Week 2: Assignment

MScA, Statistical Analysis (31007)

Scott Shepard

4/4/2018

1 Generate uniformly distributed random numbers

1.1 Use runif()

Function runif(N,a,b) simulates N pseudo-random numbers uniformly distributed on [a,b].

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

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

```
library(random)

nFlips <- 1000

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

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

1.4 Turning binary sequence to uniform random numbers

Turn your sequence of {0,1} into uniform random numbers on [0,1].

Create function that turns a sequence of zeros and ones of length n into decimal form.

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

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

Turn the sequence ofzeros 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    1    1    1    1    1    0     1
## [2,]    0    1    0    0    1    1    1    1    1     1
## [3,]    1    1    0    1    0    0    1    0    1     0
## [4,]    0    1    1    1    1    1    0    0    1     1
## [5,]    1    1    1    1    1    1    0    0    1     1
## [6,]    0    0    1    0    1    1    0    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].

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

```
## [1] 0.9970703 0.3115234 0.8222656 0.4873047 0.9873047 0.1757812
```

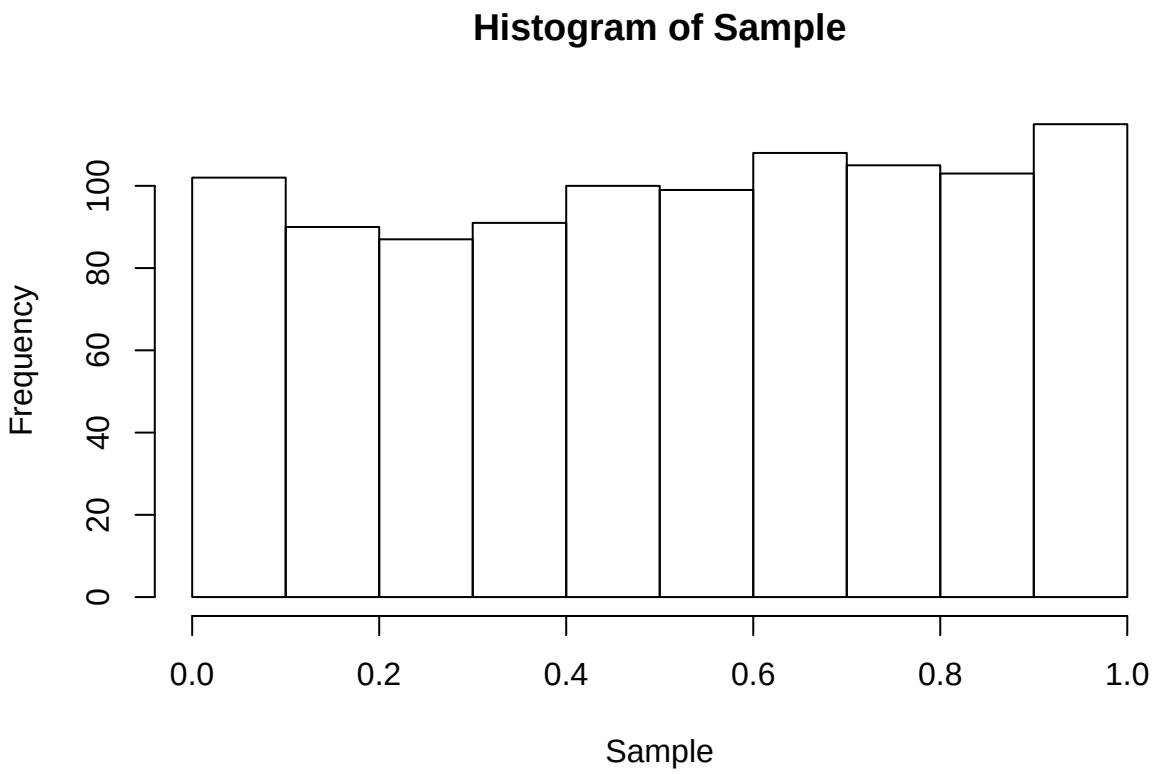
2 Test random number generators

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?

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

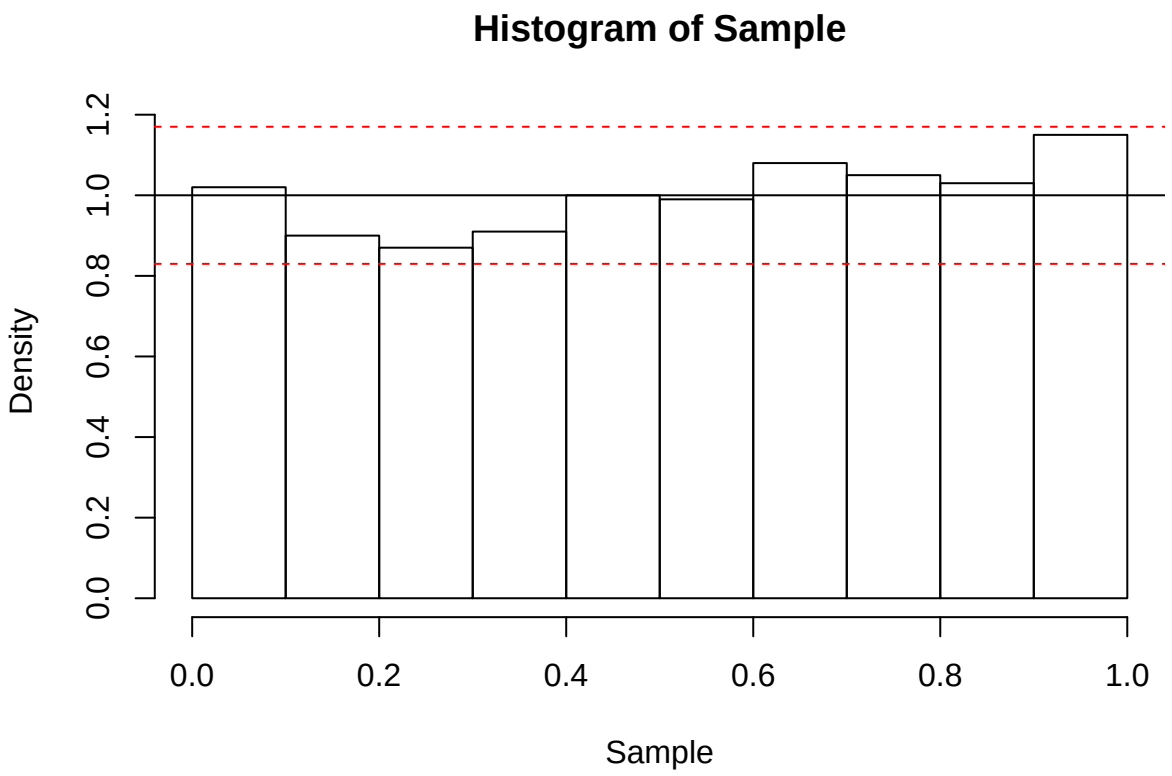
```
(Sample.histogram.mean <- mean(Sample.histogram$density))
```

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

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

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

```
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 you about the observed distribution?

Estimate moments of Sample.

```
(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?

Check the summary of the simulated sample.

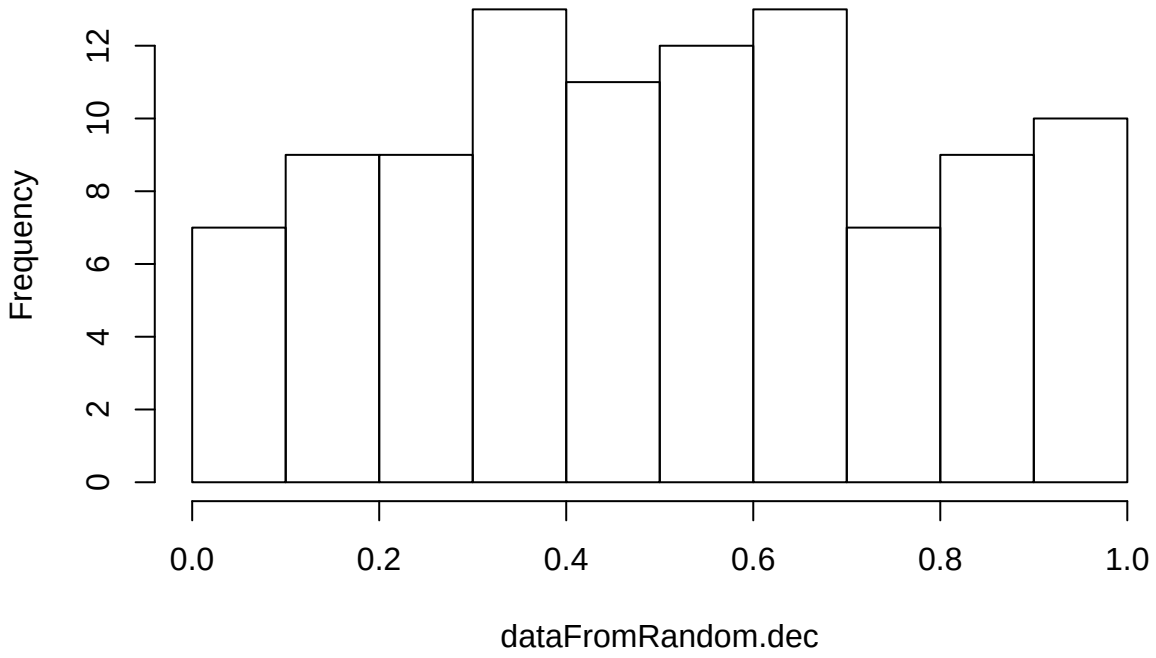
```
summary(Sample)
```

```
##       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?

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

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




```
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]  7  9  9 13 11 12 13  7  9 10
##
## $density
##  [1] 0.7 0.9 0.9 1.3 1.1 1.2 1.3 0.7 0.9 1.0
##
## $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"
```

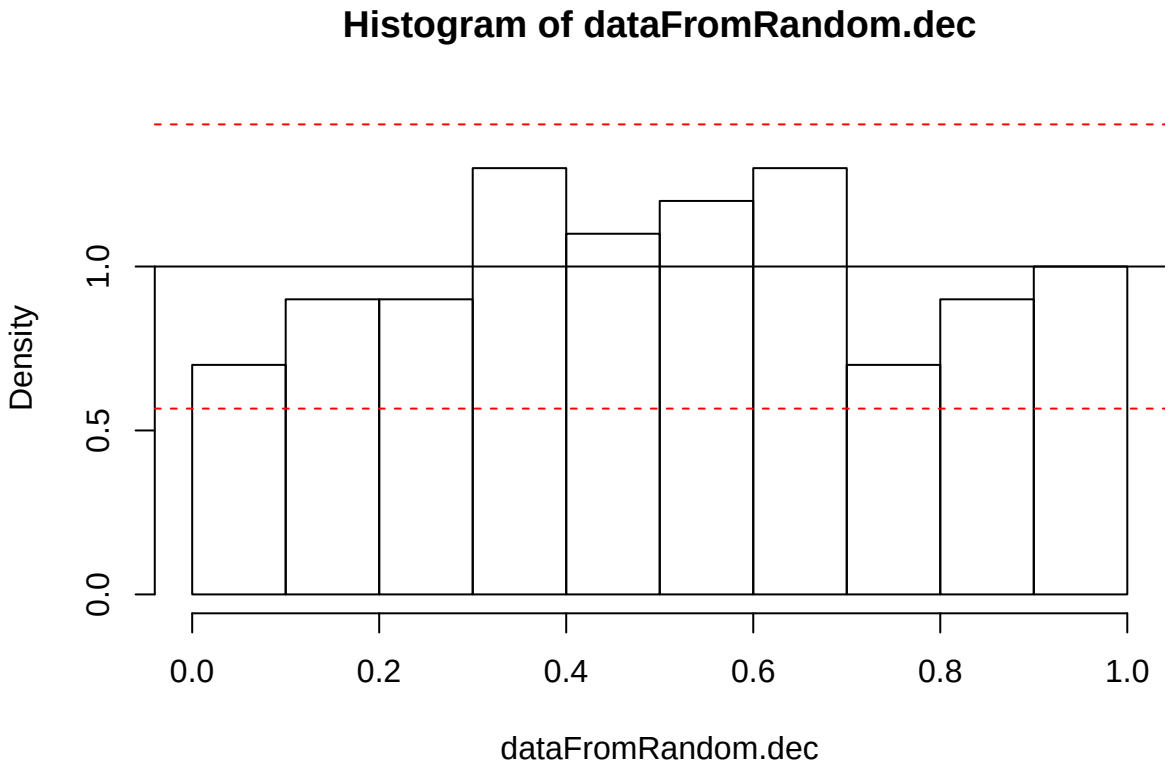
```
(Sample.histogram.mean<-mean(Sample.histogram$density))
```

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

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

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

```r
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)
```




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

```
## [1] 0.5125391
```

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

```
## [1] 0.07456875
```

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

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0009766 0.3046875 0.5043945 0.5125391 0.7128906 0.9970703
```

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 T be the number of turning points in a sample of length n large enough. Then the statistic of the test

$$z = \frac{T - \frac{2n-4}{3}}{\sqrt{\frac{16n-29}{90}}}$$

has standard normal distribution.

The test is performed by turning.point.test() in package randtests

```
library(randtests)
```

```
turning.point.test(dataFromRandom.dec)
```

```
##
##  Turning Point Test
##
## data:  dataFromRandom.dec
## statistic = -0.31913, n = 100, p-value = 0.7496
## 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.

The p-value here is quite high so the randomness cannot be rejected.

2.2.2 Test frequency by Monobit test

To perform Monobit test you need to transform your {0,1} sample into {-1,1}. Illustrate the test on the sequence simulated in the previous lecture.

We created the sequence of coin tosses:

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

Recall from the lecture notes that monobit test of randomness is based on the statistic

$$S = \frac{|\sum_{i=1}^{N} R_i|}{\sqrt{2N}} \sim erfc,$$

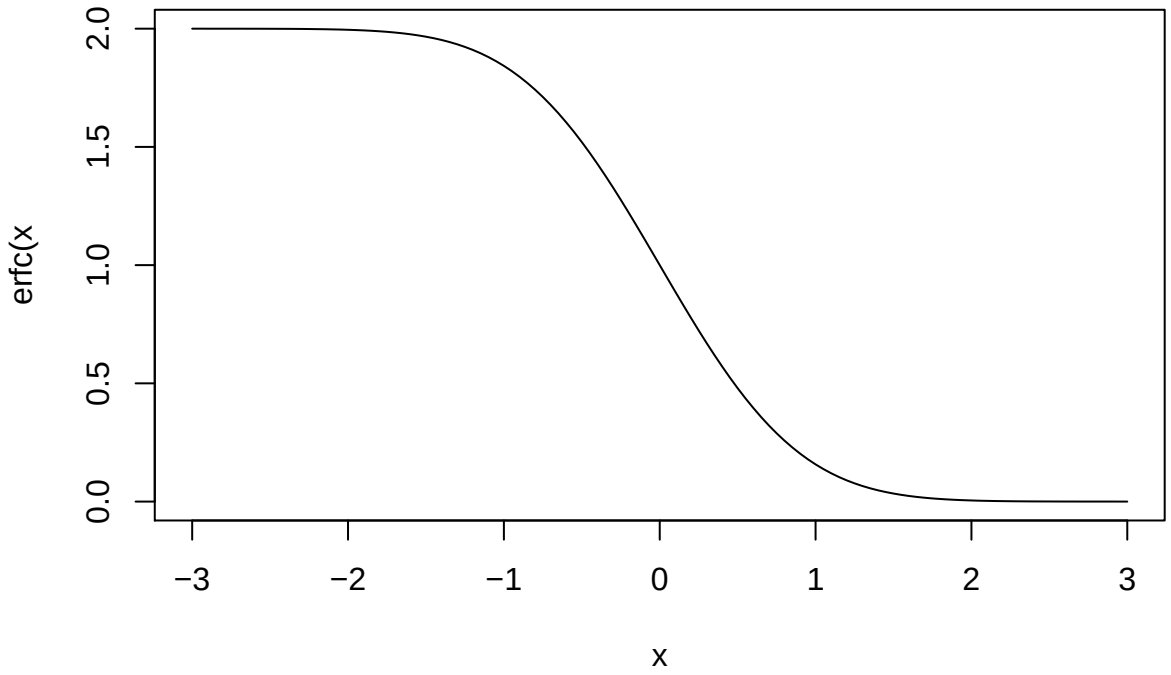
where R_i is the i-th random number, summation is done over all $N = nFlips$ random numbers.

erfc is the complimentary error function, a special function complimentary to error function erf=1-erfc.

Both functions can be easily calculated in R with the help of pnorm:

```
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 R_i check the value erfc(S).

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

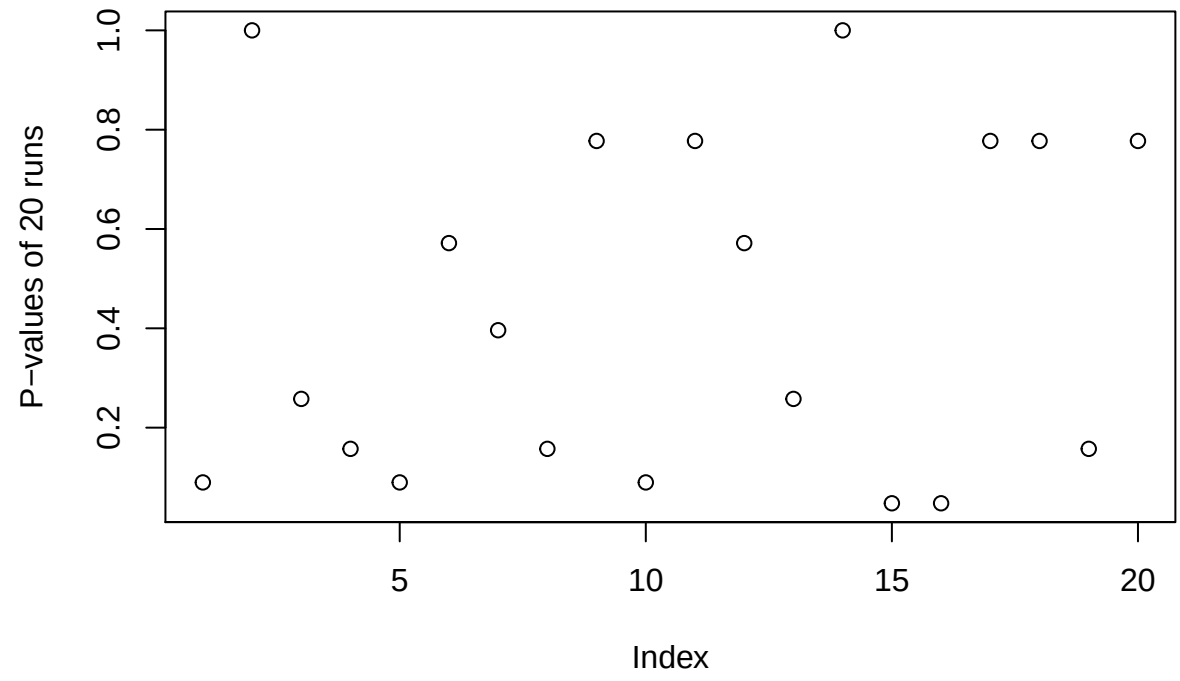
```
erfc(abs(sum(dataFromRandom.plusminus1)/sqrt(2*nFlips)))
```

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

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 r
abline(h=.01)
```



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
```

4 Monte Carlo Method

4.1 Scratch off quote of the day: fuction download

Download function `ScratchOffMonteCarlo()` contained in a binary file `ScratchOffMonteCarlo.rda` from the web site, put it in a folder with path

```
load("ScratchOffMonteCarlo.rda")
```

4.2 Simulate pseudo-random poins $[x, y]$ on $[0, 100] \times [0, 100]$