Hands-on Python – Final Project

Franziska Kranz

September 14, 2017

Preliminary Informations

- Deadline: 3 November 2017 20:00
- No Teamwork! Code will be checked for plagiarism. Plagiarism leads immediately to failing the course.
- Short presentations in a two-on-two session with me. Preferably in the time 7. November - 10. November (or earlier if you hand in you solution early!), but I need at most one workday between the hand in and the meeting.
- Please write clean code: meaningful variable names, comments where they are useful... (Makes the corrector happy, which can only be good for you)
- If you use any external modules, write a small README file, containing all the modules I might need to install.

Task1: Sequence alignment backend

The goal of the first two tasks is the alignment of two DNA sequences x and y. To achieve this goal in the first tast two steps have to be performed.

First a $(N+1) \times (M+1)$-Matrix S (where N and M are the lengths of the sequences) has to be filled with the alignment scores. Position (i,j) get the score of the best alignment for the first i positions of x against the first j positions of y. Moving from (i,j) to $(i,j+1)$ introduces a gap in the first sequence, moving to $(i+1,j)$ introduces a gap in the second sequence, while moving to $(i+1,j+1)$ keeps the sequences and may produce a mismatch. The formula for computing the entries of this scoring-matrix S is:

$$S(i,j) = \max \begin{cases} S(i-1,j-1) + s(x_i,y_j) \\ S(i-1,j) + indel \\ S(i,j-1) + indel \end{cases}$$

$$S(0,0) = 0$$

indel is the gap penalty, i.e. the negative score assigned when introducing a gap in one of the sequences and $s(x_i, y_j)$ the score we get, when comparing two bases. This scoring function s can be expressed by a 4×4 matrix, e.g.

	A	C	G	T
A	10	-5	0	-5
C	-5	10	-5	0
G	0	-5	10	- 5
T	-5	0	-5	10

The second step is to recreate the best alignment from the scoring-matrix S. We start with two empty sequences at $S(N+1, M+1)$ and go to $S(1,1)$. In each step we have to redo the calculations from the first step to see where we came from. More precisely in step (i,j) we check if $S(i,j) - s(x_{i-1}, y_{j-1}) = S(i-1, j-1)$, $S(i,j) - indel = S(i-1, j)$ or $S(i,j) - indel = S(i, j-1)$. In the first case we append x_{i-1} to the first and y_{j-1} to the second sequence. In the second case we append x_{i-1} to the first sequences and a gap to the second - indicate gaps by a "_". In the third case we do it the other way round. We follow this procedure until i or j are 0 and fill the remaining sequence with gaps. Since we went from back to front through the matrix, we have to reverse our sequences in the end.

Complete the given python file to implement the described algorithm. The function signatures (i.e. the name, parameters and return values) should not be changed, but you can of course add more internal functions. Use two global variables in the beginning of the file for $indel = -4$ and the scoring function s.

Task2: Sequence alignment frontend

The goal of this task is the provide a usable program for sequence alignment with the possibility of different output files. Use the functions for the sequence alignment of two sequences you implemented in Task1, especially the function `find_alignment` which should return a tuple with two aligned sequence strings.

Your program should work as follows:

1. Read a file with sequences, get the filename from command line option.
2. The reference sequence should be given as command line option. If none is given use the first sequence of the file as reference.
3. Output option 1 - Output to commandline: For each sequence, print the aligned sequence pair on top of each other and below a string indicating the mutations, use * for perfect match, M for a mutation, D for deletion and I for insertion (always the sequence related to the reference). Add a blank line between sequence pairs, so you have one "block" with reference - sequence - differences per input sequence. Also, the letters within the sequences should always be seperated by a space for better readability.

4. Output option 2 - Output to text file: Use the same output schema as before, but save the output in a text file..
5. Wich output option is chosen should be indicated by commandline option e.g. –file

Task3: PDB reading, writing and manipulation

- Write a reading routine for a PDB file. This file can either contain a single crystal structure or mulitple one (e.g. from a simulation). To check the structure of such a file, open the file in a text editor and have a close look. Also the exact specification can be found here: http://www.wwpdb.org/documentation/file-format-content/format33/v3.3.html The Title and Coordinate section are probably the most interesting for you.
- Print information about the file like the title or the number of contained structures and ask the user how he or she wants to proceed. This possibilities should be given, if applicable:
  - Write one specific structure (if there are more than one in the file) to a new pdb file.
  - Center the protein at $(0, 0, 0)$
  - Translate the protein by a given vector
  - Rotate the protein by a given angle and axes
  - Compute the root mean square deviation (RMSD) between the starting structure (first structure in file) and the others. Plot the result.
  - Write file: write the newly positioned protein (after centering, rotation, and/or translation) to a new pdb file. This of course only makes sense if the protein has been changed in any way.
  - Quit
- For testing a recommend to open your altered proteins in pymol and see if the can be displayed and if rotation and translation are performed in the right way.