BT3040 – BIOINFORMATICS – Assignment 10

Submitted by Sahana (BE17B038)

Question 1

Algorithm –

- Input to the program is given as two strings – the sequence and secondary structure identity.
- Compute the number of occurrences of each AA throughout the sequence.
- Compute the number of occurrences of each AA for those whose secondary str identity is Helix (H) from the sequence.
- Take ratio between the above two values for each AA.
- Find the ratio of total number of AAs in helix conformation to the length of the input sequence.
- The propensity of each AA is the ratio of the above two ratios.

Code –

```
def propensity(seq,ss):
    n = len(seq)
    AA_all
=['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y'
]

    composition = [0]*20 #Number of occurance of each AA Acid.
    H = [0]*20 #Number of occurance of each AA in a helix.
    ratio_H = [0]*20
    propensity = [0]*20

    for i in range(n):
        aa = seq[i]
        ind = AA_all.index(aa)
        composition[ind]+=1
        ty = ss[i]
        if str(ty)=='H':
            H[ind]+=1


    print('Composition')
    print(composition)
    print('\nHelix composition')
    print(H)
    nH = ss.count('H')
    percent_H = nH/n
    print('\nPercentage Helix = %4.3f' %percent_H)

    for i in range(20):
        ratio_H[i]=H[i]/composition[i]
        propensity[i] = ratio_H[i]/percent_H
    print('\nPercentage AA in helix wrt overall')
    print(ratio_H)
    print('\nThe propensity of each AA is -')
    for i in range(20):
        print('%s   %4.3f' %(AA_all[i], propensity[i]))
    print('\nMost preferred AA in helix - with cutoff = 1.000')
    for i in range(20):
```

```
        if propensity[i]>=1.000:
            print('%s   %4.3f' %(AA_all[i], propensity[i]))
seq =
'LGASGIAAFAFGSTAILIILFNMAAEVHFDPLQFFRQFFWLGLYPPKAQYGMGIPPLHDGGWWLMAGLFMTLSLGSWWIR
VYSRARALGLGTHIAWNFAAAIFFVLCIGCIHPTLVGSWSEGVPFGIWPHIDWLTAFSIRYGNFYYCPWHGFSIGFAYGCG
LLFAAHGATILAVARFGGDREIEQITDRGTAVERAALFW'
ss =
'XHHHHHHHHHHHHHHHHHHHHHHHHHHXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXHHHHHHHHHHHHHHHHHHH
HHHHHHHHHHXXHHHHHHHHHHHHHHHHHXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXHHHHHHHHHHHHHH
HHHHHHHHHHHXXXXXXXXXXXXXXXXXXXXXXXXXX'
propensity(seq,ss)
```

Output –

```
Composition
[25, 4, 5, 5, 20, 25, 7, 17, 1, 20, 4, 3, 9, 4, 9, 9, 8, 7, 11, 7]

Helix composition
[19, 2, 0, 1, 10, 13, 3, 10, 0, 12, 3, 2, 1, 0, 3, 6, 3, 2, 6, 2]

Percentage Helix = 0.490

Percentage AA in helix wrt overall
[0.76, 0.5, 0.0, 0.2, 0.5, 0.52, 0.42857142857142855, 0.5882352941176471, 0.0
, 0.6, 0.75, 0.6666666666666666, 0.1111111111111111, 0.0, 0.3333333333333333,
 0.6666666666666666, 0.375, 0.2857142857142857, 0.5454545454545454, 0.2857142
857142857]

The propensity of each AA is -
A   1.551
C   1.020
D   0.000
E   0.408
F   1.020
G   1.061
H   0.875
I   1.200
K   0.000
L   1.224
M   1.531
N   1.361
P   0.227
Q   0.000
R   0.680
S   1.361
T   0.765
V   0.583
W   1.113
Y   0.583

Most preferred AA in helix - with cutoff = 1.000
A   1.551
C   1.020
F   1.020
G   1.061
I   1.200
L   1.224
M   1.531
N   1.361
```

```
S    1.361
W    1.113
```

Question 2 (manual solution)

Q2)

→ Total number of amino acids in 'Helix' ('H') conformation = 98 = n_H

Length of given sequence of amino acids = 200 = N

$\therefore \frac{n_H}{N} = \frac{98}{200} = 0.49.$

Propensity = $\frac{\text{Ratio of AA}}{(n_H/N)}$ { The encircled / box one are most preferred AA in helix. }

→ Composition of AA in the entire sequence and in Helix conformation:

(# occurrences)

Propensity ←	AA	Whole sequence	Helix conformation	Ratio of AA
1.551	A	25	19	19/25 = 0.76
~~0.20~~ 1.020	C	4	2	2/4 = 0.5
0.000	D	5	0	0/5 = 0
0.408	E	5	1	1/5 = 0.2
1.021	F	20	10	10/20 = 0.5
1.06	G	25	13	13/25 = 0.52
0.875	H	7	3	3/7 = 0.428
1.20	I	17	10	10/17 = 0.588
0.00	K	1	0	0/1 = 0
1.224	L	20	12	12/20 = 0.6
1.631	M	4	3	3/4 = 0.75
1.361	N	3	2	2/3 = 0.667
0.227	P	9	1	1/9 = 0.111
0.00	Q	4	0	0/4 = 0
0.680	R	9	3	3/9 = 0.333
1.361	S	9	6	6/9 = 0.667
0.765	T	8	3	3/8 = 0.375
0.583	V	7	2	2/7 = 0.286
1.113	W	11	6	6/11 = 0.545
0.583	Y	7	2	2/7 = 0.286

Question 3

Algorithm –

- According to the given instructions first find those segments that belong to helix and those that belong to strands.
    - First take the (6/5)-character segment.
    - If it's parameter value if greater than (4/3), I check last three characters' summed propensity.
    - If propensity value is greater than (4/3), I consider the next character and check again for the last (4/3) characters.
    - Let's index these segments. So initially if I had 12345, I check 345, and then 456, and so on (in case of beta strands).
    - When the next segment's propensity value is less than (4/3), I terminate the entire secondary structure segment there.
    - For conflicting matches –
        - Here we check the matching parts' propensity sum, i.e, if alpha has 2345678 and beta has 123456, I check and compare propensity in alpha and beta only for 23456.

Code –

```
def matchingString(x,y):
    match=''
    for i in range(0,len(x)):
        for j in range(0,len(y)):
            k=1
            # now applying while condition untill we find a substring match and
length of substring is less than length of x and y
            while (i+k <= len(x) and j+k <= len(y) and x[i:i+k]==y[j:j+k]):
                if len(match) <= len(x[i:i+k]):
                    match = x[i:i+k]
                k=k+1
    return match

def secondary_str(seq):

    helix = {'A': 'Ha', 'C': 'ia', 'D': 'ia', 'E': 'Ha', 'F': 'ha',
             'G': 'Ba', 'H': 'ha', 'I': 'Ia', 'K': 'Ia', 'L': 'Ha',
             'M': 'ha', 'N': 'ba', 'P': 'Ba', 'Q': 'ha', 'R': 'ia',
             'S': 'ia', 'T': 'ia', 'V': 'ha', 'W': 'ha', 'Y': 'ba'}

    strand = {'A': 'Ib', 'C': 'hb', 'D': 'ib', 'E': 'Bb', 'F': 'hb',
              'G': 'ib', 'H': 'bb', 'I': 'Hb', 'K': 'bb', 'L': 'hb',
              'M': 'Hb', 'N': 'bb', 'P': 'bb', 'Q': 'hb', 'R': 'ib',
              'S': 'bb', 'T': 'hb', 'V': 'Hb', 'W': 'hb', 'Y': 'hb'}

    propensity_alpha = {'A': 1.45, 'C': 0.77, 'D': 0.98, 'E': 1.53, 'F': 1.12,
                        'G': 0.53, 'H': 1.24, 'I': 1.00, 'K': 1.07, 'L': 1.34,
                        'M': 1.20, 'N': 0.73, 'P': 0.59, 'Q': 1.17, 'R': 0.79,
                        'S': 0.79, 'T': 0.82, 'V': 1.14, 'W': 1.14, 'Y': 0.61}

    propensity_beta = {'A': 0.97, 'C': 1.30, 'D': 0.80, 'E': 0.26, 'F': 1.28,
                       'G': 0.81, 'H': 0.71, 'I': 1.60, 'K': 0.74, 'L': 1.22,
                       'M': 1.67, 'N': 0.65, 'P': 0.62, 'Q': 1.23, 'R': 0.90,
                       'S': 0.72, 'T': 1.20, 'V': 1.65, 'W': 1.19, 'Y': 1.29}
```

```
cf = {'Ha' : 1, 'ha' : 1, 'Ia' : 0.5, 'ia' : 0, 'ba' : -1, 'Ba' : -1,
      'Hb' : 1, 'hb' : 1, 'Ib' : 0.5, 'ib' : 0, 'bb' : -1, 'Bb' : -1}
#To store the final segments that are a part of either secondary structures.
helix_seq = []
strand_seq = []

n = len(seq)
#Identifying initial segments that fall into Alpha helices
print('Alpha helices: ')
i = 0
while i<n-6:
    segment = seq[i:i+6]
    value = 0
    for j in range(6):
        typ = helix[segment[j]]
        value+= cf[typ]
    if value>=4:
        done = 1
        k = 0
        while done==1:
            next_seg = seq[i+k+2:i+k+6]
            prop = 0
            for l in range(4):
                prop+=propensity_alpha[next_seg[l]]
            if prop<4.00:
                done = 0
            else:
                k+=1
        if k==0:
            print(seq[i:i+k+6])
            helix_seq.append(seq[i:i+k+6])
            i = i+k+6
        else:
            print(seq[i:i+k+5])
            helix_seq.append(seq[i:i+k+5])
            i = i+k+5

    else:
        i+=1
#Identifying initial segments that fall into Beta strands
print('\nBeta strands: ')
i = 0
while i<n-5:
    segment = seq[i:i+5]
    value = 0
    for j in range(5):
        typ = strand[segment[j]]
        value+= cf[typ]

    if value>=3:
        done = 1
        k = 0

        while done==1 and (i+k+5)<=n:
            next_seg = seq[i+k+2:i+k+5]
            prop = 0
            for l in range(3):
                prop+=propensity_beta[next_seg[l]]
            if prop<3.00:
                done = 0
```

```
                else:
                    k+=1

            if k==0:
                print(seq[i:i+k+5])
                strand_seq.append(seq[i:i+k+5])
                i = i+k+5
            else:
                print(seq[i:i+k+4])
                strand_seq.append(seq[i:i+k+4])
                i = i+k+4

        else:
            i+=1
    #Lists to store conflicting segment's (wrong) index and the entire sequence
exactly as where it belongs to
    hf = []
    sf = []
    print('\nCommon segments - Conflicting sequences')
    for i in range(len(helix_seq)):
        h = helix_seq[i]

        for j in range(len(strand_seq)):
            s = strand_seq[j]

            c = matchingString(h,s)
            m = len(c)
            if m!=0 and m>=5:
                print('Helix - %s, Strand - %s, Common segment - %s' %(h,s,c))
                prop_helix = 0
                prop_strand = 0
                for k in range(m):
                    prop_helix+=propensity_alpha[c[k]]
                    prop_strand+=propensity_beta[c[k]]
                if prop_helix > prop_strand:
                    hf.append([j,h])
                else:
                    sf.append([i,s])

    for i in range(len(hf)-1,-1,-1):
        a = strand_seq[hf[i][0]]
        strand_seq.remove(a)

    for j in range(len(sf)-1,-1,-1):
        b = helix_seq[sf[j][0]]
        helix_seq.remove(b)
    #Results
    print('\nFinal list of secondary structure segments -')
    print('\nAlpha Helix segments')
    for i in range(len(helix_seq)):
        print(helix_seq[i])
    print('\nBeta Strand segments')
    for i in range(len(strand_seq)):
        print(strand_seq[i])


sf =
'KVFGRCELAAAMKRHGLDNYRGYSLGNWVCAAKFESNFNTQATNRNTDGSTDYGILQINSRWWCNDGRTPGSRNLCNIPC
SALLSSDITASVNCAKKIVSDGNGMNAWVAWRNRCKGTDVQAWIRGCRL'
secondary_str(sf)
```

Output –

```
Alpha helices:
RCELAAAMKRH
WVCAAKFESNF
MNAWVAWRN
TDVQAWIR

Beta strands:
VFGRC
LAAAMKR
WVCAA
TDYGILQIN
AWVAWR
GTDVQAWIRGCRL

Common segments - Conflicting sequences
Helix - RCELAAAMKRH, Strand - LAAAMKR, Common segment - LAAAMKR
Helix - WVCAAKFESNF, Strand - WVCAA, Common segment - WVCAA
Helix - MNAWVAWRN, Strand - AWVAWR, Common segment - AWVAWR
Helix - TDVQAWIR, Strand - GTDVQAWIRGCRL, Common segment - TDVQAWIR

Final list of secondary structure segments -

Alpha Helix segments
RCELAAAMKRH
MNAWVAWRN

Beta Strand segments
VFGRC
WVCAA
TDYGILQIN
GTDVQAWIRGCRL
```

Question 4 (manual solution)

Alpha helix:

'KVFGRC' = 0.5 + 1 + 1 − 1 + 0 + 0 = 1.5 : Not greater than 4.0.

'VFGRCE' = 1 + 1 − 1 + 0 + 0 + 1 = 2. Not greater than 4.0.

'FGRCEL' = 1 − 1 + 0 + 0 + 1 + 1 = 2. Not greater than 4.0.

'GRCELA' = −1 + 0 + 0 + 1 + 1 + 1 = 2. Not greater than 4.0.

'RCELAA' = 0 + 0 + 1 + 1 + 1 + 1 = 4. ≥ 4 (satisfied).

Extending the segment:

checking final 4 amino acids for propensity:

'ELAA' = 1.53 + 1.34 + 1.45 + 1.45 = 5.77 ≥ 4 (satified & extending)

'LAAM' = 1.34 + 1.45 + 1.45 + 1.20 = 5.69 ≥ 4

'AAMK' = 1.45 + 1.45 + 1.20 + 1.07 = 5.55 ≥ 4.

'AMKR' = 1.45 + 1.20 + 1.07 + 0.79 = 4.51 ≥ 4.

'MKRH' = 1.20 + 1.07 + 0.79 + 1.24 = 4.3 ≥ 4

'KRHG' = 1.07 + 0.79 + 1.24 + 0.53 = 3.63 ≤ 4 (not greater than 4)

Hence, potential alpha helix segment = 'RCELAAMKRH'.

Beta strand:

'KVFGRC' = −1 + 1 + 1 + 0 + 0 = 1. < 3. (not greater than 3.0)

'VFGRC' = 1 + 1 + 0 + 0 + 1 = 3 ≥ 3. (satisfied).

Extending the segment:

checking final 3 amino acids for propensity:

'GRC' = 0.81 + 0.90 + 1.30 = 3.01 ≥ 3 (satisfied & extending)

'RCE' = 0.90 + 1.30 + 0.26 = 2.46 < 3 (not greater than 3)

Hence, potential beta strand segment = 'VFGRC'.