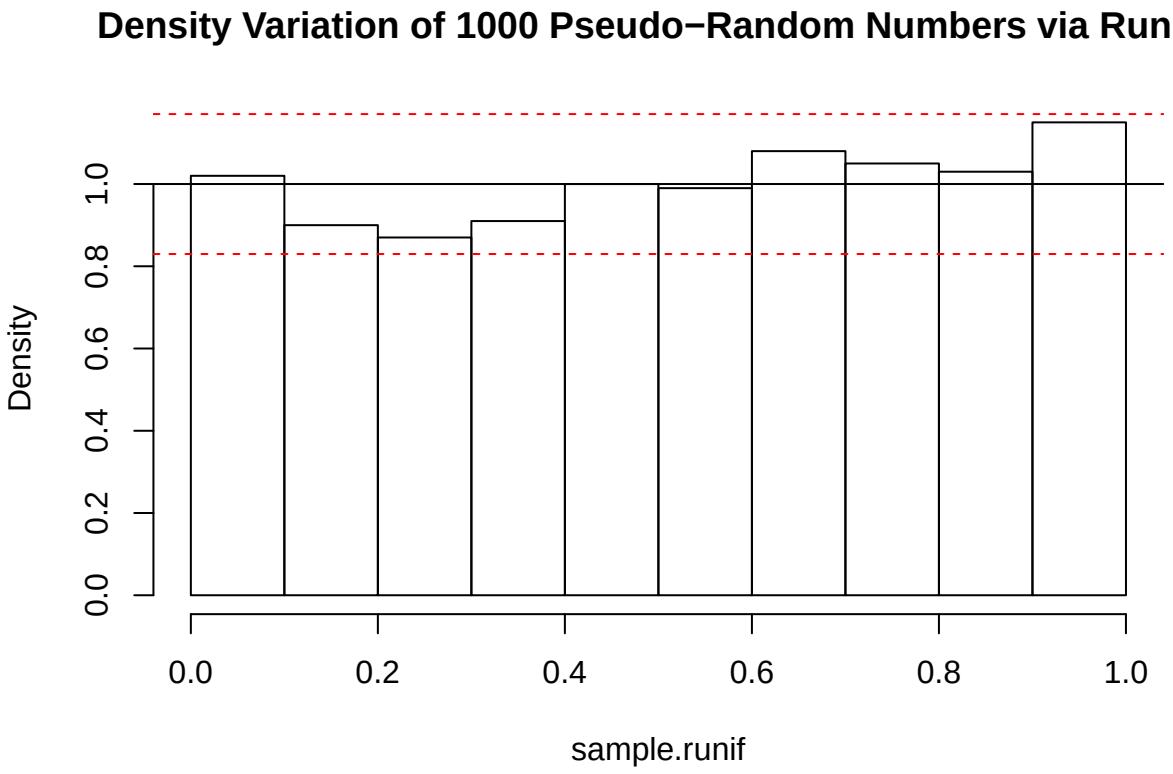
```
# Estimate mean and standard deviation of our sample's density. This is not the parameters
#   of the sample itself, but represent the central tendency and spread of the frequency
#   of the values within the sample. In other words, not how much the values themselves
#   vary, but how much the frequency of the values vary or are similar.
(hist.runif.mean <- mean(hist.runif$density))
```

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

```
(hist.runif.sd <- sd(hist.runif$density))
```

```
## [1] 0.08679478
```

```
# Plot our histogram again, adding mean and standard deviation to assist in interpreting
#   spread of the bins.
plot(hist.runif, freq = FALSE, main = paste("Density Variation of", title.runif))
abline(h = hist.runif.mean)
abline(h = hist.runif.mean + 1.96 * hist.runif.sd, col = "red", lty = 2)
abline(h = hist.runif.mean - 1.96 * hist.runif.sd, col = "red", lty = 2)
```



What does the graph tell you about the observed distribution?

The density of a histogram is calculated for each bin by dividing the number of observations in that bin by the width of the bin. Here, we can compare the density of our sample histogram's bins with the theoretical probability density of a uniform distribution.

We know that for a theoretical continuous uniform distribution, the probability density of each individual value inside its bounds is 1 divided by the distance between its maximum and minimum value. Thus for such a distribution between 0 and 1, the density for every value within its bounds will be 1 / (1-0), or simply 1.

For a sample taken from a continuous uniform distribution, we therefore would expect each of our bins to have a density close to 1, since the probability density of such a distribution remains at 1 throughout its range of values.

In other words, we know that our goal is a unified distribution on [0,1]. In a unified distribution on [0,1], as the size of the sample increases, the frequency of the outcomes should approach being equal, and their density should approach 1.

Since the densities of our sample (read, the densities of the bins of our histogram) should approach 1 and be equal, their mean of course should also approach 1 and the density across values (the bins) should not deviate excessively.

Examining the empirical density of our sample as diagrammed on our histogram, we can see that the central tendency of our 10 bins has a mean of 1, and further our bin densities are within one standard deviation of 1. Thus the central tendency and variation of these bins are consistent with a continuous uniform distribution on [0,1] and our goal.

```
# Estimate the moments of the sample.
(sample.runif.mean <- mean(sample.runif))
```

```
## [1] 0.5161848
```

```
(sample.runif.var <- var(sample.runif))
```

```
## [1] 0.08413663
```

What do you conclude about the estimated distribution from the moments?

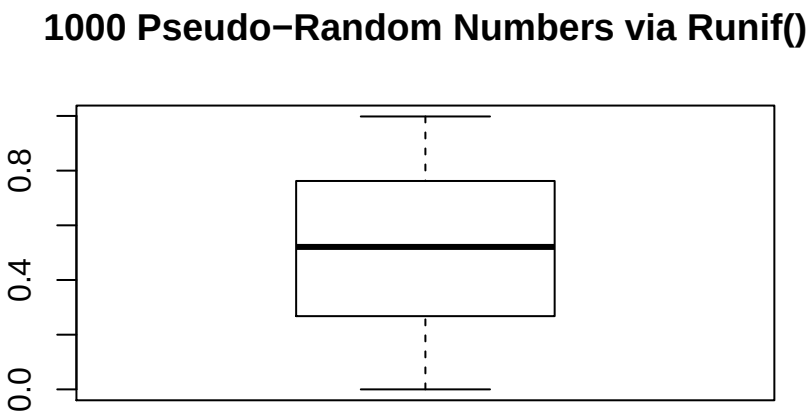
In contrast with the above exercise, we now look at the central tendency and spread of the values (the data in our sample) themselves. The first moment (mean) and the second central moment (variance) tell us, respectively, the center of our sample and the spread of our sample. These are also consistent with our goal.

For a uniform distribution we expect that the center of our data will be midway through the range of the distribution. Thus the sample mean of 0.5161848 is quite close to our expected theoretical mean of 0.5 (midway between 0 and 1, or $\frac{(1-0)}{2}$. We expect that the variance of our data will be $\frac{(b-a)^2}{12}$, or 1/12, and our sample variance of 0.0841366 is quite close to this.

```
# Check the summary of the simulated sample.
summary(sample.runif)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## 0.0000127 0.2678000 0.5210000 0.5162000 0.7620000 0.9980000
```

```
boxplot(sample.runif, main = title.runif)
```



What do you think is the best way of estimating uniform distribution over unknown interval?

Cosma Shalizi of Carnegie Mellon University explains that the histogram is a rudimentary way of examining a distribution. "If we hold the bins fixed and take more and more data, then by the law of large numbers we anticipate that the relative frequency for each bin will converge on the bin's probability."

We have used three different methods of examining our sample data:

- Examining the histogram of the data
- Examining the spread of the histogram's bins
- Examining the moments of the data itself
- Examining the interquartile range of the data (IQR)

If we define **"estimating a distribution"** as estimating, for a given data set, both its distribution (which distribution the data appears to take) and its parameters (the parameters for the distribution we suspect), are these methods helpful?

How can we confirm that data is uniform?

For any uniform distribution we know that the frequency and the density of its values should be uniform across its range. This is a distinct distribution, so eyeballing the histogram gives us a good sense of whether we can suspect a uniform distribution.

We have gone one step further by calculating the mean and standard deviation of our histogram's bins. In contrast with the mean and standard deviation of the data itself, these tell us about the extent to which our bin frequency varies from its expected mean. This method seems particularly useful for identifying a uniform distribution as it answers the question, "Is the frequency across all values spread narrowly?" If our frequency varies widely across our data points (i.e. the bins of our histogram), we know that the distribution cannot be uniform.

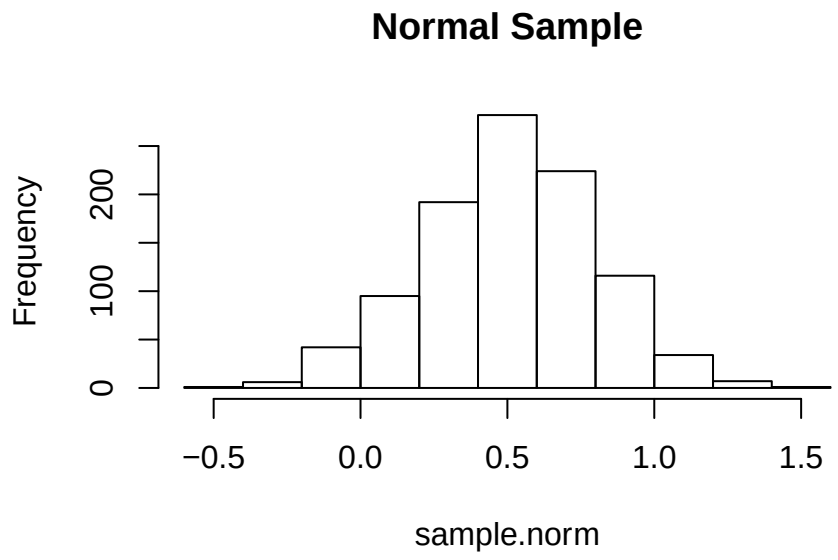
Thus, for identifying a uniform distribution, this is a more useful value than calculating the mean or standard deviation of the data itself. A uniform distribution spread across an unknown interval could have a very large standard deviation. But the standard deviation of its density (e.g. how much does the frequency of the bins themselves vary) will remain narrow.

The mean and standard deviation of the data itself can still give us a hint, as we do know how to calculate these values for a uniform distribution on any interval. However they alone cannot distinguish a uniform distribution from other famiies of distributions. This is demonstrated by the normal distribution below, which has a mean and standard deviation that matches our distribution on Unif(0,1).

```
sample.norm <- rnorm(1000, mean = 0.5, sd = sqrt(1/12))
summary(sample.norm)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.4639  0.3228  0.5063  0.5089  0.7160  1.4920
```

```
hist.norm <- hist(sample.norm, main = "Normal Sample")
```




```
(paste("Unif mean: ", sample.runif.mean))
```

```
## [1] "Unif mean:  0.51618475308083"
```

```
(paste("Unif variance: ", sample.runif.var))
```

```
## [1] "Unif variance:  0.0841366331278668"
```

```
(paste("Normal mean: ", mean(sample.norm)))
```

```
## [1] "Normal mean:  0.508865766901445"
```

```
(paste("Normal variance: ", var(sample.norm)))
```

```
## [1] "Normal variance:  0.0862176727881579"
```

To identify a uniform distribution, it appears we need to look at not just the moments of the sample itself but also the variation of the calculated frequencies or densities across the data points. However, in this case we are looking at only the first moment and the second central moment. The entire set of moments and the moment generating function likely paint a more distinguishing picture.

How can we estimate its parameters?

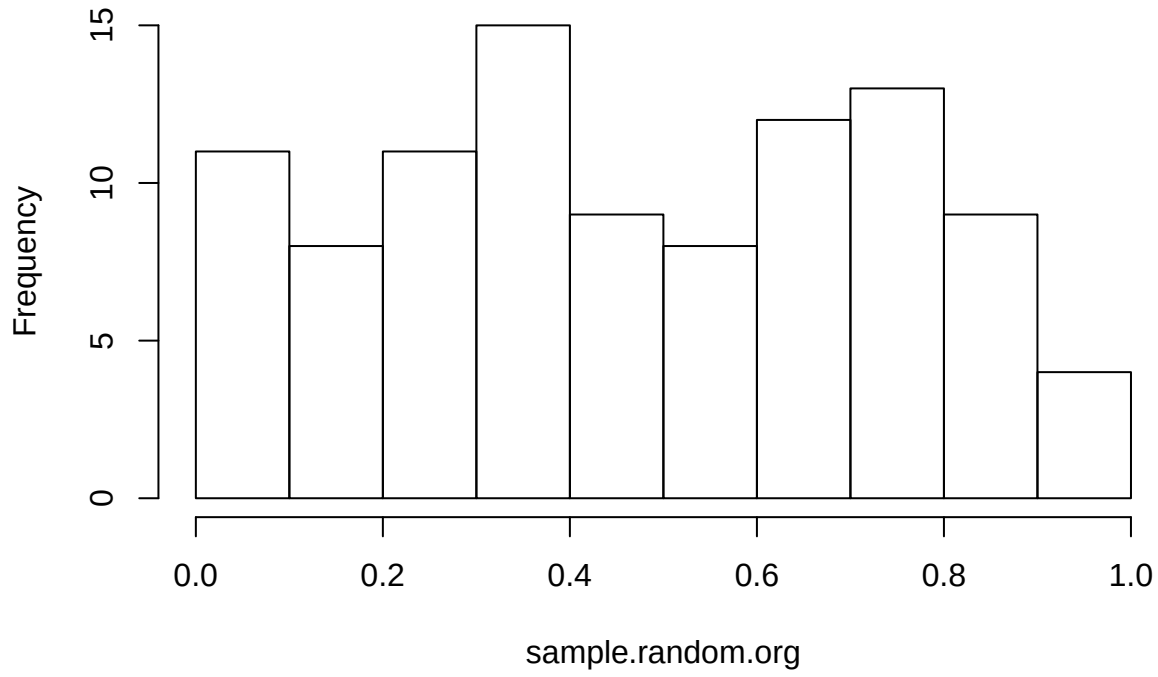
Estimating the parameters of a suspected uniform distribution is a different problem. We cannot estimate the minimum and maximum of a distribution in the same way that we identified the distribution. Knowing that the density of our histogram's bins is centered narrowly around its mean will be true for any uniform distribution of any interval.

However, if we know the minimum, maximum, mean and standard deviation of the *sample*, it seems we can come close to estimating the minimum and maximum of its hypothesized population distribution. We may need to make some assumptions regarding just how far extended from the bounds of our sample the distribution in question may be.

2.1.2. Repeat the same steps to test uniformity of the sample created from Random.org data.

Analyze what was simulated.

```
title.random.org <- "100 Pseudo-Random Numbers,\nTransformed from Random.org Binary Data"
hist.random.org <- hist(sample.random.org, main = title.random.org)
```




```
hist.random.org
```

```
## $breaks
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
##
## $counts
##  [1] 11  8 11 15  9  8 12 13  9  4
##
## $density
##  [1] 1.1 0.8 1.1 1.5 0.9 0.8 1.2 1.3 0.9 0.4
##
## $mids
##  [1] 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85 0.95
##
## $xname
## [1] "sample.random.org"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

What does the histogram tell you about the distribution? Is it consistent with the goal of simulation?

In this case, our goal was to simulate 100 random numbers having a continuous uniform distribution between [0,1], and we have used a binary sequence from a physical source, provided by the website Random.org, to produce our sample.

As seen in the histogram, when we group our 100 numbers into bins of width one tenth (0.1), each of these bins holds between 5 and 15 numbers. We expect each bin to hold approximately 10 numbers (i.e. $100/10 = 10$ and $1/10 = 0.1$). Our smaller sample based on Random.org appears to hold more variation, but the central tendency of our bin frequency, which appears to be approximately 10 from observing the histogram, is consistent with our goal.

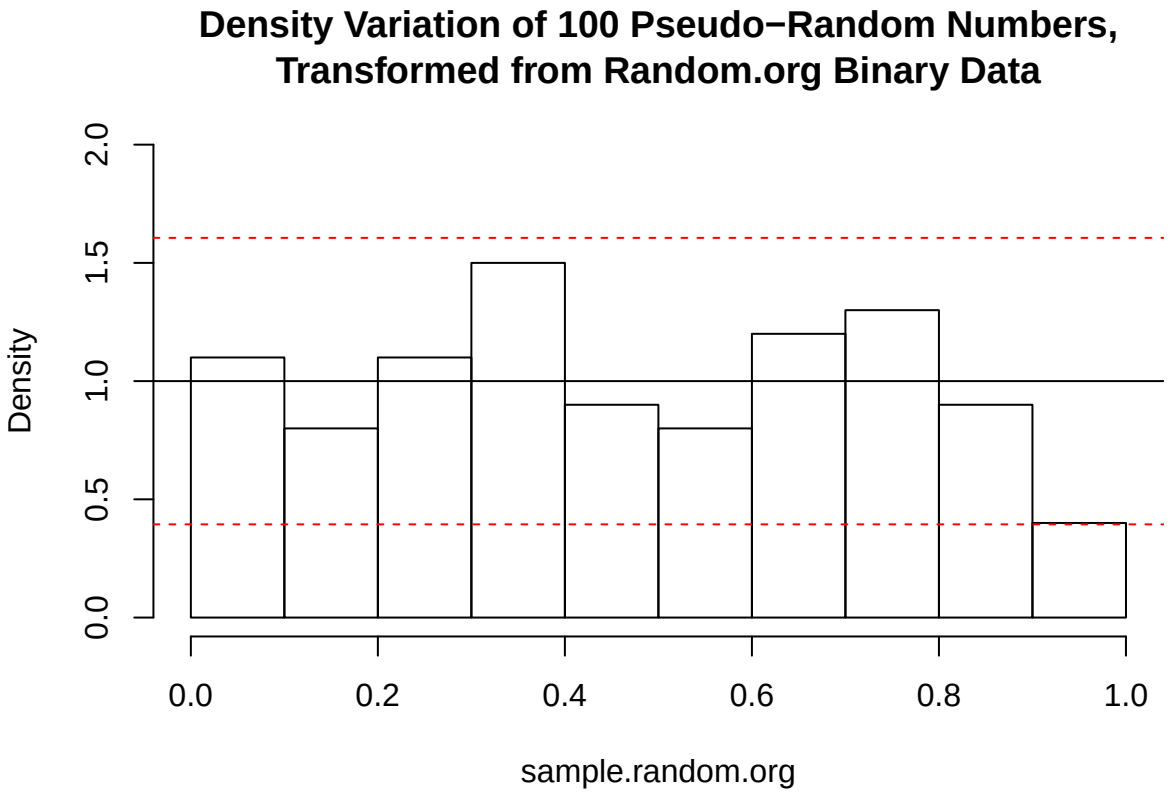
```
# Estimate mean and standard deviation of our sample's density.
(hist.random.org.mean <- mean(hist.random.org$density))
```

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

```
(hist.random.org.sd <- sd(hist.random.org$density))
```

```
## [1] 0.3091206
```

```
plot(hist.random.org, freq = FALSE, ylim = c(0, 2),
     main = paste("Density Variation of", title.random.org))
abline(h = hist.random.org.mean)
abline(h = hist.random.org.mean + 1.96 * hist.random.org.sd, col = "red", lty = 2)
abline(h = hist.random.org.mean - 1.96 * hist.random.org.sd, col = "red", lty = 2)
```



What does the graph tell you about the observed distribution?

Once again, Examining the empirical density of our sample as diagrammed on our histogram, we can see that the central tendency of our 10 bins has a mean of 1 and further our bin densities are within one standard deviation of 1. Though this smaller sample of 100 shows more variation than our earlier sample of 1000, the central tendency and variation of these bins remains approximately consistent with a continuous uniform distribution on [0,1] and our goal.

```r
# Estimate the moments of the sample.
(sample.random.org.mean <- mean(sample.random.org))
```

```
## [1] 0.4744629
```

```r
(sample.random.org.var <- var(sample.random.org))
```

```
## [1] 0.07155496
```

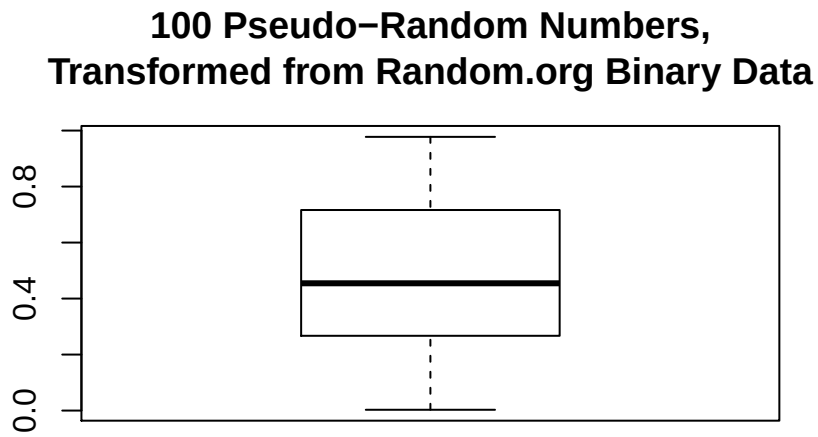
What do you conclude about the estimated distribution from the moments?

For a uniform distribution we expect that the center of our data will be midway through the range of the distribution. Thus the sample mean of 0.4744629 is quite close to our expected theoretical mean of 0.5 (midway between 0 and 1, or $\frac{(1-0)}{2}$. We expect that the variance of our data will be $\frac{(b-a)^2}{12}$, or 1/12, and our sample variance of 0.071555 is relatively close to this.

```r
# Check the summary of the simulated sample.
summary(sample.random.org)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00293 0.27470 0.45460 0.47450 0.71020 0.97750
```

```r
boxplot(sample.random.org, main = title.random.org)
```



Using the same methods to examine our random.org sample as we did our runif sample, we noted that the latter, being a sample of 100 rather than 1000, resulted in greater variation and demonstrated less strongly the expected parameters of a uniform distribution. However, given than 100 is a relatively small sample size, we suspect that given the law of large numbers that the variations we saw would even themselves out with a larger sample.

2.2. Test independence of the sequence of zeros and ones

2.2.1. Turning point test

Check if a sequence of numbers is i.i.d. (independent identically distributed) based on the number of turning points in the sequence.

The number of turning points is the number of maxima and minima in the series. The statistic of the test has a standard normal distribution and is performed by turning.point.test() in package randtests.

```
library(randtests)
```

```
turning.point.test(sample.random.org)
```

```
##
##  Turning Point Test
##
## data:  sample.random.org
## statistic = -1.0372, n = 100, p-value = 0.2997
## alternative hypothesis: non randomness
```

The null hypothesis tested by turning point test is randomness (i.i.d.). The alternative is serial correlation in the sequence. Thus, if the test returns a very small p-value the randomness needs to be rejected.

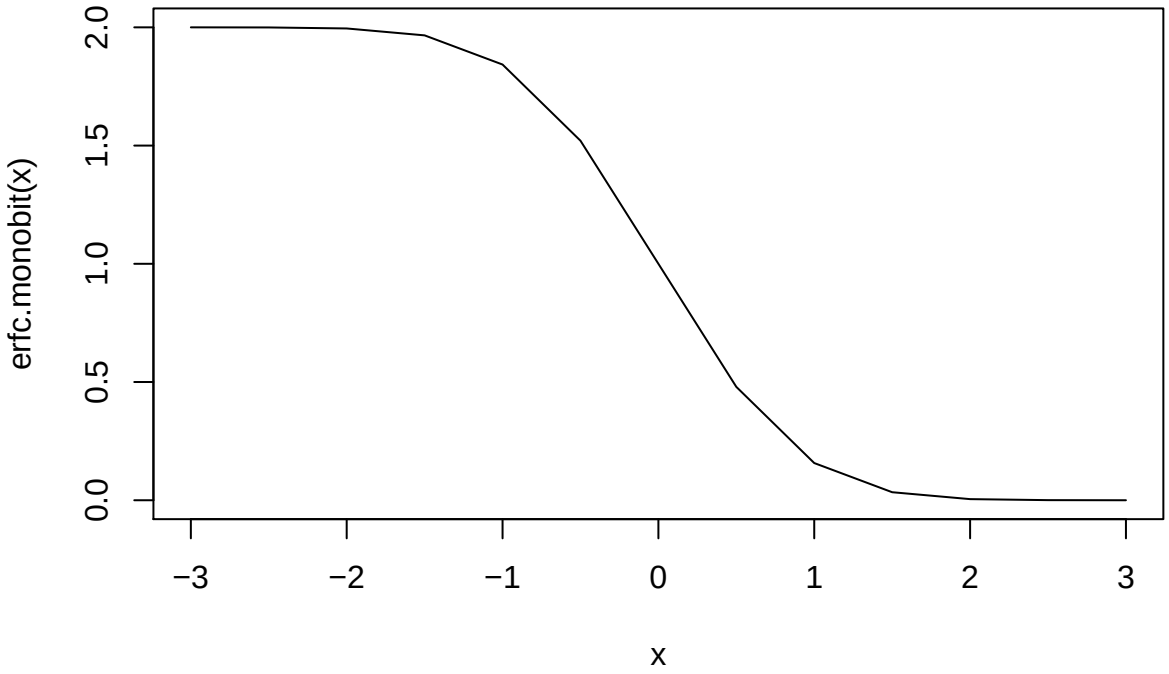
2.3. Test frequency by Monobit test

To perform a Monobit test you need to transform your {0,1} sample into {-1,1}.

```
data.random.org.plusminus1 <- (data.random.org - .5) * 2
```

The monobit test can be calculated in R with the help of pnorm. The test checks that the the p-value or complimentary error function of a statistic S is less than or equal to 0.01. If so, the sequence fails the test.

```
erf.monobit <- function(x) 2 * pnorm(x * sqrt(2)) - 1
erfc.monobit <- function(x) 2 * pnorm(x * sqrt(2), lower = FALSE)

# Chart the complimentary error function
plot(seq(from = -3, to = 3, by = 0.5), erfc.monobit(seq(from = -3, to = 3, by = 0.5)),
     type = "l", xlab = "x", ylab = "erfc.monobit(x)")
```



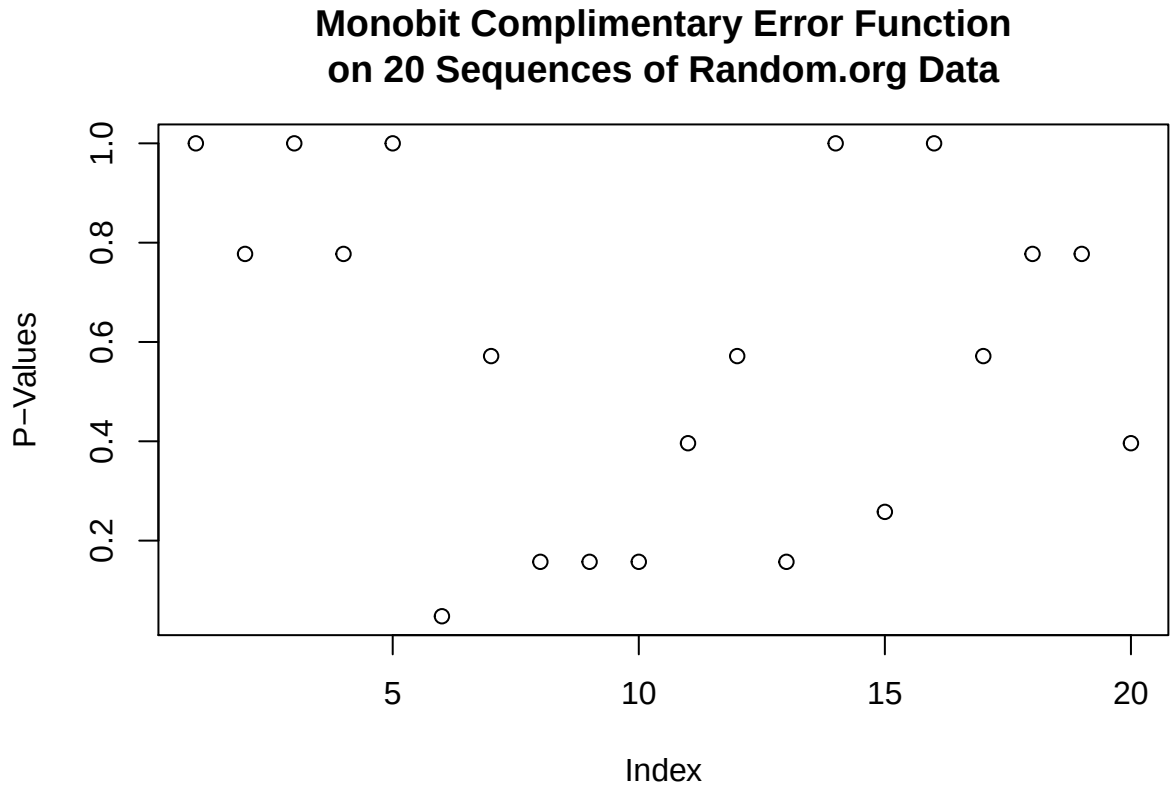

```
# Calculate our S statistic
s.monobit.random.org <- abs(sum(data.random.org.plusminus1) / sqrt( 2* num.flips))

# Find the p-value or erfc(S)
erfc.monobit(s.monobit.random.org)
```

```
## [1] 0.8495155
```

The test shows that our sequence passes.

Now check each of the sub-sequences created earlier:

```
# Create matrix of sub-sequences
matrix.plusminus1 <- matrix(data.random.org.plusminus1, ncol=50)

# Find p-value of vector of S statistics for every sub-sequence
erfc.random.org.subseq <- erfc.monobit(abs(apply(matrix.plusminus1, 1, sum)) / sqrt(2 * 50))

# Plot
plot(erfc.random.org.subseq, ylab = "P-Values",
     main = "Monobit Complimentary Error Function\non 20 Sequences of Random.org Data")
```



How many runs out of 20 fail the test?

```
sum(erfc.random.org.subseq <= 0.01)
```

```
## [1] 0
```

We can confirm that the random sequences provided by Random.org pass the monobit test.

Part 3. Invent a random number generator

Our goal in inventing a random number generator is to seek out a true source of a uniform random sequence on [0,1].

One random event that is available to all of those who live in a metropolitan area is the movement of cars on the roads that surround us. To generate our random sequence we combined the allure of statistics with the draw to spend time with our family, and spent roughly two hours recording the movement of cars along a relatively busy throughfare in the North Center neighborhood of Chicago.

For our traffic-based Bernoulli sequence, we recorded a "success" or "failure" for each car (vehicle) that passed in front of our testing location until 1000 recordings were made. Each Eastward-bound car represents a "success" and each Westward-bound car represents a "failure".

Anecdotally, we were also able to empirically estimate the length of recordings that a young elementary school student is able to retain focus on such an exercise at approximately 200 recordings, with additional predictors in the form of ambient noise from sibling YouTube activities and spousal snack preparation.

Data Import and Transformation

Let's bring this data, recorded in Excel and saved as a .csv file, into R.

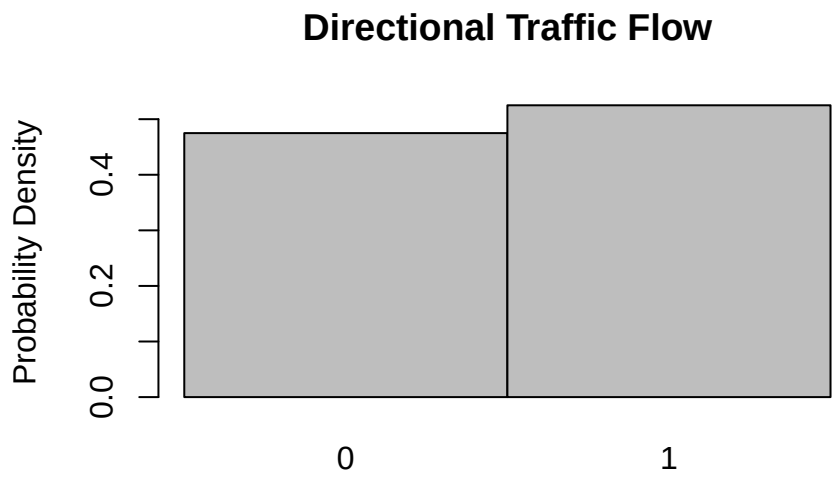
```
data.traffic <- read.csv("cars.csv", header = TRUE)
str(data.traffic)
```

```
## 'data.frame':    1000 obs. of  2 variables:
##  $ Origin.of.Car : Factor w/ 2 levels "L","R": 1 1 2 2 1 1 2 1 2 2 ...
##  $ Eastward.Bound: int  1 1 0 0 1 1 0 1 0 0 ...
```

```
traffic.n <- nrow(data.traffic)
```

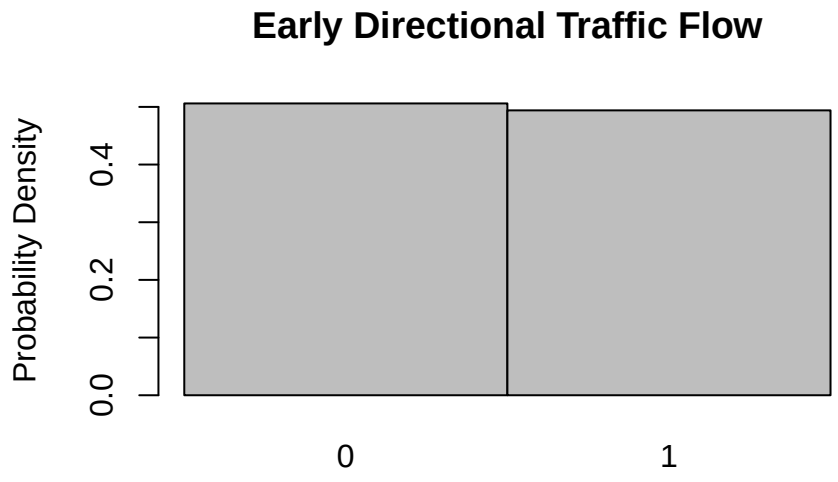
And let's take a look at our raw binary distribution.

```
data.traffic.density <- table(data.traffic$Eastward.Bound) / traffic.n
barplot(data.traffic.density, space = 0, ylab = "Probability Density",
        main="Directional Traffic Flow")
```



As we recorded our data, we casually noted that the proportion of traffic flowing in either direction seemed to grow unbalanced as the time grew later on the Sunday afternoon in which we conducted our experiment. Let's check if this observation was accurate to the data.

```
data.traffic.density.early <- table(data.traffic$Eastward.Bound[1:500]) / 500
barplot(data.traffic.density.early, space = 0, ylab = "Probability Density",
        main="Early Directional Traffic Flow")
```




```
data.traffic.density.late <- table(data.traffic$Eastward.Bound[501:1000]) / 500
barplot(data.traffic.density.late, space = 0, ylab = "Probability Density",
        main="Late Directional Traffic Flow")
```

This difference may impact our findings, but it's too costly (time is money!) to record a new data set, and we are hesitant to use only 500 observations, i.e. the first half of our data. Let's proceed with the full data set for now.

We can now convert the binary sequence of 1000 into 100 numbers on the interval [0,1], by interpreting each sequence of 10 numbers as a decimal.

We'll create two transformations. After building a matrix of the binary values in the order that they were recorded, our first transformation will interpret sequences of 10 numbers in the order they were recorded along rows, and our second will interpret sequences of 10 numbers along the columns of the matrix.

The second transformation essentially "scrambles" our sequence. Given that the sequence as recorded included periodic repetitions of length 4-10 related to stop lights on either side of our location, we are curious as to whether the scrambled column-transformed sequence will produce a more uniform distribution.

```
# Turn the sequence of 0's and 1's into a 100x10 matrix
matrix.binary <- matrix(data.traffic$Eastward.Bound, ncol = 10)
head(matrix.binary)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    0    1    1    0    0    1    1    0     1
## [2,]    1    0    0    0    0    0    1    1    0     0
## [3,]    0    1    0    1    1    1    1    0    1     0
## [4,]    0    0    1    1    1    1    1    1    1     1
## [5,]    1    1    1    1    1    1    1    0    1     0
## [6,]    1    1    0    1    1    1    0    1    1     1
```

```
str(matrix.binary)
```

```
##  int [1:100, 1:10] 1 1 0 0 1 1 0 1 0 0 ...
```

```
# Transform each row into decimal format
vector.decimal <- apply(matrix.binary, 1, BinaryToDec)
str(vector.decimal)
```

```
##  num [1:100] 717 524 378 255 1018 ...
```

```
# Divide the numbers by 2^10 to make real numbers in [0,1]
sample.traffic.1 <- vector.decimal / 2^10
str(sample.traffic.1)
```

```
##  num [1:100] 0.7 0.512 0.369 0.249 0.994 ...
```

```
sample.traffic.1
```

```
##   [1] 0.70019531 0.51171875 0.36914062 0.24902344 0.99414062 0.86621094
##   [7] 0.35839844 0.97070312 0.23730469 0.32617188 0.34082031 0.33300781
##  [13] 0.57812500 0.22167969 0.05175781 0.52734375 0.21875000 0.63769531
##  [19] 0.25097656 0.23437500 0.19726562 0.45898438 0.94042969 0.13574219
##  [25] 0.69824219 0.70800781 0.49707031 0.36914062 0.61035156 0.56152344
##  [31] 0.80273438 0.30371094 0.78320312 0.36230469 0.52441406 0.25976562
##  [37] 0.01660156 0.26757812 0.63769531 0.51953125 0.38281250 0.15527344
##  [43] 0.57031250 0.90429688 0.88281250 0.63964844 0.99707031 0.97460938
##  [49] 0.69335938 0.70800781 0.14648438 0.33886719 0.04589844 0.29882812
##  [55] 0.04199219 0.90917969 0.22949219 0.93066406 0.67871094 0.80957031
##  [61] 0.81152344 0.24609375 0.24511719 0.52050781 0.16894531 0.60742188
##  [67] 0.09179688 0.68750000 0.22851562 0.68945312 0.84667969 0.71386719
##  [73] 0.12207031 0.83105469 0.58691406 0.14355469 0.79785156 0.48535156
##  [79] 0.27246094 0.80175781 0.92773438 0.40332031 0.78027344 0.08593750
##  [85] 0.23242188 0.74511719 0.58496094 0.22753906 0.06835938 0.85449219
##  [91] 0.22363281 0.99218750 0.85156250 0.36132812 0.34375000 0.02343750
##  [97] 0.26367188 0.02832031 0.01660156 0.27832031
```

```
quantile(sample.traffic.1)
```

```
##          0%         25%         50%         75%        100%
## 0.01660156 0.23657227 0.47216797 0.70947266 0.99707031
```

```r
# Create a second transformation by column instead of row
matrix.binary <- matrix(data.traffic$Eastward.Bound, ncol = 100)
vector.decimal <- apply(matrix.binary, 2, BinaryToDec)
sample.traffic.2 <- vector.decimal / 2^10
str(sample.traffic.2)
```

```
##  num [1:100] 0.801 0.145 0.175 0.659 0.249 ...
```

```r
sample.traffic.2
```

```
##   [1] 0.800781250 0.144531250 0.174804688 0.659179688 0.249023438
##   [6] 0.022460938 0.583007812 0.852539062 0.649414062 0.375000000
##  [11] 0.184570312 0.751953125 0.386718750 0.957031250 0.605468750
##  [16] 0.333007812 0.500000000 0.577148438 0.875976562 0.477539062
##  [21] 0.599609375 0.075195312 0.992187500 0.001953125 0.874023438
##  [26] 0.529296875 0.413085938 0.269531250 0.800781250 0.750000000
##  [31] 0.749023438 0.946289062 0.935546875 0.062500000 0.139648438
##  [36] 0.257812500 0.405273438 0.972656250 0.124023438 0.968750000
##  [41] 0.248046875 0.102539062 0.014648438 0.937500000 0.011718750
##  [46] 0.249023438 0.923828125 0.637695312 0.551757812 0.968750000
##  [51] 0.236328125 0.922851562 0.280273438 0.856445312 0.321289062
##  [56] 0.819335938 0.945312500 0.499023438 0.960937500 0.334960938
##  [61] 0.977539062 0.519531250 0.606445312 0.048828125 0.992187500
##  [66] 0.418945312 0.931640625 0.406250000 0.489257812 0.403320312
##  [71] 0.836914062 0.800781250 0.274414062 0.752929688 0.342773438
##  [76] 0.627929688 0.867187500 0.127929688 0.916015625 0.512695312
##  [81] 0.249023438 0.062500000 0.973632812 0.957031250 0.333984375
##  [86] 0.994140625 0.651367188 0.830078125 0.682617188 0.070312500
##  [91] 0.587890625 0.849609375 0.245117188 0.353515625 0.274414062
##  [96] 0.436523438 0.718750000 0.999023438 0.403320312 0.506835938
```

```r
quantile(sample.traffic.2)
```

```
##           0%          25%          50%          75%         100%
## 0.001953125 0.274414062 0.540527344 0.850341797 0.999023438
```

Visualization

Let's visualize the data as an initial analysis.

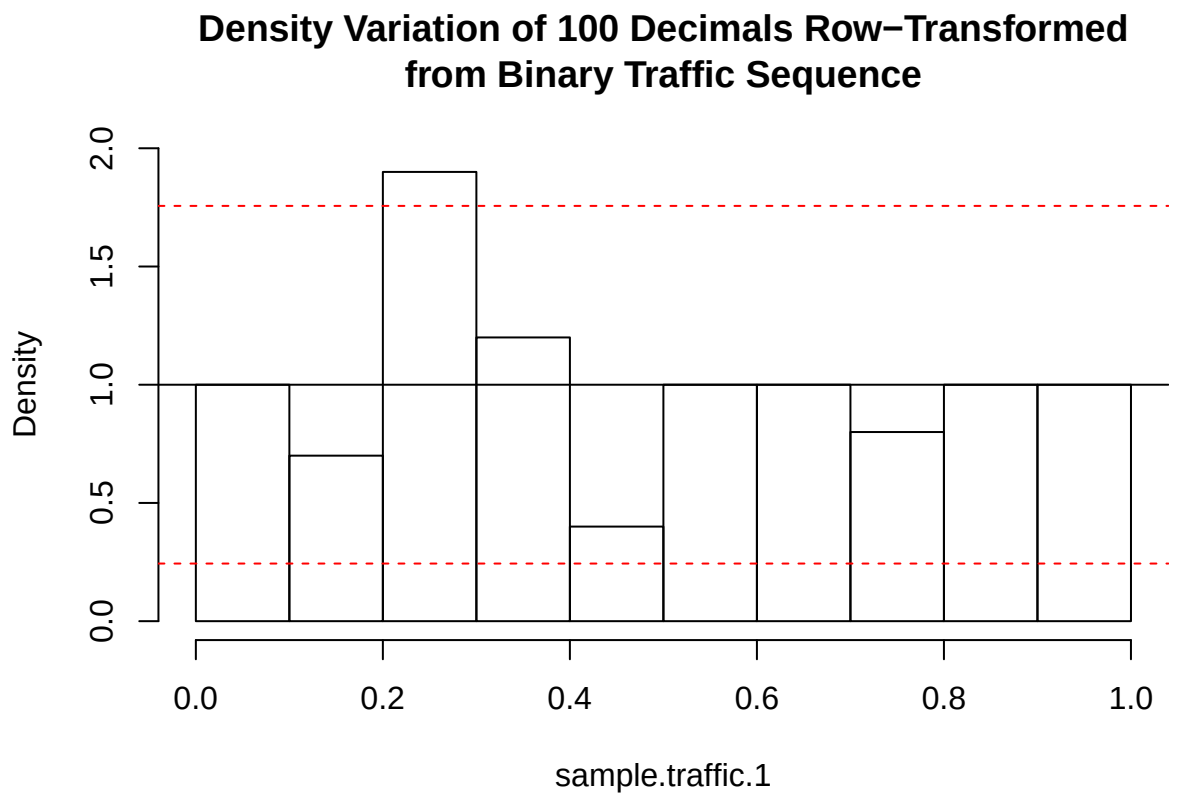
```r
# Histogram of our transformed data
title.traffic.1 <- "100 Decimals Row-Transformed\nfrom Binary Traffic Sequence"
#hist.traffic.1 <- hist(sample.traffic.1, main = title.traffic.1, plot = FALSE)
hist.traffic.1 <- hist(sample.traffic.1, plot = FALSE)
#hist.traffic.1

title.traffic.2 <- "100 Decimals Column-Transformed\nfrom Binary Traffic Sequence"
#hist.traffic.2 <- hist(sample.traffic.2, main = title.traffic.2, plot = FALSE)
hist.traffic.2 <- hist(sample.traffic.2, plot = FALSE)
#hist.traffic.2

# Estimate mean and standard deviation of our sample's density.
(hist.traffic.1.mean <- mean(hist.traffic.1$density))
```

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

```
(hist.traffic.1.sd <- sd(hist.traffic.1$density))
```

```
## [1] 0.3858612
```

```
plot(hist.traffic.1, freq = FALSE, ylim = c(0, 2),
     main = paste("Density Variation of", title.traffic.1))
abline(h = hist.traffic.1.mean)
abline(h = hist.traffic.1.mean + 1.96 * hist.traffic.1.sd, col = "red", lty = 2)
abline(h = hist.traffic.1.mean - 1.96 * hist.traffic.1.sd, col = "red", lty = 2)
```



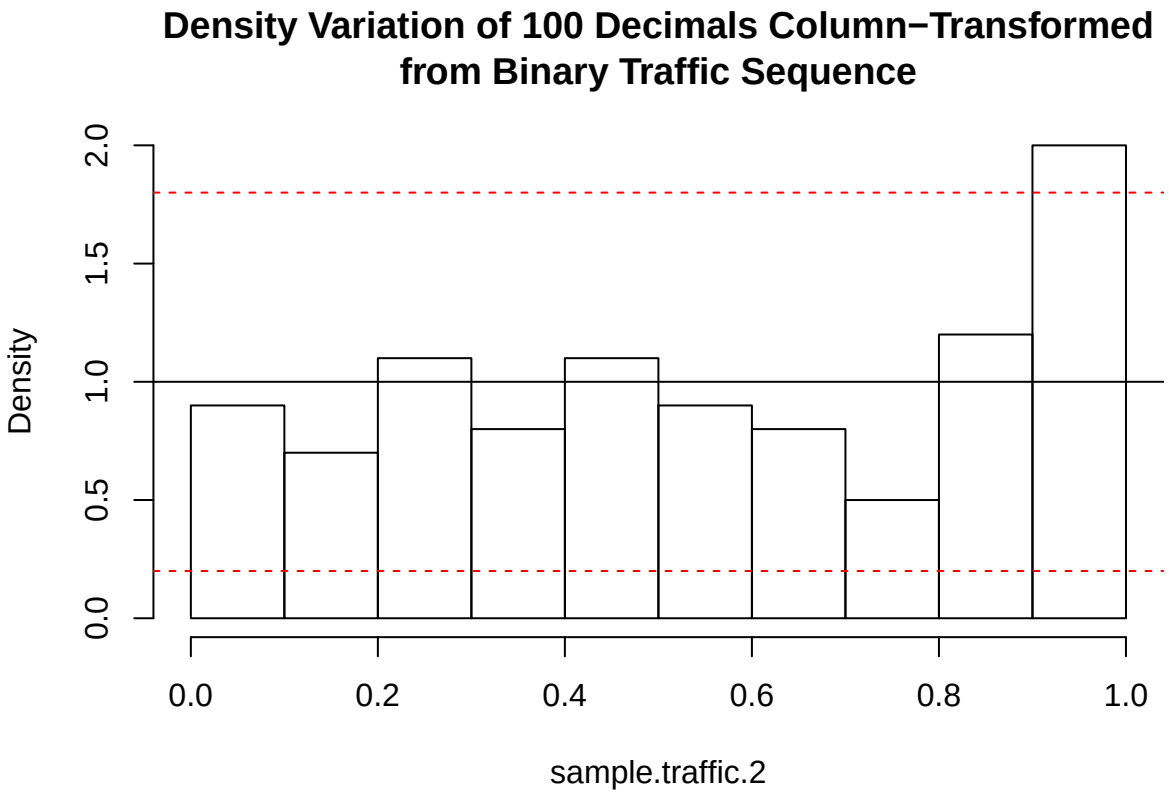

```
(hist.traffic.2.mean <- mean(hist.traffic.2$density))
```

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

```
(hist.traffic.2.sd <- sd(hist.traffic.2$density))
```

```
## [1] 0.4082483
```

```
plot(hist.traffic.2, freq = FALSE, ylim = c(0, 2),
     main = paste("Density Variation of", title.traffic.2))
abline(h = hist.traffic.2.mean)
abline(h = hist.traffic.2.mean + 1.96 * hist.traffic.2.sd, col = "red", lty = 2)
abline(h = hist.traffic.2.mean - 1.96 * hist.traffic.2.sd, col = "red", lty = 2)
```



And finally we'll look at our moments.

Row-Transformed

```
# Estimate the moments of the sample.
(sample.traffic.1.mean <- mean(sample.traffic.1))
```

```
## [1] 0.4816113
```

```
(sample.traffic.1.var <- var(sample.traffic.1))
```

```
## [1] 0.08456558
```

For a uniform distribution we expect that the center of our data will be midway through the range of the distribution. Thus the sample mean of 0.4816113 is quite close to our expected theoretical mean of 0.5 (midway between 0 and 1, or $\frac{(1-0)}{2}$. We expect that the variance of our data will be $\frac{(b-a)^2}{12}$, or 1/12, and our sample variance of 0.0845656 is quite close to this.

Column-Transformed

```
# Estimate the moments of the sample.
(sample.traffic.2.mean <- mean(sample.traffic.2))
```

```
## [1] 0.5488281
```

```
(sample.traffic.2.var <- var(sample.traffic.2))
```

```
## [1] 0.09653305
```

For a uniform distribution we expect that the center of our data will be midway through the range of the distribution. Thus the sample mean of 0.5488281 is relatively close to our expected theoretical mean of 0.5 (midway between 0 and 1, or $\frac{(1-0)}{2}$. We expect that the variance of our data will be $\frac{(b-a)^2}{12}$, or 1/12, and our sample variance of 0.0965331 is relatively close to this.

Overall Perception

It appears that our data is very nearly unified, yet both transformations seem to contain repeated values or some other factor that pulls the data outside expected uniform behavior and pulls the moments slightly askew of our expectations for a uniform distribution.

Tests for Randomness

Now that we have observed the data somewhat informally, let's test for randomness using the Turning Point and Monobit tests.

Turning Point

```
turning.point.test(sample.traffic.1)
```

```
##
##  Turning Point Test
##
## data:  sample.traffic.1
## statistic = 0.39892, n = 100, p-value = 0.69
## alternative hypothesis: non randomness
```

```
turning.point.test(sample.traffic.2)
```

```
##
##  Turning Point Test
##
## data:  sample.traffic.2
## statistic = 0.87762, n = 100, p-value = 0.3802
## alternative hypothesis: non randomness
```

The null hypothesis tested by turning point test is randomness (i.i.d.). The alternative is serial correlation in the sequence. Thus, if the test returns a very small p-value the randomness needs to be rejected.

In this case we fail to reject the null hypothesis. It appears that our data is random.

Monobit

To perform Monobit test we need to transform our {0,1} sample into {-1,1}.

```
data.traffic.plusminus1 <- (data.traffic$Eastward.Bound - .5) * 2
```

The monobit test can be calculated in R with the help of pnorm. The test checks that the the p-value or complimentary error function of a statistic S is less than or equal to 0.01. If so, the sequence fails the test.

```
erf.monobit <- function(x) 2 * pnorm(x * sqrt(2)) - 1
erfc.monobit <- function(x) 2 * pnorm(x * sqrt(2), lower = FALSE)

# Calculate our S statistic
```

```r
s.monobit.traffic <- abs(sum(data.traffic.plusminus1) / sqrt(2*traffic.n))

# Find the p-value or erfc(S)
erfc.monobit(s.monobit.traffic)
```

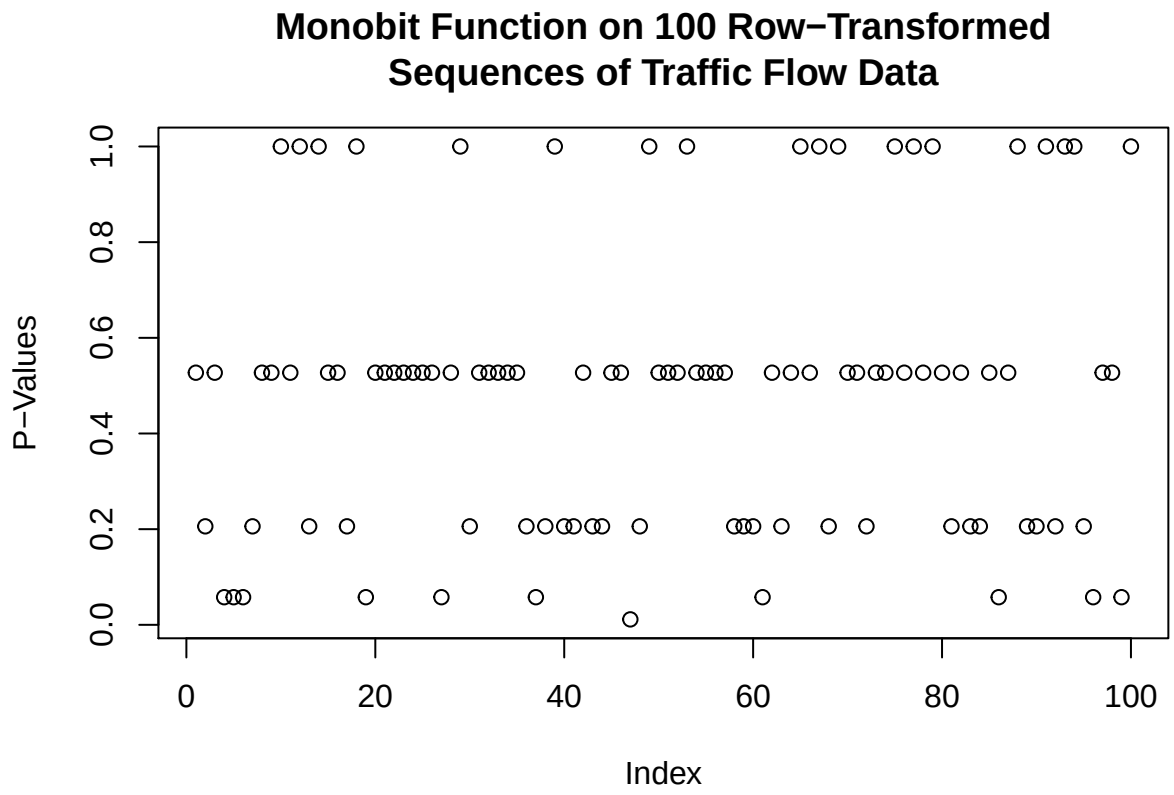
```
## [1] 0.1138463
```

Our P-value is greater than 0.01. The test shows that our sequence passes.

Now check each of the sub-sequences created earlier:

*** Row Transformed ***

```r
# Create matrix of sub-sequences
matrix.plusminus1 <- matrix(data.traffic.plusminus1, ncol=10)

# Find p-value of vector of S statistics for every sub-sequence
erfc.traffic.1.subseq <- erfc.monobit(abs(apply(matrix.plusminus1, 1, sum)) / sqrt(2 * 10))

# Plot
plot(erfc.traffic.1.subseq, ylab = "P-Values",
     main = "Monobit Function on 100 Row-Transformed\nSequences of Traffic Flow Data")
```



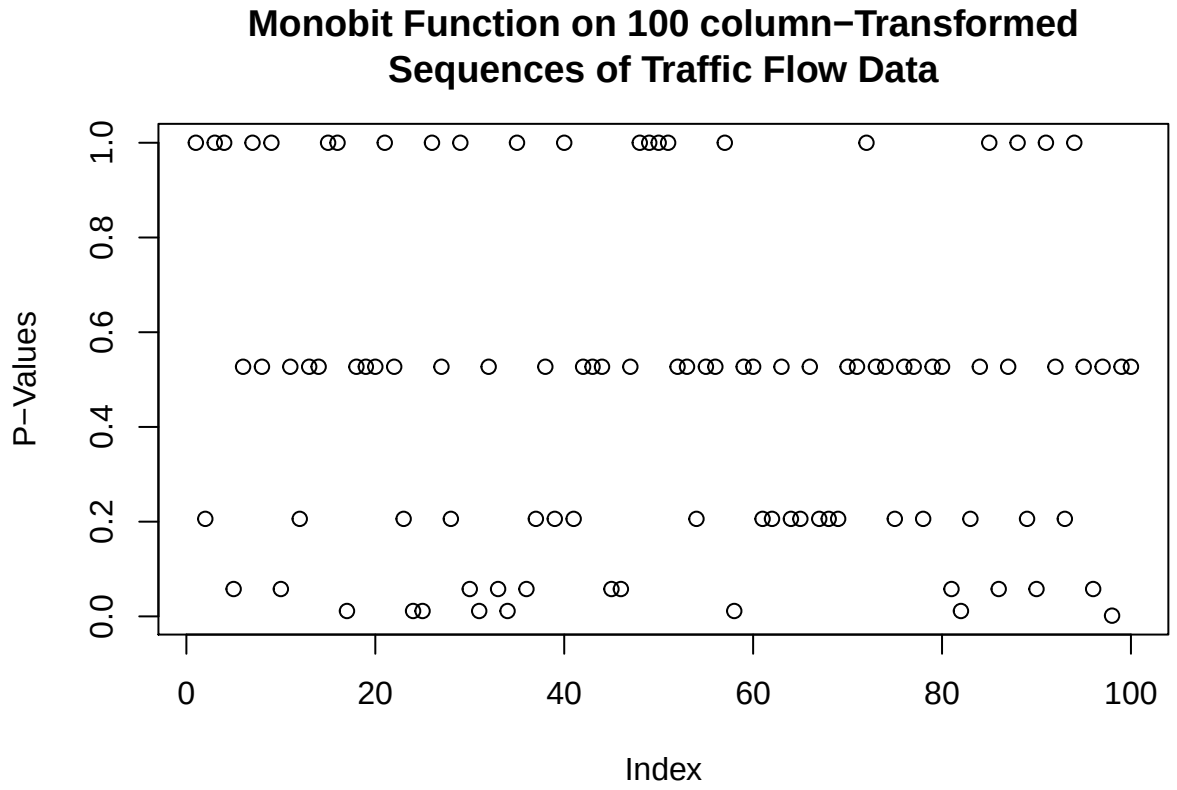
How many of the 100 row-transformed sequences fail the test?

```r
sum(erfc.traffic.1.subseq <= 0.01)
```

```
## [1] 0
```

*** Column-Transformed ***

```
# Create matrix of sub-sequences
matrix.plusminus1 <- matrix(data.traffic.plusminus1, ncol=100)

# Find p-value of vector of S statistics for every sub-sequence
erfc.traffic.2.subseq <- erfc.monobit(abs(apply(matrix.plusminus1, 2, sum)) / sqrt(2 * 10))

# Plot
plot(erfc.traffic.2.subseq, ylab = "P-Values",
     main = "Monobit Function on 100 column-Transformed\nSequences of Traffic Flow Data")
```



How many of the 100 column-transformed sequences fail the test?

```
sum(erfc.traffic.2.subseq <= 0.01)
```

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

Interestingly, the sequences which we intentionally "scrambled" thinking that they would be more random were actually less random. Our dataset has passed the turning point and monobit tests of randomness and is very nearly but not quite unified. We determined that it was not quite unified even before we analyzed the data mathematically, as demonstrated by our initial charts that showed late afternoon traffic was less unified.

Future experiements could attempt to determine factors which pull the data away from unified, such as time of day. If these factors could be controlled, this process could be used to provide a random, unified sample.

Part 4. Monte Carlo Method

4.1. Scratch off quote of the day: fuction download

```
filename <- "ScratchOffMonteCarlo.rda"

#path <- "{Path to the folder with ScratchOffMonteCarlo.rda}"
#load(file = paste(path, filename,sep= '/ '))

# Place ScratchOffMonteCarlo file in project directory or use code above
load(filename)
```

4.2. Simulate pseudo-random poins [x,y][x,y] on [0,100]×[0,100]

The ScratchOffMonteCarlo function simulates 'scratching off the paint' covering an image at each x, y coordinate. Let's try creating 1000 pseudo-random x, y coordinates and see if the image is readable.

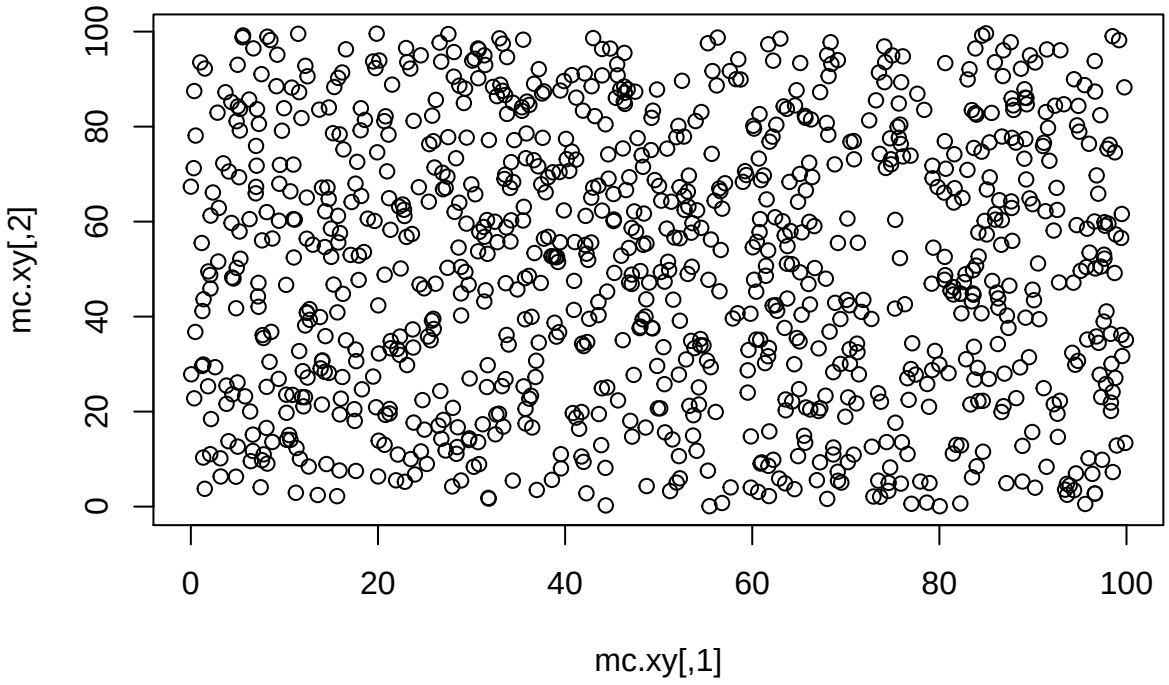
```
# Set seed
seed1 <- 938364
set.seed(seed1)

# Set sample size
mc.n <- 1000

# Simulate n * 2 pseudo-random numbers uniformly distributed on [0, 100]
mc.xy <- runif(2 * mc.n, min = 0, max = 100)

# Transform into n x 2 matrix
mc.xy <- matrix(mc.xy, ncol = 2)
str(mc.xy)
```

```
##  num [1:1000, 1:2] 11.22 15.92 2.07 6.68 39.49 ...
```

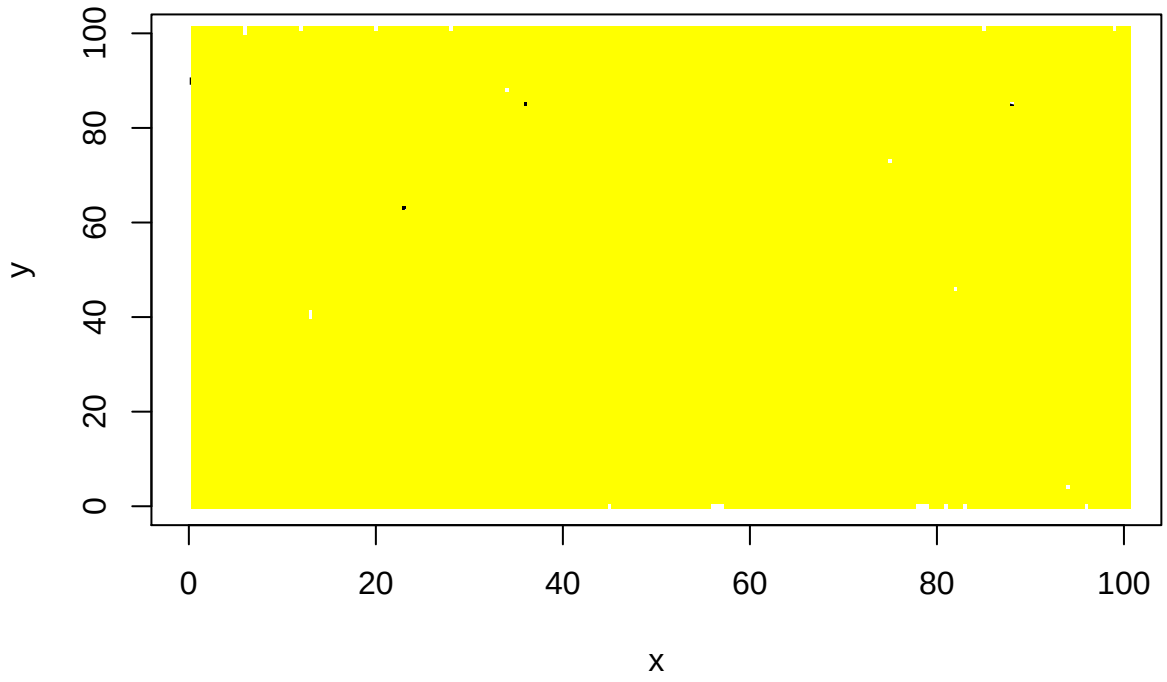
```
head(mc.xy)
```

```
##           [,1]      [,2]
## [1,] 11.224758  2.912387
## [2,] 15.922395 19.404532
## [3,]  2.069867 48.776640
## [4,]  6.684756 96.472515
## [5,] 39.490429 87.598829
## [6,] 27.526241 99.493985
```

```
plot(mc.xy)
```



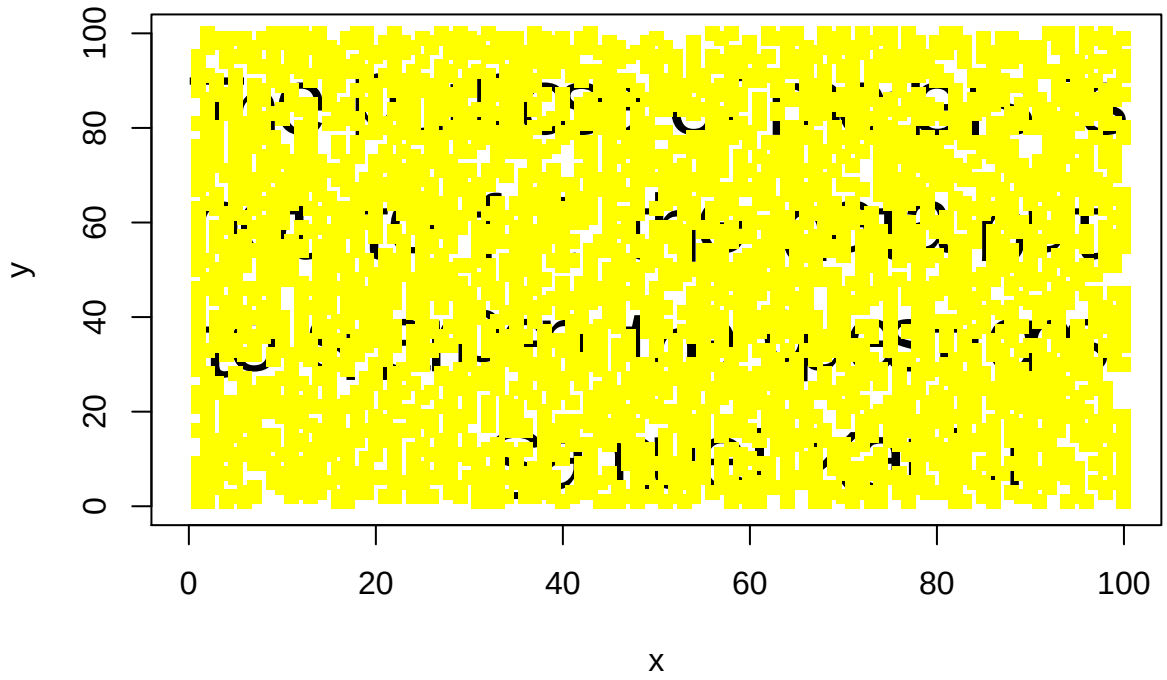

```
# Simulate 'scratching off the paint' covering an image at each x, y coordinate
ScratchOffMonteCarlo(mc.xy)
```




```
## [1] "Size = 1000"    "Open (%)= 9.52"
```

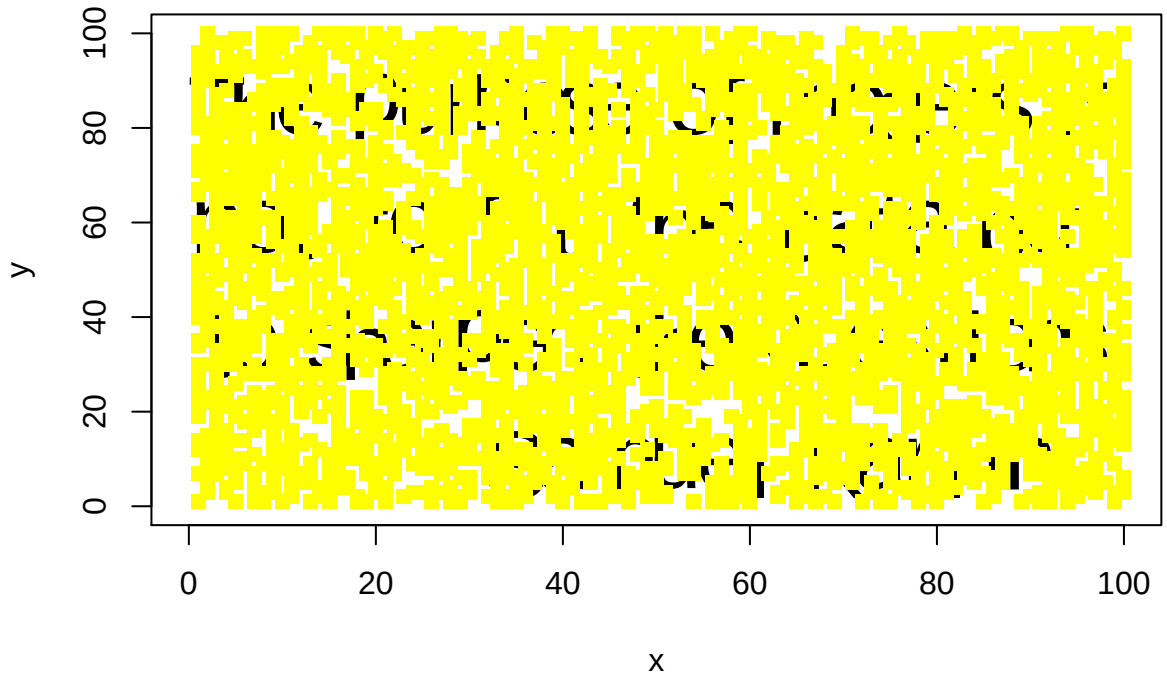
Let's adjust the sample size and seed and try to make the quote in our image readable with minimum sample size.

```
# Create function to generate coordinate matrix
xy <- function(n, seed) {
  # Argument n: Number of x, y coordinate samples to generate
  # Argument seed: Seed to ensure reproducibility

  # Simulate n * 2 pseudo-random numbers uniformly distributed on [0, 100]
  set.seed(seed)
  xy <- runif(2 * n, min = 0, max = 100)

  # Transform into n x 2 matrix
  xy <- matrix(xy, ncol = 2)
}

# Seed 1 10k:
mc.n <- 10000; mc10k.xy <- xy(mc.n, seed1); ScratchOffMonteCarlo(mc10k.xy)
```



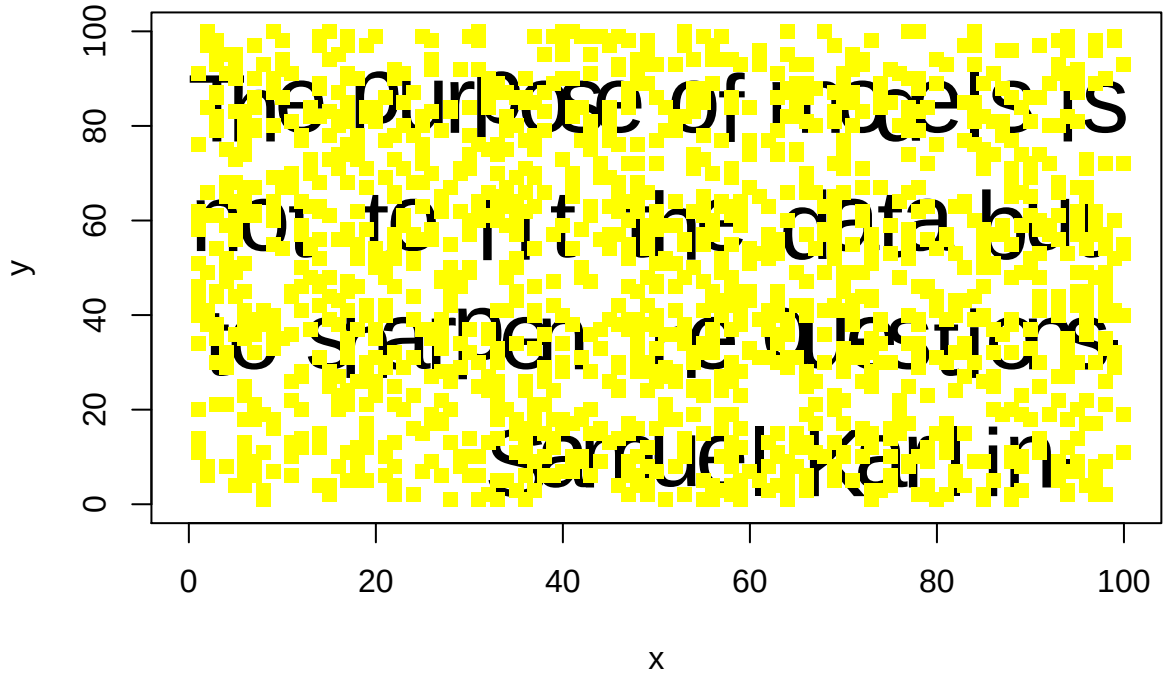

```
## [1] "Size = 10000"    "Open (%)= 63.45"
```

```
# Seed 2 10k:
seed2 <- 112233; mc10k.xy <- xy(mc.n, seed2); ScratchOffMonteCarlo(mc10k.xy)
```



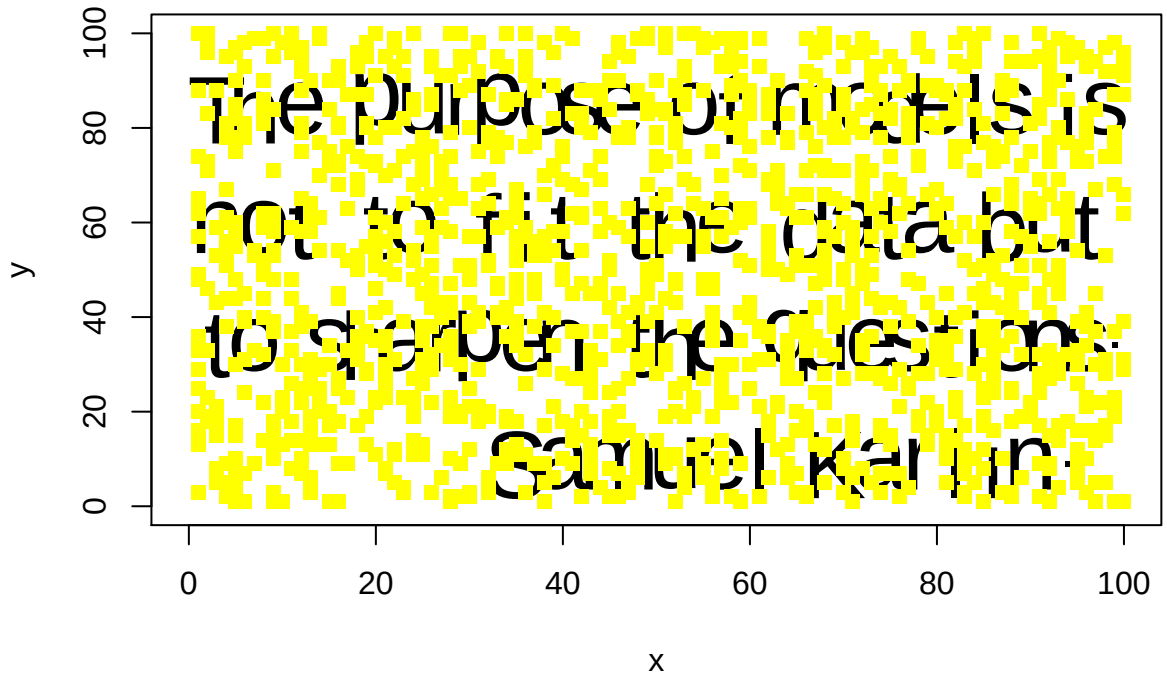

```
## [1] "Size = 10000"    "Open (%)= 63.36"
```

```
# Seed 1 20k:
mc.n <- 20000; mc20k.xy <- xy(mc.n, seed1); ScratchOffMonteCarlo(mc20k.xy)
```



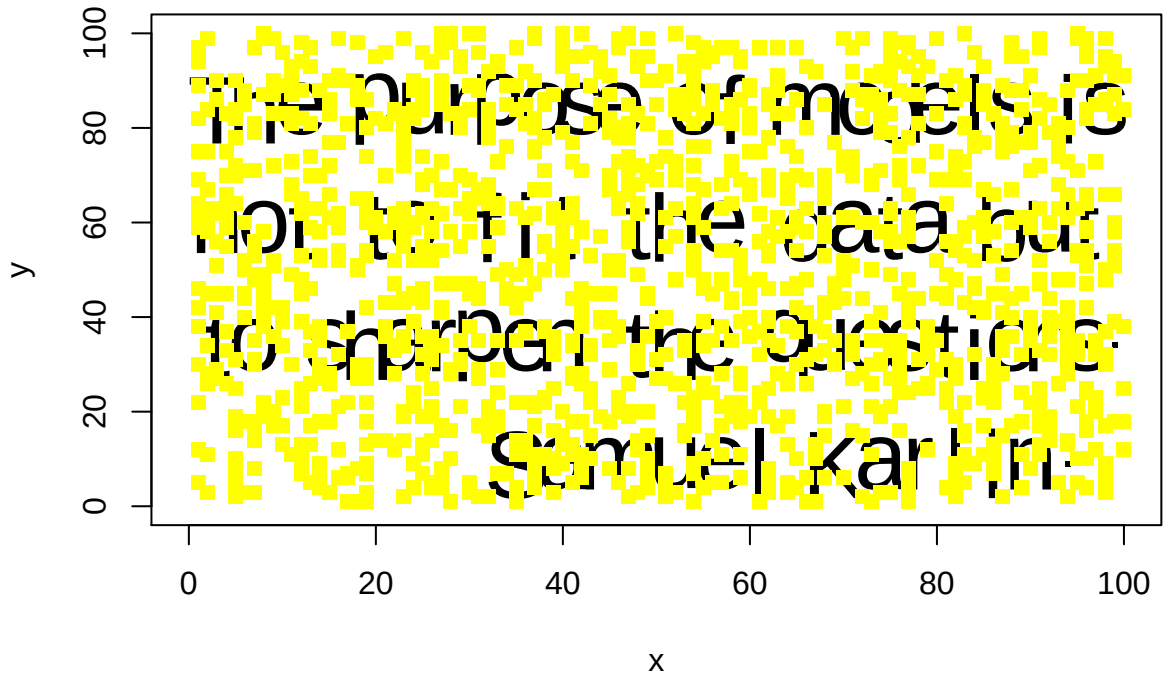

```
## [1] "Size = 20000"    "Open (%)= 86.15"
```

```
# Seed 2 20k:
mc20k.xy <- xy(mc.n, seed2); ScratchOffMonteCarlo(mc20k.xy)
```



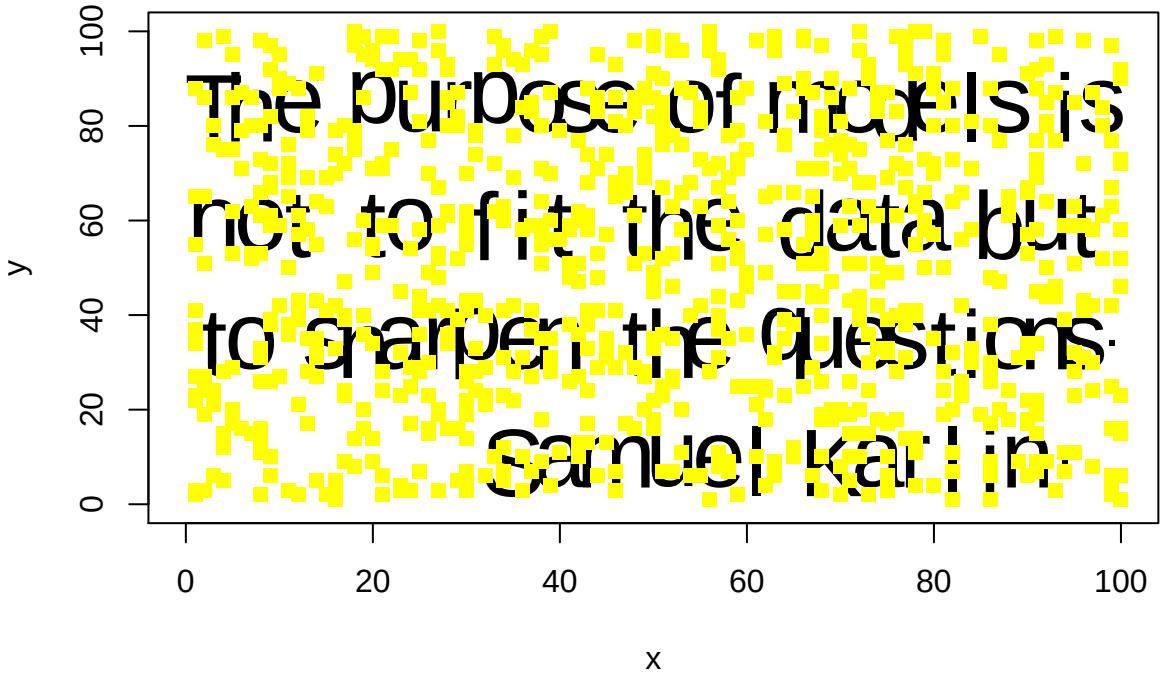

```
## [1] "Size = 20000"    "Open (%)= 86.77"
```

```
# Seed 3 20k:
seed3 <- 999999; mc20k.xy <- xy(mc.n, seed3); ScratchOffMonteCarlo(mc20k.xy)
```




```
## [1] "Size = 20000"    "Open (%)= 86.84"
```

```
# Seed 2 25k:
mc.n <- 25000; mc25k.xy <- xy(mc.n, seed2); ScratchOffMonteCarlo(mc25k.xy)
```




```
## [1] "Size = 25000"    "Open (%)= 91.91"
```

What percent you needed to scratch off to make the quote readable?

It appears we need to scratch off approximately 90% of the paint to make the quote and its author readable. (We can read the English words at less due to the cognitive ability to fill in letters in words that match existing patterns, but unknown names require more letters to be readable.)

4.3. Simulate quasi-random poins [x,y][x,y] on [0,100]×[0,100]

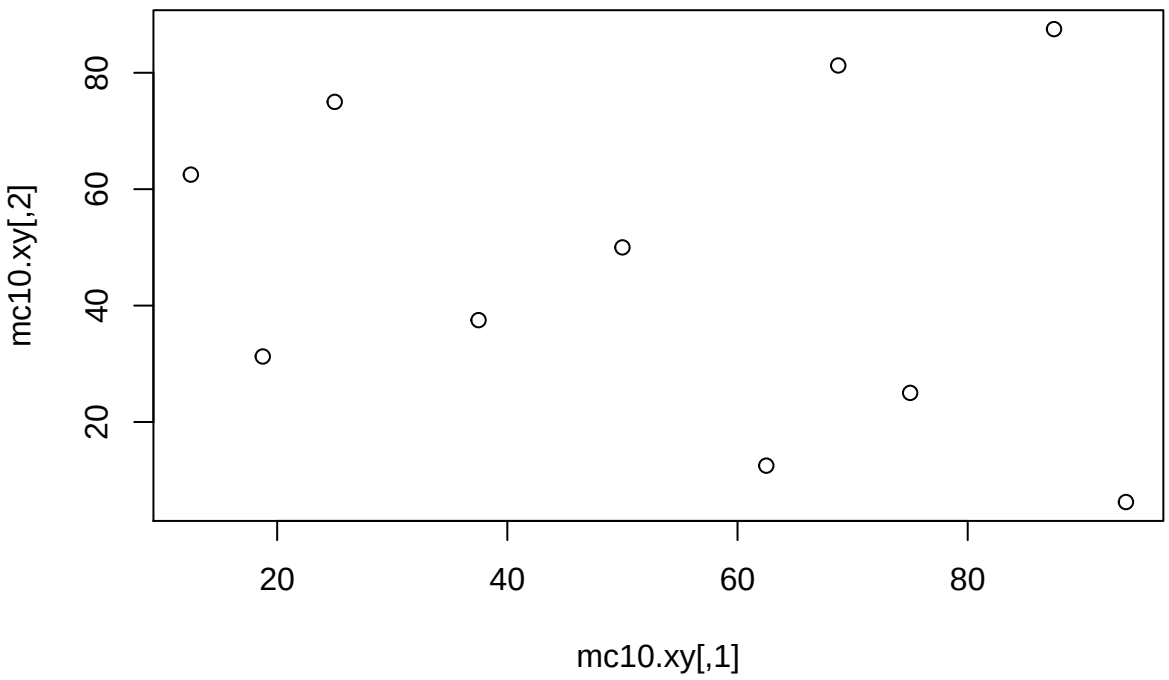
We replace function runif() with sobol() from the library randtoolbox.

```
library(randtoolbox)
```

```
seed4 <- 10
set.seed(seed4)
mc.n <- 10
mc10.xy <- sobol(mc.n, dim = 2, init = T) * 100
head(mc10.xy)
```

```
##      [,1] [,2]
## [1,] 50.0 50.0
## [2,] 75.0 25.0
## [3,] 25.0 75.0
## [4,] 37.5 37.5
## [5,] 87.5 87.5
## [6,] 62.5 12.5
```

```
plot(mc10.xy)
```

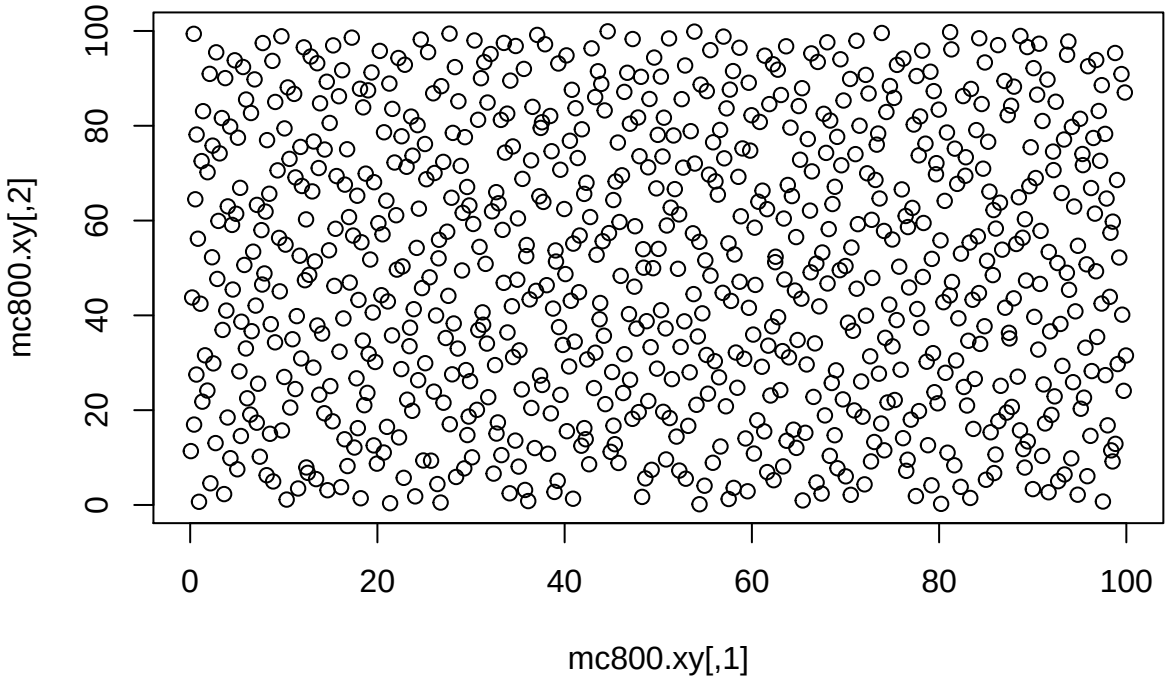


Again, by changing nSample and my.seed we try to make the quote of the day readable with minimum sample size.

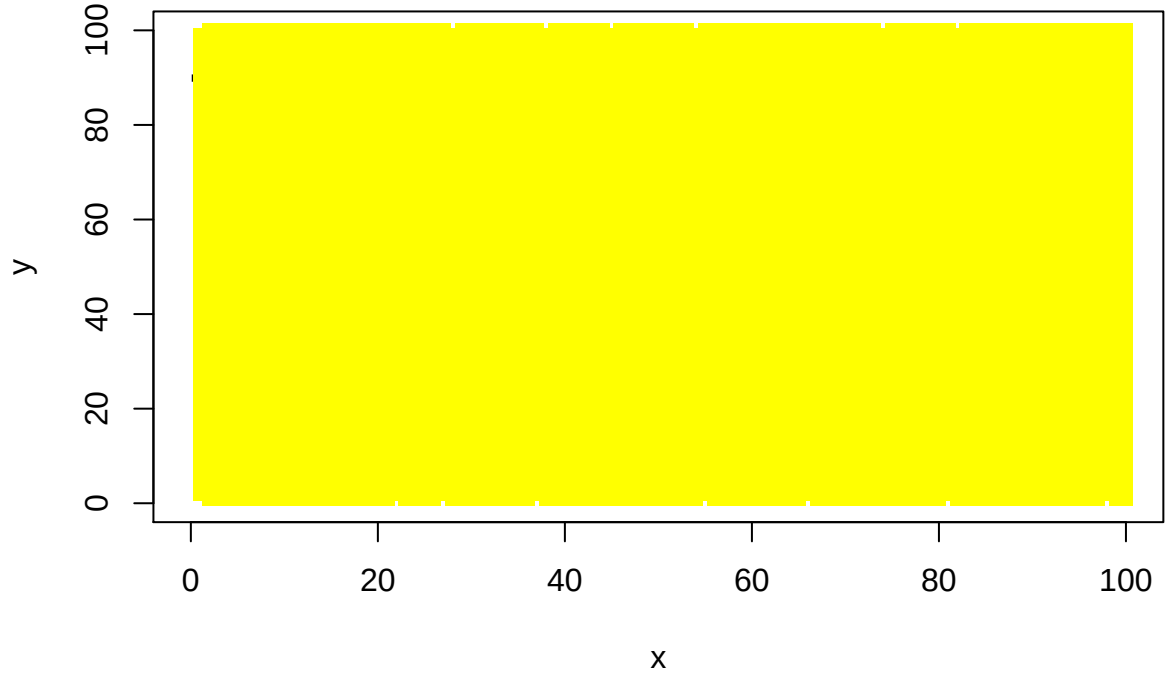
```
mc.n <- 800
mc800.xy<-sobol(mc.n, dim = 2, init = T, scrambling = T, seed = seed4) * 100
head(mc800.xy)
```

```
##           [,1]     [,2]
## [1,] 79.070514 91.40018
## [2,] 54.707766 40.42420
## [3,] 19.232512 51.77678
## [4,]  7.253371 25.57203
## [5,] 68.604934 63.48711
## [6,] 91.317755 17.19748
```

```
plot(mc800.xy)
```

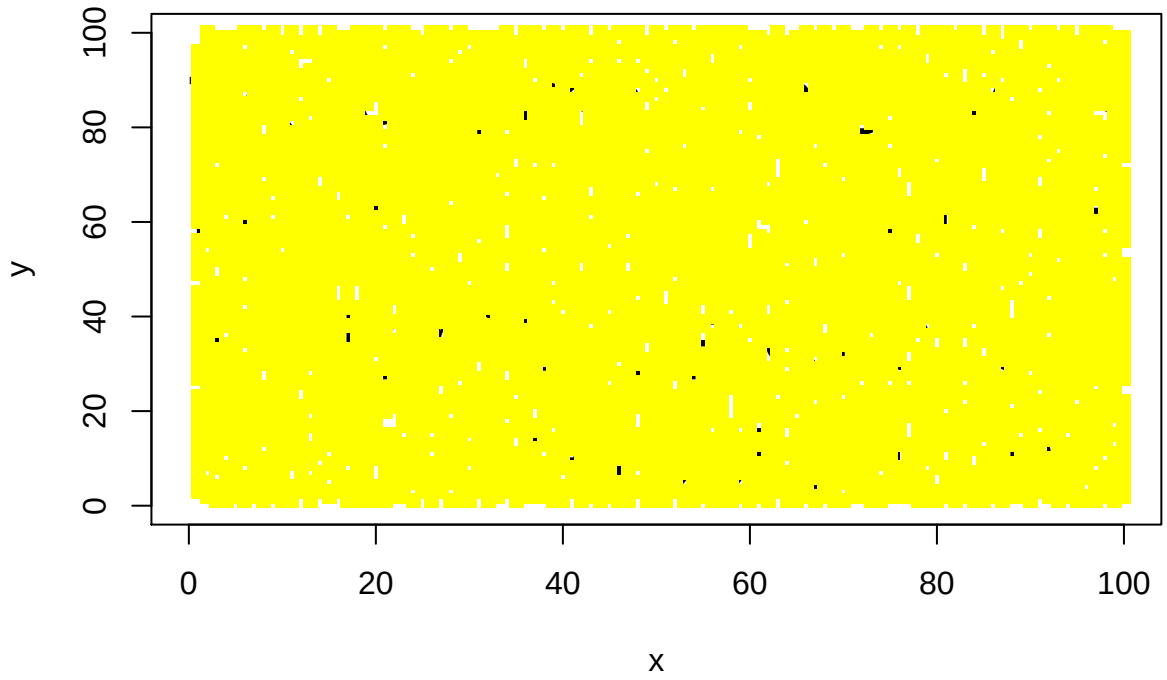



```
ScratchOffMonteCarlo(mc800.xy)
```



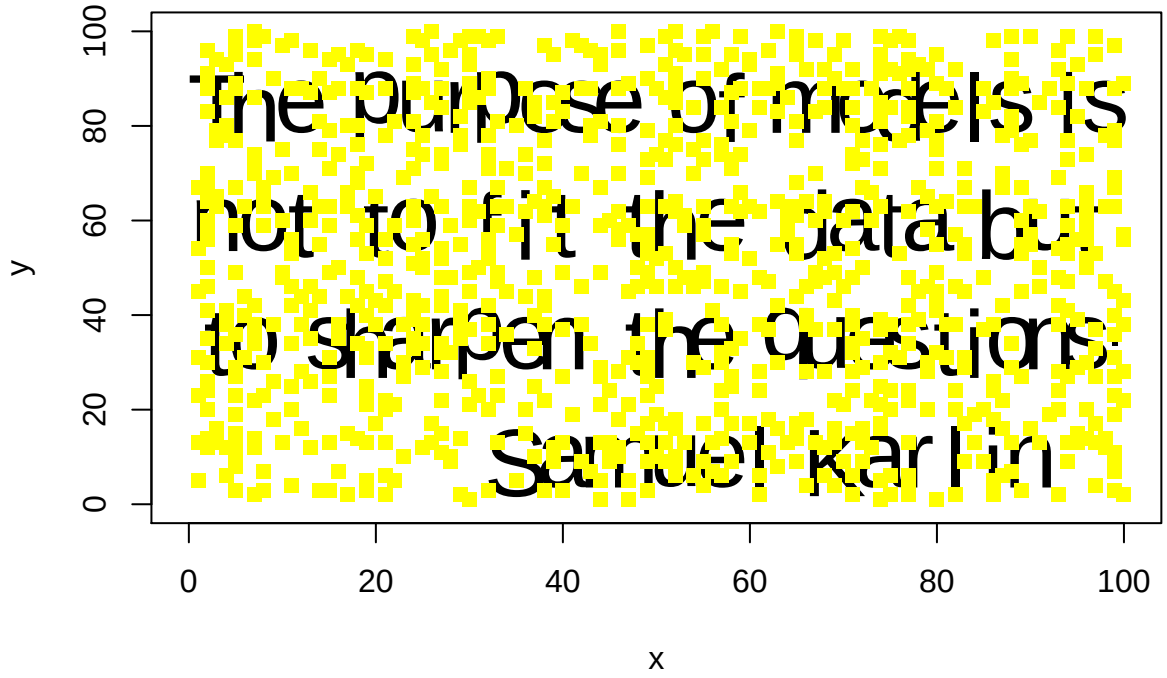
```
## [1] "Size = 800"  "Open (%)= 8"
```

```
# Seed 4 5k:
mc.n <- 5000; mc5k.xy <- sobol(mc.n, dim = 2, init = T, scrambling = T, seed = seed4) * 100
ScratchOffMonteCarlo(mc5k.xy)
```



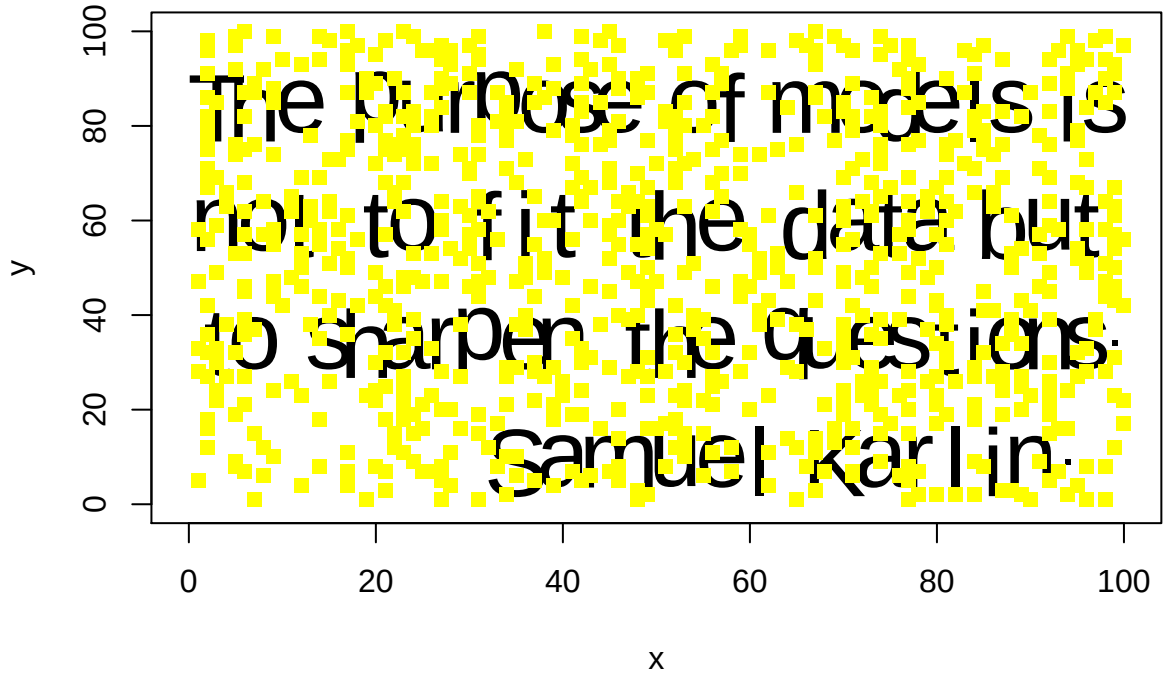

```
## [1] "Size = 5000"     "Open (%)= 45.25"
```

```
# Seed 4 15k:
mc.n <- 15000; mc15k.xy <- sobol(mc.n, dim = 2, init = T, scrambling = T, seed = seed4) * 100
ScratchOffMonteCarlo(mc15k.xy)
```



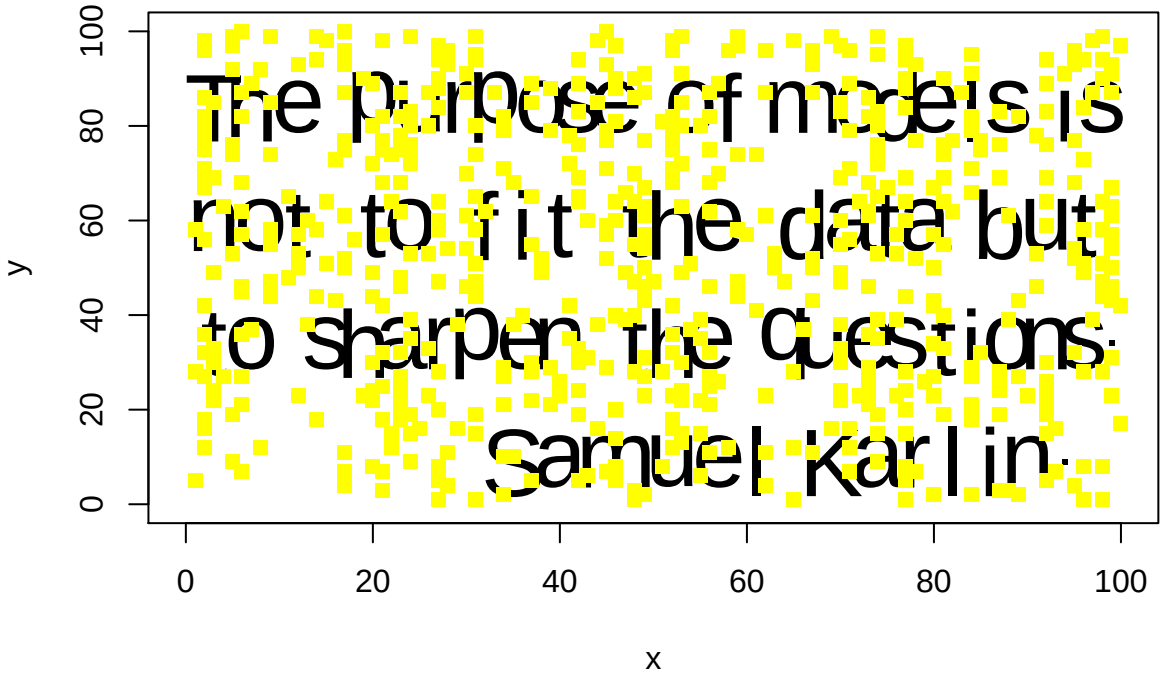

```
## [1] "Size = 15000"    "Open (%)= 90.88"
```

```
# Seed 1 15k:
mc15k.xy <- sobol(mc.n, dim = 2, init = T, scrambling = T, seed = seed1) * 100
ScratchOffMonteCarlo(mc15k.xy)
```




```
## [1] "Size = 15000"    "Open (%)= 91.78"
```

```
# Seed 1 16k:
mc.n <- 16000; mc16k.xy <- sobol(mc.n, dim = 2, init = T, scrambling = T, seed = seed1) * 100
ScratchOffMonteCarlo(mc16k.xy)
```




```
## [1] "Size = 16000"    "Open (%)= 94.25"
```

What percent you needed to scratch off to make the quote readable?

We need approximately the same percent of the image scratched off to make the quote readable, roughly 90%.

Which of the Monte Carlo methods makes the quote readable sooner?

The 90% open hurdle is reached with far fewer "scratches" or data points using the Sobol method vs. the runif() method.

Which parameters nSample and my.seed gave you the best result, what percent of the yellow paint you were able to scratch off by each method?

With the runif() method we used a sample of 25,000 to achieve ~90% scratched off. Our seed of 112233 seemed minimally optimal above the other seeds we tried.

With the Sobol method we used a sample of 16,000 to achieve ~90% scratched off. Our seed of 938364 seemed minimally optimal above the other seeds we tried.

Changing which of the two parameters plays more significant role?

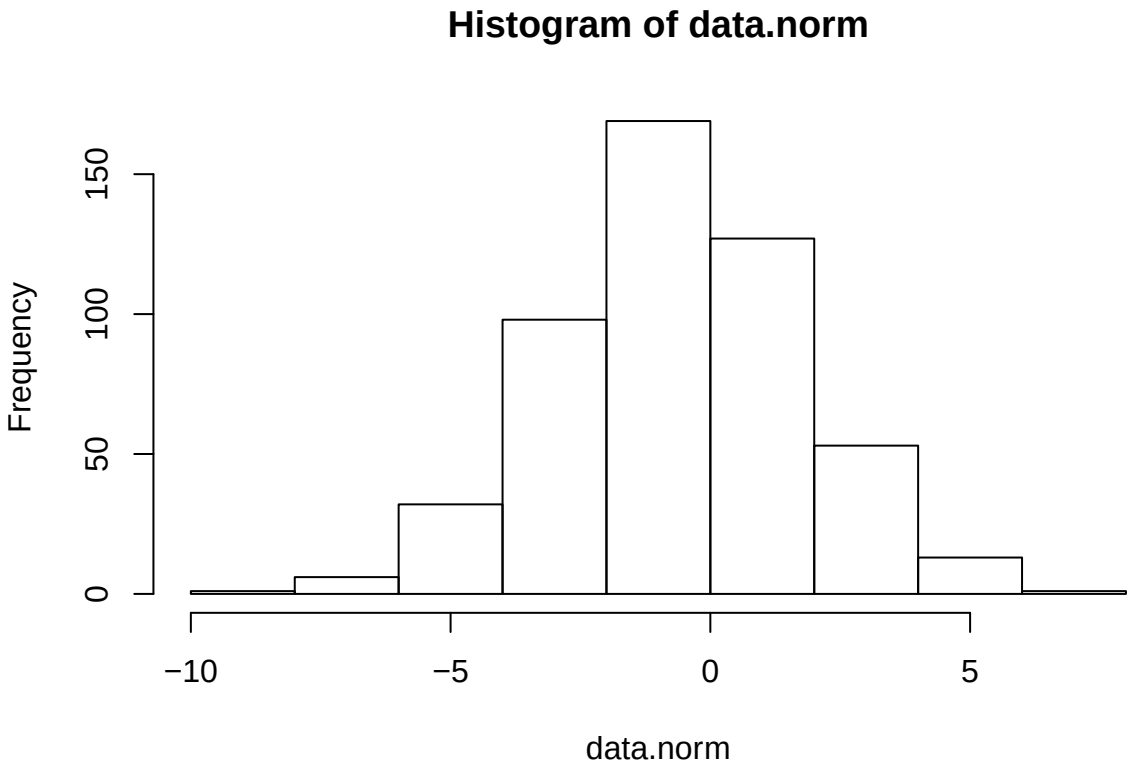
While changing the seed altered the particular dispersion of paint that was covered and sometimes revealed letters that gave more useful clues, this effect (of making the quote more readable through changing the seed) had more to do with the randomness of the letters made readable and its impact on reading ability, known informally as typoglycemia. Typoglycemia is a neologism referring to the psycholinguistic phenomenon of being able to read words without each letter or with the letters being scrambled.

Humans, as described by this phenomenon, may not actually read every letter, but instead process key letters. When we change the seed, roughly the same amount of paint is scratched and roughly the same number of letters. However, *different* letters are revealed. Therefore we suspect that typoglycemia and randomness is largely responsible for any increased ability to read the quote due to changing the seed.

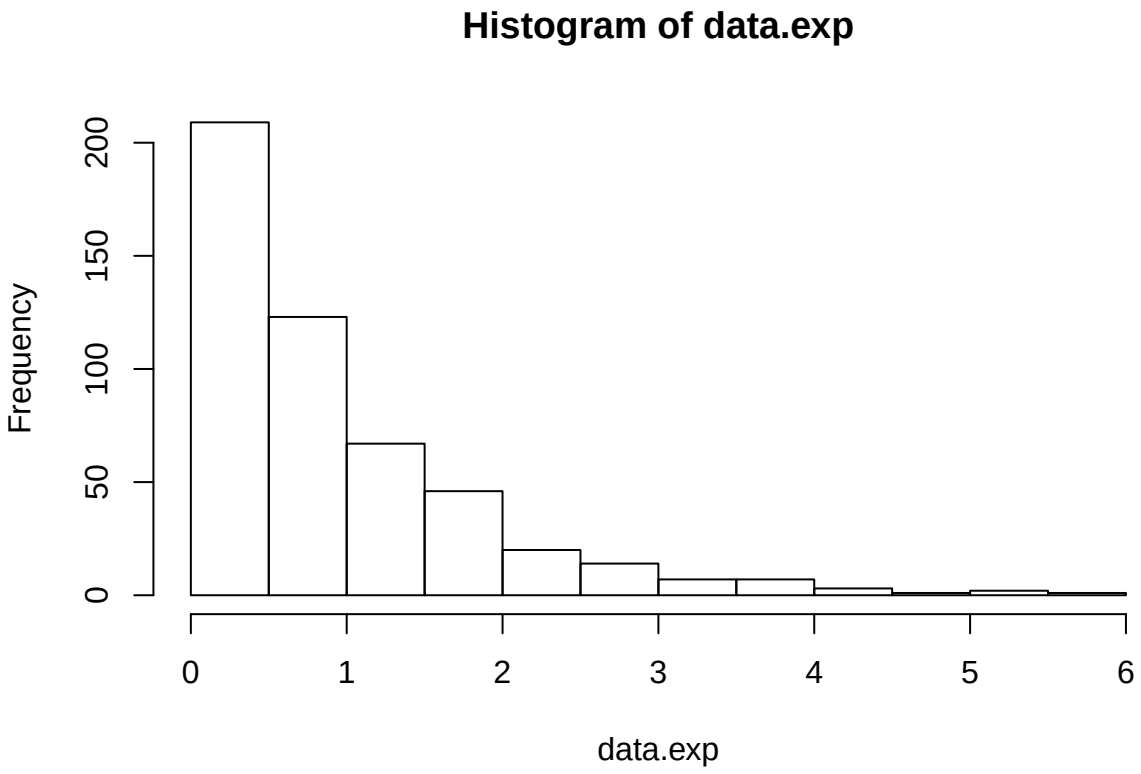
This effect is far less significant than the effect of altering the sample size, which has the effect of increasing the % of the image revealed, and correspondingly the number of readable letters. The more readable letters, the more we can read, though clearly due to typoglycemia not all letters must be distinguishable in order for our brain to link the sequence to recognizable patterns.

Part 5. Test

```
# Load data. (Data file must be located in RStudio project directory.)
data <- read.csv('Week2_Test_Sample.csv', header = TRUE)$x

# Using sample from uniform distribution, create sample from normal distribution.
data.norm <- qnorm(data[4:503], mean = data[1], sd = data[2])
hist(data.norm)
```




```
# Using sample from uniform distribution, create sample from exponential distribution.
data.exp <- qexp(data[4:503], rate = data[3])
hist(data.exp)
```




```
# Output results to RStudio project directory
result <- cbind(datNorm = data.norm, datExp = data.exp)
write.csv(result, 'result.csv', row.names = FALSE)
```