Creating simulated data sets in R

Brad Duthie

14 February 2024

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 PDF.

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.

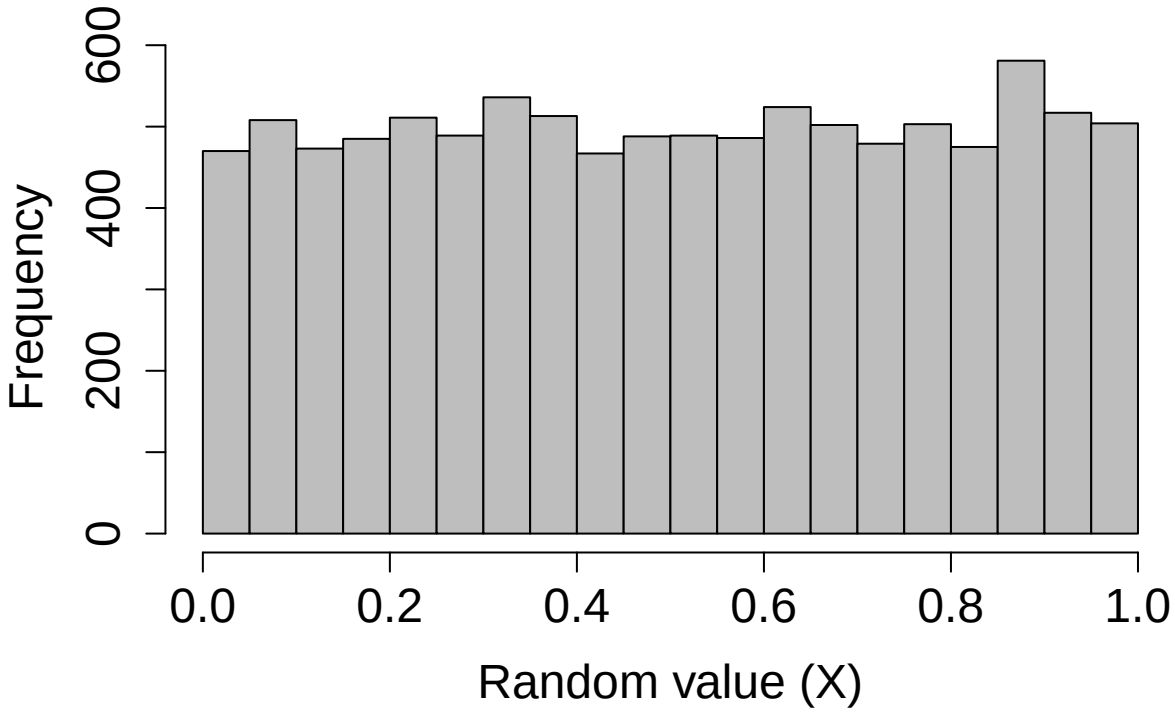
```
rand_unifs_10 <- runif(n = 10, min = 0, max = 1);
```

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] 0.26030195 0.58919966 0.07031305 0.27408554 0.57796705 0.60746196
##  [7] 0.01563063 0.67288649 0.91994100 0.55364508
```

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 $\sigma = 1$).

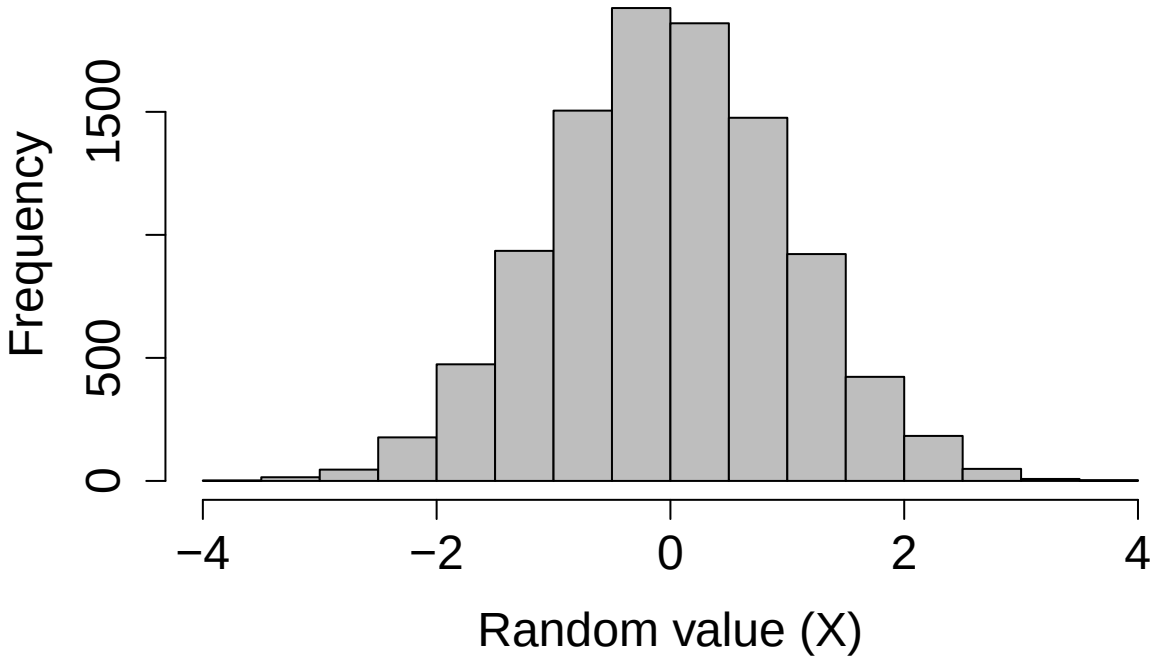
```
rand_norms_10 <- rnorm(n = 10, mean = 0, sd = 1);
```

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

```
##  [1] -0.7131560 -0.4735036  0.3816503 -1.2008142 -0.3763529  0.5344435
##  [7]  0.3729368 -0.6031522 -1.0657826 -0.4197172
```

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

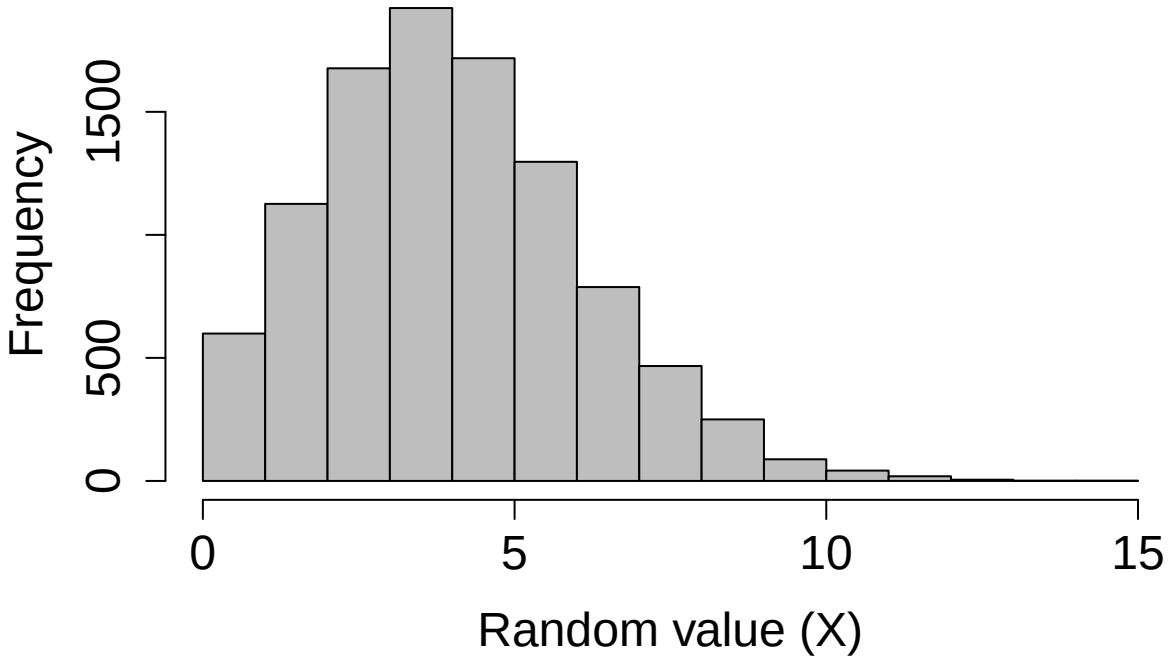
Many processes in biology can be described by a Poisson distribution. A Poisson process describes events happening with some given probability over an area of time or space such that $X \sim Poisson(\lambda)$, where the rate parameter λ is both the mean and variance of the Poisson distribution (note 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 3 3 3 4 1 3 2 2 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] 525
```

The above result shows that the coin came up heads 525 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);
```

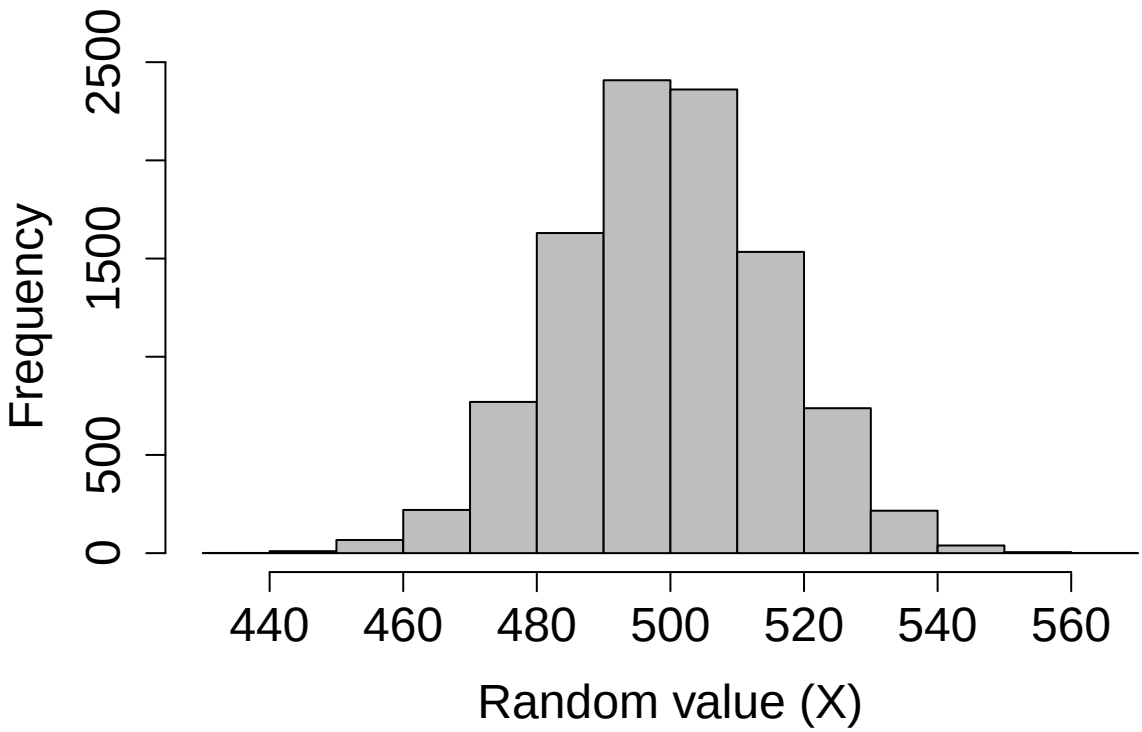
```
## [1] 521 485
```

In the above, a fair coin was flipped 1000 times and returned 521 heads, and then another fair coin was flipped 1000 times and returned 485 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] 1 0 1 0 0 0 0 0 1 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`.

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

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] 8 7 3 3 2 5 9 1 2 3
```

Note that the numbers $\{2, 3\}$ 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]  2  6  9 10  2  2  9  1  9  9
```

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_C" "species_A" "species_A"
##  [7] "species_C" "species_C" "species_A" "species_A" "species_B" "species_A"
## [13] "species_A" "species_B" "species_A" "species_B" "species_A" "species_A"
## [19] "species_A" "species_A" "species_C" "species_A" "species_A" "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] 0.2997275
```

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
163.6518	243.0293	27.50490
153.2536	234.2305	22.35618
158.1672	243.1517	24.39938
161.3257	251.3148	23.35492
162.1920	257.0307	26.26973
153.8770	243.3345	24.61258

M1	M2	M3
158.1259	251.1981	24.52671
160.5229	248.4448	27.96243
157.2187	243.2872	23.46458
158.2091	249.9774	24.22677
164.1557	249.9719	26.54893
157.7288	243.4278	24.41553
157.7326	245.0448	24.92287
158.1038	247.6874	25.55243
152.6896	231.8753	25.38396
157.4173	240.6230	22.70699
160.9749	240.0718	25.73651
156.7864	246.9460	26.26664
158.1673	244.0407	23.99257
153.8627	239.9618	24.35657
155.7099	243.9635	26.05590
157.8196	243.6506	23.02626
160.2522	249.9039	26.26784
158.3961	245.5127	26.68749
160.4448	245.6159	25.29746
161.0740	247.0418	24.73032
158.0939	247.3563	28.70769
157.4622	241.8673	24.86231
164.4666	249.6383	28.84425
156.5644	234.9602	22.79125
158.7013	244.1753	25.29089
161.5468	248.9817	27.12725
163.6412	256.3574	27.04064
158.3780	249.8303	24.71480
159.0816	244.0004	26.29937
162.4331	249.9222	25.57570
157.5452	247.0619	24.14479
156.2691	241.8423	23.96189
162.0215	240.9311	24.96541
157.5077	244.3142	26.08924

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
## 158.78931 245.28942  25.27605
```

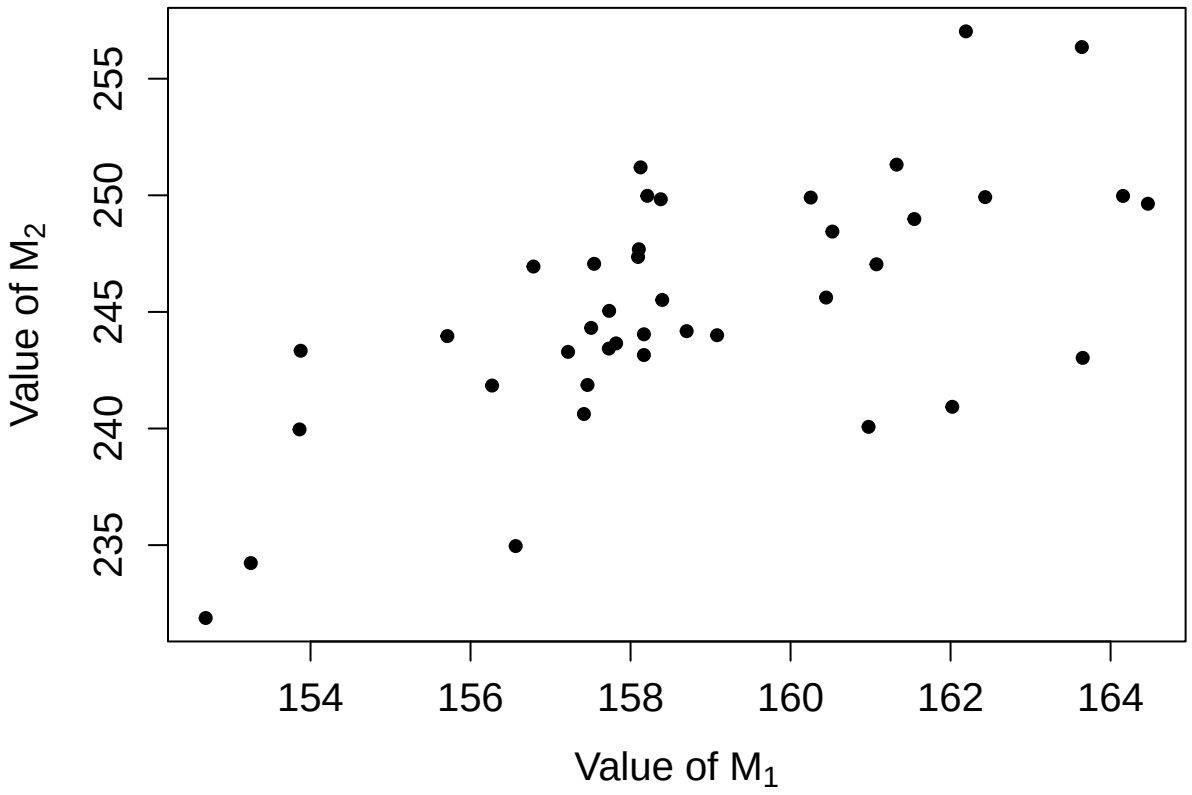
And we can check to confirm that the covariance structure of the data is correct using `cov`.

```
cov(sim_data);
```

```
##          M1        M2       M3
## M1 8.509932  9.859769 2.421537
## M2 9.859769 27.186711 3.618141
## M3 2.421537  3.618141 2.460394
```

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	159.9533	243.4227	22.72765	4
species_1	159.7499	244.5644	25.27408	0
species_1	164.5741	255.2495	26.59350	3
species_1	159.8969	250.5151	25.11469	3
species_1	153.7585	233.1831	23.16144	3
species_1	163.5869	254.4058	27.26223	2
species_1	158.7672	242.4418	24.28717	5
species_1	158.1837	240.0855	26.11573	1
species_1	153.8214	239.4643	23.31309	2
species_1	158.3107	245.9407	25.07996	2
species_1	160.0943	242.7369	24.44470	1
species_1	158.2343	237.7177	26.37711	3
species_1	159.4497	250.6303	26.76947	1
species_1	157.4304	241.9654	24.54231	4
species_1	155.9335	235.4176	25.89888	3
species_1	160.6861	247.2428	25.51683	1
species_1	158.0153	243.8761	26.17362	5
species_1	161.8486	241.5219	23.39056	2
species_1	162.3947	245.3846	25.30323	4
species_1	158.4513	243.3200	25.01094	4

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	159.9533	243.4227	22.72765	4
species_1	159.7499	244.5644	25.27408	0
species_1	164.5741	255.2495	26.59350	3
species_1	159.8969	250.5151	25.11469	3
species_1	153.7585	233.1831	23.16144	3
species_1	163.5869	254.4058	27.26223	2
species_1	158.7672	242.4418	24.28717	5
species_1	158.1837	240.0855	26.11573	1
species_1	153.8214	239.4643	23.31309	2
species_1	158.3107	245.9407	25.07996	2
species_1	160.0943	242.7369	24.44470	1
species_1	158.2343	237.7177	26.37711	3
species_1	159.4497	250.6303	26.76947	1
species_1	157.4304	241.9654	24.54231	4
species_1	155.9335	235.4176	25.89888	3
species_1	160.6861	247.2428	25.51683	1
species_1	158.0153	243.8761	26.17362	5
species_1	161.8486	241.5219	23.39056	2
species_1	162.3947	245.3846	25.30323	4
species_1	158.4513	243.3200	25.01094	4
species_2	160.2528	248.5692	24.99606	3
species_2	160.7029	240.0511	24.29649	2
species_2	158.7433	248.9468	27.24495	1
species_2	157.7189	243.6334	24.78088	1
species_2	160.0190	244.2014	25.37037	3
species_2	158.2989	243.1543	24.72782	1
species_2	158.2005	239.0357	27.30184	3
species_2	158.1541	241.9820	25.68276	2
species_2	161.1973	240.4725	24.88878	3
species_2	155.9321	242.7676	24.58200	1
species_2	156.6690	237.6551	25.27099	3
species_2	159.3251	246.9798	24.29475	1
species_2	159.6979	244.5094	26.28466	1
species_2	156.1763	240.9722	24.85592	2
species_2	155.0683	240.1483	25.14894	1
species_2	159.1114	242.0123	26.48438	0
species_2	158.8586	239.3737	26.43328	4
species_2	160.3166	242.1038	24.37108	3
species_2	161.1974	241.3731	25.05036	2
species_2	159.0816	247.3352	26.40622	4
species_3	164.3057	249.7958	26.65763	3
species_3	156.6158	245.8312	24.62549	5

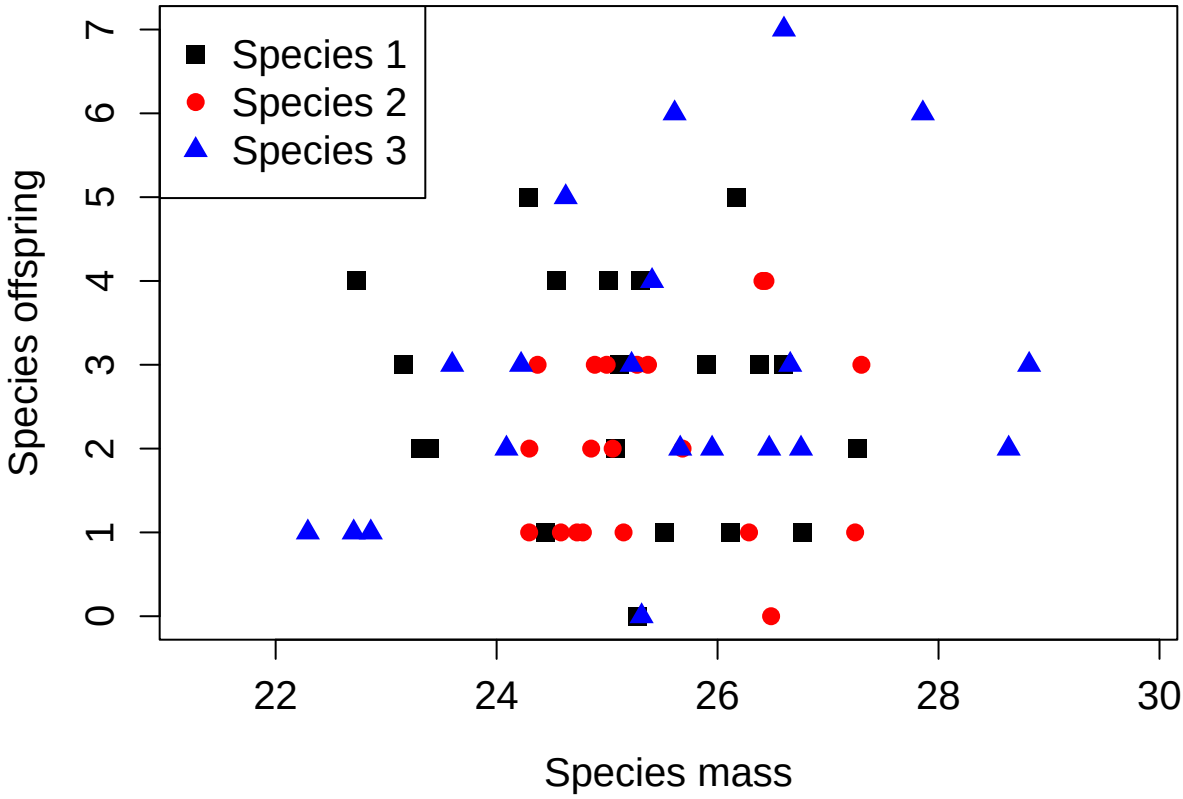
species	Length	Width	Mass	offspring
species_3	158.5840	236.4036	22.29149	1
species_3	156.9685	238.9608	23.59813	3
species_3	160.9984	242.9457	25.61123	6
species_3	158.7672	247.2871	25.40712	4
species_3	163.3584	252.2415	26.60087	7
species_3	165.5165	252.1904	28.82005	3
species_3	156.6090	240.8090	25.66192	2
species_3	157.4001	248.4543	22.86118	1
species_3	156.8166	246.6605	22.70648	1
species_3	155.8112	237.6733	25.95032	2
species_3	157.9696	241.5863	24.22165	3
species_3	160.2483	251.3143	25.31238	0
species_3	163.7988	244.1552	26.75506	2
species_3	161.2938	243.0966	26.46720	2
species_3	155.4354	235.9312	25.22167	3
species_3	154.0329	236.7923	28.63467	2
species_3	157.6275	241.5606	27.85791	6
species_3	156.4345	240.2656	24.08900	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   1.45  0.7258   0.365  0.696
## Residuals   57 113.23  1.9865
```

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");
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, : Model is nearly unide
##  - Rescale variables?
```

```
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
##    219.5    225.8   -106.8    213.5       57
##
## Scaled residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -1.6488 -0.7751 -0.1460  0.4443  2.5741
##
## Random effects:
##  Groups  Name        Variance  Std.Dev.
##  species (Intercept) 1.979e-06 0.001407
## Number of obs: 60, groups:  species, 3
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.72297    1.48979  -0.485    0.627
## Mass         0.06506    0.05843   1.113    0.266
##
## Correlation of Fixed Effects:
##      (Intr)
## Mass -0.999
## optimizer (Nelder_Mead) convergence code: 0 (OK)
## Model is nearly unidentifiable: large eigenvalue ratio
##  - Rescale variables?
```

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.