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.

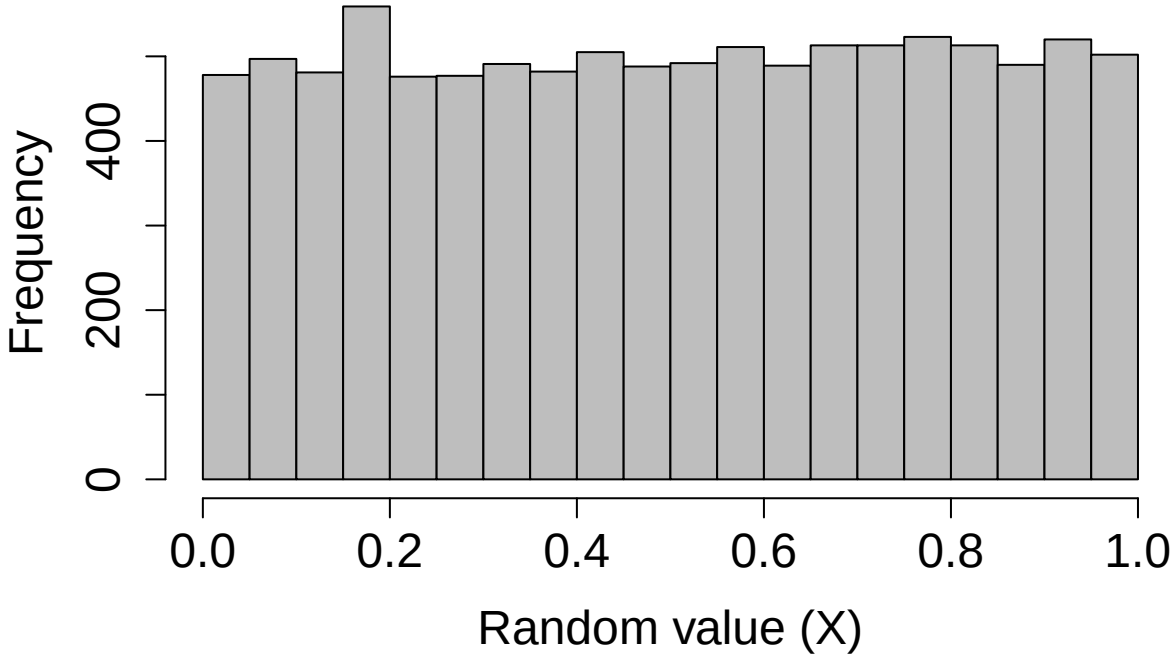
```
rand_unifs_10 <- runif(n = 10, min = 0, max = 1);
```

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] 0.7016627 0.8706902 0.5874151 0.1868191 0.9515830 0.4999142 0.3926171
##  [8] 0.2343125 0.8348191 0.6342652
```

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 $\sigma = 1$).

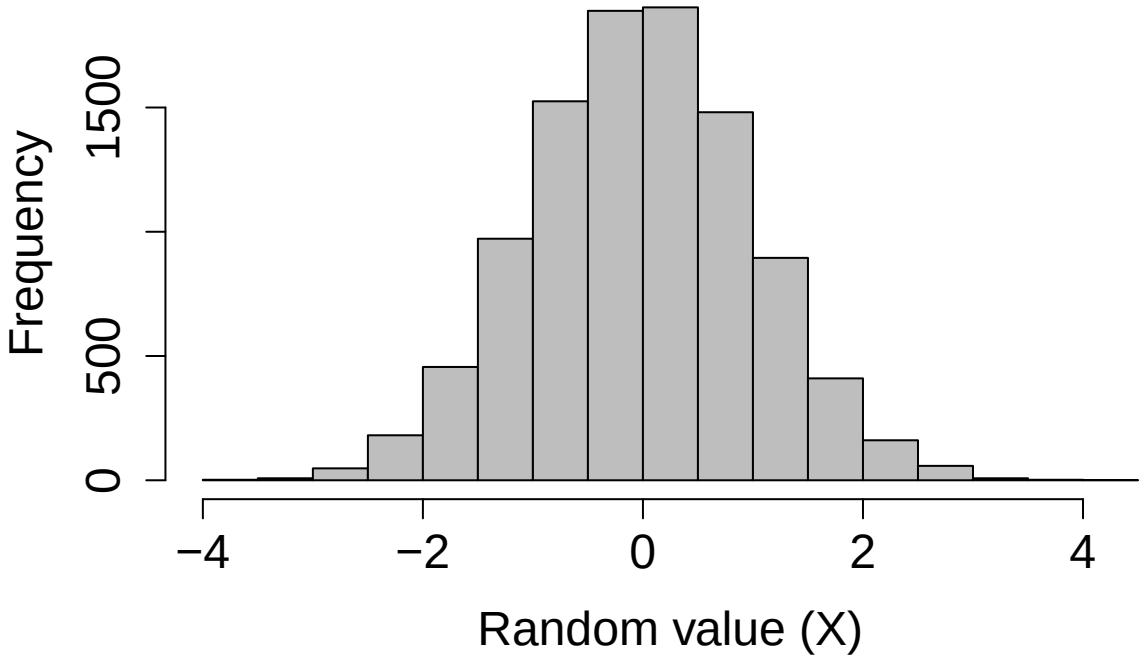
```
rand_norms_10 <- rnorm(n = 10, mean = 0, sd = 1);
```

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

```
##  [1]  0.25631514 -0.06364807 -1.10308081 -0.12529744 -1.06230734  0.49665052
##  [7] -0.08503064  0.91140182  0.12402544 -0.75528939
```

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

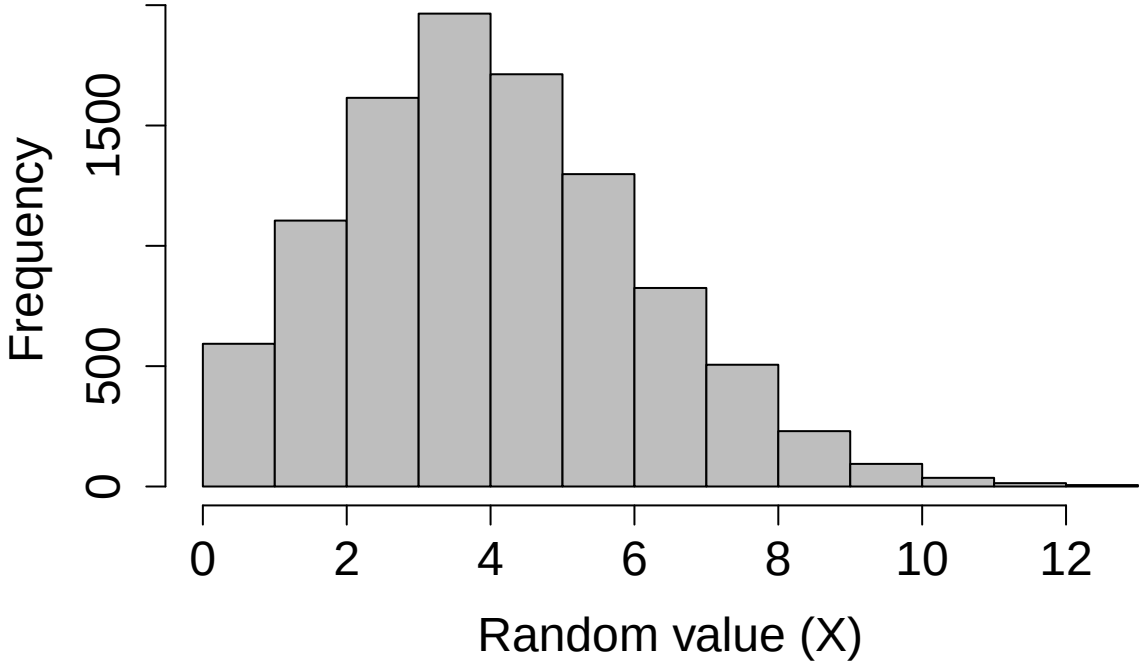
number of values to be returned (`n`) and the rate parameter (`lambda`).

```
rand_poissons <- rpois(n = 10, lambda = 1.5);
print(rand_poissons);
```

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

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] 518
```

The above result shows that the coin came up heads 518 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);
```

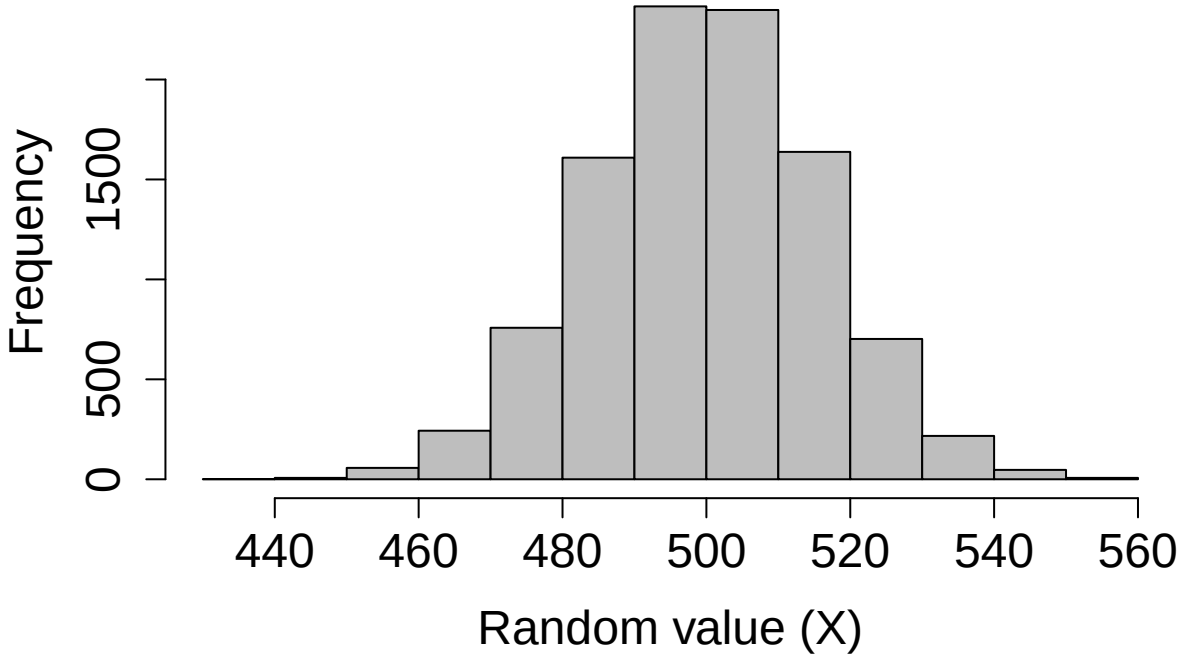
```
## [1] 524 493
```

In the above, a fair coin was flipped 1000 times and returned 524 heads, and then another fair coin was flipped 1000 times and returned 493 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 0 1 1 0 1 0 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] 7
```

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

Note that the numbers $\{3, 5\}$ 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]  8 10  6  9 10  9  5  8  7  8
```

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_A" "species_A" "species_B" "species_A" "species_A"
##  [7] "species_A" "species_B" "species_A" "species_A" "species_B" "species_C"
```

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

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] 0.2940007
```

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 $\mathbf{V}$ 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
159.2441	245.1846	23.70012
161.6302	243.2267	25.70892
157.7149	242.0864	25.64934
153.5796	236.2386	23.57932
167.0418	249.7770	26.42331
156.2224	244.0953	24.32586
158.2659	246.9075	24.79480
157.5022	242.7998	25.08861
160.7195	250.3263	27.91161
147.2343	223.5011	21.92377
159.6722	245.7136	23.36233
160.9418	248.7124	25.39382
157.0212	245.5450	24.79397
164.2168	252.8650	27.03133
161.5658	237.8498	22.99205
149.3606	231.4303	24.62678
156.4921	240.5653	24.61371
160.8303	248.7314	29.16958
157.1596	242.5213	27.36654
168.0505	257.3380	28.16901
157.1799	253.4883	26.71838
160.1161	240.7385	23.71568
159.8975	245.9273	24.39548
157.1913	243.3603	23.81638
164.3855	250.5730	26.96603
154.5454	242.1087	25.50410
161.6404	253.5170	27.93042
164.9203	247.0045	25.04879
157.8916	242.1192	25.49056
159.2734	246.9788	24.35041

M1	M2	M3
161.1968	245.3179	25.59629
162.9446	252.3643	26.92969
167.4104	253.2580	27.07034
157.1369	244.2880	25.50487
165.0468	248.2204	25.55662
159.5423	246.9659	26.52739
166.0932	253.5098	26.93427
160.7626	245.0615	24.98270
157.7186	246.3393	27.07158
166.8652	247.5106	25.53865

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
## 159.90562 245.60167  25.55683
```

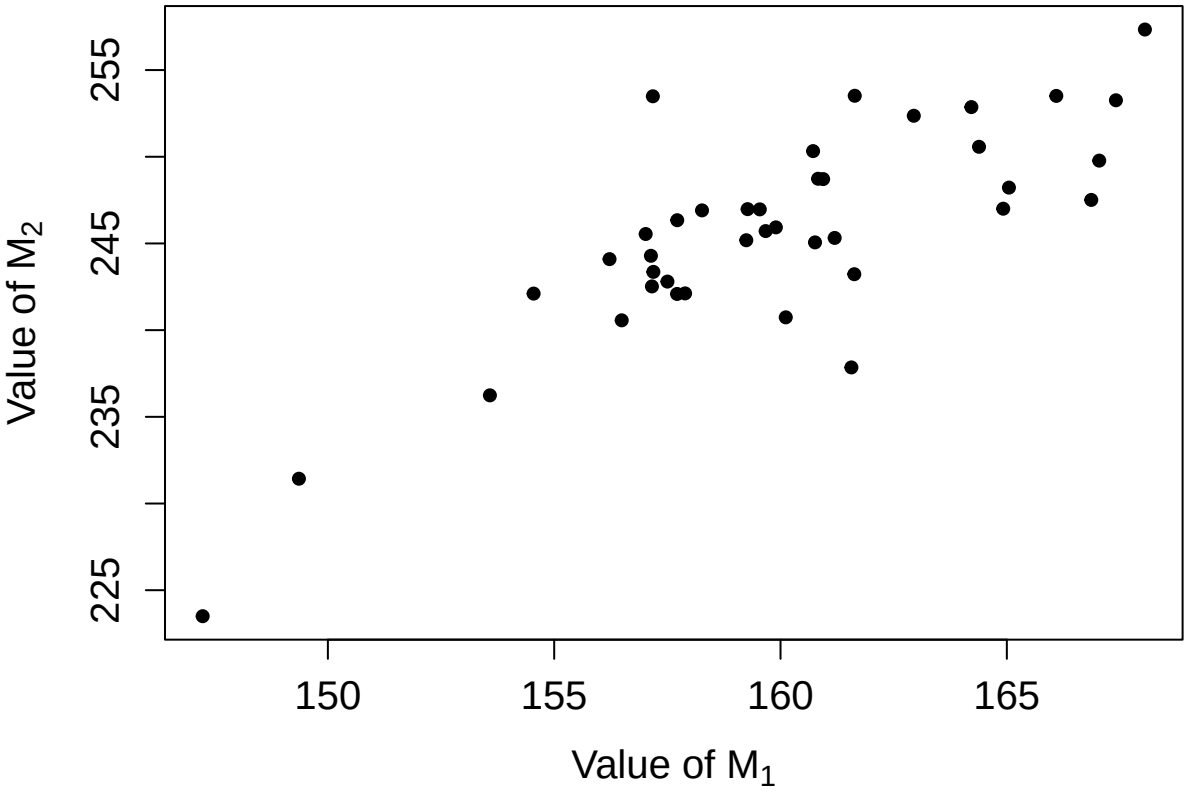
And we can check to confirm that the covariance structure of the data is correct using `cov`.

```
cov(sim_data);
```

```
##           M1        M2       M3
## M1 20.531617 22.773609 3.650357
## M2 22.773609 39.488386 7.180066
## M3  3.650357  7.180066 2.471339
```

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 $\mathbf{V}$ (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	163.8682	251.6117	24.99024	3
species_1	157.3884	248.0621	24.43774	2
species_1	155.6753	244.9217	24.16772	0
species_1	161.9808	242.7958	24.18843	0
species_1	160.1588	242.9107	27.85119	4
species_1	155.7544	231.8737	23.45579	3
species_1	162.6100	254.3312	25.82024	4
species_1	161.8533	244.1754	25.45159	1
species_1	158.6357	244.2108	23.89156	2
species_1	158.9882	248.4129	26.51485	4
species_1	163.5670	242.7449	27.26947	2
species_1	165.6599	251.3383	28.61582	2
species_1	163.7387	249.2383	27.45232	5
species_1	160.5288	247.3896	28.24590	3
species_1	161.3078	246.2869	25.12874	1
species_1	153.8870	232.9227	22.68581	5
species_1	158.7391	239.8752	25.22237	6
species_1	159.8872	243.8680	26.05544	0
species_1	155.9837	238.4647	23.60703	4
species_1	162.5246	250.5945	25.56469	6

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	163.8682	251.6117	24.99024	3
species_1	157.3884	248.0621	24.43774	2

species	Length	Width	Mass	offspring
species_1	155.6753	244.9217	24.16772	0
species_1	161.9808	242.7958	24.18843	0
species_1	160.1588	242.9107	27.85119	4
species_1	155.7544	231.8737	23.45579	3
species_1	162.6100	254.3312	25.82024	4
species_1	161.8533	244.1754	25.45159	1
species_1	158.6357	244.2108	23.89156	2
species_1	158.9882	248.4129	26.51485	4
species_1	163.5670	242.7449	27.26947	2
species_1	165.6599	251.3383	28.61582	2
species_1	163.7387	249.2383	27.45232	5
species_1	160.5288	247.3896	28.24590	3
species_1	161.3078	246.2869	25.12874	1
species_1	153.8870	232.9227	22.68581	5
species_1	158.7391	239.8752	25.22237	6
species_1	159.8872	243.8680	26.05544	0
species_1	155.9837	238.4647	23.60703	4
species_1	162.5246	250.5945	25.56469	6
species_2	151.0074	235.1868	24.43762	3
species_2	161.4289	245.5970	26.17295	1
species_2	158.5229	247.9272	25.39738	5
species_2	162.4390	247.1905	25.15459	2
species_2	152.2133	240.1924	25.39406	2
species_2	163.5624	251.5296	26.61883	1
species_2	153.3979	240.3565	24.70944	1
species_2	159.6647	254.2408	27.27693	4
species_2	157.1931	247.9641	24.04959	2
species_2	158.9740	249.0670	26.38413	2
species_2	152.8416	233.7842	24.98711	0
species_2	156.9856	237.9629	24.48126	2
species_2	160.1800	253.6052	26.09130	2
species_2	163.1142	250.6302	27.27560	1
species_2	152.0195	241.0907	25.41110	3
species_2	160.7028	247.1115	24.51558	7
species_2	167.6187	255.5554	27.50919	6
species_2	153.6705	239.8152	24.35165	2
species_2	154.6208	240.8222	24.75071	4
species_2	160.9536	245.7092	25.66063	2
species_3	158.2779	243.4118	26.41250	3
species_3	160.1410	244.5154	25.63116	6
species_3	160.5813	248.6023	26.42931	2
species_3	152.2170	242.0749	21.46261	2
species_3	151.3275	233.4769	23.82199	3
species_3	159.7834	242.9539	25.34915	3
species_3	156.0962	242.0108	26.83570	1
species_3	155.5500	252.9910	25.88176	5
species_3	157.7904	251.0123	26.23333	7
species_3	157.4988	230.4649	22.97713	4
species_3	169.3800	255.3308	26.93261	2
species_3	157.1218	252.4843	25.04706	5
species_3	153.7232	238.3739	25.69657	3
species_3	162.3686	248.3613	25.87887	2

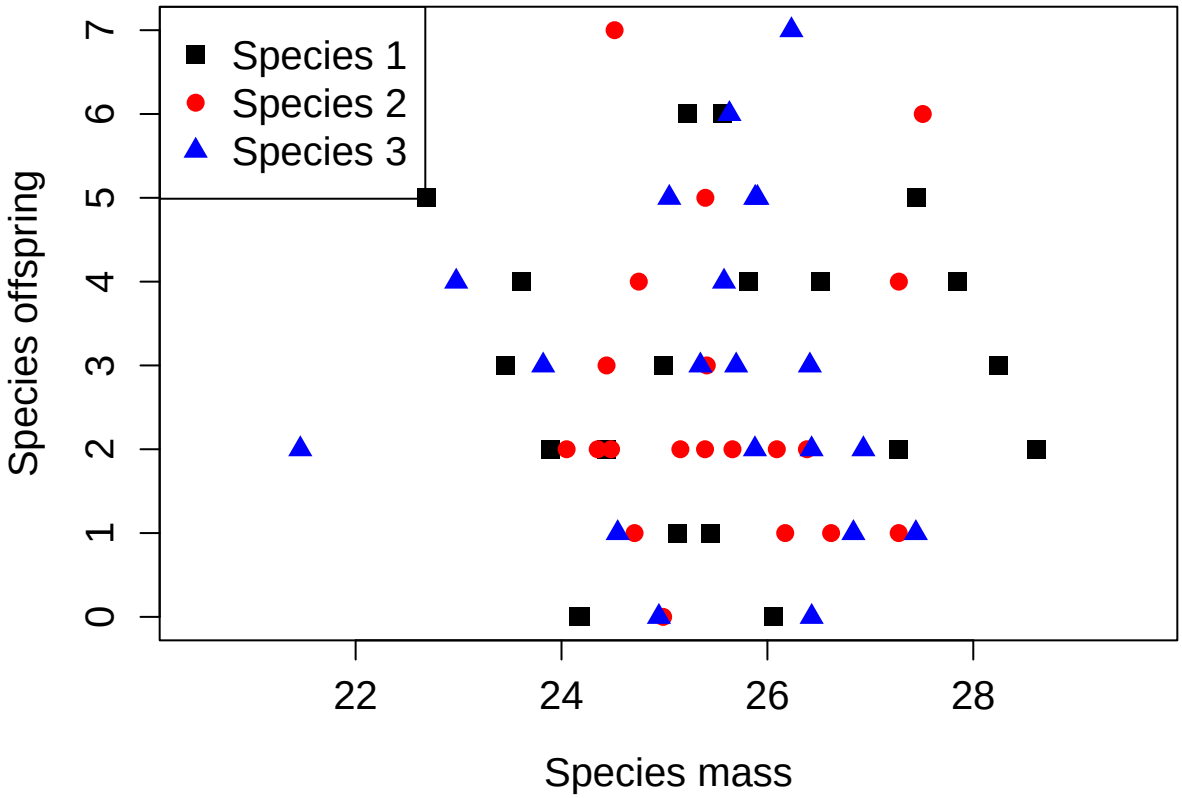
species	Length	Width	Mass	offspring
species_3	159.9588	248.5491	27.44212	1
species_3	160.5507	246.8173	24.54571	1
species_3	159.9142	240.1713	24.94603	0
species_3	161.8066	253.6563	25.90308	5
species_3	157.9241	243.5715	25.57874	4
species_3	160.0359	242.4979	26.43001	0

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   0.05  0.0235   0.012  0.988
## Residuals   57 113.99  1.9999
```

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
##    245.5    251.8   -119.7    239.5       57
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.6802 -0.6549 -0.4416  0.7198  2.5363
##
## 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) 0.801138   1.432927   0.559    0.576
## Mass        0.008953   0.056049   0.160    0.873
##
## Correlation of Fixed Effects:
##      (Intr)
## Mass -0.999
## 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.