hyang390
903320189

Homework 1: R Programming

1. Get Familiar with R

One of the most interesting insights I learned about R is the importance of vectorizing code and avoiding the use of loops. As stated from the required reading and the course lecture, vectorized code reduces complexity in the code and greatly speeds up the computational speed, sometimes hundreds of magnitudes faster than using for loops. We can demo this with the below example:

```
a = 1:15
custom_add = function(n) {
  result = 0
  for (i in seq_along(n)) {
    result = result + (n[i]^3 + log(n[i]))
  }
  return(result)
}

custom_add(a)
sum(a^3 + log(a))
```

```
> custom_add(a)
[1] 14427.9
> sum(a^3 + log(a))
[1] 14427.9
```

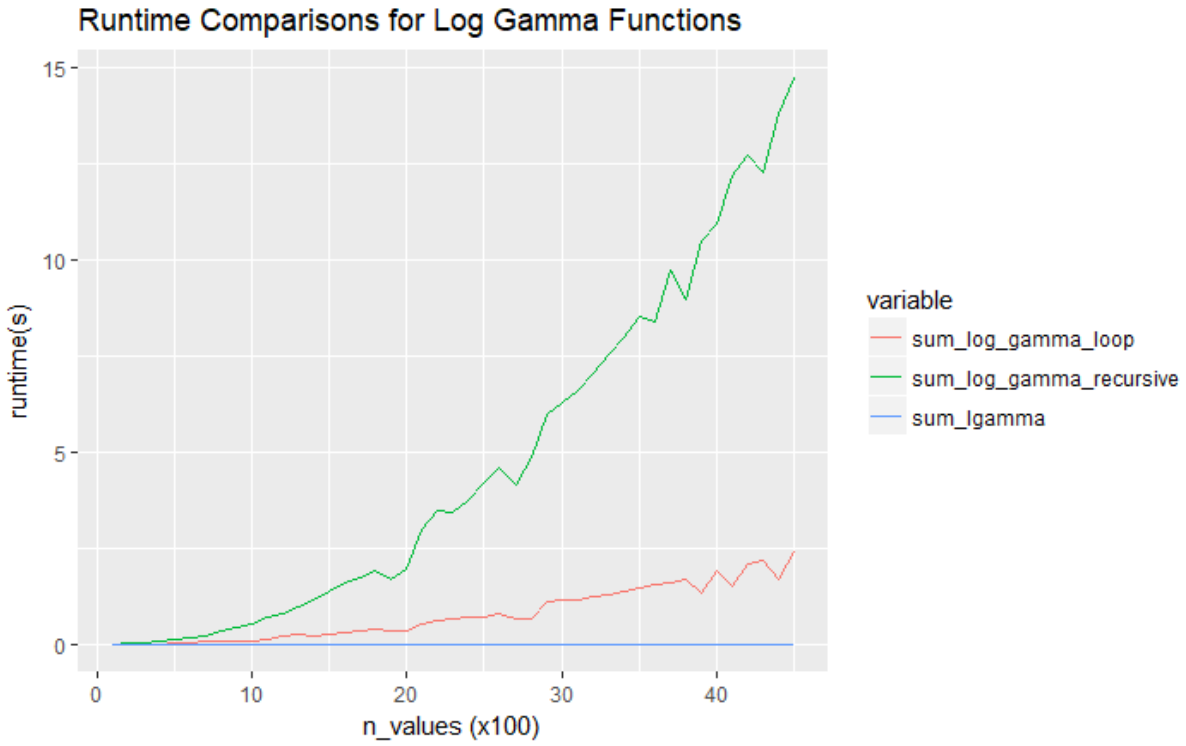
The function *custom_add* is a function that takes a vector of numbers and calculates the sum of calculating each number to the power of three added to the natural logarithm of each number. The same computation could also be done by calling the *sum* function that is built into *R* and passing in the logic, this will do element wise calculation and then conduct the sum. We can see that with a defined as a sequence from 1 to 15, the results of calling both are the same. Now, if the time is calculated for a sequence from 1 to 1 million, the times are about as follows:

```
> a = 1:1000000;
> system.time(custom_add(a))
   user  system elapsed
   0.29    0.00    0.30
> system.time(sum(a^3 + log(a)))
   user  system elapsed
   0.09    0.00    0.09
```

As we can see, the vectorized version is much faster than the for loop custom function, this makes sense as both the power and log is done element-wise and the sum then adds up all of them. In addition, note that the custom function is 7 lines whereas the vectorized code is only one line, it is much more succinct and clean.

hyang390
903320189

5. Compare Results to Built-In R Function



The above plot shows the runtime of the three functions with n values ranging from 1 to 4500 with runtimes calculated from using *system.time*. As clearly shown above, the slowest implementation is the recursive implementation, followed by the iterative loop iteration and with the built in lgamma implementation being the fastest of the three. From the observations of the runtimes, it appears that both *sum_log_gamma_loop* as well as *sum_log_gamma_recursive* have a runtime of $O(n^2)$, with the recursive function being slower than the iterative loop, hence the increase in runtime as values of n increases. The built-in function in *sum_lgamma* seems to be running in linear time $O(n)$ as values of n approaches larger values, but the runtime compared to the other two functions is exponentially faster.