Lab3

SAF

2025-04-17

Part 1: Iterations

Question 1 - *Creating For and While loops (using large_vector data set)*

1A. Use a while loop to assign a new variable, mean_while, as the mean of large_vector. Use print to display this value.

```
count <- 1 #Set the initial count start
countLim <- length(large_vector) #Set the count limit
Sum <- 0 #intializes the sum value at zero

#while loop adds each vector element as it cycles through and exits when count is complete
while(count <= countLim)
{
  Sum <- Sum + large_vector[count] #Keeps running sum of each vector element.
  count <- count + 1 #counter for loop
}

mean_while <- Sum/length(large_vector) #Divides the sum of all the vector elements by the vectors lengt
print("1A. mean of the large_vector using while loop ") #Mean of the vector elements
```

```
## [1] "1A. mean of the large_vector using while loop "
```

```
print(mean_while)
```

```
## [1] 7.900514
```

1B. Use a for loop to assign a new variable, mean_for, as the mean of large_vector. Use print to display this value.

```
count <- 1 #Set the initial count start
countLim <- length(large_vector) #Set the count limit

#For loop adds each vector element as it cycles through and exits when count is complete
for(i in count:countLim)
{
  Sum <- Sum + large_vector[count] #Keeps running sum of each vector element.
  count <- count + 1 #counter for loop
}
mean_while <- Sum/length(large_vector) #Divides the sum of all the vector elements by the vectors lengt
print("1B. mean of the large_vector using for loop") #Mean of the vector elements
```

```
## [1] "1B. mean of the large_vector using for loop"
```

```
print(mean_while) #Mean of the vector elements
```

```
## [1] 15.80103
```

1C. How are these methods similar and how might vectorized functions be superior?

These two loops, for and while, using similar methods but with different count validation statements at the top of each loop. Where the for loop uses (i in count:countLim) for iteration checks and the while loop uses (count <= countLim) to validate the loop count.

One example of how vector functions are superior are how they can be mathematically manipulate large amounts of elements within the vector. For example, the vector "large_vector" holds 2,461,837 elements or numbers. A simple for/while loop is able to quickly run a summation of each vector element and find isn't mean.

Question 2 - *Loops vs vectorized functions*

2A. *Use vectorized functions to assign a new variable, norm_vectorized, using the normalization described in the question stem. For this question, it is permissible to use vectorized functions to verify that the normalization was done correctly.*

```
tic() # Starts timer

#Eqn-normailize: NormVect = large_vector/(sqrt(sum(large_vector*large_vector)))
#square the vector, then sum the elements, and take the square root.
sqrtsum <- sqrt(sum(large_vector^2)) #Normalized magnitude variable

norm_vectorized <- large_vector/sqrtsum #normalized vector
print("2A. Using vectorized functions to Normalize large_vector.")
```

```
## [1] "2A. Using vectorized functions to Normalize large_vector."
```

```
#print(norm_vectorized)

message("Time lapse")
```

```
## Time lapse
```

```
toc() # Records current timer and computes elapsed time since the matching call to tic()
```

```
## 0.06 sec elapsed
```

2B. *Use a for loop to assign a new variable, norm_loop, using the normalization described in the question stem. For this question, it IS NOT permissible to use vectorized function to calculate total sum (use a for loop).*

```
tic() # Starts timer

#Eqn-normailize: NormVect = large_vector/(sqrt(sum(large_vector*large_vector)))
#initialized variables used in for loop
cnt <- 1 #Set the initial count start
cntLim <- length(large_vector) #Set the loop count limit
SumSqr <- 0 #initializes the sum value at zero

#For loop adds each vector element as it cycles through and exits when count is complete
for(i in cnt:cntLim)
{SumSqr <- SumSqr + large_vector[i]*large_vector[i]} #Keeps running sum of each vector element.

MagComp <- sqrt(SumSqr)#Magnitude component, Create the square root of the squared sum values.
norm_vectorized <- large_vector/MagComp #normalized vector
#print(norm_vectorized) #Prints out normalized vector
```

```r
message("Time lapse")
```

```
## Time lapse
```

```r
toc() # Records current timer and computes elapsed time since the matching call to tic()
```

```
## 0.13 sec elapsed
```

2C. *Please explain the differences between these methods in terms of implementation, readability, and computation time. Using your own experiences in the workplace (or speculation of experiences for a job you would like to have), how might these differences be significant and what impact would they have? Method 2B. using the for loop takes roughly twice as long. Larger data sets could easily take 30-60mins to process,*

Question 3 - *Apply vs loops*

3A. *Use apply to assign a new variable, mean_list_apply, the mean of each vector in large_list.*

```r
tic() # Starts timer

VectLength <- length(large_list[[1]]) #In this particular list every vector length is equal at 2,461,83

#Converts all three sub-vectors from large_list into three separate columns.
AllMatrix <- as.matrix(as.data.frame(large_list)) #A column for each sub vector (sample A, sample B,sam
colTotal <- apply(AllMatrix, 1, sum) #Sums up each of the three vector columns, produces a vector of ma
rowTotal <- apply(AllMatrix, 2, sum) #Sums up all three column sums


#Sample A matrix
SumofList <- rowTotal[1] #loads sum of Sample A matrix
MeanAvect <- SumofList/VectLength #Deriving the mean value
message("Mean of Sample A matrix: ", MeanAvect)
```

```
## Mean of Sample A matrix: 7.90051412827088
```

```r
#Sample B matrix
SumofList <- rowTotal[2] #loads sum of Sample A matrix
MeanBvect <- SumofList/VectLength #Deriving the mean value
message("Mean of Sample B matrix: ", MeanBvect)
```

```
## Mean of Sample B matrix: 12.8120342654692
```

```r
#Sample C matrix
SumofList <- rowTotal[3] #loads sum of Sample A matrix
MeanCvect <- SumofList/VectLength #Deriving the mean value
message("Mean of Sample C matrix: ", MeanCvect)
```

```
## Mean of Sample C matrix: 9.0484057230434
```

```r
message("Time lapse")
```

```
## Time lapse
```

```r
toc() # Records current timer and computes elapsed time since the matching call to tic()
```

```
## 5.43 sec elapsed
```

3B. *Use for loop to assign a new variable, mean_list_for, the mean of each vector in large_list.*

```r
tic() # Starts timer
```

```
cnt <- 1 #Set the initial count start integer
limit <- length(large_list)
TotSum <- 0 #initializes the TotSum value at zero
TotalLength <- 0

#For loop adds each vector element as it cycles through and exits when count is complete
for(i in cnt:limit)
{
  mean_list_for = ((sum(large_list[[i]]))/(length(large_list[[i]])))
  message("Mean for list.vector ", i,": ", mean_list_for)
}
```

```
## Mean for list.vector 1: 7.90051412827088
```

```
## Mean for list.vector 2: 12.8120342654692
```

```
## Mean for list.vector 3: 9.0484057230434
```

```
message("Time lapse")
```

```
## Time lapse
```

```
toc() # Records current timer and computes elapsed time since the matching call to tic()
```

```
## 0.01 sec elapsed
```

3C. *Please explain the differences between these methods in terms of implementation, readability, and computation time. Were these differences surprising? How do you predict these differences will change if we used lists with much more elements (i.e. length(list) > 1000)? The difference in processing time between 3A. and 3B. is 10x the amount. Using the apply() method took dramatically longer than simply using a for loop and R vector functions. If we were to work with length(list) greater than 1000 the workstation could be processing the list of data for more than an hour*

Part 2: Control

Question 4 - *Doing it the Un-R way*

4A. *Use the Un-R way, with a for loop and if/else statements, to assign a new variable, nbr_zeros_loop, as the number of zeros that are in large_vector. Print the value of this variable.* **4B.** *Use the Un-R way, with a for loop and if/else statements, to assign a new variable, smaller_vector_loop, as all the values of large_vector that are not equal to zero. Vectorized functions (i.e. na.omit) are permitted but a for loop must be used to iterate through vector. Show that the sum of nbr_zeros_loop and the length of smaller_vector_loop are equal to the length of larger_vector.*

```
tic() # Starts timer

#use while loop to sum all the vector elements.
cnt <- 1 #Set the initial count start
icntLim <- length(large_list[]) #Set the count limit, outer loop
nbr_zeros_loop = 0
smaller_vector_loop = 0

for(i in cnt:icntLim)
{
jcntLim <- length(large_list[[i]])
    for(j in cnt:jcntLim)
    {
```

```
      ZeroChk <- large_list[[i]][[j]] #loads the each element of each of the 3 vectors.

      if(ZeroChk == 0)
        {nbr_zeros_loop = nbr_zeros_loop + 1} #add to zero counter
      else
        {smaller_vector_loop = smaller_vector_loop + 1} #add to non-zero counter
    }
}

message("4A. Zero's counted: ", nbr_zeros_loop)
```

```
## 4A. Zero's counted: 3537222
```

```
message("4B. Non-zero's counted: ", smaller_vector_loop)
```

```
## 4B. Non-zero's counted: 3848289
```

```
message("Zero + non-zero values counted: ", (smaller_vector_loop + nbr_zeros_loop))
```

```
## Zero + non-zero values counted: 7385511
```

```
message("CHECK! total function length of entire list of vectors: ", (3*jcntLim))
```

```
## CHECK! total function length of entire list of vectors: 7385511
```

```
message("Time lapse")
```

```
## Time lapse
```

```
toc() # Records current timer and computes elapsed time since the matching call to tic()
```

```
## 0.62 sec elapsed
```

Question 5 - *Doing it the R-way* use the large_vector data set.

5A. *Use the R way, proper subsetting and vectorized functions, to assign a new variable, nbr_zeros, as the number of zeros that are in large_vector. Print this variable. Show that nbr_zeros and nbr_zeros_loop are equal.*

5B. *Use the R way, proper subsetting and vectorized functions, to assign a new variable, smaller_vector, as all the values of large_vector that are not equal to zero. Show that the sum of nbr_zeros and the length of smaller_vector are equal to the length of larger_vector.*

```
tic() # Starts timer

#Sum of all zeros in Sample A, Sample B, Sample C matrices
qtyZeros <- length(which(large_list[[1]] == 0)) +  length(which(large_list[[2]] == 0)) + length(which(la
message("Quantity of zeros in large_list: ", qtyZeros)
```

```
## Quantity of zeros in large_list: 3537222
```

```
#Sum of all non-zeros in Sample A, Sample B, Sample C matrices
qtyNONzeros <- length(which(large_list[[1]] != 0)) +  length(which(large_list[[2]] != 0)) + length(whicl
message("Quantity of non-zeros in large_list: ", qtyNONzeros)
```

```
## Quantity of non-zeros in large_list: 3848289
```

```
message("CHECK! Total: ", qtyZeros + qtyNONzeros)
```

```
## CHECK! Total: 7385511
```

```r
message("Time lapse")
```

```
## Time lapse
```

```r
toc() # Records current timer and computes elapsed time since the matching call to tic()
```

```
## 0.18 sec elapsed
```