**Lab 6 – Poisson, Geometric, Uniform, and Normal Distributions**

**To submit before your next lab: answers to all numbered questions. When the question asks you to generate output in R, such as a graph, submit the output in the Word document as part of your answer. Make sure all of your graphs have clear and descriptive labels. Also submit all commands and/or functions you used to generate your output, and submit a single .R file containing all of the scripts you wrote for this lab.**

In this lab, we will simulate experiments that are modeled by the Poisson, geometric, uniform, and normal distributions.

# Example 1: geometric probabilities

From class, we saw that we can model trials that we repeat until we obtain a success with the geometric distribution. The **dgeom()** function returns the probability that an experiment with probability **prob** of success will require **x** trials **before** obtaining a success. There are a few related functions, and their syntax should be pretty familiar to you by now!

The Geometric Distribution

**Description**

Density, distribution function, quantile function and random generation for the geometric distribution with parameter prob.

**Usage**

dgeom(x, prob, log = FALSE)

pgeom(q, prob, lower.tail = TRUE, log.p = FALSE)

qgeom(p, prob, lower.tail = TRUE, log.p = FALSE)

rgeom(n, prob)

**Arguments**

|  |  |
| --- | --- |
| x, q | vector of quantiles representing the number of failures in a sequence of Bernoulli trials before success occurs. |
| p | vector of probabilities. |
| n | number of observations. If length(n) > 1, the length is taken to be the number required. |
| prob | probability of success in each trial. 0 < prob <= 1. |
| log, log.p | logical; if TRUE, probabilities p are given as log(p). |
| lower.tail | logical; if TRUE (default), probabilities are *P[X ≤ x]*, otherwise, *P[X > x]*. |

One major difference between the geometric distribution and the binomial and hypergeometric distributions is that there is no upper bound on the number of nonzero probabilities. For example, if you roll a die 10 times, the maximum number of 3’s you can get is 10. If you draw 8 cards without replacement, you will get at most 4 aces. But if you purchase a lottery ticket every week until you get a winning one – an experiment modelled by the geometric distribution – you may theoretically be purchasing lottery tickets forever! However the probability of having to purchase a “very large” number of lottery tickets is very small and can be considered to be zero for all practical purposes.

1. A student decides to purchase lottery tickets until she wins a prize. Suppose the probability of winning a prize is 1/5. (Most of these prizes are quite small!). Use the **dgeom()** function with **x** = c(0:20) to find the exact probability distribution for the number of tickets the student will buy before getting a winner. (You may want to compare at least one of the probabilities to the one you get by using the formula from class.) Give the probability distribution as both a table and a bar plot.

> results = dgeom(c(0:20), 1/5)

> table(results)

results

0.0023058430092137 0.00288230376151712 0.0036028797018964

1 1 1

0.0045035996273705 0.00562949953421312 0.00703687441776641

1 1 1

0.00879609302220801 0.01099511627776 0.0137438953472

1 1 1

0.017179869184 0.02147483648 0.0268435456

1 1 1

0.033554432 0.04194304 0.0524288

1 1 1

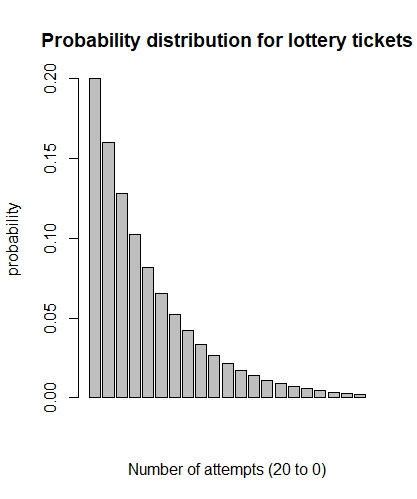
0.065536 0.08192 0.1024

1 1 1

0.128 0.16 0.2

1 1 1

> barplot(results, main="Probability distribution for lottery tickets", xlab="Number of attempts (20 to 0)", ylab="probability")



1. Approximate the probability distribution for the number of lottery tickets the student must buy before obtaining a winner in two ways:
2. By writing a function that uses the **sample()** function to simulate **n** students who each purchase lottery ticketsuntil a winner is obtained. Your function should output a table giving the relative frequencies, as well as a bar plot. Run your function for **n**=10000 and provide a table as well as a bar plot with clear and descriptive labels.

> Lottery(10000)

arr

1 2 3 4 5 6 7 8 9 10 11 12 13

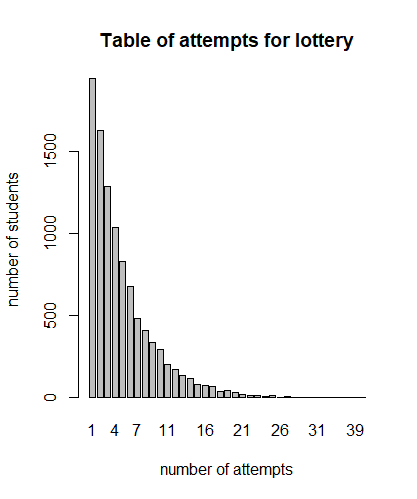
1940 1623 1285 1035 830 676 484 412 338 293 205 175 134

14 15 16 17 18 19 20 21 22 23 24 25 26

120 82 76 68 39 42 35 19 14 17 11 12 5

27 28 29 30 31 32 33 34 35 39 52

6 5 5 1 4 3 1 1 1 2 1



1. By writing a function that simulates **n** people purchasing lottery tickets until getting a winner, using the **rgeom()** function. As before, your function should output a table giving the relative frequencies, as well as a bar plot. Run your function for **n**=10000 and provide a table and a bar plot.

> Lottery2(10000)

arr

0 1 2 3 4 5 6 7 8 9 10 11 12

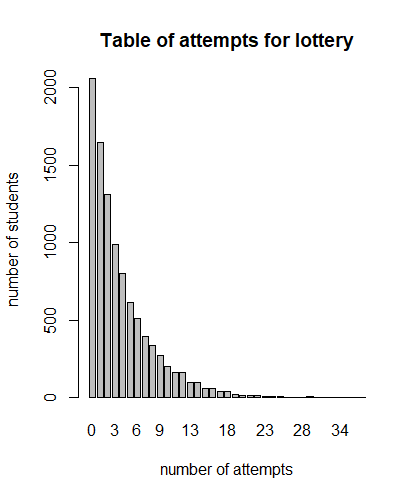
2057 1646 1314 988 804 616 514 393 336 275 204 165 161

13 14 15 16 17 18 19 20 21 22 23 24 25

102 96 63 61 39 43 20 17 14 17 7 8 9

26 27 28 29 30 31 33 34 37 38 39

3 3 4 10 1 1 4 2 1 1 1



1. How do the probabilities obtained from your simulations compare to the exact probabilities?

The results from both sample and rgeom functions are correlated with the probability result. The graphs are not 1-to-1 due to randomization, but the probability pattern persist.

# Example 2: Poisson probabilities

We saw in class that experiments in which we are interested in the number of occurrences of an event within a given interval can be modelled with Poisson probabilities. The **dpois()** function and related functions are helpful for modelling these experiments. From the help file:

The Poisson Distribution

**Description**

Density, distribution function, quantile function and random generation for the Poisson distribution with parameter lambda.

**Usage**

dpois(x, lambda, log = FALSE)

ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)

qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)

rpois(n, lambda)

**Arguments**

|  |  |
| --- | --- |
| x | vector of (non-negative integer) quantiles. |
| q | vector of quantiles. |
| p | vector of probabilities. |
| n | number of random values to return. |
| lambda | vector of (non-negative) means. |
| log, log.p | logical; if TRUE, probabilities p are given as log(p). |
| lower.tail | logical; if TRUE (default), probabilities are *P[X ≤ x]*, otherwise, *P[X > x]*. |

Like the geometric distribution, there is no theoretical upper bound on the number of nonzero probabilities. Unlike the other experiments we have modelled so far, we will not be able to simulate these ones using the **sample**() function so we will compare the exact probabilities to ones that are generated by the **rpois()** function.

The following three questions pertain to a factory that produces fibre optic cables that have an average of 0.75 flaws per meter.

1. Give an exact probability distribution for **x** = c(0:5) for the number of flaws in one metre of cable as both a table and a bar plot. Make sure your bar plot is appropriately labeled. You may want to refer to Lab 5 for a reminder how to get the labels on the x-axis. (You may want to compare at least one of the probabilities to the one you get by using the formula.)

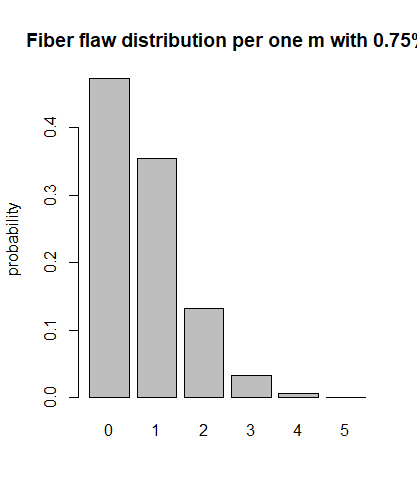
> dpois(c(0:5), 0.75)

[1] 0.4723665527 0.3542749146 0.1328530930 0.0332132732

[5] 0.0062274887 0.0009341233

> cables = dpois(c(0:5), 0.75)

> barplot(cables, main="Fiber flaw distribution per one m with 0.75%", names.arg =c(0:5), ylab= "probability")



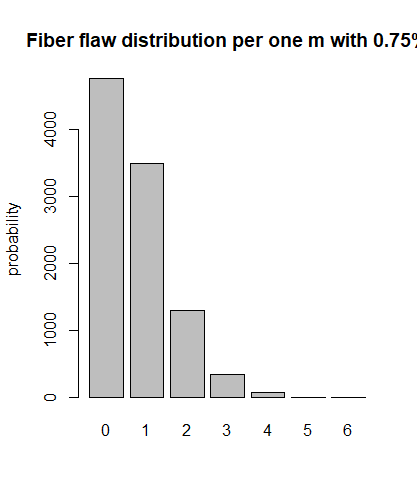
1. Write a function that simulates selecting **n** meter-long fibre optic cables and counting the number of flaws on each. As before, your function should output a table giving the relative frequencies, as well as a table. Run your function for **n**=10000 and provide a table and a bar plot.

> Cable(10000)

arr

0 1 2 3 4 5 6

4765 3493 1312 347 74 7 2



1. How do the exact values from Question 4 compare to the values obtained from the simulation you wrote for Question 5?

The exact values are perfect! Stats is magical! I love R Studio! On a serious note, our values have almost perfect comparison.

# Example 3: Continuous uniform probabilities

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 later.) 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.

By now you should have a decent feel for how the functions for the various distributions in R work. For instance, the function **rdisttype()** allows us to randomly generate random numbers that follow a distribution of type **disttype**. The **runif()** function does this for uniform distributions. 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 **n**. This generates a list of amounts of times that **n** people waited for the bus.

> options(digits=3)

> runif(n=10, min=0, max=20)

[1] 16.83 6.37 3.14 12.56 19.30 1.30 5.13 16.05 6.64 1.36

Unlike with discrete distributions, it does not make sense to create bar plots for continuous distributions because it is likely that if we generate **n** random numbers on a continuous scale, they will all be distinct. Therefore, we create histograms instead.

Generate appropriately-labelled histograms that give the frequency of waiting times for 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?

1. Using n=10000, find the proportion of people who wait less than 10 minutes for a bus. Does your answer seem reasonable? Explain.

> Bus(10000)

[1] 0.5032

My answer seems reasonable enough. 10 minutes is 50% of 20, and since we are dealing with uniform distribution, our probability must be ~(10/20) %.

# Example 4: normal probabilities

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.

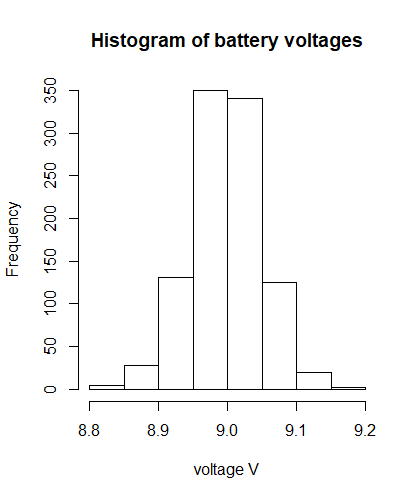
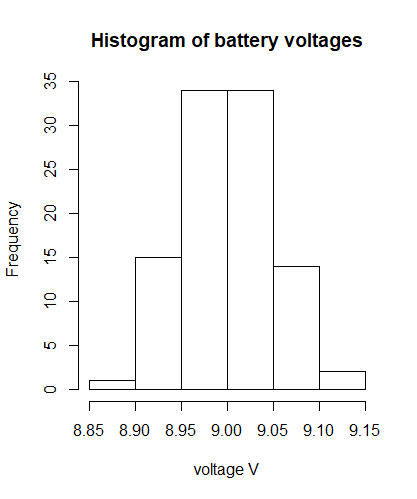
For example, a battery manufacturer produces thousands of 9V batteries. Ideally, each of the batteries should have a measured voltage of exactly 9.0000000 V. In practice, however, there is some variation in the measured voltages. Suppose the true measured voltages of 9V batteries manufactured by this company follow a normal distribution with mean 9.00 V and standard deviation 0.05 V.

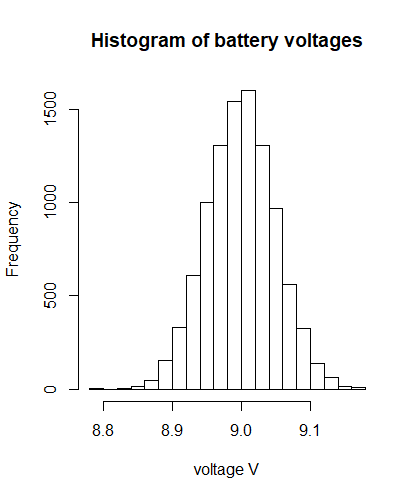
We can use the **rnorm()** command to generate **n** values to that follow a normal distribution with given mean and standard deviation. For instance,

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

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

1. Generate appropriately-labelled histograms that give the frequency battery voltages for n=100,1000, and 10000 randomly-generated batteries whose voltages are distributed normally with mean 9.00 V and standard deviation 0.05 V. No need to get fancy with bins. Do the distributions look normal?





The distribution looks normal for all 3 tests, although for n=10000 it displays the best normal distribution.

1. Suppose the battery manufacturer will ship batteries whose measured voltages are between 8.9V and 9.1V. Give a command that returns the proportion of batteries out of **n** that can be shipped, and give your results for n=100, 1000, and 10000. Do these agree with the Empirical Rule? Explain.

mean 9.00 V and standard deviation 0.05 V

By Empirical Rule, 95% should fall between +/- 2sd: between 8.9V and 9.1V.

Battery2 = function (n) {

arr = rnorm(n, mean=9.0, sd=0.05);

return (sum(arr > 8.9 & arr < 9.1)/n);

}

> Battery2(100)

[1] 0.95

> Battery2(1000)

[1] 0.957

> Battery2(10000)

[1] 0.9565

Our results agree with the Empirical Rule, because we are having a normal distribution and our randomized values are following the random distribution rule.