Project 2: Global alignment with different gap costs

By Cæcilia, Lea & Peter

Introduction:

Our global_linear program works as expected with no problems, however with the global_affine program we get the correct optimal cost of the alignment, and we are also able to get an alignment as output, but we can only get the correct alignments on some of the test cases.
We have spent approximately 10 hours making the project, and we would not have benefitted from having more time. We have also asked Christian for help, but unfortunately we still could not solve the problem.

Methods:

Linear gap cost:

We have made one global function global_linear() with the two sequences as inputs and then a linear gap cost. Inside the function, we have made a function cost() that takes two bases as input, and then returns the score of the substitution of those two bases.

```
def global_linear(s1, s2, gap_score = -5):
    # Define substitution matrix:
    def cost(i:str, j:str)->int:
        score = np.array([[10, 2, 5, 2],
                          [2, 10, 2, 5],
                          [5, 2, 10, 2],
                          [2, 5, 2, 10]])
        t = {"A":0,
            "C":1,
            "G":2,
            "T":3}
        return score[t[i],t[j]]
```

Next we have made a function C() to fill out the score matrix. This function takes two sequences as inputs as well as the gap cost and returns the matrix. First we make an empty matrix with the dimensions of the two sequences. Then we start by filling out the first row and the first column with the gap costs. At last we find the maximum value of the diagonal cell, the upper cell and the left cell for each entry i,j and then assign that value to the cell i,j.

```
# Fill out score matrix:
def C(s1,s2, gapscore = -5):
    t_mat = np.zeros(((len(str(s1))+1),len(str(s2))+1))
    for i in range(len(t_mat)):
        t_mat[i][0] = gapscore * i
    for j in range(len(t_mat[0])):
        t_mat[0][j] = gapscore * j
    for i in range(1,len(s1)+1):
        for j in range(1,len(s2)+1):
            v1 = t_mat[i][j-1] + gapscore
            v2 = t_mat[i-1][j] + gapscore
            v3 = t_mat[i-1][j-1] + cost(s1[i-1],s2[j-1])
            t_mat[i][j] = max(v1,v2,v3)
    return t_mat
```

When we have the filled out matrix, we can finally make the alignment with the get_alignment() function that takes the same arguments as the other functions and then returns the alignment. Here we compare the current entry with the diagonal, the left and the upper entry and then make the alignment from there.

```
# Make alignment:
def get_alignment(s1, s2, gap_score):
    align1, align2 = '', ''
    score = C(s1, s2, gapscore = -5)
    i,j = len(s1),len(s2) # start from the bottom right cell
    while i > 0 and j > 0: # end toching the top or the left edge
        score_current = score[i][j]
        score_diagonal = score[i-1][j-1]
        score_up = score[i][j-1]
        score_left = score[i-1][j]
        if score_current == score_diagonal + cost(s1[i-1], s2[j-1]):
            align1 = s1[i-1] + align1
            align2 = s2[j-1] + align2
            i -= 1
            j -= 1
        elif score_current == score_left + gap_score:
            align1 = s1[i-1] + align1
            align2 = '-' + align2
            i -= 1
        elif score_current == score_up + gap_score:
            align1 = '-' + align1
            align2 = s2[j-1] + align2
            j -= 1
    # Finish tracing up to the top left cell
    while i > 0:
        align1 = s1[i-1] + align1
        align2 = '-' + align2
        i -= 1
    while j > 0:
        align1 = '-' + align1
        align2 = s2[j-1] + align2
        j -= 1

    return align1 + "\n" + align2 + "\n" + 'Optimal cost: {}'.format(score[-1][-1])
return get_alignment(s1, s2, gap_score)
```

Affine cap cost:

Here we also have made one global function with the two sequences as inputs. Inside the function, we define the substitution matrix in the same way as with global_linear().

```
def global_affine(s1, s2):
    # Define score matrix:
    def cost(i:str, j:str)->int:
        score = np.array([[0, 5, 2, 5],
                          [5, 0, 5, 2],
                          [2, 5, 0, 5],
                          [5, 2, 5, 0]])
        t = {"A":0,
             "C":1,
             "G":2,
             "T":3}
        return score[t[i],t[j]]
```

When it comes to filling the matrix out, we start by defining three matrices (S, I and D). Then we fill out the I and D with the gap penalties. Then for each of the matrices we define the last entry as the minimum value.

```
# Fill out score matrix:
def affine_align(s1, s2, gap_start = 5, gap_extend = 5):
    S = np.zeros(((len(str(s1))+1),len(str(s2))+1))
    I = np.zeros(((len(str(s1))+1),len(str(s2))+1))
    D = np.zeros(((len(str(s1))+1),len(str(s2))+1))
    for i in range(1, len(s1)+1):
        S[i][0] = float('inf')
        I[i][0] = float('inf')
        D[i][0] = gap_start + (i * gap_extend)
    for i in range(1, len(s2)+1):
        S[0][i] = float('inf')
        I[0][i] = gap_start + (i * gap_extend)
        D[0][i] = float('inf')
    for i in range(1, len(s1)+1):
        for j in range(1, len(s2)+1):
            S[i][j] = cost(s1[i-1], s2[j-1]) + min(
                S[i-1][j-1],
                I[i-1][j-1],
                D[i-1][j-1])
            I[i][j] = min(
                gap_start + gap_extend + S[i-1][j],
                gap_extend + I[i-1][j],
                gap_start + gap_extend + D[i-1][j])
            D[i][j] = min(
                gap_start + gap_extend + S[i][j-1],
                gap_start + gap_extend + I[i][j-1],
                gap_extend + D[i][j-1])
    return S
```

Now we can make the alignment by backtracking. Here we start by checking if the end block is a substitution or a gap. Then if there is a gap, we want to find out how long the gap block is, which we do by updating k.

```
# Make alignment:
def backtrace(s1, s2, gap_start = 5, gap_extend = 5):
    S = affine_align(s1, s2, gap_start=5, gap_extend=5)
    align1, align2 = [], []
    i, j = len(s1), len(s2)
    while i > 0 or j > 0:
        if i > 0 and j > 0 and S[i][j] == S[i-1][j-1] + cost(s1[i-1], s2[j-1]):
            align1.append(s1[i-1])
            align2.append(s2[j-1])
            i -= 1
            j -= 1
        else:
            k = 1
            while True:
                if i >= k and S[i][j] == S[i-k][j] + (gap_start + gap_extend * k):
                    align1.append(s1[i-k:i])
                    align2.append('-' * k)
                    i = i-k
                    break
                elif j >= k and S[i][j] == S[i][j-k] + (gap_start + gap_extend * k):
                    align1.append('-' * k)
                    align2.append(s2[j-k:j])
                    j = j-k
                    break
                else:
                    k = k + 1
    return "".join(reversed(align1)) + "\n" + "".join(reversed(align2))
return backtrace(s1, s2)
```

Test:

We have tested the program using the test cases from project2_examples.txt, because here the alignments and optimal cost was known. Otherwise one could have tested the alignment algorithm by comparing it with the ones used by Biopython or MegaX.

The global_linear program performs as expected on the test cases presented in project2_examples.txt. The global_affine program does not perform as expected on the test cases presented in project2_examples.txt

Answers to the questions in project2_eval.txt:

1. Compute the score of an optimal alignment and an optimal alignment of seq1 and seq2 above using the program global_linear using the above score matrix M and gap cost g(k)=5*k

The optimal cost is 332.0 and optimal alignment is:

seq1:
TA--TG-GA-GAGAATAAAAGAACTGAGAGA--TCTAATGTCGCAGTCC-C-GCACTCGCGAGATACTCACTAAGACC
ACTGTGGACCAT-ATGGCCATAATCAA-AAA-G
seq2:
ATGGATGTCAATCCGACTCTACT--TTTCCTAAAAATTCCAGCGCAAAATGCCATAAGCA--CCAC--AT--TCCC--TTA
TACTGGAGATCCTCCATA-CAGCCATGGAA

2. Compute the score of an optimal alignment and an optimal alignment of seq1 and seq2 above using the program global_affine using the above score matrix M and gap cost g(k)=5+5*k

The optimal cost is 266.0.
Our algorithm was not able to give us the optimal alignment of these sequences.

3. Compute the optimal score of an optimal alignment for each pair of the 5 sequences above using global_linear with the score matrix M and gap cost g(k)=5*k. The result is a 5x5 table where entry (i,j) is the optimal score of an alignment of seqi and seqj.

We did not have time to solve this problem.

4. Compute the optimal score of an optimal alignment for each pair of the 5 sequences above using global_affine with the score matrix M and gap cost g(k)=5+5*k. The result is a 5x5 table where entry (i,j) is the optimal score of an alignment of seqi and seqj.

We did not have time to solve this problem.

Experiments:

We were not able to make the plots for this part of the project.