Project 3: Multiple alignment

- *By Cæcilia, Lea & Peter*

Introduction:

Everything works as expected.

Methods:

Approximation alignment program:

- The input of the alignment program is a list with sequences to be aligned.
- We then use a function find_center() to find the center string and place that at the top of the alignment.
- Then we just follow the four cases and define the new column to be added to our MSA if the case is true.
- At the end we delete the first column in MSA, which is only zeroes, and we make a new MSA_extend ready.

```
def approx_alignment(seq_list):
    seq_list[0],seq_list[find_center()] = seq_list[find_center()],seq_list[0]
    MSA = alignment_pairwise(seq_list[0],seq_list[1])
    MSA_extend = np.zeros([3,1])
    for l in seq_list[2:]:
        A = alignment_pairwise(seq_list[0],l)
        i = 0
        j = 0
        while i < MSA.shape[1] and j < A.shape[1]:
            if MSA[0][i] == '-' and A[0][j] == '-':
                new_col = []
                for l in range(0,len(MSA[:,i])):
                    new_col.append(MSA[:,i][l:l+1])
                new_col.append([A[1][j]])
                MSA_extend = np.append(MSA_extend, new_col, axis = 1)
                i+=1
                j+=1
            if MSA[0][i] == '-' and A[0][j] != '-':
                new_col = []
                for l in range(0,len(MSA[:,i])):
                    new_col.append(MSA[:,i][l:l+1])
                new_col.append(['-'])
                MSA_extend = np.append(MSA_extend, new_col,axis = 1)
                i+=1
            if MSA[0][i] != '-' and A[0][j] == '-':
                new_col = [['-'] for _ in range(np.shape(MSA)[0])]
                new_col.append([A[1][j]])
                MSA_extend = np.append(MSA_extend, new_col,axis = 1)
                j+=1
            if MSA[0][i] != '-' and A[0][j] != '-':
                new_col = []
                for l in range(0,len(MSA[:,i])):
                    new_col.append(MSA[:,i][l:l+1])
                new_col.append([A[1][j]])
                MSA_extend = np.append(MSA_extend, new_col, axis = 1)
                i+=1
                j+=1
        MSA_extend = np.delete(MSA_extend,0,1)
        MSA = MSA_extend
        MSA_extend = np.zeros([len(MSA_extend)+1,1])
    x = []
    for m in range(len(MSA)):
        x.append(''.join(MSA[m,:].tolist()))
    return x
```

Approximation score program:

- To get the score of an optimal alignment, we use the output from our approx_alignment program, which is a list of the alignments.
- We then compute the score based on the code from Christian.

```
def approx_score(list_with_alignments):
    score = 0
    for i in range(len(list_with_alignments)):
            for j in range(i+1,len(list_with_alignments)):
                for c in range(len(list_with_alignments[i])):
                    score = score + cost(list_with_alignments[i][c],list_with_alignments[j][c])
    return score
```

Exact alignment program:

- The input of the alignment program is the three sequences as well as a gapcost.
- We then create three empty lists in which we want to put our alignments.
- Then we check for the different cases and make the alignments.

```
def alignment(s1, s2, s3, gapcost = 5):
    mat = view_matrix(s1, s2, s3)
    align_seq_1 = []
    align_seq_2 = []
    align_seq_3 = []
    i = len(s1)
    j = len(s2)
    k = len(s3)
    while i > 0 or j > 0 or k > 0:
        if mat[i][j][k] == mat[i-1][j-1][k-1] + cost(s1[i-1],s2[j-1]) + cost(s1[i-1],s3[k-1]) + cost(s2[j-1],s3[k-1]):
            align_seq_1.append(s1[i-1])
            align_seq_2.append(s2[j-1])
            align_seq_3.append(s3[k-1])
            i -= 1
            j -= 1
            k -= 1
        if mat[i][j][k] == mat[i-1][j-1][k] + cost(s1[i-1],s2[j-1]) + 2*gapcost:
            align_seq_1.append(s1[i-1])
            align_seq_2.append(s2[j-1])
            align_seq_3.append('-')
            i-=1
            j-=1
        if mat[i][j][k] == mat[i-1][j][k-1] + cost(s1[i-1],s3[k-1]) + 2*gapcost:
            align_seq_1.append(s1[i-1])
            align_seq_2.append('-')
            align_seq_3.append(s3[k-1])
            i-=1
            k-=1
        if mat[i][j][k] == mat[i][j-1][k-1] + cost(s2[j-1],s3[k-1]) + 2*gapcost:
            align_seq_1.append('-')
            align_seq_2.append(s2[j-1])
            align_seq_3.append(s3[k-1])
            j-=1
            k-=1
        if mat[i][j][k] == mat[i-1][j][k] + 2*gapcost:
            align_seq_1.append(s1[i-1])
            align_seq_2.append('-')
            align_seq_3.append('-')
            i-=1
        if mat[i][j][k] == mat[i][j-1][k] + 2*gapcost:
            align_seq_1.append('-')
            align_seq_2.append(s2[j-1])
            align_seq_3.append('-')
            j-=1
        if mat[i][j][k] == mat[i][j][k-1] + 2*gapcost:
            align_seq_1.append('-')
            align_seq_2.append('-')
            align_seq_3.append(s3[k-1])
            k-=1
    return ''.join(align_seq_3[::-1]) + '\n' + ''.join(align_seq_2[::-1]) + '\n' + ''.join(align_seq_1[::-1])
```

Exact score program:

- We have the following inputs: empty score matrix, lengths of the three sequences, the three sequences and the gap cost.
- We then check for the eight different cases and choose the minimum.
- The optimal score will then be the last cell in the matrix.

```
def fill_matrix(mat,i,j,k,s1,s2,s3,gapcost = 5):
    if mat[i][j][k] == np.inf:
        v0, v1, v2, v3, v4, v5, v6, v7 = np.inf,np.inf,np.inf,np.inf, np.inf, np.inf, np.inf, np.inf
        if i > 0 and j > 0 and k > 0:
            v1 = fill_matrix(mat,i-1,j-1,k-1,s1,s2,s3) + cost(s1[i-1],s2[j-1]) + cost(s1[i-1],s3[k-1]) + cost(s2[j-1],s3[k-1])
        if i > 0 and j > 0 and k >= 0:
            v2 = fill_matrix(mat,i-1,j-1,k,s1,s2,s3) + cost(s1[i-1],s2[j-1]) + 2*gapcost
        if i > 0 and j >= 0 and k > 0:
            v3 = fill_matrix(mat,i-1,j,k-1,s1,s2,s3) + cost(s1[i-1],s3[k-1]) + 2*gapcost
        if i >= 0 and j > 0 and k > 0:
            v4 = fill_matrix(mat,i,j-1,k-1,s1,s2,s3) + cost(s2[j-1],s3[k-1]) + 2*gapcost
        if i > 0 and j >= 0 and k >= 0:
            v5 = fill_matrix(mat,i-1,j,k,s1,s2,s3) + 2*gapcost
        if i >= 0 and j > 0 and k >= 0:
            v6 = fill_matrix(mat,i,j-1,k,s1,s2,s3) + 2*gapcost
        if i >= 0 and j >= 0 and k > 0:
            v7 = fill_matrix(mat,i,j,k-1,s1,s2,s3) + 2*gapcost
        if i == 0 and j == 0 and k == 0:
            v0 = 0
        mat[i][j][k] = min(v0,v1,v2,v3,v4,v5,v6,v7)
        print(mat[i][j][k])
    return int(mat[i][j][k])
```

Experiments:

What is the score of an optimal alignment of the first 3 sequences in brca1-testseqs.fasta (i.e. brca1_bos_taurus, brca1_canis_lupus and brca1_gallus_gallus) as computed by your program sp_exact_3? What does an optimal alignment look like?

The score of the optimal alignment is 790. The optimal alignment looks like this:

brca1_gallus_gallus:
GCGAAAT-GTA-AC--A-CG-GTAGAGGTGAT-CGGGGTG-CGTTA-TAC-GTGCGTGGTGACCTCGGTCGGTGTT-GA
CGGTGCCTGGGGTTCCTCAGAGTGTTTTGGGGTCTGAAGGATG-GACTTGTCAGTGATT-GCCA-TTGGAGACGT
GCAAAATGTGCTTTCAGCCATGCAGAA-GAAC-TTGG-AGTGTCCAGTCTGTTTAGATGTGAT

brca1_canis_lupus:
ATGGATTTATCTGCGGATCGTGTTGAAGAAGTACAAAATGTTCTTAATGCTATGCA-GAAAATCTTAG--AGTGTCC
AATA-TGTCTGGAGTTGATCAAAGAGCCT-GTT-TCTACAAAGTGTGATC-A-CA-TATTTTGCAAATTTTGTATGC-TG
AAAC-T--TCTCAACCA-GAGGAAGGGGCCTTCACAGTGTCC--TTTGTGTAAGAACGA-

brca1_bos_taurus:
ATGGATTTATCTGCGGATCATGTTGAAGAAGTACAAAATGTCCTCAATGCTATGCA-GAAAATCTTAG--AGTGTCC
AATA-TGTCTGGAGTTGATCAAAGAGCCT-GTC-TCTACAAAGTGTGACC-A-CA-TATTTTGCAAATTTTGTATGC-T
GAAAC-T--TCTCAACCA-GAAGAAAGGGCCTTCACAATGTCC--TTTGTGTAAGAATGA-

What is the score of the alignment of the first 5 sequences in brca1-testseqs.fasta(i.e. brca1_bos_taurus, brca1_canis_lupus, brca1_gallus_gallus, brca1_homo_sapiens, and brca1_macaca_mulatta) as computed by your program sp_approx? Which of the 5 sequences is chosen as the 'center string'?

The score of the alignment is 3295, and the center string chosen was the first sequence.

Make an experiment comparing the scores of the alignments computed by sp_exact_3 and sp_approx that validates that the approximation ratio of sp_approx is 2(k-1)/k for k sequences. i.e 4/3 for three sequences.

You should use the test data in testseqs.zip that contains 20 fasta files (testseqs_10_3.fasta, testseqs_20_3.fasta, ..., testseqs_200_3.fasta) each containing 3 sequences of lengths 10, 20, ..., 200. For each triplet of sequences (i.e. each fasta file), you should compute the optimal score of an MSA using sp_exact_3 and the score of the alignment produced by sp_approx. Make a graph in which you plot the ratio of the computed scores for each sequence length. Comment on what you observe.

As we can see in the plot below, we observe a mean ratio between the approximation score and the exact score of 1.106. We can also see that the ratio is below 4/3 in all cases.