Creating simulated data sets in R

Brad Duthie

12 January 2022

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 a HTML.

Introduction: Simulating data

The ability generate simulated data is very useful in a lot of research contexts. Simulated data can be used to better understand statistical methods, or in some cases to actually run statistical analyses (e.g., simulating a null distribution against which to compare a sample). Here I want to demonstrate how to simulate data in R. This can be accomplished with base R functions including `rnorm`, `runif`, `rbinom`, `rpois`, or `rgamma`; all of these functions sample univariate data (i.e., one variable) from a specified distribution. The function `sample` can be used to sample elements from an R object with or without replacement. Using the MASS library, the `mvtnorm` function will sample multiple variables with a known correlation structure (i.e., we can tell R how variables should be correlated with one another) and normally distributed errors.

Below, I will first demonstrate how to use some common functions in R for simulating data. Then, I will illustrate how these simulated data might be used to better understand common statistical analyses and data visualisation.

Univariate random numbers

Below, I introduce some base R functions that simulate (pseudo)random numbers from a given distribution. Note that most of what follows in this section is a recreation of a similar section in the notes for randomisation analysis in R.

Sampling from a uniform distribution

The `runif` function returns some number (`n`) of random numbers from a uniform distribution with a range from a (`min`) to b (`max`) such that $X \sim \mathcal{U}(a, b)$ (verbally, X is sampled from a uniform distribution with the parameters a and b), where $-\infty < a < b < \infty$ (verbally, a is greater than negative infinity but less than b, and b is finite). The default is to draw from a standard uniform distribution (i.e., $a = 0$ and $b = 1$) as done below.

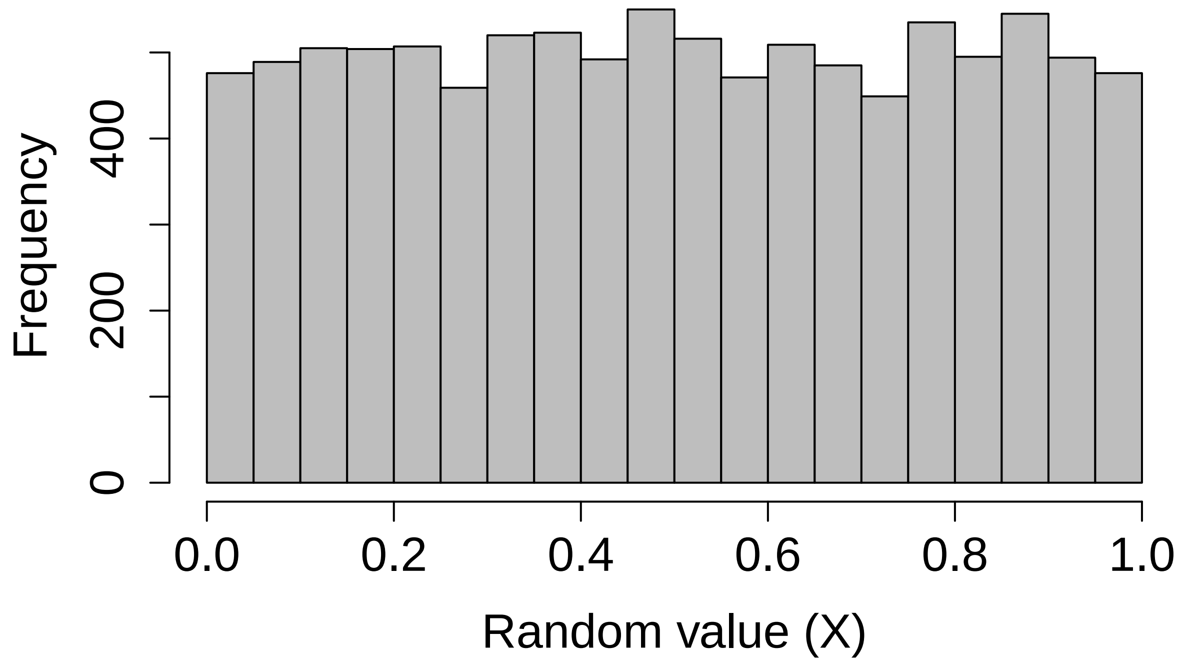
```
rand_unifs_10 <- runif(n = 10, min = 0, max = 1);
```

The above code stores a vector of ten numbers `rand_unifs_10`, shown below. Note that the numbers will be different each time we re-run the `runif` function above.

```
##  [1] 0.9411607 0.6163376 0.5848837 0.5739287 0.1306308 0.5030655 0.2729957
##  [8] 0.6870127 0.3756420 0.2735641
```

We can visualise the standard uniform distribution that 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);
```



The random uniform distribution is special in some ways. The algorithm for generating random uniform numbers is the starting point for generating random numbers from other distributions using methods such as rejection sampling, inverse transform sampling, or the Box Muller method (Box and Muller 1958).

Sampling from a normal distribution

The `rnorm` function returns some number (`n`) of randomly generated values 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 $\sigma = 1$).

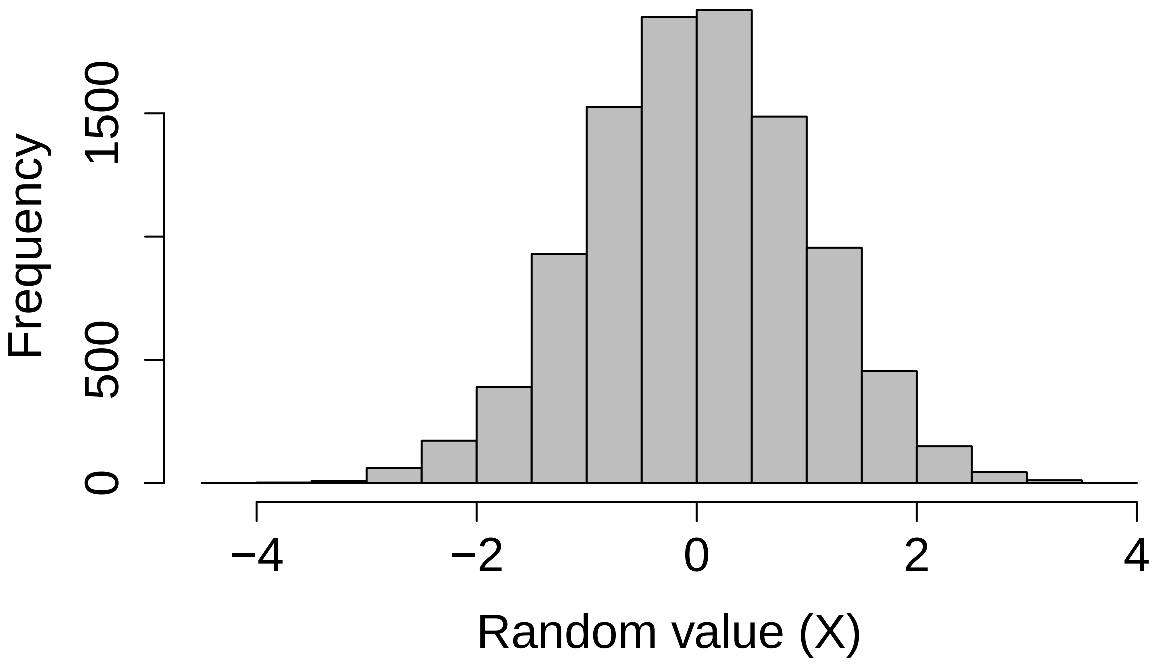
```
rand_norms_10 <- rnorm(n = 10, mean = 0, sd = 1);
```

The above code stores a vector of 10 numbers, shown below.

```
##  [1]  1.23633423  0.53181535 -1.22900317 -1.28332693 -2.05940800 -0.03666889
##  [7] -1.19813899  1.49239330  0.46235701 -0.32772583
```

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);
```



Generating a histogram using data from a simulated distribution like this is often a useful way to visualise distributions, or to see how samples from the same distribution might vary. For example, if we wanted to compare the above distribution with a normal distribution that had a standard deviation of 2 instead of 1, then we could simply sample 10000 new values in `rnorm` with `sd = 2` instead of `sd = 1` and create a new histogram with `hist`. If we wanted to see what the distribution of sampled data might look like given a low sample size (e.g., 10), then we could repeat the process of sampling from `rnorm(n = 10, mean = 0, sd = 1)` multiple times and looking at the shape of the resulting histogram.

Sampling from a poisson distribution

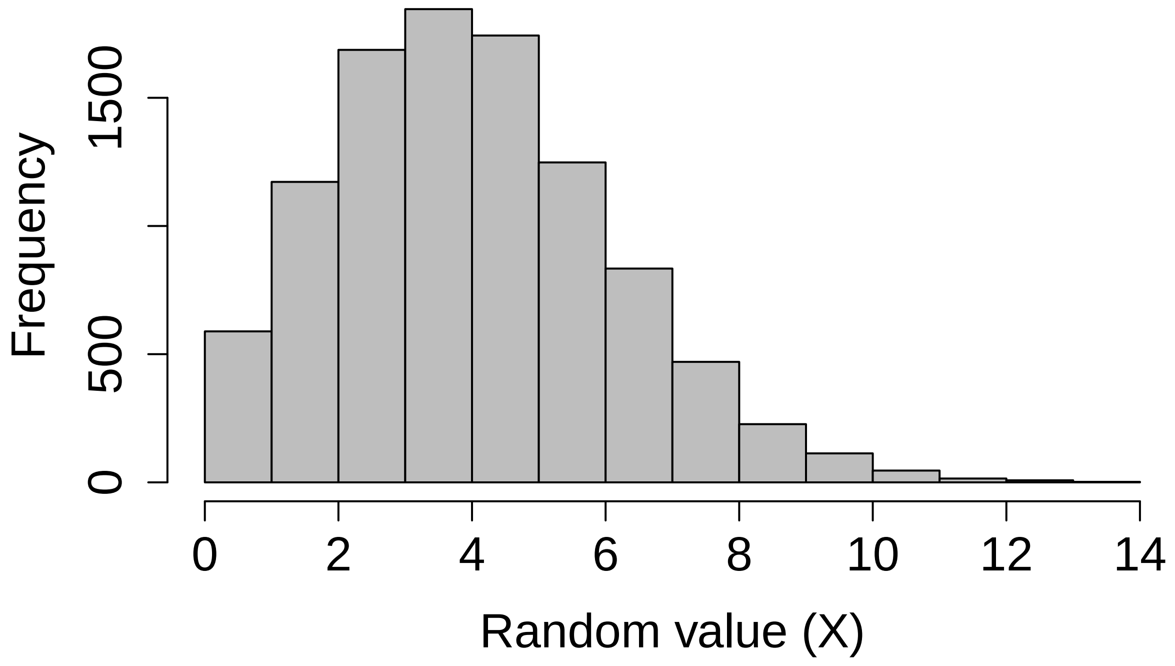
Many processes in biology can be described by a Poisson distribution. A Poisson process describes events happening with some given probability over an area of time or space such that $X \sim Poisson(\lambda)$, where the rate parameter λ is both the mean and variance of the Poisson distribution (note that by definition, $\lambda > 0$, and although λ can be any positive real number, data are always integers, as with count data). 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 = 1.5);
print(rand_poissons);
```

```
##  [1] 2 1 1 3 0 2 0 1 1 3
```

There are no default values for `rpois`. We can plot a histogram of a large number of values to see the distribution when $\lambda = 4.5$ below.

```
rand_poissons_10000 <- rpois(n = 10000, lambda = 4.5);
hist(rand_poissons_10000, xlab = "Random value (X)", col = "grey",
     main = "", cex.lab = 1.5, cex.axis = 1.5);
```



Sampling from a binomial distribution

Sampling from a binomial distribution in R with `rbinom` is a bit more complex than using `runif`, `rnorm`, or `rpois`. Like those 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] 517
```

The above result shows that the coin came up heads 517 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 a second time 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);
```

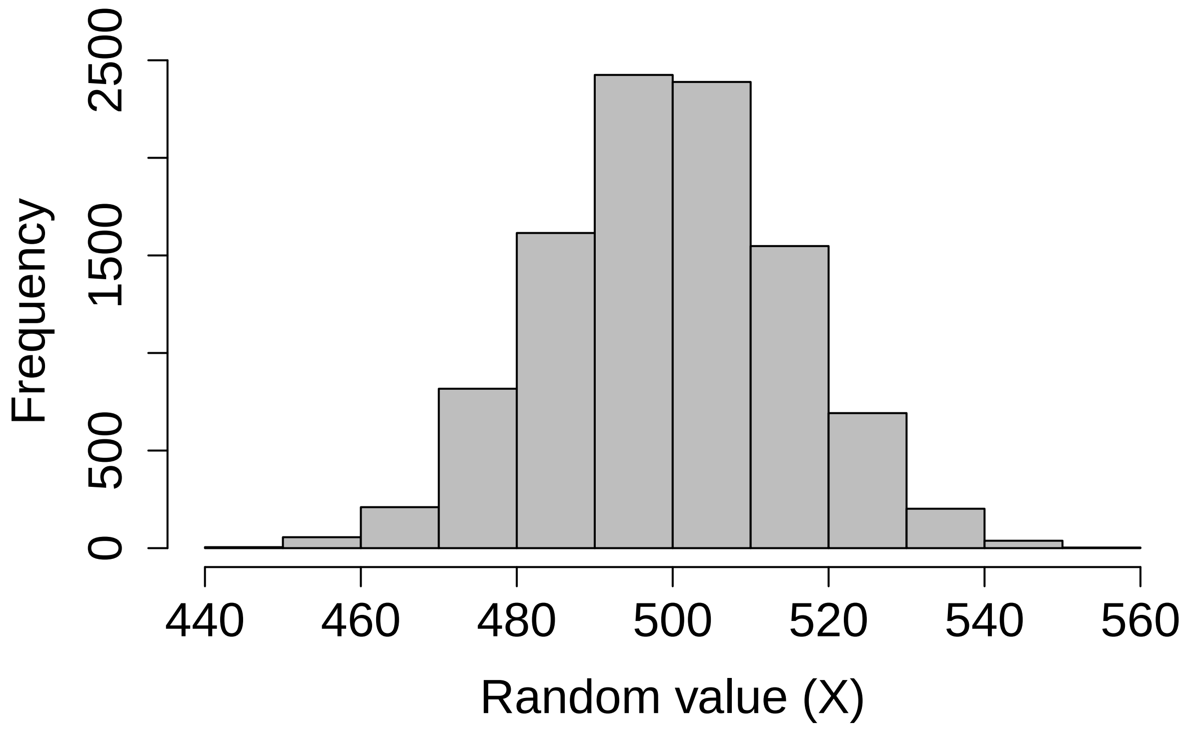
```
## [1] 517 505
```

In the above, a fair coin was flipped 1000 times and returned 517 heads, and then another fair coin was flipped 1000 times and returned 505 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 the number 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] 0 1 0 1 0 0 1 0 1 0
```

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`.

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).

Sampling random numbers from a list

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);
print(rand_number_1);
```

```
## [1] 10
```

The above code will set `rand_number_1` to a randomly selected value, in this case 10. 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] 10  1  5  4  3  7  2  8  9  6
```

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]  4  5  5  1  7  4  3  4 10  6
```

Note that the numbers {4, 5} 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] 10  6 10  1 10  8  6  7  9  6
```

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

Sampling random characters from a list

Sampling characters from a list of elements is no different than sampling numbers, but I am illustrating it separately because I find that I often sample characters for conceptually different reasons. For example, if I want to create a simulated data set that includes three different species, I might create a vector of species identities from which to sample.

```
species <- c("species_A", "species_B", "species_C");
```

This gives three possible categories, which I can now use `sample` to draw from. Assume that I want to simulate the sampling of these three species, perhaps with `species_A` being twice as common as `species_B` and `species_C`. I might use the following code to sample 24 times.

```
sp_sample <- sample(x = species, size = 24, replace = TRUE,
                    prob = c(0.5, 0.25, 0.25)
                    );
```

Below are the values that get returned.

```
##  [1] "species_A" "species_C" "species_A" "species_A" "species_B" "species_A"
##  [7] "species_A" "species_A" "species_A" "species_A" "species_C" "species_B"
## [13] "species_A" "species_A" "species_A" "species_B" "species_A" "species_B"
## [19] "species_A" "species_C" "species_A" "species_C" "species_B" "species_C"
```

Simulating data with known correlations

We can generate variables X_1 and X_2 that have known correlations ρ with with one another. The code below does this for two standard normal random variables with a sample size of 10000, such that the correlation between them is 0.3.

```
N   <- 10000;
rho <- 0.3;
x1  <- rnorm(n = N, mean = 0, sd = 1);
x2  <- (rho * x1) + sqrt(1 - rho*rho) * rnorm(n = N, mean = 0, sd = 1);
```

Mathematically, these variables are generated by first simulating the sample x_1 (`x1` above) from a standard normal distribution. Then, x_2 (`x2` above) is calculated as below,

$x_2 = \rho x_1 + \sqrt{1 - \rho^2} x_{rand}$,

Where x_{rand} is a sample from a normal distribution with the same variance as x_1. A simple call to the R function `cor` will confirm that the correlation does indeed equal `rho` (with some sampling error).

```
cor(x1, x2);
```

```
## [1] 0.3081863
```

This is useful if we are only interested in two variables, but there is a much more efficient way to generate any number of variables with different variances and correlations to one another. To do this, we need to use the MASS library, which can be installed and loaded as below.

```
install.packages("MASS");
library("MASS");
```

In the MASS library, the function `mvrnorm` can be used to generate any number of variables for a pre-specified covariance structure.

Suppose we want to simulate a data set of three measurements from a species of organisms. Measurement 1 (M_1) has a mean of $\mu_{M_1} = 159.54$ and variance of $Var(M_1) = 12.68$, measurement 2 (M_2) has a mean of $\mu_{M_1} = 245.26$ and variance of $Var(M_2) = 30.39$, and measurement 3 (M_2) has a mean of $\mu_{M_1} = 25.52$ and variance of $Var(M_3) = 2.18$. Below is a table summarising.

measurement	mean	variance
M1	159.54	12.68
M2	245.26	30.39
M3	25.52	2.18

Further, we want the covariance between M_1 and M_2 to equal $Cov(M_1, M_2) = 13.95$, the covariance between M_1 and M_3 to equal $Cov(M_1, M_3) = 3.07$, and the covariance between M_2 and M_3 to equal $Cov(M_2, M_3) = 4.7$. We can put all of this information into a covariance matrix **V** with three rows and three columns. The diagonal of the matrix holds the variances of each variable, with the off-diagonals holding the covariances (note also that the variance of a variable M is just the variable's covariance with itself; e.g., $Var(M_1) = Cov(M_1, M_1)$).

$$V = \begin{pmatrix} Var(M_1), & Cov(M_1, M_2), & Cov(M_1, M_3) \\ Cov(M_2, M_1), & Var(M_2), & Cov(M_2, M_3) \\ Cov(M_3, M_1), & Cov(M_3, M_2), & Var(M_3) \end{pmatrix}.$$

In R, we can create this matrix as follows.

```
matrix_data <- c(12.68, 13.95, 3.07, 13.95, 30.39, 4.70, 3.07, 4.70, 2.18);
cv_mat      <- matrix(data = matrix_data, nrow = 3, ncol = 3, byrow = TRUE);
rownames(cv_mat) <- c("M1", "M2", "M3");
colnames(cv_mat) <- c("M1", "M2", "M3");
```

Here is what `cv_mat` looks like (note that it is symmetrical along the diagonal).

```
##       M1    M2   M3
## M1 12.68 13.95 3.07
## M2 13.95 30.39 4.70
## M3  3.07  4.70 2.18
```

Now we can add the means to a vector in R.

```
mns <- c(159.54, 245.26, 25.52);
```

We are now ready to use the `mvrnorm` function in R to simulate some number `n` of sampled organisms with these three measurements. We use the `mvrnorm` arguments `mu` and `Sigma` to specify the vector of means and covariance matrix, respectively.

```
sim_data <- mvrnorm(n = 40, mu = mns, Sigma = cv_mat);
```

Here are the example data below.

M1	M2	M3
163.6085	251.3632	27.80722
160.0233	248.1269	25.51762
158.4631	245.0534	25.06711
156.9045	245.7987	25.62882
159.9151	245.2479	24.36989
164.3005	257.7962	27.74702
159.9607	245.1382	24.25519
157.5582	243.4723	23.50250
164.8011	248.9250	26.20114
154.2235	236.4833	24.27604
162.9883	245.3638	26.25470
159.4608	244.7730	26.00924
159.9567	247.7155	25.74406
159.5841	244.5894	24.17306
157.2536	238.0835	26.35091
163.7022	248.5122	25.56494
157.6672	245.0371	26.01742
158.1271	245.3727	24.82764
166.6379	257.8408	26.06314
154.7762	235.1836	23.92397
155.5872	240.5292	23.80275
160.1773	245.7244	27.25704
161.7522	247.9104	27.00468
163.9734	249.0662	26.27294
155.5921	247.6419	27.41349

M1	M2	M3
158.2671	247.3499	24.42533
157.6262	237.6908	22.32217
158.8771	247.3953	24.92742
157.0338	244.0608	26.08727
158.2326	247.3451	24.61032
156.3191	237.9830	24.28953
163.5261	244.8588	25.43823
162.6862	251.0953	25.03354
158.8962	238.8251	23.27923
161.5145	239.3350	25.56979
158.3486	238.4827	25.99601
159.8357	243.0909	26.07208
160.7675	242.2557	27.51231
153.1873	242.0465	23.86345
163.1338	245.6941	27.30345

We can check to confirm that the mean values of each column are correct using `apply`.

```
apply(X = sim_data, MARGIN = 2, FUN = mean);
```

```
##        M1        M2        M3
## 159.63117 244.95645  25.44457
```

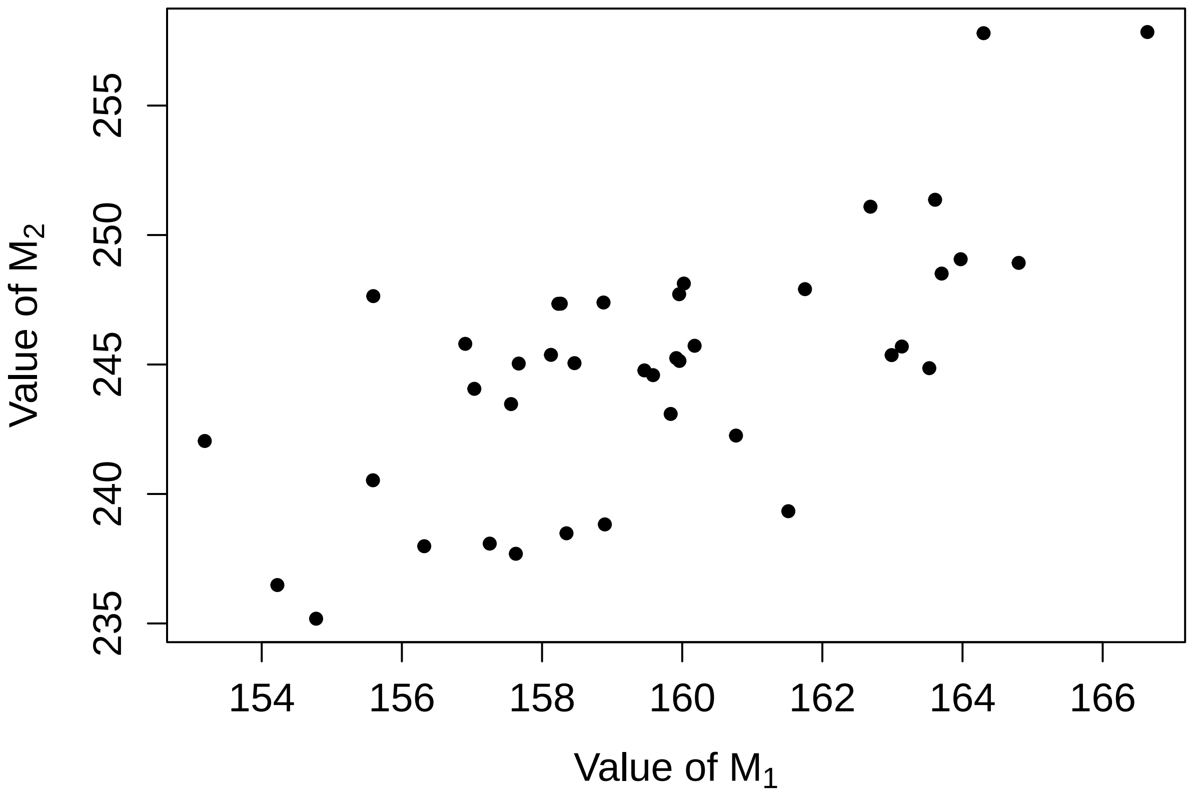
And we can check to confirm that the covariance structure of the data is correct using `cov`.

```
cov(sim_data);
```

```
##           M1        M2       M3
## M1  9.929561 10.910454 2.135212
## M2 10.910454 25.058145 3.353184
## M3  2.135212  3.353184 1.757532
```

Note that the values are not exact, but should become closer to the specified values as increase the sample size `n`. We can visualise the data too; for example, we might look at the close correlation between M_1 and M_2 using a scatterplot, just as we would for data sampled from the field.

```
par(mar = c(5, 5, 1, 1));
plot(x = sim_data[,1], y = sim_data[,2], pch = 20, cex = 1.25, cex.lab = 1.25,
     cex.axis = 1.25, xlab = expression(paste("Value of ", M[1])),
     ylab = expression(paste("Value of ", M[2])));
```



We could even run an ordination on these simulated data. For example, we could extract the principle components with `prcomp`, then plot the first two PCs to visualise these data. We might, for example, want to compare different methods of ordination using a data set with different, pre-specified properties (e.g., Minchin 1987). We might also want to use simulated data sets to investigate how different statistical tools perform. I show this in the next section, where I put a full data set together and run linear models on it.

Simulating a full data set

Putting everything together, here I will create a data set of three different species from which three different measurements are taken. We can just call these measurements 'length,' 'width,' and 'mass.' For simplicity, let us assume that these measurements always covary in the same way that we saw with **V** (i.e., `cv_mat`) above. But let's also assume that we have three species with slightly different mean values. Below is the code that will build a new data set of $N = 20$ samples with four columns: species, length, width, and mass.

```
N           <- 20;
matrix_data <- c(12.68, 13.95, 3.07, 13.95, 30.39, 4.70, 3.07, 4.70, 2.18);
cv_mat      <- matrix(data = matrix_data, nrow = 3, ncol = 3, byrow = TRUE);
mns_1       <- c(159.54, 245.26, 25.52);
sim_data_1  <- mvrnorm(n = N, mu = mns, Sigma = cv_mat);
colnames(sim_data_1) <- c("Length", "Width", "Mass");
# Below, I bind a column for indicating 'species_1' identity
species     <- rep(x = "species_1", times = 20); # Repeats 20 times
sp_1        <- data.frame(species, sim_data_1);
```

Let us add one more data column. Suppose that we can also sample the number of offspring each organism has, and that the mean number of offspring that an organism has equals one tenth of the organism's mass. To do this, we can use `rpois`, and take advantage of the fact that the argument `lambda` can be a vector rather than a single value. So to get the number of offspring for each organism based on its body mass, we can just insert the mass vector `sp_1$Mass` times 0.1 for `lambda`.

```
offspring    <- rpois(n = N, lambda = sp_1$Mass * 0.1);
sp_1         <- cbind(sp_1, offspring);
```

I have also bound the offspring number to the data set `sp_1`. Here is what it looks like below.

species	Length	Width	Mass	offspring
species_1	160.3827	249.4717	24.73354	3
species_1	158.1477	244.6999	25.54787	3
species_1	158.8443	247.0375	25.51422	4
species_1	153.0335	238.2794	25.22013	4
species_1	159.9288	246.1422	24.18515	2
species_1	159.5790	245.6793	23.74744	5
species_1	163.6944	252.7392	25.65012	5
species_1	152.8112	239.4175	22.83239	2
species_1	163.4611	245.3670	27.97450	2
species_1	156.6321	241.8300	24.35977	4
species_1	148.9986	236.7091	22.16069	2
species_1	162.1986	245.4546	23.69066	3
species_1	161.1931	241.0761	26.36017	5
species_1	162.5948	250.9681	27.31962	1
species_1	161.9920	244.1345	26.38530	1
species_1	159.6948	244.8550	24.52606	0
species_1	157.6525	248.2586	25.79400	3
species_1	159.3515	247.0168	26.85345	0
species_1	160.4292	243.7543	28.45303	2
species_1	158.0461	243.2817	27.81969	7

To add two more species, let us repeat the process two more times, but change the expected mass just slightly each time. The code below does this, and puts everything together in a single data set.

```
# First making species 2
mns_2        <- c(159.54, 245.26, 25.52 + 3); # Add a bit
sim_data_2   <- mvrnorm(n = N, mu = mns, Sigma = cv_mat);
colnames(sim_data_2) <- c("Length", "Width", "Mass");
species      <- rep(x = "species_2", times = 20); # Repeats 20 times
offspring    <- rpois(n = N, lambda = sim_data_2[,3] * 0.1);
sp_2         <- data.frame(species, sim_data_2, offspring);
# Now make species 3
mns_3        <- c(159.54, 245.26, 25.52 + 4.5); # Add a bit more
sim_data_3   <- mvrnorm(n = N, mu = mns, Sigma = cv_mat);
colnames(sim_data_3) <- c("Length", "Width", "Mass");
species      <- rep(x = "species_3", times = 20); # Repeats 20 times
offspring    <- rpois(n = N, lambda = sim_data_3[,3] * 0.1);
sp_3         <- data.frame(species, sim_data_3, offspring);
# Bring it all together in one data set
dat <- rbind(sp_1, sp_2, sp_3);
```

Our full data set now looks like the below.

species	Length	Width	Mass	offspring
species_1	160.3827	249.4717	24.73354	3
species_1	158.1477	244.6999	25.54787	3

species	Length	Width	Mass	offspring
species_1	158.8443	247.0375	25.51422	4
species_1	153.0335	238.2794	25.22013	4
species_1	159.9288	246.1422	24.18515	2
species_1	159.5790	245.6793	23.74744	5
species_1	163.6944	252.7392	25.65012	5
species_1	152.8112	239.4175	22.83239	2
species_1	163.4611	245.3670	27.97450	2
species_1	156.6321	241.8300	24.35977	4
species_1	148.9986	236.7091	22.16069	2
species_1	162.1986	245.4546	23.69066	3
species_1	161.1931	241.0761	26.36017	5
species_1	162.5948	250.9681	27.31962	1
species_1	161.9920	244.1345	26.38530	1
species_1	159.6948	244.8550	24.52606	0
species_1	157.6525	248.2586	25.79400	3
species_1	159.3515	247.0168	26.85345	0
species_1	160.4292	243.7543	28.45303	2
species_1	158.0461	243.2817	27.81969	7
species_2	164.0016	249.6304	24.17746	5
species_2	162.2563	247.0614	27.59001	5
species_2	160.1758	248.8443	26.86204	2
species_2	166.3756	249.5495	27.46349	0
species_2	156.5265	248.2137	24.12643	0
species_2	162.6832	249.6456	25.72152	0
species_2	159.3541	247.1917	25.67845	5
species_2	156.8025	237.5156	22.70427	1
species_2	164.4185	250.8626	26.85621	3
species_2	155.9606	236.9536	26.02208	3
species_2	159.7993	240.0509	25.81736	3
species_2	160.6364	248.2211	25.00965	3
species_2	159.8990	244.0576	24.73424	1
species_2	154.9236	240.3558	23.94631	5
species_2	158.6447	243.4271	24.60431	4
species_2	160.5226	240.0566	23.57381	2
species_2	155.1833	238.9279	24.50122	3
species_2	160.7832	244.2894	26.46857	2
species_2	160.8103	239.7546	27.12921	4
species_2	160.7928	243.1114	25.11401	3
species_3	154.8852	240.7362	24.86893	2
species_3	156.8818	239.3993	24.91473	2
species_3	158.5824	242.4102	23.04264	3
species_3	160.9414	254.8162	25.70268	2
species_3	162.2032	246.0668	26.61080	3
species_3	166.4624	254.7805	29.32544	3
species_3	159.9607	241.3097	24.37486	5
species_3	161.0547	255.0235	27.29515	2
species_3	157.9752	244.8211	22.94882	1
species_3	162.9361	250.8868	27.28743	3
species_3	159.2972	242.8962	27.76926	4
species_3	158.0839	240.0178	26.22109	4
species_3	154.8502	243.4432	22.13152	1
species_3	161.6293	246.4033	25.76150	1

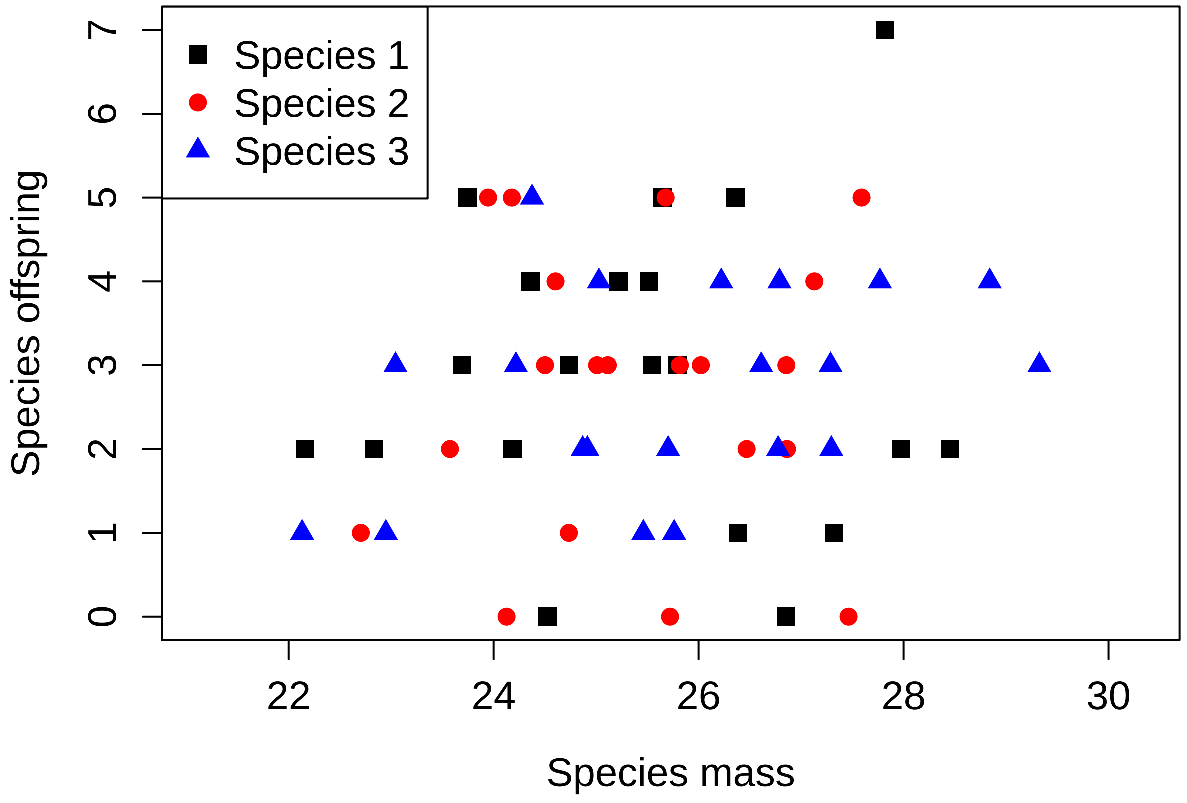
species	Length	Width	Mass	offspring
species_3	156.4545	242.7227	26.78899	4
species_3	157.0251	244.6212	25.02754	4
species_3	160.4657	244.5565	25.46118	1
species_3	153.9954	239.7789	24.21763	3
species_3	165.6975	257.4238	28.84053	4
species_3	162.5850	255.0434	26.77605	2

To summarise, we now have a simulated data set of measurements from three different species, all of which have known variances and covariances of length, width, and mass. Each species has a slightly different mean mass, and for all species, each unit of mass increases the expected number of offspring by 0.1. Because we know these properties of the data for certain, we can start asking questions that might be useful to know about our data analysis. For example, given this covariance structure and these small differences in mass, is a sample size of 20 really enough to even get a significant difference among species masses using an ANOVA? What if we tried to test for differences among masses using some sort of randomisation approach Instead? Would this give us more or less power? Let us run an ANOVA to see if the difference between group means (which we know exists) is recovered.

```
aov_result <- aov(Mass ~ species, data = dat);
summary(aov_result);
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## species      2   1.55  0.7731   0.273  0.762
## Residuals   57 161.20  2.8280
```

It appears not! What about the relationship between body mass and offspring production that we know exists? Below is a scatterplot of the data for the three different species.



This looks like there might be a positive relationship, but it is very difficult to determine just from the scatterplot. We can use a generalised linear model to test it with species as a random effect, as we might do if these were data sampled from the field (do not worry about the details of the model here; the key point is that we can use the simulated data with known properties to assess the performance of a statistical test).

```
library(lme4);
```

```
## Loading required package: Matrix
```

```
mod <- glmer(offspring ~ Mass + (1 | species), data = dat, family = "poisson");
```

```
## boundary (singular) fit: see ?isSingular
```

```
summary(mod);
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: offspring ~ Mass + (1 | species)
##    Data: dat
##
##      AIC      BIC   logLik deviance df.resid
##    228.2    234.5   -111.1    222.2       57
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -1.73476 -0.58198  0.08448  0.64577  2.25396
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  species (Intercept) 0        0
## Number of obs: 60, groups:  species, 3
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.14073    1.21225  -0.116    0.908
## Mass         0.04524    0.04713   0.960    0.337
##
## Correlation of Fixed Effects:
##      (Intr)
## Mass -0.998
## optimizer (Nelder_Mead) convergence code: 0 (OK)
## boundary (singular) fit: see ?isSingular
```

There does not appear to be any effect here either! To get one, it appears that we will need to simulate a larger data set (or a bigger effect size – or just get lucky when re-simulating a new data set).

Note that I have run a linear model that might be reasonable given the structure of our data. But the advantage of working with simulated data and knowing for certain what the relationship is between the underlying variables is that we can explore different statistical techniques. For example, we know that our response variable `offspring` is count data, so we are supposed to specify a Poisson error structure using the `family = "poisson"` argument above, right? But what would happen if we just used a normal error structure anyway? Would this really be so bad? Now is the opportunity to test because we *know* what the correct answer is supposed to be! Trying statistical methods that are normally ill-advised can actually be useful here, as it can help us see for ourselves when a technique is bad – or perhaps when it really is not (e.g., Ives 2015).

Conclusions

Simulating data can be a powerful tool for learning and investigating different statistical analyses. The main benefits of using simulated data are flexibility and certainty. Simulation gives us the flexibility to explore any number of hypotheticals, including different sample sizes, effect sizes, relationships between variables, and error distributions. It also works from a point of certainty; we know what the real relationship is between variables, and what the actual effect sizes are because we define them when generating random samples. So if we want to better understand what would happen if we were unable to sample an important variable in our system, or if we were to use a biased estimator, or if we we were to violate key model assumptions, simulated data is a very useful tool.

Literature cited

Box, G E P, and Mervin E Muller. 1958. "A note on the generation of random normal deviates." *The Annals of Mathematical Statistics* 29 (2): 610–11. https://doi.org/10.1214/aoms/1177706645.

Ives, Anthony R. 2015. "For testing the significance of regression coefficients, go ahead and log-transform count data." *Methods in Ecology and Evolution* 6: 828–35. https://doi.org/10.1111/2041-210X.12386.

Minchin, Peter R. 1987. "An evaluation of the relative robustness of techniques for ecological ordination." *Vegetatio* 69 (1-3): 89–107. https://doi.org/10.1007/BF00038690.