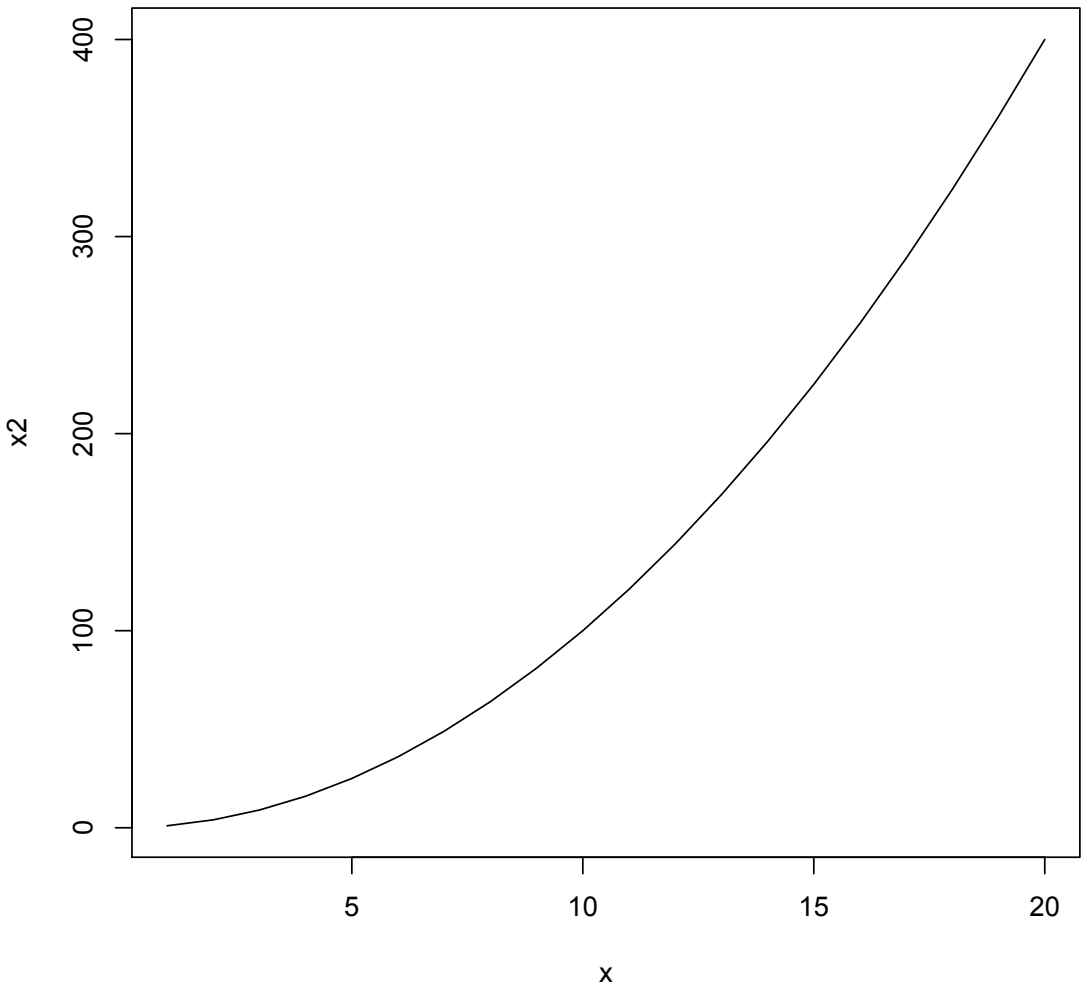
Question 1: Get Familiar with R

One insight I learned about R in my observations is R could not only take values as arguments of functions, but take expressions as input for functions as well.

```
x <- seq(1,20)
x2 <- x * x
plot(x, x2, type = "l")
```



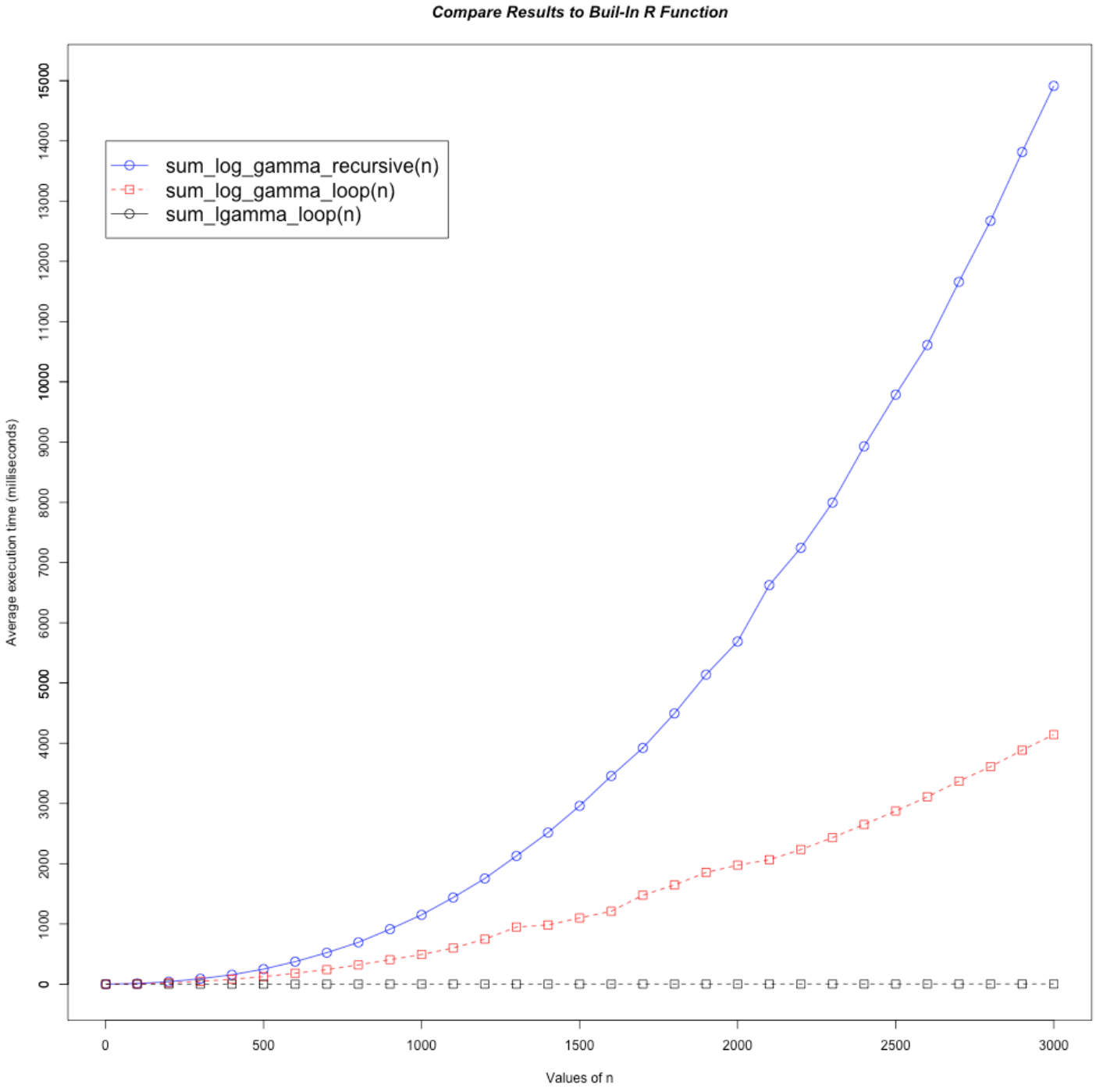
From the above figure we could see R automatically knows the variable on the x axis is called x and the variable on the y axis is called x2. In another word, R has the capability to access the code used to calculate the values of arguments of a function. And this feature makes it possible for the users to evaluate R code in non-standard ways (NSE). The reason R could do it is because the arguments of R functions belong to a special type of object called a promise. A promise collects the information of the expression computing the values of arguments, along with the information of environment in which the values of arguments were computed. Based on this, we could use substitute() function to capture the R codes used to compute the values of arguments:

```
f <- function(x) { substitute(x) }
x1 <- 10
x2 <- 12
print(f(x1 + x2 ^ 2))
## Output: x1 + x2^2
```

This feature is very useful since we could use it to dramatically reduce the coding time of carrying out interactive data analysis.

Question 5. Compare Results to Built-In R Function

I first chose n = 1 and select n from an arithmetic sequence (100, 200, 300, … 3000) with common difference of 100. And for every value of n, I run each of the 3 implementations 20 times and calculate the average of execution time (“user time” in system.time() output, unit: milliseconds) to smooth the curve. The results were summarized in the following figure:



From the above figure we could see with increasing the value of n, the implementation of sum_log_gamma_recursive(n) grows fastest and the implementation of sum_lgamma_loop(n) grows

slowest with respect to the average execution time. The growing speed of average execution time of sum_log_gamma_loop(n) is lower than that of sum_log_gamma_recursive(n) and higher than that of sum_lgamma_loop(n).

Optional Question:

The asymptotic time complexity (in Big-O notation) of each of the 3 implementations are as followings:

sum_lgamma_loop(n): $O(n)$

sum_log_gamma_loop(n): $O(n^2)$

sum_log_gamma_recursive(n): $O(n^3)$