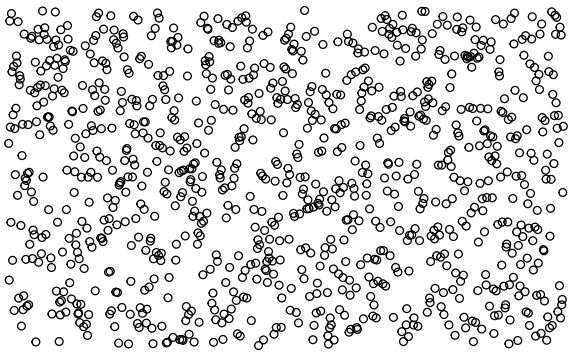
Random Numbers in R

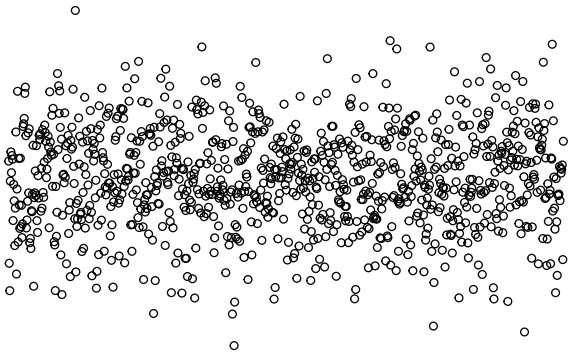
R's built-in pseudo-random-number generator is pretty solid. It works quickly and the results are practically useful. We are going to take a look at some of the effects of using different fundamental algorithms for the generation of seemingly random sequences. We will see that the most important considerations, generally speaking, are the period of the generation algorithm amd the magnitude of the sampling. A high period indicates that there is a high number of values generated by the algorithm before it begins to repeat.

You can view the documentation for RNG (the base R random number generator) by typing ?RNG into your R console. Much of the information from this blog post comes from the RNGkind documentation.

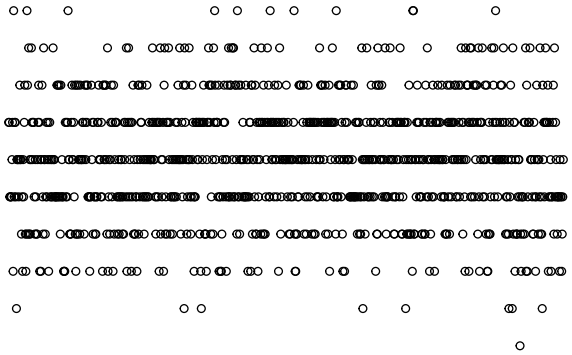
Default kind is "Mersenne-Twister". This algorithm has the highest available period of $2^{19937} - 1$ Other options "Wichman-Hill", "Marsaglia-Multicarry", "Super-Duper", "Knuth-TAOCP", "Knuth-TAOCP-2002", "L'Ecuyer-CMRG", "user-supplied". The shortest period available comes from "Marsaglia-Multicarry" at 2^{60}. It is interesting to note that this is the same order of magnitude as "Super-Duper" at 4.6×10^{18}.



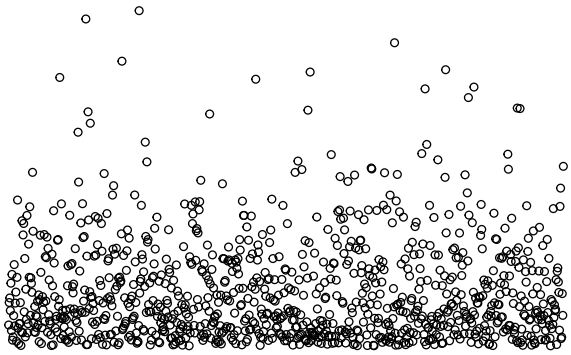
Uniform



Normal

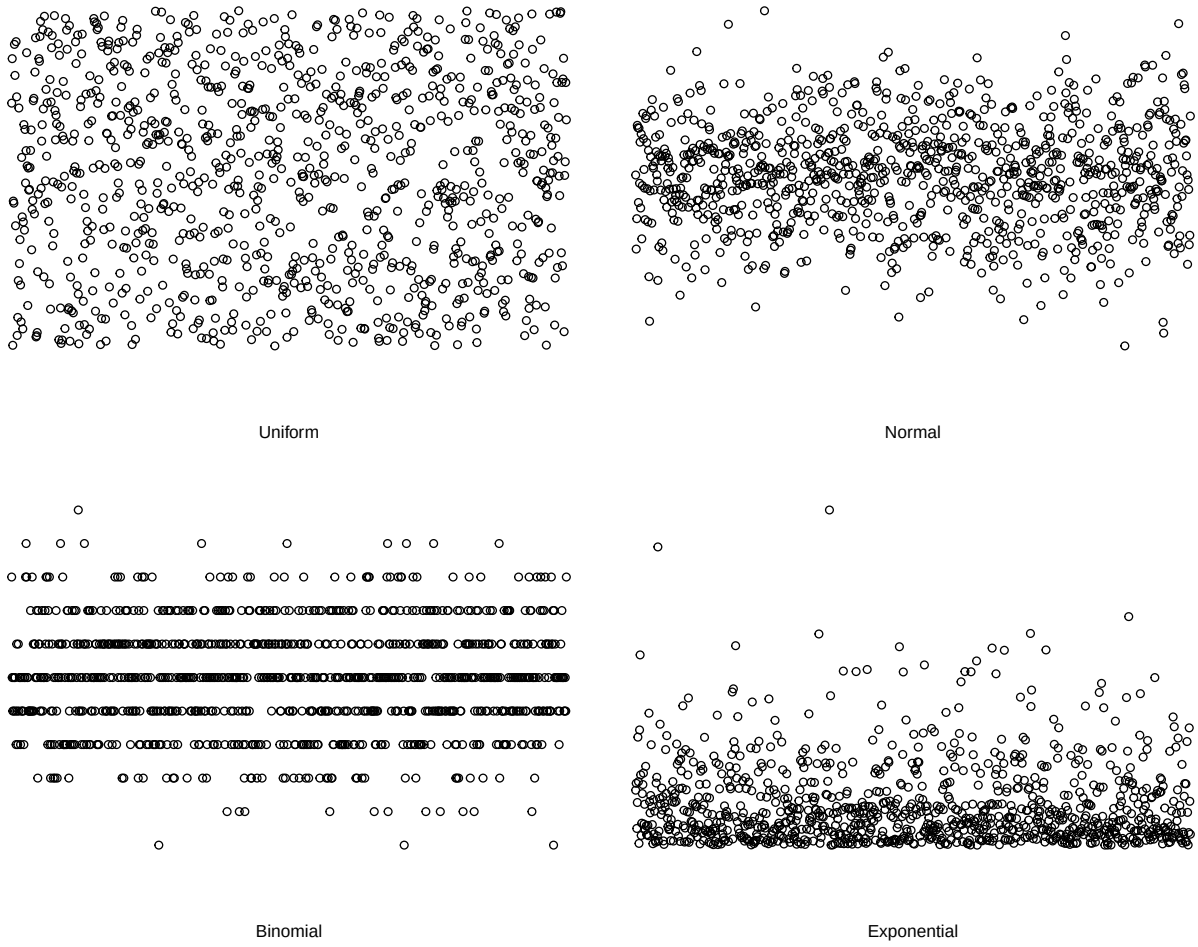


Binomial



Exponential

We try again with the simplest algorithm.



Visually, I think it is interesting to note that we can see the effects of these differences in period with only 1,000 generated values. There is more clustering and stranger outliers in the second set. If we go down to 500, the simpler algorithm does not pass the tests for randomness, and if we generate 10,000 datapoints, they are virtually indistinguishable from randomness.

Here we notice that the default number generator with the highest period pulls far ahead in terms of normal tests of randomness. Both pass the tests, but the Marsaglia-Multicarry method is noticeably weaker. It may be preferred only in scenarios where extreme amounts of computations pressure your time constraints.

```
##
##  Runs Test
##
## data:  data$un1
## statistic = -0.37966, runs = 495, n1 = 500, n2 = 500, n = 1000, p-value
## = 0.7042
## alternative hypothesis: nonrandomness

##
```

```
##  Runs Test
##
## data:  data$un2
## statistic = -1.2655, runs = 481, n1 = 500, n2 = 500, n = 1000, p-value
## = 0.2057
## alternative hypothesis: nonrandomness

##
##  Cox Stuart test
##
## data:  data$bi1
## statistic = 209, n = 421, p-value = 0.9224
## alternative hypothesis: non randomness

##
##  Cox Stuart test
##
## data:  data$bi2
## statistic = 194, n = 415, p-value = 0.2018
## alternative hypothesis: non randomness
```

I'll leave it to the reader to run this code, comb through the ANOVA information, and to assess the exponential and binomial data. The only package you will need to install is "randtests"

```
require(randtests)

generate.data <- function (kind = NULL, seed = 15) {
  # Should take a RNG kind and two-digit seed
  # Generates random data from 4 distributions
  # Outputs df

  RNGkind(kind = kind)
  set.seed(seed)

  un <- runif(1000, 0, 6)
  no <- rnorm(1000, 0, 1)
  bi <- rbinom(1000, 10, 0.5)
  ex <- rexp(1000, 0.5)

  return(data.frame(cbind(un, no, bi, ex)))
}

plot.data <- function(df, title) {
  # Takes a df from generate.data()
  # Plots the 4 samples

  par(mfrow = c(2,2),
    mai = c(1, 0.1, 0.1, 0.1))

  plot(df$un, axes = 0, xlab = '', ylab = '')
  plot(df$no, axes = 0, xlab = '', ylab = '')
  plot(df$bi, axes = 0, xlab = '', ylab = '')
  plot(df$ex, axes = 0, xlab = '', ylab = '')
```

```
}

data <- generate.data("Mersenne-Twister")
plot.data(data)

data <- cbind(data, generate.data(kind = "Marsaglia-Multicarry"), c(1:1000))
colnames(data) <- c("un1","no1","bi1","ex1","un2","no2","bi2","ex2","term")

models <- lapply(data[1:8], function(y){lm(y ~ term, data)})
anovas <- lapply(models, anova)

plot.data(data[5:8])

runs.test(data$un1)
runs.test(data$un2)

cox.stuart.test(data$bi1)
cox.stuart.test(data$bi2)
```