Practical 8

In each of the questions, the input is given in the form of FASTA files. Here is the general structure of code to read the FASTA file. Same has been used throughout to read the files for each question. The code is given below:

```
from Bio import SeqIO

input_file = "FASTA_file_location.fasta"
fasta_sequences = SeqIO.parse(open(input_file),'fasta')
for fasta in fasta_sequences:
    name, sequence = fasta.id, str(fasta.seq)

# The typical format of the FASTA file is as follows:
# >(Name of the sequence)
# Sequence of Amino acids
# Ex.
# >Seq_Name_Ex
# ATEDADEDKIFDCAEE
```

Listing 1. Code to read a FASTA file

Now, each of the following questions requiring code will contain the function that is essential to obtain the objective, and the FASTA file will be read using the above code format itself.

Question 1. Obtain the hydrophobicity profile for the sequences (Q1.fasta) and identify the α **helices** and β **strands**. **Hydrophobicity values**:
A: 13.85 **D**: 11.61 **C**: 15.37 **E**: 11.38 **F**: 13.93 **G**: 13.34 **H**: 13.82 **I**: 15.28 **K**: 11.58 **L**: 14.13
M: 13.86 **N**: 13.02 **P**: 12.35 **Q**: 12.61 **R**: 13.10 **S**: 13.39 **T**: 12.70 **V**: 14.56 **W**: 15.48 **Y**: 13.88

Q1 FASTA file content

>**seq1**
FDCAEYRSTNIYGYGLYEVSMKPAKNTGIVSSFFTYTGPAHGTQWEIDIEFLGKD
TTKVQFNYYTNGVGGHEKVISLGFDASKGFHTYAFDWQPGYIKWYVDGVLK
>**seq2**
KASEDLVKKHAGVLGAILKKKGHHEAELKPLAQSHATKAHKNIFISEAIIHVL
HSRHPGDFGADAQGAMNKALELFRKDIAAKYKELGY
>**seq3**
TVEGAGSIAAATGFVKKDQLGKNEEGAPQEGILEDMPVDPDNEAYEMPSEEGY
QDYEPEA

Solution. Firstly, I compute the hydrophobicity values of all the amino acids in the given sequence. Then, I plot the hydrophobicity values and also plot the mean of hydrophobicity values. This mean plays a crucial role in deciding the alpha helix and beta strand sections within the given sequence.

- **Identification of α helices**: It a sub-sequence of length at least 8, where you have sets of two amino acids having hydrophobicity values above and below the mean hydrophobicity value. Let's denote above mean as + and below mean as |. An example of the accepted alpha helix sub-sequence is ++||++||++.
- **Identification of β strands**: It a sub-sequence of length at least 5, where you have every consecutive 2 amino acids pair having hydrophobicity values above and below the mean hydrophobicity value. Let's denote above mean as + and below mean as |. An example of the accepted alpha helix sub-sequence is +|+|+.

```python
import numpy as np

# Computes the distance of the hyrdrophobicity value from the mean
# Intention is just to indentify whether it is above or below the mean
def dist_from_mean(hydro_value, mean_value):
    return hydro_value - mean_value

# Identifying whether there exist ranges of alpha helix in the sequence
# Identified by the pattern of hydrophobicity value with respect to mean
def identify_alpha_helices(hydro_val, hydro_mean):
    alpha_helix_range = []
    i = 0
    while i < len(hydro_val):
        count = 0
        for j in range(i, len(hydro_val)-1):
            if (dist_from_mean(hydro_val[j], hydro_mean[j]) * dist_from_mean(hydro_val[j+1], hydro_mean[j+1])) > 0 and j%2 == 0:
                count += 1
            elif (dist_from_mean(hydro_val[j], hydro_mean[j]) * dist_from_mean(hydro_val[j+1], hydro_mean[j+1])) < 0 and j%2 != 0:
                count += 1
            else:
                if count >= 8:
                    alpha_helix_range.append([i, j])
                break
        if count >= 8:
            i = j
        else:
            i += 1
    return alpha_helix_range

# Identifying whether there exist ranges of beta strand in the sequence
# Identified by the pattern of hydrophobicity value with respect to mean
def identify_beta_strands(hydro_val, hydro_mean):
    beta_strand_range = []
    i = 0
    while i < len(hydro_val):
        count = 0
        for j in range(i, len(hydro_val)-1):
            if (dist_from_mean(hydro_val[j], hydro_mean[j]) * dist_from_mean(hydro_val[j+1], hydro_mean[j+1])) < 0:
                count += 1
            else:
                if count >= 5:
                    beta_strand_range.append([i, j])
                i = j
                break
        if count >= 5:
            i = j
        else:
            i += 1
    return beta_strand_range
```

LISTING 2. Functions to compute α helix and β strand ranges

Above are the functions that will be used while iterating through the sequence to compute the ranges of alpha helix and beta strands.

```
hydrophobicity =
{"A": 13.85, "D": 11.61, "C": 15.37, "E": 11.38, "F": 13.93,
 "G": 13.34, "H": 13.82, "I": 15.28, "K": 11.58, "L": 14.13,
 "M": 13.86, "N": 13.02, "P": 12.35, "Q": 12.61, "R": 13.10,
 "S": 13.39, "T": 12.70, "V": 14.56, "W": 15.48, "Y": 13.88}

input_file = "Q1.fasta"
fasta_sequences = SeqIO.parse(open(input_file),'fasta')
for fasta in fasta_sequences:
    name, sequence = fasta.id, str(fasta.seq)
    hydrophobicity_val = []
    for i in range(len(sequence)):
        hydrophobicity_val.append(hydrophobicity[sequence[i]])

    mean = [np.mean(hydrophobicity_val)] * len(sequence)

    alpha_range = identify_alpha_helices(hydrophobicity_val, mean)
    beta_range = identify_beta_strands(hydrophobicity_val, mean)

    plt.figure(figsize=(15,5))

    plt.plot(hydrophobicity_val, color="green", linestyle='-', linewidth=2,
             label='Hydrophobicity value for each residue')
    plt.plot(mean, linestyle="dashed", color="red", linewidth=2,
             label='Hydrophobicity mean of sequence')

    for checker, x_range in enumerate(alpha_range):
        plt.axvspan(x_range[0], x_range[1], color='orange', alpha=0.3,
                    label="Alpha Helices" if checker == 0 else '')
    for checker, x_range in enumerate(beta_range):
        plt.axvspan(x_range[0], x_range[1], color='blue', alpha=0.3,
                    label="Beta Strands" if checker == 0 else '')

    plt.grid(True)
    plt.legend(loc='upper right', fontsize=8, fancybox=True, shadow=True)
    plt.gca().set_facecolor('#f0f0f0')
    plt.show()
```

LISTING 3. Plotting hydrophobicity profile with α helix and β strand ranges

In order to find the ranges of α helices, I have fixed the minimum consecutive amino acids, that have to obey the property for α helix mentioned above, (i.e. the stretch length) as 8.
In order to find the ranges of β strands, I have fixed the minimum consecutive amino acids, that have to obey the property for β strand mentioned above, (i.e. the stretch length) as 5.

Sequence No. in Q1.fasta	Ranges of α helices	Ranges of β strands
Sequence 1	-	(03, 08)
	-	(11, 15)
	-	(43, 50)
	-	(57, 62)
	-	(76, 80)
Sequence 2	(00, 10)	(23, 28)
	(34, 43)	(54, 60)
	-	(62, 66)
Sequence 3	-	(34, 38)
	-	(44, 49)

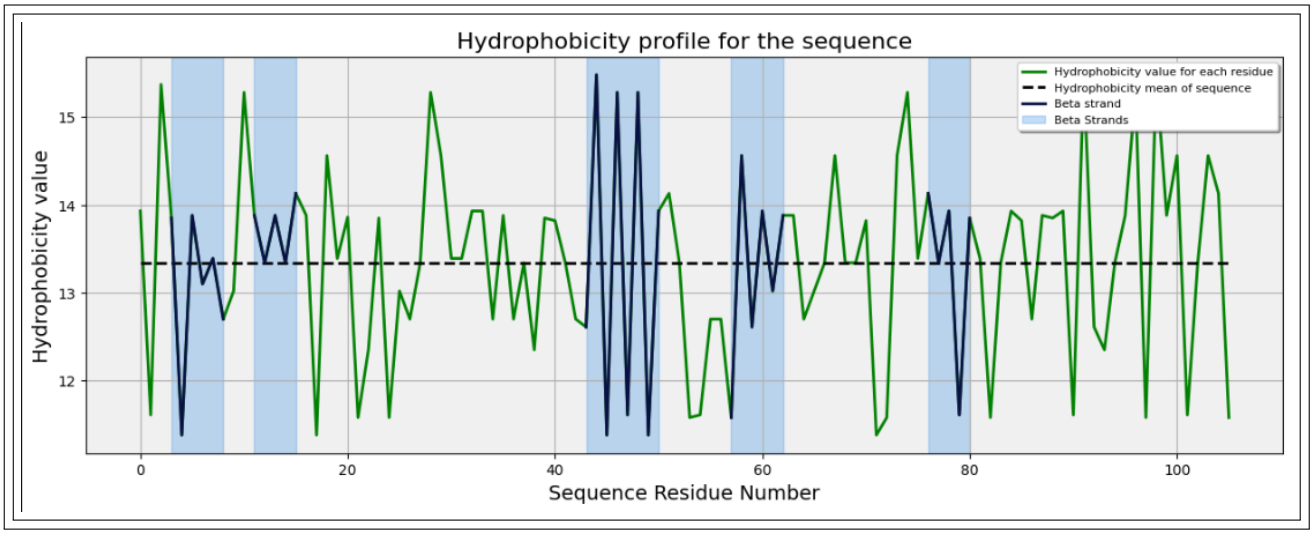

FIGURE 1. Hydrophobicity Profile of Sequence 1 in Q1.fasta

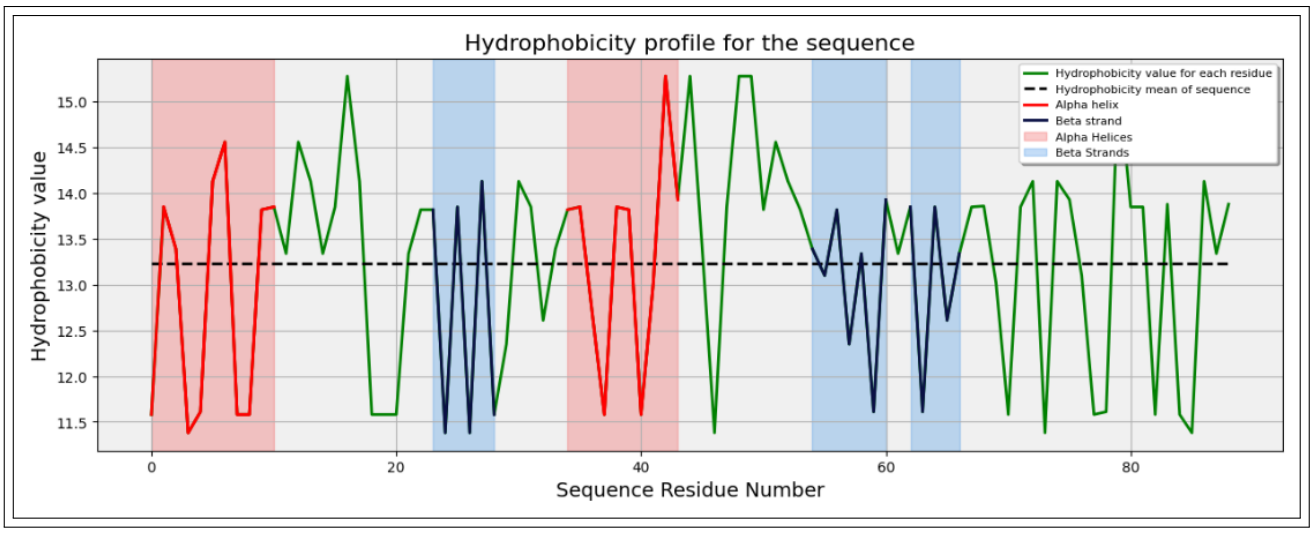

FIGURE 2. Hydrophobicity Profile of Sequence 2 in Q1.fasta

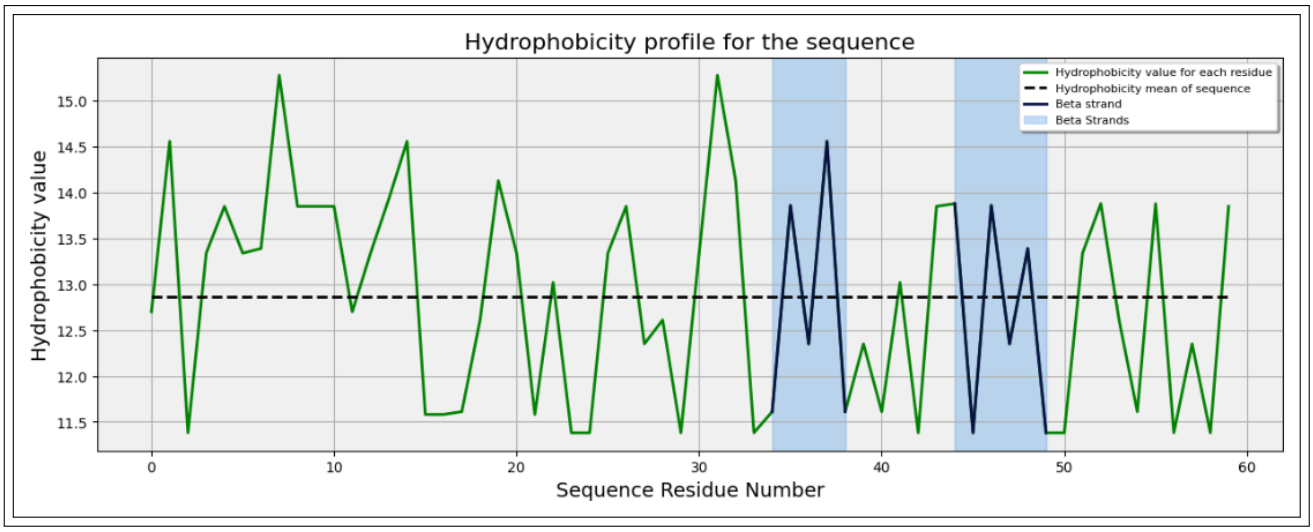

FIGURE 3. Hydrophobicity Profile of Sequence 3 in Q1.fasta

Question 2. Calculate the amphiphatic index for the helices and strands found in Q1. Use stretch lengths of 8 and 6 for α helices and β strands, respectively.

Solution. Firstly, I compute the hydrophobicity values of all the amino acids in the given sequence. Then, I plot the hydrophobicity values and also plot the mean of hydrophobicity values. This mean plays a crucial role in deciding the alpha helix and beta strand sections within the given sequence.

- **Identification of α helices**: It a sub-sequence of length at least 8, where you have sets of two amino acids having hydrophobicity values above and below the mean hydrophobicity value. Let's denote above mean as + and below mean as |. Examples of the accepted alpha helix sub-sequence include ++||++||++, +||++||++.
- **Identification of β strands**: It a sub-sequence of length at least 6, where you have every consecutive 2 amino acids pair having hydrophobicity values above and below the mean hydrophobicity value. Let's denote above mean as + and below mean as |. An example of the accepted alpha helix sub-sequence is +|+|+.
- **Amphiphatic indices for α helices**: The residues of an α-helical segment are considered on four adjacent edges along the direction of the helical axis. The average hydrophobicity of the residues constituting the edge i (i = 1,4) is given by:

(1) $$a_i = (\sum h_{i+j})/n$$

where n is the total number of residues in the edge, j increases at an interval of 4 from 0 to m, m being the number of residues in the helix; h is the hydrophobic index of the residue. The power of amphipathicity of a helix is taken to be:

(2) $$A_\alpha = |(a_1 + a_2)\text{–}(a_3 + a_4)| or |(a_1 + a_4)\text{–}(a_2 + a_3)|$$

- **Amphiphatic indices for β helices**: A β-strand segment is considered to have two faces and the average hydrophobicity of residues constituting the face i (i = 1, 2) is given by:

(3) $$b_i = (\sum h_{i+j})/n$$

where n is the total number of residues in the face, j increases at an interval of 2 from 0 to m, m being the number of residues in the strand; h is the hydrophobic index of the residue. The power of amphipathicity of a β strand is taken to be:

(4) $$A_\beta = |b_1\text{–}b_2|$$

Below is the code for the program to compute the amphipathicity of α helices and β strands

```
def amphipathicity_alpha(hydro_mean, alpha_x, alpha_y):
    amphi_values = []
    for y in alpha_y:
        a1 = np.mean([y[i] for i in range(0, len(y), 4)])
        a2 = np.mean([y[i] for i in range(1, len(y), 4)])
        a3 = np.mean([y[i] for i in range(2, len(y), 4)])
        a4 = np.mean([y[i] for i in range(3, len(y), 4)])

        if len(y) != 0:
            if (dist_from_mean(hydro_mean, y[0]) *
                        dist_from_mean(hydro_mean, y[1])) > 0:
                amphi_values.append(abs(a1+a2-a3-a4))
            else:
                amphi_values.append(abs(a1+a4-a2-a3))
        else:
            amphi_values.append(0)
    return amphi_values

```

```
def amphipathicity_beta(beta_x, beta_y):
    amphi_values = []
    for y in beta_y:
        b1 = np.mean([y[i] for i in range(0, len(y), 2)])
        b2 = np.mean([y[i] for i in range(1, len(y), 2)])
        if len(y) != 0:
                amphi_values.append(abs(b1-b2))
        else:
            amphi_values.append(0)
    return amphi_values
```

LISTING 4. Code to compute the amphipathicity of α helices and β strands

```
# Computes the distance of the hyrdrophobicity value from the mean
# Intention is just to indentify whether it is above or below the mean
def dist_from_mean(hydro_value, mean_value):
    return hydro_value - mean_value

# Identifying whether there exist ranges of alpha helix in the sequence
# Identified by the pattern of hydrophobicity value with respect to mean
def identify_alpha_helices(hydro_val, hydro_mean):
    alpha_helix_range = []
    i = 0
    while i < len(hydro_val):
        count = 0
        for j in range(i, len(hydro_val)-1):
            if (dist_from_mean(hydro_val[j], hydro_mean[j]) * dist_from_mean(
   hydro_val[j+1], hydro_mean[j+1])) > 0 and j%2 == 0:
                count += 1
            elif (dist_from_mean(hydro_val[j], hydro_mean[j]) *
   dist_from_mean(hydro_val[j+1], hydro_mean[j+1])) < 0 and j%2 != 0:
                count += 1
            else:
                if count >= 8:
                    alpha_helix_range.append([i, j])
                break
        if count >= 8:
            i = j
        else:
            i += 1
    return alpha_helix_range

# Identifying whether there exist ranges of beta strand in the sequence
# Identified by the pattern of hydrophobicity value with respect to mean
def identify_beta_strands(hydro_val, hydro_mean):
    beta_strand_range = []
    i = 0
    while i < len(hydro_val):
        count = 0
        for j in range(i, len(hydro_val)-1):
            if (dist_from_mean(hydro_val[j], hydro_mean[j]) * dist_from_mean(
   hydro_val[j+1], hydro_mean[j+1])) < 0:
                count += 1
            else:
                if count >= 6:
                    beta_strand_range.append([i, j])
```

```
                i = j
                break
        if count >= 6:
            i = j
        else:
            i += 1
    return beta_strand_range
```

LISTING 5. Functions to compute α helix and β strand ranges

```
hydrophobicity =
{"A": 13.85, "D": 11.61, "C": 15.37, "E": 11.38, "F": 13.93,
"G": 13.34, "H": 13.82, "I": 15.28, "K": 11.58, "L": 14.13,
"M": 13.86, "N": 13.02, "P": 12.35, "Q": 12.61, "R": 13.10,
"S": 13.39, "T": 12.70, "V": 14.56, "W": 15.48, "Y": 13.88}

input_file = "Q1.fasta"
fasta_sequences = SeqIO.parse(open(input_file),'fasta')
for fasta in fasta_sequences:
    name, sequence = fasta.id, str(fasta.seq)
    window_length = 6
    hydrophobicity_val = []
    for i in range(len(sequence)):
        hydrophobicity_val.append(hydrophobicity[sequence[i]])

    mean = [np.mean(hydrophobicity_val)] * len(sequence)
    alpha_range, alpha_x, alpha_y = identify_alpha_helices(hydro_val, mean)
    beta_range, beta_x, beta_y = identify_beta_strands(hydro_val, mean)

    alpha_amphi_index = amphipathicity_alpha(mean[0], alpha_x, alpha_y)
    beta_amphi_index = amphipathicity_beta(beta_x, beta_y)
    print(alpha_amphi_index)
    print(beta_amphi_index)

    plt.figure(figsize=(15,5))
    plt.plot(hydrophobicity_val, color="green", linestyle='-', linewidth=2,
            label='Hydrophobicity value for each residue')
    plt.plot(mean, linestyle="dashed", color="black", linewidth=2,
            label='Hydrophobicity mean of sequence')
    for x in range(len(alpha_x)):
        plt.plot(alpha_x[x], alpha_y[x], linestyle="-", color="red",
                    linewidth=2, label='Alpha helix' if x == 0 else '')
    for x in range(len(beta_x)):
        plt.plot(beta_x[x], beta_y[x], linestyle="-", color="#0A1045",
                    linewidth=2, label='Beta strand' if x == 0 else '')
    for checker, x_range in enumerate(alpha_range)
        plt.axvspan(x_range[0], x_range[1], color='#F15152', alpha=0.3,
                    label="Alpha Helices" if checker == 0 else '')
    for checker, x_range in enumerate(beta_range):
        plt.axvspan(x_range[0], x_range[1], color='#3C91E6', alpha=0.3,
                    label="Beta Strands" if checker == 0 else '')
    plt.xlabel('Sequence Residue Number', fontsize=14)
    plt.ylabel('Hydrophobicity value', fontsize=14)
    plt.title('Hydrophobicity profile for the sequence', fontsize=16)
    plt.show()
```

LISTING 6. Code to compute the amphipathicity of α helices and β strands

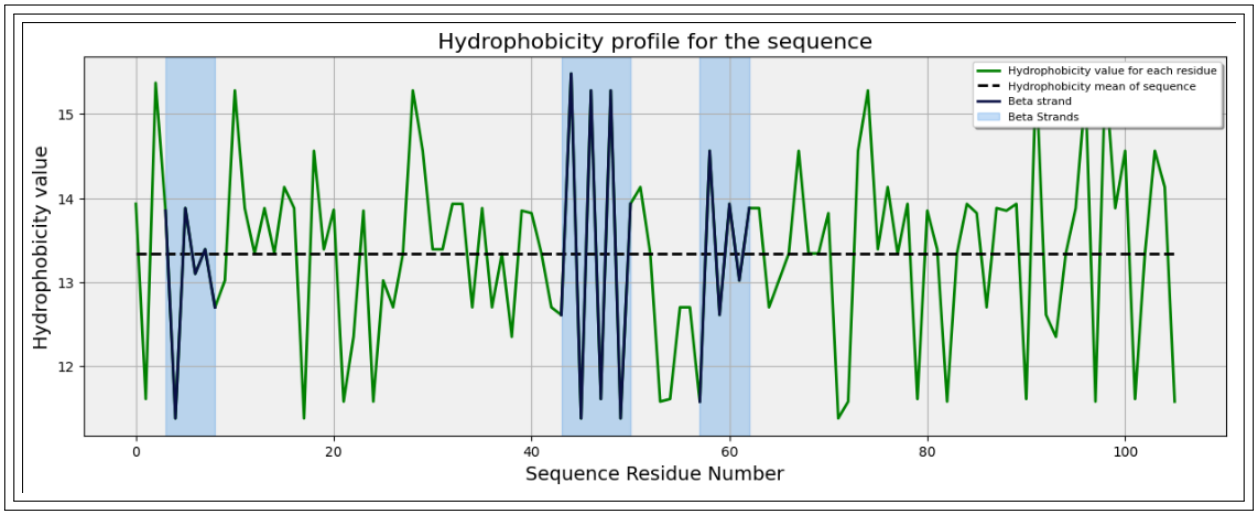

FIGURE 4. Hydrophobicity Profile of Sequence 1 in Q1.fasta

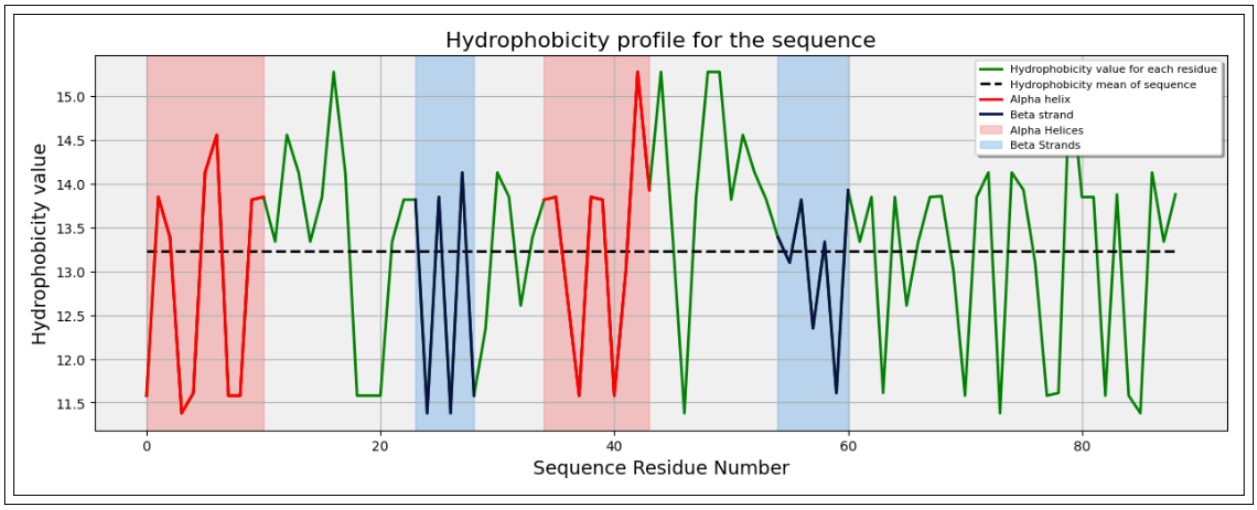

FIGURE 5. Hydrophobicity Profile of Sequence 2 in Q1.fasta

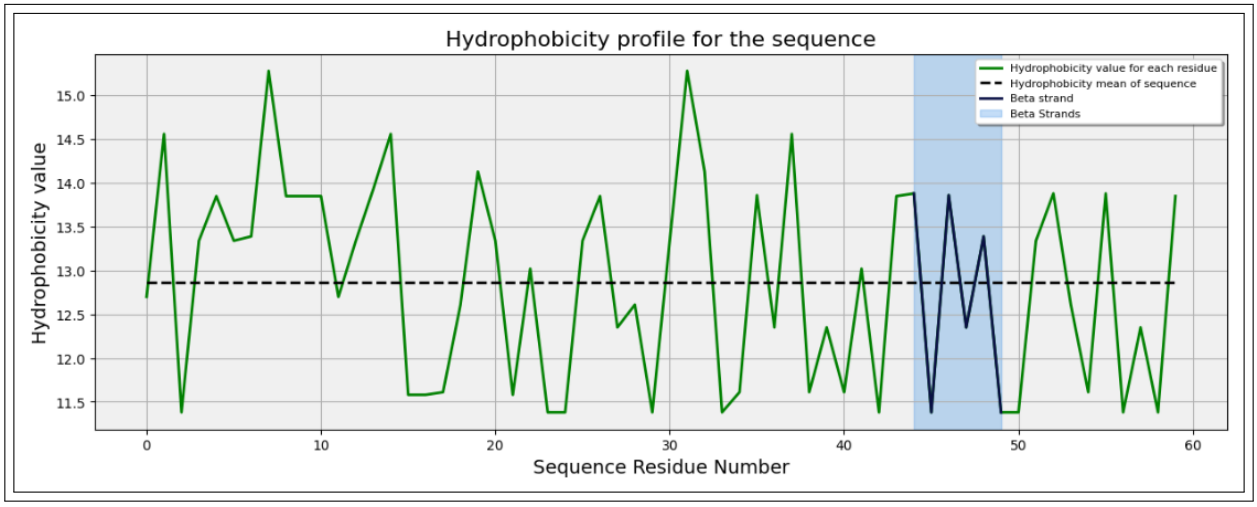

FIGURE 6. Hydrophobicity Profile of Sequence 3 in Q1.fasta

Much above is the code to compute the amphipathicity indices for the given constraints. Above are the hydrophobicity profiles for the given stretch lengths in the question.

In order to find the ranges of α helices, I have fixed the minimum consecutive amino acids, as given in the question, (i.e. the stretch length) as 8.
In order to find the ranges of β strands, I have fixed the minimum consecutive amino acids, as given in the question, (i.e. the stretch length) as 6.

Sequence No.	α helices	amphipathicity index	β strands	amphipathicity index
Sequence 1	-	-	(03, 08)	1.3133333333333361
	-	-	(43, 50)	3.2474999999999987
	-	-	(57, 62)	1.7200000000000042
Sequence 2	(00, 10)	4.796666666666667	(23, 28)	2.486666666666668
	(34, 43)	3.743333333333336	(54, 60)	1.2666666666666657
Sequence 3	-	-	(44, 49)	2.0066666666666677

Question 3. Plot the hydrophobicity profile for the sequence (Q2.fasta) with window lengths 9 and 19 and list the trans-membrane segments.

Solution. Firstly, I plotted the hydrophobicity profile for the sequence. Then, I smoothed the profile by considering the two scenarios of window lengths 9 and 19 given. This gave the modified hydrophobicity profile.
Below are the codes for computing the hydrophobicity profiles for different window sizes, along with the code for identifying trans-membrane segments.

```
# Function to identify the presence of trans-membrane segments
def identify_transmembrane(hydro_val, hydro_mean):
    transmembrane_range = []
    trans_values_x = []
    trans_values_y = []

    i = 0
    while i < len(hydro_val):
        count = 0
        for j in range(i, len(hydro_val)):
            if hydro_val[j] > hydro_mean[j]:
                count += 1
            else:
                if count >= 16:
                    transmembrane_range.append([i, j])
                    trans_values_x.append([x for x in range(i,j)])
                    trans_values_y.append([hydro_val[x] for x in range(i,j)])
                break
        i = j + 1

    return transmembrane_range, trans_values_x, trans_values_y

# Function to compute the average(smoothed) hydrophobicity value
def calculate_hydro(hydro_val, window):
    new_hydro_val = []

    for i in range(len(hydro_val)):
        start_point = max((i-window), 0)
        end_point = min((i+window+1), len(hydro_val))
        window_val = hydro_val[start_point:end_point]
        new_hydro_val.append(np.mean(window_val))
    return new_hydro_val
```

Listing 7. Functions to compute the trans-membrane segments

```
hydrophobicity =
{"A": 13.85, "D": 11.61, "C": 15.37, "E": 11.38, "F": 13.93,
"G": 13.34, "H": 13.82, "I": 15.28, "K": 11.58, "L": 14.13,
"M": 13.86, "N": 13.02, "P": 12.35, "Q": 12.61, "R": 13.10,
"S": 13.39, "T": 12.70, "V": 14.56, "W": 15.48, "Y": 13.88}

input_file = "Q2.fasta"
fasta_sequences = SeqIO.parse(open(input_file),'fasta')
for fasta in fasta_sequences:
    name, sequence = fasta.id, str(fasta.seq)
    hydrophobicity_val = []
    for i in range(len(sequence)):
        hydrophobicity_val.append(hydrophobicity[sequence[i]])

    # New hydrophobicity value list for window size 9
    new_hydrophobicity_val1 = calculate_hydro(hydrophobicity_val, 4)
    new_hydrophobicity_val1 =
                new_hydrophobicity_val1[4:len(new_hydrophobicity_val1)-4]

    # New hydrophobicity value list for window size 19
    new_hydrophobicity_val2 = calculate_hydro(hydrophobicity_val, 9)
    new_hydrophobicity_val2 =
                new_hydrophobicity_val2[9:len(new_hydrophobicity_val2)-9]
    m1 = [np.mean(new_hydrophobicity_val1)] * len(new_hydrophobicity_val1)
    m2 = [np.mean(new_hydrophobicity_val2)] * len(new_hydrophobicity_val2)

    t1_range, t1_x, t1_y = identify_transmembrane(new_hydrophobicity_val1,m1)
    t2_range, t2_x, t2_y = identify_transmembrane(new_hydrophobicity_val2,m2)

    plt.figure(figsize=(15,5))

    plt.plot(new_hydrophobicity_val1, color="green", linestyle='-',
            linewidth=2, label='Hydrophobicity value for each residue')

    plt.plot(m1, color="red", linestyle='dashed', linewidth=2,
            label=' Mean hydrophobicity value for given sequence')
    for x in range(len(t1_x)):
        plt.plot(t1_x[x], t1_y[x], linestyle="-", color="#0A1045",
                linewidth=2, label='Transmembrane segment' if x == 0 else '')
    for checker, x_range in enumerate(t1_range):
        plt.axvspan(x_range[0], x_range[1], color='#3C91E6', alpha=0.3,
                    label="Transmembrane segment" if checker == 0 else '')

    plt.xlabel('Sequence Residue Number', fontsize=14)
    plt.ylabel('Hydrophobicity value', fontsize=14)
    plt.title('Hydrophobicity profile for the sequence with window
            size = 9', fontsize=16)
    plt.grid(True)

    plt.legend(loc='upper right', fontsize=8, fancybox=True, shadow=True)
    plt.gca().set_facecolor('#f0f0f0')
    plt.show()

# Using the same ideology, we can plot for the window size 19 as well
```

LISTING 8. Code to make hydrophobicity profiles for window sizes 9 and 19

Now, in order to find the trans-membrane segments, I analyzed the modified hydrophobicity profile. If a minimum of 16 consecutive amino acids had hydrophobicity value greater than the mean, then they were considered as a trans-membrane segment. Below, I have tabulated the trans-membrane segments for the two cases.

Window size	Ranges of trans-membrane segments	Sequence indices in the segment
9	(27, 48)	(27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47)
	(106, 130)	(106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129)
	(171, 193)	(171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192)
19	(20, 42)	(20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41)
	(73, 92)	(73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91)
	(103, 130)	(103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129)
	(212, 234)	(212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233)

Below, I have enclosed the figures for the hydrophobicity profiles for the given two window sizes of 9 and 19 respectively. Also, the trans-membrane segments have been neatly highlighted. There are 3 and 4 trans-membrane segments for window sizes 9 and 19 respectively

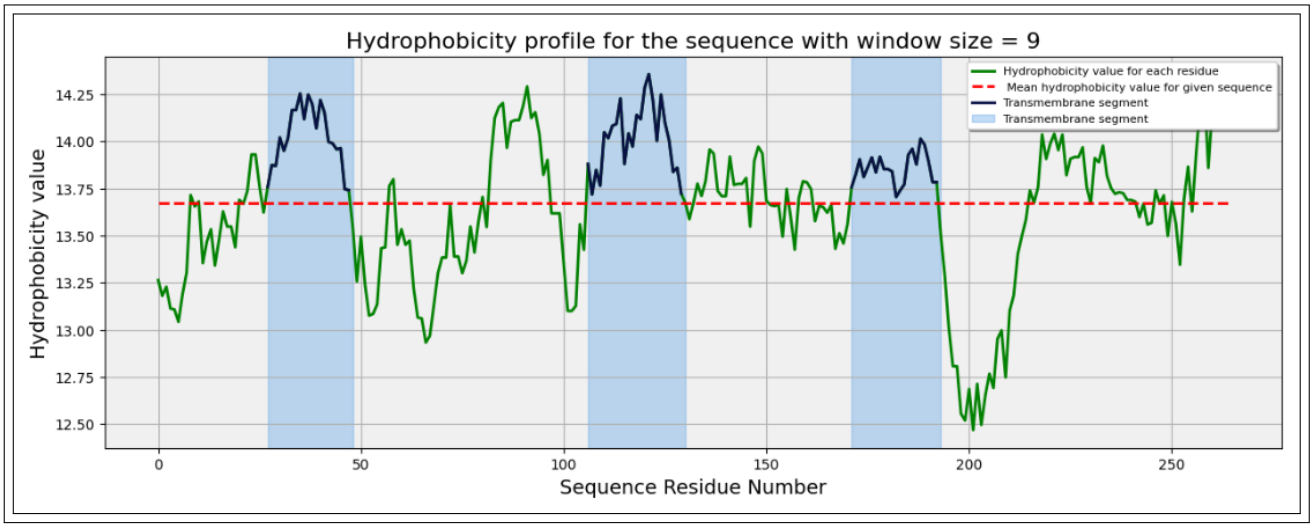

FIGURE 7. Hydrophobicity profile for window size 9

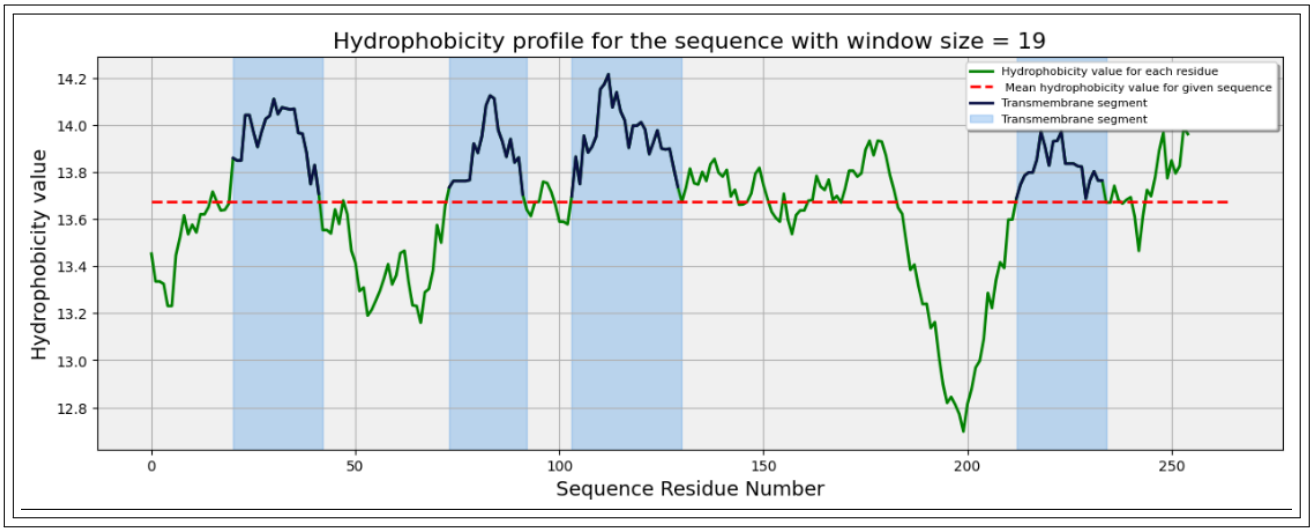

FIGURE 8. Hydrophobicity profile for window size 19

Question 4. Use **ScanProsite** tool (https://prosite.expasy.org/scanprosite/ - select option 2), to search for the patterns

(a) **[SV]-T-[VT]-[DERK](2)-IL**
(b) **[FILV]QxxxRKGxxx[RK]xx[FILVWY]**

in UniProtKB (Include Swiss-Prot, isoforms). List the number of matches for each pattern.

Solution. The ScanProsite tool allows you to scan proteins for matches against the PROSITE collection of motifs as well as against your own patterns.
It enables three options to choose from:

- Submit PROTEIN sequences to scan them against the PROSITE collection of motifs.
- Submit MOTIFS to scan them against a PROTEIN sequence database.
- Submit PROTEIN sequences and MOTIFS to scan them against each other.

We can also choose the database to match the motif or protein sequences against. It also gives us some additional options like:

- Exclude motifs with a high probability of occurrence from the scan
- Exclude profiles from the scan
- Run the scan at high sensitivity (show weak matches for profiles)

(a) The pattern is **[SV]-T-[VT]-[DERK](2)-IL**

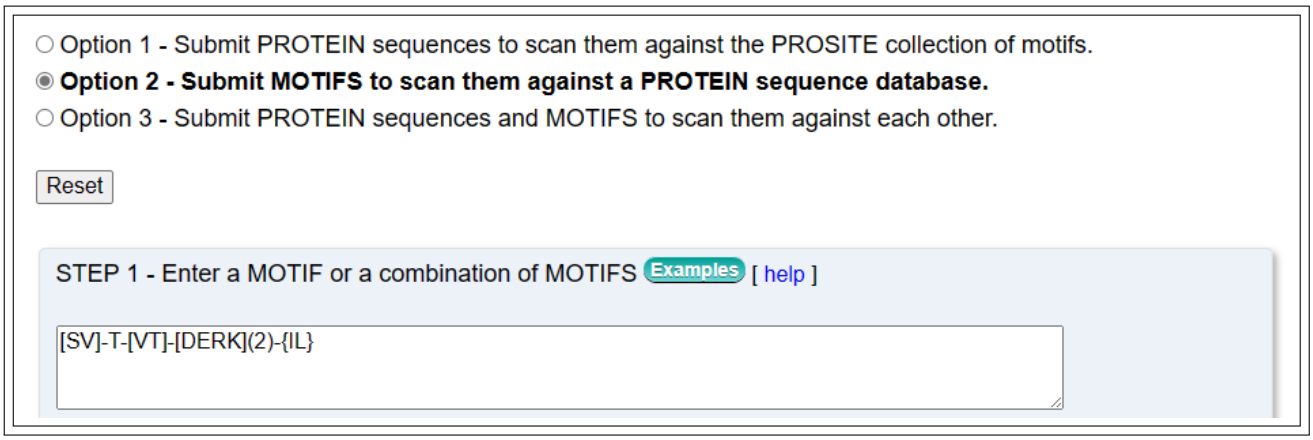

FIGURE 9. Input patter (a) to ScanProsite Tool

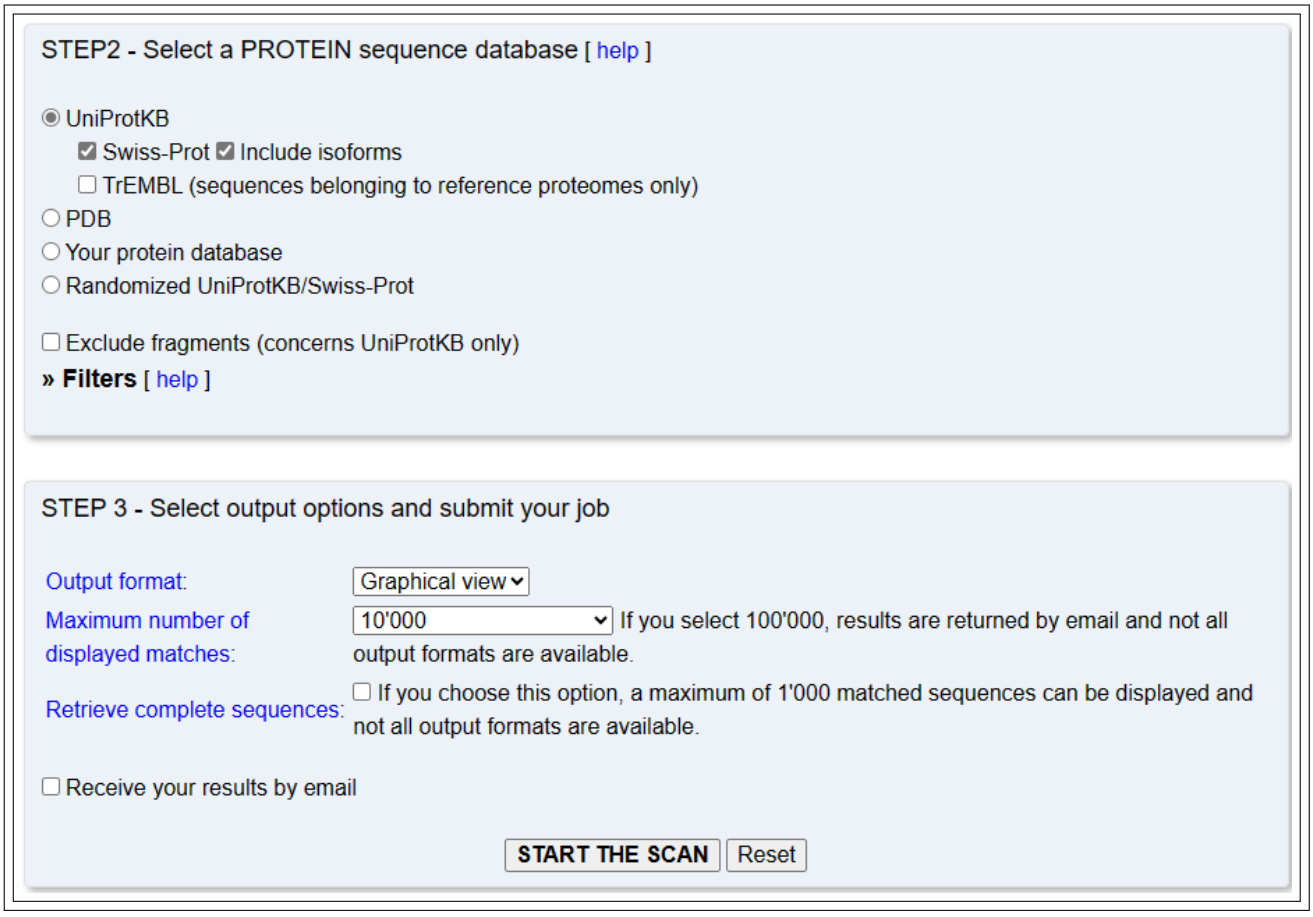

FIGURE 10. Additional input options (default chosen) for the ScanProsite tool

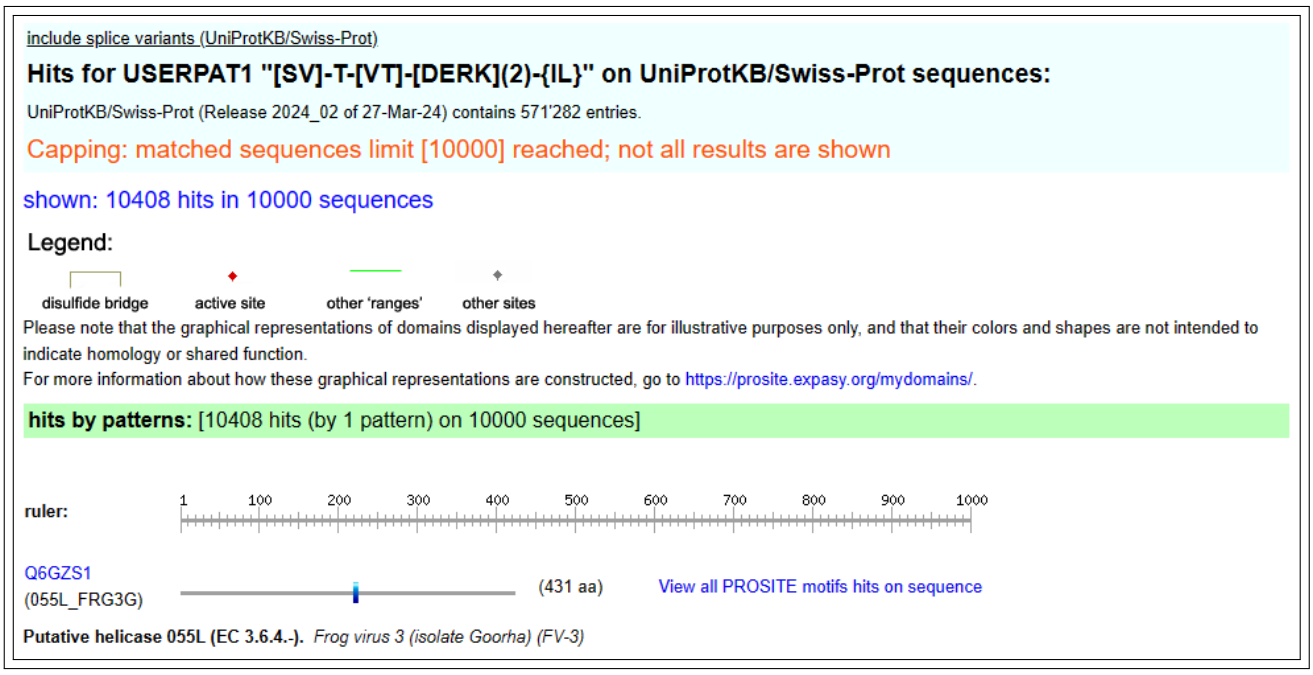

FIGURE 11. Output from ScanProsite tool (limited to 10,000)

Above is the output showing the number of hits when you limit maximum number of displayed matches as 10,000. It is seen that the output has been trimmed slightly (mentioned in the output). So, we need to extend the limit. Below is the output showing the number of hits when you extend the limit to 100,000.

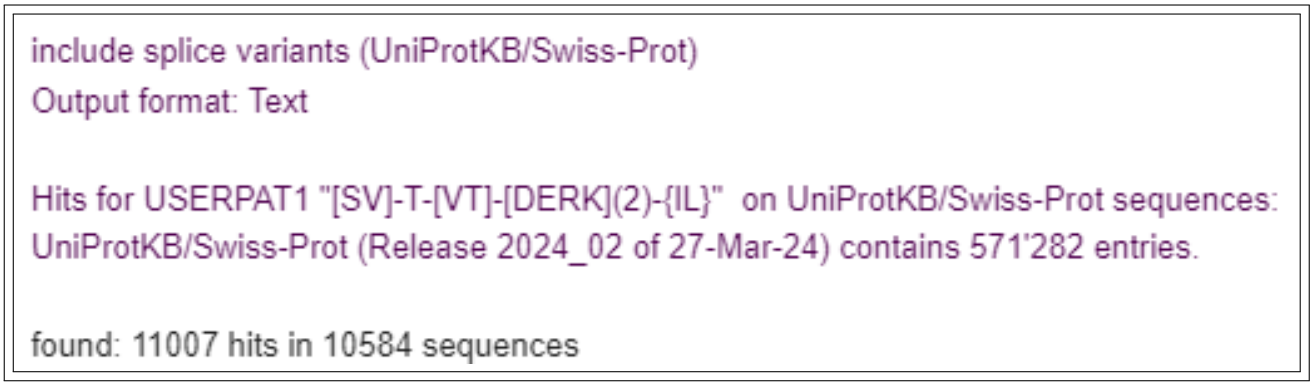

FIGURE 12. Output from ScanProsite tool (limited to 100,000)

(b) The pattern is [**FILV**]**QxxxRKGxxx**[**RK**]**xx**[**FILVWY**]

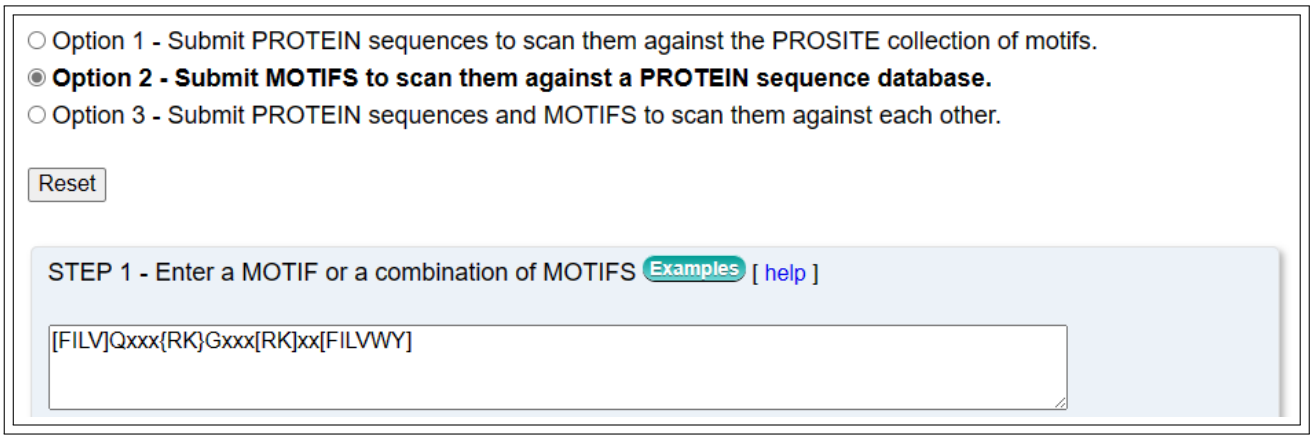

FIGURE 13. Input patter (b) to ScanProsite Tool

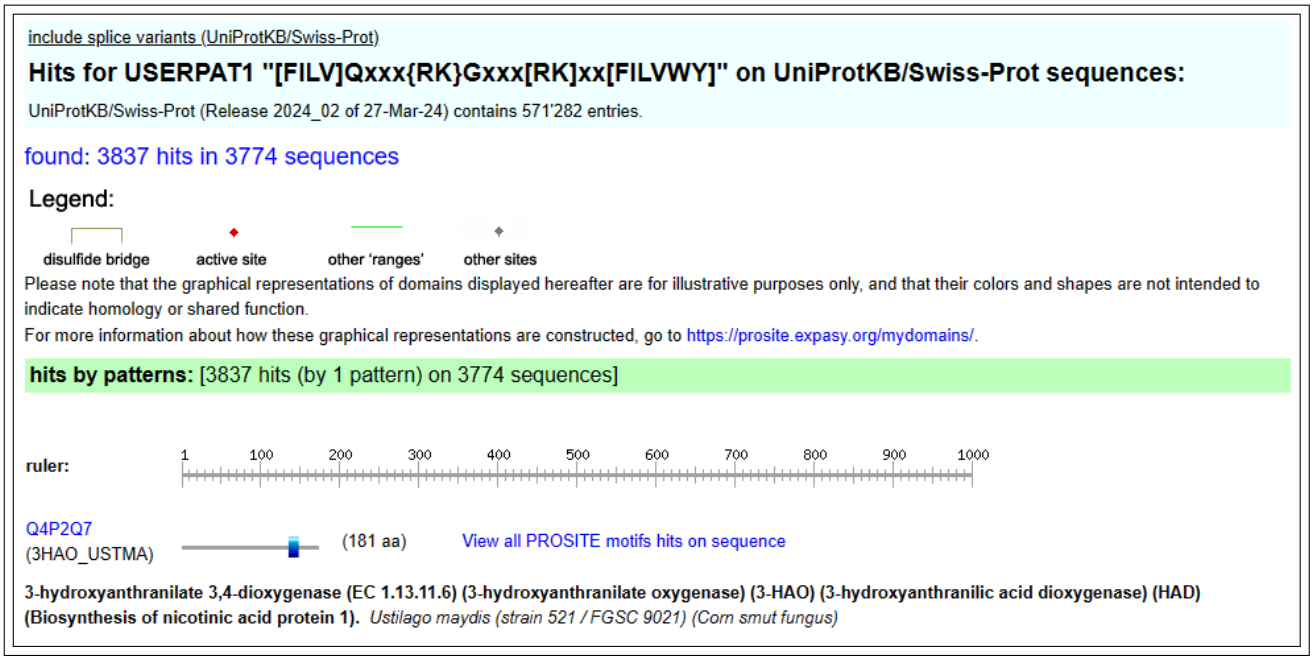

FIGURE 14. Output from ScanProsite tool (limited to 10,000)

Above is the output showing the number of hits when you limit maximum number of displayed matches as 10,000. It seems to fit well so no need to extend.

Results obtained from ScanProsite tool

(a) **[SV]-T-[VT]-[DERK](2)-IL**:
- 10,408 hits in 10,000 sequences (while capping maximum matches sequences to 10,000, **limit is reached**)
- 11,007 hits in 10,584 sequences (when capping maximum matches sequences to 100,000, **limit is not reached**)

(b) **[FILV]QxxxRKGxxx[RK]xx[FILVWY]**
- 3,837 hits in 3,774 sequences (when capping maximum matches sequences to 10,000, **limit is not reached**)

Question 5. Write a program to identify the patterns (refer Q4) in the sequence database (Q4.fasta). List the matches along with the sequence header and location of the matches in the sequence.

Solution. In order to match the patterns against the sequence database, I am going to use the **regex** library in python. The regex patterns for the above two cases in question 4 are:

(a) **[SV]T[VT][DERK]2[^IL]**
(b) **[FILV]Q.3[^RK]G.3[RK].2[FILVWY]**

There are occurrences of duplicated sequences in the FASTA file that have been removed to reduce redundancy. The python script enclosed below performs pattern matching on sequences from a FASTA file, extracts relevant information about the matches, and organizes this information into two pandas dataframes.

```
import re
from Bio import SeqIO

# Define your regex patterns

pattern1 = r'[SV]T[VT][DERK]{2}[^IL]'
pattern2 = r'[FILV]Q.{3}[^RK]G.{3}[RK].{2}[FILVWY]'

seq_desc1 = []
match_loc1 = []
match_ele1 = []

seq_desc2 = []
match_loc2 = []
match_ele2 = []

# Specify the path to your FASTA file
fasta_file = "Q4.fasta"
for record in SeqIO.parse(fasta_file, "fasta"):

    # Store the sequence and description line of FASTA separately
    record_id = record.description
    sequence = str(record.seq)

    # re function search for pattern matching
    matches1 = re.finditer(pattern1, sequence)
    matches2 = re.finditer(pattern2, sequence)

    # Iterate through pattern1 matching
    for match in matches1:
        # Note the start and end positions to store location
```

```
        start_pos = match.start()
        end_pos = match.end()
        matching_element = sequence[start_pos:end_pos]

        seq_desc1.append(record_id)
        match_loc1.append(str(start_pos) + "-" + str(end_pos))
        match_ele1.append(matching_element)

    # Iterate through pattern2 matching
    for match in matches2:
        # Note the start and end positions to store location
        start_pos = match.start()
        end_pos = match.end()
        matching_element = sequence[start_pos:end_pos]

        seq_desc2.append(record_id)
        match_loc2.append(str(start_pos) + "-" + str(end_pos))
        match_ele2.append(matching_element)

# Create datafraes for the two cases
df1 = pd.DataFrame({"Sequence Header":seq_desc1, "Match Element":match_ele1,
   "Match Location":match_loc1}, columns = ["Sequence Header", "Match Element
   ", "Match Location"])

df2 = pd.DataFrame({"Sequence Header":seq_desc2, "Match Element":match_ele2,
   "Match Location":match_loc2}, columns = ["Sequence Header", "Match Element
   ", "Match Location"])

df1 = df1.drop_duplicates().reset_index()
df1 = df1.drop("index", axis=1)
df1.head(len(df2))

df2 = df2.drop_duplicates().reset_index()
df2 = df2.drop("index", axis=1)
df2.head(len(df2))
```

LISTING 9. Code to find frequency of patterns in the given sequence database

Results from Regex pattern finding

(a) **[SV]-T-[VT]-[DERK](2)-IL**:
This pattern can be read as: Match a sequence where the first character is either S or V, followed by T, followed by either V or T, followed by exactly two occurrences of D, E, R, or K, and finally followed by any character except I or L.
Found 138 matches in the Q4.fasta

(b) **[FILV]QxxxRKGxxx[RK]xx[FILVWY]**:
This pattern can be read as: Match a sequence where the first character is either F, I, L, or V, followed by Q, followed by any character (except newline) exactly 3 times, followed by any character except R or K, followed by G, followed by any character (except newline) exactly 3 times, followed by either R or K, followed by any character (except newline) exactly 2 times, and finally followed by either F, I, L, V, W, or Y.
Found 7 matches in the Q4.fasta

	Sequence Header	Match Element	Match Location
0	4A0C_2|Chains C,E|CULLIN-4B|HOMO SAPIENS (9606)	STTERV	664-670
1	4A0K_1|Chain A|CULLIN-4A|HOMO SAPIENS (9606)	STTERV	665-671
2	5F0J_3|Chain C|Sorting nexin-3|Homo sapiens (9...	STVRRR	69-75
3	5F0L_3|Chain C|Sorting nexin-3|Homo sapiens (9...	STVRRR	69-75
4	5F0M_3|Chain C|Sorting nexin-3|Homo sapiens (9...	STVRRR	69-75
5	5F0P_3|Chain C|Sorting nexin-3|Homo sapiens (9...	STVRRR	69-75
6	5N69_1|Chains A,B|Myosin-7|Bos taurus (9913)	VTVKED	68-74
7	5N6A_1|Chain A|Myosin-7|Bos taurus (9913)	VTVKED	68-74
8	5TBY_1|Chains A,B|Myosin-7|Homo sapiens (9606)	VTVKED	68-74
9	6FSA_1|Chains A,B|Myosin-7|Bos taurus (9913)	VTVKED	68-74
10	6X5Z_3|Chains D,G,J|Myosin-7|Bos taurus (9913)	VTVKED	68-74
11	7JH7_2|Chains F,G,H|Myosin-7|Sus scrofa (9823)	VTVKED	68-74
12	1A8J_1|Chains H,L|IMMUNOGLOBULIN LAMBDA LIGHT ...	STVEKT	203-209
13	1ADQ_2|Chain L|IGM-LAMBDA RF-AN FAB (LIGHT CHA...	STVEKT	200-206
14	1AIV_1|Chain A|OVOTRANSFERRIN|Gallus gallus (9...	STVEEN	542-548
15	1AQK_1|Chain L|FAB B7-15A2|Homo sapiens (9606)	STVEKT	203-209
16	1BJM_1|Chains A,B|LOC - LAMBDA 1 TYPE LIGHT-CH...	STVEKT	203-209
17	1DCL_1|Chains A,B|MCG|Homo sapiens (9606)	STVEKT	203-209
18	1DPU_1|Chain A|REPLICATION PROTEIN A (RPA32) C...	STVDDD	85-91
19	1E8I_1|Chains A,B|EARLY ACTIVATION ANTIGEN CD6...	STVKRS	18-24
20	1FM2_2|Chain B|GLUTARYL 7-AMINOCEPHALOSPORANIC...	STVDKP	111-117
21	1FM5_1|Chain A|EARLY ACTIVATION ANTIGEN CD69|H...	STVKRS	39-45
22	1FNT_15|Chains c,d,e,f,g,h,i,j,k,l,m,n,o,p|PRO...	STVEDK	152-158
23	1FO9_1|Chain A|ALPHA-1,3-MANNOSYL-GLYCOPROTEIN...	STVRRC	18-24
24	1FOA_1|Chain A|ALPHA-1,3-MANNOSYL-GLYCOPROTEIN...	STVRRC	18-24
25	1GPJ_1|Chain A|Glutamyl-tRNA reductase|Methano...	STVEEE	315-321
26	1IQ7_1|Chain A|Ovotransferrin|Gallus gallus (9...	STVEEN	201-207
27	1JT1_1|Chain A|FEZ-1, class B3 metallo-beta-la...	STVDKV	121-127
28	1JVK_1|Chains A,B|IMMUNOGLOBULIN LAMBDA LIGHT ...	STVEKT	204-210
29	1JVZ_2|Chain B|cephalosporin acylase beta chai...	STVDKP	111-117
30	1K07_1|Chains A,B|FEZ-1 beta-lactamase|Fluorib...	STVDKV	121-127
31	1KEH_1|Chain A|precursor of cephalosporin acyl...	STVDKP	280-286
32	1KVD_2|Chains B,D|SMK TOXIN|Pichia farinosa (4...	STVKDG	45-51
33	1KVE_2|Chains B,D|SMK TOXIN|Pichia farinosa (4...	STVKDG	45-51
34	1L9Y_1|Chains A,B|FEZ-1 b-lactamase|Fluoribact...	STVDKV	121-127
35	1LGV_1|Chains A,B|IMMUNOGLOBULIN LAMBDA LIGHT ...	STVEKT	204-210
36	1LHZ_1|Chains A,B|IMMUNOGLOBULIN LAMBDA LIGHT ...	STVEKT	204-210
37	1LIL_1|Chains A,B|LAMBDA III BENCE JONES PROTE...	STVEKT	199-205
38	1MCB_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
39	1MCC_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
40	1MCE_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
41	1MCF_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
42	1MCH_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
43	1MCJ_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
44	1MCL_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
45	1MCO_1|Chain L|IGG1 MCG INTACT ANTIBODY (LIGHT...	STVEKT	203-209

FIGURE 15. 1-46 matches for pattern1

	Sequence Header	Match Element	Match Location
46	1MCR_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
47	1MCS_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
48	1NL0_1|Chain L|anti-factor IX antibody, 10C12,...	STVEKT	203-209
49	1OVT_1|Chain A|OVOTRANSFERRIN|Gallus gallus (9...	STVEEN	542-548
50	1Q1J_1|Chains L,M|Fab 447-52D, light chain|Hom...	STVEKT	204-210
51	1R5M_1|Chain A|SIR4-interacting protein SIF2|S...	STVERE	11-17
52	1RP1_1|Chain A|PANCREATIC LIPASE RELATED PROTE...	STVRED	436-442
53	1RYX_1|Chain A|Ovotransferrin|Gallus gallus (9...	STVEEN	542-548
54	1RZF_1|Chain L|Fab E51 light chain|Homo sapien...	STVEKT	203-209
55	1S5J_1|Chain A|DNA polymerase I|Sulfolobus sol...	STVKKA	603-609
56	1UYP_1|Chains A,B,C,D,E,F|BETA-FRUCTOSIDASE|TH...	STVEDE	370-376
57	1VQT_1|Chain A|Orotidine 5'-phosphate decarbox...	STVERS	71-77
58	1W72_5|Chains L,M|HYB3 LIGHT CHAIN|HOMO SAPIEN...	STVEKT	201-207
59	1WF5_1|Chain A|sidekick 2 protein|Homo sapiens...	STVERR	28-34
60	1Z1D_1|Chain A|Replication protein A 32 kDa su...	STVDDD	89-95
61	1Z7Q_15|Chains c,d,e,f,g,h,i,j,k,l,m,n,o,p|pro...	STVEDK	152-158
62	1ZTM_1|Chains A,B,C|Fusion glycoprotein|Human ...	STVDKY	227-233
63	1ZVO_1|Chains A,B|myeloma immunoglobulin D lam...	STVEKT	201-207
64	2APC_1|Chain A|Alpha-1,3-mannosyl-glycoprotein...	STVRRC	12-18
65	2B0S_1|Chain L|Fab 2219, light chain|Homo sapi...	STVEKT	205-211
66	2B1A_1|Chain L|Fab 2219, light chain|Homo sapi...	STVEKT	205-211
67	2B1H_1|Chain L|Fab 2219, light chain|Homo sapi...	STVEKT	205-211
68	2BB0_1|Chains A,B|Imidazolonepropionase|Bacill...	STVKDT	113-119
69	2BB5_1|Chains A,B|Transcobalamin II|Homo sapie...	STVEDV	334-340
70	2DD8_2|Chain L|IGG Light Chain|Homo sapiens (9...	STVEKT	200-206
71	2DVV_1|Chain A|Bromodomain-containing protein ...	STVKRK	53-59
72	2E3K_1|Chains A,B,C,D|Bromodomain-containing p...	STVKRK	53-59
73	2E7N_1|Chain A|Bromodomain-containing protein ...	STVKRK	60-66
74	2ES7_1|Chains A,B,C,D|putative thiol-disulfide...	STVDDW	24-30
75	2FB4_1|Chain L|IGG1-LAMBDA KOL FAB (LIGHT CHAI...	STVEKT	203-209
76	2FH6_1|Chain A|pullulanase|Klebsiella aerogene...	STVEEV	521-527
77	2FH8_1|Chain A|pullulanase|Klebsiella aerogene...	STVEEV	521-527
78	2FHB_1|Chain A|pullulanase|Klebsiella aerogene...	STVEEV	521-527
79	2FHC_1|Chain A|pullulanase|Klebsiella aerogene...	STVEEV	521-527
80	2FHF_1|Chain A|pullulanase|Klebsiella aerogene...	STVEEV	521-527
81	2FL5_1|Chains A,C,E,L|Immunoglobulin Igg1 Lamb...	STVEKT	199-205
82	2G3F_1|Chains A,B|Imidazolonepropionase|Bacill...	STVKDT	113-119
83	2G4A_1|Chain A|Bromodomain-containing protein ...	STVKRK	49-55
84	2G75_2|Chains B,D|IGG Light Chain|Homo sapiens...	STVEKT	200-206
85	2GAN_1|Chains A,B|182aa long hypothetical prot...	STVKDE	10-16
86	2H32_2|Chain B|Immunoglobulin omega chain|Homo...	STVEKT	108-114
87	2H3N_2|Chains B,D|Ig lambda-5|Homo sapiens (9606)	STVEKT	107-113
88	2IDR_1|Chains A,B|Eukaryotic translation initi...	STVEDF	34-40
89	2IDV_1|Chain A|Eukaryotic translation initiati...	STVEDF	34-40
90	2IG2_1|Chain L|IGG1-LAMBDA KOL FAB (LIGHT CHAI...	STVEKT	203-209
91	2J28_23|Chain O|50S RIBOSOMAL PROTEIN L18|ESCH...	STVEKA	51-57

FIGURE 16. 46-92 matches for pattern1

	Sequence Header	Match Element	Match Location
92	2J42_1|Chain A|C2 TOXIN COMPONENT-II|CLOSTRIDI...	STVDDT	337-343
93	2J6E_3|Chains L,M|IGM|HOMO SAPIENS (9606)	STVEKT	222-228
94	2JB5_2|Chain L|FAB FRAGMENT MOR03268 LIGHT CHA...	STVEKT	204-210
95	2JB6_1|Chains A,L|FAB FRAGMENT MOR03268 LIGHT ...	STVEKT	204-210
96	2JE8_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	VTTERY	348-354
97	2JE8_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	STVKEF	474-480
98	2JQ3_1|Chain A|Apolipoprotein C-III|Homo sapie...	STVKDK	54-60
99	2KC8_1|Chain A|Toxin relE|Escherichia coli (83...	STVREQ	22-28
100	2KC9_1|Chain A|Toxin relE|Escherichia coli (83...	STVREQ	22-28
101	2LDX_1|Chains A,B,C,D|APO-LACTATE DEHYDROGENAS...	STVKEQ	0-6
102	2MCG_1|Chains 1,2|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
103	2MXC_1|Chain A|Sorting nexin-3|Homo sapiens (9...	STVRRR	74-80
104	2OAJ_1|Chain A|Protein SNI1|Saccharomyces cere...	STVEKN	752-758
105	2OAJ_1|Chain A|Protein SNI1|Saccharomyces cere...	STVKEQ	824-830
106	2OLD_1|Chains A,B|Bence Jones KWR Protein - Im...	STVEKT	204-210
107	2OMB_1|Chains A,B,C,D|Bence Jones KWR Protein ...	STVEKT	204-210
108	2OMN_1|Chains A,B|Bence Jones KWR Protein - Im...	STVEKT	204-210
109	2PI2_1|Chains A,B,C,D|Replication protein A 32...	STVDDD	256-262
110	2QA2_1|Chain A|Polyketide oxygenase CabE|Strep...	STVRKA	165-171
111	2RCJ_2|Chains C,D,G,H,K,L,O,P,S,T|IgA1 heavy c...	STVEKT	201-207
112	2RDO_14|Chain O|50S ribosomal protein L18|Esch...	STVEKA	51-57
113	2RDO_34|Chain 8|Ribosome recycling factor|Esch...	VTVEDS	56-62
114	2VJX_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	VTTERY	346-352
115	2VJX_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	STVKEF	472-478
116	2VL4_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	VTTERY	346-352
117	2VL4_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	STVKEF	472-478
118	2VMF_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	VTTERY	346-352
119	2VMF_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	STVKEF	472-478
120	2VO5_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	VTTERY	346-352
121	2VO5_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	STVKEF	472-478
122	2VOT_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	VTTERY	346-352
123	2VOT_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	STVKEF	472-478
124	2VQT_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	VTTERY	346-352
125	2VQT_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	STVKEF	472-478
126	2VQU_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	VTTERY	346-352
127	2VQU_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	STVKEF	472-478
128	2VR4_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	VTTERY	346-352
129	2VR4_1|Chains A,B|BETA-MANNOSIDASE|BACTEROIDES...	STVKEF	472-478
130	1AA0_1|Chain A|FIBRITIN|Enterobacteria phage T...	STVEER	40-46
131	1FO8_1|Chain A|ALPHA-1,3-MANNOSYL-GLYCOPROTEIN...	STVRRC	13-19
132	1JKM_1|Chains A,B|BREFELDIN A ESTERASE|Bacillu...	STVRDV	344-350
133	1JW0_2|Chain B|cephalosporin acylase beta chai...	STVDKP	111-117
134	1MCD_1|Chains A,B|Immunoglobulin lambda-1 ligh...	STVEKT	203-209
135	1MCI_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
136	1MCK_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209
137	1MCN_1|Chains A,B|IMMUNOGLOBULIN LAMBDA DIMER ...	STVEKT	203-209

FIGURE 17. 92-138 matches for pattern1

	Sequence Header	Match Element	Match Location
0	4FXG_2|Chains B,E|Complement C4-A alpha chain|...	LQIEKEGAIHREEL	251-265
1	4FXK_2|Chain B|Complement C4-A Alpha chain|Hom...	LQIEKEGAIHREEL	251-265
2	4XAM_2|Chains C,E|Complement C4-A|Homo sapiens...	LQIEKEGAIHREEL	174-188
3	5JPM_2|Chains B,E|Complement C4-A|Homo sapiens...	LQIEKEGAIHREEL	251-265
4	5JPN_2|Chain B|Complement C4-A|Homo sapiens (9...	LQIEKEGAIHREEL	251-265
5	5JTW_2|Chains B,E|Complement C4-A|Homo sapiens...	LQIEKEGAIHREEL	174-188
6	6YSQ_1|Chains A,B|Complement C4-B,Complement C...	LQIEKEGAIHREEL	853-867

FIGURE 18. 1-7 matches for pattern2

Question 6. Identify the beta barrel membrane proteins with the following pattern: [K,R,H,Q,F,E]-x-G-[I,V,L,F,A,C]-x-[I,V,L,F,M,Y,W]-x-[I,V,L,F,W]
Use:
http://www.bioinformatics.org/sms2/protein_pattern.html,
http://prosite.expasy.org/scanprosite/
Hint: Modify the patterns according to the input format of the server.

Solution.

- Sequence Manipulation Suite: Protein Pattern Find (http://www.bioinformatics.org/sms2/protein_pattern.html):

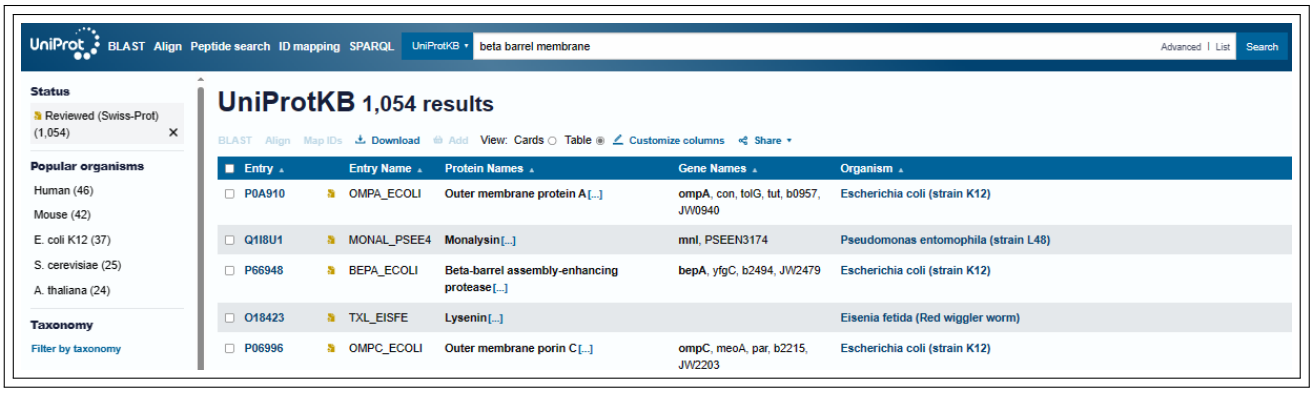

FIGURE 19. "Beta barrel membrane" proteins in UniProtKB

Firstly, a FASTA file that contains all the beta barrel membrane proteins in SwissProt is downloaded. The image for the same is given above.

In this Sequence Manipulation Suite, the modified pattern is **[KRHQFE].G[IVLFAC].[IVLFMYW].[IVLFW]** to work well with the server.

The FASTA file is then given as input to the Sequence Manipulation Suite, and submitted.

Sequence Manipulation Suite Output

495 hits were found. The results are shown below:

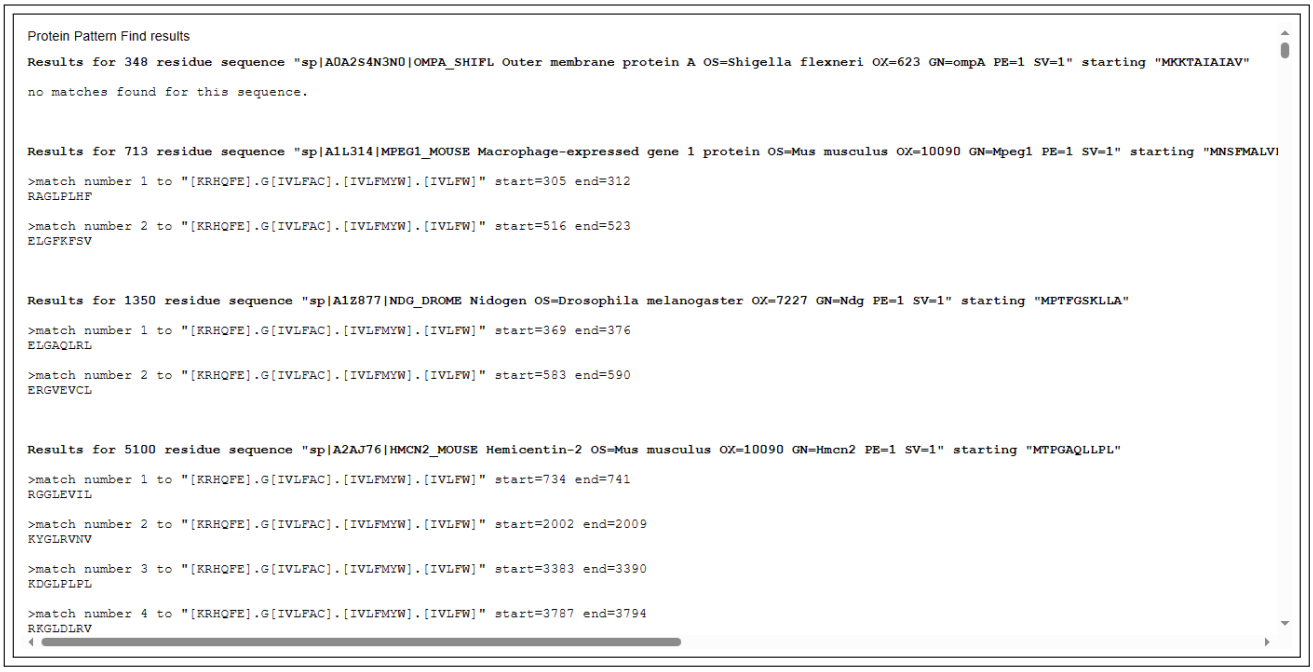

FIGURE 20. Sequence Manipulation Suite output

- ScanProsite tool (http://prosite.expasy.org/scanprosite/):

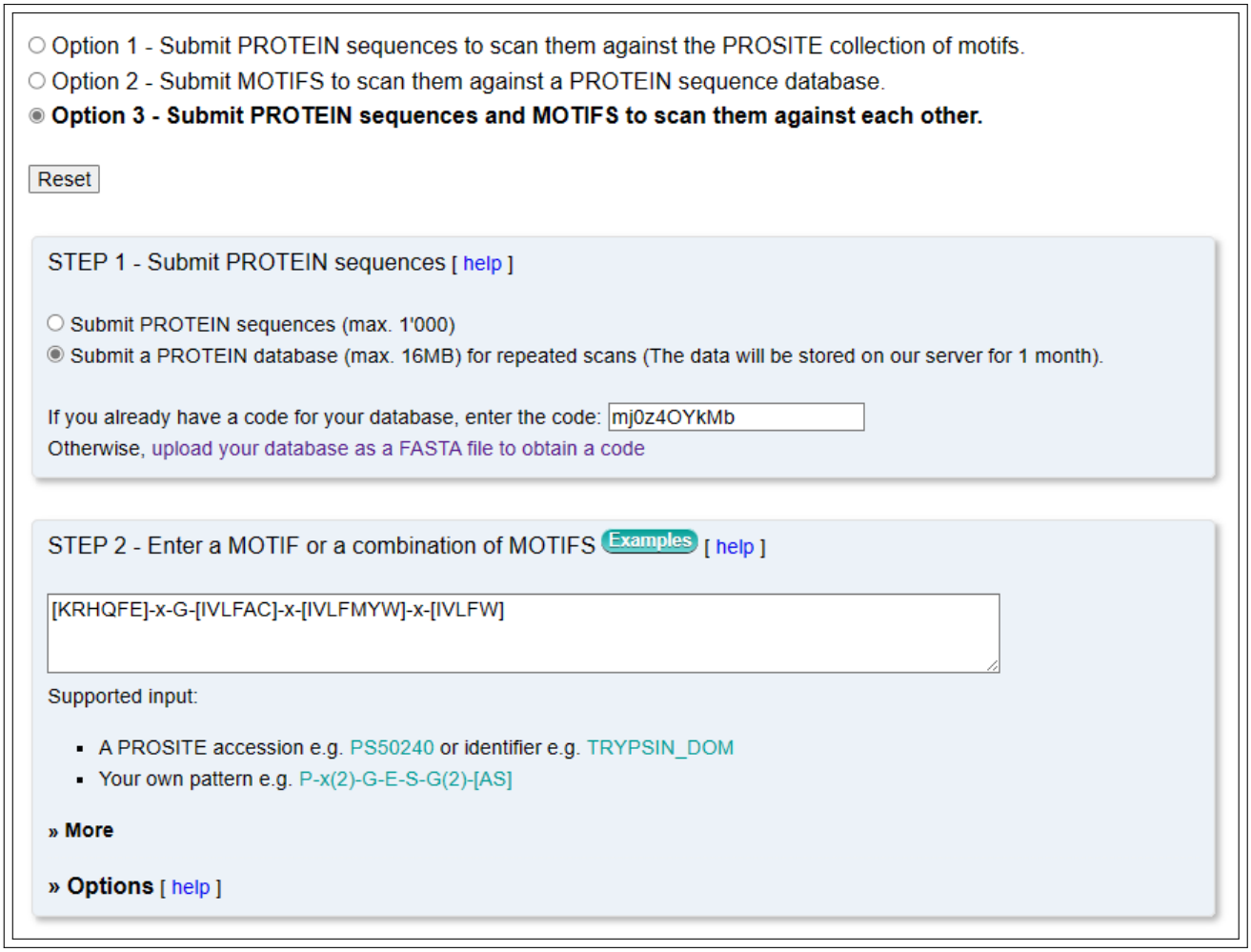

FIGURE 21. Input to ScanProsite tool

Firstly, a FASTA file that contains all the beta barrel membrane proteins in SwissProt is downloaded. The image for the same was enclosed in the previous part. Now, using the ScanProsite interface, I upload my database to obtain the code for the database to be given as input to the tool.

In this ScanProsite Tool, the modified pattern is

[KRHQFE]-x-G-[IVLFAC]-x-[IVLFMYW]-x-[IVLFW] to work well with the server.

Option 3 is selected for the purpose. The code of the generated FASTA file, alongwith the above modified pattern is then given as input to the ScanProsite tool.

Sequence Manipulation Suite Output

495 hits were found in 350 sequences. The results are shown below:

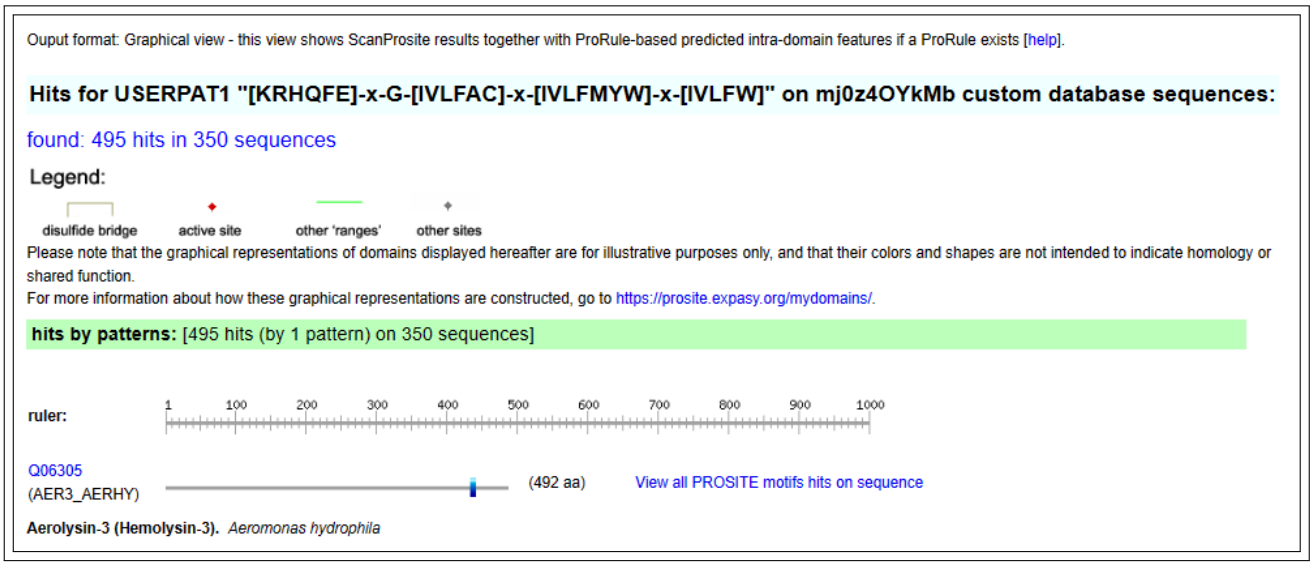

FIGURE 22. Results of ScanProsite tool