# Lab 1: Introduction to Data Generation

## EDP 380C.28: Simulation in R

Fall 2022– Unique: 12104

Brian T. Keller, Ph.D.
bk@utexas.edu
University of Texas at Austin
August 22, 2022

---

### Objectives

- Learn about Directed Acyclic Graphs and their use for data generation.
- Learn how to generate univariate normally distributed data.
- Learn how to generate bivariate normally distributed data.
- Learn how to create and use functions in R and use `apply` on an array.

---

This lab will introduce you to using ***directed acyclic graphs*** (DAGs) to describe simple data generation processes. We will use these diagrams throughout the semester to abstractly visualize statistical models (i.e., joint probability distributions). From there, we can use a ***functional*** notation to translate the graphs into equations allowing us to specify distributional restrictions on variables. Throughout the course labs, we will use boxes like the one below to describe the steps you must implement in R.

**In R:**

This section will include ordered steps that you must implement in R. In the initial labs, these steps will be very concrete and specific, but as we continue, the steps will grow more general and require you to make your own decisions.

**(1)** Run the following included code to set up a `list()` to store answers.

```
answers <- vector('list', 9L)
```

***Note:*** It is essential to follow the steps in the listed order to have answers graded as correct.

**Answer:**

This section will discuss the outputs' requirements, including object structure and labels you ***must*** obey. In this lab specifically, you will produce results and save them to elements in the `answers` object created above.

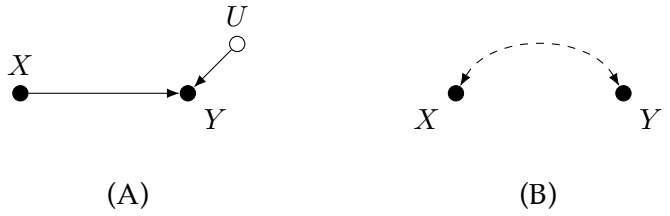


Figure 1: Two examples of Directed Acyclic Graphs

# 1 Directed Acyclic Graph (DAG)

As a general framework, ***directed acyclic graphs*** (DAGs, graphs) span multiple disciplines (e.g., family trees, computational scheduling, citation networks). Our use closely resembles the usage in path analysis, structural equation modeling, causal inference, and missing data literature (known as $m$-graphs).

Figure 1A provides an example of a DAG for the regression of $Y$ on $X$, symbolically represented by $X \rightarrow Y$. In this figure, we have three ***nodes*** represented by the circles labeled with $X$, $Y$, and $U$. The $U$ node is an empty circle representing an unobserved or ***latent*** variable. In addition, we have two arrows or ***edges*** representing the directed relationship between two variables. Note, in casual modeling the direction implies a causal relationship (i.e., $X$ causes $Y$), but throughout this course, that will not always be the case. Figure 1B illustrates how a DAG represents a correlation between two variables, $X \longleftrightarrow Y$. In the casual literature, a correlation represents an unspecified common cause between two variables (e.g., $X \leftarrow U \rightarrow Y$). Throughout this course, we often use this specification to represent two variables with an unspecified linear association.

We will use DAGs to help us in two distinct ways. First, they provide us a graphical representation of the statistical model from which we are trying to generate data. Returning to Figure 1, both A and B offer two different statistical models. Figure 1A describes a statistical model where $X$ causes $Y$ and an unobserved $E$ causes $Y$. In contrast, Figure 1B depicts an unspecified joint relationship between $X$ and $Y$. While under specific distributional assumptions (i.e., the distributions of $X$, $Y$, and $U$), we cannot statistically distinguish between the two statistical models, we can still generate the data based on the distinct processes.

The second way DAGs will help us in this class is to map out the algorithmic way we will proceed to solve the computational task; said differently, a DAG provides us with the algorithm we will use to generate the data. We algorithmically will proceed by generating all exogenous nodes (i.e., nodes with no edges pointing towards them). Returning to Figure 1A, we would first generate data for $X$ and $E$. Using this data, we will then generate the data for $Y$. For Figure 1B, as we will see, we can generate both $X$ and $Y$ from a joint distribution.

# 2 Generating Normally Distributed Data

In this section, we will discuss generating normally distributed data. This section ought to be review and may come off overly didactic, but we will quickly build on these fundamental concepts throughout the following labs. As a review, recall that the normal density is paramterizied with

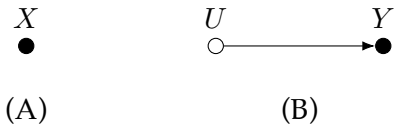


Figure 2: Two Data Generating Processes for Normal data

two parameters, a mean and a standard deviation:

$$f\left(x \mid \mu, \sigma\right) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\} \tag{1}$$

Note, throughout this course we will parameterize the normal distribution in terms of its standard deviation to map onto R.

## 2.1 Generating Data via the Built-in Function

To begin, consider the two DAGs in Figure 2 that describe data-generating processes. Figure 2A is a simple univariate problem where we only generate one variable, $X$. As the section title implies, we will assume $X$ follows a normal distribution with a mean representing the distribution's location and a standard deviation representing the scale.

Throughout this course, we will use a functional notation to provide an abstract representation of how we generate data from a graph. In addition, as we will discuss later this semester, this notation directly relates to the probabilistic models used to analyze multivariate data. Using a functional notation we write Figure 2A as follows.

$$f\left(X \mid \mu, \sigma\right) \Rightarrow \quad X \sim \mathcal{N}\left(\mu, \sigma\right) \tag{2}$$

The first part of Equation (2) represents the functional notation. We can read this as $X$ follows an arbitrary density (represented by $f$) conditional on $\mu$ and $\sigma$—i.e., the mean and standard deviation, respectively. The arrow ($\Rightarrow$) represents that this density is equivalent to $X$ being distributed as (i.e., '$\sim$' can be read as "distributed as") a normal distribution with a mean ($\mu$) and standard deviation ($\sigma$). Note that the functional notation is general, and we only obtain the normal distribution because assumed $X$ follows a normal distribution—i.e., Equation (1).

The primary function that generates normally distributed data in R is `rnorm()`:

```
rnorm(n, mean, sd)
```

Note, to see the manual page for the function type `?rnorm` into the console. The `rnorm()` function returns a one dimensional numeric vector of type double (i.e., double precision floating point numbers) with `n` normally distributed variables. In R, we will first generate 1000 normally distributed observations. As a reminder, follow the Style Guide with good commenting practices.

**In R:**

**(1)** Predefine the parameters for the normal distribution. Do this via a `list()` with each element's name representing the mean and standard deviation. This list should be named informative, referencing the variable $X$ (e.g., `p_x` for parameters of $X$). Set $\mu = 10$ and $\sigma = 5$.

**(2)** Define a global variable to represent the sample size we wish to draw. For this lab, we will use a sample size of $N = 1000$.

**(3)** Draw $1000$ observations from the normal distribution. Make sure to set the seed to `12345`. Note, you can use the `with()` command to simplify this code—e.g., `with(p_x, your_code_here)`.

**Answer:**

Create a `list()` with elements of the empirical mean and standard deviation from the randomly sampled observations. The first element should be labeled `mu` and the second element should be labeled `sigma`. Store this in your `answers` list under element `[[1]]`.

## 2.2 Generating Data via Transformation

Although `rnorm()` has arguments for mean and standard deviation, we can also transform the data ourselves. By transforming the data, we can intuitively understand how the mean and variance affect the distribution of values. Furthermore, data transformation will be necessary to generate data following complex models. The DAG in Figure 2B represents the data generation process via a transformation approach. Figure 2B illustrates that we can generate $Y$ as a consequence of an unobserved variable, $U$. First, let's specify the functional notation for $U$.

$$f\left(U\right) \Rightarrow \quad U \sim \mathcal{N}\left(0, 1\right) \tag{3}$$

There are other choices for $U$ (e.g., uniform distribution), but we have used a standard normal (i.e., mean of zero and variance of one) because it is straightforward. Tracing the DAG, the next step is to specify the functional notation for $Y$ with an arbitrary mean ($\mu$) and standard deviation ($\sigma$) conditional on $U$.

$$f\left(Y \mid U, \mu, \sigma\right) \Rightarrow \quad Y = \mu + \sigma U \tag{4}$$

As you might notice, this is a solution for the inverse of a $Z$-score transformation. Importantly, this illustrates the fact that what we have is a linear regression. The mean represents the intercept of our regression, and the standard deviation is the slope (i.e., the weight maintaining the relationship between $U$ and $Y$).

**In R:**

**(1)** Draw 1000 observations from a standard normal distribution and assign the result to an informative name. Make sure to reset the seed to `12345`.

**(2)** Using the parameters you have already defined, transform the drawn values by implementing Equation (4). Note, you can use the `with()` command to simplify this code—e.g., `with(p_x, your_code_here)`.

**Answer:**

Create a `list()` with elements of the empirical mean and standard deviation from the randomly generated $Y$ observations. The first element should be labeled `mu` and the second element should be labeled `sigma`. Store this in your `answers` list under element `[[2]]`. Next check if answer 1 and 2 are identical. Store the result from this check into `answers[[3]]` by creating a list with the first element labeled "identical". Note, this should be a single bool.

# 3 Generating Bivariate Normally Distributed Data

This section will discuss generating data from a bivariate normal distribution. While there are R packages to accomplish this, the goal is to lay the groundwork for the following labs which build on this fundamental process. First, let us look at the paramterization of the bivariate normal distribution. Recall, that five parameters characterize the distribution:

$$f\left(x, y \mid \mu_x, \mu_y, \sigma_x, \sigma_y, \rho\right) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2(1-\rho^2)}\left[\left(\frac{x-\mu_x}{\sigma_x}\right)^2 + \left(\frac{y-\mu_y}{\sigma_y}\right)^2 - 2\rho\left(\frac{x-\mu_x}{\sigma_x}\right)\left(\frac{y-\mu_y}{\sigma_y}\right)\right]\right\} \tag{5}$$

As an intuition, if we focus on the term in between the square brackets, we can see that the bivariate distribution is characterized by the sum of two $Z$-scores (as opposed to one in Equation 1) minus a term that represents the covariance between $X$ and $Y$. Conceptually, when $\rho = 0$, we can see that it completely drops out of the equation (and with some algebra, they can be factored out into two separate normal densities).

## 3.1 Generating Bivariate Data via Direct Cause

Let us reconsider the DAG in Figure 1B, where $X$ and $Y$ are correlated and (in our case) follow a bivariate normal distribution. Figure 3 illustrates two separate ways we can accomplish this computational task. Figure 3A should be familiar (because it is a duplicate of Figure 1A) and will be our starting point for generating bivariate normal data. Using the functional notation, I've written out the first two steps from Figure 3A.

$$\begin{aligned} f\left(X \mid \mu_x, \sigma_x\right) \Rightarrow &\quad X \sim \mathcal{N}\left(\mu_x, \sigma_x\right) \\ f\left(U\right) \Rightarrow &\quad U \sim \mathcal{N}\left(0, 1\right) \end{aligned} \tag{6}$$

The above distributions above are straightforward and already discussed in the previous section.

To generate $Y$ is more complicated because we need to consider including both variables with an edge pointing towards $Y$. In addition, we must maintain $Y$'s marginal mean ($\mu_y$) and standard

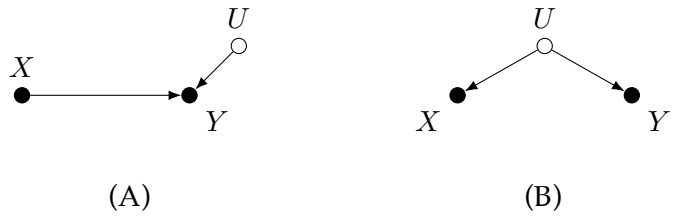


Figure 3: Two Data Generating Processes for Bivariate Normal

deviation ($\sigma_y$). Let us first focus on the standard deviation. We can decompose $Y$'s marginal ***variance*** based upon our model.

$$\text{Var}\left(Y\right) = \text{Var}\left(Y \mid X\right) + \text{Var}\left(Y \mid U\right) \tag{7}$$

The Var ( ) function represents a variance computation, which is formally defined via expectations as $\text{Var}\left(x\right) = \text{E}\left(\left[x - \text{E}\left(x\right)\right]^2\right)$. Conceptually, Equation (7) is saying that $Y$'s total variance is equal to the sum of two parts: (a) the variance conditional on knowing $X$ and (b) a variance conditional on knowing $U$.

If $X$ and $Y$ were standardized, the squared correlation ($\rho^2$) would define the variance conditional on knowing $X$. However, because they are not standardized, we must weigh this squared correlation by the ratio of $Y$ and $X$'s variances.

$$\begin{aligned}\text{Var}\left(Y \mid X\right) &= \rho^2 \left(\frac{\text{Var}\left(Y\right)}{\text{Var}\left(X\right)}\right) \\ &= \rho^2 \left(\frac{\sigma_y^2}{\sigma_x^2}\right)\end{aligned} \tag{8}$$

In contrast, the variance conditional on $U$ is the unexplained correlation (i.e., $1 - \rho^2$) weighted by the ratio of $Y$ and $U$'s variances.

$$\begin{aligned}\text{Var}\left(Y \mid U\right) &= \left(1 - \rho^2\right) \left(\frac{\text{Var}\left(Y\right)}{\text{Var}\left(U\right)}\right) \\ &= \left(1 - \rho^2\right) \left(\frac{\sigma_y^2}{1}\right) = \left(1 - \rho^2\right) \sigma_y^2\end{aligned} \tag{9}$$

Like how the square root of the variance of $Y$ weighted the $U$ in Equation (4), the square root of the conditional variances constitutes the weights for $X$ and $U$.

Now that we have derived the variance of $Y$, we only need to define the mean structure of $Y$. Instead of going through any derivation, we can remove the location of each variable (i.e., center each variable at its expectation/mean) and add $Y$'s mean to set the location of the distribution. Putting this together allows us to compute $Y$ with the following expression.

$$f\left(Y \mid X, U\right) \Rightarrow \quad Y = \mu_y + \left(\frac{\rho\sigma_y}{\sigma_x}\right)\left(X - \mu_x\right) + U\sigma_y\sqrt{1 - \rho^2} \tag{10}$$

Note, I've not included the parameters (i.e., $\mu_x$, $\sigma_x$) in the function as a simplification.

**In R:**

**(1)** Predefine the parameters for $Y$'s distribution. Do this via a `list()` with each element's name representing the mean and standard deviation. Set $\mu_y = 5$ and $\sigma_y = 7$.

**(2)** Define a variable to represent the correlation with $\rho = 0.3$.

**(3)** Set the seed to `12345`.

**(4)** Draw $1000$ observations from the normal distribution using $\mu_x = 10$ and $\sigma_x = 5$.

**(5)** Draw $1000$ observations from a standard normal distribution.

**(6)** Compute $Y$ by applying the computation described in Equation (10).

**(7)** Using the parameters you have already defined, transform the drawn values by implementing Equation (10).

**Answer:**

Create a `list()` with three elements labeled `x`, `y`, and `cor`. Elements `x` and `y` should follow the form of `list(mu = , sigma = )` and be set to the empirical mean and standard deviation from the randomly generated observations. The `cor` element should be the empirical correlation between the generated $X$ and $Y$. Store this in `answers[[4]]`.

### 3.2 Generating Bivariate Data via Joint Cause

Now that we have generated $X$ and $Y$ using Figure 3A, let us generate data via Figure 3B. This graph demonstrates what is sometimes called a phantom variable specification for a correlation. In essence, the correlation we see is a manifestation of a common cause between two variables. With linear relationships, the correlation we observe is equal to the sum of the two edges. Starting from the first node, we define the functional notation of $U$.

$$f(U) \Rightarrow \quad U \sim \mathcal{N}(0, 1) \tag{11}$$

The equation above is a straightforward application of everything we have already done for $U$.

The functional notation of $X$ and $Y$ are more tricky but almost identical. First, let us expand Figure 3B to include residuals for both $X$ and $Y$, denoted as $E_x$ and $E_y$, respectively.

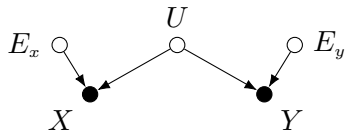


Figure 4: Representing Figure 3B with Residuals

The graph in Figure 4 demonstrates what we have already illustrated in the previous section. We can define most random processes in terms of residual structures; said differently, both graphs in Figure 2 generated identical data.

If we focus on only $X$, we can see that this problem is similar to the data generation process in Figure 4A. The critical difference is that the edge for $U \rightarrow X$ is equal to half the correlation ($\rho$);

thus, we apply Equation (10) using half the correlation for $U$ and $E_x$.

$$f\left(X \mid U, E_x\right) \Rightarrow \quad X = \mu_x + U\left(\sigma_x\sqrt{|\rho|}\right) + E_x\sigma_x\sqrt{1-|\rho|} \tag{12}$$

Notice that we have now taken the absolute value of $\rho$ in the equation. This is because $\rho$ ranges from $-1$ to $1$ and we cannot take the square root of a negative number. Finally, we can simplify the above equation to remove $E_x$ entirely:

$$f\left(X \mid U\right) \Rightarrow \quad X \sim \mathcal{N}\left(\mu_x + U\left(\sigma_x\sqrt{|\rho|}\right), \sigma_x\sqrt{1-|\rho|}\right) \tag{13}$$

Similarly, we can apply the same equation for $Y$ with one key difference. We must include the sign of the correlation to maintain the direction.

$$f\left(Y \mid U\right) \Rightarrow \quad Y \sim \mathcal{N}\left(\mu_y + \text{sgn}(\rho)\cdot U\left(\sigma_y\sqrt{|\rho|}\right), \sigma_y\sqrt{1-|\rho|}\right) \tag{14}$$

Note, $\text{sgn}(\rho) = \frac{\rho}{|\rho|}$, which returns a $1$ for positive $\rho$ or a $-1$ for negative $\rho$. In R, this function is called `sign()`.

**In R:**

**(1)** Set the seed to `12345`.

**(2)** Draw 1000 observations from a standard normal distribution.

**(3)** Draw 1000 observations from the distribution described in Equation (13)

**(4)** Draw 1000 observations from the distribution described in Equation (14)

**Answer:**

Create a `list()` with three elements labeled `x`, `y`, and `cor`. Elements `x` and `y` should follow the form of `list(mu = , sigma = )` and be set to the empirical mean and standard deviation from the randomly generated observations. The `cor` element should be the empirical correlation between the generated $X$ and $Y$. Store this in `answers[[5]]`. Next check if answer 4 and 5 are identical. Store the result from this check into `answers[[6]]` by creating a list with the first element labeled "identical". Note, this should be a single bool.

## 4 Constructing Our First Simulation

You may be surprised by the fact that the two graphs in Figure 3 do not produce the exact same data. This is because they are two different data generation processes with the same model expectations. Said differently, they are indistinguishable in terms of means and variances, but $X$ and $Y$ are a combination of different random processes.

To demonstrate that the two processes are indistinguishable, we will empirically verify that these processes produce the same population parameters in expectation. Note that this will be a crucial step when generating any data set, and we will be doing it throughout this semester. We will accomplish this via simulation (as the name of this course might imply).

### 4.1 Creating Functions

We first will define two functions to construct a simulation comparing the two data generation processes. This process may appear a bit tedious for what we are trying to accomplish, but it will serve you quite well as you start to implement and run more complex simulations.

**In R:**

Create two functions, one to generate from Figure 3A and one from Figure 3B. Both functions should have four inputs with the following form

```
function(n, p_x, p_y, cor)
```

and output a 2 column matrix (i.e., `cbind(x = x, y = y)`). As a reminder, follow the Style Guide with good commenting practices to document both functions' inputs and outputs.

### 4.2 Using the Replicate Function

A quick and concise way to run the same function multiple times is using the `replication()` function. The `replicate()` function belongs to a broader family of 'apply' functions that we will use throughout this course. There are three inputs to this function,

```
replicate(n, expr, simplify = "array")
```

where `n` is the number of replications, `expr` is an R expression that will be evaluated, and `simplify` defines the output from replicate. By default this is an array. We will discuss arrays more in subsequent labs.

**In R:**

**(1)** Set the seed to `12345`. Reuse the same parameters and sample size from the previous section.

**(2)** Use `replicate()` to generate 500 replications from your data generation function for Figure 3A.

**(3)** Use `replicate()` to generate 500 replications from your data generation function for Figure 3B.

**(4)** Use `dim()` to investigate the output dimensions from the previous steps. Is this what you expected?

**Answer:**

Create a `list()` with two elements labeled `dim_a` and `dim_b`. Store the appropriate dimensions into these two elements. Store the list in `answers[[7]]`.

### 4.3 Analyzing the Data with the Apply Function

The advantage of using something `replicate()` to generate our data is that we can then take its output and use other functions from the apply family. This will allow us to have compact code that

can quickly scale up when we introduce more complexities, parallelization, and other optimizations. Investigation of the dimensions of the results ought to make it clear how the data structure is set up. We have a three-dimensional array with the first dimension equal to the number of observations (1000), the second dimension the number of variables (2), and the third dimension equal to the number of replications (500).

First, we want to focus on what it would take to analyze a single data set. Recall that this simulation aims to compare the expectation of our sample parameters from the two data generation processes. Therefore, for each data set, we must obtain the sample estimates of all five parameters (two means, two standard deviations, and a correlation).

**In R:**

Create an analyze functions that takes one data set and outputs a one-dimensional vector (i.e., `c()`) with the following labels and ordering:

`m_x`, `m_y`, `s_x`, `s_y`, and `cor`.

As a reminder, follow the Style Guide with good commenting practices to document the function's input and output.

Now that we have created a function that analyzes a single data set, we can take that function and apply the function to each generated data. As one might guess, this is done via the `apply()` function.

```
apply(X, MARGIN, FUN, ..., simplify = TRUE)
```

Note, we can read the entire manual page for the function type `?apply` into the console, but let's briefly review the primary inputs. The `X` input is the array that we want to call the function to—i.e., our array that contains the data sets. `MARGIN` is the dimension that we want to apply the function across; said differently, we are subsetting the array across this dimension. For example, `MARGIN = 1` will be iterating over `X[i, , ]` and `MARGIN = 2` will be iterating over `X[, i, ]`. Finally, `FUN` is the function we want to call on across the dimension we defined.

**In R:**

**(1)** Use `apply()` to call the analysis function on the data generated for Figure 3A.

**(2)** Use `apply()` to call the analysis function on the data generated for Figure 3B.

**(3)** Use `dim()` to investigate the output dimensions from the previous steps. Is this what you expected?

**Answer:**

- Create a `list()` with two elements labeled `dim_a` and `dim_b`. Store the appropriate dimensions into these two elements. Store the list in `answers[[8]]`.
- Create a `list()` with two elements labeled `a` and `b`. Store the average of each parameter across all 500 replications. Look at the results. Do they make sense? How similar are they to the population parameters? Store the list in `answers[[9]]`.

  ***Hint:*** This should be accomplished by using `rowMeans()`.

# 5 Conclusion

The purpose of this lab was to provide you with an introduction to several key concepts that we will use throughout this semester. We discussed using directed acyclic graphs to represent data generating processes and statistical models. We then took these graphs and constructed equations using the abstract functional notation. These two concepts will be paramount throughout this course as we discuss more complicated data generation processes and statistical models.

Using DAGs and functional notation, we discussed several ways to simulate normally distributed data. We illustrated how using an unobserved residual can produce the equivalent data as drawing directly from the distribution of interest. When looking at bivariate normally distributed data, we illustrated how we could generate data from this distribution via a directed cause model ($X \rightarrow Y$) or a joint cause model ($X \leftarrow U \rightarrow Y$). We demonstrated that these data generation processes are not equivalent but still produce identical expectations.

In R, we introduced taking equations and translating them into runnable code. We demonstrated the use of lists and arrays to quickly store multiple data sets and results. We showed that using these data structures allows us to think of a single data set, set up a function to analyze it, and use the Apply Family of R functions to run the operation on each data set efficiently.

**In R:**

Make sure to run the final line of code to produce the answers for this Lab. This function writes out an R binary file into the "answers" folder.

**Answer:**

Create a commit including your answers and R script. Make sure to push this commit to GitHub to be graded. GitHub will automatically check the solutions you provide for correctness.

***Reminder:*** GitHub will check the solutions based on exact file matching; as a result, be sure to follow the directions in constructing answers and setting random number generator seeding.