# Lab 5 – Binomial distribution; distributions of sample means

This lab has two components. The first builds on our last lab, in which we simulated various simple experiments: flipping a fair coin, rolling a fair die, and drawing cards from a deck. In lecture, we saw that certain kinds of experiments could be modelled by special distributions. So far in class we encountered one of these: the binomial distribution. We will now see how to model experiments described by those distributions and compute their associated probabilities in R.

In the second part, we are going to be investigating the distributions of **means** of samples taken from a population. For instance, suppose we take the means and standard deviations of **n** samples of a population, each of size **m**, and then looked at the distribution of those **n** sample means. We will be adapting yourfunctions from Lab 4 for this purpose.

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

## Binomial probabilities

We can calculate binomial probabilities in R using the **dbinom** function. The **dbinom** function is one of several in a family of functions involving binomial probabilities. Here is what the Help file tells us about these functions:

**Usage**

dbinom(x, size, prob, log = FALSE)

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

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

rbinom(n, size, prob)

**Arguments**

|  |  |
| --- | --- |
| x, q | vector of quantiles. |
| p | vector of probabilities. |
| n | number of observations. If length(n) > 1, the length is taken to be the number required. |
| size | number of trials (zero or more). |
| prob | probability of success on each trial. |
| 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]*. |

If we wish to simulate rolling ten dice and finding the probability of obtaining one 3, **x** would equal 1, **size** would equal 10, and **prob** would equal 1/6.

> dbinom(1, 10, 1/6)

[1] 0.3230112

This tells us that the probability of obtaining a single 3 when we roll ten dice is 0.3230112. Does this agree with those formula from class? (Take a minute to check – this is good practice for your next quiz.)

Notice that in the help file, we could also have entered a vector of quantiles instead of just a single value for **x**. If we enter the vector c(0:10), then we will obtain a table that gives the probabilities of obtaining **x** heads, for x=0,1,2,…10.

If we set

> options(digits=3)

then we obtain the following probability distribution for the number of 3’s obtained when ten fair dice are rolled.

> dbinom(c(0:10), 10, 1/6)

[1] 1.62e-01 3.23e-01 2.91e-01 1.55e-01 5.43e-02 1.30e-02 2.17e-03 2.48e-04 1.86e-05 8.27e-07 1.65e-08

Or in a nicer format:

> cbind(dbinom(c(0:10), 10, 1/6))

[,1]

[1,] 1.62e-01

[2,] 3.23e-01

[3,] 2.91e-01

[4,] 1.55e-01

[5,] 5.43e-02

[6,] 1.30e-02

[7,] 2.17e-03

[8,] 2.48e-04

[9,] 1.86e-05

[10,] 8.27e-07

[11,] 1.65e-08

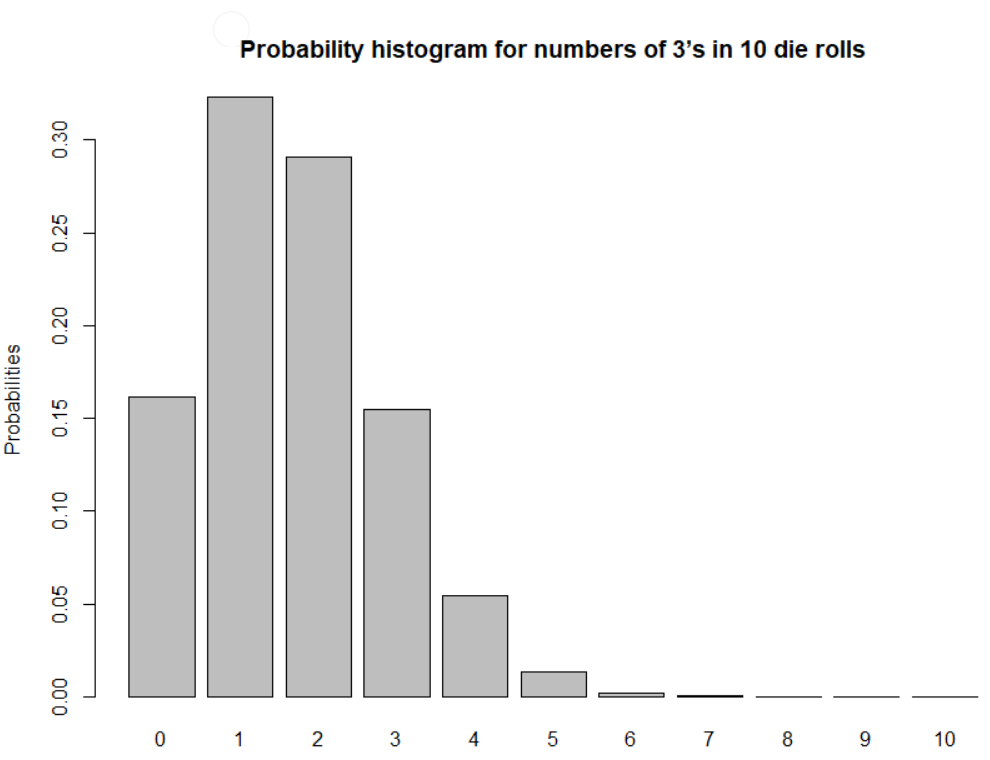
(Notice that the rows are indexed from 1 to 11, rather than the more useful indexing of 0 to 10. We can stick in a column that gives us the actual values of **x**, which run from 0 to 10:)

|  |
| --- |
| > cbind(c(0:10), dbinom(c(0:10), 10, 1/6))  [,1] [,2]  [1,] 0 1.62e-01  [2,] 1 3.23e-01  [3,] 2 2.91e-01  [4,] 3 1.55e-01  [5,] 4 5.43e-02  [6,] 5 1.30e-02  [7,] 6 2.17e-03  [8,] 7 2.48e-04  [9,] 8 1.86e-05  [10,] 9 8.27e-07  [11,] 10 1.65e-08 |
|  |
|  |

This tells us that, eg, the probability of getting two 3’s when we roll ten dice is 0.291.

And, of course, we can create a probability histogram for this results using the **barplot()** command. Note that we need to use the **names.arg** argument to get labels for each of the bars:

> barplot(dbinom(c(0:10), 10, 1/6), names.arg =c(0:10), ylab="Probabilities", main="Probability histogram for numbers of 3’s in 10 die rolls")



The **pbinom()** function allows us to compute cumulative probabilities. The following command will find the probability of obtaining two *or fewer* three’s when rolling a fair die ten times:

> pbinom(2,10,1/6)

[1] 0.7752268

This is particularly useful when we have binomial experiments with very large numbers of trials. questions which would be almost impossible to answer by hand. Example: We toss a fair coin 200 times. What is the probability of getting 90 heads or fewer? To do this by hand we would need to add the probability of getting 0 head + probability of getting 1 head + … + probability of getting 90 heads. Each of these would use the formula for the Binomial distribution. This would be really long. R allows us to find this value with just one command:

> pbinom(90,200,1/2)

[1] 0.08948202

1. During one stage in the manufacture of integrated circuits, a coating must be applied. Suppose that it is known that one third of chips do not receive a thick enough coating. 300 chips are randomly selected for testing. Give the commands, along with your output, to compute the following probabilities (answers are in brackets):
   1. Exactly 100 do not receive a thick enough coating (ans: 0.04881. Note: you can check this one on your calculator using the formulas from lecture)

Dbinom(100,300,1/3)

* 1. 100 or fewer do not receive a thick enough coating (ans: 0.5271)

Pbinom(100,300,1/3)

* 1. Fewer than 100 do not receive a thick enough coating (ans: 0.4783)

“More than 200 receive a thick enough coating”

“At least 200 receive a thick enough coating”

“100% - 200 or fewer receive a think enough coating”

1-pbinom(200,300,2/3)

* 1. At least 110 do not receive a thick enough coating (ans: 0.1228)

“190 or fewer receive a thick enough coating”

Pbinom(190,300,2/3)

* 1. Between 90 and 110 (inclusive) do not receive a thick enough coating (ans: 0.8017)

“Fewer than 110 do not receive a thick enough coating – fewer than 90 do not receive a thick enough coating”

“100% - 190 or fewer receive a thick enough coating – (100% - 210 or fewer receive a thick enough coating)”

1-pbinom(190,300,2/3)-(1-pbinom(210,300,2/3)) + dbinom(190,300,2/3)

R also allows us to generate random numbers that follow binomial (and other kinds of) distributions. For example, if we wish to simulate rolling a die 10 times and counting the number of 3’s, we can model this by generating random numbers that follow a binomial distribution with n=10 and p=1/6. (We consider a 3 to be a success.)

> rbinom(1, 10, 1/6)  
[1] 2

This result tells us that when we simulated rolling ten dice, we had two successes – ie, two of the dice came up as 3’s.

We could obtain a good approximation of the distribution of 3’s by running this simulation many times, say 10000. (R is a bit confusing here – we would assign the value 10 to the parameter **size**, and the number of simulations (in this case 10000) would be **n**).

We don’t want R to display 10000 values in the console, so here is a command that simulates rolling a set of 10 dice 20 times, and returns the number of 3’s for each of those 20 experiments:

> numthrees=rbinom(20, 10, 1/6)  
> numthrees

[1] 0 0 0 1 0 2 4 1 0 2 2 1 3 2 1 1 1 4 1 1

It looks like most of the time, when we roll ten dice, we get zero, one, or two 3’s. (Does this seem right?) Less commonly, we get three or four 3’s.

We can organize those results in a table:

> table(numthrees)  
numthrees

0 1 2 3 4

5 8 4 1 2

Alternatively, we can display the table vertically:

> cbind(table(numthrees))

[,1]

0 5

1 8

2 4

3 1

4 2

We can also convert the number of 3’s to relative frequencies by dividing the right hand side of our table by the total number of die rolls (in this case 20):

> table(numthrees)/20

numthrees

0 1 2 3 4

* 1. 0.40 0.20 0.05 0.10

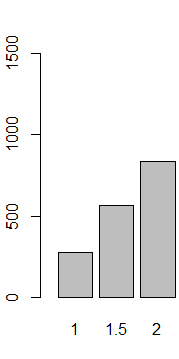
1. Approximate the probability distribution for the number of 3’s obtained when rolling 10 dice with relative frequency distributions in two different ways, as follows:  
   1. By writing a function that simulates rolling **m** dice **n** times, using the **sample()** function and techniques from Lab 4. (You can adapt one of the functions you wrote from Lab 4.) Your function should output a table giving the relative frequencies of obtaining different numbers of 3’s, as well as a bar plot. Run your function for **m**=10, **n**=10000 and provide a table and a bar plot.
   2. By writing a function that simulates rolling **m** dice **n** times, using the **rbinom()** function. As before, your function should output a table giving the relative frequencies, as well as a bar plot. Run your function for **m**=10, **n**=10000 and provide a table and a bar plot.

Note: both of your bar plots should look very similar to the exact probability distribution obtained

with the **dbinom()** function. If yours look significantly different, there is something wrong with your code.

## Distribution of sample means in a uniformly distributed population

We are going look at the distribution of sample means when the underlying population is uniform. A good example of such a population is die rolls: if you roll a fair die many times, the results 1, 2, 3, 4, 5, and 6 should occur with roughly equal frequency. (You saw this result in Lab 4.) We will now look at what happens when we take many samples of size **m** from a uniformly-distributed population and look at the means and standard deviations of those means. We can imagine this as rolling **m** dice many times, and looking at the means and standard deviations of the results of those dice.

1. Get out the script you created in Lab 4. Create a new function called **DiceMeans(n,m)** that will be based on your **RollSomeDice(n,m)** function. Your **DiceMeans(n,m)** function will simulate rolling **m** dice **n** times, and will compute (and keep track of) the sample mean for each of those **n** die rolls.Your function should return the following:

* The mean of the **n** sample means
* The standard deviation of the **n** sample means
* A bar plot of the **n** sample means

Note: The title of your bar plot should be “Means for **n** rolls of **m** dice”, where **n** and **m** are the actual values of **n** and **m** you entered. Choose your axis labels appropriately.

Run your function for **n=10000** and for the following values of **m**: 1 (ie, roll a single die 10000 times), 2 (roll 2 dice 10000 times and generate a list of the 10000 sums), 10, 50, 100. The left portion of the output for m=2 is shown to the right – yours should look similar.

Submit five outputs (bar plots, with the corresponding means and standard deviations pasted below each plot). How do the means and standard deviations compare as **m** increases? How do the shapes of the graphs compare?