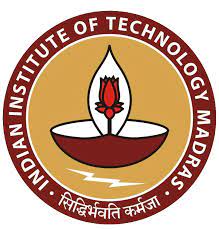
..

**BT 3040: Bioinformatics**

**Assignment 4**



Indian Institute of Technology Madras

Atharva Mandar Phatak | BE21B009

Department of Biotechnology

## **Q1) Write a program to construct a dot plot for the alignment of human and chicken hemoglobin β chain. Identify the segments, which are same in both sequences.**

# Atharva Mandar Phatak | BE21B009 | BT3040 Assignment 4

#Q1

import matplotlib.pyplot as plt

def dot\_plot(seq1, seq2):

    len1 = len(seq1)

    len2 = len(seq2)

    dot\_plot = [[0] \* len2 for \_ in range(len1)]

    # Fill the dot plot

    for i in range(len1):

        for j in range(len2):

            if seq1[i] == seq2[j]:

                dot\_plot[i][j] = 1

    # Create the plot

    plt.imshow(dot\_plot,cmap='Accent', interpolation='nearest')

    plt.xlabel('Chicken Hb')

    plt.ylabel('Human Hb')

    plt.title('Dot Plot for Alignment of Human and Chicken Hemoglobin β Chains')

    plt.xticks(range(len2), seq2)

    plt.yticks(range(len1), seq1)

    plt.gca().invert\_yaxis()

    plt.show()

def similar\_seq(seq1, seq2):

    posn\_vals= []

    for i in range(len(seq1)):

        if seq1[i] == seq2[i]:

            posn\_vals.append(i+1)

    return posn\_vals

#Given Sequences

human\_hb = "MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH"

chicken\_hb = "MVHWTAEEKQLITGLWGKVNVAECGAEALARLLIVYPWTQRFFASFGNLSSPTAILGNPMVRAHGKKVLTSFGDAVKNLDNIKNTFSQLSELHCDKLHVDPENFRLLGDILIIVLAAHFSKDFTPECQAAWQKLVRVVAHALARKYH"

dot\_plot(human\_hb, chicken\_hb)

segments\_indx = similar\_seq(human\_hb, chicken\_hb)

#Align Output Sequence

def align\_seq\_output(segments):

    segs=segments

    seq=[]

    counter=0

    templst=[]

    while(counter<(len(segs)-1)):

        if counter==0:

            templst.append(segments[counter])

        if segs[counter+1]==(segs[counter]+1):

            templst.append(segments[counter+1])

        else:

            seq.append(templst)

            templst=[]

            templst.append(segments[counter+1])

        counter=counter+1

    seq\_len=[len(i) for i in seq ]

    alignstr=""

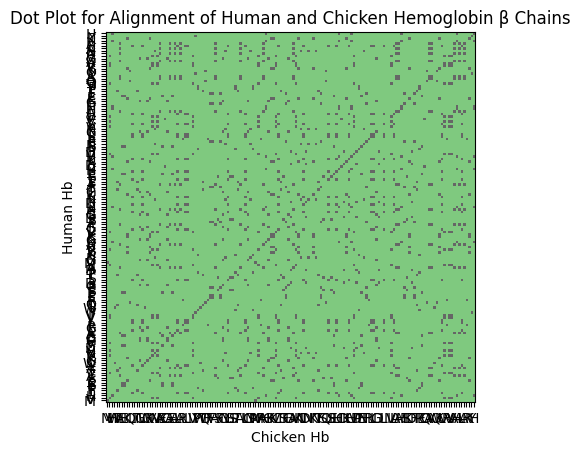
    for i in seq[seq\_len.index(max(seq\_len))]:

        alignstr=alignstr+str(human\_hb[i-1])

    print("Maximum matching sequence:", alignstr)

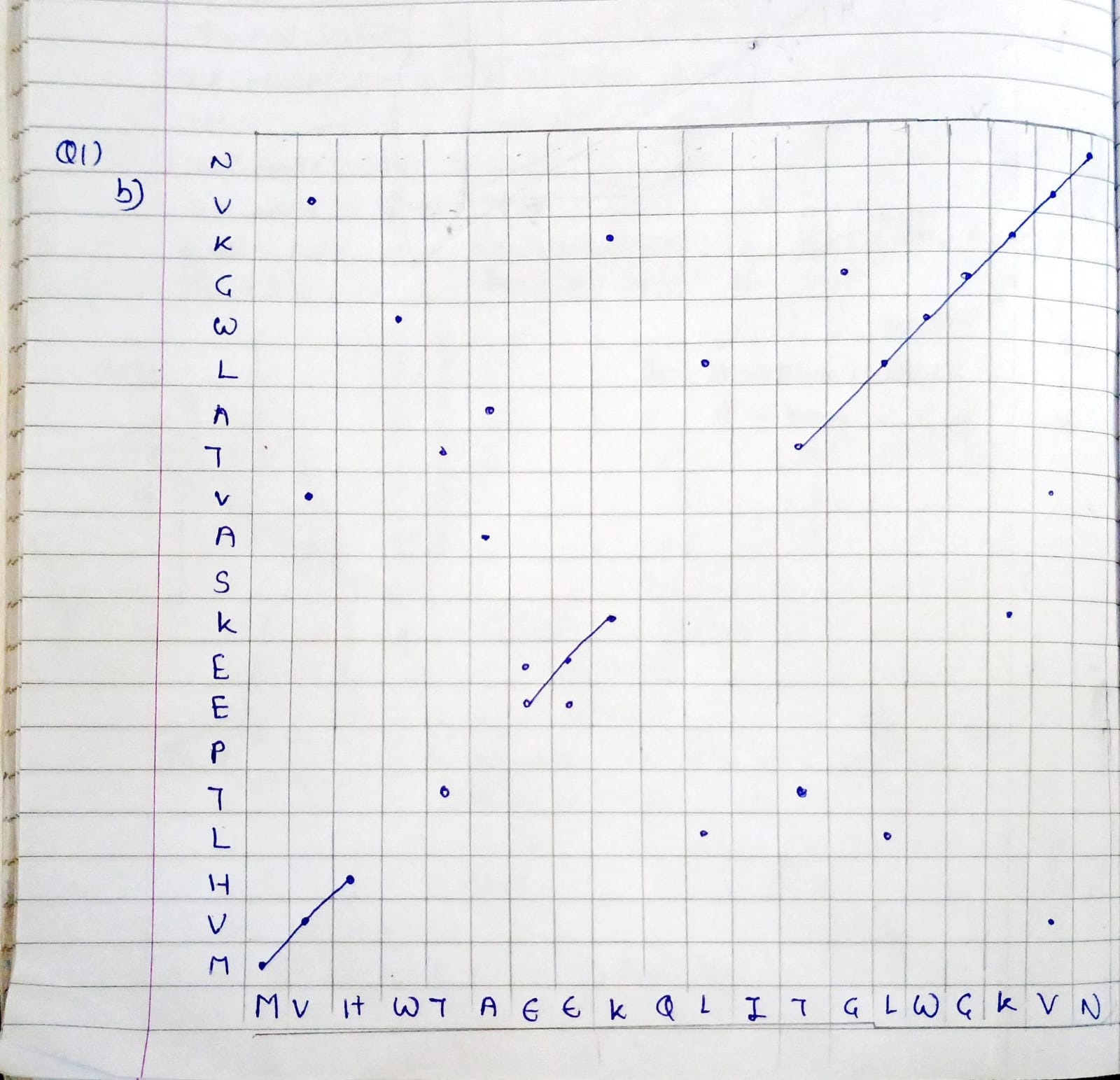
align\_seq\_output(segments\_indx)

Dot Plot (Output):



Maximum matching sequence: LSELHCDKLHVDPENFRLLG

### Manual Dot Plot





Longest Matching Sequence: WGKVN…

## **Q2) Calculate the score for the following alignments using code: AATCTATAAAG—ATA. Assume that the match score is 1, mismatch score is 0, origination penalty is -2, and length penalty is -1**

# Atharva Mandar Phatak | BE21B009 | BT3040 Assignment 4

#Q2

inp\_seq1='AATCTATA'

inp\_seq2='AAG--ATA'

#Finding the total score (excluding penalty)

def score\_find(seq1,seq2):

    score=0

    for i in range(len(seq1)):

        if(seq1[i]==seq2[i]):

            score=score+1

        elif(seq1[i]!=seq2[i]):

            score=score+0

    return score

F#Finding the penalty, and later subtracting from the total score

def penalty\_find(seq2):

    count\_arr=[]

    counter=0

    for i in range(len(seq2)):

        if(seq2[i]=='-'):

            counter+=1

        else:

            count\_arr.append(counter)

            counter=0

    tot\_penalty\_vals=[]

    for i in count\_arr:

        if(i!=0):

            tot\_penalty\_vals.append(i\*(-1)-2)

    return(sum(tot\_penalty\_vals))

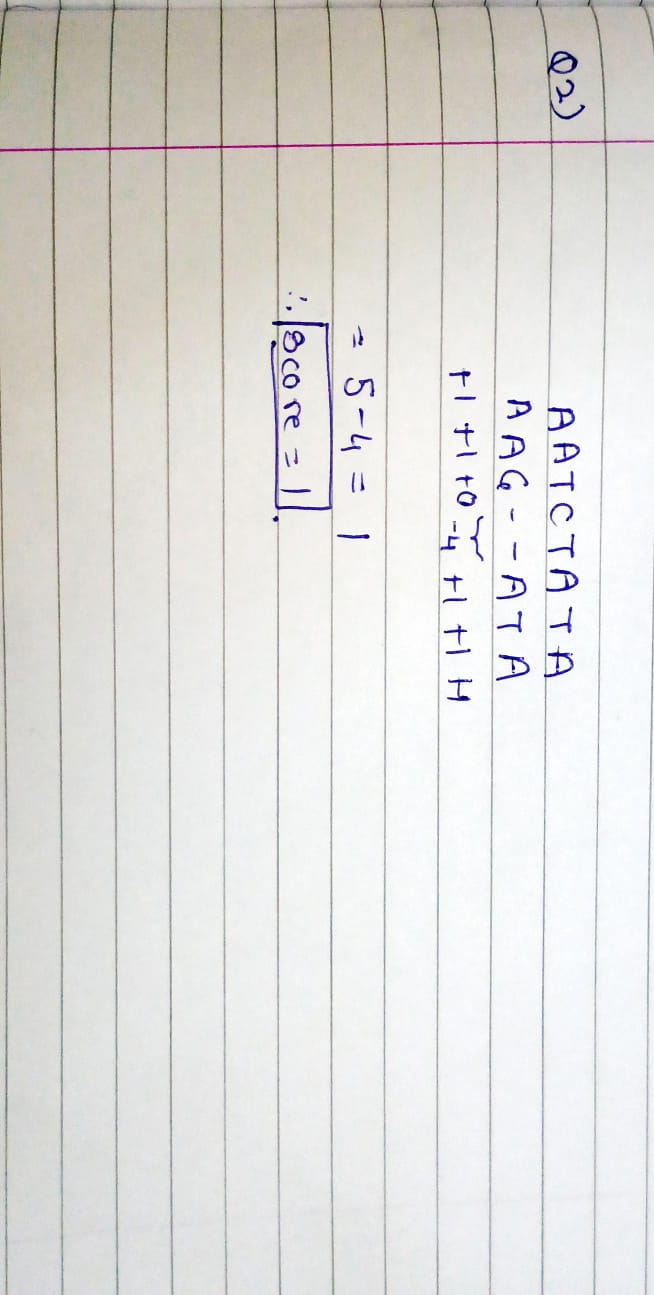
final\_score= score\_find(inp\_seq1,inp\_seq2)+penalty\_find(inp\_seq2)

print("Final Score =",final\_score)

Output:

l core = 1

## **Q3)** **Verify the Q2 manually**



## **Q4) Using the Needleman and Wunsch dynamic programming method, construct the partial alignment score table and align the following two sequences (using code): ACAGTCGAACG and ACCGTCCG use the scoring parameters: match score: +2; mismatch score: -1 and gap penalty: -2**

# Atharva Mandar Phatak | BE21B009 | BT3040 Assignment 4

#Q4

def needleman\_wunsch(seq1, seq2, match\_score=+2, mismatch\_penalty=-1, gap\_penalty=-2):

    # Initialize the score matrix

    score\_matrix = [[0] \* (len(seq2) + 1) for \_ in range(len(seq1) + 1)]

    # Initialize the traceback matrix to store directions

    traceback\_matrix = [[0] \* (len(seq2) + 1) for \_ in range(len(seq1) + 1)]

    # Initialize the first row and column of the score matrix

    for i in range(1, len(seq1) + 1):

        score\_matrix[i][0] = gap\_penalty \* i

        traceback\_matrix[i][0] = 'U'  # 'U' indicates up

    for j in range(1, len(seq2) + 1):

        score\_matrix[0][j] = gap\_penalty \* j

        traceback\_matrix[0][j] = 'L'  # 'L' indicates left

    # Fill the score and traceback matrices

    for i in range(1, len(seq1) + 1):

        for j in range(1, len(seq2) + 1):

            match = score\_matrix[i - 1][j - 1] + (match\_score if seq1[i - 1] == seq2[j - 1] else mismatch\_penalty)

            delete = score\_matrix[i - 1][j] + gap\_penalty

            insert = score\_matrix[i][j - 1] + gap\_penalty

            score\_matrix[i][j] = max(match, delete, insert)

            # Update traceback matrix

            if score\_matrix[i][j] == match:

                traceback\_matrix[i][j] = 'D'  # 'D' indicates diagonal

            elif score\_matrix[i][j] == delete:

                traceback\_matrix[i][j] = 'U'  # 'U' indicates up

            else:

                traceback\_matrix[i][j] = 'L'  # 'L' indicates left

    # Traceback to find the alignment

    alignment\_seq1 = ''

    alignment\_seq2 = ''

    i, j = len(seq1), len(seq2)

    while i > 0 or j > 0:

        if traceback\_matrix[i][j] == 'D':

            alignment\_seq1 = seq1[i - 1] + alignment\_seq1

            alignment\_seq2 = seq2[j - 1] + alignment\_seq2

            i -= 1

            j -= 1

        elif traceback\_matrix[i][j] == 'U':

            alignment\_seq1 = seq1[i - 1] + alignment\_seq1

            alignment\_seq2 = '-' + alignment\_seq2

            i -= 1

        else:

            alignment\_seq1 = '-' + alignment\_seq1

            alignment\_seq2 = seq2[j - 1] + alignment\_seq2

            j -= 1

    return score\_matrix, alignment\_seq1, alignment\_seq2

# Example sequences

sequence1 = "ACAGTCGAACG"

sequence2 = "ACCGTCCG"

# Call the needleman\_wunsch function

score\_table, aligned\_seq1, aligned\_seq2 = needleman\_wunsch(sequence1, sequence2)

# Print the alignment and the partial alignment score table

print("Sequence 1:", aligned\_seq1)

print("Sequence 2:", aligned\_seq2)

print("\nPartial Alignment Score Table:")

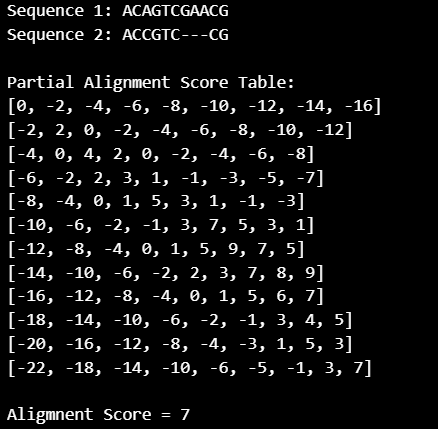
for row in score\_table:

    print(row)

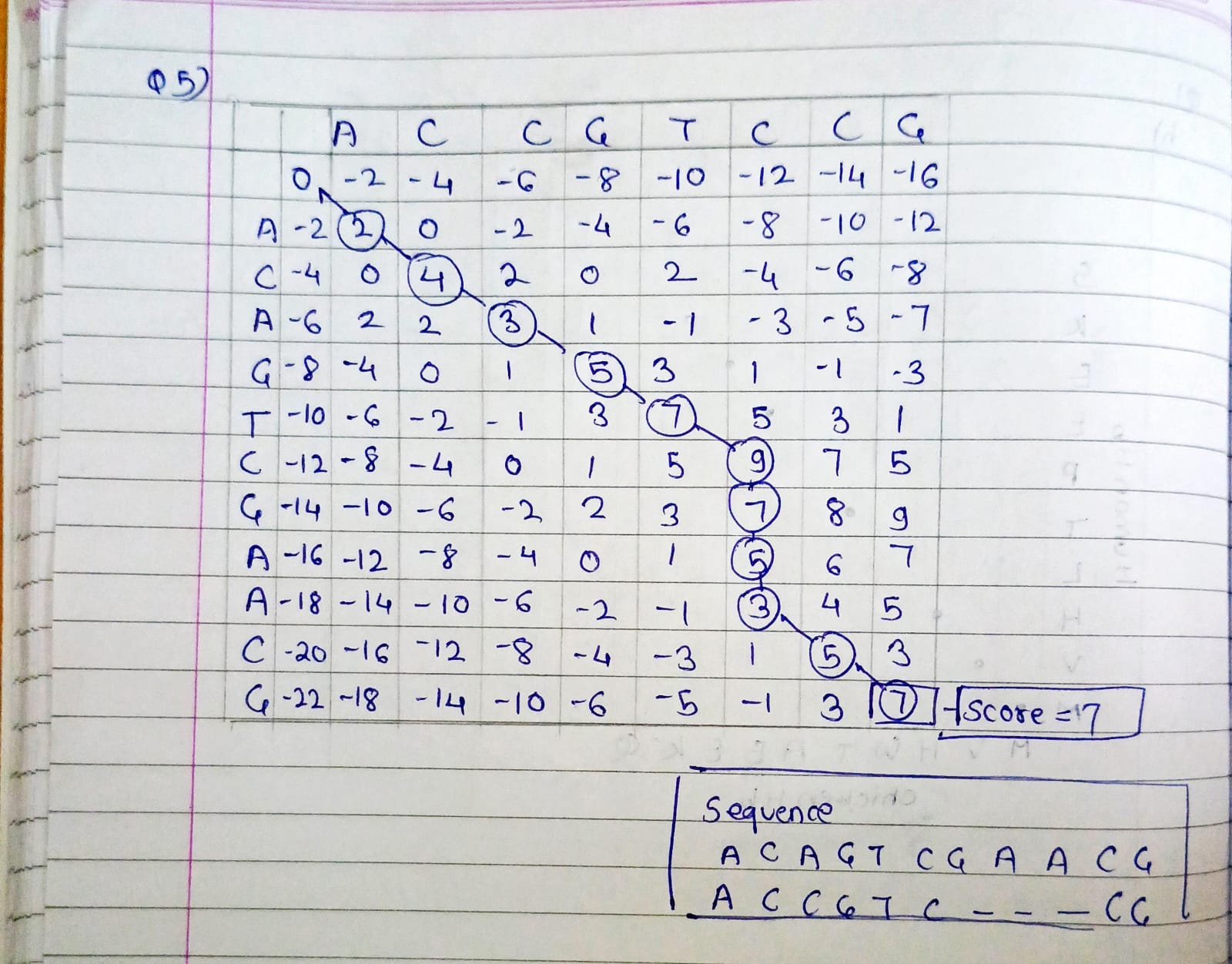
print(" ")

print(f"Aligmnent Score={max(score\_table[-1])}")

Output



## **Q5) Verify Q4 manually**



**Q6) Using the Smith-Waterman method, construct the partial alignment scoring table and align the following two sequences (using code): ACGTATCGCGTATA and GATGCGTATCG scoring parameters: match score: +2; mismatch score: -1 and gap penalty: -2**

# Atharva Mandar Phatak | BE21B009 | BT3040 Assignment 4

#Q6

def smith\_waterman(seq1, seq2, match=2, mismatch=-1, gap=-2):

    # Initialize the scoring matrix

    rows = len(seq1) + 1

    cols = len(seq2) + 1

    score\_matrix = [[0] \* cols for \_ in range(rows)]

    # Fill the scoring matrix

    for i in range(1, rows):

        for j in range(1, cols):

            if seq1[i - 1] == seq2[j - 1]:

                diag\_score = score\_matrix[i - 1][j - 1] + match

            else:

                diag\_score = score\_matrix[i - 1][j - 1] + mismatch

            up\_score = score\_matrix[i - 1][j] + gap

            left\_score = score\_matrix[i][j - 1] + gap

            score\_matrix[i][j] = max(0, diag\_score, up\_score, left\_score)

    # Find the maximum score in the matrix

    max\_score = 0

    max\_i, max\_j = 0, 0

    for i in range(rows):

        for j in range(cols):

            if score\_matrix[i][j] > max\_score:

                max\_score = score\_matrix[i][j]

                max\_i, max\_j = i, j

    # Traceback to find the alignment

    alignment\_seq1 = ""

    alignment\_seq2 = ""

    i, j = max\_i, max\_j

    while score\_matrix[i][j] != 0:

        if score\_matrix[i][j] == score\_matrix[i - 1][j - 1] + (match if seq1[i - 1] == seq2[j - 1] else mismatch):

            alignment\_seq1 = seq1[i - 1] + alignment\_seq1

            alignment\_seq2 = seq2[j - 1] + alignment\_seq2

            i -= 1

            j -= 1

        elif score\_matrix[i][j] == score\_matrix[i - 1][j] + gap:

            alignment\_seq1 = seq1[i - 1] + alignment\_seq1

            alignment\_seq2 = '-' + alignment\_seq2

            i -= 1

        else:

            alignment\_seq1 = '-' + alignment\_seq1

            alignment\_seq2 = seq2[j - 1] + alignment\_seq2

            j -= 1

    return alignment\_seq1, alignment\_seq2, max\_score, score\_matrix

def print\_matrix(matrix):

    for row in matrix:

        print(row)

def main():

    seq1 = "ACGTATCGCGTATA"

    seq2 = "GATGCGTATCG"

    alignment\_seq1, alignment\_seq2, max\_score, score\_matrix = smith\_waterman(seq1, seq2)

    print("Alignment Sequence 1:", alignment\_seq1)

    print("Alignment Sequence 2:", alignment\_seq2)

    print("Alignment Score:", max\_score)

    print("\nScoring Matrix:")

    print\_matrix(score\_matrix)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output:

