

Instructions

This problem set is provided in the form of a Jupyter notebook. Problems are posed within this notebook file and you are expected to provide codes and/or written answers when prompted. Remember that you can use Markdown cells to format written responses where necessary.

Before submitting your assignment, be sure to do a clean run of your notebook and **verify that your cell outputs (e.g., prints, figures, tables) are correctly shown**. To do a clean run, click *Kernel*→*Restart Kernel and Run All Cells*....

You are required to submit this notebook to Gradescope in two forms:

1. Submit a PDF of the completed notebook. To produce a PDF, you can use *File*→*Save and Export Notebook As...*→*HTML* and then convert the HTML file to a PDF using your preferred web browser. **Verify that your code, written answers, and cell outputs are visible in the submitted PDF.**
2. Submit a zip file (including the `.ipynb` file) of this assignment to Gradescope.

Online Resources and Collaborators

Please list the online resources you used and the names of other students you collaborated with while working on this problem set.

- Online resources:
[https://www.biostars.org/p/19122/#:~:text=high%20gap%20opening%20penalty:%20point,aligned%20across%20their%20full%20length.\)](https://www.biostars.org/p/19122/#:~:text=high%20gap%20opening%20penalty:%20point,aligned%20across%20their%20full%20length.)
- Student Names:

✓ Problem set 0: introduction to genomic data.

In this problem set, you will access the genome of the original Sars-CoV-2 strain (causal agent of the covid-19 pandemic) and translate one of its genes into a protein product. The learning objectives are:

1. Become familiar with BioPython
2. Become familiar with fasta formatted sequence files

3. Build understanding of how information from DNA is translated into protein by implementing your own translation function

✓ 00. Install BioPython (5 points)

You can find the installation instructions here: <https://biopython.org/wiki/Download>

```
# install BioPython and import it
!pip install biopython
```

```
from Bio import SeqIO
import pandas as pd
```

```
Requirement already satisfied: biopython in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
# Unzip the ncbi_dataset.zip file into the Colab environment's filesystem (in-n
!unzip -o '/content/drive/MyDrive/690U_assignments/ncbi_dataset.zip' -d '/conte
```

```
# Optional: List the contents of the unzipped directory to verify
!ls '/content/ncbi_dataset/data/'
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
Archive: /content/drive/MyDrive/690U_assignments/ncbi_dataset.zip
  inflating: /content/README.md
  inflating: /content/ncbi_dataset/data/data_report.jsonl
  inflating: /content/ncbi_dataset/data/genomic.fna
  inflating: /content/ncbi_dataset/data/protein.faa
  inflating: /content/ncbi_dataset/data/virus_dataset.md
  inflating: /content/ncbi_dataset/data/dataset_catalog.json
  inflating: /content/md5sum.txt
data_report.jsonl    genomic.fna  virus_dataset.md
dataset_catalog.json protein.faa
```

```
!ls -F /content
```

```
drive/  md5sum.txt  ncbi_dataset/  README.md  sample_data/
```

✓ 01. Use Bio.SeqIO to read in the genome sequence of Sars-CoV-2 (15 points)

The covid-19 pandemic was caused by a virus called Sars-Cov-2. One of the original Sars-CoV-2 genomes sequenced is this one:

<https://www.ncbi.nlm.nih.gov/datasets/taxonomy/2697049/>

Use the "download" button to access the genome sequence and the protein sequence data from this genome. You can view a graphical layout of the genes in the genome on the website as well. DO NOT click the "download all genomes" button! This downloads all the viral sequences on NCBI

Use Bio.SeqIO to read in the fasta-formatted **genomic** data file that you downloaded. This should contain a single nucleotide sequence, representing the full genome

SeqIO documentation: <https://biopython.org/docs/1.75/api/Bio.SeqIO.html> (hint: use the 'parse' function)

```
# for protein_record in SeqIO.parse("/content/ncbi_dataset/data/protein.faa", "
#     print("%s %i" % (protein_record.id, len(protein_record)))
```

```
for gene_record in SeqIO.parse("/content/ncbi_dataset/data/genomic.fna", "fasta
    print("%s %i" % (gene_record.id, len(gene_record)))
    print("Gene: ", gene_record)
```

```
NC_045512.2 29903
Gene: ID: NC_045512.2
Name: NC_045512.2
Description: NC_045512.2 Severe acute respiratory syndrome coronavirus 2 isolate
Number of features: 0
Seq('ATTAAAGGTTTATACCTTCCCAGGTAACAAACCAACCACTTTCGATCTCTTGT...AAA')
```

02. Translate the sequence of the Spike protein (25 points total)

The spike protein is the protein responsible for facilitating entry of covid-19 into human cells. It is also the protein targeted by antibodies produced by our immune system. Thus, mutations in the spike protein are extremely interesting - they may facilitate increased ability of the virus to evade immunity or enter cells. We are going to translate the gene sequence of the spike protein (the gene is called "S") into a protein sequence. This gene is encoded from position 21563 to 25384 (1-indexed) on the chromosome in the + direction (you can see info on this gene here:

<https://www.ncbi.nlm.nih.gov/gene/43740568/>).

- ✓ 02.A extract the gene sequence (subsequence of the genome sequence) corresponding to the spike protein (10 points)

```
spike_start = 21562
spike_end = 25384
spike_seq = gene_record.seq[spike_start:spike_end]

print("Spike gene length (nt):", len(spike_seq))
print(spike_seq)
```

```
Spike gene length (nt): 3822
ATGTTTGTGTTTCTTGTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCCCTGC
```

- ✓ 02.B Use the translation table below to translate the sequence of the spike gene ("S") into a protein sequence (15 points)

```
# Use the translation table
# _ indicates a stop codon
translation_table = {
    "TTT": "F", "TTC": "F", "TTA": "L", "TTG": "L",
    "TCT": "S", "TCC": "S", "TCA": "S", "TCG": "S",
    "TAT": "Y", "TAC": "Y", "TAA": "_", "TAG": "_",
    "TGT": "C", "TGC": "C", "TGA": "_", "TGG": "W",
    "CTT": "L", "CTC": "L", "CTA": "L", "CTG": "L",
    "CCT": "P", "CCC": "P", "CCA": "P", "CCG": "P",
    "CAT": "H", "CAC": "H", "CAA": "Q", "CAG": "Q",
    "CGT": "R", "CGC": "R", "CGA": "R", "CGG": "R",
    "ATT": "I", "ATC": "I", "ATA": "I", "ATG": "M",
    "ACT": "T", "ACC": "T", "ACA": "T", "ACG": "T",
    "AAT": "N", "AAC": "N", "AAA": "K", "AAG": "K",
    "AGT": "S", "AGC": "S", "AGA": "R", "AGG": "R",
    "GTT": "V", "GTC": "V", "GTA": "V", "GTG": "V",
    "GCT": "A", "GCC": "A", "GCA": "A", "GCG": "A",
    "GAT": "D", "GAC": "D", "GAA": "E", "GAG": "E",
    "GGT": "G", "GGC": "G", "GGA": "G", "GGG": "G"
}
```

```
def translate_seq(nuc_seq, translation_table, unknown='X'):
    protein_seq = ""
    for i in range(0, len(spike_seq)-2, 3):
        codon = str(spike_seq[i:i+3])
        translation = translation_table.get(codon, "X")
        protein_seq += translation
    return protein_seq, len(protein_seq)
```

```
# translate the sequence of the spike gene into a protein sequence and print it

protein_seq = translate_seq(spike_seq, translation_table)[0]
```

✓ 03. Check your work (5 points)

The protein.faa file you downloaded contains the translated sequences of all genes in the genome. The identifier ("id") for the S protein is called **YP_009724390.1:1-1273**. Again using Bio.SeqIO, read in the protein sequences from the protein.faa file, select the sequence corresponding to the spike protein based on its identifier, and compare it to your translated sequence. They should match, except that our translation table above puts a "_" at the end of sequences to represent the stop codon!

I realize that comparisons like this can be somewhat boring, but it's very important to build checks into your pipelines! Biological data can be very heterogenous.

```
for record in SeqIO.parse("/content/ncbi_dataset/data/protein.faa", "fasta"):
    if record.id.startswith("YP_009724390.1:1-1273"):
        reference_spike = str(record.seq)
        break
```

```
print("Length of reference spike:", len(reference_spike))
print("Reference spike:", reference_spike)
```

```
Length of reference spike: 1273
Reference spike: MFVFLVLLPLVSSQCVNLTTRTQLPPAYTNSFTRGVYYPDKVFRSSVLHSTQDLFLPFFSNVT
```

```
matches = protein_seq[:-1] == reference_spike
print("Matches:", matches)
```

```
Matches: True
```

✓ 04. Implement the Needleman-Wunsch algorithm and interpret (45 points total)

✓ 04A. Implement the Needleman-Wunsch algorithm using the BLOSUM80 substitution matrix. (25 points)

I suggest using the `blosum` python package which provides the substitution matrices

<https://github.com/not-a-feature/blosum>

Use a gap penalty of -8. Provide an alignment for the two sequences below, as well as their score:

```
!pip install blosum

import blosum as bl
import numpy as np

def NW(sequence1, sequence2, S, gap_penalty=-8):
    """
    sequence1: str
        the first protein sequence
    sequence2: str
        the second protein sequence

    S: blosum._blosum.BLOSUM matrix
        the substitution matrix to score matches and mismatches
    gap_penalty: int, optional (default=-8)
        the gap penalty to use (note: does not support affine gap penalty)
    """
    # initialising the table
    n = len(sequence1)
    m = len(sequence2)

    score = np.zeros((n+1, m+1), dtype=int)
    traceback = np.zeros((n+1, m+1), dtype=int)
    # ensures F(0,0) = 0

    # Filling the first row - F(i,0) = F(i-1,0) - d
    for i in range(1, n+1):
        score[i, 0] = i * gap_penalty
        traceback[i, 0] = 1

    # Filling the first column - F(0,j) = F(0,j-1) - d
    for j in range(1, m+1):
        score[0, j] = j * gap_penalty
        traceback[0, j] = 2

    # getting max score
    for i in range(1, n+1):
        for j in range(1, m+1):
            # calculating for match.mismatch - F(i-1,j-1)+s(xi,yj)
            match = score[i-1, j-1] + S[sequence1[i-1]][sequence2[j-1]]

            # calculating for deletion - F(i,0) = F(i-1,j)-d
            delete = score[i-1, j] + gap_penalty
```

```

# calculating for insertion -  $F(i,j) = F(i,j-1) - d$ 
insert = score[i, j-1] + gap_penalty

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

# Keeping a track of match/insertion/deletion
if score[i, j] == match:
    traceback[i, j] = 0
elif score[i, j] == delete:
    traceback[i, j] = 1
else:
    traceback[i, j] = 2

align1 = ""
align2 = ""
i, j = n, m

# Aligning final match sequences
while i > 0 or j > 0:
    if traceback[i, j] == 0:
        align1 = sequence1[i-1] + align1
        align2 = sequence2[j-1] + align2
        i -= 1
        j -= 1
    elif traceback[i, j] == 1:
        align1 = sequence1[i-1] + align1
        align2 = "-" + align2
        i -= 1
    else:
        align1 = "-" + align1
        align2 = sequence2[j-1] + align2
        j -= 1

return align1, align2, int(score[n, m]), score, traceback # Replace it and

```

Requirement already satisfied: blosum in /usr/local/lib/python3.12/dist-packages

```

def visualize_traceback(score, traceback, seq1_labels=None, seq2_labels=None, cell_w=10):
    """
    Print a grid of scores with an arrow in each cell.

    Args:
        score (array-like): score matrix shape (n+1, m+1)
        traceback (array-like): traceback matrix same shape (0=diag,1=up,2=left)
        seq1_labels (str or list, optional): labels for rows (original seq1). If None, use 0..n
        seq2_labels (str or list, optional): labels for cols (original seq2). If None, use 0..m
        cell_w (int): width of each printed cell
    """
    import numpy as np

```

```

score = np.asarray(score)
tb = np.asarray(traceback)
rows, cols = score.shape # rows = n+1, cols = m+1
n = rows - 1
m = cols - 1

# prepare labels (index 0 is the gap/corner)
def to_labels(x, length):
    if x is None:
        return [' '] * length
    if isinstance(x, str):
        x = list(x)
    x = list(x)
    # truncate or pad with '-' to required length
    if len(x) > length:
        return x[:length]
    return x + ['-'] * (length - len(x))

row_labels = to_labels(seq1_labels, n)
col_labels = to_labels(seq2_labels, m)

arrow = {0: '^', 1: '↑', 2: '←'}

# header: corner then col labels (j=1..m)
corner = ' ' * cell_w
header = corner
for j in range(1, m+1):
    header += f"{col_labels[j-1]:^{cell_w}}"
print(header)

# rows: i=0..n
for i in range(0, n+1):
    left_label = ' ' if i == 0 else row_labels[i-1]
    row_str = f"{left_label:^{cell_w}}"
    for j in range(0, m+1):
        sc = int(score[i, j])
        tb_code = int(tb[i, j]) if (0 <= i < rows and 0 <= j < cols) else 1
        arr = arrow.get(tb_code, ' ')
        # show number and arrow; adjust formatting if numbers become wide
        cell = f"{sc:>{cell_w-2}}{arr:>2}"
        row_str += cell
    print(row_str)

```

```

sequence1 = "HEAGAWGHEE"
sequence2 = "PAWHEAE"
blosum_80_matrix = bl.BLOSUM(80)

```

```

align1, align2, final_score, score_mat, tb = NW(sequence1, sequence2, S=blosum_
print("Alignment:")

```

```

print(align1)
print(align2)
print("Final score:", final_score)

visualize_traceback(score_mat, tb, seq1_labels=sequence1, seq2_labels=sequence2)

```

Alignment:

HEAGAWGHE-E

--P-AW-HEAE

Final score: -5

	P	A	W	H	E	A	E	
	0 ↖	-8 ←	-16 ←	-24 ←	-32 ←	-40 ←	-48 ←	-56 ←
H	-8 ↑	-3 ↖	-10 ↖	-18 ←	-16 ↖	-24 ←	-32 ←	-40 ←
E	-16 ↑	-10 ↖	-4 ↖	-12 ←	-18 ↖	-10 ↖	-18 ←	-26 ↖
A	-24 ↑	-17 ↖	-5 ↖	-7 ↖	-14 ↖	-18 ↑	-5 ↖	-13 ←
G	-32 ↑	-25 ↑	-13 ↑	-9 ↖	-10 ↖	-17 ↖	-13 ↑	-8 ↖
A	-40 ↑	-33 ↖	-20 ↖	-16 ↖	-11 ↖	-11 ↖	-12 ↖	-14 ↖
W	-48 ↑	-41 ↑	-28 ↑	-9 ↖	-17 ←	-15 ↖	-14 ↖	-16 ↖
G	-56 ↑	-49 ↑	-36 ↑	-17 ↑	-12 ↖	-20 ↖	-15 ↖	-17 ↖
H	-64 ↑	-57 ↑	-44 ↑	-25 ↑	-9 ↖	-12 ↖	-20 ←	-15 ↖
E	-72 ↑	-65 ↑	-52 ↑	-33 ↑	-17 ↑	-3 ↖	-11 ←	-14 ↖

✓ 04B: Now run the same code using a gap penalty of -10. (20 points)

You should get different results. Explain why, including discussing the total alignment score and the match/mismatch scores of the relevant amino acids (1-2 sentences).

Answer: A higher gap penalty makes gaps more costly, causing the algorithm to prefer mismatches in some positions. This changes the dynamic programming scores, traceback path, and ultimately final alignment.

```

sequence1 = "HEAGAWGHEE"
sequence2 = "PAWHEAE"
blosum_80_matrix = bl.BLOSUM(80)

align1, align2, final_score, score_mat, tb = NW(sequence1, sequence2, S=blosum_80_matrix)
print("Alignment:")
print(align1)
print(align2)
print("Final score:", final_score)

# visualize using the ORIGINAL sequences (sequence1 down rows, sequence2 across columns)
visualize_traceback(score_mat, tb, seq1_labels=sequence1, seq2_labels=sequence2)

```

Alignment:

HEAGAWGHEE

--P-AWHEAE

Final score: -13

	P	A	W	H	E	A	E
--	---	---	---	---	---	---	---

	0 ↖	-10 ←	-20 ←	-30 ←	-40 ←	-50 ←	-60 ←	-70 ←
H	-10 ↑	-3 ↖	-12 ↖	-22 ←	-22 ↖	-32 ←	-42 ←	-52 ←
E	-20 ↑	-12 ↖	-4 ↖	-14 ←	-22 ↖	-16 ↖	-26 ←	-36 ↖
A	-30 ↑	-21 ↖	-7 ↖	-7 ↖	-16 ↖	-23 ↖	-11 ↖	-21 ←
G	-40 ↑	-31 ↑	-17 ↑	-11 ↖	-10 ↖	-19 ↖	-21 ↑	-14 ↖
A	-50 ↑	-41 ↖	-26 ↖	-20 ↖	-13 ↖	-11 ↖	-14 ↖	-22 ↖
W	-60 ↑	-51 ↑	-36 ↑	-15 ↖	-23 ↖	-17 ↖	-14 ↖	-18 ↖
G	-70 ↑	-61 ↑	-46 ↑	-25 ↑	-18 ↖	-26 ↖	-17 ↖	-17 ↖
H	-80 ↑	-71 ↑	-56 ↑	-35 ↑	-17 ↖	-18 ↖	-27 ↑	-17 ↖
E	-90 ↑	-81 ↑	-66 ↑	-45 ↑	-27 ↑	-11 ↖	-19 ↖	-21 ↖
E	-100 ↑	-91 ↑	-76 ↑	-55 ↑	-37 ↑	-21 ↖	-12 ↖	-13 ↖

05. Construct an alignment of the S protein against the S protein from MERS (5 points)

Sars-Cov-2, the virus that causes covid-19, is related to other viruses including the one that causes Middle East respiratory syndrome (MERS). This can be seen by aligning the sequences of the S proteins from the two viruses. Read in the MERS_S_protein.faa sequence from the data folder, and use your Needleman-Wunsch code to align the two sequences (using a gap penalty of -8) and report their alignment score.

```
for record in SeqIO.parse("/content/drive/MyDrive/690U_assignments/SARS_data/ME
    mers_s_spike = str(record.seq)

print("MERS_S spike length:", len(mers_s_spike))
print("MERS_S spike:", mers_s_spike)
```

```
MERS_S spike length: 1353
MERS_S spike: MIHSVFLLMFLLTPTESYVDVGPDSVKSACIEVDIQQTFFDKTWPRPIDVSKADGIIYPQGRITYSN
```

```
for gene_record in SeqIO.parse("/content/ncbi_dataset/data/genomic.fna", "fasta
    print("%s %i" % (gene_record.id, len(gene_record)))
    print("Gene: ", gene_record)
```

```
NC_045512.2 29903
Gene: ID: NC_045512.2
Name: NC_045512.2
Description: NC_045512.2 Severe acute respiratory syndrome coronavirus 2 isolate
Number of features: 0
Seq('ATTAAAGGTTTATACCTTCCCAGGTAACAAACCAACCACTTTCGATCTCTTGT...AAA')
```

```
reference_spike, reference_spike_len = translate_seq(gene_record, translation_t
print(mers_s_spike)
print(reference_spike)
print("lengths: ", mers_s_spike_len, reference_spike_len)
```

```
MIHSVFLLMFLLTPTESYVDVGPDSVKSACIEVDIQQTFFDKTWPRPIDVSKADGIIYPQGRTYSNITITYQGLFPYQGD
MFVFLVLLPLVSSQCVNLTTRTQLPPAYTNSFTRGVYYPDKVFRSSVLHSTQDLFLPFFSNVTWFHAIHVSGTNGTKRFD
lengths: 1274 1274
```

```
# sequence matching
```

```
blosum_80_matrix = bl.BLOSUM(80)
align1, align2, final_score, score_mat, tb = NW(protein_seq[:-1], mers_s_spike,
print("Alignment:")
print(align1)
print(align2)
print("Final score:", final_score)
```

```
# visualize using the ORIGINAL sequences (sequence1 down rows, sequence2 across
# visualize_traceback(score_mat, tb, seq1_labels=sequence1, seq2_labels=sequence2)
```

```
Alignment:
```

```
MF--VFLVLLPLVSSQC-VNLTTRTQLPPAYTNSFTRGVYYPDKVFRSSVLHSTQD-LFLP---FFSNVTW-FHAIH-VS
MIHSVFLLMFLLTPTESYVDV-GPDSVKSACIEVDIQQTFF-DKTWPRPIDVSKADGIIYPQGRTYSNITITYQGLFPYQ
Final score: 989
```

While it may be challenging to see the similarity in these sequences by eye, I encourage you to view figure 1 in this paper:

<https://www.biorxiv.org/content/10.1101/2020.04.17.047548v1.full> which shows the structural similarity in the S protein between the two sequences. Alignments like this were used in the early days of the pandemic to establish that covid-19 was related to previously observed respiratory viruses such as MERS.

Start coding or [generate](#) with AI.