Protein folding project

Haakon T. Simensen

February 23, 2018

In this project, you will do statistical numerics on protein folding. An introduction to protein folding, necessary concepts and methods, as well as a proposed algorithm, are introduced in Sec. 1. The exercises are given in Sec. 2. To help you get started, we have added a "Getting started"-chapter in Sec. 3.

1 Introduction

1.1 Protein folding

A protein is a large molecule (polymer) which is built up by chains of amino acids (monomers). The amino acid sequence is referred to as the protein's primary structure [1]. Proteins are large molecules, that is long, and have in principle a huge degree of freedom in its spatial form. However, as any molecule in general "tries" to minimize its potental energy[1], the possible bondings between atoms at different positions in the molecule will eventually make the molecule fold. At sufficiently low temperatures, the molecule will stabilize in a local minimum in the potential energy. The protein's secondary structure refers to its local spatial structure. For instance, if hydrogen bonds are able to form at regularly spaced intervals, the protein may locally form a helix- or sheet-structure (for illustration, see *e.g.*: Protein secondary structure - Wikipedia). If we zoom further out, and look at the full molecule, we look at the protein's tertiary structure (for illustration, see *e.g.*: Protein tertiary structure - Wikipedia). The tertiary structure is what largely determines its biological properties. The energy of hydrogen bonds between amino acids in a water solution is $\approx (0.5 - 1.5)$ kcal/mol [2], which corresponds to $\approx (3.47 - 10.4)\cdot 10^{-21}$ J per bond. Compared to the thermal energy scale, $k_B T_0$, where k_B is Boltzmann's constant and $T_0 = 313$ K is typical room temperature, we find that the bonds have energies $\approx (0.8 - 2.4) k_B T_0$. These hydrogen bonds thus have energies within the temperature scale where life exists, which allows for dynamics to occur.

Proteins are important molecules in living organisms (at least for life as we know it). They perform a vast amount of jobs in the body, say DNA replication and intracellular transportation to name a few examples. A protein's structure is to a large extent what determines its biological function. That is, living organisms highly depend upon the protein folding happening in the correct manner. If the folding is not

[1] This is a bit simplified, but it will suffice to create an intuition of the physics in this problem.

performed correctly, we refer to it as misfolding. There actually is a separate class of diseases called *proteopathies*, recognized by misfolding of certain proteins, which include diseases like Alzheimer's disease, Creutzfeldt–Jakob disease and Parkinson's disease. Whether the protein misfolding is the cause or a product of the diseases is still under discussion, but we know for a fact that protein misfolding may cause the protein to fully alter its function. In fact, a vital protein may turn toxic if it forms certain structures.

Protein folding physics

Protein folding is, due to the discussion above alone, no doubt a very interesting topic. However, analyzing protein folding from a physics point of view is very hard indeed, as the number of possible structures is huge. We will therefore make some simplifications before we roll up our sleeves and start with the analysis. Firstly, the problem will be much easier to assess if we reduce the number of dimensions in which the protein may move to two. Secondly, the curvature of the protein is formally a continuous parameter, leading to an infinite number of possible protein configurations. We simplify this by allowing the protein to twist $\pm 90°$. In other words, our amino acids live on a two-dimensional grid, as illustrated in Fig. 1, where the dots represent amino acids and the lines represent bonds between them. We further assume that all amino acids occupy the same amount of space, that is one grid point. The amino acids differ only in their binding potential with respect to the other amino acids, that is in the possible binding strengths. In that sense, we end up with what seems like an incredibly simplified model. However, we can still learn from the results, as the results might still reflect the underlying physics which the realistic protein folding is subject to. As will be evident after you have completed this project, the generalization from 2D to 3D is straightforward.[2]

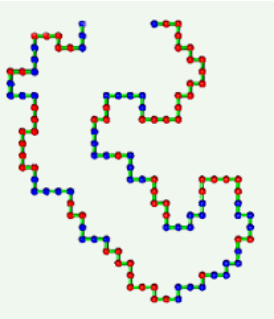

Figure 1: Illustrative image of a 2D protein. The image is downloaded from Ref. [3].

1.2 Monte Carlo methods

Having made major simplifications in the last paragraph, the problem is still a (too) challenging one. Say we want to calculate the mean binding energy of the protein, given that the potential between each amino acid is known. Denote the potential between the amino acids i and j at positions $\mathbf{r}_i$ and $\mathbf{r}_j$ as $U(\mathbf{r}_i - \mathbf{r}_j)$, and the expression for the mean energy would then follow as

[2]The reason to why you are going to do this exercise in 2D is that implementing the algorithms in 3D is harder from a programming point of view, as you will have to deal with an additional rotational degree of freedom. However, once you have succeeded in two dimensions, going to 3D should not be too hard!

$$\langle E \rangle = \sum_{S \in states} f(S) \sum_{i \neq j} U(\mathbf{r}_i - \mathbf{r}_j), \tag{1}$$

where S denotes a certain state (protein structure), f is the probability distribution of states, and we sum over all states. The challenge here is that we do not know the (discrete) function $f(S)$, and we know of no easy way of obtaining this function either. An analytic answer to this problem is therefore difficult to obtain. However, if we allow for a numerical method to solve the problem, we can solve it *approximately* with relative ease with the help of a computer.

Monte Carlo methods is an umbrella term which incorporates numerous algorithms that solve problems with the use of random sampling. It might sound weird that random sampling may in fact solve anything of interest, but the randomness is in fact what ensures that these methods work! To be concrete, we will in this project use the Metropolis algorithm, which is a Markov chain Monte Carlo method (MCMC). A Markov chain can be defined as a sequence of events, in which the next event only depends upon the present event. That is, for any event there is a probability distribution for the next events. There is a finite amount of possible events, and all events must somehow be within the reach from any other event. However, there is no requirement that all events must be within one step from any given event. A simple example of a Markov chain is if you decide to buy a random plane ticket (from a pool of all available tickets) at Værnes Airport, and fly to whatever destination you got. At your arrival, you immediately buy another random ticket, and fly to your next destination. During your journey, you keep record of which airports you visit. This is a Markov process, as the next possible destinations only depend on where you are, and not where you have been (we neglect difference in weekdays, time of day and such complicating factors). If you look at your record of airports after having travelled for a long time, you will probably see that airports such as Frankfurt, Dubai, Chicago O'Hare, Los Angeles, Beijing, Heathrow etc. are well represented, while you have been lucky if you have had the opportunity to visit *e.g.* Funafuti International Airport. Now, if we make a bar graph of airports vs. number of visits, what exactly are we looking at? We are looking at an empirical sample of the probability distribution of where to find you after a long time of travelling in the way described above! The law of large numbers states that if we have sampled over a sufficiently long time, the sample distribution will approach the real equilibrium distribution of the Markov chain.

If you don't like flying that much, we are able to reproduce the distribution above only by knowing the plane ticket distribution at each airport, and then simulate the ticket lottery on a computer with the use of (pseudo-)random[3] numbers. Instead of travelling for years, we only need a couple of seconds to obtain a good approximation of the real probability distribution. Now, that's an improvement in computation time! This is an example of a Monte Carlo method.

We are going to use the MCMC method on protein folding as well! Given that the protein starts as a linear structure, we know that it must twist somewhere in order to

[3]Computers are not able to generate completely random numbers, as the random-functions always follow some underlying algorithm. However, for this problem, it is sufficient that the probability distribution of the pseudorandom numbers equals that of random numbers.

start the folding. In the 2D system defined above, it may either twist clockwise or anti-clockwise (or no twist at all). After this twist, we allow for another twist, and so on. If we know the relative probability of these twists, we are able to approximate the probability distribution of each structure by iterating through a sufficiently long Markov chain of protein structures. Having approximated the probability distribution, that is $f(S)$ from Eq. (1), calculating quantities such as the (approximate) mean energy follows trivially.

1.3 System description and algorithm

We define a two-dimensional grid of size $m \times n$ on which a polymer consisting of N monomers (A_1, A_2, ..., A_N) lives. The polymer is defined by the following rules:

- The polymer is not allowed to split. The only situation in which a polymer is connected, is if all monomers have at least 2 nearest neighbours (except the end points, which must have at least 1 nearest neighbour). In other words, if we search for other monomers above, below, to the left and to the right of a given monomer, we need to find at least 2 other monomers (1 for the end points).
- The permutation of the monomers cannot be changed. That is, A_2 must be a nearest neighbour of A_1, A_3 must be a nearest neighbour of A_2 etc.
- Two monomers cannot occupy the same point on the 2D grid.

To summarize the points above: if we set our finger at the first monomer (A_1), we should be able to follow the monomers from A_1 to A_N by going up, down, right or left, without ever crossing the polymer. We introduce no boundary constraints, so we set $m = n \geq N$, so that all possible configurations can be placed inside the grid.

Subsequent monomers in the polymer are bound by strong chemical bonds, which by assumption will never break. We assume that apart from this strong interaction, the other non-subsequent monomers interact weakly, but only if they are nearest neighbours. That is, if A_1 is nearest neighbour to A_4, a weak bond forms between these two monomers. We define this interaction with a nearest neighbour interaction matrix U_{ij}, of which elements are decimal numbers chosen randomly between $-3.47 \cdot 10^{-21}$ and $-10.4 \cdot 10^{-21}$ (in units of J). This interaction matrix have elements equal to the function $U(\mathbf{r}_i - \mathbf{r}_j)$ defined in Eq. (1), but evaluated only at nearest neighbour positions. Note that to avoid weak interactions between connected monomers, we define U_{ij} to be zero if $|i - j| \leq 1$. Also define the binary nearest neighbour function n_{ij} to be equal to 1 if monomers A_i and A_j are nearest neighbours, and 0 otherwise. The energy function of the polymer is then defined as

$$E = \sum_{i,j}^{N} U_{ij}\, n_{ij}, \tag{2}$$

where the summation over both i and j runs from 1 to N.

The polymer is subject to twists (or "rotations"), in which a part of the polymer is rotated by 90°, clockwise or anticlockwise, with the rotation center at a given

monomer. For the twist to be allowed, the new twisted structure must fulfill the three requirements stated above. There are two mechanisms which generate the twists:

- The polymer naturally seeks to minimize its potential energy. Therefore, if the energy of the twisted structure E^t is less than the previous structure E^0, *i.e.* if $E^t - E^0 \equiv \Delta E < 0$, the twist will be physically realized.
- The second driving force is thermal fluctuations, which may cause the polymer to enter a twisted state even though $\Delta E > 0$ for the process. Formally, we use the rule that a twist is realized if a randomly chosen decimal number between 0 and 1 is less than $e^{-(E^t-E^0)/k_\mathrm{B}T}$, where k_B is Boltzmann's constant.

Given a polymer structure, we thus know how to find the next structure. That is, we can now make a Markov chain of protein structures, which we eventually will use to approximate a probability distribution. Note that protein folding is not only dependent upon a simple relative probability, but also depends on a discrete accept/reject-requirement given by the energy difference. However, we are still able to assess the probability for the protein to make a twist, and the Markov chain requirements are still fulfilled. This special case of MCMC is called the Metropolis algorithm.

Physical justification of the algorithm

The first mechanism above is quite intuitive, as we are used to physical systems which are driven towards a potential energy minimum. An illustrative example is that things you drop from a height have a very consistent tendency to hit the ground. Note however that if an expensive china vase balances at the edge of a table at zero temperature (and if we neglect the extremely small and indeed negligible quantum fluctuations), it will not fall to the ground, although the potential energy for the system would be reduced if the vase hit the floor. This is an example of a physical system which is trapped in a *local minimum* (if the vase balances perfectly, it is strictly speaking a *saddle point*). That is, in the case of a local minimum, the vase would require a temporary increase in potential energy in order to fall over the edge of the table, and this process does therefore not happen without help. This is however where thermal fluctuations come into the picture and (possibly) breaks apart your expensive vase! Thermal fluctuations are of a statistical nature, and we therefore need to apply statistical physics[4] in order to evaluate the effect. In statistical physics, we define a quantity Z named the (canonical) partition function, defined by

$$Z = \sum_{\{\mathrm{ms}\}} e^{-\beta E_\mathrm{ms}}, \tag{3}$$

where ms is shorthand notation for microstates, and $\beta \equiv 1/k_\mathrm{B}T$. That is, we sum over all microstates of the system, and weight each microstate by $e^{-\beta E_\mathrm{ms}}$, where E_ms is the energy of the microstate. It turns out, that a thermal expectation value of an the energy E can be expressed as

[4] Statistical physics is a 3rd year course on NTNU (TFY4230). We introduce a taste of it here, but do not be afraid, as we only need it to justify the proposed algorithm.

$$\langle E \rangle = \frac{1}{Z} \sum_{\{\text{ms}\}} E_{\text{ms}} e^{-\beta E_{\text{ms}}}. \tag{4}$$

By analyzing this equation from a statistical viewpoint, we immediately recognize $p(i) \equiv \frac{1}{Z} e^{-\beta E_i}$ as the probability for finding the system in microstate i, and that the partition function enters as a normalization constant. It follows that the probability ratio for finding the system in state i rather than in state j is

$$\frac{p(i)}{p(j)} = e^{-\beta(E_i - E_j)}. \tag{5}$$

This is a very useful result for us! We only need the energy difference $\Delta E = E_i - E_j$ and the temperature T in order to know the probability ratio. We can now go back to our protein folding problem. Eq. (5) gives us the probability ratio between the twisted and un-twisted state if we let $E_i \to E^t$ and $E_j \to E^0$, and takes a value between 0 and 1 as long as $E^t > E^0$. The thermal fluctuations are of a random nature, and we use a (pseudo-)random number generator to simulate this randomness. The selection rule for the thermal process must therefore be that the twist is realized if the probability ratio exceeds the random number.

1.4 Example of a numerical procedure

We will now go through an illustrative example which later will help you solve the exercises. This is however not an exercise, only an example where we show the proposed algorithm: We want to calculate the mean energy and mean diameter of a polymer as function of temperature. The procedure is as follows:

1. Start with a straight polymer, that is with all monomers aligned. Start at $T = 0$.

2. *Try* to twist the polymer around a random monomer in a random direction. If the twist is not allowed by the rules we defined for the protein in the beginning of Sec. 1.3, it does not count as a twist, and you will have to try new twists until you find a legal one before you proceed. An illustration of twists is given in Fig. 2. For each protein state, calculate the binding energy and polymer diameter. Repeat this procedure a suitable amount of times d. We emphasize *try*, as if the twist is not energetically favored, nor "forced" by thermal fluctuations, the polymer will not move. The requirement for d is that the structure iterates through a sufficient amount of states so that the convergence is satisfactory. The lower the temperature, the slower is the physics, and the larger d you will need. For a polymer of 15 monomers, you should for instance *definitely* not use d lower than 10000 for low temperatures. A natural choice of d is $d(T) = d_{\max} \exp\{-sT\}$, where $d_{\max}$ is the maximum d (for $T = 0K$), and s is a parameter which defines the slope of $d(T)$. We will not give an absolute requirement for $d_{\max}$ and s, so you will have to find reasonable values yourself.

3. Define a temperature interval from $T = 0$ K to $T = T_{\text{end}}$, and split this into N_T intervals. Increase the temperature stepwise by $\Delta T = T_{\text{end}}/N_T$, repeat the

procedure above for every temperature between $T = 0$ K and T_{end}. Calculate for instance the mean binding energy and mean diameter for each T.

To be more concrete, an example of a possible pseudocode for a protein of N monomers follows below. It assumes that N is odd, but it can easily be generalized to even N's. It also assumes that the indexation of arrays starts at 1. For simplicity, we have also assumed a constant d in the pseudocode.

Algorithm 1 Protein folding on a 2D grid

```
grid ← (N × N) array of zeros
grid ← assign values 1 to N from left to right in the center row. The numbers
symbolize monomers A_1, ..., A_N in the polymer.
ϵ ← (N_T × d) array of zeros
L ← (N_T × d) array of zeros
ϵ(1, 1) ← 0
L(1, 1) ← N
for i = 1 to N_T do
    Reset grid s.t. the polymer is horizontal
    for j = 1 to d do
        E ← energy of grid
        Pick a random point around which to perform twist
        Pick clockwise or anticlockwise twist (randomly)
        copyGrid ← twisted version of grid
        if legal twist then
            E_new ← energy of copyGrid
            if E_new < E then
                grid ← copyGrid
            else if RAND(0,1) < exp{−β(E_new − E)} then
                grid ← copyGrid
            else
                Do nothing
            end if
        else
            Try a new twist until the twist is legal
        end if
        ϵ(i, j) ← E
        L(i, j) ← diameter of polymer
    end for
end for
```

Useful tips:

- You are always free to choose which "end" of the polymer you want to rotate when you twist. Without loss of generalization, you can choose the twist s.t. the center monomer $A_{(N+1)/2}$ is stationary in all twists. Doing so, you ensure that the

polymer won't escape your grid. Alternatively, you can use periodic boundary conditions.

- This task is suited for object oriented programming. You can for instance define two classes, Protein and Grid, and give them useful member variables and functions. Doing this, we can easily repeat the procedure above for different values of N_T and d, and the code will (hopefully) stay tidy. You are however free to structure your code as you like.

- Depending on your code, you may experience that the calculations require a lot of runtime, especially for long polymers. There is a number of ways to cope with this problem. Firstly, try to use built-in numpy functions (if you use Python) where they can replace a for-loop. These will generally run faster, especially if you end up doing matrix operations. Secondly, develop your code at low numerical accuracy, and use the desired accuracy first when you want to produce results. The inevitable debugging process will then hopefully speed up. Those who want a challenge may also try to run some computations (those which can be parallelized) in parallel, that is use several CPU cores. Python libraries such as joblib make this rather easy. However, this is absolutely not a requirement, and is not recommended for those who do not enjoy deciphering obscure errors.

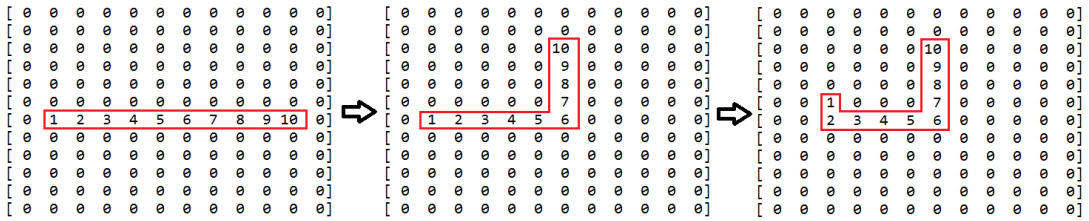

Figure 2: A visualization of two twists performed on a protein, where the visualization is as described in the problem text. The protein has been twisted around monomers nr. 6 and 2. Note that the twists are performed s.t. monomer nr. 5 is fixed.

2 Exercises

The exercises follow below. Those exercises where you are expected to give an explicit figure or answer a specific question in the report are marked in **bold** font. If you find the exercises hard, we have added a *Getting started*-section after these exercises which may push you in the right direction.

1. Visualization to ensure correct behaviour

   1.1. Find a suitable way to visualize a protein of arbitrary length on a 2D grid. You can for instance follow the example under Sec. 1.4, and associate the number 1 with the first monomer, 2 with the second, etc., and associate 0 with the grid points where there are no monomers.

   1.2. Define the necessary functions that twist a part of the protein clockwise and anti-clockwise around a random position. Use the visualization of the protein to confirm that the functionality is correct. Also make sure that no illegal structures are allowed to appear (see Sec. 1.3). **Include** a figure that contains three plots: a plot of a straight polymer consisting of 10 monomers, a plot of the polymer after one random twist, and a plot of the polymer after two random twists. The plots may for instance be screen-shots of your way to visualize the protein, placed side-by-side inside one figure.

2. Energy as function of temperature

   2.1. Define a straight polymer consisting of 15 monomers. Follow the example in Sec. 1.4, and **plot** the mean energy, $\langle E \rangle$, as function of temperature. Choose the relevant parameters as in Sec. 1.3. That is, choose U_{ij} randomly between $-3.47 \cdot 10^{-21}$ and $-10.4 \cdot 10^{-21}$ (in units of J), and vary T between 0 K and 1500 K. You need to do $d(T)$ twists at each temperature T, where $d(T) = d_{\max} \exp\{-sT\}$, as explained in Sec. 1.4. **Choose** $d_{\max}$ and s so that you obtain a sufficiently convergent[5] result,.

   2.2. Start with a straight 15-monomer long polymer at $T = 0$ K and perform 5000 twists. **Plot** the binding energy E as function of number of twists for this temperature. That is, show how E evolves when the protein is subject to twists. Repeat this procedure with $T = 500$ K (5000 twists), and **place** these plots side-by-side. **Compare** the plots, and explain why these plots can be used to assess whether or not the number of twists was sufficient to approximate the probability distribution of states.

   2.3. For temperatures $T \gtrapprox 100$ K, the trend should be that $\langle E \rangle$ increases for increasing T. **How** can we interpret this result physically? **What** happens to the slope of $\langle E \rangle$, that is $\mathrm{d}\langle E \rangle / \mathrm{d}T$, when T is large? **Give** a physical explanation for this.

[5] Convergence in this sense means that the results stabilize. If sufficient convergence is reached, increasing the number of twists per temperature will not affect the results within the desired level of accuracy.

2.4. **What** is the behaviour of $\langle E \rangle$ for low T? In order to see this, it is useful to calculate $\langle E \rangle (T = 0 \text{ K})$ several times, and compare the results. **Give** a physical interpretation of this.

2.5. Repeat point 2.1. for a polymer consisting of 30 monomers. **Plot** $\langle E \rangle (T)$, and **suggest** the parameters $d_{\max}$ and s to obtain a sufficiently convergent result. **Which** factors make the numerical complexity for this structure larger compared to the polymer of 15 monomers?

3. **Plot** the protein's mean diameter, $\langle L \rangle$, as function of temperature for a polymer of both 15 and 30 monomers. **Give** a physical interpretation of the results. *We define the diameter of the protein as the longest possible distance between two monomers. Define each monomer to have length 1 both horizontally and vertically.*

4. Gradual cooling of a polymer

4.1. Start with a straight polymer consisting of 15 monomers at $T = 1500$ K. Gradually decrease the temperature as you perform twists on the polymer. For instance, do 600 twists at $T = 1500$ K, 600 twists at $T = 1470$ K, and keep on doing this down to $T = 0$ K. The grid should not be reset to horizontal position when the temperature is lowered. **Plot** E as function of number of twists, and mark the points where the temperature drops. **Comment** on the behaviour of E as the temperature drops. At $T = 0$ K, **does** the protein settle in (what seems like) the global minimum or a local minimum? *Note: if you use too many rotations per temperature, the plot will be completely unreadable. Therefore, I suggest to use about 30000 rotations in total from* $T = 1500$ K *to* $T = 0$ K, *as well as plotting with a thin line*.

4.2. **Plot** $\langle E \rangle$ as function of temperature for the cooldown process. **What** do you observe for $\langle E \rangle (T = 0 \text{ K})$ when you gradually cool down the system, as compared to the result for $\langle E \rangle (T = 0)$ when you started with a straight polymer and did twists only at $T = 0$ K? **Give** a physical interpretation of this. *Hint: relate this to the concept "local minima".*

4.3. **Plot** $\langle L \rangle$ as function of temperature for the cooldown process. **Describe** the function, and **reason** why it looks the way it does.

4.4. Do several such cooldowns for a polymer and visualize the polymer at the end ($T = 0$ K). **Plot/draw** one of these polymers, and **comment** on why the polymer got "trapped".

4.5. Repeat the points above for a polymer of length 30, and **compare** it with the polymer of length 15. That is, **give** the same plots as above and **compare**. A tidy way of doing this is to present the plots side-by-side as subfigures.

Good luck and happy programming!

Something to think about (not an exercise!):
Eggs consist of a lot of proteins. In a raw (cold) egg, these are curled up into their tertiary structures. What happens to the proteins when the egg is boiled? Can you relate this to the hardening of the egg? Why doesn't the egg become liquid-like when you cool it down after having boiled it for 10 minutes?

3 Getting started

If you have trouble getting started with the exercises, we will give you some help here. We will suggest how to structure your program, but note that this is only one of many possible ways to do so, and we recommend that you make your own solution to this. These suggestions do not form a complete solution of anything, so you will have to fill in the blanks yourself.

1. Define a $n \times n$ zero-matrix with the python syntax:

```
import numpy as np
grid = np.zeros((n,n)).astype(np.int16)
```

Then place the polymer of length N inside the grid. Do this by associating the number 1 with the first monomer, the number 2 with the second, etc. This can be done with the following syntax:

```
grid[int(np.round(n/2)), int(np.round(n/2 - N/2)): \
int(np.round(n/2 - N/2)) + N] \
= np.linspace(1,N,N).astype(np.int16)
```

You should then make a function which twists the protein around monomer nr. x (with $1 < x < N$). With *clockwise* being a boolean variable, it should look something like this:

```
def isLegalTwist(grid,x,clockwise):
        #Check if twist is legal
        return True / False

def twist(grid,x,clockwise):
        if isLegalTwist(grid,x,clockwise):
                #Twist the protein and continue
        else:
                #Try again
        return twistedGrid
```

We have now defined the basic functionality of the protein!

2. It is useful to declare functions which calculate the energy of the polymer.

```
def nearestNeighbours(grid, n, U, x, y):
#Calculates the energy contribution from nearest
#neighbour interactions on site (x,y)
        E = 0
        if x+1 < n:
                E = E + U[grid[x+1,y],grid[x,y]]
        if x-1 >= 0:
                E = E + U[grid[x-1,y],grid[x,y]]
```

```
        if y+1 < n:
                E = E + U[grid[x,y+1],grid[x,y]]
        if y-1 >= 0:
                E = E + U[grid[x,y-1],grid[x,y]]
        return E

def calculateEnergy(grid, N):
#Returns the total energy of the grid
        E = 0
        for i in range(1,N+1):
                pos = np.argwhere(grid == i)[0]
                E = E + 0.5 * nearestNeighbours(pos[0],pos[1])
                # multiply by 0.5 to avoid double counting
        return E
```

3. We can now compare the twisted and untwisted grid. Below is a sketch of the selection mechanism in the Markov chain.

```
import numpy.random as rand

x = rand.randint(2,N)
clockwise = rand.randint(0,2)
twistedGrid = twist(grid, x, clockwise)
E_untwisted = calculateEnergy(grid,N)
E_twisted = calculateEnergy(twistedGrid,N)
if (E_twisted < E_untwisted):
        #Continue with the twisted grid
elif (rand.rand(1,1)[0,0] < \
np.exp(-(E_twisted-E_untwisted)/(k_B * T))):
        #Continue with the twisted grid
else:
        #Continue with untwisted grid
```

References

[1] "Protein structure", *Wikipedia*. URL: https://en.wikipedia.org/wiki/Protein_structure (Accessed: 16th February 2018).

[2] Williams, D. H., Searle, M. S., Mackay, J. P., Gerhard, U. and Maplestone, R. A. (1993) *Proc. Natl. Acad. Sci. USA* **90**, 1172–1178.

[3] Lam, F., Negara, G. and Stewart, A. *The ProFolding Project*. URL: http://www.brown.edu/Research/Istrail_Lab/ProFolding.php (Accessed: 16th February 2018).