**Lab 5 – Binomial and hypergeometric probability 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. Note: you have two weeks to complete this lab but it is recommended you complete as much as possible, and especially questions 1 and 4, before your midterm, as your midterm may contain “calculator-friendly” versions of these types of questions.**

In our last lab, 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 two of these: binomial, and hypergeometric. We will now see how to model experiments described by those distributions and compute their associated probabilities in R.

# Example 1: 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 the formula from class? (Take a minute to check – this is good practice for your midterm.)

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

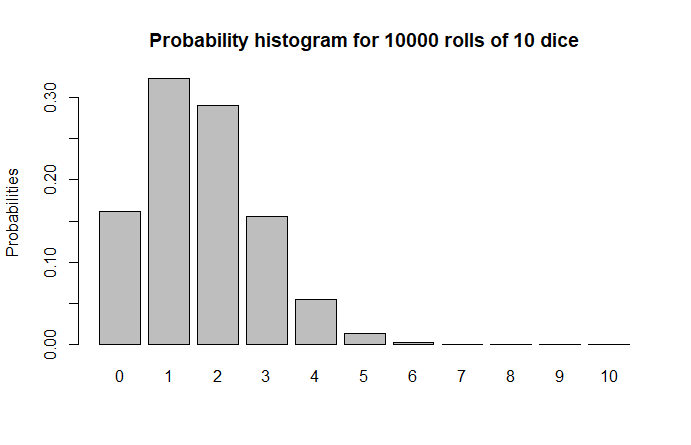
(Notice that the rows are indexed from 1 to 11, rather than the more useful indexing of 0 to 10. We can stick in a column that gives us the actual values of **x**, which run from 0 to 10:)

|  |
| --- |
| > 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 0.0291.

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 10000 rolls of 10 dice")



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] 0.04881277

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

> pbinom(100, 300, 1/3)

[1] 0.5271102

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

> pbinom(99, 300, 1/3)

[1] 0.4782974

* 1. At least 110 do not receive a thick enough coating (ans: 0.1229)

> pbinom(190, 300, 2/3)

[1] 0.1227525

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

> pbinom(110, 300, 1/3) - pbinom(89, 300, 1/3)

[1] 0.8016926

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

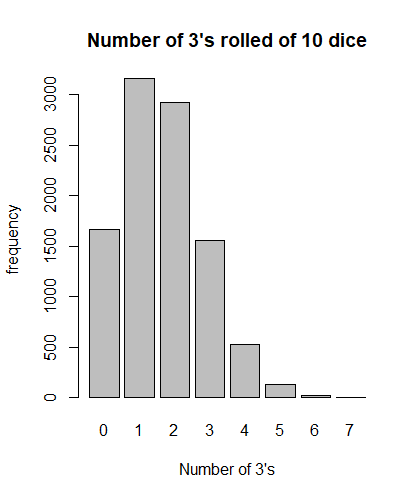
1. Approximate the probability distribution for the number of 3’s obtained when rolling 10 dice with relative frequency distributions in two different ways, as follows:
   1. By writing a function that simulates rolling **m** dice **n** times, using the **sample()** function and techniques from Lab 4. (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.

> Dice(10, 10000)

arr

0 1 2 3 4 5 6 7

1663 3161 2923 1559 531 135 23 5



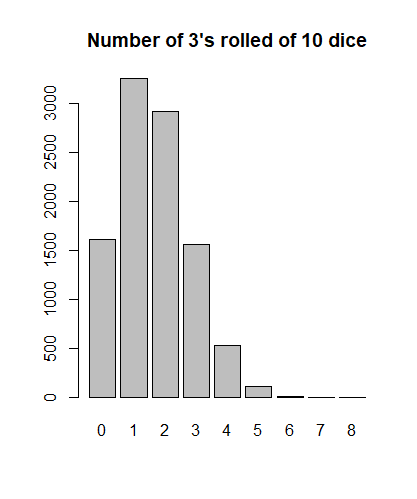
* 1. By writing a function that simulates rolling **m** dice **n** times, using the **rbinom()** function. As before, your function should output a table giving the relative frequencies, as well as a bar plot. Run your function for **m**=10, **n**=10000 and provide a table and a bar plot.

> Dice2(10, 10000)

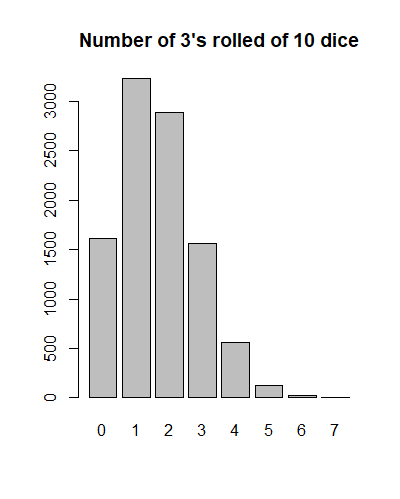
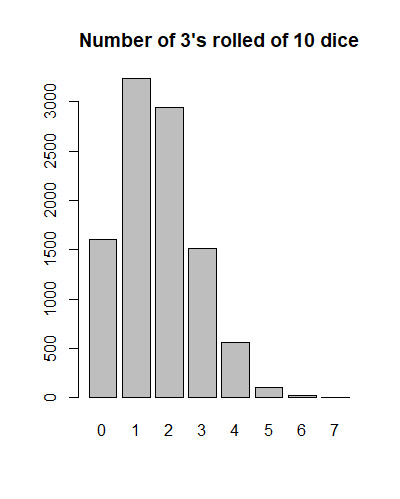
arr

0 1 2 3 4 5 6 7 8

1610 3248 2914 1564 529 113 18 3 1



1. How do the tables and bar plot from parts (a) and (b) compare to the exact probabilities obtained with the **dbinom()** function?



The graphs are identical, so we can say that our sample function was working well. From the coding perspective, it was much longer and harder to implement compared to dbinom.

# Example 2: hypergeometric probabilities

We saw in class that sampling with replacement can be modelled by hypergeometric distributions. R provides a family of functions involving the hypergeometric distribution, similar in syntax and usage to the family of functions we saw for binomial distributions. The **dhyper()** function computes hypergeometric probabilities directly. We can look it up, along with the other functions in the family, in the help file. Note that the help file gives a very concrete example involving drawing balls from an urn, but it can be generalized to any situation that involves sampling without replacement. (We can ignore the **log** argument.)

The Hypergeometric Distribution

**Description**

Density, distribution function, quantile function and random generation for the hypergeometric distribution.

**Usage**

dhyper(x, m, n, k, log = FALSE)

phyper(q, m, n, k, lower.tail = TRUE, log.p = FALSE)

qhyper(p, m, n, k, lower.tail = TRUE, log.p = FALSE)

rhyper(nn, m, n, k)

**Arguments**

|  |  |
| --- | --- |
| x, q | vector of quantiles representing the number of white balls drawn without replacement from an urn which contains both black and white balls. |
| m | the number of white balls in the urn. |
| n | the number of black balls in the urn. |
| k | the number of balls drawn from the urn. |
| p | probability, it must be between 0 and 1. |
| nn | number of observations. If length(nn) > 1, the length is taken to be the number required. |
| 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]*. |

The **dhyper()** and **phyper()** functions are analogous to the **dbinom()** and **pbinom()** functions. For instance, we can find the probability of getting exactly two jacks when drawing ten cards from a standard 52-card deck:

> dhyper(2,4,48,10)

[1] 0.1431157

We can also find the probability of drawing two or fewer jacks when drawing ten cards without replacement from a standard 52-card deck:

> phyper(2,4,48, 10)

[1] 0.9806076

1. During one stage in the manufacture of integrated circuits, a coating must be applied. Suppose that in a batch of 999 chips, 333 did not receive a thick enough coating. 300 of the 999 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: 0.05835. Note: you can check this one on your calculator using the formulas from lecture)

> dhyper(100, 333, 666, 300)

[1] 0.05834763

* 1. 100 or fewer do not receive a thick enough coating (ans: 0.5305)

> phyper(100, 333, 666, 300)

[1] 0.5304624

* 1. Fewer than 100 do not receive a thick enough coating (ans: 0.4721)

> phyper(99, 333, 666, 300)

[1] 0.4721148

* 1. At least 110 do not receive a thick enough coating (ans: 0.08254)

> 1 - phyper(109, 333, 666, 300)

[1] 0.08253631

* 1. Between 90 and 110 (inclusive) do not receive a thick enough coating (ans: 0.8759)

> phyper(110, 333, 666, 300) - phyper(89, 333, 666, 300)

[1] 0.8759101

1. Imagine that we are drawing cards from a standard 52-card deck without replacement. Using the **dhyper()** function, give a table and a barplot that gives that exact probability distribution for the number of aces obtained when **8** cards are drawn without replacement. (You may want to compare at least one of the probabilities to the one you get by using the formula from class.) Be sure to give clear and descriptive labels for your barplot. Note: there are four aces in a standard 52-card deck.

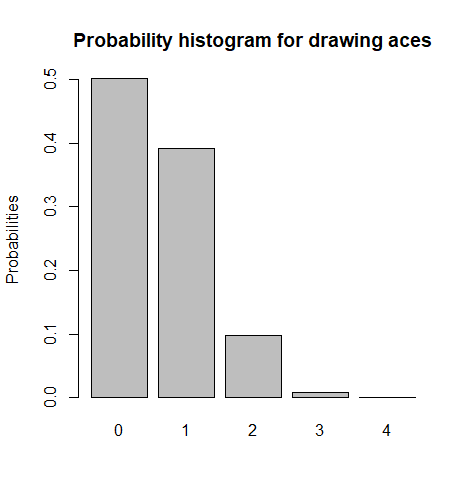
> Deck()

0.000258564964447318 0.00910148674854558 0.0978409825468648

1 1 1

0.39136393018746 0.501435035552683

1 1



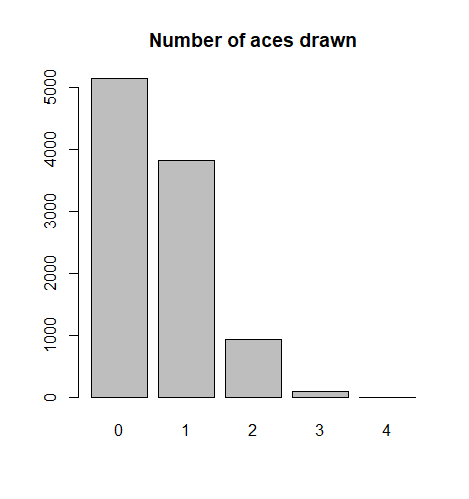
1. Approximate the probability distribution for the number of aces obtained when drawing 8 cards from a standard 52-card deck using relative frequencies obtained from two different simulations, as follows:
2. By writing a function that simulates drawing **m** cards **n** times, using the **sample()** 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, as well as a bar plot. Run your function for **m**=8, **n**=10000 and provide the table and a bar plot.

> Aces(8,10000)

arr

0 1 2 3 4

5057 3854 1002 84 3



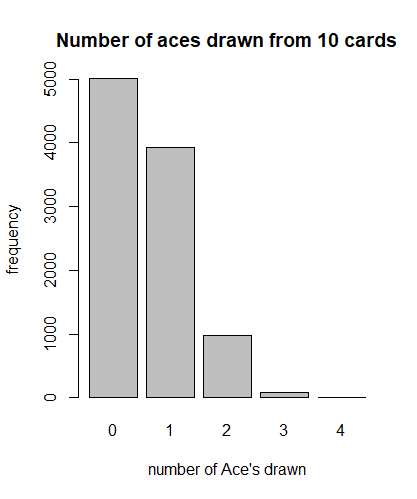
1. By writing a function that simulates drawing **m** cards **n** times, using the **rhyper()** function. As before, your function should output a table giving the relative frequencies, as well as a bar plot. Run your function for **m**=8, **n**=10000 and provide a table and a bar plot.

> Aces2(8,10000)

arr

0 1 2 3 4

5007 3927 971 92 3



1. How do the tables and bar plot from parts (a) and (b) compare to the exact probabilities obtained with the **dhyper()** function?

Both a and b correspond to the exact probabilities that we goe from dhyper(). It’s almost as if they were twins!