Creating simulated data sets in R

Brad Duthie

8 July 2020

Contents

The ability to simulate data is a useful tool for better understanding statistical analyses and planning experimental designs. These notes illustrate how to simulate data using a variety of different functions in the R programming language, then discuss how data simulation can be used in research. These notes borrow heavily from a Stirling Coding Club session on randomisation, and to a lesser extent from a session on linear models. After working through these notes, the reader should be able to simulate their own data sets and use them to explore data visualisations and statistical analysis. These notes are also available as PDF and DOCX documents.

Introduction: Simulating data

Univariate random numbers

There are several functions in R that generate random numbers from particular distributions, or sample elements from vectors or lists. Below, I introduce some of these functions and show how to use them.

Sampling from a normal distribution

The `rnorm` function returns some number (`n`) of (pseudo)randomly generated numbers given a set mean (μ; `mean`) and standard deviation (σ; `sd`), such that $X \sim \mathcal{N}(\mu, \sigma^2)$. The default is to draw from a standard normal (a.k.a., "Gaussian") distribution (i.e., $\mu = 0$ and $\sigma = 1$).

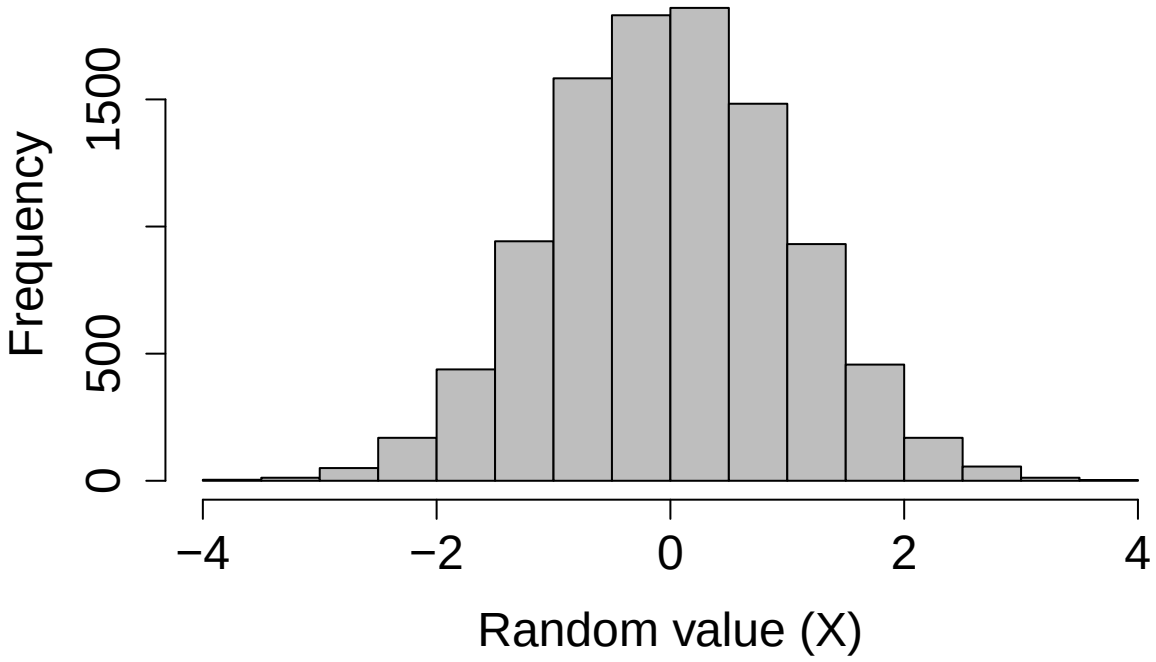
```
rand_norms_10 <- rnorm(n = 10, mean = 0, sd = 1);
```

The above code stores a vector of ten numbers, shown below.

```
##  [1]  1.2482611 -0.7385943  0.3577943  0.6399430  0.1834261 -1.3612220
##  [7] -0.4100112  1.3191546  0.4780209  1.7010319
```

We can verify that a standard normal distribution is generated by plotting a histogram of a very large number of values created using `rnorm`.

```
rand_norms_10000 <- rnorm(n = 10000, mean = 0, sd = 1);
hist(rand_norms_10000, xlab = "Random value (X)", col = "grey",
     main = "", cex.lab = 1.5, cex.axis = 1.5);
```



Sampling from a uniform distribution

Like the `rnorm` function, the `runif` function returns some number (`n`) of random numbers, but from a uniform distribution with a range from a (`min`) to b (`max`) such that $X \sim \mathcal{U}(a, b)$, where $-\infty < a < b < \infty$. The default is to draw from a standard uniform distribution (i.e., $a = 0$ and $b = 1$).

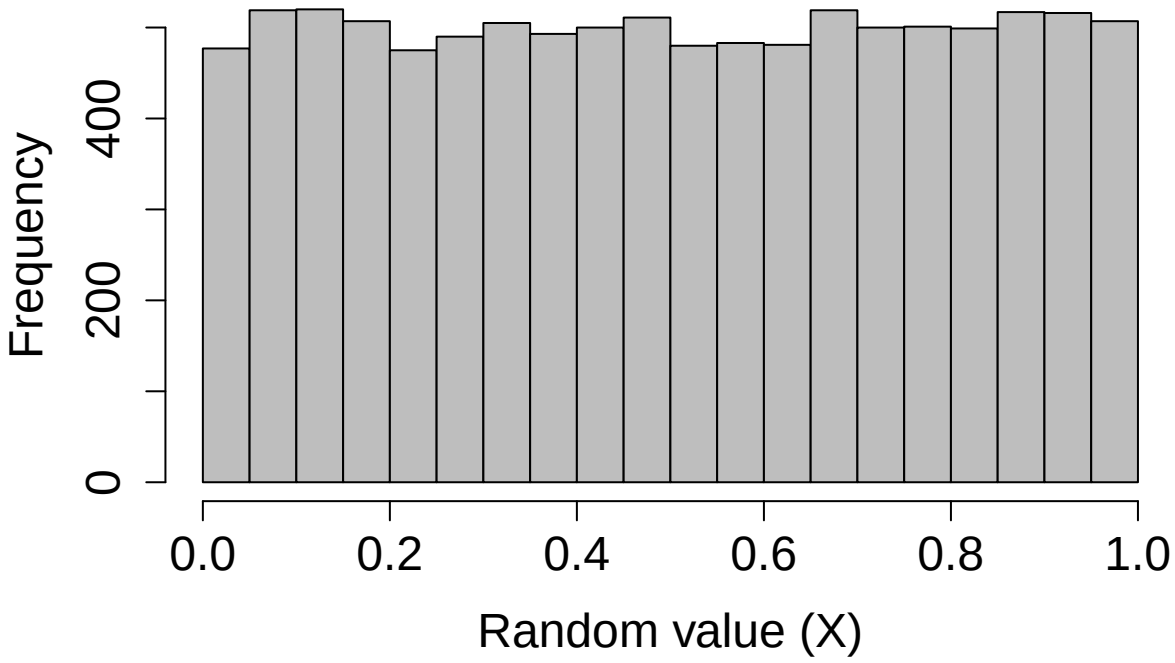
```
rand_unifs_10 <- runif(n = 10, min = 0, max = 1);
```

The above code stores a vector of ten numbers, shown below.

```
##  [1] 0.9414265 0.9932959 0.7405133 0.7974370 0.1340870 0.9551097 0.5676347
##  [8] 0.7153564 0.2659642 0.1398133
```

As with the randomly generated normally distributed numbers, we can verify that a standard uniform distribution is generated by plotting a histogram of a very large number of values created using `runif`.

```
rand_unifs_10000 <- runif(n = 10000, min = 0, max = 1);
hist(rand_unifs_10000, xlab = "Random value (X)", col = "grey",
     main = "", cex.lab = 1.5, cex.axis = 1.5);
```



Sampling from a binomial distribution

Like previous functions, the `rbinom` function returns some number (`n`) of random numbers, but the arguments and output can be slightly confusing at first. Recall that a binomial distribution describes the number of 'successes' for some number of independent trials ($\Pr(success) = p$). The `rbinom` function returns the number of successes after `size` trials, in which the probability of success in each trial is `prob`. For a concrete example, suppose we want to simulate the flipping of a fair coin 1000 times, and we want to know how many times that coin comes up heads ('success'). We can do this with the following code.

```
coin_flips <- rbinom(n = 1, size = 1000, prob = 0.5);
print(coin_flips);
```

```
## [1] 526
```

The above result shows that the coin came up heads 526 times. Note, however, the (required) argument `n` above. This allows the user to set the number of sequences to run. In other words, if we set `n = 2`, then this could simulate the flipping of a fair coin 1000 times once to see how many times heads comes up, then repeating the whole process to see how many times heads comes up again (or, if it is more intuitive, the flipping of two separate fair coins 1000 times).

```
coin_flips_2 <- rbinom(n = 2, size = 1000, prob = 0.5);
print(coin_flips_2);
```

```
## [1] 511 527
```

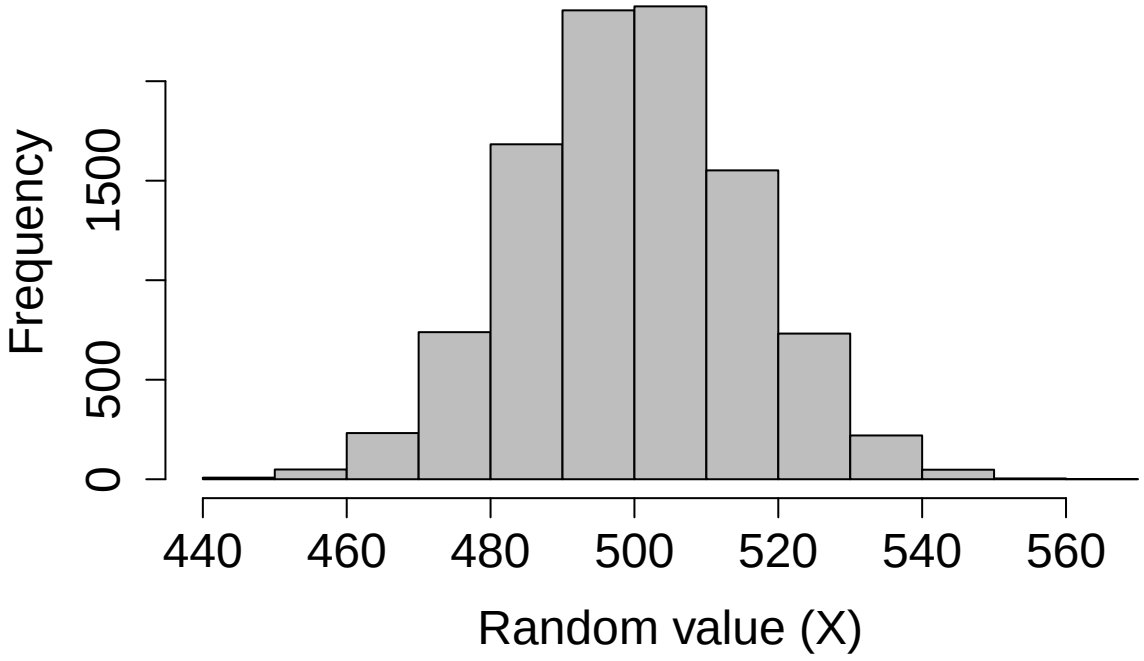
In the above, a fair coin was flipped 1000 times and returned 511 heads, and then another fair coin was flipped 1000 times and returned 527 heads. As with the `rnorm` and `runif` functions, we can check to see what the distribution of the binomial function looks like if we repeat this process. Suppose, in other words, that we want to see the distribution of times heads comes up after 1000 flips. We can, for example, simulate

the process of flipping 1000 times in a row with 10000 different coins using the code below.

```
coin_flips_10000 <- rbinom(n = 10000, size = 1000, prob = 0.5);
```

I have not printed the above `coin_flips_10000` for obvious reasons, but we can use a histogram to look at the results.

```
hist(coin_flips_10000, xlab = "Random value (X)", col = "grey",
     main = "", cex.lab = 1.5, cex.axis = 1.5);
```



As would be expected, most of the time 'heads' occurs around 500 times out of 1000, but usually the actual number will be a bit lower or higher due to chance. Note that if we want to simulate the results of individual flips in a single trial, we can do so as follows.

```
flips_10 <- rbinom(n = 10, size = 1, prob = 0.5);
```

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

In the above, there are `n = 10` trials, but each trial consists of only a single coin flip (`size = 1`). But we can equally well interpret the results as a series of `n` coin flips that come up either heads (`1`) or tails (`0`). This latter interpretation can be especially useful to write code that randomly decides whether some event will happen (`1`) or not (`0`) with some probability `prob`.

Sampling from a poisson distribution

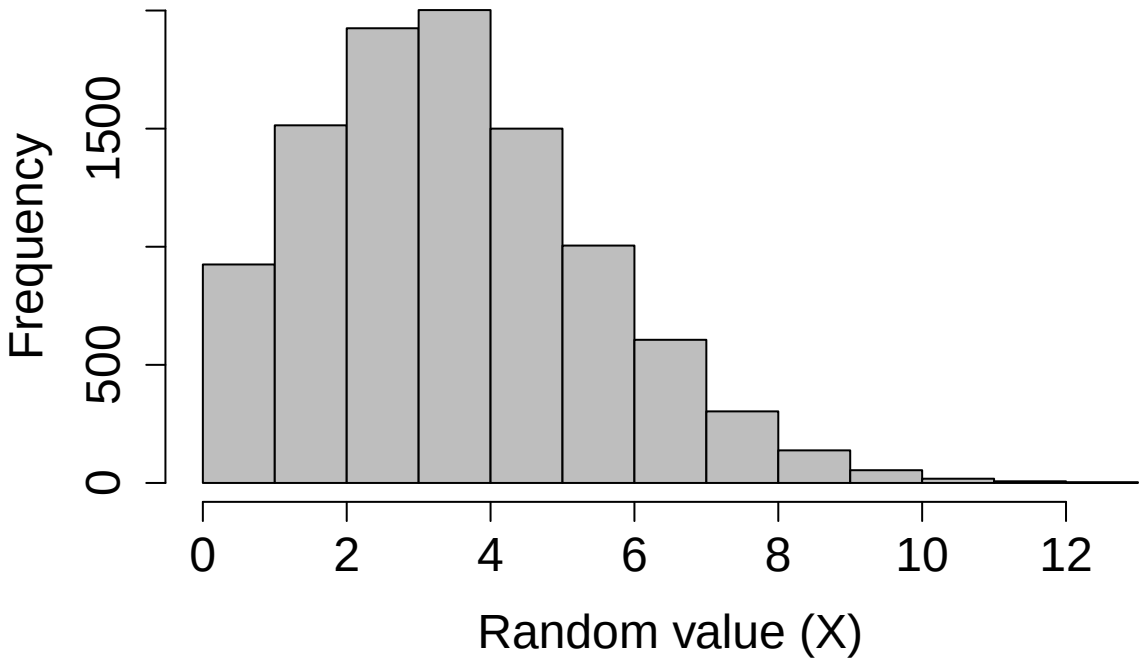
Many processes in biology can be described by a Poisson distribution. A Poisson process describes events happening with some given probability over an area of time or space such that $X \sim Poisson(\lambda)$, where the rate parameter λ is both the mean and variance of the Poisson distribution (note that by definition, $\lambda > 0$). Sampling from a Poisson distribution can be done in R with `rpois`, which takes only two arguments specifying the number of values to be returned (`n`) and the rate parameter (`lambda`).

```
rand_poissons <- rpois(n = 10, lambda = 1);
print(rand_poissons);
```

```
##  [1] 2 0 1 0 0 1 1 2 0 0
```

There are no default values for `rpois`. We can plot a histogram of a large number of values to see the distribution when $\lambda = 4$ below.

```
rand_poissons_10000 <- rpois(n = 10000, lambda = 4);
hist(rand_poissons_10000, xlab = "Random value (X)", col = "grey",
     main = "", cex.lab = 1.5, cex.axis = 1.5);
```



Random sampling using `sample`

Sometimes it is useful to sample a set of values from a vector or list. The R function `sample` is very flexible for sampling a subset of numbers or elements from some structure (`x`) in R according to some set probabilities (`prob`). Elements can be sampled from `x` some number of times (`size`) with or without replacement (`replace`), though an error will be returned if the `size` of the sample is larger than `x` but `replace = FALSE` (default). To start out simple, suppose we want to ask R to pick a random number from one to ten with equal probability.

```
rand_number_1 <- sample(x = 1:10, size = 1);
```

The above code will set `rand_number_1` to a randomly selected value, in this case 2. Because we have not specified a probability vector `prob`, the function assumes that every element in `1:10` is sampled with equal probability. We can increase the `size` of the sample to `10` below.

```
rand_number_10 <- sample(x = 1:10, size = 10);
print(rand_number_10);
```

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

Note that all numbers from 1 to 10 have been sampled, but in a random order. This is becaues the default is to sample with replacement, meaning that once a number has been sampled for the first element in `rand_number_10`, it is no longer available to be sampled again. To change this and allow for sampling with replacement, we can change the default.

```
rand_number_10_r <- sample(x = 1:10, size = 10, replace = TRUE);
print(rand_number_10_r);
```

```
##  [1]  9  6 10  8  6  6  5  8  6 10
```

Note that the numbers {6, 8, 10} are now repeated in the set of randomly sampled values above. We can also specify the probability of sampling each element, with the condition that these probabilities need to sum to 1. Below shows an example in which the numbers 1-5 are sampled with a probability of 0.05, while the numbers 6-10 are sampled with a probability of 0.15, thereby biasing sampling toward larger numbers.

```
prob_vec      <- c( rep(x = 0.05, times = 5), rep(x = 0.15, times = 5) );
rand_num_bias <- sample(x = 1:10, size = 10, replace = TRUE, prob = prob_vec);
print(rand_num_bias);
```

```
##  [1]  5  9  8  7  3  7  8 10  8  9
```

Note that `rand_num_bias` above contains more numbers from 6-10 than from 1-5.

Simulating data with known correlations

Simulating a full data set

Conclusions