

HW 3: Random Number Streams

Sokona Mangane

2023-02-04

DCS 307 - HW 3

```
#tinytex::install_tinytex(), to knit as a PDF  
library(simEd)
```

Question 1: What are the first ten service times from the first run?

These are the first 10 service times from the second run: 0.6906071, 0.2777054, 0.1499705, 0.6604378, 0.5704503, 0.1073709, 0.3050448, 0.640531, 1.4994352, 0.2054382

Question 2: What are the first ten service times from the second run?

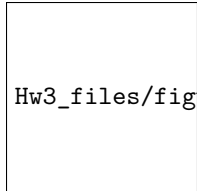
These are the first 10 service times from the second run: 0.6906071, 0.2777054, 0.1499705, 0.6604378, 0.388547, 2.7028233, 1.4994352, 0.8472361, 1.8154944, 2.7168156

Question 3: For what customer, and what are the different values, where the first service time diverges between the two runs?

The service time for the 5th customer is where we see different values, where for the first run, we get a service time of 0.5704503 and for the second run, we get a service time of 0.388547

Question 4: In the first run, how many arrivals occur after the fourth arrival, but before that fourth customer completes service?

```
par(mfrow = c(2,1))  
  
indices = seq_along(output1$numInSystemT[output1$numInSystemT <= 5])  
plot(output1$numInSystemT[indices], output1$numInSystemN[indices], type = "s", xlim = c(0,5), bty = "n")  
abline(v = 4.562, col = "red", lwd=3, lty=2)  
  
indices = seq_along(output2$numInSystemT[output2$numInSystemT <= 5])  
plot(output2$numInSystemT[indices], output2$numInSystemN[indices], type = "s", xlim = c(0,5), bty = "n")  
abline(v = 4.2, col = "red", lwd=3, lty=2)
```



Hw3_files/figure-latex/unnamed-chunk-1-1.pdf

2 arrivals

Question 5: In the second run, how many arrivals occur after the fourth arrival, but before that fourth customer completes service?

3 arrivals

Question 6: What are the arrival times ($\text{rate} = 1$) generated using `rexp` (no streams)?

```
set.seed(8675309)
rexp(1,rate = 1)
```

```
## [1] 1.662163
```

```
rexp(1,rate = 1)
```

```
## [1] 1.223265
```

```
rexp(1,rate = 1)
```

```
## [1] 0.7673412
```

What are the service times ($\text{rate} = 10/9$) generated using `rexp` (no streams)?

```
set.seed(8675309)
rexp(1,rate = 10/9)
```

```
## [1] 1.495947
```

```
rexp(1,rate = 10/9)
```

```
## [1] 1.100939
```

```
rexp(1,rate = 10/9)
```

```
## [1] 0.6906071
```

Now what are the arrival times ($\text{rate} = 1$) generated using `rexp` (no streams)?

```
set.seed(8675309)
a1 = rexp(1,rate = 1)
s1 = rexp(1,rate = 10/9)
a2 = rexp(1,rate = 1)
s2 = rexp(1,rate = 10/9)
a3 = rexp(1,rate = 1)
s3 = rexp(1,rate = 10/9)
```

The arrival times generated using rexp is 1.6621634, 0.7673412, and 0.3085615.

Now what are the service times (rate = 10/9) generated using rexp (no streams)?

The service times generated using rexp is 1.1009387, 0.3114826, and 0.6088186.

What are the arrival times (rate = 1) generated using vexp (with streams)?

```
set.seed(8675309)
vexp(1,rate = 1, stream = 1)
```

```
## [1] 0.2334595
```

```
vexp(1,rate = 1, stream = 1)
```

```
## [1] 1.07828
```

```
vexp(1,rate = 1, stream = 1)
```

```
## [1] 0.7855611
```

What are the service times (rate = 10/9) generated using vexp (with streams)?

```
set.seed(8675309)
vexp(1,rate = 10/9, stream = 2)
```

```
## [1] 0.0305915
```

```
vexp(1,rate = 10/9, stream = 2)
```

```
## [1] 0.1224158
```

```
vexp(1,rate = 10/9, stream = 2)
```

```
## [1] 1.42499
```

Now what are the arrival times (rate = 1) generated using vexp (with streams)?

```

set.seed(8675309)
a4 = vexp(1,rate = 1, stream = 1)
s4 = vexp(1,rate = 10/9, stream = 2)
a5 = vexp(1,rate = 1, stream = 1)
s5 = vexp(1,rate = 10/9, stream = 2)
a6 = vexp(1,rate = 1, stream = 1)
s6 = vexp(1,rate = 10/9, stream = 2)

```

The arrival times generated using vexp is 0.2334595, 1.0782802, and 0.7855611.

Now what are the service times (rate = 10/9) generated using vexp (with streams)?

The arrival times generated using vexp is 0.0305915, 0.1224158, and 1.4249899.

What is the result of the following?

```

myArr1ws <- function() {vexp(1, rate = 1, stream = 1)}
myArr2ws <- function() {vexp(1, rate = 11/10, stream = 1)}
mySvc1ws <- function() {vexp(1, rate = 10/9, stream = 2)}

#using first arrival rate
output1ws <- ssq(maxArrivals = 20, seed = 8675309, interarrivalFcn = myArr1ws(), serviceFcn = mySvc1ws())

#using second arrival rate, yet the same service function
output2ws <- ssq(maxArrivals = 20, seed = 8675309, interarrivalFcn = myArr2ws(), serviceFcn = mySvc1ws())

interarrival1 <- output1ws$interarrivalTimes
service1 <- output1ws$serviceTimes

interarrival2 <- output2ws$interarrivalTimes
service2 <- output2ws$serviceTimes

sum(output1ws$interarrivalTimes - output2ws$interarrivalTimes)

## [1] 0

```

What is the result of the following?

```

sum(output2ws$serviceTimes - output2ws$serviceTimes)

## [1] 0

```

Reflection / Investigation:

What is the default RNG used by Python? What is its period? List your source(s).

The default RNG used by Python is Mersenne - Twister, with a period of $2^{19937} - 1$ (Source).

What is the default RNG used by R? What is its period? List your source(s).

The default RNG used by R is also Mersenne - Twister, with the same period (Source).

What additional RNGs are available in Python, and how do you set up code to use them? List your source(s).

According to this source, the Permuted Congruential Generator and the PCG-64 DXSM, Philox Counter-based RNG, SFC64 Small Fast Chaotic PRNG are available in Python, you can see the `numpy.random.[generator that you want to use]` class to use the respective generators.

What additional RNGs are available in R, and how do you set up code to use them? List your source(s).

According to this source, Super-Duper, Wichmann-Hill, Marsaglia-Multicarry, KnuthTAOCP-2002, Knuth-TAOCP, L'Ecuyer-CMRG, etc, are also RNG'S available to use in R and you can use `RNGkind()` to alter the algorithm. R also has utility functions that you can use to work with RNGs, using the `rngtools` package (Source).

Given the article posted to Lyceum last class, which of the RNGs you mention in 1-4 above are presented in the article, and how does the article describe and/or rank their relative “goodness”? Include pointers to specific passages in the article.

Only the PCG-32, and Mersenne Twister RNGS are mentioned in the article. The article also classified into different types of PRNG's (LFSR-based and LCG Based PRNG's) and the those two are in two different classes. Looking at page 46, both RNG's have great 1st Level rankings for their respective type of RNG. But for the final ranking, MT19937-64 is 4th and PCG 32 is 5th, which are good but aren't the best according to their criteria. The article includes PCG-32, so I'm sure that PCG-64 (the one Python uses) is faster but not drastically.