Biological Data Analysis (CSE 182) : Assignment 1

Name: Amit Elia

Date: April 23rd 2022

Libraries used:

```
import copy
import math
import sys
import argparse
import random
import matplotlib.pyplot
import numpy
```

References:

BENG181 chapter 5 homework solved by me.

https://web.stanford.edu/class/cs262/notes/lecture3.pdf

1. **File name:** locAL.py

   Usage: python locAL.py <seq_file.fasta> -m [match] -s [mismatch] -d [indel] -a [alignment]

   Commands:

   - read_fasta_file(file_name) - reads a fasta sequences from a text-based file into a dictionary with sequence headers as keys.
   - regular_local_alignment(seq1, seq2, m, s, d) - computing the score matrix, as well as backtracking matrix as presented in class, similar to Hirschberg's Algorithm. Max score among the score_matrix is computed to get the indices of the local alignment's end.
   - outputLocalAlignment - recursively build the path based on the backtracking algorithm. Stops when S[i,j] is 0. Returns a tuple list of characters.
   - format_alignment - formats the output and computes the score of the alignment based on outputLocalAlignment's output.

   Input: python locAL.py p1seqs.txt -m 1 -s -10 -d -1 -a

   Output: q1_results.txt - in proper format

```
Score:119
Length:1245
CCTA-AAACCACTCC----GCAGAA--AAAG--AATA--AG--GCCAAAACACGACTAAAATCGAAAGAC-ATGACAAGTA
AACGAGAAAAGAAA-A-ATA-AA-CGACATACACACTTGTAGGA--A-A--A-ATAA-GAAA-A-AGGGGGAGACGAAG
CAAAGA-AAGGGCAGCTAACCCT-CA-A---GGA-AGAACCAGACA-GAATAAGA--A-AA---ACCCGAAA-GCCACC-A
AA-TGAAA-G-GAC-AATAACACCTAA-GAGCAA--AAT--CAATAA--A--A-CACCGATCCTC-----C---GAGGAT-AACCA-
AGA-GAGACCTAAGAACGAC-A--AG-AAACCAATG--A--A-A-GA-AA--AAG---AA-A--ATGGA-CATCAGAACGA-CT
TAGAA-TGCTGGGAA-AA-AGAAAAATT------ATAAACGAA-G-G-A-TG-G---G--CATAAATTG-G--AC-GAAG-C-C-A-A
GAGATAG-GC-CGA--G--ATAAAACGGAGAACAATAAG--GGAGAC-C--AT-G-GAGAGC--AAAC--CAACCGCAA-CA
AA--TAA-A-GGGGGGGACAAA-AACAAGACCAAC--CC-AAA-C-TGT-CA-G----A-CA-G-GA-A-GAGC-AATAAC--CA
AGACA-GAA-GAA-G-AAACAGGA--GACAAACA-AC--AT--AA-TA--TA-AGA-GCA-CCTAGCTAACAAAAAAGA---C
CAGCAAACGGATTAAGA-AGAT--AAAGAAA----AC---G--T----AA-AGAA-C-A-GTC-AAG---GAACAAGCGA----TAAT
AA-ATG-CAGG-G-AAAAAATGG-G--GA-CAG-ACGAAG--GAAACAACCA-G-AAATAATCTA-ACGCATCGCAGAAG
ATGACACTGCGA--GAA-AATACGAGCCGT-ATACGACAC-A-AAAC--C--G--GGAA-TAA-A-GA-AA---AAAACCATA
CC-CAA-AA-AGA-ACA-AC-GCGA-AAGATGAAACGCTCC--C-AAC--TC-G----G-A---TGAG-CAAAGCCGCCAG--GC
CAAAA-AAGAGAACCA--GAGC-AG-AGCGA--AGCTATGG-GT-A--GAAA-AC---ACCCTAAGCGCGGGTAGTAGA-G
ACGAAAA-A-TAA-AAAC-AGGC-TGAC-C-CGAACATAAGAGCCCACACAAGTAGAAGAACGGAAAGAAAACGAA
AA
CC-AGAAACCA-TCCTAAA--AGAAGGAAAGCAAATAGGAGAA--CAAAACA--A-T--AATC-AAAG-CGAT----A-TAAA
-G-G---AGAAACATA-ACAACCG-C-TACAC-C----A--ATCACACCACA-AAGGAAAGATA-----AG-C---GCAAAGAG-A--
--AG-T-ACCCTGCATACCT--AC--AACCA-A-AT-AA-AAGAGGAGAACTGA---GAAACGCCACCAAAAC--AAACGTG
ACG-AT-A-A-CTAATGA--AACGAATGA-AA-AAGGAGGAT-A--GA-CCTCAAATTCAAAGA-GATGAA-CAT-GAC-AG-
-CTAA-AA-GACAACGAGCAAA--AATGCTAGGAGAC-ATAACCAAGCTAAAGACCA-GGACC--C--AACGACC---GA
AC-GC----AAGAAT-G-AAAATTAGCCCCA-AAA--AACGCGCAC-GAGAAAGAAC-TAAA--GAGCCACA-AAGACAC
AATAGA-A-AGTGCTCGACGGACA-AAAA---A-AA-AA-AAGAA-GAGACACAAATAGA-A-A-CAAAAACAA-AA--G-
AAGCAAACG-AATAT------GACAAAG-A-AA-ACCAACTACCAAAAGCA-GTACATGACACATCATGC-ACAC-AGCGA
A-AACAACAA-A-ATGAAC-AACGAAAACA--ACC-A-AAA-AGACGAATCGAAC-AGG-AGAGAGG-ATCC---C---CAA
AAAAGAGGGCC--C-AAC----TAAGACA-ATGCAAAGAAACGCGACAAAGCCTCGCCAACAGAATCAAC--CAAAGC
ATGAAC-AGC-ACTTTTAA-AACATGT--GGCGC-------GGCGTCGAGCAGTACG--GTT----CAA--ATGCAAA-AAT-TAC
A---A----A-AA-A-GACA-TGC-ACTGAAC----C---CCGTAA-A-GA-ACGAGAAACTTCAAGAAGGAAG-AAGAC-ATAA
GCCAAAA--A-ACCA-AATAATAGACACAG-CT-CGAG-A-A--AAA-GC-CCAACAAACGA-CAGAAAAGAAGGG-GAG
T--AAG--G--AGAAG-CAAAAC---AGAA-CAGGG-GCGAGAA-CGACT-GC---GGAGTAATCGAAAGACATGA--C-AA--
----GTA--A-ACGA-GAAAAGATTAATAAACGA--CAT-ACACA--AAC--AA-A----A-ACAA-TAG-A-AAC-GAAA-AAAA--
AAAA
```

2. **File names:** randomDNA.py, P1P2.py, lp1lp2.py

    a. **Random sequences + summary:**

    File name: randomDNA.py

    Usage: python randomDNA.py [Number of sequences] [Length of a sequence]

    Commands:

    - Random_seq(length) - returns a random sequence of the alphabet "ACGT" with length = length.
    - get_sequences(num, length) - returns a list of random sequences by calling random_seq num times.
    - get_summary(seqs) - returns a dictionary of frequencies of each letter in the alphabet "ACGT" in the list of sequences provided.

    Example Input: 5 50

    Example Output:

```
Namespace(num=5, len=50)
TTGAACGGTATGAATAATAGCACTCAACCACAGCGCTTAAGGGAAGTCTT
CCTCTGTTGAATATAAGACATTGACGATCACCGAGCTCCATCCCAAGCAT
ACAGGTAATATTAATGTTGCAGCAGATCGCCCTTTGATACAAAGGCCCAA
GTTTTAGTGGCTGAATAGGTCTGATGCTCTTATAAAAGCCCCTGCAGGTC
AAGTCGGATATCACAACGGGTCGTCGCACGCAACGGCATAAAAGAGTAGG
{'A': 0.308, 'C': 0.228, 'G': 0.228, 'T': 0.236}
```

b. P1P2 histograms:

File name: P1P2.py

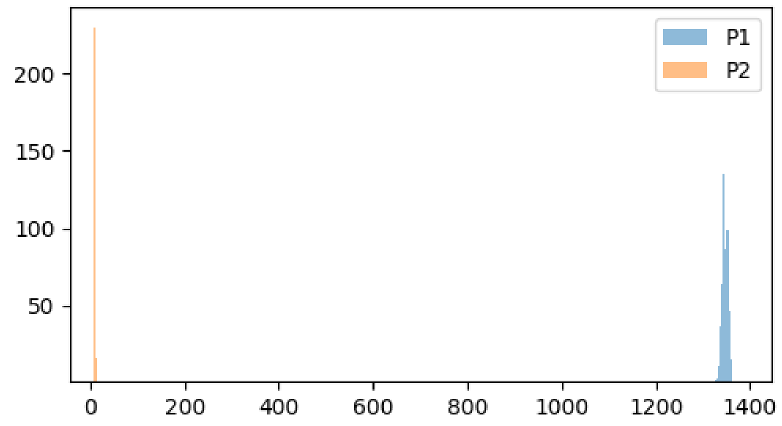
Input: python P1P2.py -m 1 -s -30 -a

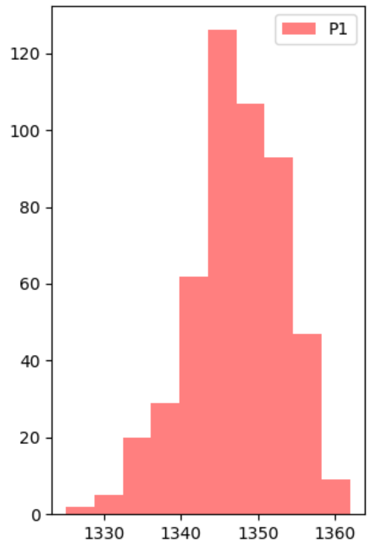
Output:

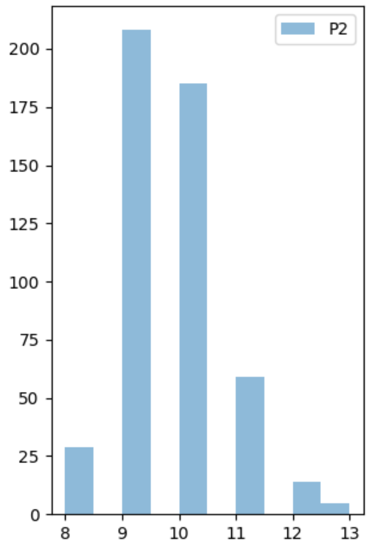
Distribution lists: P1P2.txt

Histograms: Y-axis: number of alignments, X-Axis: length of alignment.

The alignments lengths are different because using P1 parameters where the indel score is 0, the start point of the local alignment will be the first match. The alignment will include all of seq1 and seq2 from that point because insertions and deletions are not negatively scored. This leads the alignment to be larger than the actual sequence length. When the absolute value of the indel score is less than that of the match score, the alignment will more likely find another match before the score of that alignment becomes 0, resetting the local alignment, so the alignment will include more indels than letters.







c. Form of $l_{p1}(n)$ and $l_{p2}(n)$:

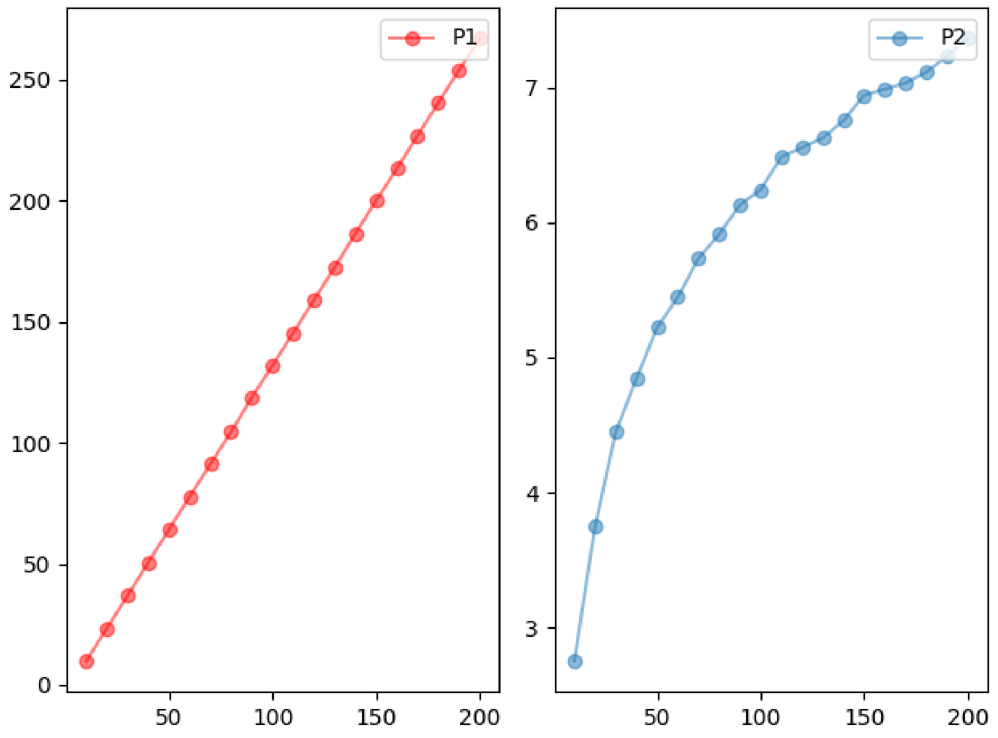
File name: lp1lp2.py

Input: python lp1lp2.py -m 1 -s -30 -a

Output:

Values of $l_{p1}(n)$ and $l_{p2}(n)$: lp1lp2.txt

X - lengths of sequences, Y = length of global alignment.



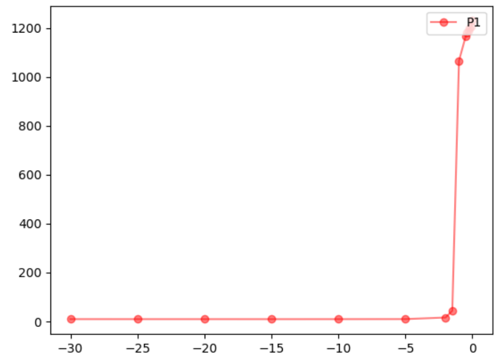
$l_{p1}(n)$form:$y = x$

$l_{p2}(n)$form:$y = \sqrt{x}$

3. **File Names:** lp_n.py
Usage: python lp_n.py <seq_file.fasta> -m [match] -s [mismatch] -d [indel] -a [alignment]

a. Mismatch = Indel, values: [-30, -25, -20, -15, -10, -5, -2, -1.5, -1, -0.5, -0.4, -0.3, -0.2, -0.1 , 0]

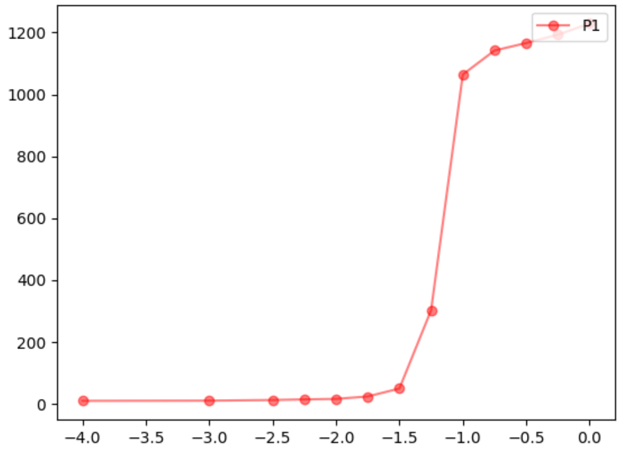
{-30: 9.71, -25: 9.64, -20: 9.65, -15: 9.59, -10: 9.65, -5: 9.84, -2: 16.29, -1.5: 44.375, -1: 1065.59, -0.5: 1165.44, -0.4: 1179.5, -0.3: 1188.14, -0.2: 1198.37, -0.1: 1205.34, 0: 1229.72}



Constant length of around ~10 before score = -5.

b. Mismatch = Indel, values: [-4,-3,-2.5,-2.25,-2,-1.75,-1.5,-1.25,-1,-0.75,-0.5,-0.25,0]

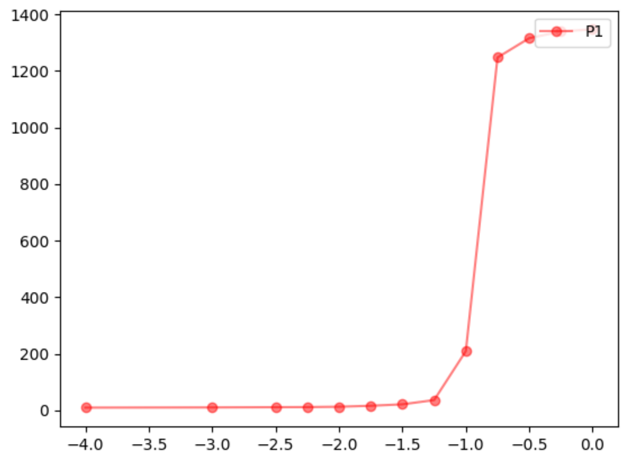
{-4: 9.895, -3: 10.495, -2.5: 12.58, -2.25: 14.625, -2: 16.245, -1.75: 24.04, -1.5: 49.705, -1.25: 302.83, -1: 1064.43, -0.75: 1141.93, -0.5: 1166.175, -0.25: 1192.935, 0: 1229.3}



The mean length of the alignments with mismatch=indel scores up to -2 is constant. The length of the alignment abruptly changes at the -2, -1 range. The length then approaches a constant value of ~1230 as score gets closer to 0.

c. Mismatch = 0, indel values: [-4,-3,-2.5,-2.25,-2,-1.75,-1.5,-1.25,-1,-0.75,-0.5,-0.25,0]

```
{-4: 9.905, -3: 10.61, -2.5: 11.39, -2.25: 11.765, -2: 13.05, -1.75: 16.495, -1.5: 21.585, -1.25: 36.51, -1:
209.51, -0.75: 1247.695, -0.5: 1316.05, -0.25: 1339.64, 0: 1347.455}
```



When setting mismatch score to 0, the jump in length is shifted to the right.

4. **File name:** locAL.py

   I started this homework when it was uploaded and managed parts 1-3 in quickly. I was stuck on this part for more than 10 days. I tried implementing the middleEdge dynamic programming algorithm before realizing it is used for global alignments. I couldn't find a resource online on alterations to this algorithms to make it work for local alignments, or think of any myself. I followed an advice from my friend Ben Hadler that heard in office hours saying we could use the two row linear space score matrix to get the end of the alignment. Then reversing the sequences sliced up to that point and aligning them to find the starting position. A good upper bound on local alignments of length r is r^2 < m+n where m,n are the lengths of the sequences. Therefore we can now normally align the sliced sequences to find the alignment while using linear space overall.

   Usage: python locAL.py <seq_file.fasta> -m [match] -s [mismatch] -d [indel] -a [alignment]

   Commands:

   - read_fasta_file(file_name) - reads a fasta sequences from a text-based file into a dictionary with sequence headers as keys.
   - locAl2(seq1, seq2, m, s, d) - slices the sequences by calling linear_local_alignment3 twice then returning the sliced strings.
   - linear_local_alignment3(seq1, seq2, m, s, d) - computes the score matrix using two rows. The next row in the matrix is calculated using the previous row. This way the memory required to store the score_matrix is now linear (2n=>O(n)).
   - outputLinearLocalAlignment(backtrack_matrix, seq1, seq2, i, j) - recursively build the path based on the backtracking algorithm. Stops when backtrack_matrix[i,j] = 'N'. Returns a tuple list of characters.
   - format_alignment - formats the output and computes the score of the alignment based on outputLocalAlignment's output.

   Input: python locAL.py p4seqs.txt -m 1 -s -10 -d -1 -a

   Output: q4_results.txt - in proper format

```
(21226, 3647)
Score:56
Length:140
TGTCCTACGC---TAGCC--G-CTAATG-GTATCGAAC--CC-TGTGTTTCCCGCGAGCAAT--CGTTAT
GAGCG---AT-GGGACCTCATCCCGGATTAGGATACTTCACGCTTACGAACTCTCAGGGAC--AGT
TCCG
TGT-CTAC-CATATA-CCTGGTC--ATGTG-A-C-AACAGCCA--TG-TT--C-CGAG---TGGC----TGAGC
GCTT-TC--GACCTCATCCCGGATTAGGATACTTCACGCTTACGAACTCTCAGGGACAGAGTTCC
G
```

5. I used Python, and Pycharm IDE. I cannot recall the total time, because it took a lot of time to run the algorithms. I tried to brainstorm ideas with Amit Klein, a friend of mine that's in the class.