Project BIO - Genetic Code in R

Carlos Adir **Ely Murussi Leite**

June 18, 2020

1 Introduction

This project has the objective of using the concepts in R learned during the classes to apply in biology: Make the processus of transcription, translation and alignment.

In the first moment there are some exercises to reminder of the language in R, the use of a GUI(Graphical User Interface) known as Shiny to help in data analysis.

The 2 last exercises(7.5 and 7.6, bonus) were not made.

2 Remider on R

Exercise 2.1

a) Write a function **compte** which takes two arguments: a sequence of characters and a letter, and which returns the number of occurrences of the letter in the sequence.

We code the folowed function

```
compte <- function(sequence, letter){
        return(sum(sequence == letter))
}
```

b) Apply your function to find the number of occurrences of the letter "a" in the biological sequence

```
sequence<-c("a","a","t","g","a","g","c","t","a","g","c","t","g")
```

Using the code below, we get

```
sequence <- c("a","a","t","g","a","g","c","t","a","g","c","t","g")
qtt_a <- compte(sequence, "a")
print(qtt_a)
```

OUTPUT:

```
[1] 4
```

c) Use a for loop to find the numbers of occurences of each letter in the sequence.

```
bases = c("a","c","g","t")
composition = rep(0,4)
for(i in 1:4){
        composition[i] = compte(sequence, bases[i])
}
print(composition)
```

OUTPUT:

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

3 Dealing with sequences

Exercise 3.1

a) What are the **seqtype** and **forceDNAtolower** options for?

If we read the documentation, we get:

seqtype: the nature of the sequence: DNA or AA, defaulting to DNA

forceDNAtolower: whether sequences with seqtype == "DNA" should be returned as lower case letters

b) Print the first 50 nucleotides of your sequence:

For do that, we use the code:

```
library("seqinr")
filename <- "~/ECN/Biologie/13-TP-ProjR/Students/Mysterious_seq.txt"
sequences <- read.fasta(file = filename, seqtype = "DNA", forceDNAtolower = T)
mysequence <- sequences$seq0
print(mysequence[1:50])
```

OUTPUT:

```
 [1] "g" "c" "g" "c" "c" "g" "g" "g" "a" "a" "g" "a" "a" "a" "g" "g" "a" "a" "c"
[20] "a" "t" "g" "g" "c" "t" "c" "c" "t" "g" "a" "g" "g" "c" "g" "c" "a" "c" "a"
[39] "g" "c" "g" "c" "c" "g" "a" "g" "c" "g" "c" "g"
```

c) Print the length of the sequence.

To do so, we just need

```
print(length(mysequence))
```

OUTPUT

```
[1] 5080
```

d) Print the number of occurences of nucleotides (a, t, c, g).

To do so, we just need

```
bases <- c("a", "c", "g", "t")
composition <- matrix(integer(4), ncol=4)
for(i in 1:4){
  composition[i] <- sum(mysequence == bases[i])
}
colnames(composition) <- bases
print(as.table(composition))
```

OUTPUT:

```
     a    c    g    t
A 1391 1079 1202 1408
```

Exercise 3.2

a) Why is the computation of GC-content relevant in molecular biology?

It give us the stability of the chain, because as the link $G \equiv C$ is stronger than the link $A = T$. We call that *thermostability*. Its entirely correlated with the quantity of hidrogen bonds.

Other reason in use CG content instead of AT, is because we can use the same concept in RNA. That means, we just need to say CG content instead of AT/AU content.

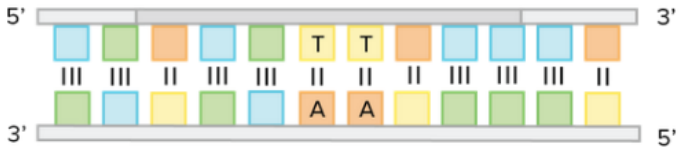


Figure 1: Links between the nucleotides
Source: Khan Academy

b) Compute the GC-content of your sequence.

```
GCcount = composition[2] + composition[3]
GCcontent = GCcount/length(mysequence)
print(paste("Value GC-content:", GCcontent))
```

OUTPUT:

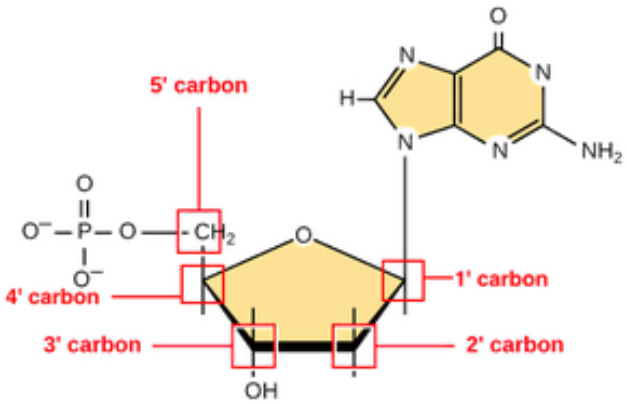
```
[1] "Value GC-content: 0.449015748031496"
```

Exercise 3.3

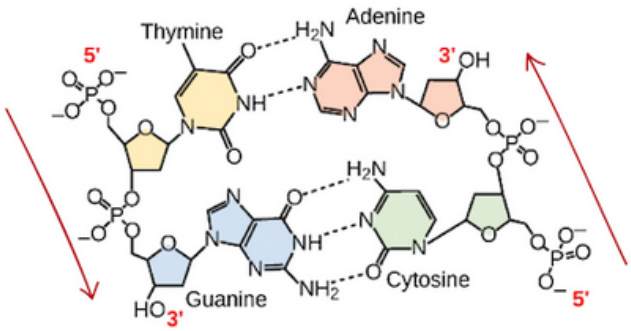
a) By convention, in which direction ($5' \rightarrow 3'$ or $3' \rightarrow 5'$) do we write DNA sequences? What do $5'$ and $3'$ represent?

The convention is using the direction $5' \rightarrow 3'$.

As we know, the molecules are in 3D and it's hard to say what is below, what is above, and so on. For that, the convention is numerate using the number of the carbon that the structure is linked, as seen in the Figure (2).



(a) Number of carbons



(b) Sequencing

Figure 2: Structure
Source: Khan Academy

b) Write a function complementary that takes a sequence as input and returns its complementary sequence (i.e. a becomes t; t, a; c, g; and g, c).

```
complementary <- function(seq){
  bases <- "acgt"
  bases_rev <- "tgca"
  complement_seq <- chartr(bases, bases_rev, seq)
  return(rev(complement_seq)) # To adjust from 5->3 again
}
```

c) Compute and print the reverse complementary of the reference sequence.

```
mysequence_complement <- complementary(mysequence)
print("Complementary sequence:")
print(mysequence_complement[1:25])
```

OUTPUT:

```
[1] "Complementary sequence:"
 [1] "t" "t" "c" "t" "g" "c" "t" "a" "c" "t" "a" "t" "a" "t" "c" "a" "a" "g" "t"
[20] "t" "t" "t" "t" "a" "t"
```

4 Introduction to Shiny

Exercise 4.1

a) Add the function *compte* from *exercise 2.1* in the Shiny application (*util.R* file)

As we have already done the code in 2.1, it's only copy and paste.

b) Run the application and test with *Mysterious_seq.txt*. Alternatively, you can test it with a dataset of 31 PKD2 mRNA sequences (*PKD2_mRNA_sequences.txt*).

When we run, we can select a file. If we select the dataset of 31 sequences given, we obtain the Figure (3)

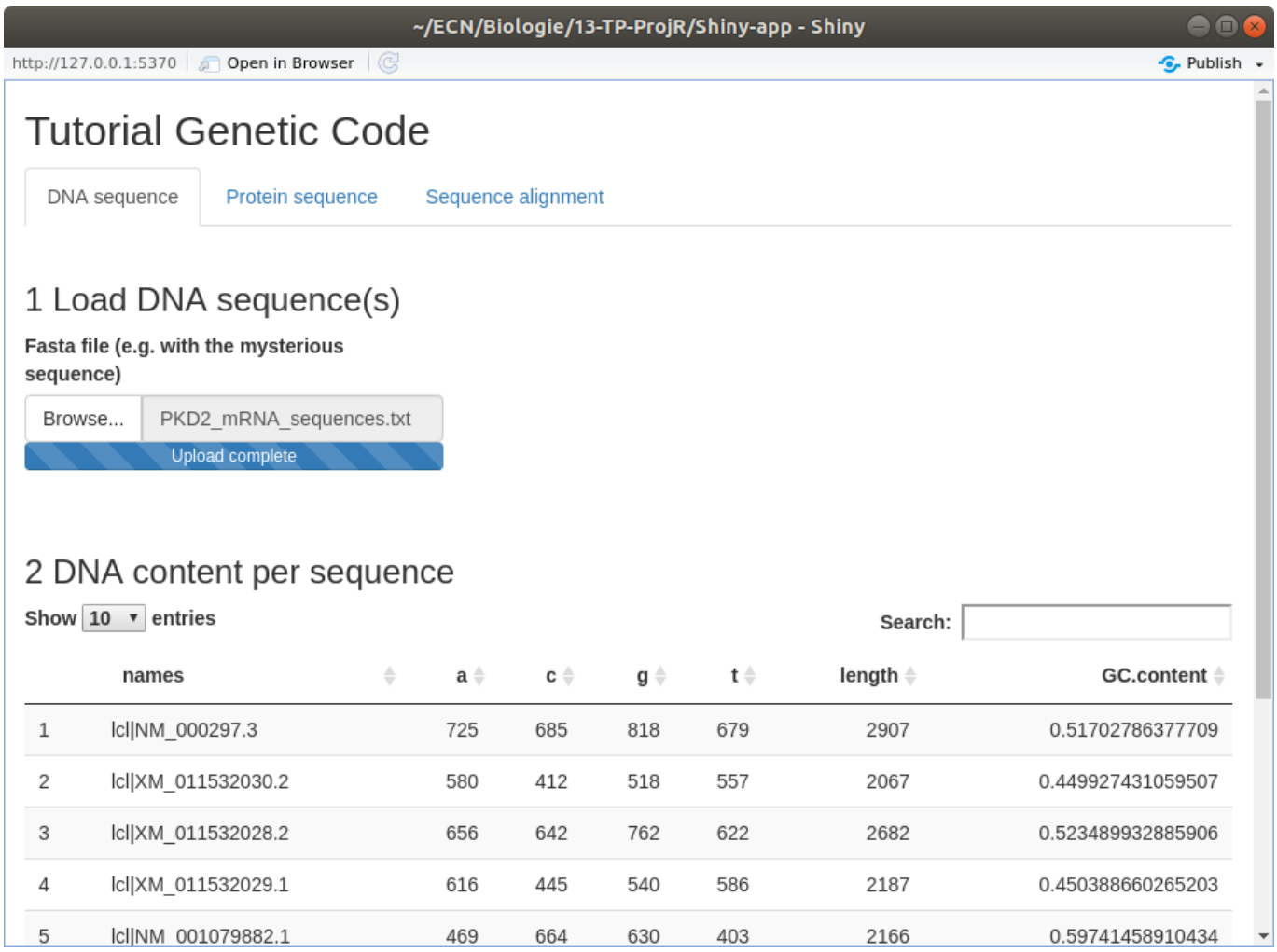


Figure 3: Frist screen

5 Transcription

Exercise 5.1

a) What is the transcription mechanism for? Describe briefly the main steps of transcription.

The transcription is the processing of getting the DNA's sequence, copying it and making a RNA molecule.

As we can see in the Figure (4), it's the processus os creating the RNA.

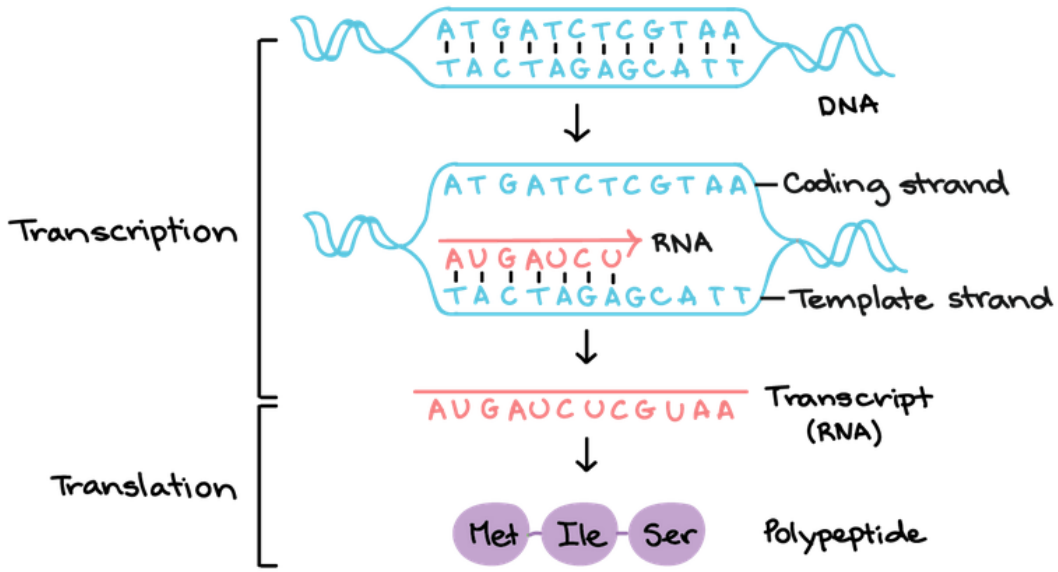


Figure 4: Transcription Mechanism
Source: Khan Academy

The processus follow the order:

Begin of the transcription The enzyme polymerase goes to the DNA in a region that we call **promoter**.

Elongation The enzyme begins the elongation: The RNA grows because the enzyme puts more and more nucleotides while it walks on the DNA, as we can see in the Figure (5).

End of transcription It keeps going until it reaches the signal to **stop**. When it happens, the polymerase writes the sequence known as terminator

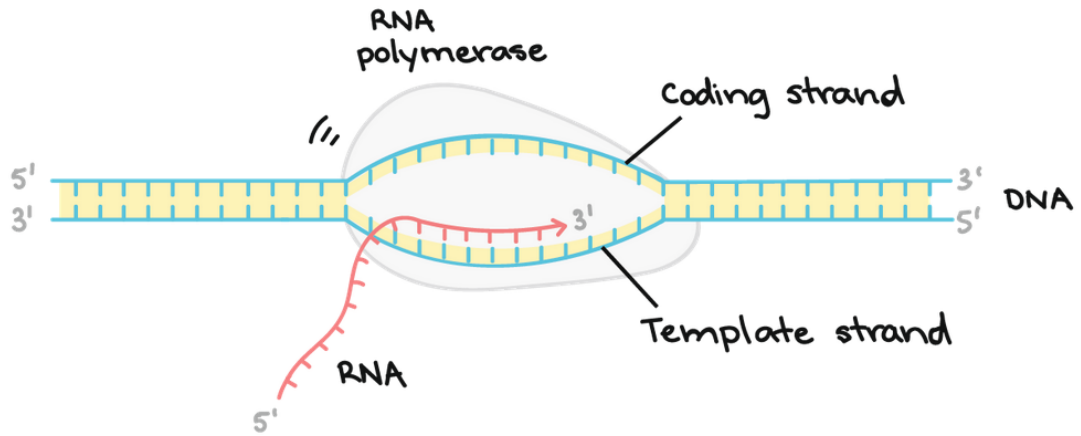


Figure 5: Polimerase produces pre-mRNA from DNA
Source: Khan Academy

b) What is(are) the difference(s) between **mRNA** and **cDNA**?

The first difference that we have, is that RNA has the base U instead of T that we find in the DNA. The second difference is that it's a single-stranded similar to the DNA sequence

We have that $mRNA$ is the RNA produced by transcription, and $cDNA$ is the DNA synthesized using the $mRNA$ sequence.

c) What enzyme catalyzes the **mRNA** to **cDNA** reaction?

While we have the polymerase that produces $mRNA$ from the DNA(Figure 5), there is the **enzyme reverse transcriptase** that gets the $mRNA$ and transforms into $cDNA$(complementary DNA).

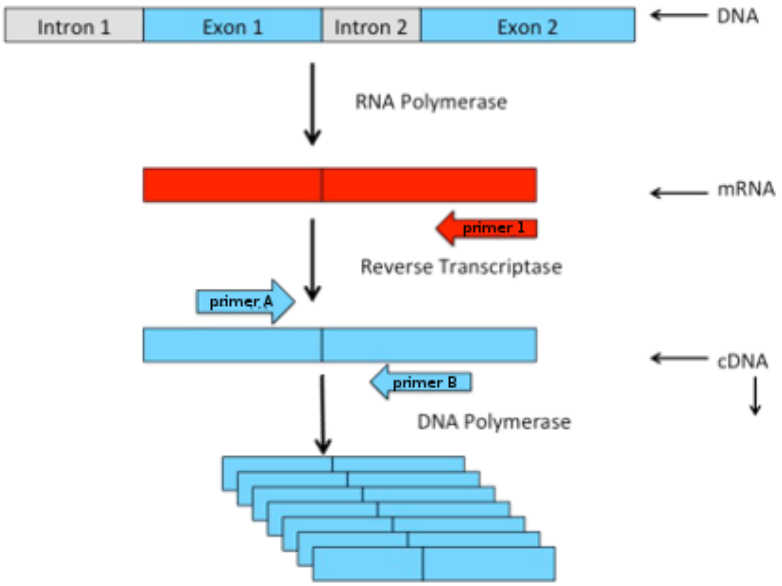


Figure 6: Reverse trans produces cDNA from mRNA
Source: Khan Academy

d) What is the difference between cDNA and the original DNA sequence (if any)?

During the processus of transcription of the DNA, we get the **pre-mRNA**. During its maturation, introns are thrown away, remaining only the extrons(as we can see in the Figure 7), resulting in the **mRNA** that is converted into **cDNA**

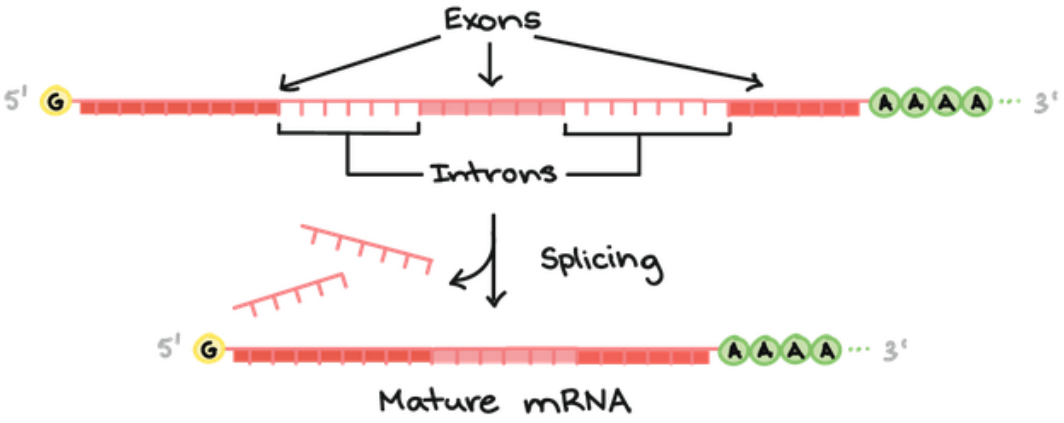


Figure 7: Slicing
Source: Khan Academy

So, we could say that cDNA is a mix of slices(in order) of the original DNA.

But we have to say that cDNA and DNA are different only in eukaryote cells, because the prokaryotes are not able to remove the introns.

e) Write a function **transcribe** that converts the mysterious cDNA sequence into RNA.

```
transcribe <- function(cDNA){
  RNA <- chartr("t", "u", cDNA)
  return(RNA)
}
```

6 Translation

Exercise 6.1

a) What is the translation mechanism for? Describe briefly the main steps of translation.

In short, that's showed in the Figure (8)

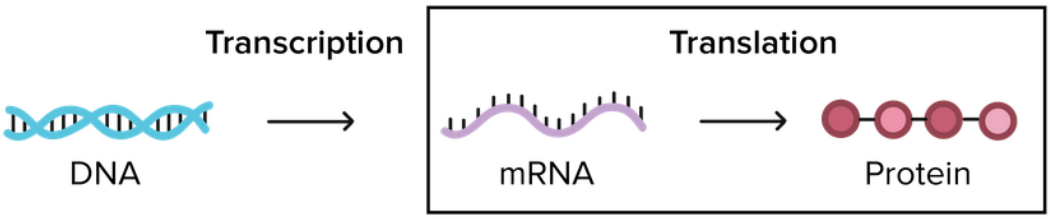


Figure 8: Translation Mechanism
Source: Khan Academy

We can say that the **translation mechanism** is the processus of getting the **mRNA** and transforming it into a chain of amino-acids that we call as a **protein**.

The processus is

Begin The ribosome gets the **mRNA** and the first tRNA(Transfer RNA).

Middle Ribosome gets amino-acids from tRNAs and link them to make a chain

End The sequence of amino-acids is released to make its job as protein.

b) What is the genetic code? What are its main characteristics (number of nucleotides used for decoding, sliding properties of the ribosome, etc.)?

The genetic code are the rules to transform the genetic material(DNA or RNA) into proteins (sequence of peptides/amino-acid).

As there is 4 nucleotides(A, C, G, T/U), it's possible to create 64 combinations(called codons), as we can see in the Figure (9).

As 3 of them is a code to **Stop**, the 61 remaining are translated into 20 amino-acids(including the **Met**).

During the translation, the ribosomes can slide over the non coding sequence(but keeps in the **ORF** region) to synthesize a single protein. We call that process as translation bypassing.

c) How many amino acids are there in the standard genetic code? Why is the genetic code defined as "redundant'?

As seen in the Figure (9), the same amino-acid can be given from differents codons.

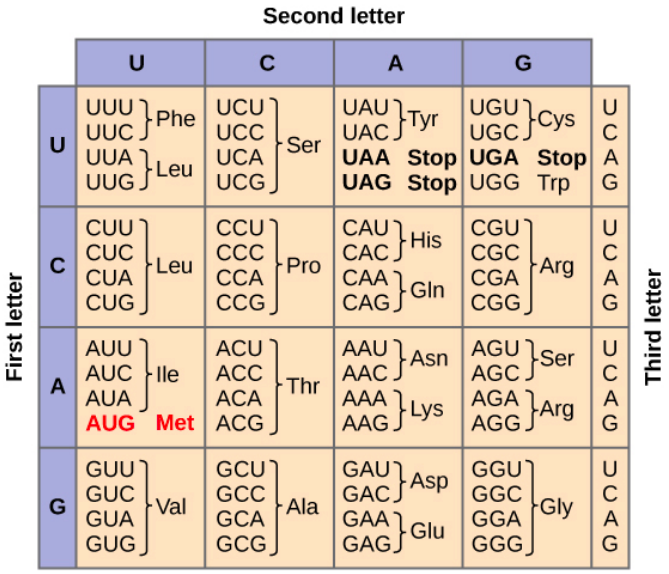


Figure 9: 64 codons, 20 amino-acids
Source: Khan Academy

That say us that some information is irrelevant. For example, if we have the sequence CU$*$, doesn't matter the value of $*$, because we already know that it's going to be translated into **Leu**. We can put them in a sequence and get:

- CU$*$ $\rightarrow$ Leu
- CC$*$ $\rightarrow$ Pro
- CG$*$ $\rightarrow$ Arg
- UC$*$ $\rightarrow$ Ser
- GU$*$ $\rightarrow$ Val
- GC$*$ $\rightarrow$ Ala
- GG$*$ $\rightarrow$ Gly
- AC$*$ $\rightarrow$ Thr

d) What is an "open reading frame" (ORF)? How many ORF is there for one sequence?

The **ORF** is a part of the **DNA** that when translated into amino-acids, doesn't have any **X**(stop codon).

As the amino-acid is built with 3 nucleotides sequence, it's possible to:

Read in the original sequence order and

- Not slide
- Slide 1 nucleotide
- Slide 2 nucleotides

Read in the reverse sequence order and

- Not slide
- Slide 1 nucleotide
- Slide 2 nucleotides

So, there is 6 possibilities to read and so get 6 differents ORF for the same sequence of DNA.

e) What is the amino acid acting as the Start codon? Which codon correspond to this amino acid?

The **Start** codon is named by **Met**, or just **M**

It's Constitution, as we can see in the Figure (9) is **AUG**.

f) Which are the Stop codons? Which one-letter code is used here to describe Stop codons?

The **End** codon is named by **Stop**, or just **X**

It's constitution, as we can see in the Figure (9) can be one of three: **UAA**, **UAG** or **UGA**.

Exercise 6.2

In your working R script, Write a function *dna2peptide* that translates a DNA sequence into an amino acid chain.

```
codon_filename <- "~/ECN/Biologie/13-TP-ProjR/Code_shortcut/Genetic_code.txt"
codon.table <- read.table(file = codon_filename, col.names = c("codon", "aa", "letter"),
                          stringsAsFactors = F)
codon.table$codon <- tolower(codon.table$codon) # use only lower case characters
rownames(codon.table) <- codon.table$codon # use codons as table keys
head(codon.table)

dna2peptide <- function(sequence, frame=0, sens='F'){
  if(sens == 'R'){
    sequence <- complementary(sequence)
  }
  step = 3
  len <- length(sequence)
  l <- step*((len - frame)%/%step)
  init = frame + 1
  end = l + frame
  seq_coupe <- split(sequence[init:end], ceiling(seq(l)/step))
  for( i in 1:length(seq_coupe) ){
    seq_coupe[i] <- paste(unlist(seq_coupe[i]), collapse='')
  }
  peptides <- codon.table[unlist(seq_coupe), "letter"]
  return( paste(peptides, collapse = "") )
}
sequence <- c('a', 't', 'g', 't', 't', 'c', 't', 't', 't', 'a')
dna2peptide(sequence, 0, 'F')
```

OUTPUT:

```
[1] "MFF"
```

Exercise 6.3

Define a function *find.mysterious.proteins* function that takes a cDNA sequence as input and returns a list of proteins as defined above. Apply this function to the mysterious sequence, save the resulting sequences in a text file and join it to your report.

```
find.mysterious.proteins <- function(cDNA){
  output_file <- "~/ECN/Biologie/13-TP-ProjR/Students/proteins.txt"
  pattern = 'M[ACDEFGHIKLMNPQRSTVWY]{80,}(X|$)'
  proteins <- c()
  step <- 3
  for( frame in 0:(step-1) ){
    for( sens in c("L", "R") ){
      supposed_peps <- paste(dna2peptide(cDNA, frame, sens), collapse = "")
      reg <- gregexpr(pattern, supposed_peps)
      match <- regmatches(supposed_peps, reg)[[1]]
      proteins <- c(proteins, match)
    }
  }
  write(proteins, file = output_file)
  return(proteins)
}

library("seqinr")
filename <- "~/ECN/Biologie/13-TP-ProjR/Students/Mysterious_seq.txt"
sequences <- read.fasta(file = filename, seqtype = "DNA", forceDNAtolower = T)
mysequence <- sequences$seq0
proteins <- find.mysterious.proteins(mysequence)
print(proteins)
```

OUTPUT:

```
[1] MVNSSRVQPQQPGDAKRPPAPRAPDPGRLMAGCAAVGASLAAPGGLCEQRGLEIEMQRIRQAAARDPPAGAAASP
    SPPLSSCSRQAWSRDNPGFEAEEEEEEVEGEEGGMVVEMDVEWRPGSRRSAASSAVSSVGARSRGLGGYHGAGHP
    SGRRRRREDQGPPCPSPVGGGDPLHRHLPLEGQPPRVAWAERLVRGLRGLWGTRLMEESSTNREKYLKSVLRELV
    TYLLFLIVLCILTYGMMSSNVYYYTRMMSQLFLDTPVSKTEKTNFKTLSSMEDFWKFTEGSLLDGLYWKMQPSNQ
    TEADNRSFIFYENLLLGVPRIRQLRVRNGSCSIPQDLRDEIKECYDVYSVSSEDRAPFGPRNGTAWIYTSEKDLN
    GSSHWGIIATYSGAGYYLDLSRTREETAAQVASLKKNVWLDRGTRATFIDFSVYNANINLFCVVRLLVEFPATGG
    VIPSWQFQPLKLIRYVTTFDFFLAACEIIFCFFIFYYVVEEILEIRIHKLHYFRSFWNCLDVVIVVLSVVAIGIN
    IYRTSNVEVLLQFLEDQNTFPNFEHLAYWQIQFNNIAAVTVFFVWIKLFKFINFNRTMSQLSTTMSRCAKDLFGF
    AIMFFIIFLAYAQLAYLVFGTQVDDFSTFQECIFTQFRIILGDINFAEIEEANRVLGPIYFTTFVFFMFFILLNM
```

```
    FLAIINDTYSEVKSDLAQQKAEMELSDLIRKGYHKALVKLKLKKNTVDDISESLRQGGGKLNFDELRQDLKGKGH
    TDAEIEAIFTKYDQDGDQELTEHEHQQMRDDLEKEREDLDLDHSSLPRPMSSRSFPRSLDDSEEDDDEDSGHSSR
    RRGSISSGVSYEEFQVLVRRVDRMEHSIGSIVSKIDAVIVKLEIMERAKLKRREVLGRLLDGVAEDERLGRDSEI
    HREQMERLVREELERWESDDAASQISHGLGTPVGLNGQPRPRSSRPSSSQSTEGMEGAGGNGSSNVHVX
[2] MWYPCVSTMCHISIMVTRKSLYDTFYCTVFHFTCKILQNSSFLPINYIYFSSLVMENSPPHLFPIIFPCVLVLLK
    RHYIRKSFSTIKNVINVX
[3] MQTLVCSNKELCVAYSRESKDRDEAMRKNLQKPSMFHFQPGKVYLLMFSLEIGLTFYQLNEQLVKLKYSIKRQDS
    LGSPSEGYSTALHALLLISAQNVTLLNISVKEDFFFX
[4] MQRVPAADWAGARRALVLSPAPPPARVARAVVAPKPPAPRAHGAHGRGGGRPPAARAPLYVHLHHHSAFFPFHLL
    LLLLGLEAGVIAAPRLPGARRERRRRGGRGSGRGVPRGRLPDALHLDLQAPLLAEAARGGEAGAHGRAASHQPAR
    VRRAGRGRPLGVPGLLRLHATGVHHRGHWRPARGCAAPRSALCASGAMFLSSRR
[5] MHKTMRKRRYVTSSRKTLLRYFSRLVLLSSMSLVPQRPRSPRTSLSAQATRGGCPSRGRWRCSGSPPPTGLGHGG
    PWSSRRRRLPLGWPAPWX
```

Exercise 6.4

Fill the declaration of *output$aa.sequences* in *server.R* so that if you load a sequence file via shiny, the protein sequences appear in the main panel.

```
1 output$aa.sequences <- renderUI({
2   inFile.2 <- input$dna.file.2
3   if (is.null(inFile.2))
4     return(NULL)
5   dna.sequences <- read.fasta(file = inFile.2$datapath,
6                               seqtype = "DNA", forceDNAtolower = T)
7   list.of.proteins <- find.mysterious.proteins(dna.sequences$seq0)
8   HTML(proteins.to.html(list.of.proteins))
9 })
```

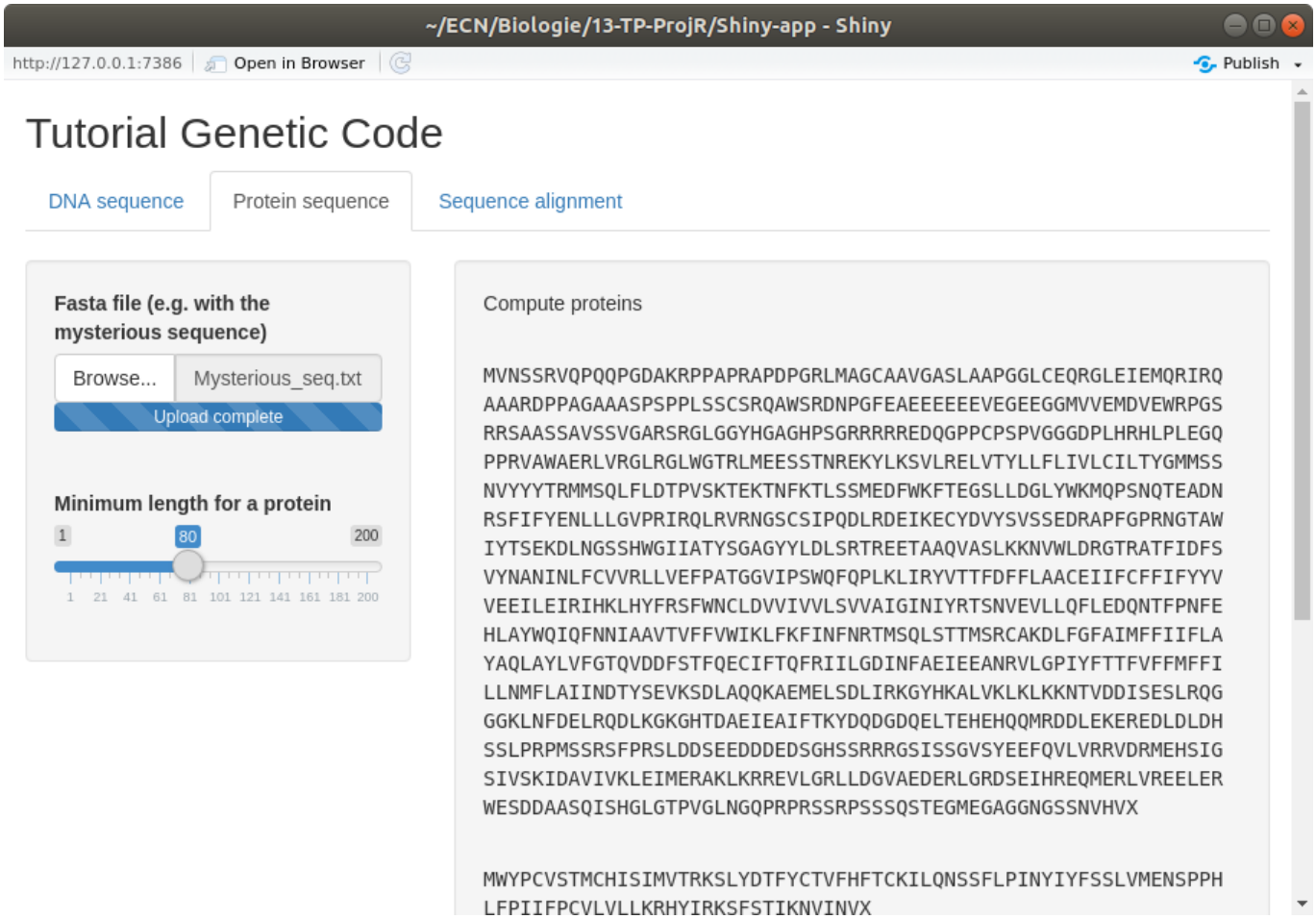


Figure 10: Proteins

7 Alignment

Exercise 7.1

a) What are the pros and cons of aligning amino acids instead of nucleotides?

The pros are

- It's easier to treat, because if we have a sequence of $3n$ nucleotides, when we translate we get a sequence of n amino-acids
- We cut off the redundancies. As we know, from the 64 codons, we have only 21 amino-acids.

And the cons are

- There is not only one sequence, but 6 sequences of amino-acids that we have to pay attention, because it's possible to slide over the sequence.
- The alphabet of amino-acids is 21, while the alphabet of nucleotides is only 4.

b) Back to your working R script (not in Shiny), align the longest mysterious translated protein from the previous part against the two reference proteins - you generated the PKD1 and PKD2 sequences in the fasta format during the Unix session (exercise 2). What was the mysterious sequence ?

```
requireNamespace("Biostrings", quietly = TRUE)
library("seqinr")

TPfolder <- "~/ECN/Biologie/13-TP-ProjR/"
mysterious_filename <- paste(TPfolder, "Students/Mysterious_seq.txt", sep = "")
codon_filename <- paste(TPfolder, "Code_shortcut/Genetic_code.txt", sep = "")
reference_filename <- paste(TPfolder, 'Students/PKD1-PKD2_seq.txt', sep = "")
proteins_filename <- paste(TPfolder, 'Students/Protein_sequences.txt', sep = "")
alignment_filename <- paste(TPfolder, 'Students/alignment.txt', sep = "")

ref.prot.sequences <- Biostrings::readAAStringSet(reference_filename)
target.prot.sequences <- Biostrings::readAAStringSet(proteins_filename)
myst_sequences <- read.fasta(file = mysterious_filename, seqtype = "DNA", forceDNAtolower = T)

myst_sequence <- myst_sequences$seq0
myst_proteins <- find.mysterious.proteins(myst_sequence)
biggest_myst_protein <- myst_proteins[which.max(nchar(myst_proteins))]
pair.alignment = Biostrings::pairwiseAlignment(pattern = ref.prot.sequences,
                                               subject = biggest_myst_protein,
                                               substitutionMatrix = "BLOSUM62",
                                               type = "global")
if( which.max(pair.alignment@score) == 1 ){
  print("[ PKD1 - 4303aa ] is the sequence")
}else{
  print("[ PKD2 - 968aa ] is the sequence")
}
```

OUTPUT:

```
[1] "[ PKD2 - 968aa ] is the sequence"
```

c) Align your pair's sequence against the two reference proteins and identify which of the two reference proteins was assigned to you (use pairwiseAlignment). Write the result in a file (writePairwiseAlignments).

My sequence is the sequence 12. So, I comparated them.

```
mysequence <- target.prot.sequences$seq12
pair.alignment = Biostrings::pairwiseAlignment(pattern = ref.prot.sequences,
                                               subject = mysequence,
                                               substitutionMatrix = "BLOSUM62",
                                               type = "local")
Biostrings::writePairwiseAlignments(pair.alignment, alignment_filename)
if( which.max(pair.alignment@score) == 1 ){
  print("[ PKD1 - 4303aa ] is the sequence")
```

```
}else{
  print("[ PKD2 - 968aa ] is the sequence")
}
```

OUTPUT:

```
[1] "[ PKD1 - 4303aa ] is the sequence"
```

d) By looking at the alignment, identify by eye the mutation/deletion/insertion between your sequence and the reference sequence for your protein by looking at the alignment result. Mutations are annotated in this way : G7602T means that a Glycine is mutated into a Threonine at the 7602th reference position.

When written in the file, we could see well the comparison of the proteins. The sequence matches with the proteins PKD1 almost entirely between the values 3641 and 4196, except the cases:

- At position **3778**, **seq12** has **R**(or Arg = CG∗/AGA/AGG) while **PKD1** has **S**(or Ser = TC∗/AGT/AGC)
- Between the positions **4152** and **4155**, **seq12** has **PQFR** while **PKD1** has —-

So, there is a mutation **S3778R**.

Exercise 7.2

a) Compute the pairwise alignment for each of the 32 AA sequences against all the others in *Protein_sequences.txt*. Build a 32 by 32 scores matrix. Use heatmap to plot the clustered scores matrix and comment the results.

```
len = length(target.prot.sequences)
matrix.scores = matrix(0, len, len)
for(i in 1:len){
  for(j in 1:len){
    pair.alignment <- Biostrings::pairwiseAlignment(pattern = target.prot.sequences[i],
                                                    subject = target.prot.sequences[j],
                                                    substitutionMatrix = "BLOSUM62",
                                                    type = "global")
    matrix.scores[i, j] = pair.alignment@score
  }
}
heatmap(matrix.scores, symm = T, main = "Alignment of the scores of 32 proteins")
```

OUTPUT:

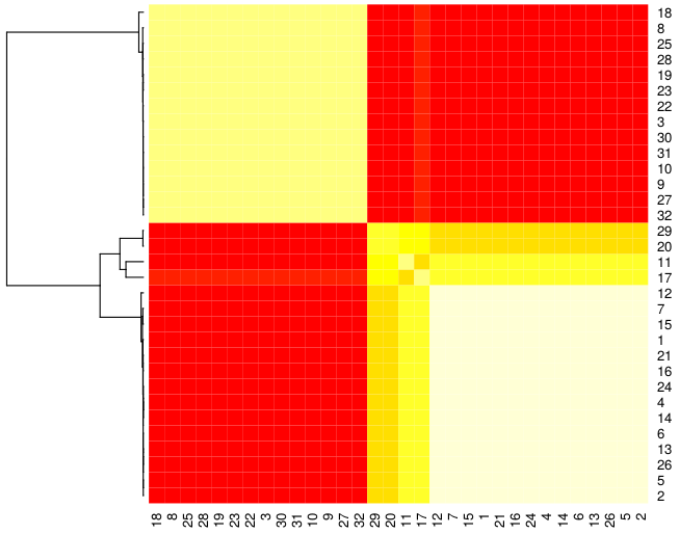


Figure 11: Heatmap

b) Do you identify clusters? How can you retrieve the sequences? What is your interpretation?

Exercise 7.3

a) Why do we use substitution matrices to deal with amino acids changes instead of constant penalties ?

Exercise 7.4

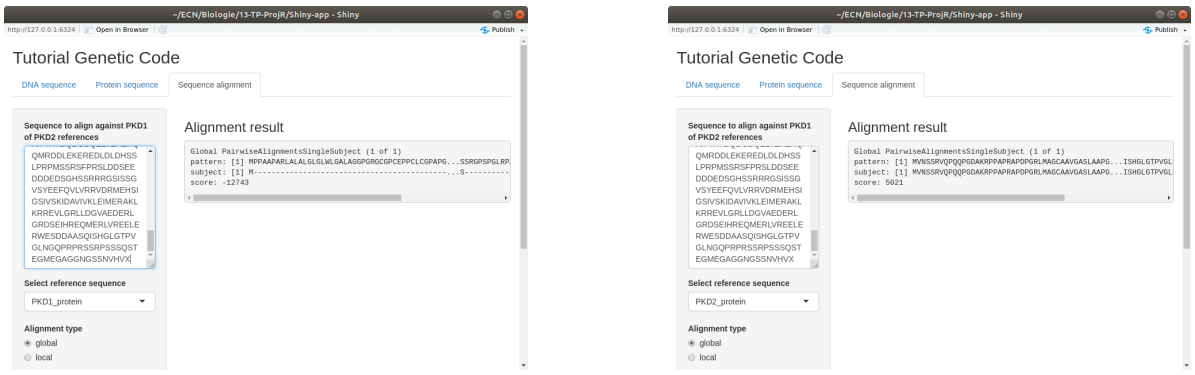
a) Fill the **output$align.out** part of the **server.R** code.

```
output$align.out <- renderPrint({
  ref.sequence <- clean.sequence(ref.prot.sequences[input$select.ref.seq])
  in.seq <- clean.sequence(input$subject.seq)
  if(is.null(in.seq))
    return(NULL)
  pair.alignment <- pairwiseAlignment(attern = ref.sequence,
                                      subject = in.seq,
                                      substitutionMatrix = input$align.matrix,
                                      type = input$align.type)
  return(pair.alignment)
})
```

b) Add two **sliderInput** objects in the UI, one to pilot the gap opening cost (default=10) from pairwiseAlignment, the other one to pilot the gap extension cost (default=4). Their min and max will be 0 and 100.

```
tabPanel("Sequence alignment",
         br(),
         sidebarLayout(sidebarPanel(
           textAreaInput(inputId = "subject.seq",value = "
               MARVPRVRPPHGFALFLAKEEARKVKRLHGMLRSLLVYMLFLLVTLLASYGDASCHGHAYRLQSAIKQELH",
                         label = "Sequence to align against PKD1 of PKD2 references",
                         resize = 'both',
                         cols = 80,
                         rows = 10),
           selectInput("select.ref.seq", "Select reference sequence", c("PKD1_protein", "
               PKD2_protein"), "PKD1_protein", multiple = F),
           radioButtons("align.type", "Alignment type", c("global", "local"), "global"),
           selectInput("align.matrix", "Substitution matrix", c("BLOSUM45", "BLOSUM50",  "
               BLOSUM62", "BLOSUM80", "BLOSUM100", "PAM30", "PAM40", "PAM70", "PAM120", "PAM250")
               , "BLOSUM62"),
           sliderInput("slider1", "LabelSlider1", 0, 100, 10),
           sliderInput("slider2", "labelSlider2", 0, 100, 4)
         ),
         mainPanel( h3("Alignment result"), verbatimTextOutput("align.out") )
         )
)
```

c) Copy and paste the mysterious sequence from panel 2 and align it to the two reference sequences.



(a) Comparison with PKD1 (b) Comparison with PKD2

Figure 12: Comparison of Mysterious Sequence

d) Comment the effects of every parameter.

Exercise 7.5 (bonus)

Combine the **sapply** lines from the second chunk of code **util.R** into a single call.

Exercise 7.6 (bonus)

In the graphical rendering of the shiny application, color each aminoacid letter according to its physico-chemical properties.