

FPP: PDB parser and parameters calculation in Python

Alessandro Lambertucci

15-02-2023

The bottom half of the page features a decorative design with large, overlapping triangles. A large yellow triangle points upwards from the bottom right corner. Two orange triangles point downwards from the bottom left corner, overlapping each other and the yellow triangle.

1 Abstract

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

With this script we want to demonstrate the usefulness of python to automate various tasks in computational biology.

2 Introduction

FPP is a software written in python which is capable of fastly computing various proteins parameters, such as: **RMSD**, **center of mass**, **molecular weight**, **volume** and **end-to-end distance** from protein-data-bank (**PDB**) files.

RMSD (root mean square deviation of atomic position) is a metric often used to evaluate:

- the difference between the predicted binding pose and the crystallographic pose.
- the diversity in multiple generated conformers.
- the accuracy of protein structure prediction software once the original structure has been resolved with NMR or x-ray crystallography.

Such program is more a proof of concept, rather than a stable production-ready software, of what could be theoretically achievable by integrating the use of python in biology.

3 Implementation

To choose which **PDB** files you want to plug in the script you can either run them as arguments, bearing in mind you need to be in the same folder as where they are located or plug the relative path to them, as shown below:

```
python FPP.py xyz_1.pdb xyz_2.pdb
```

or you can edit the code and point the two **PATH** variables to the absolute path of where the files are located in your home directory depending on your os:

```
PATH_1 = "C:/Users/username/Documents/3epo.pdb"  
PATH_2 = "C:/Users/username/Documents/1elo.pdb"
```

3.1 RMSD

The **RMSD** is defined as:

$$RMSD = \sqrt{\frac{1}{n} \sum_{i=1}^n (d_i)}$$

where:

$$d_i = (x_{i,nat} - x_{i,model})^2 + (y_{i,nat} - y_{i,model})^2 + (z_{i,nat} - z_{i,model})^2$$

This means we iterate across n atoms, and take the difference in their x -, y -, and z -coordinates. We then take the square root of the average deviation.

This formula can be either used to calculate **RMSD** across all atoms of two protein structures, or a subset, as long as n is identical between the two structures.

This is implemented in python as follows:

```
import numpy

def compute_RMSD(fixed, moving):
    if fixed.size == moving.size:
        rmsd = numpy.sqrt((((fixed - moving) ** 2) * 3).mean())

    return rmsd
```

3.2 Align

Before we calculate the **RMSD**, we need to align our structures, for two reasons:

- PDB structures may be located in different starting coordinates in our force field.
- There can be small differences in length between the two structures.

In python, the easiest way to align the two structures is to use the package **Bio.SVDSuperimposer** from **BioPython**.

SVDSuperimposer finds the best rotation and translation to put two point sets on top of each other (minimizing the **RMSD**).

SVD stands for singular value decomposition, which is used in the algorithm.

This is implemented in python as follows:

```
try:
    superimposed = SVDSuperimposer()
    superimposed.set(atom_pos_1, atom_pos_2)
    superimposed.run()
    rmsd = superimposed.get_rms()
    print("The computed RMSD is: ", round(rmsd, 6), "Å")
except:
    print("Couldn't compute RMSD; proteins have different backbone length")
```

3.3 Center of Mass

The **center of mass** of a distribution of mass in space is the unique point where the weighted relative position of the distributed mass sums to zero.

This means the **center of mass** is the **mean location of a distribution of mass in space**, similarly to a weighted average.

The formula for the **center of mass** on the **x**- axis is:

$$\bar{x} = \frac{\sum_{i=1}^n m_i x_i}{\sum_{i=1}^n m_i}$$

And similarly for the **y**- and **z**- axis.

This is implemented in python as follows:

```
for line in pdb.readlines():
    if line.startswith("ATOM"):
        x = line.split()
        atom_mass_sum += float(atom_mass_dict[x[11]])
        sum_of_mass_x += float(x[6]) * float(atom_mass_dict[x[11]])
        sum_of_mass_y += float(x[7]) * float(atom_mass_dict[x[11]])
        sum_of_mass_z += float(x[8]) * float(atom_mass_dict[x[11]])

center_mass_coord = (center_mass_x, center_mass_y, center_mass_z)
```

Atomic mass is determined for every atom through a *key:value* reference on an **atomic mass dictionary** displayed below:

```
atom_mass_dict = {
    'C': '12.011',
    'O': '15.9994',
    'N': '14.0067',
    'S': '32.06',
    'H': '1.00784'
}
```

Hydrogens are/aren't considered for computing the center of mass depending on the **PDB** file; if hydrogens are present they will be considered, else they will not.

3.4 Molecular Mass

The **molecular mass** is the mass of a given molecule, measured in **Dalton (Da)**.

This is implemented in python as follows:

```
for line in pdb.readlines():
    if line.startswith("ATOM"):
        x = line.split()
        if x[3] != 'AA':
            AA = x[3]
            pro_mass += float(residue_mass_dict[x[3]])
```

Residue mass is determined for every residue through a *key:value* reference on a **monoisotopic aminoacid mass dictionary**, in part displayed below:

```
residue_mass_dict = {  
    'ALA': '71.03711',  
    'CYS': '103.00919',  
    ...  
    'VAL': '99.06841',  
    'TRP': '186.07931',  
    'TYR': '163.06333'  
}
```

In this case **hydrogens** are always considered for every **residue** in the chain.

3.5 Volume

Molecular volume is computed by dividing the **molecular mass** for the **average protein density** 1.35 g/cm³ converted to roughly 0.8129891 Da/Å³:

3.6 End-to-End Distance

The distance between the **N-terminal end** and **C-terminal end** can be simply calculated with the formula to determine the distance between two points in a 3 dimensional plane:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

This is implemented in python as follows:

```
for line in pdb.readlines():  
    if line.startswith("ATOM"):  
        x = line.split()  
        if flag:  
            first_x = float(x[6])  
            first_y = float(x[7])  
            first_z = float(x[8])  
            flag = False  
        first_last_x = float(x[6]) - first_x  
        first_last_y = float(x[7]) - first_y  
        first_last_z = float(x[8]) - first_z  
  
distance_FL = round((((first_last_x ** 2) + (first_last_y ** 2) +  
                        (first_last_z ** 2)) ** (1 / 2), 3)
```

3.7 Results

Although it is built in python, **FPP** is designed to be fast and it achieves this by iterating only one time through the atoms in the **PDB** files and at the same time it computes the various parameters.

Thanks to the module **tabulate**, results are outputted in an easy to read and nicely formatted table, such as:

	1elo.pdb	3epo.pdb
-----	-----	-----
Protein Mass	65.964 KDa	112.894 KDa
Protein Volume	81.137 nm ³	138.863 nm ³
Center of Mass	(40.088, 49.256, -23.914)	(54.711, 41.195, 17.318)
End-to-End distance	75.601	31.793
Atoms Not Counted	0	247
Residues Cot Counted	0	18

4 Discussion

Despite being fast **FPP** isn't reliable on big **PDB** files, or files incorporating more than one chain for every protein or multiples structures.

There's no built in logic to distinguish between chains/structures and to choose which one you want to compute the parameters for and which one you want to omit.

Also there's little to none error handling, the script simply skips **PDB** lines with less than expected data and warns the user there might be some errors.

Computing **RMSD** is unreliable too, there's no logic to superimpose just one chain or a smaller backbone area of one or both structures; most of the times, this leads to skipping the **RMSD** completely if the compared files differs for just one **Ca** than the other.

5 Conclusion

Although **FPP** is more a proof of concept of what an user could do by writing a script in python, it still provides a fast way to compute various parameters for every **PDB** file parsed.

By being written in python **FPP** could also be easily expanded with the introduction of new functions or modules.