Computer_Assignment1

KhanhLinhNguyen

9/10/2020

1. Reading and manipulating DNA sequences:

a

Load the packages

```r
library("ape", quietly = T)
library("Biostrings", quietly = T)
```

Read DNA string

```r
# Use 'read.GenBank' function from Biostrings package
# It reads the DNA directly from GenBank and converts it as a 'DNAString'
seq <-read.GenBank("MF770243.1", as.character = TRUE)[[1]]
seqStr <- paste(seq, sep="", collapse="")
DNAstr <- DNAString(seqStr)


# 1. Count length of the sequence
sequenceLength <- length(DNAstr)


# 2. Count alphabet frequency and pick 'T'


freq = alphabetFrequency(DNAstr)[DNA_BASES]
Tsequence <- freq['T']

# 3. The number of nucleotide C at base positions 100 to 200

DNAstr100to200 = substr(DNAstr, 100, 200)
freq100to200 = alphabetFrequency(DNAstr100to200)[DNA_BASES]
Csequence <- freq100to200['C']
```

Length of the sequence: 16851.
The number of nucleotide T in the sequence: 4794.
The number of nucleotide C at base positions 100 to 200: 29.

2. General use of R

a. Vectors, matrices and lists in R

i. Basic operations

```r
x = c(2,3,1)
y = c(3,5,2)
# 1. Modify the vector x by adding 3 to it
for (i in 1 : length(x))
  x[i] <- x[i] + 3
# Print X
print(x)
```

```
## [1] 5 6 4
```

```r
# 2. Modify y by raising each of its elements to the power of 2
for (i in 1 : length(y))
  y[i] <- y[i] ^ 2
# Print Y
print(y)
```

```
## [1]  9 25  4
```

```r
# 3. Create a matrix which has x as its first column and y as its second column
matrix <- matrix(c(x,y), ncol = 2)

# Print matrix
print(matrix)
```

```
##      [,1] [,2]
## [1,]    5    9
## [2,]    6   25
## [3,]    4    4
```

```r
# 4. Create a new matrix A by multiplying x with the transpose of y
A <- x %*% t(y)

# Print matrix A
print (A)
```

```
##      [,1] [,2] [,3]
## [1,]   45  125   20
## [2,]   54  150   24
## [3,]   36  100   16
```

```r
# 5. Create a random matrix B consisting of integer elements that are drawn
# from a discrete uniform distribution from 1 to 10, of the same size as A. (help(sample))
v <- c(1,2,3,4,5,6,7,8,9,10)
```

```r
B <- matrix(replicate(3,(sample(v,3))), ncol = 3, nrow = 3)

# Print B
print(B)
```

```
##      [,1] [,2] [,3]
## [1,]    4    3    5
## [2,]    5    8    1
## [3,]    3    5    8
```

```r
# 6. Perform element-wise multiplication between A and B.
print (A*B)
```

```
##      [,1] [,2] [,3]
## [1,]  180  375  100
## [2,]  270 1200   24
## [3,]  108  500  128
```

```r
# 6. Perform matrix multiplication between A and B.
print (A %*% B)
```

```
##      [,1] [,2] [,3]
## [1,]  865 1235  510
## [2,] 1038 1482  612
## [3,]  692  988  408
```

ii. How to access elements of vectors

```r
x=c(1,2,3,4,5)

# Access element with index
x[3]
```

```
## [1] 3
```

```r
# Access a series
x[1:3]
```

```
## [1] 1 2 3
```

```r
x[1:length(x)]
```

```
## [1] 1 2 3 4 5
```

```r
x[1:length(x)-1]
```

```
## [1] 1 2 3 4
```

```
# Sequence
x[seq(5,1,-2)]
```

```
## [1] 5 3 1
```

```
#Subset
x[c(1,3,5)]
```

```
## [1] 1 3 5
```

iii. How to access elements of matrices

```
A=matrix(c(1,2,3,4,5,6),nrow=2)

A[3]
```

```
## [1] 3
```

```
A[1,]
```

```
## [1] 1 3 5
```

```
A[,2:3]
```

```
##      [,1] [,2]
## [1,]    3    5
## [2,]    4    6
```

```
A[A>3]
```

```
## [1] 4 5 6
```

iv. How to use lists

```
# This is a list.
a = list()
# An empty list can be appended as
a[[1]] = 0.0
a[[2]] = c(TRUE,FALSE)
a[['vector']] = c(1,2,3)
print(a)
```

```
## [[1]]
## [1] 0
##
## [[2]]
## [1]  TRUE FALSE
##
## $vector
## [1] 1 2 3
```

```
print(a[[1]]) # double brackets select the content of the index in a list
```

```
## [1] 0
```

```
# Three ways to access named entries in a list:
a[[3]]
```

```
## [1] 1 2 3
```

```
a[['vector']]
```

```
## [1] 1 2 3
```

```
a$vector
```

```
## [1] 1 2 3
```

```
# Get names of entries in a list
names(a)
```

```
## [1] ""       ""       "vector"
```

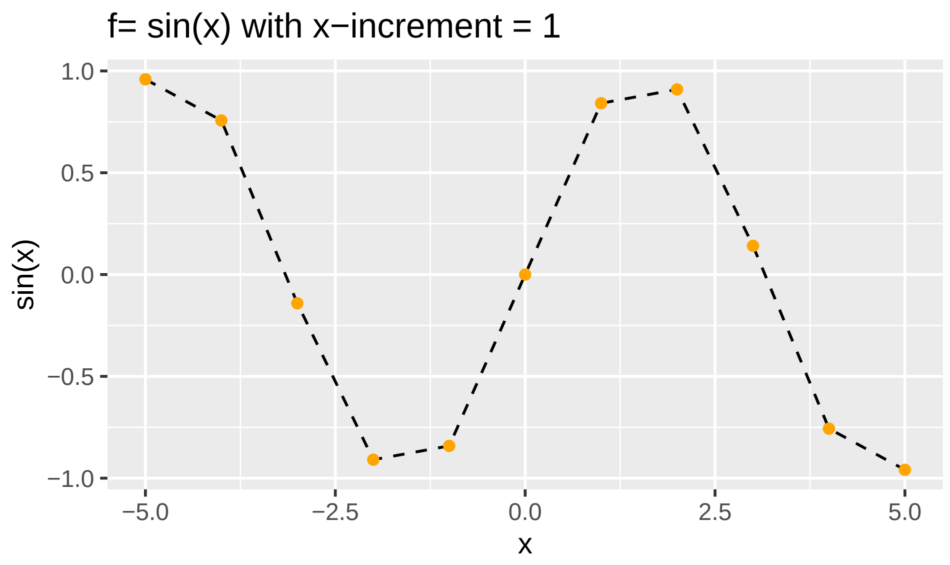
```
# Print structure of any variable (can be a list, vector,...)
str(a)
```

```
## List of 3
##  $       : num 0
##  $       : logi [1:2] TRUE FALSE
##  $ vector: num [1:3] 1 2 3
```

b. Graphics in R

Load package

```
library("ggplot2")
x = seq(-5,5)
y = sin(x)
data <- data.frame(x,y)
ggplot(data = data, aes(x = x, y = y)) +
  geom_line(linetype = "dashed") +
  geom_point( color = "orange") +
  ggtitle("f= sin(x) with x-increment = 1") +
  labs(y="sin(x)", x = "x")
```

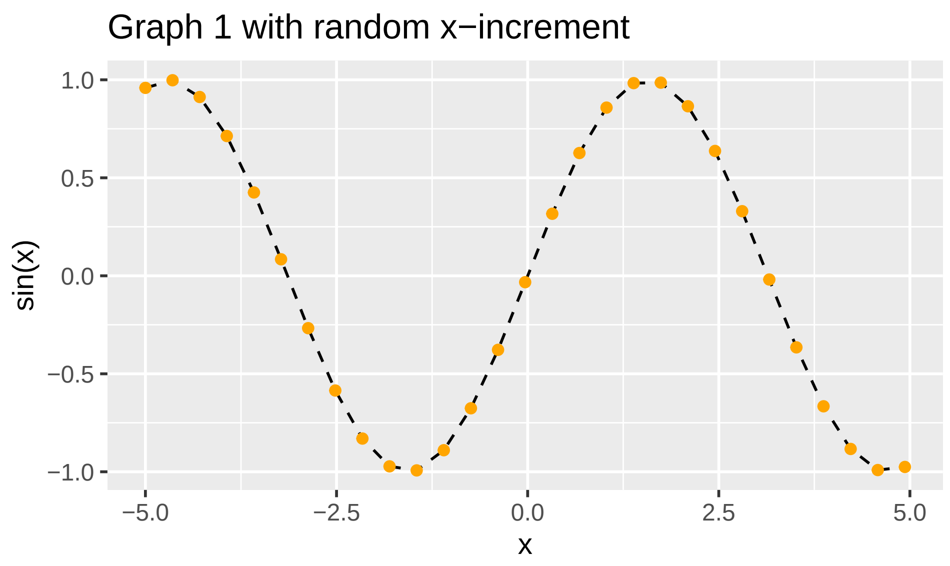


Try to change the increment with a random figure from 0 to 1

```
library("ggplot2")
randIncre = runif(1)
print(randIncre)
```

```
## [1] 0.3548082
```

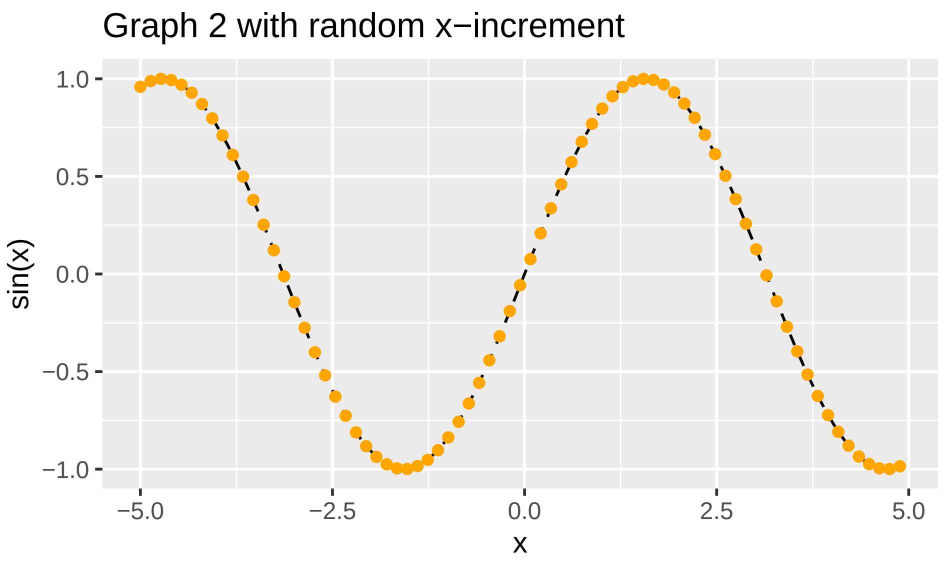
```
x = seq(-5,5, randIncre)
y = sin(x)
data <- data.frame(x,y)
ggplot(data = data, aes(x = x, y = y)) +
  geom_line(linetype = "dashed") +
  geom_point( color = "orange") +
  ggtitle("Graph 1 with random x-increment") +
  labs(y="sin(x)", x = "x")
```



```r
library("ggplot2")
randIncre = runif(1)
print(randIncre)
```

```
## [1] 0.1335831
```

```r
x = seq(-5,5, randIncre)
y = sin(x)
data <- data.frame(x,y)
ggplot(data = data, aes(x = x, y = y)) +
  geom_line(linetype = "dashed") +
  geom_point( color = "orange") +
  ggtitle("Graph 2 with random x-increment") +
  labs(y="sin(x)", x = "x")
```

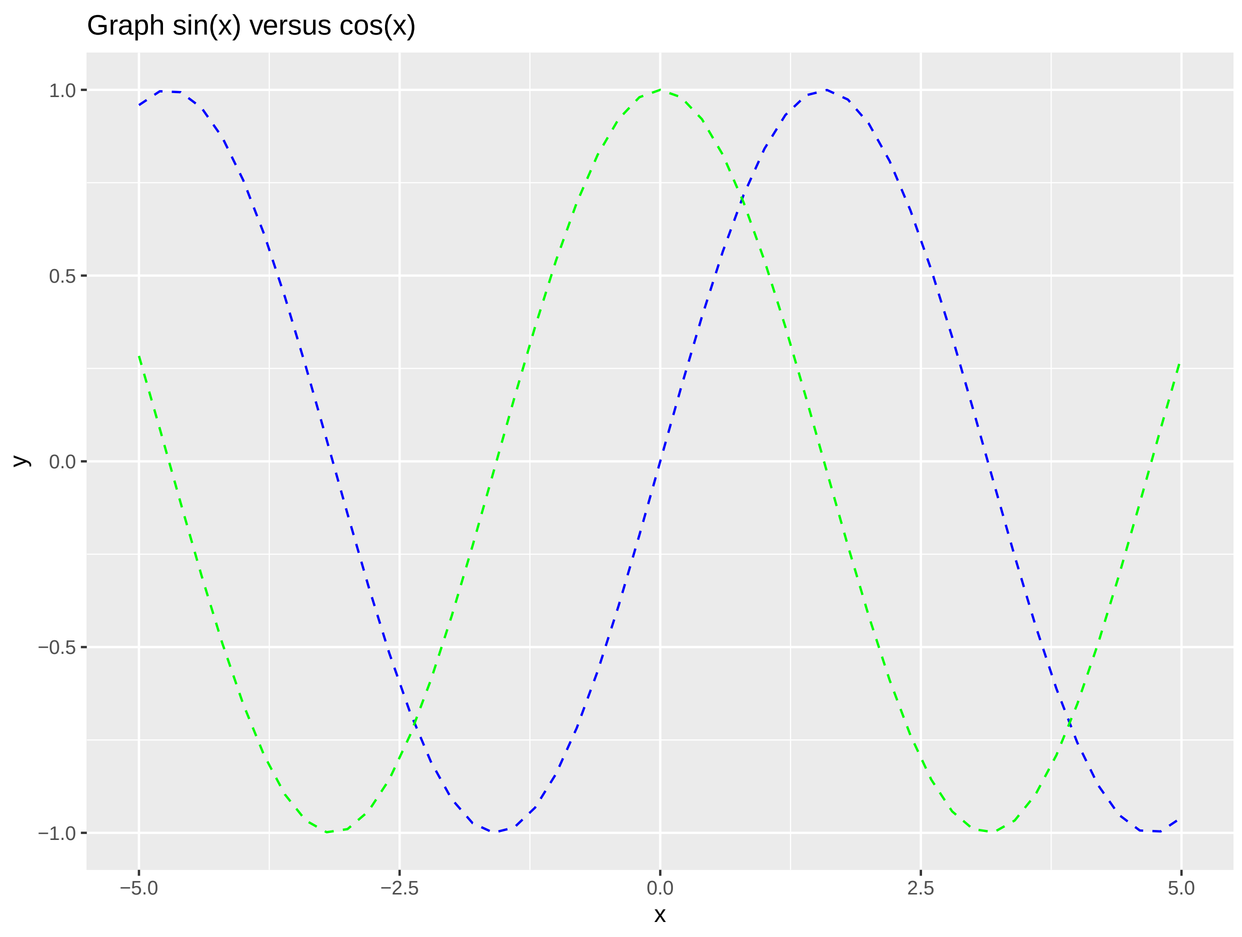


From the figures, it can be observed that the graph was presented more detailed and accurate when the increment was smaller.

As when the increment is smaller, there are more data points in the data frame which help illustrating the function (the line) better.

Add plot of a function g = cos(x) to the current figure and add legend

```r
library("ggplot2")
x = seq(-5,5,0.2)
y = sin(x)
g = cos(x)
data <- data.frame(x,y)
data2 <- data.frame(x,g)
ggplot() +
  geom_line(data = data, aes(x = x, y = y),linetype = "dashed", color="blue") +
  geom_line(data = data2, aes(x = x, y = g),linetype = "dashed", color="green") +
  geom_point( color = "orange") +
  ggtitle("Graph sin(x) versus cos(x)") +
  labs(y="y", x = "x", color = "Legend")
```



c. How to use loops

```
t = -1:5
u = c()
for (i in 1: length(t)){
  if(t[i] > (pi/2)){
      y = 1
  } else if (t[i] >= 0 & t[i] <= (pi/2) ) {
      y = sin(x)
  } else {
      y = 0
  }
  u[i] <- y
}

print(u)
```

```
## [1] 0.0000000 0.9589243 0.9589243 1.0000000 1.0000000 1.0000000 1.0000000
```

d. How to use functions

```r
calculateBMI <- function(weight, height) {
   BMI <- (weight) / (height^2)
       if(BMI < 18.5){
            category <- "underweight"
       } else if (BMI >= 18.50 & BMI < 25 ) {
            category <- "normal"
       } else if (BMI >= 25) {
            category <- "overweight"
      }
       result = list()
       result[[1]] = BMI
       result[[2]] = category
       return(result)
}

#Test function input weight in kg and height in m

calculateBMI(50, 1.6)
```

```
## [[1]]
## [1] 19.53125
##
## [[2]]
## [1] "normal"
```