Chapter 37: Random Numbers Generator

Section 37.1: Random permutations

To generate random permutation of 5 numbers:

```
sample(5)
# [1] 4 5 3 1 2
```

To generate random permutation of any vector:

```
sample(10:15)
# [1] 11 15 12 10 14 13
```

One could also use the package `pracma`

```
randperm(a, k)
# Generates one random permutation of k of the elements a, if a is a vector,
# or of 1:a if a is a single integer.
# a: integer or numeric vector of some length n.
# k: integer, smaller as a or length(a).

# Examples
library(pracma)
randperm(1:10, 3)
[1] 3 7 9

randperm(10, 10)
[1]  4  5 10  8  2  7  6  9  3  1

randperm(seq(2, 10, by=2))
[1]  6  4 10  2  8
```

Section 37.2: Generating random numbers using various density functions

Below are examples of generating 5 random numbers using various probability distributions.

Uniform distribution between 0 and 10

```
runif(5, min=0, max=10)
[1] 2.1724399 8.9209930 6.1969249 9.3303321 2.4054102
```

Normal distribution with 0 mean and standard deviation of 1

```
rnorm(5, mean=0, sd=1)
[1] -0.97414402 -0.85722281 -0.08555494 -0.37444299  1.20032409
```

Binomial distribution with 10 trials and success probability of 0.5

```
rbinom(5, size=10, prob=0.5)
[1] 4 3 5 2 3
```

Geometric distribution with 0.2 success probability

```
rgeom(5, prob=0.2)
[1] 14  8 11  1  3
```

Hypergeometric distribution with 3 white balls, 10 black balls and 5 draws

```
rhyper(5, m=3, n=10, k=5)
[1] 2 0 1 1 1
```

Negative Binomial distribution with 10 trials and success probability of 0.8

```
rnbinom(5, size=10, prob=0.8)
[1] 3 1 3 4 2
```

Poisson distribution with mean and variance (lambda) of 2

```
rpois(5, lambda=2)
[1] 2 1 2 3 4
```

Exponential distribution with the rate of 1.5

```
rexp(5, rate=1.5)
[1] 1.8993303 0.4799358 0.5578280 1.5630711 0.6228000
```

Logistic distribution with 0 location and scale of 1

```
rlogis(5, location=0, scale=1)
[1]  0.9498992 -1.0287433 -0.4192311  0.7028510 -1.2095458
```

Chi-squared distribution with 15 degrees of freedom

```
rchisq(5, df=15)
[1] 14.89209 19.36947 10.27745 19.48376 23.32898
```

Beta distribution with shape parameters a=1 and b=0.5

```
rbeta(5, shape1=1, shape2=0.5)
[1] 0.1670306 0.5321586 0.9869520 0.9548993 0.9999737
```

Gamma distribution with shape parameter of 3 and scale=0.5

```
rgamma(5, shape=3, scale=0.5)
[1] 2.2445984 0.7934152 3.2366673 2.2897537 0.8573059
```

Cauchy distribution with 0 location and scale of 1

```
rcauchy(5, location=0, scale=1)
[1] -0.01285116 -0.38918446  8.71016696 10.60293284 -0.68017185
```

Log-normal distribution with 0 mean and standard deviation of 1 (on log scale)

```
rlnorm(5, meanlog=0, sdlog=1)
[1] 0.8725009 2.9433779 0.3329107 2.5976206 2.8171894
```

Weibull distribution with shape parameter of 0.5 and scale of 1

```
rweibull(5, shape=0.5, scale=1)
[1] 0.337599112 1.307774557 7.233985075 5.840429942 0.005751181
```

Wilcoxon distribution with 10 observations in the first sample and 20 in second.

```
rwilcox(5, 10, 20)
[1] 111  88  93 100 124
```

Multinomial distribution with 5 object and 3 boxes using the specified probabilities

```
rmultinom(5, size=5, prob=c(0.1,0.1,0.8))
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    1    1    0
[2,]    2    0    1    1    0
[3,]    3    5    3    3    5
```

Section 37.3: Random number generator's reproducibility

When expecting someone to reproduce an R code that has random elements in it, the `set.seed()` function becomes very handy. For example, these two lines will always produce different output (because that is the whole point of random number generators):

```
> sample(1:10,5)
[1]  6  9  2  7 10
> sample(1:10,5)
[1]  7  6  1  2 10
```

These two will also produce different outputs:

```
> rnorm(5)
[1]  0.4874291  0.7383247  0.5757814 -0.3053884  1.5117812
> rnorm(5)
[1]  0.38984324 -0.62124058 -2.21469989  1.12493092 -0.04493361
```

However, if we set the seed to something identical in both cases (most people use 1 for simplicity), we get two identical samples:

```
> set.seed(1)
> sample(letters,2)
[1] "g" "j"
> set.seed(1)
> sample(letters,2)
[1] "g" "j"
```

and same with, say, `rexp()` draws:

```
> set.seed(1)
> rexp(5)
[1] 0.7551818 1.1816428 0.1457067 0.1397953 0.4360686
> set.seed(1)
> rexp(5)
[1] 0.7551818 1.1816428 0.1457067 0.1397953 0.4360686
```