CSE 486

Bioinformatics

Assignment

Ahmed Amr Elgayar
19p8349

Table of Contents

Code link: https://github.com/aGayar30/BioInformatics/tree/master/Assignment

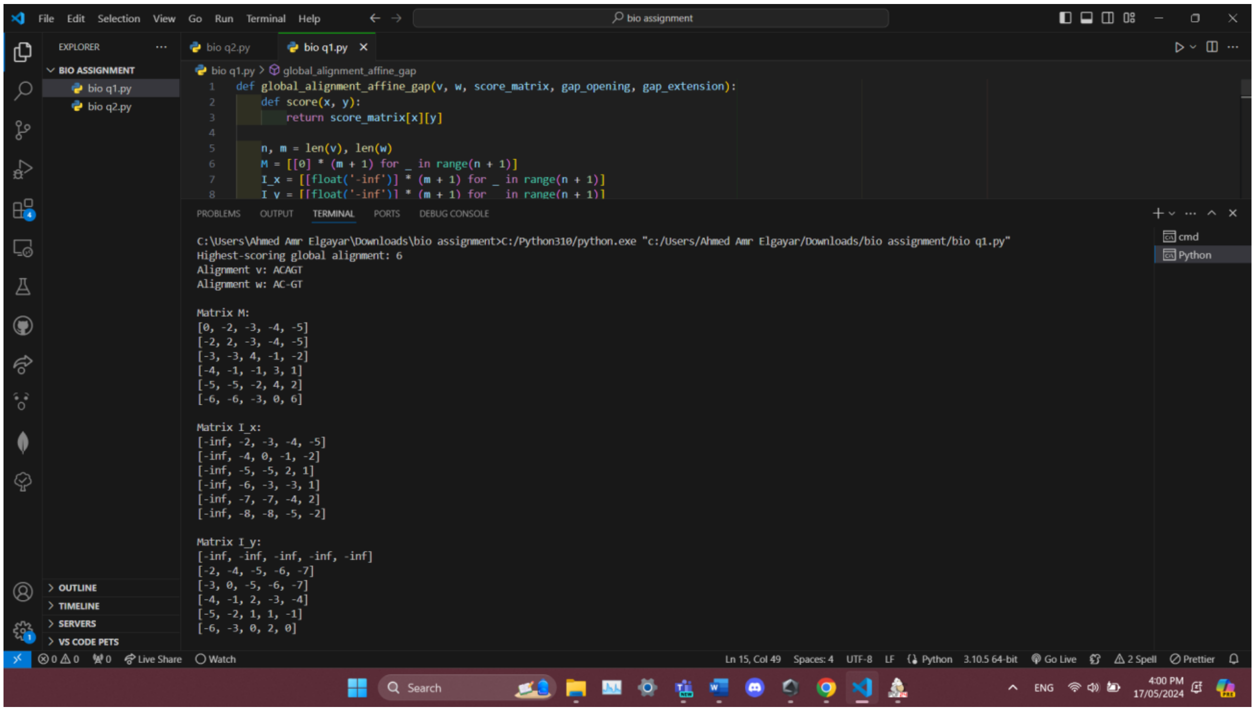
Question 1

Alignment with affine gap penalties problem: Construct a highest- scoring global alignment between two strings (with affine gap penalties). The inputs are Two strings v and w, a scoring matrix Score. The output is highest-scoring global alignment between these strings, as defined by the scoring matrix Score and by the gap opening and extension penalties. Implement it using three matrices.

Code

```
def global_alignment_affine_gap(v, w, score_matrix, gap_opening, gap_extension):
    def score(x, y):
        return score_matrix[x][y]

    n, m = len(v), len(w)
    M = [[0] * (m + 1) for _ in range(n + 1)]
    I_x = [[float('-inf')] * (m + 1) for _ in range(n + 1)]
    I_y = [[float('-inf')] * (m + 1) for _ in range(n + 1)]
    traceback_matrix = [[None] * (m + 1) for _ in range(n + 1)]

    # Initialization
    for i in range(1, n + 1):
        I_y[i][0] = gap_opening + (i - 1) * gap_extension
        M[i][0] = I_y[i][0]
        traceback_matrix[i][0] = '↑'  # Gap in w
    for j in range(1, m + 1):
        I_x[0][j] = gap_opening + (j - 1) * gap_extension
        M[0][j] = I_x[0][j]
        traceback_matrix[0][j] = '←'  # Gap in v

    # Fill DP tables
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            M[i][j] = max(M[i-1][j-1], I_x[i-1][j-1], I_y[i-1][j-1]) + score(v[i-1], w[j-1])
            I_x[i][j] = max(I_x[i][j-1] + gap_extension, M[i][j-1] + gap_opening)
            I_y[i][j] = max(I_y[i-1][j] + gap_extension, M[i-1][j] + gap_opening)

            if M[i][j] >= I_x[i][j] and M[i][j] >= I_y[i][j]:
                traceback_matrix[i][j] = '↖'  # Match/mismatch
            elif I_x[i][j] > I_y[i][j]:
                traceback_matrix[i][j] = '←'  # Gap in v
            else:
                traceback_matrix[i][j] = '↑'  # Gap in w

    # Traceback
    alignment_v, alignment_w = '', ''
    i, j = n, m
    while i > 0 or j > 0:
        if traceback_matrix[i][j] == '↖':
            alignment_v = v[i-1] + alignment_v
            alignment_w = w[j-1] + alignment_w
            i -= 1
            j -= 1
        elif traceback_matrix[i][j] == '←':
            alignment_v = '-' + alignment_v
            alignment_w = w[j-1] + alignment_w
            j -= 1
        elif traceback_matrix[i][j] == '↑':
            alignment_v = v[i-1] + alignment_v
            alignment_w = '-' + alignment_w
            i -= 1

    return M, I_x, I_y, max(M[n][m], I_x[n][m], I_y[n][m]), alignment_v, alignment_w

# Example usage
v = "ACAGT"
w = "ACGT"
score_matrix = {
    'A': {'A': 2, 'C': -1, 'G': -1, 'T': -1},
    'C': {'A': -1, 'C': 2, 'G': -1, 'T': -1},
    'G': {'A': -1, 'C': -1, 'G': 2, 'T': -1},
    'T': {'A': -1, 'C': -1, 'G': -1, 'T': 2}
}
gap_opening = -2
gap_extension = -1

M, I_x, I_y, result, alignment_v, alignment_w = global_alignment_affine_gap(v, w, score_matrix,
gap_opening, gap_extension)

print("Highest-scoring global alignment:", result)
print("Alignment v:", alignment_v)
print("Alignment w:", alignment_w)
print("\nMatrix M:")
for row in M:
    print(row)
print("\nMatrix I_x:")
for row in I_x:
    print(row)
print("\nMatrix I_y:")
for row in I_y:
    print(row)
```

Output Screenshot



```
def global_alignment_affine_gap(v, w, score_matrix, gap_opening, gap_extension):
    def score(x, y):
        return score_matrix[x][y]

    n, m = len(v), len(w)
    M = [[0] * (m + 1) for _ in range(n + 1)]
    I_x = [[float('-inf')] * (m + 1) for _ in range(n + 1)]
    I_y = [[float('-inf')] * (m + 1) for _ in range(n + 1)]
```

```
C:\Users\Ahmed Amr Elgayar\Downloads\bio assignment>C:/Python310/python.exe "c:/Users/Ahmed Amr Elgayar/Downloads/bio assignment/bio q1.py"
Highest-scoring global alignment: 6
Alignment v: ACAGT
Alignment w: AC-GT

Matrix M:
[0, -2, -3, -4, -5]
[-2, 2, -3, -4, -5]
[-3, -3, 4, -1, -2]
[-4, -1, -1, 3, 1]
[-5, -5, -2, 4, 2]
[-6, -6, -3, 0, 6]

Matrix I_x:
[-inf, -2, -3, -4, -5]
[-inf, -4, 0, -1, -2]
[-inf, -5, -5, 2, 1]
[-inf, -6, -3, -3, 1]
[-inf, -7, -7, -4, 2]
[-inf, -8, -8, -5, -2]

Matrix I_y:
[-inf, -inf, -inf, -inf, -inf]
[-2, -4, -5, -6, -7]
[-3, 0, -5, -6, -7]
[-4, -1, 2, -3, -4]
[-5, -2, 1, 1, -1]
[-6, -3, 0, 2, 0]
```

Question 2

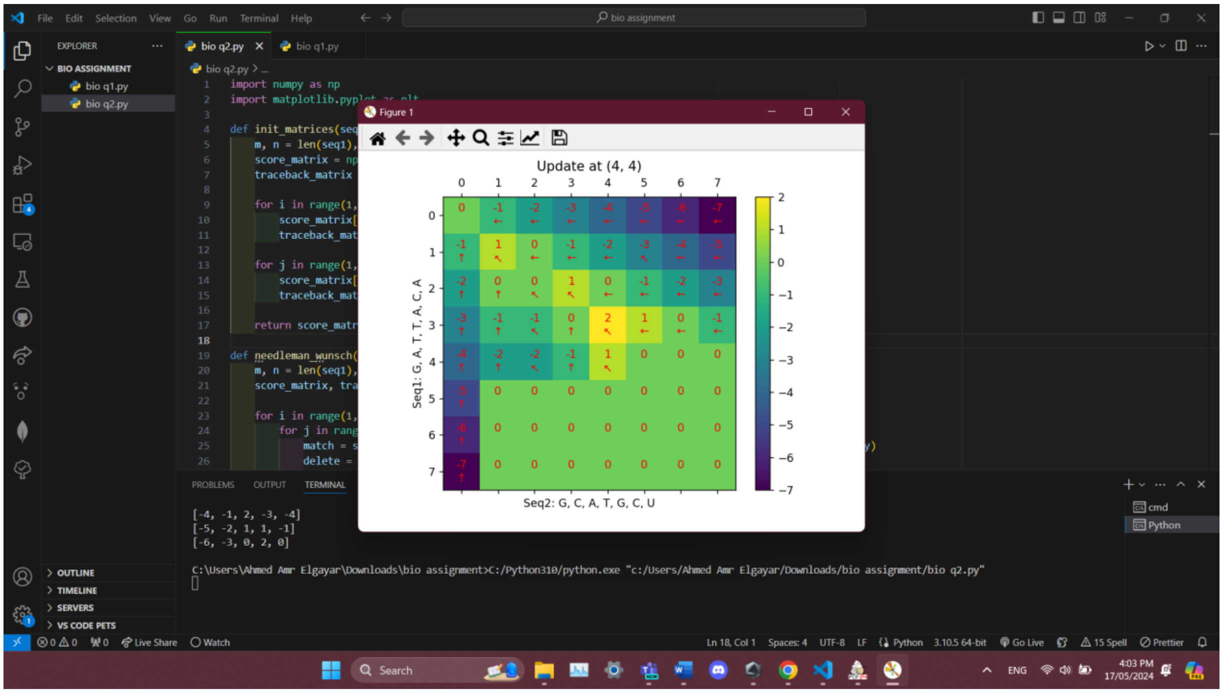
Implement a function (and any helper functions) that takes two DNA sequences and shows the graphical representation of all required steps to get their global alignment using the 2. Needleman-Wunch Algorithm.

Code

```
import numpy as np
import matplotlib.pyplot as plt

def init_matrices(seq1, seq2, gap_penalty):
    m, n = len(seq1), len(seq2)
    score_matrix = np.zeros((m+1, n+1))
    traceback_matrix = np.zeros((m+1, n+1), dtype=str)

    for i in range(1, m+1):
        score_matrix[i, 0] = score_matrix[i-1, 0] + gap_penalty
        traceback_matrix[i, 0] = '↑'

    for j in range(1, n+1):
        score_matrix[0, j] = score_matrix[0, j-1] + gap_penalty
        traceback_matrix[0, j] = '←'

    return score_matrix, traceback_matrix

def needleman_wunsch(seq1, seq2, match_score, mismatch_penalty, gap_penalty):
    m, n = len(seq1), len(seq2)
    score_matrix, traceback_matrix = init_matrices(seq1, seq2, gap_penalty)

    for i in range(1, m+1):
        for j in range(1, n+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)

            if score_matrix[i, j] == match:
                traceback_matrix[i, j] = '↖'
            elif score_matrix[i, j] == delete:
                traceback_matrix[i, j] = '↑'
            else:
                traceback_matrix[i, j] = '←'

            # Visualization after each step
            visualize_matrix(score_matrix, traceback_matrix, i, j, seq1, seq2)
    alignment_a, alignment_b = traceback(seq1, seq2, traceback_matrix)
    return score_matrix, traceback_matrix, alignment_a, alignment_b

def visualize_matrix(score_matrix, traceback_matrix, i, j, seq1, seq2):
    fig, ax = plt.subplots()
    cax = ax.matshow(score_matrix, cmap='viridis')
    plt.title(f"Update at ({i}, {j})")
    fig.colorbar(cax)

    # Annotate the matrix with the values
    for x in range(score_matrix.shape[0]):
        for y in range(score_matrix.shape[1]):
            ax.text(y, x, f'{int(score_matrix[x, y])}\n{traceback_matrix[x, y]}', va='center', ha='center',
color='red')

    plt.xlabel('Seq2: ' + ', '.join(seq2))
    plt.ylabel('Seq1: ' + ', '.join(seq1))
    plt.show()

def traceback(seq1, seq2, traceback_matrix):
    alignment_a = ""
    alignment_b = ""
    i, j = len(seq1), len(seq2)

    while i > 0 or j > 0:
        if traceback_matrix[i, j] == '↖':
            alignment_a = seq1[i-1] + alignment_a
            alignment_b = seq2[j-1] + alignment_b
            i -= 1
            j -= 1
        elif traceback_matrix[i, j] == '↑':
            alignment_a = seq1[i-1] + alignment_a
            alignment_b = '-' + alignment_b
            i -= 1
        else:  # traceback_matrix[i, j] == '←'
            alignment_a = '-' + alignment_a
            alignment_b = seq2[j-1] + alignment_b
            j -= 1

    return alignment_a, alignment_b

# Example DNA sequences
seq1 = "GATTACA"
seq2 = "GCATGCU"

# Perform Needleman-Wunsch and visualize each step
score_matrix, traceback_matrix, alignment_a, alignment_b = needleman_wunsch(seq1, seq2,
match_score=1, mismatch_penalty=-1, gap_penalty=-1)
```

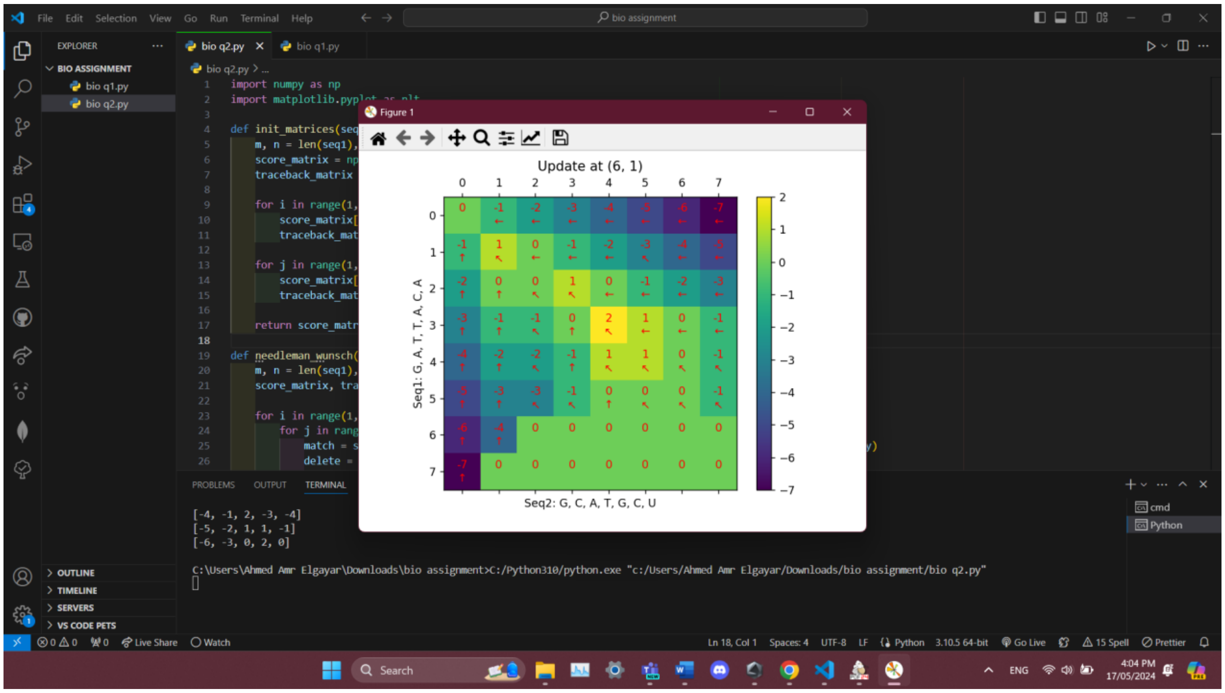
```
# Print the final alignment
print("Alignment:")
print(alignment_a)
print(alignment_b)
```

Output Screenshot



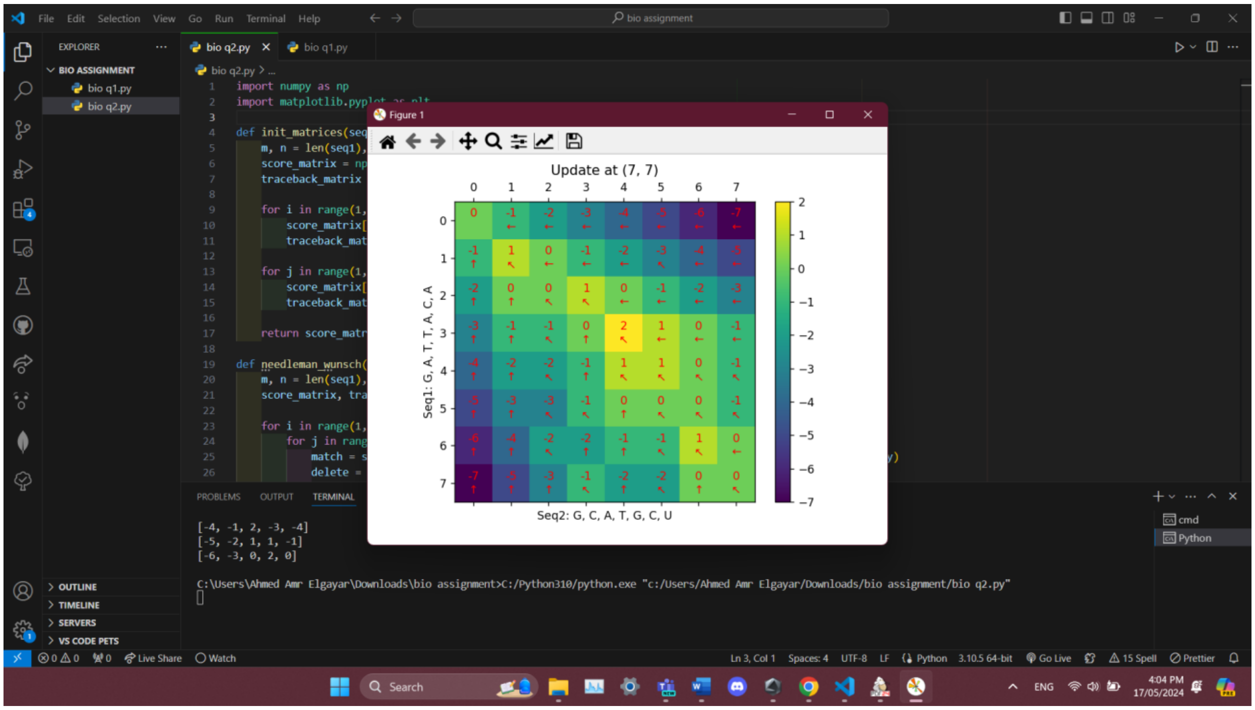
```
import numpy as np
import matplotlib.pyplot as plt

def init_matrices(seq
    m, n = len(seq1),
    score_matrix = np
    traceback_matrix

    for i in range(1,
        score_matrix[
        traceback_mat

    for j in range(1,
        score_matrix[
        traceback_mat

    return score_matr

def needleman_wunsch(
    m, n = len(seq1),
    score_matrix, tra

    for i in range(1,
        for j in rang
            match = s
            delete =
```

```
[-4, -1, 2, -3, -4]
[-5, -2, 1, 1, -1]
[-6, -3, 0, 2, 0]

C:\Users\Ahmed Amr Elgayar\Downloads\bio assignment>C:/Python310/python.exe "c:/Users/Ahmed Amr Elgayar/Downloads/bio assignment/bio q2.py"
```



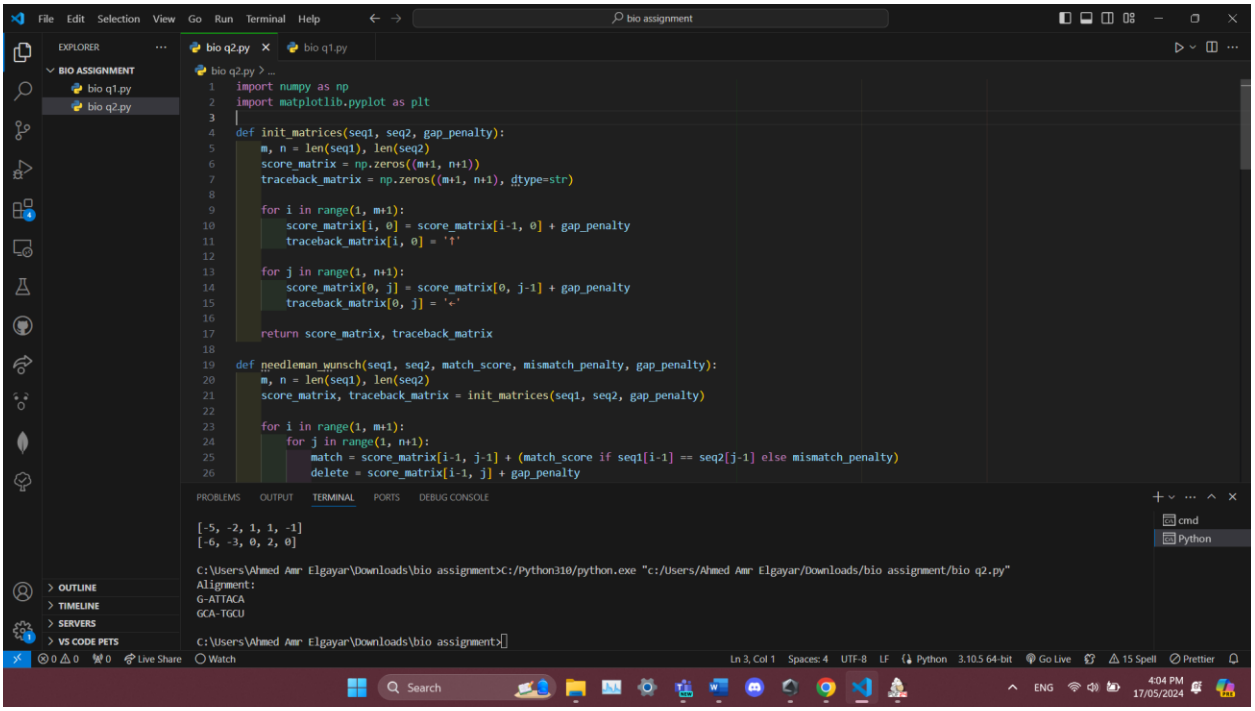
```
import numpy as np
import matplotlib.pyplot as plt

def init_matrices(seq
    m, n = len(seq1),
    score_matrix = np
    traceback_matrix

    for i in range(1,
        score_matrix[
        traceback_mat

    for j in range(1,
        score_matrix[
        traceback_mat

    return score_matr

def needleman_wunsch(
    m, n = len(seq1),
    score_matrix, tra

    for i in range(1,
        for j in rang
            match = s
            delete =
```

```
[-4, -1, 2, -3, -4]
[-5, -2, 1, 1, -1]
[-6, -3, 0, 2, 0]

C:\Users\Ahmed Amr Elgayar\Downloads\bio assignment>C:/Python310/python.exe "c:/Users/Ahmed Amr Elgayar/Downloads/bio assignment/bio q2.py"
```



```python
import numpy as np
import matplotlib.pyplot as plt

def init_matrices(seq
    m, n = len(seq1),
    score_matrix = np
    traceback_matrix

    for i in range(1,
        score_matrix[
        traceback_mat

    for j in range(1,
        score_matrix[
        traceback_mat

    return score_matr

def needleman_wunsch(
    m, n = len(seq1),
    score_matrix, tra

    for i in range(1,
        for j in rang
            match = s
            delete =
```

```
[-4, -1, 2, -3, -4]
[-5, -2, 1, 1, -1]
[-6, -3, 0, 2, 0]

C:\Users\Ahmed Amr Elgayar\Downloads\bio assignment>C:/Python310/python.exe "c:/Users/Ahmed Amr Elgayar/Downloads/bio assignment/bio q2.py"
```



```python
import numpy as np
import matplotlib.pyplot as plt

def init_matrices(seq1, seq2, gap_penalty):
    m, n = len(seq1), len(seq2)
    score_matrix = np.zeros((m+1, n+1))
    traceback_matrix = np.zeros((m+1, n+1), dtype=str)

    for i in range(1, m+1):
        score_matrix[i, 0] = score_matrix[i-1, 0] + gap_penalty
        traceback_matrix[i, 0] = '↑'

    for j in range(1, n+1):
        score_matrix[0, j] = score_matrix[0, j-1] + gap_penalty
        traceback_matrix[0, j] = '←'

    return score_matrix, traceback_matrix

def needleman_wunsch(seq1, seq2, match_score, mismatch_penalty, gap_penalty):
    m, n = len(seq1), len(seq2)
    score_matrix, traceback_matrix = init_matrices(seq1, seq2, gap_penalty)

    for i in range(1, m+1):
        for j in range(1, n+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
```

```
[-5, -2, 1, 1, -1]
[-6, -3, 0, 2, 0]

C:\Users\Ahmed Amr Elgayar\Downloads\bio assignment>C:/Python310/python.exe "c:/Users/Ahmed Amr Elgayar/Downloads/bio assignment/bio q2.py"
Alignment:
G-ATTACA
GCA-TGCU

C:\Users\Ahmed Amr Elgayar\Downloads\bio assignment>
```