ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

**Laboratory for Computation and Visualization in Mathematics and Mechanics (LCVM$^2$)**

Institut des mathématiques Bernoulli (IMRI) - Section de mathématiques (SMA)
Faculté des Sciences de Base (SB)

---

# Computational physics project
# Final report

## $C_\alpha$ circles

---

Christian Schlatter[(1)]

11, avenue de Mont-Goulin
CH-1008 Prilly
Switzerland

July, 2002

Made with LaTeX $2_\varepsilon$

(1) student in physics, christian.schlatter@epfl.ch

# Contents


**1 Introduction** **1**

**2 Theoretical background** **2**
2.1 Proteins . . . 2
2.2 Global curvature, thickness and f-ratio . . . 4
2.3 The Protein Data Bank . . . 6

**3 The algorithm** **7**
3.1 EXTRACTOR (Release 1.31) . . . 7
3.2 CALCULATOR (Release 1.62) . . . 8
3.2.1 Fitting circles through three points . . . 9
3.2.2 Fitting spheres through four points . . . 9
3.2.3 Data saving . . . 10
3.3 Sourcecode details . . . 10

**4 Discussion** **11**
4.1 Program testing . . . 11
4.2 Results . . . 11
4.2.1 Global and local radius . . . 11
4.2.2 Dislocation of circles and spheres . . . 14
4.2.3 Optimality of protein shaping . . . 15
4.2.4 Thickness of proteins . . . 15
4.3 Runtime analysis . . . 19
4.4 Conclusion . . . 20
4.4.1 Acknowledgements . . . 20

**A Sample data files** **22**
A.1 $C_\alpha$ input data file . . . 22
A.2 $C_\alpha$ output data file (circles) . . . 23
A.3 $C_\alpha$ output data file (spheres) . . . 24

**B Sourcecode of the algorithm** **25**
B.1 EXTRACTOR . . . 25
B.2 CALCULATOR . . . 42

# List of Figures


2.1 The backbone of a protein. . . . . 2
2.2 The bovine pancreatic trypsin inhibitor (BPTI) protein. . . . . 3

4.1 Local and global circular radius of curvature. . . . . 12
4.2 Local and global spherical radius of curvature. . . . . 12
4.3 Distribution of the smallest global circular radii of curvature. . . . . 13
4.4 Distribution of the smallest global spherical radii of curvature. . . . . 13
4.5 Behavior of the nine smallest circle radii for each $C_\alpha$ atom. . . . . 14
4.6 Behavior of the nine smallest sphere radii for each $C_\alpha$ atom. . . . . 15
4.7 Dislocation number ✠ for circles fitted through BPTI. . . . . 16
4.8 Dislocation number ✠ for spheres fitted through BPTI. . . . . 16
4.9 Packing of $\alpha$-helix in the chicken cartilage matrix protein. . . . . 17
4.10 Optimality of packing (circular). . . . . 18
4.11 Optimality of packing (spherical). . . . . 18
4.12 Runtime analysis for sphere calculation. . . . . 19

# List of Tables

2.1 Description of the bovine pancreatic trypsin inhibitor in the Protein Data Bank (PDB). Parts of the header and the description of the backbone are shown. . . . 6

4.1 Circular and spherical thickness $\triangle$ for some processed proteins. Note that the number of atoms corresponds to the number of extracted $C_\alpha$ atoms. The thickness doesn't depend on the length of the proteins backbone. . . . . 17

# Chapter 1

# Introduction

This computational physics project makes part of the two-semestrial lecture "Simulation numérique de systèmes physiques" taught by professor Alfredo Pasquarello to students in physics. I had the pleasure to get a project at the Bernoulli institute from professor John Maddocks (Chaire d'analyse appliquée).

Proteins show a high packing density of residues. This packing has not been completely characterized(1) yet. The molecule packing effects on functional dynamics.
Very recently, Bagci, Jernigan and Bahar studied the packing of amino acids of different sizes and shapes [10] and [11]. Bagci and al. found that the residue of coarse-grained(2) protein structures show a quite uniform density. Too, their analysis of the packing geometry leads to the conclusion, that the idealization of the molecule as homopolymeric chain (formed of equivalent residues) packs as spheres(3) do.

Other recent work [7], [8] suggested that the packing of 3-dimensional structures is related to the concept of global curvature [9]. Using this approach, Maritan and al. [7] studied the packing of the most densely packed core regions, which are found to be those where the backbone of the protein forms special tertiary structures like helices. Such compact strings are found to be optimally shaped.

The algorithm developed within this project serves to analyze the presence and organization of several 3-dimensional protein configurations (determined by experiment) using the concept of global curvature. The whole backbone (or only sequences forming helices) of proteins can be treated using the coarse grained idealization of geometrical structure. The algorithm permits to study the behavior of the global curvature based on the fitting of circles and spheres through three or four residues.

The next chapter lights up some facts about proteins, the concept of the global radius and the Protein Data Bank. In chapter 3 the organization and content of the code is presented. The last chapter gives the results of the calculations and opens the interpretation phase. The appendix lists the program source code and some sample data files.

(1) Analyzing the genome could be simplified if packing metrics would be known.
(2) One space point per residue.
(3) Closest packing of spheres corresponds to a fcc-lattice, as suggested already by Kepler and definitely proven in 1998 only! [12].

# Chapter 2

# Theoretical background

## 2.1 Proteins

A protein molecule is a linear polymer that is made from a long chain of amino acids, each linked to its neighbor through a covalent peptide bond. The repeating sequence of atoms along the chain is referred to as the polypeptide backbone (see figure 2.1).

Different[(1)] amino acid side chains are attached to one of the central carbon atoms (called the $C_\alpha$ atom) of each backbone amino acid. The side chains are these portions of the amino acids that are not involved in making a peptide bond and which gave each amino acid its unique properties.

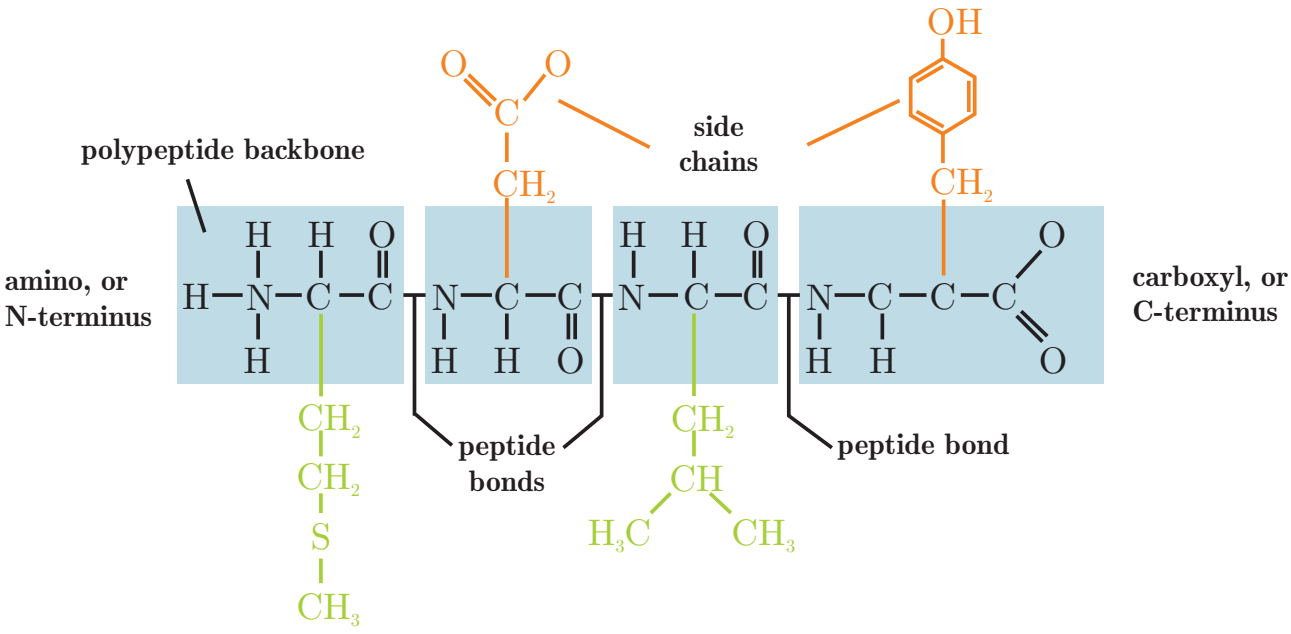


**Figure 2.1: The backbone of a protein.** Four residues (blue blocks) are shown. From the left to the right a methionine, an aspartic acid, a leucine and a tyrosine are bonded together to form the backbone of a protein. The two ends of a polypeptide chain are always different. The beginning group consists of an $NH_2$ - the end group of COOH. Proteins are always read from the N to the C direction.

(1) Out of 20 kinds.

Proteins are extremely varied and of complicated shape. The most simple protein has about 30 amino acids, the most complicated about 10'000. Its backbones and side chains are folded into a conformation of lowest energy. The over all conformation of each protein is unique, but locally, two regular folding patterns are often found: the $\alpha$-helix (the backbone turns around itself to form a cylinder) and the $\beta$-sheet (several parallel aligned backbones sequences).

The bovine pancreatic trypsin inhibitor (BPTI) is a protein which has been largely studied in chemistry, biology, physics and numerical experiments. BPTI is present in many tissues of the human body where it inhibits the proteolytic action of the enzyme trypsin. BPTI is a monomeric polypeptide containing 58 amino acid residues. Figure 2.2 shows BPTI and is colored by secondary structure.

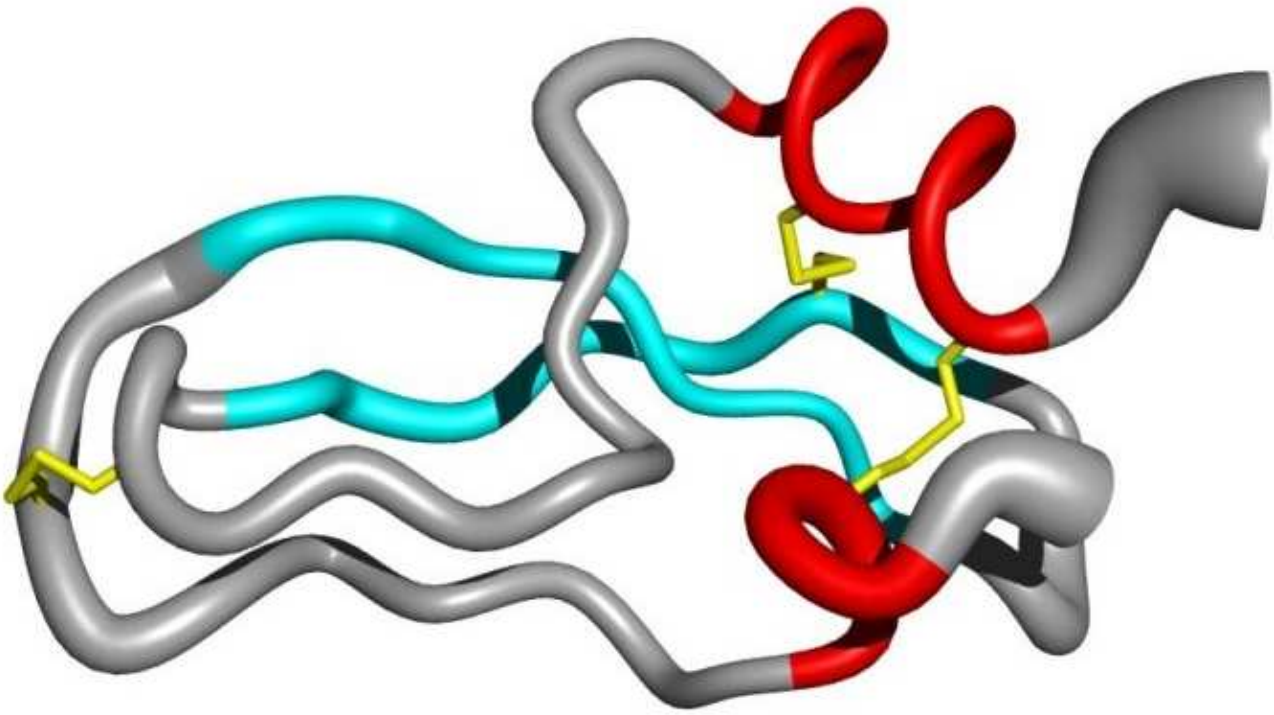

**Figure 2.2: Bovine pancreatic trypsin inhibitor (BPTI) protein.** The coil (in grey) is colored by secondary structure. In red the two $\alpha$-helices, in blue the $\beta$-sheets. The three disulfide bonds are shown in yellow.

The study of the spatial configuration of the backbone at a coarse grained level can be done by considering the coordinates of only one repetitive backbone amino acid atom, as for example the $C_\alpha$ carbon atom (hence the name of this project.

## 2.2 Global curvature, thickness and f-ratio

The $n$ atoms of a chain-like molecule (as the backbone of a protein) can be represented as discrete curve $\mathcal{C}_n$ by a list of distinct points $\{\mathbf{q}_1, .., \mathbf{q}_n\}$ in three-dimensional space. The points are given by a repetitive atom type figuring in the backbone (coarse-grain level consideration).

Let $\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k \in \mathbb{R}^3$, with $i, j, k \in \{1, .., n\}$ be three such non-collinear points. An unique circle passing through each of these points can be construed whose radius $r(\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k)$ is given by

$$r(\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k)^2 = \frac{-D_{ij}^2 D_{ik}^2 D_{jk}^2}{D_{ij}^4 + D_{ik}^4 + D_{jk}^4 - 2(D_{ij}^2 D_{ik}^2 + D_{ik}^2 D_{jk}^2 + D_{ij}^2 D_{jk}^2)} \tag{2.1}$$

where $D_{ij} = |\mathbf{q}_i - \mathbf{q}_j|$ is the euclidian distance between points $i$ and $j$.

The local radius of curvature $\rho$ at point $\mathbf{q}_i$ of curve $\mathcal{C}_n$ corresponds to the circle which locally best approximates the curve. In the discrete case it is simply defined by

$$\rho(\mathbf{q}_i) = r(\mathbf{q}_{i-1}, \mathbf{q}_i, \mathbf{q}_{i+1}), \quad 1 < i < n \tag{2.2}$$

The global radius of curvature $\rho_G$ at point $\mathbf{q}_i$ of $\mathcal{C}_n$ generalizes the concept of local curvature. It is the radius of the smallest circle containing $\mathbf{q}_i$ and any two other distinct points $\mathbf{q}_j$ and $\mathbf{q}_k$ i.e.

$$\rho_G(\mathbf{q}_j) = \min_{\substack{1 \leqslant i,k \leqslant n \\ i \neq j \neq k \neq i}} r(\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k), \quad 1 < j < n \tag{2.3}$$

The global radius of curvature $\rho_G$ takes into account both, the local bending of the curve and the non-local proximity of different parts of the curve and is therefore predestined for a characterization of its packing.

Finally the thickness $\Delta[\mathcal{C}_n]$ associated to $\mathcal{C}_n$, i.e.

$$\Delta[\mathcal{C}_n] = \min_{1 \leqslant i \leqslant n} \rho_G(\mathbf{q}_i) \tag{2.4}$$

denotes the maximum radius if a uniform tube with the curve passing through its axis, beyond which the tube either ceases to be smooth, owing to tight local bends, or it self-intersects.

To characterize optimal strings (such as helices) in terms of the sphere packing problem, Maritan and al. [7] introduced the f-ratio. Optimal helices show equality of the local radius of curvature $\rho$ and the radius that is associated with a suitable triplet of non-consecutive points lying in two successive turns of the helix. Let us call the non-local radius of this special non-local triplet $\rho_N$, defined by

$$\rho_N(\mathbf{q}_j) = \min_{\substack{1 \leqslant i,k \leqslant n \\ (k-j)+(j-i) > 2 \\ i \neq j \neq k \neq i}} r(\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k), \quad 1 < j < n \tag{2.5}$$

The local f-ratio $f(\mathbf{q}_i)$ is then defined by

$$f(\mathbf{q}_i) = \frac{\rho_N}{\rho} \tag{2.6}$$

For $f(\mathbf{q}_i) > 1$ the global radius of curvature $\rho_G(\mathbf{q}_i)$ corresponds to the local radius of curvature $\rho(\mathbf{q}_i)$ (local regime), for $f(\mathbf{q}_i) < 1$ the global radius of curvature is strictly lower than the local radius, and the thickness of the helix is determined basically by the distance between two consecutive helix turns. Optimal packing selects the very special helix with $f(\mathbf{q}_i) = 1$.
The global f-ratio $F[\mathcal{C}_n]$, averaged over the whole curve (or an interval of the curve) measures then the optimality of its packing.

All the above notions can be extended to the case where spheres are fitted through four points of the backbone.
Let $\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k, \mathbf{q}_l \in \mathbb{R}^3$, with $i, j, k, l \in \{1, .., n\}$ be four non-coplanar points. An unique sphere passing through each of these points can be construed whose radius $r(\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k, \mathbf{q}_l)$ is given by

$$r(\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k, \mathbf{q}_l)^2 = \frac{-\begin{vmatrix} 0 & D_{ij}^2 & D_{ik}^2 & D_{il}^2 \\ D_{ij}^2 & 0 & D_{jk}^2 & D_{jl}^2 \\ D_{ik}^2 & D_{jk}^2 & 0 & D_{kl}^2 \\ D_{il}^2 & D_{jl}^2 & D_{kl}^2 & 0 \end{vmatrix}}{2\begin{vmatrix} 0 & D_{ij}^2 & D_{ik}^2 & D_{il}^2 & 1 \\ D_{ij}^2 & 0 & D_{jk}^2 & D_{jl}^2 & 1 \\ D_{ik}^2 & D_{jk}^2 & 0 & D_{jl}^2 & 1 \\ D_{il}^2 & D_{jkl}^2 & D_{kl}^2 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{vmatrix}} \tag{2.7}$$

where $D_{ij} = |\mathbf{q}_i - \mathbf{q}_j|$ is the euclidian distance between points $i$ and $j$.

The local radius of curvature $\rho$ at point $\mathbf{q}_i$ is logically

$$\begin{gathered} \rho(\mathbf{q}_2) = r(\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4) \\ \rho(\mathbf{q}_i) = min\left\{r(\mathbf{q}_{i-2}, \mathbf{q}_{i-1}, \mathbf{q}_i, \mathbf{q}_{i+1}), r(\mathbf{q}_{i-1}, \mathbf{q}_i, \mathbf{q}_{i+1}, \mathbf{q}_{i+2})\right\}, 2 < i < (n-1) \\ \rho(\mathbf{q}_{n-1}) = r(\mathbf{q}_{n-3}, \mathbf{q}_{n-2}, \mathbf{q}_{n-1}, \mathbf{q}_n) \end{gathered} \tag{2.8}$$

The local radius is not defined for points $\mathbf{q}_1$ and $\mathbf{q}_n$. To be consistent, the global radius of curvature $\rho_G$ at point $\mathbf{q}_i$ is

$$\rho_G(\mathbf{q}_j) = \min_{\substack{1 \leqslant i,k,l \leqslant n \\ k \neq i \neq j \neq k \neq l \neq i \\ j \neq l}} r(\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k, \mathbf{q}_l), \quad 1 < j < n \tag{2.9}$$

and the non-local radius $\rho_N$ will then be

$$\rho_N(\mathbf{q}_j) = \min_{\substack{1 \leqslant i,k,l \leqslant n \\ (l-k)+(k-j)+(j-i) > 3 \\ k \neq i \neq j \neq k \neq l \neq i \\ j \neq l}} r(\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k, \mathbf{q}_l), \quad 1 < j < n \tag{2.10}$$

The thickness and the f-ratio are calculated accordingly.

## 2.3 The Protein Data Bank

The algorithm uses real protein structure data which has been determined by experiment. Protein data can been downloaded from the huge Protein Data Bank (PDB) database[(2)], which is hosting almost 20'000 biological macromolecular structures.

The Protein Data Bank uses a proprietary data format to save many information about protein. Each single protein is documented in so called PDB-files. Table 2.1 shows some parts of the file describing the BPTI protein.

A comprehensive description of the PDB file format can be found in [14]. The algorithm at issue conforms to release 2.2 of the PDB file format description.

(2) http://www.rcsb.org/pdb/

```
HEADER  PROTEINASE INHIBITOR (TRYPSIN)              13-MAY-87   6PTI       6PTI    3
COMPND  BOVINE PANCREATIC TRYPSIN INHIBITOR (/BPTI$\$$,CRYSTAL FORM        6PTI    4
...
REMARK   2 RESOLUTION. 1.7 ANGSTROMS.                                      6PTI   36
...
HELIX    1  H1 PRO     2  GLU     7  5 ALL DONORS,ACCEPTORS INCLUDED       6PTI   58
HELIX    2  H2 SER    47  GLY    56  1 ALL DONORS,ACCEPTORS INCLUDED       6PTI   59
...
ATOM     1  N   ARG     1      12.579  -6.640  11.376  1.00 24.04         6PTI   73
ATOM     2  CA  ARG     1      12.139  -5.568  10.464  1.00 21.52         6PTI   74
ATOM     3  C   ARG     1      11.304  -6.259   9.366  1.00 18.34         6PTI   75
ATOM     4  O   ARG     1      11.724  -7.230   8.757  1.00 17.22         6PTI   76
ATOM     5  CB  ARG     1      13.190  -4.672   9.878  1.00 22.85         6PTI   77
ATOM     6  CG  ARG     1      13.820  -4.993   8.563  1.00 24.90         6PTI   78
ATOM     7  CD  ARG     1      14.478  -3.884   7.852  1.00 26.62         6PTI   79
ATOM     8  NE  ARG     1      15.347  -3.042   8.595  1.00 29.03         6PTI   80
ATOM     9  CZ  ARG     1      16.555  -3.097   9.091  1.00 29.05         6PTI   81
ATOM    10  NH1 ARG     1      17.400  -4.129   9.005  1.00 29.36         6PTI   82
ATOM    11  NH2 ARG     1      17.042  -2.028   9.750  1.00 29.94         6PTI   83
ATOM    12  N   PRO     2      10.099  -5.724   9.193  1.00 15.90         6PTI   84
ATOM    13  CA  PRO     2       9.166  -6.247   8.181  1.00 14.63         6PTI   85
ATOM    14  C   PRO     2       9.700  -6.139   6.774  1.00 12.14         6PTI   86
ATOM    15  O   PRO     2      10.428  -5.188   6.389  1.00 11.64         6PTI   87
ATOM    16  CB  PRO     2       7.915  -5.381   8.391  1.00 14.77         6PTI   88
ATOM    17  CG  PRO     2       8.076  -4.722   9.708  1.00 15.32         6PTI   89
ATOM    18  CD  PRO     2       9.578  -4.563   9.914  1.00 15.68         6PTI   90
ATOM    19  N   ASP     3       9.324  -7.061   5.899  1.00 12.40         6PTI   91
ATOM    20  CA  ASP     3       9.805  -7.020   4.499  1.00 12.67         6PTI   92
ATOM    21  C   ASP     3       9.443  -5.726   3.781  1.00  9.95         6PTI   93
ATOM    22  O   ASP     3      10.077  -5.313   2.802  1.00  9.25         6PTI   94
ATOM    23  CB  ASP     3       9.193  -8.193   3.699  1.00 18.98         6PTI   95
ATOM    24  CG  ASP     3       9.814  -9.527   4.031  1.00 26.04         6PTI   96
ATOM    25  OD1 ASP     3       9.236 -10.570   3.638  1.00 29.28         6PTI   97
ATOM    26  OD2 ASP     3      10.892  -9.510   4.691  1.00 26.07         6PTI   98
```

**Table 2.1:** Description of the bovine pancreatic trypsin inhibitor in the Protein Data Bank (PDB). Parts of the header and the description of the backbone are shown.

# Chapter 3

# The algorithm

The present algorithm consists mainly of two programs, called **EXTRACTOR** and **CALCULATOR**. **EXTRACTOR** has to be used to generate the data files which will be read by **CALCULATOR**. Its input are data files downloaded from PDB. **EXTRACTOR** loads and analyzes these files and extracts the information we are interested in. The output are proprietary data files in the $C_\alpha$ source format (see appendix A, $C_\alpha$ input data file format). Input and output files are read and written from and to a subdirectory tree on the local disk. This tree structure can be created using the batch file **DIRECTORIES.BAT**.

**CALCULATOR** does only accept the $C_\alpha$ data files as input, which will be used for the 'circling' and 'sphereing' of the protein. The results of the calculation are stored into a subdirectory tree on disk in the proprietary $C_\alpha$ result format (see appendix A, $C_\alpha$ output data file format).

## 3.1 EXTRACTOR (Release 1.31)

When calling the **EXTRACTOR** the user has to provide the filename of the PDB file to be processed. Alternatively, several PDB files can be listed[(1)] in a file called *Files.ini*. **EXTRACTOR** will then process file by file given by this list. All PDB files have to be stored in the subfolder *PDB*.

Next, the user has to provide the type of atoms to extract. For alpha-carbons ($C_\alpha$) the corresponding identifier is **CA** (the same identifier as in PDB files is used).

The third step is to select if the whole backbone of the protein should get extracted or only atoms which are located in secondary structure $\alpha$-helices. The latter case has been implemented for comparison purposes with existing results done by [7] (see figure 4.9 in section 4.2).

The PDB files are processed accordingly to the PDB format description guide [14] from December 20, 1996. PDB ASCII files are read and processed line by line by the **EXTRACTOR**. Each line starts with a record name followed by data fields of different lengths (see table 2.1). These fields are addressed by dividing the line string up in substrings of the corresponding length.

(1) One complete filename (including the file extension) by line, no blank lines.

From PDB files the `HEADER`, `REMARK 2`, `SETQRES`, `HELIX` and `ATOM` records are interpreted. The `HEADER` entry contains classification details about the protein, the unique PDB ID, the date of acquisition of the PDB data and sometimes experimental techniques. The `REMARK 2` entry may[2] contain information about the highest resolution (in Å) that was used in building the model. The resolution is extracted for error calculation purposes, but the release at issue doesn't make use of it (if present its written to the header of the output file). The `SETQRES` record lists the number of residues in the backbone. In case the whole backbone is extracted, this number is used for initialization of variables. By contrast, when extracting is limited to $\alpha$-helices only, the `HELIX` record is used to determine the location and number of the concerned atoms. There may be several $\alpha$-helices in the protein, both types - right- and left-handed - are considered. Finally, the atom coordinates and the corresponding chain identifier are provided from the `ATOM` record. If multiple chains are present, all are getting extracted. If the PDB file contains more than one model[3], only the first is taken into consideration.

Please be aware that PDB files may contain errors. If the number of residues given in the `SETQRES` or HELIX record are not correct, then the algorithm shrinks or expands the variables by re-initialization to extract the data correctly.

Rarely neighboring C$\alpha$ atoms have been observed, what is obviously impossible. In these cases only the first atom is considered.

If for a given file there were no helices, the file is ignored.

The gathered data is then stored to disk in the folder *source*. For an example of the proprietary file format see appendix A. The output filename are the same as the input filenames. The file extension is changed to *bck* for whole backbone extractions and *hel* for helices only.

## 3.2 CALCULATOR (Release 1.62)

**CALCULATOR** follows the philosophy introduced with **EXTRACTOR**. It can be run in batch or single file mode. In batch mode the files have to be listed[4] in the *Source.ini* file; in single mode the user provides the filename. **CALCULATOR** only accepts input files in the $C_\alpha$ input file format. Therefore, any PDB file has previously to be processed by **EXTRACTOR**.

The user has the choice between fitting circles only or circles and spheres. Fitting spheres does take much more time than fitting circles, see section 4.3 for more information.

[2] For diffraction experiments only.
[3] Using different experimental techniques or mutants.
[4] One complete filename (including the file extension) by line, no blank lines.

### 3.2.1 Fitting circles through three points

The radius of a circle going through 3 points is calculated according to formula 2.1.

When three points are collinear one can't draw a circle through them of course. The denominator in equation 2.1 vanishes for three collinear points, so this criterion is used to prevent collinearity.

The calculated radii are classed in the following three categories :

- local radii : three neighboring atoms i, j and k with $i < j = i + 1 < k = j + 1$ define the local radius for the atom j in the middle.

- partly adjacent radii : two of the three radii are neighbors.

- non-adjacent radii : no atoms are neighbors.

For each atom the smallest radii are ordered with increasing size, the number of memorized radii is limited by the flag *memory*.

When all combinations of circles have been calculated the global radius, the non-local radius, the f-number and the thickness are determined.

To know the distribution of the smallest circles the thickness of the protein has to be known (it is impossible to store all combinations of circles and doing statistics on the whole set of radii). To do that, a second call of the circling routine is initiated and the number of smallest radii with a resolution of 1 % is counted. The upper limit radius is given with the flag *limit*.

### 3.2.2 Fitting spheres through four points

The radius of a sphere going through 4 points is calculated according to formula 2.7.

Again, when the four points are coplanar, no sphere can be built with them. For four coplanar points the determinant in the denominator of equation 2.7 vanishes, what can be used as criterion.

The determinant is calculated using the partial pivoting Gaussian elimination scheme [1].

The calculated radii are classed in the following two categories :

- local radii : four neighboring atoms i, j, k and l with $i < j = i + 1 < k = j + 1 < l = k + 1$ define the local radius for the two atoms j and k in the middle.

- non-local radii : all other combinations of atoms.

Afterwards, the global radii, the thickness and the f-number are determined. A second run determines the distribution of the smallest spheres.

### 3.2.3 Data saving

The results are stored to the folder *results*. If the input file contained $\alpha$-helices the data is saved to *results/helix*, backbone data to *results/whole*. The radii are always saved twice : one file is written to the subfolder *human* whose data is formatted to be easy readable by the user (see appendix A), another to the subfolder *machine*, which contains raw data assigned for further processing, e.g. for plotting with Gnuplot.

**Please be aware that uninitialized values, incalculable values and infinity are noted by 0.0 (zero).**

The content of these output files is determined by their file extensions, which are :

- **cir** : local and global circles for each atom, circular thickness, circular f-ratio.
- **sph** : local and global sphere for each atom, spherical thickness, spherical f-ratio.
- **cdt** : distribution of the circles.
- **sdt** : distribution of the spheres.

## 3.3 Sourcecode details

The algorithm has been developed under Microsoft Windows XP Professional using Compaq's Visual Fortran 6.6 Development Environment (a fully Fortran 95 compiler, the former Digital compiler). The code at issue has been made compatible with SUN's Fortran 90 compiler to run the program on the SUN workstation in the LCVM$^2$ calculation laboratory.

The algorithm has been written as flexible as possible. Most variables are allocatables and the heaviest variables are pointers. The final code is a compromise between memory demands and execution speed.

The source code is structured using modules and subroutines. Adjustable parameters are grouped in a special module `parameters` just at the beginning of the files `extractor.f90` and `calculator.f90`. The data type of the main variable (*alphacarbons*) which is used in both programs is also defined (data type *carbons*) in a single module `var`.

**EXTRACTOR** calls the subroutines *helices* (or *backbones*) which are located in the module `reading` to extract the coordinates of the atoms in the helix (or in the whole backbone).

**CALCULATOR** reads its input files using the subroutine *import* in the module `loading`. The circles (or spheres) are then calculated using the subroutines *circles* (or *spheres*) in module `distance`. The subroutine *spheres* makes use of the function *determinant* in module `mathe` to calculate the determinant of the expressions in formula 2.7.

For further details please refer to the source code of the algorithm in appendix B.

# Chapter 4

# Discussion

## 4.1 Program testing

During development, the algorithm has been tested on several PDB files. All mutants or variants[(1)] of the BPTI protein have been successfully processed. From April to August 2002 a bunch of as different as possible proteins have been processed. At present different output files are ready for interpretation.
The plausibility of the resulting global radii has been confirmed by comparing to previous work, see section 4.2.3

## 4.2 Results

The following figures illustrate the data which is produced using **CALCULATOR**. All the calculations have been done using $C_\alpha$ backbone atoms only.

### 4.2.1 Global and local radius

Figures 4.1 and 4.2 show local and global radii of curvature along the backbone of the bovine pancreatic trypsin inhibitor protein (BPTI) consisting of 58 residues. PDB file **1PTI** has been used. The local radius shows quite strong oscillations along the backbone. The circular thickness is achieved with residues 11, 35 and 36 and equals 2.46 Å. The spherical thickness is achieved with residues 49 to 52 and equals 2.84.

During the second run of **CALCULATOR** (i.e. when the thickness is known), the statistics of the smallest circles and spheres can be revealed. Figure 4.3 and 4.4 show the distribution regarding to the thickness for BPTI. The abscissa is given by the normalized global radius $\rho_G/\triangle$. The behavior of the distribution of the radii depends on the number of atoms of the protein. A gap as it can be seen on the distribution of the smallest radii of the circular global radii (figure 4.3) is not always observed.

A more illustrative plot of the behavior of the global radii is given with figures 4.5 and 4.6. They show the smallest found circles and spheres along the backbone. The lowermost curve corresponds to the global radius of curvature. The polygons rely points which belongs to the same position in the hierarchy of increasing global radii.

(1) The Protein Data Bank returns at present 59 entries on a query for 'BPTI'.

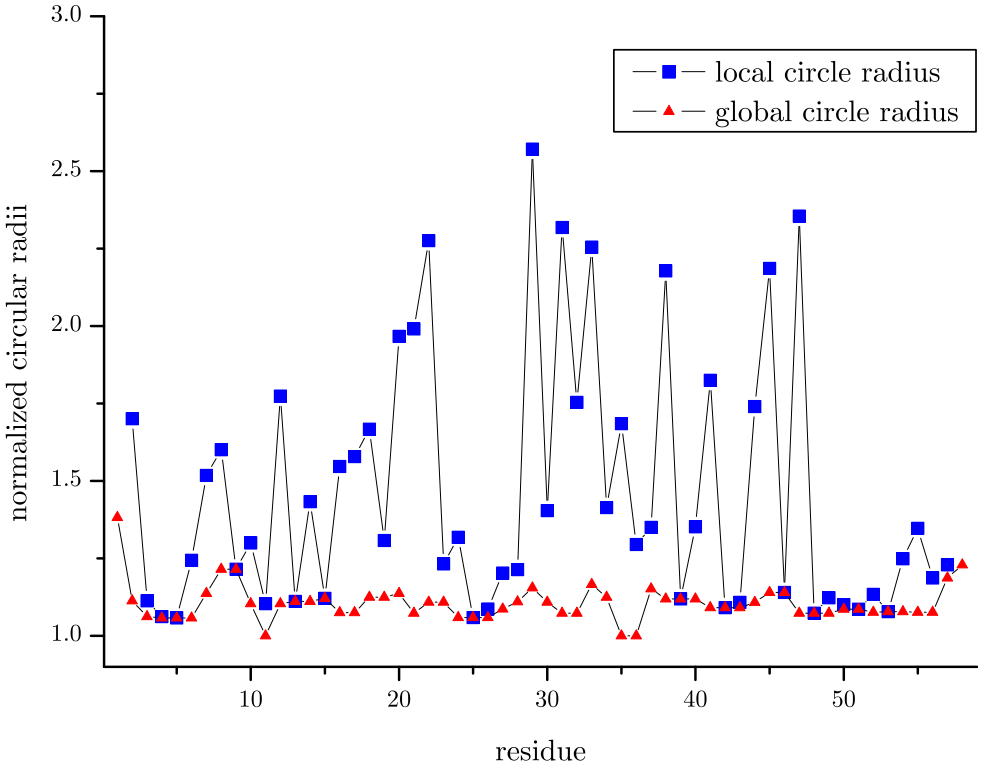


**Figure 4.1: Local and global circular radius of curvature for BPTI.** Blue squares show the local radii, the red triangles the global radii. The vertical axis show normalized values, i.e. all radii has been divided by the circular thickness of BPTI.

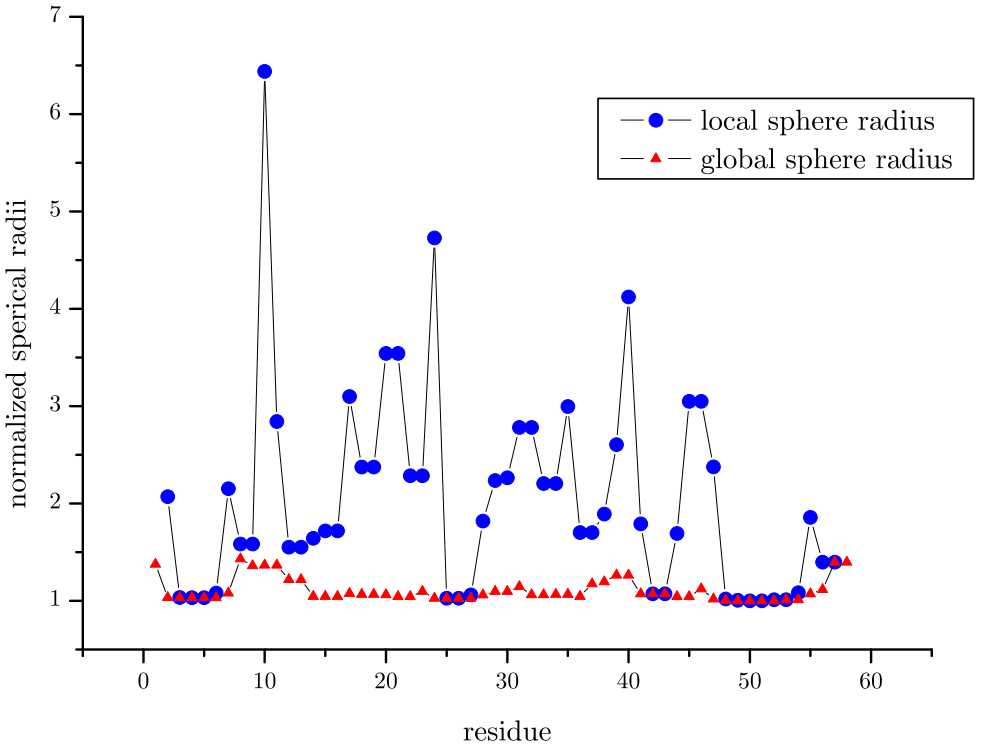


**Figure 4.2: Local and global spherical radius of curvature for BPTI.** Blue squares show the local radii, the red triangles the global radii. The vertical axis show normalized values, i.e. all radii has been divided by the spherical thickness of BPTI.

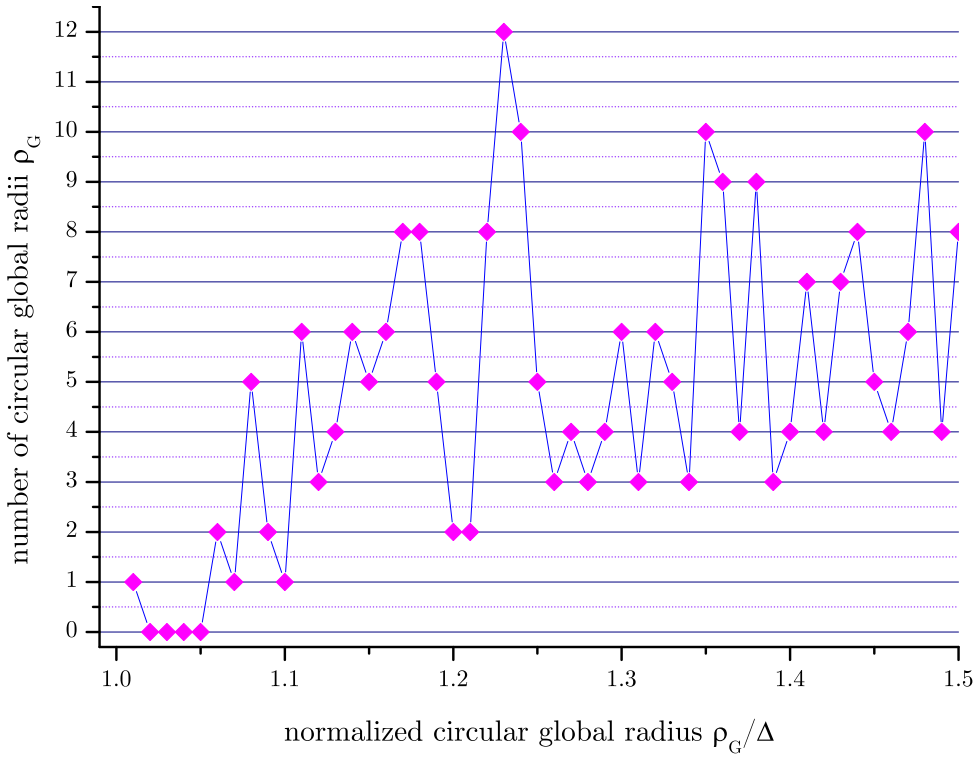


**Figure 4.3: Distribution of the smallest global circular radii of curvature for BPTI.** The resolution of the normalized global radius is 1 %. There are no other circles up to 5 % of the thickness.

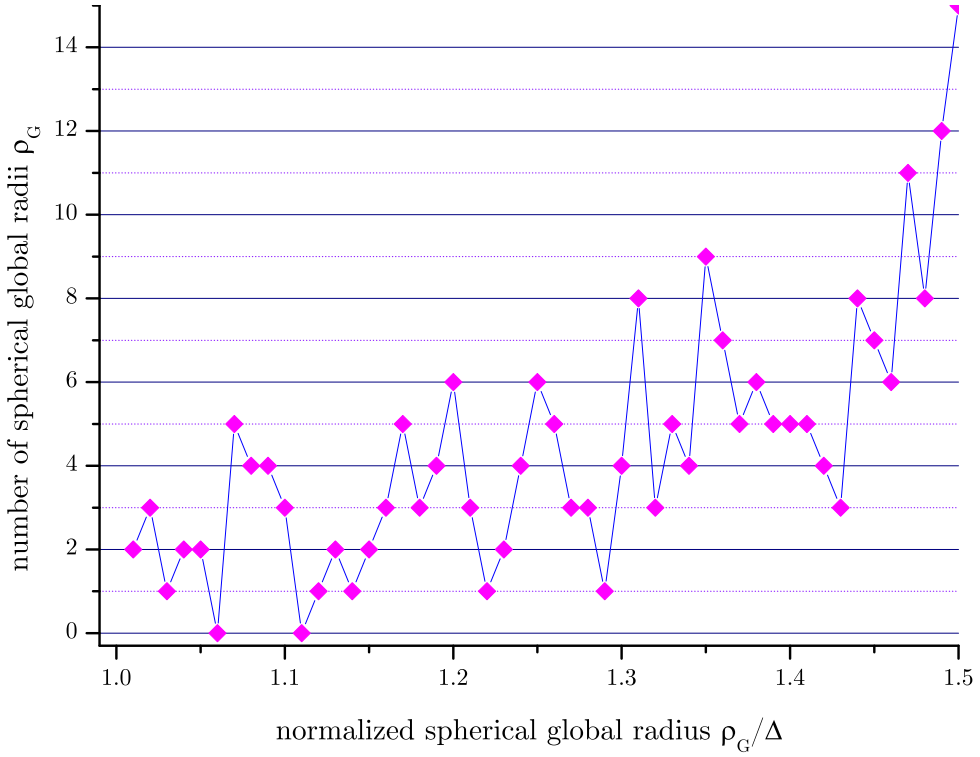


**Figure 4.4: Distribution of the smallest global spherical radii of curvature for BPTI.** The resolution of the scaling factor is 1 %. Two spheres have radius in the interval {1.00, 1.01} times the thickness.

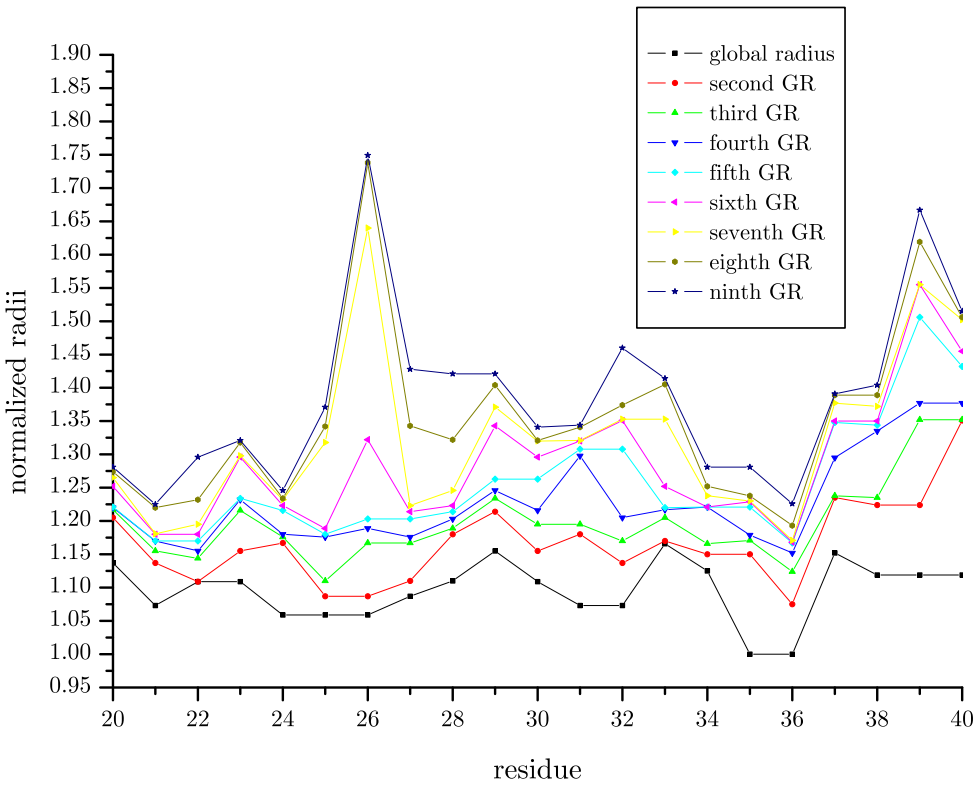


**Figure 4.5: Behavior of the nine smallest circle radii for each $C_\alpha$ atom.** The lowermost curve shows the global circular radii along the backbone. The curve just above shows the circles which have radius most next to those realizing the global radius, one could call them second global radii, and so on.

### 4.2.2 Dislocation of circles and spheres

An interesting feature available in the output files is the dislocation number ✠ which is given by

$$✠ = (k - j) + (j - i) \tag{4.1}$$

with $i, j, k$ in ascending order. It is some kind of measure of how un-local the global radius is.

For ✠ = 2 the global circle is a local circle. Figure 4.7 shows circular ✠ for BPTI.

For spheres the dislocation number becomes

$$✠ = (l - k) + (k - j) + (j - i) \tag{4.2}$$

with $i, j, k, l$ in ascending order. For ✠ = 3 the global sphere is a local sphere. Figure 4.8 shows spherical ✠ for BPTI.

Please note that circles are not that much scattered than spheres.

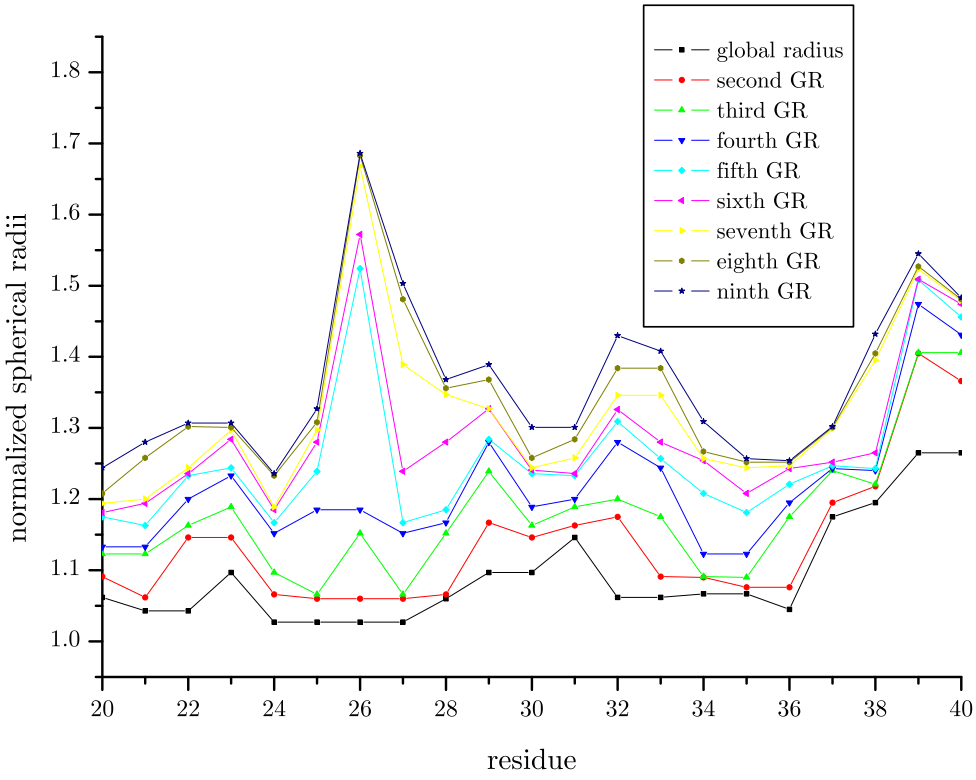


**Figure 4.6: Behavior of the nine smallest sphere radii for each $C_\alpha$ atom for BPTI.** The lowermost curve shows the global spherical radii along the backbone. Did you remark the strange gap between forth and fifth smallest sphere for residue 26?.

### 4.2.3 Optimality of protein shaping

Maritan and al. calculated the local circular f-ratio for the helix in the the chicken cartilage protein. They used the PDB entry 1AQ5 and published a plot in their publication[(2)]. A good way to test the developed algorithm was to reproduce this plot. Figure 4.9 shows the f-ratio calculated with **CALCULATOR**. The plots coincide and Maritan's and al. global f-ratio for residues 14 to 42 stating the optimality of helices is also found : $F = 1.01$.

The f-ratio is now also calculated for the whole backbone of proteins. Figure 4.10 shows the local f-ratio calculated along BPTI. The concept of optimality measured by the f-ratio has been extended to spherical radii. Figure 4.11 shows the spherical f-ratio for BPTI. Please note the stability $f \approx 1.0$ for the residues forming the $\alpha$-helix.

### 4.2.4 Thickness of proteins

Table 4.1 shows the circular and spherical thickness for some processed proteins. The absence of the spherical thickness for larger proteins is simply due to the fact that it took too much time to calculate them (see next section).

(2) Figure 3 on page 289 in [7]

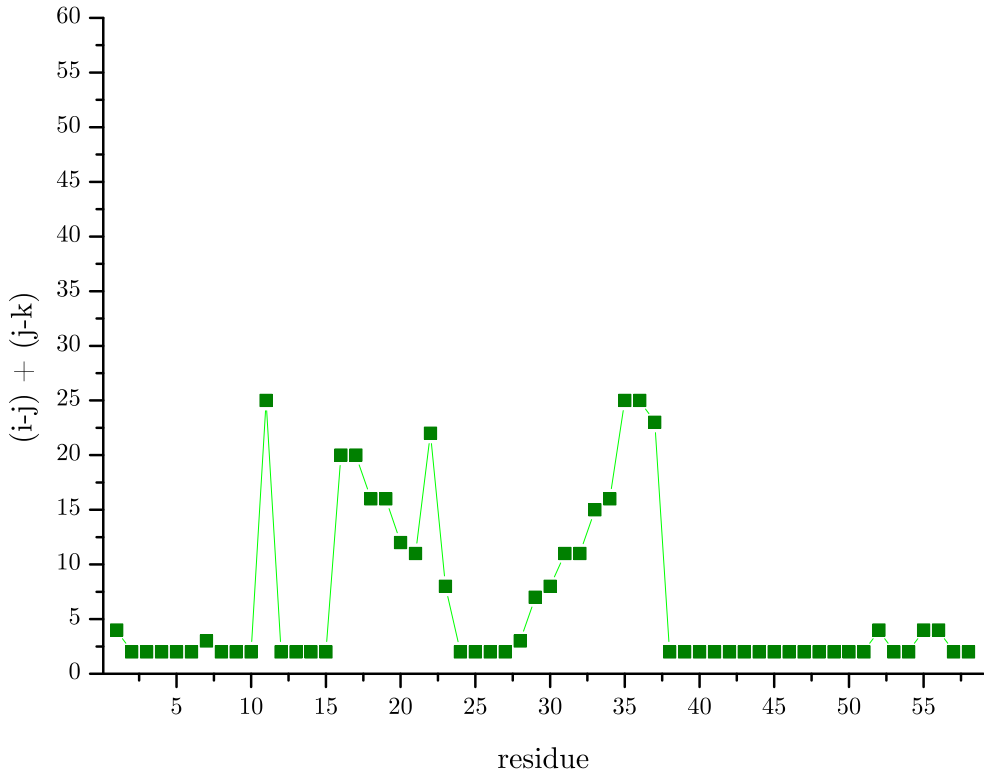


**Figure 4.7: Dislocation number ✠ for circles fitted through BPTI.** ✠ = 2 means that the global circle corresponds to the local circle.

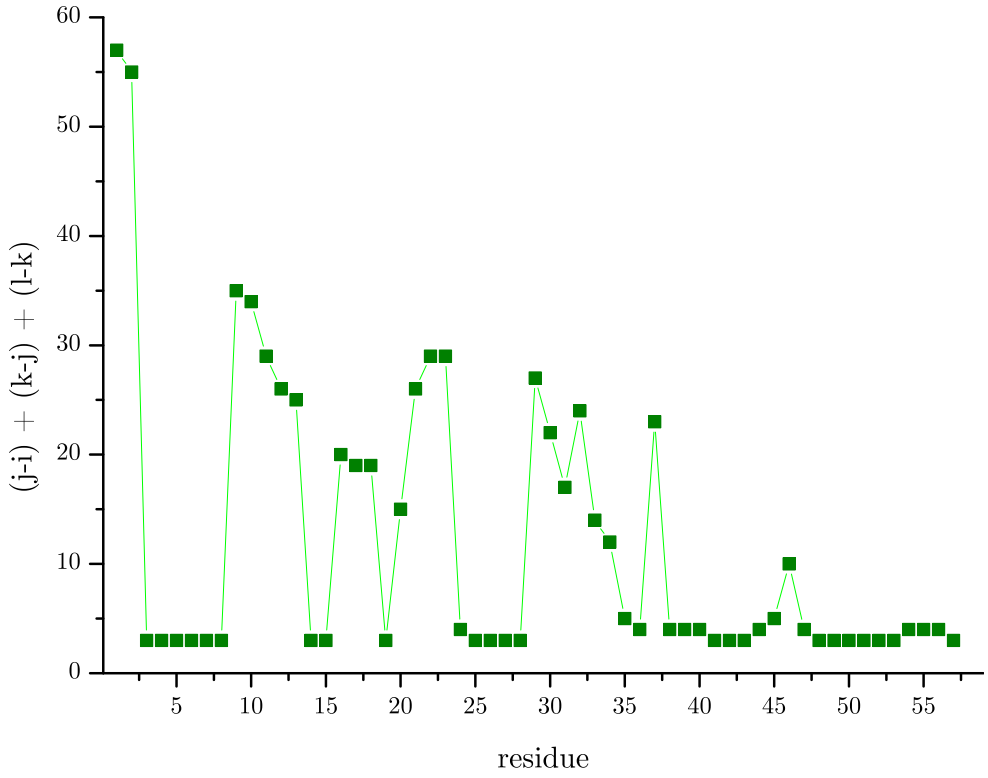


**Figure 4.8: Dislocation number ✠ for spheres fitted through BPTI.** ✠ = 3 means that the global sphere corresponds to the local sphere.

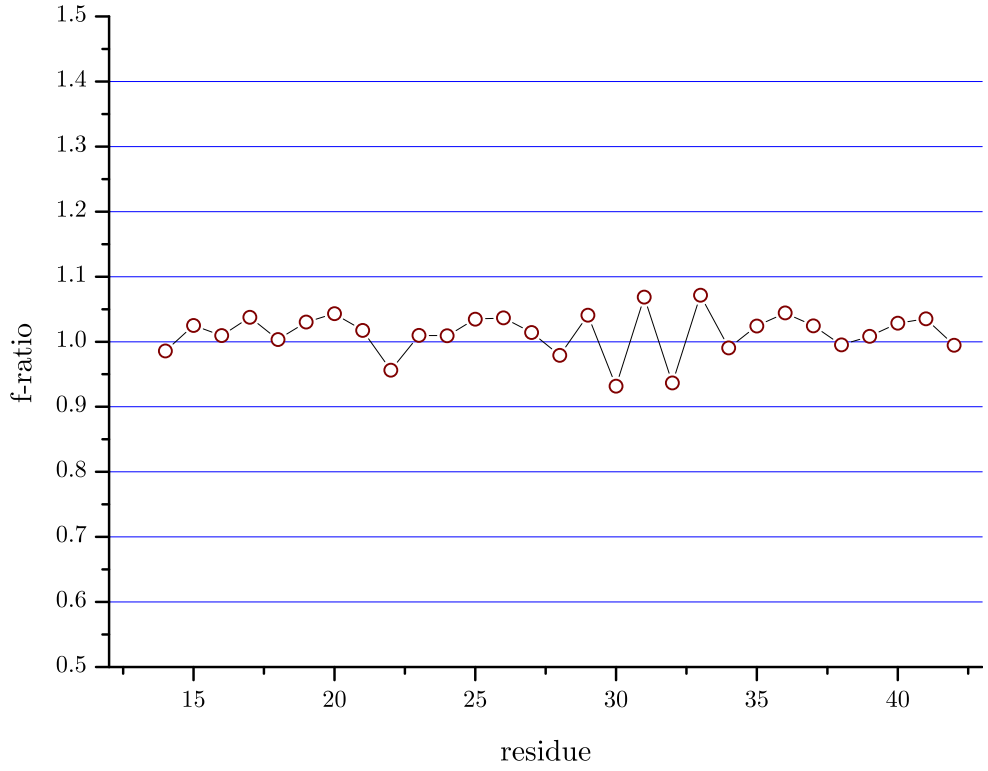


**Figure 4.9: Packing of $\alpha$-helix in the chicken cartilage matrix protein.** The local f-ratio is is plotted for residues 14 to 42 which shape the helix.

| Protein | # atoms | circular △ [Å] | spherical △ [Å] | Description |
|---|---|---|---|---|
| **4INS** | 21 | 2.60 | 2.83 | Insulin |
| **1PPB** | 36 | 2.50 | 2.80 | Thrombin |
| **1PTI** | 58 | 2.46 | 2.84 | Proteinase inhibitor |
| **7HVP** | 97 | 2.39 | 2.78 | HIV-1 Protease |
| **1IGT** | 214 | 2.30 | 2.79 | Antibodies |
| **1FBB** | 224 | 2.46 | 2.70 | Bacteriorhodopsin |
| **2PLV** | 288 | 2.38 | 2.89 | Poliovirus |
| **1YVN** | 372 | 2.51 | 2.75 | Actin |
| **1HTB** | 374 | 2.31 | 2.77 | Alcohol Dehydrogenase |
| **1ASZ** | 423 | 2.44 | 2.72 | Aminoacyl-tRNA Synthetases |
| **1N2C** | 478 | 2.35 | 2.76 | Nitrogenase |
| **1PRH** | 522 | 2.33 | - | Cyclooxygenase |
| **1ACC** | 665 | 2.37 | - | Anthrax Toxin |
| **1JB0** | 740 | 2.30 | - | Photosystem I |
| **1B7T** | 766 | 2.41 | - | Myosin |
| **6GPB** | 828 | 2.34 | - | Gycogen Phosphorylase |

**Table 4.1:** Circular and spherical thickness △ for some processed proteins. Note that the number of atoms corresponds to the number of extracted $C_\alpha$ atoms. The thickness doesn't depend on the length of the proteins backbone.

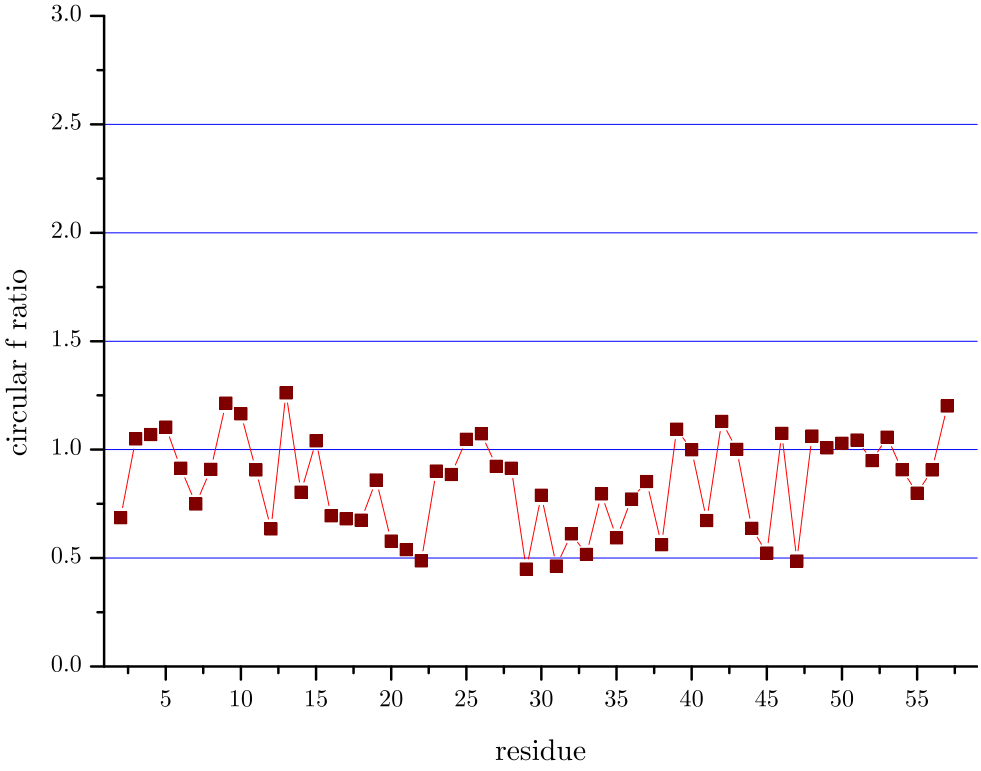


**Figure 4.10: Optimality of packing.** Circular f-ratio for BPTI. Two helices are present: a right-handed $3_{10}$ helix from residue 3 to 7, a right-handed $\alpha$-helix from residue 47 to 56.

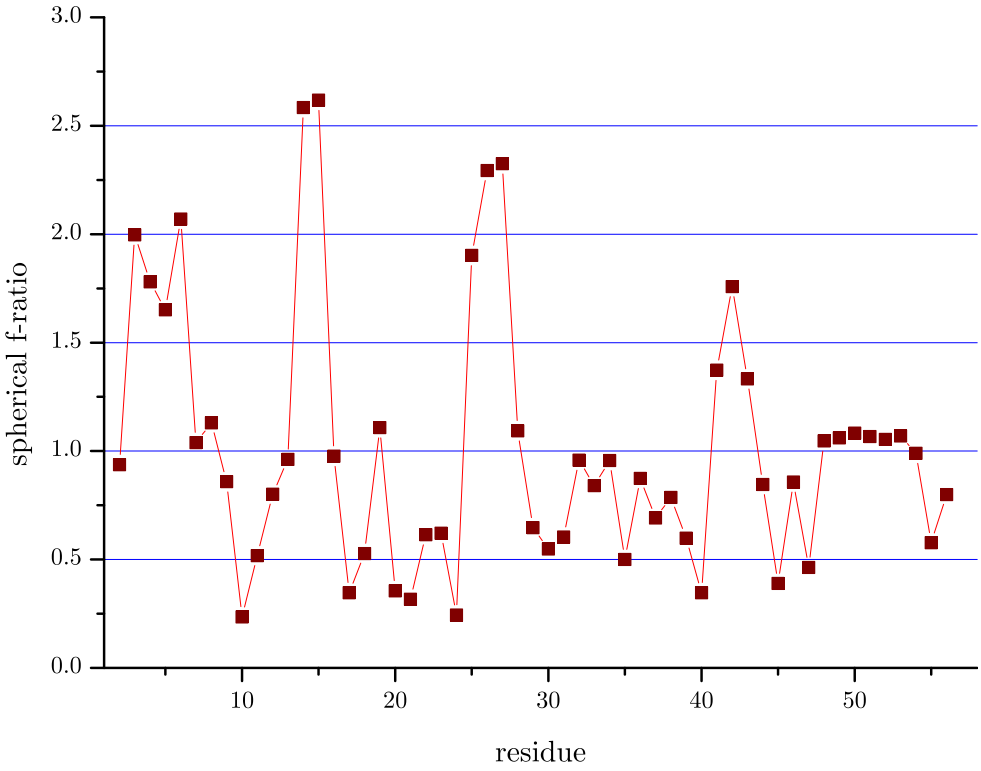


**Figure 4.11: Optimality of packing.** Spherical f-ratio for BPTI. Two helices are present: a right-handed $3_{10}$ helix from residue 3 to 7, a right-handed $\alpha$-helix from residue 47 to 56.

## 4.3 Runtime analysis

Large macromolecules require an astronomical number of radii to be calculated. Using the Fortran package StopWatch [2], the execution time for different proteins has been analyzed. The measurements have been done on a SUN 450 machine [(3)] running SUN's operating system Solaris 7.

Figure 4.12 shows the number of hours necessary to process several bigger proteins. The time dependence t is linear in the number of radii to calculate, i.e.

$$t = A \cdot \binom{n}{k} \tag{4.3}$$

where $n$ is the number of extracted atoms (i.e. the number of residues in the protein), $k = 3$ for circles or $k = 4$ for spheres.

For the 1N2C file (a complex of nitrogenase proteins containing 478 $C_\alpha$ atoms) the calculation of the spheres takes up to twelve (!) days, whilst for its circles, only 30 seconds were necessary. On LCVM$^2$'s computer the coefficient $A$ equals $4.95 \cdot 10^{-4}$ for spheres if $t$ is measured in seconds.

Please bear in mind that the Sun system is not very fast. For comparison purposes some files have also been processed on an AMD Athlon running at 1.2 GHz under Microsoft Windows XP Professional, where all the jobs were fulfilled more than 6 times faster.

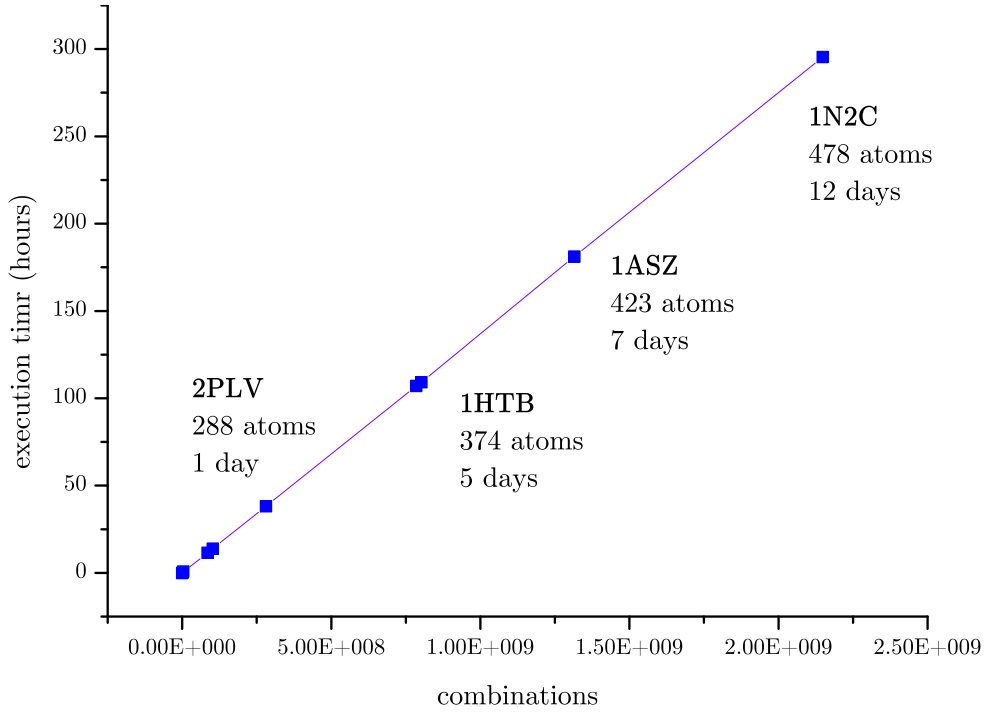


**Figure 4.12: Runtime analysis for sphere calculation.** Execution time is linear in the number of combinations of spheres to calculate. During the twelve days necessary to process PDB file 1N2C, 2'148'006'525 spheres were calculated.

(3) SUN Enterprise 4-CPU server

## 4.4 Conclusion

The algorithm has been tested and data for several proteins has been produced. Further data can easily be produced and the gathered data is open for interpretation.

The present implementation of **CALCULATOR** works fine and delivers results for smaller proteins within short time. The processing of longer macromolecules (what was not the aim of this project) as for example DNA, requires a precalculation to select neighboring atoms.

### 4.4.1 Acknowledgements

I would like to thank professor John H. Maddocks for accepting me as project student and for taking counsel with me during the project period. A big thank goes to Jana Smutny, doctoral candidate at LCVM$^2$. She was providing me with the formulae 2.1 and 2.7 [6] and assisted my test runs on LCVM$^2$ workstations helping me to make my algorithms compatible with the Sun Fortran 90 compiler. Thanks also to Mister Caussignac, for the help solving computer problems, hard disk crashes and launching remote jobs on Sun OS in the computer laboratory at the institute.

Prilly, August 2002

Christian Schlatter

# References


[1] Jacques Rappaz and Marco Picasso; **Introduction à l'analyse numérique**, Presses polytechniques et universitaires romandes, Lausanne 1998.

[2] William F. Mitchell, **StopWatch: A Module for Portable Measurement of Execution Time**, Fortran Journal Vol. 9, No. 1, 1997.

[3] Bruce Alberts, Dennis Bray et al.; **Essential Cell Biology**, An introduction to the molecular biology of the cell; Garland Publishing, New York 1998.

[4] J. C. Adams, W. S. Brainerd et al.; **The Fortran 95 Handbook**, MIT Press, Cambridge 1997.

[5] M. Posternak, Alfonso Baldereschi; **Introduction aux moyens informatiques**, EPFL 2000.

[6] Jana Smutny, **Radii and centres of spheres and circles**, EPFL 2001, internal document.

[7] Amos Maritan, Cristian Micheletti, Antonio Trovato and Jayanth R. Banavar; **Optimal shapes of compact strings**, Nature, vol. 406, pp. 287-290, London, july 2000.

[8] Andrzej Stasiak and John H. Maddocks; **Best packing in proteins and DNA**, Nature, vol. 406, pp 251-253, London, july 2000.

[9] Oscar Gonzalez and John H. Maddocks; **Global curvature, thickness, and the ideal shape of knots**, Proc. Natl. Acad. Sci. USA, vol. 96, pp. 4769-4773, april 1999.

[10] Zerrin Bagci, Robert L. Jernigan and Ivet Bahar, **Residue packing in proteins: Uniform distribution on a coarse-grained scale**, Journal of chemical physics, vol. 116, nr. 5, pp 2269-2276, february 2002.

[11] Zerrin Bagci, Robert L. Jernigan and Ivet Bahar, **Residue coordination in proteins conforms to the closest packing of spheres**, polymer, vol. 43, pp. 451-459, 2002.

[12] N. J. A. Sloane, **Kepler's concecture confirmed**, Nature, vol. 395, pp. 435-436, London, 1998.

[13] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I.N. Shindyalov, P.E. Bourne; **The Protein Data Bank**. Nucleic Acids Research, vol. 28, pp. 235-242, 2000

[14] The Protein Data Bank Contents Guide: **Atomic Coordinate Entry Format Description**, http://www.rcsb.org/pdb/docs/format/pdbguide2.2/guide2.2_frame.html

# Appendix A

# Sample data files

The following sections give illustrations of the proprietary $C_\alpha$ data formats. They show generated files for the PDB file 4INS, describing the insulin hormone from pigs. The $C_\alpha$ atoms coordinates of chain B from molecule I has been processed.

The three first lines of $C_\alpha$ files always identify what is inside the file. The release of the corresponding algorithms is listed there for compatibility reasons (the data format has been modified throughout several releases).

The fourth line states the atom type (CA for $C_\alpha$) to which the coordinates in the body of the file applies to. Line 5 gives the resolution used to determine the coordinates by experiment. If the value is 0.0 then the resolution information was not present in the PDB file. The entry column only serializes the atoms (the residue numbering sometimes doesn't start with 1 or a $C_\alpha$ atom is missing in a residue).

## A.1 $C_\alpha$ input data file

The $C_\alpha$ input data file is generated by **EXTRACTOR** by translating the PDB file (here 4INS.ent). If more chains are present they are identified by the chain entry. All coordinates are given in Å.

```
Atom cartesian coordinates from PDB
C-alpha-circle project. File created using Extractor.
Release© 1.31 (2002/04/28).  2002 EPFL Lausanne
Atom type :  CA
ResolutionÅ [] : 1.50

Entry     xÅ []     yÅ []     zÅ []  residue  chain

    1   -20.374     1.368     4.053        1      B
    2   -18.754     0.598     7.406        2      B
    3   -16.872     2.691     9.950        3      B
    4   -13.137     2.542     9.571        4      B
    5   -10.083     3.000    11.779        5      B
    6    -7.468     4.709     9.611        6      B
    7    -4.058     5.835    10.569        7      B
    8    -1.070     7.261     8.632        8      B
    9    -0.863     6.544     4.980        9      B
   10    -3.940     4.379     5.188       10      B
```

| | | | | | |
|---|---|---|---|---|---|
| 11 | −6.071 | 7.518 | 5.617 | 11 | B |
| 12 | −4.579 | 9.057 | 2.495 | 12 | B |
| 13 | −5.432 | 5.946 | 0.542 | 13 | B |
| 14 | −9.061 | 6.164 | 1.719 | 14 | B |
| 15 | −9.224 | 9.854 | 0.717 | 15 | B |
| 16 | −7.838 | 8.961 | −2.686 | 16 | B |
| 17 | −10.566 | 6.290 | −3.167 | 17 | B |
| 18 | −13.351 | 8.545 | −1.933 | 18 | B |
| 19 | −12.433 | 11.758 | −3.756 | 19 | B |
| 20 | −10.685 | 10.359 | −6.851 | 20 | B |
| 21 | −10.398 | 13.043 | −9.501 | 21 | B |

## A.2 $C_\alpha$ output data file (circles)

On line 6 and 7 of the file the calculated thickness in Åand the entry numbers of the atoms realizing this radius are given. The memory flag on line 8 corresponds to the number of smallest global radii which are listed - together with local radius, f-ratio - in the body of the file. The dislocation number ✠ is also given there (see section 4.2.2 for its definition).

The illustration below shows the human readable version of the file. The machine readable version contains the same information, but without formatting (as parenthesis and additional explanatorily text).

```
Circling of proteins
C–alpha–circle project. File created using Calculator.
Release© 1.61 (2002/07/07)  2002 EPFL Lausanne
Atom type : CA
ResolutionÅ [] : 1.50
Circular thicknessÅ [] : 2.60
Thickness combination :  16,  17 and  18
Memory :  2

Entry     Local   f–ratio  Dis     1. global radius      2. global radius

    1     0.000     0.000    2 (1.114;   1   2   3)  (1.135;   1   2   4)
    2     1.114     0.973    2 (1.053;   2   3   4)  (1.084;   2   4   5)
    3     1.053     1.042    2 (1.053;   2   3   4)  (1.076;   3   4   5)
    4     1.076     1.008    2 (1.053;   2   3   4)  (1.069;   4   5   6)
    5     1.069     1.014    2 (1.045;   5   6   7)  (1.069;   4   5   6)
    6     1.045     0.962    5 (1.005;   6  10  11)  (1.045;   5   6   7)
    7     1.166     0.945    2 (1.045;   5   6   7)  (1.090;   7   8   9)
    8     1.090     0.995    4 (1.084;   5   8   9)  (1.090;   7   8   9)
    9     1.682     0.644    4 (1.082;   6   9  10)  (1.084;   5   8   9)
   10     1.848     0.544    5 (1.005;   6  10  11)  (1.082;   6   9  10)
   11     1.964     0.512    5 (1.005;   6  10  11)  (1.264;  11  12  15)
   12     1.748     0.625    2 (1.016;  12  13  14)  (1.093;  12  14  15)
   13     1.016     1.058    2 (1.016;  12  13  14)  (1.033;  13  14  15)
   14     1.033     1.044    2 (1.016;  12  13  14)  (1.033;  13  14  15)
   15     1.041     1.032    2 (1.033;  13  14  15)  (1.035;  15  16  17)
   16     1.035     1.038    2 (1.000;  16  17  18)  (1.035;  15  16  17)
   17     1.000     1.092    2 (1.000;  16  17  18)  (1.022;  17  18  19)
   18     1.022     1.094    2 (1.000;  16  17  18)  (1.022;  17  18  19)
   19     1.106     1.005    2 (1.022;  17  18  19)  (1.106;  18  19  20)
   20     1.431     0.777    2 (1.106;  18  19  20)  (1.112;  17  19  20)
   21     0.000     0.000    2 (1.431;  19  20  21)  (1.655;  18  19  21)
```

## A.3 $C_\alpha$ output data file (spheres)

The same remarks as in the previous section are valid.

```
Sphering of proteins
C–alpha–circle project. File created using Calculator.
Release© 1.61 (2002/07/07).  2002 EPFL Lausanne
Atom type : CA
ResolutionÅ [] : 1.50
Spherical thicknessÅ [] : 2.83
Thickness combination :  13,  14,  15 and  16
Memory :  2
```

| Entry | Local | f–ratio | Dis | 1. global radius | | | | 2. global radius | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.000 | 0.000 | 4 (1.060; | 1 | 2 | 3 | 4) | (1.088; | 1 | 2 | 4 | 5) |
| 2 | 1.060 | 1.027 | 3 (1.009; | 2 | 3 | 4 | 5) | (1.060; | 1 | 2 | 3 | 4) |
| 3 | 1.009 | 1.082 | 3 (1.009; | 2 | 3 | 4 | 5) | (1.050; | 3 | 4 | 5 | 6) |
| 4 | 1.009 | 2.274 | 3 (1.009; | 2 | 3 | 4 | 5) | (1.027; | 4 | 5 | 6 | 7) |
| 5 | 1.027 | 1.048 | 3 (1.009; | 2 | 3 | 4 | 5) | (1.027; | 4 | 5 | 6 | 7) |
| 6 | 1.027 | 1.061 | 3 (1.027; | 4 | 5 | 6 | 7) | (1.050; | 3 | 4 | 5 | 6) |
| 7 | 1.070 | 1.006 | 3 (1.027; | 4 | 5 | 6 | 7) | (1.070; | 5 | 6 | 7 | 8) |
| 8 | 1.078 | 0.999 | 4 (1.070; | 5 | 6 | 7 | 8) | (1.077; | 4 | 5 | 7 | 8) |
| 9 | 1.997 | 0.546 | 4 (1.078; | 6 | 7 | 8 | 9) | (1.090; | 5 | 6 | 8 | 9) |
| 10 | 1.997 | 0.564 | 5 (1.126; | 5 | 6 | 9 | 10) | (1.177; | 6 | 7 | 9 | 10) |
| 11 | 3.690 | 0.339 | 6 (1.252; | 5 | 6 | 10 | 11) | (1.350; | 5 | 6 | 9 | 11) |
| 12 | 2.129 | 0.557 | 4 (1.017; | 12 | 13 | 14 | 15) | (1.186; | 12 | 13 | 15 | 16) |
| 13 | 1.017 | 1.100 | 3 (1.000; | 13 | 14 | 15 | 16) | (1.017; | 12 | 13 | 14 | 15) |
| 14 | 1.000 | 1.119 | 3 (1.000; | 13 | 14 | 15 | 16) | (1.014; | 14 | 15 | 16 | 17) |
| 15 | 1.000 | 1.186 | 3 (1.000; | 13 | 14 | 15 | 16) | (1.014; | 14 | 15 | 16 | 17) |
| 16 | 1.014 | 1.104 | 3 (1.000; | 13 | 14 | 15 | 16) | (1.014; | 14 | 15 | 16 | 17) |
| 17 | 1.037 | 1.080 | 3 (1.014; | 14 | 15 | 16 | 17) | (1.037; | 16 | 17 | 18 | 19) |
| 18 | 1.037 | 1.276 | 3 (1.037; | 16 | 17 | 18 | 19) | (1.046; | 15 | 16 | 17 | 18) |
| 19 | 1.052 | 1.319 | 3 (1.037; | 16 | 17 | 18 | 19) | (1.052; | 17 | 18 | 19 | 20) |
| 20 | 1.716 | 0.000 | 4 (1.052; | 17 | 18 | 19 | 20) | (1.387; | 16 | 18 | 19 | 20) |
| 21 | 0.000 | 0.000 | 4 (1.625; | 17 | 18 | 19 | 21) | (1.716; | 18 | 19 | 20 | 21) |

# Appendix B

# Sourcecode of the algorithm

## B.1 EXTRACTOR

```
! Projet de simulation énumrique
! student : Christian Schlatter, Physique 4e éanne EPFL
! professeur : John Maddocks, DMA EPFL
!
! ********************************************************************** 5
!
! This release meets the version 2.2 of the PDB format description guide
! which is available on
! http://www.rcsb.org/pdb/docs/format/pdbguide2.2/guide2.2_frame.html
! Future modification of the field lengths (as proposed for the atom 10
! entry will require a modification of this code.

! I'm using Visual Fortran 6.6A from Compaq (formerly Digital),
! fully Fortran 95 compatible compiler.
! It should also be runnnable on a fully Fortran 90 compiler. 15

! For release information see module release

module release

                                                                      20
  ! Release information
  character (LEN=25), parameter :: releaseinfo = 'Release 1.31 (2002/04/28)'

end module release

                                                                      25
module var

  ! Data type definitions
  type carbons

                                                                      30
    integer :: residue
    real, dimension(3) :: coordinates
    character(LEN=1) :: chainident

  end type carbons 35

end module var

module reading

                                                                      40
```

```
! Subroutines
implicit none

contains

subroutine helices (filename, resolution, alphacarbons, talkmode, atomtype)

  ! Extracting of alpha helices

  use var

  implicit none

  ! Variables
  integer                                          :: i, j, k, sequencenumber,
  integer                                          :: io, oldatom, newatom
  integer                                          :: numbercarbon,
  integer                                          :: beginsequence,
  integer                                          :: endsequence, numberhelix
  integer, allocatable, dimension(:,:)             :: chainseq
  ! lines : found helices
  ! begin at sequence end at sequence

  character (LEN=1)                                :: test3, ident
  character (LEN=2)                                :: helixtype
  character (LEN=4)                                :: atomname, seq, beginseq,
  character (LEN=4)                                :: endseq, test6, atomtype
  character (LEN=5)                                :: res, test4
  character (LEN=6)                                :: atom, helix
  character (LEN=8)                                :: x, y, z, test2
  character (LEN=9)                                :: test1
  character (LEN=10)                               :: test5
  character (LEN=22)                               :: test
  character (LEN=32)                               :: test32
  character (LEN=60)                               :: molecule
  character *(*), intent(in)                       :: filename
  character (LEN=1), allocatable, dimension(:,:)   :: chain
  ! lines : found helices
  ! rows : location, orientation

  real                                             :: xpos, ypos, zpos
  real, intent(out)                                :: resolution

  type(carbons), pointer, dimension(:)             :: alphacarbons, dummy

  logical                                          :: flag, talkmode


  OPEN (UNIT=1, FILE=filename, FORM='FORMATTED', STATUS='OLD')

    rewind 1

    ! Reading of the PDB file header line

    read (1, '(a10, a60)', ADVANCE='YES') test5, molecule

    if (talkmode) then

      write (*, '("   The following protein is going to be analysed:")')
```

```
  write (*, *)
  write (*, '("    ", a60)') molecule

end if

read (1, '(a10, a60)', ADVANCE='YES') test5, molecule

if (talkmode) then

  write (*, '("    ", a60)') molecule
  write (*, *)

end if

do

  ! Reading of the experimental data resolution value

  read (1, '(a22, a5)', ADVANCE='YES') test, res

  if (test == 'REMARK   2 RESOLUTION.') then

     exit

  endif

end do

if (res == 'NOT ') then

  if (talkmode) then

    write (*, '("    No resolution specified !!! ")')
    write (*, *)

  end if

else

  read(res, '(f5.0)') resolution

  if (talkmode) then

    write (*, '("    The structure data has a resolution of ", f4.2, "
       angstroems")') resolution
    write (*, *)

  end if

end if

! Determination of the number of present alpha helices

rewind 1

numberhelix = 0

do
```

```
      read(1, '(a6, a32, a2)', ADVANCE='YES', IOSTAT = io) helix, test32,
          helixtype

      if (io < 0) then                                                    160

        if ((talkmode) .and. (numberhelix == 0)) then

          write (*, '("   No alpha helices in this file !!")')
          write (*, *)                                                    165
          return

        end if

        exit                                                              170

      end if

      if ((helix == 'HELIX ') .and. ((helixtype == '6') .or. (helixtype
          == '1'))) then
                                                                          175
        ! Alpha helix types : 1 means right-handed; 6 means left-handed helix

        numberhelix = numberhelix + 1

      end if                                                              180

    end do

    allocate(chainseq(numberhelix,2))
    allocate(chain(numberhelix,2))                                        185

    ! Determination of location of the alpha helix

    rewind 1

                                                                          190
    i = 1
    chain = X
    flag = .false.

HEL : do                                                                  195

      if (i > numberhelix) then

        ! All alpha helices have been read

                                                                          200
        exit

      end if

      read(1, '(a6, a13, a1, a1, a4, a8, a4, a1, a2)', ADVANCE='YES', &   205
                IOSTAT = io) helix, test1, ident, test3, beginseq, &
                test2, endseq, test3, helixtype

      if ((helix == 'HELIX ') .and. ((helixtype == '6') .or. (helixtype
          == '1'))) then
                                                                          210
        chain(i,1) = ident
```

```
        read(beginseq, '(i4)') chainseq(i,1)  ! Helix starting sequence
           number
        read(endseq, '(i4)') chainseq(i,2)    ! Helix stopping sequence
           number

        flag = .true.

        if (helixtype == '1') then

           if (talkmode) then

             write (*, '("   Right-handed alpha helix found in chain ", a1)')
                 chain(i,1)

           end if

           chain(i,2) = 'R'
           i = i + 1

        else

           if (talkmode) then

             write (*, '("   Left-handed alpha helix foundin chain ", a1)')
                chain(i,1)

           end if

           chain(i,2) = 'L'
           i = i + 1

        endif

     endif

     if ((helix /= 'HELIX ') .and. (flag)) then

        exit HEL  ! End of helix specificator list reached

     endif

  end do HEL

  if (talkmode) then

     write (*, *)
     write (*, '("    ", i3, " helices found.")') i - 1
     write (*, *)

  end if

  j = 1

  do k = 1, i - 1

     numbercarbon = chainseq(k,2) - chainseq(k,1) + 1

     if (associated(alphacarbons)) then
```

```
        ! Extending the variable 'alphacarbons' for the new helix

        allocate(dummy(size(alphacarbons)))                                    270
        dummy = alphacarbons
        deallocate(alphacarbons)
        allocate (alphacarbons(size(dummy) + numbercarbon))
        alphacarbons (1 : j-1) = dummy (:)
        deallocate (dummy)                                                     275

      else

        ! Allocating the variable 'alphacarbons' for the first helix
                                                                               280
        allocate (alphacarbons(numbercarbon))

      end if

      if (talkmode) then                                                       285

        write (*, '("   There are ", i10, " carbon atoms located in the &
                  alpha helix of chain ", a1)') &
                  numbercarbon, chain(k,1)
        write (*, *)                                                           290
        write (*, '("   The carbon atoms have the following coordinates:")')
        write (*, *)
        write (*, '("       #    SEQ    x [A]    y [A]    z [A]")')
        write (*, *)
                                                                               295
      end if

      rewind 1
      sequencenumber = 0
      oldatom = -10                                                            300

ATM : do

        ! Reading of the coordinates of the atoms located in the alpha helices
                                                                               305
        read (1, '(a6, a5, a5, a5, a1, a4, a4, a8, a8, a8)', ADVANCE='YES') &
                  atom, test4, atomname, test5, ident, seq, test6, x, y, z

        if ((atom == 'ATOM  ') .and. (ident == chain(k,1))) then
                                                                               310
          read (test4, '(i5)') newatom

          if ((atomname == atomtype) .and. (newatom == oldatom + 1)) then

            ! Error in PDB file or strange mutation : The same atom found as   315
              neighbours

            oldatom = newatom
            cycle

          end if                                                               320

          if (atomname == atomtype) then

            oldatom = newatom
                                                                               325
```

```
        end if

        read(seq, '(i4)') sequencenumber
        read(x, '(f8.0)') xpos
        read(y, '(f8.0)') ypos
        read(z, '(f8.0)') zpos

      endif

      if ((atom == 'ATOM  ') .and. (atomname == atomtype) .and. &
          & (sequencenumber >= chainseq(k,1)) .and. (sequencenumber <=
              chainseq(k,2))) then

        ! Atom of type 'atomtype' found inside a helix

        if (talkmode) then
          write (*, '("  ", i5, i6, f8.3, f8.3, f8.3)') j, sequencenumber,
             xpos, ypos, zpos
        end if

        alphacarbons(j)%residue = sequencenumber
        alphacarbons(j)%coordinates(1) = xpos
        alphacarbons(j)%coordinates(2) = ypos
        alphacarbons(j)%coordinates(3) = zpos
        alphacarbons(j)%chainident = ident
        j = j + 1

      endif

      if (sequencenumber >= chainseq(k,2) .and. (atomname == atomtype)) then

        ! End of helix reached

        if (talkmode) then

          write (*, *)

        end if

        exit ATM

      endif

    end do ATM

  end do

  CLOSE (UNIT=1, STATUS='KEEP')

  deallocate(chainseq, chain)

end subroutine helices

subroutine backbones (filename, resolution, alphacarbons, talkmode, atomtype)

  ! Extracting the backbone of the protein

  use var
```

```
implicit none

! Variables                                                              385

integer, parameter                              :: maxhelices = 10
! maximum possible number of helices

integer                                         :: i, j, k, sequencenumber, io,    390
integer                                         :: oldatom, newatom
integer                                         :: oldsequencenumber
integer                                         :: numbercarbon, beginsequence,
integer                                         :: endsequence, residues
integer, dimension(maxhelices,2)                :: chainseq                        395
! lines : found helices
! begin at sequence end at sequence

character (LEN=1)                               :: ident
character (LEN=4)                               :: atomname, seq, test6, resi      400
character (LEN=4)                               :: atomtype
character (LEN=5)                               :: res, test4
character (LEN=6)                               :: atom, helix, seqres
character (LEN=7)                               :: test1
character (LEN=8)                               :: x, y, z                         405
character (LEN=10)                              :: test5
character (LEN=22)                              :: test
character (LEN=60)                              :: molecule
character *(*), intent(in)                      :: filename
character (LEN=1), dimension(maxhelices,2)      :: chain                           410
! lines : found helices
! rows : location, orientation

real                                            :: xpos, ypos, zpos
real, intent(out)                               :: resolution                      415

type(carbons), pointer, dimension(:)            :: alphacarbons, dummy

logical                                         :: flag, talkmode
                                                                                   420

OPEN (UNIT=1, FILE=filename, FORM='FORMATTED', STATUS='OLD')

  ! Reading of the PDB file header line
                                                                                   425
  rewind 1

  read (1, '(a10, a60)', ADVANCE='YES') test5, molecule

  if (talkmode) then                                                               430

    write (*, '("   The following protein is going to be analysed:")')
    write (*, *)
    write (*, '("   ", a60)') molecule
                                                                                   435
  end if

  read (1, '(a10, a60)', ADVANCE='YES') test5, molecule

  if (talkmode) then                                                               440
```

```
    write (*, '("    ", a60)') molecule
    write (*, *)

  end if

  do

    ! Reading ot the experimental data resolution value

    read (1, '(a22, a5)', ADVANCE='YES') test, res

    if (test == 'REMARK   2 RESOLUTION.') then

       exit

    endif

  end do

  if (res == 'NOT ') then

    if (talkmode) then

      write (*, '("   No resolution specified !!! ")')
      write (*, *)

    end if

  else

    read(res, '(f5.0)') resolution

    if (talkmode) then

      write (*, '("   The structure data has a resolution of ", f4.2, "
         angstroems")') resolution
      write (*, *)

    end if

  end if

  do

    ! Determination of the number of residues in the backbone

    read (1, '(a6, a7, a4)', ADVANCE='YES') seqres, test1, resi

    if (seqres == 'SEQRES') then

       exit

    endif

  end do

  read(resi, '(i4)') residues

  j = 1
```

```
      allocate (alphacarbons(residues))

      if (talkmode) then

        write (*, '("   There are ", i10, " residues in the backbone.")')
           residues
        write (*, *)
        write (*, '("   The carbon atoms have the following coordinates:")')
        write (*, *)
        write (*, '("       #   SEQ    x [A]    y [A]    z [A]")')
        write (*, *)

      end if

      rewind 1
      sequencenumber = 0
      oldatom = -10

ATM :   do

        oldsequencenumber = sequencenumber

        ! Reading of the coordinates of the atoms

        read (1, '(a6, a5, a5, a5, a1, a4, a4, a8, a8, a8)', ADVANCE='YES', &
                  IOSTAT = io) atom, test4, atomname, test5, ident, seq, &
                   test6, x, y, z

        if (atom == 'ATOM  ') then

          read (test4, '(i5)') newatom

          if ((atomname == atomtype) .and. (newatom == oldatom + 1)) then

            ! Error in PDB file or strange mutation : The same atom found as
               neighbours

            oldatom = newatom
            cycle

          end if

          if (atomname == atomtype) then

            oldatom = newatom

          end if

          read(seq, '(i4)') sequencenumber
          read(x, '(f8.0)') xpos
          read(y, '(f8.0)') ypos
          read(z, '(f8.0)') zpos

        endif

        if ((sequencenumber < oldsequencenumber) .or. (io < 0)) then

          ! If there were more than one atom of the chosen type per
```

```
    ! residue, the 'alphacarbons' variable has to be extended

    allocate(dummy(oldsequencenumber))
    dummy = alphacarbons(1:oldsequencenumber)
    deallocate(alphacarbons)
    allocate(alphacarbons(size(dummy)))
    alphacarbons = dummy
    deallocate (dummy)

    write(*,*)
    exit ATM

  end if

  if ((atom == 'ATOM  ') .and. (atomname == atomtype)) then

    ! An atom has been found

    if (j > residues) then

      ! Extending the variable 'alphacarbons' if the number of
      ! residues given in the PDB file header was not correct.

      allocate(dummy(size(alphacarbons)))
      dummy = alphacarbons
      deallocate(alphacarbons)
      allocate(alphacarbons(size(dummy) + 1))
      alphacarbons(1:j-1) = dummy(:)
      deallocate (dummy)

    end if

    alphacarbons(j)%residue = sequencenumber
    alphacarbons(j)%coordinates(1) = xpos
    alphacarbons(j)%coordinates(2) = ypos
    alphacarbons(j)%coordinates(3) = zpos
    alphacarbons(j)%chainident = ident

    if (talkmode) then

      write (*, '("  ", i5, i6, f8.3, f8.3, f8.3)') j, sequencenumber,
          xpos, ypos, zpos

    end if

    j = j + 1

  endif

  if (((sequencenumber == residues) .and. (atomname == atomtype)) .or. (
      atom == 'TER   ')) then

    ! All atoms have been processed.

    if (talkmode) then

      write (*,*)

    end if
```

```
            if (j < residues) then

              ! Correction if number of found alphacarbons is smaller
              ! than the size of its allocated variable

              allocate(dummy(j-1))
              dummy = alphacarbons(1:j-1)
              deallocate(alphacarbons)
              allocate(alphacarbons(size(dummy)))
              alphacarbons(:) = dummy(:)
              deallocate (dummy)

            end if

            exit ATM

          endif

        end do ATM

    CLOSE (UNIT=1, STATUS='KEEP')

  end subroutine backbones

end module reading

! ***********************************************************************

program extract

  ! Main program

  use reading
  use var
  use release

  implicit none

  ! Variables
  character (LEN=12)                          :: export, dummy
  character (LEN=3)                           :: subfolder, input
  character (LEN=4)                           :: extension, atomtype
  character (LEN=6)                           :: exportfolder
  character (LEN=12), allocatable, dimension(:) :: filename
  character, dimension(:), pointer            :: name

  real                                        :: resolution, min
  real(8)                                     :: initial, final
  real, allocatable, dimension(:,:)           :: allalphacarbons, oldcarbons
  real, pointer, dimension(:,:)               :: alphacircles
  real, allocatable, dimension(:)             :: minradius

  type(carbons), pointer, dimension(:)        :: alphacarbons

  integer                                     :: i, j, k, numberfiles, io,
  integer                                     :: position, length
  integer                                     :: what, resourceflag
```

```
  integer , allocatable , dimension(:,:)          :: triple

  logical                                         :: talkmode
  logical , dimension(:) , allocatable            :: goodfiles

!  resourceflag = 0       ! either 0 : a single PDB-file specified by filename
                          ! or     1 : a bunch of PDB-files listed in Files.ini
                          !            is loaded.

  subfolder = 'PDB'       ! name of the subdirectory where the PDB-resource
                          ! files are stored in (with extension .ent or .pdb)

  exportfolder = 'source' ! name of the subfolder for ASCII export files

  talkmode = .true.       ! if true, detailed comments are printed on screen

  if (talkmode) then

    write (*, *)
    write (*, '(" Computational physics project")')
    write (*, '(" Christian Schlatter, student of physics, 4th year")')
    write (*, *) releaseinfo
    write (*, '("
        ************************************************************")')
    write (*, *)

  end if

  ! PDB input file collection

  write (*, '(" PDB File coordinates extractor")')
  write (*, *)
  write (*, '(" How many PDB files would you like to process ?")')
  write (*, *)
  write (*, '(" -> Type 1 for a single file or")')
  write (*, '(" -> Type 2 for a list of files provided in Files.ini")')
  write (*, *)
  write (*, '(" Please choose your option now : ")', advance = 'no')

  read (*,*) resourceflag

  write (*, *)

  if (resourceflag == 2) then    ! Batch mode using Files.ini

    write (*, '(" Opening Files.ini... -> ")', advance = 'no')

    open (UNIT=1, FILE='Files.ini', FORM='FORMATTED', STATUS='OLD')

    rewind 1

    numberfiles = 0

    do

      ! Determination of the number of PDB files

      read (1, '(a12)', IOSTAT = io) dummy
```

```
      if (io == 0) then

        numberfiles = numberfiles + 1

      else                                                              735

        exit

      end if

                                                                        740
    end do

    allocate (filename(numberfiles))

    rewind 1                                                            745

    do i = 1, numberfiles

      ! Determination of the PDB file names

                                                                        750
      read (1, '(a12)') filename(i)

    end do

    close (UNIT=1, STATUS='KEEP')                                       755

    write (*, '(i10, " files to process")') numberfiles
    write (*,*)

  else                                                                  760

    allocate (filename(1))

    ! User provided PDB file name

                                                                        765
    write (*, '(" Please provide the filename of the PDB file : ")', advance = '
       no')
    read (*, *) filename(1)
    write (*, *)

    numberfiles = 1                                                     770

  endif

  allocate (goodfiles(numberfiles))
  goodfiles = .true.                                                    775

  ! Atom type to be extracted

  input = '   '

                                                                        780
  write (*, *)
  write (*, '(" Which type of atoms would you like to extract ?")')
  write (*, *)
  write (*, '(" Example : For alpha carbons type the two characters CA")')
  write (*, *)                                                          785
  write (*, '(" Please choose your atom now : ")', advance = 'no')

  read (*, *) input
```

```
atomtype(1:1) = " "

do i = 1, 3

  atomtype(i+1:i+1) = input(i:i)

end do


! Special tertiary configuration selection

write (*, *)
write (*, '(" What kind of information would you like to extract ?")')
write (*, *)
write (*, '(" -> Type 1 for coordinates of atoms located in alpha helices.")')
write (*, '(" -> Type 2 for coordinates of atoms of the whole backbone of the
   protein.")')
write (*, *)
write (*, '(" Please choose your option now : ")', advance = 'no')

read (*, *) what

do i = 1, numberfiles

  ! Processing file by file

  if (talkmode) then

    write (*, *)
    write (*, *)
    write (*, '(" Extracting structure information from file ", a12)')
       filename(i)
    write (*, '(" ******************************************************")')
    write (*, *)
    write (*,'(" Processing now file ", a12)') filename(i)
    write (*, *)

  end if

  position = scan(filename(i),'.')

  select case (what)

    case (1)

      ! Extracting helix subsequences only

      call helices(subfolder//'/'//filename(i), resolution, alphacarbons,
         talkmode, atomtype)

      filename(i)(position+1:position+1) = 'h'
      filename(i)(position+2:position+2) = 'e'
      filename(i)(position+3:position+3) = 'l'

    case (2)

      ! Extracting the whole backbone of the protein
```

```
      call backbones(subfolder//'/'//filename(i), resolution, alphacarbons,
         talkmode, atomtype)

      filename(i)(position+1:position+1) = 'b'
      filename(i)(position+2:position+2) = 'c'
      filename(i)(position+3:position+3) = 'k'

    case default

  end select

  if (.not.(associated(alphacarbons))) then

    ! This may occur when no alpha helices were present in the selected
       protein

    if (talkmode) then

      write (*, '(" This file will be ignored. Please check if it is
         applicable.")')
      write (*, *)
      pause

    else

      write (*, '("File ", a8, " has not been created. No helices present.")')
           filename(i)

    end if

    goodfiles(i) = .false.

    cycle

  end if

  ! Saving the gathered informations to disk

  write (*, *)
  write (*, '(" Exporting C-alpha coordinates to file ", a12)') filename(i)

  if (talkmode) then

    write (*, '(" ************************************************")')
    write (*, *)

  end if

  open (UNIT=2, FILE=exportfolder//'/'//filename(i), FORM='FORMATTED', STATUS
     ='REPLACE')

  ! Definition of the C-alpha circle input file header

  write(2, '("Atom cartesian coordinates from PDB")')
  write(2, '("C-alpha-circle project. File created using Extractor.")')
  write(2, '(a25)', advance='no') releaseinfo
  write(2, '("©.  2002 EPFL Lausanne")')
  write(2, '("Atom type : ", a4)') atomtype
  write(2, '("ResolutionÅ [] : ", f4.2)') resolution
```

```
    write(2, *)
    write(2, '("Entry     xÅ []     yÅ []     zÅ []  residue  chain")')
    write(2, *)

    do j = 1, size(alphacarbons)
      write (2, '(i5, 3f10.3, i9, a7)') j, alphacarbons(j)%coordinates(1), &
                                           alphacarbons(j)%coordinates(2), &
                                           alphacarbons(j)%coordinates(3), &
                                           alphacarbons(j)%residue, &
                                           alphacarbons(j)%chainident
    end do

    close (UNIT=2, STATUS='KEEP')

    deallocate (alphacarbons)

  end do

  ! Protocolling of which files were processed sucessfully

  open (UNIT=3, FILE = 'goodfiles.txt', FORM='FORMATTED', STATUS='REPLACE')

  do i = 1, numberfiles

    if (goodfiles(i)) then

      write (3, *) filename(i)

    end if

  end do

  close (UNIT=3, STATUS='KEEP')

  deallocate (filename)

  write(*, *)
  write(*,'(" DONE ! ")')
  write(*, *)

end program extract
```

# B.2 CALCULATOR

```
! Projet de simulation énumrique
! student : Christian Schlatter, SB, section physique, EPFL
! professeur : John Maddocks, SB, section émathmatique, EPFL

! This release meets the version 2.2 of the PDB format description guide
! which is available on
! http://www.rcsb.org/pdb/docs/format/pdbguide2.2/guide2.2_frame.html
! Future modifications of the field lengths (as proposed for the atom entry)
! will require a modification of this code.

module parameters

  ! User adjustable parameter module
  integer, parameter :: memory    =    12   ! Memory limit of smallest radii
  integer, parameter :: limit     =  200    ! Upper limit for distribution
     vector

  logical, parameter :: talkmode  = .false. ! Comments on screen
  logical, parameter :: secondrun = .true.  ! Statistics run to classify
     distribution of radii

  character (LEN=25), parameter :: releaseinfo = 'Release 1.61 (2002/07/07)'

end module parameters

! ********************************************************************

module var

  ! Data type definitions
  type carbons

    integer             :: residue
    real, dimension(3)  :: coordinates
    character (LEN=1)   :: chainident

  end type carbons

end module var

! ********************************************************************

module mathe

  ! Mathematical routines
  contains

  function crossproduct (a, b)

    ! Calculates the cross product of two vectors
    implicit none

    real, dimension(3), intent(in) :: a, b
    real, dimension(3) :: crossproduct

    crossproduct(1) = a(2)*b(3) - a(3)*b(2)
```

```
    crossproduct(2) = a(3)*b(1) - a(1)*b(3)
    crossproduct(3) = a(1)*b(2) - a(2)*b(1)

  end function crossproduct

  function determinant (A)

    ! Calculates the determinant of a matrix using
    ! the partial pivoting Gaussian elimination scheme.

    implicit none

    real, pointer, dimension(:,:) :: A
    real, allocatable, dimension(:,:) :: B
    real, allocatable, dimension(:) :: C

    integer, allocatable, dimension(:) :: indexer
    integer :: i, j, k, n, tempindex, msgn

    real :: scaler, pivot, pivoti, pivotj
    real :: determinant

    n = size(A(:,1))

    allocate (B(n,n))
    B = A

    ! index records the pivoting order

    allocate (indexer(n))

    ! A is the original matrix in the input. B is A transformed
    ! into the matrix plus the pivoting element ratios below
    ! the diagonal in the output.

    do i = 1, n

      indexer(i) = i

    end do

    ! Find the rescaling factors, one from each row

    allocate (C(n))

    do i = 1, n

      scaler = 0.0

      do j = 1, n

        scaler = amax1(scaler,abs(B(i,j)))

      end do

      C(i) = scaler

    end do
```

```
! Search the pivoting (largest) element from each column
do j = 1, n-1
  pivot = 0.0
  do i = j, n
    pivoti = abs(B(indexer(i),j)) / C(indexer(i))
    if (pivoti.GT.pivot) then
      pivot = pivoti
      k = i
    endif
  end do
  ! Interchange the rows via indexer(n) to record pivoting order
  tempindex = indexer(j)
  indexer(j) = indexer(k)
  indexer(k) = tempindex
  do i = j+1, n
    pivotj  = B(indexer(i),j)/B(indexer(j),j)
    ! Record pivoting ratios below the diagonal
    B(indexer(i),j) = pivotj
    ! Modify other elements accordingly
    do k = j+1, n
      B(indexer(i),k) = B(indexer(i),k)-pivotj*B(indexer(j),k)
    end do
  end do
end do
determinant = 1.0
do i = 1, n
  determinant = determinant*B(indexer(i),i)
end do
msgn = 1
do i = 1, n
  do while (i.NE.indexer(i))
```

```
        msgn = -msgn
        j = indexer(i)
        indexer(i) = indexer(j)
        indexer(j) = j

      end do

    end do

    determinant = msgn*determinant

    deallocate(indexer)
    deallocate(B)
    deallocate(C)

  end function determinant

end module mathe

! ************************************************************************

module distance

  implicit none

  contains

  subroutine circles (alphacarbons, f, g, truelocal, globalradius, &
                      globallocation, smallestradius, smallestlocation, &
                      distribution, thickness)

    ! fits circles through the macromolecule
    use var
    use mathe
    use parameters

    implicit none

    type(carbons), pointer, dimension(:) :: alphacarbons

    real                                 :: r2, A, B, C, thickness
    real, dimension (3)                  :: D, first, second, third
    real, dimension (3)                  :: crossed, radii
    real, pointer, dimension(:)          :: f, g, localradius, nonadjacentradius
    real, pointer, dimension(:)          :: globalradius, nonlocalradius
    real, pointer, dimension(:)          :: partlyadjacentradius, truelocal
    real, pointer, dimension(:,:)        :: smallestradius

    integer, pointer, dimension(:)       :: firstlocal
    integer, pointer, dimension(:,:)     :: nonadjacentlocation
    integer, pointer, dimension(:,:)     :: partlyadjacentlocation
    integer, pointer, dimension(:,:)     :: globallocation
    integer, pointer, dimension(:,:,:)   :: smallestlocation
    integer, dimension(3)                :: combination
    integer                              :: numbercarbon, i, j, k, l, m, q, r
    integer                              :: mem, index, pos, location
    integer, dimension(1)                :: selector
    ! due to compatibility issues with Fortran 90
```

```
integer(4), dimension(limit)          :: distribution

numbercarbon = size (alphacarbons)
                                                                    235
allocate (localradius(numbercarbon))
allocate (truelocal(numbercarbon))
allocate (nonadjacentradius(numbercarbon))
allocate (firstlocal(numbercarbon))
allocate (nonadjacentlocation(numbercarbon,3))                      240
allocate (partlyadjacentradius(numbercarbon))
allocate (partlyadjacentlocation(numbercarbon,3))

if (memory .gt. numbercarbon) then
                                                                    245
  ! I can't remember more than present atoms !
  mem = numbercarbon

else

                                                                    250
  mem = memory

end if

localradius = 0.0                                                   255
truelocal = 0.0
nonadjacentradius = 0.0
partlyadjacentradius = 0.0
smallestradius = 0.0
index = 0                                                           260

if(thickness .ne. 0.0) then

  ! Statistical classification of the radii possible now
  write(*,'(" Second run. Statistics running...")')                 265
  write(*, *)
  write(*, *)

  distribution = 0.0
                                                                    270
end if

! Checking all possible combination of 3 points
do i = 1 , numbercarbon - 2
                                                                    275
  do j = i + 1 , numbercarbon - 1

    do k = j + 1 , numbercarbon

      ! colinearity check -> ((b-a)\times(c-a))^2 = 0               280

      first = alphacarbons(i)%coordinates
      second = alphacarbons(j)%coordinates
      third = alphacarbons(k)%coordinates
                                                                    285
      crossed = crossproduct((second-first), (third-first))

      ! radii calculation

      if (dot_product(crossed, crossed) == 0) then                  290
```

```
    r2 = 0.0

  else

    ! Radius formula when three points are given
    D(1) = (sum((alphacarbons(i)%coordinates - &
                 alphacarbons(j)%coordinates)**2))
    D(2) = (sum((alphacarbons(i)%coordinates - &
                 alphacarbons(k)%coordinates)**2))
    D(3) = (sum((alphacarbons(j)%coordinates - &
                 alphacarbons(k)%coordinates)**2))

    A = sum(D**2)
    B = 2*((D(1)*D(2))+(D(1)*D(3))+(D(2)*D(3)))

    if((A-B) == 0) then

      C = -1  ! The points are colinear

    else

      C = (- product(D) / (A - B))

    end if

    if (C < 0) then

      ! This may also occur if the points are almost colinear,
      ! so that the radius of the circle is quite large

      write(*,'("   Problem with alpha carbons : ", i10, ", ", i10, ", "
          , i10)') i, j, k
      write(*,'("   Colinearity check dot product equals: ", f10.3)')
         dot_product(crossed, crossed)
      write(*,*)

      r2 = 0.0

    else

      r2 = sqrt (C)

      if(thickness .ne. 0.0) then

        ! This is the second run :
        ! Classification of the found radius

        pos = floor(((r2/thickness) - 1) * 100) + 1

        if (pos .gt. limit) then

          ! Found radius is bigger than the given limit

          pos = limit

        end if

        distribution(pos) = distribution(pos) + 1
```

```
end if

combination(1) = i    ! This is due to Fortran 90 compatibility
combination(2) = j
combination(3) = k

do m = 1, 3

  ! Completing for all three points

  l = combination(m)

  do q = 1, mem

    if (r2 .lt. smallestradius(l,q)) then

      ! A smaller radius has been found

      do r = mem, q+1, -1

        ! Shifting the radius vector for insertion of the
        ! new radius

        smallestradius(l,r) = smallestradius(l,r-1)
        smallestlocation(l,r,1) = smallestlocation(l,r-1,1)
        smallestlocation(l,r,2) = smallestlocation(l,r-1,2)
        smallestlocation(l,r,3) = smallestlocation(l,r-1,3)

      end do

      ! Saving the new radius

      smallestradius(l,q) = r2
      smallestlocation(l,q,1) = i
      smallestlocation(l,q,2) = j
      smallestlocation(l,q,3) = k

      exit

    end if

    if (smallestradius(l,q) == 0.0) then

      ! This is the first radius for the given point

      smallestradius(l,q) = r2
      smallestlocation(l,q,1) = i
      smallestlocation(l,q,2) = j
      smallestlocation(l,q,3) = k

      exit

    end if

  end do

end do
```

```
      end if

    end if

    if (((k-j) + (j-i)) == 2) then

      ! The found radius is local

      combination(1) = i
      combination(2) = j
      combination(3) = k

      if ((truelocal(j) == 0) .or. (truelocal(j) > r2)) then

        ! Only for the middle point j (i < j < k) this is a true
        ! local radius

        truelocal(j) = r2

      end if

      do m = 1, 3

        l = combination(m)

        ! Completion for all three points

        if ((localradius(l) == 0) .or. (localradius(l) > r2)) then

          localradius(l) = r2
          firstlocal(l) = i     ! Indicates the first point

        end if

      end do

    elseif (((k-j) .gt. 1) .and. ((j-i) .gt. 1)) then

      ! None of the atoms are neighbors

      combination(1) = i
      combination(2) = j
      combination(3) = k

      do m = 1, 3

        l = combination(m)

        if ((nonadjacentradius(l) == 0) .or. (nonadjacentradius(l) > r2))
           then

          nonadjacentradius(l) = r2
          nonadjacentlocation(l,1) = i
          nonadjacentlocation(l,2) = j
          nonadjacentlocation(l,3) = k

        end if

      end do
```

```
      else

        ! Two of the three atoms are direct neighbors

        combination(1) = i
        combination(2) = j
        combination(3) = k

        do m = 1, 3

          l = combination(m)

          if ((((k-j) .ne. 1) .and. ((j-i) .ne. 1)) .or. (((k-j) == 1) .and
              . ((j-i) == 1))) then

            write (*, '("Error !!!")')

          end if

          if ((partlyadjacentradius(l) == 0) .or. (partlyadjacentradius(l)
               > r2)) then

            partlyadjacentradius(l) = r2
            partlyadjacentlocation(l,1) = i
            partlyadjacentlocation(l,2) = j
            partlyadjacentlocation(l,3) = k

          end if

        end do

      end if

      index = index + 1

    end do

  end do

end do


if (talkmode) then

  write (*, '(" ", i10, " circles have been calculated")') index
  write (*, *)

end if

! Finalizing

allocate (globalradius(numbercarbon))
allocate (globallocation(numbercarbon,3))
allocate (nonlocalradius(numbercarbon))

do i = 1, numbercarbon

  ! Which class of radii provides the smallest radius?
```

```
    radii(1) = localradius(i)
    radii(2) = nonadjacentradius(i)
    radii(3) = partlyadjacentradius(i)

    selector = minloc(radii, mask = radii .ne. 0.0)
    location = selector(1)

    ! Global radius lists the smallest circle for each point
    globalradius(i) = radii(location)

    select case (location)

      ! Indexes of the points realizing the smallest circle
      case (1)

        globallocation(i,1) = firstlocal(i)
        globallocation(i,2) = firstlocal(i) + 1
        globallocation(i,3) = firstlocal(i) + 2

      case (2)

        globallocation(i,:) = nonadjacentlocation(i,:)

      case (3)

        globallocation(i,:) = partlyadjacentlocation(i,:)

    end select

    ! Doing the same but by ignoring the local radii

    radii(1) = 0.0
    radii(2) = nonadjacentradius(i)
    radii(3) = partlyadjacentradius(i)

    selector = minloc(radii, mask = radii .ne. 0.0)
    location = selector(1)

    nonlocalradius(i) = radii(location)

  end do

  allocate (f(numbercarbon))
  allocate (g(numbercarbon))

  f = 0.0
  g = 0.0

  do i = 2, numbercarbon - 1

    ! Calculation of the f and g factor

    f(i) = nonlocalradius(i)/truelocal(i)
    g(i) = globalradius(i)/truelocal(i)

  end do

  deallocate (nonadjacentradius, nonadjacentlocation)
```

```
  deallocate (partlyadjacentradius, partlyadjacentlocation)
  deallocate (nonlocalradius, firstlocal, localradius)

end subroutine circles

! ****************************************************************

subroutine spheres (alphacarbons, f, g, localradius, globalradius, &
                    globallocation, smallestradius, smallestlocation, &
                    distribution, thickness)

  ! fits spheres through the macromolecule
  use var
  use mathe
  use parameters

  implicit none

  type(carbons), pointer, dimension(:) :: alphacarbons

  real                                 :: A, B, maxradius, rquadrat
  real                                 :: r3, thickness, detn, detz
  real                                 :: prevtime, time
  real, dimension(4,4)                 :: D, E, H
  real, dimension(3)                   :: C, first, second, third, fourth
  real, dimension(2)                   :: radii
  real, dimension(4)                   :: minima
  real, pointer, dimension(:)          :: f, g, localradius, nonlocalradius
  real, pointer, dimension(:)          :: globalradius
  real, pointer, dimension(:,:)        :: smallestradius
  real, pointer, dimension(:,:)        :: coplanarity, N, Z

  integer                              :: numbercarbon, i, j, k, l, m, o
  integer                              :: q, r, s, index, error
  integer                              :: location, pos, mem
  integer, pointer, dimension(:)       :: firstlocal
  integer, pointer, dimension(:,:)     :: nonlocallocation, globallocation
  integer, pointer, dimension(:,:,:)   :: smallestlocation
  integer, dimension(4)                :: combination
  integer, dimension(1)                :: selector
  integer(4), dimension(limit)         :: distribution

  numbercarbon = size (alphacarbons)

  allocate (localradius(numbercarbon))
  allocate (firstlocal(numbercarbon))
  allocate (nonlocalradius(numbercarbon))
  allocate (nonlocallocation(numbercarbon,4))

  if (memory .gt. numbercarbon) then

    ! I can't remember more than present atoms !
    mem = numbercarbon

  else

    mem = memory

  end if
```

```
640
index = 0
error = 0
localradius = 0.0
nonlocalradius = 0.0
smallestradius = 0.0                                                    645

allocate(coplanarity(4,4))
allocate(N(4,4))
allocate(Z(5,5))

                                                                        650
if(thickness .ne. 0.0) then

  ! Statistical classification of the radii possible now
  write(*,'(" Second run. Statistics running...")')
  write(*, *)                                                           655
  write(*, *)

  distribution = 0.0

end if                                                                  660

! Checking all possible combination of 4 points
do i = 1 , numbercarbon - 3

  do j = i + 1 , numbercarbon - 2                                       665

    do k = j + 1 , numbercarbon - 1

      do l = k + 1, numbercarbon

                                                                        670
        ! coplanarity check -> (d-a).((b-a)\times(c-a)) = 0

        coplanarity = reshape((/alphacarbons(i)%coordinates, 1.0, &
                                alphacarbons(j)%coordinates, 1.0, &
                                alphacarbons(k)%coordinates, 1.0, &      675
                                alphacarbons(l)%coordinates, 1.0/), (/4, 4/)
                                   )

        ! radii calculation

        D(1:4,1:4) = 0.0                                                680
        D(1,2) = (sum((alphacarbons(i)%coordinates - &
                       alphacarbons(j)%coordinates)**2))
        D(1,3) = (sum((alphacarbons(i)%coordinates - &
                       alphacarbons(k)%coordinates)**2))
        D(1,4) = (sum((alphacarbons(i)%coordinates - &                  685
                       alphacarbons(l)%coordinates)**2))
        D(2,3) = (sum((alphacarbons(j)%coordinates - &
                       alphacarbons(k)%coordinates)**2))
        D(2,4) = (sum((alphacarbons(j)%coordinates - &
                       alphacarbons(l)%coordinates)**2))                 690
        D(3,4) = (sum((alphacarbons(k)%coordinates - &
                       alphacarbons(l)%coordinates)**2))
        N(1:4,1:4) = 0.0
        Z(1:5,1:5) = 0.0

                                                                        695
        N = D + transpose(D)
        Z(1:4,1:4) = N
```

```
Z(1:4,5) = 1.0
Z(5,1:4) = 1.0

detn = determinant(N)
detz = determinant(Z)

if (detz == 0) then

  ! The points are coplanar
  rquadrat = -1

else

  rquadrat = (- detn / (2 * detz))

end if

if ((abs(determinant(coplanarity)) == 0) .OR. (rquadrat <0)) then

  ! This may also occur if the points are almost coplanar, so that
     the radius of
  ! the sphere is quite large

  if (talkmode) then

    write(*,'(" Problem with alpha carbons : ", i10, ", ", i10, ", "
       , i10, ", ", i10)') i, j, k, l
    write(*,'(" Determinant of Z equals: ", f10.3)') detz
    write(*,'(" Coplanarity check determinant equals: ", f10.3)')
       determinant(coplanarity)
    write(*,*)

  end if

  r3 = 0.0

  error = error + 1

else

  r3 = sqrt(rquadrat)

  if(thickness .ne. 0.0) then

    ! This is the second run :
    ! Classification of the found radius

    pos = floor(((r3/thickness) - 1) * 100) + 1

    if (pos .gt. limit) then

      ! Found radius is bigger than the given limit

      pos = limit

    end if

    distribution(pos) = distribution(pos) + 1
```

```
      end if

      combination(1) = i ! This is due to Fortran 90 compatibility
      combination(2) = j
      combination(3) = k
      combination(4) = l

      do m = 1, 4

        ! Completing for all four points

        s = combination(m)

        do q = 1, mem

          if (r3 .lt. smallestradius(s,q)) then

            ! A smaller radius has been found

            do r = mem, q+1, -1

              ! Shifting the radius vector for insertion of the
              ! new radius

              smallestradius(s,r) = smallestradius(s,r-1)
              smallestlocation(s,r,1) = smallestlocation(s,r-1,1)
              smallestlocation(s,r,2) = smallestlocation(s,r-1,2)
              smallestlocation(s,r,3) = smallestlocation(s,r-1,3)
              smallestlocation(s,r,4) = smallestlocation(s,r-1,4)

            end do

            ! Saving the new radius

            smallestradius(s,q) = r3
            smallestlocation(s,q,1) = i
            smallestlocation(s,q,2) = j
            smallestlocation(s,q,3) = k
            smallestlocation(s,q,4) = l

            exit

          end if

          if (smallestradius(s,q) == 0.0) then

            ! This is the first radius for the given point

            smallestradius(s,q) = r3
            smallestlocation(s,q,1) = i
            smallestlocation(s,q,2) = j
            smallestlocation(s,q,3) = k
            smallestlocation(s,q,4) = l

            exit

          end if

        end do
```

```
          end do
        end if
        if (((l-k) + (k-j) + (j-i)) == 3) then
          ! The found radius is local
          if ((localradius(j) > r3) .or. (localradius(j) == 0.0)) then
            localradius(j) = r3
          end if
          ! The local radius is attributed to the both inner points
          localradius(k) = r3
        else
          ! The radius is not an local one
          combination(1) = i
          combination(2) = j
          combination(3) = k
          combination(4) = l
          do m = 1, 4
            o = combination(m)
            if ((nonlocalradius(o) == 0) .or. (nonlocalradius(o) > r3)) then
              nonlocalradius(o) = r3
              nonlocallocation(o,1) = i
              nonlocallocation(o,2) = j
              nonlocallocation(o,3) = k
              nonlocallocation(o,4) = l
            end if
          end do
        end if
        index = index + 1
      end do
    end do
  end do
end do

if (talkmode) then
  write (*, '(" ", i10, " spheres have been calculated.")') index
```

```
    write (*, '(" For ", i10, " spheres the determination of its radius was
       not possible.")') error
    write (*, *)

  end if

  deallocate(coplanarity)
  deallocate(N)
  deallocate(Z)

  allocate (globalradius(numbercarbon))
  allocate (globallocation(numbercarbon,4))

  ! Finalizing

  do i = 1, numbercarbon

    ! Which class of radii provides the smallest radius?

    radii(1) = localradius(i)
    radii(2) = nonlocalradius(i)

    selector = minloc(radii, mask = radii .ne. 0.0)
    location = selector(1)

    ! Global radius lists the smallest circle for each point
    globalradius(i) = radii(location)

    select case (location)

      ! Indexes of the points realizing the smallest circle
      case (1)

        globallocation(i,1) = i - 1
        globallocation(i,2) = i
        globallocation(i,3) = i + 1
        globallocation(i,4) = i + 2

      case (2)

        globallocation(i,:) = nonlocallocation(i,:)

    end select

  end do

  allocate (f(numbercarbon))
  allocate (g(numbercarbon))

  f = 0.0
  g = 0.0

  do i = 2, numbercarbon - 2

    ! Calculation of the f and g factor

    f(i) = nonlocalradius(i)/localradius(i)
    g(i) = globalradius(i)/localradius(i)
```

```
    end do

    deallocate (nonlocalradius, nonlocallocation)
    deallocate (firstlocal)

  end subroutine spheres

end module distance

! ************************************************************************

module loading

  implicit none

  contains

  subroutine import (inputfile, alphacarbons, atomtype, resolution)

    ! Loads the preprocessed input data files

    use var

    implicit none

    real                                :: resolution

    integer                             :: i, io, numbercarbons, temp

    character (LEN=18), intent(in)      :: inputfile
    ! Compatibility with Fortran 90

    character (LEN=4)                   :: atomtype
    character (LEN=12)                  :: dummy
    character (LEN=13)                  :: dummy2
    character (LEN=17)                  :: dummy3
    character (LEN=35)                  :: test

    type(carbons), pointer, dimension(:) :: alphacarbons

    OPEN (UNIT=2, FILE=inputfile, FORM='FORMATTED', STATUS='OLD')

    ! Reading of the header line and the resolution value

    rewind 2

    read (2, '(a35)') test

    if (test .ne. 'Atom cartesian coordinates from PDB') then

      write(*, '("This is not an atom coordinates file")')
      return

    end if

    read (2, *)

    read (2, *)
    read (2, '(a13, a3)') dummy2, atomtype
```

```
    read (2, '(a17, f4.0)') dummy3, resolution

    read (2, *)
    read (2, *)
    read (2, *)

    numbercarbons = 0

    do

      ! Counting the numbers of atoms

      read (2, '(a12)', IOSTAT = io) dummy

      if (io == 0) then

        numbercarbons = numbercarbons + 1

      else

        exit

      end if

    end do

    allocate (alphacarbons(numbercarbons))

    rewind 2

    do i = 1, 8

      read (2, *)

    end do

    do i = 1, numbercarbons

      ! Reading the coordinates

      read (2, '(i5, 3f10.3, i9, i6, a1)') temp, &
                  alphacarbons(i)%coordinates(1), &
                  alphacarbons(i)%coordinates(2), &
                  alphacarbons(i)%coordinates(3), &
                  alphacarbons(i)%residue, temp, &
                  alphacarbons(i)%chainident
    end do

    CLOSE (UNIT=2, STATUS='KEEP')

  end subroutine import

end module loading

! ***************************************************************

program calculator
```

```
! Main program

use loading                                                                    1050
use var
use distance
use parameters

implicit none                                                                  1055

character (LEN=12)                                  :: export, dummy
character (LEN=7)                                   :: exportfolder, machinefolder
character (LEN=5)                                   :: subfolder, humanfolder
character (LEN=6)                                   :: importfolder            1060
character (LEN=12), allocatable, dimension(:)       :: filename
character, dimension(:), pointer                    :: name
character (LEN=4)                                   :: extension, atomtype

real                                                :: resolution, thickness   1065
real(8)                                             :: initial, final
real, pointer, dimension(:)                         :: f, g, localradius
real, pointer, dimension(:)                         :: globalradius
real, pointer, dimension(:,:)                       :: alphacircles, smallestradius
                                                                               1070
type(carbons), pointer, dimension(:)                :: alphacarbons
type(carbons), allocatable, dimension(:)            :: allalphacarbons, oldcarbons

integer                                             :: i, j, k, numberfiles, io
integer                                             :: position, length, dislocation  1075
integer                                             :: a, b, c, d, what, sourceflag
integer                                             :: fitflag, numbercarbon, mem
integer, allocatable, dimension(:,:)                :: comb
integer, pointer, dimension(:,:)                    :: globallocation
integer, pointer, dimension(:,:,:)                  :: smallestlocation        1080
integer(4), dimension(limit)                        :: distribution

logical, dimension(:), allocatable                  :: goodfiles

exportfolder       = 'results'                                                 1085
! name of the subdirectory where the results are stored to
! Adjust the above field length of the string !!!

importfolder       = 'source'
! name of the subfolder for ASCII export files                                 1090
! Adjust the above field length of the string !!!

humanfolder        = 'human'
machinefolder      = 'machine'
                                                                               1095
if (talkmode) then

  write (*, *)
  write (*, '(" Computational physics project")')
  write (*, '(" Christian Schlatter, student, physics, 4th year")')            1100
  write (*, *) releaseinfo
  write (*, '("
      ************************************************************")')
  write (*, *)

end if                                                                         1105
```

```
! sourceflag = 1                    ! > TEMPORARY
! allocate (filename(1))            ! > TEMPORARY
! filename(1) = '1acc.hel'          ! > TEMPORARY
! numberfiles = 1                   ! > TEMPORARY

write (*, '(" Global curvature analyser")')
write (*, *)
write (*, '(" How many proteins would you like to process ?")')
write (*, *)
write (*, '(" -> Type 1 for a single input file or")')
write (*, '(" -> Type 2 for a list of files provided in Source.ini")')
write (*, *)
write (*, '(" Please choose your option now : ")', advance = 'no')

sourceflag = 2                      ! < TEMPORARY

! read (*,*) sourceflag             ! > TEMPORARY

write (*, *)

if (sourceflag == 2) then ! Batch mode using Source.ini

  write (*, '(" Opening Source.ini...")')
  write (*, *)

  open (UNIT=1, FILE='Source.ini', FORM='FORMATTED', STATUS='OLD')

  rewind 1

  numberfiles = 0

  do

    ! Determination of the number of input files

    read (1, '(a12)', IOSTAT = io) dummy

    if (io == 0) then

      numberfiles = numberfiles + 1

    else

      exit

    end if

  end do

  allocate (filename(numberfiles))

  rewind 1

  do i = 1, numberfiles

    ! Determination of the input file names

    read (1, '(a12)') filename(i)
```

```
      write (*,*) filename(i)

    end do

    close (UNIT=1, STATUS='KEEP')

    write (*,*)
    write (*, '(" You have chosen ", i3, " files to process")') numberfiles
    write (*,*)

  else

    allocate (filename(1))

    ! User provided input file name

    write (*, '(" Please provide the filename of the import file: ")', advance
        = 'no')

    read (*, *) filename(1)

    write (*, *)

    numberfiles = 1

  endif

  write (*, '(" What would you like to fit into the protein ?")')
  write (*, *)
  write (*, '(" -> Type 1 for fitting circles only")')
  write (*, '(" -> Type 2 for fitting both - circles and spheres")')
  write (*, *)
  write (*, '(" Please choose your option now : ")', advance = 'no')

  fitflag = 2                          ! < TEMPORARY

  ! read (*,*) fitflag                 ! > TEMPORARY
  write (*, *)

  allocate (goodfiles(numberfiles))
  goodfiles = .true.

  do i = 1, numberfiles

    if (talkmode) then

      write (*, *)
      write (*, '(" Extracting structure information from file ", a12)')
         filename(i)
      write (*, '("
          ************************************************************")')
      write (*, *)

    end if

      write (*,'(" Processing now file ", a12)') filename(i)
      write (*, *)

    nullify(alphacarbons)
```

```
if (associated(alphacarbons)) then

  deallocate(alphacarbons)

end if

! Reading the input file

call import(importfolder//'/'//filename(i), alphacarbons, atomtype,
    resolution)

numbercarbon = size(alphacarbons)

if (memory .gt. numbercarbon) then

  mem = numbercarbon

else

  mem = memory

end if

allocate (smallestradius(numbercarbon, mem))
allocate (smallestlocation(numbercarbon,mem,3))

if (talkmode) then

  write (*, '(" Fitting circles through the alpha carbons")')
  write (*, '("
      ************************************************************")')
  write (*, *)

end if

thickness = 0.0

! Fitting circles through three atoms

call circles (alphacarbons, f, g, localradius, globalradius, &
              globallocation, smallestradius, smallestlocation, &
              distribution, thickness)

! Circular thickness of the macromolecule

thickness = minval(globalradius)

if (secondrun) then

  ! Doing a second run for statistics on the smallest circles

  deallocate(localradius, globalradius, f, g, globallocation)

  call circles (alphacarbons, f, g, localradius, globalradius, &
                globallocation, smallestradius, smallestlocation, &
                distribution, thickness)

end if
```

```
position = scan(filename(i),'.')

if (filename(i)(position+1:position+1) == 'h') then

  ! An alpha helix input file has been used
  subfolder = 'helix'

else

  ! An backbone input file has been used
  subfolder = 'whole'

end if

! Saving the circles to disk

filename(i)(position+1:position+1) = 'c'
filename(i)(position+2:position+2) = 'i'
filename(i)(position+3:position+3) = 'r'

open (UNIT=3, FILE=exportfolder//'/'//subfolder//'/'//humanfolder//'/'//
    filename(i), FORM='FORMATTED', STATUS='REPLACE')
open (UNIT=4, FILE=exportfolder//'/'//subfolder//'/'//machinefolder//'/'//
    filename(i), FORM='FORMATTED', STATUS='REPLACE')

! Making the output file header

write(3,'("Circling of proteins")')
write(4,'("Circles")')
write(3,'("C-alpha-circle project. File created using Calculator.")')
write(3,'(a25)', advance='no') releaseinfo
write(4,'(a25)') releaseinfo
write(3,'("©  2002 EPFL Lausanne")')
write(3,'("Atom type : ",a4)') atomtype
write(4,'(a4)') atomtype

write(3,'("ResolutionÅ [] : ",f4.2)') resolution
write(4,'(f4.2)') resolution

write(3,'("Circular thicknessÅ [] : ",f4.2)') thickness
write(4,'(f4.2)') thickness

write(3,'("Thickness combination : ",i3,", ",i3," and ",i3)') &
                          globallocation(minloc(globalradius),1), &
                          globallocation(minloc(globalradius),2), &
                          globallocation(minloc(globalradius),3)

write(4,'(3i4)') globallocation(minloc(globalradius),1), &
                          globallocation(minloc(globalradius),2), &
                          globallocation(minloc(globalradius),3)

write(3,'("Memory : ",i3)') mem
write(4,'(i3)') mem

if (secondrun) then

  ! Saving statistics results to disk
```

```
   filename(i)(position+1:position+1) = 'c'
   filename(i)(position+2:position+2) = 'd'
   filename(i)(position+3:position+3) = 't'

   open (UNIT=5, FILE=exportfolder//'/'//subfolder//'/'//machinefolder//'/'//
       filename(i), FORM='FORMATTED', STATUS='REPLACE')

   do j = 1, limit

     ! Distribution of the smallest radii

     write(5, '(f10.2, i10)') ((j / 100.0) + 1.0), distribution(j)

   end do

   close (UNIT=5, STATUS='KEEP')

end if

write(3, *)
write(4, *)
write(3, '("Entry      Local    f-ratio  Dis")', advance = 'no')

do j = 1, mem

   write(3, '("    ", i2)', advance = 'no') j
   write(3, '(". global radius ")', advance = 'no')

end do

write(3, '(" Residue")')
write(3, *)

! Writing data to file

do j = 1, (size (alphacarbons))

   a = minval(globallocation(j,:))
   c = maxval(globallocation(j,:))
   b = minval(globallocation(j,:), MASK = globallocation(j,:) .NE. a)

   ! Spacing of the indexes

   dislocation = (c - b) + (b - a)

   write (3, '(i5, 2f10.3, i5)', advance = 'no') j, &
              localradius(j)/thickness, f(j), dislocation

   write (4, '(i5, 2f10.3, i5)', advance = 'no') j, &
              localradius(j)/thickness, f(j), dislocation

   do k = 1, mem

     write(3, '(" (", f5.3, ";", 3i4, ") ")', advance = 'no') &
                              smallestradius(j,k)/thickness, &
                              smallestlocation(j,k,1), &
                              smallestlocation(j,k,2), &
                              smallestlocation(j,k,3)
```

```
      write(4, '(f7.3)', advance = 'no') smallestradius(j,k)/thickness

    end do

    write (3, '(i8)' ) alphacarbons(j)%residue
    write (4, '(i8)' ) alphacarbons(j)%residue

  end do

  write (*, '(" -> This protein has a thickness of circularily ", f5.3, "
     Angstroem.")') thickness
  write(*,*)

  close (UNIT=3, STATUS='KEEP')
  close (UNIT=4, STATUS='KEEP')

  deallocate(localradius, globalradius, f, g, globallocation, smallestradius)
  deallocate(smallestlocation)

  if (fitflag == 2) then

    ! Fitting spheres through 4 atoms of the macromolecule

    allocate (smallestradius(numbercarbon, mem))
    allocate (smallestlocation(numbercarbon,mem,4))

    if (talkmode) then

      write (*, '(" Fitting spheres through the alpha carbons")')
      write (*, '("
          ************************************************************")
         ')
      write (*, *)

    end if

    thickness = 0.0

    call spheres (alphacarbons, f, g, localradius, globalradius, &
                  globallocation, smallestradius, smallestlocation, &
                  distribution, thickness)

    ! Spherical thickness of the macromolecule

    write(*, '("First run successfully completed...")')

    thickness = minval(globalradius)

    if (secondrun) then

      ! Doing a second run for statistics on the smallest spheres

      deallocate(localradius, globalradius, f, g, globallocation)

      call spheres (alphacarbons, f, g, localradius, globalradius, &
                    globallocation, smallestradius, smallestlocation, &
                    distribution, thickness)

    end if
```

```
position = scan(filename(i),'.')

! Saving the spheres to disk

filename(i)(position+1:position+1) = 's'
filename(i)(position+2:position+2) = 'p'
filename(i)(position+3:position+3) = 'h'

open (UNIT=3, FILE=exportfolder//'/'//subfolder//'/'//humanfolder//'/'//
    filename(i), FORM='FORMATTED', STATUS='REPLACE')
open (UNIT=4, FILE=exportfolder//'/'//subfolder//'/'//machinefolder//'/'//
    filename(i), FORM='FORMATTED', STATUS='REPLACE')

! Making the output file header

write(3, '("Sphering of proteins")')
write(4, '("Spheres")')
write(3, '("C-alpha-circle project. File created using Calculator.")')
write(3, '(a25)', advance='no') releaseinfo
write(4, '(a25)') releaseinfo
write(3, '("©.  2002 EPFL Lausanne")')
write(3, '("Atom type : ", a4)') atomtype
write(4, '(a4)') atomtype

write(3, '("ResolutionÅ [] : ", f4.2)') resolution
write(4, '(f4.2)') resolution

write(3, '("Spherical thicknessÅ [] : ", f4.2)') thickness
write(4, '(f4.2)') thickness

write(3, '("Thickness combination : ", i3, ", ", i3, ", ", i3, " and ", &
                        i3)') globallocation(minloc(globalradius),1), &
                              globallocation(minloc(globalradius),2), &
                              globallocation(minloc(globalradius),3), &
                              globallocation(minloc(globalradius),4)

write(4, '(4i4)') globallocation(minloc(globalradius),1), &
                  globallocation(minloc(globalradius),2), &
                  globallocation(minloc(globalradius),3), &
                  globallocation(minloc(globalradius),4)

write(3, '("Memory : ", i3)') mem
write(4, '(i3)') mem

if (secondrun) then

  ! Saving statistics results to disk

  filename(i)(position+1:position+1) = 's'
  filename(i)(position+2:position+2) = 'd'
  filename(i)(position+3:position+3) = 't'

  open (UNIT=5, FILE=exportfolder//'/'//subfolder//'/'//machinefolder
      //'/'//filename(i), FORM='FORMATTED', STATUS='REPLACE')

  do j = 1, limit

    ! Distribution of the smallest radii
```

```
      write(5, '(f10.2, i10)') ((j / 100.0) + 1.0), distribution(j)

    end do

    close (UNIT=5, STATUS='KEEP')

  end if

  write(3, *)
  write(4, *)
  write(3, '("Entry     Local   f-ratio  Dis")', advance = 'no')

  do j = 1, mem

    write(3, '("         ", i2)', advance = 'no') j
    write(3, '(". global radius ")', advance = 'no')

  end do

  write(3, '(" Residue")')
  write(3, *)

  ! Writing data to file

  do j = 1, (size (alphacarbons))

    a = minval(globallocation(j,:))
    d = maxval(globallocation(j,:))
    b = minval(globallocation(j,:), MASK = globallocation(j,:) .NE. a)
    c = maxval(globallocation(j,:), MASK = globallocation(j,:) .NE. d)

    ! Spacing of the indexes

    dislocation = (d - c) + (c - b) + (b - a)

    write (3, '(i5, 2f10.3, i5)', advance = 'no') j, &
                localradius(j)/thickness, f(j), dislocation

    write (4, '(i5, 2f10.3, i5)', advance = 'no') j, &
                localradius(j)/thickness, f(j), dislocation

    do k = 1, mem

      write(3, '(" (", f5.3, ";", 4i4, ") ")', advance = 'no') &
                                smallestradius(j,k)/thickness, &
                                smallestlocation(j,k,1), &
                                smallestlocation(j,k,2), &
                                smallestlocation(j,k,3), &
                                smallestlocation(j,k,4)

      write(4, '(f7.3)', advance = 'no') smallestradius(j,k)/thickness

    end do

    write (3, '(i8)' ) alphacarbons(j)%residue
    write (4, '(i8)' ) alphacarbons(j)%residue

  end do
```

```
      write (*, '(" -> This protein has a thickness of spherically ", f5.3, "
         Angstroem.")') thickness
      write(*,*)

      close (UNIT=3, STATUS='KEEP')
      close (UNIT=4, STATUS='KEEP')

      deallocate(localradius, globalradius, f, g, globallocation, smallestradius
         )
      deallocate(smallestlocation)

    end if

  end do

  deallocate(filename)
  deallocate(goodfiles)

  write(*, '("Program execution finished")')
  write(*, *)

end program calculator
```