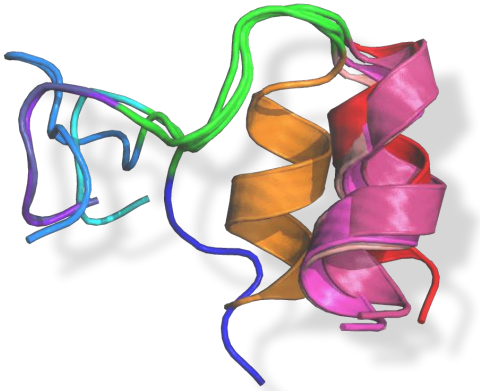
DEMOCRITUS UNIVERSITY OF THRACE
SCHOOL OF HEALTH SCIENCES
DEPT. OF MOLECULAR BIOLOGY AND GENETICS

Final Year Thesis

"Two residue periodicities in protein structures: Results from a systematic search in 4-dimensional Ramachandran space"



Author:
Ioannis Riziotis

Advisor:
Dr. Nicholas M. Glykos
Assistant Professor of Structural and Computational Biology

Alexandroupolis
March 2017

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ ΥΓΕΙΑΣ
ΤΜΗΜΑ ΜΟΡΙΑΚΗΣ ΒΙΟΛΟΓΙΑΣ ΚΑΙ ΓΕΝΕΤΙΚΗΣ

Διπλωματική Εργασία

"**Περιοδικότητες δύο αμινοξικών καταλοίπων σε πρωτεϊνικές δομές**: Αποτελέσματα από μία συστηματική έρευνα στον τετραδιάστατο χώρο Ramachandran"

Ιωάννης Ριζιώτης
(Α.Ε.Μ.:1224)

Επιβλέπων Καθηγητής:
Δρ. Νικόλαος Γλυκός

Αλεξανδρούπολη
Μάρτιος 2017

Acknowledgements

While working in a science laboratory, no matter the field, one should be trained not only in theory and technical matters, but also one should learn how to think and simplify problems that occur in every level of the work. I would like to thank my supervisor Dr. Nicholas Glykos for without his guidelines I would not have managed to meet up with many challenges that I had to confront, and for the ways of teaching me how to think. A big thanks also to my laboratory partners, the "*NMG* group" and my friends, for all the help and encouragement that provided me, and for the long chats we had while procrastinating in the lab. I would finally like to thank my family for all the moral support through the academic years.

"Science, my lad, is made up of mistakes, but they are mistakes which it is useful to make, because they lead little by little to the truth."

-Jules Verne, A Journey to the Center of the Earth

Table of contents

Abstract

The various secondary structure elements in proteins, are formed by amino acid residues that share similar backbone dihedral angle values. Each residue has a limited range of φ,ψ angles, due to steric hindrance of the side chain. We can easily depict the value range of φ,ψ angles of all residues in 2-dimensional space, as a Ramachandran distribution, and distinguish three major, highly-populated regions, that correspond to each of the known secondary structure elements. The typical assumption on protein structure, is that most of the secondary structure elements are characterised by a specific hydrogen bond pattern, and repeating φ,ψ angles.

Our research focuses on tracing and describing motifs in protein structure, which are formed of consecutive residues that do not have repeating φ,ψ angle values, but *two* distinct φ,ψ values alternating between any two residues. Viewing such motifs in a Ramachandran plot, we can see the residues occupying two different regions alternately. An example of a hypothetical model is five continuous transitions between the β-sheet and α_L-helix regions.

In order to evaluate this hypothesis, we performed a series of *in silico* studies in a large dataset of protein molecules. We developed a probabilistic algorithm to cull and score structures that follow motifs like the one described above, using as data, X-ray solved protein structures from the Protein Data Bank. Furthermore, a *UPGMA* clustering algorithm was used to group the potential structures that follow the pattern, and characterise them. Our current results show that such motifs occur in proteins a significant extent.

Περίληψη

Τα διάφορα στοιχεία δευτεροταγούς δομής στις πρωτεΐνες, σχηματίζονται από αμινοξικά κατάλοιπα με παρόμοιες τιμές δίεδρων γωνιών (*φ,ψ*) στην κεντρική τους αλυσίδα. Κάθε κατάλοιπο μπορεί να λάβει τιμές γωνιών *φ,ψ* περιορισμένου εύρους, λόγω στερεοχημικής παρεμπόδισης από την πλευρική αλυσίδα. Μπορούμε έυκολα να απεικονίσουμε το εύρος τιμών των γωνιών *φ,ψ* όλων των αμινοξέων, σε ένα δισδιάστατο επίπεδο, γνωστό και ως διάγραμμα Ramachandran. Διακρίνουμε τρεις περιοχές όπου εμφανίζονται αμινοξέα με μεγάλη συχνότητα. Καθεμία από τις περιοχές αυτές αντιστοιχεί σε ένα μοτίβο δευτεροταγούς δομής και αντικατοπτρίζει το επιτρεπτό εύρος τιμών των γωνιών *φ,ψ*. Η γενική θεώρηση επάνω στις δομές πρωτεϊνών, περιγράφει ότι τα στοιχεία δευτεροταγούς δομής χαρακτηρίζονται από ένα συγκεκριμένο μοτίβο δεσμών υδρογόνου, καθώς επίσης και από επαναλαμβανόμενες τιμές *φ,ψ*.

Η έρευνα μας επικεντρώνεται στον εντοπισμό και την περιγραφή δομικών μοτίβων, τα οποία σχηματίζονται από γειτονικά αμινοξικά κατάλοιπα τα οποία δεν έχουν επαναλαμβανόμενες τιμές *φ,ψ,* αλλά δύο διαφορετικά εύρη τιμών, εναλλασσόμενα, μεταξύ δύο οποιωνδήποτε καταλοίπων. Απεικονίζοντας τέτοια μοτίβα σε ένα διάγραμμα Ramachandran, βλέπουμε τα κατάλοιπα να καταλαμβάνουν εναλλάξ, δύο διαφορετικές περιοχές. Ένα παράδειγμα του υποθετικού μας μοντέλου, είναι ένα μοτίβο πέντε καταλοίπων, τα οποία μεταπίπτουν μεταξύ των περιοχών των *β*-πτυχωτών φύλλων και των αριστερόστροφων *α*-ελίκων.

Για την επαλήθευση της υπόθεσης μας, πραγματοποιήσαμε μία σειρά *in silico* μελετών σε ένα μεγάλο δείγμα πρωτεϊνικών μορίων. Αναπτύξαμε έναν πιθανοτικό αλγόριθμο ο οποίος επιλέγει δομές που πιθανόν ακολουθούν ένα μοτίβο όπως το προαναφερθέν, και τις βαθμολογεί ανάλογα με την πιστοτητά τους σε αυτό. Ο αλγόριθμος χρησιμοποιεί ως δεδομένα αληθινές δομές, λυμένες με κρυσταλλογραφία ακτίνων Χ, απο την βάση δεδομένων *PDB* (*Protein Data Bank*). Επιπλέον, για την ομαδοποίηση και τον ευκολότερο χαρακτηρισμό των δομών που βρέθηκαν, χρησιμοποιήθηκε ένας *UPGMA* αλγόριθμος ομαδοποίησης. Τα τρέχοντα αποτελέσματα των υπολογισμών, δείχνουν ότι μοτίβα τέτοιου είδους απαντώνται σε πρωτεΐνες, σε σημαντικό βαθμό.

Section 1
Introduction

1.1 Proteins: A prologue

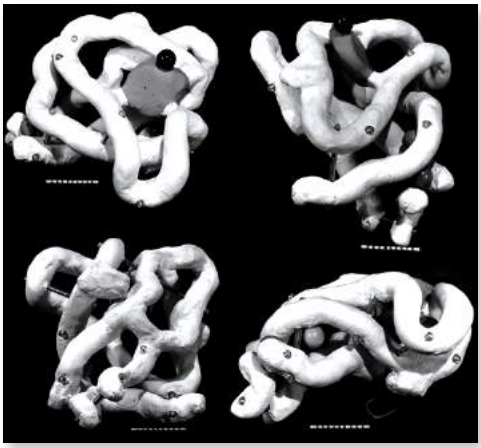


Figure 1.1: The 3D model of myoglobin as presented by J. Kendrew (adapted without permission from *J.C. Kendrew et al., Nature, 1958*)

The nature of proteins as the building blocks of life has been a major concern to the scientific society, and aspects of them regarding structure and function are still being unraveled. Several breakthrough methods, such as X-ray crystallography and Nuclear Magnetic Resonance, have been developed in order to approach and reveal the structure of these molecules and consequently their functional roles. John Kendrew and his myoglobin model in 1958[1] (**Figure 1.1**) kickstarted the era of structure solving, and of that time, nobody could predict the vast number of known structures that would have been available sixty years later. One could wonder why is protein structure of so much significance, and the answer is that structure and function are two interdependent characteristics. It is a matter of necessity to study and identify the principal components of which proteins are formed, in order to fully understand and interpret the various mechanisms carried on by them, such as enzymatic catalysis or cell structure formation.

Protein structure can be analysed into four major classes: primary structure or amino acid sequence; secondary structure; tertiary, and quaternary structure. Each level of this hierarchy is strictly dependent on its subordinate level, with the primary structure being the determinant for the final 3-dimensional structure, hence the native, functional conformation. Structure and function are physically linked, and proteins must undergo a complex folding procedure, on which their conformation changes and goes through multiple transition states, until it reaches the native state. Among the significant number of unique solved protein structures, we can identify some common conformational patterns which can help us organise and comprehend the architecture of proteins.[2] These patterns, or secondary structure elements, can fold further and form the tertiary structure, which can be self-contained and functional. Finally, several folded polypeptide chains can be combined and form a quaternary structure, which is a multi-subunit protein (see **Figure 1.2**).

At the time of discovery of the myoglobin structure, Kendrew was disappointed from the complexity and lack of symmetry the molecule seemed to have, however, we now understand that this complexity is what makes proteins functional[3]. Despite the fact that

tertiary and quaternary structure might seem tangled, one could simplify it in a significant extent by observing the elements of lower hierarchy. By studying the secondary structure, patterns and periodical occurrences of specific residues can be found, and these periodicities are what drew our attention.

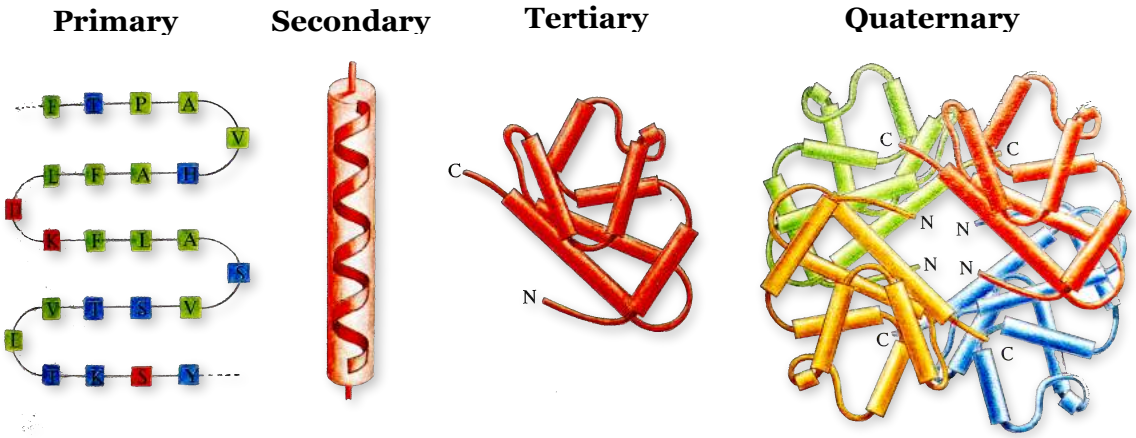


Figure 1.2: The hierarchy of protein structure (adapted without permission from *Branden & Tooze, Introduction to Protein Structure*)

1.2 Secondary structure elements

Peptide folding is carried through the packing of the hydrophobic side chains towards the centre of the protein molecule, creating a hydrophobic core and a hydrophilic outer surface. Something that should be indicated, is that the backbone is highly hydrophilic, due to the occurrence of imine groups (NH) and carbonyl groups (C'=O) in each peptide group (**Figure 1.3**), which act as proton donors and proton receptors respectively. These groups need to be neutralised by forming hydrogen bonds and maintain hydrophobicity in the core.

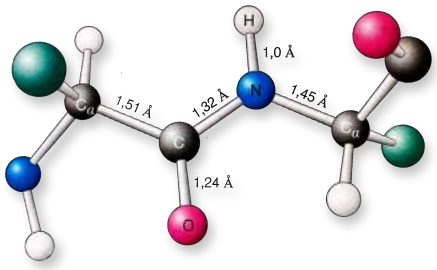


Figure 1.3: A trans peptide group (the four atoms in the centre and C_αs on each side) and the normal distances between the atoms (adapted without permission from *Stryer, Biochemistry*)

The consequence of this, is the formation of stable conformational patterns, known as secondary structure elements. The most abundant pattern is a helical configuration known as the α-helix which is characterised by the presence of 3,7 residues per turn, in right-handed direction[4]. The second most common element is the β-sheet, a pleated surface conformation. It is formed of β-strands, which are configurations of 3 to 10 residues with extended backbone, and has a different hydrogen bond pattern than the α-helix[5]. These elements are kept stable inside the hydrophobic core and provide a scaffold to the molecule[6].

Linus Pauling and Robert Corey first proposed the above-mentioned secondary structure elements in 1951, after collecting information about features, such as bond distances and angles, derived from the crystal structures of several small molecules. In **Figure 1.4** we can see the crystal structure of the atoms that form the *α*-helix and the *β*-sheet as well as the hydrogen bonds that keep the structures stable.

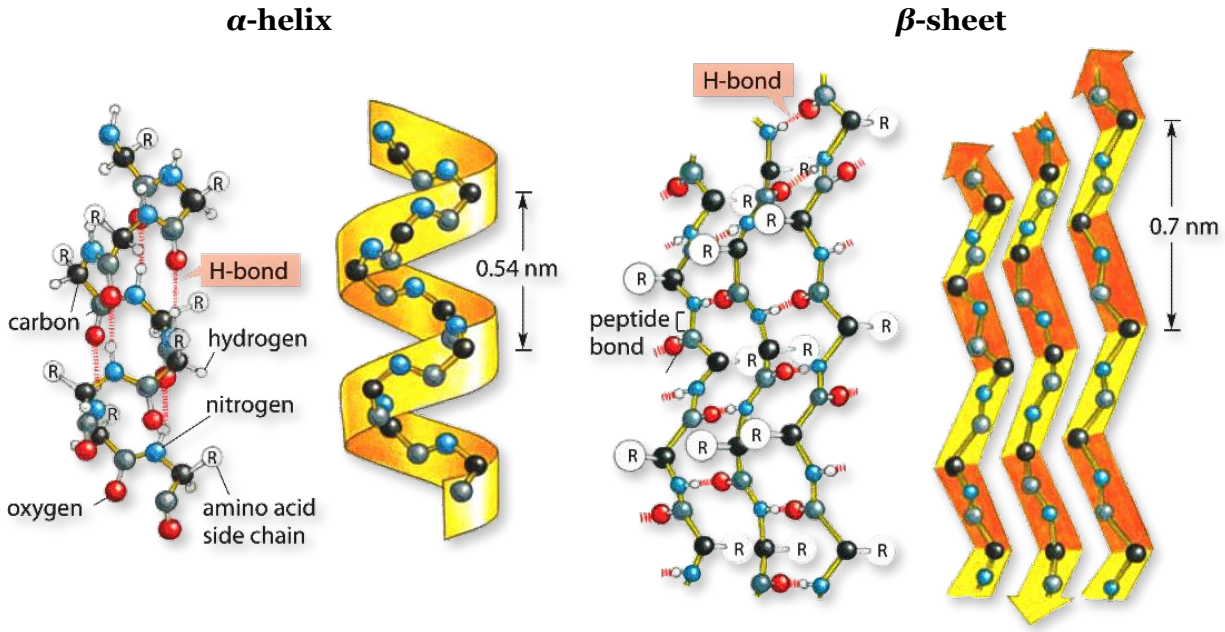


Figure 1.4: The *α*-helix and *β*-sheet crystal structure (adapted without permission from *Essential Cell Biology, 2004, Garland Science*)

However, the two elements described above are not the only ones that can be found in protein structures. Many conformational patterns, that are either variations of the basic two motifs described above, or completely different from them, are parts of the secondary structure, and many of them may have significant functionality. Some examples are the α_L-helix or left-handed *α*-helix, a rare type of helix; the 3_{10}-helix[7] (3 residues and 10 atoms per turn) and π-helix[8] (4.1 residues per turn) that differ from the α-helix in the number of residues per turn; the β-turns, and the random coils. The latter are located mainly on the protein surface, they are mostly hydrophilic and often involved in the formation of the active site of enzymes and in other crucial functional roles. Although random coils seem to be of unsymmetrical and non-periodic structure on first sight, there are major insights of potential periodicity in the level of the primary structure. In the next sections, the periodical occurrences of residues in coils will be expanded, and described by a systematic research in the protein world.

1.3 φ, ψ dihedral angles

In order to strictly define a secondary structure element, we need to comprehend the basic parameters that determine the conformation of a peptide. Assuming a dipeptide of residues *n* and *n+1*, the peptide group contains the C_α and C'=O group of the residue *n,*

as well as the NH group and the C_α atom of the residue $n+1$[6]. A peptide group is uncharged, and forms an inflexible plane, as the C'-N peptide bond cannot rotate, due to magnetic resonation with the C'=O bond[4]. Each amino acid residue backbone has two degrees of freedom that correspond to the torsion angles of the N-C_α and C_α-C' bonds (**Figure 1.5**). These dihedral torsion angles are called φ and ψ respectively. Every residue has a specific range of φ and ψ angles that can take, due to stereochemical restriction of the side chains. The dihedral angles can span from -180° to 180° and we conventionally define φ=0° and ψ=0° when the two bonds on each side of a C_α atom are on the same level[2]. The restriction of the angle values is very definitive for the formation of the secondary structure, and this explains the residue preferences on the various structure patterns. What we can conclude considering the above is that by knowing the two dihedral angles of the backbone, we can define the crystal structure of the backbone of a protein. Additionally, knowing the torsion angles of the bonds of the side chains (x_1,x_2...x_i), we can completely define the structure of the whole molecule.

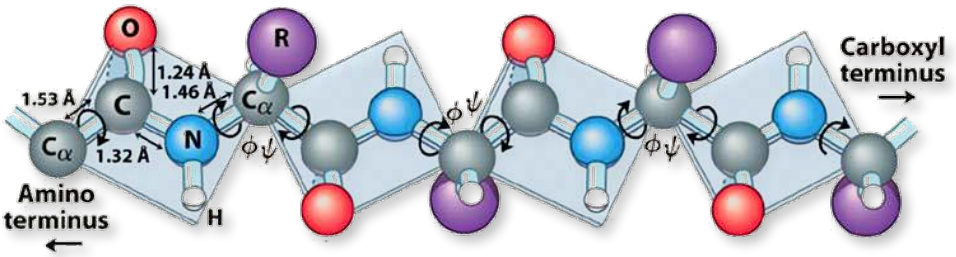


Figure 1.5: The planar peptide groups. The φ and ψ dihedral angles are the torsion measure of the N-C_α and C_α-C' bonds respectively (adapted without permission from *Nelson & Cox, Lehningher Principles of Biochemistry*)

1.4 The Ramachandran plot

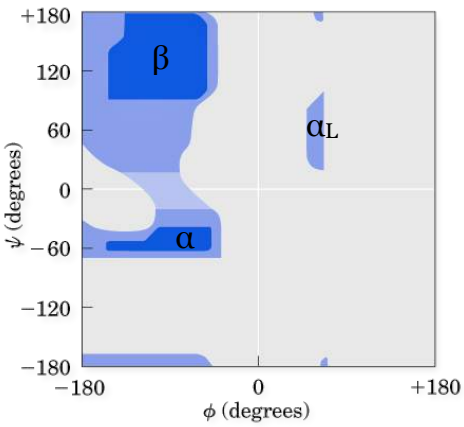


Figure 1.6: A typical Ramachandran plot with the favoured regions indicated (adapted without permission from *Nelson & Cox, Lehninger Principles of Biochemistry*)

As mentioned before, not all conformations of a residue backbone are energetically and stereochemically allowed, due to the short contacts between the atoms of adjacent residues[9]. In fact, the only amino acid that has a firmly broad range of allowed φ,ψ angles, is glycine, due to its symmetry as it has no side chain. The flexibility of glycine is very important, because this allows it to form plenty of different conformations. Other amino acids on the other hand, contain side chains that cause large steric hindrance, so their dihedral angle range is restricted. A good example is proline, which has a pyrrolidine side chain and this causes it to have a narrow range of allowed conformations.

In 1965 G.N. Ramachandran specified all the possible stereochemical conformations of the amino acid residues and plotted each one as dots in a 2-dimensional diagram, now known as the Ramachandran plot[9, 10]. As seen in **Figure 1.6**, the Ramachandran plot contains distinct regions, that correspond to the allowed φ,ψ angle values of the residues that form the various secondary structure elements. The three major regions are the α-helix region in the lower left quadrant, the β-sheet region in the upper left quadrant, and the α_L-helix region in the upper right quadrant. In more detailed Ramachandran plots, we can distinguish more secondary structure motifs and variations of the basic ones. For example, the broad β-sheet region contains distinct clusters, that correspond to the parallel and anti-parallel β-sheets.

An interesting exception that modifies the standard Ramachandran plot is the occurrence of glycine and proline in a polypeptide chain. These amino acids, as mentioned before, have a much different range of φ,ψ values than the other 18 amino acids, so they occupy different regions on the plot.

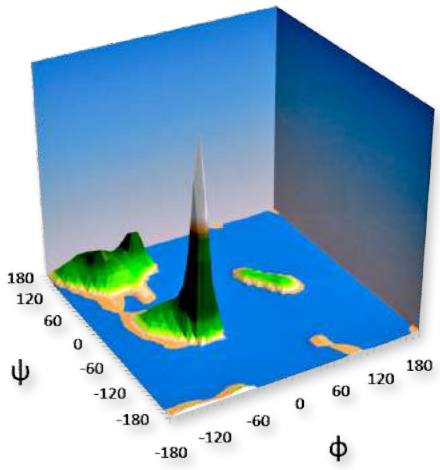


Figure 1.7: A 3D landscape Ramachandran plot showing the distribution of the secondary structure elements (adapted without permission from *Hollingsworth & Karplus, Biom. Con., 2010*)

Examining a Ramachandran plot like the one in **Figure 1.6,** we cannot easily understand the population and the frequency of the secondary structure elements, due to the collision of the data points on the 2D space. **Figure 1.7** shows a 3D Ramachandran plot created by S. Hollingsworth and P.A. Karplus using real protein X-ray crystallography data in resolution *<1.2Å*[11]. The plot clearly shows the high frequency of α-helices and β-sheets in proteins, as well as the scarcity of other secondary structure elements such as the α_L-helix.

In this thesis we will use the Ramachandran plot as a powerful tool for our studies, and it will be analysed furthermore, as it can give useful information on protein structure.

1.5 Linear groups

The conventional assumption on secondary structure, defines the secondary structure elements mainly by their hydrogen-bond patterns and the repetition of specific φ,ψ angle values on each residue. However, structural motifs formed of residues sharing similar φ,ψ angles and not following a regular pattern of hydrogen bonding, can be classified as secondary structure elements. A good example is the P_{II} (poly-L-proline *II*) motif[12], which is part of the secondary structure, although is does not have a strict hydrogen bond pattern.

To generalise issues on terminology, we could use the term linear groups to describe structural motifs characterised by a single φ,ψ-pair repetition[13, 14], not considering the hydrogen bonding. The common linear groups are shown in **Figure 1.8**.

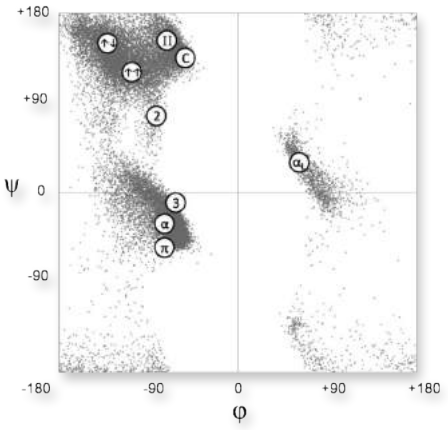


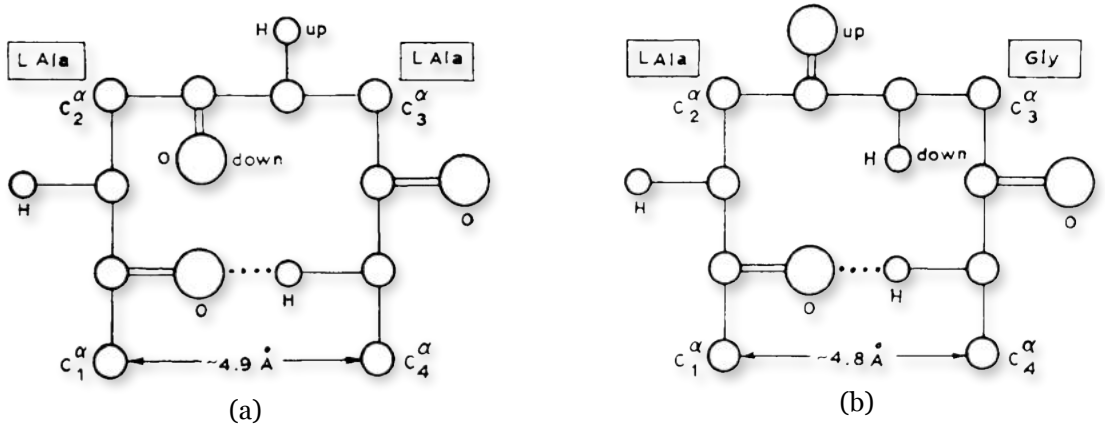
Figure 1.8: The nine common linear groups shown on the Ramachandran plot (adapted without permission from *Hollingsworth & Karplus, Port Sc, 2009*)

Hollingsworth and Karplus in an interesting publication (2009) on linear groups, define the shortest length of a linear group as three consecutive residues with similar φ,ψ angles ($\pm 10^0$). Their survey is based on real structures, and they recognise as true linear groups conformational patterns residing in three regions on the Ramachandran plot: The α-helix region (that contains the 3_{10}-helix and the π-helix), the β-sheet region, and the P_{II} region. The interesting fact is that the α_L-helix is not classified as linear group, as it does not satisfy the requirements of at least three adjacent residues with similar φ,ψ pairs[14].

Nevertheless, this introduction to linear groups was made in order to comprehend the non-linear conformations which is the subject of our research. For this reason, we will persist on the classic definition of linear groups, and consider them as conformations of repeating φ,ψ pairs.

1.6 Non-linear conformations - $(\varphi,\psi)_2$-motifs

Besides the standard, one-residue periodical conformations, there are structural motifs, in which two adjacent residues have distinct φ,ψ pairs. A representative example is the reverse turns, three-peptide group (four C_αs) conformations with a hydrogen bond between O_i and N_{i+3}[13]. According to Venkatachalam[15], there are three types of reverse turns, *I*, *II* and *III*, and their mirror conformations, *I'*, *II'* and *III'*. **Figure 1.9** shows the conformation of reverse turns of type *I* and *II* and the NH...O hydrogen bond.



(a) (b)

Figure 1.9: Conformation of type I (a) and II (b) reverse turns. C_α^2 (residue i+1) and C_α^3 (residue i+2) are the two central C_αs. The two central residues have different φ,ψ pairs. (adapted without permission from *Venkatachalam, Biopolymers, 1968*)

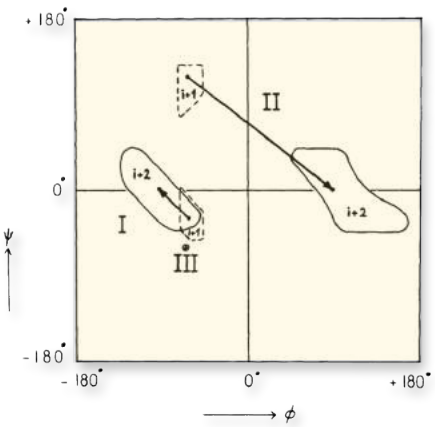


Figure 1.10: The two central residues of reverse turns as shown on the Ramachandran plot. Type *III* turns reside in the same region which is the 3_{10}-helix region (adapted without permission from *Schulz, Principles of protein structure*)

Representing reverse turns on a Ramachandran plot (**Figure 1.10**), we see the transition of the residues *i+1* and *i+2* between two distinct regions. Although not a linear group, reverse turns are indeed secondary structure elements. They are characterised in fact by a $(\varphi,\psi)_2$-motif and are very abundant in proteins.

$(\varphi,\psi)_2$-motifs are conformations formed by two similar consecutive φ,ψ-pairs[16]. Hollingsworth et el. on a 2013 publication, used real, four-residue fragments of protein structures to search for $(\varphi,\psi)_2$-motifs, and grouped these motifs according to their abundance. A considerable number of motifs on which the residues *i+1* and *i+2* have distinct φ,ψ angle values were found, including the reverse turns. These conformations are non-linear and are much of significance in our research, as the main goal is the identification of structures adopting continuous and recurrent $(\varphi,\psi)_2$-motifs.

1.7 Our goal

Considering the reverse turns and the general broad group of $(\varphi,\psi)_2$-motifs, we raised the question whether there is some type of extended conformation that is formed of consecutive repetitions of *two* or more φ,ψ pairs. In other words, a peptide fragment of certain length (e.g. five residues), with the adjacent residues residing in distinct regions on the Ramachandran plot. **Figure 1.11** shows a diagram of our hypothetical model.

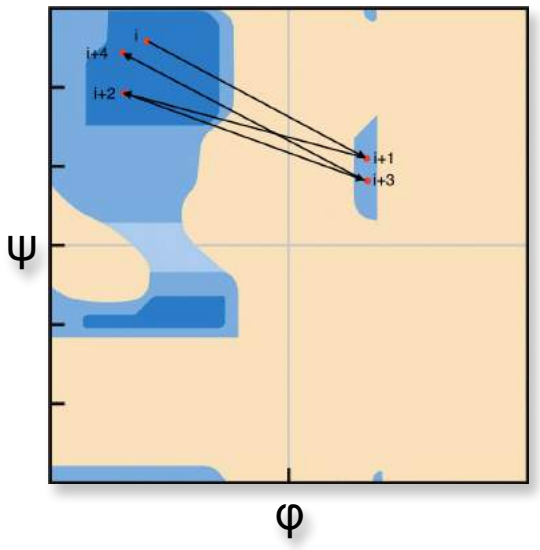


Figure 1.11: The hypothetical model of our research. The five central residues of the peptide fragment make continuous transitions between two distinct Ramachandran regions, in this case the β and α_L regions.

We algorithmically searched a large dataset of real protein structures, for peptide fragments that follow a pattern of transitions between two regions in the Ramachandran plot. For example, a peptide of which the first residue is in the β-sheet region, the second in the α_L-region, the third back in the β-sheet and the pattern continues up to five or more residues. Although these two regions are referred as an example, our algorithm is able to search for all possible transitions between any two (or more) regions.

This structural computational research aims to identify some standard patterns in random conformations such as coils, that seem non-periodic in the level of one residue. Therefore, taking into account the periodicity found in reverse turns, we thought that two-residue periodicities may also apply to random coils. The implementation was carried through the development of a probabilistic algorithm that uses as input real X-ray crystallography data from the *Protein Data Bank*[17]. The whole procedure and results will be thoroughly described in the following sections.

Something that must be mentioned, is that we searched for structures not containing glycine and proline residues. We did this in order to exclude structures such as reverse turns, which firmly contain glycine, and avoid any biased or false positive results.

Section 2
Methods

2.1 Programming languages

2.1.1 The *C* programming language

The *ANSI C* language is a programming language developed by Dennis Ritchie in the late 60's/early 70's at Bell Labs. It is a general-purpose, medium level language, mainly used for structured and linear programming, supporting all the fundamental control-flow constructions such as decision making, looping and statement grouping[18]. It was originally designed for the development of the *UNIX* operating system (which is almost exclusively used in our research), so it provides a perfect integration with it, in terms of functions and commands. Moreover, programs written in *C* are easily portable in other operating systems, as the language itself is, in a large extent, architecture independent.

We used *C* for the implementation of the main algorithm developed for the purposes of this research, as it is a straightforward, robust and easy-to-use language, perfect for handling large datasets and mathematical procedures. The algorithm does not demand complex parallel operations or object-oriented programming, so we chose *C* as the ideal language for our project.

2.1.2 The *Perl* programming language

Perl is a high-level, multi-purpose programming language developed by Larry Wall in 1987. It is interpreted, so it does not demand the use of a compiler, and highly portable. As *Perl* is truly open-source (under GPL licence), there is a vast variety of modules available for any purpose, such as *BioPerl*[19], which is package of great utility in bioinformatics. The fact that *Perl* is interpreted, makes it rather slow in comparison with compiled languages such as *C*, especially in mathematical calculations (~60 times slower). However, it is of great use in bioinformatics and computational biology, as it supports regular expressions.

The various ancillary scripts developed for processing our data and results, are all written in *Perl* and take advantage of its high-level built-in functions and regular expressions. We prefer to use *Perl* in our lab for various data processing needs, due to the high integration with our software and data, and for the flexibility and easiness of use.

2.1.3 The *R* statistical package

R is an open-source programming language and environment used for statistical computing and graphics[20]. It is highly capable of handling large datasets and has built-in functions for almost any statistical calculation and data mining. Like *Perl*, it is interpreted, and libraries available for free, expand its capabilities in many computational fields, such as artificial neural networks (ANNs).

We used *R* for creating and plotting various histograms needed in the research and for the clustering of the structures returned by our algorithm.

2.1.4 Other languages used

We mainly work on *UNIX* systems so we take advantage of *Bash* scripting and languages such as *AWK*, for task automation and quick data and text processing respectively. The source code of all the programs or scripts used, can be found in the **Appendix**.

2.2 Data Preparation

2.2.1 The *PDB* and *PISCES* databases

The data used in our research, derive from protein structures solved by X-ray crystallography. We collected these structures from the *Protein Data Bank* or *PDB*[17], a database available online, that contains a large archive of protein structures solved by the scientific community. By the time this thesis was written, the *PDB* contained 128,783 structures. A screenshot of the user interface of the *PDB* is shown in **Figure 2.1** and an example of a *PDB* protein structure file in **Figure 2.2**.

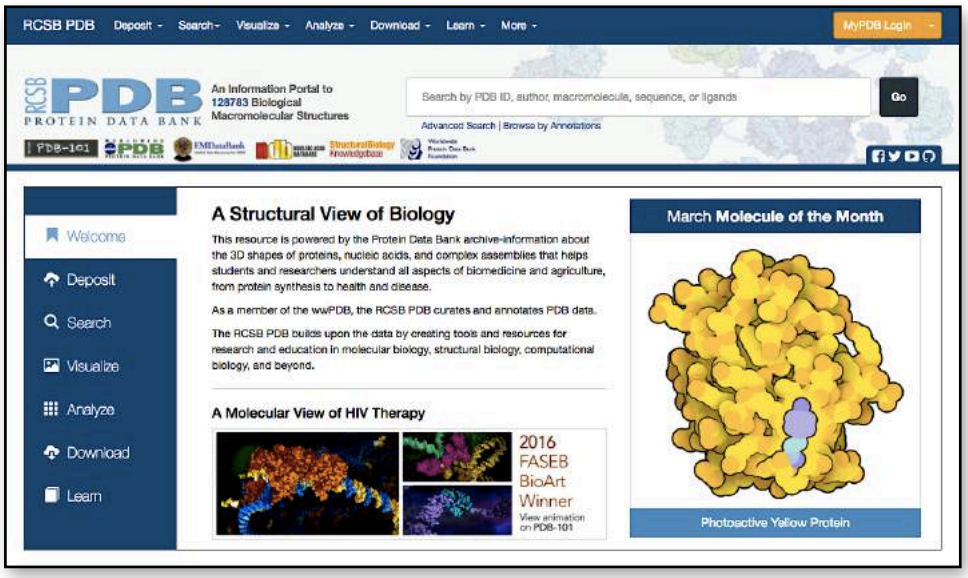


Figure 2.1: The homepage of the *Protein Data Bank*. (Available from: *http://www.rcsb.org/pdb/home/home.do*)

```
HEADER    EXTRACELLULAR MATRIX                    22-JAN-98   1A3I
TITLE     X-RAY CRYSTALLOGRAPHIC DETERMINATION OF A COLLAGEN-LIKE
TITLE    2 PEPTIDE WITH THE REPEATING SEQUENCE (PRO-PRO-GLY)
...
EXPDTA    X-RAY DIFFRACTION
AUTHOR    R.Z.KRAMER,L.VITAGLIANO,J.BELLA,R.BERISIO,L.MAZZARELLA,
AUTHOR   2 B.BRODSKY,A.ZAGARI,H.M.BERMAN
...
REMARK 350 BIOMOLECULE: 1
REMARK 350 APPLY THE FOLLOWING TO CHAINS: A, B, C
REMARK 350   BIOMT1   1  1.000000  0.000000  0.000000        0.00000
REMARK 350   BIOMT2   1  0.000000  1.000000  0.000000        0.00000
...
SEQRES   1 A    9  PRO PRO GLY PRO PRO GLY PRO PRO GLY
SEQRES   1 B    6  PRO PRO GLY PRO PRO GLY
SEQRES   1 C    6  PRO PRO GLY PRO PRO GLY
...
ATOM      1  N   PRO A   1       8.316  21.206  21.530  1.00 17.44           N
ATOM      2  CA  PRO A   1       7.608  20.729  20.336  1.00 17.44           C
ATOM      3  C   PRO A   1       8.487  20.707  19.092  1.00 17.44           C
ATOM      4  O   PRO A   1       9.466  21.457  19.005  1.00 17.44           O
ATOM      5  CB  PRO A   1       6.460  21.723  20.211  1.00 22.26           C
...
HETATM  130  C   ACY   401       3.682  22.541  11.236  1.00 21.19           C
HETATM  131  O   ACY   401       2.807  23.097  10.553  1.00 21.19           O
HETATM  132  OXT ACY   401       4.306  23.101  12.291  1.00 21.19           O
...
```

Figure 2.2: A sample from a *PDB* file (*1A3I*). (Available from: *https://en.wikipedia.org/wiki/Protein_Data_Bank_(file_format)*)

A large and non-redundant sample of proteins needed to be obtained from the *PDB* via *file transfer protocol* (*ftp*). A non-redundant *PDB* dataset is a dataset not containing duplicate entries. *PDB* contains many identical entries with different IDs (four-character codes that correspond to each individual structure entry, e.g. 2DOV). We wanted to exclude these, in order to avoid false results and artefacts, such as finding a conformational pattern, which is in fact a recurring sequence in many identical molecules. In order to do this, a list of *PDB IDs* needed to be created. We created this list using the protein sequence culling server *PISCES* [21]. *PISCES* is a tool that is able to produce lists of non-redundant entries from the entire *PDB* according to some criteria defined by the user. The criteria used by *PISCES* are the structure quality and the mutual sequence identity among the molecules. A screenshot of the interface of *PISCES* is shown on **Figure 2.3**. The criteria we used were:

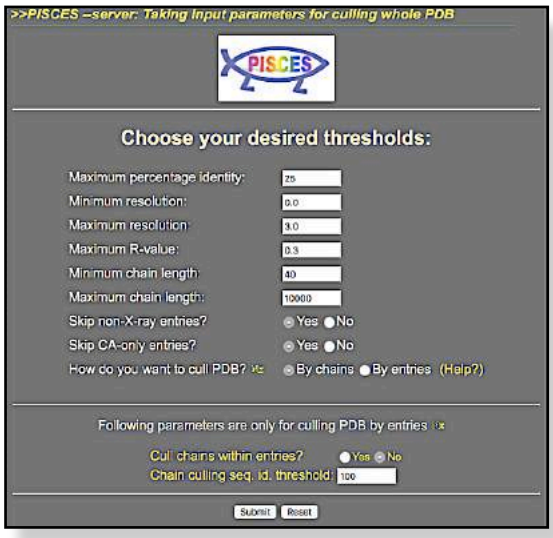


Figure 2.3: The *PISCES* interface (available at: *http://dunbrack.fccc.edu/Guoli/PISCES_ChooseInputPage.php*)

- Resolution ≤ 3.0*Å*
- 80% identity cut-off

The server returned a list of 29,211 *PDB* entries.

2.2.2 *ftp* scripts

The list created by *PISCES* is a one-column text file containing *PDB IDs* in capitals. **Script 1** (*list_lowercase.pl*) (source code in **Appendix**) modifies the list, converting in to lowercase, so it can be used as input to an ftp script. The modified list was used as input to **Script 2** (*pdb_ftp.pl*) which downloads all the entries in the list from the *PDB* server (*ftp.wwpdb.org/pub/pdb/data/structures/all/pdb*). 27.300 compressed files (*.tar.gz*) were downloaded. The discrepancy of 1911 unique IDs between the *PISCES* list and the downloaded files, is due to the multiple IDs for each polypeptide chain in the list (e.g. *2DoVA*, *2DoVB* etc.).

2.2.3 Extraction of the dihedral angles - *PROCHECK*

In order to search for any structural motif in a dataset of atom coordinates files, we needed to calculate all the *φ/ψ* dihedral angle values of the residues in all the molecules. As described in **Paragraph 1.3** and **Figure 1.5**, the *φ* angle measures the torsion of the N-C_α bond and the *ψ* angle measures the torsion of the C_α-C' bond. So in order to define the *φ* dihedral angle of a residue in the 3-dimensional space, we need the atom coordinates of the carbonyl C', the C_α, the amide N, and the next carbonyl C'. Respectively for the *ψ* angle, we need the coordinates of the amide N, the carbonyl C', the C_α and the next amide N atoms.

All these *x,y,z*-coordinates are provided from the *PDB* files and the angles can be calculated using structural analysis software. In our case, we used the program *PROCHECK*[22], which is a suite of tools for analysing the stereochemical parameters of a given protein molecule. It uses as input a *.pdb* file and outputs a series of text and PostScript files that contain information such as Ramachandran plots, bond lengths and main-chain or side-chain properties. *PROCHECK* can be run in a *UNIX* terminal as shown below:

```
>$ procheck [pdbfile] [chain (blank if all chains)] [resolution]
```

For example:

```
>$ procheck 2d0v.pdb A 3.0
```

One of the output files has a *.rin* extension and contains, among others, a list of the all the main-chain and side-chain bond angles of all the residues of a specific protein. An example of a *.rin* angle file is shown in **Figure 2.4** (φ and ψ columns are indicated):

```
                 φ       ψ
 1ASN A   1 h 999.90 110.76-171.51-166.89 -75.83 999.90 999.90 ...
 2ASP A   2 H -71.15 -29.75 178.71 -68.38 -35.04 999.90 999.90 ...
 3LYS A   3 H -64.11 -40.44 171.36 -55.60-166.66-152.03 147.09 ...
 4LEU A   4 H -58.99 -41.64 178.87 -79.41 174.72 999.90 999.90 ...
 5ILE A   5 H -56.02 -48.85 179.32 -53.96-157.02 999.90 999.90 ...
 6GLU A   6 H -64.52 -49.33 176.49 179.63 149.46 -66.39 999.90 ...
 7LEU A   7 H -61.43 -36.35 176.18 -62.51 154.76 999.90 999.90 ...
 8SER A   8 H -61.76 -23.31 174.48  68.51 999.90 999.90 999.90 ...
 9ASN A   9 h -71.93  -4.70 178.60 -58.27 -32.92 999.90 999.90 ...
10SER A  10 t-104.78 126.64 172.67-179.74 999.90 999.90 999.90 ...
...
```

Figure 2.4: A part of pdb2dov.rin file produced by *PROCHECK*. The first two floating point value columns contain the *φ* and *ψ* angles.

As mentioned above, two atoms on each side of a bond are needed to define the torsion angle of this bond (4 atoms in total). Thus, we cannot set a value for the *φ* and *ψ* angles on the residues of the -NH and -COOH termini respectively. *PROCHECK* assigns a 999.90 value in the angles of terminal residues or chain breaks. For our calculations we needed to run *PROCHECK* for all the *PDB* files (all chains, resolution 3.0*Å*), and extract only the specific φ/ψ columns from the *.rin* output file. **Script 3** (*angles.pl*) runs *PROCHECK* for a file using the same arguments as shown above, deletes all the output files except for the *.rin* file, extracts the *φ/ψ* angles and writes them in a new file. I.e. *angles.pl* uses as input a *PDB* file and outputs a *φ/ψ* angle file with a *.ang* extension (**Figure 2.5**).

PDB ID	Residue number	Chain	Residue ID	φ	ψ
...					
12AS	22GLU	A	25	-87.58	-40.52
12AS	23ARG	A	26	-75.45	-28.86
12AS	24LEU	A	27	-123.52	00.26
12AS	25GLY	A	28	999.90	44.28
12AS	26LEU	A	29	-104.16	137.59
12AS	27ILE	A	30	-102.19	152.92
12AS	28GLU	A	31	-76.58	134.05
12AS	29VAL	A	32	-124.31	155.11
12AS	30GLN	A	33	-73.82	129.10
12AS	31ALA	A	34	-86.98	137.64
...					

Figure 2.5: Angle file (*12AS.ang*) for the molecule 12AS after processing the *PDB* file with *angles.pl.*

The script needed to be run for every *PDB* file, using **Script 4** (*run_angles.pl*) for the automation of the process. The angle files produced were concatenated in one large file. The file however contained several occurrences of non-standard amino acid residues, so any residue not included in the 20 common, needed to be skipped. This process was

carried on by **Script 5** (*unknown_omit.pl*) which assigns a 999.90 *φ*-value in any residue with non-standard three letter code (e.g. UNK or SEC). The new dataset was named *res3.0_noUnk.ang*. Moreover, a second data set that excludes glycine and proline residues needed to be created. **Script 6** (*gp_omit.pl*), which assigns a 999.90 φ-value in glycines and prolines, was used to do this. This dataset was named *res3.0_nonUnkGP.ang*. Finally, the two data sets were processed with **Script 7** (*input_correction.c*) in order to refine them and edit some minor format issues (details shown in source code).

The dataset we used for the purposes of this thesis was the one not containing glycine and proline residues for the reasons explained in **Paragraph 1.7**.

2.3 Definition of a Ramachandran cluster

2.3.1 Fundamentals

The main point of the algorithm developed for the purposes of our research relies on the search of 5-residue fragments that follow three fundamental rules (see **Figure 1.11**):

I. Residues *i, i+2, i+4* must reside on a specific Ramachandran region.
II. Residues *i+1, i+3* must reside on another Ramachandran region.
III. The two regions must be *distinct*.

The method we developed to search for the pattern described, is based on the calculation of euclidian distances between two protein residues on the 2-dimensional Ramachandran space. These distances are then used in order to figure out whether two residues are likely to reside in the same Ramachandran cluster, or in different ones (more details on the algorithm are given in the following paragraphs). To do this, we first needed to define a Ramachandran region or cluster.

A generally admitted definition of a Ramachandran cluster, is a highly populated region that contains residues of the same secondary structure (thus similar *φ,ψ* pairs). However, it is difficult to strictly define it, because it is not possible to set clear and binary limits of the cluster. However, by using statistics, we are able to interpret fuzzy, experimental biological data like X-ray solved structures, and make them human-comprehensible.

Considering each residue as a dot in the 2-dimensional cartesian space, and smoothing the *φ,ψ* values in a reasonable range (e.g. 5^{o}), we can easily notice that the residues which populate a certain Ramachandran cluster, tend to converge around a maximum value (**Figure 1.7**). Furthermore, the data scatter around the local maxima smoothly, forming a Gaussian-like distribution. A Gaussian or normal distribution is the most common type of data distribution and represents the symmetrical convergence of

observations around a maximum (mean) value. The key parameters that define a Gaussian distribution of a sample *N*, are the *mean* (μ) which is the maximum value, and the *standard deviation* (σ) which measures the scatter of the observations around the around the *mean*.

- The basic Gaussian function is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \qquad \textbf{Equation 2.1}$$

- The *mean* (μ) equation is:

$$\mu = \frac{\sum x}{N} \qquad \textbf{Equation 2.2}$$

- The *standard deviation* (σ) equation is:

$$\sigma = \sqrt{\frac{\sum (x-\mu)^2}{\mathrm{N}}} \qquad \textbf{Equation 2.3}$$

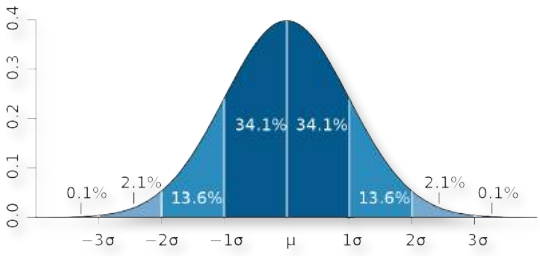


Figure 2.6: The normal or Gaussian distribution, or bell curve, and the percentage of the observations in the interval

Figure 2.6 shows a typical Gaussian distribution along with the probability percentage of the observations inside the bell curve. We see that the 68.2% of observations scatter in a range of 1σ away from the *mean,* 27.2% scatter 2σ away from the *mean* and the rest are in a range >2σ away from the *mean*. These percentages correspond to the probability of a random observation to be within each value range, and we used this principle for our calculations.

The conclusion derived from the above insights, is that we can define a Ramachandran cluster as a Gaussian distribution, with a strictly defined *mean* (μ) and *standard deviation* (σ). However, our calculations use euclidian distances between residues as data, and we need the distribution of the distances instead of the residues themselves. Thus, the *mean* value of the distribution is set to 0, which is the minimum possible distance between two residues (the two φ,ψ pairs are identical, according to the smoothing rate). Also, only positive values, on the right side of the *mean* make sense. More details on the construction of the Gaussian curve are given in the following paragraphs.

2.3.2 The problem of circular periodicity in dihedral angles

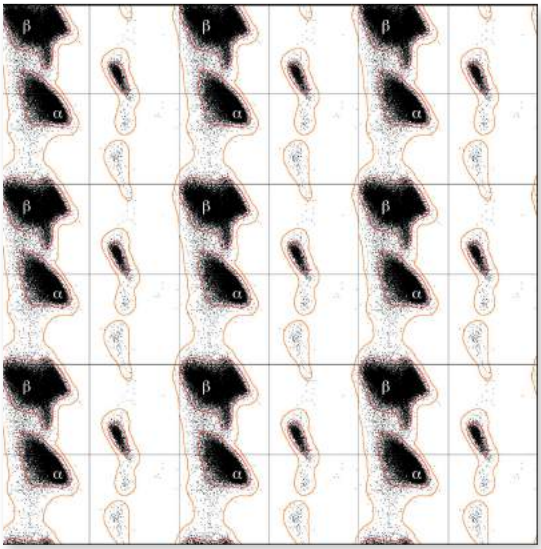


Figure 2.7: The periodicity of the Ramachandran plot regions. 9 identical plots shown as a grid.

The process of euclidian distance calculation requires the φ,ψ coordinates of two residues, and although the procedure might seem straightforward, there is an issue that must be considered. Some regions on the Ramachandran plot are not limited between the -180° and +180° range. For example, the β-sheet region stops at the top of the plot and continues on the bottom. Apparently, two residues of the same region, may seem distant on the typical Ramachandran plot, with one located at the top, and the other at the bottom. **Figure 2.7** shows 9 copies of a Ramachandran plot in a grid order. We can clearly see the periodicity of the various regions, and this is explained by the circular range of the dihedral angles (e.g. an angle with 180° value is identical with -180° angle).

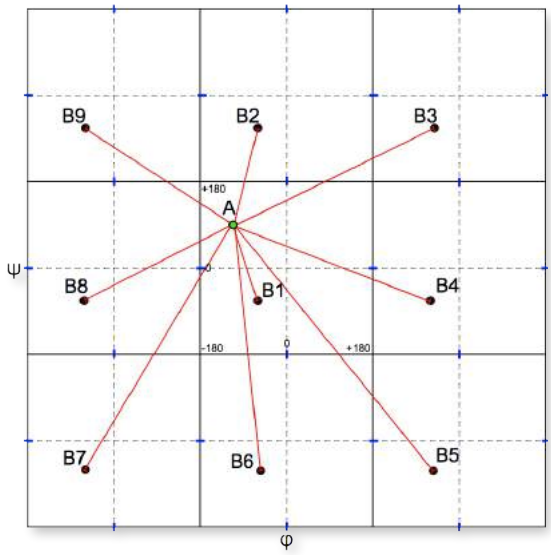


Figure 2.8: A diagram that summarises the method to solve the problem of periodicity in the Ramachandran plot. 9 plots shown as a grid. Red lines represent the possible the difference vectors on the 2D space. The one with the minimum measure is the true one.

In order to calculate the distance between any two residues on the 2-dimensional space, we needed to eliminate the problem of periodicity. **Figure 2.8** summarises the method of solving the issue. The euclidian distance d between two points $a(x_1 , y_1)$ and $b(x_2 , y_2)$ on the 2-dimensional cartesian space is:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad \textbf{Equation 2.4}$$

Considering a grid of 9 Ramachandran plots, with one plot in the centre and 8 complementary around it, we need to calculate the euclidian distance between a residue A and a residue B. There are 9 possible distances, between the static residue $A(\varphi_o , \psi_o)$ in the central plot, and the symmetric residues $B_i(\varphi_i , \psi_i)$ in all the 9 plots. After calculating them, the minimum distance is the one that makes sense and will be used further as data for creating the distribution. Residue A φ,ψ coordinates are (φ_o , ψ_o), residue B φ,ψ coordinates are (φ_1 , ψ_1), and knowing that the length of the Ramachandran square is 360, the coordinates of all the B symmetric residues will be:

- $B_2(\varphi_1 , \psi_1+360)$
- $B_3(\varphi_1+360 , \psi_1+360)$
- $B_4(\varphi_1+360 , \psi_1)$
- $B_5(\varphi_1+360 , \psi_1-360)$
- $B_6(\varphi_1 , \psi_1-360)$
- $B_7(\varphi_1-360 , \psi_1-360)$
- $B_8(\varphi_1-360 , \psi_1)$
- $B_9(\varphi_1-360 , \psi_1+360)$

Using the **Equation 2.4** and replacing the *x,y* values with the *φ,ψ* coordinates, we calculate the following distances and find the minimum one:

- $d_1 = AB_1$
- $d_2 = AB_2$
- $d_3 = AB_3$
- $d_4 = AB_4$
- $d_5 = AB_5$
- $d_6 = AB_6$
- $d_7 = AB_7$
- $d_8 = AB_8$
- $d_9 = AB_9$

The method described was used not only for gathering the data to construct a histogram and define the Ramachandran cluster, but also on the main algorithm where there is also a distance calculation procedure.

2.3.3 Histogram construction

In order to find the distribution that corresponds to the Ramachandran regions, we needed to construct a histogram of all the euclidian distances (i.e. difference vectors) between two residues *i* and *i+2* (the ones that need to be in the same Ramachandran cluster, as defined by the rules of the pattern we search for). For this purpose we developed a program that uses as input φ,ψ dihedral angles (*res3.0_noUnkGP.ang* file in this case) and outputs a list of *[i - i+2]* distances. The program is written in *C* and the source code is available on the **Appendix** (*histogram1-3.c*). The algorithm used is based on the distance calculation method described in the previous paragraph. **Figure 2.9** shows a short pipeline of the program:

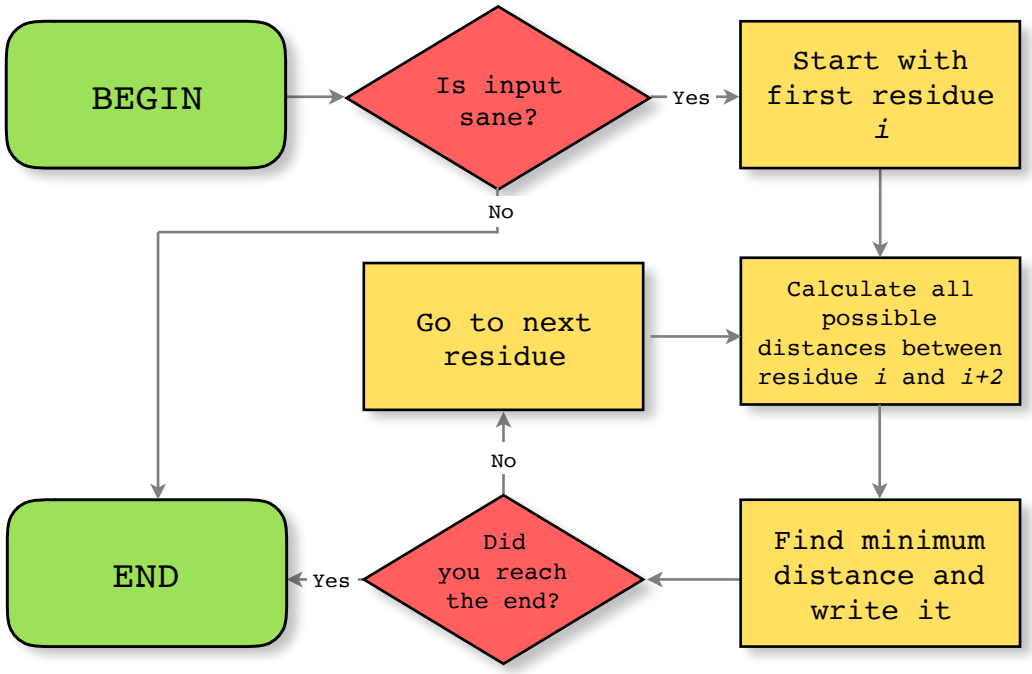


Figure 2.9: Flow chart of the program *histogram1-3.c*

The distances are written in a file named *distances1-3_noUnkGP_hist.dat*. A sample of the output is shown in **Figure 2.10**:

```
12.902238
11.915625
25.765066
32.565891
25.221869
37.106285
15.809918
12.218562
13.582341
15.192897
19.437786
22.414783
28.101343
 8.660682
...
```

Figure 2.10: Sample of the file containing the residue pair euclidian distances on the 2D Ramachandran space

The next step was the construction of a histogram using as data the distances of the above file. The histogram was created using the *R* statistical package, and is shown in **Figure 2.11**:

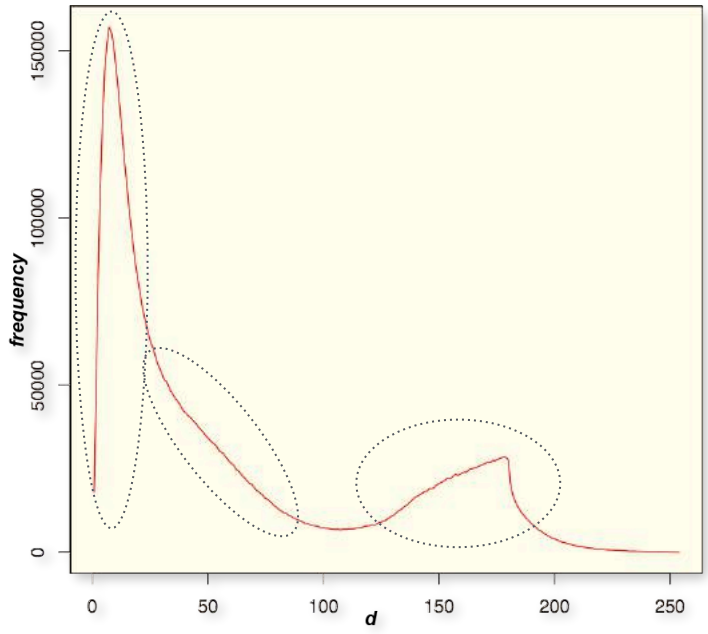


Figure 2.11: The distances (*d*) histogram. The circles indicate the three separate curves that form a larger one.

The distribution of the data, as shown on the histogram, is rather interesting, and gives us insights to define a Ramachandran cluster. The curve can be analysed, and described as the aggregate of three subordinate curves as shown in the circles in **Figure 2.11**. The curve on the left has a bell form and contains the most frequent distances. This fact leads us to conclude that this curve corresponds to the distances of residues found in the same Ramachandran region. The curve in the middle is the shoulder of the first bell curve and it

probably corresponds to distances between residues of a broader Ramachandran cluster, most likely the β-sheet region, which contains two distinct sub-regions: the parallel and anti-parallel β-sheet. The third curve (in the circle on the right), is most likely to show the frequency of distances between residues of clearly distinct clusters as we can notice that is has a global maximum in ~180°. The conclusion of all the above is that we can use the curve on the left as the one that most accurately represents a cluster on the Ramachandran plot. The curve is of Gaussian form, an the next step was to find the Gaussian function that fits it.

2.3.4 Non-linear regression fitting

To estimate the parameters that best represent the Ramachandran cluster, we needed to fit the data in a function, specifically a Gaussian function. With a quick look on the curve of the data points in **Figure 2.11**, we can safely say that the data do not follow a linear model, but instead, it is a non-linear aggregate of three bell curves.

The method used to fit the mixture of the three distributions is the *non-linear regression*. The basic steps of this method are:

a. *Initialisation*, by setting the function we want the data to fit in (*chi-by-eye* method).
b. Definition of *starting values* for the parameters of the function.
c. *Alteration* of the parameters until the *Root Mean Square Error (RMSE)* minimizes, thus the curve is most accurately fitted in the function.

Non-linear regression requires many value alterations and computations until the standard error minimizes (*brute-force* method), something a human is not able to do quickly. Many computational tools can be used to efficiently carry on this process; we chose the *R* statistical language which has a built-in function (*nls*) specifically for *non-linear regression*. The *nls* function requires the user to define the fitting function and the starting values of the function parameters. A key principle of *nls,* is that the user has to estimate the starting values as accurately as possible, by studying the data curve (large deviation of the starting values from the final values will cause the process to crash).

The first factor of the Gaussian function is the height of the curve (see **Equation 2.4**), and can be simplified, so the function can be written as:

$$f(x) = ke^{\frac{-1(x-\mu)^2}{2\sigma^2}} \qquad \textbf{Equation 2.5}$$

where μ is the *mean*, and σ the *standard deviation*.

The mixture of the three Gaussians is a function of the form:

$$f(x) = ke^{\frac{-(x-\mu_1)^2}{2\sigma_1^2}} + le^{\frac{-(x-\mu_2)^2}{2\sigma_2^2}} + me^{\frac{-(x-\mu_3)^2}{2\sigma_3^2}} \qquad \textbf{Equation 2.6}$$

The *mean* of the first Gaussian (μ_1) is set to 0 for the reasons described in **Paragraph 2.2.2**. Studying the histogram we estimated the following starting parameters:

- $k = 100000$
- $l = 30000$
- $m = 15000$
- $\mu_2 = 50$
- $\mu_3 = 125$
- $\sigma_1 = 10$
- $\sigma_2 = 10$
- $\sigma_3 = 20$

The *R* script uses as input the histogram data, it fits them in the function using the *nls* function and outputs a summary of the parameters of the function calculated along with a plot of the histogram and the fitted function, in superposition. The source code of the *R* script can be found in the **Appendix**. To optimise the fit we removed the data points on the left side of the first peak. **Figure 2.12** shows the summary of the parameters of the fitted function. The *standard deviation* (10.22) of the first Gaussian is highlighted. The script returned a negative *sd* because in the function it was squared, and could converge in +10.22 as well as -10.22. *Standard deviation* however is always positive, so we use its absolute value.

```
Formula: y ~ k * exp(-(x - 0)^2/(2 * s1^2)) + l * exp(-(x - m2)^2/(2 *
    s2^2)) + m * exp(-(x - m3)^2/(2 * s3^2))

Parameters:
     Estimate Std. Error t value Pr(>|t|)
k    1.250e+05  7.594e+03  16.456  < 2e-16 ***
l    9.538e+04  1.475e+04   6.465 2.56e-10 ***
m    4.321e+04  3.911e+02 110.487  < 2e-16 ***
m2 -2.908e+01  1.376e+01  -2.113   0.0351 *
m3   1.640e+02  2.637e-01 622.085  < 2e-16 ***
s1 -1.022e+01  4.165e-01 -24.534  < 2e-16 ***
s2   6.198e+01  4.774e+00  12.982  < 2e-16 ***
s3   2.235e+01  2.693e-01  83.015  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2707 on 466 degrees of freedom

Number of iterations to convergence: 12
Achieved convergence tolerance: 9.151e-06
```

Figure 2.12: The summary output of the *R* script after fitting the histogram curve. The σ of the first Gaussian distribution is highlighted in yellow.

Figure 2.13 shows the plot of the histogram and the fitted curve.

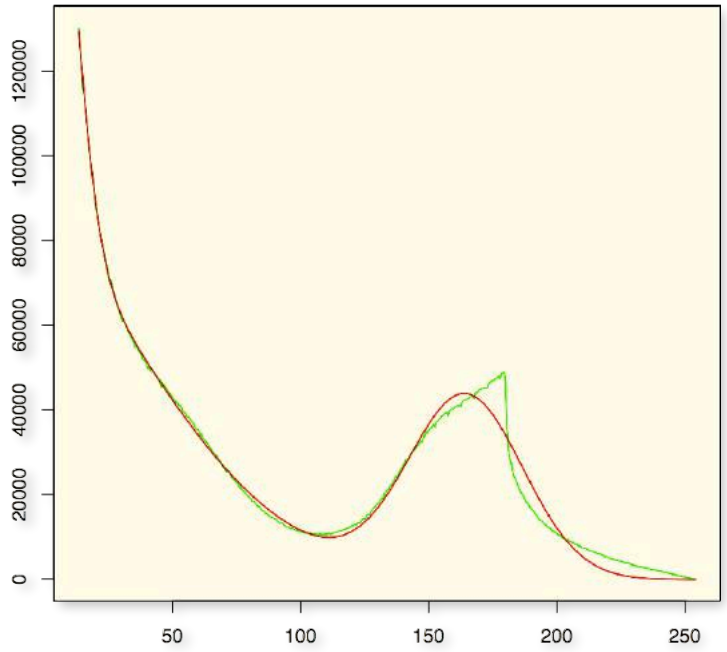


Figure 2.13: Plot of the histogram curve (green) and the fitted function curve (red)

2.3.5 Clustering parameters

The parameters returned by the *non-linear regression* script can be used to finally define the Ramachandran region in the terms of our research methods. We can safely say that a Ramachandran region follows a Gaussian distribution with *mean=0* and *standard deviation=10.22*. The value of *standard deviation* is a fundamental part of the main algorithm, which is described in the following paragraph. The conclusion after considering the above insights, is that a Ramachandran region is a Gaussian distribution with the following function: (**Equation 2.7**):

$$f(x) = 125{,}000\, e^{\frac{-x^2}{208.8968}}$$ **Equation 2.6**

2.4 Main algorithm

2.4.1 Principles

The algorithm we developed in the quest for the *two-residue periodicity* patterns in protein structures, relies again on distances on the 2-dimensional Ramachandran space. However, when using such tools like euclidian distances, that are not directly structure-based, but geometry-based, it is difficult to classify the peptide fragments in a binary form. In other words, we cannot say that a structure found by this method strictly follows the pattern or not. Therefore, we needed to find a method to characterise our results in a

probabilistic way. A widely used, *fuzzy logic* based method is the scoring of the results using the principles of probability theory.

Before we dwell into details on probabilities, we need to recall the three rules of the pattern we search for, stated in **Paragraph 2.2.1**. We can summarise these rules in a diagram, where the necessary distances are indicated: Δ_n are the distances between residues of the same Ramachandran cluster, and d_n the distances between residues of distinct clusters (**Figure 2.14**).

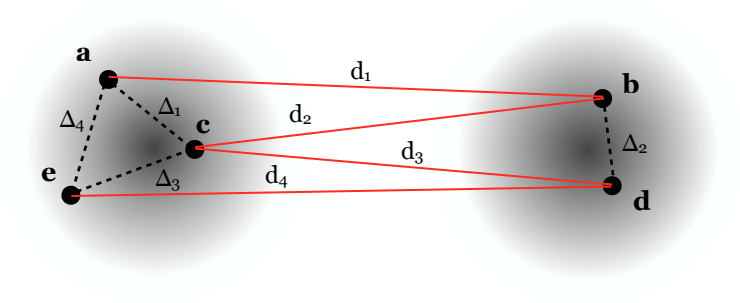


Figure 2.14: A diagram of the two-residue periodicity pattern in a peptide fragment which contains five central residues. The two gradient circles represent two different Ramachandran clusters. The continuous lines are the distances between the residues of distinct clusters (Δ_n distances) and the dotted lines are the distances between residues of the same cluster (d_n distances).

Our program uses a simple algorithm to find the pattern. The basic steps of the algorithm are stated below:

1. Calculation of the above mentioned distances in all the possible 5-residue fragments -while skipping the chain breaks.
2. Conversion of the distances to probabilities, using the *standard deviation* that defines the Ramachandran cluster
3. Calculation of the log-odds of every probability.
4. Aggregation of the log-odds that correspond to every distance; the log-odds sum is the score of the peptide fragment
5. Sorting of all the results by their score; high-scored fragments have a higher probability to match the hypothetical model.

The second step, which is the conversion of the distances to probabilities, is the most important part of the algorithm, and the reason why we needed to define a Ramachandran cluster as a Gaussian distribution. We used the *error function* (*erf*)[23] for this procedure. The *error function* is the integral of a Gaussian distribution and expresses the probability of an observation x. It is defined as:

$$erf(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-(x-\mu)^2}{2\sigma^2}} dx$$

Equation 2.7

We can also calculate the reverse probability by using the *complementary error function* (*erfc*) which is defined as:

$$erfc(x) = 1 - erf(x) \qquad \textbf{Equation 2.8}$$

Consequently, considering a Ramachandran cluster as a normal distribution of distances *d,* with $\sigma=10.22$ and $\mu=0$, the *error function* can convert the distances into probabilities. We need to calculate two kinds of probabilities:

- The probability $P_{\Delta n}$ of two residues to be in the same cluster (distances Δ_n). This means that when the distance between residues i and $i+2$ increases, the probability decreases (distance and probability are inversely proportional). We used the *erf* for this:

$$P_{\Delta_n} = erf\left(\frac{\Delta_n}{2\sigma\sqrt{2}}\right) \qquad \textbf{Equation 2.9}$$

- The probability P_{dn} of two residues to be in different clusters. This means that when the distance between residues i and $i+1$ increases, the probability also increases (distance and probability are proportional). This is the reverse probability so we used *erfc*:

$$P_{d_n} = erfc\left(\frac{d_n}{2\sigma\sqrt{2}}\right) \qquad \textbf{Equation 2.10}$$

For the 5-residue pattern, we calculated a total of 8 probabilities for each peptide fragment. Every probability contributes to the final score, so the higher the aggregate of the 8 probabilities, the higher the match to the hypothetical model. However, we found out that the aggregate of the probabilities themselves as a score for the results, is not very representative; some high scored peptides did not match the pattern, while peptides with lower score did. To solve this problem we used the aggregate of the log-odds ratio, instead of the probabilities themselves, and the scoring was significantly more representative. The log-odds ratio of a probability *P*, is given by the following equation:

$$\log odds(P) = \frac{P}{1-P} \qquad \textbf{Equation 2.11}$$

Our program implements all the above principles in a systematic way, for every possible peptide fragment, using as input the file containing the dihedral angles. It should be noted that the calculation of the distances is carried on by the method described in **Paragraph 2.2.2**.

2.4.2 Pipeline

A schematic summary of the algorithm can be shown in the following pipeline (**Figure 2.14**):

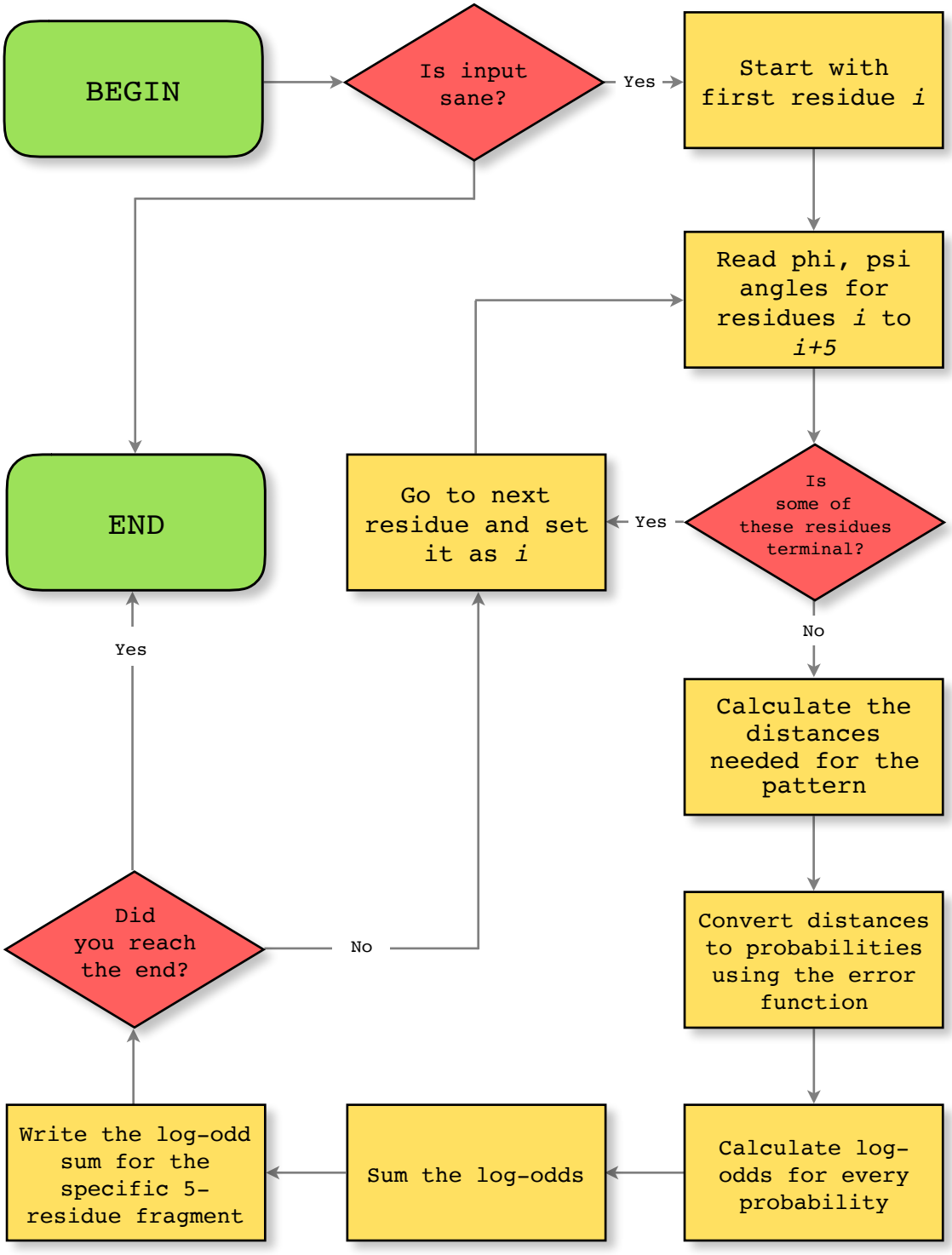


Figure 2.14: Flow chart of the program *dif_vectors.c*

The source code of the program (*dif_vectors.c*) can be found in the **Appendix**.

2.5 Structure clustering and overall procedure

The output of the program *dif_vectors* is a text file containing all the possible 5-residue fragments, scored according to the match on our hypothetical model. Every line of the file corresponds to a certain fragment, and contains, beside the score, information such as the first and the last residue, the *PDB ID* of the molecule and the chain. A sample of the output file is shown in **Figure 2.15**:

```
...
 -2.2269513468047 2D0S     70LYS  A   70     74TRP  A   74
 -2.4814835273328 2D0S     71ILE  A   71     75VAL  A   75
 -0.1424664302906 2D0S     72VAL  A   72     76LEU  A   76
 -5.4832997065578 2D0S     73ARG  A   73     77THR  A   77
-27.9728959773400 2D0S     74TRP  A   74     78LEU  A   78
 -3.4051897562880 2D0V      2ASP  A    2      6GLU  A    6
 -0.6846187450998 2D0V      3LYS  A    3      7LEU  A    7
 -3.0800667320617 2D0V      4LEU  A    4      8SER  A    8
 -7.3779921939430 2D0V      5ILE  A    5      9ASN  A    9
-51.5885065028282 2D0V      6GLU  A    6     10SER  A   10
 17.5482587940678 2D0V      7LEU  A    7     11ASN  A   11
 -1.7855995844273 2D0V      8SER  A    8     12GLU  A   12
-12.3305597026372 2D0V      9ASN  A    9     13ASN  A   13
-19.0835522155755 2D0V     10SER  A   10     14TRP  A   14
-40.3097017553709 2D0V     11ASN  A   11     15VAL  A   15
 -5.1063778528191 2D0V     12GLU  A   12     16MET  A   16
  7.0104079613309 2D0V     19LYS  A   19     23SER  A   23
  0.5462487411286 2D0V     20ASN  A   20     24ASN  A   24
 49.0366337270417 2D0V     21TYR  A   21     25ASN  A   25
 88.2788064456197 2D0V     22ASP  A   22     26TYR  A   26
  2.1951844095155 2D0V     23SER  A   23     27SER  A   27
...
```

Figure 2.15: Sample of the *probabilities.dat* file. The first column contains the log-odds sum score, the second is the *PDB ID*, the third and the sixth columns contain the first and the last residue IDs respectively. The fourth and seventh columns contain the chain, and the fifth and eighth columns contain the residue numbers respectively.

The next step of the procedure is the culling of the high-scored peptide fragments and then the classification according to their structure similarity. We are going to describe a protocol for extracting the peptides, and clustering them according to their structure similarity. The procedure combines programming, task automation and usage of structure analysis software such as *Carma*[24], *Grcarma*[25], *PyMol*[26] and *VMD*[27]:

- *Carma* is a molecular dynamics trajectory analysis program; we used it to create artificial trajectories and for cartesian cluster analysis.
- *Grcarma* is more user-friendly version of *Carma,* which supports graphical interface.
- *VMD* and *PyMol* are molecular visualisation and graphics programs which we used to illustrate the structures we found. *PyMol* supports graphical rendering with *Ray Tracing* to produce high quality 3D models.

The protocol we used is stated below. All the *UNIX* commands are presented in distinct font:

1. Reverse sorting of the score file using:

```
sort -s -n -r -k probabilities.dat > probs_sorted.dat
```

2. Production of *PDB* files for every high scored hit (in this case a threshold of 100 was set, so we needed the first 16.000 hits) by doing the following:

```
head -16000 probs_sorted.dat > top16000.dat
./pdb_extractor.pl -d -h top16000.dat
```

pdb_extractor.pl is a *Perl* script used for culling the structures found by the program *dif_vectors*. It downloads the specific files from the *PDB* via *ftp* and extracts only the residues of the peptide fragments specified in the score file. The flag *-d* is for deleting the initial *PDB* files after extracting the fragments, and *-h* is for suppressing any duplicate entries of homopolymeric molecules. The source code can be found in the **Appendix** (**Script 9**)

3. 8190 *PDB* files remained after excluding the duplicates. Some entries have duplicate atoms probably due to protein discrete disorder. To remove such entries we can filter the *PDB* files with two *Bash* scripts, that implement an *AWK* command:

   - For filtering the molecules that contain a sane number of C_a atoms, we used the *ca_filter.sh Bash* script (source code in the **Appendix**, **Script 10**).
   - For filtering the molecules that contain a sane number of backbone atoms (N, C_a, C, O) we used the *backbone_filter.sh Bash* script (source code in the **Appendix**, **Script 11**)

   390 files were removed, and 7800 remained.

4. Having a pure database of potential hits, the next step was to extract the C_as or the backbone atoms from every molecule, to use them for creating an *Root Mean Square Deviation* matrix for clustering. The *Root Mean Square Deviation* (*RMSD*) is a way to measure the average distance of atomic positions of two peptides in superposition, and is defined as:

$$RMSD = \sqrt{\frac{1}{n}\sum_{i=1}^{n} d_i^2}$$

where n is the number of pairs of equivalent atoms and d_i the distance of the atoms of the i^{th} pair[28]. The lower the *RMSD* between two peptides, the higher their structure similarity. The *RMSD* matrix contains the *RMSD* values between all the frames (pseudo-frames in our case) of a trajectory. The matrix can be analysed and the structures can be clustered by setting an *RMSD* cut-off. We chose this method because it is directly structure-based (it uses cartesian coordinates) and easy to implement using the *R* statistical package. The extraction of the atoms was carried on with the following commands:

```
grep –no-filename ‘ CA’ * > all_CA.pdb
 or
awk ‘{$3==”CA” || $3==”C” || $3==”N” || $3==”O” {print}’ *.pdb
>> all_backbone.pdb
```

5. Validation of the sanity of the files:

```
ls -l *.pdb | wc -l
```

The two files are sane if the number returned is dividable by 5 for the C_a file, or by 20 for the backbone file. Validation in every step is crucial, in order to avoid unexpected problems during the whole procedure.

6. The method we chose for the clustering of the structures, required the construction of a fake molecular dynamics trajectory, which contains all the structures found by our program. In order to do this, an “END” had to be put in the end of every molecule in the global *.pdb* file (*all_CA.pdb* or *all_backbone.pdb*):

`awk ‘{print}; NR%5==0 {print “END”}’ all_CA.pdb > out` (for *all_CA.pdb*)
or
`awk ‘{print}; NR%20==0 {print “END”}’ all_CA.pdb > out` (for *all_backbone.pdb*)

There must be an “END” every 5 (or 20) lines, and in the last line.

7. The large *.pdb* file looks like a trajectory and contains all the hits in alphabetical order. The filtered *.pdb* files were stored, and a numbered list of them was created. This list was used in the late stages of the procedure, in order to assign the structures of the trajectory to the initial files containing them.

8. The trajectory *.pdb* file was used as input to *VMD*. The structures were not yet superimposed so they could not be visualised. *VMD* was used to produce a *.dcd* file for the cluster analysis.

9. The cluster analysis was carried on by the program *Carma*. *Carma* requires a *.dcd* file and a *.psf* file (*protein structure file)*. A pseudo *.psf* file was created with the program *pdb2psf* (*Glykos NM*, **Script 12**, source code in the **Appendix**) and the atom and residue numbers were modified to be 1,2,3 etc.

10. An *RMSD* matrix of the C_α atoms was created using *Carma* (The input files were *all_backbone.dcd* and *all_backbone.psf*. Although the above steps described how to create a C_α file as well, we used the backbone file in our calculations). For the 7800 structures of the pseudo-trajectory, we created an 7800x7800 matrix with step of 1 frame, using the following command:

```
carma -verbose -cross -step -segid A all_backbone.dcd
all_backbone.psf
```

11. Construction of a hierarchical dendrogram of the structures, using the *UPGMA* algorithm. The clustering was done using *R*, with an *RMSD threshold* = 1*Å* (structures with *RMSD* ≤ 1*Å* are joined in the same cluster. An *R* script was used (**Script 13**, source code in the **Appendix**) as shown below:

```
Rscript clustering.R | tee LOG
```

LOG file contains a summary of the clustering procedure, *all_clusters.list contains* the list of clusters found, and a *PostScript* file contains the dendrogram (*RMSD* matrix and dendrogram can be found in **Section 3: Results**).

12. The cluster list was then separated into distinct lists, each one containing one cluster. This was done by running *lists.sh* (**Script 14**, source code in the **Appendix**). Every list links the frames with the initial structures of the data set. The structures can be assigned to the initial *pdb* files according to the list created in step 7.

13. The structures were reordered by running *reorder.sh* (**Script 15**, source code in the **Appendix**). The script runs *Carma* to sort the structures of the lists into new *.dcd* files.

14. The *.dcd* file contains non-superimposed structures. To superimpose them we used *superimpose.sh* (**Script 16**, source code in the **Appendix**). The script fits the

structures using *Carma*.

15. Production of *.pdb* files of superimposed structures for every cluster, using *final_pdbs.sh* (**Script 17**, source code in the **Appendix**). The script takes as input the *.dcd* files and runs *Carma* for every one of them to produce *.pdb* files.

16. To create 3-dimensional visualisations of the clusters, we used *PyMol*, which is capable of rendering high-quality textures and 3D models.

17. The final step was to check the structures for residue conservation; we did this by creating a *sequence logo* for each cluster of aligned peptides[29]. Sequence logos are constructed by letters that correspond the residues of a protein (or nucleotides in a DNA/RNA chain). The letters are stacked in each position, and their relative size represents their frequency in the cluster. The total height of each stack measures the conservation of the residues in this position in *bits*. For proteins, the *bits* range between 0 and 4. The *sequence logos* were created using the online tool *Weblogo*[30].

Section 3
Results

Our research returned a significant number of hopeful results, which are presented in this section. As stated previously, the clustering procedure included the construction of a 7800x7800 *RMSD* matrix for 7800 candidate 5-residue peptide fragments (*RMSD* of the C_α atoms). The scores of the fragments range from *206.9* (highest score) to *100* (cut-off). 39 clusters were found by the *UPGMA* algorithm, using *1.0Å RMSD* cut-off.

It is generally accepted from empirical observations, that superimposed structures with *RMSD* < *2.0Å* have close structure similarity[31]. We chose the low *RMSD* threshold of *1.0Å* after studying the hierarchical dendrogram constructed by the *UPGMA* algorithm (**Figure 3.2**). The dendrogram shows a high increase in the number of clusters in *RMSD* values lower than *1.0Å,* so we considered this threshold to be optimal for clustering peptides of high structure similarity. Before presenting the clusters, we are showing the *RMSD* matrix (**Figure 3.1**) as well as the hierarchical dendrogram (**Figure 3.2**). Also **Table 3.1** summarises the members of all the clusters found:

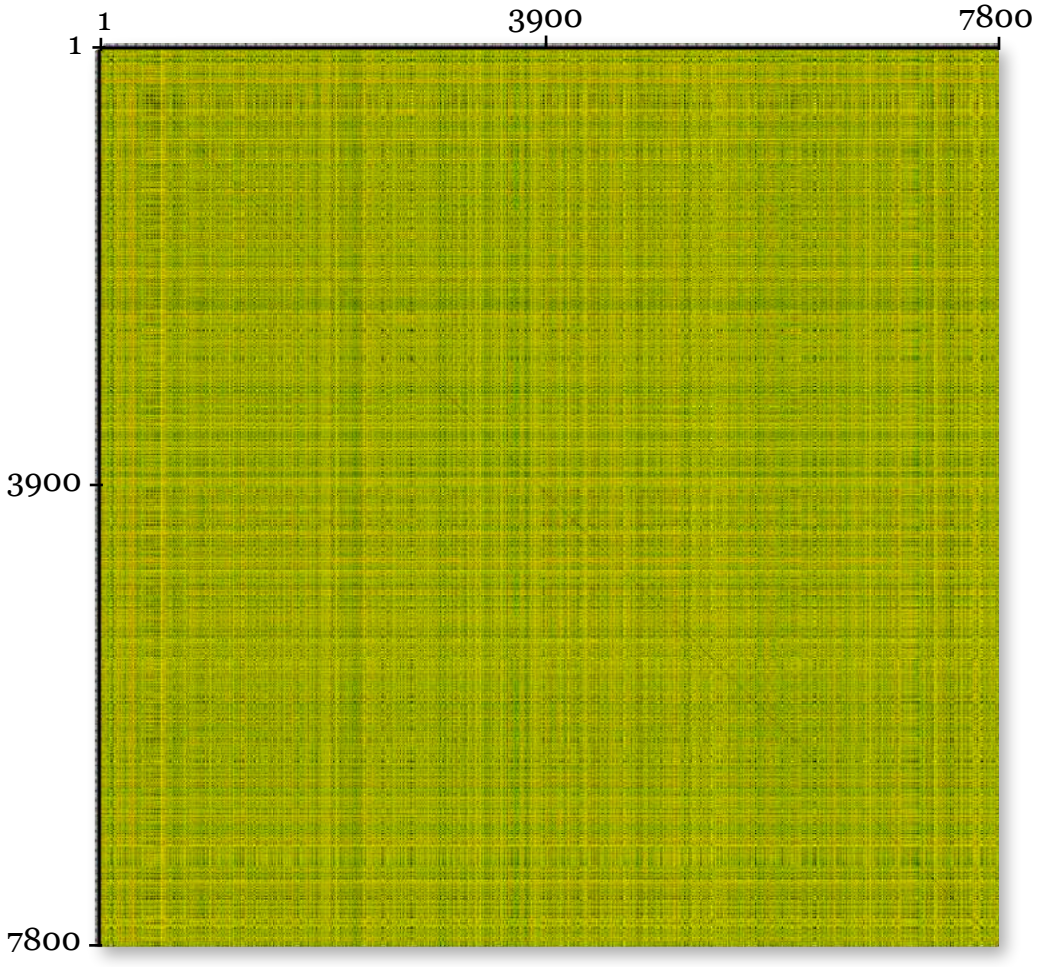


Figure 3.1: The 7800x7800 *RMSD* matrix for the C_αs of 7800 potential hits returned by the scoring algorithm. The colours range from yellow (high *RMSD* value between two peptides) to blue (low *RMSD*).

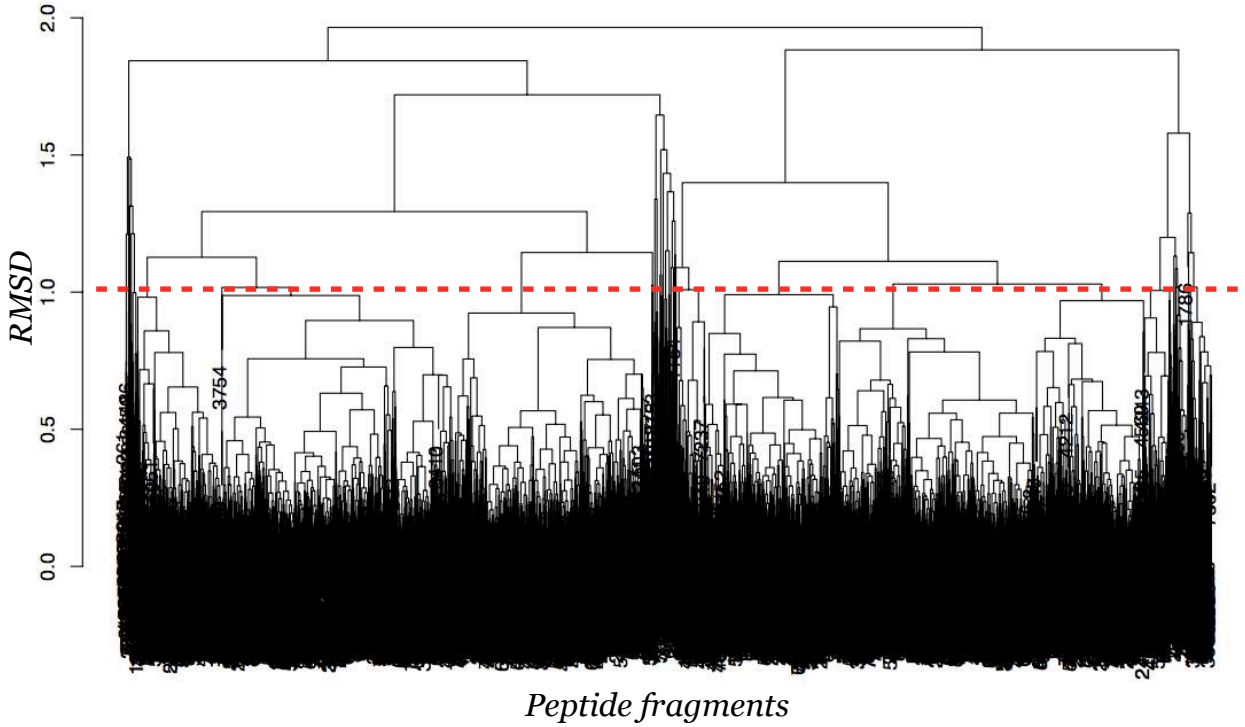


Figure 3.2: The hierarchical dendrogram of *RMSD* as created by the *R* clustering script (*UPGMA* algorithm). The red dotted line indicates the *RMSD* cut-off we set for the clustering (40 clusters at *1.5Å RMSD* cut-off).

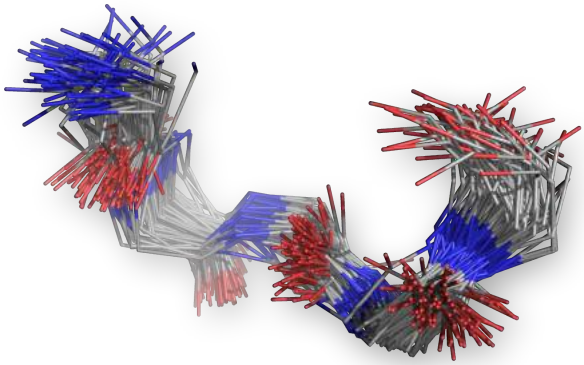
Cluster	Number of structures	% of all hits
1	1675	21,47%
2	137	1,75%
3	961	12,32%
4	606	7,76%
5	776	9,94%
6	1411	18,08%
7	1417	18,16%
8	38	0,48%
9	75	0,96%
10	13	0,17%
11	23	0,29%
12	3	0,04%
13	151	1,94%
14	17	0,22%
15	12	0,15%
16	6	0,08%
17	177	2,27%
18	18	0,23%
19	2	0,03%
20	30	0,38%

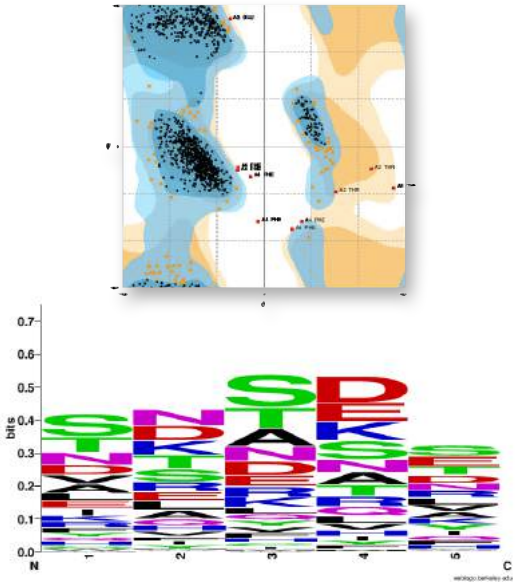
Cluster	Number of structures	% of all hits
21	12	0,15%
22	79	1,01%
23	15	0,19%
24	21	0,27%
25	13	0,17%
26	34	0,43%
27	6	0,08%
28	12	0,15%
29	15	0,19%
30	9	0,12%
31	6	0,08%
32	2	0,03%
33	1	0,01%
34	12	0,15%
35	3	0,04%
36	6	0,08%
37	1	0,01%
38	3	0,04%
39	2	0,03%

Table 3.1: The population of the clusters along with the percentage of clustered structures in the dataset of total 7800 structures. The most populated clusters (≥0,9%) are highlighted in yellow, and will be illustrated in the following pages.

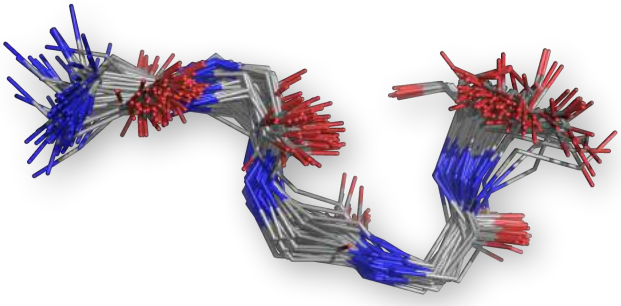
The following images show the 11 most populated clusters (highlighted in yellow in **Table 3.1**) in descending order. 100 backbone structures of each cluster (except 22 and 9) are shown as sticks (blue = N atom, red = O atom, grey = C atom), along with their *Ramachandran* plots and the *sequence logos*. The 3D models were created in *PyMol*[26] and the *Ramachandran* plots were constructed using the online tool *Rampage*[32]. The sequence *logos* were created in *Weblogo*[30].

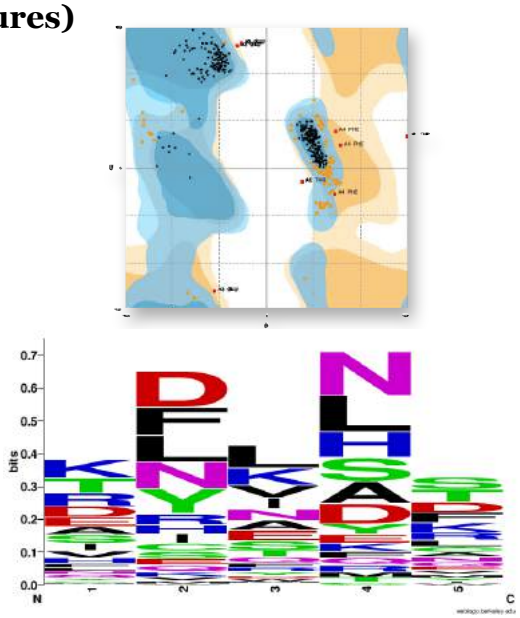
Cluster 1 (1675 structures)



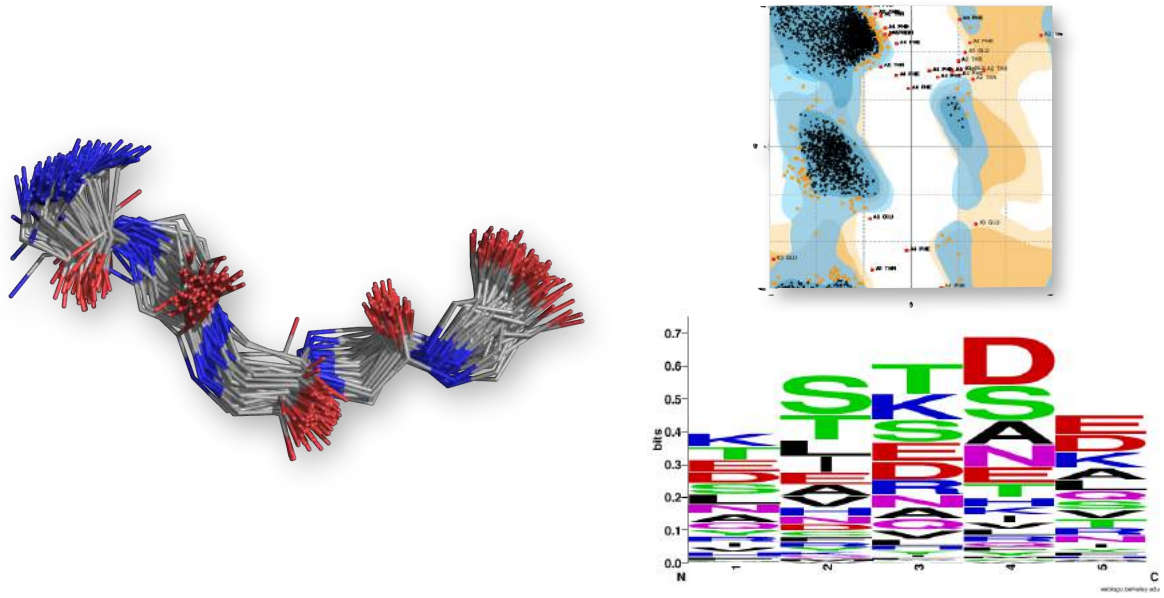


Cluster 2 (137 structures)

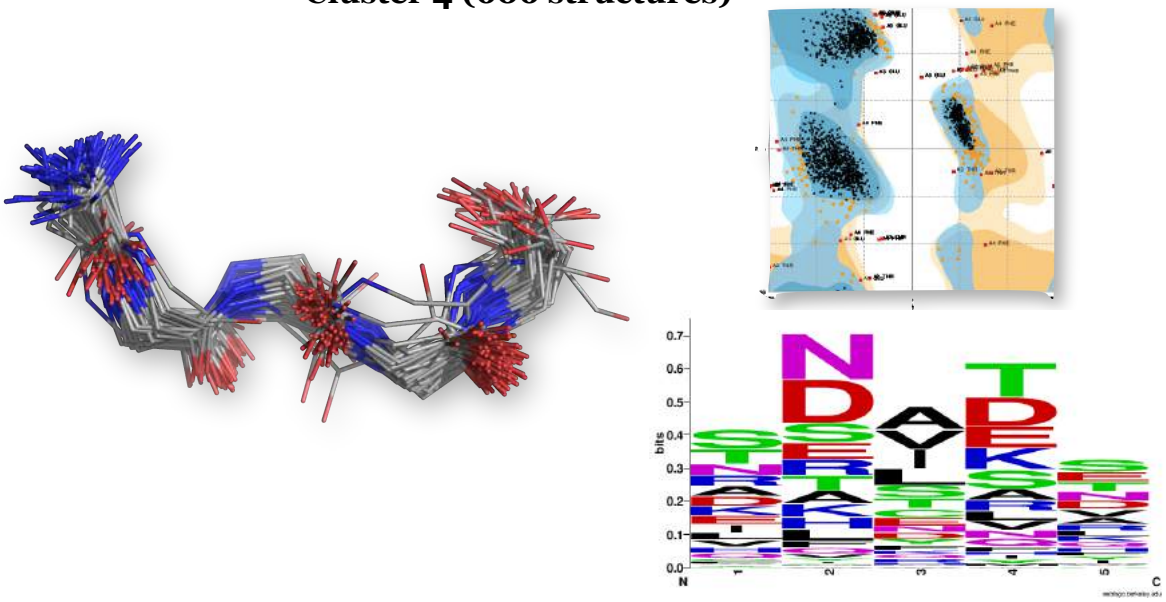




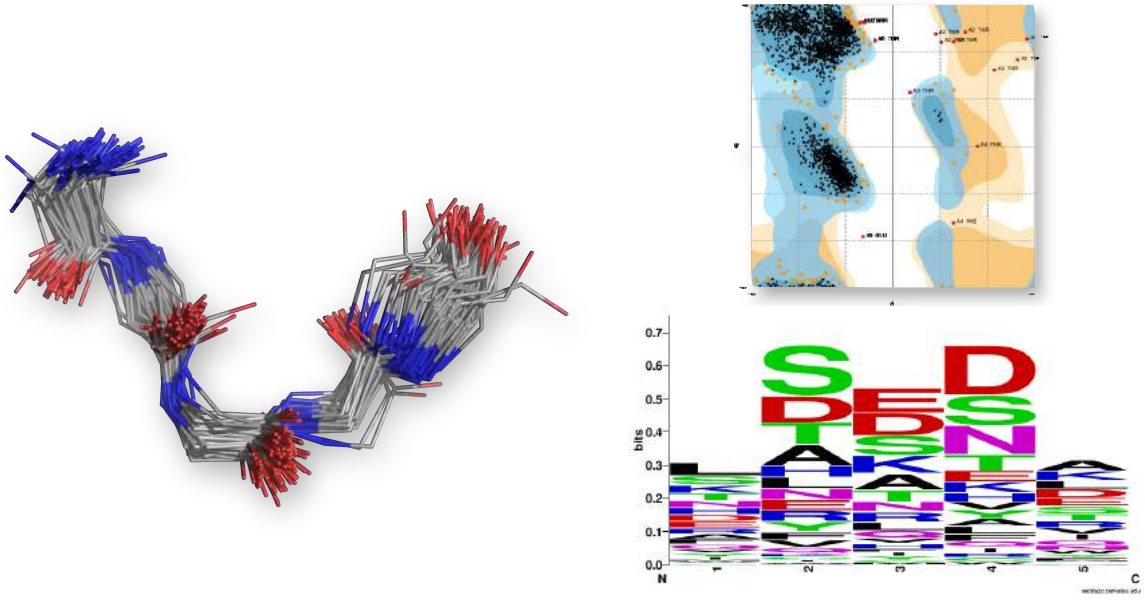
Cluster 3 (961 structures)



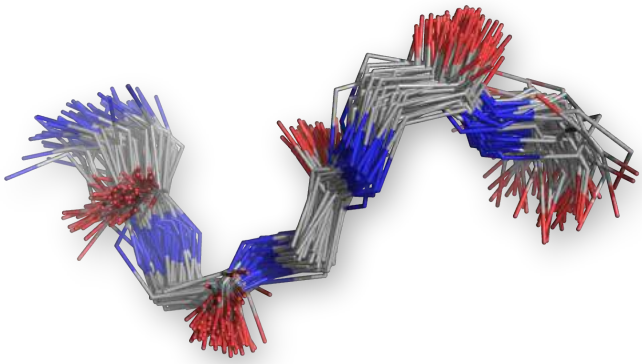
Cluster 4 (606 structures)

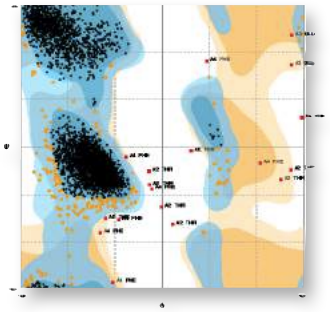


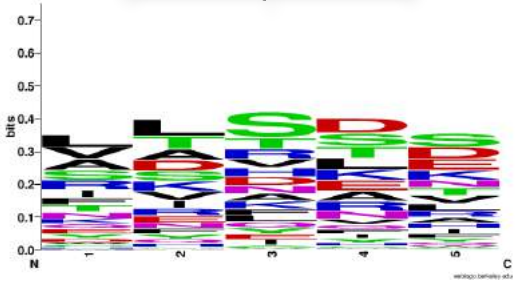
Cluster 5 (776 structures)



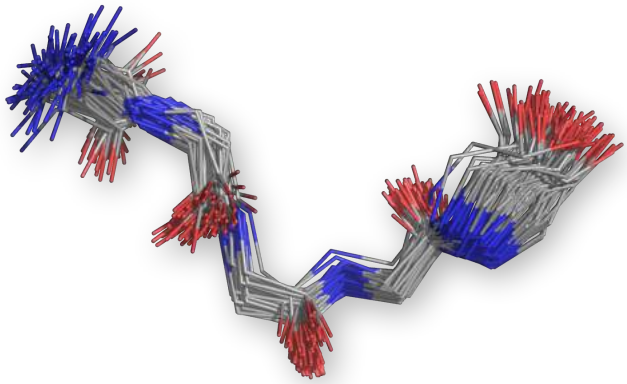
Cluster 6 (1411 structures)

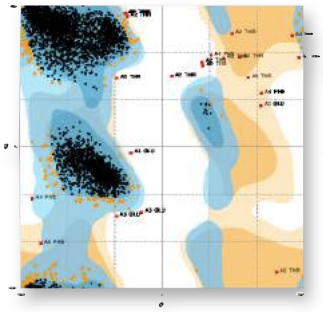




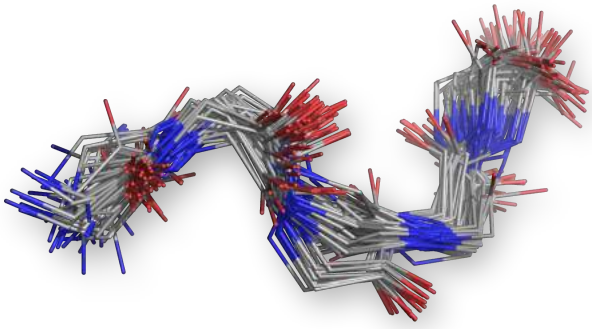


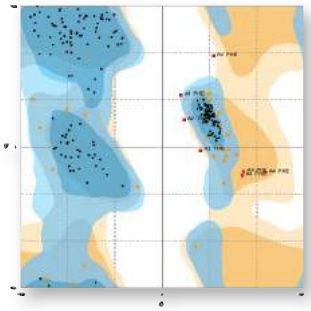
Cluster 7 (1417 structures)

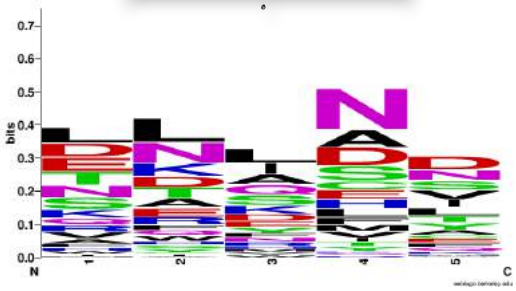




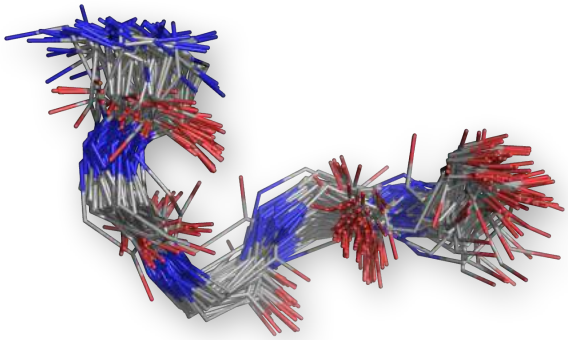
Cluster 9 (75 structures)

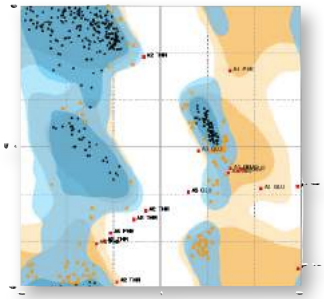


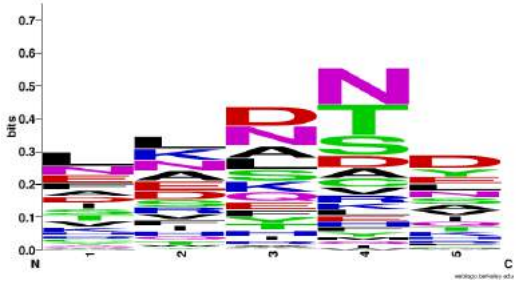




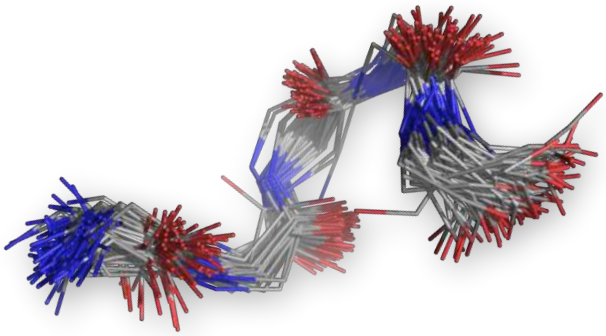
Cluster 13 (151 structures)

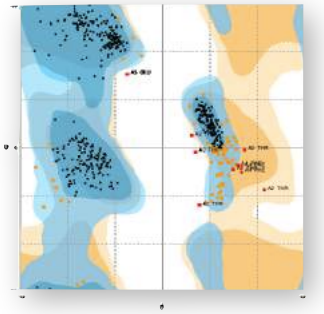


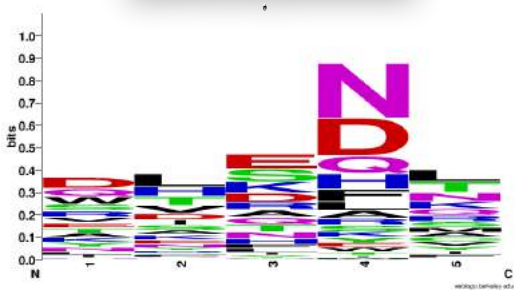




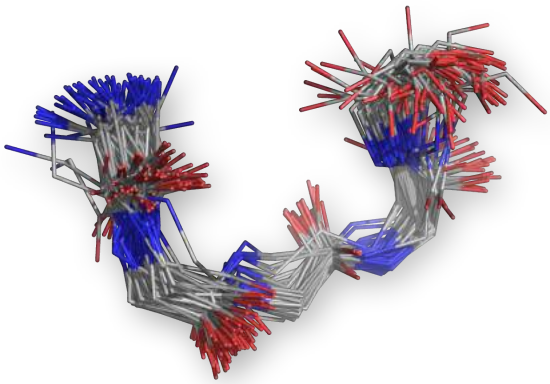
Cluster 17 (177 structures)

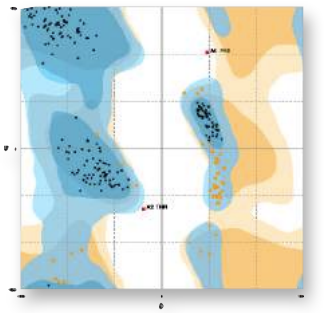


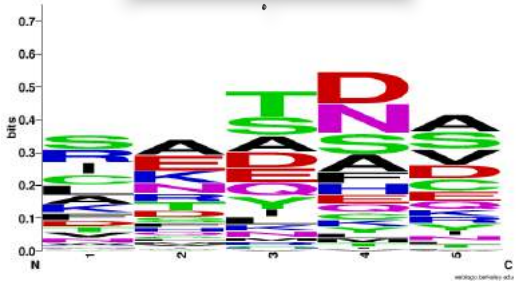




Cluster 22 (79 structures)







Section 4
Conclusions and Discussion

To conclude, we should review our initial hypothesis in contrast with the insights given by the results. Our goal was to search for a 5-residue-long, periodical motif in the known protein structures. We performed a series of *in silico* studies to scan a large sample of *X-ray diffraction*-solved protein molecules. The motif we searched for, is characterised by alternating *φ,ψ*-pairs, between two distinct ranges. We did not specify two strict *φ,ψ* ranges, but we let the clustering algorithm group the high-scored structures according to their geometric similarity in the 3-dimensional cartesian space. Therefore, the study was not sequence-specific, but secondary structure-specific.

These early results presented in the previous section, show the occurrence of recurrent two-residue periodical patterns in peptide fragments of five residues. Specifically, two dominant motifs are the most abundant (described as seen on a Ramachandran plot):

1. Transitions between the α-helix region and β-sheet region (**Figure 4.1**)

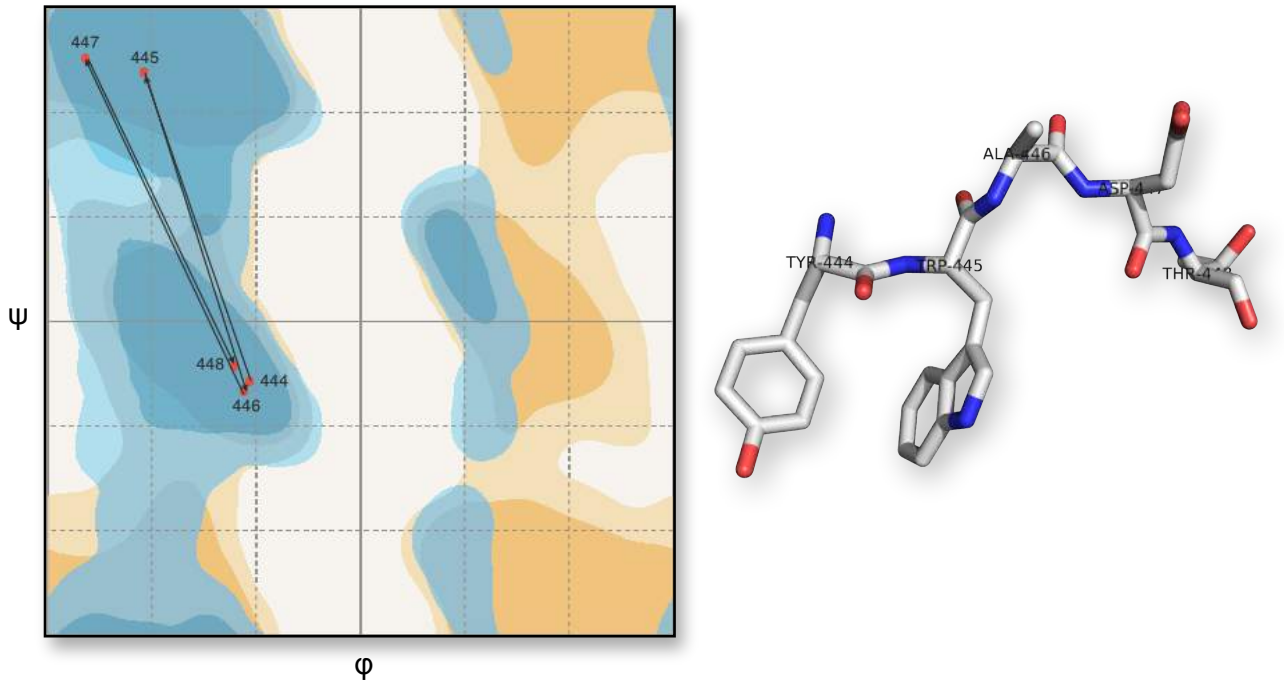


Figure 4.1: A representative example of a pattern of transitions between the *α*-helix and *β*-sheet regions. The image shows the Ramachandran plot and 3D structure of the residues Y444-447 of the molecule with *PDB ID: 2ZF5*. Two residues, one residue on each terminus of the fragment, were added in order to plot the 5 central residues. The fragment belongs to cluster 7. The Ramachandran plot was created in *Rampage* and the 3D model in *PyMol*.

2. Transitions between the α_L-helix region and the β-sheet region (**Figure 4.2**)

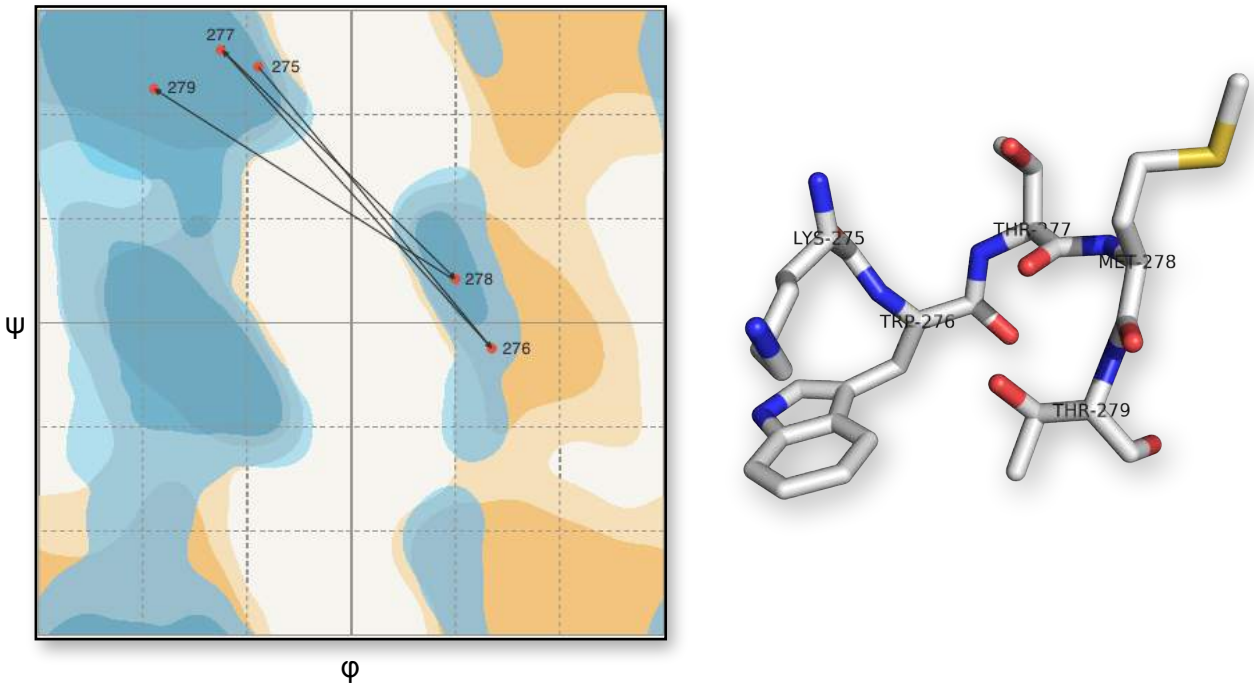


Figure 4.2: A representative example of a pattern of transitions between the *β*-sheet and α_L-helix regions. The image shows the Ramachandran plot and 3D structure of the residues A275-279 of the molecule with *PDB ID: 2DoV*. Two residues, one residue on each terminus of the fragment, were added in order to plot the 5 central residues. The fragment belongs to cluster 7. The Ramachandran plot was created in *Rampage* and the 3D model in *PyMol*.

Although a possible α_L-helix - *α*-helix pattern may occur, this case has not been evaluated by our current studies. An clue that might support the existence of such pattern, is that residues of some clusters (e.g. 4, 17, 22) populate more than two Ramachandran regions. Therefore, the pattern is likely to exist, but to be not clearly distinguishable in the graphical representations. For the reason that the current studies do not prove the existence of this pattern, right now we consider this as noise, but our future intentions include the elimination of it. If we cluster the structures with a lower *RMSD* threshold, or use the backbone atoms instead of the C_αs, we will be able to group peptides of closer structure similarity, and reduce the noise in the Ramachandran plots. Another way to do this, is to modify the scoring algorithm so it is able to set limits to the *φ,ψ* ranges, and search for patterns between two specific Ramachandran regions (e.g. a program that searches only for an α_L-*α* transition pattern).

As regards the sequence-specificity in the peptides found, we can study the *sequence logos* of the aligned structures in each cluster. As mentioned before, the conservation in a particular position is measured in *bits*, and the relative height of the letters in this position indicates the frequency of the corresponding residues. The standard scaling of *bits* (for proteins) is 0-4, however, the logos we made have a much narrower

range (0-0.7 and 0-1) . **Figure 4.3** shows the sequence logo of the first cluster, but with the normal scaling:

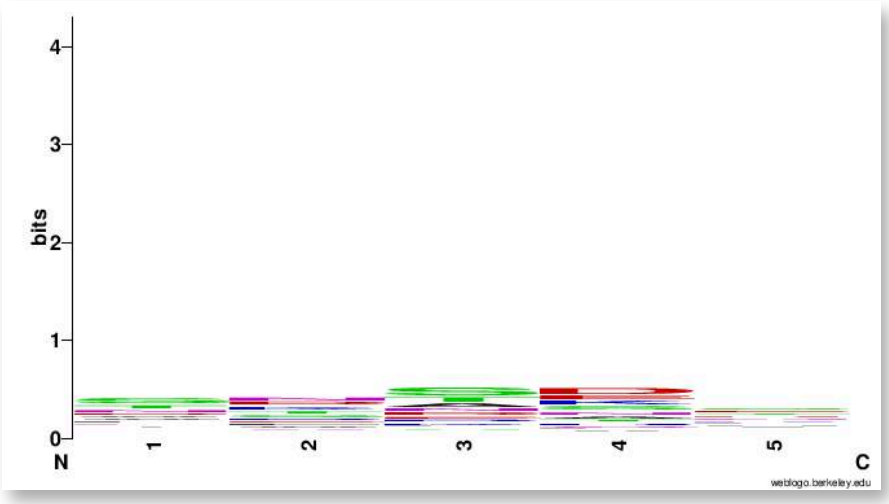


Figure 4.3: The sequence logo of Cluster 1, with the normal scaling of the *bits* (0-4). The logos of the rest of the clusters have a similar form.

We can notice that the height of the letter stacks in all the positions is low, when using normal *bits* scaling. This indicates that the residue conservation is rather insignificant, and none of the clusters seem to have a consensus sequence.

By studying the relative height of the letters in the logos, we can assess the preference of some particular residues in the peptide fragments. Some residues such as serine (S), threonine (T), glutamic acid (E), asparagine (N) and aspartic acid (D) seem to be preferred in the sequences, especially in the three central positions. This is expected, as these residues are common in loops. Nevertheless, we cannot assume that there is a strict residue preference, as there is a variety of different residues in all the five positions. The conclusion of the above observations is that the peptides which follow the 2-residue periodical pattern, do not seem to have a particular sequence specificity, although they are highly similar in the level of secondary structure. The question that is raised considering this statement, is whether this similarity of structure is translated into a specific functional role. To answer this, further research is needed, by studying the gene topology of the protein molecules that contain these peptide fragments, something that is indeed included in our future work.

These first steps we made in assessing the existence of some standard conformations in random coils, relying on the knowledge on $(\varphi,\psi)_2$-motifs, can help us in future studies. Our plans (besides the improvement of the algorithm) are the characterisation of the 2-residue periodical patterns found, in terms of sequence and function. There are some clues that support the hypothesis that these patterns might play a functional role. For instance, some observations we made, show potential conservation in some structures in Cluster 6. If we add five extra residues on each terminus of the fragments, and then superimpose the five central residues, we can distinguish a rather

interesting conformation, recurring in the specific cluster: the five central residues that follow the α_L-β transition pattern take an S shape, and the five C-terminal residues form an α-helix. Also, the five N-terminal residues seem to take a random coil conformation. **Figure 4.4** shows five members of cluster 6 that seem to follow the above mentioned pattern:

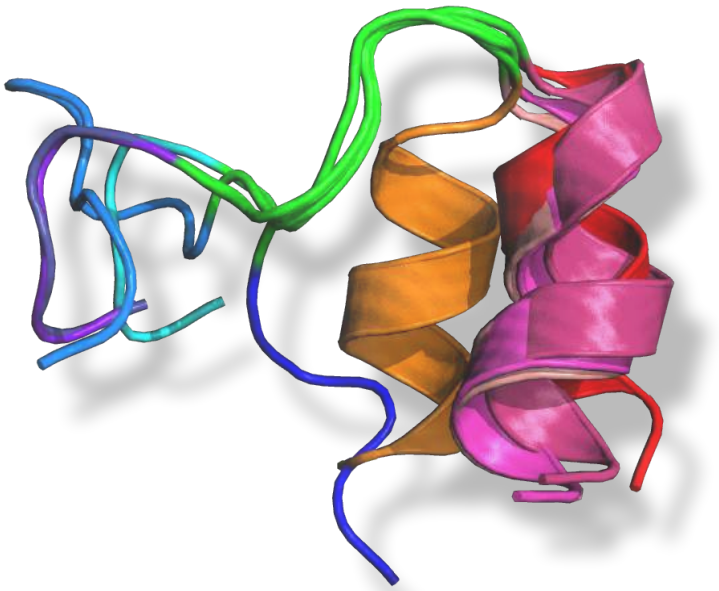


Figure 4.4: Cartoon visualisation of five selected members of cluster 6, in superposition (5-central residues shown in green colour). N-terminus added residues are shown in shades of blue and C-terminus added residues are shown in shades of red. The image was created in *PyMol,* and was made to show a potential structural conservation and functionality of the 2-residue periodical peptide fragment.

It must be indicated that the above hypothesis is not a product of systematic research, but an early observation, carried on loosely. Nevertheless, transferring this hypothesis to a new project, on strictly characterising our results, can give us new leads on the quest for uncommon secondary structure motifs.

"Science never solves a problem without creating ten more."

-George Bernard Shaw

References

1. Kendrew, J.C., et al., A three-dimensional model of the myoglobin molecule obtained by x-ray analysis. Nature, 1958. **181**(4610): p. 662-6.
2. Lehninger, A.L., D.L. Nelson, and M.M. Cox, *Lehninger principles of biochemistry*. 2013, W.H. Freeman: New York. p. 151-3.
3. Kuriyan, J., B. Konforti, and D. Wemmer, The molecules of life : physical and chemical principles. xxii, 1008 pages.
4. Pauling, L., R.B. Corey, and H.R. Branson, The structure of proteins; two hydrogen-bonded helical configurations of the polypeptide chain. Proc Natl Acad Sci U S A, 1951. **37**(4): p. 205-11.
5. Pauling, L. and R.B. Corey, The pleated sheet, a new layer configuration of polypeptide chains. Proc Natl Acad Sci U S A, 1951. **37**(5): p. 251-6.
6. Brändén, C.-I. and J. Tooze, *Introduction to protein structure*. 1999, Garland Pub.: New York. p. 15-7.
7. Toniolo, C. and E. Benedetti, *The polypeptide 310-helix*. Trends Biochem Sci, 1991. **16**(9): p. 350-3.
8. Low, B.W. and R.B. Baybutt, The pi-helix -a hydrogen bonded configuration of polypeptide chains. J Am Chem Soc, 1952. **74**(22): p. 5806-5807.
9. Ramachandran, G.N., C. Ramakrishnan, and V. Sasisekharan, *Stereochemistry of polypeptide chain configurations*. J Mol Biol, 1963. **7**: p. 95-9.
10. Ramakrishnan, C. and G.N. Ramachandran, Stereochemical criteria for polypeptide and protein chain conformations. II. Allowed conformations for a pair of peptide units. Biophys J, 1965. **5**(6): p. 909-33.
11. Hollingsworth, S.A. and P.A. Karplus, A fresh look at the Ramachandran plot and the occurrence of standard structures in proteins. Biomol Concepts, 2010. **1**(3-4): p. 271-283.
12. Arnott, S. and S.D. Dover, The structure of poly-L-proline II. Acta Crystallogr B, 1968. **24**(4): p. 599-601.
13. Schulz, G.E. and R.H. Schirmer, *Principles of protein structure*. Springer advanced texts in chemistry. 1979, New York: Springer-Verlag. x, 314 p.
14. Hollingsworth, S.A., D.S. Berkholz, and P.A. Karplus, *On the occurrence of linear groups in proteins*. Protein Sci, 2009. **18**(6): p. 1321-5.
15. Venkatachalam, C.M., Stereochemical criteria for polypeptides and proteins. V. Conformation of a system of three linked peptide units. Biopolymers, 1968. **6**(10): p. 1425-36.
16. Hollingsworth, S.A., et al., (phi,psi)(2) motifs: a purely conformation-based fine-grained enumeration of protein parts at the two-residue level. J Mol Biol, 2012. **416**(1): p. 78-93.

17. Berman, H.M., et al., *The Protein Data Bank*. Nucleic Acids Res, 2000. **28**(1): p. 235-42.
18. Kernighan, B.W. and D.M. Ritchie, *The C programming language*. 2nd ed. 1988, Englewood Cliffs, N.J.: Prentice Hall. xii, 272 p.
19. Stajich, J.E., et al., The Bioperl toolkit: Perl modules for the life sciences. Genome Res, 2002. **12**(10): p. 1611-8.
20. *Introduction to R*. [cited 2017 March]; Available from: https://www.r-project.org/about.html.
21. Wang, G. and R.L. Dunbrack, Jr., PISCES: a protein sequence culling server. Bioinformatics, 2003. **19**(12): p. 1589-91.
22. Laskowski, R.A., et al., PROCHECK - a program to check the stereochemical quality of protein structures. J App Cryst, 1993. **26**: p. 283-291.
23. Weisstein, E.W. *"Erf."*. MathWorld--A Wolfram Web Resource.; Available from: http://mathworld.wolfram.com/Erf.html.
24. Glykos, N.M., Carma: a molecular dynamics analysis program. J Comput Chem, 2006. **27**(14): p. 1765-8.
25. Koukos, P.I. and N.M. Glykos, Grcarma: A fully automated task-oriented interface for the analysis of molecular dynamics trajectories. J Comput Chem, 2013. **34**(26): p. 2310-2.
26. Schrodinger, LLC, The PyMOL Molecular Graphics System, Version 1.8. 2015.
27. Humphrey, W., A. Dalke, and K. Schulten, *VMD: visual molecular dynamics*. J Mol Graph, 1996. **14**(1): p. 33-8, 27-8.
28. Kufareva, I. and R. Abagyan, Methods of protein structure comparison. Methods Mol Biol, 2012. **857**: p. 231-57.
29. Schneider, T.D. and R.M. Stephens, Sequence logos: a new way to display consensus sequences. Nucleic Acids Res, 1990. **18**(20): p. 6097-100.
30. Crooks, G.E., et al., WebLogo: a sequence logo generator. Genome Res, 2004. **14**(6): p. 1188-90.
31. Krissinel, E., On the relationship between sequence and structure similarities in proteomics. Bioinformatics, 2007. **23**(6): p. 717-23.
32. Lovell, S.C., et al., Structure validation by Calpha geometry: phi,psi and Cbeta deviation. Proteins, 2003. **50**(3): p. 437-50.

Appendix

Source code

Script 1: list_lowercase.pl

```
#!/usr/bin/perl

use warnings;

@ARGV == 1 || die "Usage: p1_scr2.pl [list_path]";

open (INFILE, "$ARGV[0]") || die "Cannot open file for input: $!\n";
open (OUTFILE, ">$ARGV[0].lst") || die "Cannot open file for output:
$!\n";

while ($input = <INFILE>){

        if($input =~ m/^IDs*/){
                next;
        }

        else{
                $pdb_code = lc substr("$input", 0, 4);
                print OUTFILE "$pdb_code\n";
        }
}

close INFILE;
close OUTFILE;

exit(0);
```

Script 2: pdb_ftp.pl

```
#!/usr/bin/perl
use warnings;

use Net::FTP;

@ARGV == 1 || die "Usage: p1_scr3.pl [list_path]";

$ftp = Net::FTP->new("ftp.wwpdb.org", Debug => 0) || die "Cannot
connect to ftp.wwpdb.org: $!\n";
print "Connected!\n";

```

```
11 $ftp->login("john_ree", "randompasswd") || die "Cannot login to
ftp.wwpdb.org: $!\n";
12 print "You're in mah dawg!\n";
13
14 my $fetching_directory = "/pub/pdb/data/structures/all/pdb/";
15
16 $ftp->cwd($fetching_directory);
17
18 open (INFILE, "$ARGV[0]") || die "Cannot open file for input: $!\n";
19
20 my $input;
21 my $file_to_transfer;
22
23 while ($input = <INFILE>){
24
25         chop $input;
26         $file_to_transfer = "pdb$input.ent.gz";
27         print "Getting file: $file_to_transfer\n";
28         $ftp->get($file_to_transfer) || warn "Couldn't get
$file_to_transfer, skipped: $!\n";
29 }
30
31 $ftp->quit;
32 close INFILE;
33
34 exit(0);
```

Script 3: `angles.pl`

```
 1 #!/usr/bin/perl
 2
 3 use warnings;
 4
 5                 ## INPUT FILE SANITY CHECK ##
 6
 7 if (@ARGV == 2){
 8
 9         $pdbfile = $ARGV[0];
10         if ($pdbfile =~ m/(\d...)([.]...)/){
11                 $pdbcode = "$1";
12                 print "Now processing: $pdbcode\n";
13         }
14         else{
15                 die "Wrong input file: Must be a .pdb or .ent\n";
16         }
17
18         $resolution = $ARGV[1];
19 }
20
```

```
21 elsif (@ARGV == 3){
22
23         $pdbfile = $ARGV[0];
24         if ($pdbfile =~ m/(\d...)([.]...)/){
25                 $pdbcode = "$1";
26                 print "Now processing: $pdbcode\n";
27         }
28         else{
29                 die "Wrong input file: Must be a .pdb or .ent\n";
30         }
31
32         $chain = $ARGV[1];
33         $resolution = $ARGV[2];
34 }
35
36 else{
37         die "Usage: prog.pl [pdb_file_path] [chain (leave blank for
all chains] [resolution]\n";
38 }
39                 ## RUN PROCHECK TO PRODUCE PHI/PSI DATA FILE ##
40
41 if (@ARGV == 2){
42
43         system('bash', '-i', '-c', "procheck $pdbfile $resolution >/
dev/null 2>&1");
44 }
45
46 elsif (@ARGV == 3){
47
48         system('bash', '-i', '-c', "procheck $pdbfile $chain
$resolution >/dev/null 2>&1");
49 }
50
51                 ## DELETE JUNK FILES ##
52
53 #`mv $pdbcode.pdb a$pdbcode.pdb`;
54 `mv *.rin temp1.dat`;
55 `rm *$pdbcode*`;
56 #`mv a$pdbcode.pdb $pdbcode.pdb`;
57 `rm fort.27`;
58 `rm *.log`;
59 `rm procheck.prm`;
60
61                 ## MODIFY PHI/PSI ANGLE FILE ##
62
63 open (INFILE1, "temp1.dat");
64 open (OUTFILE1, ">$pdbcode.ang");
65
66 while ($line = <INFILE1>){
67
68         if($line =~ m/(\d+\w\w\w)/){
```

```
69                  $residue = $&;
70                  $residue =~ s/ //;
71          }
72          else{
73                  next;
74          }
75          $resid = substr($line, 8, 5);
76          $phi = substr($line, 15, 7);
77          $phi =~ s/ //;
78          $psi = substr($line, 22, 7);
79          $psi =~ s/ //;
80
81          printf OUTFILE1 ("%s\t%7s\t%7s\t%7s\t%7s\n", $pdbcode,
$residue, $resid, $phi, $psi);
82 }
83
84 close INFILE1;
85 close OUTFILE1;
86
87 `rm temp1.dat`;
88
89 exit(0);
```

Script 4: run_angles.pl

```
1 #!/usr/bin/perl
2 use warnings;
3
4 @files = <~/Desktop/test_pdb/*.ent>;
5 foreach $file (@files) {
6
7         system("~/Desktop/ang_test/angles.pl $file 3.0”);
8 }
9 exit(0);
```

Script 5: unknown_omit.pl

```
 1 #!/usr/bin/perl -w
 2
 3 open (INFILE, "$ARGV[0]");
 4 open (OUTFILE, ">no_unknown.ang");
 5
 6 while ($line = <INFILE>){
 7
 8                  if ($line !~ m/(\s\d+ALA|ARG|ASN|ASP|ASX|CYS|GLN|GLU|
GLY|GLX|HIS|ILE|LEU|LYS|MET|PHE|SER|THR|TRP|TYR|VAL|PRO)(.........)
(.......)/){
 9                          $substr1 = substr($line, 0, 21);
```

```
10                          $substr2 = " 999.90";
11                          $substr3 = substr($line, 28);
12                          $line = $substr1.$substr2.$substr3;
13                          print OUTFILE  "$line";
14                  }
15                  else{
16                          print OUTFILE "$line";
17                  }
18
19  }
```

Script 6: gp_omit.pl

```
 1  #!/usr/bin/perl -w
 2
 3  open (INFILE, "$ARGV[0]");
 4  open (OUTFILE, ">no_gp.ang");
 5
 6  while ($line = <INFILE>){
 7
 8                  if ($line =~ m/(\s\d+GLY|PRO)(.........)(.......)/){
 9                          $line =~ s/$3/ 999.90/;
10                          print OUTFILE "$line";
11                  }
12                  else{
13                          print OUTFILE "$line";
14                  }
15
16  }
```

Script 7: input_correction.c

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #include <string.h>
 4  #include <stdlib.h>
 5
 6  int main(){
 7
 8          char pdbid1[5], pdbid2[5];
 9          char resnum1[20], resnum2[20];
10          char chain1, chain2;
11          int resid1, resid2;
12          float phi1, psi1, phi2, psi2;
13
14          scanf("%s %s %c %d %f %f", pdbid1, resnum1, &chain1, &resid1,
&phi1, &psi1);
15          printf("%s %11s %c %4d %8.2f %8.2f\n", pdbid1, resnum1,
chain1, resid1, phi1, psi1);
```

```
16          while(scanf("%s %s %c %d %f %f", pdbid2, resnum2, &chain2,
&resid2, &phi2, &psi2) == 6){
17                  if(strcmp(pdbid1, pdbid2) == 1 && resid2 != resid1+1)
{
18                          printf("%s %11s %c %4d 999.90 %8.2f\n",
pdbid1, resnum1, chain1, resid1, psi1);
19                          printf("%s %13s %c %5d %10.2f 999.90\n",
pdbid2, resnum2, chain2, resid2, phi2);
20                  }
21                  else{
22                          printf("%s %11s %c %4d %8.2f %8.2f\n",
pdbid2, resnum2, chain2, resid2, phi2, psi2);
23                  }
24                  strcpy(pdbid1, pdbid2);
25                  strcpy(resnum1, resnum2);
26                  chain1 = chain2;
27                  resid1 = resid2;
28                  phi1 = phi2;
29                  psi1 = psi2;
30          }
31  }
```

Script 8: `gauss_fit.R`

```
 1
 2 # Variable and function declarations
 3
 4
 5          input.data <- read.table("~/Dropbox/Lab/proj1/R/
distances1-3_noGP.histogram", header = TRUE, sep = "", dec = ".")
 6          x <- input.data$x
 7          y <- input.data$y
 8
 9
10          fit.data <- data.frame(x,y)
11
12          gaussian.formula <- "y ~
13                                  k * exp(-(x-0)^2/(2*s1^2)) +
14                                  l * exp(-(x-m2)^2/(2*s2^2)) +
15                                  m * exp(-(x-m3)^2/(2*s3^2))"
16
17
18 # Fit
19
20          fit <- nls(gaussian.formula, data = fit.data, start =
list(k=100000, l=30000, m=15000, m1=10, m2=50, m3=125, s1=10, s2=10,
s3=20), trace = FALSE)
21
22          pdf(file="fit_plot.pdf")
23          plot(x, y, type="l", col="green")
24          lines(x, predict(fit), type="l", col="red")
25          summary(fit)
```

Script 9: pdb_extractor.pl

```
#!/usr/bin/perl

use strict;
use Net::FTP;

@ARGV == 1 || @ARGV == 2 || @ARGV == 3 || @ARGV == 4 || die
"\nUsage: pdb_extractor.pl [input_file] [options]\nOptions:\n-d : delete
initial PDB files after the process\n-c : concatenate hits of the same
molecule into one PDB file\-h : Skip homopolymer duplicates\n\n";

my $i;
my $delete = "NO";
my $concatenate = "NO";
my $no_homopolymer = "NO";
my $file_exists;
my $input_file = $ARGV[0];
my $ftp;
my $fetching_directory;
my $line_1;
my $line_2;
my $pdb_id;
my $chain;
my $res1_id;
my $res5_id;
my $nameres1;
my $nameres5;
my $num_of_entries = 0;
my $pdb_entry;
my $dh;
my $indir_name = "pdb_entries";
my $outdir_name = "out_pdb";
my $outfile_name;

my @dir_contents;

                                        ##Arguments##
        for($i=1; $i<=3; $i++){

                if($ARGV[$i] =~ m/-.*h.*/){

                        if($ARGV[$i] =~ m/c/){

                                die "That's insane. Goodbye.\n";
                        }
                        else{
                                $no_homopolymer = "YES";
                        }
                }
        }
        if ($ARGV[1] eq "-d" || $ARGV[2] eq "-d"){

                $delete = "YES";
        }
```

```
 51          elsif ($ARGV[1] eq "-c" || $ARGV[2] eq "-c"){
 52
 53                  $concatenate = "YES";
 54          }
 55          elsif ($ARGV[1] eq "-dc" || $ARGV[1] eq "-cd"){
 56
 57                  $concatenate = "YES";
 58                  $delete = "YES";
 59          }
 60
 61          elsif(@ARGV == 1){
 62          }
 63          else{
 64                  die "\nUsage: pdb_extractor.pl [input_file]
[options]\nOptions:\n-d : delete initial PDB files after the process\n-c
: concatenate hits of the same molecule into one PDB file\n\n";
 65          }
 66
 67                                  ##Establish ftp connection##
 68
 69          $ftp = Net::FTP->new("ftp.wwpdb.org", Debug => 0) || die
"Cannot login to ftp server 'ftp.wwpdb.org': $!\n";
 70          print "\nConnection to PDB server succesful!\n";
 71
 72          $ftp->login("john_ree", "randompasswd") || die "Cannot login
to server: $!\n";
 73          print "You're in dawg!\n";
 74
 75          $fetching_directory = "/pub/pdb/data/structures/all/pdb/";
 76
 77          $ftp->cwd($fetching_directory);
 78
 79                                  ##Open input file and check sanity##
 80
 81          open (INFILE_1, "$input_file");
 82
 83          while($line_1 = <INFILE_1>){
 84
 85                  if($line_1 !~ m/(\S+)(\s)(\w\w\w\w)(\s+)(\w+)(\s+)
(\w)(\s+)([0-9]+)(\s+)(\w+)(\s+)(\w)(\s+)([0-9]+)/){
 86
 87                          die "Input file not valid: Must contain 8
columns. Cheers.\n";
 88                  }
 89
 90                  else{
 91                          $num_of_entries++;
 92                  }
 93          }
 94          print "$num_of_entries hits will be processed.\n";
 95          seek (INFILE_1, 0, 0);
 96
 97                                  ##Download and process every file
listed##
 98
 99          mkdir("./$indir_name");
```

```
100         mkdir("./$outdir_name");
101
102 MAIN_LOOP: while($line_1 = <INFILE_1>){
103
104                 if($line_1 =~ m/(-*\S+)(\s)(\w\w\w\w)(\s+)(\w+)(\s+)
(\w)(\s+)(-*[0-9]+)(\s+)(\w+)(\s+)(\w)(\s+)(-*[0-9]+)/){
105
106                         if($1 =~ m/inf|nan/){
107
108                                 next;
109                         }
110
111                         else{
112
113                                 $file_exists = "NO";
114                                 $pdb_id = lc($3);
115
116                                 opendir($dh, $indir_name);
#check if pdb file exists#
117                                 @dir_contents = readdir($dh);
118                                 foreach $pdb_entry (@dir_contents){
119
120                                         if ($pdb_entry eq
"pdb$pdb_id.ent"){
121
122                                                 $file_exists =
"YES";
123                                                 last;
124                                         }
125                                 }
126
127                                 $chain = $7;
128                                 $res1_id = $9;
129                                 $res5_id = $15;
130
131                                 if ($file_exists eq "NO"){
132
133                                         print "Downloading and
processing file: pdb$pdb_id.ent.gz\n";
134
135                                         $ftp->get
("pdb$pdb_id.ent.gz") || warn "Couldn't get pdb$pdb_id.ent.gr, skipped:
$!\n";
136                                         system ("gunzip
pdb$pdb_id.ent.gz");
137                                         system ("mv
pdb$pdb_id.ent ./$indir_name");
138                                         $nameres1 = $res1_id;
139                                         $nameres5 = $res5_id;
140                                         $outfile_name =
"$pdb_id\_$chain$nameres1-$chain$nameres5.pdb";
141                                         open (OUTFILE, ">./
$outdir_name/$outfile_name");
142                                 }
143
```

```
144                                     elsif ($file_exists eq "YES" &&
$concatenate eq "YES"){         #to concatenate in single file#
145
146                                             opendir($dh, $outdir_name);
147                                             @dir_contents =
readdir($dh);
148                                             foreach $pdb_entry
(@dir_contents){
149
150                                                     if ($pdb_entry =~ m/
$pdb_id/){
151
152                                                             close
OUTFILE;
153                                                             open
(OUTFILE, ">>$outdir_name/$pdb_entry");
154                                                             last;
155                                                     }
156                                             }
157                                     }
158                                     elsif ($file_exists eq "YES" &&
$concatenate eq "NO"){
159
160                                             if($no_homopolymer eq "YES")
{
161
162                                                     opendir($dh,
$outdir_name);
163                                                     @dir_contents =
readdir($dh);
164                                                     foreach $pdb_entry
(@dir_contents){
165
166                                                             if
($pdb_entry =~ m/$pdb_id.+$res1_id/){
167
$pdb_id = uc($pdb_id);
168
print "Skipped homopolymer in entry $pdb_id\n";
169                                                                     next
MAIN_LOOP;
170                                                             }
171                                                     }
172                                             }
173                                             $nameres1 = $res1_id;
174                                             $nameres5 = $res5_id;
175                                             $outfile_name =
"$pdb_id\_$chain$nameres1-$chain$nameres5.pdb";
176                                             open (OUTFILE, ">./
$outdir_name/$outfile_name");
177                                     }
178
179                                     open (INFILE_2, "./$indir_name/
pdb$pdb_id.ent");
180
181                                     while ($line_2 = <INFILE_2>){
```

```
182
183
184                                                 if( $line_2 =~ m/(^ATOM)\s+
(\w+)\s+(\w+)\s+(\w+)\s+(\w)\s+([0-9]+)/ ){
185
186                                                         if( $5 eq $chain &&
$6 >= $res1_id && $6 <= $res5_id ){
187
188                                                                 print
OUTFILE "$line_2";
189                                                         }
190                                                 }
191                                         }
192                                         close INFILE_2;
193                                         close OUTFILE;
194
195                                         if( $file_exists eq "YES" &&
$concatenate eq "YES" ){
196
197                                                 system("mv ./$outdir_name/
$pdb_id* ./$outdir_name/temp");
198                                                 system("sort -u ./
$outdir_name/temp > ./$outdir_name/$pdb_id\_all.pdb");
199                                                 system("rm ./$outdir_name/
temp");
200                                         }
201                                 }
202                         }
203                 }
204                 close INFILE_1;
205
206                 if ($delete eq "YES"){
207
208                         system("rm -rf $indir_name");
209                 }
210 exit();
```

Script 10: ca_filter.sh

```
 1 #!/bin/bash
 2
 3 for f in *.pdb
 4    do
 5          awk 'BEGIN{a=0} if($3=="CA"){a++} END{if(a!=5) print
FILENAME}' $f > to_delete.list
 6    done
 7
 8 for a in $(<to_delete.list)
 9    do
10          rm $a
11    done
```

Script 11: backbone_filter.sh

```
#!/bin/bash

for f in *.pdb
   do
         awk 'BEGIN{a=0} if($3=="CA" || $3=="C" || $3=="N" || $3=="O")
{a++} END if(a!=20){print FILENAME}' $f >> to_delete.list

   done

for a in $(<to_delete.list)
   do
         rm $a
   done
```

Script 12: pdb2psf

```
#!/usr/bin/perl -w

#
# Open input-output files
#
if ( @ARGV == 1 )
  {
    if ( $ARGV[0] =~ /(\w+)\.(p|P)(d|D)(b|B)/ )
      {
        $outname = $1 . ".psf";
        open( IN , $ARGV[0] ) or die "Can not open input file\n";
        open( OUT, ">$outname" ) or die "Can not open output file\n";
      }
    else
      {
        print "Usage: pdb2psf in.pdb out.psf\n";
        exit;
      }
  }
elsif ( @ARGV == 2 )
  {
        open( IN , $ARGV[0] ) or die "Can not open input file\n";
        open( OUT, ">$ARGV[1]" ) or die "Can not open output file\n";
  }
else
  {
    print "Usage: pdb2psf in.pdb out.psf\n";
    exit;
  }

print OUT "PSF\n\n";
print OUT "       2 !NTITLE\n";
print OUT " REMARKS This is a _pseudo_ PSF file for sole use with the
program carma.\n";
```

```
34 print OUT " REMARKS It will not work with any other PSF-reading
program.\n\n";
35
36 $nof_atoms = 0;
37 while ( $line = <IN> )
38 {
39   if ( $line =~ /^ATOM\s*(\d*)\s*(\w*)\s*(\w*).(.)\s*(\d*)/ )
40     {
41       $nof_atoms++;
42     }
43   elsif ( $line =~ /^HETATM\s*(\d*)\s*(\w*)\s*(\w*).(.)\s*(\d*)/ )
44     {
45       $nof_atoms++;
46     }
47 }
48
49 printf OUT "%8d !NATOM\n", $nof_atoms;
50
51 print "Found $nof_atoms atoms. Writing ...\n";
52
53 close( IN );
54 open( IN , $ARGV[0] );
55
56 while ( $line = <IN> )
57 {
58   if ( $line =~ /^ATOM\s*(\d*)\s*(\w*)\s*(\w*).(.)\s*(\d*)/ )
59     {
60       printf OUT "%8d %1s%5d    %-5s%-5sDUMMY  0.000000        0.0000
0\n", $1, $4, $5, $3, $2;
61     }
62   elsif ( $line =~ /^HETATM\s*(\d*)\s*(\w*)\s*(\w*).(.)\s*(\d*)/ )
63     {
64       printf OUT "%8d %1s%5d    %-5s%-5sDUMMY  0.000000        0.0000
0\n", $1, $4, $5, $3, $2;
65     }
66
67 }
68
```

Script 13: `clustering.R`

```
 1 A <- matrix(scan("all_backbone.RMSD.matrix", n=7800*7800), 7800,
7800, byrow = TRUE)
 2
 3 hc<-hclust( as.dist(A), method="average")
 4 postscript()
 5 plot(hc)
 6 dev.off()
 7 cutree(hc, h=1)
 8
 9 clusters<-cutree(hc, h=1)
10 a<-as.data.frame(clusters)
11 names(a) <- NULL
12
13 write.table(a, file = "all_clusters.list", sep = " ", quote = FALSE)
```

Script 14: lists.sh

```
#!/bin/bash

mkdir cluster_lists

for i in `seq 1 39` #change me depending on the number of clusters
do
        awk -v c="$i" '$2 == c {print $1}' all_clusters.list > cluster_lists/cluster$i.list
done
```

Script 15: reorder.sh

```
#!/bin/bash

mkdir reordered_clusters
for i in `seq 1 39` #change me depending on the number of clusters
do
        IF1=cluster_lists/cluster$i.list
        OF1=reordered_clusters/cluster$i.dcd
        carma -sort $IF1 ../all_backbone.dcd
        mv carma.reordered.dcd $OF1
done
```

Script 16: superimpose.sh

```
#!/bin/bash

for i in `seq 1 39` #change me depending on the number of clusters
do
        IF=./reordered_clusters/cluster$i.dcd
        OF=./reordered_clusters/cluster$i.fitted.dcd
        carma -v -fit -atmid ALLID $IF ../all_backbone.psf
        mv carma.fitted.dcd $OF
        rm $IF
done
```

Script 17: final_pdbs.sh

```
#!/bin/bash

mkdir final_pdbs
for i in `seq 1 39` #change me depending on the number of clusters
do
        IF_DCD=./reordered_clusters/cluster$i.fitted.dcd
        IF_PSF=./all_backbone.psf

        carma -v -pdb -atmid ALLID $IF_DCD $IF_PSF
        cat cluster*fitted*.pdb > ./final_pdbs/cluster$i.pdb
        rm cluster*fitted*.pdb
done
```

Distances histogram construction program: `histogram1-3.c`

```
/****Plot histogram.dat using NMG's 'plot' ('>$ plot -h < histogram.dat' or '>$ plot -hs < histogram.dat' to
     produce data file along with image)****/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{

        FILE *fp;
        FILE *ofp;

        char line[200];
        char pdbid[4];
        char residue[10];
        char chain;
        int resid;
        float d[9];
        float dmin;
        float phi, psi;
        float phi1, phi2, phi3;
        float psi1, psi2, psi3;
        float x0, x1;
        float y0, y1;
        int i;



                /******Argument sanity check******/

        if (argc != 2){

                printf ("Usage: histogram [angle_file_name]\n");
                exit(1);
        }

                /******Open file and check if it is correct******/

        fp = fopen (argv[1], "r");
        puts ("\nChecking file...");

        while (fgets(line, sizeof(line), fp) != NULL){

                if(sscanf(line, "%s %s %c %d %f %f", pdbid, residue, &chain, &resid, &phi, &psi) != 6){

                        puts("Error: Wrong input file format. It must contain four columns");
                        exit(1);
```

```
 49                 }
 50                 sscanf(line, "%s %s %c %d %f %f", pdbid, residue,
&chain, &resid, &phi, &psi);
 51                 if(phi == 180.00 || psi == 180.00){
 52
 53                         puts("Error: Angle value +180.00 found.
Replace with -180.00 and re-run.");
 54                         exit(1);
 55                 }
 56         }
 57
 58                 /******Calculating distances for histogram
dataset******/
 59
 60         puts ("Data is good! Passing dataset to calculate distances
for histogram..");
 61         rewind(fp);
 62
 63         ofp = fopen ("histogram1-3.dat", "w");
 64
 65         fscanf(fp, "%s %s %c %d %f %f", pdbid, residue, &chain,
&resid, &phi1, &psi1);
 66         if (phi1 > 180.0 || psi1 > 180.0){
 67                 fscanf(fp, "%s %s %c %d %f %f", pdbid, residue,
&chain, &resid, &phi1, &psi1);
 68         }
 69         fscanf(fp, "%s %s %c %d %f %f", pdbid, residue, &chain,
&resid, &phi2, &psi2);
 70
 71         while(fscanf(fp, "%s %s %c %d %f %f", pdbid, residue,
&chain, &resid, &phi3, &psi3) == 6){
 72
 73                 if(phi1 > 180.00 || psi1 > 180.00 || phi2 > 180.00
|| psi2 > 180.00 || phi3 > 180.00 || psi3 > 180.00){
 74
 75                         phi1 = phi2;
 76                         psi1 = psi2;
 77
 78                         phi2 = phi3;
 79                         psi2 = psi3;
 80                         continue;
 81                 }
 82
 83                 else{
 84
 85                         x0 = phi1;
 86                         y0 = psi1;
 87
 88                         /* Calculate all possible distances between
residue #1 and all symmetrics to residue #3*/
 89                         x1 = phi3;
 90                         y1 = psi3;
 91                                 d[0] = sqrt( pow((x1-x0), 2) +
pow((y1-y0), 2));
 92
 93                         x1 = phi3;
```

```
 94                          y1 = psi3 + 360;
 95                                  d[1] = sqrt( pow((x1-x0), 2) +
pow((y1-y0), 2));
 96
 97                          x1 = phi3 + 360;
 98                          y1 = psi3 + 360;
 99                                  d[2] = sqrt( pow((x1-x0), 2) +
pow((y1-y0), 2));
100
101                         x1 = phi3 + 360;
102                         y1 = psi3;
103                                 d[3] = sqrt( pow((x1-x0), 2) +
pow((y1-y0), 2));
104
105                         x1 = phi3 + 360;
106                         y1 = psi3 - 360;
107                                 d[4] = sqrt( pow((x1-x0), 2) +
pow((y1-y0), 2));
108
109                         x1 = phi3;
110                         y1 = psi3 - 360;
111                                 d[5] = sqrt( pow((x1-x0), 2) +
pow((y1-y0), 2));
112
113                         x1 = phi3 - 360;
114                         y1 = psi3 - 360;
115                                 d[6] = sqrt( pow((x1-x0), 2) +
pow((y1-y0), 2));
116
117                         x1 = phi3 - 360;
118                         y1 = psi3;
119                                 d[7] = sqrt( pow((x1-x0), 2) +
pow((y1-y0), 2));
120
121                         x1 = phi3 - 360;
122                         y1 = psi3 + 360;
123                                 d[8] = sqrt( pow((x1-x0), 2) +
pow((y1-y0), 2));
124
125                         /*Find minimum distance in distance array*/
126
127                         dmin = 9999;
128                         for(i=0; i<=8; i++){
129
130                                 if(d[i] < dmin){
131                                         dmin = d[i];
132                                 }
133                         }
134
135
136                         fprintf(ofp, "%10.6f\n", dmin);
137
138                         phi1 = phi2;
139                         psi1 = psi2;
140
141                         phi2 = phi3;
```

```
142                         psi2 = psi3;
143
144                 }
145         }
146         fclose(ofp);
147         fclose(fp);
148 }
```

Main program: `dif_vectors_5residues.c`

```
  1 #include <stdio.h>
  2 #include <math.h>
  3 #include <stdlib.h>
  4 #include <string.h>
  5
  6 #define SD 10.22L
  7 #define LM_SQRT2 1.41421356237309504880168872420969 81L
  8
  9         /*********************************/
 10         /*                               */
 11         /*      Variable Declarations    */
 12         /*                               */
 13         /*********************************/
 14
 15 double min_distance(float phi1, float psi1, float phi2, float psi2);
 16 long double probability(double i);
 17
 18 int main(int argc, char *argv[])
 19 {
 20
 21         FILE *fp;
 22         FILE *ofp;
 23
 24         char line[200];
 25         char pdbcode[5];
 26         char chain;
 27         int resid, resid1, resid2, resid3, resid4, resid5;
 28         char residue[10], residue1[10], residue2[10], residue3[10],
residue4[10], residue5[10];
 29         float phi, psi;
 30         float phi1, phi2, phi3, phi4, phi5;
 31         float psi1, psi2, psi3, psi4, psi5;
 32         double d1, d2, d3, d4;
 33         double s1, s2, s3, s4;
 34         //double global_min_distance = 999.0;
 35         long double prob_d1, prob_d2, prob_d3, prob_d4, prob_s1,
prob_s2, prob_s3, prob_s4;
 36         long double logodd_d1, logodd_d2, logodd_d3, logodd_d4;
 37         long double logodd_s1, logodd_s2, logodd_s3, logodd_s4;
 38         long double logodd_sum;
```

```
 39
 40
 41
 42          /*********************************/
 43          /*                               */
 44          /*      Argument sanity check    */
 45          /*                               */
 46          /*********************************/
 47
 48
 49          if (argc != 2){
 50
 51                  printf ("Usage: dif_vectors [angle_file_name]\n");
 52                  exit(1);
 53          }
 54
 55
 56          /*********************************/
 57          /*                               */
 58          /*      Open file and check if   */
 59          /*      if it is correct         */
 60          /*                               */
 61          /*********************************/
 62
 63
 64          fp = fopen (argv[1], "r");
 65          puts ("\nChecking file...");
 66
 67          while (fgets(line, sizeof(line), fp) != NULL){
 68
 69                  if(sscanf(line, "%s %s %c %d %f %f", pdbcode,
residue, &chain, &resid, &phi, &psi) != 6){
 70
 71                          puts("Error: Wrong input file format. It
must contain six columns");
 72                          exit(1);
 73                  }
 74                  sscanf(line, "%s %s %c %d %f %f", pdbcode, residue,
&chain, &resid, &phi, &psi);
 75                  if(phi == 180.00 || psi == 180.00){
 76
 77                          puts("Error: Angle value +180.00 found.
Replace with -180.00 and re-run.");
 78                          exit(1);
 79                  }
 80          }
 81
 82
 83          /*********************************/
 84          /*                               */
 85          /*      Euclidian distance,      */
```

```
 86         /*      probability, and        */
 87         /*      logodd calculation      */
 88         /*                              */
 89         /********************************/
 90
 91         puts ("Data is good! Passing dataset to calculate distances
and probabilities...");
 92         rewind(fp);
 93         ofp = fopen("probabilities.dat", "w");
 94
 95         if(ofp == NULL) {
 96
 97                 printf("Error: Cannot open output file\n");
 98                 exit(1);
 99         }
100
101                 // Read phi, psi angle values for 5 consecutive
residues at a time
102
103         fscanf(fp, "%s %s %c %d %f %f", pdbcode, residue1, &chain,
&resid1, &phi1, &psi1);
104         if (phi1 > 180.0 || psi1 > 180.0){
105                 fscanf(fp, "%s %s %c %d %f %f", pdbcode, residue1,
&chain, &resid1, &phi1, &psi1);
106         }
107
108         fscanf(fp, "%s %s %c %d %f %f", pdbcode, residue2, &chain,
&resid2, &phi2, &psi2);
109         fscanf(fp, "%s %s %c %d %f %f", pdbcode, residue3, &chain,
&resid3, &phi3, &psi3);
110         fscanf(fp, "%s %s %c %d %f %f", pdbcode, residue4, &chain,
&resid4, &phi4, &psi4);
111
112         while(fscanf(fp, "%s %s %c %d %f %f", pdbcode, residue5,
&chain, &resid5, &phi5, &psi5) == 6){
113
114                 // Skip terminal residues, and everything that has
999.90 angle value
115
116                 if(phi1 > 180.00 || psi1 > 180.00 || phi2 > 180.00
|| psi2 > 180.00 || phi3 > 180.00 || psi3 > 180.00 || phi4 > 180.00 ||
psi4 > 180.00 || phi5 > 180.00 || psi5 > 180.00){
117                         phi1 = phi2;
118                         psi1 = psi2;
119
120                         phi2 = phi3;
121                         psi2 = psi3;
122
123                         phi3 = phi4;
124                         psi3 = psi4;
125
```

```
126                         phi4 = phi5;
127                         psi4 = psi5;
128
129                         strcpy(residue1, residue2);
130                         strcpy(residue2, residue3);
131                         strcpy(residue3, residue4);
132                         strcpy(residue4, residue5);
133
134                         resid1 = resid2;
135                         resid2 = resid3;
136                         resid3 = resid4;
137                         resid4 = resid5;
138
139                         continue;
140                 }
141
142                 else{
143
144                         // Calculate i - i+1 euclidian distances in
2D space
145
146                         s1 = min_distance(phi1, psi1, phi2, psi2);
//Calculate minimum distance using the min_distance() function
147                                 prob_s1 = probability(s1);
//Calculate erfc using the probability() funcion
148                                 logodd_s1 = log1pl(-prob_s1)-
logl(prob_s1);     //Calculate the reverse probability log-odds
(probability of two consecutive residues to be in different regions of
the Ramachandran distribution plot
149
150                         s2 = min_distance(phi2, psi2, phi3, psi3);
151                                 prob_s2 = probability(s2);
152                                 logodd_s2 = log1pl(-prob_s2)-
logl(prob_s2);
153
154                         s3 = min_distance(phi3, psi3, phi4, psi4);
155                                 prob_s3 = probability(s3);
156                                 logodd_s3 = log1pl(-prob_s3)-
logl(prob_s3);
157
158                         s4 = min_distance(phi4, psi4, phi5, psi5);
159                                 prob_s4 = probability(s4);
160                                 logodd_s4 = log1pl(-prob_s4)-
logl(prob_s4);
161
162
163                         // Calculate i - i+2 euclidian distances in
2D space
164
165                         d1 = min_distance(phi1, psi1, phi3, psi3);
166                                 prob_d1 = probability(d1);
```

```
167                                 logodd_d1 = logl(prob_d1)-log1pl(-
prob_d1);     //Log-odds of the probability of a residue i and a residue
i+2 to be in the same region of the Ramachandran distribution plot
168
169                             d2 = min_distance(phi2, psi2, phi4, psi4);
170                                 prob_d2 = probability(d2);
171                                 logodd_d2 = logl(prob_d2)-log1pl(-
prob_d2);
172
173                             d3 = min_distance(phi3, psi3, phi5, psi5);
174                                 prob_d3 = probability(d3);
175                                 logodd_d3 = logl(prob_d3)-log1pl(-
prob_d3);
176
177                             d4 = min_distance(phi1, psi1, phi5, psi5);
178                                 prob_d4 = probability(d4);
179                                 logodd_d4 = logl(prob_d4)-log1pl(-
prob_d4);
180
181                             //Sum of log-odds
182
183                             logodd_sum =
logodd_s1+logodd_s2+logodd_s3+logodd_s4+logodd_d1+logodd_d2+logodd_d3+lo
godd_d4;
184
185                             fprintf(ofp, "%17.13Lf %s\t%9s %2c %4d %9s
%2c %4d\n", logodd_sum, pdbcode, residue1, chain, resid1, residue5,
chain, resid5);
186
187                             //Prepare to go to the next 5 redidues
188
189                             phi1 = phi2;
190                             psi1 = psi2;
191
192                             phi2 = phi3;
193                             psi2 = psi3;
194
195                             phi3 = phi4;
196                             psi3 = psi4;
197
198                             phi4 = phi5;
199                             psi4 = psi5;
200
201                             strcpy(residue1, residue2);
202                             strcpy(residue2, residue3);
203                             strcpy(residue3, residue4);
204                             strcpy(residue4, residue5);
205
206                             resid1 = resid2;
207                             resid2 = resid3;
208                             resid3 = resid4;
```

```
209                         resid4 = resid5;
210
211                 }
212         }
213         fclose(fp);
214         fclose(ofp);
215         return(0);
216 }
217
218         // This function caclulates all the possible distances of
the two given residues in the 2D space, and finds the one with the
minimum value. This is for avoiding the periodicity of dihedral angles.
219
220 double min_distance(float phi1, float psi1, float phi2, float psi2)
221 {
222         double d[10];
223         double dmin;
224         float x0, y0, x1, y1;
225         int i;
226
227         x0 = phi1;
228         y0 = psi1;
229
230         // Calculate all possible distances between residue #1 and
all symmetrics to residue #2
231
232         x1 = phi2;
233         y1 = psi2;
234                 d[0] = sqrt( pow((x1-x0), 2) + pow((y1-y0), 2));
235
236         x1 = phi2;
237         y1 = psi2 + 360;
238                 d[1] = sqrt( pow((x1-x0), 2) + pow((y1-y0), 2));
239
240         x1 = phi2 + 360;
241         y1 = psi2 + 360;
242                 d[2] = sqrt( pow((x1-x0), 2) + pow((y1-y0), 2));
243
244         x1 = phi2 + 360;
245         y1 = psi2;
246                 d[3] = sqrt( pow((x1-x0), 2) + pow((y1-y0), 2));
247
248         x1 = phi2 + 360;
249         y1 = psi2 - 360;
250                 d[4] = sqrt( pow((x1-x0), 2) + pow((y1-y0), 2));
251
252         x1 = phi2;
253         y1 = psi2 - 360;
254                 d[5] = sqrt( pow((x1-x0), 2) + pow((y1-y0), 2));
255
256         x1 = phi2 - 360;
```

```
257         y1 = psi2 - 360;
258                 d[6] = sqrt( pow((x1-x0), 2) + pow((y1-y0), 2));
259
260         x1 = phi2 - 360;
261         y1 = psi2;
262                 d[7] = sqrt( pow((x1-x0), 2) + pow((y1-y0), 2));
263
264         x1 = phi2 - 360;
265         y1 = psi2 + 360;
266                 d[8] = sqrt( pow((x1-x0), 2) + pow((y1-y0), 2));
267
268         /*Find minimum distance in distance array*/
269
270         dmin = 9999;
271         for(i=0; i<=8; i++){
272
273                 if(d[i] < dmin){
274                         dmin = d[i];
275                 }
276         }
277         return dmin;
278 }
279
280         // Probability function using the complementary error
function
281
282 long double probability(double i){
283         long double prob;
284
285         prob = erfcl(i/(2*SD*LM_SQRT2));         /*Complementary
Error Function*/
286         return prob;
287 }
```