Simulating data in R

stirlingcodingclub.github.io/simulating_data

Stirling Coding Club

14 December 2022

Why simulate data?

Simulating data uses generating random data sets with known properties using code (or some other method). This can be useful in a lot of contexts.

- Better understand statistical methods
- Plan ahead for actual data collection
- Visualise data sets and distributions
- Run some statistical analyses (randomisation)

How can data be simulated in R?

Random data with different properties can be generated in R using several base R functions.

- ‘runif‘ generates random values from a uniform distribution.

How can data be simulated in R?

Random data with different properties can be generated in R using several base R functions.

- ‘runif‘ generates random values from a uniform distribution.
- ‘rnorm‘ generates random values from a normal distribution.

How can data be simulated in R?

Random data with different properties can be generated in R using several base R functions.

- ‘runif‘ generates random values from a uniform distribution.
- ‘rnorm‘ generates random values from a normal distribution.
- ‘rpois‘ generates random values from a poisson distribution.

How can data be simulated in R?

Random data with different properties can be generated in R using several base R functions.

- ‘runif‘ generates random values from a uniform distribution.
- ‘rnorm‘ generates random values from a normal distribution.
- ‘rpois‘ generates random values from a poisson distribution.
- ‘rbinom‘ generates random values from a binomial distribution.

How can data be simulated in R?

Random data with different properties can be generated in R using several base R functions.

- ‘runif‘ generates random values from a uniform distribution.
- ‘rnorm‘ generates random values from a normal distribution.
- ‘rpois‘ generates random values from a poisson distribution.
- ‘rbinom‘ generates random values from a binomial distribution.
- ‘sample‘ samples values from any given vector with or without replacement.

Other R packages, such as the MASS library, can simulate full data sets with pre-defined correlation stuctures.

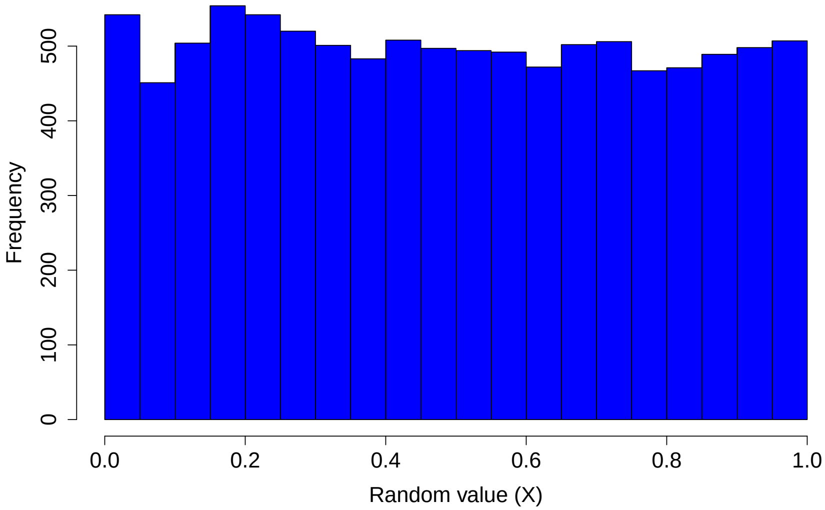
The runif function in R

```r
rand_unifs <- runif(n = 10000, min = 0, max = 1);
```

```
##  [1] 0.8670857768 0.5615415513 0.6288375359 0.7923040655
##  [6] 0.1095961148 0.2355039727 0.8978011261 0.6879017937
## [11] 0.5624286113 0.9375209641 0.3825901572 0.5472148873
## [16] 0.7698536073 0.4487606222 0.8535867573 0.7288415928
## [21] 0.0934886276 0.0524177891 0.6597115695 0.1274242669
## [26] 0.7071376585 0.7218166539 0.9343405771 0.0041327698
## [31] 0.8715531109 0.1296939508 0.3870098493 0.0009932041
## [36] 0.6261875923 0.5429373912 0.1288557602 0.7124097436
```

The `runif` function in R

```
rand_unifs <- runif(n = 10000, min = 0, max = 1);
```



The runif function in R

```
int verify_seed(int x){
  x=abs(x) % 30000;
  return(++x);
 } /* Easy way of getting seeds */

double as183(int seeds[]){
  double unidev; /* Code below verifies the 3 seeds */
  seeds[0] = verify_seed(seeds[0]);
  seeds[1] = verify_seed(seeds[1]);
  seeds[2] = verify_seed(seeds[2]);
  /* Code below gets a decimal to be added to unidev */
  seeds[0] = (171 * seeds[0]) % 30269;
  seeds[1] = (172 * seeds[1]) % 30307;
  seeds[2] = (170 * seeds[2]) % 30323;
  /* unidev gets a random uniform number between zero and one */
  unidev = seeds[0]/30269.0 + seeds[1]/30307.0 + seeds[2]/30323.
  /* Return just the decimal, subtract integer of unidev */
  return(unidev - (int)unidev);
} /* We now have one random uniform number */
```

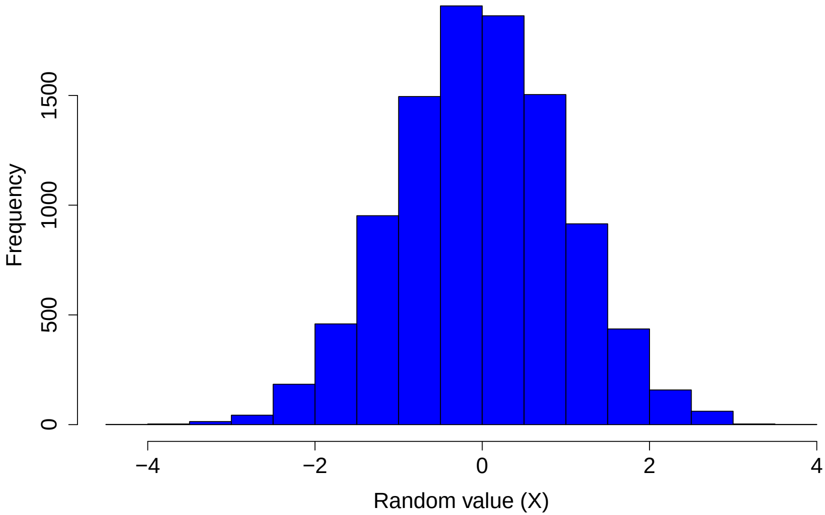
The rnorm function in R

```
rand_rnorms <- rnorm(n = 10000, mean = 0, sd = 1);
```

```
##  [1] -0.238818449 -0.410411347  0.277909056  0.305590858
##  [6] -0.492572290 -0.397795630  0.135767266  0.312741598
## [11]  0.247555343 -0.565200403  0.486173016  0.835464508
## [16] -0.282386781 -0.490972178 -0.415299952  0.303614862
## [21]  0.533060260 -0.866063628  0.153560803 -0.122009959
## [26] -0.024995795  1.538472460  0.909788815 -2.154001653
## [31] -0.441665950 -0.681505811  0.746111719  1.192287230
## [36]  0.286682273  0.518682453  0.871770365  0.961401125
```

The rnorm function in R

```
rand_rnorms <- rnorm(n = 10000, mean = 0, sd = 1);
```



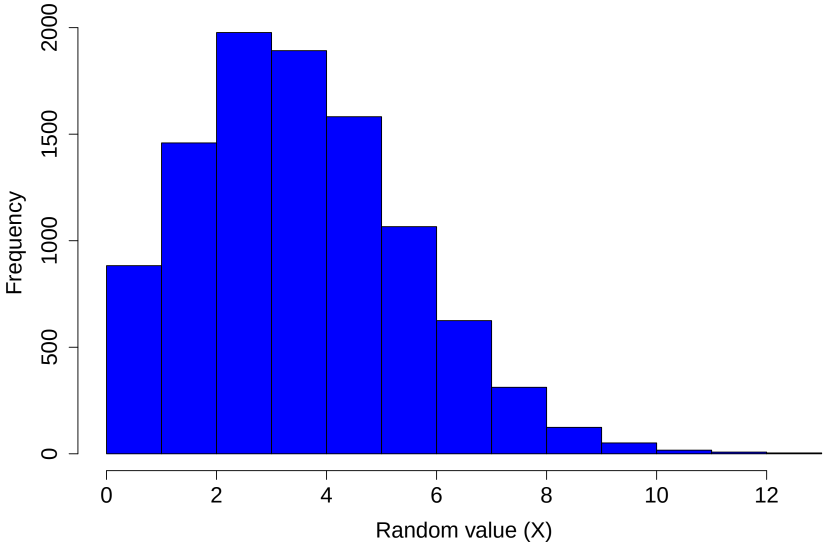
The rpois function in R

```r
rand_rpois <- rpois(n = 10000, lambda = 4);
```

```
##  [1]  3  6  3 10  5  4  3  3  5  7  4  4  3  7  4  3  5
## [26]  7  1  4  4  4  7  3  2  3  1  4  3  5  2  1
```

The rpois function in R

```
rand_rpois <- rpois(n = 10000, lambda = 4);
```



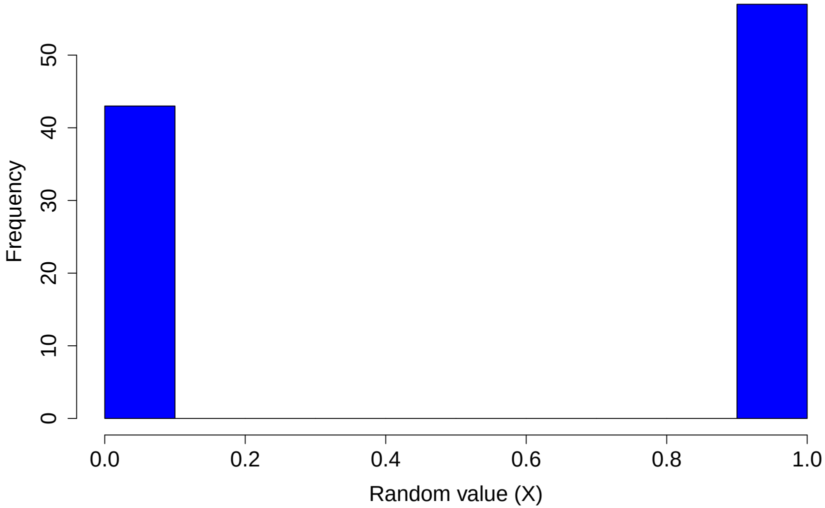
The rbinom function in R

```r
rand_rbinom <- rbinom(n = 100, size = 1, prob = 0.5);
```

```
##  [1] 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 1 0 0 1 1 0 0 1 1 1
## [39] 0 1
```

The rpois function in R

```
rand_rbinom <- rbinom(n = 100, size = 1, prob = 0.5);
```



Using sample in R

Create a vector of numbers from which to sample.

```r
my_sample_vec <- 1:10;
```

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

Using `sample` in R

Create a vector of numbers from which to sample.

```r
my_sample_vec <- 1:10;
```

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

Use `sample` to randomly sample numbers from `my_sample_vec`

```r
my_sample <- sample(x = my_sample_vec, size = 4);
```

```
## [1]  7 10  4  2
```

Using `sample` in R

Can sample with or without replacement.

```r
sample_no_replace <- sample(x = my_sample_vec,
                            size = 10, replace = FALSE);
```

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

Using sample in R

Can sample with or without replacement.

```
sample_no_replace <- sample(x = my_sample_vec,
                            size = 10, replace = FALSE);
```

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

```
sample_replace <- sample(x = my_sample_vec,
                         size = 10, replace = TRUE);
```

```
##  [1] 4 2 9 2 3 8 5 9 9 6
```

Using `sample` in R

Can also change the probabilities of being sampled

```
# Vector values must sum to 1
pr_vector  <- c(0, 0, 0, 0, 0,
                0.2, 0.2, 0.2,
                0.2, 0.2);
new_sample <- sample(x = 1:10, size = 10,
                     replace = TRUE,
                     prob = pr_vector);
```

```
##  [1]  9 10  9  6 10  7 10  8  7  7
```

Using `sample` in R

Can also sample strings instead of numbers

```
species   <- c("species_A", "species_B", "species_C");
sp_sample <- sample(x = species, size = 12,
                    replace = TRUE,
                    prob = c(0.5, 0.25, 0.25)
                    );
```

```
##  [1] "species_A" "species_A" "species_C" "species_C" "sp
##  [7] "species_C" "species_C" "species_B" "species_A" "sp
```

Building a simple simulated dataset

```r
N          <- 12;
species    <- c("species_A", "species_B");
sp_sample <- sample(x = species,
                    size = N, replace = TRUE);
sp_mass    <- rnorm(n = N, mean = 100, sd = 4);
for(i in 1:N){
  if(sp_sample[i] == "species_A"){
    sp_mass[i] <- sp_mass[i] + rnorm(n = 1,
                                     mean = 2, sd = 1);
  }
}
sim_data  <- data.frame(sp_sample, sp_mass);
```

Building a simple simulated dataset

sp_sample	sp_mass
species_B	102.12150
species_B	103.63621
species_B	100.80428
species_A	97.01569
species_A	104.27227
species_A	104.20936
species_A	96.13937
species_A	99.41683
species_B	106.52525
species_A	103.54237
species_A	99.59500
species_B	105.38609

Building a simple simulated dataset

```
t.test(sp_mass ~ sp_sample, data = sim_data);
```

```
##
##  Welch Two Sample t-test
##
## data:  sp_mass by sp_sample
## t = -1.8637, df = 9.9981, p-value = 0.09197
## alternative hypothesis: true difference in means between
## 95 percent confidence interval:
##  -6.7975205  0.6055819
## sample estimates:
## mean in group species_A mean in group species_B
##                100.5987                103.6947
```

Building a simple simulated dataset

```r
N          <- 120;
species    <- c("species_A", "species_B");
sp_sample <- sample(x = species, size = N,
                    replace = TRUE);
sp_mass    <- rnorm(n = N, mean = 100, sd = 4);
for(i in 1:N){
  if(sp_sample[i] == "species_A"){
    sp_mass[i] <- sp_mass[i] + rnorm(n = 1,
                                     mean = 2, sd = 1);
  }
}
sim_data  <- data.frame(sp_sample, sp_mass);
```

Building a simple simulated dataset

```r
t.test(sp_mass ~ sp_sample, data = sim_data);
```

```
##
##  Welch Two Sample t-test
##
## data:  sp_mass by sp_sample
## t = 3.5696, df = 117.23, p-value = 0.0005193
## alternative hypothesis: true difference in means between
## 95 percent confidence interval:
##  1.09329 3.81812
## sample estimates:
## mean in group species_A mean in group species_B
##                102.25269                99.79699
```

Setting a seed gives the same numbers

Try it once

```
set.seed(10);
rnorm(n = 10);
```

```
##  [1]  0.01874617 -0.18425254 -1.37133055 -0.59916772  0.
##  [7] -1.20807618 -0.36367602 -1.62667268 -0.25647839
```

Try it again

```
set.seed(10);
rnorm(n = 10);
```

```
##  [1]  0.01874617 -0.18425254 -1.37133055 -0.59916772  0.
##  [7] -1.20807618 -0.36367602 -1.62667268 -0.25647839
```