Random number generation in R

Nathan Brouwer

11/5/2019

Introduction

Generating random numbers is key to many tasks in statistics and computational biology. Usually we can focus on our analysis or simulation experiments and not worry about the details, but its good to be familiar with some basic concerns when it comes to random number generations

Using set.seed() to make simulations reproducible

The way that random numbers are generated in practice uses algorithms that produce random-looking numbers and random-behaving sets up numbers. However, these algorithms are actually deterministic based on a given numberic input. Normally when we generate a random number in R, R grabs a constantly value to use as the input.

The sys.time() function can be used to tell us the time

```
Sys.time()
```

```
## [1] "2019-11-18 23:19:49 EST"
```

R can keep track of very small scales of time; here, I set to measure seconds to 3 decimal places (1000s of sectons)

```
options(digits.secs = 3)
Sys.time()
```

```
## [1] "2019-11-18 23:19:49.656 EST"
```

```
Sys.time()
```

```
## [1] "2019-11-18 23:19:49.660 EST"
```

Its a little more complicated than this, but basically R uses a constantly chaning baseline as a starting point for random number generation.

Normally, when we generate random numbers this means that every time we call a random number generation fucntion we get a different results.

For example, runif() generates a random value from 0 to 1. Run this 4 times and you get four values.

```
runif(n = 1)
```

```
## [1] 0.1169965
```

```
runif(n = 1)
```

```
## [1] 0.2605173
```

```
runif(n = 1)
```

```
## [1] 0.06432931
```

```
runif(n = 1)
```

```
## [1] 0.2296483
```

Prior to generating a random number we can fix the starting point of the random number generator, called the **seed**. R will use the same input (seed) each time it runs its random number generator, and so the same output will be given. Every time you run this, you should get 0.5074782.

```
set.seed(10)
runif(n = 1)
```

```
## [1] 0.5074782
```

```
set.seed(10)
runif(n = 1)
```

```
## [1] 0.5074782
```

```
set.seed(10)
runif(n = 1)
```

```
## [1] 0.5074782
```

```
set.seed(10)
runif(n = 1)
```

```
## [1] 0.5074782
```

Note that the runif() generates randum uniform data, and rnorm() generates random normal data, so even with the same seed they give different results. For random normal data, the random value shoudl be 0.01874617.

```
set.seed(10)
rnorm(n = 1)
```

```
## [1] 0.01874617
```

A different seed will give you a different value

```
set.seed(11)
rnorm(n = 1)
```

```
## [1] -0.5910311
```

And again

```
set.seed(12)
rnorm(n = 1)
```

```
## [1] -1.480568
```

The value of the seed has no inherent importance.

Types of random number generators

R has many random number generators, with fancy names like

- "Marsaglia-Multicarry"
- "Super-Duper"
- "Mersenne-Twister"
- "L'Ecuyer-CMRG"

You can see the default random number generator with RNGkind()

```
RNGkind()
```

```
## [1] "Mersenne-Twister" "Inversion"        "Rejection"
```

Reporting simulations

As long as you know the version of R someone used and the seed they used, you should be able to reproduce their results using their code. Ideally the code definitive version of a simulation reported in a paper should report the seed so that the **exact** results of the paper can be reproduced by running the code. So, in the case of my BLAST-related project, when I"m finally ready to write up a report, I should put set.seed() in the proper places in the code, give it a number, run the whole workflow, and report the results from that run.

Notes:

From the set.seed help file: "Initially, there is no seed; a new one is created from the current time and the process ID when one is required. Hence different sessions will give different simulation results, by default."

```
Sys.time()
```

```
## [1] "2019-11-18 23:19:50.100 EST"
```

```
set.seed(Sys.time())
```

R's sample() function actually had some problems https://arxiv.org/abs/1809.06520