LAB100

Week 18: Probability and Simulation

In this workshop you will learn how to use RStudio to calculate the probabilities of discrete events such as results of coin tosses, rolls of dice or removal of balls from bags. In addition you will also learn how to simulate random events of this kind. Upon completing this workshop you should be able to:

- Simulate data with and without replacement
- Seed the random number generator
- Use RStudio to compute the probabilities of particular events

Remember that you need to change the working directory and open a new R script at the start of each workshop.

1 Sample Space and Probabilities

Before creating simulations, we first need to define the sample space and the probability vector where each element corresponds to the probability of each event in the sample space. For example, a fair coin can either land on a head (H) or a tail (T), where each event is equally probable:

$$\mathbb{P}(\{H\}) = \frac{1}{2} \quad \mathbb{P}(\{T\}) = \frac{1}{2}$$

We can represent this information by these two vectors:

```
SampleSpace <- c( "H", "T" )
ProbSet <- c( 0.5, 0.5 )
```

All probability trials that have a discrete and finite sample space can be represented by these two vectors. Below are some more examples:

Biased Coin Toss: where the probability of a head is twice the probability of obtaining a tail

```
SampleSpace <- c( "H", "T" )
ProbSet <- c( 2/3, 1/3 )
```

Role of a Six Sided Dice:

```
SampleSpace <- c( 1, 2, 3, 4, 5, 6 )
ProbSet <- c( 1/6, 1/6, 1/6, 1/6, 1/6, 1/6 )
```

The Main Draw of the EuroMillion Lottery Numbers:

```
SampleSpace <- 1:50
ProbSet <- rep( 1/50, times=50 )
```

It is important that the vector `ProbSet` satisfies the axioms of probability. A simple function that performs a logical test can be used to assess whether each event in sample space has an associated probability value and the set of probabilities satisfy the positivity and finitivity conditions. Carefully type the following command into your editor and ensure that you understand what each logical statement is assessing.

```
is.ProbSet <- function( space, probs ){
    SizeCond <- length( space ) == length( probs )
    Positve <- all( probs >= 0 )
    Finite <- sum( probs ) == 1
    Valid <- SizeCond && Positve && Finite
    return( Valid )
}
```

The `all` command returns `TRUE` only is every element in the supplied logical statement is true. By considering each of the four examples, check that the pairs of vectors are valid probability sets:

```
is.ProbSet( SampleSpace, ProbSet)
```

2 Sampling with Replacement

Simulating trials that have a discrete and finite sample space can easily be performed by the `sample` command. This function generates a random sample from the sample space according to the probability vector. The arguments used in this command are as follows:

To simulate a single toss of a fair coin, copy the following and execute the trial.

```
SampleSpace <- c( "H", "T" )
ProbSet <- c( 0.5, 0.5 )
sample( SampleSpace, size=1, replace=TRUE, prob=ProbSet )
```

Argument	Description
x	A vector contains the sample space.
size	The number of events to simulate.
replace	logical argument stating whether the event should be replaced (default is FALSE).
prob	Vector of probabilities.

Run the last line of code a few times. Did you get a different output? Instead of repetitively executing this command to replicate numerous coin tosses, increase the number of simulations by setting a larger value for size.

```
sample( SampleSpace, size=10, replace=TRUE, prob=ProbSet )
sample( SampleSpace, size=100, replace=TRUE, prob=ProbSet )
sample( SampleSpace, size=1000, replace=TRUE, prob=ProbSet )
```

How long do you think it would take for you to toss a coin 1000 or more times? Understanding the features of 1000 coin tosses is difficult to see from this large vector. Therefore it is best to summarise the simulation in a table or histogram.

```
CoinToss <- sample( SampleSpace, size=1000, replace=TRUE, prob=ProbSet )
TAB <- table( CoinToss )
TAB
histogram( TAB, main="Simulation of 1000 Coin Tosses" )
```

3 Seeding Simulations

Sometimes it is helpful to be able to generate the same set of random numbers again and again. This is particularly useful for identifying and fixing coding errors. In order to simulate the same random sequence we first need to seed the random number generator by using the set.seed command that accepts a single positive integer number called the seed number. To illustrate, simulate the outcome of rolling five dice.

```
SampleSpace <- 1:6
ProbSet <- rep( 1/6, times=6 )
set.seed( 30 )
sample( SampleSpace, size=5, replace=TRUE, prob=ProbSet )
set.seed( 75 )
sample( SampleSpace, size=5, replace=TRUE, prob=ProbSet )
set.seed( 30 )
```

```
sample( SampleSpace, size=5, replace=TRUE, prob=ProbSet )
sample( SampleSpace, size=5, replace=TRUE, prob=ProbSet )
```

Notice that the numbers generated in the first and third dice role sequences are identical because the random number generator has been seeded with the same seed number.

4 Sampling with Replacement

Generating simulations of a random trial where the occurrence of each event can only occur once is called sampling without replacement. For instance, suppose we have a bag containing 6 red balls and 4 blue balls. Suppose we remove 4 balls in turn from the bag without replacing them. The probability that all the balls are red is equal to

$$\frac{6}{10} \times \frac{5}{9} \times \frac{4}{8} \times \frac{3}{7} = \frac{1}{14}$$

.

The probability that exactly three balls are red can be calculated by noting that there are four different orderings of balls that can lead to this situation:

RRRB
RRBR
RBRR
BRRR

each of these individual sequences has probability

$$\frac{6 \cdot 5 \cdot 4^2}{10 \cdot 9 \cdot 8 \cdot 7}$$

so the overall probability is

$$\frac{6 \cdot 5 \cdot 4^3}{10 \cdot 9 \cdot 8 \cdot 7} = \frac{1}{4}$$

To simulate the event of removing 4 balls from the bag of 10 balls we can use

```
SampleSpace <- rep(c("R","B"),c(6,4))
ProbSet <- rep(1/10, 10)
sample(SampleSpace, size=4, replace=FALSE, prob=ProbSet)
```

Simulation can be used as a way of approximating the probability of events. For instance if we repeat the simulation above many times, we can observe the proportion of cases where all the balls are red and the proportion of cases where exactly three balls are red.

```
set.seed(245)
allred <- 0
threered <- 0
N <- 1000
for (i in 1:N) {
    balls <- sample(SampleSpace, size=4, replace=FALSE, prob=ProbSet)
    if(sum(balls=="R")==4) allred <- allred + 1
    if(sum(balls=="R")==3) threered <- threered + 1
}
p4 <- allred/N
p3 <- threered/N
```

with this particular seed number we get values of 0.079 and 0.376 for the probabilities of observing all red and three reds, respectively. This is reasonably close to the true value. We will explore this idea of using simulation to estimate probabilities, known as Monte Carlo evaluation, in more detail in the next Workshop.

Quiz 1: Seeded Dice Throws

Use the seed number 565 and simulate 200 rolls of a fair 6 sided dice. How many times did the number 5 occur?

(**A**) 31 (**B**) 32 (**C**) 33 (**D**) 34 (**E**) 35

Quiz 2: Simulating the Lottery

The UK National lottery (Lotto) involves the drawing of 6 numbered balls from a set of balls numbered 1 to 49. Which of the following R commands will simulate a single realisation of a National lottery draw.

(**A**) `sample(1:49, 6, replace=TRUE , prob=rep(1/49,49))`
(**B**) `sample(1:6, 49, replace=TRUE, prob=c(1/49,1/48,1/47,1/46,1/45,1/44))`
(**C**) `sample(1:49, 6, replace=FALSE, prob=rep(1/49,49))`
(**D**) `sample(1:6, 49, replace=FALSE, prob=c(1/49,1/48,1/47,1/46,1/45,1/44))`
(**E**) `sample(1:49, 6, replace=FALSE, prob=1/(49:1))`

Quiz 3: Balls in a bag

Suppose a bag contains 6 red balls and m blue balls. If 4 balls are drawn from the bag without replacement then the probability that 3 or more of the drawn balls are red is given by

$$p_m = \frac{6 \cdot 5 \cdot 4 \cdot 3 + 6 \cdot 5 \cdot 4^2 \cdot m}{(m+6) \cdot (m+5) \cdot (m+4) \cdot (m+3)}$$

Find the minimum number of blue balls needed to ensure this probability is less than 0.05.

Quiz 4: Design of a lottery

A company wishes to launch a new lottery. They plan to charge £1 per ticket and the only prize will be the jackpot which will be fixed at £1000000 They want to ensure the expected pay-out is as close to 50% of the total revenue as possible. The probability of matching all balls in a match n from N lottery can be computed as

$$1/\binom{N}{n}$$

so the expected pay out of a single ticket is

$$1000000/\binom{N}{n}.$$

Find the combination of n and N, for $N < 100$, that is closest to achieving an expected pay-out equal to 0.5.
[Hint: Recall that $\binom{N}{n}$ can be computed in R using `choose(N,n)`.]