Week 2 Assignment

Part 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

```
##       V1
##  [1,]  1
##  [2,]  1
##  [3,]  0
##  [4,]  0
##  [5,]  1
##  [6,]  1
##  [7,]  0
##  [8,]  1
##  [9,]  0
## [10,]  0
## [11,]  1
## [12,]  0
## [13,]  0
## [14,]  1
## [15,]  0
## [16,]  1
## [17,]  1
## [18,]  1
## [19,]  1
## [20,]  0
## [21,]  1
## [22,]  0
## [23,]  1
## [24,]  0
## [25,]  0
```

Learn how to turn your sequence of {0,1} into uniform random numbers on [0,1]

```
suppressMessages(library(compositions))
bin2dec<-function(Bin.Seq){
  unbinary(paste(Bin.Seq,collapse=""))
}
bin2dec(c(1,1,1,1,1,0))
```

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

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

Transform each row of the matrix into decimal format using bin2dec() and divide the numbers by 2^10 to make real numbers in [0,1]

```
Decimal.Sample<-apply(Binary.matrix,1,bin2dec)/2^10
Decimal.Sample
```

```
##   [1] 0.71582031 0.85058594 0.47265625 0.40039062 0.61816406 0.58984375
##   [7] 0.34570312 0.79296875 0.36914062 0.03320312 0.76074219 0.41113281
##  [13] 0.43847656 0.75585938 0.03906250 0.68847656 0.66796875 0.51464844
##  [19] 0.81250000 0.04785156 0.52539062 0.11035156 0.54296875 0.30859375
##  [25] 0.42187500 0.88378906 0.95996094 0.75585938 0.68750000 0.21191406
##  [31] 0.80175781 0.14257812 0.64941406 0.79101562 0.82910156 0.37988281
##  [37] 0.64648438 0.35644531 0.47851562 0.83105469 0.07910156 0.91210938
##  [43] 0.94238281 0.39941406 0.15820312 0.19335938 0.89257812 0.46679688
##  [49] 0.73144531 0.00781250 0.40625000 0.55566406 0.76953125 0.62500000
##  [55] 0.44628906 0.34960938 0.05957031 0.26464844 0.15722656 0.52148438
##  [61] 0.93652344 0.54394531 0.47363281 0.49707031 0.33984375 0.55468750
##  [67] 0.13183594 0.39550781 0.02441406 0.08886719 0.05175781 0.07421875
##  [73] 0.68945312 0.20410156 0.98339844 0.19335938 0.99902344 0.15429688
##  [79] 0.98144531 0.58105469 0.66210938 0.76464844 0.97753906 0.09667969
##  [85] 0.95507812 0.19433594 0.29687500 0.87695312 0.49609375 0.89941406
##  [91] 0.56445312 0.56152344 0.52539062 0.38671875 0.78222656 0.31738281
##  [97] 0.87207031 0.22070312 0.49609375 0.77343750
```

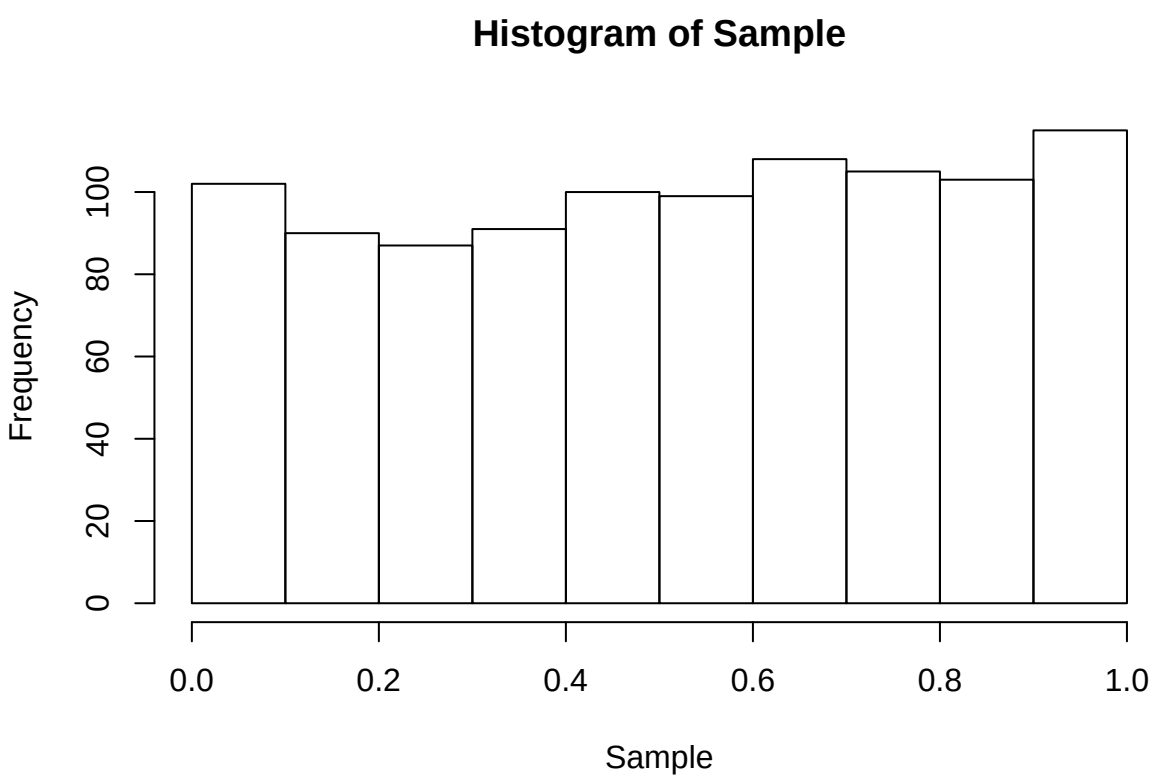
Part 2. Test random number generator

2.1 Test uniformity of distribution of both random number generators

2.1.1. Sample obtained by runif()

Analyze what was simulated 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? This histogram is a uniform distribution. Yes, it is consistent with the goal of the simulation. Which was to create a random uniform set of 1000 points all between 0 and 1.

Estimate mean and standard deviation of Sample.histogram$density. mean = 1, sd = .25

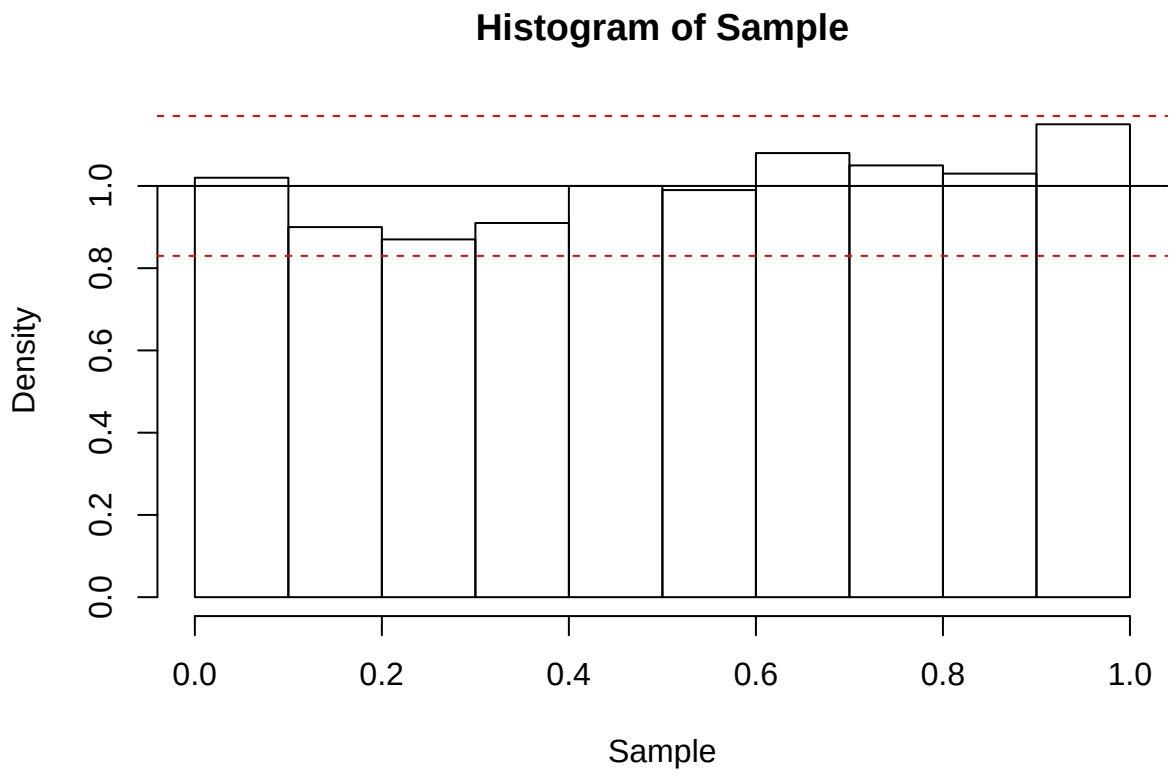
```
(Sample.histogram.mean<-mean(Sample.histogram$density))
```

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

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

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

```
plot(Sample.histogram,freq=FALSE)
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 you about the observed distribution?

It looks like a uniform distribution with mean of 1 and a standard deviation of 0.0868

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

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

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

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

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

The mean of Sample is 0.516 and the variance is 0.0841

Check the summary of the simulated sample.

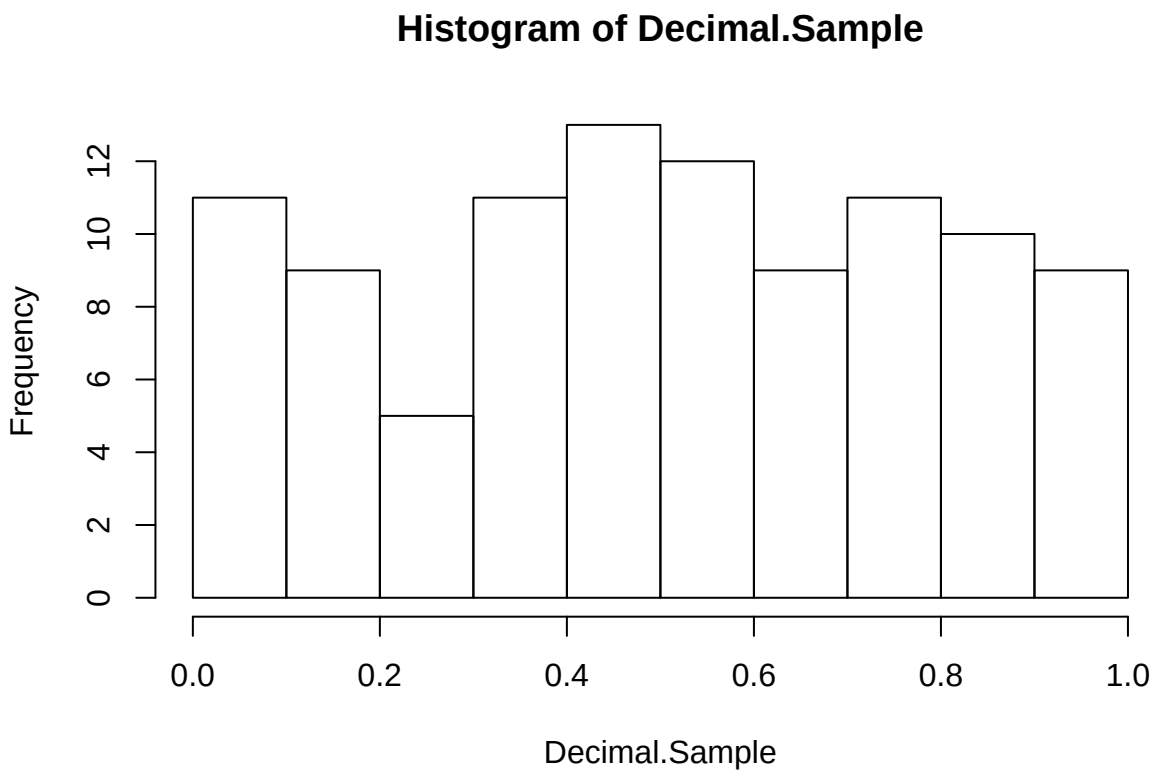
```
summary(Sample)
```

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

What do you think is the best way of estimating uniform distribution over unknown interval? The best way to estimate uniform distribution is by analyzing the moments.

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

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




```
D.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] 11  9  5 11 13 12  9 11 10  9
##
## $density
##  [1] 1.1 0.9 0.5 1.1 1.3 1.2 0.9 1.1 1.0 0.9
##
## $mids
##  [1] 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85 0.95
##
## $xname
## [1] "Decimal.Sample"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

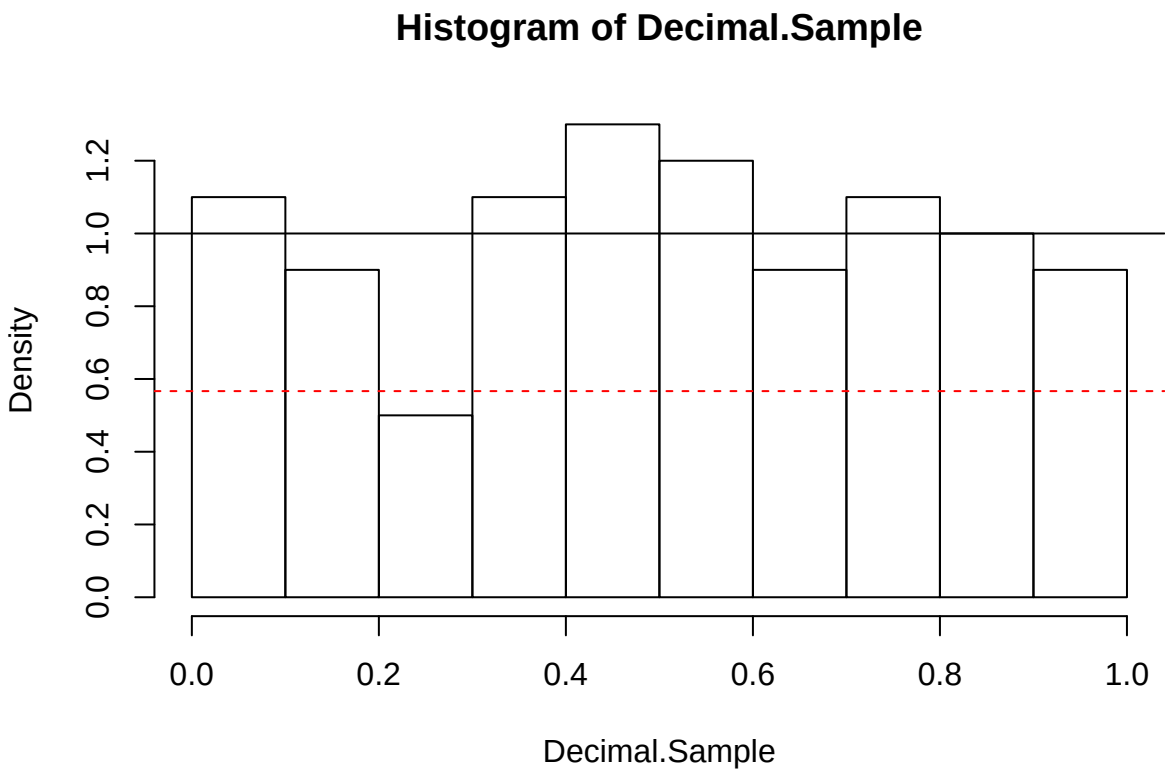
```
(D.Sample.histogram.mean<-mean(D.Sample.histogram$density))
```

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

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

```
## [1] 0.2211083
```

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




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

```
## [1] 0.5119629
```

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

```
## [1] 0.08145397
```

```
summary(Decimal.Sample)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.007812 0.305700 0.518100 0.512000 0.761700 0.999000
```

This distribution has a similar mean, but a larger variance. It still gives a uniform distribution.

2.2. Test independence of the sequence of zeros and ones

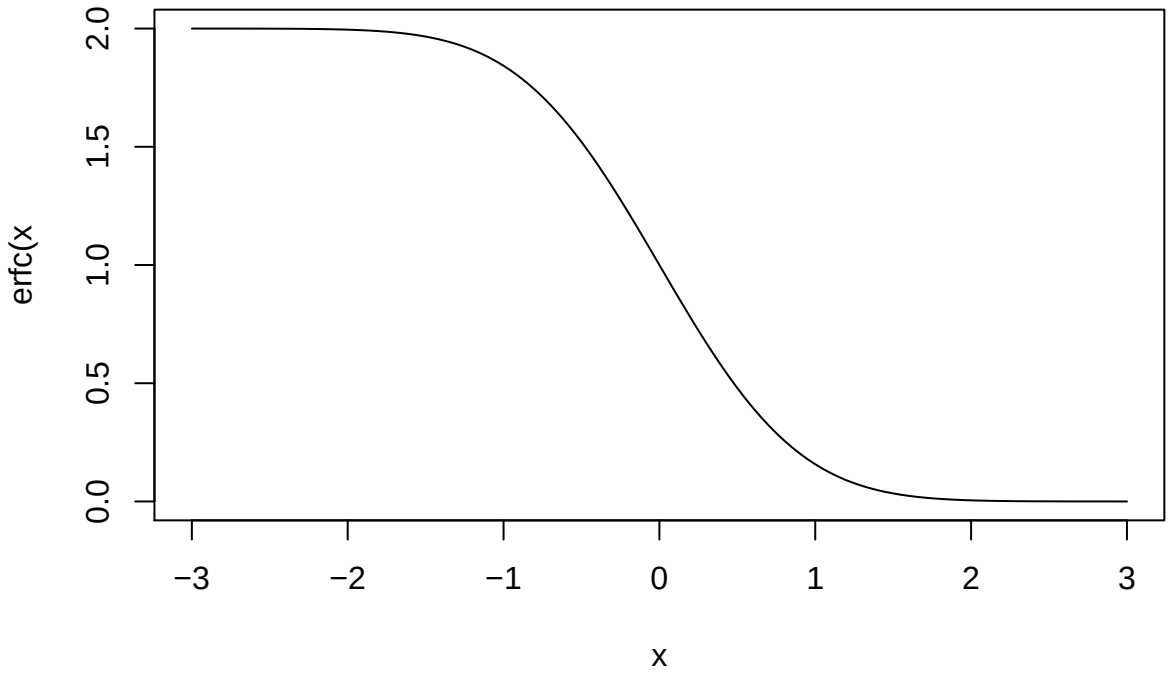
2.2.1. Turning point test

```
library(randtests)
turning.point.test(Decimal.Sample)
```

```
##
##  Turning Point Test
##
## data:  Decimal.Sample
## statistic = 0.63827, n = 100, p-value = 0.5233
## alternative hypothesis: non randomness
```

2.3. Test frequency by Monobit test

```
dataFromRandom.plusminus1<-(dataFromRandom-.5)*2
erf <- function(x) 2 * pnorm(x * sqrt(2)) - 1
erfc <- function(x) 2 * pnorm(x * sqrt(2), lower = FALSE)
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 RiRi check the value erfc(S).

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

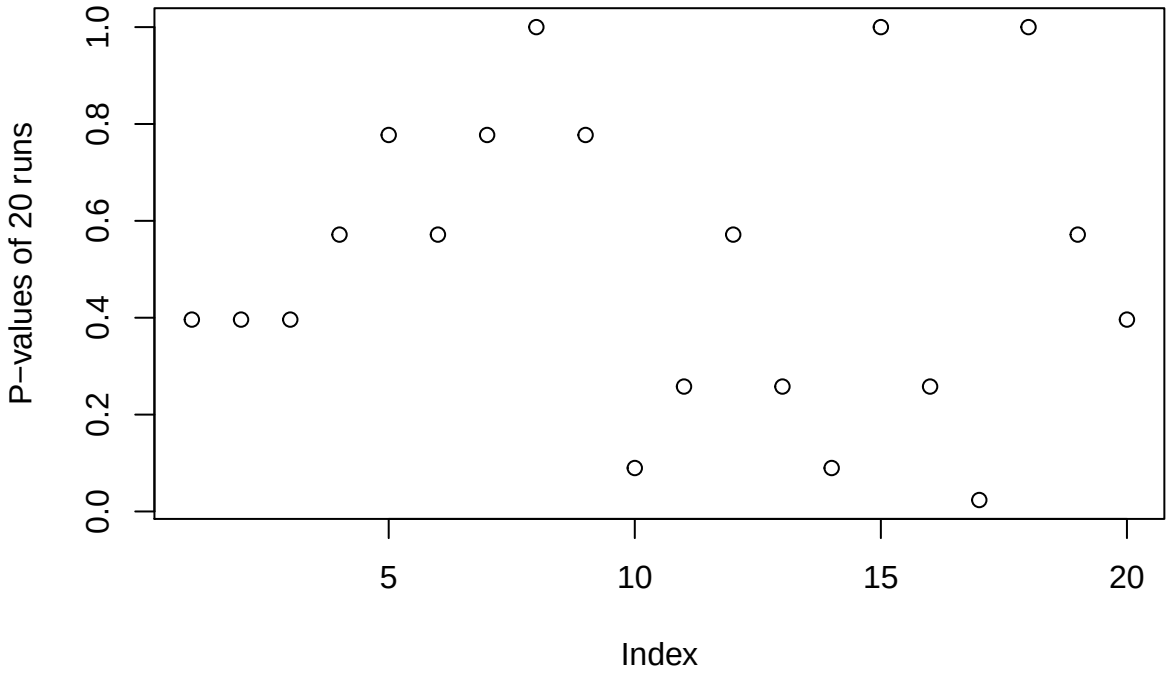
```
erfc(abs(sum(dataFromRandom.plusminus1)/sqrt(2*nFlips)))
```

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

The test shows that our 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 r
```



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] 0
```

None of them fail, there are none less than 0.01

Part 3. Invent a random number generator

Part 4. Monte Carlo Method

4.1. Scratch off quote of the day: fuction download

```
datapath<-"C:/Users/JohntheGreat/Downloads/documents%2FMScA Statistical Analysis 31007%2FMScA 31007 Lec
#load(file=paste(datapath,'ScratchOffMonteCarlo.rda',sep='/'))

##load(file = datapath)
load("C:/Users/JohntheGreat/Downloads/documents%2FMScA Statistical Analysis 31007%2FMScA 31007 Lecture 1
```

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

Select a number o points nSample.

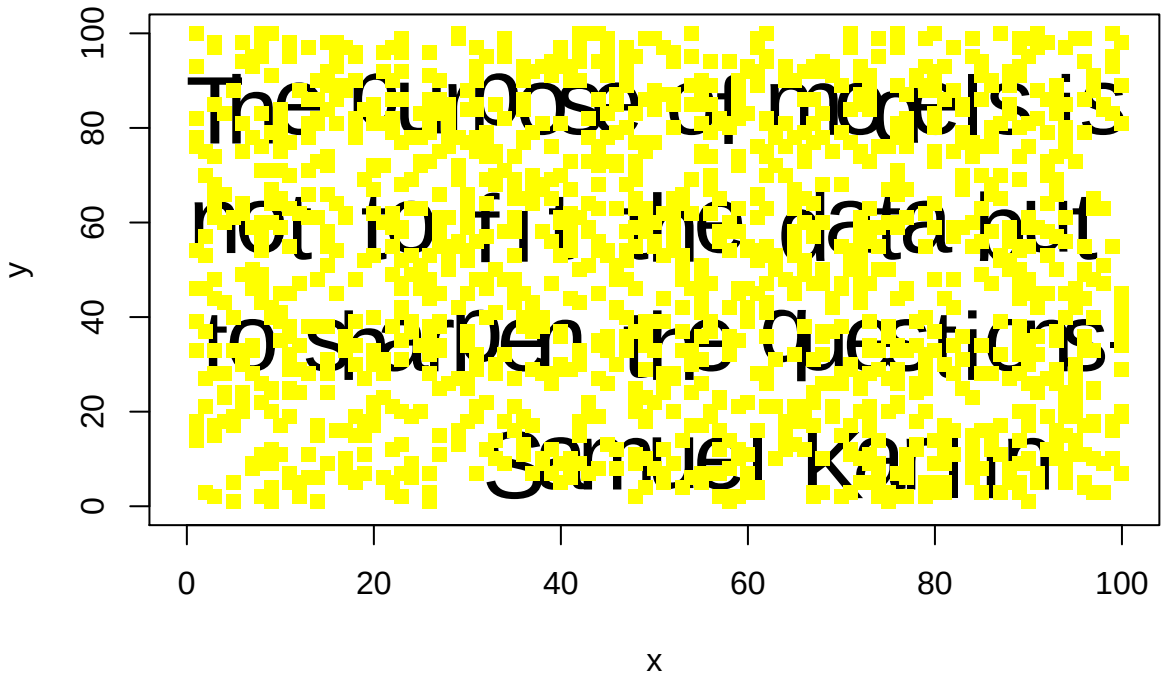
Simulate a sample of length 2*nSample from uniform distribution on [0,100] and turn it into a (nSample×2)(nSample×2) matrix. Use a seed of your choice my.seed.

```
nSample <- 20000
my.seed <- 5555
set.seed(my.seed)
xy<-runif(2*nSample,0,100)
xy<-matrix(xy,ncol=2)
head(xy)
```

```
##           [,1]     [,2]
## [1,] 22.089697 54.15111
## [2,] 12.923022 91.32639
## [3,] 32.718565 83.45741
## [4,] 16.986400 20.32621
## [5,]  8.011574 27.21023
## [6,]  6.727130 28.40253
```

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

```
ScratchOffMonteCarlo(xy)
```



```
## [1] "Size = 20000"    "Open (%)= 86.64"
```

Take a note of the percentage scratched off returned by ScratchOffMonteCarlo(xy)

By changing nSample and my.seed try to make the quote of the day readable with minimum sample size. What percent you needed to scratch off to make the quote readable?

I needed over 86% to make the quote readable, which was nSample <- 20000

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

```
library(randtoolbox)
```

```
## Loading required package: rngWELL
```

```
## This is randtoolbox. For overview, type 'help("randtoolbox")'.
```

```
##
## Attaching package: 'randtoolbox'
```

```
## The following object is masked from 'package:randtests':
##
##     permut
```

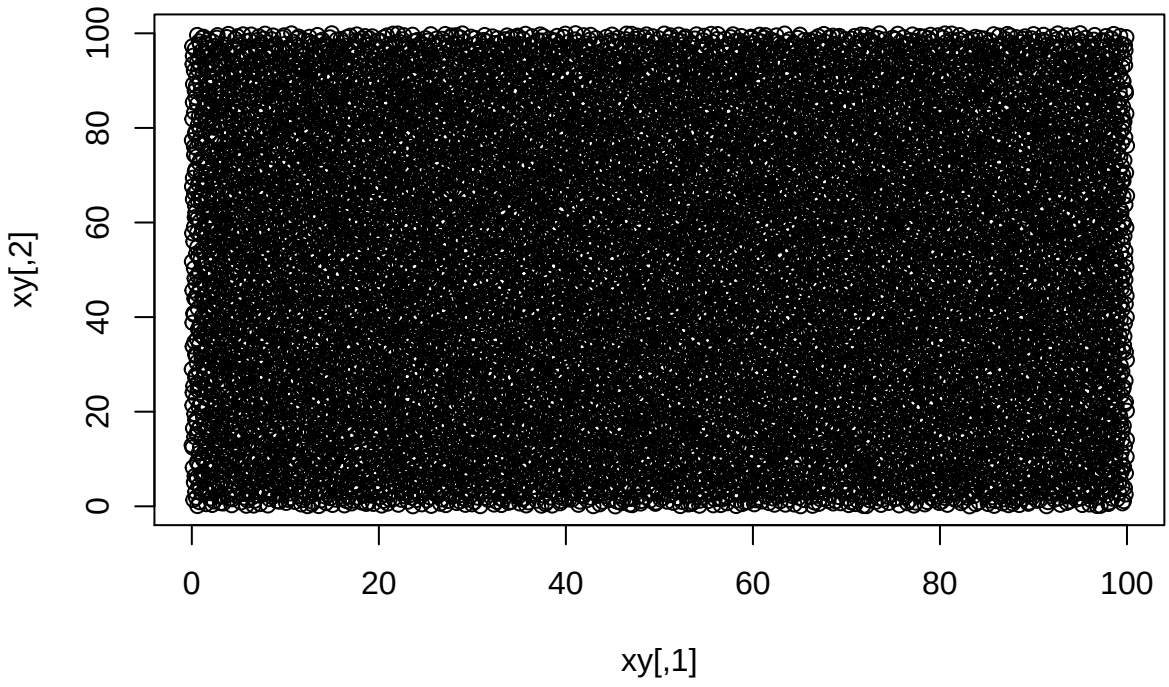
Run sobol() first time with the default set for parameter init=T.

```
my.seed<-6565
set.seed(my.seed)
nSample<-10
xy<-sobol(nSample,dim=2,init=T)*100
```

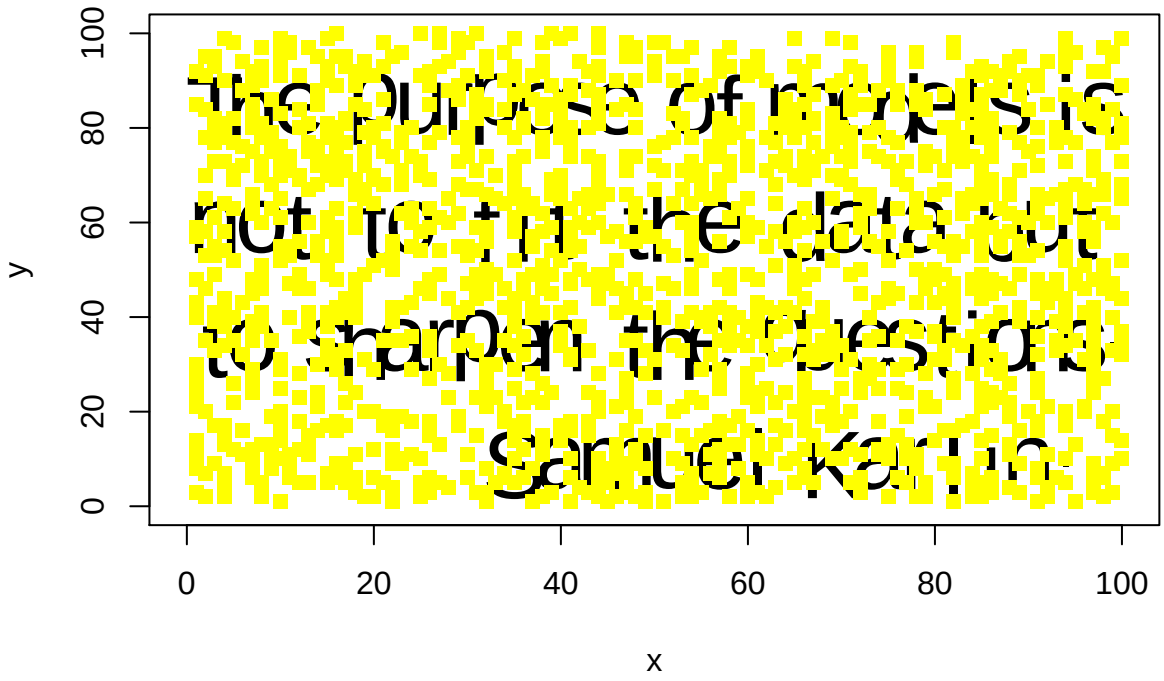
Then make init=F if you want to generate different sample every time or keep it equal to T if you want repeated samples.

```
my.seed<-123
nSample<-13000
xy<-sobol(nSample,dim=2,init=T,scrambling = T,seed=my.seed)*100

plot(xy)
```




```
ScratchOffMonteCarlo(xy)
```




```
## [1] "Size = 13000"    "Open (%)= 86.31"
```

Again, by changing nSample and my.seed try to make the quote of the day readable with minimum sample size. What percent you needed to scratch off to make the quote readable? Again, I needed about 86% open to make it readable

Which of the Monte Carlo methods makes the quote readable sooner? The Sobol method was able to reach a readable percentage with a smaller nSample size

Which parameters nSample and my.seed gave you the best result, what percent of the yellow paing you were able to scratch off by each method? For runif(), I used my.seed <- 5555 with nSample <- 20000 For Sobol, I used my.seed <- 123 with nSample <- 13000 Both got the percent open to around 86% *Changing which of the two parameters plays more significant role?* Changing nSample played a more significant role.