AiB Project 2 - Global alignment with different gap costs

Peter and Ane
https://github.com/PeterKADam/Bioinformatics/tree/main/Projects/Alignmentproject/AIBproject2

Introduction:

We successfully implemented Linear and affine global alignment functions in quadratic time and space. File input and output is mostly functional, Input works as a single multisequence fasta file or as two single sequence files. output only saves a single optimal alignment.

Methods:

The program is split into two files: **global_linear.py** and **global_affine.py** for calling the programs linear and affine implementation through the command line. The program expects at minimum: A fasta file with at least two sequences or a pair of fasta files, a Match matrix, and gapcost function parameters.

Options

–multifile file
    fasta file with two or more sequences to be aligned. only the top two sequences will be used
–seq1 file
    fasta file with a single sequence entry to be aligned. requires –seq2 argument
–seq2 file
    fasta file with a single sequence entry to be aligned. requires –seq1 argument
–matchscore file required
    Matchscore file to be used. Is a required argument, formatting of the file can be seen in matchscores.txt
–gapcost int required
    cost of gaps*k in linear alignments or cost of extending existing gap in affine alignments
–gapopen int
    cost to open gap in affine implementation. required for Affine.

Example:

Global linear alignment with 2 sequences in 1 file:
$python global_linear.py –multifile multifasta.fa –matchscore matchscores.txt –gapcost 5
Global affine alignment with 2 sequences in 2 files:
$python global_affine.py –seq1 seq1.fa –seq2 seq2.fa –matchscore matchscores.txt –gapcost 5 –gapopen 5

File IO is mostly handled through biopython SeqIO. if theres any problems, its probably their fault.

the Matchscore are implemented in this format:
A 0 5 2 5
C 5 0 5 2
G 2 5 0 5
T 5 2 5 0

in a short summary, it is a 4x4 matrix of match, transversion and transitions, that is expanded to a 5x5 matrix using the row headers rotated to also apply symmetrically as column headers

IO is implemented in the frontend file **global_linear.py** and **global_affine.py** aswell as in **files.py**. all matrix computation is done in **AIBP2.py** which is called by the frontends.

The output of the programs is the very first optimal alignment in the array of optimal alignments, in a timestamped fasta file. We did not see any way to output multiple possible alignments in a readable format in a fasta file, although all optimal alignments are computed.

Linear Gapcost:

We have used the exact same function for this as we did at the last project.

Affine Gapcost:

We made a function that if given two sequences, a match score matrix, *m*, and a gap opening (*beta*) and gap extension score (*alpha*), could produce an alignment/score matrix *S*, a deletion matrix *D*, and an insertion matrix *I* by executing this for each cell in the three matrices:

$$S(i,j) = \min \begin{Bmatrix} 0 & if\ i = 0\ and\ j = 0 \\ S(i-1,j-1) + s[A[i],B[j]]) & if\ i > 0\ and\ j > 0 \\ D(i,j) & if\ i > 0\ and\ j \geq 0 \\ I(i,j) & if\ i \geq 0\ and\ j > 0 \end{Bmatrix}$$

$$D(i,j) = \min \begin{Bmatrix} S(i-1,j) + \alpha + \beta & if\ i > 0\ and\ j \geq 0 \\ D(i-1,j) + \alpha & if\ i > 1\ and\ j \geq 0 \end{Bmatrix}$$

$$I(i,j) = \min \begin{Bmatrix} S(i,j-1) + \alpha + \beta & if\ i \geq 0\ and\ j > 0 \\ I(i,j-1) + \alpha & if\ \geq 0\ and\ j > 1 \end{Bmatrix}$$

Afterwards we made a function that if given an alignment matrix, a row number *i*, a column number *j*, a match score matrix, and a gap opening and gap extension score, could find traceback arrows, by examining whether a specific score *(i, j)* could be obtained by adding the score diagonally above it *(i-1, j-1)* with the match/mismatch score, or adding either the upper *(i-1, j)* or left *(i, j-1)* score with the gap opening and gap extension score. The function also tests if the score matches an extension of a gap that has already been opened. For simplicity we will just explain it for an Insertion, but it is the exact same method for Deletions.

The functions loop over all possible values *i<=k<0*, and examine whether the value of the gap function g=k*alpha+beta matches the score difference between *(i, j)* and *(i-k, j)*.

Then we made a traceback recursion function, that for each arrow obtained from the previous function could find their traceback arrows. And lastly we made a traceback function that could follow the arrows, and thereby obtain all optimal alignments.

Test:

There are correct ways to do unittests and there is our way.
During development we tested the code to provide results similar to the examples provided in **project2_examples.txt** we did not do any robust testing, as the final code behaved as expected.

formal testing of the implementation can get messy because the alignments are unordered and we did not feel like the time to test all possible alignments were worth the time, when no issues were observed during development.

Experiments: