uc3m | Universidad Carlos III de Madrid

University Degree in Biomedical Engineering
2018-2019

Bachelor Thesis

"Image Processing Algorithms for Electron Microscopy"

David Herreros Calero

Supervisor: Mr. Carlos Óscar Sánchez Sorzano Ph.D.

Leganés, 12nd March 2019

ACKNOWLEDGEMNTS

I would like to express my gratitude to the following people for their support and help during this project.

To my tutor, Mr. Carlos Óscar Sánchez Sorzano Ph.D., for offering me the opportunity to work in this project, for his advice, support and for being always ready to solve any problem encountered during the project.

To my academic tutor, Mr. Jorge Ripoll Ph.D., for the support he gave me in the writing of the bachelor thesis and all the useful information he provided to improve it.

To all the team of the Biocomputing Unit at the National Centre for Biotechnology (CNB), for the advices and the time they dedicated to help with the project.

To my teachers, who prepared me to face a project like this thesis and to take the best out of me.

To my family and friends for being always by my side, supporting, encouraging and trusting me in all my decisions and advising me when I needed it.

ABSTRACT

The development of the electron microscopy (EM) has led to new algorithms devoted to the analysis of the images obtained by the EM know as micrographs. These images are characterized for being noisy, making the extraction of information a difficult task. To that end, several reconstruction packages are being developed that provide with powerful tools to denoise, extract and analyze the micrographs to obtain the 3D model (volume) of the sample.

Since there is no a gold standard that can be followed to process all micrographs in a single way, researches rely on the combination of algorithms of several packages to get the optimal result. However, the lack of consensus between the packages makes difficult to combine different programs. Also, the algorithms should be made as simple and automatic as possible to facilitate the task of the researchers without degrading the quality of the results.

The purpose of this bachelor thesis is to introduce required algorithms that can be used to reconstruct the 3D model of a sample out of the micrographs acquired by the electron microscope. The analysis will be focused on two individual and complementary tasks: the development of a mathematical basis aimed at the automatization and simplification of the deformation calculations carried out to optimize the 3D templates used to reconstruct new 3D models, and the protocolization of a package that uses *ab initio* methods to avoid the requirements of a 3D template to reconstruct a model. The algorithms will be tested in order to validate and analyze their performance, so they can be used in real problems and applications.

Key words: Normal Mode Analysis (NMA); Electron Microscope (EM); Spherical Harmonics (SH); Zernike polynomials; Fourier Transform (FT); Multidimensional Scaling (MDS); SCIPION; Transmission Electron Microscope (TEM); Scanning Electron Microscope (SEM).

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

The analysis of the different hierarchical structures present in an organism is carried out by sophisticated technologies. Progressively, scientists started to study the smallest structures living organism are composed of: cells. By the middle of the 19th century, light microscopes started to introduce limitations to the maximum resolution to which the internal structures of the cells could be resolved [1]. Overcoming these limitations was crucial for the understanding of the different mechanism and machineries used by the cells to perform their functions. The invention of the EM by Max Knoll and Ernst Ruska in 1931 proved that it was possible to go beyond the resolutions achievable with the light microscope, leading to a revolution in this field.

Since their invention, the improvement of the EM has allowed to resolve structures with a resolution range between near atomic resolution and 3 nm [2]. The main advantage of EM against light microscopes is the use of a beam of accelerated electrons instead of photons to illuminate the sample. According to de Broglie equation, the wavelength of an electron depends on its mass and speed. In general, the wavelength of an electron is several orders of magnitude below the wavelength of a photon, allowing scientists to obtain the structure of smaller objects.

Since the discovery of the electron microscopy, two main types of EM have been developed depending on their mode of operation [3]:

- Transmission Electron Microscope (TEM): The TEM is based in a cathode able to emit accelerated electron towards the sample and focused by magnetic lenses. As the electron traversed the sample, they will carry the structural information of the specimen. The image is then recorded thanks to the electrons hitting a fluorescent screen, photographic plate or a light sensitive sensor. This was the first mode of operation of an EM in history.
- Scanning Electron Microscope (SEM): Nowadays, EM has experienced a quantum leap by the introduction of direct electron detectors, that allow imaging at much better resolution. The SEM detects the secondary electron emitted by the sample due to the interactions of the primary accelerated electrons with the atoms of the specimen. The primary electron beam is scanned along the surface of the sample and secondary electrons are collected by detectors able to relate the detected signal to the position of the beams.

1.1. Motivation

Three-dimensional electron microscopy (3D-EM) is a technique used to reconstruct large biomolecules in their native state. The main requirement to obtain a 3D model (volume) of any molecule is to take several micrographs that contain thousands of projections of the molecule of interest [3].

There are several difficulties to overcome during the reconstruction of the 3D model of any molecule:

Firstly, the projections are noisy as the samples are illuminated with low doses to avoid damaging the specimen and due to the presence of amorphous ice with a contrast very similar to that of the macromolecules. Secondly, a large amount of projections are needed (in the order of 10^3 to 10^5 projections) to reconstruct the maps with an admissible resolution. This makes the reconstruction process demanding in terms of computational complexity and resources.

There are several packages available to reconstruct a 3D model from a set of particles including EMAN [4], SPIDER [5] or XMIPP [6]. Since there is not an optimal methodology to obtain the maps, several techniques have been developed to overcome the different difficulties a researcher may encounter.

The motivation of this work is to include in SCIPION new techniques to simplify different processes involved in the reconstruction and analysis of a 3D model. SCIPION is a program developed by the Biocomputing Unit of the National Center for Biotechnology (CNB) in Spain. The purposed of SCIPION is the integration of several tools in a unified interface to reconstruct 3D models of macromolecular complexes [7]. SCIPION is mainly distributed for Linux OS.

The techniques exposed during this work aim in the automatization of processes, and the simplification of the requirements needed to execute a program in order to reduce the background knowledge a user should have to obtain a proper result.

1.2. Objetives

The purpose of this report is to introduce and discuss two techniques implemented in SCIPION.

The first technique is an algorithm based on a mathematical basis defined over the sphere to find the strain and rotation gradients that minimize the distances between two similar models (e.g. two maps representing different conformations of a macromolecular complex). The objective sought with this application was to facilitate and automatize the process of finding such strain and rotation gradients, so any user can apply this analysis in an easy manner.

The methodology followed for the implementation of the mathematical basis was the following:

1. Definition of the mathematical tools required to define the basis on the sphere (this work was done together with the Dr. Roy Lederman from Yale University).
2. Implementation of each part of the basis in C++.
3. Testing of the mathematical basis with basic 3D volumes to confirm that the behavior of the basis is appropriate.
4. Testing of the mathematical basis with real 3D models to compare the results with validated resources.
5. Adapt the mathematical tool to compute the deformation and rotational gradients if the input data is a 3D model and a micrograph.
6. Analysis of the deformation and rotational gradients and visualization of the results.

The second technique is the implementation of SIMPLE package [8] in SCIPION. The main objective is to provide with alternative algorithms that may be useful for some specific reconstruction applications, as there is no a standard to reconstruct 3D models from the micrographs.

The methodology followed during the implementation of SIMPLE into SCIPION was the following:

1. Understating the basic concepts required to protocolize a reconstruction package into an integration framework like SCIPION.
2. Understanding of the algorithms and workflows used by SIMPLE to approach the reconstruction procedure.
3. Protocolization of the programs provided in the newer version of SIMPLE (v2.5) that introduce newer approaches/algorithms to SCIPION.
4. Testing of the SIMPLE programs implemented in SCIPION to ensure their correct functioning and correction of errors.

The implementation of the first technique was done in C++ programing language and it was included inside a package of SCIPION known as XMIPP which is also developed by the Biocomputing Unit. The second technique was written in Python programing language and it was directly implemented within SCIPION.

The regulatory framework of this work is described in chapter 7 of this document.

2. BACKGROUND

The main purpose of this chapter is to provide to the reader a brief summary of the state of the art and the basic concepts required to understand the methodology develop along the work.

2.1. Normal mode analysis

The assembling of several atoms in a repetitive manner leads to the formation of large molecules (macromolecules) known as polymers. These polymers have many degrees of freedom (as much as $3N - 6$ being N the number of monomers forming the polymer) [11]. Thanks to this large amount of degrees of freedom, a polymer can exist in several shapes known as conformations.

One of the most important polymers in biology are proteins. Like other polymers, proteins may be reshaped to many different conformations, but in nature these conformations are constrained to a series of microstates that will determine (together with other properties of the proteins) the function of a specific protein inside cell. In principle, a protein can be found in nature in any of its possible microstates although they are not equally probable.

The microstates of a protein follow a probabilistic model being the most probable microstate known as the "conformation" of that protein.

The introduction of computation for biological applications made possible the analysis of the different conformations of a given protein once its structure (usually the conformation of the protein) is known.

Among the different techniques developed for this purposed, this section will be focused on normal mode analysis (NMA). NMA is "a molecular modelling technique used to model the vibrations, fluctuations and conformational changes of proteins" [12].

To that end, NMA considers that the atoms composing any protein have a given amount of energy (i.e. they are not static, but they present a dynamic behavior usually manifested through vibration). Apart from this intrinsic vibration coming from the energy content of the atoms, atoms also suffer different forces (hydrophobic interactions, Van der Waals forces, electrostatic forces…) which keep them in a specific position within the macromolecule.

NMA tries to model the behavior of the atoms inside a protein through an elastic network model (ENM). An ENM of a protein is created by assuming that the atoms inside the protein are connected to the atoms surrounding them through springs with a given elastic energy (which is the addition of the intrinsic energy and the extrinsic forces acting on the atom). An example of an ENM is provided in **Figure 1**.

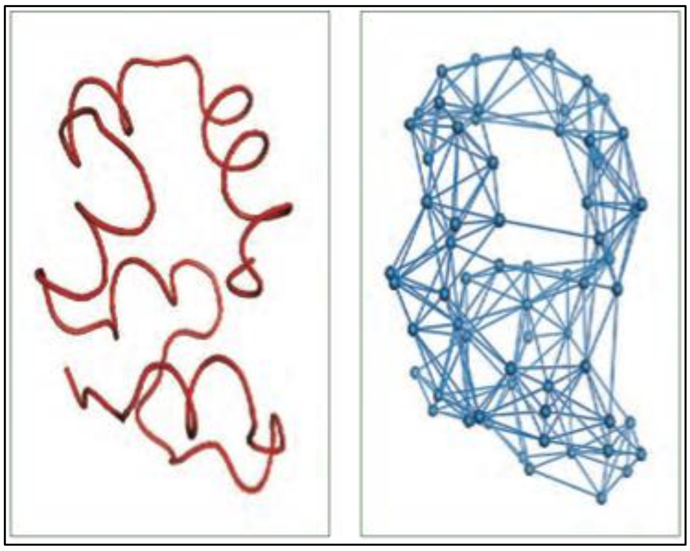


Figure 1: ENM of a macromolecule. Extracted from [13].

Once the ENM of a protein is known, it can be used to compute the total elastic energy of the system. Then, the Hessian matrix of this elastic energy is obtained in order to analyze the eigenvalues and eigenvectors associated to the energy of the system.

This process is known as NMA [12]. It is important to mention that the eigenvalues and the eigenvectors should be analyzed in pairs: the eigenvectors represent a movement of the macromolecule and the eigen value is the associated energy cost per unit of that movement.

Since NMA is an appropriate and affordable way to compute the conformations and deformations a given protein may suffer, it is possible to use them to solve the problem of finding the rotation and deformation fields that bring two conformation of similar proteins as close as possible [14]. An example of this application is shown in **Figure 2**.

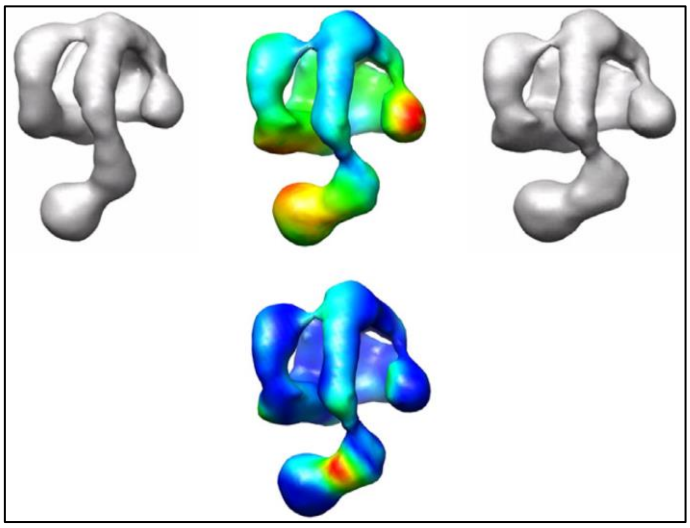


Figure 2: Example of the deformation and rotation fields computed using NMA. Extracted from [14].

The main disadvantage of using NMA for this application is the lack of full automatization. The method will provide to the user a set of NM representing the possible movements a macromolecule may suffer. If the user does not pick the adequate combination of NM, the finding of the deformation and rotation fields may be compromised leading to unexpected results. The improvement introduced with the mathematical basis is the complete automatization of the process to reduce the dependency of the user in the result.

2.2. Principle of operation of an EM

The next subsection provides a summary of the physical principles of EM. The whole description of the principles is available at [15].

Imaging a given sample requires to measure the interaction between a source of light and the sample. In the 20th century, it was discovered that the electron exhibits a wave-particle duality whose characteristics wavelength is expressed as:

$$\lambda = \frac{h}{p} \tag{2.1}$$

Where h is the Plank constant and p is the linear momentum of the electron. This equation was first proposed by Louis de Broglie.

The linear momentum of the electron depends on the potential used to accelerate the particles in the EM. If the potential is small, the electron will move slowly leading to a strong diffraction. If the potential is progressively increased, the energy and velocity of the electron will be increased accordingly. At some point, the electron will be able to penetrate a solid several microns. Depending on the internal structure of the sample (and the energy of the electrons), the electron penetrating it will be absorbed, scattered or transmitted. Electron that are capable of interacting with the sample followed by their transmission carry information about the structure of the sample. **Figure 3** shows the possible fates of an electron interacting with a given sample.

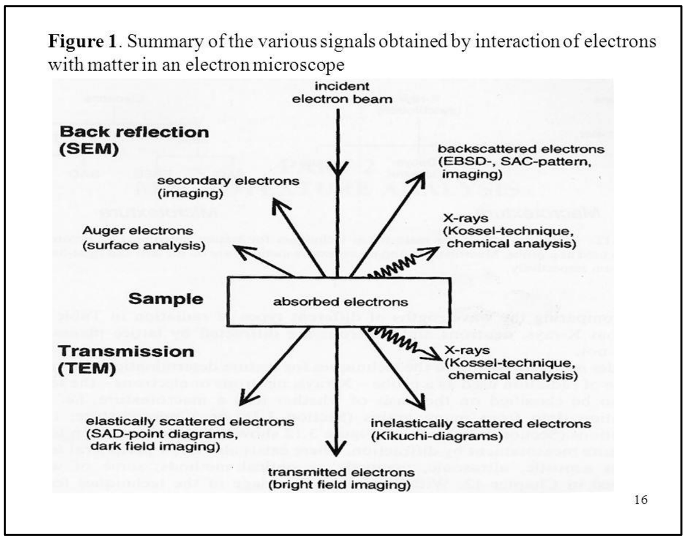


Figure 1. Summary of the various signals obtained by interaction of electrons with matter in an electron microscope
incident electron beam
Back reflection (SEM)
backscattered electrons (EBSD-, SAC-pattern, imaging)
secondary electrons (imaging)
X-rays (Kossel-technique, chemical analysis)
Auger electrons (surface analysis)
Sample
absorbed electrons
Transmission (TEM)
X-rays (Kossel-technique, chemical analysis)
elastically scattered electrons (SAD-point diagrams, dark field imaging)
inelastically scattered electrons (Kikuchi-diagrams)
transmitted electrons (bright field imaging)
16

Figure 3: Possible outcomes of the interaction of an electron with matter. Extracted from [16].

Since the electron is a particle negatively charged, it can be influenced by the presence of electric or magnetic fields that may be surrounding the molecule at a given moment. Thanks to this property of the electrons, it is possible to redirect the electrons transmitted through the microscopic structure of a material to create an image. This is the principle of operation of the Transmission Electron Microscope (TEM). **Figure 4** shows a summary of the operation of the TEM compared to the light microscope.

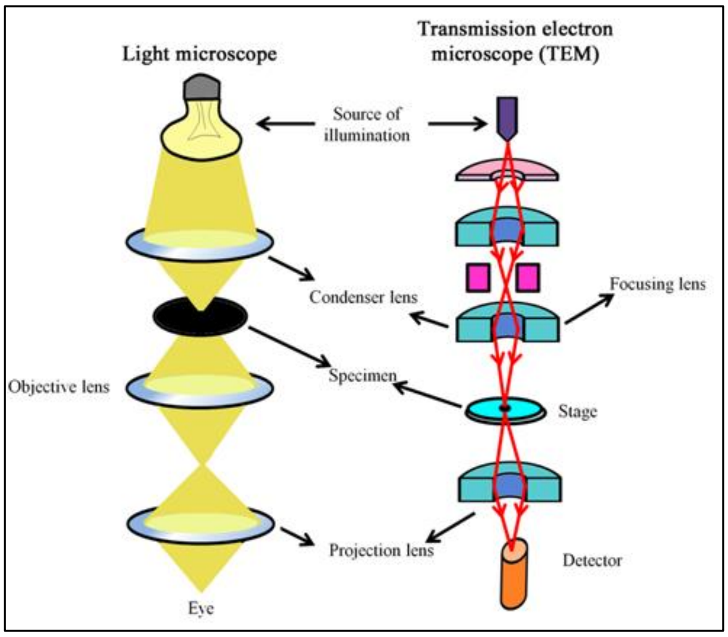


Figure 4: Operation of the TEM compared to the light microscope. Extracted from [17].

As it can be seen from equation (2.1), if the potential used to accelerate the electrons is made large, very low wavelengths can be achieved that would lead to an increase in the resolution of the images obtained by the TEM. The main drawback is that if highly energetic electrons are used to image the sample, some damage may be induced leading to a wrong result. To that end, it is necessary to use the appropriate potential, so good enough resolutions are achieved without damaging our samples. Currently, it is common to use low energy beams of electrons to get a movie from the sample rather than a single image (that will be postprocessed to improve the quality of the image). This reduces the risk of damaging the sample.

TEM are usually used when the samples are thin, so electron can be transmitted through the sample. If the thickness of the sample increases, electrons are scattered or absorbed by the sample rather than being transmitted. This make impossible to image bulky samples by using the TEM.

As it can be seen from **Figure 3**, electron can suffer two other fates apart from the absorption, transmission and scatter. Electrons may be backscattered by the microscopic structure of the material or, if the energy is large enough, they can induce the release of an electron bounded to an atom forming the sample. These electrons are known as secondary electrons and they can be also used to produce an image of the sample.

The main approach is to focus two primary beams of electrons (perpendicular to each other) in each region of the samples to scan a square area of the specimen known as raster. Since the specimens are bulky, the electrons coming from the primary beams won't be transmitted, but they will induce a secondary emission of electrons from the material. These secondary electrons can be used to produce an image of the area that is being scanned by the primary beams. This procedure is known as Scanning Electron Microscopy (SEM). One drawback of this technique is that the resolution of the images obtained is below the TEM, but it is better than the light microscope. **Figure 5** provides a summary of the operation of the SEM compared to the light microscope.

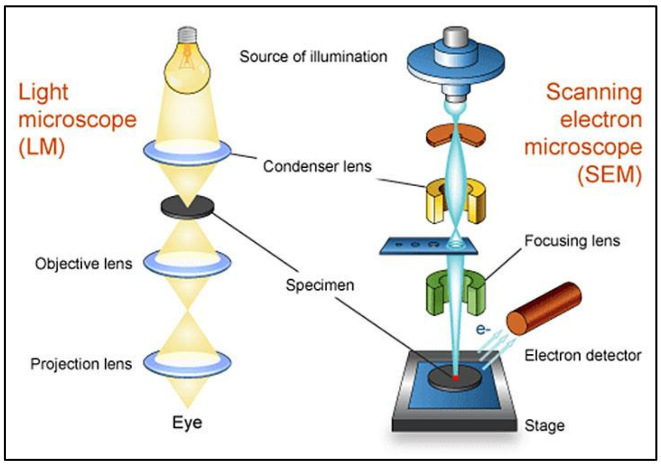


Figure 5: Operation of SEM compared to the light microscope. Extracted from [18].

It is also possible to combine the principles of the TEM and the SEM to image a thin sample. This gave rise to the Scanning Transmission Electron Microscope (STEM). This EM uses the electrons coming out from the opposite side to the one irradiated by the beam of electrons (although the electrons may not emerge from the sample in the direction of the beam but with a given direction). **Figure 6** shows the main differences between the operation of TEM and STEM.

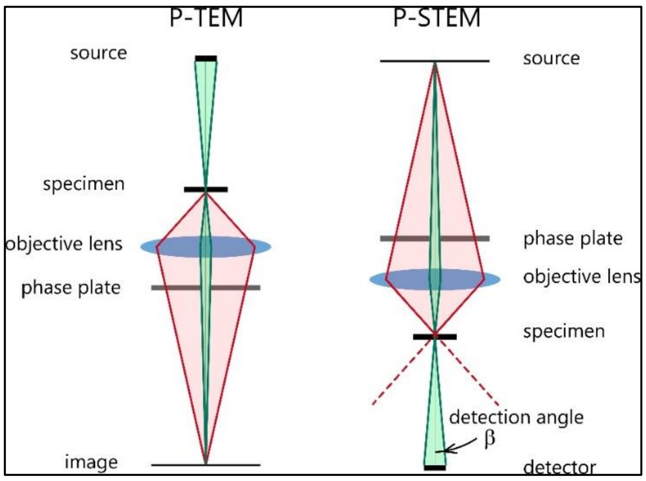


Figure 6: Comparison between the operation of TEM and STEM. Extracted from [19].

Apart from the TEM, SEM and STEM, there are other EMs. The description of these devices is provided at [15].

In order to produce a good image, the electrons must be focused properly in the detectors. In fact, the physical principles applied to the optics of the light microscope can be also applied to the EM, as the electrons can be considered as waves that follow a given path like light rays. The main difference are the lenses needed to focus the electrons compared to those used on the light microscope.

In EM, it is possible to divide the lenses in two different groups: electrostatic and magnetic lenses. Although the mechanisms of the lenses will not be explained here, they can be considered as normal glass lenses that take advantage of the electrical charge of the electron to focus them properly in the detectors. The whole description of the optics of both the light microscope and the electron microscope is available at [15].

3. METHODS

The following chapter is intended to be an analysis of the mathematical tools required to define the basis in the sphere that will be explained posteriorly in this work. After the definition of the tools, a brief description of the operation and functioning of SIMPLE package, SCIPION and XMIPP is also provided.

3.1. Infinite dimensional function space

Before explaining the set of functions that will be used to develop the mathematical basis, it is worth to introduce the concept of Hilbert space that will be useful to understand the final behavior of the basis.

In mathematics, infinite dimensional function spaces are defined as a Hilbert space [20]. The Hilbert space is a vector space that includes the inner product. This provides the space with new information about the vectors, as it is possible to measure their length, angle, orthogonality… Another important restriction of the Hilbert space is that the introduction of the inner product and the norm makes it a complete metric space [21]. If this property is not verified, then the space is known as inner product space.

The mathematical tool developed during this work represents a function space (with an infinite number of dimensions) defined over the sphere. In comparison with a vector space, a function space can be considered as a vector space whose points are functions [22], so they verify the properties of any vector space. Moreover, as an infinite dimensional space, the functions must lie in an infinite dimensional vector space like the Hilbert space [23].

One useful characteristic of the infinite dimensional vector spaces is that they are a generalization of finite spaces like the Euclidean space. This should also be true for our mathematical basis. As it will be explained lately in this work, our mathematical tool tries to find the rotational and strain fields that minimizes the distances between the voxels of two related EM volumes. Apart from the more localize deformations and rotations that will be applied to the volumes, it is also possible to include a rigid transformation that is defined over the Hilbert space. Our mathematical basis should also be able to identify and apply this transformation, as it is defined over a finite vector space contained in the generalization of the infinite dimensional vector spaces.

Since the basis is defined over the space contained in the sphere, it is convenient to work with angles and radii to simplify the formulation of the mathematical basis. In addition, it will allow to apply a “divide and conquer” strategy, as it is convenient to decompose the mathematical basis into its radial and angular components. The next two subsections are intended to be a simplified explanation of these two components. The result of combining the radial and angular components to define the final tool is described in the following chapter.

3.2. Angular part of the basis

Keeping in mind that the final objective of the analysis is to define a tool able to decompose any function defined over the sphere, it is needed to impose two constrains to the behavior of the angular part of the basis:

- In general, an angular basis working in a 3D space must belong to the *3D Rotation Group (SO_3)*. In mechanics and geometry, SO_3 represents any rotation about the origin of the Euclidian 3D Space (or $\mathbb{R}^3$) [12]. This is an important constrain as any sphere $S \in \mathbb{R}^3$ and $[0,2\pi] \in S$.

- The angular part should be a complete set of (orthogonal) functions over the surface of the sphere (i.e. they can represent any function defined over the surface of the sphere).

Considering the 2D case of the problem, in general sine and cosine functions can be considered as the angular part for a basis define over $\mathbb{R}^2$ (if polar coordinates are used). These two functions verify the previous two properties and, in fact, are already use in the Fourier Transform [24].

The 3D functions analogue to the sine and cosine function for the case treated along project are a set of functions known as *spherical harmonics (SHs)*. SHs are derived from the Laplace equation in spherical coordinates.

$$\nabla^2 f = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial f}{\partial r}\right) + \frac{1}{r^2 sin\theta}\frac{\partial}{\partial \theta}\left(sin\theta\frac{\partial f}{\partial \theta}\right) + \frac{1}{r^2\sin^2\phi}\frac{\partial^2 f}{\partial \phi^2} = 0 \qquad (3.1)$$

The previous equation is usually solved through separation of variables of the form:

$$f(r,\theta,\phi) = R(r)Y(\theta,\phi) \qquad (2.2)$$

Using (3.2) in (3.1) and after some algebra, it is possible to find an equation for the angular part for both angles θ and ϕ that reads:

$$\frac{1}{sin\theta}\frac{\partial}{\partial \theta}\left(sin\theta\frac{\partial Y(\theta,\phi)}{\partial \theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2 Y(\theta,\phi)}{\partial \phi^2} + l(l+1)Y(\theta,\phi) = 0 \qquad (3.3)$$

It is possible to apply separation of variables to (3.3) following the methodology shown in (3.2) to get a separated equation for the two angular coordinates. According to the full derivation available in [25], the resulting equations after the second separation of variables depend on two functions that fulfill some periodicity and regularity conditions. This simplifies the solution of (3.3) to a Sturm-Liouville problem which is useful to find the value of the first constant appearing after the first separation of variables. Moreover, an appropriate change of variables transforms (3.3) into a Legendre equation.

Combining all the previous properties, the solution to equation (3.3) reads:

$$Y_l^m(\theta,\phi) = Ne^{im\phi}P_l^m(cos\theta) \qquad (3.4)$$

Where Y_l^m represent the SHs of degree l and order m, P_l^m is the associated degree polynomial of same degree and order that Y_l^m, N is a normalization constant and θ and ϕ reprenst the angular variables (colatitude and azimuthal angles respectively).

The constants l and m are the parameters derived from the Sturm-Liouville problem and the separation of variables. **Figure 7** shows an example of some spherical harmonics represented along the surface of the unit sphere.

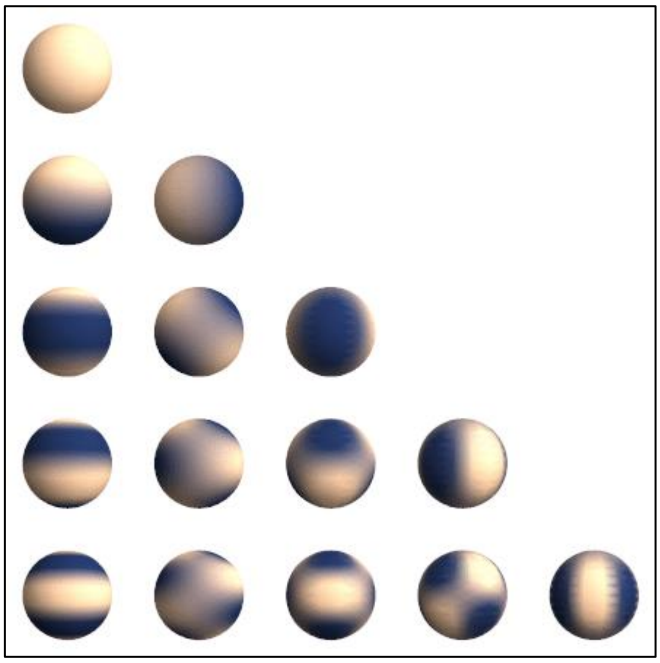


Figure 7: Representation of some SHs over the sphere. Extracted from [24].

The normalization constant can be written in terms of l and m in a convenient way to implement it in a programing language, although the calculation won't be needed for this application as the formulas of the SHs are already tabulated (the discussion of the application of the tabulated formulas instead of its computation is provided in the following chapter).

It is important to note that SHs are complex functions for all values of m different from zero. In general, working with complex numbers with a computer suppose a high computational complexity and it should be avoided. Fortunately, SHs admit a real form thanks to the complex conjugation conjugate. The complex conjugate of any SH can be written as:

$$Y_l^{m*}(\theta,\phi) = (-1)^m Y_l^{-m}(\theta,\phi) \tag{3.5}$$

After performing a linear combination of the form:

$$(-1)^m\sqrt{2}\,(Y_l^{-m} \pm Y_l^m) \tag{3.6}$$

An expression for the real SH is obtained (also known as tesseral SH):

$$Y_l^m = \begin{cases} (-1)^m\sqrt{2}\,NP_l^{|m|}(cos\theta)\sin|m|\phi & if\ m<0 \\ N_0P_l^m(cos\theta) & if\ m=0 \\ (-1)^m\sqrt{2}\,NP_l^m(cos\theta)\cos m\phi & if\ m>0 \end{cases} \tag{3.7}$$

To finish with the discussion of the angular part of the basis, it is important to mention that by convention the degree and order of the SHs are named l and m respectively because in quantum mechanics they represent the azimuthal and magnetic quantum numbers. In fact, for each value of $l, m = -l, \dots 0, \dots l$.

3.3. Radial part of the basis

Before introducing the set of functions that will constitute the radial part of the basis, the desired properties of this component will be described:

- The radial part of the basis should be formed by a set of (orthogonal) functions that can be contained in the unit sphere (i.e. the functions begin at the origin and finish at $r = 1$) and that are continuous on this interval.
- The value of the functions should oscillate in the interval defined previously as the degree of the functions is increased. These variations will allow the basis to affect differently the voxels forming the volume. In general, the expected behavior is to modify progressively higher levels of details (high frequencies) of the 3D model as the degree of the basis is increased.
- The values the radial part is taken in the unit sphere should belong to the close interval $[0,1]$. Thanks to this property, the basis won't affect the original proportions of the 3D model. This is desirable as the purpose of the basis is to find a different microstate of the molecule, but without altering the original macromolecule.

According to these properties, Zernike polynomials were chosen as the radial part of the basis. Zernike polynomials are a set of orthogonal and continuous functions that form a basis in the unit circle [26]. These functions are usually used in optics as they are useful to describe the spherical aberrations that may be present in a lens (for example, those affecting the human eye lenses like astigmatism or hypermetropy).

Since Zernike polynomials are usually defined over the unit disk, they are usually written together with the sine and cosine functions (that form an angular basis in the 2D space). Zernike polynomials over the unit disk are defined as:

$$\begin{cases} Z_l^m(r,\theta) = R_n^m(r)\cos m\theta \quad for\; m \geq 0 \\ Z_l^{-m}(r,\theta) = R_l^m(r)\sin m\theta \;\; for\; m < 0 \end{cases} \tag{3.8}$$

For the purpose of the project, only the radial part of the Zernike polynomials is of interest. The radial Zernike polynomials can be defined as:

$$R_n^m(\rho) = \sum_{k=0}^{\frac{n-m}{2}} \frac{(-1)^k (n-k)!}{k!\left(\frac{n+m}{2}-k\right)!\left(\frac{n-m}{2}-k\right)!}\rho^{n-2k} \tag{3.9}$$

According to the previous definition, the radial Zernike polynomials different from zero only for $n-m$ even. It is also possible to redefine the radial Zernike polynomials, so they take values when $n-m$ is odd. This new set of polynomials are known as the pseudo-Zernike polynomials [27].

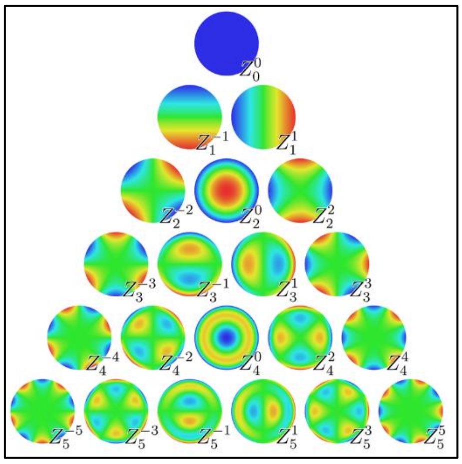


Figure 8: Representation of the first Zernike polynomials. Extracted from [28].

For this application, the standard Zernike polynomials will be used**. Figure 8** shows the Zernike polynomials (using the sine and cosine functions as the angular part of the basis)

Although it will be discussed in the following sections of this work, it is worth to mention now the relation that is present between the degree and the order of the spherical harmonics and the radial Zernike polynomials. Just for simplicity, the properties of the degree and order of both sets of functions are listed below:

- The degree and the order of the radial Zernike polynomials are restricted to combinations of numbers such that $n-m$ is an even number (otherwise their value will be zero as discussed before) being $n \geq m$ (where n is the degree of the polynomials and m the order).
- The degree l and the order m of the spherical harmonics must verify that $l \geq 0$ and $|m| \leq l$.

According to the previous restrictions, the set of polynomials defining the mathematical basis depend on a set of three numbers $\{l, n, m\}$ that must verify:

- l takes values from 0 to infinity (in theory).
- n takes values from 0 to l (included). In addition, it must be verified that if l is even/odd so is n. Following this definition, the degree of the radial Zernike polynomials corresponds to the value of l.
- m takes value from $-l$ to l.

3.4. SIMPLE

SIMPLE (Single-particle Image Processing Linux Engine) is a package developed by Dominika and Hans Elmlund whose purpose is to provide the tools necessary to reconstruct the 3D model of a macromolecule using 2D projections coming from a set of micrographs acquired with the EM. SIMPLE is designed to work with images showing a single type of particle at a time.

The main advantage of SIMPLE over other reconstruction procedures is the implementation of an *ab initio 3D reconstruction* algorithm [8]. Most of the procedures used to reconstruct the 3D model coming from the projections of a particle require an input reference volume already defined, that will be used to find an approximation of the projection angles of the particles along the 3D model. *Ab initio 3D reconstruction* is a procedure that generates an initial 3D model without the requirement of a reference volume (i.e. all the data required to find the approximate orientations of the projections is obtained from the projections themselves). The initial model is then refined to get a better result.

SIMPLE makes use of other packages (EMAN, SPIDER…) to preprocess the images before performing the *ab initio 3D reconstruction* process. This section is not intended to be a deep explanation of the functioning of SIMPLE and the packages it uses, but a brief introduction/summary to its novel *ab initio reconstruction* algorithm[1].

After correcting the motion artifacts and other errors that may be present in the original micrographs, it is necessary to extract the particles of interest present on the images. SIMPLE uses an automatic algorithm to extract the particles present in a micrograph based on reference particles that may be extracted manually from the image (using, for example, EMAN or other package that includes manual tools to select a specific region of interest in the micrographs).

Although the micrographs have been corrected, the particles extracted are in most of the cases noisy. This is mainly due to the low electron doses used to acquire images from a sample to avoid radiation damage to the particles. Since the 3D model reconstruction highly depends on the number of projections available and their quality, it is useful to reduce the noise of the images before feeding them to the reconstruction algorithm.

SIMPLE includes a package known as PRIME (Probabilistic Initial 3D Model Generation) that includes a protocol to cluster those particles that represent a similar projection of the real 3D macromolecule. The clustering of several images into different classes (one per projection present in the micrograph) increases the SNR of the particles, leading to more accurate results when fed to the reconstruction algorithm. This clustering method is based on a "stochastic hill climbing algorithm". The main difference between a stochastic hill climbing based algorithm and other optimization algorithms (such as steepest-ascent hill climbing) is that the direction chosen to move towards the maximum is randomly chosen among all those directions that maximize the function (unlike steepest that chooses the direction of maximum movement towards the maximum of the function). This method is less sensitive to get trapped in a local maximum.

[1] The interested reader is referred to [8], [26], [27] to find a complete description of the algorithms used by SIMPLE

Once the clusters have been found, they are used as an input to the *ab initio reconstruction* algorithm provided by SIMPLE. The main idea of the method is to generate an initial random model that will be projected in 2D images. These projections will be compared to the clusters through a correlation function to find the range of orientations that maximize the correlation between the clusters and the 2D projections of the initial model (in the case of deterministic methods, the orientation assigned to each projection is fixed as unique) giving to each orientation in the range a given weight. In this way, the reconstruction of the model is based on a weight average of the clustering images at the different possible orientations. The newly generated models are submitted to the same analysis until the final *ab initio 3D model* is found. An example of the reconstruction process results of a ribosome for different iterations using this *ab initio* algorithm is provided in **Figure 9.**

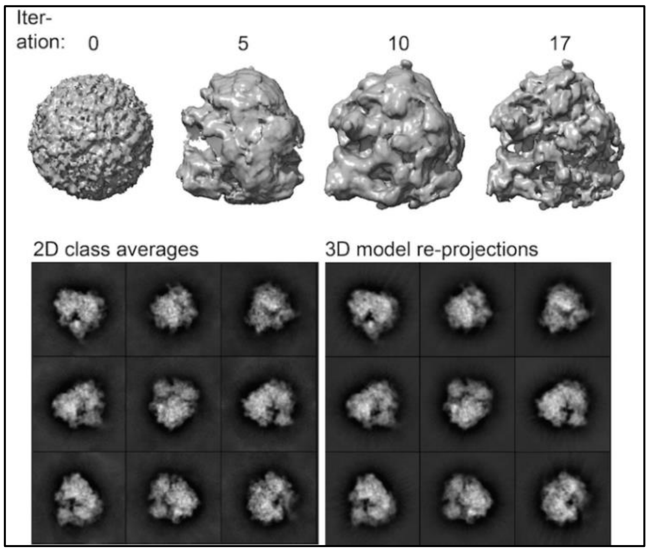


Figure 9: Ab initio 3D reconstruction and convergence of clusters and 3D projections. Extracted from [31].

3.5. SCIPION

SCIPION is an image processing framework developed by the Biocomputing Unit at the National Center for Biotechnology (CNB) in Spain [7]. The main objective of SCIPION is to integrate and manage the interaction among several packages oriented toward the reconstruction of 3D models of molecules using the images coming from the EM.

Nowadays, there are several packages devoted to the reconstruction of 3D models like SIMPLE [8], EMAN [4], SPIDER [5] or RELION [9] among others. All the packages available have advantages and disadvantages, so it is usually needed to combine the algorithms of different packages to get the desired result.

One of the main problems when using different packages for a given application is the lack of standardization in the format of the output and input files required to call a given program. This usually complicates the task of the user, as he needs to be aware of the differences between the working flows and the results provided by each package in order to combine them properly to get an appropriate result.

In order to simplify the interaction of different programs, integration software packages like SCIPION were developed to handle and manage the different output formats of the files generated by the reconstruction packages.

In the case of SCIPION, the management is performed trough special object types. Some examples of these types include: *micrographs, movies, volume, set of averages*… These object types are used to match the requirements of each program inputs and outputs, so the interaction among them is no longer problem of the user.

Apart from the integrative function of SCIPION, this software also allows to perform a traceability of the workflows executed to reconstruct a given 3D model. SCIPION saves the inputs and outputs of each step executed in the workflow in a different folder and creates a "*log"* file. This allows the user to examine the results and the possible errors or warnings that may appear during the execution of the workflow to isolate the steps that are leading to a given problem.

Figure 10 shows the typical workflow of execution of SCIPION and provides with some examples of the programs integrated to perform different tasks.

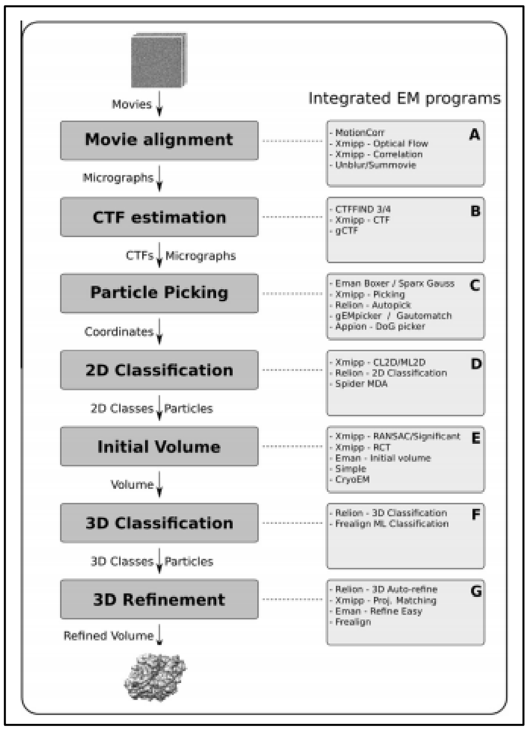


Figure 10: Example of the typical workflow of SCIPION executed for a 3D model reconstruction. Extracted from [7].

SCIPION works by means of projects that can be created by the user to reconstruct a 3D model. The project name and path can be determined during its creation. It is also possible to import an already existing project saved in the computer to modify or include new steps on it.

Figure 11 shows the GUI (Graphical User Interface) of a project in SCIPION. As it can be seen from the image, the GUI of SCIPION is mainly formed by three different windows that are described below:

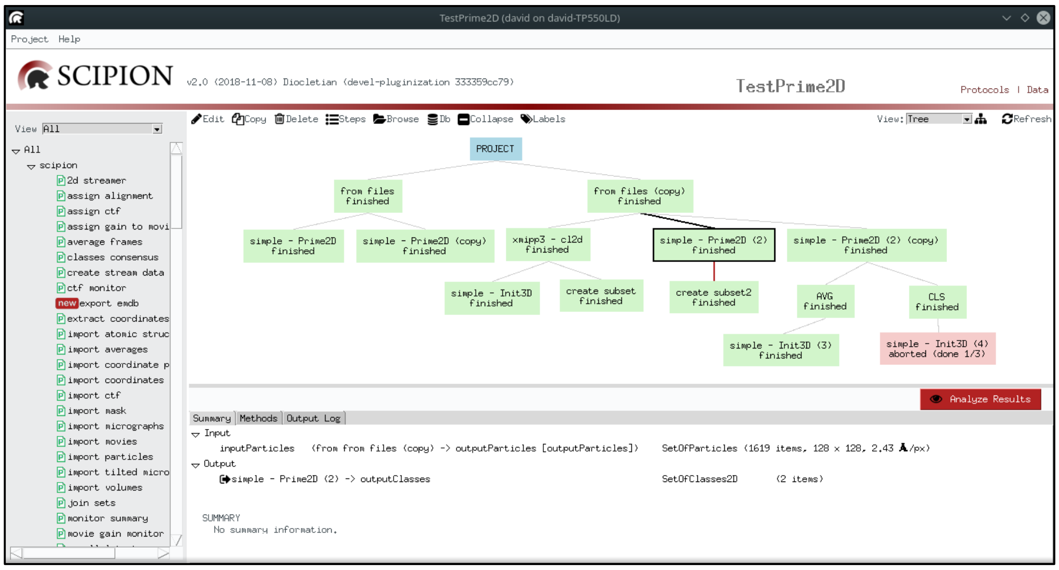
- The window placed at the left of the GUI contains all the programs integrated in SCIPION for the reconstruction of a 3D model. They can be organized in different ways to make easier finding a specific algorithm. When a program is selected, a prompt window will appear showing the inputs required to execute the program. **Figure 12** shows an example of this window.

  The central window shows the workflow of a given reconstruction procedure. The different steps of the workflow are specified by blocks that are inserted by clicking in any program present in the left window and filling up the inputs. The edges are used to stablish the block that is sending its outputs as the inputs of the actual block, and to show which block is going to receive the output of the actual block.

  Blocks can be shown in three different colors. A green color means that the block has been executed without errors; a red color means that the execution was stopped due to an error appearing during the execution; an orange color indicates that the program is being executed.

  By double clicking in each block, the prompt window shown in **Figure 12** will appear again. This allows the user to execute again a given program or to change the input parameters to get a better result.

- The window below the central window is mainly used to show the outputs of the block that is being analyzed. It has three different tabs: The first tab shows the input and the output files of the block that can be visualized by clicking on the "*Analyze Results*" button; the tab "*Methods*" is used to show the log file generated by SCIPION for each block; the last tab show the "*.stdout*" file that contains all the information about the execution of the block, including the messages indicating the error that stopped the execution. This tab is updated in real time during the execution of the block.

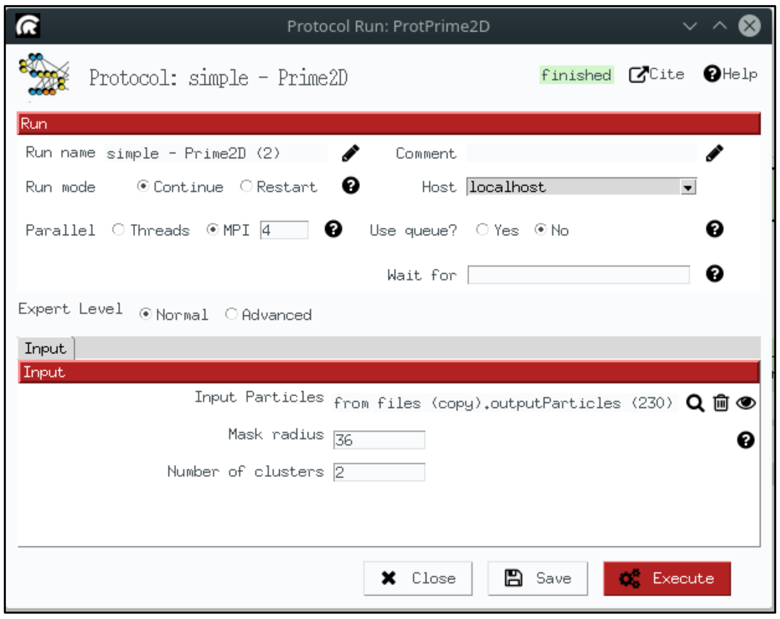


```
▽ Input
    inputParticles   (from from files (copy) -> outputParticles [outputParticles])    SetOfParticles (1619 items, 128 x 128, 2.43 Å/px)
▽ Output
    simple - Prime2D (2) -> outputClasses                                             SetOfClasses2D     (2 items)


  SUMMARY
    No summary information.
```

Figure 11: Example of the GUI for a project in SCIPION.

It is also important to mention that SCIPION is mainly composed of two different type of programs listed below:

- Protocols: These programs are written in Python and they are mainly devoted to managing the execution of the algorithms included in the different packages. They are also responsible of the format handling of the inputs and the outputs
- Libraries: These programs are written in C++ and they contain the algorithms that will be afterwards called by the protocols to perform a specific task in the reconstruction procedure.


Protocol Run: ProtPrime2D

Protocol: simple - Prime2D   finished   Cite   Help

Run

Run name: simple - Prime2D (2)   Comment:

Run mode: [✓ Continue] Restart   Host: localhost

Parallel: Threads [✓ MPI] 4   Use queue? Yes [✓ No]

Wait for:

Expert Level: [✓ Normal] Advanced

Input

Input Particles: from files (copy).outputParticles (230)

Mask radius: 36

Number of clusters: 2

Figure 12: Example of the window used to provide the inputs for a given program.

3.6. XMIPP

X-windows based microscopy image processing package (XMIPP) [6] is a package that provides a set of programs and algorithms aimed to the reconstruction of a 3D model of a molecule from a set of particle images obtained from the electron microscope. This package is currently being developed at the National Center for Biotechnology in Spain by the Biocomputing Unit.

The programs provided by XMIPP are grouped into three different libraries listed below:

- Data structure library: This library includes the programs necessary to create and handle structures like matrices, volumes…
- Classification library: This library includes the programs devoted to the analysis of the data structures to find relationship among them that allows to create clusters (or to classify the data contained in the structures).

- Reconstruction library: This library includes all the programs and algorithms devoted to the reconstruction of a 3D model through the data analysis coming from the images of the EM.

Figure 13 shows the common steps followed to reconstruct a 3D model using XMIPP package. XMIPP package is totally compatible with SPIDER as it shares most of the formats of the files both packages handle. Compatibility with other packages is also possible if the formats are converted appropriately to those accepted by the package. This task is (as explained in the previous subsection) perform by SCIPION itself.

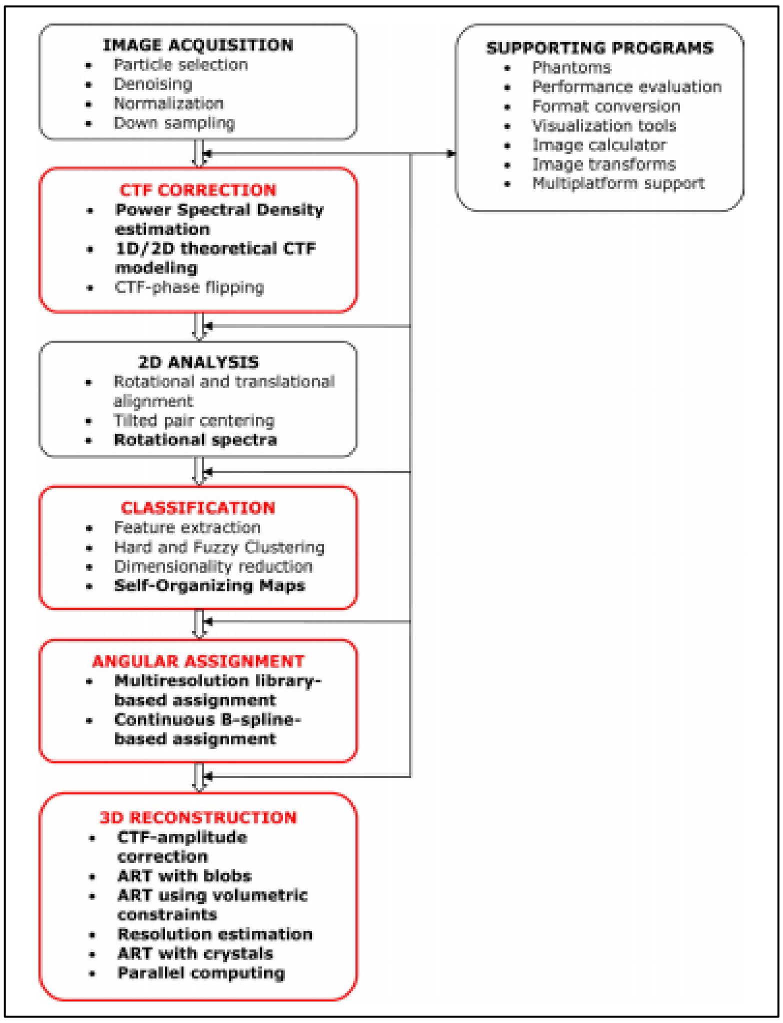


Figure 13: Typical workflow of XMIPP used to reconstruct a 3D model. Extracted from [6].

4. RESULTS AND DISCUSSION

The following chapter explains the results of the projects developed during this work: the implementation and application of a mathematical basis defined in the sphere and the inclusion of SIMPLE package into SCIPION.

4.1. Mathematical basis

The following subsections are devoted to the definition, analysis and application of the mathematical basis and the discussion of the results obtained for this part of the work.

4.1.1. Definition of the basis

Combining the results obtained in the sections 3.1.1 and 3.1.2, it is obtained the definition of the mathematical basis that reads:

$$Z_l^{n,m}(x_r, y_r, z_r, r) = R_l^n(r) Y_l^m(x_r, y_r, z_r) \qquad (4.1)$$

Being r the radial distance to any point of the sphere where the basis is applied. The values of $\{x_r, y_r, z_r\}$ are normalized with respect to the radius according to the following formulas:

$$x_r = \frac{x}{r}, y_r = \frac{y}{r}, z_r = \frac{z}{r} \qquad (4.2)$$

As explained before, the goal of the mathematical basis is to compute the deformation that minimizes the distances between the voxels of two volumes (V_1, V_2) such that:

$$||V_1(\boldsymbol{r}) - V_2(g(\boldsymbol{r}))||^2 = min \qquad (4.3)$$

Being $g(\boldsymbol{r})$ the deformation mentioned before which is described as:

$$g(\boldsymbol{r}) = \boldsymbol{r} + \sum c_{l,n,m} Z_l^{n,m} \qquad (4.4)$$

Where $c_{l,n,m}$ represents the coefficients needed to decompose the deformation in the mathematical basis previously defined.

It is important to mention that the deformation of a volume involves a 3D deformation gradient. In order to be consistent, a coefficient for each of the three Cartesian directions needs to be defined (x, y, z) in order to minimize the distances for each different component.

The behavior of the mathematical basis depends on the degree of the functions used. As the degree of the functions is made progressively higher, the deformation will affect smaller details in the volume. In fact, when the degree of the functions is set to zero, the result is the application of a rigid transformation (rotation+translation) to the volume and the deformation will be more localized as the degree increases.

This is useful as the degree of the functions can be set for each application depending on how locally the basis will deform the original volume. It is important to remind that the problem to be solved is a minimization and, in general, it is not possible to achieve a 0 distance (total equality) between the volumes involved. All in all, it may not be interesting to increase the degree to high levels as it will increase the computational complexity without affecting too much the result of the minimization.

4.1.2. MatLab implementation of the basis

The first step taken after the mathematical definition of the basis was its implementation in MatLab. Although this high-level programming language is not the final language decided to be used for the application of the basis, it provides with several tools that were useful to analysis the behavior of the basis and it made clearer its implementation and functioning.

The implementation in MatLab was carried out using the recursion formulas for the radial Zernike polynomials and spherical harmonics. As it has been seen before, it is easier to handle the basis if it is decomposed into the angular and radial components. For the implementation, this “divide and conquer” strategy was applied again as the result only requires multiplying both components (but keeping in mind the relation between the orders and degrees of both set of functions).

The recursive formulas for both set of functions can be found at [27] and [28]. The main advantage of using the recursive formulas is the decreased in the computational complexity they suppose compared to common implementation using the ordinary definitions.

The MatLab code used to compute the mathematical basis is provided at the end of this document (**Annex A**). The main purpose of the code is to serve as a validation of the mathematical basis to see if it behaves as theoretically expected. To that end, the mathematical basis was represented in 3D and it was compared with the 3D representation of the spherical harmonics to see what the effect of the addition of the radial Zernike polynomials is.

The decomposition of the volume only considers those voxels contained in a sphere of radius:

$$R = \sqrt[3]{\left(\frac{size(x)}{2}\right)^2 + \left(\frac{size(y)}{2}\right)^2 + \left(\frac{size(z)}{2}\right)^2} \tag{4.5}$$

For the final implementation, the radius of the sphere is set by default to that containing the whole 3D volume of the molecule, although the user can modify this parameter according to his application. Here only half of the sizes are used for testing purposes. As explained in the previous section, the value of R will be also used to normalize the values of (x, y, z) as the mathematical basis is defined in the unit sphere.

The results obtained for several combinations of (l, n, m) are shown in **Figure 14**. Thanks to the functions provided by MatLab, the result of the mathematical basis can be rendered in 3D to compare it with the spherical harmonics and the radial Zernike polynomials.

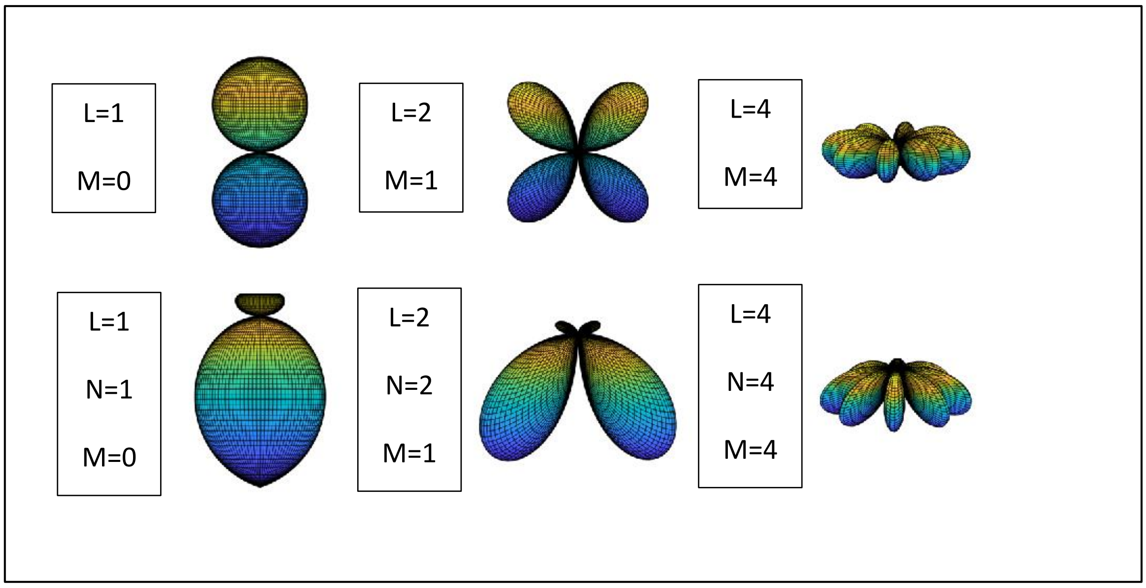


Figure 14: Examples of some SH and the corresponding mathematical basis.

As we it can be seen from the images, the mathematical basis retains the basic features of the spherical harmonics, but with a slight deformation in the radial direction that is more important in the central region of the representation. Moreover, the deformation of the radial Zernike polynomials is more intense as the order is increased due to the oscillations progressively appearing in this set of functions.

4.1.3. C++ implementation of the basis

Once the basis was successfully implemented and tested in MatLab, it was required to implement it in C++, so it can be used inside SCIPION.

For simplicity and efficiency purposes, the implementation did not follow the recursive formulas used in MatLab. It was decided to write directly the tabulated formulas that can be found in [28] and [32] up to 4th order. The main reason behind this decision is that the application of this basis won't usually reach such a high order in most of the cases. In case a higher order is required, it is possible to use the MatLab code described before to find the coefficients of the new formulas (as it was not possible to find any table/document including the formulas of the radial Zernike polynomials and spherical harmonics for degrees higher than four).

The final program requires several inputs to execute. The inputs and outputs of the program are listed (and explained) below:

- Input volume: path to the working volume (i.e. the volume that will be deform towards a reference volume).
- Reference volume: path to the volume used as a template towards which the input volume will be deformed.
- Output volume: (path+) filename indicating where the deformed version of the input volume will be saved. If it is not specified, it will rewrite the input volume data by default.

- Harmonical depth: parameter used to determine the maximum degree of the basis that will be used. As the degree of the basis increases, the number of coefficients $c_{l,n,m}$ will increase accordingly making the program more demanding in terms of computational complexity. This number should not be larger than 4 or smaller than 0. By default, it is set to 1.
- Maximum radius of the transformation: it determines the radius of the sphere used to compute the decomposition of the volumes in the components defined by the basis. By default, is set to the radius of the sphere that circumscribes the volume.

The parameters can be specified through the GUI provided by SCIPION or, in case the program is called through the command line, by using the specific flag for each parameter.

Apart from the outputs, the program also provides with some feedback through the command line. The minimization algorithm used to compute the coefficients of the basis that minimizes the distances between the deformed and reference volumes shows its progression and the values obtained for each coefficient once they are calculated. This algorithm (based on a gradient method to find the maxima/minima of a function numerically) was already implemented in SCIPION.

The execution flow of the program is simple. For each iteration, the value of the coefficients is computed. After the execution of the minimization algorithm, the value of the mathematical basis is obtained for a given voxel and it is used (together with the coefficients) to compute the deformation of the input volume. The value of the deformation suffered by the input volume (in pixels) is computed through the following formula:

$$Def = \sqrt{\frac{\sum_{j,i,k=0}^{n} \lVert g_{ijk}(x,y,z) \rVert}{V_{total}}} \tag{4.6}$$

Where $g_{ijk}(x,y,z)$ represents the result of multiplying the coefficients $c_{l,n,m}$ by the value of the spherical harmonics for each voxel and V_{total} represents the total number of voxels composing the volume. The minimization algorithm tries to make this deformation maximum since minimizing the distance between the deformed and the reference volume is analogous to maximizing the deformation of the input volume towards the reference volume.

The previous steps are repeated until the minimization algorithm finishes and it is also repeated for each harmonical depth sequentially. (i.e. the program is executed for $l = 0,1,2,\ldots$). After a complete execution for a given harmonical depth, the final deformation in pixels is also printed in the command line.

The next sections will be devoted to the explanation and analysis of the results obtained for 3 different examples of the application of the mathematical basis to a real problem: volume to volume deformation, image to volume deformation and strain-deformation gradient analysis.

4.1.4. Volume to volume deformation

One possible application of the mathematical basis is to use it to find the deformation (strain+rotation) needed to reduce as much as possible the distance between a reference volume and a working volume with some similarity as explained in section 4.1.1.

As mentioned in previous chapters of this work, this task was previously developed by means of the NMA. The main drawback of this method is the lack of full automation and that the user should have some knowledge about the analysis to pick the normal modes that best solve the problem. The main advantage of the mathematical basis is that it overcomes these two difficulties, so it yields an output with a lower degree of variability and with no extra user implication apart from the introduction of the reference and working volumes and other parameters explained in the previous section.

Before showing and analyzing the results obtained, it is important to mention that [33] already solve this problem using NMA. In order to test the validity of this new methodology, the same volumes as [33] will be used and the results obtained will be also compared to the ones in the reference. If the mathematical basis works properly, it is expected to obtain similar results to those obtained by [33].

The testing volumes used correspond to a set of eight human mitochondrial ribosomes. This structure can be accessed from the "Protein Data Bank in Europe" web page with the following codes: EMD-1717, EMD-1718, EMD-1719, EMD-1720, EMD-1721, EMD-1722, EMD-1723 and EMD-1724 (**Figure 15** shows the set of volumes in order). The molecules represent slightly different structures (conformations) of the mitochondrial ribosome obtained through electron microscopy imaging techniques.

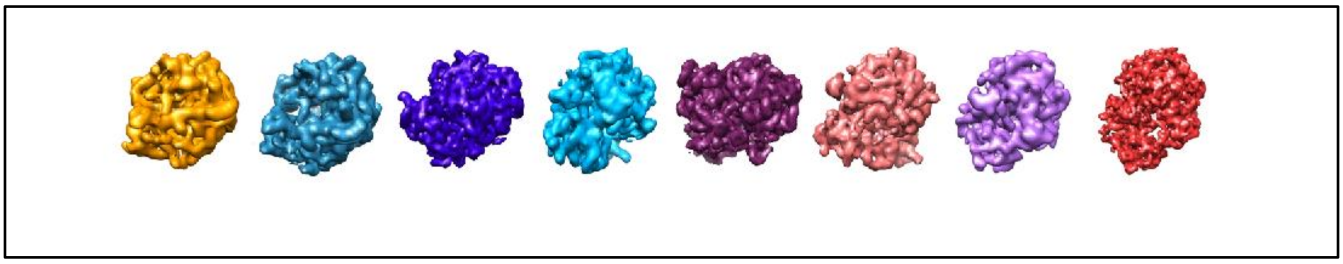


Figure 15: Test volumes used during the analysis. Extracted from [34].

The first step is to align both working volumes through a rigid transformation (rotation+translation). Although this task can be also performed by the basis as explained in previous sections, its main purpose is not to find the rigid transformation that minimizes the distances between the working volumes but to find more localized deformation and rotation gradients. To simplify the application of the basis, the volumes were aligned and normalized with another protocol of SCIPION (although the basis may introduce some deviations from the result of this program in case it is needed to continue with the execution).

After the images are aligned and normalized, the maps were input to the C++ code containing the implemented basis. The input parameters were adapted to save several output volumes obtained by different harmonical depths (the default maximum radius was left untouched).

The test was performed in such a way that each volume was deformed towards the other members of the set and the resulting coefficients and deformations were stored for further analysis. As an example of the test, **Figure 16** shows the result of deforming EMD-1718 towards EMD-1717 (with a harmonical depth equal to 4) together with the comparison of the initial and the reference maps.

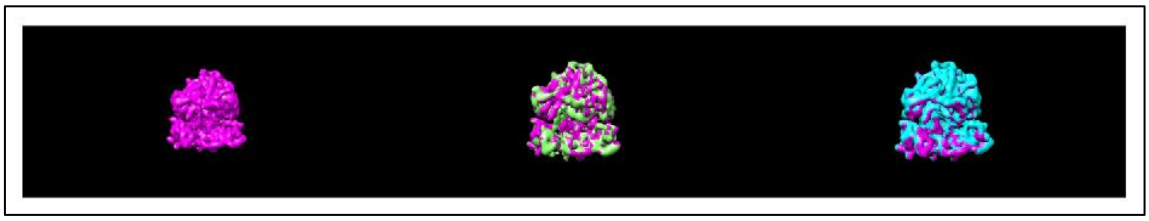


Figure 16: First image: deformed volume; Second image: EMD-1717 (Ref) and deformed volume; Third image: EMD-1718 (Input) and deformed volume.

As it can be seen from **Figure 16**, most of the deformation has occurred in the small subunit of the ribosome while the large subunit remains almost undeformed. This behavior shows that the main difference between the two conformations is the position of the small subunit, so its correction will increase the most the correlation between the deformed and the reference volumes.

The deformation of the input volume and the error committed between the deformed and the reference volume after the execution of the program (both in pixels) were stored in a table to analyze more easily their relation. These results were obtained for the first and second harmonical depths (without considering the zeroth harmonical depth). The tables obtained are included as a supplementary material (**Annex B**) to this work. As it can be seen from the tables, the initial error decreases after the execution of the program and the deformation increases accordingly. Moreover, if the depth is increased to 2 the error and the deformation decreases and increases respectively.

Another important result is that the matrices obtained for the deformation and the errors are not symmetric as the deformation required to reduce the differences between two volumes is dependent on the selection of the input and the reference volume (i.e. the strains and rotations required to deform A towards B are different to the strains and rotations required to deform B towards A).

The final validation for this test is to perform a multidimensional scaling analysis of the results to compare it with the one presented in [33]. **Figure 17** shows the comparison between the analysis obtained at [33] and our analysis.

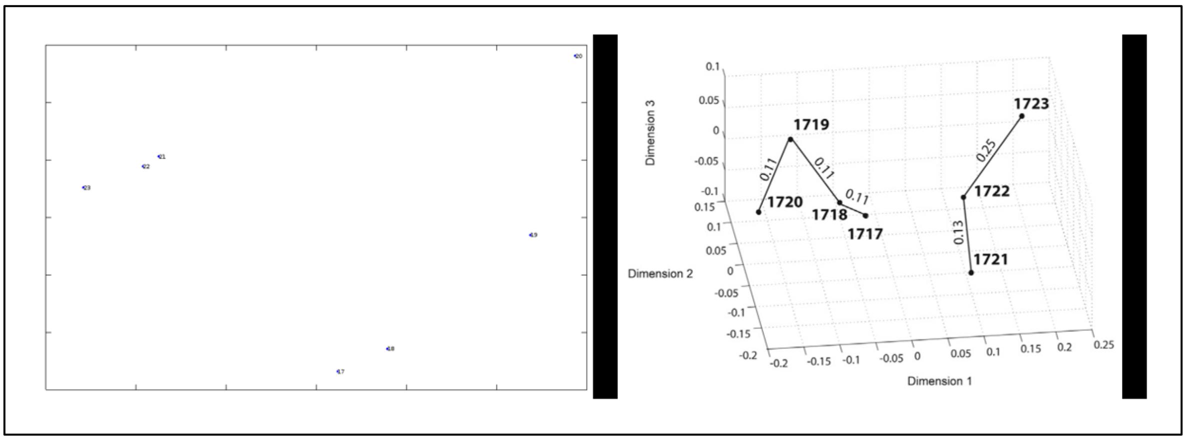


Figure 17: Left image shows our MDS analysis; right image shows the MDS analysis extracted from [33].

As it can be seen from **Figure 17** the MDS analysis obtained with the mathematical basis resembles the results obtained with the analysis extracted from [33]. A group can be clearly seen formed between ribosomes 1723, 1722 and 1721 indicating that they are more closely related among them and a second group formed by the rest of the ribosomes (1717, 1718. 1719, 1720).

Together with the tables mentioned previously it is also included the final distance (in pixels) between each pair of ribosomes. A comparison between this table and the one provided in the supplementary material of [33] was also performed.

Some difficulties were found during this analysis as the tables provided by [33] do not specify the units of the values present in the table. This complicates the comparison as it is not possible to convert our values to their units to find if there is any resemblance of the results obtained. The conclusions derived from the analysis is that the numbers obtained deviate from those proposed by [33] (probably due to the units), although, generally higher distances are obtained for those pairs that are more dissimilar according to the reference table.

According to the two previous results, it is concluded that the mathematical basis has performed well in this application although it should be further tested with real sets of data to asses any possibility of improvement or any source of error that may be present.

4.1.5. Volume to image deformation

The next step is to apply the mathematical basis to the case of having a reference 2D image and a volume that will be deformed towards the reference image. The main objective is to apply the basis in a similar manner to the volume to volume deformation in order to obtain a good result automatically and in a simpler way.

Although the mathematics used are the same to the previous program, it was necessary to modify the scripts in order to adapt them to this new case. Firstly, the new input and output parameters of the program will be listed below:

- Input angular assignment: the input of the program will be a file containing the metadata of the reference image.
- Output: the output of the program will be a metadata file containing the angular alignment and the deformation parameters.
- Maximum shift allowed in pixels.
- Sampling rate of the image.
- Boolean parameters to determine if it is needed to optimize the alignment, the deformation or both.
- A value "lambda" use for regularization (explained lately in this section).

The data set used for this new application was the same as the data set of the previous subsection. This allowed to compare the results obtained before (that were already validate as pointed out in the previous application) with the new ones. In this way, it is expected to get results that resemble those obtained previously. Since now the reference is not a volume but a 2D image, it was performed a random angular assignment for the volumes to get a projection that can be used as a reference.

The execution flow of the program is the following. First, it is needed to compute the projection of the working volume whose correlation is maximum with respect to the reference image. This is only executed if the Boolean parameter "*optimizeAlignment*" is set to true. This is in general advisable as the deformation that will be introduced in the original volume may deviate from the initial assignation of the angles of the projection whose correlation was maximum with the original reference image.

After the previous step, the error between the projection and the reference image is computed. The successive steps are similar to the volume to volume deformation case. The minimization algorithm computes the value of the coefficients in such a way that the correlation/error between the projection and the reference image is maximized/minimized. These steps are repeated sequentially for all the harmonical depths contained between the value introduce by the user and 1 (zeroth harmonical depth is no longer needed as it is not needed to align the volume to anything. The "alignment" step is already performed by the angular assignation method).

As in the case before, the program provides with some feedback in the command line including the output of the minimization algorithm, the deformation of the volume in pixels and the error committed between the projection of the deformed volume and the reference image.

Before showing the results obtained for this part of the project, it is worth to mention what is the purpose of the regularization parameter "lambda" required by this new application. In some cases, the deformations in pixels obtained using the program without the regularization parameter where much larger than those of the volume to volume deformation case (in some case they were and order of magnitude larger). This introduced a problem as it was expected to get similar values for both cases.

Otherwise, the results for both experiments do not have any relation (this is undesirable as the same data set as before was used) thus impacting the reliability of the results.

The main reason of this effect is the lack of constrains in the third dimension. As the algorithm is working with a reference 2D image, it does not know how to deform the volume in the direction perpendicular to the projection. As a result, the correlation is unaffected when the deformation is maximized in that specific direction (as explained in subsection 4.1.3, maximizing the deformation is analogous to minimizing the error between the input and the reference). This led to a volume that was deformed in an expected manner in the directions of the projection but largely deformed in the perpendicular direction (due to the lack of constrains) explaining the values obtained. **Figure 18** shows the behavior explained before.

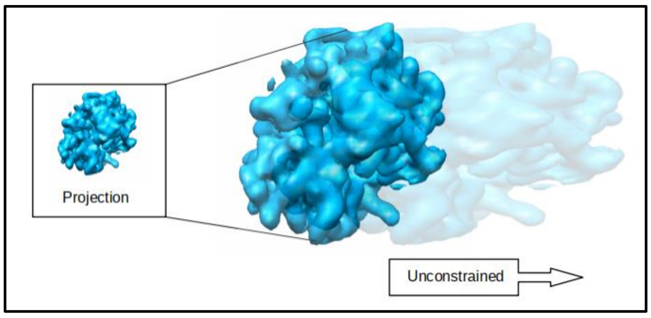


Figure 18: Representation of the unconstrained deformation suffered by a volume

To avoid excessive deformations, extra constrains are included manually in order to reduce the value of the deformation coefficients. Although the final deformation may be smaller than the one obtained by deforming a volume with respect to another volume, the error is not as large as the one obtained previously, and the results were closer to the volume to volume deformation case. The parameter "lambda" serves as a penalization adder to the deformation.

The value of lambda is, by default, set to zero to avoid the penalization. In case it is needed, the value should be around 10^{-2}. Values larger than 10^{-2} can be used but it is important to choose a value that does not halt the maximization of the correlation performed by the program (this means that the importance of the penalization factor is too large compared to the correlation).

Figure 19 shows the results obtained for a projection of the input volume EMD-1718. The reference image corresponds to EMD-1717. As it can be seen from the image, the apparent deformation of the volume is not so obvious. In fact, the total deformation suffered by the volume was equal to 1.914350 pixels. In order to make clearer the changes, the difference between the initial projection of the input volume and the projection of the final deformed volume is also provided. As it can be seen from the image, most of the changes occur in the middle of the molecule and the surface features remain almost unchanged (it is difficult to find any mark indicating a difference in the border of the image that corresponds to the outer most region of the volume). This result resembles the one obtained in the previous application as the deformation was affecting less the surface/details of the volume.

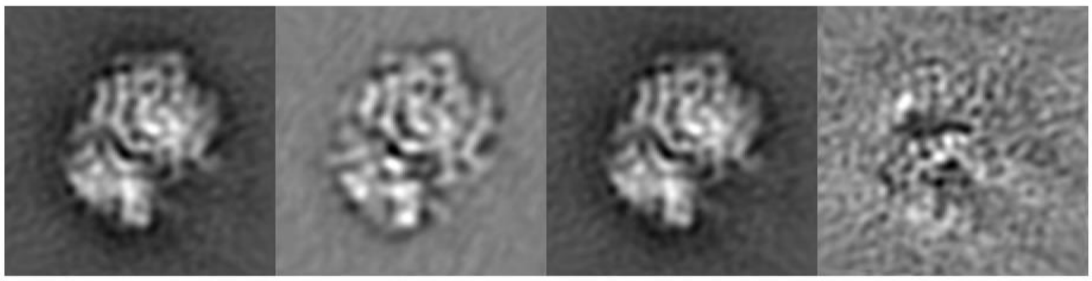


Figure 19: First image: input; second image: reference; third image: projection of the deformed volume; fourth image: difference between the input image and the deformed projection

4.1.6 Deformation gradient analysis

The last application of the mathematical basis is to analyze the deformation gradient of the volumes deformed in the previous applications. In general, the deformation gradient can be decomposed in two different components: one representing the strain suffered by the volume and another for the rotation.

As previously done in subsection 4.1.4, the results obtained will be compared to those provided in [14]. This will allow to validate the output of the program and to get precise conclusions about the results.

According to [14], the strain and rotation gradients can be obtained if the continuous deformation field is known. In the case of this work, the continuous deformation filed is represented by $G(r)$, which is the matrix containing the coefficients (g_x, g_y, g_z). These coefficients can be defined as:

$$g_k = c_{l,n,m} Z_l^{n,m} \qquad (4.7)$$

Where k represents one of the possible directions (x, y, z) in the Cartesian 3D space. Using the definition of the coefficients and $G(r)$, the continuous deformation field can be obtained following the definition described in [5]. The definition is provided below for convenience:

$$G(r) = \left(g_x(r), g_y(r), g_z(r)\right)^T \in \mathbb{R}^3 \qquad (4.83)$$

In order to compute the continuous deformation field, a new input parameter was added to the volume to volume deformation script. The parameter is described below:

- Analyze strain: Boolean parameter used to indicate the program if the values of the coefficients (g_x, g_y, g_z) should be saved to perform a local analysis of the strain and rotations present in the deformed version of the input volume.

Once $G(r)$ is available, the displacement vector field is determined as:

$$u(r) = G(r) - r \qquad (4.9)$$

The gradient of the displacement vector field will be a matrix that can be decomposed into the symmetric and antisymmetric part obtaining the following matrices:

$$\begin{aligned} D(r) &= \frac{1}{2}\left(\nabla \mathrm{u}(\mathrm{r}) + \nabla \mathrm{u}^{\mathrm{T}}(r)\right) \\ H(r) &= \frac{1}{2}\left(\nabla \mathrm{u}(\mathrm{r}) - \nabla \mathrm{u}^{\mathrm{T}}(r)\right) \end{aligned} \tag{4.10}$$

The eigenvalues of the matrix $D(r)$ contain the information of the local strains present in the deformed volume. The value determines the strength of the deformation and the sign the direction. Similarly, the eigenvalues of $H(r)$ represent the rotations.

The analysis was tested with the deformed version of EMD-1720 (using EMD-1724 as a reference) and the results are shown in **Figure 21**.

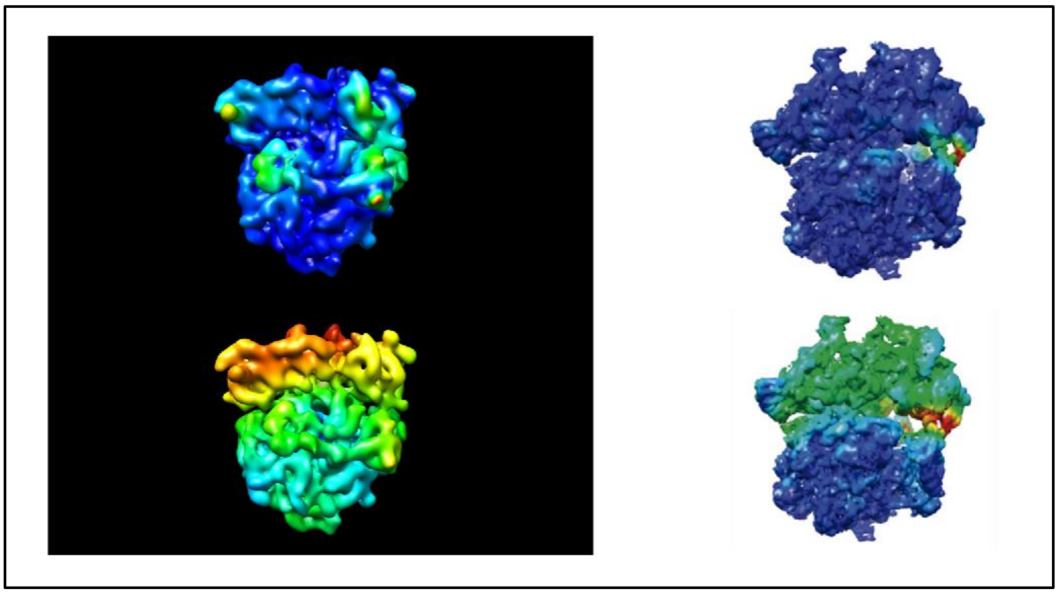


Figure 20: First column of images represents the local strain and rotation analysis (respectively) of our program; second column of images shows the results extracted from [14].

As it can be seen from **Figure 21**, the results obtained after applying the basis led to larger deformations and rotations compared to those obtained by [14]. In addition, the analysis of the local strain and rotations suffered by the molecules shows a more scattered distribution over the molecule (mainly in the strain analysis). It is possible to attribute this result to a more localized behavior of the basis compared to the NMA: while in NMA usually large motions of the molecules are considered to represent a conformational change, the mathematical basis can also analyze the deformation leading to motions of smaller areas in the molecule.

Taking a general view from the results, it is possible to see that the area which is more affected after the deformation is the small subunit of the ribosome and the region surrounding the subunit. This result resembles the one reported by [14], so it can be considered that the behavior of the basis is consistent with the analysis of the local strain and rotations obtained by NMA performed by [14].

4.2. SIMPLE in SCIPION

The following subsections are intended to be an explanation of the steps followed and the results obtained during the introduction of SIMPLE program (developed by Hans and Dominika Elmlund) into SCIPION.

4.2.1. Protocols

Apart from the libraries written in C++ language, SCIPION uses another type of codes written in Python devoted to call a program located in the internal libraries or in an external package. These programs are known as protocols and they are useful to maintain the workflow of execution of a given project in SCIPION.

The purposes of a protocol can be divided into different functions listed below:

- Protocols are responsible of generating the GUI of a specific program. In general, the GUI provides with an intuitive way of introducing the input parameters of a program and, in some cases, it is possible to reduce their number or structure them into different levels (beginner, intermediate, advance…) allowing the users to adapt the execution to their necessities.
- Protocols call a given program through the command line specifying the flags corresponding to the different input parameters appropriately.
- Protocols convert the output and input formats of the files between those used by SCIPION and any other external package. In general, different packages developed for EM reconstruction applications used their own format to store the data. SCIPION tries to mediate between different packages so the input/output format files are converted to those that are understood by each different package.
- The protocol is also responsible of managing other options like the parallelization of the CPUs to improve the efficiency of a program among others.

The documentation of SCIPION [35] provides with an explanation of the structure of a protocol and the purpose of each of the different parts that composed this type of programs.

4.2.2. SIMPLE protocolization

The main objective of this part of the work is to include the software SIMPLE into SCIPION using protocols. SIMPLE is a package composed of several scripts that perform different activities directed toward the reconstruction of a final volume starting with the micrographs coming from the EM.

The first step in the protocolization of an external package is to understand the execution and functioning of the different programs that it includes.

Although there is available a detailed explanation of the workflow of SIMPLE in the documentation web page [31], it is included here a summary.

Most of the micrographs obtained by the electron microscope are taken with low energy electron beams. This generates noisy images with no appreciable particles in the image. Although increasing the energy of the electrons used to interact with the samples would increase also the SNR, it may also damage the sample. In order to overcome this problem, the EMs are able to take movies of the sample instead of individual images. As the noise has a stochastic behavior (i.e. it appears in different places when comparing two images of the same object taken at different moments) and the sample is not moving, the frames of the movies can be used to increase the SNR without affecting the integrity of the samples. **Figure 21** shows an example of a raw frame of a movie and the final results after processing the whole movie.

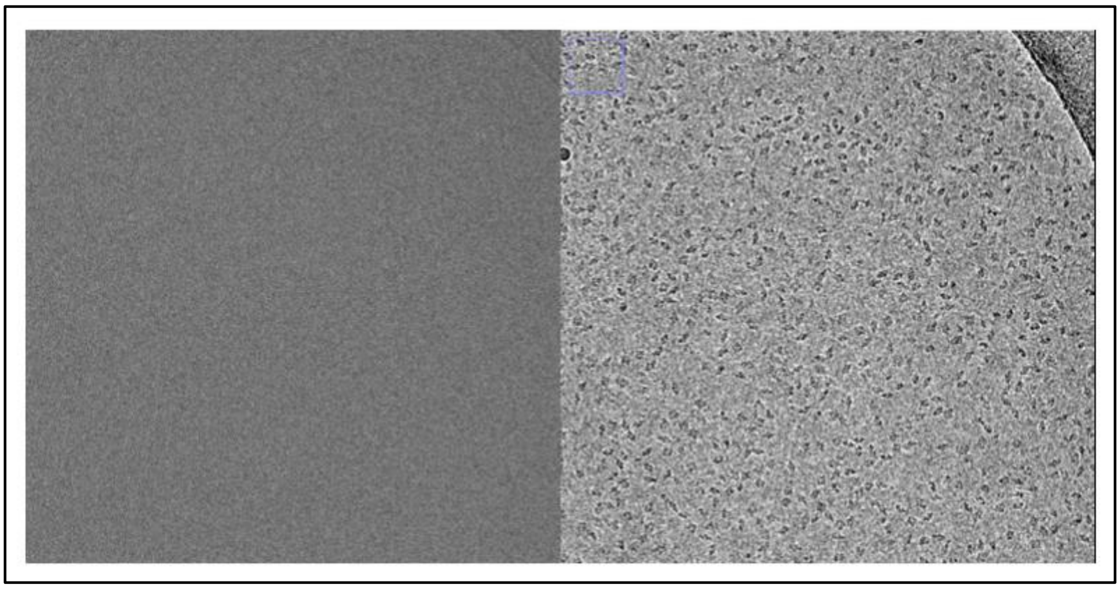


Figure 21: Example of an unprocessed and processed micrographs.

SIMPLE offers a program called "unblur" that uses and algorithm to process the frames of a movie to increase the SNR. Moreover, it can also remove the motion artifacts that may appear in the during the acquisition of the images.

Once the micrographs have been processed, the particles present in the image are extracted. This involves two different steps.

Firstly, it is needed to identify the coordinates of the particles in the set of micrographs obtained. SIMPLE does not include a program to perform this task, but it relies on EMAN package to pick up the coordinates corresponding to the particles.

Picking the particles can be done manually or automatically. The automatic version of the extraction requires the user to pick several particles (the higher the number the better the result) that the program will use to sweep the whole micrograph trying to find regions of high similarity to the reference images provided by the user. EMAN store the coordinates of the particles in a "*.box*" file.

The coordinates are then used to extract the particles in individual images. SIMPLE uses the program "*extract*" to perform this task.

After the extraction of the particles, it was obtained a set of images that are still noisy. In fact, it is not possible to determine the structure of the particles in the image as the shape it is not completely clear. **Figure 22** shows an example of the particles obtained after the extraction process of a process micrograph.

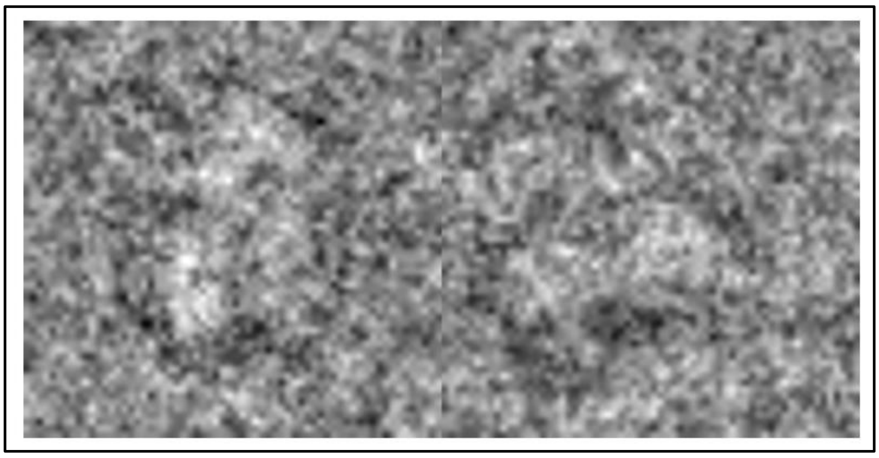


Figure 22: Example of two particles extracted from a micrograph.

The images extracted correspond to several projections of the same particle seen from different perspectives. Although the projections may not be the same, it may be possible to find several particles representing the approximate same relative projection of the particle. By grouping these particles, a set of 2D class averages is created. In the same way as it was done with the movies, by grouping several images of the "same" particle projection together the SNR is maximized, making the shape of the molecule more obvious. SIMPLE includes a program called "*PRIME2D"* capable of perform an angular alignment of the projections that are similar to create the set of 2D class averages. **Figure 23** shows an example of the output of the program when a set of particles is provided as an input.

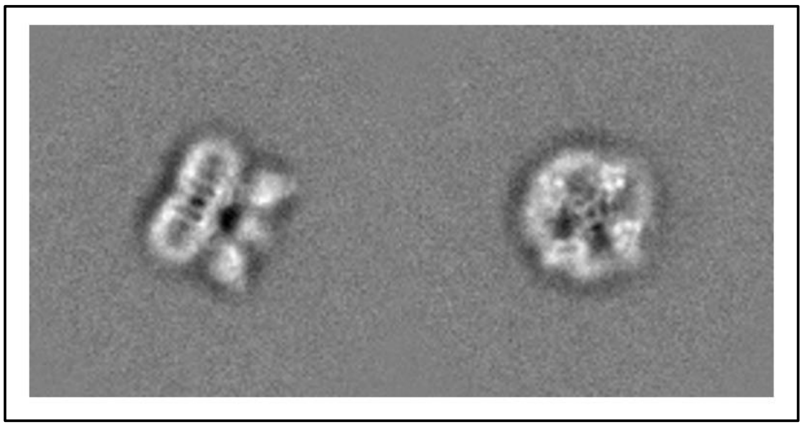


Figure 23: 2D class averages from PRIME2D (TRPV1).

The information provided by the 2D class averages can be used to reconstruct the final 3D volume of the molecule.

Although there are several methods to reconstruct a 3D volume out of a set of projections, the most commonly used relies on the FT and the Fourier slice theorem [36]. This theorem postulates that there is a relation between the particle images obtained before and the 3D volume. In fact, if there exists a projection of the 3D volume as seen along a given direction perpendicular to the projection that corresponds to a given particle, then the values of the FT of the particle image will be equal to the values of the FT of the volume along a given slice.

In order to reconstruct a volume, a large set of different 2D class averages are needed. In this way, there will be enough projections corresponding to different directions to reconstruct the volume.

The main difficulty in the reconstruction is the assignation of the orientation corresponding to each projection (this is analogous to assign the vector perpendicular to the projection to each 2D class average). In general, this orientation is unknown. The most common way to proceed is to use a template 3D model (that may be obtain from a similar and already reconstructed molecule or by deforming this template map towards the 2D class averages using the method developed during subsection 4.1.5) to get an initial approximation of the orientations. These orientations are used to reconstruct an initial 3D model that will be used as a new template for the following iteration. This procedure continuous until the 3D model deviation from the template is small.

SIMPLE includes a novel algorithm to get an initial 3D model using only the class averages obtained through PRIME2D. This "*ab initio 3D reconstruction"* method (distributed under the name "*ini3D_from_cavgs*") [29] relies on a probabilistic approach to determine the orientation of the images. If a 3D template is used for the reconstruction, the assignment of the orientation for a given image is deterministic as the algorithm finds the direction that provides the maximum correlation between the class average and a given projection of the template. In a probabilistic approach, the algorithm assigns to each image a weight for a range of possible directions that provides a high score (the assignment of the score and directions is performed by a hill climbing optimization method). Then, the final reconstruction is just a weight average of the class averages assigned to the different directions.

Figure 24 provides an example of the map of a ribosome obtained using "ini3D_from_cavgs".

The following subsections explain the process of protocolization of the programs "*unblur*", "*Prime2D*" and "*ini3D_from_cavgs*" so they can be used as an external package in SCIPION.

The main reason why all the programs included in SIMPLE were not ported to SCIPION is that most of the functions of the programs are already implemented by other algorithms inside SCIPION. Only those performing new functions and/or similar functions but using novel approaches compared those already present in SCIPION were protocolized.

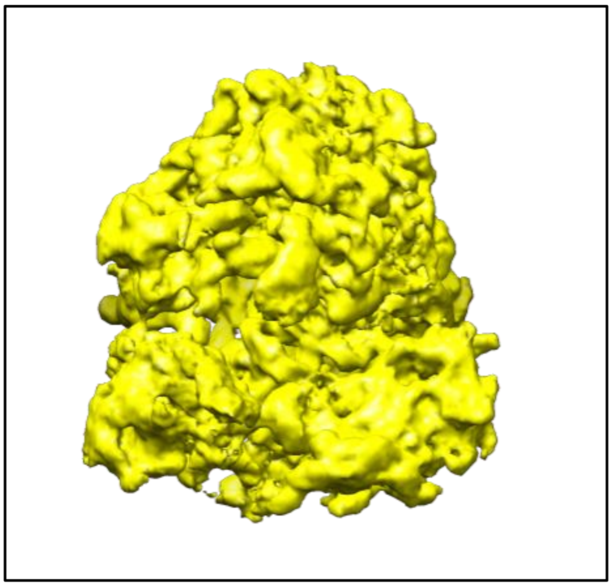


Figure 24: Ab initio reconstruction of a ribosome through Prime3D.

4.2.3. Unblur

The first step in any protocolization is to define the dependencies and the input parameters that will be included in the GUI. The input parameters of *unblur* are available at [31].

As it can be seen from the documentation, there are many input parameters to be defined to call the program *unblur*. One good practice when writing protocols is to keep the call of a program as SIMPLE as possible in such a way that a normal user can execute the program without difficulties.

Among all the possible inputs present in the list, the user is only allowed to introduce the following parameter in the GUI:

- InputMovies: absolute path to the movies that will process by *unblur*.

This simplifies the GUI, making the program easier to use. The rest of mandatory inputs will be obtained during the execution of the program. The sampling rate can be obtained directly from one of the frames present in the movies and the partitions will be set by default to one. Among the optional parameters, only use "*fbody*" will be used to determine the final name of the files. This name is the combination of the name that gives by default SIMPLE to the output files and the name of the original movies.

In the input section of the protocols it is also asked if the user wants to apply parallelization to improve the efficiency of the program. This means that it is possible to operate different sections of the program or different micrographs using one out of all the available threads that the computer has. SCIPION handles by itself the parallelization of the programs when this option is present in the protocol (if parallelization is not desired, it will be omitted from the protocol). The only requirement is that the user determines in the GUI if he wants to use the option "*threads*" or "*MPI*" to parallelize the process.

Once the input parameters are defined in the protocol, the different steps that will be executed are defined. The function "*insertAllSteps*" will be responsible to execute the different steps defined in the order they have been written written them inside this function. It can be seen as the function "*main*" in C++: it should not contain the definition of any function/method but to call them. It is also possible to defined parameters that are required by the steps inside the function "*insertAllSteps*".

In the case of *unblur*, there is only one step that will be called: "*unblurStep*". The main function of this step is to get the input parameter defined by the user and create the "*.txt*" file required by *unblur* known as "*filetab*". This file contains the path to the micrographs that will be processed by the program. The step also defined the string that will be passed to the command line to call and execute "*unblur*" with the following parameters:

- Filetab: defined before.
- Smpd: sampling frequency of the movies. It is determined automatically by analyzing one frame of the input movies.
- Nparts: number of partitions in distributed execution. It is set to 1 by default.
- Fbody: determines the name of the output file. The name is automatically generated by the protocol and it is a combination of the name of the input movies and the name SIMPLE gives by default to the output files.

Once the string is defined, the program *unblur* is called and executed.

Most of the programs in SIMPLE provide with some feedback to the user through the command line (where they are usually called). SCIPION can get this feedback and append it to a file with the extension "*.stdout*". This file can be seen from the GUI and it allows to keep organize all the data provided by SIMPLE (and any other program).

One problem found during the execution of *unblur* was that the parallelism could not be apply directly to improve the efficiency of the execution. This difficulty was coming from the way SIMPLE was design. The execution of SIMPLE does not contemplate the possibility of CPU parallelism as all the outputs the program generates are append to a file called "*nohup.out*". As the name of the file is always the same, if CPU parallelism is used, all the outputs will be appended to the same file leading to wrong results. SIMPLE admits the introduction of different inputs as long as they are executed in a sequential manner making the process time consuming if the user needs to process a large set of data.

The solution found to this problem was to define for each input a different working directory. In this way, each CPU will work with a "*nohup.out*" with a different path so the files will be independent of each other. The working directories were created inside a temporal directory that is emptied as soon as the execution of *unblur* finishes. Before the removal of the files, the outputs that are more interesting are moved to an extra directory, so they can be handled after the execution. The files are also renamed during their movement to give them the appropriate format described before.

Using this methodology, it is possible to take advantage of the parallelization handling of SCIPION inside the package SIMPLE without complications or errors in the output.

As explained in the previous sections, SCIPION is mainly devoted to handle the data coming from different programs and to adapt their format, so they can be used by any other package. This essential function of SCIPION is determined in the protocols in a step called "*createOutputStep"*. This also allows SCIPION to "see" the outputs generated by a program. The step is only executed when all the jobs running are finished so no outputs are lost during the process.

In order to create this step, it was first needed to define an object (among all the ones defined in SCIPION such as volumes, movies, class averages…) that matches the type of output data generated by a program. In this case, the input of the program is a set of movies and the output is a set of micrographs, so we it was required to create a "mic" type object.

Any object in SCIPION has a series of fields that need to be filled up in order to define properly the object (so it can be seen and used by SCIPION). In the case of a micrograph, the name of the file and the sampling rate need to be defined among others.

The last part of the step is to stablish a relation/dependency between the input and the output data.

Apart from the part defined previously, a protocol may have other sections that can be found at [30]. For the protocols defined in this work, it was included also a "*citation"* section, so the user can go directly to the publications related to SIMPLE.

Figure 25 shows an example of the output of the protocol after its execution. The input movies are not provided as they are noisy, and it is not possible to extract any information nor differentiating them. An example of a frame of a raw movie was shown in **Figure 21**.

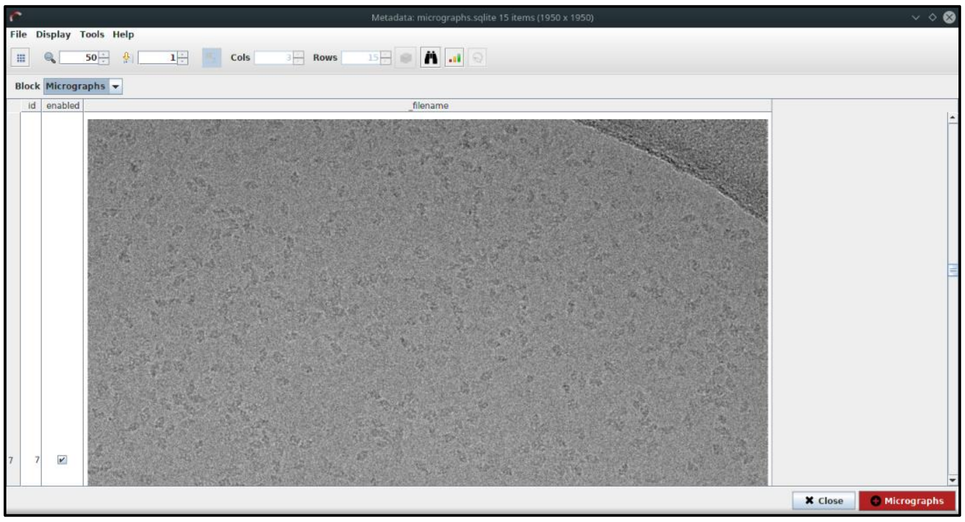


Figure 25: Example of the output of unblur after the execution of the protocol.

Apart from the protocol itself, the protocolization of any program requires also a test. A test is just a program that calls a protocol and provides the parameters necessary to execute it.

The output of a protocol may not be a good result. The only purpose of a test is to serve as an easy way to call the protocol trough the command line, so the developer can analyze the errors present during the execution to isolate them and correct them more easily. The test is just a tool for the developer, and it should not be used by the user to call the program. The test also creates a new project in SCIPION where all the outputs are saved in order to manage them more easily.

Once the output of the test is correct and it does not show any error, it is also a good practice to execute again the protocol but now through the GUI of SCIPION. The main purpose of this execution is to simulate a real scenario with a real set of data to analyze the quality of the results. It is also useful to identify errors associated with the GUI or the management of the outputs by SCIPION.

The output of *unblur* is composed by four files described below:

- The file ending with "*_intg1.mrc*" is the binary of the processed movie with the correction for the motion artifacts and the increased SNR coming from the weighted average of the frames. This file has been renamed to end with "*_psd.mrc*" to match the nomenclature used by SCIPION.
- The file ending with "*_forctf1.mrc"* is the binary file of the processed movie by applying a simple average of the aligned frames of the movie.
- The file ending with "*_pspec1.mrc*" is the power spectrum of the first (left) and second (right) images that the program yield as an output. This file has been renamed to end with "*_psd.mrc*" to match the nomenclature used by SCIPION.
- The file ending with "*_thumb1.mrc*" is a down-scaled version of "*_intg1.mrc*".

It was decided to keep only the last two files to simplify the output of the program.

4.2.4. Prime2D

Firstly, it is needed to define the input parameters of the protocol as done in the previous section. As a reminder, the input parameters of *Prime2D* are available at [31]. The parameters are listed below:

- Input particles: Path indicating the location of the stack of particles extracted from the micrographs. As explained before, this can be done with external packages like EMAN. SCIPION already includes protocols to extract the particles in a manual and an automatic way that can be used to obtain the stack of particles.
- Mask: This parameter is used to define the radius of the mask (in pixels) that will be apply to the particles. The mask is used to process the particle images to remove the noise, so the program can handle them more easily. By default, it is set to 36.
- Clusters: Indicates the number of class averages that will be obtained at the end of the program (i.e. how many conformations will be taken into account to group the images in a single class average). By default, it is set to 5.

- MaxIter: Indicates the number of iterations the program can execute to get the class averages. This parameter is set to advance level as choosing a number higher/lower than the optimal one may lead to an increase of the computational complexity or to a wrong result. By default, the number is set to 0 to indicate the protocol that *Prime2D* will choose this number by its own internal algorithms.

As done in the previous subsection, the remaining parameters will be set automatically either by internal functions of SCIPION or by the default values determined by SIMPLE itself. This helps to reduce the complexity of the program.

The function "*_inserAllSteps*" executes three different steps in this protocol apart from the initialization of the variables that may be required to execute the before mentioned steps.

Firstly, the step "*_convertInput*" is called. This step is commonly used inside the protocols. In the previous sections it was explained that SCIPION is mainly devoted to handle files and binaries written in several formats and to adequate these files to the execution of other packages. All these jobs are declared inside this step, when the input of the protocol is not expected to be written in an appropriate way to call a given program.

The purpose of this step inside this protocol is to write a single stack of particles out of all the files the user may specify as an input parameter. Since it is expected that all the particles provided by the user belong to the same molecule, they can be grouped to work more easily.

The next step is "*prime2DStep*" that is analogous to "*unblurStep*". The main function of this step is to retrieve all the data necessary to call *Prime2D* through the command line. Apart from the input parameters specified by the user, there are another two mandatory inputs that will be defined in this step. They are listed below:

- Smpd: As before, this parameter is obtained directly by analyzing one image from the stack of particles created before. The sampling rate needs to be the same for all the particles specified in the input, otherwise the output of the program will include errors.

- CTF: This parameter indicates if the particles need to be corrected by means of the CTF (Contrast Transfer Function) of the micrograph. This parameter is always set to "*no*" as this step is already done by other algorithms inside SCIPION. In case the any correction is needed, the protocols containing these algorithms should be called before the execution of *Prime2D* and their output would be fed as an input to this protocol.

Once all the required parameters are available, the step writes the string that will be used in the command line to call and execute *Prime2D*.

In order to improve the efficiency of the program, it was also applied CPU parallelism. In this case, it is not necessary to include the trick used with *unblur* as the program works with a single stack of particles (coming from the unification of all the inputs specified by the user). Anyway, it was decided to move the working directory to the temporal file to automatically remove those outputs that are not interesting. The application of CPU parallelism is still a good practice as SCIPION can use it to execute more efficiently loops or other structures that admit parallelization.

Once the output is available, the step "*_createOutputStep*" is executed. As before, it is needed first to define the type of object that best represents the output data. In this case, the program retrieves a set of 2D class averages. This object does not only contain the output 2D averages created by *Prime2D,* but it also stablishes a relation between the class representative and the particles used to create that representative. In order to define the object, it was needed to extract the rigid transformation (translation in (x, y) directions and the rotation) used to align each particle to create the representative, and the representative the micrograph belongs to.

Prime2D includes a "*.txt*" file with all this information, so it is possible to use the file to extract the data needed to define the object. SCIPION includes already several functions to analyze files containing this type of information but, as explained in other sections, the lack of consensus makes impossible to have a way to analyze each different file with its own format. For this case, it was needed to create a new function to analyze the files generated by *Prime2D*. This function is simple, as each line of the file correspond to the information of a given particle image. The function opens the file and reads each line to extract the data defined with the flags "*class*", "*x*", "*y*" and "*e3*" that contains the information of the class representative, x translation, y translation and the angle of rotation respectively. The information is saved in a dictionary structure to access it more easily.

The information extracted is used to fill up the class. The last step is to stablish the relation between the input and the output. **Figure 26** shows an example of the output of the protocol for a set of particles.

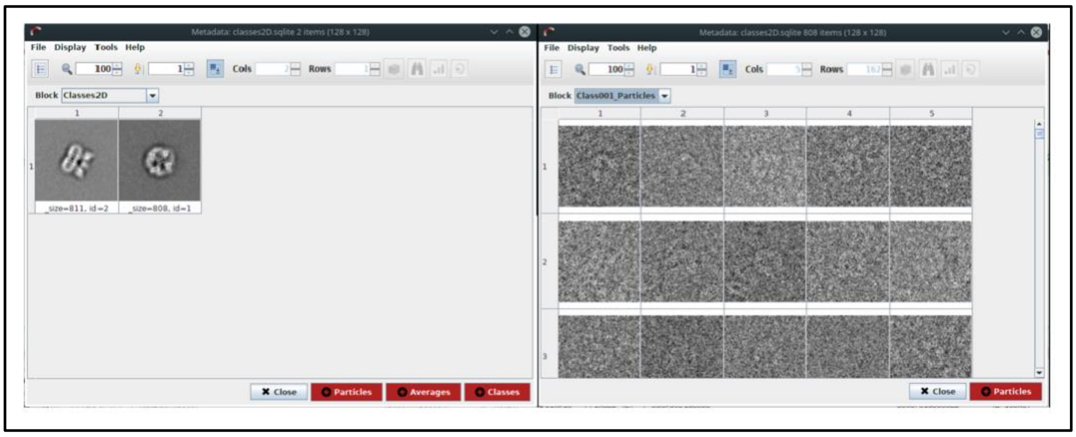


Figure 26: Output of the protocol written for Prime2D. First image: class representatives; second image: particles used to create the first representative.

As it can be seen from **Figure 26**, the GUI includes two red buttons with the labels "*Averages*" and "*Classes*". This allows to create new set that can be used as an input for other programs. The main different between a set of averages and a set of classes is that the set of averages only contains the class representatives.

A test was also created to manage more easily the errors that appeared during the development of the protocol.

4.2.5. Prime3D (ini3D_from_cavgs)

The next protocolization corresponds to the program *"ini3D_from_cavgs"* that will be referred as *Prime3D* during the subsection for simplicity. The inputs required by the protocol includes:

- Input classes: Path indicating the location of the 2D class averages obtained by *Prime2D*. As mentioned in the previous subsections, it is possible to save the class averages as a set of classes or as a set of averages. The input was defined to allow the user to introduce both types of objects as an input.
- Mask: Mask radius (in pixels). The mask is used to process the class averages to remove the noisy they may have.
- Symmetry: In order to reconstruct the volume of any molecule, *Prime3D* requires the specification of the point group symmetry of the molecule. A point group symmetry is "a group of geometric symmetries (isometries) that keep at least one point fixed" [34]. For example, a cube has octahedral symmetry because if the center of the cube is chose as a fixed point, there are a total of 24 axis of symmetry crossing that specific fixed point as shown in **Figure 27**. In the same way, it is possible to classify the molecules according to their symmetries that cross a specific point. *Symmetry* parameter determines a range used by *Prime3D* to search automatically the symmetry of the molecule (e.g. if symmetry c5 is selected, *Prime3D* will evaluate symmetry groups ranging from c1 to c5, selecting the most appropriate one). It is also possible to specify *Prime3D* to assume the point-group symmetry provided as an input as the correct one through a Boolean parameter.

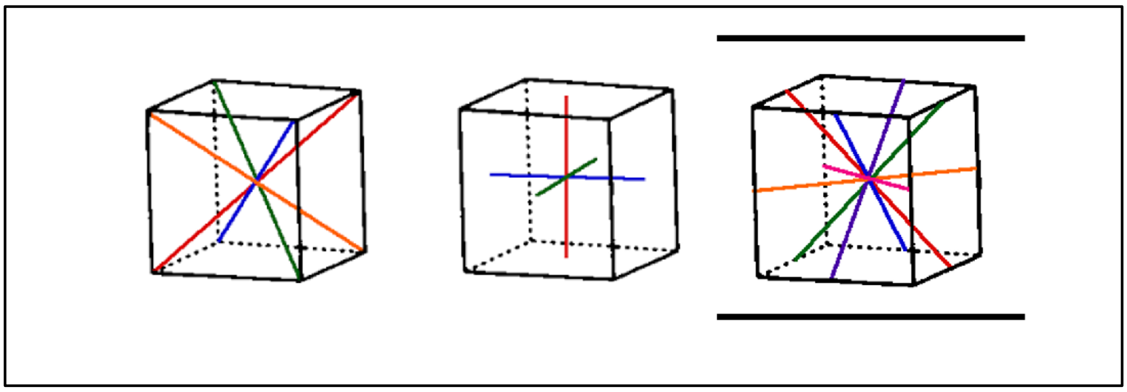


Figure 27: Symmetry axis of the cube crossing the center. Extracted from [37].

As in the previous cases, it was decided to exclude those inputs that may complicate the execution of the program. The list with all the inputs for *Prime3D* is also available at [31].

The function "*_insertAllSteps"* executes again three different steps as in the previous protocol.

The first step is a "*convertInputStep"* that has the same objective as in the case of *Prime2D*.

The user may introduce the path to several files containing the class averages corresponding to the same molecule. In order to handle this data more easily, it was decided to create a new stack of 2D class averages that joins all the images present in the input files into a single object that facilitates handling the data inside the protocol.

The following step is called *"init3DStep"* that is also used to retrieve and define all the data required to call *Prime3D* through the command line. Parallelization was also used with a similar objective as in the *Prime2D* protocol.

The step whose definition is more interesting from the protocol is the last step: *"_createOutputStep"*. In this case, it was needed to define a new object which is either a volume or a set of volumes. The output of *Prime3D* is a series of binaries containing the information needed to represent the volume using the appropriate viewers. These binaries correspond to the different iterations the program has performed to reach the final *ab initio* 3D model. Although it was decided to save only the volume corresponding to the final iteration, it is also possible to create a set of volumes with all the models generated after each iteration.

A volume object in SCIPION only requires two fields to be filled up to be considered a properly defined object. This fields are the path to the volume to be saved, and the sampling rate obtained from the class averages. As always, it was needed to stablish the relationship between the input and the output to finish the step.

Figure 28 shows the result obtained after the execution of the protocol created for *Prime3D*.

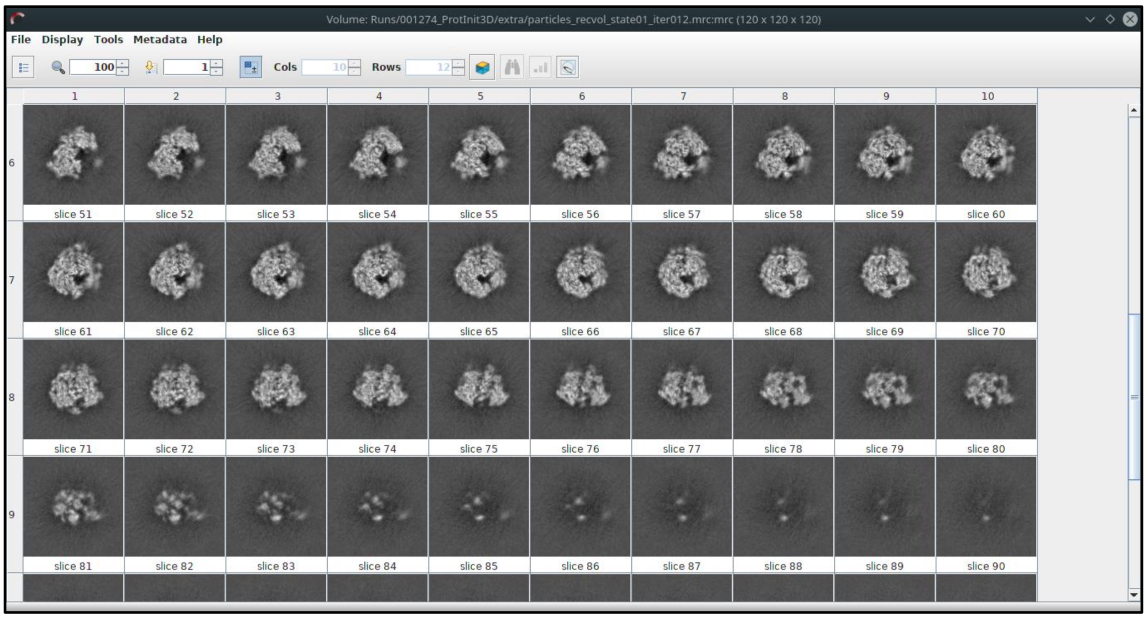


Figure 28: Output of the protocol written for Prime3D.

A test was also defined to simplify the execution and error handling during the development of the protocol.

5. SOCIO-ECONOMIC IMPACT

Structure analysis is useful in several fields highlighting pharmacology and cell biology. The introduction of the mathematical basis explained along the work supposes a new methodology to understand the behavior and interactions of biomolecules. The analysis of the output of the algorithm can be applied to the development of new pharmaceutical to treat diseases (as it is possible to simulate the changes that may be present during the interaction of the pharmaceuticals with their receptors (drug–receptor interactions), leading to an increment in the knowledge and understanding of the operation of the molecules). Moreover, the increment in the knowledge of different molecular entities will translate in a higher understanding of the complex mechanisms taking place in the cells and their functions.

In the case of SIMPLE, its implementation as part of SCIPION will simplify the reconstruction of the volumes obtained by processing the micrographs coming from the EM. The *ab initio* algorithms included in SIMPLE are of great use when the data set obtained from the microscope cannot be related to any other already reconstructed volume (3D model), that will provide with an initial assignment of the angles needed for the reconstruction. In combination with the algorithms present in SCIPION, SIMPLE will provide with extra powerful tools for those cases where standard algorithms do not retrieve the output expected.

6. CONCLUSION AND FUTURE WORK

6.1. Conclusion

Nowadays EM techniques are extremely useful to analyses molecular entities. The images coming from the microscope suffer an extensive postprocessing, directed towards the reconstruction of 3D models of the samples. The algorithms used to achieve this goal are complex and, in some cases, there is the need to have *a priori* information coming from already known molecules to obtain the results.

Among the different steps applied during the reconstruction of the volumes, angular assignment is essential for the reconstruction of the samples from the 2D class averages of the different particles retrieved from the micrograph. In general, angular assignment depends on the information obtained from related molecules already reconstructed. These molecules will be used to get an initial guess of the projection angles of each class average in the final volume.

The mathematical basis introduce along the work can be applied to deform 3D models of molecules towards another related volume or a set of related 2D class average. This will improve the initial angular assignment of the projections, (as the reference model will be more similar to the final output of the reconstruction) leading to an increase in the accuracy of the results and a decrease in the computational complexity of the reconstruction process.

Although the information coming from a reference volume is highly used in the reconstruction process, it may not be possible to find a good initial reference to assign the projection angles. *Ab initio* methods can be used when this is the case. The algorithms implemented in SIMPLE are directed towards an efficient *ab initio* reconstruction of the volumes, solving the problem found when reconstructing a volume from a set of class averages with no obvious relation with other molecular entities.

The implementation of both, SIMPLE and the mathematical basis, are of great use in the analysis of molecules in cell biology and pharmacology, as they provide with powerful tools to obtained and analyze the interactions and conformations of the biomolecules. These analyses will increase our knowledge and will improve the understanding on the handling of cellular processes.

6.2. Future work

Future steps in both projects (SIMPLE inclusion in SCIPION and development of the mathematical basis) should be directed towards the optimization and testing of the algorithms.

All the programs generated during this project were intended to run using the CPU (Central Processing Unit) of the computer. Nowadays, many computers incorporate GPU (Graphics Processing Unit) processing unit apart from the CPU. The main advantage of GPUs over CPUs alone is the possibility of using the GPU of the computer to perform graphical or float operations, while the CPU is involved in other processes of the system.

The combination of both units greatly increases the performance of the programs, reducing their computational complexity. The scripts created should be optimized so they can be used with GPUs.

Apart from the optimization, it is also needed to test the algorithms applied by SIMPLE and the mathematical basis in a wide range of situations and data sets. This will suppose a good stating probe about the performance and efficacy of the programs to ensure a proper functioning.

Focusing on the mathematical basis, it would be helpful to perform a multivariate analysis of the deformation coefficients retrieved after the execution of the programs. The main objective is to use the aforementioned analysis to estimate the energy landscapes of the input volumes/molecules to get some insights about all the possible conformations of the involved biomolecules.

In the case of SIMPLE, it will be needed to keep updated the different scripts included in SCIPION, as well as adding new functionalities that may be released in future version of the package.

7. REGULATORY FRAMEWORK

The techniques introduced along this work are not regulated by any legislation or subjected to any intellectual property and they do not break any code of professional ethics. However, several packages and programing languages were used for the development of the aforementioned techniques with their own regulations.

Scipion is under the GNU General Public License, which is a free, copyleft license for software. In the case of Simple, the protocolization was done under the confirmation/knowledge of the developers (Hans Elmlund - Monash University).

In the case of the programming languages used for the development of the scripts, both, C++ and Python are open source, although the IDEs (Integrated Development Environment) used may be subjected to regulatory frameworks. The IDE used for C++ codes was Eclipse, whose foundation has its own open source license (EPL) similar to the CPL (Common Public License) but with some modifications. In the case of Python, PyCharm was preferred which also has its own agreement (which approves its usage for academic research).

The regulations of MatLab are under the academic license, which supports academic research but not commercial purposes (the codes developed for this work were not intended to be included in the final product).

Scipion releases are distributed for Linux OS (Operative System) which is also an open source software under the GNU license.

8. WORKFLOW OF THE PROJECT

The following chapter is intended to be a brief description of the different task develop during the project to achieve the final goals. For simplicity, two tables are provided for the description of the workflow followed during the development of the mathematical basis (**Table 1**) and the implementation of SIMPLE into SCIPION (**Table 2**). All the tasks are shown in the order of execution. It is worth to remind that, before starting the project, it was needed to acquire the skills necessary to program fluently in Python and C++ programing languages.

Workflow: Mathematical basis	
Mathematical Definition	Understanding of the functions involved and elaboration of the tool. This work was done together with the Dr. Roy Lederman from Yale University
MatLab implementation	Assessment of the mathematical basis through a MatLab script.
C++ implementation	Introduction of the mathematical basis inside SCIPION as a library.
Volume to volume deformation (implementation + testing)	Implementation of the algorithm in C++ required to deform an input volume towards a reference volume. The script was introduced as a library of XMIPP package (developed at the National Center for Biotechnology).
Image to volume deformation (implementation + testing)	Implementation of the algorithm in C++ required to deform an input volume towards a reference image (particle/projection/class average). The script was introduced as a library of XMIPP package (developed at the National Center for Biotechnology).
Deformation gradient analysis (implementation + testing)	Implementation of the algorithm in C++ required to analyze the local strain and rotational fields applied to the molecules stored in the deformation coefficients. The script was introduced as a library of XMIPP package (developed at the National Center for Biotechnology). The implementation of the rotational filed and final testing was done by Carlos Óscar Sánchez Sorzano.

Table 1: Workflow followed during the development of the mathematical basis

Workflow: SIMPLE protocolization	
Protocols in SCIPION	Understanding and training on the development of protocols to call packages or libraries in SCIPION.
SIMPLE algorithms and workflow	Understanding of the tools provided by SIMPLE for the reconstruction of volumes and the normal execution workflow to achieve this task.
Testing of SIMPLE programs	Analysis of the execution of the programs executed by SIMPLE through the command line (useful to develop a strategy to create the final protocol).
***Unblur* protocolization and testing**	Implementation of the protocol needed to call and handle the data used/retrieved by *Unblur* (including input/output format conversion among packages and input simplification). The protocol was tested to check if the outputs are properly shown in SCIPION and if they were the desired ones.
***Prime2D* protocolization and testing**	Implementation of the protocol needed to call and handle the data used/retrieved by *Prime2D* (including input/output format conversion among packages and input simplification). The protocol was tested to check if the outputs are properly shown in SCIPION and if they were the desired ones.
***Prime3D* protocolization and testing**	Implementation of the protocol needed to call and handle the data used/retrieved by *ini3D_from_cavgs* (including input/output format conversion among packages and input simplification). The protocol was tested to check if the outputs are properly shown in SCIPION and if they were the desired ones.

Table 2: Workflow followed during the protocolization of SIMPLE

9. PROJECT COST

The costs involving the project are divided in two different sections: the technical equipment (**Table 3**) and the human resources (**Table 4**). The technical equipment. refers to the software and hardware required to develop the project and the human resources includes the salaries of the members involved in the work.

Technical equipment	Cost/Unit (€)	Cost/Month (€)	Months used	Total Cost (€)
Computer	809	20	7	140
MatLab	0	0	1	0
Pycharm	0	0	3	0
Eclipse	0	0	3	0
				140

Table 3: Cost associated to the technical equipment.

Human resources	Hours	Cost/Hour (€)	Total Cost (€)
Student	380	20	7600
Tutor	20	55	1100
			8700

Table 4: Cost associated to the human resources.

The final cost of the project is shown in the following table (**Table 5**):

Concept	Total Cost (€)
Technical equipment	140
Human resources	8700
	8840

Table 5: Total cost of the project.

10. BIBLIOGRAPHY

[1] T. Palucka. “Electron microscopy”. History of recent science & technology. https://authors.library.caltech.edu/5456/1/hrst.mit.edu/hrs/materials/public/ElecMicr.htm (access December 20th, 2018).

[2] J. A. Velazquez-Muriel, M. Valle, A. Santamaría-Pang, I. A. Kakadiaris, and J. M. Carazo, “Flexible Fitting in 3D-EM Guided by the Structural Variability of Protein Superfamilies”, *Structure*, vol. 14, no. 7, pp. 1115-1126, July 2006. [On line]. Available at: https://doi.org/10.1016/j.str.2006.05.013. Access: December 20th, 2018.

[3] “Electron microscope”. Wikipedia. https://en.wikipedia.org/wiki/Electron_microscope. (access December 20th, 2018).

[4] S. J. Ludtke, P. R. Baldwin, and W. Chiu, “EMAN: Semiautomated Software for High-Resolution Single-Particle Reconstructions”, *Journal of Structural Biology*, vol. 128, no. 1, pp. 82-97, December 1999. [On line]. Available at: https://doi.org/10.1006/jsbi.1999.4174. Access: December 20th, 2018.

[5] J. Frank et al., “SPIDER and WEB: Processing and Visualization of Images in 3D Electron Microscopy and Related Fields”, *Journal of Structural Biology*, vol. 116, no. 1, pp. 190-199, January 1996. [On line]. Available at: https://doi.org/10.1006/jsbi.1996.0030. Access: December 20th, 2018.

[6] C. O. S. Sorzano et al., “XMIPP: a new generation of an open-source image processing package for electron microscopy”, *Journal of Structural Biology*, vol. 148, no. 2, pp. 194-202, November 2004. [On line]. Available at: https://doi.org/10.1016/j.jsb.2004.06.006. Access: December 20th, 2018.

[7] J. M. de la Rosa-Trevín et al., “SCIPION: A software framework toward integration, reproducibility and validation in 3D electron microscopy”, *Journal of Structural Biology*, vol. 195, no. 1, pp. 93-99, July 2006. [On line]. Available at: https://doi.org/10.1016/j.jsb.2016.04.010. Access: December 20th, 2018.

[8] D. Elmlund, and H. Elmlund, “SIMPLE: Software for ab initio reconstruction of heterogeneous single-particles”, *Journal of Structural Biology*, vol. 180, no. 3, pp. 420-427, December 2012. [On line]. Available at: https://doi.org/10.1016/j.jsb.2012.07.010. Access: December 20th, 2018.

[9] S. H. Scheres, “RELION: Implementation of a Bayesian approach to cryo-EM structure determination”, *Journal of Structural Biology*, vol 180, no. 3, pp. 519-530, December 2012. [On line]. Available at: https://doi.org/10.1016/j.jsb.2012.09.006. Access: December 20th, 2018.

[10] N. Grigorieff, “FREALIGN: High-resolution refinement of single particle structures”, *Journal of Structural Biology*, vol. 157, no. 1, pp. 117-12, January 2007. [On line]. Available at: https://doi.org/10.1016/j.jsb.2006.05.004. Access: December 20th, 2018.

[11] I. Bahar, T. R. Lezon, A. Bakan, and I. H. Shrivastava, "Normal Mode Analysis of Biomolecular Structures: Functional Mechanisms of Membrane Proteins", *Chemical Reviews*, vol. 110, no. 3, pp. 1463-1497, September 2009. [On line]. Available at: https://dx.doi.org/10.1021%2Fcr900095e. Access: December 20st, 2018.

[12] C. O. S. Sorzano et al., "Survey of the analysis of continuous conformational variability of biological macromolecules by electron microscopy", *Structural Biology Communications*, vol. 75, no. 1, pp. 19-32, January 2019. [On line]. Available at: https://doi.org/10.1107/S2053230X18015108 (access: December 29th, 2018).

[13] J. Bray. "Normal Mode Analysis: Calculation of the Natural Motions of Proteins". Biomedical Computation Review. http://biomedicalcomputationreview.org/content/normal-mode-analysis-calculation-natural-motions-proteins (access: December 21st, 2018).

[14] C. O. S. Sorzano et al., "Local analysis of strains and rotations for macromolecular electron microscopy maps", *Journal of Structural Biology*, vol. 195, no. 1, pp. 123-128, July 2016. [On line]. Available at: https://doi.org/10.1016/j.jsb.2016.04.001. Access: December 21st, 2018.

[15] R. F. Egerton, *Physical Principles of Electron Microscopy*, 1st ed. New York: Springer, 2005. [On line]. Avalailbe at: https://doi.org/10.1007/B136495.

[16] A. D. Rollet, and P. N. Kalu. "Microscopy: Overview of Different Methods". Slide Player. https://slideplayer.com/slide/2320013/ (access: December 21st, 2018).

[17] Healthtard. "Guide to Electron Microscopy". Healthtard. http://www.healthtard.com/guide-to-electron-microscopy/ (access: December 21st, 2018).

[18] M. A. Hill. "Scanning Electron Microscopy". Embriology. https://embryology.med.unsw.edu.au/embryology/index.php/Scanning_Electron_Microscopy (access: December 21st, 2018).

[19] H. Lijima, M. Minoda, T. Tamai, Y. Kondo, and F. Hosokawa, "Development of Phase Contrast Scanning Transmission Electron Microscopy", exposed in 18th International Microscopy Congress, Prague, 7-12 September, 2014. [On line]. Available at: http://www.microscopy.cz/proceedings/all.html.

[20] "Hilbert space". Wikipedia. https://en.wikipedia.org/wiki/Hilbert_space (access: January 21st, 2019).

[21] E. Weisstein. "Hilbert Space". MathWorld. http://mathworld.wolfram.com/HilbertSpace.html (access: January 21st, 2019).

[22] E. Weisstein. "Function Space". MathWorld. http://mathworld.wolfram.com/FunctionSpace.html (access: January 21st, 2019).

[23] "Infinite-dimensional vector function". Wikipedia. https://en.wikipedia.org/wiki/Infinite-dimensional_vector_function (access: January 21st, 2019).

[24] "Spherical Harmonics". Wikipedia. https://en.wikipedia.org/wiki/Spherical_harmonics (access: December 21st, 2018).

[25] "3D rotation group". Wikipedia. https://en.wikipedia.org/wiki/3D_rotation_group (access: December 21st, 2018).

[26] V. Lakshminarayanan, and A. Fleck, "Zernike polynomials: a guide", *Journal of Modern Optics*, vol. 58, no. 7, pp. 545-561, April 2011. [On line]. Available at: https://doi.org/10.1080/09500340.2011.633763. Access: December 21st, 2018.

[27] C. Clemente, L. Pallotta, A. de Maio, J. J Soraghan, and A. Farina, "A novel algorithm for radar classification based on doppler characteristics exploiting orthogonal Pseudo-Zernike polynomials", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 1, pp. 417-430, April 2015. [On line]. Available at: https://doi.org/10.1109/TAES.2014.130762. Access: December 21st, 2018.

[28] "Zernike polynomials". Wikipedia. https://en.wikipedia.org/wiki/Zernike_polynomials (access: December 21st, 2018).

[29] H. Elmlund, D. Elmlund, and S. Bengio, "PRIME: Probabilistic Initial 3D Model Generation for Single-Particle Cryo-Electron Microscopy", *Structure*, vol. 21, no.8, pp. 1299-1306, August 2013. [On line]. Available at: https://doi.org/10.1016/j.str.2013.07.002. Access: December 21st, 2018.

[30] C. F. Reboul, F. Bonnet, D. Elmlund, and H. Elmlund, "A Stochastic Hill Climbing Approach for Simultaneous 2D Alignment and Clustering of Cryogenic Electron Microscopy Images", *Structure*, vol. 24, no. 6, pp. 988-996, June 2016. [On line]. Available at: https://doi.org/10.1016/j.str.2016.04.006. Access: December 21st, 2018.

[31] D. Elmlund. "SIMPLE Documentation". SIMPLE cryoem. https://SIMPLEcryoem.com/index.html (access: December 21st, 2018).

[32] "Table of spherical harmonics". Wikipedia. https://en.wikipedia.org/wiki/Table_of_spherical_harmonics (access: December 21st, 2018).

[33] C. O. S. Sorzano, A. L. Alvarez-Cabrera, and S. J. Jonic, "StructMap: Elastic Distance Analysis of Electron Microscopy Maps for Studying Conformational Changes", *Biophysics Journal,* vol. 110, no. 8, pp. 1753-1765, April 2016. [On line]. Available at: https://doi.org/10.1016/j.bpj.2016.03.019. Access: December 21st, 2018.

[34] Protein Databank in Europe. http://www.ebi.ac.uk/pdbe/node/1 (access: December 21st, 2018).

[35] "SCIPION Documentation". Git Hub. https://github.com/I2PC/SCIPION/wiki (access: December 21st, 2018).

[36] F. J. Sigworth, "Principles of cryo-EM single-particle image processing", *Microscopy*, vol. 65, no. 1, pp. 57-67, February 2016. [On line]. Available at: https://doi.org/10.1093/jmicro/dfv370. Access: December 21st, 2018.

[37] M. Zabrocki. "The Rotational Symmetries of the Cube". Introduction to combinatorics. http://garsia.math.yorku.ca/~zabrocki/math4160w03/cubesyms/ (access: December 21st, 2018).

11. SUPPLEMENTARY MATERIAL (ANNEXES)

11.1. Annex A (MatLab scripts)

MathBasis.m

```
clc, clear
close all;

%Sphere discretization (To plot the Basis)
Res = [100 100]; %Resolution
phi = linspace(0,2*pi,Res(1));
theta = linspace(0,pi,Res(2));
[Theta,Phi] = meshgrid(theta,phi);

%Computation of the Basis
L=2; N=2; M=1;
R = CompRZernike(N,L,Res(1));
Ylm = RealSPH(L,M,Theta,Phi);
Zlnm = R.*Ylm;

%Plot Basis
[Xr,Yr,Zr]=sph2cart(Phi,Theta+pi/2,Zlnm); %Plotting Real Spherical
Harmonics
figure; hold on;
h = surf(Xr,Yr,Zr);
axis equal off; %rot3d;
grid on
```

ComRZernike.m

```
function Vals = CompRZernike(N,L, Res) %Where L is the degree and N
the order

%Discretizacion del disco unitario
r = linspace(0,1,Res);
phi = linspace(0,2*pi,Res);
[r,phi] = meshgrid(r,phi);

%Radial Zernike Polynomials
R = RZernike(N,L);
Vals = R(r);

end
```

RealSPH.m

```
function Ylm = RealSPH(L,M, Theta, Phi) %Where L is the degree and M
the order

%Legendre assciated polynomials
Plm = LPoly(M,L);
Plm = Plm(cos(Theta));
```

```
%Spherical harmonics computation
a1 = (2*L+1)/(4*pi);
a2 = factorial(L-abs(M))/factorial(L+abs(M));
Alm = sqrt(a1*a2); %Normalization Lengendre Polynomials

%Real Spherical Harmonics
if M<0
    Ylm = ((-1)^M)*sqrt(2).*Alm.*Plm.*sin(abs(M).*Phi);
elseif M==0
    Ylm = sqrt(a1).*Plm;
else
    Ylm = ((-1).^M).*sqrt(2).*Alm.*Plm.*cos(M.*Phi);
end

%Representation
Ylm = abs(Ylm);
[Xr,Yr,Zr]=sph2cart(Phi,Theta+pi/2,Ylm); %Plotting Real Spherical
Harmonics
figure; hold on;
h = surf(Xr,Yr,Zr);
axis equal off; %rot3d;
grid on

end
```

RZernike.m

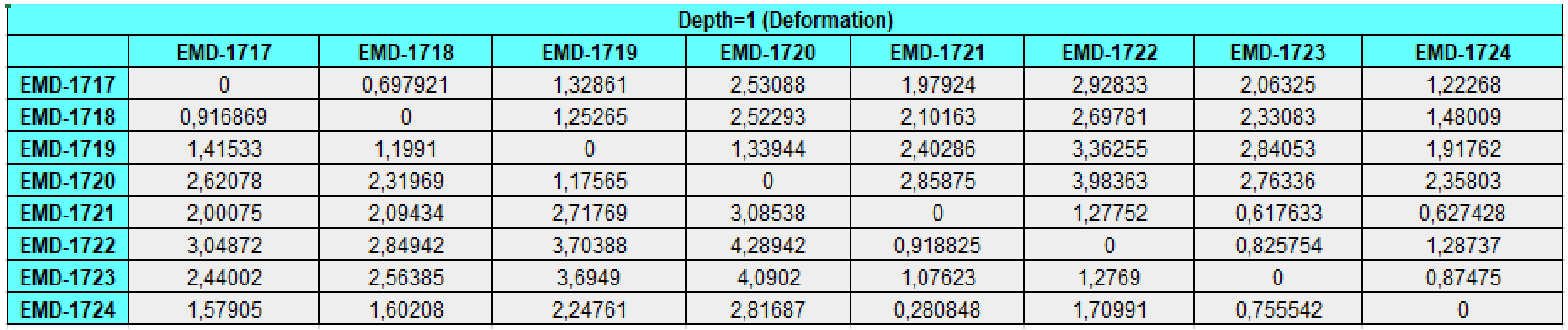
```
function R = RZernike(m,n) %Where n is the degree and m the order

%Computaton of the coefficients of the radial Zernike polynomials

R{1} = zeros(1,m+1); R{1}(m+1) = 1;
R{2} = zeros(1,m+1+2); R{2}(m+1) = -(m+m+2)/2; R{2}(m+1+2) = m+2;

if (ceil(n/2)>2)
    for i = 2:ceil(n/2)
        R{i+1} = -((2*(i-1)+m)*(2*(i-1)+m+1)*(2*(i-1)+m+2))/(2*((i-
1)+1)*((i-1)+m+1)*(2*(i-1)+m))*(-2*[0 0 R{i}])-(2*((i-1)+m)*(i-
1)*(2*(i-1)+m+2))/(2*((i-1)+1)*((i-1)+m+1)*(2*(i-1)+m))*[R{i-1} 0 0 0
0];
        R{i+1}= R{i+1}-((2*(i-1)+m+1)*m^2+(2*(i-1)+m)*(2*(i-
1)+m+1)*(2*(i-1)+m+2))/(2*((i-1)+1)*((i-1)+m+1)*(2*(i-1)+m))*[R{i} 0 0
];
    end
end

if (m==n)
    RC = fliplr(R{1});
else
    RC = fliplr(R{end});
end
R = poly2sym(sym(RC));
R = matlabFunction(R);

end
```

LPoly.m

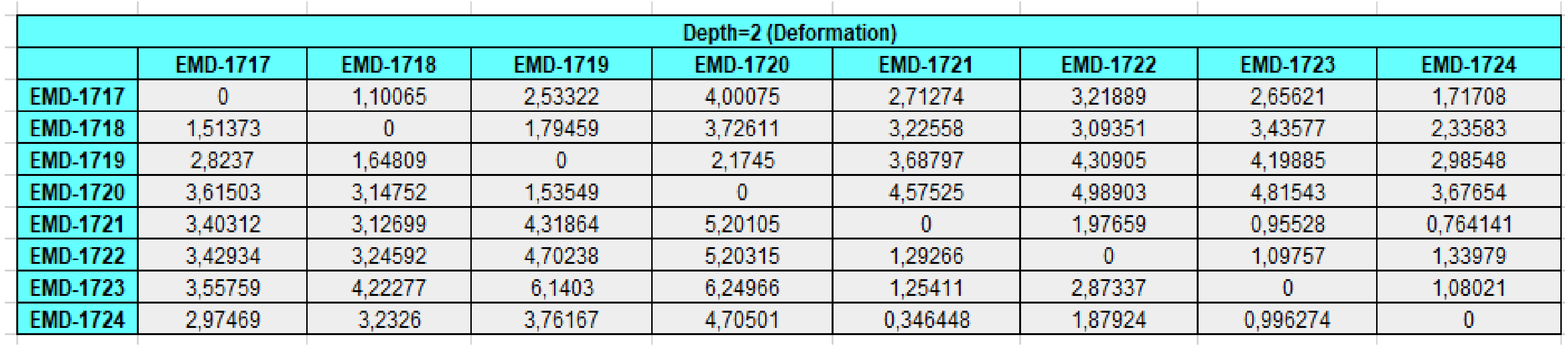
```
function P = LPoly(m,l) %Where l is the degree and m the order

%Coefficients Legendre Polynomials
P{1} = 1;
P{2} = [0 1];
for i = 2:l
        P{i+1} = ((2*i-1)/i)*[0 P{i}]-((i-1)/i)*[P{i-1} 0 0];
end

%Associated Legendre Polynomials coefficients
%Since we have a derivative of order "m", we can compute the new
%coefficiens directly and realocate the values of the vector so the
%position of each coefficient corresponds to its power.
for i = (m+1):(l+1)
    for j = (m+1):length(P{i})
        Plm{i-m}(j-m) = ((-1)^m)*P{i}(j)*factorial(m)*factorial(j-
1)/(factorial(m)*factorial(j-1-m));
    end
end

syms x
PC = fliplr(Plm{end});
P = poly2sym(sym(PC))*(1-x^2)^(m/2);
P = matlabFunction(P);
end
```

11.2. Annex B (Tables)

Deformation of the models

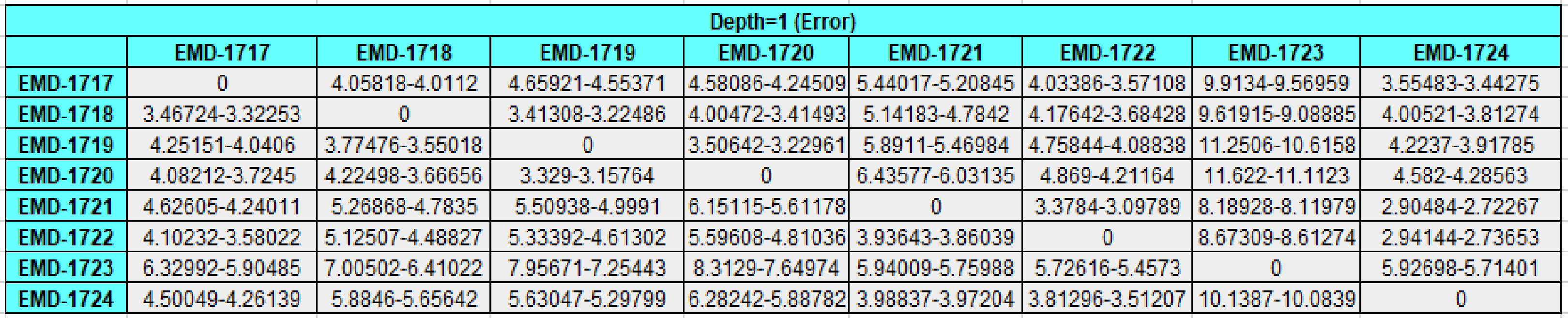


Depth=1 (Deformation)								
	EMD-1717	EMD-1718	EMD-1719	EMD-1720	EMD-1721	EMD-1722	EMD-1723	EMD-1724
EMD-1717	0	0,697921	1,32861	2,53088	1,97924	2,92833	2,06325	1,22268
EMD-1718	0,916869	0	1,25265	2,52293	2,10163	2,69781	2,33083	1,48009
EMD-1719	1,41533	1,1991	0	1,33944	2,40286	3,36255	2,84053	1,91762
EMD-1720	2,62078	2,31969	1,17565	0	2,85875	3,98363	2,76336	2,35803
EMD-1721	2,00075	2,09434	2,71769	3,08538	0	1,27752	0,617633	0,627428
EMD-1722	3,04872	2,84942	3,70388	4,28942	0,918825	0	0,825754	1,28737
EMD-1723	2,44002	2,56385	3,6949	4,0902	1,07623	1,2769	0	0,87475
EMD-1724	1,57905	1,60208	2,24761	2,81687	0,280848	1,70991	0,755542	0

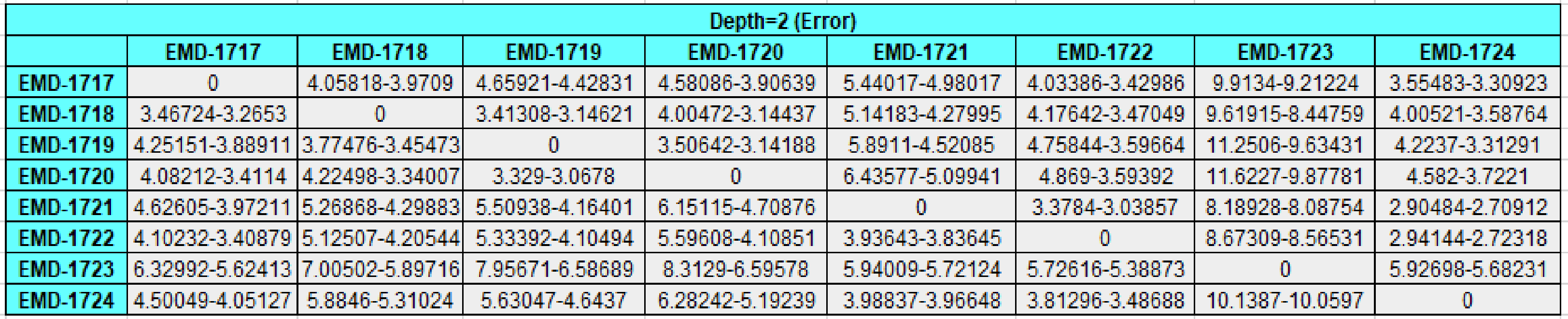


Depth=2 (Deformation)								
	EMD-1717	EMD-1718	EMD-1719	EMD-1720	EMD-1721	EMD-1722	EMD-1723	EMD-1724
EMD-1717	0	1,10065	2,53322	4,00075	2,71274	3,21889	2,65621	1,71708
EMD-1718	1,51373	0	1,79459	3,72611	3,22558	3,09351	3,43577	2,33583
EMD-1719	2,8237	1,64809	0	2,1745	3,68797	4,30905	4,19885	2,98548
EMD-1720	3,61503	3,14752	1,53549	0	4,57525	4,98903	4,81543	3,67654
EMD-1721	3,40312	3,12699	4,31864	5,20105	0	1,97659	0,95528	0,764141
EMD-1722	3,42934	3,24592	4,70238	5,20315	1,29266	0	1,09757	1,33979
EMD-1723	3,55759	4,22277	6,1403	6,24966	1,25411	2,87337	0	1,08021
EMD-1724	2,97469	3,2326	3,76167	4,70501	0,346448	1,87924	0,996274	0

Error before-after deformation of the models



Depth=1 (Error)								
	EMD-1717	EMD-1718	EMD-1719	EMD-1720	EMD-1721	EMD-1722	EMD-1723	EMD-1724
EMD-1717	0	4.05818-4.0112	4.65921-4.55371	4.58086-4.24509	5.44017-5.20845	4.03386-3.57108	9.9134-9.56959	3.55483-3.44275
EMD-1718	3.46724-3.32253	0	3.41308-3.22486	4.00472-3.41493	5.14183-4.7842	4.17642-3.68428	9.61915-9.08885	4.00521-3.81274
EMD-1719	4.25151-4.0406	3.77476-3.55018	0	3.50642-3.22961	5.8911-5.46984	4.75844-4.08838	11.2506-10.6158	4.2237-3.91785
EMD-1720	4.08212-3.7245	4.22498-3.66656	3.329-3.15764	0	6.43577-6.03135	4.869-4.21164	11.622-11.1123	4.582-4.28563
EMD-1721	4.62605-4.24011	5.26868-4.7835	5.50938-4.9991	6.15115-5.61178	0	3.3784-3.09789	8.18928-8.11979	2.90484-2.72267
EMD-1722	4.10232-3.58022	5.12507-4.48827	5.33392-4.61302	5.59608-4.81036	3.93643-3.86039	0	8.67309-8.61274	2.94144-2.73653
EMD-1723	6.32992-5.90485	7.00502-6.41022	7.95671-7.25443	8.3129-7.64974	5.94009-5.75988	5.72616-5.4573	0	5.92698-5.71401
EMD-1724	4.50049-4.26139	5.8846-5.65642	5.63047-5.29799	6.28242-5.88782	3.98837-3.97204	3.81296-3.51207	10.1387-10.0839	0



Depth=2 (Error)								
	EMD-1717	EMD-1718	EMD-1719	EMD-1720	EMD-1721	EMD-1722	EMD-1723	EMD-1724
EMD-1717	0	4.05818-3.9709	4.65921-4.42831	4.58086-3.90639	5.44017-4.98017	4.03386-3.42986	9.9134-9.21224	3.55483-3.30923
EMD-1718	3.46724-3.2653	0	3.41308-3.14621	4.00472-3.14437	5.14183-4.27995	4.17642-3.47049	9.61915-8.44759	4.00521-3.58764
EMD-1719	4.25151-3.88911	3.77476-3.45473	0	3.50642-3.14188	5.8911-4.52085	4.75844-3.59664	11.2506-9.63431	4.2237-3.31291
EMD-1720	4.08212-3.4114	4.22498-3.34007	3.329-3.0678	0	6.43577-5.09941	4.869-3.59392	11.6227-9.87781	4.582-3.7221
EMD-1721	4.62605-3.97211	5.26868-4.29883	5.50938-4.16401	6.15115-4.70876	0	3.3784-3.03857	8.18928-8.08754	2.90484-2.70912
EMD-1722	4.10232-3.40879	5.12507-4.20544	5.33392-4.10494	5.59608-4.10851	3.93643-3.83645	0	8.67309-8.56531	2.94144-2.72318
EMD-1723	6.32992-5.62413	7.00502-5.89716	7.95671-6.58689	8.3129-6.59578	5.94009-5.72124	5.72616-5.38873	0	5.92698-5.68231
EMD-1724	4.50049-4.05127	5.8846-5.31024	5.63047-4.6437	6.28242-5.19239	3.98837-3.96648	3.81296-3.48688	10.1387-10.0597	0

Final distance of the models

Depth=2 (Distance)							
	EMD-1717	EMD-1718	EMD-1719	EMD-1720	EMD-1721	EMD-1722	EMD-1723
EMD-1717	0	0,08728	0,2309	0,67447	0,46	0,604	0,70116
EMD-1718	0,20194	0	0,26687	0,86035	0,86188	0,70593	1,17156
EMD-1719	0,3624	0,32003	0	0,36454	1,37025	1,1618	1,61629
EMD-1720	0,67072	0,88491	0,2612	0	1,33636	1,27508	1,74489
EMD-1721	0,65394	0,96985	1,34537	1,44239	0	0,33983	0,10174
EMD-1722	0,69353	0,91963	1,22898	1,48757	0,09998	0	0,10778
EMD-1723	0,70579	1,10786	1,36982	1,71712	0,21885	0,33743	0