Exercise #2: Normal distributions and the Galton board

Patrick Applegate, patrick.applegate@psu.edu

Learning Goals

After completing this exercise, you should be able to

- describe under what circumstances normal distributions arise
- use quantile-quantile plots to determine whether individual data sets are normal or not
- install and load R packages from CRAN, the Comprehensive R Archive Network
- write simple `for` loops in R

Introduction

Suppose that we ask a class of students to walk individually from one end of a field to the other and count the number of steps that they take in order to cover the distance. More than likely, each student will get a somewhat different answer. After each student has paced off the field, we then make a histogram of the number of steps taken by the individual students.

If we were to perform this experiment with a large number of students, the histogram would likely resemble a normal distribution. Normal distributions arise when a large number of measurements are taken of a single quantity, and the individual measurements are affected by random processes that are additive and that can be either positive or negative. In our example, the random processes have to do with the varying stride lengths of the students (some take longer strides and some shorter), and counting errors (which are likely to be small and can cause any given student to either over- or underestimate the actual number of paces they took).

A Galton board is a physical device that demonstrates this concept (Fig. 1). A Galton board has rows of pins arranged in a triangular shape, with each row of pins offset horizontally relative to the rows above and below it. The top row has one pin, the second row has two pins, and so forth.

If a ball is dropped into the Galton board, it falls either to the right or the left when it bounces off the top pin. The ball then falls all the way to the bottom of the board, moving slightly to the right or left as it passes through each row of pins. Bins at the bottom of the board capture the ball and record its final position. If this experiment is repeated with many balls, and if the Galton board includes many rows of pins, the number of balls in each bin resembles a normal distribution (Fig. 1).

In this exercise, you'll experiment with a simple representation of a Galton board in R and examine the distributions of final ball positions that it produces. You'll also examine two data sets that are "built in" to R and see whether they are well described by a normal distribution or not.

Tutorial

There is a ready-made, animated representation of the Galton board in R's `animation` package. An R package is a set of functions or commands that have been written by R users and posted to CRAN, the Comprehensive R Archive Network. R packages have to meet some minimum standards of quality in order to be included on CRAN, and the packages are open-source and free to use.

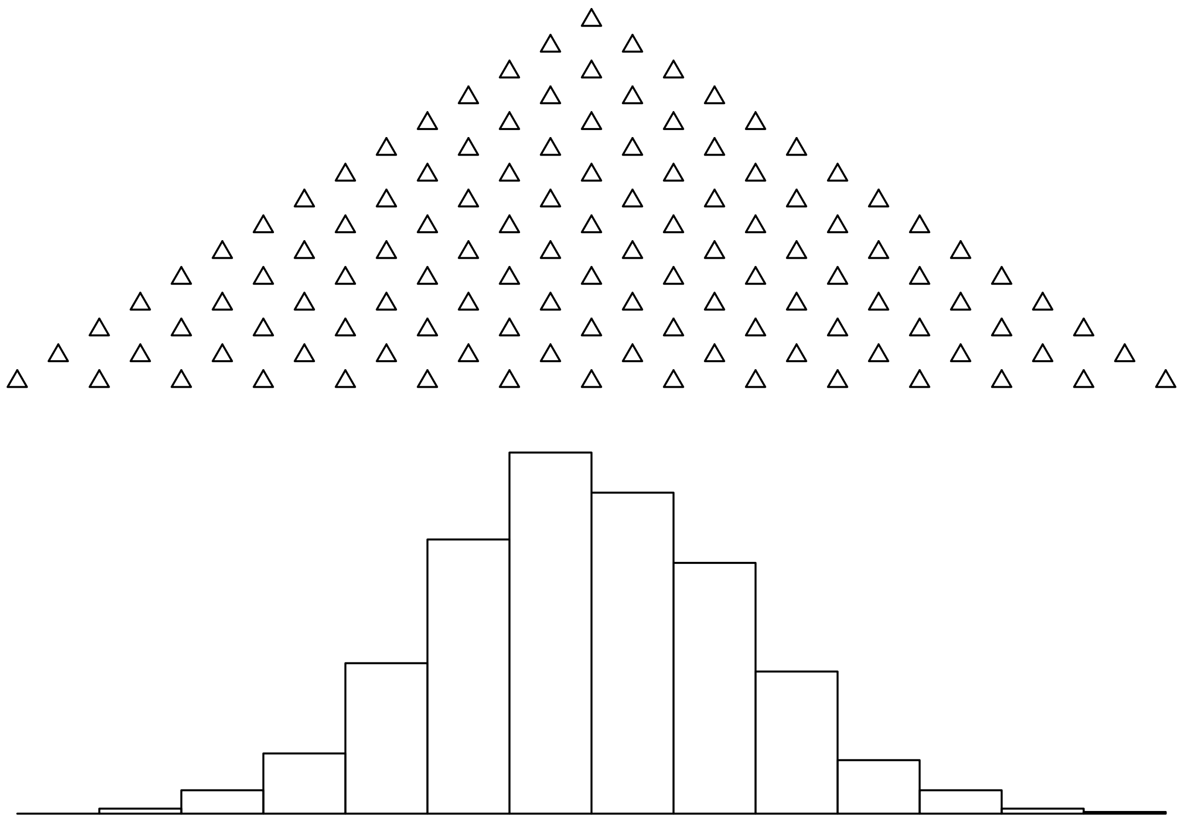

Figure 1: A Galton board with 15 rows of pins (top), and one possible distribution of final ball positions resulting from dropping 1000 balls through the board (bottom). Triangles, pins in the Galton board; bars, heights of balls in each bin.

Installing and loading packages from CRAN in R

Making a particular package available for use in R is a two-step process: first we download the package, and then we load it into R's working environment. To download and load the `animation` package, open RStudio and type the following commands into the Console window (pressing Enter after each command). You can ignore the text after the pound signs (`#`): these text fragments are *comments* that are not processed by R.

```r
install.packages('animation') # downloads the animation package from CRAN
require('animation') # loads the animation package
```

Once the package has been loaded, we can experiment with the Galton board animation. Typing `help(quincunx)` in the Console window in RStudio and pressing Enter lets us look at the arguments of the `quincunx()` function. ("Quincunx" is another name for a Galton board.)

```r
# from help(quincunx)
quincunx(balls = 200, layers = 15, pch.layers = 2, pch.balls = 19,
    col.balls = sample(colors(), balls, TRUE), cex.balls = 2)
```

Here, the `balls` argument represents the number of balls to drop through the Galton board, and `layers` indicates the number of rows of pins in the board. The numbers after each of these arguments (`balls = 200, layers = 15`) indicate the arguments' default values.

Let's try just one ball at first. Type the following two commands into the Console window in RStudio and press Enter after each one.

```r
ani.options(nmax = 14) # sets the number of animation frames
quincunx(balls = 1) # runs the animation
```

You'll see the ball move downward through the rows of pins and land in a bin, where it is represented with a tall bar.

Try entering the second command above into RStudio repeatedly. With your cursor at the prompt in the Console window, press the Up arrow and then Enter to reevaluate the last command you entered. How does the path of the ball differ among evaluations of the `quincunx()` function? Which bins receive balls most often?

Once you've examined the single-ball case, let's move on to a case with many balls. Enter the following commands into RStudio's Console window, pressing Enter after each command.

```r
n.balls <- 200 # sets the number of balls to drop
n.rows <- 15 # sets the number of rows of pins in the Galton board
ani.options(nmax = n.balls+ n.rows- 2)
quincunx(balls = n.balls, layers = n.rows)
```

This example will take several minutes to run. If you get tired of watching the animation, you can stop RStudio by holding down Control and pressing C on your keyboard, or by clicking on the little STOP sign that appears in the upper right-hand corner of the Console window while R is working.

This time, we see many balls of different colors fall from the top of the Galton board to the bottom. The heights of the bars in the different bins grow as the balls reach the bottom of the board. How well does the final histogram agree with Figure 1?

Writing our own simple Galton board script

Building computer models involves thinking about how to represent the behavior of physical (or chemical, or social) systems in a form that can be understood by a computer. In thinking about how we could represent the Galton board in terms of R code, we might start by looking at the diagram of the Galton board in the top part of Figure 1. The horizontal spacing between the pins is always the same, but the rows are offset relative to one another; that is, none of the pins in any given row are immediately above or below the pins in the adjacent rows. This offset is exactly half of the horizontal spacing between pins in the same row. Thus, we can imagine that the ball bounces 0.5 distance units to one side whenever it hits a pin.

We can then represent the path that a single ball takes through the Galton board by

```r
path <- sample(x = c(-0.5, 0.5), size = (n.rows- 1), replace = TRUE)
print(path)
```

```
##  [1] -0.5 -0.5  0.5  0.5 -0.5  0.5  0.5  0.5  0.5 -0.5 -0.5 -0.5  0.5 -0.5
```

Positive values indicates that the ball bounces to the right; negative values indicate that the ball bounces to the left. So, in this example, the ball bounces twice to the left (-0.5, -0.5), twice to the right (0.5, 0.5), and so on.

The `sample()` command generates sequences of values that come from a vector of possibilities supplied to the argument `x`. Have a look at `help(sample)` to see what the other arguments mean. We set the `size` argument set to `(n.rows- 1)`, rather than just `n.rows`, for consistency with the `quincunx()` function, which yields a number of bins equal to `n.rows -1`. To check this statement, count the number of bins in the lower part of Figure 1.

The bin that the ball lands in is then just

```r
bin <- sum(path)
print(bin)
```

```
## [1] 0
```

In this case, the ball lands immediately below the top pin of the Galton board, in the middle bin (0).

Doing things over and over again: `for` loops

The commands above tell us about the path that any single ball takes through the Galton board, as well as the bin it finally lands in. But, suppose we want to write a piece of R code that performs this calculation for many balls and gives us a histogram of the bins that they land in. How could we do that?

Most, perhaps all, programming languages include a method for carrying out a set of instructions a fixed number of times. In R, this method is called a `for` loop. The syntax of `for` loops looks like this:

```r
n.times <- 3 # not part of the for loop syntax
for (i in 1: n.times) { # note the open curly brace!
  print(i) # instructions to be repeated go here
} # note the close curly brace!
```

```
## [1] 1
## [1] 2
## [1] 3
```

This sample `for` loop is not very exciting: it simply `print()`s the contents of the *index variable* `i` during each iteration of the commands within the curly braces (`{}`).

To see why this sample `for` loop `print()`s the values that it does, we can examine the vector of values taken by the index variable `i`:

```
1: n.times # equivalent to seq(1, n.times, by = 1)
```

```
## [1] 1 2 3
```

So, this vector contains the values 1, 2, 3. As the `for` loop runs through these values, it `print()`s them to the screen successively.

In "real" `for` loops, the name of the index variable, the vector of values that the index variable can take (here `1:  n.times`), and the instructions in the curly braces can all change. In particular, there can be more than one line of instructions between the curly braces.

In general, we will want to create a variable that will store the results from the commands carried out within the `for` loop. The following block of code generates and `print()`s the first five Fibonacci numbers:

```
n.times <- 5 # not part of the for loop syntax
output <- rep(1, n.times) # makes a vector of five 1's
for (i in 3: n.times) {
  # stores the sum of the preceding two elements of output in the ith element
  # of output
  output[i] <- sum(output[(i- 2): (i- 1)])
}
print(output)
```

```
## [1] 1 1 2 3 5
```

Here, `output` is a vector with `n.times` elements that receives the results of the calculations performed within the `for` loop.

In the Exercise, below, you'll put together the information above to write your own script for making histograms of the outcomes from a Galton board. First, though, let's examine how we determine whether a particular data set is normal or not.

Normal or not normal?: Generating and interpreting quantile-quantile plots

It can be tempting to assume that a particular process gives normally-distributed output, or that data are normally distributed; however, assuming normality when it isn't present can lead to large errors. How can we check individual data sets for normality?

One robust and simple, but approximate, test for normality involves making a quantile-quantile (Q-Q) plot. The details of how Q-Q plots work aren't important for our purposes here. However, if data are drawn from a normal distribution, they will lie on a line connecting the first and third quartiles of the data when drawn on an appropriately-constructed Q-Q plot (see `help(qqline)`). R makes drawing this type of Q-Q plot easy.

For example, the following block of code generates 100 random numbers from a normal distribution, makes a Q-Q plot from them, and draws a 1:1 line on the plot:

```
norm.vals <- rnorm(100, mean = 5, sd = 3) # generates normally-distributed values
qqnorm(norm.vals) # makes the q-q plot
qqline(norm.vals) # adds a line; do the points lie on this line?
```

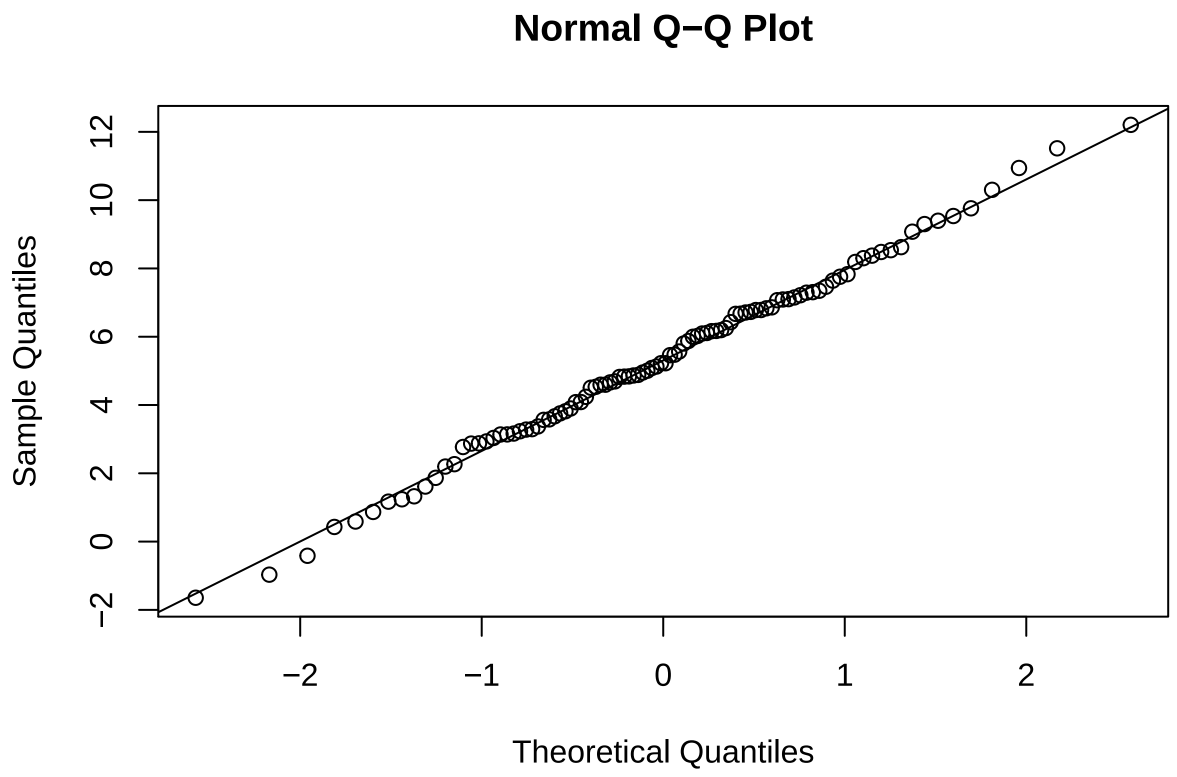

Figure 2: A Q-Q plot showing 100 normally-distributed random values. The points, which represent the random values, lie approximately on the line, confirming that they come from a normal distribution.

Exercise

Part 1. Building on your R script from Exercise 1 and the information presented in this document, write an R script that

1. has comments at the top that explain what the script does and who wrote it
2. clears existing variables from memory and closes open figures
3. sets values for the number of balls to drop and the number of rows of pins in the Galton board (you might use the variable names `n.balls` and `n.rows`)
4. creates a vector `output` that contains a number of elements equal to `n.balls`; these elements should be populated with `NA`s to start (hint: see `help(rep)`)
5. uses a `for` loop that

- runs through values of the index variable `i` from `1:  n.balls`,
- determines which bin of the Galton board each ball lands in, and
- stores these values in the different elements of `output`

6. makes a histogram of `output`. The code block below shows one way of carrying out this step.

```
# Make a histogram of the results.
bin_edges <- seq(-n_rows/ 2, n_rows/ 2, by = 1)
hist(output, bin_edges, xlab = 'Bin', ylab = 'Frequency')
```

Part 2. Add commands to your R script so that it also generates a Q-Q plot, with a line, for the results in `output`.

Part 3. Generate Q-Q plots for two built-in data sets in R,

1. the waiting time between Old Faithful eruptions (`faithful[, 2]`), and
2. the sepal length of setosa irises on the Gaspe Peninsula (`iris3[, 1, 1]`).

You can learn about the sources of these data with `help(faithful)` and `help(iris3)`.

Questions

1. Set the number of rows of pins in your Galton board R script to 10. Also set the number of balls to drop to a low value, say 10 to begin.

- How does the histogram change as you repeatedly `source()` the script?
- What is the smallest number of balls that you can drop that gives more or less the same histogram each time you run the script, for 10 rows of pins?

2. Recall from the discussion above that a Galton board only gives an approximately-normal distribution if both the number of balls that are dropped and the number of bins are large. Leaving the number of rows of pins at 10, increase the number of balls to a large value like 10^4 (in R code, that would be just `10^4`). Confirm that the `output` under these circumstances is not normally distributed. How large does the number of rows of pins have to be to get approximately-normal results?
3. Which of the built-in data sets listed above is approximately normally distributed, and which is not? How can you tell?