Calculating Protein Mass

Purpose, and theory

Polypeptides are chains of amino acids joined by a peptide bond. Knowing the mass of a polypeptide is important for many applications, such as identification of an unknown protein, and assessment of unique features the protein may possess.

The theory behind protein mass is that the mass of each amino acid must be accounted for when calculating the mass of a polypeptide. That is easily done. However, there is a complication that means that it is not as simple as summing the mass of each amino acid.

During the formation of a protein, lets say two amino acids combine together, they undergo a condensation reaction; this is where a molecule of water of given off when forming a peptide bond. Therefore, we cannot simply sum the masses of the amino acids because their atomic constitution changes with the formation of a peptide bond.

However, the problem is further complicated because we need to know the number of molecules of water given off. To find this, we must first cover the bonding process during the formation of a peptide.

Shape

Description automatically generated with medium confidence

Rectangle

Description automatically generated with low confidenceFigure 1. This is the structure of the simplest amino acid – glycine

The red square indicates the location of where the peptide bond forms. This is a condensation reaction where the hydroxyl group from the amino acid on the left, and the hydrogen atom from the amino acid on the right, react together in an elimination reaction to form a peptide and water.

Icon

Description automatically generated

Icon

Description automatically generated with medium confidenceIn the red square is the newly formed peptide bond. For every two amino acids, there is one molecule of water given off.

There are three amino acids joined together here, as indicated by the two peptide bonds.

We can now formulate a solution to calculating the number of water molecules released when a peptide is formed.

***Number of water molecules produced = number of amino acids - 1***

We can now form a general formula for calculating the mass of the protein:

***Mass of protein = sum of amino acid masses – number of water molecules produced***

This can be written as:

***MP = ∑ (MA) – (NA – 1)***

Where:

* **MP** is the ***mass of the protein***
* **MA** is the ***mass of the amino acid***
* **NA** is the ***number of amino acids***

Solution Implementation

First of all, the amino acid sequence of the polypeptide must be retrieved from the user. This proves a trickier problem to solve because unlike reading a DNA sequence which is composed of uniform triplets, amino acids have names with different numbers of letters. I think the most holistic solution for this is to have the user input a sequence of amino acids separated by a comma. If there are spaces between the commas it can interfere with identifying the amino acid, so the spaces should be removed before further processing. This can be achieved by using the *.replace(find, replace with)* command:

aa\_sequence = aa\_sequence.replace(" ", "")

The sequence is also made lowercase to reduce problems in identifying the amino acid by using the *.lower()* command.

On the end of the sequence a comma (,) is added because of the way the amino acids are read, which will be explained next.

The way this application reads a sequence is it finds the first comma in the sequence and stores its index. Up to this index is the position of the amino acid, which is read, and stored as a string under the ‘aa’ variable. This amino acid is then compared to a list of amino acids to find the index of it within the list. This new index is then used to find the complementary mass of the amino acid in the ‘aa\_masses’ list. Once the amino acid mass has been appointed, the user sequence is shortened by cutting off the prefix including the just analysed amino acid, and its corresponding comma:

aa\_sequence = aa\_sequence[len(aa)+1:]

This is achieved by finding the length of the currently analysed amino acid, and adding 1 to it to accommodate the comma. The analysis process is repeated via a *while* loop until an error is returned, in which case the loop stops and the processed values are returned for final processing.

The program also accepts three-letter amino acid names, as well as one-letter name sequences. Depending on which one you use, the corresponding class is chosen for analysis. The theory behind reading, comparing, and finding the mass of the amino acid is the same process whether you are using the full-name, three-letter name, or one-letter name. The difference for the latter two is that the user sequence is shortened by three-units or one-unit respectively. This is achieved by:

aa\_sequence = aa\_sequence[1:]

The user’s sequence is called ‘aa\_sequence’. Using the indexing in Python, with each repetition of the *while* loop the sequence is read from the next letter in the sequence (or three letters).

The mass of the water is calculated as mentioned above. The number of amino acids is stored as the integer variable *n\_aa* whose value increases by 1 with each repetition of the *while* loop. The value stored within the variable has 1 subtracted from it (as explained earlier), and is then multiplied by the mass of water (18.01056 Da). However, it is important to mention that this code within the application is optional, and can be enabled if required. It is not necessary for calculating the protein mass because the values stored within the .CSV file are the ***residue*** masses which means they account for the loss of a water molecule. If however, you are using basic amino acid weights, then the algorithm needs to be enabled by removing the corresponding hashtags under the heading ‘*water mass calculator*’.

Improvements to the Solution

The method I used for locating the complementary mass of each amino acid in the sequence could be made more efficient. My aim with this application was to try and use basic Python libraries so that it is a ready-to-use application, which meant that my solution had to write the contents from the .CSV file to a list variable. Although computationally this is negligible because the list is only written to once, and there are only 23 amino acids, there could be a better solution. One such solution I have found was using the Pandas library. It is said that if there is anything to do with .CSV, Pandas is the number one choice. However, the issue I ran into was that the documentation for .CSV file analysis with Pandas was ambiguous (at least at the minute), and the online forums didn’t help me get a working prototype. To overcome this issue, one of my projects in the near future will be to get a working solution to find and extract data from .CSV files using Pandas.