# **Lab 9 – Uniform and normal probability simulations; more sampling distributions of the mean**

In Lab 4, we used the sample() function in order to perform simulations. However, many experiments are modelled by continuous probability distributions such as the ones we have encountered in class – eg - uniform, normal, and exponential. Therefore, in simulating such experiments, it is useful to be able to generate sets of values that follow these kinds of distributions. For instance, if we want to simulate an experiment in which outcomes can be modelled by a normally distributed random variable with mean 10 and standard deviation 2, then if we run the experiment 10000 times, we would want most of the results to be fairly close to 10, and the values to be distributed normally.

As in Lab 4, we are going to simulate experiments and estimate probabilities using the relative frequency approach to probability. That is, we will simulate an experiment *n* times, and count the number of “successes” *k*. This will allow us to estimate the probability of the event as *k.*

We will also revisit an earlier question about the sampling distribution of the mean. In Lab 6, we looked at samples generated from a discrete, uniform distribution: the population of die rolls. Now we will look at samples from a population that follows a normal distribution.

**To submit: answers to all numbered questions. When the question asks you to write code, submit the code in the Word document as part of your answer. Also submit a single .R file that contains all of your code.**

# Experiment 1: Waiting for a bus (uniform distribution)

In our first experiment, we imagine a person waiting for a bus that comes very reliably every 20 minutes. (This ideal situation is not very realistic; we will refine the model soon.) However, there’s a problem: the person waiting does not know the bus’s schedule! The person may have been lucky and arrived just before the bus came. Or, they may have just missed the last bus and will have to wait nearly 20 minutes for the next one. The amount of time the person will be waiting before the bus arrives can be modelled by a continuous uniform variable with minimum of 0 and maximum 20.

We can simulate a single person waiting for bus with the **runif()** function.

> runif(n=1, min=0, max=20)

[1] 3.7336

Here, my person was fairly lucky and only had to wait 3.7336 minutes for a bus.

Note that R is rounding the result to 5 digits. We can change the display by using the **options()** command:

> options(digits=20)

> runif(n=1, min=0, max=20)

[1] 14.162643705494702

Technically, the **runif()** function is discrete and not continuous, because R (like all software) can only store finitely many digits. However, it’s pretty close and we can treat it as continuous for our purposes.

We can simulate multiple bus-waiters by changing the first argument to **n**. This generates a list of amounts of times that **n** people waited for the bus.

1. Generate appropriately-labelled histograms that give the frequency of waiting times for n=100,1000, and 10000 people who are waiting for a bus that comes every 20 minutes. No need to get fancy with bins. Do the distributions look uniform? That is – when 100 people show up to catch the bus, were there approximately equal numbers of people waiting “short” amounts of time as “medium” and “long” amounts of time? How about when there are 10000 people?
2. Using n=10000, find the proportion of people who wait less than 10 minutes for a bus. Does your answer seem reasonable? Explain.

# Experiment 2: Quality control (normal distribution)

In mass-production, companies aim to produce large quantities of identical goods. In practice, the goods are not completely identical, and the variation is typically modelled by a normal distribution.

For example, a battery manufacturer produces thousands of 9V batteries. Ideally, each of the batteries should have a measured voltage of exactly 9.0000000 V. In practice, however, there is some variation in the measured voltages. The true measured voltages of 9V batteries manufactured by this company follow a normal distribution with mean 9.01 V and standard deviation 0.05 V.

We can use the **rnorm()** command to generate **n** values to that follow a normal distribution with given mean and standard deviation. For instance,

> rnorm(n, mean=mu, sd=sigma)

gives n normally-distributed values with mean **mu** and standard deviation **sigma**.

1. Suppose the battery manufacturer will ship batteries whose measured voltages are between 8.9V and 9.1V. Give a command that simulates selecting **n** batteries and returns the proportion that can be shipped. Give your results for n=100, 1000, and 10000.

We can also work with normal distributions directly, without doing simulations. This is an alternative to finding z-scores and using the z-table. The **pnorm()** command finds cumulative normal probabilities. For instance,

> pnorm(q, mean=mu, sd=sigma)

returns the probability that a value in a normally-distributed population with mean **mu** and standard deviation **sigma** is less than **q**.

1. Without doing a simulation, find the proportion of batteries whose measured voltages are between 8.9V and 9.1V. Now use your Z-table to answer this question. How do your numbers compare to one another, and to the values you got for Question 3?

We can also find measurements corresponding to given proportions. For instance,

> qnorm(p, mean=mu, sd=sigma)

returns the **p**th percentile in a normally-distributed population.

1. Find the voltage that is larger than 80% of the batteries’ measured voltages.
2. Find the range of voltages that contains the middle 50% of voltages.

You can check your answers to Questions 5 and 6 by using the Z-table.

## Distribution of sample means in a normally distributed population

Let’s revisit the question of finding distributions of sample means. This time, we will investigate the means of samples that are drawn from a normally-distributed population.

1. Create a function called **NormalMeans(n,m)** that simulates taking **n** samples of size **m** from normally-distributed populations with mean 0 and standard deviation 1. Like the **DiceMeans(n,m)** function you created in Lab 6, this new function will compute (and keep track of) the sample mean for each of those **n** samples.Your function should return the following:

* The mean of the **n** sample means
* The standard deviation of the **n** sample means
* An appropriately-named **histogram** of the **n** sample means (use the default bins)

Run your function for **n=10000** and for the following values of **m**: 1, 2, 10, 50, 100.

Submit five outputs (five histograms, each with the means and standard deviations listed below). How do the means and standard deviations compare as **m** increases? How do the shapes of the graphs compare to one another?