dgiron3

Homework 1 Report

1. Get Familiar with R

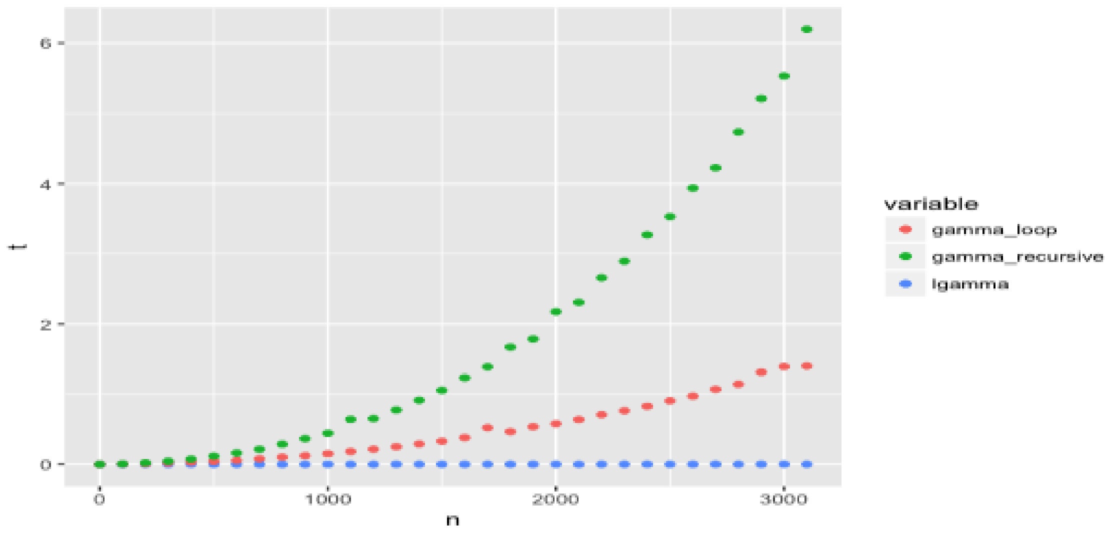
A cool insight I have learned about R in my observations is that coding in R is very flexible. What I mean by that is that there are multiple ways to output the same thing. This code snippet and output is one example that shows you can create the same vector with the same function, but using different parameters. w1 was created by passing a "by = 0.1" parameter. This function tells the program to create a vector from 0 to 1 and increment each value (starting at 0) by = 0.1 until 1 is reached. In turn, a vector with 11 values is created. w2 was created by passing a "length.out = 11" parameter. This function tells the program to create a vector from 0 to 1 and have a total of 11 equally spaced values between 0 and 1. In turn a vector of values are created and each value is spaced out by = 0.1.

```
w1 = seq(0,1, by = 0.1)
w2 = seq(0,1, length.out = 11)
stopifnot(w1 == w2)
print(w1)

> w1 = seq(0,1, by = 0.1)
> w2 = seq(0,1, length.out = 11)
> stopifnot(w1 == w2)
> print(w1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

5. Compare Results to Built-In R Function

This was an interesting exercise. In short I conducted three different experiments. The first experiment was to try and compare all three functions until the recursion function overflowed (which happened around where n = 3164). The below plot corresponds with the data frame titled "df" from the script *hw1.r:*



In this experiment, I used equal increments of n by 100 and outputted the runtimes using the system.time() function. As you can see, the recursive function has the worst performance. Another observation is that the lgamma function was not phased in terms of runtime for any n value between 1 and 3100. This lead me to look more closely into the R lgamma function and loop function performances (uncomment the second section to produce data frame titled “df2”:

```
> df2
      n lgamma gamma_loop
1 1e+00  0.000      0.000
2 1e+01  0.000      0.000
3 1e+02  0.000      0.003
4 1e+03  0.001      0.160
5 1e+04  0.008     14.388
6 1e+05  0.036   1430.369
```

This code increments n exponentially where each following in value is 10x the previous. As was the case with the previous experiment, the lgamma function was barely phased (even when n = 100,000). However, the looping function took much longer at n = 100,000 (almost 24 minutes). Plotting this did not make sense any further because lgamma was basically zero in runtime while the gamma_loop function increased exponentially. This lead me to do one more final experiment to just look into the R function lgamma(). The results are displayed in data frame “df3”:

```
> df3
       n  lgamma
1  1e+00   0.000
2  1e+01   0.000
3  1e+02   0.000
4  1e+03   0.001
5  1e+04   0.005
6  1e+05   0.045
7  1e+06   0.355
8  1e+07   3.184
9  1e+08  31.031
10 1e+09 310.568
```

Fascinatinly enough, lgamma started to show its first sign of significant slow down when n = 10,000,000. Even then t only was equal to 3 seconds. When n = 1,000,000,000, it was still faster than gamma_loop when n = 100,000. That is a significant advantage for the built in R function.

Overall, the R function was the best performer of the 3 functions. The recursive function was the most limiting and the loop function fell somewhere in between.