post01-cole-stern

Cole Stern

11/30/2017

Random Numbers in R:

Understanding their function and application

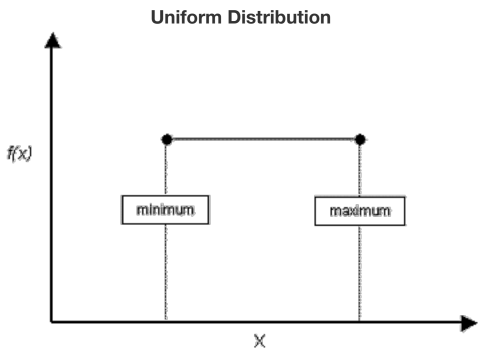
So far in our exploration of R, we have only slightly played around with using random numbers. In this blog post, I aim to delve deeper into the world of random and pseudo-random numbers for varying types of distributions. Hopefully, you (the reader) will therefore have a more intuitive understanding of what is going on behind the scenes in R when you work with random numbers.

Quick note about random numbers

When we talk about generating random numbers, it's not like these numbers are coming from thin air! Rather, computers are what we call *deterministic* meaning that they rely on algorithms and set rules in order to generate random numbers. Additionally, these machines may rely on a 'seed' (which you can think of as a starting position within a string of numbers) in order to start off the process and ensure that the results are replicable across separate computers. In this sense, these numbers are not truly random and instead are 'pseudo-random', however for the majority of work that we will be doing this will be sufficient as it still lets us get a relatively even distribution from a large data set.

The workhorse functions: runif, rbinom, rnorm

One way to generate a set of random numbers is by using the `runif` function. As evident from its name, this function generates numbers from a **uniform** distribution, meaning that the distribution has a constant probability across all values being sampled. In order to use this function, you need to supply both the number of samples as well as the lower and upper limits of the distribution (note that these upper and lower limits are exclusive). For better visual reference, see this image of one hypothetical uniform distribution:



For example, the code below will show how to get 5 values between 1 and 25, with each value between 1 and 25 having an equal probability of being sampled.

```
runif(n = 5, min = 1, max = 25)
```

```
## [1]  5.574182 15.212183 15.103767 15.863338  3.041924
```

In the code above, since we are not specifying a start seed, we will get different numbers every time we execute it. To simulate this, see the code below which uses a for loop to run the code multiple times.

```
for (i in 1:4) {
  print(runif(n = 5, min = 1, max = 25))
}
```

```
## [1] 15.713276 20.363307  4.565590  6.265175  3.512024
## [1] 11.82208  5.73511 18.92308 18.48452 16.04907
## [1]  2.674305  9.657395 20.871139 10.822276 23.640399
## [1] 12.128126 17.601752 11.656936  7.126234 17.217826
```

Compare the answers you saw in the previous code chunk with the one below where a start seed is specified. Using `set.seed()` you can see that R will give you the same "random" values every time the code is executed. This is what allows us to ensure replicability no matter who is running the code. In regards to what to set the `set.seed()` value to, this is up to you, it just has to be a positive integer—I personally was inspired by the Tommy Tutone single of the same name.

```
for (i in 1:4) {
  set.seed(8765309)
  print(runif(n = 5, min = 1, max = 25))
}
```

```
## [1] 11.43465 17.00913 24.41000 15.02991 16.09720
## [1] 11.43465 17.00913 24.41000 15.02991 16.09720
## [1] 11.43465 17.00913 24.41000 15.02991 16.09720
## [1] 11.43465 17.00913 24.41000 15.02991 16.09720
```

Being that `runif()` samples from a uniform distribution, you would want to use this function to do things like randomly select students from a class, where every student has an equal probability of being selected. But if you're wanting to see what other distributions you can sample from in R you can visit this website supplied by CRAN. Here you can explore other different types of distributions readily available and get a better understanding of when to use each one. Additionally, you can also use the code `help.search("distribution", package = "stats")` and enter it into the console to search for common distributions without having to search online.

Which transitions us nicely to talking about another common R distribution function, `rbinom()`. Again, using its name as a hint, this function generates random numbers but from a **binomial distribution**. The binomial distribution is most often used in statistical settings to model the number of successes within a specified sample as drawn from the larger population. The classic example of this is the flipping of a coin and counting up either the number of heads or tails. To simulate this scenario we can use `rbinom()`, making sure to specify the number of observations, the number of trials, and the probability of success on each trial. In the following example the code will generate 25 observations from a single trial with a 50/50 change of success (let us say getting heads is considered a success).

```
coin_tosses_25 <- rbinom(25, 1, 0.5)
coin_tosses_25
```

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

To get a more readable tally of the counts, we can use `table()` to display the number of Zero's and One's, showing us number of failures and successes, respectively.

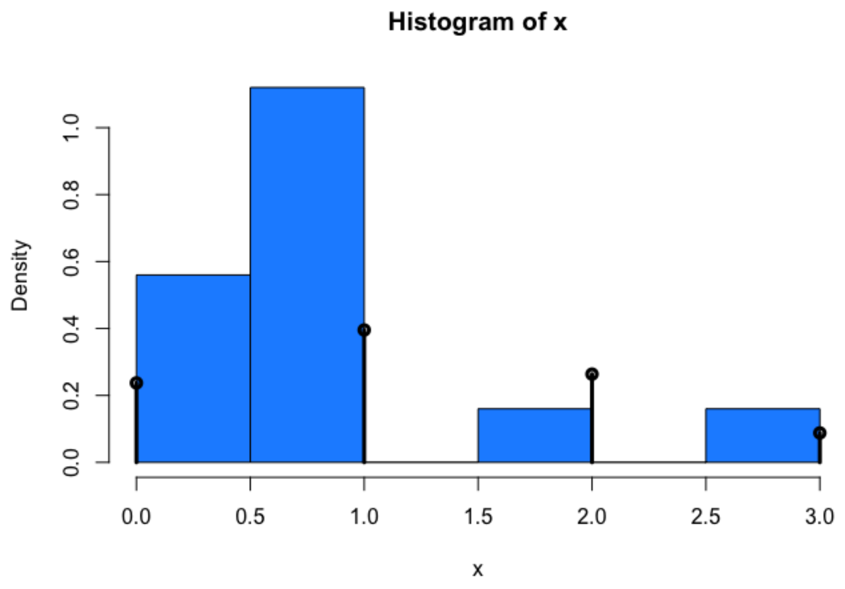
```
table(coin_tosses_25)
```

```
## coin_tosses_25
##  0  1
## 11 14
```

Playing around with the parameters, we can explore how with a larger and larger value for n (our sample size), our resultant distributions starts to look almost normal. Demonstrating this, our code below will show 25 binomial distributed random numbers with 5 trials and probability equal to 0.25.

```
# 25 random numbers
x = rbinom(25, size = 5, p = 0.25)
hist(x, probability = TRUE, col = "#1E90FF")

# use points, not curve as dbinom() wants integers only for x
xvals = 0:5
points(xvals, dbinom(xvals, 5, 0.25), type = "h", lwd = 3)
points(xvals, dbinom(xvals, 5, 0.25), type = "p", lwd = 3)
```

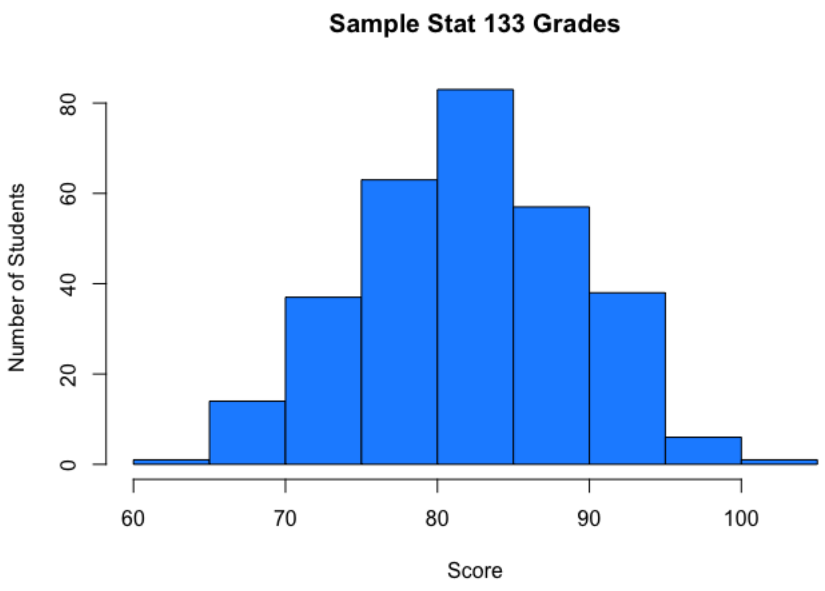


Moving forward, the last of the workhorse functions we are covering here would be `rnorm()`, and you guessed it, this will pull numbers from a **normal distribution**. To use this function, you need to specify the number of observations, the mean, and the standard deviation.

For example, let's say we want to explore a hypothetical Stat 133 class with 300 students. On their recent exam, knowing that the average was an 82 with a standard deviation of 7, we can use `rnorm()` to model the hypothetical normal distribution. See the code below:

```
# Normal disribution of 300 students, mean = 82, sd = 7
students <- rnorm(n = 300, mean = 82, sd = 7)

# Graphing the results
hist(students, col = "#1E90FF", xlab = "Score", ylab = "Number of Students", main = "Sample Stat 133 Grades")
```



Outside applications: the pareto distribution

After exploring some of the basic random number generator distributions in R, you may want to see what other things you can play around with. While the database at CRAN which was linked earlier does detail the numerous other functions available that come packaged with R, you are by no means limited to these functions. In this next section we will explore how to create our own functions to represent a specific distribution.

In this example, drawing from PennState's Center for Astrostatistics page, we will be exploring the power-law a.k.a The Pareto Distribution. This distribution is fittingly named after the Italian civil engineer, economist, and sociologist Vilfredo Pareto. The Pareto distribution has many applications in various fields used to describe social, scientific, and geographically observable phenomena. However, in this example we will concentrate on the Pareto Distribution's use in astrostatistics and therefore our functions will reflect that. The main use that the distribution has in astrostatistics is to model the brightness of standard candles in space, allowing astronomers to calculate distances to nearby stars or far-away galaxies.

The probability density function for the Pareto Distribution is given by $f(x) = ab^a/x^(a+1)$, where *x* is greater than *b*. Additionally, *a* and *b* are fixed positive values, where *b* is the minimum possible value.

To create our function we will first start off by assigning the probability density function equal to `dpareto()`. Additionally, we will set default values for both *a* and *b* which can later be changed by the user.

```
dpareto <- function(x, a = 0.5, b = 1) {
  a*b^a/x^(a+1)
}
```

From here we will integrate the density function to obtain the distribution function. (Note that we account for *x* being greater than *b* in the function by coercing that inequality to either 0 or 1)

```
ppareto <- function(x, a = 0.5, b = 1) {
  (x>b)*(1-(b/x)^a)
}
```

Next we will invert the distribution function in order to yield the quantile function.

```
qpareto <- function(u, a = 0.5, b = 1) {
  b/(1-u)^(1-a)
}
```

And lastly, understanding that when the quantile function is applied to a uniform random variable, what we get is a random variable with the desired distribution simulating random Pareto variables.

```
rpareto <- function(n, a = 0.5, b = 1) {
  qpareto(runif(n), a, b)
}
```

From here we can find random Pareto Deviates by using our `rpareto()` function, yielding 25 hypothetical standard candles randomly distributed throughout space.

```
set.seed(8)
rpareto(25)
```

```
##  [1] 1.368830 1.123541 2.234158 1.694845 1.214026 1.886213 1.187512
##  [8] 3.842456 2.081288 1.677161 1.357119 1.047882 1.327320 1.482437
## [15] 1.077216 3.721930 1.000651 1.165995 1.175683 1.445043 1.134964
## [22] 1.299606 1.611386 1.128646 1.723288
```

Wrapping Up

I hope that this lesson has made you a little more aware of how random numbers in R are both generated and applied toward different types of analysis. Moreover, while most of the time you will be using the main base functions in order to generate random number from different distributions, you also have the capability to create your own functions to represent outside distributions. If there is one piece of advice I could give, I would say don't be afraid to explore. Google is your best friend, and most likely if you have a question, it has been tackled (or at least asked) on any number of forums.

So Go Out and Start Exploring!

References

- http://www.simonqueenborough.info/R/stats-advanced/sampling.html

> Good source for explaining the uses behind the "workhorse" functions.

- https://en.wikibooks.org/wiki/R_Programming/Random_Number_Generation

> Explains how `set.seed()` works.

- https://engineering.mit.edu/engage/ask-an-engineer/can-a-computer-generate-a-truly-random-number/

> Shows the difference between random and pseudo-random, and how machines generate random numbers.

- https://cran.r-project.org/web/views/Distributions.html

> The link to all the distributions that come package with R.

- http://astrostatistics.psu.edu/

> Talks about the pareto distribution and its application towards astrostatistics.

- https://en.wikipedia.org/wiki/Binomial_distribution

> Wikipedia on Binomial Distribution

- https://en.wikipedia.org/wiki/Pareto_distribution

> Wikipedia on Pareto Distribution

Processing math: 100%