# Contents



# COMP1730/6730 2023 Semester 1, Project-3 (Bioinformatics)

*The assignment is an exercise in Bioinformatics which requires to align short gene sequences and calculate the mutation likelihood of one sequence transforming into another. Computationally, it involves strings, lists and doubly-nested lists, and some operations which need to be performed on such lists. Alternatively, one can use Numpy arrays, 1-dimensional arrays instead of lists, 2-dimensional arrays instead of nested lists. The estimated size (including the code comments and docstrings) is approximately 250 lines of code. The time effort is 10–12 hours.*

This is slightly modified problem which was used as a programming assignment in Princeton University in 2010s.

## The problem

The science of *Bioinformatics* is broadly defined as "the application of tools of computation and analysis to capture and interpret biological data" (A. Bayat, "Science, medicine, and the future: Bioinformatics", *British Medical Journal*, **324** (2002), p.1018-1022). One fundamental part of it is a sequence analysis of DNA and proteins, which is often called *Genomics*. In this assignment, we will learn how to solve two simple problems of genomics: likelihood of gene mutation and gene sequence alignment.

Both problems deal with two gene sequences which can be described as strings of symbols representing four *nucleotides* (or, *bases*), `A`, `C`, `G` and `T`:

```
GATACAAAACTATGATAATTGCTAGATACATAGATACAGATAGATACAGATAGTCG
```

The first problem quantifies the difference between two sequences (of possibly unequal length), so that the two can be *aligned* and considered as mutational variation of one another. The second problem calculates the likelihood that the two sequences are related by mutation.

### Problem One: DNA Sequence Alignment

Often a protein function encoded in a genetic sequence is determined by comparing this sequence to another sequence (perhaps belonging to a different organism) which

is already known. Comparing, or *aligning*, the two sequences is a viable method which may ease the task of determining protein functions.

The alignment is performed by calculating the *edit-distance* of sequences. The edit-distance is also used in coding theory, spell checking, plagiarism detection, version control, computational linguistics and other areas. The method to calculate the edit-distance of two sequences consists in aligning them (allowing gaps to make up for missing symbols since the sequences can have unequal length). Each mismatch or gap in an alignment incurs a penalty. In genomics applications, the following penalty function is often used:

| **Penalty** | **Cost** |
|---|---|
| Gap | 2 |
| Mismatch | 1 |
| Match | 0 |

Two examples next illustrate how this penalty assignment works:

| **Penalty** | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 2 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Sequence 1** | A | A | C | A | G | T | T | A | C | C |
| **Sequence 2** | T | A | A | G | G | T | C | A | - | - |

and

| **Penalty** | 1 | 0 | 2 | 0 | 0 | 1 | 0 | 2 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Sequence 1** | A | A | C | A | G | T | T | A | C | C |
| **Sequence 2** | T | A | - | A | G | G | T | - | C | A |

The total cost of alignment is just a sum of all penalties; in the above examples, it's 8 for the first alignment and 7 for the second. The edit-distance is defined as the smallest total penalty among all possible alignments. One can tackle this problem in two ways: a recursive one (relatively simple) and a dynamic programming.

A *recursive* (naive) approach is quite simple, bit it can be very demanding computationally since the number of possible alignments grows exponentially with the length of sequences. The algorithm of recursive approach for calculating the edit-distance goes like this:

- Take two original sequences $x, y$ of the length $M, N$ correspondingly, and denote their edit-distance:

$$\text{edist}(x, y) \equiv \text{edist}(0, 0),$$

where edist$(i,j)$ is the edit-distance of two *suffix sequences* $x[i..M]$ and $y[j..N]$ (in other words if $x = x_0 \ldots x_M$, its suffix sequence $x[i..k]$ is the same as a subsequence $x_i \ldots x_k$, $i \geqslant 0$, $k \leqslant M$, and similarly for the suffix sequence $y[j..N]$). The task now is to write down a recursive relation for edist$(i,j)$.

- In the optimal alignment, there are three possibilities:
  - $x[i]$ and $y[j]$ align with penalty 0 or 1 depending whether they match or not, the remaining contribution is edist$(i+1, j+1)$;
  - there is a gap in the $y$ sequence which contributes penalty 2, and the remaining term comes from aligning $x[i+1..M]$ and $y[j..N]$ with the contribution edist$(i+1, j)$;
  - there is a gap in $x$ sequence (penalty 2), and the remaining alignment contributing edist$(i, j+1)$.

  The optimal alignment for $x[i..M]$ and $y[j..N]$ must be obtained by minimising edist:

$$\text{edist}(i,j) = \min\Big\{\text{edist}(i{+}1, j{+}1){+}(0 \text{ or } 1), \text{edist}(i{+}1, j){+}2, \text{edist}(i, j{+}1){+}2\Big\}, \quad i < M,\ j < N.$$

- When $i = M$, the remaining alignment with an empty sequence contributes $2(N-i)$, and analogously for $j = N$ case:

$$\text{edist}(M, j) = 2(N-j) \quad \text{and} \quad \text{edist}(i, N) = 2(M-i)$$

The above cases are enough to calculate edist$(0,0)$ for any $x$ and $y$, and thus calculate edist$(x,y)$. Your task is to implement this algorithm.

A more sophisticated *dynamic programming* approach allows to overcome the inefficiency of the recursive approach related to multiple repetitions of the same computation when one moves from one recursive level to the previous one. In a case of two sequences having the length $N$, the number of recursive calls (almost all of them redundant) grows like $2^N$. The dynamic programming allows to avoid unnecessary computation by breaking the original problem into subproblems, *storing* the result of their solution (when those are obtained for the first time), and then simply re-using those results when they are needed instead of repeating same computation over and over again. This technique is also called *memoization*. We shall discuss it in the lectures on a simple example. If you decide to employ this approach here, you need research this topic to the greater extent yourself (by using references provided in *Bibliography*) and apply what you will have learnt.

**Problem Two: Likelihood of gene mutation**

Once you found two aligned sequences (which belong to different species), you may ask the question whether they are related by mutation. Here, we have data

(two aligned sequences) and the problem is to test if the model which describes the mutation of genes can describe these data with sufficient likelihood.

The model here is represented by the probability values with which every nucleotide occurs in a gene sequence, and by the mutation matrix $\mathbb{M}$ which describes the probabilities of each nucleotide `A, C, G` and `T` turning into some other in the process of copying the genetic information (when a new sequence is created by copying an existing one). We shall assume that the model is fixed: the probability of every base to occur is given by four values (which sum up to 1):

$$\pi_A = 0.1,\ \pi_C = 0.4,\ \pi_G = 0.2,\ \pi_T = 0.3.$$

The probabilities of base changes during a single step of copying are described by the matrix:

$$\mathbb{M} = \begin{bmatrix} 0.976 & 0.010 & 0.007 & 0.007 \\ 0.002 & 0.983 & 0.005 & 0.010 \\ 0.003 & 0.010 & 0.979 & 0.007 \\ 0.002 & 0.013 & 0.005 & 0.979 \end{bmatrix}$$

where we assumed an alphabetical ordering of the nucleotides `A, C, G` and `T`, and the Python-like indexing rule (the first element has the index 0), the matrix element, for example, $\mathbb{M}_{23}$ gives the probability of one-step mutation `G` $\longrightarrow$ `T`.

If we have two *aligned sequences* which may be related by a *single* act of mutation,

$$\underbrace{CCAT}_{s_1} \longrightarrow \underbrace{CCGT}_{s_2},$$

the likelihood for going from one sequence to the second is calculated as follows:

$$P(s_1, s_2, 1) = \pi_C\, \mathbb{M}_{cc}\, \pi_C\, \mathbb{M}_{cc}\, \pi_A\, \mathbb{M}_{ag}\, \pi_T\, \mathbb{M}_{tt} = 0.4{\cdot}0.983{\cdot}0.4{\cdot}0.983{\cdot}0.1{\cdot}0.007{\cdot}0.3{\cdot}0.979 = 0.000300$$

If the number of copying is 2, the likelihood of mutation is calculated as above in, except that the matrix elements are now taken from the matrix $\mathbb{M}^2$, the square of $\mathbb{M}$,

$$\mathbb{M}^2 = \begin{bmatrix} 0.976 & 0.010 & 0.007 & 0.007 \\ 0.002 & 0.983 & 0.005 & 0.010 \\ 0.003 & 0.010 & 0.979 & 0.007 \\ 0.002 & 0.013 & 0.005 & 0.979 \end{bmatrix}^2 = \begin{bmatrix} 0.953 & 0.020 & 0.013 & 0.015 \\ 0.005 & 0.966 & 0.010 & 0.020 \\ 0.007 & 0.020 & 0.959 & 0.015 \\ 0.005 & 0.026 & 0.010 & 0.959 \end{bmatrix}.$$

For three-step copying, the matrix used is cube of $\mathbb{M}$, $\mathbb{M}^3$, and so on. The general tendency is that the diagonal elements decrease (the probability of no mutation

falls), and the off-diagonal elements increase (the mutation probability rises with the number of copying).

Your task will be to calculate the probability $P(s_1, s_2, n)$ of converting a sequence $s_1$ into a sequence $s_2$ in $n$ mutation steps. This probability has a maximum $n_{\max}$ which can be used to analyse the mutation model. You will have to confirm existence of the maximum and calculate $n_{\max}$. When two sequences belong to taxonomically related species, the calculated maximum may be used to verify the validity of model (by comparing the number of steps corresponding to the maximum and the genes position on the taxonomy tree).

There is a question of how to deal with gaps in the alignment of two sequences. A reasonable approach would be to accept that the gap could contain any of the four nucleotides, so that the mutation probability factor can be 1. For example (extending on the case of the equation for $P(s_1, s_2, 1)$):

$$CCGAT \to CC{-}GT: \quad \pi_C \mathbb{M}_{cc} \pi_C \mathbb{M}_{cc} \pi_G \mathbb{M}_{g\to\text{any}} \pi_A \mathbb{M}_{ag} \pi_T \mathbb{M}_{tt}$$

$$P_{CCGAT\to CC-GT} = 0.4 \cdot 0.983 \cdot 0.4 \cdot 0.983 \cdot 0.2 \cdot 1 \cdot 0.1 \cdot 0.007 \cdot 0.3 \cdot 0.979 = 0.0000600$$

Computationally, this is a straightforward problem: you will have to represent the $4 \times 4$ mutation matrix as a nested list, and define the operation of matrix multiplication on such doubly-nested lists. Alternatively (and more performant), one could use a NumPy 2D array, and use the built-in matrix multiplication.

As a part of the analysis, you will plot the found likelihood $P(s_1, s_2, n)$ as the function of $n$.

## The programming tasks

Your task will be to make sense out of the above algorithms and implement them in Python. Namely, this is what your program should be able to do:

1. When started, make the program to read two strings of nucleotide symbols (*A, C, G* and *T*, the case doesn't matter) from a data file, whose name is either passed as a command-line argument, or typed in on the program prompt (the second option is for Windows-challenged). Also pass a second command-line argument which will set the maximum length of the sequences (for prompt-based interface, make the program to request it as the second input). Verify that the content of sequences is valid, and their length is within the acceptable limit.
2. Calculate the edit-distance and the corresponding alignment of the sequences represented by the above strings using either a recursive or a dynamic programming approach. The found value of edit-function and the aligned sequences must be reported to the user.

3. Find the value of $n_{\max}$ where the likelihood has maximum and calculate this maximum. Both quantities must be communicated to the user.
4. Plot the likelihood-vs-$n$ function using the `matplotlib` package.

## References


- "The Idiot's Guide to the Zen of Likelihood in a Nutshell in Seven Days for Dummies, Unleashed" by Peter G Foster. A preprint of this article (apparently, it has never been published) is not easily found online these days, therefore, we provide its local copy, The Idiot Guide. The first half of the paper contains discussion of how to calculate the likelihood of mutating aligned sequences.
- The Wikipedia article Sequence alignment.