

## PRACTICAL 1

**Question 1.** Download the EMBOSS package (**Download EMBOSS**) and copy it to your Windows system. In the case of Linux, use the command: `sudo apt-get install jemboss`. For Mac users, use the online tool links given in the instructions document.

**Solution.** The EMBOSS package has been downloaded and installed on the Windows system.



FIGURE 1. The logo of the Emboss software

**Question 2.** Using EMBOSS, find the complementary strand for the sequence:  
**CTCGGATTTGTAAAGATCATGATCTCATACATAGTACCTAGCCATTG**

**Solution.** On the Emboss software, we go to the **revseq** tool. It reads nucleotide sequences and computes the reverse complement (anti-sense) of each sequence. It can also output just the reversed sequence or just the complement of the sequence.

Given a DNA sequence, reading from 5' to 3', its complementary strand from 5' to 3' will basically be the **reverse-complement** counterpart of the DNA sequence.

**REVSEQ**  
Reverse and complement a nucleotide sequence

**input section**  
Enter the 'seqall' input as:  
☐ file/emboss-query or ☒ paste or ☐ list of files

**Cut and Paste**  
CTCGGATTTGTAAAGATCATGATCTCATACATAGTACCTAGCCATTG

FIGURE 2. Input to the revseq tool

The complementary strand for the given sequence, in the 5' to 3' direction, as obtained from the **revseq** tool is:

**CAATGGCTAGGTACTATGTATGAGATCATGATCTTTACAAATCCGAG**

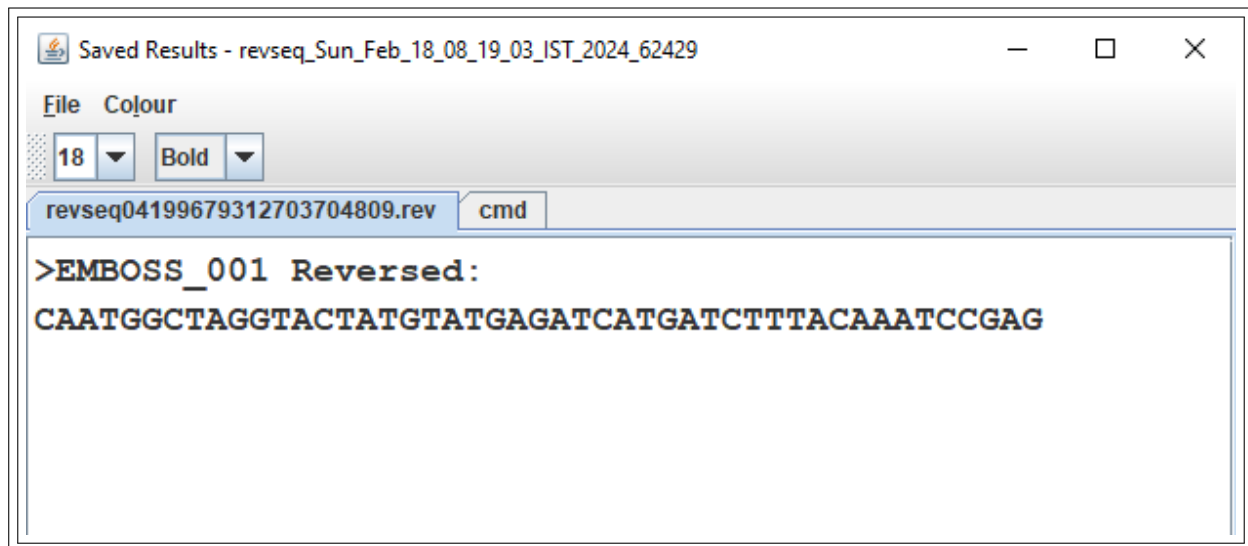


FIGURE 3. Output of the revseq tool

**Question 3.** Write a program to find the complementary strand for the sequence given in Q2.

```

1 def complementary(seq):
2     """Given a DNA string, find the reverse complement of the DNA string
3
4     Args:
5         seq (str): Takes the given DNA string as input
6
7     Returns:
8         str: Returns the reverse complement of the DNA strand
9     """
10    # Best way to take the reverse of a string
11    seq = seq[::-1]
12    ans = ""
13    # dictionary with the complement of each key
14    complement = {"A": "T", "T": "A", "G": "C", "C": "G"}
15    for i in seq:
16        ans = ans + complement[i]
17
18    return ans
19
20 s = "CTCGGATTTGTAAAGATCATGATCTCATACATAGTACCTAGCCATTG"
21 print(complementary(s))

```

LISTING 1. Reverse Complement

The output of the above code, in 5' to 3' direction, when s is passed to the function, is:  
**CAATGGCTAGGTACTATGTATGAGATCATGATCTTTACAAATCCGAG**

**Question 4.** Using EMBOSS,

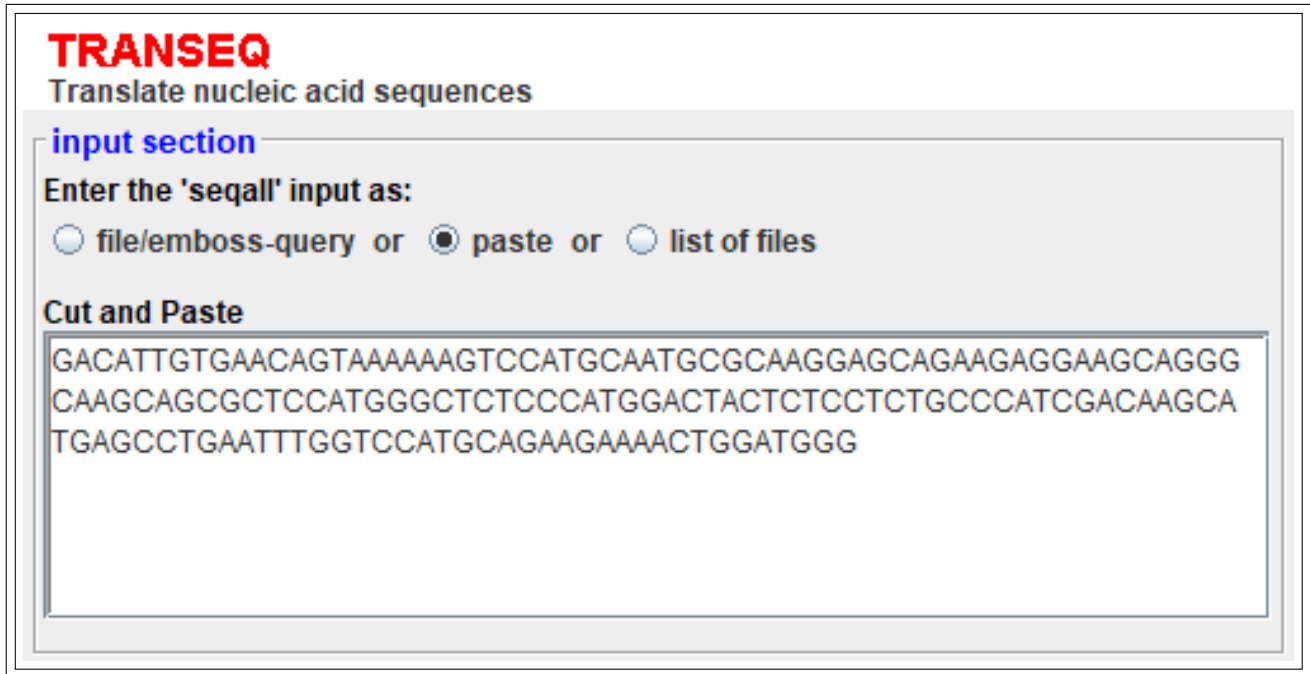
(i) Find the protein sequence for:

**GACATTGTGAACAGTAAAAAAGTCCATGCAATGCGCAAGGAGCAGAAGA  
GGAAGCAGGGCAAGCAGCGCTCCATGGGCTCTCCCATGGACTACTCTCC  
TCTGCCCATCGACAAGCATGAGCCTGAATTTGGTCCATGCAGAAGAAAA  
CTGGATGGG**

(ii) Identify the reading frame equivalent to the following sequence:

**PIQFSSAWTKFRLMLVDGQRRVVHGRAHGALLALLPLLLLAHCMDFFTVHNV**

**Solution.** (i) On the Emboss software, we go to the **transeq** tool. It reads the nucleotide sequence and computes the corresponding protein sequence translations. It can translate in any of the 3 forward or 3 reverse sense frames, or in all six frames. It can be translated using the standard **Universal genetic code** and also with a selection of non-standard codes. The protein sequence for the given nucleotide sequence is:  
**DIVNSKKVHAMRKEQKRKQGKQSRMSGSPMDYSPLPIDKHEPEFGPCRRKLDG**



**TRANSEQ**  
Translate nucleic acid sequences

input section

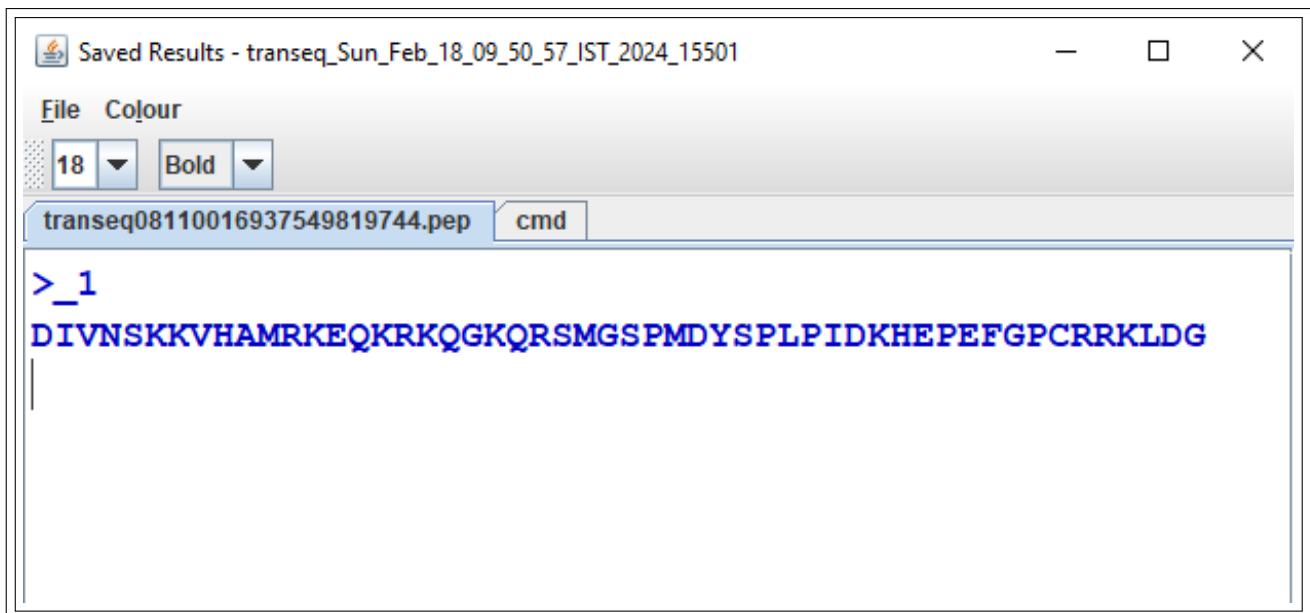
Enter the 'seqall' input as:

☐ file/emboss-query or ☒ paste or ☐ list of files

Cut and Paste

```
GACATTGTGAACAGTAAAAAGTCCATGCAATGCGCAAGGAGCAGAAGAGGAAGCAGGG
CAAGCAGCGCTCCATGGGCTCTCCCATGGACTACTCTCCTCTGCCCATCGACAAGCA
TGAGCCTGAATTTGGTCCATGCAGAAGAAAAGTGGATGGG
```

FIGURE 4. Input to the transeq tool



Saved Results - transeq\_Sun\_Feb\_18\_09\_50\_57\_IST\_2024\_15501

File Colour

18 Bold

transeq08110016937549819744.pep cmd

```
>_1
DIVNSKKVHAMRKEQKRKQGKQSRMSGSPMDYSPLPIDKHEPEFGPCRRKLDG
```

FIGURE 5. Output of the transeq tool

(ii) In the second part of the question, given the protein sequence comprising amino acids, the corresponding reading frame equivalent has to be found. On the Emboss tool, we go to the **transeq** tool. For the sequence given in 4(i), we print the translated protein output for all 6 frames. Then we compare the protein sequences of each frame with the one given in 4(ii).

**TRANSEQ**  
 Translate nucleic acid sequences

☐ file/emboss-query or ☒ paste or ☐ list of files

Cut and Paste

GACATTGTGAACAGTAAAAAGTCCATGCAATGCGCAAGGAGCAGAAGAGGAAGCAGGG  
 CAAGCAGCGCTCCATGGGCTCTCCCATGGACTACTCTCCTCTGCCCATCGACAAGCA  
 TGAGCCTGAATTTGGTCCATGCAGAAGAAACTGGATGGG

FIGURE 6. Input to the transeq tool

```

>_1
DIVNSKKVHAMRKEQKRKQKGKQSRMSGSPMDYSPLPIDKHEPEFGPCRRKLDG
>_2
TL*TVKKSMMQCARSRGRSRASSAPWALPWTTLLCPSTSMSLNLVHAEENWMG
>_3
HCEQ*KSPCNAQGAEAEAGQAALHGLSHGLLSSAHRQA*A*IWSMQKKTGWX
>_4
PIQFSSAWTKFRLMLVDGQRRVVHGRAHGALLALLPLLLLAHCMDFFTVHNV
>_5
PSSFLLHGPNSSGCLSMGRGE*SMGEPMERCLPCFLFCSLRIAWTFLLFTMS
>_6
HPVFFCMDQIQAHACRWAEESSPWESPWSAACPASSAPCALHGLFYCSQCX
  
```

FIGURE 7. Output of the transeq tool for all reading frames

The above Figure 7 is the output of the **transeq** tool with nucleotide sequence as input. We observe that the protein sequence corresponding to **4** matches the protein sequence in 4(ii). This implies that the reading frame equivalent to the protein sequence is **4** or **-1**, which is the reverse complement (or) complementary strand of the given nucleotide sequence in Q4.

**Question 5.** Write a program to translate the given DNA sequence (refer to Q4) to the protein sequence.

```

1 # Dictionary to address the edge cases when left with last two nucleotides
2 edge_case = {"UU": "X", "CU": "L", "AU": "X", "GU": "V",
3             "UC": "S", "CC": "P", "AC": "T", "GC": "A",
4             "UA": "X", "CA": "X", "AA": "X", "GA": "X",
5             "UG": "X", "CG": "R", "AG": "X", "GG": "G"}
  
```

```

6 # Dictionary containing the genetic code for codons
7 sequence =
8 {"UUU": "F", "CUU": "L", "AUU": "I", "GUU": "V", "UUC": "F", "CUC": "L",
9  "AUC": "I", "GUC": "V", "UUA": "L", "CUA": "L", "AUA": "I", "GUA": "V",
10 "UUG": "L", "CUG": "L", "AUG": "M", "GUG": "V", "UCU": "S", "CCU": "P",
11 "ACU": "T", "GCU": "A", "UCC": "S", "CCC": "P", "ACC": "T", "GCC": "A",
12 "UCA": "S", "CCA": "P", "ACA": "T", "GCA": "A", "UCG": "S", "CCG": "P",
13 "ACG": "T", "GCG": "A", "UAU": "Y", "CAU": "H", "AAU": "N", "GAU": "D",
14 "UAC": "Y", "CAC": "H", "AAC": "N", "GAC": "D", "UAA": "*", "CAA": "Q",
15 "AAA": "K", "GAA": "E", "UAG": "*", "CAG": "Q", "AAG": "K", "GAG": "E",
16 "UGU": "C", "CGU": "R", "AGU": "S", "GGU": "G", "UGC": "C", "CGC": "R",
17 "AGC": "S", "GGC": "G", "UGA": "*", "CGA": "R", "AGA": "R", "GGA": "G",
18 "UGG": "W", "CGG": "R", "AGG": "R", "GGG": "G"}
19
20 def translation(rna_seq):
21     """Given an RNA string corresponding to a strand of mRNA, calculate the
22     protein string encoded by it
23     Args:
24         rna_seq (str): RNA string corresponding to a strand of mRNA
25     Returns:
26         str: Returns the protein string encoded by the RNA string
27     """
28     protein_string = ""
29     for i in range(0, len(rna_seq)-2, 3):
30         trimer = rna_seq[i:i+3]
31         protein_string += sequence[trimer] + " "
32     return protein_string
33
34 def transcription(dna_seq):
35     new_seq = ""
36     for i in range(len(dna_seq)):
37         if dna_seq[i] == "T":
38             new_seq += "U" + " "
39         else:
40             new_seq += dna_seq[i] + " "
41     return new_seq
42
43 def edge_cases(dna_seq):
44     if len(dna_seq) % 3 == 0:
45         return ""
46     elif len(dna_seq) % 3 == 1:
47         return "X"
48     else:
49         two_nucleotide = dna_seq[-2:]
50         codon = edge_case[two_nucleotide] + " "
51         return codon
52
53 seq = "GACATTGTGAACAGTAAAAAGTCCATGCAATGCGCAAGGAGCAGAAGAGGAAGCAGGGCAAGCA
54 GCGCTCCATGGGCTCTCCCATGGACTACTCTCCTCTGCCATCGACAAGCATGAGCCTGAATTTGG
55 TCCATGCAGAAGAAAACTGGATGGG"
56 new_seq = transcription(seq)
57 prot = translation(new_seq)
58 protein = prot + " " + edge_cases(new_seq)
59 print(protein)

```

LISTING 2. Translation

The above code aims to translate the DNA sequence into a protein sequence. The transcription function converts a DNA sequence into an RNA sequence. The translation function then iterates through the RNA sequence in codons, using a predefined dictionary to map each codon to its corresponding amino acid in protein synthesis. An edge function addresses the cases where the DNA sequence length is not a multiple of 3.

The output of the above code, when `seq` is passed to the function, is:

**DIVNSKKVHAMRKEQKRKQGKQRSMGSPMDYSPLPIDKHEPEFGPCRRKLDG**

**Question 6.** Write a code to search for the following strings: **AAG**, **GTC**, **GAG**, **ACTA**, and **ATAT** in the DNA sequence provided in Q4. The program should print the total number of matches for each of the given strings and the start positions of the matches.

**Solution.** I am using an iterative brute-force approach to search for the strings within the DNA sequence. For each sequence, it iterates through the entire DNA sequence and checks for instances where the subsequence matches the search string. The locations of these instances are stored in a list and later converted to a string separating each instance with a `(,)`. The time complexity of this operation is  $O(n^m)$ , where  $n$  is the length of entire sequence and  $m$  is the number of search strings.

```

1 def find_seq(dna_seq, search_seq):
2     """Given a DNA sequence of nucleotides and a search string, find the
3     total number of times the search string occurs, alongwith its location
4     Args:
5         dna_seq (str): A big DNA sequence comprising nucleotides
6         search_seq (str): A small string sequence comprising nucleotides
7     Returns:
8         int: Gives the number of times the search string has appeared in
9         given DNA sequence
10        str: Gives a string of positions separated by (,) indicating the
11        positions of those occurrences
12    """
13    count_match = 0
14    match_loc = []
15    # Iterate through the length of the DNA sequence
16    for i in range(len(dna_seq) - len(search_seq) + 1):
17        subseq = dna_seq[i:i+len(search_seq)]
18        if search_seq == subseq:
19            count_match += 1
20            match_loc.append(str(i))
21
22    match_location = ', '.join(match_loc)
23    return count_match, match_location
24
25 dna_seq = "GACATTGTGAACAGTAAAAAGTCCATGCAATGCGCAAGGAGCAGAAGAGGAAGCAGGGCAAGCA
26 GCGCTCCATGGGCTCTCCCATGGACTACTCTCCTCTGCCATCGACAAGCATGAGCCTGAATTTGG
27 TCCATGCAGAAGAAACTGGATGG"
28 search_sequences = ["AAG", "GTC", "GAG", "ACTA", "ATAT"]
29 for search_seq in search_sequences:
30     c,l = find_seq(dna_seq, search_seq)
31     print("Enter the string:", search_seq)
32     print("Total match:", c)
33     print("Position of match:", l)
34     print()

```

LISTING 3. Search string in sequence

The outputs generated for each of the search strings mentioned in the question are as follows:

AAG

Enter the string: AAG  
Total match: 7  
Position of match: 19, 36, 45, 51, 60, 111, 140

GTC

Enter the string: GTC  
Total match: 2  
Position of match: 21, 130

GAG

Enter the string: GAG  
Total match: 3  
Position of match: 39, 47, 117

ACTA

Enter the string: ACTA  
Total match: 1  
Position of match: 88

ATAT

Enter the string: ATAT  
Total match: 0  
Position of match:

**Question 7.** Familiarize with other applications in EMBOSS. For example, melting temperature, bending, curvature etc.

**Solution.** (i) **dan** tool calculates the melting temperature( $T_m$ ) and the GC content for windows over a nucleic acid sequence, optionally plotting them. The window is moved along the sequence with the properties being calculated at each new position. Along with the above content, the change in enthalpy (H) and entropy (S) for dissociation of the oligomers may also be obtained. In order to calculate these, the user must provide the salt and DNA concentrations. Optionally, the percent formamide, percent of mismatches allowed and product length may be specified.

DAN

Calculate nucleic acid melting temperature

input section

Enter the 'seqall' input as:

☐ file/emboss-query or ☒ paste or ☐ list of files

Cut and Paste

CTCGGATTGTAAAGATCATGATCTCATACATAGTACCTAGCCATTG

required section

20

Enter window size  
(min:1 max:100 default:20)

1

Enter Shift Increment  
(min:1 default:1)

50

Enter DNA concentration (nM)  
(min:1. max:100000. default:50.)

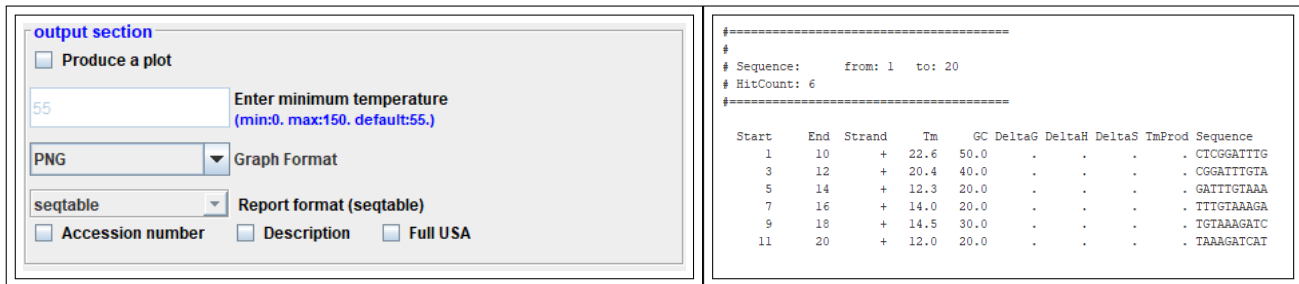
50

Enter salt concentration (mM)  
(min:1. max:1000. default:50.)

(A) Input section of dan tool

(B) Required section of dan tool

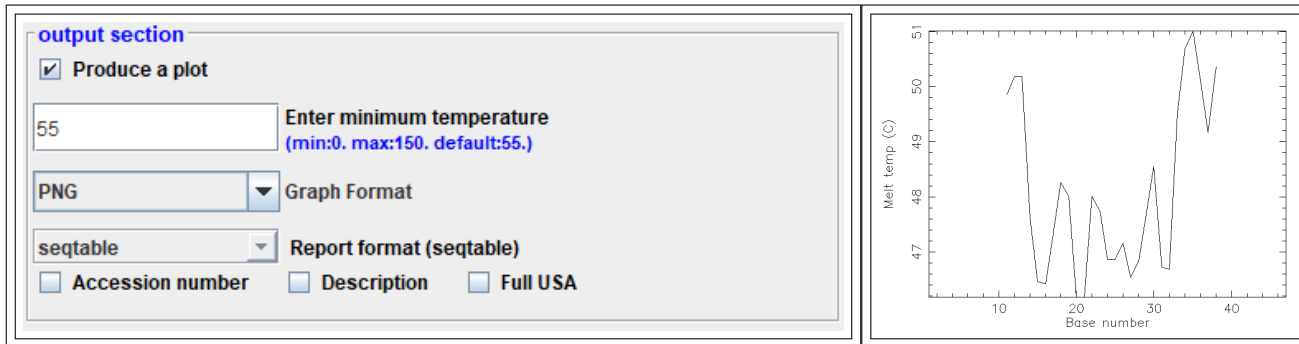
FIGURE 8. Input to the dan tool



(A) Output section of dan tool without a plot

(B) Output information by dan tool

FIGURE 9. Output of the dan tool



(A) Output section of dan tool with a plot

(B) Output graph by dan tool

FIGURE 10. Output of the dan tool

(ii) **banana** tool plots the bending and curvature data for normal DNA. It predicts the bending of the DNA double helix. The program calculates the magnitude of local bending and macroscopic curvature at each point along the DNA sequence, using any bending model with parameters.

**input section**

Enter the 'sequence' input as:

☐ file/emboss-query or ☒ paste or ☐ list of files

Cut and Paste

CTCGGATTTGTA

Eangles\_tri.dat DNA base trimer roll angles data file (default:Eangles\_tri.dat)

**output section**

PNG Graph Format

50 Number of residues to be displayed on each line (default:50)

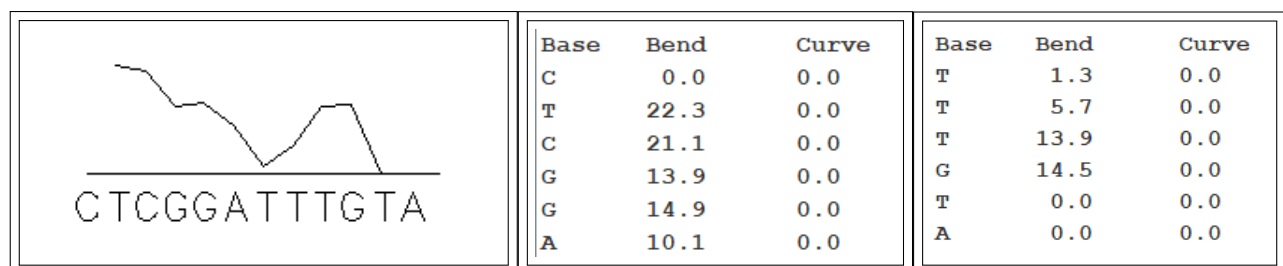
banana.profile (default:banana.profile)

outfile file name

(A) Input section of banana tool

(B) Output section of banana tool

FIGURE 11. Input and output sections of the banana tool



(A) Graphical output

(B) banana profile

(C) banana profile

FIGURE 12. Output representation of the banana tool



**Question 8.** Write a program to compute the average base stacking energy for the sequence given in Q2 (AA: -4; AT: -7; AC: -5; AG: -11; TA: -7; TT: -2; TC: -3; TG: -4; CA: -9; CT: -5; CC: -6; CG: -7; GA: -9; GT: -6; GC: -4; GG: -11).

**Solution.** In order to calculate the average base stacking energy, I first created a dictionary with dinucleotide stacking energy. Then, iterating over the entire nucleotide sequence, I added the dinucleotide stacking energy for every consecutive overlapping nucleotide. The mean of the so obtained energy is the answer.

```
1 dinucleotide_energy = {"AA": -4, "AT": -7,
2                          "AC": -5, "AG": -11,
3                          "TA": -7, "TT": -2,
4                          "TC": -3, "TG": -4,
5                          "CA": -9, "CT": -5,
6                          "CC": -6, "CG": -7,
7                          "GA": -9, "GT": -6,
8                          "GC": -4, "GG": -11}
9
10 def calculate_stack_energy(dna_seq):
11     """Compute the average base stacking energy for the given nucleotide
12     sequence
13
14     Args:
15         dna_seq (str): The input nucleotide sequence
16
17     Returns:
18         float: The average base stacking energy for the sequence
19     """
20     total_energy = 0
21     for i in range(len(dna_seq)-1):
22
23         dinucleotide = dna_seq[i:i+2]
24         total_energy += dinucleotide_energy[dinucleotide]
25
26     avg_energy = total_energy / (len(dna_seq)-1)
27     return avg_energy
28
29 dna_seq = "CTCGGATTTGTAAAGATCATGATCTCATACATAGTACCTAGCCATTG"
30
31 avg_stack_energy = calculate_stack_energy(dna_seq)
32
33 print(avg_stack_energy)
```

LISTING 4. Average base stacking energy

The output of the above code, when **dnaseq** is passed to the function, is:  
**-6.282608695652174** units

**Question 9.** Compute the average melting temperature of the following sequences using the Seq2Feature tool <https://www.iitm.ac.in/bioinfo/SBFE/index.html> and comment on the results (Enter one sequence at a time in fasta format) (i) ATATATATAT (ii) GCGCGCGCGC

**Solution.** The Seq2Feature tool is a web-based feature extraction tool that computes protein and DNA driven features. Major protein sequence based descriptors include physical-chemical, mutation matrices, and contact potentials. Major DNA-based properties are physical-chemical, conformational, and composition based. Seq2Feature is a Python-based toolkit to compute the numerical values associated with DNA and protein sequences.

(i) The average melting temperature for the sequence **ATATATATAT** is: **48.0022 °C**

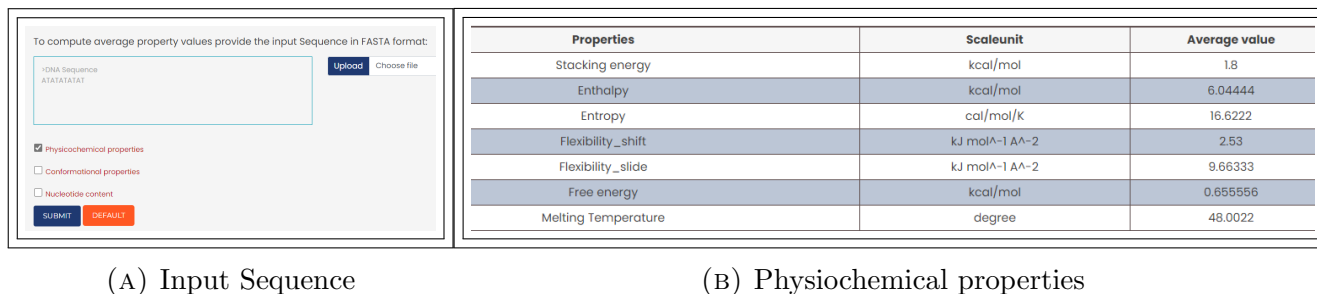


FIGURE 13. Seq2Feature tool for physio-chemical properties

(ii) The average melting temperature for the sequence **GCGCGCGCGC** is: **107.867 °C**

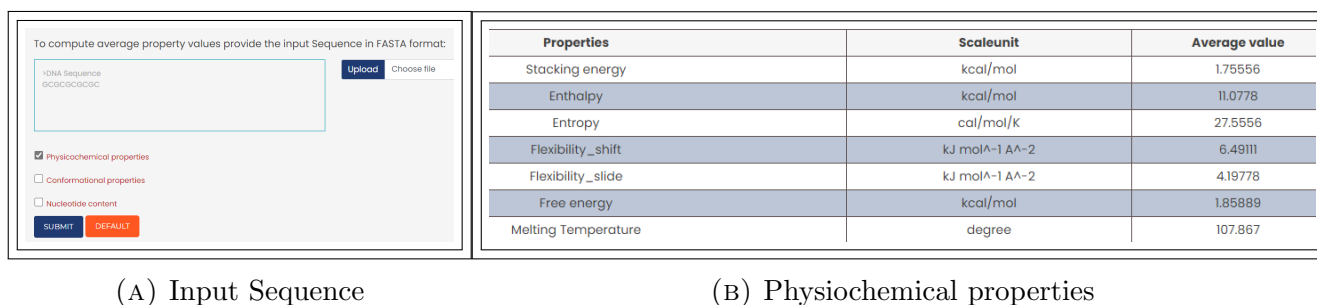


FIGURE 14. Seq2Feature tool for physio-chemical properties

**Question 10.** Calculate the AT and GC content of the sequence **AAATGGCCCTAA** using Seq2Feature tool

**Solution.** The Seq2Feature tool is also capable of giving the nucleotide content of a given sequence. The nucleotide content (AT and GC specifically) for the given sequence is:

**AT content: 58.333333%**  
**GC content: 41.666667%**

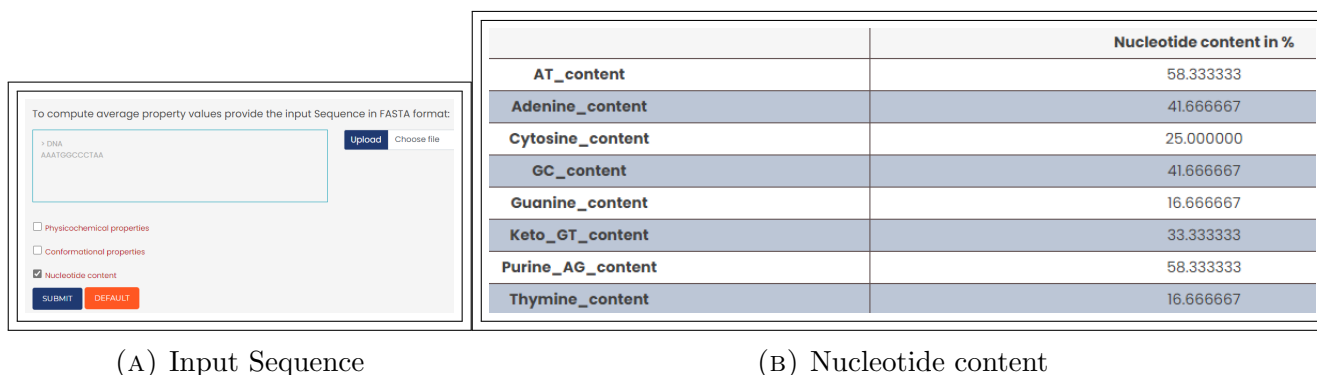


FIGURE 15. Seq2Feature tool for nucleotide content