ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ | DEMOCRITUS UNIVERSITY OF THRACE

DEMOCRITUS UNIVERSITY OF THRACE
SCHOOL OF HEALTH SCIENCES
DEPARTMENT OF MOLECULAR BIOLOGY & GENETICS

Atomic density distributions in proteins: structural and functional implications

BSc Thesis

Author: Sotirios Touliopoulos
Department register number: 2375

Supervisor: Dr. Nicholas M. Glykos
Associate Professor of Structural and Computational Biology

Alexandroupolis, Greece
July 2024

Abstract

Atomic density is the number of atoms per Å^3 (atoms / Å^3). It is calculated by dividing the sum of atomic masses of atoms inside a hypothetical sphere, with the volume of the sphere. Each sphere comes with a certain radius in Angstrom values (Å), with the atom's position as the center. Atomic density in a protein structure is a measure of proximity between protein's atoms. A protein's atomic density distribution shows how well packed is a structure and it may include information on potentially identifying proteins with special folding patterns. In this preliminary report we examine atomic density distributions derived from 21.255 protein structures and show that statistically significant differences between those distributions are present. The biounit assembly format was chosen as a representative for these structures. Hydrogenation alongside hydration of the structures was also completed with modeling programs, to simulate the structures in-vitro. Several protein structures deviate significantly and systematically from the average behaviour and —not unexpectedly— include proteins with characteristic structures. Hierarchical clustering of the atomic density distributions indicated that a far distinct cluster occurs. It supports the existence of a protein group with uncommon atomic density distributions. Search for persistent patterns in this cluster's proteins might be an indication for a structural implication of atomic density. Current efforts focus on identifying putative patterns connecting the structure and function of those proteins with their corresponding distributions. Different clustering methods (kmeans, hdbscan) and application of different radius cutoffs (5Å–7Å) removes bias from our approach and enhances validity.

Keywords:
1. Atomic density
2. Protein structures
3. Clustering

Περίληψη

Η ατομική πυκνότητα ορίζεται ως το πλήθος ατόμων ανα $Å^3$. Υπολογίζεται διαιρώντας το πλήθος των ατομικών μαζών ατόμων που βρίσκονται μέσα σε μια υποθετική σφαίρα, με τον όγκο της σφαίρας αυτής. Κάθε σφαίρα ορίζεται με μια συγκεκριμένη ακτίνα σε Å, με το κέντρο του ατόμου στο κέντρο της. Η ατομική πυκνότητα αποτελεί μέτρο εγγύτητας ατόμων. Μια κατανομή ατομικής πυκνότητας φανερώνει το επίπεδο πακεταρίσματος της δομής και μπορεί να περιέχει πληροφορία για ειδικά μοτίβα αναδίπλωσης πρωτεϊνών. Σε αυτήν την πρώιμη αναφορά, εξετάζουμε κατανομές ατομικής πυκνότητας από ένα δείγμα 21255 πρωτεϊνών και δείχνουμε πως υπάρχουν στατιστικά σημαντικές διαφορές μεταξύ των κατανομών. Ως αντιπροσωπευτική δομή για την ανάλυση επιλέχθηκε η βιολογική οντότητα των πρωτεϊνικών δομών. Ακολούθησε, υδρογόνωση και ενυδάτωση των δομών, με χρήση προγραμμάτων, για να γίνει προσομοίωση τους σε in-vitro συνθήκες. Αρκετές πρωτεΐνες, αποκλίνουν σημαντικά και συστηματικά από την μέση κατάσταση και –καθόλου αναπάντεχα– έχουν χαρακτηριστικές δομές. Η ιεραρχική ομαδοποίηση των κατανομών ατομικής πυκνότητας, φανερώνει επίσης την ύπαρξη μιας απομακρυσμένης συστάδας πρωτεϊνών. Αυτό ενισχύει τις ενδείξεις για ομάδες πρωτεϊνών με ασυνήθιστες κατανομές. Η αναζήτηση επαναλαμβανόμενων δομικών μοτίβων στις πρωτεΐνες αυτής της συστάδας, μπορεί να αποκαλύψει κάποιες δομικές επιπτώσεις της ατομικής πυκνότητας. Η εφαρμογή διαφορετικών μεθόδων ομαδοποίησης (k-means, hdbscan) και η εφαρμογή διαφορετικών ακτίνων (5Å–7Å), ενισχύει την αξιοπιστία της ανάλυσης και μειώνει την όποια προκατάληψη στην ερμηνεία των αποτελεσμάτων.

Λέξεις-κλειδιά:
1. Ατομική πυκνότητα
2. Δομές πρωτεϊνών
3. Ομαδοποίηση

Acknowledgements

I would like to thank my mentor Nicholas M. Glykos, for his support and guidance throughout my 2 years at the NMG group. With patience he helped me grow analytical skills in order to solve complex problems and overcome many obstacles. I would also like to thank my parents for their support and love, the friends I made in this journey and my colleagues at the NMG group, who made working at the lab a pleasing experience.

Table of Contents

Abstract....2
Περίληψη....3
Acknowledgements....4
1. Introduction....6
1.1 Proteins: A prologue....6
1.2 Secondary Structure Elements....8
1.3 Packing in proteins....9
1.4 Purpose of the present thesis....9
2. Methods....11
2.1 Linux operating system....11
2.2 Python programing language....11
2.3 R programming language....11
2.4 Protein Data Bank....12
2.5 PISCES culling server....12
2.6 File Transfer Protocol....12
2.7 Protein Sample Handling....13
2.8 Structures hydrogenation with OpenBabel....13
2.9 Structures hydration with solvate....14
2.10 Atomic density calculation algorithm....14
2.11 Reduce computational time with Parallel....15
2.12 Scaling distributions....16
2.13 Scatter plots....16
2.14 Distance matrix....16
2.15 Plot program for data visualization....17
2.16 Clustering algorithms....17
2.17 Pymol for protein structure visualization....18
3. Results....19
3.1 Raw data visualization and clustering....19
3.2 Structures Examination....23
4. Discussion....32
5. References....33
Appendix....35

1. Introduction

1.1 Proteins: A prologue

Proteins are complex macromolecules that participate in many biological processes such as enzyme catalysis, transport and immune responses [1]. Their wide functionality contributes to them being a major research topic.

Proteins are made up of folded chains of amino acids (polypeptide chains), whose sequence is determined by the sequence of the gene that encodes them (DNA sequence).

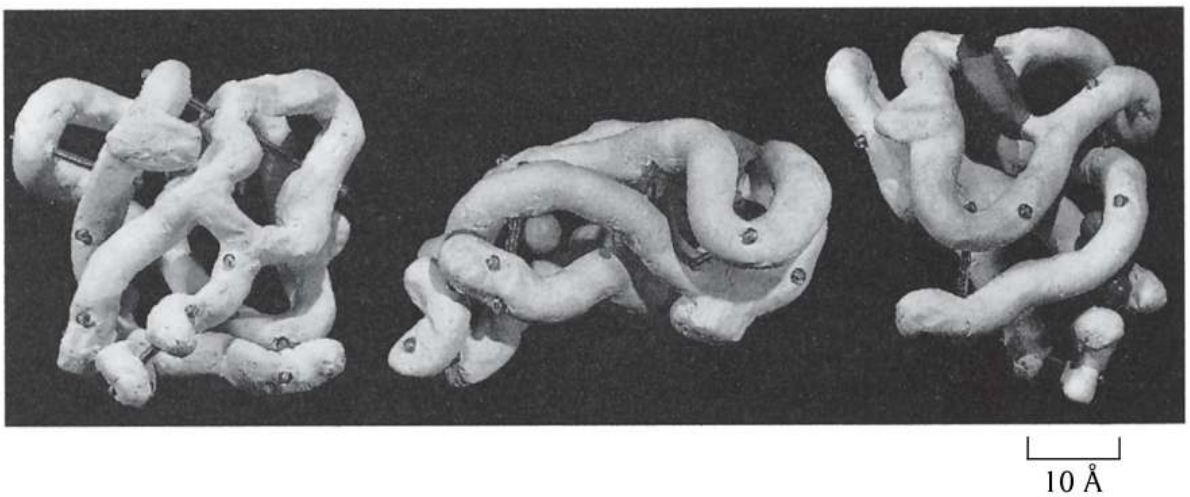

Figure 1.1: Kendrew's model of the low-resolution structure of myoglobin shown in three different views. Reproduced without permission from Carl Ivar Branden & Tooze, J. *Introduction to Protein Structure*.

The linear sequence of the amino acids is what is called the first structural level of proteins and refers to the primary protein structure. The sequence of amino acids encompasses all required information to lead the protein to a folded state [2]. Each amino acid comes with a general structure of an amino group, a carboxyl group and a side-chain R connected to the central alpha-carbon (Ca), as shown in Figure 1.2a. Although more than 100 amino acids occur in nature, only 20 of them are commonly found in most proteins.

Amino acids can be linked via a type of covalent bond, called peptide bond. This bond is formed between the carbonyl carbon of the first amino acid and the nitrogen atom of the second amino acid (Figure 1.2b). These bonds are mainly found in trans-conformation, which prevents neighboring side chains from coming to a non-favorable thermodynamically contact.

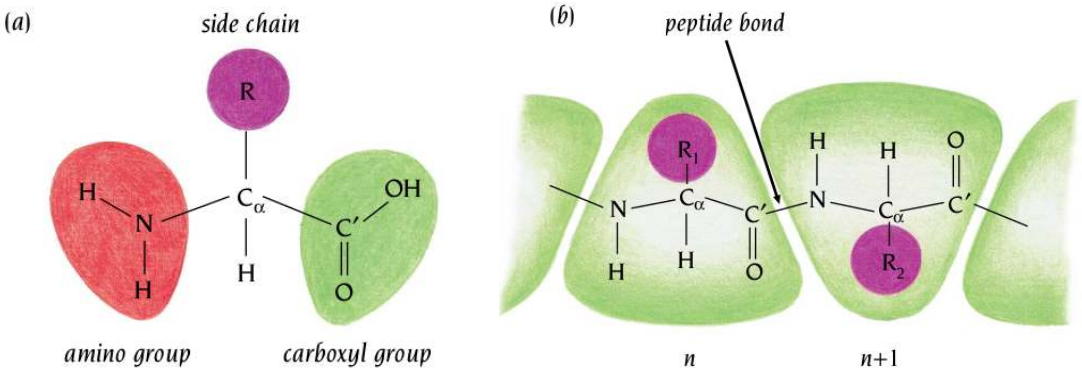

Figure 1.2: (a) amino acid composition, (b) peptide bond between 2 amino acids. Reproduced without permission from Carl Ivar Branden & Tooze, J. *Introduction to Protein Structure.*

Due to the side chains of amino acids having different chemical properties, amino acids can be classified into four different groups, based on the polarity of the side chain. These groups include the non-polar, the polar, the negative charged and the positive charged amino acids.

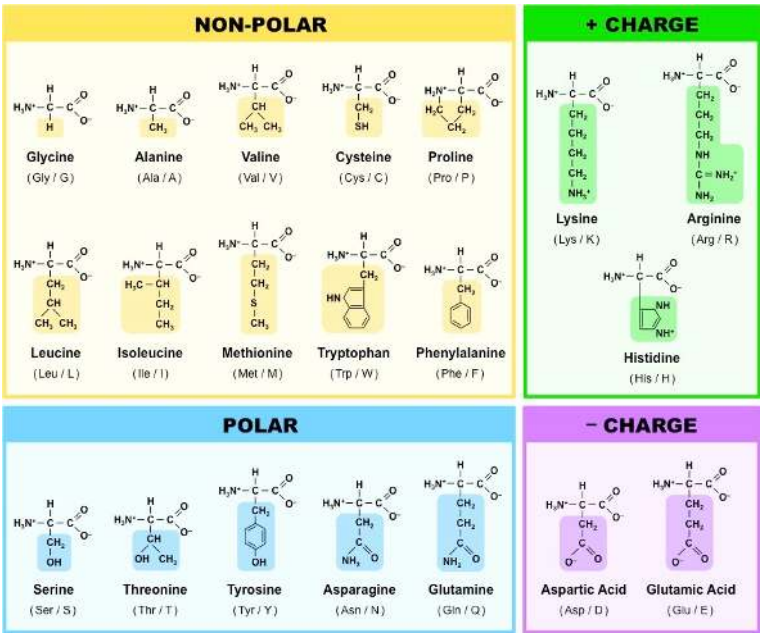

Figure 1.3: Different amino acids groups. Reproduced without permission from https://old-ib.bioninja.com.au/standard-level/topic-2-molecular-biology/24-proteins/amino-acids.html

In addition to the first structural level of proteins, the next levels are the second, which refers to the secondary structure and the third level, which refers to the tertiary structure. Combination of different polypeptide chains may also lead to a fourth structural level. All the 4 hierarchy structural levels can be seen in Figure 1.4.

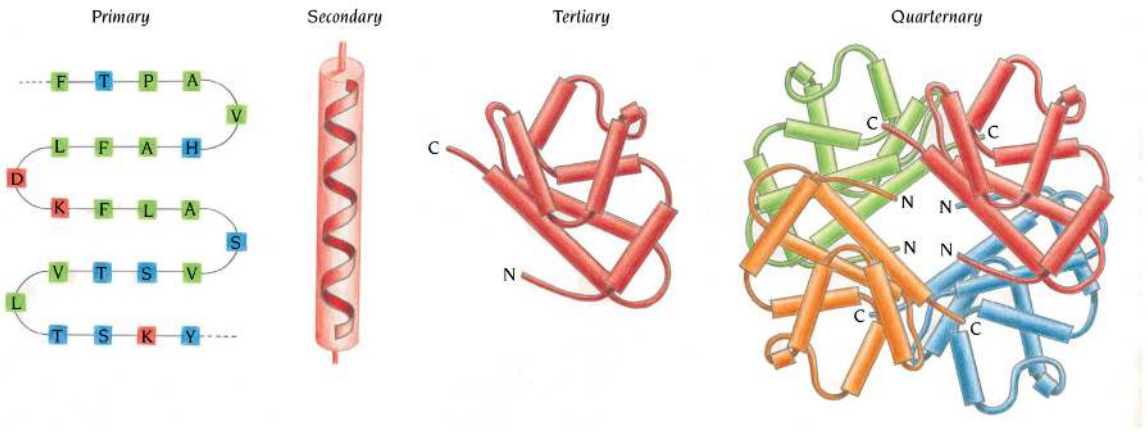

Figure 1.4: The 4 structural levels of proteins. Reproduced without permission from Carl Ivar Branden & Tooze, J. *Introduction to Protein Structure.*

1.2 Secondary Structure Elements

Secondary structure elements are stable conformational patterns that consist of various shapes formed via hydrogen bonding between imine (NH) and carbonyl groups (C'=O). These shapes include alpha helices, beta-pleated sheets and beta-turns. These elements, when present, stabilize protein structures. Alpha helix, is the most abundant pattern, which is characterized by the presence of 3.7 residues per turn, in right-handed direction. Another common element is the b-sheet. It consists of b-strands, which are configurations of 3 to 10 residues [3].

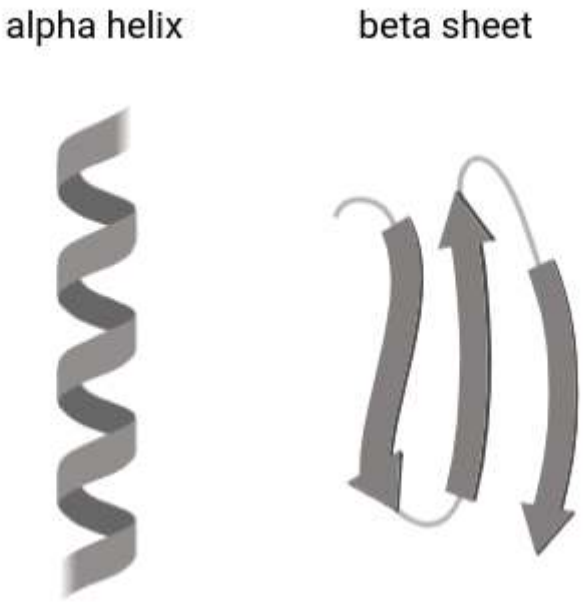

Figure 1.5: Secondary structure elements. Created with BioRender.com.

Elements of regular secondary structure, such as helices and sheets, not only are extensively hydrogen-bonded but also are densely packed. Disulfide bridges not only impose connectivity constraints on conformational motions, but, by forcing backbone segments together, also increase the local atomic density [4].

1.3 Packing in proteins

A general principle in structural biology is that protein structure affects protein function. Thus, some factors affecting protein structure, like atomic packing, radius of gyration and amino acid composition also affect protein function. Atomic packing has been recognized as an important metric for characterizing protein structures since it was observed in 1974 that the average packing density for the interior of proteins is approximately the same as that for crystals of small organic molecules [5].

Atomic packing is a metric of the proximity between protein's atoms. During protein folding a polypeptide chain takes up a three-dimensional structure that is characterized by close packing of atoms. This helps atoms interact with other atoms, developing secondary structure elements that stabilize the structure. Most protein atoms therefore cannot be displaced much without also displacing some of their nonbonded neighbors. However, proteins are not uniformly packed across their structure. Localised defects in packing show up as cavities and when present they can reduce the stability of the structure [6]. The distribution of voids (cavities) is also highly inhomogeneous across proteins [4]. Another fact is that larger proteins tend to be packed more loosely than smaller proteins [7].

Several approaches have been tested to reveal the relationship between packing, structure and function, such as the Voronoi procedure which has been used to assign a unique volume to individual atoms [8]. In this work, we compute atomic density distributions from a sample of 21255 proteins, obtained from the Protein Data Bank (PDB). The centers of atoms across the structure are assigned with the same sphere, which occurs from a standard Angstrom radius. No correction is made for the part of the atom’s volume that is inside or outside of each sphere.

A hypothetical sphere with a certain radius can be imagined surrounding the center of an atom. Inside this sphere, other atoms may exist. If these atoms are counted and then divided by the sphere's volume, the atomic density for this atom is calculated. If this procedure is recursively performed for every atom in the structure an atomic density distribution is created. Another approach to create this distribution is to count the atomic weights of the atoms inside spheres. Each approach will be tested with 3 different radii: 5Å, 6Å, 7Å (Angstrom values), to test the effect of sphere radius in atomic density distributions.

1.4 Purpose of the present thesis

Work on atomic density has already been done in terms of contact density in a small sample of proteins [4]. Our aim is to extend this application to a large sample and simulate the in-vitro

conditions by adding water molecules around the surface of structures. Identification of outliers might reveal uncommon structures with an atomic density distribution deviating significantly from the mean distribution. In addition, identification of clusters containing a small number of structures might reveal structures with common elements or ligands. Our aim is to identify a functional implication in these groups of structures too.

2. Methods

Methods for collection, filtering, modification, processing and analysis of data are presented in this section. This analysis is based on running computational scripts and programs from the linux command line.

2.1 Linux operating system

Linux is a family of open-source Unix-like operating systems based on the Linux kernel, an operating system kernel first released on September 17, 1991, by Linus Torvalds. Ubuntu is a Linux distribution based on Debian and composed mostly of free and open-source software. All computational scripts and analysis were performed in a Ubuntu 22.04 LTS operating system [9].

2.2 Python programing language

Python [10] is a high-level, interpreted and general-purpose programming language. Python was invented in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands. All computational procedures from downloading PDB files to processing them, were completed using Python version 3.10.12.

Figure 2.1: Python logo. Reproduced without permission from https://www.python.org/

2.3 R programming language

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS [11]. The whole statistical analysis for this diploma thesis was performed in R. This includes scaling distributions, visualization and clustering. To help with analysis reproducibility, a Quarto document is created in Rstudio, that works as a notebook. It executes specific code chunks that contain specific analysis steps [12].

Figure 2.2: R logo. Reproduced without permission from https://www.r-project.org/

2.4 Protein Data Bank

The Protein Data Bank (PDB) is the single worldwide archive of structural data of biological macromolecules. It was established at Brookhaven National Laboratories (BNL) in 1971 as an archive for biological macromolecular crystal structures [13].
As of the time this thesis is written (24/06/2024), 221.371 experimentally-determined structures have been deposited in PDB while there are also 1.068.577 structures available derived from computer models predictions.

Figure 2.3: Protein Data Bank logo. Reproduced without permission from https://www.rcsb.org/

2.5 PISCES culling server

PISCES server [14] was used to obtain a set of proteins (PDB entries) with diverse structural and functional characteristics. A list of 22478 PDB files was returned. Thresholds for culling the PDB are shown in table 2.1.

Table 2.1: Thresholds for culling the PDB through PISCES server.

Resolution	0.0 - 2.2
R-factor	0.25
Sequence length	40 - 10000
Sequence percentage identity	<= 50.0
X-ray entries	Included
EM entries	Excluded
NMR entries	Excluded
Chains with chain breaks	Included
Chains with disorder	Included

2.6 File Transfer Protocol

A script for the FTP transfer of the PDB files was created (Appendix: ftp.py). This script enabled the download of the structures the PISCES server returned. The protocol was built

inside a Python script using the corresponding FTP class from the *ftplib* module. This protocol will be phased out on November 1st 2024 for the PDB. The alternative option to download files from PDB, is through *wget*.

The PISCES file was given as input to the ftp script, which saved the first 4 letters from lines containing structure identifiers to a list. These letters (PDB ids) were given as input to the function that retrieves files. Chain identifiers starting from letter 5 were not used, as the aim was to obtain the whole protein and not a specific polypeptide chain.

The Biological Assembly (biounit format) of proteins was selected instead of the common PDB format, as our analysis is focused on both structural and functional implications of atomic density. To acquire such files a ".pdb1.gz" suffix was appended at the end of the 4-letters string. After acquisition and decompression of files locally, some of them were removed, as they did not contain structure information. These files existed in suffixes other than the "pdb1.gz" we provided, and were identified by executing the linux commands:

```
wc * | sort -n -k1
```

2.7 Protein Sample Handling

A cutoff for removal of structures with less than 50 aminoacids was applied. This was done by counting alpha carbons (CA's) Another cutoff was applied for structures with more than 80.000 atoms.

In the "Solvate" section, the 2 final scripts used to handle the proteins' sample are explained. These scripts deal with renumbering residues and mutating "ATOM" to "HETATM" entries. These are necessary format changes in order the solvate program can work properly.

2.8 Structures hydrogenation with OpenBabel

Most hydrogen atoms from crystallographic data are missing due to their very small electron density. OpenBabel program [15] was used to add missing hydrogen atoms to PDB files.

The program's algorithm is based on a geometric approach, it does not perform any simulation and thus completes the hydrogenation fast. Hydrogen atoms may not be placed in their exact structural position, but this creates small or no error at all, as the density algorithm is counting atoms inside spheres and is not affected by the validity of the hydrogen coordinates. Presence or absence of specific atoms inside a sphere is what we examine based on our program's implementation. Newly added hydrogen atoms are appended at the end of PDB files. OpenBabel was executed through bash with the command:

```
for file in ./* ; do obabel ''$file'' -opdb -h > ''$file''.h 2>
''$file''.error ; done
```

Except for the hydrogenated structure with “.h” suffix, another file with “.error” suffix is created for each file that contains warnings and errors. Some of these files are examined and a systematic "Failed to kekulize aromatic bonds” warning appears. This however, does not affect the analysis, as molecules complete protonation successfully. Counting of this warning’s string characters with word count command (wc), helps the identification of other warnings by sorting and accessing larger warnings. This is done with:

```
wc *.error | sort -n -k1 | awk '{print substr( $4, 1, 4 )}'
```

63 files were seen to contain other warnings, which occured from element charges information, but the protonation was completed successfully in all of them.

2.9 Structures hydration with solvate

Solvate program written by Helmut Grubmüller and Volker Groll was used to perform hydration of structures and create a sphere of water molecules around them. Thus, simulation of the in-vitro conditions and removal of low atomic density values can be achieved. The parameter *-bulk* prevented solvate from adding water molecules in buried residues inside cavities. Bulk waters around the protein's surface were appended at the end of PDB files with a “TIP” entry, in columns 17-20. Solvate can not parse files with a residue integer identifier starting at 0, so we renumbered the PDB files. This was done with a script from pdb-tools’ github page. Solvate also removes hydrogens added from OpenBabel to crystallographic waters, so a mutation of their corresponding entries from “HETATM” to “ATOM” was performed too (Appendix: mutate_pdb_formats.py). This mutation was the final step before running the atomic density computational script, so it does not affect the efficiency of programs used to handle PDB files. Solvate is recursively run with bash command:

```
for line in ./* ; do solvate -bulk ${file} ${file}.water 2>
"$file".error ; done
```

2.10 Atomic density calculation algorithm

After sample preparation was completed, files were given as an input to a python script that calculated the atomic density distribution of a structure. This script (Appendix: Z_parser.py), in general, saved the x-y-z coordinates of atoms and calculated the euclidean distance between every possible atoms’ combination. For an Angstrom radius R given as a constant, the script calculated whether the distance between 2 atoms i and j was smaller than R and if so, it counted the presence of atom j inside the sphere of atom i. This was calculated recursively for every atom and the atomic spheres abundance distribution occurred. Then, after division with the spheres’ volume the atomic density distribution was created. This approach was based on counting only the presence or absence of atoms inside spheres centralised in the center of each atom. These atoms were equally treated independently of

their element type and atomic mass. Each atom contributed 1 point to the density calculation, when inside a sphere.

The constant radius R ranged from 5.0 to 7.0 Angstrom with a step of 1.0. Greater values (> 7Å) approach the diameter of 4-alpha helical bundle proteins.

Some limitations were applied to the script, to reduce the level of error and the computational time. The script saved the coordinates of all the atoms from a structure except of the coordinates of solvent's hydrogen atoms added by solvate. An estimation was made that if the oxygen atom from the solvent was inside a sphere, then its corresponding 2 hydrogen atoms would be inside that sphere too. This reduced computational time, as the abundance of these hydrogen atoms was about half of the whole structure's atom abundance, i.e. for every non-hydrogen solvent atom in a structure, another hydrogen solvent atom exists.

In addition, some other limitations were applied. Atomic spheres were created only for atoms of the protein structure itself and thus water molecules and ligands did not contribute with their own atomic sphere creation. The same limitations were applied to atoms with atomic mass smaller than 12 (atomic mass of carbon).

This approach, however, returned a systematic artifact in histogram creation, as the abundance of atoms inside spheres can only be an integer and histogram bins are integers too. Thus, in specific bin widths, histogram frequencies deviated significantly from the expected value and the histogram was discontinuous with extreme outliers. To deal with this artifact, the algorithm was modified to count the atomic mass of atoms inside spheres.

The algorithm for atomic density calculation based on atomic weight counts the atomic mass of each element and not just the presence of it. Unknown atoms were assigned with the average mass across all the atoms with an atomic mass greater than 12. Thus, the density is now counted in Dalton per Angstrom values. After checking the resulting histogram data, the artifact disappeared as expected.

2.11 Reduce computational time with Parallel

The 21255 PDB files that have to be parsed by the atomic density script, lead to a long computational time of about 16 days for each radius. This stimulates the need for procedures running in parallel. GNU parallel [16] is a shell tool for executing jobs in parallel using one or more computers. The 21255 files are equally split into 8 directories using bash and the atomic density script is running simultaneously for the files in the 8 different directories. This reduces computational time for each radius from 16 to 2 days. The bash command for running parallel is:

```
parallel -u ::: dir1.sh dir2.sh dir3.sh ... dir8.sh
```

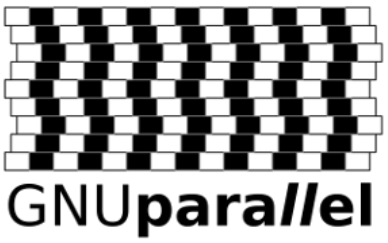

Figure 2.4: GNU Parallel logo. Reproduced without permission from https://www.gnu.org/software/parallel/

2.12 Scaling distributions

The next step, after obtaining the atomic density distribution for each structure, was the creation of the raw data table (Appendix: scaling_distributions.r). Distributions were recursively scanned and the Freedman-Diaconis (FD) rule was applied to each one of them, to find the optimal number of histogram bins. However, the maximum value returned from the FD rule did not work well with small proteins in terms of scaling, so a reduction of the number to 100 was performed. After acquiring the minimum and maximum values across all atomic densities, it was possible to scale distributions in the same density limits and number of bins and create the raw data for analysis. In this data table, every row corresponds to a different structure and every column to a different bin. The first column contains PDB id information. The format of this table is shown in table 2.2.

Table 2.2: Format of raw data table, with N*M dimensions. N equals to 21255 as the length of the protein sample and M equals the maximum of the FD values (different for each radius).

PDB id	bin 1	bin 2	bin M
id 1	value 1,1	value 1,2	value 1,M
id 2	value 2,1	value 2,2	value 2,M
id N	value N,1	value N,2	value N,M

2.13 Scatter plots

Raw data were visualized with scatter plots (Appendix: scatter_plot.r), to gain insight on how data is distributed. In the same plot, mean values for each bin plus standard deviations for bins with a specific step were visuzalised too. Distribution of some outlier proteins was added to the plot too.

2.14 Distance matrix

A distance matrix was created by calculating the euclidean distance between distributions from the raw data. Algorithms for this calculation are already implemented like numpy's

"*linalg.norm"* method (Appendix: distance_matrix.py). Distance matrix was given as an input to the hierarchical clustering algorithm (Appendix: hierarchical_clustering.r) and to the heatmap function (Appendix: heatmap.r). By calculating the sum of each row from this matrix, protein outliers were found too. These proteins were then removed from the distance matrix to better distinguish color scale alterations in heat-maps. To compare cross-radius heatmaps, the maximum of distance values across every distance matrix was applied at the matrix 0,0 position.

2.15 Plot program for data visualization

Plot is a command line program, created for Linux and Mac operating systems by Nicholas M. Glykos [17]. Plot can do simple x-y plots, overlay two x-y plots, draw histograms, create scatter plots and much more. Plot was used to create heat maps from distance matrices and for the comparison of 2 distance matrices with a Pearson correlation as an output. Plot's contribution in matrix comparison is explained in the "Clustering algorithms" section. Heat-maps were created using plot with the "-cc" flag:

```
plot -cc < distance.matrix
```

2.16 Clustering algorithms

3 different clustering methods were used to group proteins with similar atomic density distributions: hierarchical, kmeans and hdbscan.

Hierarchical clustering algorithms take a distance matrix as an input and create a dendrogram that reveals hierarchical relationships between objects. After creating a dendrogram with the "complete" hierarchical method for each radius and extracting the clusters, a cross-radius comparison revealed the level of alteration in the composition of clusters in different radius. A Pearson correlation coefficient, indicated whether different distance matrices were highly correlated or not, going from one radius to another. This was achieved using plot in combination with linux commands:

```
fmt -1 dis.mat.1 > t1 ; fmt -1 dis.mat.2 > t2 ; paste t1 t2 | plot
```

K-Means clustering was applied directly to the raw data. This algorithm supports a built-in dimension reduction based on PCA. However because of the complexity of the data and many outlier distributions present, kmeans did not manage to identify most of the outliers in 2 dimensions. Thus, we used PCA's first 3 Principal components as an alternative to identify outliers. Systematic removal of them and application of PCA from scratch was performed until all far distinct outliers were removed. This was done for plot preparation purposes (outliers were not removed from the analysis).

HDBSCAN clustering was used to validate results from the other clustering methods and to cluster uncommon structures with similar distributions as an alternative method too.

2.17 Pymol for protein structure visualization

Pymol is a molecular graphics program built with the Python programming language. It supports many features such as mutations and atom-atom distance counting. In this analysis, Pymol was used to examine the structure of outlier proteins and gain knowledge on why their distribution is uncommon compared to the rest sample. Figures with structures across this document are created from this software.

3. Results

3.1 Raw data visualization and clustering

Raw data were visualized with scatter plots and heatmaps. The scatter plots (Figure 3.1) overlay the values of specific bins across all distributions. Bins numeric ids are converted to atomic density values. Scatter plots show that certain distributions deviate significantly from the mean distribution (colored in black). This arises from the fact that several data points are significantly distant from the mean, by more than 3σ (three sigmas). This can be seen even more clearly by comparing the distributions of outlier structures (colored in cyan and salmon) with the mean distribution.

Heatmaps show that some distributions shift to higher and some others to lower values of atomic density (Figure 3.2a, 3.2b, 3.2c). This can also be depicted in the corresponding clusters that appear on the left side of the heatmaps. A subset of the raw data was selected for plotting to remove the zero frequencies that appear on the left and right side. This subsetting of the data also affects the clustering algorithm, so a separate hierarchical clustering was performed to the full set of the raw data.

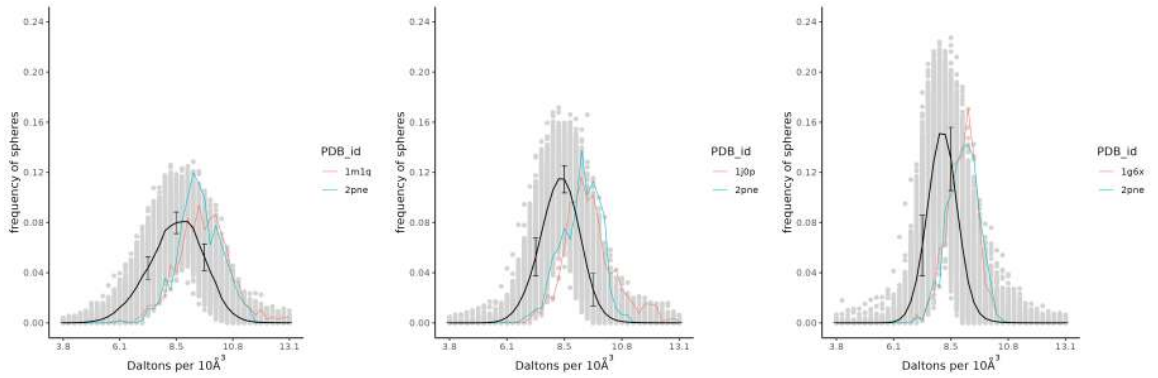

Figure 3.1: Scatter plots of the raw data from 3 different radii. From left to right is the 5Å, the 6Å and the 7Å radius. Some outlier distributions are also plotted.

It can be seen that the increase of the radius from 5Å to 7Å narrows the edges of the distribution (more spheres appear in the middle) and standard deviation values increase too. In addition, a 'chi-by-eye' inspection shows that the scatter plot of 6Å is closer to the shape of the normal (gaussian) distribution.

Heatmaps are depicted in a 3-color code, with low, medium and high values (also appearing in the legends in the top right corner). The 5Å radius appears with some noise, while the 7Å radius has outliers that are difficult to distinguish. The 6Å radius seems ideal for the analysis as it balances between the previous observations. This comes in alignment to the scatter plots visualizations too.

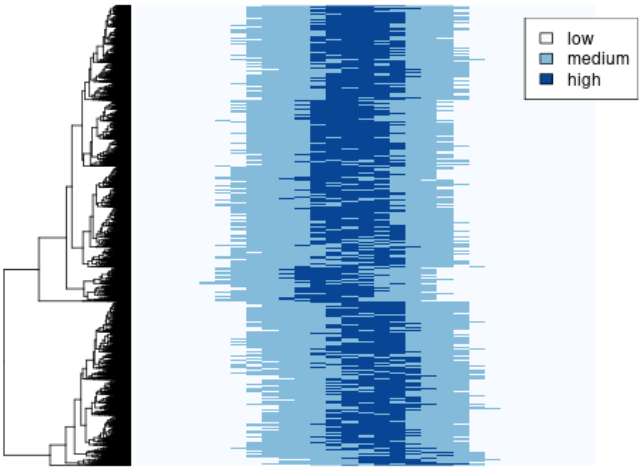

Figure 3.2a: Heatmap of the raw data from 5Å radius.

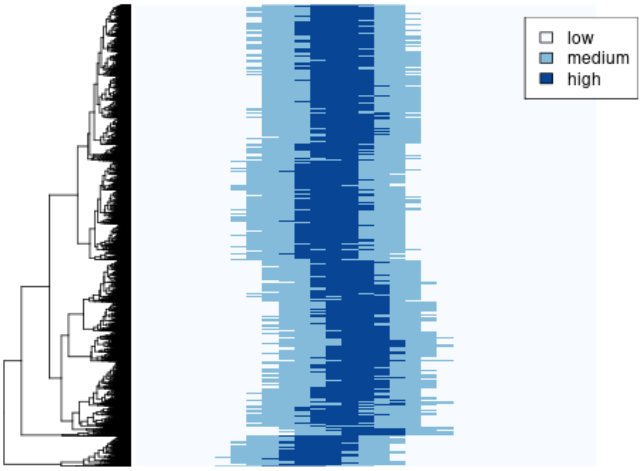

Figure 3.2b: Heatmap of the raw data from 6Å radius.

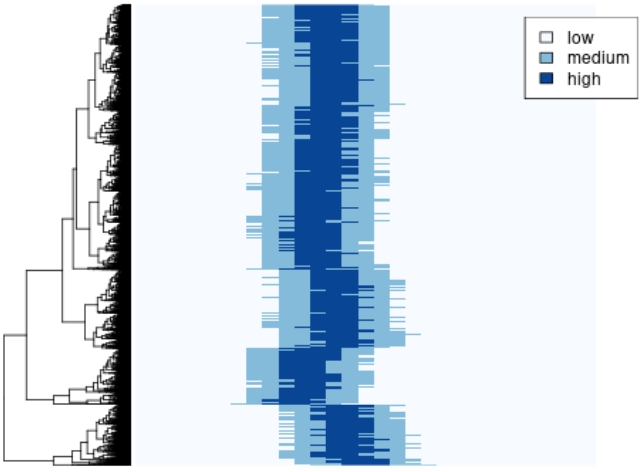

Figure 3.2c: Heatmap of the raw data from 7Å radius.

Distance matrices were created for each different radius and visualized with heatmaps too. These matrices were also given as an input to hierarchical clustering algorithms. Distance matrices (only for the heatmap visualization) were scaled to the same maximum value placed at the point 0.0 of each matrix. The plot program has a color scale code, ranging from blue for small distances to red for big distances. The minimum distance across all different matrices is the same, equaling 0 (occurs from self to self comparisons). The maximum distance however is different and thus we applied the maximum value across all matrices at the point 0.0 of each matrix. This way, the red color of the plot corresponded to this value and a cross-radius color scaling was achieved. This enabled a cross-radius comparison of the results (Figure 3.3), as we were able to observe comparable changes in distances between different radii. The 7Å radius heatmap reveals distributions with similar distances, which is not the case for 5Å and 6Å radii.

Distance matrices were also compared by converting the matrix to a single column and then calculating a pearson correlation for each pairwise radius combination. The 5Å-6Å comparison returned a value of +0.86346943, the 6Å-7Å comparison returned +0.92953539 and the 5Å-7Å returned +0.75881490. As expected, the 5Å-7Å comparison is the least correlated. However, the other comparisons indicate that distance matrices have similar distances between distributions and may produce similar clusters too.

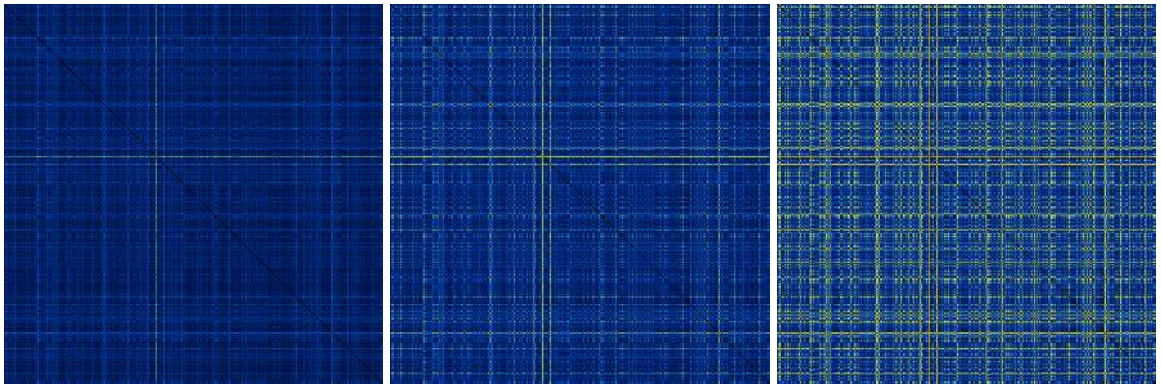

Figure 3.3: Heatmaps of the distance matrices.
From left to right is the 5Å, the 6Å and the 7Å radius.

Existence of outliers did not allow the color code to better visualize the differences between structures. The top 100 most distanced structures were removed from the raw data (only for the plot creation, they remained in the analysis data) and a new distance matrix was created for each radius (Figure 3.4).

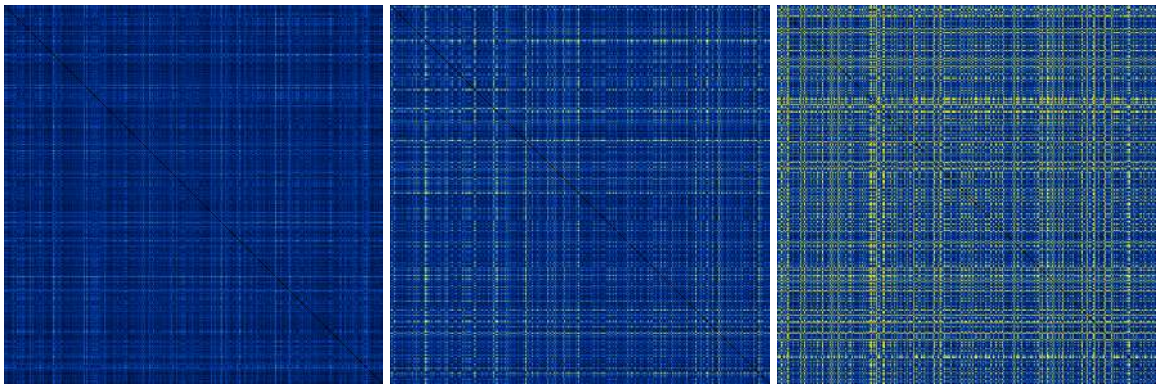

Figure 3.4: Heatmaps of distance matrices after removal of 100 outliers.
From left to right is the 5Å, the 6Å and the 7Å radius.

It can be seen that groups of proteins can be easily identified in the 7Å radius, while the 5Å radius still contains the most outliers, preventing homogenous grouping between structures. Hierarchical clustering (Figure 3.5) supports this, as 7Å is the radius that creates 2 evenly distributed clusters. From what has been observed until now 6Å seems to be the most appropriate radius for our analysis. However identification of outliers will be performed in all of the 3 different radii.

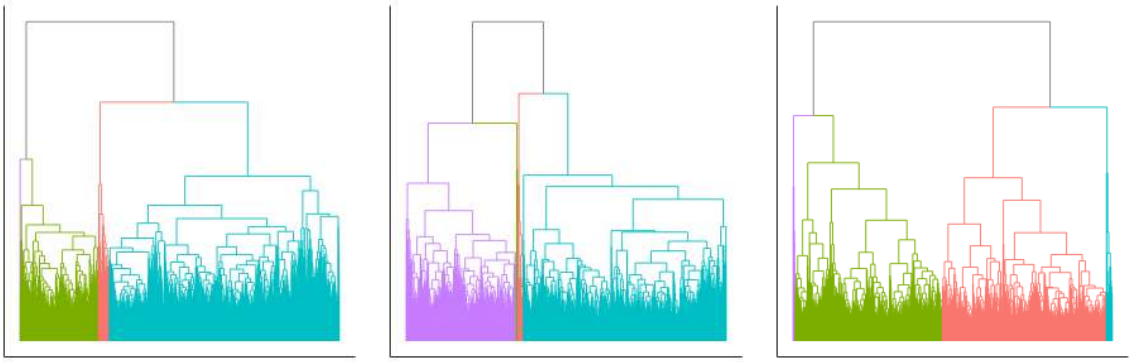

Figure 3.5: Hierarchical clustering of the distance matrices. From left to right is the 5Å, the 6Å and the 7Å radius. Different colors correspond to different clusters for a k=4 parameter (4 clusters in each dendrogram).

Distinct clusters with only a few structures inside them, appear in each radius and contain the structures of interest. Extraction of structures from these clusters, followed by cross-method and cross-radii validation, will return unbiased outlier structures.

Principal Components Analysis (PCA) was performed in the raw data of the 6Å radius. A 3D plot of the first 3 components was returned (Figure 3.6). This plot, showed that the existence of outliers, compresses the majority of structures in a small portion of the space. Thus, 318 outliers (signal) were removed (only for the plot creation) and PCA was performed again, until uniform noise in the 2D plots was achieved.

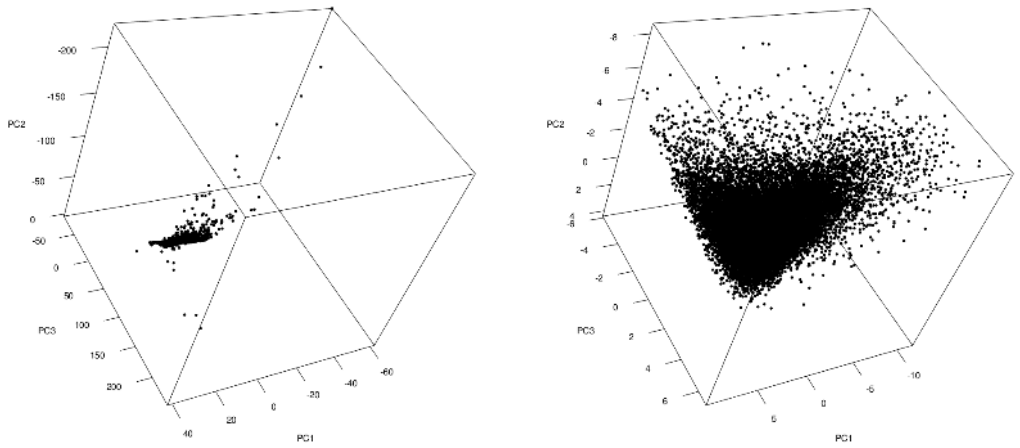

Figure 3.6: Principal Components Analysis (PCA) of the raw data for 6Å radius before (left) and after (right) removal of signal from outliers.

Pairwise plots of the first 3 principal components were also created (Figure 3.7)

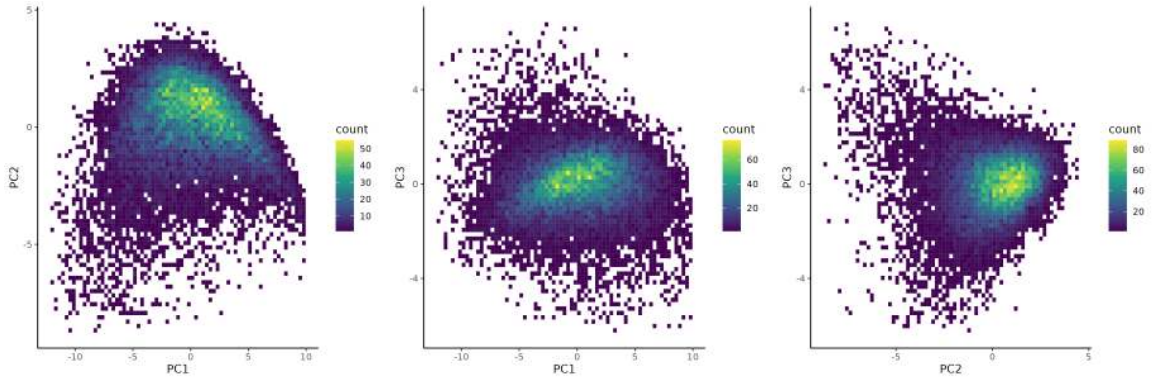

Figure 3.7: Pairwise plots of the first 3 principal components. (left) PC1 with PC2. (middle) PC1 with PC3. (right) PC2 with PC3.

3.2 Structures Examination

The intersection of outliers from the sum of euclidean distances in the distance matrix, hierarchical clustering and Principal Components Analysis returned 67 structures. HDBSCAN results were analyzed separately as the algorithm was executed 4 times with a different number of minimum points (2-5) to produce more clusters of interest.

6 of the 67 structures were cytochromes c in complex with heme. Heme is a ligand with 616 Da average mass. The large mass of heme contributes to higher values of atomic density and shifts distributions to the right compared to the mean distribution, as seen in Figure 3.8.

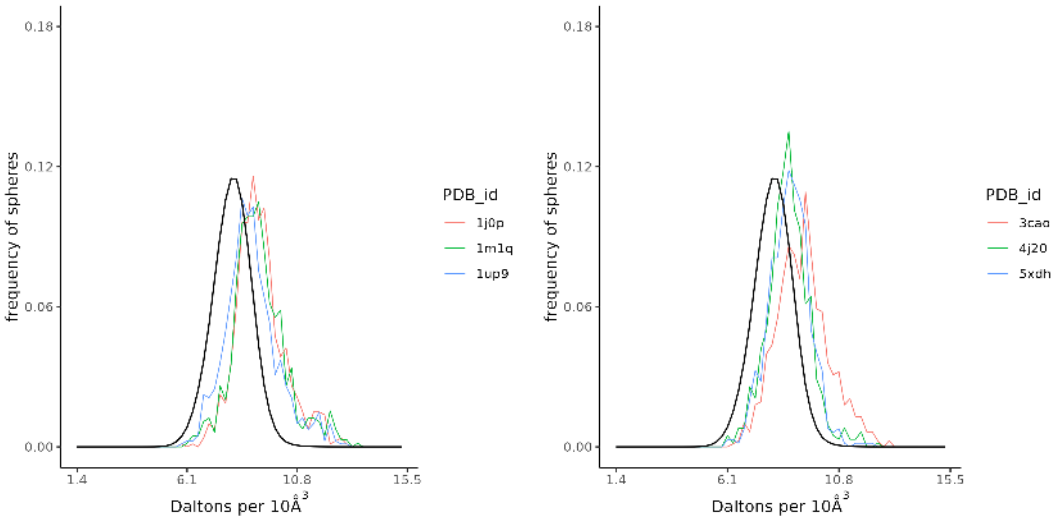

Figure 3.8: Distributions of outlier structures with cytochrome function, compared to the mean distribution colored in black.

Cytochromes form a diverse group of proteins with only a few features in common. They all contain protoheme IX or one of its derivatives and function in electron transport. They are found in nearly all forms of life [18].

Cytochromes c are a class of small, ubiquitously distributed heme proteins that include eukaryotic mitochondrial cytochrome c, the bacterial photosynthetic cytochromes C2, and several others of prokaryotic origin [19] [20]. Cytochrome c is important for oxidative phosphorylation in mitochondria, where it assists with production of life-sustaining ATP by participating in electron transport. The three-dimensional native structure of this protein consists of a compact core around the heme moiety [21]. The heme group in cytochrome c is not only the redox center of the protein, but is also critical for maintaining the native structure: its removal causes disruption of the native fold and loss of most of the secondary structure.

Porphyrin-cytochrome c appears to have a compact structure similar to native cytochrome c, although it is less stable to heat denaturation. It appears that the coordination of the iron atom of the heme results in an increase in the stability of this protein, but is not required for the folding into a compact conformation [22].

The shift of the atomic density distributions of cytochromes to the right comes in agreement to the fact that heme stabilizes the structure. The cytochromes examined are small proteins and thus presence of heme increases the stability of the monomer. The corresponding structures can be seen in Figure 3.9a. 3.9b, 3.9c.

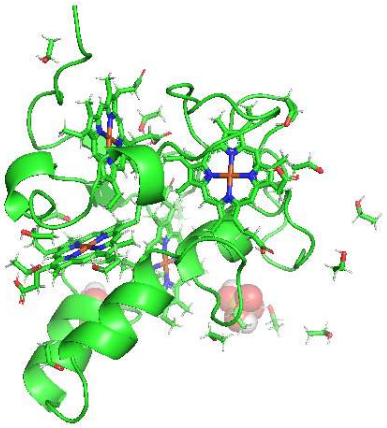

Figure 3.9a: Structure of a cytochrome in complex with heme (PDB id: 1j0p)

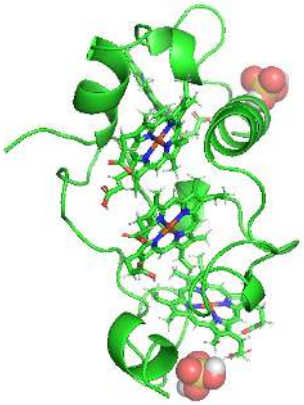

Figure 3.9b: Structure of a cytochrome protein in complex with heme (PDB id: 1m1q)

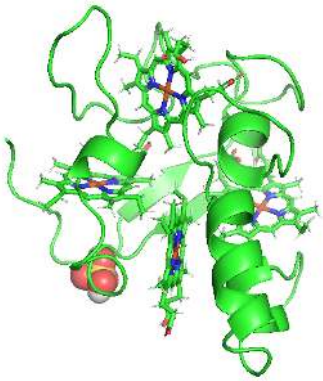

Figure 3.9c: Structure of a cytochrome protein in complex with heme (PDB id: 1up9)

Another group of outlier proteins with similarity in structure and function is the light-harvesting protein group. These structures are complexes with chlorophyll.

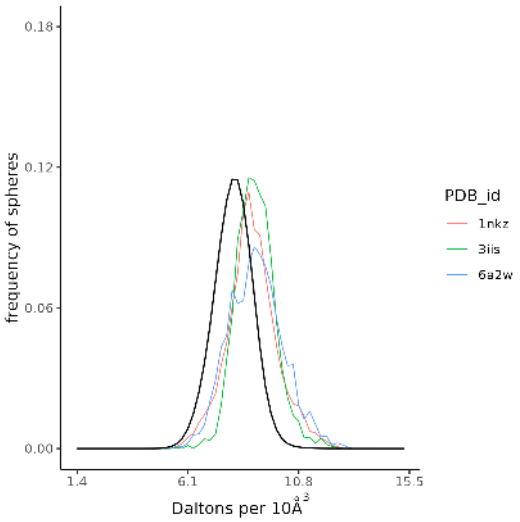

Figure 3.10: Distributions of outlier structures with light-harvesting function, compared to the mean distribution colored in black.

Photosynthetic organisms contain light-harvesting antenna systems to gather light energy required for driving photochemical reactions to initiate a series of charge separation and electron transfer reactions [23]. Many naturally occurring light-harvesting pigment-protein complexes use chlorophyll (Chl) to collect incoming photons [24]. The close packing in the structures, as seen in Figure 3.10 helps the energy transfer between chlorophylls [25] [26].

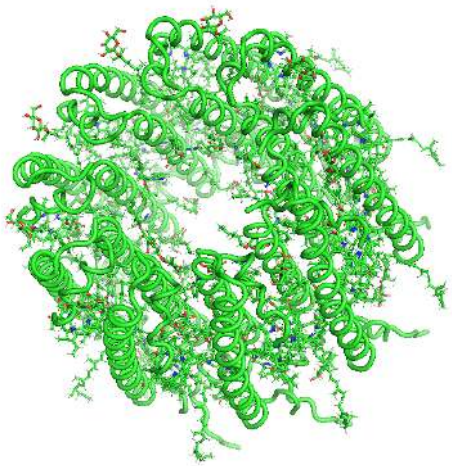

Figure 3.11a: Structure of a light-harvesting protein (PDB id: 1nkz)

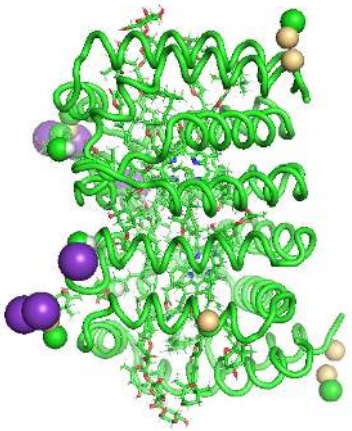

Figure 3.11b: Structure of a light-harvesting protein (PDB id: 3iis)

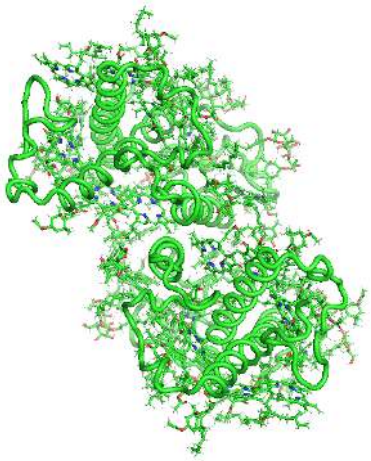

Figure 3.11c: Structure of a light-harvesting protein (PDB id: 6a2w)

Regarding the HDBSCAN algorithm, 2 clusters each containing 3 structures were returned, one from the minimum points set to 2 and one from the minimum points set to 3 parameter.

The first cluster contains structures that function as storage for copper.

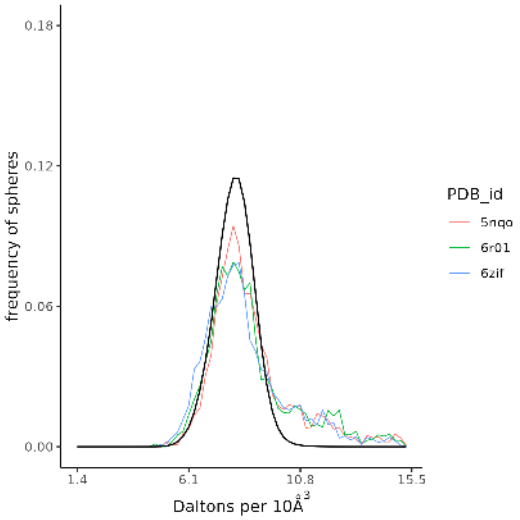

Figure 3.12: Distributions of outlier structures with copper storage function, compared to the mean distribution colored in black.

With these copper storage (Csps) proteins bacteria can maintain appreciable amounts of intracellular copper to prevent toxicity [27] [28] [29]. Compared to previous distributions already examined, this cluster has certain atoms in very high density values. This is due to copper's high atomic mass of 63 Da. This seems to be a unique characteristic of copper-storage proteins, as the overall distribution does not shift to the right too. The corresponding structures can be seen in Figure 3.13a, 3.13b, 3.13c.

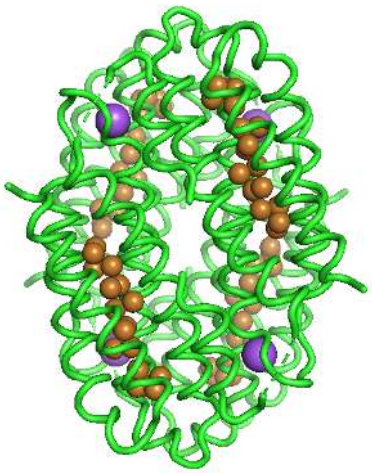

Figure 3.13a: Structure of a copper storage protein (PDB id: 5nqo)

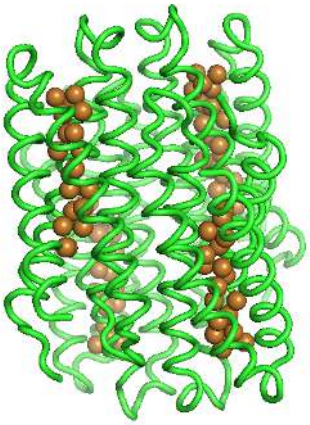

Figure 3.13b: Structure of a copper storage protein (PDB id: 6r01)

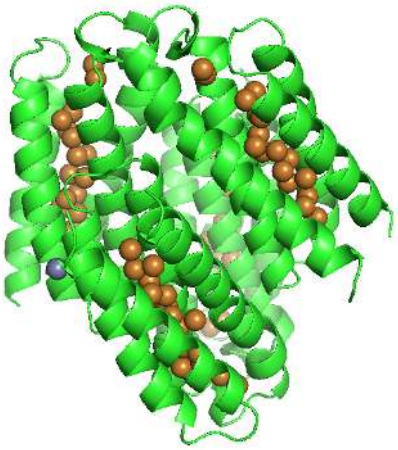

Figure 3.13c: Structure of a copper storage protein (PDB id: 6zif)

The last cluster to be examined, also derived from HDBSCAN but with minimum points set to 3, is a cluster of ferritin proteins.

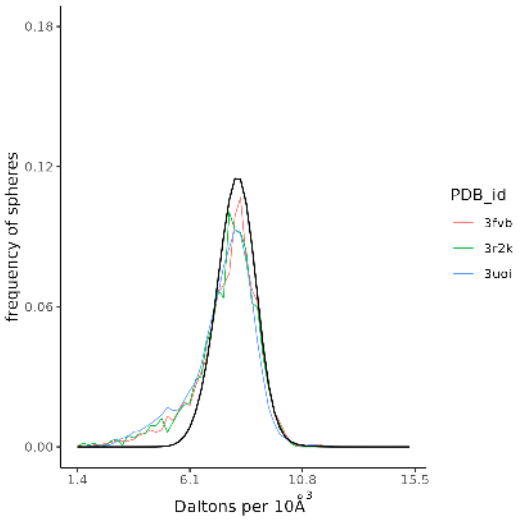

Figure 3.14: Distributions of outlier structures with ferritin-like function, compared to the mean distribution colored in black.

By examining the distributions, certain atoms with very low values of atomic density can be observed. The distributions do not shift to any side like the distributions from the copper-storage proteins too.

Two distinct types of ferritin-like molecules often coexist in bacteria, the heme binding bacterioferritins (Bfr) and the non-heme binding bacterial ferritins (Ftn). Ferritin-like molecules function by oxidizing Fe^{2+} using O_2 and H_2O_2 as electron acceptors and internalize the resultant Fe^{3+} in the form of a mineral.

When environmental iron concentrations are low, Fe^{3+} stored in ferritin-like molecules is mobilized for its incorporation in metabolism, which is why ferritin-like molecules act as dynamic regulators of cytosolic iron concentrations [30].

The corresponding structures can be seen in Figure 3.15a, 3.15b, 1.15c. Their ball-like structures with an empty internal space is responsible for the low atomic density values.

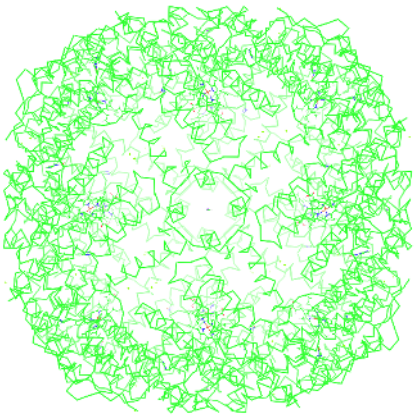

Figure 3.15a: Structure of a ferritin-like protein (PDB id: 3fvb)

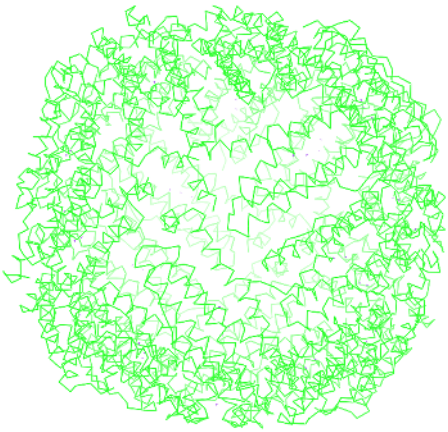

Figure 3.15b: Structure of a ferritin-like protein (PDB id: 3r2k)

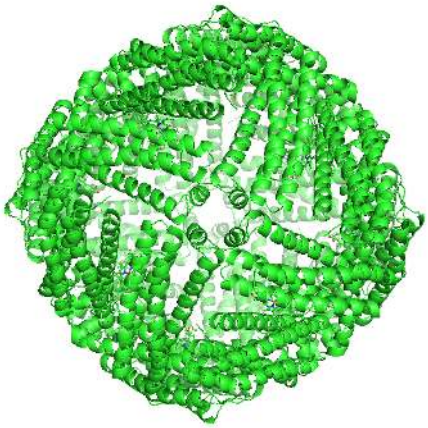

Figure 3.15c: Structure of a ferritin-like protein (PDB id: 3uoi)

4. Discussion

Our first approach for calculating atomic density distributions, failed to generate good quality data, as artifacts appeared in the scatter plots.

Our final approach that dealt with the atomic masses of the corresponding atoms, eliminated these artifacts. However, even after the addition of hydrogens to structures, distributions still had 2 peaks, one in the small and one in the big density values areas. We removed the low density values that were produced from atoms in the surface of the structures with the addition of water molecules around them and the creation of a solvation shell.

Comparison of different radius cutoffs (5Å, 6Å, 7Å), showed that 6Å returns quality clusters without failing to identify outlier structures, compared to the other radii that did not manage to maintain both attributes.

Our final approach for calculating atomic density distributions in proteins seems to reveal some special structural patterns. In the case of copper-storage proteins, a unique type of distribution was revealed and thus these structures were clustered in the same group. The same thing happened for bacterioferritin proteins that had a unique distribution too.

From the final selection of 67 outliers, we managed to group some proteins manually after examination of them with molecular graphics. Groups of interest that eventually came up with similar distributions were created. Some of them were presented in the results section.

Regarding the algorithms used, hierarchical clustering did not provide clusters with a small number of structures (the smallest one had 103 structures in 6Å radius). Thus, we benefit from the algorithm only with information regarding outliers.

Principal Components Analysis was sensitive to the wide range of outlier distributions, but helped us identify extreme outliers both in 2D and in 3D space.

A promising algorithm, that provided small clusters of interest, is the HDBSCAN algorithm. From this algorithm, we managed to identify 2 clusters of proteins with similar structure and function. It seems a promising tool to further examine small clusters in our analysis.

Small clusters and most of the outliers returned, were protein complexes with some ligands. Ligands, because of their excessive molecular mass, seem to play a primary role in obtaining structural information from the atomic density distributions.

Overall, atomic density is a promising metric that might help in the prediction of missing ligands or protein functionality. Examination of structures from non-outliers clusters and use of Gene Ontology (GO) terms, will advance our findings and provide a quantitative correlation between atomic density and protein functionality.

5. References

1. Alberts, B. *et al.* Protein Function. in *Molecular Biology of the Cell. 4th edition* (Garland Science, 2002).

2. Rumbley, J., Hoang, L., Mayne, L. & Englander, S. W. An amino acid code for protein folding. *Proc. Natl. Acad. Sci.* **98**, 105–112 (2001).

3. Carl Ivar, Branden, jj & John, Tooze. *Introduction to Protein Structure*. (Garland Publishing Inc., New York, NY, USA).

4. Halle, B. Flexibility and packing in proteins. *Proc. Natl. Acad. Sci. U. S. A.* **99**, 1274–1279 (2002).

5. Richards, F. M. The interpretation of protein structures: Total volume, group volume distributions and packing density. *J. Mol. Biol.* **82**, 1–14 (1974).

6. Sonavane, S. & Chakrabarti, P. Cavities and Atomic Packing in Protein Structures and Interfaces. *PLOS Comput. Biol.* **4**, e1000188 (2008).

7. Liang, J. & Dill, K. A. Are Proteins Well-Packed? *Biophys. J.* **81**, 751–766 (2001).

8. Pontius, J., Richelle, J. & Wodak, S. J. Deviations from Standard Atomic Volumes as a Quality Measure for Protein Crystal Structures. *J. Mol. Biol.* **264**, 121–136 (1996).

9. Haines, N. Installing Ubuntu. in *Beginning Ubuntu for Windows and Mac Users : Start Your Journey into Free and Open Source Software* (ed. Haines, N.) 1–61 (Apress, Berkeley, CA, 2023). doi:10.1007/978-1-4842-8972-3_1.

10. Welcome to Python.org. *Python.org* https://www.python.org/ (2024).

11. R: The R Project for Statistical Computing. https://www.r-project.org/.

12. Bauer, P. C. & Landesvatter, C. Writing a reproducible paper with RStudio and Quarto. Preprint at https://doi.org/10.31219/osf.io/ur4xn (2023).

13. Berman, H. M. *et al.* The Protein Data Bank. *Nucleic Acids Res.* **28**, 235–242 (2000).

14. Wang, G. & Dunbrack, R. L., Jr. PISCES: a protein sequence culling server. *Bioinformatics* **19**, 1589–1591 (2003).

15. O'Boyle, N. M. *et al.* Open Babel: An open chemical toolbox. *J. Cheminformatics* **3**, 33 (2011).

16. Tange, O. *GNU Parallel 2018*. (Lulu.com, 2018).

17. plot, spherical harmonics, Pepinsky's machine, mcps, hp48, Nicholas M Glykos' group. https://utopia.duth.gr/~glykos/other.html.

18. Scott Mathews, F. The structure, function and evolution of cytochromes. *Prog. Biophys. Mol. Biol.* **45**, 1–56 (1985).

19. Salemme, F. R. Structure and Function of Cytochromes C. *Annu. Rev. Biochem.* **46**,

299–330 (1977).

20. Capaldi, R. A., Malatesta, F. & Darley-Usmar, V. M. Structure of cytochrome *c* oxidase. *Biochim. Biophys. Acta BBA - Rev. Bioenerg.* **726**, 135–148 (1983).

21. Kang, X. & Carey, J. Role of Heme in Structural Organization of Cytochrome c Probed by Semisynthesis. *Biochemistry* **38**, 15944–15951 (1999).

22. Fisher, W. R., Taniuchi, H. & Anfinsen, C. B. On the Role of Heme in the Formation of the Structure of Cytochrome c. *J. Biol. Chem.* **248**, 3188–3195 (1973).

23. Wang, W. *et al.* Structural basis for blue-green light harvesting and energy dissipation in diatoms. *Science* **363**, eaav0365 (2019).

24. Schulte, T. *et al.* Identification of a single peridinin sensing Chl-a excitation in reconstituted PCP by crystallography and spectroscopy. *Proc. Natl. Acad. Sci. U. S. A.* **106**, 20764–20769 (2009).

25. Croce, R. & van Amerongen, H. Natural strategies for photosynthetic light harvesting. *Nat. Chem. Biol.* **10**, 492–501 (2014).

26. Büchel, C. Evolution and function of light harvesting proteins. *J. Plant Physiol.* **172**, 62–75 (2015).

27. Vita, N. *et al.* A four-helix bundle stores copper for methane oxidation. *Nature* **525**, 140–143 (2015).

28. Vita, N. *et al.* Bacterial cytosolic proteins with a high capacity for Cu(I) that protect against copper toxicity. *Sci. Rep.* **6**, 39065 (2016).

29. Baslé, A., Platsaki, S. & Dennison, C. Visualizing Biological Copper Storage: The Importance of Thiolate‐Coordinated Tetranuclear Clusters. *Angew. Chem. Int. Ed Engl.* **56**, 8697–8700 (2017).

30. Yao, H. *et al.* Two distinct ferritin-like molecules in P. aeruginosa: The product of the bfrA gene is a bacterial ferritin (FtnA) not a bacterioferritin (Bfr). *Biochemistry* **50**, 5236–5248 (2011).

Appendix

ftp.py

```
#python ftp module
import ftplib
import sys

#connect to : ww.pdb.org
try:
    ftp = ftplib.FTP("ftp.wwpdb.org")
except:
    print('Unable to connect \n')
finally:
    print('Connection established \n')

#log in
try:
    ftp.login("anonymous", "password")
except:
    print('Unable to log in \n')
finally:
    print('Logged in Successful \n')

#change directory
try:
    ftp.cwd('/pub/pdb/data/biounit/coordinates/all')
except:
    print('Cannot access :  /pub/pdb/data/biounit/coordinates/all \n')
finally:
    print('Current directory :  /pub/pdb/data/biounit/coordinates/all \n')

pdb_code_list = []
with open(sys.argv[1], 'r') as file:
    for line in file:
        list = line.split()
        first_column = list[0]
        if len(first_column) == 5 :
            pdb_code = line[:4]
            pdb_code_list.append(pdb_code.lower())

for i in range( 0 , len(pdb_code_list) , 1 ):
    filename = pdb_code_list[i] + '.pdb1.gz'

    try:
        ftp.retrbinary("RETR " + filename ,open(filename, 'wb').write)
    except:
        print('failed to download : ' , filename )
    finally:
        print('Successful download of : ' , filename )

try:
    ftp.quit()
```

```
except:
    print('Unable to Disconnect from : wwpdb.org')
finally:
    print('Disconnected from : wwpdb.org')
```

mutate_pdb_formats.py

```
# script that converts HETATM records from crystallographic water to ATOM records

import sys

try:
    with open(sys.argv[1], 'r') as file:
        for line in file:

            if line.startswith('ATOM'):
                if line[16] == 'A' or line[16] == ' ' :
                    print(line[:-1])

            elif line.startswith("HETATM"):
                residue = line[17:20]
                if residue == "HOH":
                    atom_number = line[6:13]
                    print("ATOM ", atom_number, line[14:-1])

                else:
                    print(line[:-1])

except:
    print("error in mutating")
```

Z_parser.py

```
import sys
import math

# define a dictionary with elements and their corresponding atomic mass
atomic_mass_dictionary = {
       "H":   {"mass": 1.00797},
       "D":   {"mass": 2.014},
       "HE":  {"mass": 4.00260},
       "LI":  {"mass": 6.941},
       "BE":  {"mass": 9.01218},
       "B":   {"mass": 10.81},
       "C":   {"mass": 12.011},
       "N":   {"mass": 14.0067},
       "O":   {"mass": 15.9994},
       "F":   {"mass": 18.998403},
```

```
"NE":  {"mass": 20.179},
"NA":  {"mass": 22.98977},
"MG":  {"mass": 24.305},
"AL":  {"mass": 26.98154},
"SI":  {"mass": 28.0855},
"P":   {"mass": 30.97376},
"S":   {"mass": 32.06},
"CL":  {"mass": 35.453},
"K":   {"mass": 39.0983},
"AR":  {"mass": 39.948},
"CA":  {"mass": 40.08},
"SC":  {"mass": 44.9559},
"TI":  {"mass": 47.90},
"V":   {"mass": 50.9415},
"CR":  {"mass": 51.996},
"MN":  {"mass": 54.9380},
"FE":  {"mass": 55.847},
"NI":  {"mass": 58.70},
"CO":  {"mass": 58.9332},
"CU":  {"mass": 63.546},
"ZN":  {"mass": 65.38},
"GA":  {"mass": 69.72},
"GE":  {"mass": 72.59},
"AS":  {"mass": 74.9216},
"SE":  {"mass": 78.96},
"BR":  {"mass": 79.904},
"KR":  {"mass": 83.80},
"RB":  {"mass": 85.4678},
"SR":  {"mass": 87.62},
"Y":   {"mass": 88.9059},
"ZR":  {"mass": 91.22},
"NB":  {"mass": 92.9064},
"MO":  {"mass": 95.94},
"TC":  {"mass": 98},
"RU":  {"mass": 101.07},
"RH":  {"mass": 102.9055},
"PD":  {"mass": 106.4},
"AG":  {"mass": 107.868},
"CD":  {"mass": 112.41},
"IN":  {"mass": 114.82},
"SN":  {"mass": 118.69},
"SB":  {"mass": 121.75},
"I":   {"mass": 126.9045},
"TE":  {"mass": 127.60},
"XE":  {"mass": 131.30},
"CS":  {"mass": 132.9054},
"BA":  {"mass": 137.33},
"LA":  {"mass": 138.9055},
"CE":  {"mass": 140.12},
"PR":  {"mass": 140.9077},
"ND":  {"mass": 144.24},
"PM":  {"mass": 145},
"SM":  {"mass": 150.4},
"EU":  {"mass": 151.96},
"GD":  {"mass": 157.25},
"TB":  {"mass": 158.9254},
```

```
"DY":  {"mass": 162.50},
"HO":  {"mass": 164.9304},
"ER":  {"mass": 167.26},
"TM":  {"mass": 168.9342},
"YB":  {"mass": 173.04},
"LU":  {"mass": 174.967},
"HF":  {"mass": 178.49},
"TA":  {"mass": 180.9479},
"W":   {"mass": 183.85},
"RE":  {"mass": 186.207},
"OS":  {"mass": 190.2},
"IR":  {"mass": 192.22},
"PT":  {"mass": 195.09},
"AU":  {"mass": 196.9665},
"HG":  {"mass": 200.59},
"TL":  {"mass": 204.37},
"PB":  {"mass": 207.2},
"BI":  {"mass": 208.9804},
"PO":  {"mass": 209},
"AT":  {"mass": 210},
"RN":  {"mass": 222},
"FR":  {"mass": 223},
"RA":  {"mass": 226.0254},
"AC":  {"mass": 227.0278},
"PA":  {"mass": 231.0359},
"TH":  {"mass": 232.0381},
"NP":  {"mass": 237.0482},
"U":   {"mass": 238.029},
"PU":  {"mass": 242},
"AM":  {"mass": 243},
"BK":  {"mass": 247},
"CM":  {"mass": 247},
"NO":  {"mass": 250},
"CF":  {"mass": 251},
"ES":  {"mass": 252},
"HS":  {"mass": 255},
"MT":  {"mass": 256},
"FM":  {"mass": 257},
"MD":  {"mass": 258},
"LR":  {"mass": 260},
"RF":  {"mass": 261},
"BH":  {"mass": 262},
"DB":  {"mass": 262},
"SG":  {"mass": 263},
"UUN": {"mass": 269},
"UUU": {"mass": 272},
"UUB": {"mass": 277},

"X":   {"mass": 13.53277},
"*":   {"mass": 13.53277},

"TIP":  {"mass": 18},

"HOH O":  {"mass": 15.9994},
"HOH H":  {"mass": 1.00797},
```

```
        "DNA P": {"mass": 30.97376},
        "DNA O": {"mass": 15.9994},
        "DNA C": {"mass": 12.011},
        "DNA N": {"mass": 14.0067},
        "DNA H": {"mass": 1.00797}

}


#create an empty list to store atom type
atom_type = []

#create an empty list to store x coordinates
x = []
#create an empty list to store y coordinates
y = []
#create an empty list to store z coordinates
z = []

#create an empty list to store element types
element_type = []


#identify errors in parsing the PDB
try:
        #opens a pdb file given as 1st command line parameter
        with open(sys.argv[1], 'r') as file:

        #for every line
                for line in file:

                        if( ( line.startswith("ATOM") or
                              line.startswith("HETATM") ) and
                            ( line[13:21] != 'H1  TIP3') and
                            ( line[13:21] != 'H2  TIP3') ):

                                #creates a variable from the column of atom type
                                atom = line[0:3]
                                #appends it to a list
                                atom_type.append(atom)

                                #creates a variable from the column of x coordinate
                                coord_x = line[30:38]

                                x.append(float(coord_x))

                                #creates a variable from the column of y coordinate
                                coord_y = line[38:46]

                                y.append(float(coord_y))

                                #creates a variable from the column of z coordinate
                                coord_z = line[46:54]

                                z.append(float(coord_z))
```

```
#creates a variable from the column of the element type
element_id = line[76:78]

#creates a variable from the column of residue/solvent type
residue = line[17:20]


#if TIP water molecule defined in residue columns
if residue == "TIP":
       element_type.append("TIP")

#if crystallographic oxygen from water molecule

elif residue == "HOH" and element_id == " O":
       element_type.append("HOH O")

#if crystallohraphic/obabel hydrogen from water molecule

elif residue == "HOH" and element_id == " H":
       element_type.append("HOH H")


# for DNA molecules
elif (residue == "DA "):
       element_type.append("DNA " + element_id[1].upper())

elif (residue == "DT "):
       element_type.append("DNA " + element_id[1].upper())

elif (residue == "DG "):
       element_type.append("DNA " + element_id[1].upper())

elif (residue == "DC "):
       element_type.append("DNA " + element_id[1].upper())

elif (residue == "DU "):
       element_type.append("DNA " + element_id[1].upper())

elif (residue == "DI "):
       element_type.append("DNA " + element_id[1].upper())

elif (residue == "I  "):
       element_type.append("DNA " + element_id[1].upper())

elif (residue == "A  "):
       element_type.append("DNA " + element_id[1].upper())

elif (residue == "U  "):
       element_type.append("DNA " + element_id[1].upper())

elif (residue == "C  "):
       element_type.append("DNA " + element_id[1].upper())

elif (residue == "G  "):
       element_type.append("DNA " + element_id[1].upper())
```

```
                        elif (residue == "N  "):
                                element_type.append("DNA " + element_id[1].upper())


                        #if non-water element defined in 1 char in element column
                        elif (element_id[0] == ' '):
                                element_type.append(element_id[1].upper())
                        #if non-water non-DNA element defined in 2 chars
                        elif (element_id[0] != ' '):
                                element_type.append(element_id[0:2].upper())


except:
    print('error in parsing')


#identify errors in density calculations
try:
        #creates an empty list to store all atomic weights
        atomic_weight_dist = []

        #radius in which we calculate weighting density
        #given as 2nd command line parameter
        radius = float( sys.argv[2] )

        #creates a loop with range as the counted atoms
        #which is the same as the counted x coordinates that we use
        for i in range(len(atom_type)):

                if( (atom_type[i] != "HET") and
                (element_type[i][0:3] != "HOH") and
                (element_type[i] != "TIP") and
                (element_type[i][0:3] != "DNA") and
                (atomic_mass_dictionary[element_type[i]]["mass"] > 12) ):

                        #set count variable to 0 for the current atom i
                        count = 0

                         #creates a loop to compare every atom i with every other j
                        for j in range(len(atom_type)):

                                #counts the distance between atom i and atom j
                                distance = (  math.sqrt((( x[i] - x[j] ) *
                                                         ( x[i] - x[j])) +
                                                        (( y[i] - y[j] ) *
                                                         ( y[i] - y[j] )) +
                                                        (( z[i] - z[j] ) *
                                                         ( z[i] - z[j] ))  )  )

                                #call the corresponding element id from its list
                                element_id = element_type[j]

                                #if distance is smaller than given radius
                                if distance < radius:

                                        #return the corresponding atomic mass
```

```
                                        Z_weight = atomic_mass_dictionary[element_id]["mass"]

                                        #sum the atomic number for the current atom
                                        count += Z_weight

                                #add the sum of atomic numbers of current atom to a list
                                atomic_weight_dist.append(count)


        #calculate the volume of the sphere we want to calculate atomic density
        #based on the radius we gave as parameter
        volume = ((4*math.pi*radius*radius*radius)/(3))

        #create an empty list to store all densities
        density_dist=[]

        #for every item in atomic weights list
        for i in atomic_weight_dist:

                #calculate its corresponding density by dividing with sphere volume
                atom_density = ( i / volume )
                #add the calculated density to a list
                density_dist.append(atom_density)

        #print the list
        for density in density_dist:
                print(density)

except:
        print('error in calculating distance')
```

scaling_distributions.r

```
Z_5A = "~/R_pipeline/5A/5A"

Z_6A = "~/R_pipeline/6A/6A"

Z_7A = "~/R_pipeline/7A/7A"

# get a vector with all distribution file names
distributions_files = c(
                        list.files(path = Z_5A,
                                   recursive = TRUE,
                                   pattern = ".dist$",
                                   full.names = TRUE) ,

                        list.files(path = Z_6A,
                                   recursive = TRUE,
                                   pattern = ".dist$",
                                   full.names = TRUE) ,
```

```
                        list.files(path = Z_7A,
                                recursive = TRUE,
                                pattern = ".dist$",
                                full.names = TRUE) )


# create empty vector to store all optimal bins
all_bins = c()
min_values = c()
max_values = c()

# iterate through every distribution file
for (file in distributions_files)
{
  # read distribution file
  distribution = scan( file=file , quiet=T)

  # find minimum distribution value
  dist_min = min(distribution)
  # append to vector
  min_values = append(min_values, dist_min)

  # find maximum distribution value
  dist_max = max(distribution)
  # append to vector
  max_values = append(max_values, dist_max)

  # hist function to gain histogram
  hist = hist(distribution, breaks = "FD" , plot=FALSE )
  # frequencies/bin ==> counts, so length equals the number of bins
  bins = length(hist$counts)
  # append to vector
  all_bins = append(all_bins, bins)
}

# find maximum value of optimal bins
max(all_bins)
hist(all_bins)
bins = 100
min_density = min(min_values)
max_density = max(max_values)

# calculate bin width
bin_width = ( (max_density - min_density ) / bins )
# calculate custom breaks
breaks = seq( min_density , max_density , by=bin_width)



# set working directory
setwd("~/R_pipeline/7A/7A")

distributions_files = list.files(path = ".",
                                 recursive = TRUE,
                                 pattern = ".dist$",
                                 full.names = TRUE)
```

```
# dataframe to store bin frequencies from all distributions
raw_data = data.frame()

# iterate every file
for (file in distributions_files)
{
  # read distribution file
  distribution = scan( file=file , quiet=TRUE)
  # number of atoms
  atoms = length(distribution)

  # compute bin frequencies
  bin_frequencies = as.data.frame( table( cut( distribution,
                           breaks=breaks,
                           right=TRUE,
                           include.lowest = TRUE,
                           dig.lab = min(nchar(breaks) )
                           ) ) )

  # scale bin frequencies
  scaled_bin_frequencies = bin_frequencies[,2]/atoms
  # bind to dataframe
  raw_data = rbind(raw_data,scaled_bin_frequencies)

}

# Check scaling outcome by calculating area
# under every histogram (must be equal to 1)
histograms_areas = rowSums( raw_data )


# creation of a vector with pdb ids
PDBs = c()
for (i in distributions_files){
  PDBs = append(PDBs, substr(i,3,6) )
}
# we now pass this vector as first column in the dataset
raw_data = cbind( PDBs , raw_data )


# We also create column integer identifiers
# to name our columns appropriately
columns = c("PDB_id")
for (i in 1:bins){
  columns = append( columns , i )
}
colnames(raw_data) = columns


setwd("~/R_pipeline/7A")
write.table(raw_data , file="raw_data_7A")
```

scatter_plot.r

```
setwd("~/R_pipeline/5A")
raw_data = read.table("raw_data_5A")
dim(raw_data)

# subset bins for scatter plot
raw_data_subset = cbind( "PDB_id" = raw_data[,1] , raw_data[,11:51] )
dim(raw_data_subset)

# first we have to melt our dataframe
library(reshape2)
melt_raw_data = reshape2::melt( raw_data_subset , id.var = "PDB_id")

# compute mean and standard deviation of each bin and combine them to dataframe
library(dplyr)
melt_raw_data = melt_raw_data %>%
  group_by(variable) %>%
  mutate(mean = mean(value), sd = sd(value)) %>%
  as.data.frame()


# add this loop to eliminate the values of some standard deviation
for(i in 1:(ncol(raw_data)-1) )
{
  if( ( i %% 5) != 0 )
  {
      melt_raw_data$sd[melt_raw_data$variable == paste("X",i,sep='')] = NA
      melt_raw_data$mean[melt_raw_data$variable == paste("X",i,sep='')] = NA
  }
}

melt_raw_data$sd[melt_raw_data$variable == "X15"] = NA
melt_raw_data$mean[melt_raw_data$variable == "X15"] = NA

melt_raw_data$sd[melt_raw_data$variable == "X20"] = NA
melt_raw_data$mean[melt_raw_data$variable == "X20"] = NA

melt_raw_data$sd[melt_raw_data$variable == "X40"] = NA
melt_raw_data$mean[melt_raw_data$variable == "X40"] = NA

melt_raw_data$sd[melt_raw_data$variable == "X45"] = NA
melt_raw_data$mean[melt_raw_data$variable == "X45"] = NA

# for 7A only
#melt_raw_data$sd[melt_raw_data$variable == "X35"] = NA
#melt_raw_data$mean[melt_raw_data$variable == "X35"] = NA


# scatter_bins will be replaced by scatter_breaks in the scatter plot
scatter_bins = c(1,11,21,31,41)
scatter_breaks = 10*( round( c(breaks[11], breaks[21], breaks[31],
                               breaks[41], breaks[51] ) ,2 ) )


# we may want to plot some protein-outliers, separately with line plots
```

```
outliers = c("1m1q","2pne")

# identify rows in which there is outliers data
outliers_int_id = which(melt_raw_data$PDB_id %in% outliers)
# store outliers data in another dataframe
outliers_data = melt_raw_data[outliers_int_id,]

# plot
library(ggplot2)

g = ggplot(melt_raw_data, aes(x=as.numeric(variable) ) )+
    geom_point( aes( y=value ), color="lightgrey", size=1.5 )+
    geom_line( aes(y=value, color=PDB_id),
                   data=outliers_data , size=0.3)+

    geom_errorbar( aes(ymin=mean-sd , ymax=mean+sd),
                color="black" , size=.2)+
    stat_summary(fun="mean" , geom="line" , aes(y=value) , size=0.5)+

    theme(axis.text.x=element_text(size=12),axis.title.x=element_text(size=8))+
    theme(axis.text.y=element_text(size=11),axis.title.y=element_text(size=12))+

    theme_classic()+

    scale_x_continuous( breaks=scatter_bins,
                        labels=scatter_breaks )+

    scale_y_continuous( breaks=c(0.00, 0.04,
                                 0.08, 0.12, 0.16, 0.20, 0.24),
                        limits=c(0,0.24))+

    labs( y="frequency of spheres" ,
          x=expression(paste("Daltons per 10",
          ring(A)^3 ) ) )
```

distance_matrix.py

```
import numpy as np
from contextlib import redirect_stdout

numpy_file = np.loadtxt("./7A/raw_data_7A",
                        usecols=sorted(set(range(2,102))),
                        skiprows=1 )
with open('distance.matrix', 'w') as output:
    with redirect_stdout(output):
        for array_a in numpy_file:
            for array_b in numpy_file:
                dist = np.linalg.norm( array_a - array_b )
                print('%.5f' % dist , end='\t')

            print('')
```

heatmap.r

```
library(RColorBrewer)

setwd("~/R_pipeline/7A")
raw_data = read.table("raw_data_7A", header = T)

PDBs = c(raw_data[,1])

heatmap(as.matrix(raw_data[,18:46]),
        Colv = NA,
        labRow = NA,
        labCol = NA,
        keep.dendro=TRUE,
        col = colorRampPalette(brewer.pal(8,"Blues"))(3))

legend(x="topright", legend = c("low", "medium", "high"),
     fill=colorRampPalette(brewer.pal(8, "Blues"))(3))
```

hierarchical_clustering.r

```
library(ggplot2)
library(ggdendro)
library(dendextend)
library(dplyr)

# get the PDB names
setwd("~/R_pipeline/7A/7A")
distributions_files = list.files(path = ".",
                                 recursive = TRUE,
                                 pattern = ".dist$",
                                 full.names = TRUE)

PDBs = c()
for (i in distributions_files){
  PDBs = append(PDBs, substr(i,3,6) )
}


setwd("~/R_pipeline/5A")

# Load the distance matrix
distance_matrix = matrix(scan("./distance.matrix.5A",
                  n = 21255*21255), 21255, 21255, byrow = TRUE)

# Convert the data into a distance matrix format and perform clustering
hc = hclust( as.dist(distance_matrix) , method = "complete")
```

```
# extract clusters
clusters = cutree( hc, k = 4)
clusters_df = as.data.frame(clusters)
rownames( clusters_df ) = PDBs
clusters_df %>% count(clusters)
write.table( clusters_df , file="./hierarchical_clusters_df")


dend <- as.dendrogram(hc) %>%
  set("branches_k_color", k = 4)

ggd1 <- as.ggdend(dend)


ggplot(ggd1$segments) +
  geom_segment(aes(x = x, y = y, xend = xend, yend = yend, col=col))+
  labs( y="" , x="" )+

  theme_classic()
```