Assignment 2

Matthew Dunne

1 Generate uniformly distributed random numbers

1.1 Use runif()

```
set.seed(15)
Sample<-runif(1000,0,1)
```

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

```
suppressWarnings(library(random))
nFlips<-1000
dataFromRandom<-randomNumbers(n=nFlips, min=0, max=1, col=1, base=2, check=TRUE)
head(dataFromRandom)
```

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

1.3 Downloading data from Random.org directly

As I had no trouble downloading data from random.org through the procedure in 1.2, I will not go through this step.

1.4 Turning binary sequence to uniform random numbers

You can create a function that turns turns a sequence of zeros and ones of length n into decimal form. For example the 6-bit binary code representing the number 62 is: 1,1,1,1,1,0. If we take this sequence and convert it to decimal form, we get:

```
bitsToInt<-function(x) {
    packBits(rev(c(rep(FALSE, 32-length(x)%%32), as.logical(x))), "integer")
}
bitsToInt(c(1,1,1,1,1,0))/2^6
```

```
## [1] 0.96875
```

Let us turn the sequence of zeros and ones dataFromRandom of length 1000 into a matrix with 10 columns and 100 rows.

```
Binary.matrix<-matrix(dataFromRandom,ncol=10)
head(Binary.matrix)
```

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

Now, we transform the binary code contained within each row of the matrix into decimal format. Let us look at the first few numbers.

```
dataFromRandom.dec<-as.vector(apply(Binary.matrix,1,bitsToInt))/2^10
head(dataFromRandom.dec)
```

```
## [1] 0.9042969 0.8857422 0.2480469 0.0625000 0.7490234 0.8964844
```

These are all decimals between 0 and 1. This is the equivalent of the sample obtained by runif(), although they are not the same numbers in the result. It is only the process that is equivalent.

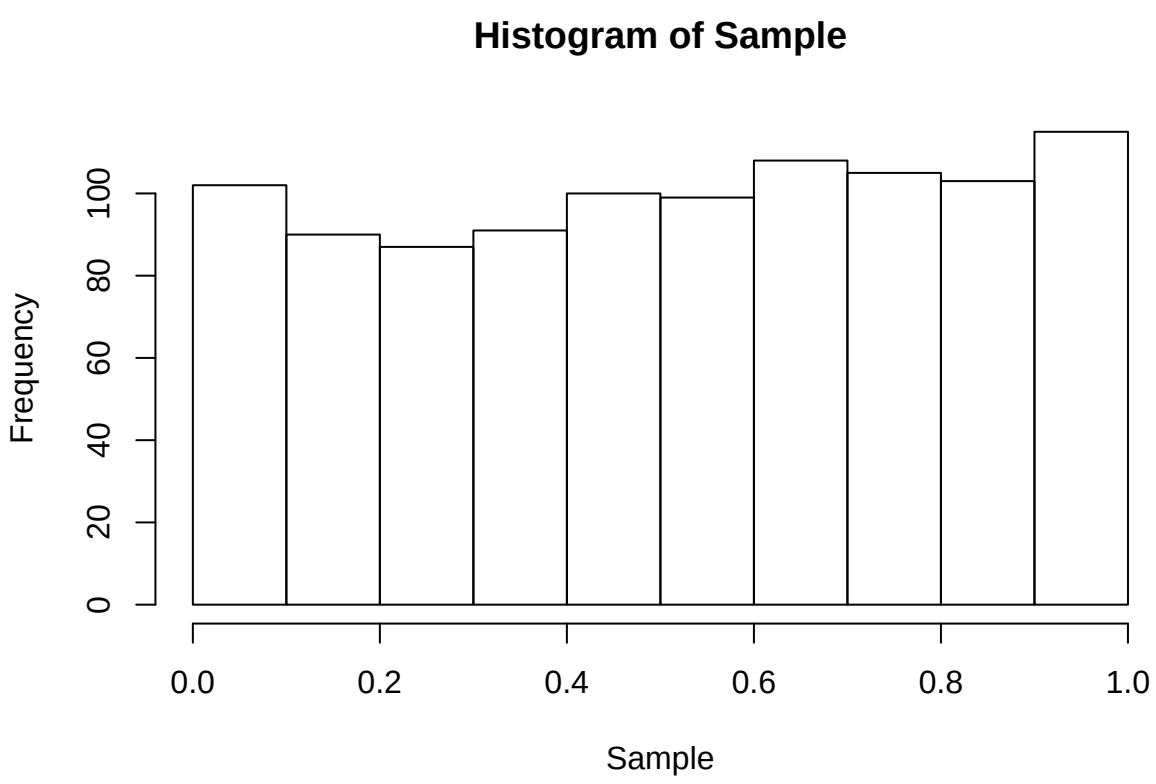
2. Test random number generators

2.1 Test uniformity of distribution of both random number generators

2.1.1 Using Sample generated by runif()

Analyze what was simulated by first looking at the histogram.

```
Sample.histogram<-hist(Sample)
```



```
Sample.histogram
```

```
## $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"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

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

The freqency of each bin in the histogram is about 100. So the distribution is approximately uniform. As we intended to generate a vector or numbers that is more or less uniformly distributed the goals of the simulation are met.

Estimate mean and standard deviation of Sample.histogram$density.

the mean:

```
(Sample.histogram.mean<-mean(Sample.histogram$density))
```

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

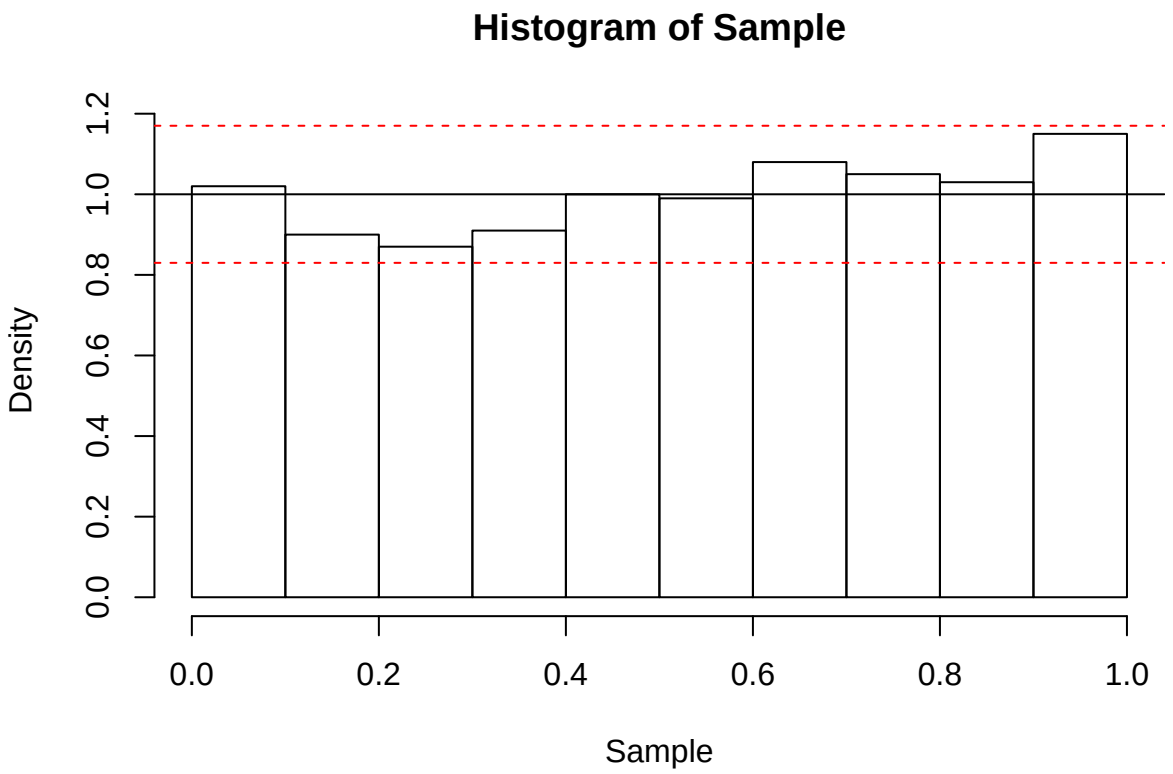
the standard deviation:

```
(Sample.histogram.sd<-sd(Sample.histogram$density))
```

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

And then re-plot the histogram with lines representing the 95% confidence interval from the mean.

```
plot(Sample.histogram,freq=FALSE,ylim=c(0,Sample.histogram.mean+2*Sample.histogram.sd))
abline(h=Sample.histogram.mean)
abline(h=Sample.histogram.mean+1.96*Sample.histogram.sd,col="red",lty=2)
abline(h=Sample.histogram.mean-1.96*Sample.histogram.sd,col="red",lty=2)
```



What does the graph tell us about the observed distribution? The values of the bins (density) are within the 95% confidence interval of the mean of the density. If we were conducting a hypothesis test, we would not reject the null hypothesis of the density being equal to 1, i.e. that it is a random uniform distribution.

Estimate moments of Sample.

mean of sample:

```
(Sample.mean<-mean(Sample))
```

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

variance of sample:

```
(Sample.variance<-var(Sample))
```

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

What can we conclude about the estimated distribution from the moments? The mean is about 0.5, and there is a very small variance. Furthermore, 0.5 is within the 95% confidence interval of the mean of Sample. So, we can conclude that the mean is probably 0.5, or at least we would not reject a null hypothesis to that effect.

Check the summary of the simulated sample.

```
summary(Sample)
```

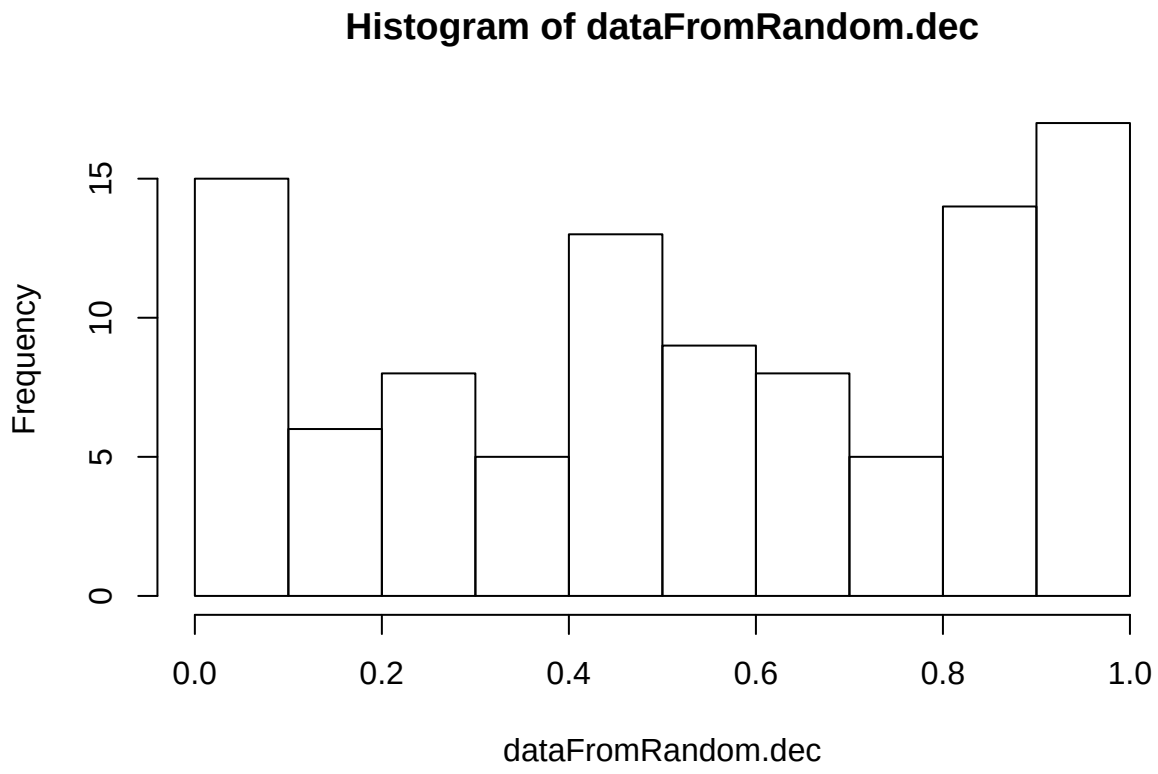
```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## 0.0000127 0.2678376 0.5209934 0.5161848 0.7619871 0.9980255
```

What do you think is the best way of estimating uniform distribution over unknown interval? Get the mean of the uniform distribution. (the right most point in the interval + the leftmost part of the interval)/2 or (b+a)/2

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

First, create an histogram from the data we generated by downloading from random.org, converting to binary, and then converting to decimal.

```
Sample.histogram<-hist(dataFromRandom.dec)
```



and then look at the relevant information:

```
Sample.histogram
```

```
## $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] 15  6  8  5 13  9  8  5 14 17
##
## $density
##  [1] 1.5 0.6 0.8 0.5 1.3 0.9 0.8 0.5 1.4 1.7
##
## $mids
##  [1] 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85 0.95
##
## $xname
## [1] "dataFromRandom.dec"
```

```
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

The mean of the density is:

```
(Sample.histogram.mean<-mean(Sample.histogram$density))
```

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

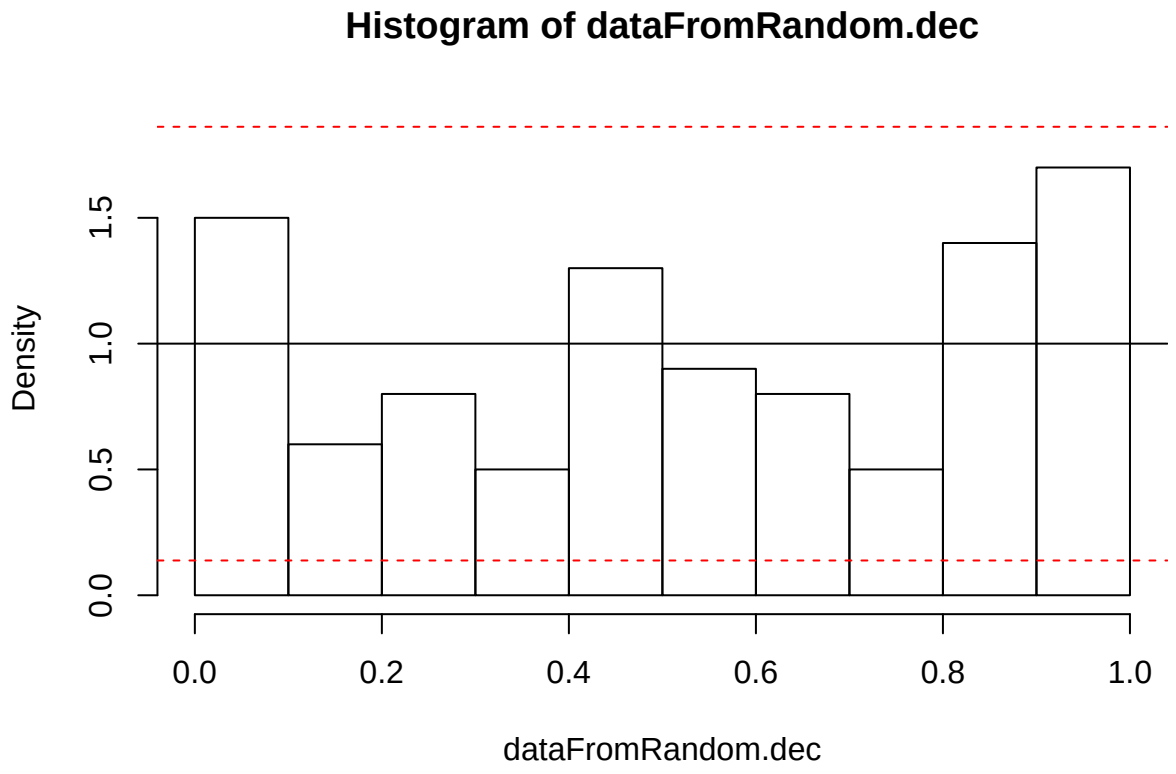
and its standard deviation is:

```
(Sample.histogram.sd<-sd(Sample.histogram$density))
```

```
## [1] 0.4396969
```

Plotting this distribution as before we see the following:

```
plot(Sample.histogram,freq=FALSE,ylim=c(0,Sample.histogram.mean+2*Sample.histogram.sd))
abline(h=Sample.histogram.mean)
abline(h=Sample.histogram.mean+1.96*Sample.histogram.sd,col="red",lty=2)
abline(h=Sample.histogram.mean-1.96*Sample.histogram.sd,col="red",lty=2)
```



The mean of this distribution is:

```
(Sample.mean<-mean(dataFromRandom.dec))
```

```
## [1] 0.5340039
```

And the variance is:

```
(Sample.variance<-var(dataFromRandom.dec))
```

```
## [1] 0.1041134
```

Here is the summary of the data:

```
summary(dataFromRandom.dec)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.003906 0.249512 0.561523 0.534004 0.863037 0.999023
```

2.2 Test independence of the sequence of zeros and ones

2.2.1 Turning point test
Turning point test is used to check if a sequence of numbers is i.i.d. (independent identically distributed). The test is 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.

Let us test if the random decimal data we generated from random.org is i.i.d.

```
suppressWarnings(library(randtests))
turning.point.test(dataFromRandom.dec)
```

```
##
##  Turning Point Test
##
## data:  dataFromRandom.dec
## statistic = -1.5159, n = 100, p-value = 0.1295
## 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. Here we see a p-value well above 0.05 so we do not reject the null hypothesis. We do not reject randomness.

2.2.2 Test frequency by Monobit test
To perform Monobit test you need to transform your {0,1} sample into {-1,1}.

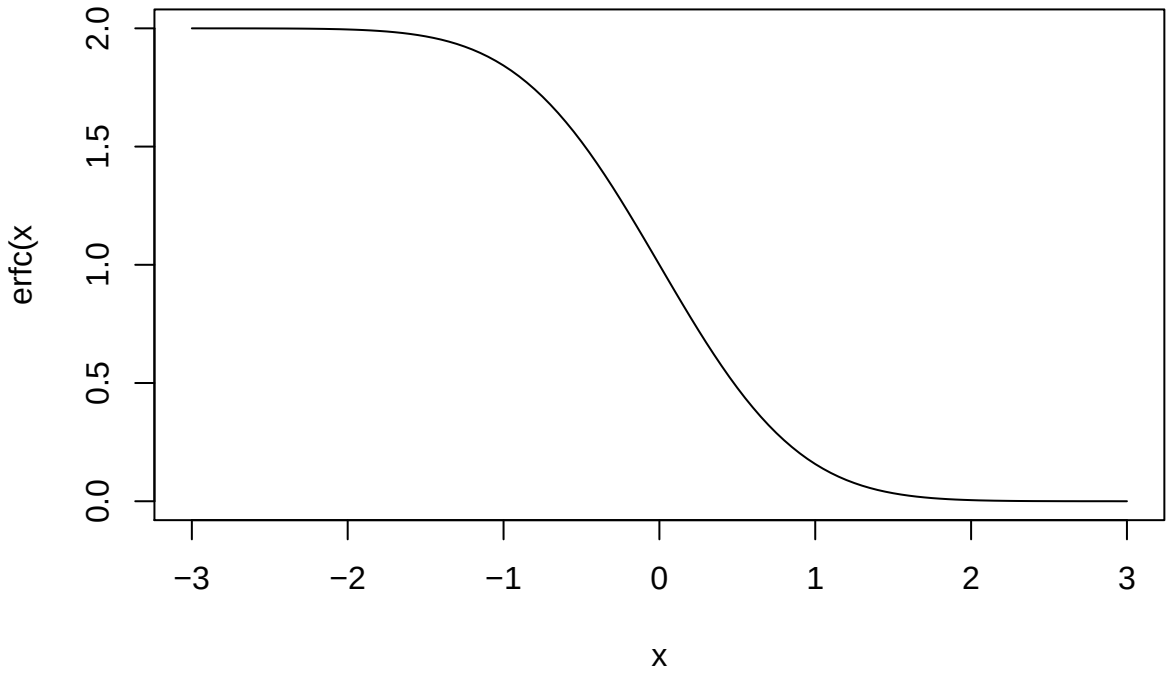
```
dataFromRandom.plusminus1<-(dataFromRandom-.5)*2
```

We need to calculate two functions: erf and erfc. erfc is the complimentary error function, a special function complimentary to error function erf=1-erfc.

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

The complimentary error function looks as follows:

```
plot(seq(from=-3,to=3,by=.05),erfc(seq(from=-3,to=3,by=.05)),type="l",xlab="x",ylab="erfc(x")
```



To test the sequence Ri check the value erfc(S). If the P-value or erfc(S) is less or equal than 0.01 the sequence fails the test.

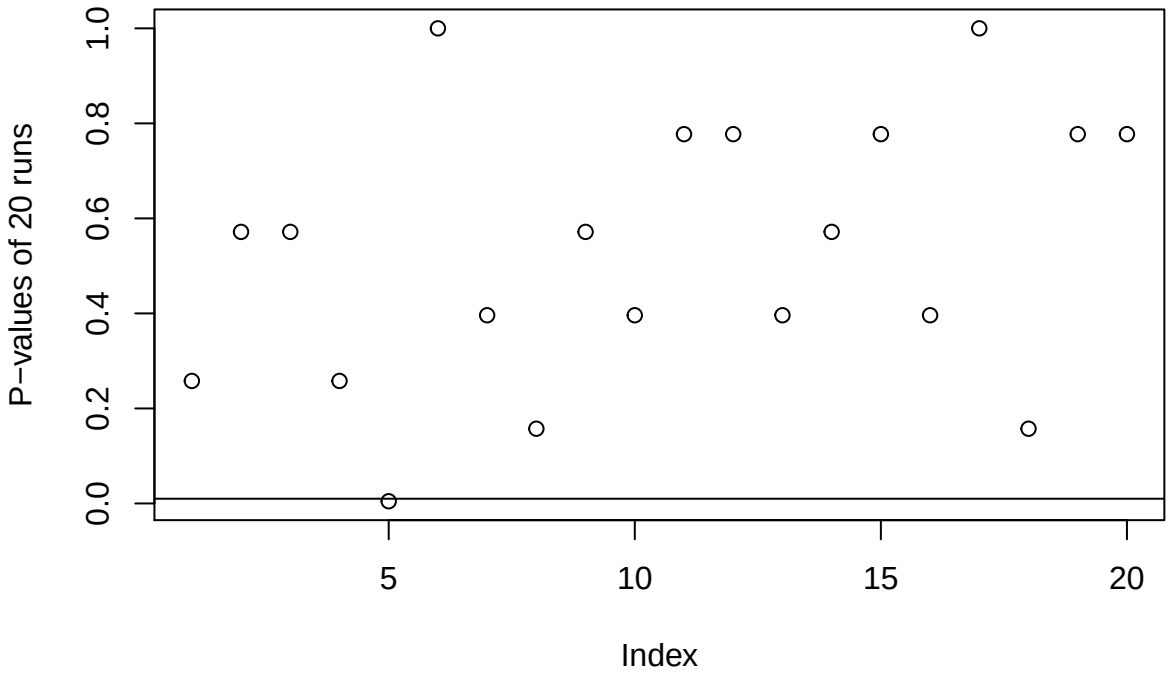
```
erfc(abs(sum(dataFromRandom.plusminus1)/sqrt(2*nFlips)))
```

```
## [1] 0.486616
```

The test shows that the Random.org sequence passes.

Now check each of the sub-sequences created earlier:

```
plot(erfc(abs(apply(matrix(dataFromRandom.plusminus1,ncol=50),1,sum))/sqrt(2*50)),ylab="P-values of 20 
abline(h=.01)
```



These are the p-values of 20 simulated runs. How many runs out of 20 fail the test?

```
sum(erfc(abs(apply(matrix(dataFromRandom.plusminus1,ncol=50),1,sum))/sqrt(2*50))<=.01)
```

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

3 Invent a random number generator

Description of your random number generator

The data in this section is an online random number generator that is meant to randomly generate numbers for the Illinois Lottery Quick Pick Game. It can be found at https://www.random.org/quick-pick/?tickets=20&lottery=6x52.0x0.

I generated 20 tickets, each with 6 numbers. Because a Quick Pick ticket will give numbers in ascending order, I scrambled the vector of 120 numbers so there was no such effect.

Generated sequence

The generated sequence is as follows:

```
IllinoisLotto<-c(02,07,25,39,41,42,01,04,12,31,43,46,02,04,11,15,19,25,26,28,33,39,40,48,05,12,15,43,50
IllinoisLotto<-c(IllinoisLotto, 05,12,30,35,41,45, 27,35,39,45,46, 51, 35,42,44,46,49,52,15,25,33,38,39
IllinoisLotto<-c(IllinoisLotto, 10,14,28,30,33,45,10,11,12,18,29,43, 14,20,30,35,45,52, 11,21,39,41,44,5
IllinoisLotto<-c(IllinoisLotto, 27,29,32,42,48,50, 03,08,30,36,47,52, 11,17,18,19,20,26, 20,22,23,29,44
set.seed(10)
```

```
randomlotto<-sample(IllinoisLotto, size=120, replace = FALSE)
print(randomlotto)
```

```
##   [1] 10 27 33 41 31 12 12  5 12 52 43 14  2 30 35 46 42 43 46  4 16 51 35
##  [24] 41 39 11 11 40 29 30 15 12 11 42 46 20 18 21 23 52 32 26  7 33 36 26
##  [47] 52 45  4 45  5 33 19 46 30 19 51 47 51 44 20  1 11 44 49 51 30 48 41
##  [70] 46 39 15 44 20 47 22 48  9 43 21  3  8 39 39 22  2 27 31 17 10 19 33
##  [93] 42  4 27 50 23 45 35 45 28 18 28 25 28 51 15 39 29 38 14 19 50 25 35
## [116] 11 39 25 29 44
```

If we convert this to 8-bit binary it looks, in part, like this:

```
randomlottobinary<-(as.vector(sapply(randomlotto,function(z) head(intToBits(z),8)))==1)*1
head(randomlottobinary)
```

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

And if we use that vector to generate a random uniform distribution we get:

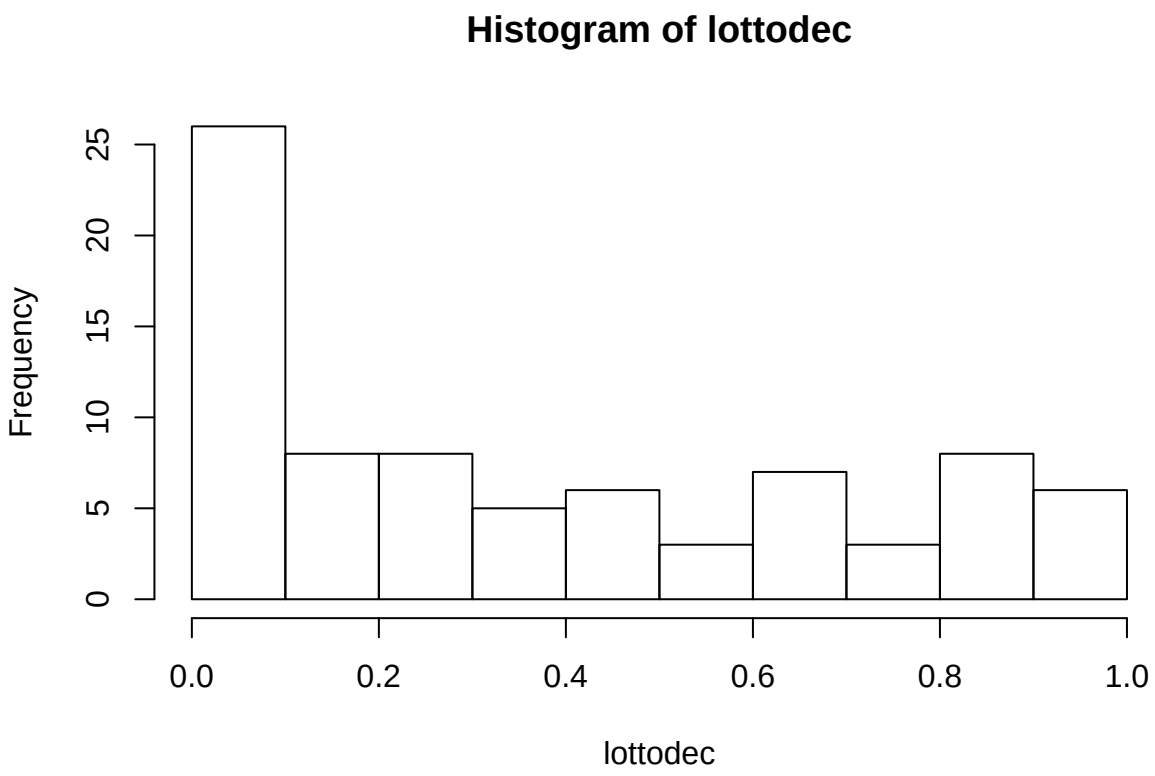
```
Binary.matrix<-matrix(randomlottobinary,ncol=12)
lottodec<-as.vector(apply(Binary.matrix,1,bitsToInt))/2^12
lottodec
```

```
##  [1] 0.33496094 0.81958008 0.09057617 0.81323242 0.13427734 0.28515625
##  [7] 0.00000000 0.00000000 0.65356445 0.91088867 0.31640625 0.84960938
## [13] 0.65698242 0.14160156 0.00000000 0.00000000 0.74414062 0.49536133
## [19] 0.03759766 0.07568359 0.01635742 0.63208008 0.00000000 0.00000000
## [25] 0.65893555 0.33007812 0.28027344 0.96166992 0.25463867 0.74414062
## [31] 0.00000000 0.00000000 0.88793945 0.96020508 0.74072266 0.58349609
## [37] 0.52685547 0.48071289 0.00000000 0.00000000 0.14916992 0.43725586
## [43] 0.81640625 0.90649414 0.12255859 0.25927734 0.00000000 0.00000000
## [49] 0.14428711 0.46459961 0.54077148 0.88525391 0.12402344 0.30102539
## [55] 0.00000000 0.00000000 0.86694336 0.26806641 0.61279297 0.42895508
## [61] 0.07250977 0.43115234 0.00000000 0.00000000 0.21850586 0.33300781
## [67] 0.96948242 0.88745117 0.20581055 0.27832031 0.00000000 0.00000000
## [73] 0.03613281 0.13525391 0.99780273 0.18286133 0.69140625 0.61889648
## [79] 0.00000000 0.00000000
```

This at first glance appears to meet the requirements of a random uniform distribution. All the numbers are between 0 and 1, and there is no discernible pattern. Although there are a noticeable amount of 0's in this vector. But does this variable pass the tests outlined in Section 2?

Results of the uniformity test

```
lottodechist<-hist(lottodec)
```



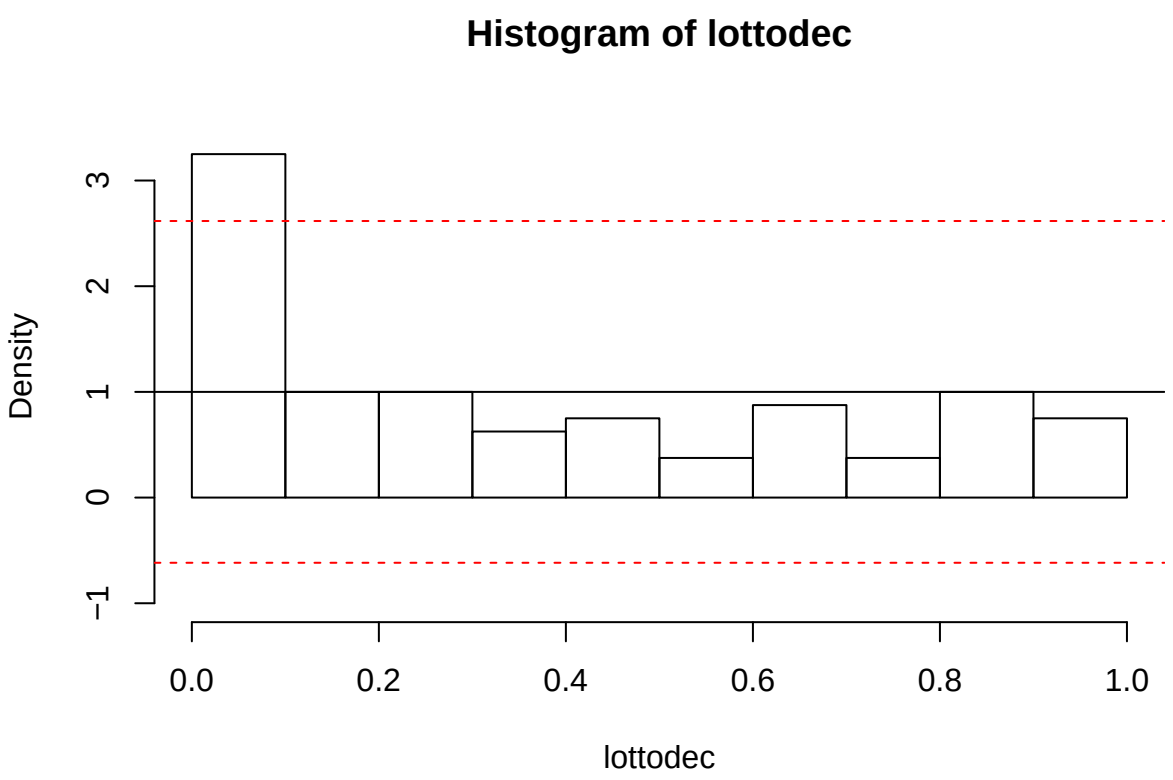

```
(lottodechist.mean<-mean(lottodechist$density))
```

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

```
(lottodechist.sd<-sd(lottodechist$density))
```

```
## [1] 0.8249579
```

```
plot(lottodechist,freq=FALSE,ylim=c(-1,lottodechist.mean+3*lottodechist.sd))
abline(h=Sample.histogram.mean)
abline(h=lottodechist.mean+1.96*lottodechist.sd,col="red",lty=2)
abline(h=lottodechist.mean-1.96*lottodechist.sd,col="red",lty=2)
```



What does the graph tell us about the observed distribution?

There are a great many values in the 0 bin, which makes us suspect something is amiss. The values of all the bins (density) are not within the 95% confidence interval of the mean of the density. If we were conducting a hypothesis test, we would reject the null hypothesis of the density being equal to 1. This suggests that the variable is not random.

Results of the frequency test

To perform Monobit test first transform the data fr {0,1} sample into {-1,1}.

```
lottobinary.plusminus1<-(randomlottobinary-.5)*2
length(lottobinary.plusminus1)
```

```
## [1] 960
```

To test the sequence Ri check the value erfc(S). If the P-value or erfc(S) is less or equal than 0.01 the sequence fails the test. Here there are 960 “flips” or values

```
erfc(abs(sum(lottobinary.plusminus1)/sqrt(2*960)))
```

```
## [1] 7.005482e-14
```

This values is far less than 0.01. So our sequence of data fails the test.

Results of the turning point test

Turning point test is used to check if a sequence of numbers is i.i.d. (independent identically distributed). The test is 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.

Let us test if the random decimal data we generated from random.org is i.i.d.

```
suppressWarnings(library(randtests))
turning.point.test(lottodec)
```

```
##
##  Turning Point Test
##
## data:  lottodec
## statistic = -0.095739, n = 70, p-value = 0.9237
## 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. Here we see a p-value well above 0.05 so we do not reject the null hypothesis. We do not reject randomness.

4 Monte Carlo Method

4.1 Scratch off quote of the day: fuction download

First load the relevant function.

```
datapath<-"C:/Users/mjdun/Desktop/Master Classes/Q1/Statistical Analysis/Lecture 2"
load(file=paste(datapath,'ScratchOffMonteCarlo.rda',sep='/'))
```

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

First select a number of points nSample. We will make this 100.

Simulate a sample of length 2*nSample from uniform distribution on [0,100] and turn it into a (nSample×2) matrix. Use a seed of your choice my.seed.

```
nSample<-14000
set.seed(0)
xy<-runif(2*nSample,0,100)
xy<-matrix(xy,ncol=2)
```

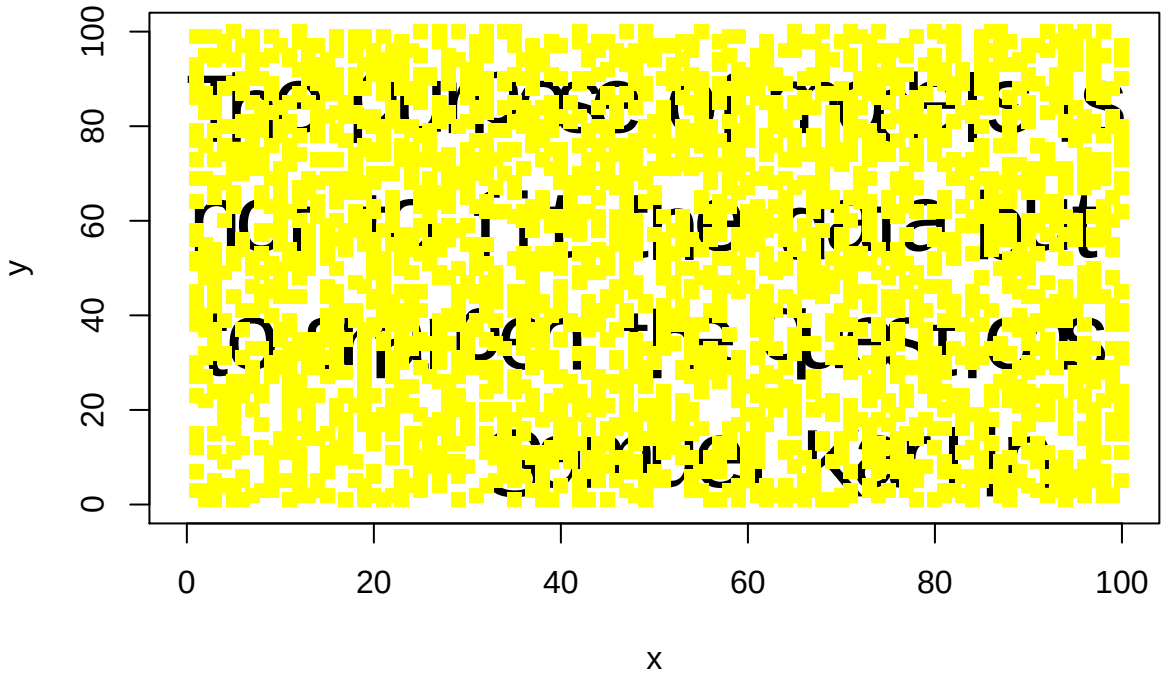
Throw nSample simulated points on square [0,100]×[0,100] to scratch off some of yellow paint.

```
nSample<-14000
set.seed(0)
xy<-runif(2*nSample,0,100)
xy<-matrix(xy,ncol=2)
head(xy)
```

```
##          [,1]      [,2]
## [1,] 89.66972 72.318627
## [2,] 26.55087  9.251632
## [3,] 37.21239 44.674406
## [4,] 57.28534 82.944637
```

```
## [5,] 90.82078 81.237396
## [6,] 20.16819 80.510669
```

```
ScratchOffMonteCarlo(xy)
```




```
## [1] "Size = 14000"    "Open (%)= 75.17"
```

What percent you needed to scratch off to make the quote readable? It appears we need a minimum of about 75% scratched off or 14,000 in nSample in order to make the quote readable.

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

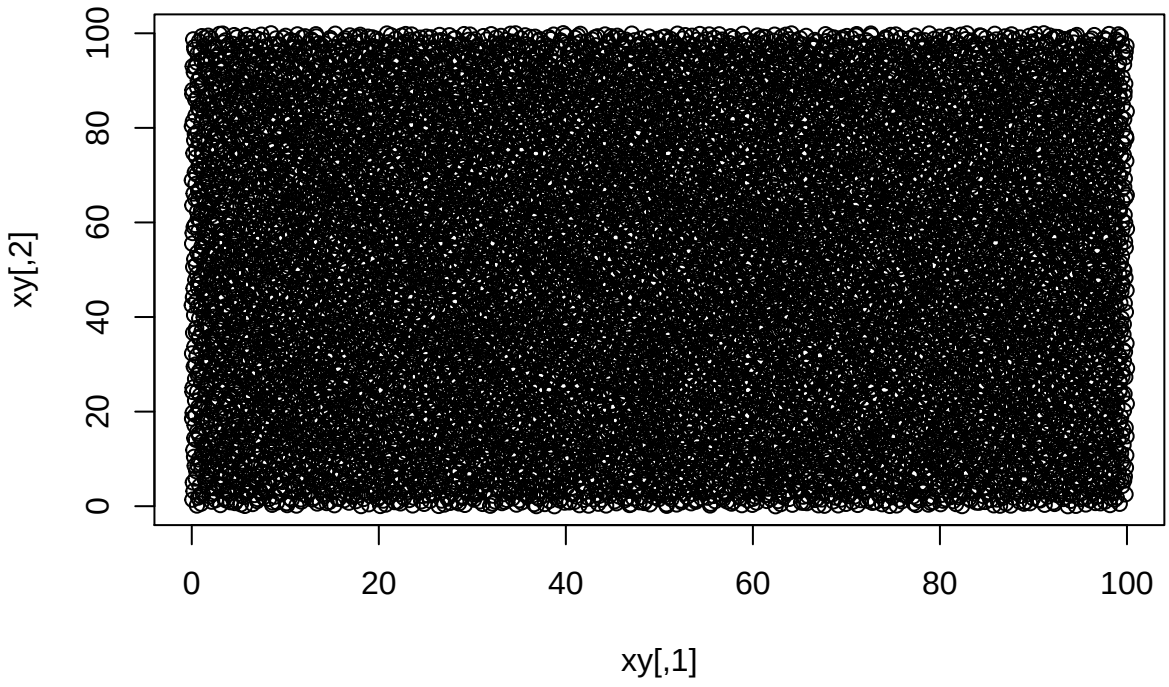
Run sobol() first time with the default set for parameter init=T.

```
suppressMessages(library(randtoolbox))
my.seed<-10
set.seed(my.seed)
nSample<-10
xy<-sobol(nSample,dim=2,init=T)*100
```

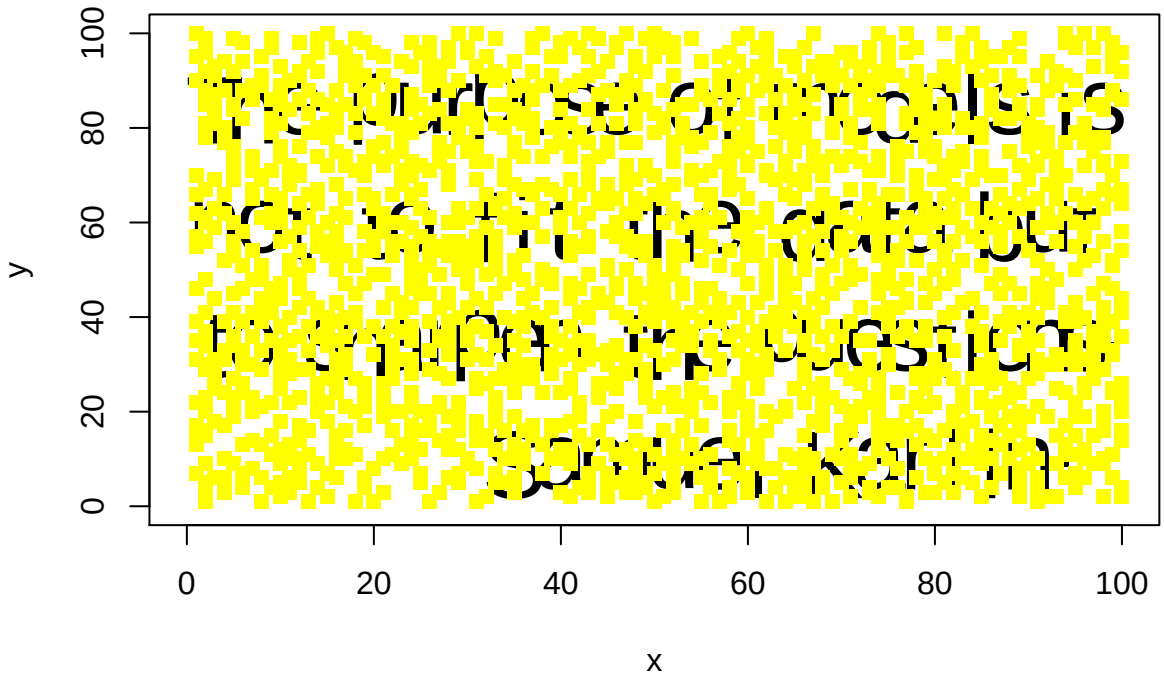
Again, by changing nSample and my.seed try to make the quote of the day readable with minimum sample size.

```
nSample<-12000
my.seed<-9
set.seed(my.seed)
xy<-sobol(nSample,dim=2,init=T,scrambling = T,seed=my.seed)*100
```

```
plot(xy)
```




```
ScratchOffMonteCarlo(xy)
```




```
## [1] "Size = 12000"    "Open (%)= 81.86"
```

What percent you needed to scratch off to make the quote readable? At about 80-82% the quote is readable.

Which of the Monte Carlo methods makes the quote readable sooner? The Sobol method makes the quote readable sooner in terms of nSample. We only had to use 12,000 as opposed to the pseudo random method where we had to use 14,000. However, the Sobol method scratched off a higher percentage, even with the lower sample size, i.e. 82% instead of 75%

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? The Sobol method with nSample 12,000 and seed of 9 gave the best result. It scratched off 82% whereas the pseudo random method scratched off about 75%.

Changing which of the two parameters plays more significant role? Changing nSample had more effect on the result. Changing the seed changed the percentages a little, but not much.

5 Test

First load Week2_Test_Sample.csv,

```
dataPath<-"C:/Users/mjdun/Desktop/Master Classes/Q1/Statistical Analysis/Lecture 2"
dat <- read.csv(paste(dataPath,'Week2_Test_Sample.csv',sep="/"), header=TRUE)$x
```

Extract the relevant variables:

```
mean<-dat[1]
sd<-dat[2]
```

```
intensity<-dat[3]
vector<-dat[4:503]
```

Then create we create vectors in which the random uniform vector is used to generate a normal and exponential distribution. Combine them and write them to a .csv file.

```
datNorm<-qnorm(vector, mean = mean, sd=sd)
datExp<-qexp(vector, rate = intensity)
res<-cbind(datNorm=datNorm,datExp=datExp)
#don't execute this line
#write.csv(res, file = paste(dataPath,'result.csv',sep="/"), row.names = F)
```