First Set of Exercises to Deliver

Álvaro Ponce Cabrera

April 04, 2016

Exercise 1.

Load the following function in your R workspace

```
  generateExpressions<-function(nrow=5,ncol.1=1,mean.1=0,sd.1=1,samples.2=F,
ncol.2=NULL,mean.2=NULL,sd.2=NULL){
out<-matrix(rnorm(nrow*ncol.1,mean.1,sd.1),nrow,ncol=ncol.1)
if(samples.2){
if(is.null(ncol.2)) ncol.2<-ncol.1
if(is.null(mean.2)) mean.2<-mean.1
if(is.null(sd.2)) sd.2<-sd.1
matrix.2<-matrix(rnorm(nrow*ncol.2,mean.2,sd.2), nrow, ncol=ncol.2)
out<-cbind(out,matrix.2)
}else{
ncol.2<- 0
}
colnames(out)<-paste("sample",1:(ncol.1+ncol.2),sep=".")
rownames(out)<-paste("gene",1:nrow,sep=".")
return(out)
}
```

1.1 Invoke generateExpressions with different input parameters.

```
generateExpressions(3, 2, 12, 1)
```

```
##        sample.1 sample.2
## gene.1 13.50821 11.18989
## gene.2 12.00203 11.96079
## gene.3 12.19745 11.94699
```

```
generateExpressions(2,5,1,1,samples.2=TRUE, ncol.2=3)
```

```
##         sample.1  sample.2   sample.3   sample.4   sample.5  sample.6
## gene.1 0.1398592 0.9288157  1.1201286 1.64048815 -0.8822202 0.1965009
## gene.2 0.5311469 1.0039982 -0.5893304 0.09419956  3.4051295 1.6320580
##        sample.7   sample.8
## gene.1 1.015990 -0.1179371
## gene.2 2.611992 -1.1262021
```

1.2 Rewrite the function in order to provide a more human readable structure. That is, try to introduce the indentation, spaces and carriage-return linefeeds when they are required as well as to write appropriate comments for clarifying the action that is being performed.

```r
generateExpressions<-function(nrow=5, ncol.1=1, mean.1=0, sd.1=1, samples.2=F,
                              ncol.2=NULL, mean.2=NULL, sd.2=NULL)
  {
  # Generate a matrix with nrow and ncol indicate in the firt and second arguments of the function
  # Elements = rnorm(nrow * ncol) (first argument of matrix, generate a normal distribution with
  #mean = mean.1 and sd = sd.1, arguments of the function)
  out<-matrix (rnorm (nrow*ncol.1, mean.1, sd.1), nrow, ncol=ncol.1)

  # Data for a second matrix if samples.2 = TRUE.

  #If data of second matrix is null, the function use this data from matrix 1.
  if(samples.2){
    if(is.null(ncol.2)) ncol.2<-ncol.1
    if(is.null(mean.2)) mean.2<-mean.1
    if(is.null(sd.2)) sd.2<-sd.1

    #Creating a second matrix binded to the first one.
    #This matrix is generated exactly like the first one but with arguments ended in ".2".
    matrix.2<-matrix (rnorm (nrow*ncol.2, mean.2, sd.2), nrow, ncol=ncol.2)
    out<-cbind (out, matrix.2)
  }

  else{
    ncol.2<- 0
  }

  #The function automaticaly name the columns and the rows like-> sample.numberofcolum and gene.numbero
  colnames(out)<-paste("sample",1:(ncol.1+ncol.2),sep=".")
  rownames(out)<-paste("gene",1:nrow,sep=".")
  return(out)
}

generateExpressions(6,3)
```

```
##          sample.1   sample.2   sample.3
## gene.1  1.9087492  0.9423103  0.4833011
## gene.2 -0.4797148 -1.6223822 -2.0417390
## gene.3 -0.8530208  0.4759918  2.7616003
## gene.4 -0.8659309  0.9619073 -0.9764069
## gene.5  1.1455070  0.9017986  0.1867650
## gene.6 -0.7582223 -0.3812429  0.1626937
```

```r
generateExpressions( )
```

```
##           sample.1
## gene.1 -0.59281893
## gene.2 -0.47836946
## gene.3  0.01719178
## gene.4  0.21350656
## gene.5  0.08739617
```

1.3 Briefly, describe what the function does and show an example.

This function generates a matrix with values from a normal distribution. We can decide number of rows, number of colums, mean, and standar deviation of the distribution.

```
generateExpressions<-function(nrow=5,ncol.1=1,mean.1=0,sd.1=1,samples.2=F,
ncol.2=NULL,mean.2=NULL,sd.2=NULL){
out<-matrix(rnorm(nrow*ncol.1,mean.1,sd.1),nrow,ncol=ncol.1)
if(samples.2){
if(is.null(ncol.2)) ncol.2<-ncol.1
if(is.null(mean.2)) mean.2<-mean.1
if(is.null(sd.2)) sd.2<-sd.1
matrix.2<-matrix(rnorm(nrow*ncol.2,mean.2,sd.2), nrow, ncol=ncol.2)
out<-cbind(out,matrix.2)
}else{
ncol.2<- 0
}
colnames(out)<-paste("sample",1:(ncol.1+ncol.2),sep=".")
rownames(out)<-paste("gene",1:nrow,sep=".")
return(out)
}
```

Also we can create a second matrix which is binded to the first one using the argument samples.2=TRUE and adding the data in the arguments ended in ".2".

An example of a matrix using sample.2 argument and binding this one to the first matrix.

```
generateExpressions(samples.2=TRUE,ncol.2=2,mean.2=2,sd.2=1)
```

```
##          sample.1  sample.2 sample.3
## gene.1  0.8966055 2.1236336 2.664443
## gene.2 -2.5246823 1.9771016 4.035587
## gene.3  1.2091499 0.8559063 3.311291
## gene.4  1.2309578 1.7078635 1.846229
## gene.5  0.7815282 2.1941046 2.875403
```

Exercise 2.

File gene exp.data contains a subset of gene expression values from a human Affymetrix array. Each probe consists of six values associated with three different experimental conditions (control, placebo and treatment)

2.1 Input this file in your R workspace.

```
gene <- read.table("gene_expr.data", header = TRUE)
head(gene)
```

```
##       Probe Symbol        Control.1        Control.2        Placebo.1
## 1   1053_at   RFC2 6,72016220324364 6,86886484679274 6,38506567503364
## 2    117_at  HSPA6  4,4781045172568 4,81989871319116 4,46559095249476
## 3    121_at   PAX8  6,9178421386246 7,31719533479415 7,16307864478947
## 4 1255_g_at GUCA1A 2,97901053177759 2,94111690858829  2,8487215206575
## 5   1294_at   UBA7 7,78163614353785 7,62807029539797 7,88665735253274
## 6   1316_at   THRA 5,34593713904358 4,69217591719547 4,90701996746067
##          Placebo.2      Treatment.1      Treatment.2
## 1 6,72615256197592 6,63745292639532 6,67710275299777
## 2 4,64622173052489 4,51909953871631 4,65576406143708
## 3 7,17033776701622  6,7943158410971  7,1280127856557
## 4 2,92333254904510 2,98090625940440 3,09877275268451
## 5 7,85488055565392 7,92496308464751 8,04952969621382
## 6 5,22475412294243 4,48366281161626 5,52166796604203
```

2.2 How many probes are in the dataset?

```
length(gene$Probe)
```

```
## [1] 33619
```

2.3 How many genes are in the dataset?

```
length(unique(gene$Symbol))
```

```
## [1] 15629
```

2.4 How many probes are in the dataset per each gene?

```
(length(gene$Probe)) / (length(unique(gene$Symbol)))
```

```
## [1] 2.151065
```

2.5 Write a function that reads an arbitrary file like gene exp.data and gives you these three numbers

```
numbers.f<- function (file) {

  gene.f <- read.table(file, header = TRUE)
  ProbeNumber <- length((gene.f$Probe))
  GeneNumber <- length(unique(gene.f$Symbol))
  ratio <- (length(gene.f$Probe)) / (length(unique(gene.f$Symbol)))
  return(c("ProbeNumber" = ProbeNumber,"GeneNumber" = GeneNumber, "Probe per Gene" = ratio))

}
numbers.f("gene_expr.data")
```

```
##    ProbeNumber     GeneNumber Probe per Gene
##   33619.000000   15629.000000       2.151065
```

Exercise 3.

Given a numeric vector, write a function called descriptive, that yields a new vector containing the following descriptive statistics: 1. n: the number of measurements 2. missing: the number of NA's 3. Mean: the average 4. Std.Dev: the standard deviation 5. Min: the minimum value 6. Q1: the quantile 1 (i.e. percentile 25%) 7. Median: the median 8. Q3: the quantile 3 (i.e. percentile 75%) 9. Max: the maximum value

```
descriptive <- function (vector) {

  n<- length(vector)
  missing <- sum(is.na(vector))
  Mean <- mean(vector)
  Std.Dev<- sd(vector)
  Min <- min(vector)
  Q1 <- quantile(vector, 0.25, na.rm=TRUE)
  Median <- median(vector)
  Q3<- quantile(vector, 0.75,na.rm=TRUE)
  Max<- max(vector)

  return(c("number of measurements"= n, "NA's Number" = missing,"Mean" = Mean,
           "Standar deviation" = Std.Dev, "Minumun value" = Min, "Quantile 1" = Q1, "Median" = Median,
           "Quantile 3"= Q3, "Maximun value" = Max))
}

v<- c(rnorm(15,1,1))
descriptive(v)
```

```
## number of measurements            NA's Number                   Mean
##             15.0000000              0.0000000              0.7887515
##      Standar deviation          Minumun value        Quantile 1.25%
##              0.8972242             -0.7010821              0.4821818
##                 Median        Quantile 3.75%          Maximun value
##              0.8150028              1.1073544              2.4472095
```

Exercise 4.

Consider the data.frame that you read in exercise 1. Write a function called exprDescriptive that invokes descriptive function for describing the expression values of either samples (columns) or probes (rows). The result must be a matrix where on rows are the samples (or probes depending on the option selected by the user) and on columns are the statistics

```
exprDescriptive <- function (data = NULL, colpos = -(1:2), row.col) {
  data <- gene [,colpos]
  if (row.col== "row") {
    d.row <- c()
    for (i in 1:nrow(data)) {
      d.row<-c(d.row, descriptive(as.numeric(data[i,])))
    }
    out <- matrix (d.row, nrow = nrow (data), byrow = TRUE)
  }
  if (row.col == "col") {
      d.col <- c()
    for (i in 1:ncol(data)) {
      d.col<-c(d.col, descriptive(as.numeric(data[,i])))
    }

  out <- matrix (d.col, nrow= ncol(data), byrow = TRUE)
  }
  return(out)
}

head(exprDescriptive(gene, row.col  ="col"))
```

```
##       [,1] [,2]     [,3]     [,4] [,5]   [,6]  [,7]    [,8]  [,9]
## [1,] 33619    0 15665.02 8992.199    1 7882.5 15676 23377.5 31422
## [2,] 33619    0 15709.89 9007.833    1 7945.5 15721 23444.5 31466
## [3,] 33619    0 15706.69 9036.468    1 7880.5 15707 23473.0 31532
## [4,] 33619    0 15704.40 9000.716    1 7914.5 15738 23425.5 31445
## [5,] 33619    0 15717.93 9005.536    1 7921.5 15755 23469.5 31455
## [6,] 33619    0 15756.40 9037.214    1 7936.5 15790 23522.5 31548
```

Exercise 5.

The GC-content is a characteristic of a specific fragment of DNA or RNA, or that of the whole genome of an organism. It is expressed as the percentage of nitrogenous bases on a DNA molecule that are either guanine (G) or cytosine (C). Write a function called gc that calculates the GC-content of a gicen DNA sequence.

```
s<- "gccaattcgtatcgatctgacgt"
gc <- function (sequence) {
  c<-length(gregexpr("c",sequence)[[1]])
  g<-length(gregexpr("g",sequence)[[1]])
  out <- (g+c) / length(unlist(strsplit(sequence,"")))*100
  return(out)
}
gc(s)
```

```
## [1] 47.82609
```

Exercise 6.

Write a function called wc that invokes the function readLines for reading a text file and gives you the number of lines of that file. You can use file dna seqs.fasta for testing your function.

```
wc <- function (file) {
    out <- length (readLines (file))
    return (out)
}
wc ("dna_seqs.fasta")
```

```
## [1] 64
```

Exercise 7.

Write a function called summarySingleFasta that reads a fasta file with a single sequence and yields a vector with the following four elements: 1. Description: the description provided in the first line of the fasta sequence 2. Sequence: the DNA sequence 3. Width: the length of the DNA sequence 4. GC.content: the GC-content of the sequence You can use the file single dna seq.fasta for testing your function.

```
summarySingleFasta <- function (file) {
my.sequence <- readLines (file)
description <- my.sequence[1]
sequence <- my.sequence[2]
width <- length (unlist (strsplit (my.sequence[2], "")))
gc.content <- gc (my.sequence[2])
return (list ("Description" = description, "Sequence" = sequence, "Width" = width, "GC.content" = gc.co
}
summarySingleFasta ("single_sequence.fasta")
```

```
## $Description
## [1] ">read.2 Sus_domesticus"
##
## $Sequence
## [1] "ggtattggcaccgtataatgggggatggcgtgcttctaccagtgacaattaagatgcgcttcggaactaagt"
##
## $Width
## [1] 72
##
## $GC.content
## [1] 48.61111
```

Exercise 8.

Write a function called summaryMultipleFasta that reads a fasta file with multiple DNA sequences, whose description consists of two elements (a potential name of a gene and a potential name of a specie) separated by a single space, and yields a data.frame such that for each sequence (rows) provides the following information (columns): 1. Gene: the name of the potential gene 2. Specie: the name of the potential specie 3. Sequence: the DNA sequence 4. Width: the length of the DNA sequence 5. GC.content: the GC-content of the sequence You can use the file dna sequence.fasta for testing your function.

```
summaryMultipleFasta <- function (file) {
multi.fasta <- readLines (file)
description <- strsplit(multi.fasta[seq(1,length(multi.fasta), by = 2)]," ")
```

```r
gene <- c(NULL)
  for (i in 1:length (description))
  {
    gene <- c(gene, description[[i]][1])
    }
specie <- c(NULL)

  for (i in 1:length (description)) {
    specie <- c(specie, description[[i]][2])
  }

sequence <- multi.fasta[seq(2,length(multi.fasta), by = 2)]
width <- c(NULL)

for (i in 1:length (sequence)) {
  width <- c(width, length (unlist(strsplit(sequence[i], ""))))
  }
gc.content <- c(NULL)

  for (i in 1:length (sequence)) {
    gc.content <- c(gc.content, gc(sequence[i]))
  }

out <- data.frame ("Gene" = gene, "Specie" = specie, "Sequence" = sequence, "Width" = width, "GC.conten
return(out)
}
head (summaryMultipleFasta ("dna_seqs.fasta"))
```

```
##       Gene           Specie
## 1  >read.2   Sus_domesticus
## 2  >read.3   Sus_domesticus
## 3 >read.10   Sus_domesticus
## 4 >read.12     Homo_sapiens
## 5  >read.1 Canis_familiaris
## 6  >read.4   Sus_domesticus
##                                                                      Sequence
## 1   ggtattggcaccgtataatgggggatggcgtgcttctaccagtgacaattaagatgcgcttcggaactaagt
## 2                       tcagaggccaccaagtagaagtacgagtgcttgtctacagatttgatgttctc
## 3        gaagctagcgaaagaaccttgtgacgagcctctaatacatggtcaactctttatagtcctcgtttcg
## 4 ccgacacttttgtcgtcgcttagtggttgtgaccgagtgggcgttagtaccaatacagtccggatagtgtcagg
## 5  gtcactcgcggtaacttcaataagttcccgccctaaggccaactggaaccgaggcaccacaggacttgactgt
## 6    ccttccccccagtagttattgcttcctacgcgattattttttttatatatagtcgcgacctggtcgcgcgt
##   Width GC.content
## 1    72   48.61111
## 2    53   45.28302
## 3    67   44.77612
## 4    74   52.70270
## 5    73   54.79452
## 6    71   46.47887
```