Exercise in Bioinformatics

Modelling biological systems, BIOS13, Lund University
Mikael Pontarp and Yun-Ting Jang

Work on these exercises on your own or in pairs. You find example solutions on Canvas, but do try on your own first! Try to work your way through at least exercise 8.
Before you start, it may be worthwhile to freshen your memory on how to write functions in R (slides 6-12, "Structure and Plotting in R.pdf"), and how vectors, lists, and strings work in R (slides 9-14, 19-20, "Basic programming in R.pdf").

1. Download the human mitochondrial sequence 'NC_012920' from the genbank database (https://www.ncbi.nlm.nih.gov/genbank/). Save it as a 'fasta' file. Alternatively, download NC_012920.fasta from the course webpage.

2. Open Rstudio. Install the 'seqinr' package, if it is not already installed, and load it:

   ```
   > install.packages('seqinr')
   > library('seqinr')
   ```

   (Read more about seqinr here:
   https://cran.r-project.org/web/packages/seqinr/seqinr.pdf)

3. In the console window, load the sequence data and have a look at it:

   ```
   > mito_fasta <- read.fasta('NC_012920')
   > print(mito_fasta) # the whole sequence will not be printed
   ```

   The actual sequence is stored in '`mito_fasta$NC_012920.1`'. As you may see, it is a vector of characters ['g','a','t','c'…], corresponding to the DNA nucleotides. You may want to extract the sequence vector into a separate variable:

   ```
   > mito_seq <- mito_fasta$NC_012920.1
   ```

4. Write a *function* `nucleo_content` that takes a sequence as input (in form of a vector of characters), calculates the proportions of the G, C, A and T nucelotides and prints the result according to:

   ```
   G : …%
   C : …%
   A : …%
   T : …%
   ```

   Test your function, for example with a simple input:

   ```
   > nucleo_content(c('a','g','c'))
   G : 33.33333%
   C : 33.33333%
   A : 33.33333%
   T : 0%
   ```

5. Write another *function* `create_complement` that takes a sequence as input and *returns* the *complementary* sequence (see lecture notes, remember to read 'backwards').
   Hints:
   Logical indexing is useful here. For example: `v[v==1] <- 2` will change all elements equal to 1 in vector v to the value 2.
   To reverse a vector v you can use `v <- v[length(v):1]`.

   Test your function, for example like:

```
> create_complement(c('a','c','g','t'))
[1] "a" "c" "g" "t"
```

6. Write a *script* (in a separate file) that

   a. Clears all variables:

```
> rm(list=ls())
```

   b. Loads the mitochondrial sequence, as above

   c. Prints the length of the sequence

   d. Calls `nucleo_content` to print the proportions of the G, C, A and T nucelotides.

   e. Calls `create_complement` to create the complementary sequence and then `nucleo_content` to print its nucelotide proportions.

   Was the proportions of the complementary as expected (what did you expect)?

7. (****extra****) Write a *function* that takes a sequence as input and plots the GCAT proportions as *moving averages* across the sequence. In other words, plot the proportion of G in a frame of, say, 200 nucleotides, and then let the frame slide across the whole sequence. Make one plot per nucleotide.

8. Triplets of nucleotides correspond to codons (GGC, CGT, ATG, etc.). Each codon codes for an amino acid, a part of a protein. The interpretation of a DNA sequence can be done in no less than six different ways. In each direction (forward and the complement), there are three different *reading frames*, corresponding to starting the first codon in position 1, 2 or 3 (see lecture notes).

   Of particular interest are the *start* (ATG, ATA, ATT, ATC, GTG) and *stop* (TAA, TAG, AGA, AGG) codons. (These may differ between organisms and nuclear origin. The ones listed here apply to vertebrate mitochondria).

Write a new *function* `find_codon` that takes a sequence (a vector), a codon (as a string of length 3) and a *reading frame* (a number 1, 2 or 3) as input and *returns* a vector of positions of the codon in the sequence.

Hint: The function `substr(s,p,p)` gives the character at position `p` of a string `s`. Another option is to use `strsplit(s,'')[[1]]`, which converts a string to a vector of characters, which can be indexed as usual (the return value of `strsplit` is a list, and we want the first component of that list, hence the '`[[1]]`' at the end).

As before, *test* your function with simple input, for example:

```
> find_codon( c('a','c','g','g','a','t'), 'gat', 1)
[1] 4
```

To further check your results, I got this when searching for the start codon ATG in the first reading frame of the forward sequence:

```
find_codon( mito_seq, 'atg', 1 )
 [1]    73   415   772  1090  1399  1567  1798  2629  2749
[10]  3313  3355  3367  4444  5665  5869  6085  7039  7237
[19]  7390  7495  7504  7894  8071  8170  8389  8527  8695
[28]  8836  9037  9223  9385  9943  9964 10369 10429 10450
[37] 10723 10804 11164 11344 11401 11557 11833 12004 12343
[46] 12499 13357 14833 14836 15232 15559 15880 15931 16087
[55] 16567
```

9. Now use the `find_codon` function in another function `find_start_codon` that takes a sequence and reading frame as input and returns a (sorted) vector of the positions of *all* start codons in the sequence.
   To check your code, I got the following positions in the first 500 nucleotides of the first reading frame of the forward sequence:

```
> find_start_codon(mito_seq[1:500],1)
 [1]  13  49  73  82 148 160 178 205 211 232 235 238 244 397 415 448
[17] 451 484
```

10. Write the function `find_stop_codon` in a similar fashion.
    For reference, I got this:

```
> find_stop_codon(mito_seq[1:500],1)
[1]   7 199 217 226 274 334 388 424 442
```

11. An *open reading frame* is the DNA sequence between a start-codon and the next stop-codon (including both the start and the stop). It can include several start-codons in between. Can you, just by looking at the output above, find the first open reading frame of the first reading frame?
    (The answer should be (13-199), or rather (13-201) to include the whole stop codon. Print out the sequence if you wish: `> mito_seq[13:201]`)

12. (*advanced*) Write a function `find_all_open_frames` that finds all open reading frames of an input sequence in a given reading frame. The output should be a matrix with two columns, where each row contains the start and end position of a single frame.
    I got this from the first 500 nucleotides of the first reading frame:

```
> find_all_open_frames(mito_seq[1:500],1)
     [,1] [,2]
[1,]   13  201
[2,]  205  219
[3,]  232  276
[4,]  397  426
```

    You may want to filter out very short frames (such as 205-219, which only contains 5 codons).

13. Which of all the 6 possible reading frames contains the longest open reading frame?

14. (*advanced*) Write a function that *translates* a sequence to its corresponding sequence of amino acids. There are 20 amino acids, each labelled with a single letter. The mitochondrial translation scheme is as follows (you read a codon as a column and find the amino acid label at the bottom. Labels '*' are stop codons):

```
Base 1    = TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
Base 2    = TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
Base 3    = TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
Am. acid  = FFLLSSSSYY**CCWWLLLLPPPPHHQQRRRRIIMMTTTTNNKKSS**VVVVAAAADDEEGGGG
```

(reference: https://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?chapter=cgencodes#SG2)

15. (*advanced*) Plot the frequency of the different amino acids of the longest reading frame in a histogram. Something similar to this (using `barplot`):