Title: hw1_report

Date: September 10, 2017

1. Get Familiar with R [10 points]

R is very convenient to visualize and manipulate multidemtional data. Although the execution time is slower than C++, it able to implement C++ in R programming. Various high-quality packages are available for R, which further enhance the capability of R for data analysis. In addition, R is very good at generating high quality graphics for data visualization.

I think the structure of R programing is very similar to other language. We are able to write for loop, if-else, while loop, and functions in R. R has its own built-in function as well. It is very convenient to call R-building functions. Functions in R are also very similar to other language. Each parameter in function has been defined with default value. When you call a function in R, you only need to call the function along with the parameter and its values you want to change. The rest parameters will take default value.

The coding of if-else, while loop and for loop and logical operators in R are similar to these in Java. Here is an example in R using while loop to repeatedly calculate sm=sm+1 under the condition b>1

```
----------------
sm=0
a=1;b=10
while(b>a){
  sm=sm+1
  a=a+1
  b=b-1
}
Sm
#[1] 5
---------------
```

The initial valve for sm, a, and b are 0, 1 and 10, respectively. “b>a” generates a bealoon value of TRUE or FALSE. The while loop will repeatedly calculate sm=sm+1; a=a+1; b=b-1 when the bealoon value of “b>a” is TRUE. One major difference between R and Java program is that each line has to be finished with “;” in Java, but R doesn’t have such requirement.

2. Log Gamma (Loop) [20 points]

#Code: Function log_gamma_loop (n)

```
log_gamma_loop=function(n=5){
result=0
```

```
if(n==1){
 result=0
 return (result)
}else{
  for(var in seq(1,(n-1),by=1)){
    temp = log(var)
    result = result+temp
  }
  return (result)
  }
}
print(log_gamma_loop(5))
```

3. Log Gamma (Recursive) [20 points]

#Code: log_gamma_recursive (n)

```
log_gamma_recursive=function(n=5){
    if(n==1) result=0
    else  result=log(n-1)+log_gamma_recursive(n-1)
    return (result)
}
```

4. Sum of Log Gamma [20 points]

#Code: Function sum_log_gamma_loop(n):

```
sum_log_gamma_loop=function(n=7){
  sum_log_gamma_loop=0
  for(var in seq(1,n,by=1)){
    temp=log_gamma_loop(var)
    sum_log_gamma_loop=sum_log_gamma_loop+temp
    }
  return(sum_log_gamma_loop)
}
```

#Code: Function sum_log_gamma_recursive(n)

```
sum_log_gamma_recursive=function(n=7){
  sum_log_gamma_recursive=0
  for(var in seq(1,n,by=1)){
```

```
   temp=log_gamma_recursive(var)
   sum_log_gamma_recursive=sum_log_gamma_recursive+temp
  }
  return(sum_log_gamma_recursive)
}
```

5. Compare Results to Built-In R Function [30 points]

 # Code: Implementation of sum_lgamma(n) using built-in lgamma(n) function.

```
sum_lgamma=function(n=7){
  sum_lgamma=0
  for(var in seq(1,n, by=1)){
    temp=lgamma(var)
    sum_lgamma=sum_lgamma+temp
  }
    return (sum_lgamma)
}
```

#Code to call the three summation functions with increasing values of n over a reasonable range and measure execution times.

```
max_n=3000
x=seq(200,max_n,by=200)
```

#calculate user time for equation sum_lgamma with n= 200, 400, ..., 3000

```
user_sum_lgamma=x
for (var in seq(200,max_n,by=200)){
  i=var/200
  user_sum_lgamma[i]=system.time(sum_lgamma(var))[1]
  }
```

#calculate user time for equation sum_log_gamma_loop n= 200, 400, ..., 3000

```
user_sum_log_gamma_loop=x
for (var in seq(200,max_n,by=200)){
  i=var/200
  user_sum_log_gamma_loop[i]=system.time(sum_log_gamma_loop(var))[1]
  }
```

#calculate elapsed time for equation _sum_log_gamma_recursive n= 200, 400, ..., 3000

```
user_sum_log_gamma_recursive=x
for (var in seq(200,max_n,by=200)){
   i=var/200     user_sum_log_gamma_recursive[i]=system.time(sum_log_gamma_recursive(var))
   [1]
}
```

#plot different system of three function with increase of n values

```
plot(x,user_sum_log_gamma_recursive,type="o", main="Figure 1: Comparision of execution time for three functions",col="red", xlab="integer n", ylab = "Time (s)",pch = 8)
points(x,user_sum_log_gamma_loop,type="o",col="green",pch = 8)
points(x,user_sum_lgamma,type="o",col="blue",pch = 8)
legend("topleft",
c("sum_log_gamma_recursive","sum_log_gamma_loop","sum_lgamma"),lty=1,col=c("red","green","blue"),pch = 8)
```

Report: A brief writeup explaining my observations, with a plot of running times for comparison.

I have run the system.time function with a series of n values from 200 to 3000 [seq(200, 3000, by=200)] , and plotted each user time with the corresponding n values.

As shown in Figure 1, the red line (stand for user time of sum_log_gamma_recursive(n)) grows fastest with increasing n and the blue line (stand for user time of sum_lgamma(n)) grows slowest with increasing n. Thus, sum_log_gamma_recursive(n) takes the longest time to calculate and sum_lgamma(n) take shortest time to calculate.