BT2042: Fundamentals of Biophysical Chemistry

ASSIGNMENT

NAME AND ROLL NO: RITVIK PANDEY(BS21B028)
S DEEPAK KUMAR(BS21B013)
RAHUL SRINIVAS(BS21B029)

1. For the 2D 16-mer polymer discussed in the class, run the unfolding simulations starting from the fully folded state for four interaction energy values of -0.25, -0.75, -1.25 and -2.5 (assume kT=1). Note that these energy values are assigned to only those pairs of non-covalent interactions present in the folded structure as discussed in the class.
2. For every energy value, the simulation should be run for 10^5 steps and for 50 molecules (i.e. a total of 10^5 x 50 steps for every energy value). What differences do you observe across the different energy values? Rationalize them.
3. What reaction coordinate can you use to monitor the folding or unfolding of the protein? Using your own take on this, construct a one-dimensional free energy profile.
4. How many different microstates are sampled and how would you say a given molecule is unfolded?
5. How long does the 16-mer take to unfold? Is there a distribution in the number of steps taken to unfold? If so, why?
6. What would happen if the beads in the polymer were assigned a favorable non-covalent interaction energy of -1.25 irrespective of whether it is present or absent in the folded structure? In other words, any bead can interact with another bead as long as it is not covalently linked. Would the molecule take longer to unfold? If so, by how many more steps?
7. Movies of the unfolding runs can also be made.

Algorithm Explanation

Mapping the Residues

To map the residues to generate the native protein structure, I used a 2D NumPy array to store the coordinates of each residue according to their order. Given below is the code snippet that I used for that:

```python
residues = np.array([[0, 0, 1, 1, 2, 2, 3, 3, 3, 2, 1, 0, 0, 1, 2, 3], [1, 0, 0, 1, 1, 0, 0, 1, 2, 2, 2, 2, 3, 3, 3, 3]])
res_prob = residues.copy()
```

[res_prob] is the temporary 2D array that stores the temporary movement of the residue that goes to the metropolis criterion for checking.

Parameters

Radius of Gyration

One of the parameters that I used for unfolding was the Radius of Gyration of a particular conformation of the protein. First, I calculated the centre of mass of the protein, and then, using the equation given below, I calculated the Radius of Gyration for a particular conformation:

$$r_g = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_i - r_{cm})^2}$$

I used the code given below to calculate that:

```python
cmx = sum(residues[0]) / 16
cmy = sum(residues[1]) / 16
srg = 0
```

```
for k in range(16):
    srg = srg + (math.dist([residues[0][k], residues[1][k]], [cmx, cmy]) ** 2)
Rg = [(srg / 16) ** 0.5]
```

End-to-End Distance

End to end distance is the distance between first and last residue and can be a good measure of how much the protein unfolded. To measure that, I used the following code:

Python

```
e2e = [math.dist([residues[0][0], residues[1][0]], [residues[0][15], residues[1][15]])]
```

Interaction Energy

The total Interaction Energy of the whole protein can be calculated using the formula:
E = N * ES; where N is the number of native interactions and ES is the given energy of a single **non-covalent interaction**. I used the following code to map the number of native interactions remaining in the given conformation:

Python

```
#Mapping the non-covalent Interactions
   n = 0
   if math.dist([residues[0][0], residues[1][0]], [residues[0][3], residues[1][3]]) == 1:
       n += 1
   if math.dist([residues[0][0], residues[1][0]], [residues[0][11], residues[1][11]]) == 1:
       n += 1
   if math.dist([residues[0][3], residues[1][3]], [residues[0][10], residues[1][10]]) == 1:
       n += 1
   if math.dist([residues[0][2], residues[1][2]], [residues[0][5], residues[1][5]]) == 1:
```

```
        n += 1
    if math.dist([residues[0][4], residues[1][4]], [residues[0][7],
residues[1][7]]) == 1:
        n += 1
    if math.dist([residues[0][4], residues[1][4]], [residues[0][9],
residues[1][9]]) == 1:
        n += 1
    if math.dist([residues[0][8], residues[1][8]], [residues[0][15],
residues[1][15]]) == 1:
        n += 1
    if math.dist([residues[0][9], residues[1][9]], [residues[0][14],
residues[1][14]]) == 1:
        n += 1
    if math.dist([residues[0][10], residues[1][10]],
[residues[0][13], residues[1][13]]) == 1:
        n += 1
```

Mapping the Interactions when the residues interact with all residues

In the case when each residue can interact with any other residue instead of just the native configuration residues, I used the following code:

Python

```
for m in range(16):
        for m1 in range(m+2, 16):
            if math.dist([residues[0][m], residues[1][m]],
[residues[0][m1], residues[1][m1]]) == 1:
                n += 1
```

This loop checks the distance between nth residue and (n + 2)th residue and if it is 1, it counts it as an interaction. This loop checks for each residue against all the other residues apart from the ones it has a covalent bond with, i.e., (n - 1) and (n + 1).

Unfolding

Then I go into the loop to iterate over 100000 iterations or turns or steps.

Choosing a random residue

I randomly chose a residue, and based on if it was an end residue or anything in between, I went on to the move. If it was an end residue (1 or 16), I used the End Move code block, and if it was any other residue, I used a corner move code block.

Python

```
cm = np.random.randint(16); #Randomly choosing one residue
```

End Move

In the case of the end residues, they have 4 possibilities that they can move through

1. (x-1, y-1)
2. (x+1, y-1)
3. (x-1, y+1)
4. (x+1, y+1)

I randomly chose one of these 4 possibilities and initiated the move. If the move that was chosen satisfies the following criteria, it was accepted:

1. No overlap with any other residue.
2. The bond length should remain constant.

I used the following code for this:

Python

```
#End move (Available only for residues 1 and 16)
        r = np.random.randint(4)
        xpt = residues[0][cm] + x_shift[r]
        ypt = residues[1][cm] + y_shift[r]

        if cm == 0:
            m = 0
```

```
            if math.dist([xpt, ypt], [residues[0][1],
residues[1][1]]) == 1:
                for k in range(0, 16):
                    if math.dist([xpt, ypt], [residues[0][k],
residues[1][k]]) == 0:
                        m = 1
                        break

                if m == 0:
                    res_prob[0][cm] = xpt
                    res_prob[1][cm] = ypt

        elif cm == 15:
            m = 0
            if math.dist([xpt, ypt], [residues[0][14],
residues[1][14]]) == 1:
                for k in range(0, 16):
                    if math.dist([xpt, ypt], [residues[0][k],
residues[1][k]]) == 0:
                        m = 1
                        break

                if m == 0:
                    res_prob[0][cm] = xpt
                    res_prob[1][cm] = ypt
```

Corner Move

A residue at the corner can only move in one direction provided the previous and next residues are fixed, and that is diagonally. Suppose the coordinates of the nth residue are (x,y), for the (n-1)th residue is (x',y) and for the (n+1)th residue is (x,y'). Then the new coordinates of the residue after the corner move will be (x',y'). Furthermore, the criteria to accept the move was the same as the End move. The following code was used to initiate a corner move:

Python

```
#Corner Move
        xpt = 0
        ypt = 0
        if residues[0][cm] == residues[0][cm-1] and residues[1][cm]
== residues[1][cm+1]:
            xpt = residues[0][cm + 1]
            ypt = residues[1][cm - 1]
        elif residues[0][cm] == residues[0][cm + 1] and
residues[1][cm] == residues[1][cm - 1]:
            xpt = residues[0][cm - 1]
            ypt = residues[1][cm + 1]

        m = 0
        if math.dist([xpt, ypt], [residues[0][cm + 1], residues[1][cm
+ 1]]) == 1 and math.dist([xpt, ypt], [residues[0][cm - 1],
residues[1][cm - 1]]) == 1:
            for k in range(0, 16):
                if math.dist([xpt, ypt], [residues[0][k],
residues[1][k]]) == 0:
                    m = 1
                    break

            if m == 0:
                res_prob[0][cm] = xpt
                res_prob[1][cm] = ypt
```

Metropolis Criterion

To check the favourability of the move, I implemented the metropolis criterion:

Metropolis Criterion

$$E_i \text{ be the energy of the } i^{th} \text{ state and } E_{i+1} \text{ be the energy of } (i+1)^{th} \text{ state}$$

$$\downarrow$$

$$\Delta E = E_{i+1} - E_i$$

$$\downarrow$$

$$\text{Calculate } w = e^{(-\Delta E/kT)}$$

$$\downarrow$$

$$\text{If } w > 1 \text{ then move to the next step/conformation}$$

$$\downarrow$$

$$\text{If } w < 1 \text{ then generate a random number } r\ [0,1]$$

$$\downarrow$$

$$\text{If } w > r \text{ then move to the next step or conformation, if not stay put}$$

To implement that into the simulation, I used the following code:

Python

```
Et = n * ES #Energy for the current configuration (Temporary)

    dE = Et - Ei
    w = np.exp(-dE)
    if w>1:
        Ei = Et
        residues = res_prob
    else:
        tn = np.random.rand()
        if w>tn:
            Ei = Et
            residues = res_prob
        else:
```

```
            res_prob = residues
```

Number of Microstates sampled

To calculate the number of microstates sampled, I multiplied the Interaction energy (unless it was zero in which case, I took it as 1), the Radius of Gyration and the End to end distance to get a score for each conformation. Each unique score was a unique microstate that was sampled. To implement that, I used the following code:

Python
```python
if E != 0:
        Score = E[i] * e2e[i] * Rg[i]
    else:
        Score = e2e[i] * Rg[i]
    microstates.append(Score)
micro_unique = len(set(microstates))
```

The function set(), is used to find the unique values in our arrays.

Exporting Animation

To export the animation, I used the FuncAnimation function from the library matplotlib.animation. The given code shows the implementation of the function:

Python
```python
ani = animation(fig, update, frames = range(turns))
ani.save('Anim_E.mp4', writer='ffmpeg', fps=30)
```

Free Energy Calculation

To calculate the free energy, I used the following thermodynamic properties:

$$dG = dH - TdS$$
$$dH = dU + PdV$$
Since, $dV = 0$
$$dG = dU - TdS$$

dU is basically our interaction energy and $dS = k\ln W$

lnW = (Number of microstates at a given n)/(Total number of microstates sampled)
Since kT = 1 (given),

dG = (n * ES) - ln((microstates at given n)/(total microstates sampled))

This will give us a value of dG at each interaction level (n=0 to n=9). Since we will get a value of dG at a particular n for the 50 molecules, I took the average of microstates sampled and used it to calculate (kind of) the mean free energy for each interaction level. I took the value of n as my reaction coordinate in the form of (n/9). This gave us the value of the reaction coordinate from 0 to 1 and a nice distribution I have attached below.

To count the number of microstates at each energy level, I used the following code:

Python

```
if n == 0:
            n0.append(Score)
        elif n == 1:
            n1.append(Score)
        elif n == 2:
            n2.append(Score)
        elif n == 3:
            n3.append(Score)
        elif n == 4:
            n4.append(Score)
        elif n == 5:
            n5.append(Score)
        elif n == 6:
            n6.append(Score)
        elif n == 7:
            n7.append(Score)
        elif n == 8:
            n8.append(Score)
        elif n == 9:
            n9.append(Score)



    n_sum[0] = n_sum[0] + len(set(n0))
```

```
    n_sum[1] = n_sum[1] + len(set(n1))
    n_sum[2] = n_sum[2] + len(set(n2))
    n_sum[3] = n_sum[3] + len(set(n3))
    n_sum[4] = n_sum[4] + len(set(n4))
    n_sum[5] = n_sum[5] + len(set(n5))
    n_sum[6] = n_sum[6] + len(set(n6))
    n_sum[7] = n_sum[7] + len(set(n7))
    n_sum[8] = n_sum[8] + len(set(n8))
    n_sum[9] = n_sum[9] + len(set(n9))
```

The code to calculate dG is given below:

Python

```
for mi in range(10):
    n_mean[mi] = n_sum[mi] / 50

delG_arr = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
smarr = sum(n_mean)
for i in range(10):
    W = n_mean[i] / smarr
    delG_arr[i] = (i * (-0.25)) - np.log(W)
```

Exporting the Data

To export all the data that was obtained, I used the following code:

Python

```
import pandas as pd
pd.DataFrame(E_matrix).to_csv('Total Energy at Random Interactions E
= -1_25.csv')
pd.DataFrame(Rg_matrix).to_csv('Radius of Gyration at Random
Interactions at E = -1_25.csv')
pd.DataFrame(e2e_matrix).to_csv('End-to-End distance at Random
Interactions at E = -1_25.csv')
```

```
pd.DataFrame(E_max_turn_array).to_csv('Unfolding turns at Random
Interactions at E = -1_25.csv')
pd.DataFrame(microstates_50).to_csv('Number of microstates at Random
Interactions E = -1_25.csv')
pd.DataFrame(n_mean).to_csv('Microstates at Random Interactions at
each n for E = - 1_25.csv')
pd.DataFrame(delG_arr).to_csv('Delta G at Random Interactions at
each n for E = - 1_25.csv')
```

All the data that was exported is in the zip file

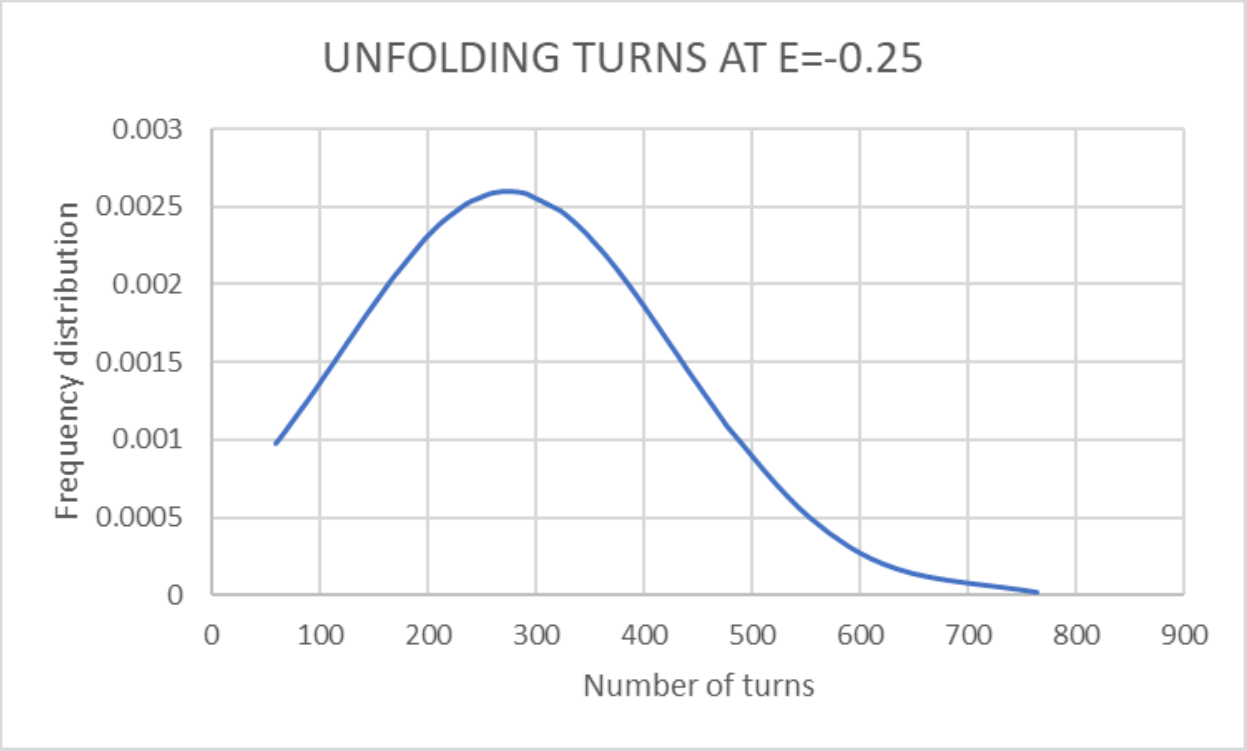
Rationalizing the difference between different Energy values

In the protein folding simulation, as the non-covalent interaction energy values decreased, the non-covalent interaction between the residues became stronger. Hence, we expect that the molecule will take more turns to unfold. This was indeed evident from the data that was obtained after running the simulation. I have given the relevant data in the table given below along with the distribution of the number of turns it took to unfold at each energy level.

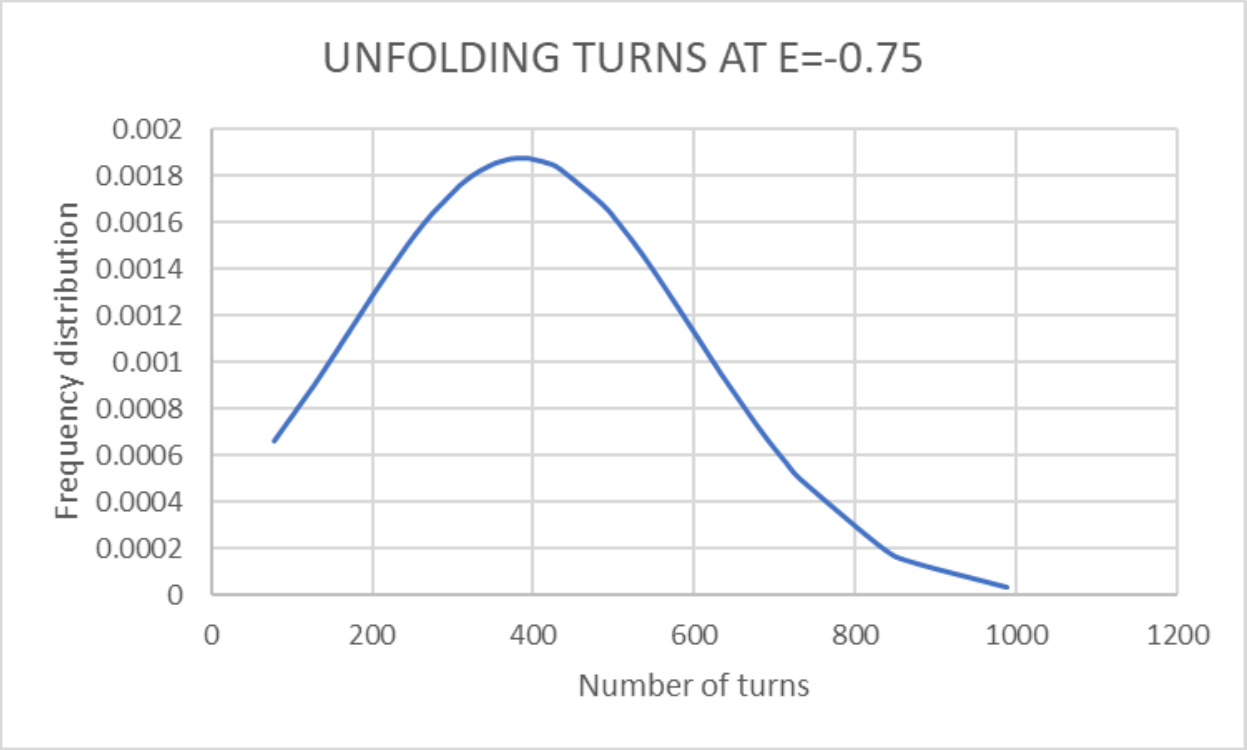
Energy (kT units)	***Mean***	***Median***	***Standard Deviation***	***Minimum Value***	***Maximum Value***
-0.25	279.42	267	149.8885	59	763
-0.75	385.38	343	212.8434	78	989
-1.25	375.7843	330	270.0979	79	1484
-2.50	995.38	850	614.9961	199	2634

Distribution of Unfolding Turns

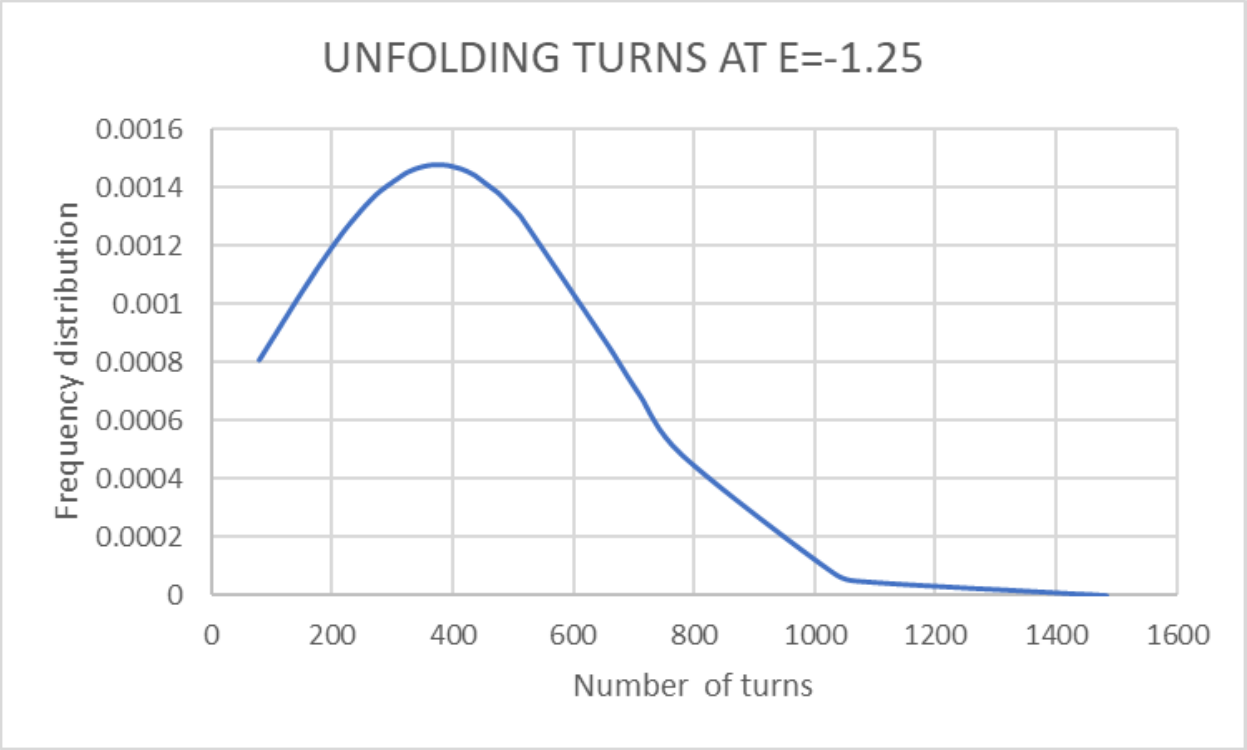
1. **E = -0.25**



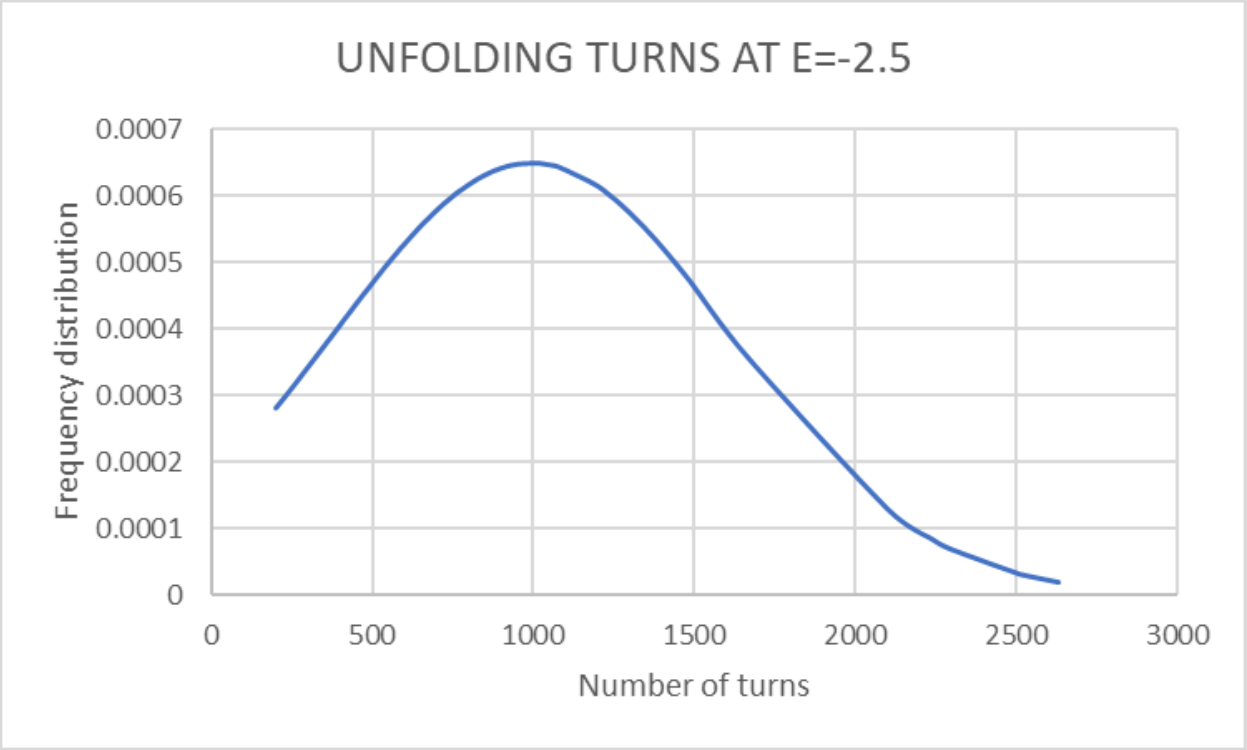
2. **E = -0.75**


UNFOLDING TURNS AT E=-0.75
Frequency distribution
0.002
0.0018
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
0
200
400
600
800
1000
1200
Number of turns

3. **E = -1.25**


UNFOLDING TURNS AT E=-1.25
Frequency distribution
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
0
200
400
600
800
1000
1200
1400
1600
Number of turns

4. **E = -2.50**

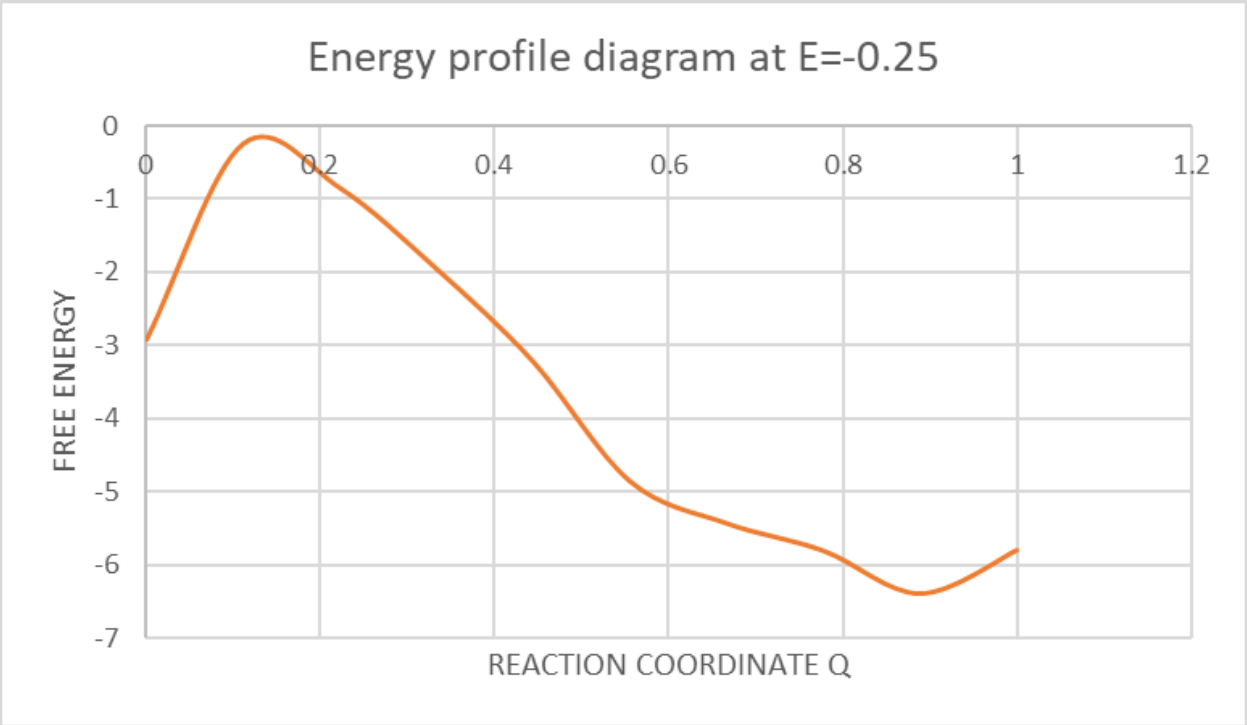


Reaction-coordinate and one-dimensional free energy profile

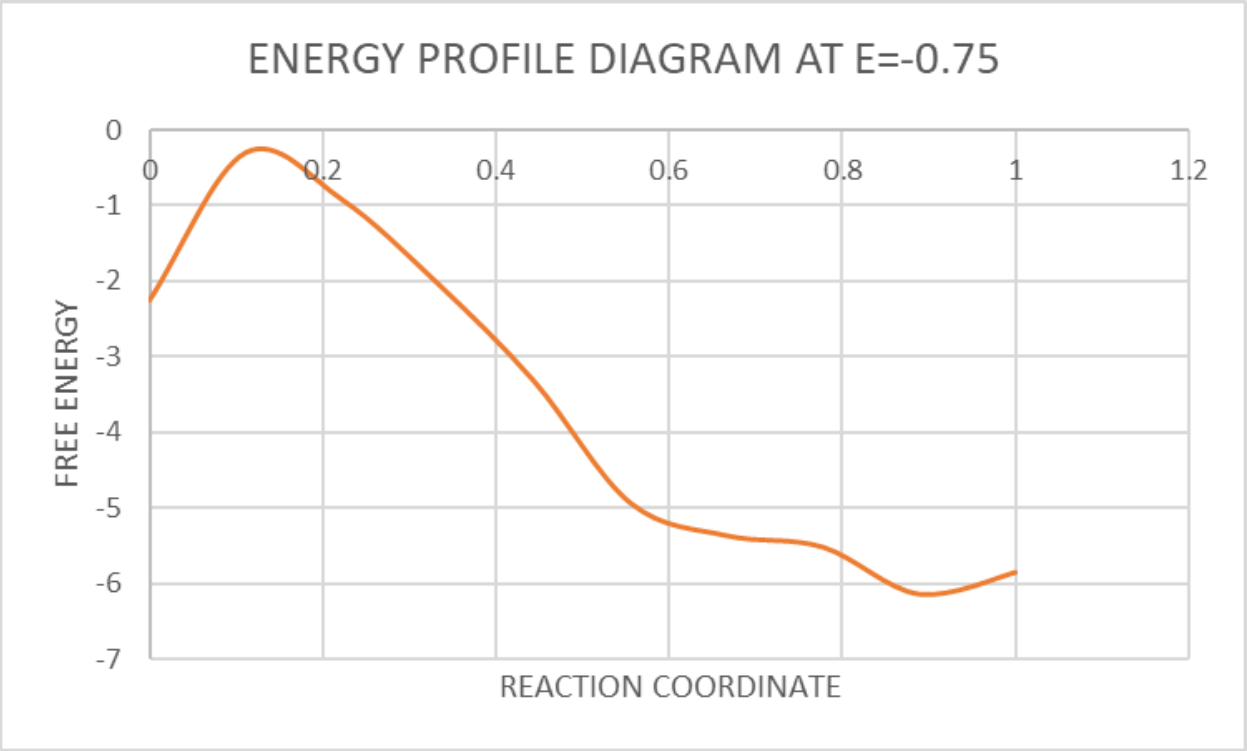
Reaction Coordinate

Since the Free Energy that I calculated was variable in the number of interactions (as detailed above), I took the reaction coordinate as **n/9**, where n is the number of interactions that go from 0 to 9 and I divided it by 9 to get it nicely from 0 to 1 as I have attached below:

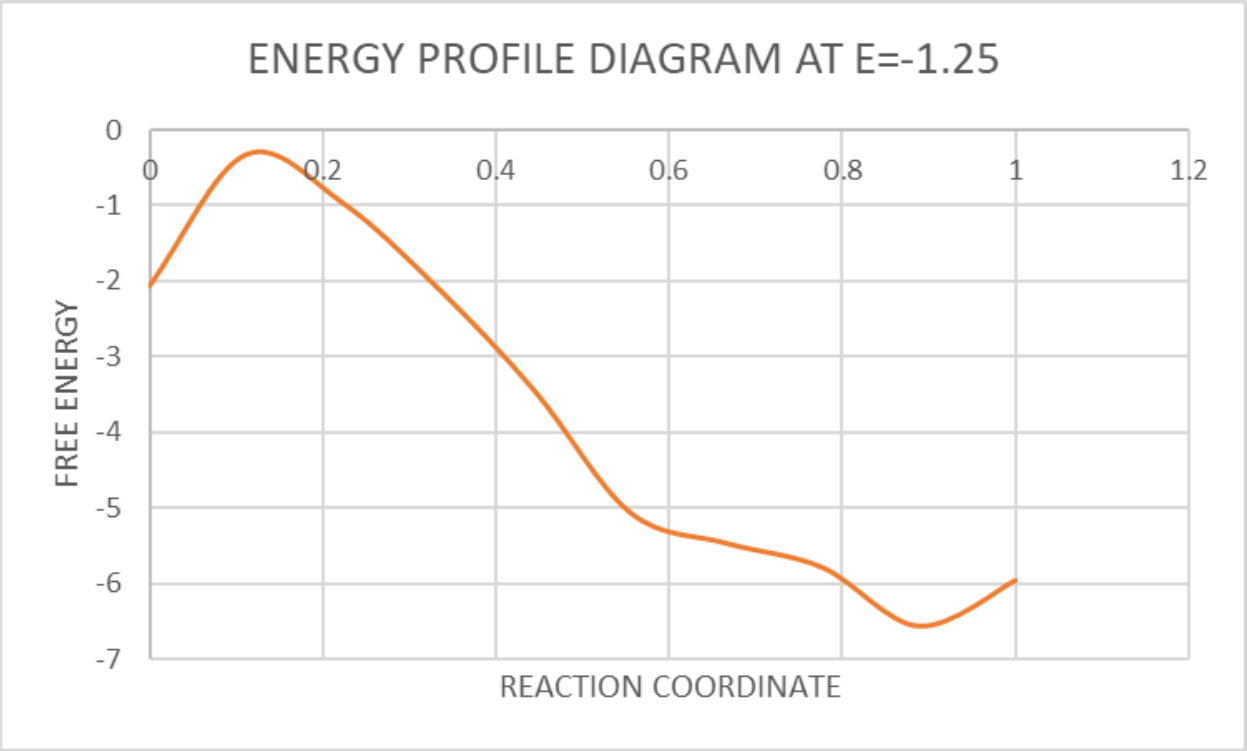
1. E = -0.25


Energy profile diagram at E=-0.25
FREE ENERGY
0
-1
-2
-3
-4
-5
-6
-7
0
0.2
0.4
0.6
0.8
1
1.2
REACTION COORDINATE Q

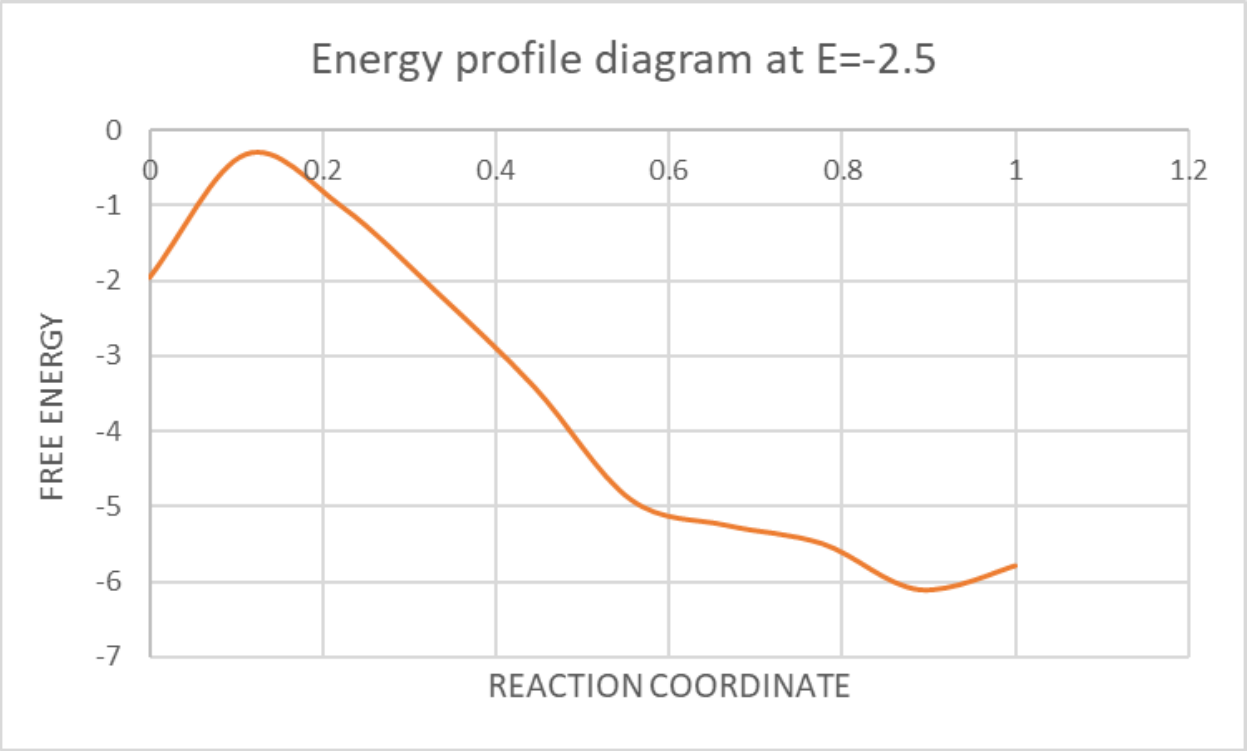
2. E = -0.75


ENERGY PROFILE DIAGRAM AT E=-0.75
FREE ENERGY
REACTION COORDINATE
0
-1
-2
-3
-4
-5
-6
-7
0
0.2
0.4
0.6
0.8
1
1.2

3. E = -1.25


ENERGY PROFILE DIAGRAM AT E=-1.25
FREE ENERGY
REACTION COORDINATE
0
-1
-2
-3
-4
-5
-6
-7
0
0.2
0.4
0.6
0.8
1
1.2

4. E = -2.50


Energy profile diagram at E=-2.5
FREE ENERGY
REACTION COORDINATE
0
-1
-2
-3
-4
-5
-6
-7
0
0.2
0.4
0.6
0.8
1
1.2

Microstates Sampled and Unfolding Criteria

Microstates Sampled

The table given below shows the number of microstates sampled for each of the 50 molecules at each energy value:

ENERGY:	-0.25 kT	-0.75 kT	-1.25 kT	-2.50 kT
1	5756	5954	5863	5124
2	5840	6360	5384	5060
3	6182	5979	5601	5491
4	5975	6239	5876	5061
5	5997	5802	5837	5059
6	5901	5487	5762	4921
7	6127	5778	5716	5368
8	6215	6313	6036	4703
9	5955	5635	5825	5277
10	6056	5529	5310	4920
11	6237	5624	5446	4878
12	5624	5872	5959	5186
13	5781	5763	5605	4894
14	6011	5861	5768	5329
15	6041	5843	5569	5202
16	5841	5368	6285	4979
17	5769	5731	6228	4915
18	5849	6059	5813	4937
19	5972	6442	6058	4924

20	6014	5629	5589	4743
21	6086	6015	5674	5230
22	5887	5344	5682	5009
23	5896	6179	5842	4600
24	6052	5906	5791	5050
25	5891	5922	5897	5211
26	5970	5703	5751	5099
27	5508	6019	5969	5185
28	5811	6226	5743	4943
29	6219	5614	5696	4934
30	6457	5677	5986	5181
31	6082	5937	5527	4764
32	5686	5811	5712	5010
33	5824	6040	5832	5080
34	6076	5946	6092	4955
35	6223	6091	5695	4750
36	6017	5670	5629	5003
37	5904	5694	5950	5242
38	5572	6045	6031	4748
39	6277	5938	5935	5032
40	6059	5933	6041	5097
41	5852	6055	5812	4866
42	6015	5738	6041	5125
43	5763	6122	5937	4867
44	5903	5911	6054	4944

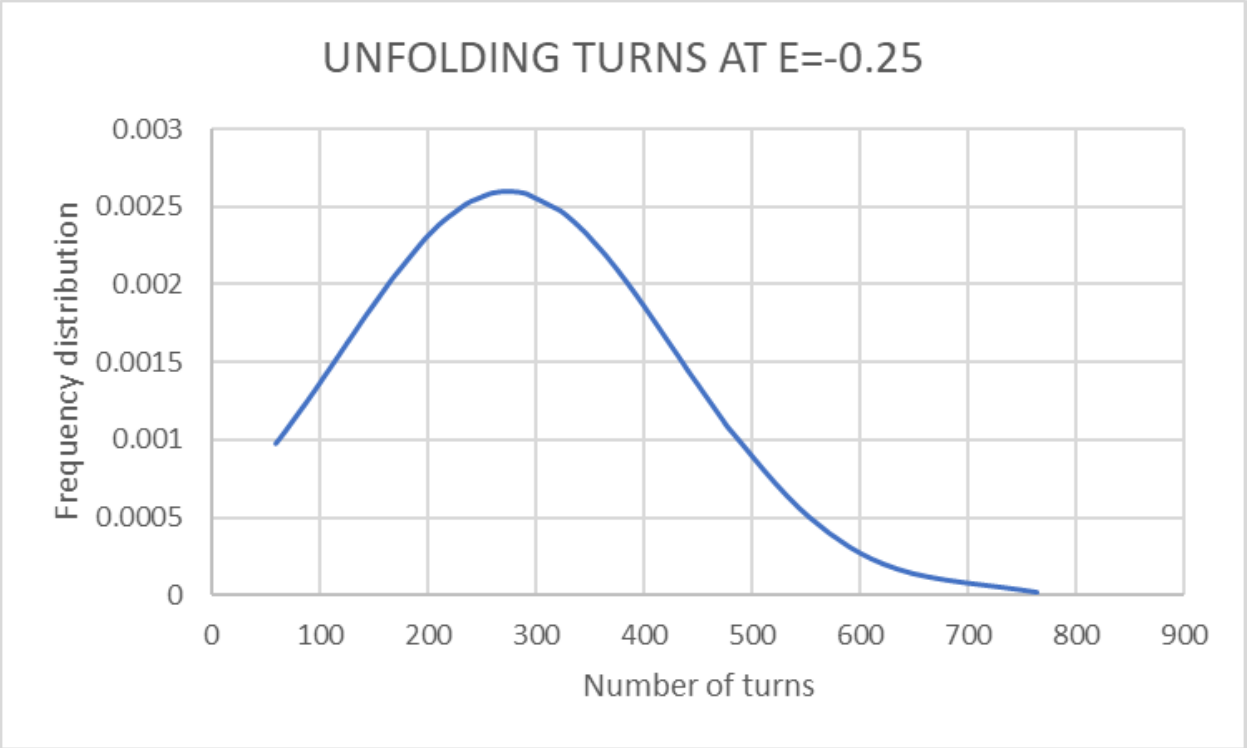
45	5719	5587	5838	4999
46	5751	6210	5726	5015
47	5677	5424	5540	5354
48	5846	5893	5988	5080
49	5714	5975	6086	5221
50	5998	6286	5566	5103

Unfolding Criteria

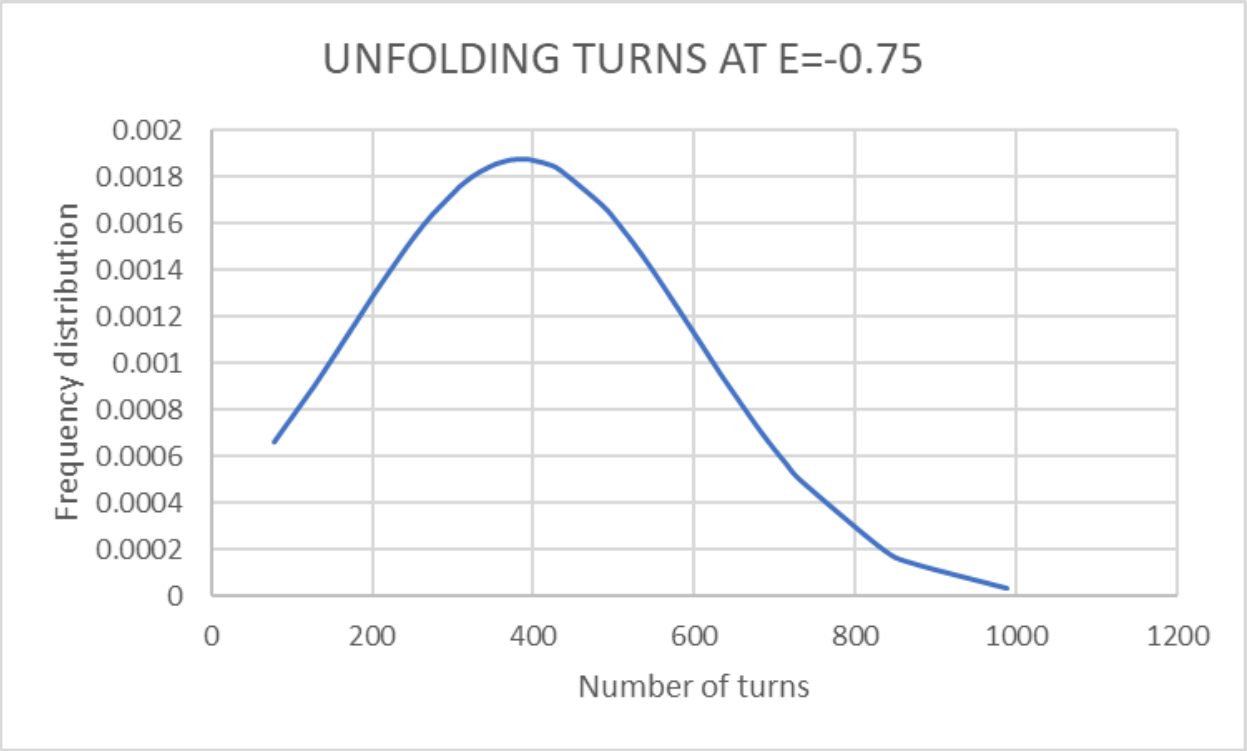
The criteria that I took for unfolding was the number of native interactions in a particular conformation. Basically, the number of native non-covalent interactions tell us about the unfoldedness of the molecule. Since when all the native interactions are lost, the protein loses its structure and function. Hence, the point at which the total interaction energy (or number of native interactions) becomes zero is considered as the unfolding point. I have already tabulated and plotted all the relevant data above.

Unfolding Distribution

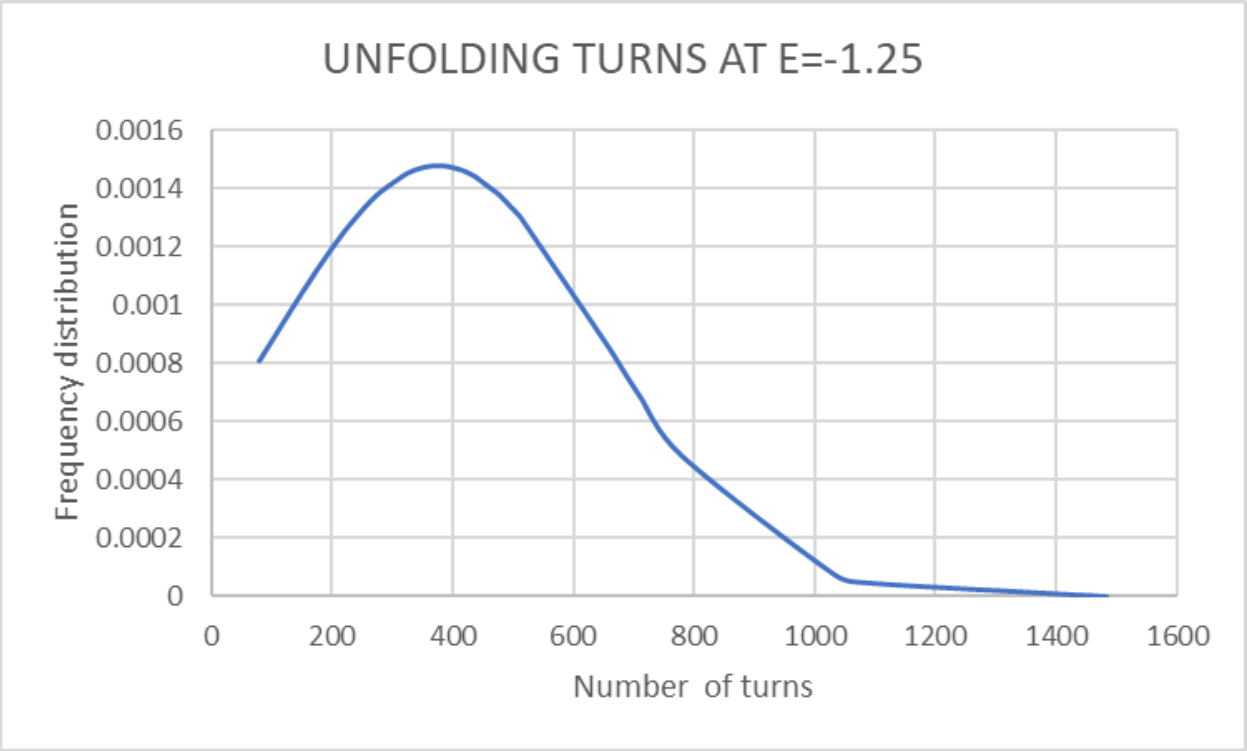
1. E = -0.25



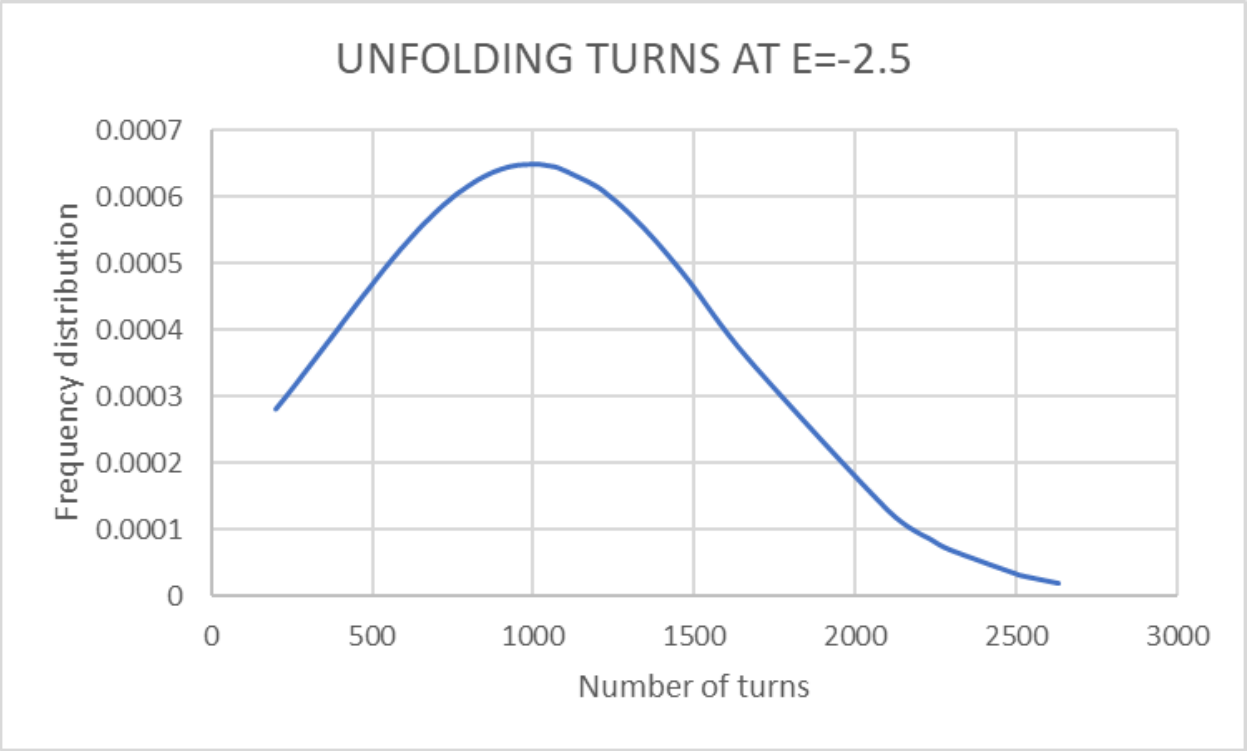
2. E = -0.75


UNFOLDING TURNS AT E=-0.75
Frequency distribution
0.002
0.0018
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
0
200
400
600
800
1000
1200
Number of turns

3. E = -1.25


UNFOLDING TURNS AT E=-1.25
Frequency distribution
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
0
200
400
600
800
1000
1200
1400
1600
Number of turns

4. E = -2.50



When all interactions are allowed

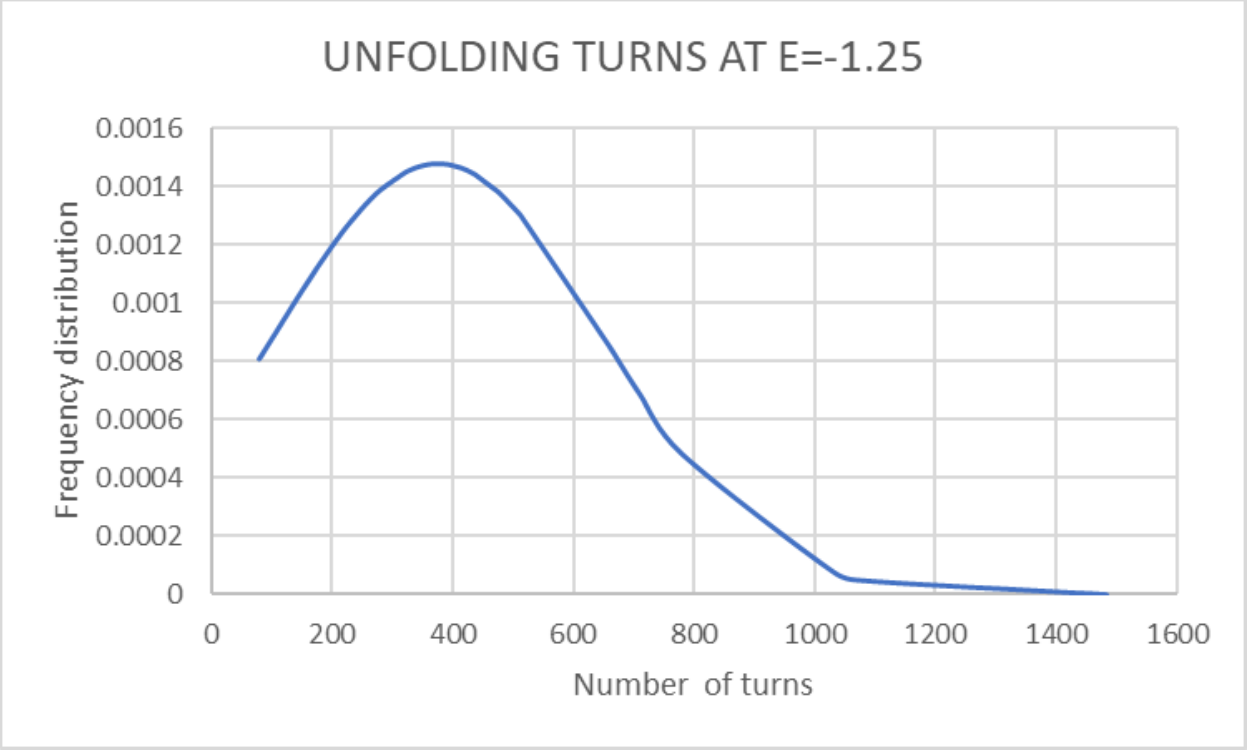
When all residues interact non-covalently with each other instead of just the native non-covalent interactions, we expect the molecule to take longer to unfold. Indeed, that's what we see when we simulate the unfolding and obtain the data.
I have tabulated the data below, in which I have compared the unfolding properties like the mean, median, standard deviation, minimum and maximum turns it took to unfold the molecule completely. These values are collected over 50 molecules each at E = -1.25. The second column is the one where only native interactions are allowed and the third column is where all residues interact non-covalently with each other.

UNFOLDING PROPERTY	**NATIVE INTERACTIONS**	**ALL INTERACTIONS**
MEAN	375.7843	743.68
MEDIAN	330	592

STANDARD DEVIATION	270.0979	421.7941
MINIMUM VALUE	79	200
MAXIMUM VALUE	1484	1947

Plots of Unfolding Turns:

1. Native Only



2. All Interactions