Homework 6

Steven Gan

2022-10-16

Table of contents

Section 1

A

```
df <- data.frame(a = 1:10, b = seq(200, 400, length = 10),c = 11:20, d = NA)
df
```

```
   a        b  c  d
1  1 200.0000 11 NA
2  2 222.2222 12 NA
3  3 244.4444 13 NA
4  4 266.6667 14 NA
5  5 288.8889 15 NA
6  6 311.1111 16 NA
7  7 333.3333 17 NA
8  8 355.5556 18 NA
```

```
9   9 377.7778 19 NA
10 10 400.0000 20 NA
```

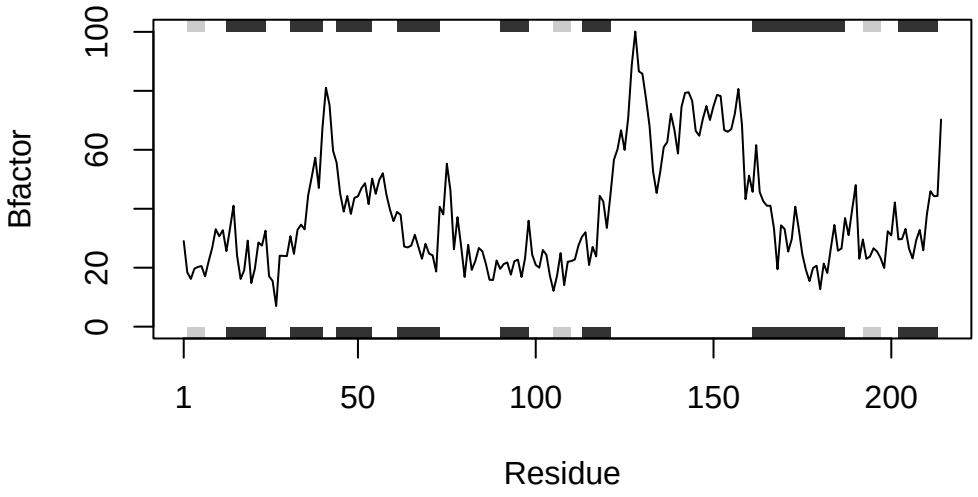
```r
a <- function(x) {
  x <- (x - min(x)) / (max(x) - min(x))
}

df <- apply(df, 2, a)
df
```

```
              a         b         c  d
 [1,] 0.0000000 0.0000000 0.0000000 NA
 [2,] 0.1111111 0.1111111 0.1111111 NA
 [3,] 0.2222222 0.2222222 0.2222222 NA
 [4,] 0.3333333 0.3333333 0.3333333 NA
 [5,] 0.4444444 0.4444444 0.4444444 NA
 [6,] 0.5555556 0.5555556 0.5555556 NA
 [7,] 0.6666667 0.6666667 0.6666667 NA
 [8,] 0.7777778 0.7777778 0.7777778 NA
 [9,] 0.8888889 0.8888889 0.8888889 NA
[10,] 1.0000000 1.0000000 1.0000000 NA
```

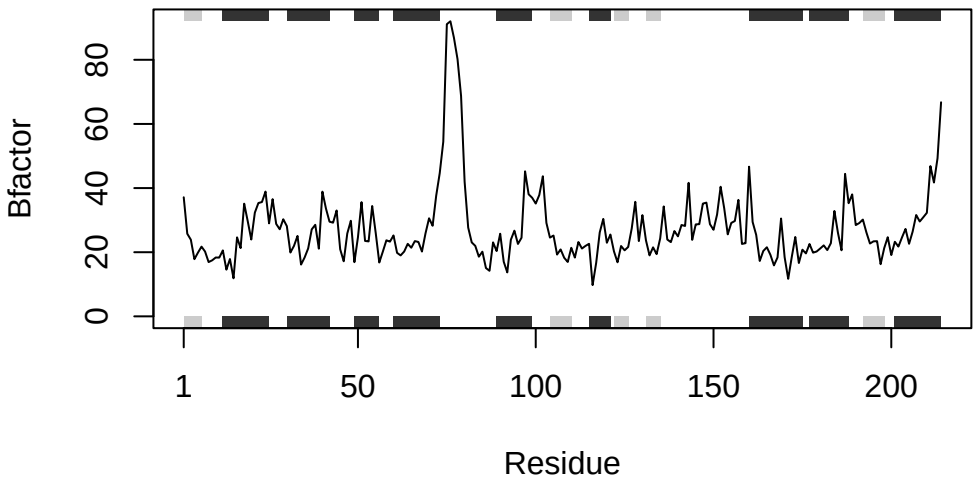
B

```r
# Output the Bfactor of a protein by inputing its PDB ID
b_factor <- function(pdb, plot = TRUE){
  if(!requireNamespace("bio3d", quietly = TRUE))
  install.packages("bio3d")
  library(bio3d)

  s <- read.pdb(pdb)
  s.chainA <- trim.pdb(s, chain = "A", elety = "CA")
  s.b <- s.chainA$atom$b

  if (plot == TRUE){plotb3(s.b, sse = s.chainA, typ = "l", ylab = "Bfactor")}
  return(s.b)
}

s.b <- apply(as.data.frame(c("4AKE", "1AKE", "1E4Y")), 1, b_factor)
```

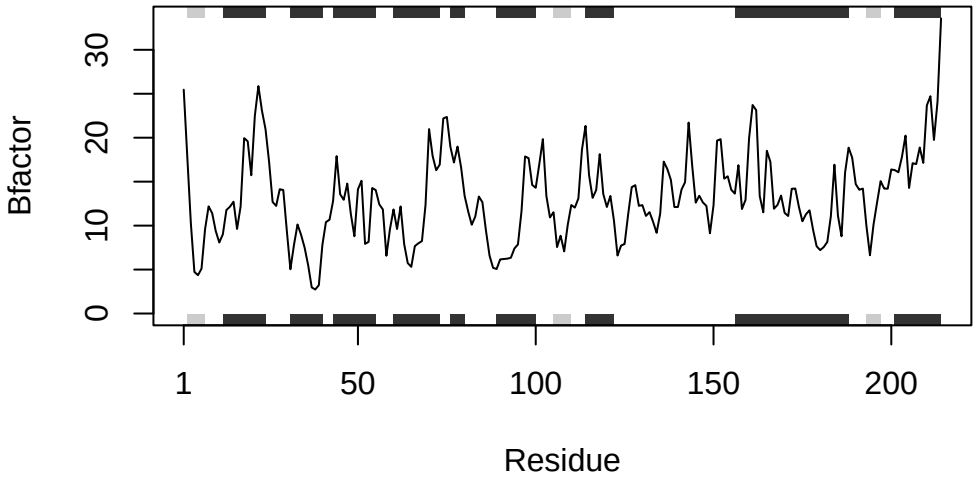
```
Note: Accessing on-line PDB file
```



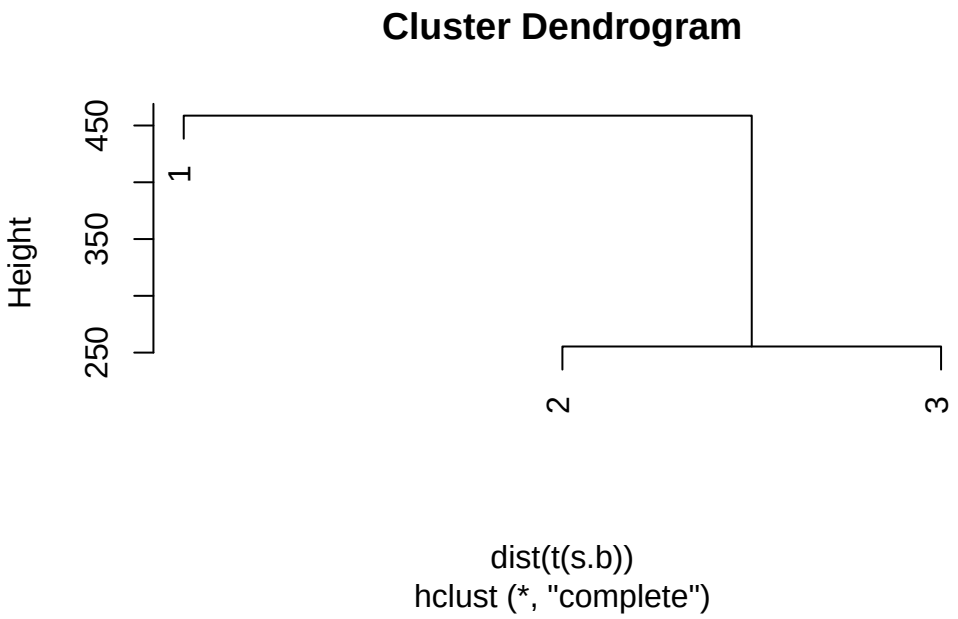

```
Note: Accessing on-line PDB file
 PDB has ALT records, taking A only, rm.alt=TRUE
```




```
Note: Accessing on-line PDB file
```



```
# Hierarchical cluster analysis
hc <- hclust(dist(t(s.b)))
plot(hc)
```



Q1:

A list with 8 elements is returned from the read.pdb()

Q2:

trim.pdb() trim residues and filter the structures from a pdb object to a new pdb object. In this case, it select the C-alpha atoms from the chain A of the protein.

Q3:

Delete the sse argument will turn off the marginal black and grey rectangules. sse indicates the secondary structure object, in this case, the the chain A of the protein.

Q4:

To compare the similarity between proteins, the intuitive way would be superposite the strucutre of two proteins and label the physical distance between two structures. Hierarchical cluster analysis is also a way to plot the phylogenetic tree of proteins.

Q5:

Protein 1AKE and 1E4Y exhibit higher similarity with each other.

The analysis was performed based on measuring the distance matrix between proteins and followed by hierarchical cluster analysis.

Q6:

```
#Input PBD of proteins and plot the Hierarchical cluster analysis result
hc_pdbs <- function(pdbs){
  pdbs <- as.data.frame(pdbs)
  s.b <- apply(pdbs, 1, b_factor, plot = FALSE)
  hc <- hclust(dist(t(s.b)))
  plot(hc)
}

hc_pdbs(c("4AKE", "1AKE", "1E4Y"))
```

```
  Note: Accessing on-line PDB file
```

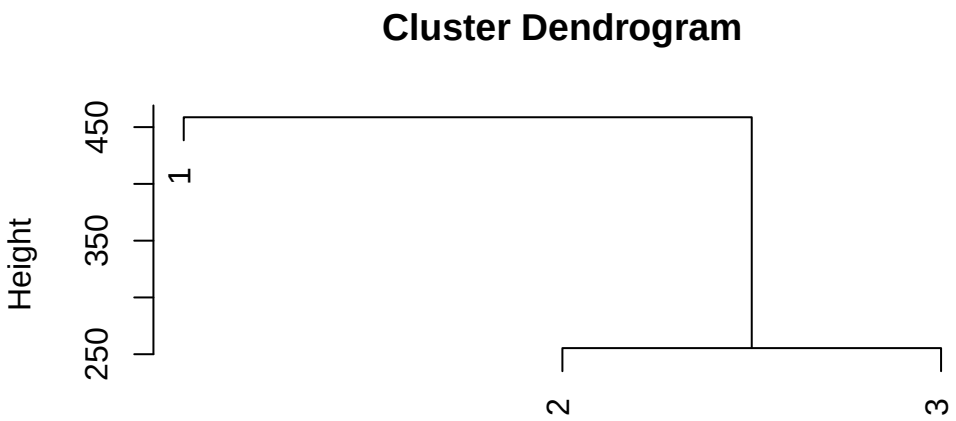
```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE): /var/folders/pb/
rjqmlzp924v7cg247gmrwd3w0000gn/T//RtmpSnmTNi/4AKE.pdb exists. Skipping download
```

```
  Note: Accessing on-line PDB file
```

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE): /var/folders/pb/
rjqmlzp924v7cg247gmrwd3w0000gn/T//RtmpSnmTNi/1AKE.pdb exists. Skipping download
```

```
   PDB has ALT records, taking A only, rm.alt=TRUE
  Note: Accessing on-line PDB file
```

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE): /var/folders/pb/
rjqmlzp924v7cg247gmrwd3w0000gn/T//RtmpSnmTNi/1E4Y.pdb exists. Skipping download
```



Section 2

```
square.it <- function(x) {
 square <- x * x
 return(square)
}

# square a number
square.it(5)
```

```
[1] 25
```

```
# square a vector
square.it(c(1, 4, 2))
```

```
[1]  1 16  4
```

```r
# square a character (not going to happen)
#square.it("hi")
#Error in x * x : non-numeric argument to binary operator

matrix1 <- cbind(c(3, 10), c(4, 5))
square.it(matrix1)
```

```
     [,1] [,2]
[1,]    9   16
[2,]  100   25
```

```r
fun1 <- function(x) {3 * x - 1}
fun1(5)
```

```
[1] 14
```

```r
fun2 <- function(x) {y <- 3 * x - 1}
fun2(5)
```

Section 3

```r
my.fun <- function(x.matrix, y.vec, z.scalar) {
 # use my previous function square.it() and save result
 sq.scalar <- square.it(z.scalar)
 # multiply the matrix by the vector using %*% operator
 mult <- x.matrix %*% y.vec
 # multiply the resulting objects together to get a final ans
 final <- mult * sq.scalar
 # return the result
 return(final)
}

# save a matrix and a vector object
my.mat <- cbind(c(1, 3, 4), c(5, 4, 3))
my.vec <- c(4, 3)
# pass my.mat and my.vec into the my.fun function
my.fun(my.mat, my.vec, 9)
```

```
     [,1]
[1,] 1539
[2,] 1944
[3,] 2025
```

```
#Returning a list of objects
another.fun <- function(sq.matrix, vector) {
 # transpose matrix and square the vector
 step1 <- t(sq.matrix)
 step2 <- vector * vector
 # save both results in a list and return
 final <- list(step1, step2)
 return(final)
}
# call the function and save result in object called outcome
outcome <- another.fun(sq.matrix = cbind(c(1, 2), c(3, 4)), vector = c(2, 3))
# print the outcome list
print(outcome)
```

```
[[1]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4

[[2]]
[1] 4 9
```

```
# extract first in list
outcome[[1]]
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

```
# extract second in list
outcome[[2]]
```

```
[1] 4 9
```

Section 4

```
my.fun <- function(x.matrix, y.vec, z.scalar) {
 print("xmatrix")
 print(x.matrix)
 print("yvec")
 print(y.vec)
 print("Dimensions")
 print(dim(x.matrix))
 print(length(y.vec))
 # use previous function square.it() and save result
 sq.scalar <- square.it(z.scalar)
 print(paste("sq.scalar=", sq.scalar))
 # multiply the matrix by the vector using %*% operator
 mult <- x.matrix %*% y.vec
 # multiply the two resulting objects
 final <- mult * sq.scalar
 # return the result
 return(final)
}
#my.fun(my.mat, c(2, 3, 6, 4, 1), 9)

my.second.fun <- function(matrix, vector) {
 if (dim(matrix)[2] != length(vector)) {
 stop("Can't multiply matrix%*%vector because the
dimensions are wrong")
 }
 product <- matrix %*% vector
 return(product)
}
# function works when dimensions are right
my.second.fun(my.mat, c(6, 5))
```

```
     [,1]
[1,]   31
[2,]   38
[3,]   39
```

```
# function call triggered error
# my.second.fun(my.mat, c(6, 5, 7))
```

```
# Error in my.second.fun(my.mat, c(6, 5, 7)) : Can't multiply matrix%*%vector because the
```