

UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR



**Grado en ingeniería de tecnologías
y servicios de telecomunicación**

TRABAJO FIN DE GRADO

**Métodos computacionales para el análisis de
estructuras macromoleculares**

**Ricardo Serrano Gutiérrez
Tutor: Carlos Oscar S. Sorzano
Ponente: Roberto Marabini Ruiz**

Junio de 2022

UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR



**Grado en ingeniería de tecnologías
y servicios de telecomunicación**

TRABAJO FIN DE GRADO

**Computational methods for the analysis of
macromolecular structures**

**Ricardo Serrano Gutiérrez
Tutor: Carlos Oscar S. Sorzano
Lecturer: Roberto Marabini Ruiz**

June 2022

Computational methods for the analysis of macromolecular structures

Ricardo Serrano Gutiérrez

**TUTOR: Carlos Oscar Sánchez Sorzano
LECTURER: Roberto Marabini Ruiz
ADVISER: James M Krieger**

**Escuela Politécnica Superior
Universidad Autónoma de Madrid
June 2022**

Agradecimientos (Acknowledgments)

Firstly I would like to thank James Krieger for his involvement during the time we have been working on this project. I have learned many things from him, and this project have been finished thanks to his help and guidance.

I would also like to thank my tutors Carlos Oscar Sorzano and Roberto Marabini for giving me the opportunity to do this project and to the people from the Scipion team who helped me in the early stages of the project as well as Mohamad Harastani who develops continuousflex.

Me gustaría agradecer a mis padres, que me han acompañado desde siempre y han confiado en mi para llegar hasta este día. Ellos me han transmitido los valores sobre los que se cimienta este logro y el final de esta bonita etapa. No estaría escribiendo estas líneas si no fuera por ellos.

A mi hermano que aún no es consciente de lo que me ha ayudado a mantener la motivación y la cabeza fría para afrontar cada situación con valentía.

A toda mi familia y en especial a mis abuelas por ser un ejemplo de amor, perseverancia y trabajo duro para mi y a mis abuelos por haberme acompañado y ayudado desde el Cielo a llegar hasta aquí.

A mis compañeros y amigos que me han acompañado en estos cuatro años y que me han demostrado que por caminos pedregosos se camina mejor en grupo.

A toda la gente nueva que he conocido este año estando fuera de España y a Pablo Mora por pelear a mi lado en aquellos meses.

Por último, a mi pueblo, Bercimuel, pero sobre todo a su gente, ellos hacen que el mundo se quede en pausa cuando atravieso las montañas que me separan de sus calles y que olvide (si es que alguna vez existieron) cada uno de mis problemas

Gratias ago Deo.

Resumen (castellano)

En este proyecto se van a diseñar e implementar en el gestor de flujos de Scipion, nuevas funcionalidades para el análisis de la dinámica de ciertos grupos de proteínas mediante la inclusión dentro de la herramienta de Scipion de las posibilidades que ofrece ProDy. Otro software cuyo fin es este tipo de análisis. El lenguaje de programación en el que se realizará este proyecto es Python ya que es el lenguaje en el que se concibieron inicialmente estos dos programas.

En bioinformática existen distintos tipos de modelados y análisis que son muy útiles para conocer el comportamiento de ciertas moléculas presentes en la mayoría de seres vivos. Alguno de estos modelados son el modelado gaussiano de redes y la descomposición del dominio dinámico. El modelado gaussiano nos permite analizar el comportamiento en el espacio de ciertas moléculas y sus interacciones con estímulos físicos. Por su parte la descomposición del dominio dinámico de moléculas nos permite agrupar los resultados obtenidos en el modelado gaussiano para conocer los diferentes comportamientos dinámicos de los diferentes sectores dentro de la propia molécula. Este tipo de análisis se puede realizar a través de las diferentes funciones programadas en ProDy pero, sin embargo, algunas de sus funcionalidades quedan lejos del alcance de aquellos usuarios que no tengan un avanzado nivel en Python. Es por ello que el objetivo principal de este trabajo será el crear un entorno adaptado a cualquier tipo de usuario sin necesidad de que éste tenga que tener conocimientos avanzados de programación.

Con el objetivo de comprobar que los resultados obtenidos a lo largo del proceso de diseño son los correctos, se procederá a recopilar información científica y académica sobre una serie de proteínas (AMPA, NDMAR y ubiquitina) para realizar una simulación de uso de nuestro programa y cotejar los resultados obtenidos con la información científica recopilada. Para poder analizar estas proteínas en concreto, tendremos que acceder a la base de datos PDB por sus siglas en inglés *Protein Data Bank*, con el objetivo de descargar los archivos que modelan las diferentes moléculas a analizar para poder incluirlas dentro de nuestro programa.

Palabras clave

Scipion, gestor de flujos, dinámica, ProDy, Python, software, modelado gaussiano de redes (GNM), descomposición de dominio dinámico, herramientas, AMPAR, NDMAR, ubiquitina, pdb, neurotransmisor, estructura molecular.

Abstract (English)

In this project, new functionalities for the analysis of the dynamics of certain groups of proteins, will be designed and implemented in the Scipion workflow engine by including some of the possibilities that ProDy, another software whose objective is the analysis of the dynamics of proteins, offers. This Project will be developed entirely in Python as it is the language in which these two programs were initially conceived.

In bioinformatics, different types of modelling and analysis exist that are very useful in order to get to know the behaviour of some certain molecules which are present in the majority of the living beings. Some of these models are the Gaussian Network Model and the Dynamic Domain Decomposition

The Gaussian Network Model (GNM) lets us analyse the behaviour in space of some certain molecules and their interactions with physical stimuli. For its part, the dynamical domain decomposition of molecules, lets us group the obtained results in the Gaussian Network Model to get to know the different dynamic behaviours of the different sectors inside the molecule itself. These kinds of analysis can be done thanks to the different functions programmed in ProDy. Nonetheless, some of its functionalities remain far beyond the reach of researchers who do not have an advanced programming level in Python. That is why the main objective of this project is to create a work environment adapted to any kind of user without it being necessary to have advanced knowledge in programming.

In order to make all the checks of the obtained results along the design process, we have collected scientific and academic information about some proteins (AMPA, NMDAR and ubiquitin) so as to make a simulation of a case of use of our program and compare that with the collected scientific information. To be able to analyse these proteins in detail requires access to the Protein Data Bank (PDB) to make downloads of the files that contain experimental models of the different molecules to be analysed so to include them in our program.

Keywords

Scipion, workflow, dynamics, ProDy, Python, software, Gaussian network model (GNM), dynamical domain decomposition, tools, AMPAR, NMDAR, ubiquitin, pdb, neurotransmitter, molecular structure.

Content Index

1.	Introduction	1
1.1	Motivation	1
1.2	Objectives.....	1
1.3	Structure of the report	2
2.	State of the art	3
2.1	Introduction.....	3
2.2	The Scipion environment	3
2.3	ProDy Project	5
2.4	Gaussian Network Modelling (GNM)	6
2.5	Dynamical domain decomposition	9
3.	Design and implementation	11
3.1	Introduction.....	11
3.2	Development Environment.....	12
3.2.1	Windows Subsystem Linux (WSL) and MobaXterm.....	12
3.2.2	Jupyter Notebook and iPython.....	13
3.2.3	Git and GitHub	13
3.3	Main influences of the code developed	14
3.3.1	The ANM analysis	14
3.3.2	The compare protocol and viewer	16
3.4	The implementation of GNM	16
3.5	The implementation of the dynamical domain decomposition.....	24
4.	Tests and results.....	27
4.1	Introduction.....	27
4.2	Creating the Scipion project for the tests.....	27
4.3	Analysis of the results of GNM in a use case	31
4.4	Analysis of the results of the dynamical domain decomposition	35
5.	Conclusions and future work	37
5.1	Conclusions.....	37
5.2	Future work	38
	Bibliography.....	39

FIGURE INDEX

FIGURE 2.1: Scipion Plugin manager	5
FIGURE 2.2: Workflow in ProDy	6
FIGURE 2.3: Schematic representation of nodes in elastic network of GNM [13]	7
FIGURE 2.4: Schematic description of GNM [16]	9
FIGURE 3.1: Hessian Matrix of the potential V	15
FIGURE 3.2: Calculation of the Hessian matrix in ANM (file: protocol_anm.py)	15
FIGURE 3.3: Calculation of the Kirchhoff matrix in GNM (file: protocol_gnm.py)	16
FIGURE 3.4: Graphical interface for the modes.xmd file	20
FIGURE 3.5: GNM viewer interface display in Scipion	24
FIGURE 4.1: Protocol Pwem for the import of the AMPA molecule	28
FIGURE 4.2: Protocol Atom selection for the pwem import of the NMDA molecule	29
FIGURE 4.3: Protocol GNM configuration for the AMPA molecule	30
FIGURE 4.4: Scipion final project for the tests.	31
FIGURE 4.5: Obtained Cross-correlation matrices in the article [28] vs in Scipion	32
FIGURE 4.6: Results obtained from Scipion vs the figure 8B of the article [28].	33
FIGURE 4.7: Desired behaviour to replicate the results of the file [29] programmed on jupyter.	34
FIGURE 4.8: Example of the use of a selection string in Scipion.	34
FIGURE 4.9: Results Scipion vs ProDy analysis of Ubiquitin.	35
FIGURE 4.10: VMD results of the dynamical domains for 5 (left) and 20 modes (right) for NMDA	36
FIGURE 4.11: VMD results of the dynamical domains for 5 (left) and 20 modes (right) for AMPA	36

1. Introduction

1.1 Motivation

Recent times have shown us the importance of having efficient tools so as to be capable of analysing biological, molecular, and atomic structures so as to be able to study how these biological microstructures interact with ourselves or with their own environment. Setting the example of the covid-19 pandemic, thanks to tools like the one developed in this project, humankind was able to prevent new virus strains, and has been able to develop the vaccines against it. In essence, computing and the huge advances we have witnessed in the past few years, are a fantastic way to help in getting a deep knowledge about these molecular structures.

Programming is something I always liked, and this project seemed to be a challenge the first time I heard about it. As is obvious, I had learned nothing about biology or biochemistry during the degree, so getting into a project like this, arises out of my own intention of using the knowledge acquired during the degree of telecommunications engineering in such an important (but at the same time unknown to me) field of biochemistry.

The tools used in this project (Scipion and ProDy) already existed and are commonly used by scientists all over the globe, but I played my part in developing new functionalities into this environment. Because, as I learned in these four years, no matter how good a program or a tool is, you can always add new functionalities to it in order to make it better.

1.2 Objectives

The main objective of this project is to offer the people interested in studying the behaviour of certain molecules an efficient and intuitive tool to do it. Particularly, aimed to implement a functionality using some of the functions that already existed in ProDy (and some new ones that we have included into Scipion) that enable us to make a GNM (Gaussian Network Model) analysis. The work will be developed in the following way:

- Making the code able to make all the calculations needed for the GNM analysis using the functions packed in the ProDy repository.
- Add to the scipion-em-prody repository the necessary changes and the new functions needed to make the most efficient GNM analysis.

- Integrate this ProDy functionality with the Scipion workflow engine and make a friendly interface so anyone can use it, even those users that do not know many things about programming.

Among my personal objectives, the main ones are to understand more deeply the kind of analysis that the scientist does for some specific molecules and learn more about how to integrate and expand code that already exists. In fact, the collaborative coding using GitHub, has been one of the key elements in the developing of this project. Making use of the skills I have obtained in Python during the degree is also an important objective for me.

1.3 Structure of the report

This report will be structured in the following chapters:

- **Chapter 1: Introduction.** Talking about the general concepts. It includes the motivation and the main objectives of the project.
- **Chapter 2: State of the art.** In this chapter the current situation of programs and technologies like those developed will be explained. Also, I will be talking about the main influences of the code developed, and go more deeply into the explanation of the analysis implemented (GNM and Dynamical Domain Decomposition).
- **Chapter 3: Design and implementation.** This chapter's objective is to make an explanation of the way the functionalities have been developed
- **Chapter 4: Tests and Results.** In this chapter we will see an example of use of the tool with the AMPAR (PDB: 3kg2) and the NMDAR (PDB: 4PE5) molecules whose GNMs analysis have been done already, in order to check whether the results obtained are the correct ones.
- **Chapter 5: Conclusions and future work.** I will check whether the objectives described in chapter 1 have been achieved. It will also be discuss which are the perspectives for programs like Scipion and ProDy and some possible future work to be done after this project.

2. State of the art

2.1 Introduction

All the work that has been done through the years on ProDy has allowed researchers to make their work in a more efficient way since all the calculations for complex molecules can now be done by a computer in seconds to minutes. The problem was that, despite the fact that ProDy has some useful tools that can be used almost by any user, for some important functionalities, there was not an intuitive interface for those researchers that were not really into programming, until a few years ago when the main developers started making more friendly interfaces such as the DynOmics webserver [1] and more recently a plugin for Scipion. Hongchun Li and James Krieger were the main players in this process.

The best way to use Scipion is by installing it on a Linux distribution (e.g., Ubuntu). This can be a pain in the neck for users that have not ever used this operating system because it works slightly differently than Windows, which is the most used operating system. Nevertheless, you can install the tool on Windows using a WSL (Windows Subsystem Linux), but this is even more complicated than installing it on a simple Linux distribution currently running on a device. However, the instructions from the documentation on the main website of Scipion are extremely useful and will work even for the less advanced users.

So, in summary, Scipion has many plugins that are integrated in its environment and work in an efficient way. Some of those plugins, for example, can perform sophisticated analyses of images of biological molecules from electron microscopy or make a 3D animation of molecular movements or display some 2D results such as graphs. ProDy, for its part, work also as a Scipion plugin, and can display many results of the dynamic behaviour of some certain molecules from modelling them with elastic networks. The main modelling, I have worked on, is the GNM (Gaussian Network Model) and the dynamical domain decomposition, which is obtained from the GNM results, but I have also inspired myself on the existing code for the ANM (Anisotropic Network Model).

2.2 The Scipion environment

The first thing I had to do before starting this project was installing Scipion (version 3). Despite the fact that installing it on a native Linux operative system was the easiest option, I decided to install it on a Windows Subsystem for Linux (WSL) so it would be less intrusive for the rest of the work that I do on my computer. The process is remarkably similar to the installation described on the Scipion webpage [2]

The only difference is that the interface would not work as it would on a native Linux. We need to install an Xserver such as MobaXterm and after configuring it properly, use its own command window to display the Scipion interface.

One more thing I had to deal with the first time that I tried to make Scipion work, was that I have not got an NVidia card on my pc so, although the install documentation says that I must install CUDA, I just ignored that part as Scipion is not using CUDA in most of the cases.

After installing Scipion and the required plugins, (see below) it is ready to work. . It creates an environment with various Python packages including the three core Scipion ones – scipion-em (pwem), scipion-pyworkflow, and scipion-app – and others for the plugins and their dependencies such as NumPy, SciPy and Biopython, which are also common dependencies of other packages such as ProDy.

For this project, I also needed to download and install ProDy and its plugin. One can do that from the command line or from the Scipion plugin manager. To complete the installation, I made use of the following commands:

```
git clone git@github.com:scipion-em/scipion-em-prody.git
scipion3 installp -p ./scipion-em-prody --devel
```

I installed the developer version of the plugin scipion-em-prody as I was going to make changes to the source code of the plugin. This downloads and installs ProDy itself too, which I also did using the development version as I made changes to it as well. From this procedure, the directory /scipion-em-prody is created and inside it you can find a README where more commands like the one mentioned above are detailed.

In the figure 2.2.1 we can see the display of the Scipion Plugin manager and the necessary plugins installed for this particular project. For example xmipp is necessary not only for ProDy but for Scipion to work. The way of installing these plugins is the same as that described for ProDy except for xmipp whose installation guide is described on the Scipion website.

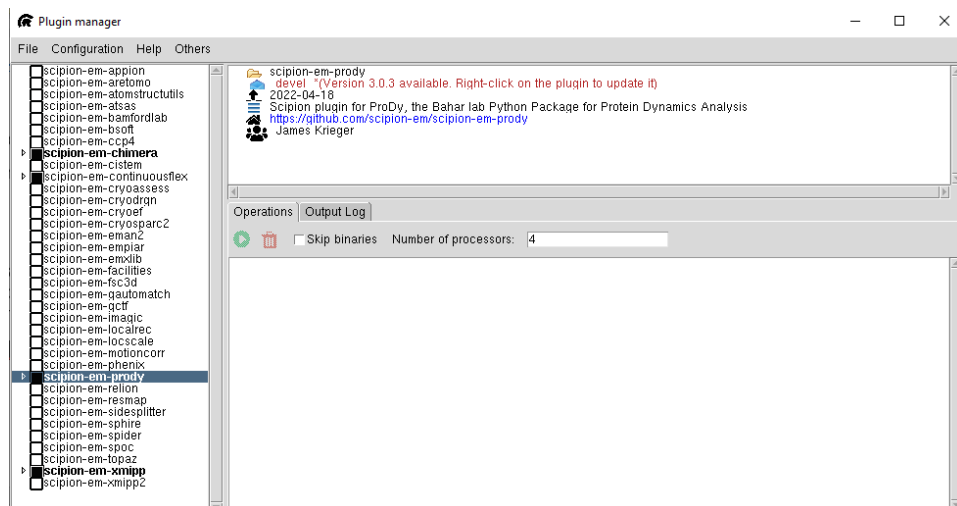


FIGURE 2.1: Scipion Plugin manager

Most of these plugins are for different software developed by different developers that actually do not need Scipion to be executed. But thanks to Scipion we can integrate all this software to work together and use the results of different programs to get more complex results in a more efficient way [3]. Installed Xmipp [4], ChimeraX [5] and ContinuousFlex [6] as well as ProDy [7] of course. I also manually installed VMD [8], which doesn't have a Scipion plugin but is important for analysing ProDy and ContinuousFlex results and can be used by Scipion for that.

2.3 ProDy Project

As explained in the main website of ProDy [7]. It is a free and open-source Python package for protein structural dynamics analysis and some associated programs. The ProDy API (Application Programming Interface) was released in 2011 in order to provide a unified environment for the dynamical analysis of proteins or molecular structures [9]. It was updated in 2014 and a new module was added for sequence evolution analysis (Evol) [10]. The Normal Mode Analysis (NMA) and these tools including the Gaussian Network Model were available already on the first API version. A lot of work has been done for the upgrading of ProDy, resulting in the recent release of version 2.0, and it has seen wide utilization with more than 2 million downloads and 150.000+ unique website visits [11].

ProDy primarily works with atomic structure files, which are usually obtained from the Protein Data Bank (PDB) the files from this data base have been obtained through physical experiments. They can be retrieved using its ID, its sequence or from a local directory. The outputs of the program instances depend on the analysis done, there can be sequences, normal modes, ensembles or plots thanks to the use of Python libraries such as NumPy, SciPy and Matplotlib. For this project it has been used Matplotlib. ProDy is also capable of generating NMD files for the visualization tool NMWiz (Normal Mode Wizard) [7] which is a plugin for VMD [8]. We

can also use TCL scripts that are written on the Scipion side to give VMD instructions without NMWiz as used in the part of the Dynamical domain decomposition to display the results.

The ProDy workflow is illustrated in the following scheme (Figure 2.2):

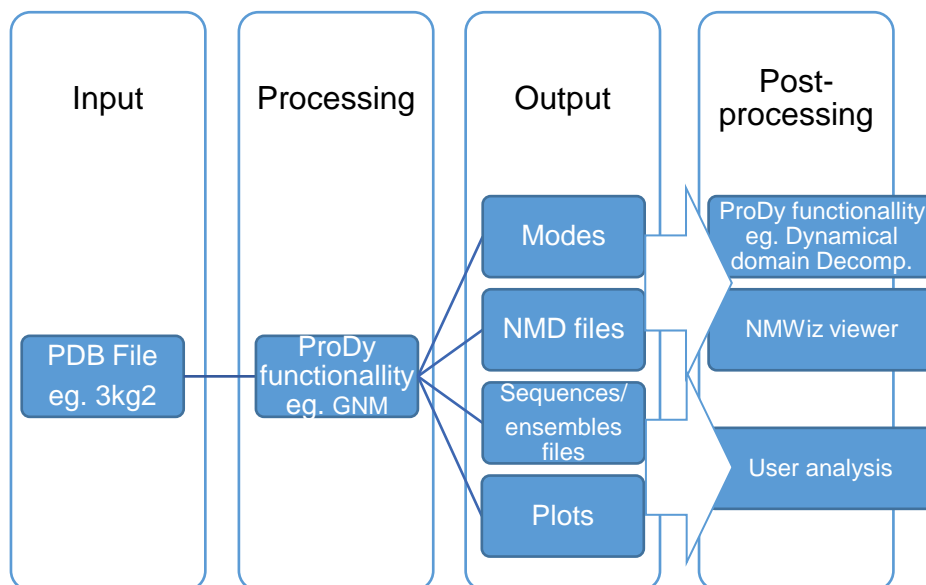


FIGURE 2.2: Workflow in ProDy

2.4 Gaussian Network Modelling (GNM)

At a molecular level, many biological phenomena occur within the scale of nanoseconds to milliseconds. Predicting the movement of a certain molecule could be a hard task, specially if we wanted to make a simulation, because most of this simulation would not be longer than a few microseconds. The elastic network models (in which GNM is included), would provide us a longer time-scale behaviour of the macromolecules.

The Gaussian network model (GNM) is a representation of a biological macromolecule as an elastic network model so it would be easier to understand and study the mechanical aspects of its dynamics. That is, thanks to the GNM analysis we can focus on how the different forces that act over the molecule make it “move” through space [12]. However, this is an incomplete definition of what really happens. It only considers forces within the molecule and but we need to bare in mind that there are essentially random collisions and thermal fluctuations driving the molecule to move at all. Nevertheless, such models have been shown to reproduce experimentally observed dynamic properties, showing that internal contacts are sufficient to predict the motions that happen in response to collisions and thermal energy

The GNM analysis, is rooted in the elastic network (EN) theory, better known as the statistical mechanical theory of elasticity, where other methods such as Anisotropic Network

Model (ANM) are also included. Specifically, in GNM we have the assumption of Gaussian and isotropic fluctuations which means that we will get 1D results.

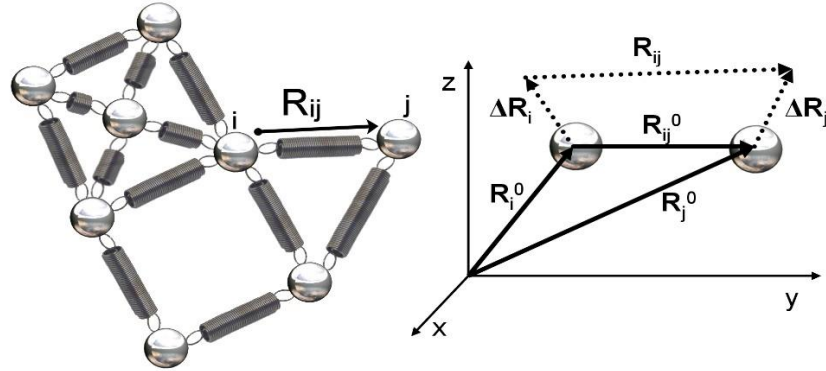


FIGURE 2.3: Schematic representation of nodes in elastic network of GNM [13]

In figure 2.3 we can see a schematic representation of an elastic network studied in GNM. Each node of the net usually represents a residue of the protein/molecule [14]. The different vectors represented on the image are the equilibrium position (R_i^0 and R_j^0) and distance (R_{ij}^0) vectors, and the instantaneous fluctuation (ΔR_i and ΔR_j) and distance (R_{ij}) vectors.

“The Potential Energy of a Gaussian Network Model is defined as the summation of harmonic potentials over all unique (i, j)-pairs and is a function of only the square of inter-residue distance vector, $\Delta R_{ij}=R_i-R_j^0$. It can be given in terms of the Kirchhoff matrix for inter-residue contacts, Γ ” [13] as in the following equation:

$$V^{GNM} = \frac{\gamma}{2} \Delta \mathbf{R}^T (\mathbf{\Gamma} \otimes \mathbf{I}_{3 \times 3}) \Delta \mathbf{R},$$

where γ is a force constant that is usually uniform for all the connections between nodes and Γ is an (N×N) matrix and is defined as:

$$\mathbf{\Gamma}_{ij} = \begin{cases} -1 & \text{if } i \neq j \text{ and } R_{ij} \leq R_{cut}, \\ 0 & \text{if } i \neq j \text{ and } R_{ij} > R_{cut}, \\ -\sum_{j, j \neq i}^N \mathbf{\Gamma}_{ij} & \text{if } i = j. \end{cases}$$

R_{cut} is the cutoff distance, is usually taken as $7.5e-10$ m.

Hence, we can obtain the GNM normal modes from the diagonalization of the Kirchhoff matrix. Their frequency and shape are represented by its eigenvalue and eigenvector, respectively. The different modes represent different parameters of the net, for example the algebraic multiplicity of zero eigenvalues means that everything is connected and moves as a rigid block. Higher eigenvalues are related to more localised motions with lower frequency modes showing domain motions of relevance to biological functions and the highest ones showing high energy movements of individual residues that may be important for stability.

The following equation reflexes the value of Hessian matrix (which will be explained later that is used in ANM (Anisotropic Network Model)). Here we can see the importance of the slow modes as the lowest frequency modes contribute most to the spatial partition function because $\det(H^{-1})$ is the product of the reciprocal nonzero eigenvalues of H.

$$\widetilde{\mathbf{H}}^{-1} = \sum_{k=1}^{3N-6} \frac{\widetilde{\mathbf{u}}_k \widetilde{\mathbf{u}}_k^T}{\omega_k^2} \quad [15]$$

Key mechanical properties of the protein are related to the inverse of the Kirchhoff matrix, which can be approximated by a weighted sum of the contribution of several modes to the dynamics is represented with the following equations that refers to the GNM modes in terms of the K^{th} eigenvalue λ_k and K eigenvectors u_k of Γ

$$\langle \Delta R_i \cdot \Delta R_j \rangle = \sum_K [\Delta R_i \cdot \Delta R_j]_k = \frac{3K_B T}{\gamma} [\Gamma^{-1}]_{ij}$$

$$[\Delta R_i \cdot \Delta R_j]_k = \frac{3K_B T}{\gamma} [\lambda_k^{-1} u_k u_k^T]_{ij}$$

The GNM theory does not provide information of the direction of the fluctuations due to the isotropic nature of the analysis. Hence, the theory come up with N-dimensional predictions (where N is the number of nodes) on the mean-square fluctuation (MSFs) of residues and the covariances and cross correlation between the fluctuations [12] by summing the above dot product contributions from different modes. This means that we can predict the fluctuations of the net by the calculation of the covariance matrix, where the diagonal values are the MSFs. Purely Orientational cross-correlations can be obtained by normalising this matrix between -1 and +1.

Concluding, lets introduce the main application of this network analysis method. GNM is mostly common when analysing proteins. It is remains powerful for predicting the equilibrium dynamics of large proteins and their complexes (existing in the PDB). Thanks to its efficiency and

robustness, it becomes a remarkably interesting resource that can also be applied to analysing chromosomal dynamics, because it can offer a robustly accurate result even when having poor resolution data. This is because it is not necessary to have fully accurate data of the spatial coordinates of the nodes. When using GNM, it is just necessary to get a contact map of a particular system to get its square fluctuations as it is shown in the figure 2.4 [16].

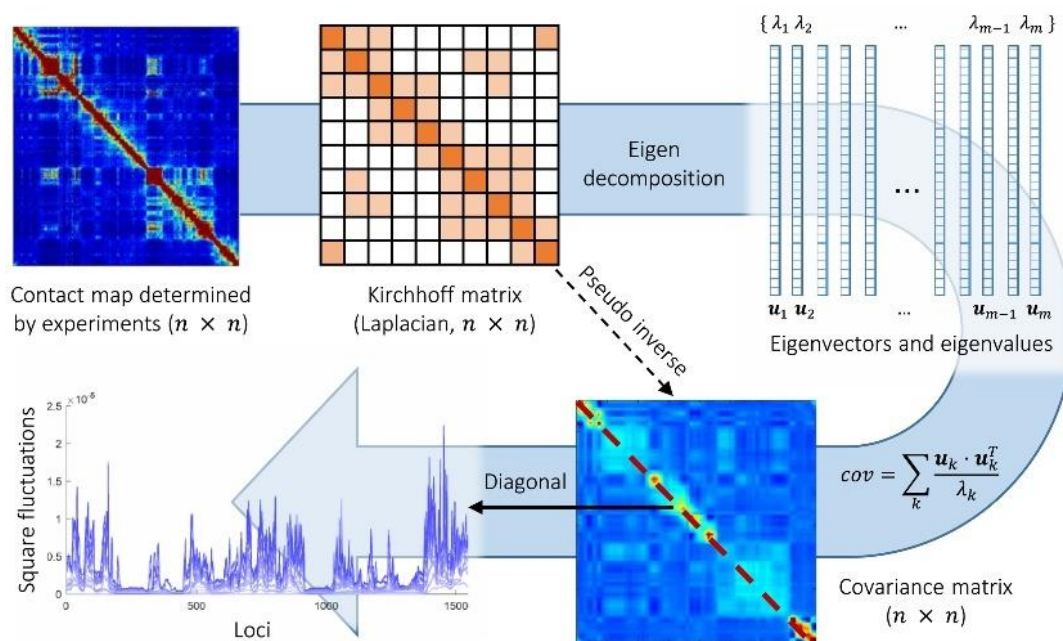


FIGURE 2.4: Schematic description of GNM [16]

The contact map is usually calculated from atomic coordinates in the PDB, but can also come from other data such as Hi-C data for chromosomes.

2.5 Dynamical domain decomposition

The Dynamical domain decomposition consists of grouping regions of similar dynamics obtained from domains of GNM modes. This will result on obtaining at least (or approximately) the same number of dynamical domains as GNM modes.

ProDy has already a method that can calculate these domains by spectral clustering [16] [17]. The way it is done is quite complex to be explained in detail but basically, after calculating the eigenvalues of the GNM modes and (by default) discretizing the calculated matrix of eigenvalues we will be splitting the close values of the matrix into different homogeneous groups [16]. The main objective in this project is to develop a Scipion method capable to do all the necessary steps in order to make the calculations that this ProDy method needs and display the results.

3. Design and implementation

3.1 Introduction

First of all, the entire project will be programmed in python, because is the programming language in which both environments (Scipion and ProDy) where created.

Two new methods have been developed for the Scipion environment GNM and Dynamical Domain Decomposition. Although programs doing the calculations needed for both protocols, already existed in ProDy, it was necessary to have new protocols inside Scipion to get all the outputs, make all the process of calculation and display 3D views easier to take out for an average user.

In order to achieve our goal of making a fully functional program capable of calculating and displaying different results for GNM and for Dynamical Domain decomposition we will need to structure each of the protocols as following:

- i. Define the params required for the functions used.**
- ii. Calculate the modes and write initial files.**
- iii. For the GNM case create the matrices used in the modes calculations.**
- iv. Compute the files and store the modes and calculations.**
- v. Compute some extra calculations (e.g. atoms shift).**
- vi. Create the outputs.**

My main work compromises four files, one for the protocol of GNM, where all the points mentioned above will take place, another one for the visualization of the GNM results (viewer), one more for the computing of the dynamical domain decomposition and another short file for the dynamical domain visualization.

Before starting with explanation of the outcome of these methods, I will first go through the developing and testing environment as well as the main influences and methods in which the development of this new implementation have been based. The ANM (anisotropic network modelling) protocol (already existing in the ProDy plugin, scipion-em-prody) has been used as inspiration for the development of the GNM protocol. For Its part, the GNM viewer is somewhat based in the viewer for comparing normal modes.

For the full development of this project, it was necessary to seek for some certain ProDy functions and understand their behaviour and the way they were defined and programmed. In this section, all the used functionalities will be summed up.

It is also important to mention which files from the directory `scipion-em-prdy` have been modified so as to be able to include the new protocols inside the workflow engine of Scipion and so the user can access to them from Scipion. The main parts are inside the directory **prody2**, which acts as a Python package for the plugin. I modified two `_init_.py` files inside (one in `prody2/protocols` and the other in `prody2/viewers`). In these files we can find the initial definition of all the protocols that are linked to Scipion and all its respective viewers. We have just added two lines to each of them. For the case of the protocols file we have added the following:

```
from .protocol_gnm import ProDyGNM
from .protocol_domdec import ProDyDomainDecomp
```

And for the viewers file we have added these two lines:

```
from .viewer_gnm import ProDyGNMViewer
from .viewer_domdec import ProDyDomainViewer
```

The file `protocols.conf` has also been modified so the user can get the protocol from the ProDy shortcut on the left side panel of the Scipion main screen without needing to list or search through all the protocols from Scipion. It was necessary to include two lines to this protocol (one for GNM and one for the dynamical domains):

```
{"tag": "protocol", "value": "ProDyGNM", "text": "GNM NMA"}
{"tag": "protocol", "value": "ProDyDomainDecomp", "text": "Domain Decomposition"}
```

Continuing with the introduction, I will sum up the different types software packages that exist inside all the code and the names we are giving to them. Scipion and ProDy are the main programs. ProDy is being run by both importing the API library **prody** in lower case and through command-line apps (in our case **prody gnm**). On the Scipion side, it relies heavily on *pwem* (`scipion-em`) and *pyworkflow* (`scipion-pyworkflow`) as will be explained in section 3.4 below. In particular, we use method inside `scipion-em` programmed to link the functionalities from ProDy to Scipion, called protocols which in the Scipion project display may appear as boxes one after another.

3.2 Development Environment

3.2.1 Windows Subsystem Linux (WSL) and MobaXterm

I mentioned before that we were using the WSL of Windows to run a Linux Kernel inside of the native Windows operating system so it would be less intrusive with the rest of things that the user could do in their daily life. This is done thanks to a virtualization of Linux. The way of installing the WSL is fully explained in the Microsoft support [18]. As we said before, the way of installing Scipion is very similar to that described on its website for a native Linux machine. It is

important to make sure of installing everything in a non administrator path. To install everything in the Linux home directory is the recommended option.

The interface of Linux will not work in WSL but its console will do, so if we wanted to display anything on Windows, we would need to use an X server. An X server is a graphic system that provides a limited Graphical User Interface (GUI) of Linux so almost anything that needs the Linux interface can be displayed on any system where the X server is installed. Particularly I have been using MobaXterm, and its installation is quite simple (I just needed to download the .exe and execute it on windows) [19]. Could then use the MobaXterm' s terminal.

A problem that occurred during the configuration of the VMD for the visualizations of molecules in the Linux interface of MobaXterm. As it was working on the WSL, the viewer was not working at all. So it was necessary to download the program for windows and create a symbolic link for the Linux virtual Machine. After that the VMD viewer worked fine.

3.2.2 Jupyter Notebook and iPython

Jupyter Notebook is a coding environment that offers the possibility of debugging code thanks to its principal feature: It is able to execute the code step by step. It also lets the user to save projects as “notebooks” that can be used later or shared for teaching.

iPython is an alternative to Jupyter. It offers some similar features but it is integrated in the IDE terminal and it let us do the main feature of Jupyter Notebook (execute the code step by step), but it will only work on the terminal (MobaXterm or VScode) as it is an interactive Python terminal. Jupyter in contrast can work in a web browser, such as Mozilla Firefox. [20].

Many of the tutorials from ProDy are available as files from Jupyter Notebook. This tool has been used to understand more easily the code behind ProDy functionalities. iPython terminals have also been used as a way to debug the code implemented.

3.2.3 Git and GitHub

Git and GitHub have been main characters in this project. As ProDy and Scipion are public domain software which are being developed constantly by many different people is important to use the forks and branches environment that GitHub offers in other to work as efficiently as possible. This environment consist on a kind of schematical way of uploading the code to the repository where different users may create different branches and then merge the work when everyone relevant has had the chance to check the code and agree. The bugs on each part of the programs keep being solved by the developing community. Thanks to this tool all the developers can keep their software completely updated and the development of the code is fluent. In our

particular case, all the code has been branched with the scipion-em-prody repository [21] and the code in the ProDy repository has been forked to my own repository (user: RicardoSerr) [22] and then branched there.

It is important to make the distinction between Git and GitHub. Git is a program which is installed in the developer computer and that through the command line will be able to upload their code to the GitHub repositories which is the biggest Git based repository host. A repository is a website where there is code stored.

During the development of the work, some changes were needed to be done to some of the ProDy. Some bugs were fixed, and some implementations were included in the ProDy repository.

The RMSF calculation and displaying needed to be fully developed. Nonetheless the implementation was very simple as it would be the same as for the existing calculation of Square Fluctuations and raise its results to 0.5 (taking the square root). With that, we created two new functions in the ProDy repository: *showRMSFlucts* which was added to the *plotting.py* module from ProDy, and *calcRMSFlucts* that was added to the *analysis.py* module in the *prody/dynamics* [23].

Also the *showAtomicLines* needed to be fixed as there were some conflicts when trying to plot the Lines as overlaid chains for multiple atom chains. The way of fixing this was very simple. It was just needed to make the length of the residues of the atoms to be added one after the other [24]

All the changes done in the GitHub repositories of scipion-em/scipion-em-prody are in the appropriate link to GitHub [25] and all the changes done in the prody/ProDy repository are on the corresponding link to GitHub [26]

3.3 Main influences of the code developed

3.3.1 The ANM analysis

The Anisotropic Network Model is used with the normal mode analysis in order to study the dynamics of a specific molecule (e.g. a protein). In a normal mode system, all the oscillating parts of it move sinusoidally with the same frequency and with a related phase (see below). ANM is useful when analysing bio-molecular systems, for example Haemoglobin (*Chunyan 2003*), influenza virus Hemagglutinin A (*Isin 2002*) or the Nicotinic acetylcholine receptor (*Hung 2005 and Taly 2005*)

Comparing ANM and GNM analysis, both are based on an elastic network model. GNM tends to be more accurate when predicting fluctuations than ANM. Nevertheless, GNM is limited to the evaluation of its mean squared displacements and cross correlations between fluctuations. ANM gives us the opportunity to get a 3D description of the 3N-6 internal modes while evaluating internal preferences.

In short, when calculating both methods, the main difference is that with ANM instead of calculating the Kirchhoff matrix as in GNM (described on chapter 2.4) we need the Hessian Matrix (Figure 3.1 of second derivatives of the potential. The Hessian will deploy 3N-6 non zero eigenvectors which means that we will have minimum 6 non zero normal modes. This is due to the properties of the Hessian Matrix, and is translated into having 6 zero modes in ANM. In those modes, the system moves as a connected block. Makes sense to have 6 modes because we are on a 3D representation and we are having rotation and traslation and this modes are complex combinations of this two types of movements.

$$H_{ij} = \begin{bmatrix} \frac{\partial^2 V_{ij}}{\partial x_i \partial x_j} & \frac{\partial^2 V_{ij}}{\partial x_i \partial y_j} & \frac{\partial^2 V_{ij}}{\partial x_i \partial z_j} \\ \frac{\partial^2 V_{ij}}{\partial y_i \partial x_j} & \frac{\partial^2 V_{ij}}{\partial y_i \partial y_j} & \frac{\partial^2 V_{ij}}{\partial y_i \partial z_j} \\ \frac{\partial^2 V_{ij}}{\partial z_i \partial x_j} & \frac{\partial^2 V_{ij}}{\partial z_i \partial y_j} & \frac{\partial^2 V_{ij}}{\partial z_i \partial z_j} \end{bmatrix}$$

FIGURE 3.1: Hessian Matrix of the potential V

We will understand later how the GNM and the ANM protocols work and as they are very similar, we will see now the main differences of our GNM code with the already programmed ANM protocol. The first one is the obvious, GNM is calculating the Kirchhoff matrix from the eigenvalues while ANM is using the eigenvalues to calculate the Hessian matrix, translating this into code we can have a look at figures 3.2 and 3.3. We can notice that the calculations of the eigenvalues are the same for both cases.

```

176 eigvecs = self.anm.getEigvecs()
177 eigvals = self.anm.getEigvals()
178 hessian = prody.parseArray(self._getPath('modes_hessian.txt'))
179
180 self.anm.setHessian(hessian)
181 self.anm.setEigens(eigvecs, eigvals)
182 prody.saveModel(self.anm, self._getPath('modes_anm.npz'), matrices=True)

```

FIGURE 3.2: Calculation of the Hessian matrix in ANM (file: protocol_anm.py)

```

156     eigvecs = self.gnm.getEigvecs()
157     eigvals = self.gnm.getEigvals()
158     kirchhoff = prody.parseArray(self._getPath('modes_kirchhoff.txt'))
159
160     self.gnm.setKirchhoff(kirchhoff)
161     self.gnm.setEigens(eigvecs, eigvals)
162     prody.saveModel(self.gnm, self._getPath('modes.gnm.npz'), matrices=True)

```

FIGURE 3.3: Calculation of the Kirchhoff matrix in GNM (file: protocol_gnm.py)

We have mentioned the second difference a few lines above. While ANM will have 6 zero eigenvalues, GNM would have one. This is because as the Kirchhoff matrix “*is defined as the summation of harmonic potentials over all unique (i, j)-pairs*” [13] we will always have at least one pair of non zero eigenvalues, otherwise it would not make sense to make the calculations for GNM.

It will be important to take into account the number of non zero eigenvalues in order to make the necessary code checks to ensure the accurate and adequate use of this two-network analysis and avoid errors.

Finally, when picking one of the two models it is important to know the kind of prediction that will be done by both. In GNM the distributions of the fluctuations prediction is **Gaussian and isotropic** (there is no variations dependant of the direction measured which means that we will not need to consider this parameter)).

3.3.2 The compare protocol and viewer

Now we will go through the other main influence of the project which is an already existing program inside of the scipion-em-prody (file: viewer_compare.py). This file has been studied mainly to understand the handle of matrix files on scipion. These viewer depends on a protocol inside ProDy (file: protocol_compare.py) that would compare two sets of normal modes with the same number of nodes (Calphas) and degrees of freedom and would calculate different kinds of Overlap or spectral overlap within the two sets of modes. Regarding the outputs, this program will generate matrices files where all the contact points of the modes will be loaded for being displayed.

3.4 The implementation of GNM

Now I can finally explain the way in which the GNM analysis tools have been developed to work inside Scipion and display the convenient results, and also make a brief explanation on how the existing protocol of ANM was working for ProDy. For the development of the file protocol_gnm.py the already existing protocol_anm.py has been used as a base for development as we explained before. The necessity of making the file of viewer_gnm.py comes because ANM

(in fact, every Normal Mode Analysis) is a 3D analysis so it is represented in three dimensions and can be represented straight from the protocol using the ProDy modes viewer, which uses the VMD plugin NMWiz to read the NMD files and make the representation. This mode viewer also work for GNM and is included at the bottom of our display. The Gaussian Network Model results requires a 2D analysis so we should do a different viewer for the representation as it happens with the protocol compare since we want to plot things like the covariance matrix, as well as 1D features such as the mode shape. In this sense, a fast check that need to be done is, whether the analysis done is ANM or GNM. We can do this by checking if the data to be displayed is of a length 3 times the number of atoms. In that case we will be displaying Normal Mode Analysis.

File: protocol_gnm.py

The first thing we need to do for this file is to include all the libraries and functions needed for this program to work. There is two important imports here needed to create the parent class of the following functions defined in the code. These are *pwem* and *pyworkflow*.

pwem is the library from scipion-em from where we will create the main class of this file (the parent class) which is *ProDyGNM(EMProtocol)*. The *EMProtocol* is imported from *pwem.protocols*. We will also be importing some objects from *pwem* (*AtomStruct*, *SetOfNormalModes*, *String* and *EMfile*) which are mainly used in the output creations section of the code and in the params assignment. The *EMlib* from *pwem* (which is actually a link to *Xmipp* when installed as in our case) was also imported and it is used to write metadata in the qualifying of the GNM modes.

Regarding the *pyworkflow* import, this is used to create the special params from the first class of the code (*BooleanParam*, *PointerParam* which is used to make the import (point to the place of the memory) of the input structure, and other params like the int or float param). This import is important so the scipion-em is able to read correctly the params from the GNM protocol. We also have used some utilities such us the *makePath* function which is used to load the calculation of the atom shift.

Some other imports were also needed like *prody* and the *math* library and the *os* library used to create and read from the different paths. It is necessary to mention the meaning of the use of the *** in the imports this is used so we can get rid of the initial indication of the position of a certain function. So for example, if we make the following import:

```
from prody import *
```

We will not need to name ProDy at the beginning of its functions so for example we could just *calcCorrelation()* instead of *prody.calcCorrelation()*. However, this can sometimes cause problems is different libraries have functions or classes with the same or similar names.

To conclude with this section, we should explain the structure that our code in Python has. We have a class, *ProDyGNM* and all the functions inside it will be able to access any of the properties of the *ProDyGNM* which at the same times is inheriting from the *EMProtocol*. In order to refer to this common namespace it is uses the *self* variable, which is one of the common conventions for this.

After assigning all the params and variables for the whole program, we will need also to define the functions (steps) that our code will be following in order to make the Scipion workflow to run the program. For the case of GNM we will have the following functions which will be executed in the following order:

i. (line 59) `def _defineParams(self, form):`

In this part of the code we will just define all the variables that may appear on the displayed form of the GNM protocol and that will be passed to the rest of the functions. We are using methods from the parent class *EMProtocol* in order to build the initial form and these also determine when and how this function gets executed. This function also includes the 'help' messages for that display.

ii. (line 114) `def _insertAllSteps(self):`

In this function, all the steps of the code are assigned an order to run so that protocol knows what to do. We will also obtain some inputs values such as the number of modes or files like the atomic structure from the previous step that are passed into to these other steps..

iii. (line 131) `def computeModesStep(self, inputFn, n):`

In this function, we will first get all the files needed and then after calculating the modes file (*modes.gnm.npz*) using an app from *ProDy*, we will then load the results by using the *ProDy* function *loadModel()* storing this information in a class attribute (*self*), *self.gnm*. As we mentioned above, a *self* variable is used to refer to the main class function in python to which all the methods and objects of the code belong.

This attribute contains its own attributes including he eigenvalues and the eigenvectors. That were calculated from different apps of *ProDy* and then use the functions, *getEigvals()* and *getEigvecs()* to obtain those results. Then, we can set the Kirchhoff matrix (which was saved in a separate file) as an attribute too and restore the eigenvalues and eigenvectors to it using the functions *setKirchhoff()* and *setEigens()*. In this step, we will also compute the covariances and the cross-correlation matrices (which is not done in the ANM protocol).


```
(line 164) covariances = prody.calcCrossCorr(self.gnm[1:],norm=False)
(line 167) crossCorr = prody.calcCrossCorr(self.gnm[1:])
```

It is important to mention that the cross correlation is equals to the covariance matrix but normalized and that's why we are using the same function for both calculation. The reason to discard the zero eigenvalues in this method is because the contribution of each mode to the covariance is related to 1/eigenvalue and we can't divide by zero.

```
iv. (line 170) def qualifyModesStep(self, numberOfModes,
collectivity-Threshold,
structureEM, suffix='')
```

In this step, we will calculate the collectivity and the related score so to complete the modes.xml file which is necessary for one of the displays, the metadata viewer mode (figure 3.4). The implementation of this is not so interesting (it does not change a lot from one protocol to other), although is very common in scipion-em-prody and scipion-em-continuousflex..

```
v. (line 231) def computeAtomShiftsStep(self, numberOfModes):
```

This function will calculate the atoms shift profile which is the variation of position of individual atoms. The operations done in this function are basically a sort of loops to calculate the maximum value of a place in the array of computed modes. Once it is calculated you will have to deal with the same difficulty as when opening the necessary files in order to write the results. We will be writing that results in the maxAtomsShift.xml The main difference between ANM and GNM protocols here is to handle the presence or absence of 3D information..

```
vi. (line 265) def createOutputStep(self):
```

Finally the last step would be to create the outputs for Scipion, we will create two objects around the matrix files, which will be handled by the viewer GNM, and of course the SetofNormalModes which are needed by Scipion in other to handle the results from ProDy. These are the output modes.

	enabled	nmaCollectivity	nmaModefile	nmaScore	nmaEigenval	order
1	<input type="checkbox"/>	1.0000	Runs/000436_ProDyGNM/modes/vec.1	0.0500	0.0000	1
2	<input type="checkbox"/>	0.5019	Runs/000436_ProDyGNM/modes/vec.2	0.1250	1.8774	2
3	<input type="checkbox"/>	0.5186	Runs/000436_ProDyGNM/modes/vec.3	0.1250	5.9400	3
4	<input type="checkbox"/>	0.4838	Runs/000436_ProDyGNM/modes/vec.4	0.2000	11.1757	4
5	<input type="checkbox"/>	0.3270	Runs/000436_ProDyGNM/modes/vec.5	0.2500	13.3649	5
6	<input type="checkbox"/>	0.1531	Runs/000436_ProDyGNM/modes/vec.6	0.3250	15.3569	6
7	<input type="checkbox"/>	0.0337	Runs/000436_ProDyGNM/modes/vec.7	0.6000	17.9435	7
8	<input type="checkbox"/>	0.0621	Runs/000436_ProDyGNM/modes/vec.8	0.5250	17.9945	8
9	<input type="checkbox"/>	0.1318	Runs/000436_ProDyGNM/modes/vec.9	0.4250	19.4044	9
10	<input type="checkbox"/>	0.0988	Runs/000436_ProDyGNM/modes/vec.10	0.4750	20.5061	10
11	<input type="checkbox"/>	0.0188	Runs/000436_ProDyGNM/modes/vec.11	0.7250	21.9565	11
12	<input type="checkbox"/>	0.0402	Runs/000436_ProDyGNM/modes/vec.12	0.7000	22.2703	12
13	<input checked="" type="checkbox"/>	0.2304	Runs/000436_ProDyGNM/modes/vec.13	0.4750	22.8269	13
14	<input type="checkbox"/>	0.0168	Runs/000436_ProDyGNM/modes/vec.14	0.8250	23.3918	14
15	<input type="checkbox"/>	0.0109	Runs/000436_ProDyGNM/modes/vec.15	0.8750	23.9350	15
16	<input type="checkbox"/>	0.0684	Runs/000436_ProDyGNM/modes/vec.16	0.7000	24.1589	16
17	<input type="checkbox"/>	0.0583	Runs/000436_ProDyGNM/modes/vec.17	0.7750	24.4996	17
18	<input type="checkbox"/>	0.0784	Runs/000436_ProDyGNM/modes/vec.18	0.7250	24.7323	18
19	<input type="checkbox"/>	0.0433	Runs/000436_ProDyGNM/modes/vec.19	0.8500	25.3799	19
20	<input type="checkbox"/>	0.0824	Runs/000436_ProDyGNM/modes/vec.20	0.7500	25.9953	20

FIGURE 3.4: Graphical interface for the modes.xmd file

File: viewer_gnm.py

Having explained the process of making the GNM calculations, this section focuses on the way of showing all the data so it is as useful as possible for the users as well as be done in a friendly interface. As we did before, we will explain each of the functions of the code in a sorted way.

First, we will import all the necessary libraries from scipion-em and from ProDy as we did in the previous described file. This time, *pyworkflow* and *pwem* will be also imported. Their libraries focused on the visualization are the majority of the imports. One of the most important imports being done here is the ProDyGNM class, which will be used as the main target of the program so that we get an appropriate viewer.

We will also be creating the main Class (child) that will inherit from the ProtocolViewer from Scipion (parent) and will include some of its properties as we explained above:

ProDyGNMViewer (ProtocolViewer)

We will also define the variables and the environments of the program. The principal class (self refers to) ProDyGNMViewer and the embedded methods to this class are:

i. (Line 63) `def _defineParams(self, form):`

As in the previous code, here is where all the params shown in the GUI display are defined. The 'help' messages are also be defined here as before (figure 3.5). In this method, a lot of the params are actually for telling the programs which button was clicked rather than actually having a value that is entered. In this part we will group some parameters so the display is more intuitive for the user using methods from ProtocolViewer.

ii. (Line 126) `def _getVisualizeDict(self):`

This function, creates a dictionary so the reading of the parameters will be indexed and each function can know which kind of param is being used as an argument. Here it will also define the output modes and get the atoms from the GNM protocol to handle them in the rest of the code.

iii. (Line 143) `def _viewAllModes(self, paramName):`

This method, is for plotting the covariance and cross-correlations matrices for all the computed modes of GNM with its own title each. We will also set up the VMD view (calling `createVmdNmwizView`) assigning the task to do when the parameter name is 'displayVmd2'.

iv. (line 161) `def _viewParam(self, paramName):`

This method will plot the maximum distance profile by using the calculated values in the file `maxAtomShifts.xml` from the previous code or will display the modes metadata viewer from `prody` that we showed in the figure 3.4.

v. (Line 170) `def _viewSQF(self, paramName):`

This function calculates and plots the mean square fluctuation (MSF) and its square root, the root mean square fluctuation (RMSF). The MSF represents the average displacement that a particle have with respect to its reference position.

Regarding the coding of the function, we will be displaying these two types of square fluctuations from all the computed modes or for a range of them. It will also be displayed an

RMSF or MSF for a single mode if in the modes range section both numbers typed are the same (that mode number will be plotted in such case). We will need to make some checks in order to see which case are we plotting and also to see whether there are enough computed modes to plot. It is also necessary to check that the modes are not lower than 2 (to account for having one zero mode) and that the second mode is bigger than the first one (correct range). This is done between the lines 170 and 203. The range checks will be done with *if* instructions and the check of the existence of enough modes to compute will be done with *try/except* instructions. In both cases an error message will be displayed in case the user would make any of these mistakes. These looks as follows:

```

try:
    mode1 = self.modes[modeNumber1]
except IndexError:
    return [self.errorMessage("Invalid initial mode number *%d*\n"
                              "Display the output Normal Modes to see "
                              "the available ones." % modeNumber1+1,
                              title="Invalid input")]
if modeNumber1+1 > modeNumber2:
    return [self.errorMessage("Invalid mode range\n"
                              "Initial mode number can not be "
                              "bigger than the final one.", title="Invalid
                              input")]

```

The second part of the method (from line 204 to 245) will make the plotting work. It will just check in which scenario the user is (whether they want to plot single or multiple modes) and will do it by using the following lines:

- (line 207) `plot = prody.showSqFlucts(self.modes[1:], atoms=self.atoms)`

Here we will be plotting all the modes MSF:

- (line 215) `plot = prody.showRMSFlucts(self.modes[modeNumber1], atoms=self.atoms)`

Here we will be plotting the RMSF of a single mode (both modes have the same number):

- (line 218) `plot = prody.showSqFlucts(self.modes[modeNumber1:modeNumber2], atoms=self.atoms)`

Here we will be plotting the MSF of a range of modes

vi. (Line 224) `def _viewSingleMode(self, paramName):`

On a first instance, in this method we were going to be displaying just the shape of a single mode and we have the option to show it as overlaid curves of different colours for different chemical chains forming the subunits of the protein or one after the other with bars underneath to indicate them. For doing this it was necessary to modify some files from ProDy in order to fix the bug that would not let me plot multiple chains as it was explained in section 3.2.3.

After doing some tests we found also necessary to make this function able to plot the covariance and the cross correlation matrices for a single mode. This was done inside the protocol viewerGNM itself just by using the output modes of the main GNM protocol and the ProDy functions `showCovarianceMatrix()` and `showCrossCorrelation()` but not the generated matrix files.

vii. (line 247) `def createShiftPlot(mdFn, title, ylabel):`

This basic function will just create the plot where the atoms shift calculations will be plotted.

viii. (Line 263) `def createDistanceProfilePlot(protocol, modeNumber):`

This will display the maximum distance profile of all the computed modes. And we will need to use the last function to be able to display it:

```
(line 282) def showDistanceProfilePlot(protocol, modeNumber):
```

ix. (line 293) `def createVmdNmwizView(protocol):`

Finally we will seek for the vmd file so the program is able to plot the modes. Notice that this method and the previous 2 (`createDistanceProfilePlot()` and `createShiftPlot()`) are not inside the main class. NMWiz (Normal Mode Wizard) is another repository from the ProDy universe capable of plotting the different modes of an analysis and assigning them colours. It can be downloaded in development mode too, but has been a core plugin shipped with VMD since 2014.

The final result shown below in figure 3.5 is what the user would see, and may use in order to make all the necessary calculations and get as much important information as possible from the GNM analysis.

It is structured in three groups of plotting all modes, single mode and range of modes. As I mentioned above, this is done thanks to the grouping of parameters in the first step of the viewer program and the protocol uses files and packages that have already been built in Scipion.

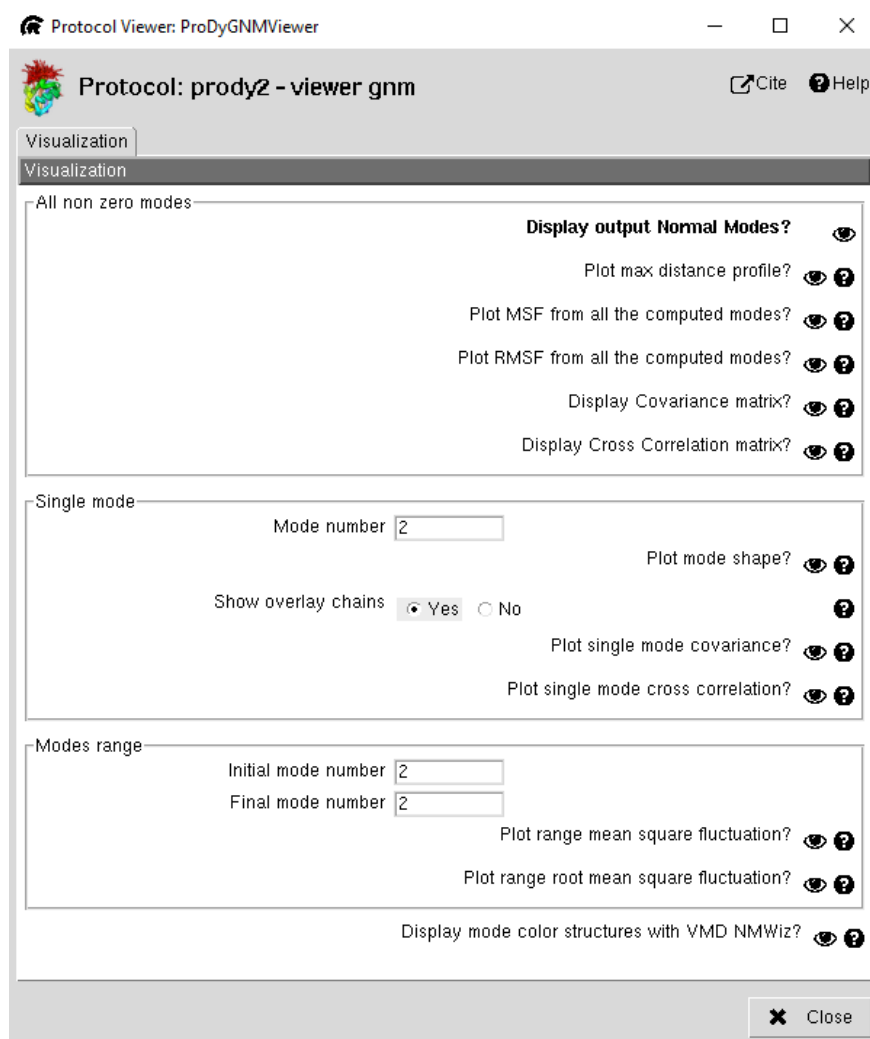


FIGURE 3.5: GNM viewer interface display in Scipion

3.5 The implementation of the dynamical domain decomposition

As explained above, domain decomposition is used in bioinformatics to group the parts of a certain molecule in order to predict its behaviour and dynamical domain decomposition does this by using the lower energy GNM modes. In this section of the project we will make all the necessary code in two files: *protocol_domdec.py* and *viewer_domdec.py*

This can be calculated in ProDy by its function `calcGNMDomains()`. The way my codes have been planned is by getting the GNM modes calculated in the previous protocol and use them as inputs to this new protocol. After assigning all the variables, and getting the input from the user of the number of modes to be used, using the mentioned function the dynamical domains modes will be calculated and loaded in the file `atoms.pdb`. After this, the analysis will be plotted out.

The outputs on Scipion consist of two files, The following code lines are those used to generate the outputs of the protocol `ProDyDomainDecomp ()`:

(Line 94)

```
def createOutputStep(self):
    fhCmd=open(self._getPath("domains.vmd"),'w')
    fhCmd.write("mol new %s\n" % self._getPath("atoms.pdb"))
    fhCmd.write("mol modcolor 0 0 Beta\n")
    fhCmd.write("mol modstyle 0 0 Beads\n")
    fhCmd.close()

    outputPdb = AtomStruct()
    outputPdb.setFileName(self._getPath("atoms.pdb"))

    outputvmd = EMFile()
    outputvmd.setFileName(self._getPath("domains.vmd"))

    self._defineOutputs(outputStructure=outputPdb, outputvmd=outputvmd)
```

In the first 5 lines it its created the vmd file in which we will include the color mode and the style of the molecule drawing that in this case consists of beads (a sort of dots). Then we will create the PDB atom struct and the EMfile so we can display the VMDview in the `viewer_domdec.py`. Which after targeting the main protocol `ProDyDomainDecomp` and the pertinent environment, will just make the return of the vmd view as follows:

```
return [VmdView('-e "%s"' % self.protocol._getPath("domains.vmd"))]
```


4. Tests and results

4.1 Introduction

In this chapter there are some tests of the implemented code. In particular I have made the GNM analysis of the AMPA and NMDA glutamate receptors as in the published article: *Cooperative Dynamics of Intact AMPA and NMDA Glutamate Receptors: Similarities and Subfamily-Specific Differences* [27]. We will try to replicate that study and see if the obtained results are congruent and correct. As this study will not be enough to test all the implementations, it will be necessary to make some other analysis that will be detailed later.

This part will be explained in the form of a tutorial, allowing readers to reproduce the steps and use my code more easily.

First, it will be necessary to create a project inside Scipion to make a real case of work. In the following points, it will be explained how to create the complete project and the protocols from the ProDy plugin needed to complete the full analysis. Some of the errors in the program (not only in our own code but in the whole ProDy project) will also be treated.

The results will be explained in order to make them easier to understand despite being detailed in the article [27]. Such comparisons were performed in the paper by selecting corresponding atoms between the two structures but this is beyond the scope of this project.

To conclude with this point, I will introduce the molecules to be analysed. The glutamate receptors molecules, are a type of protein that recognise the excitatory neurotransmitter glutamate. This kind of receptors are classified into various types. In particular, the AMPAR (PDB structure: 3kg2) and the NMDAR (PDB structure: 4PE5) are classified inside of the ionotropic group which are usually cations channels. creating electrical signals relevant for information processing, learning and memory. Their inner chemistry is too complex to be explained in this report.

4.2 Creating the Scipion project for the tests

As we explained in the previous point, we will need to create a whole Scipion project to make the test. We will need to use some protocols from the ProDy plugin (and one from pwem) in order to make the full analysis of the two molecules. The Protocols used for these tests will be:

- pwem – import atomic structure
- ProDy atom selection

- Protocol GNM
- Protocol Dynamic Domain Decomposition
- Protocol compare.

After creating a new Scipion project by opening Scipion and clicking new project, we will need to import some new protocols to it as listed above. . These are all listed under the ProDy protocols option in the left side panel.

First is the import atom structure protocol from Scipion itself (provided by the pwem), this is needed to, as its own name says, make the import of the pdb file to be read for the protocols that follow this one.

We can see the example for the AMPAR molecule in the figure 4.1. For the NMDAR it would be similar we would just need to substitute the atomic structure ID and put the NMDAR pdb: 4PE5.

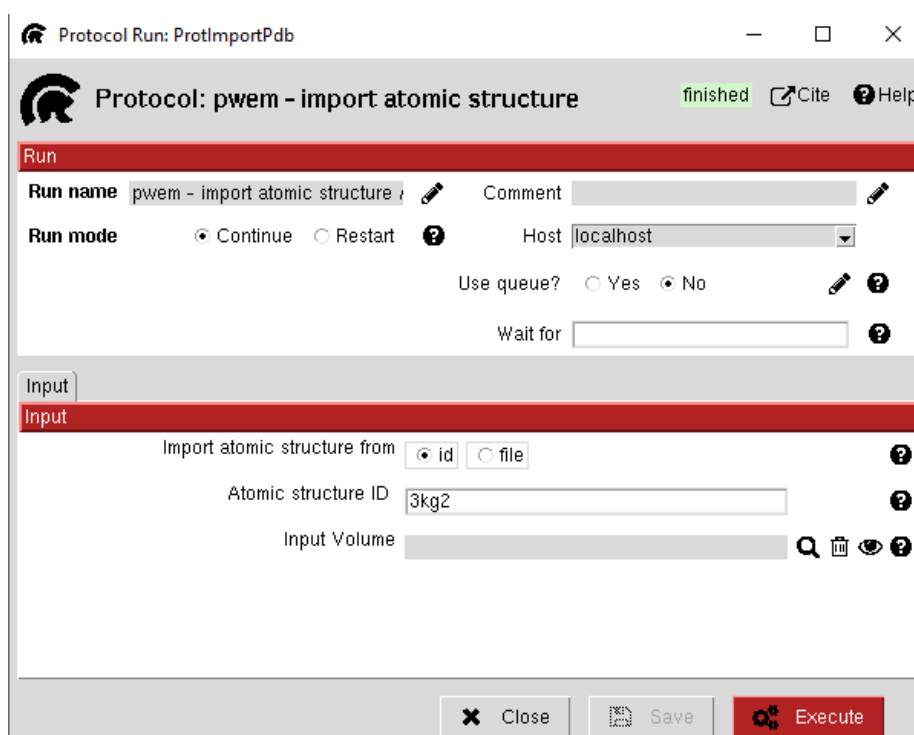



FIGURE 4.1: Protocol Pwem for the import of the AMPA molecule

For these two cases the seek of the atomic structure done by pwem would be successful, but in some cases we would need to go to the pdb data base through the internet (<https://www.rcsb.org>) and download the complete pdb file. Some other times it may be needed to

use our own file, those cases are when the file has not been added to the data base yet or when we have modified the file for example in a selection or alignment. If this were this the case, we would just need to change the *import atomic structure from* option and select file. After that, we can select the path of the downloaded pdb file and the protocol should do the rest of the work.

The next step is to use the atom selection protocol, this is used to convert the pdb file into an atomic file able to be processed by the rest of protocols in ProDy where only the Calphas are selected. The display of this protocol is showed in figure 4.2 and after clicking in the  symbol from the dialog *input structure* we would just need to select the pwem import structure of the desired molecule. After that just click *Execute* as the default selection string is fine.

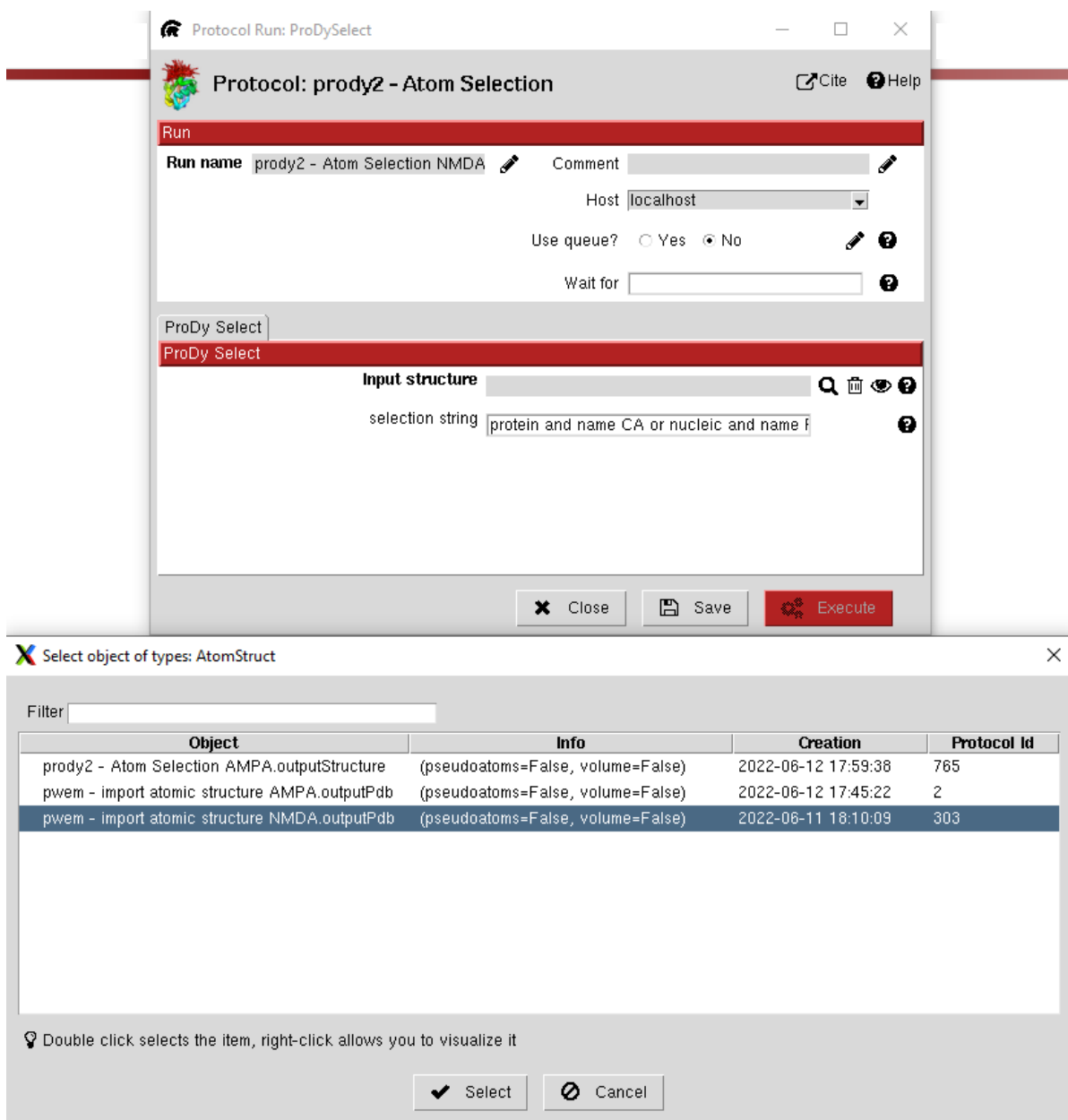


FIGURE 4.2: Protocol Atom selection for the pwem import of the NMDA molecule.

The next step is to use the protocol GNM for both of the ProDy atom selection boxes. We will compute 40 modes as that's the number of modes used in the article [27]. By following the steps from the previous protocol, we will get a similar dialog and we will just need to pick the ProDy atom selection for the particular molecule to be computed. We can select the advanced option and it would be great to get rid of the zero if we wanted to, but we need them for dynamical domain decomposition. So in this sense every dialog box from the advance display can remain as default. (figure 4.3).

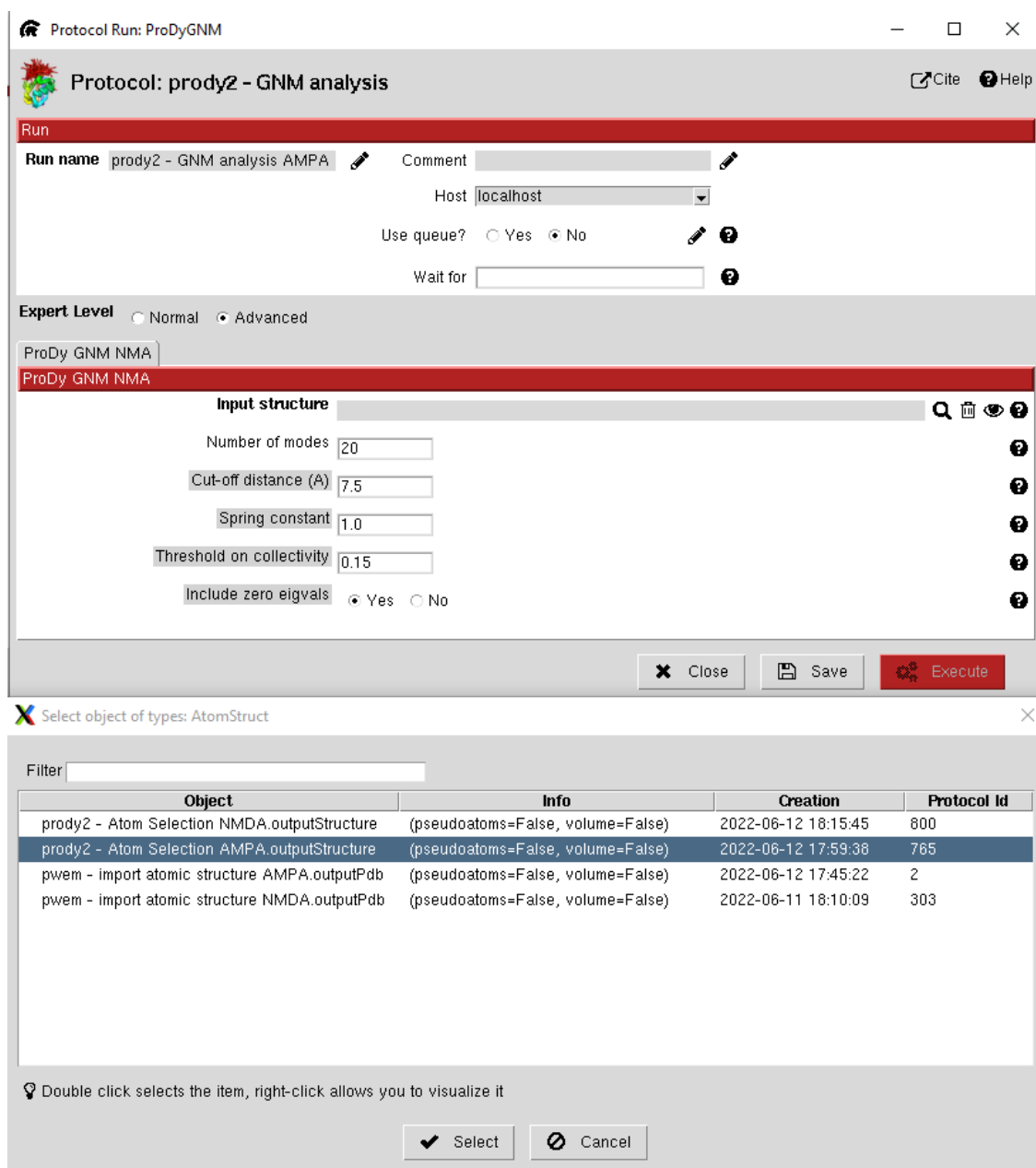


FIGURE 4.3: Protocol GNM configuration for the AMPA molecule.

Now, we will attach the protocol dynamical domain decomposition to this output of the GNM. It would also be interesting to attach the protocol compare to this output. The way of including this protocols is exactly the same as in the previous ones. For the dynamical domain decomposition we will do two versions of analysis, one for 5 and another one for 20 modes (less than in the GNM analysis). If we try to execute a protocol compare attached to this GNM outputs, it will not work. The reason for this to happen is because this protocol is not designed to work with different number of atoms. It is more oriented to the comparison of the same molecule when doing different configurations of analysis (e.g. Different cut-off number or different structural state).

Additionally we will need to include another branch to the project tree for some of the analysis/tests of the GNM. As the studies won't cover all the implementations we have done, we will need to include the ubiquitin molecule whose analysis was done in the ProDy tutorial files. With all that, the final aspect that our Scipion workflow should have is the one shown in the figure 4.4:

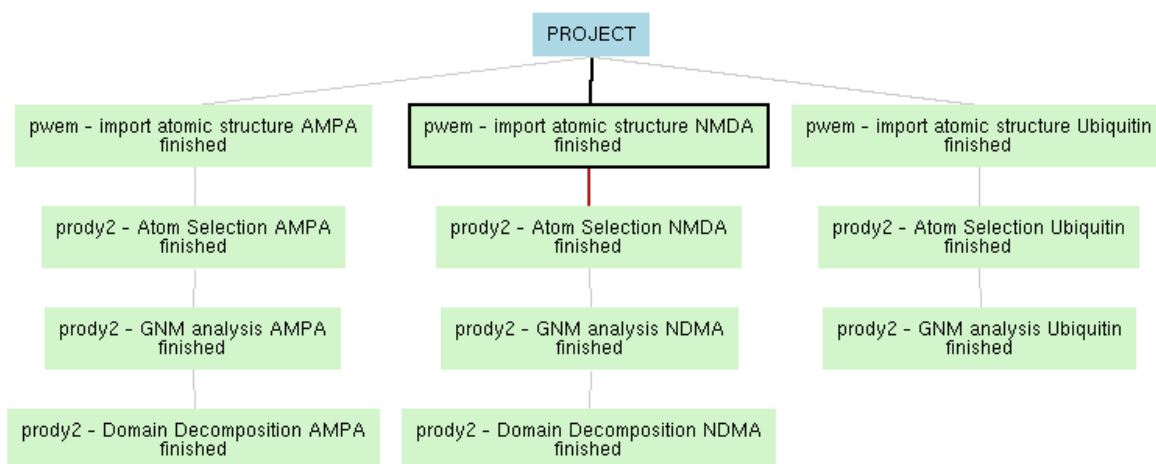


FIGURE 4.4: Scipion final project for the tests.

4.3 Analysis of the results of GNM in a use case

In this section I will be analysing the results obtained in the gnm stage of the workflow and comparing them to the existing results studied in the article [27] and also in the prody tutorial [28]. This way we will be simulating a use case.

Reading the article I found that most of the analyses are done for the non-zero mode 2. We could begin plotting the cross correlation matrix of both molecules for that specific mode, but

initially we had configured the code so it would only display the cross correlations of all the computed modes. So, after the mentioned changes over the viewer protocol we manage to plot a single mode covariance and cross correlation. We can notice that the plots given by our program and the ones observed in the article [27] are exactly the same (some little colour variances are due to the quality of the images) (figure 4.5).

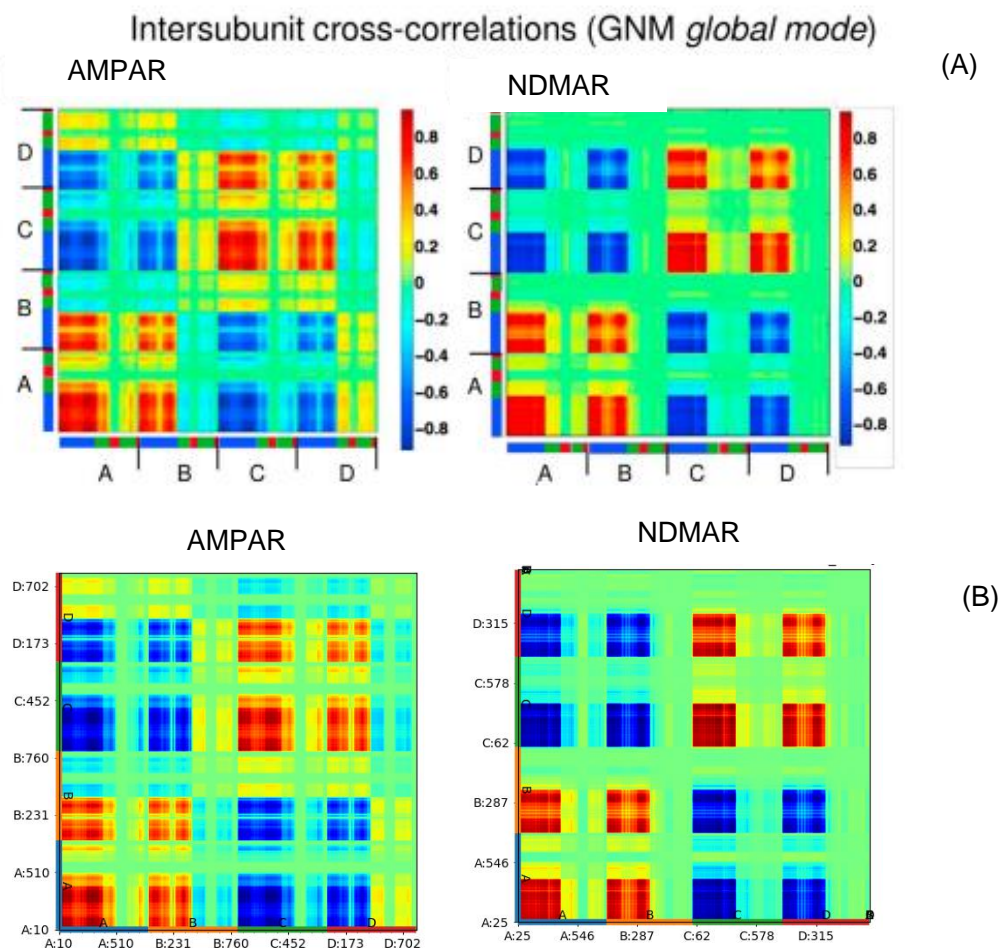


FIGURE 4.5: Obtained Cross-correlation matrices in the article [27] vs in Scipion .

Figure 2 from article [27] representing the cross-correlations for the GNM mode 2. (pair of figures A) vs obtained results from Scipion of the cross-correlation for GNM mode 2 (pair of figures B)

We can now move on to the next analysis, in the article [27] the second non-zero GNM mode is shown for both molecules as overlaid chains. We can accurately do this on our new viewer by typing the number 3 on the box from the single mode analysis of the display, as the article is discarding the first mode and we are including it so our numbers are higher by 1. This condition is the same as for the previous test (covariance and cross correlation matrices). We can see that the results match perfectly so we could give the check for this analysis and take them as correct.

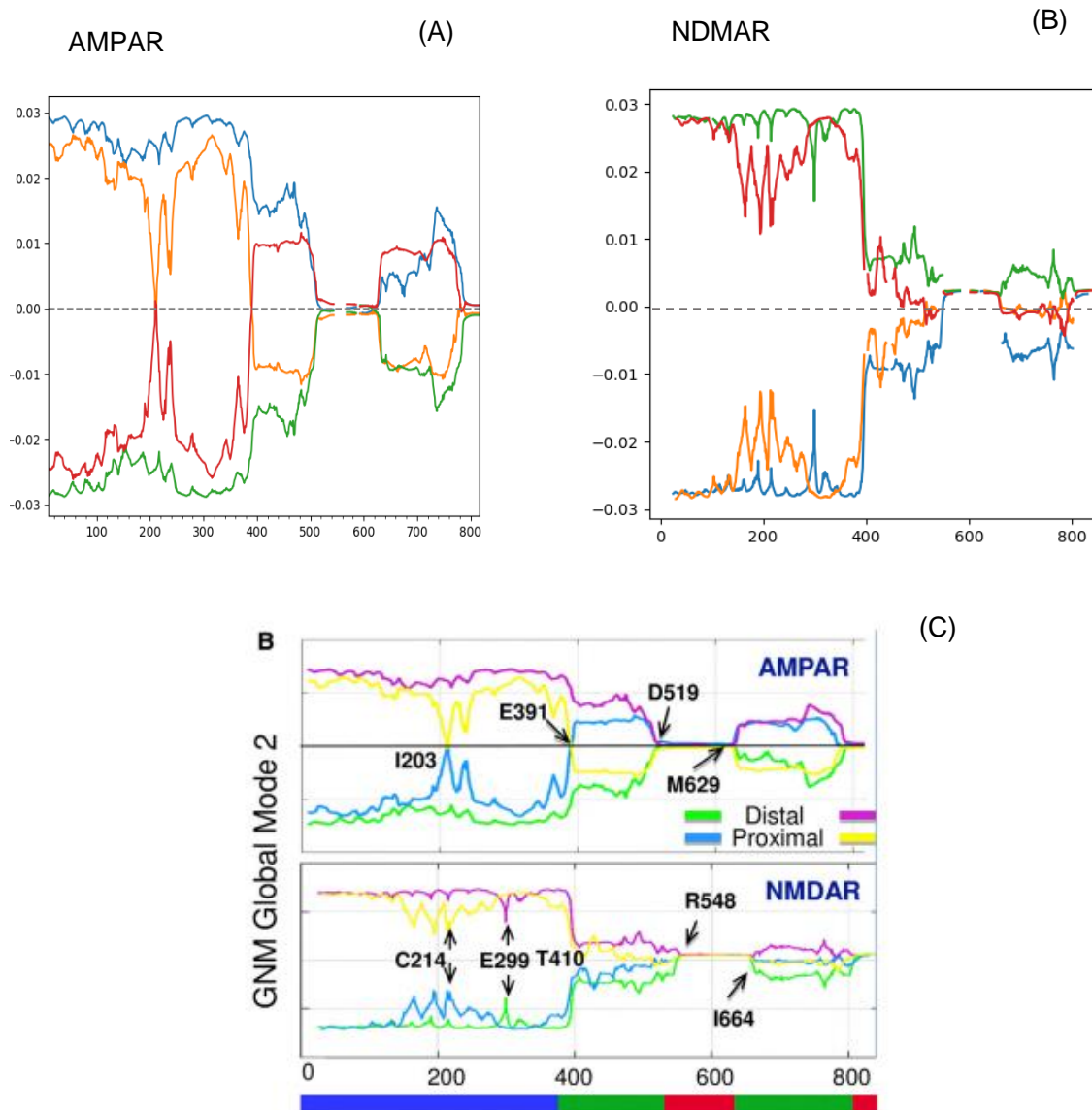


FIGURE 4.6: Results obtained from Scipion vs the figure 8B of the article [27].

GNM mode 2 for AMPAR and NDMAR showed as overlaid chains. (Scipion results are figures a and b vs Scientific file which is figure c)

For the checking of the mean square fluctuations we will follow the ProDy tutorial of GNM [28] and compare the results with that. In the ProDy tutorial, the analysis that has been done is the GNM analysis of Ubiquitin (PDB structure: 1aar). This is a small protein which is very common in the eukaryotic cells and its function is the recycling of other proteins. It attaches itself to other proteins to mark them for degradation [29].

When we first try to make the analysis in Scipion we notice that in the tutorial, the analysis is not being done to the whole structure but only for some certain number of atoms as we see in figure 4.7 obtained from a simulation in Jupiter notebook. See that the selection is done only for the Calphas of the first 70 atoms of the chain A. We can check the ProDy website to see more string selections examples [30] as noted in the help of the selection protocol.

```

In [4]: ubi = parsePDB("/home/ricardo/testing/laar.pdb", compressed=False)
        ubi

@> 1218 atoms and 1 coordinate set(s) were parsed in 0.03s.
@> Secondary structures were assigned to 94 residues.

<AtomGroup: laar (1218 atoms)>

In [5]: calphas = ubi.select('calpha and chain A and resnum < 71')
        calphas

<Selection: 'calpha and chai...and resnum < 71' from laar (70 atoms)>

In [6]: calphas.numAtoms()

70

In [7]: gnm = GNM('Ubiquitin')
        gnm.buildKirchhoff(calphas)

@> Kirchhoff was built in 0.00s.

```

FIGURE 4.7: Desired behaviour to replicate the results of the file [28] programmed on jupyter.

We can implement this in our Scipion project using the *prody – atom selection* box as in figure 4.8:

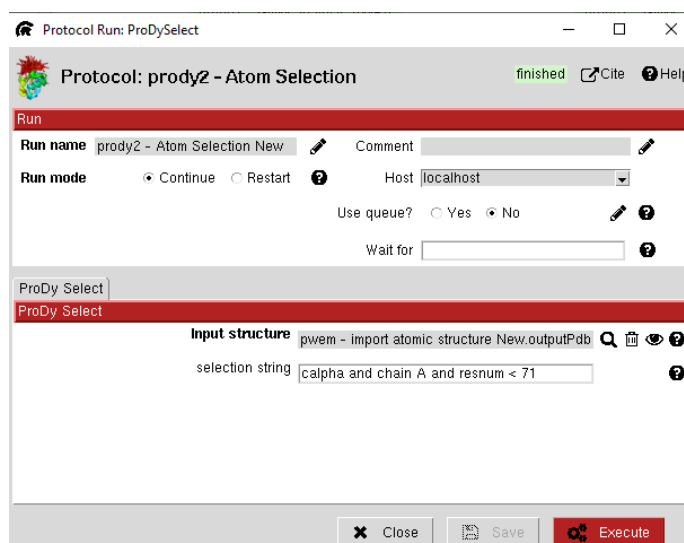


FIGURE 4.8: Example of the use of a selection string in Scipion.

The following figures are obtained from our analysis on Scipion and we can see that they are exactly the same pictures as in the file [28]. (As the pictures are the same they will not be included in duplicate in this report). The tests for the gnm are finished and we can confirm that the results are valid. We can now move to the test for the dynamical domain decomposition.

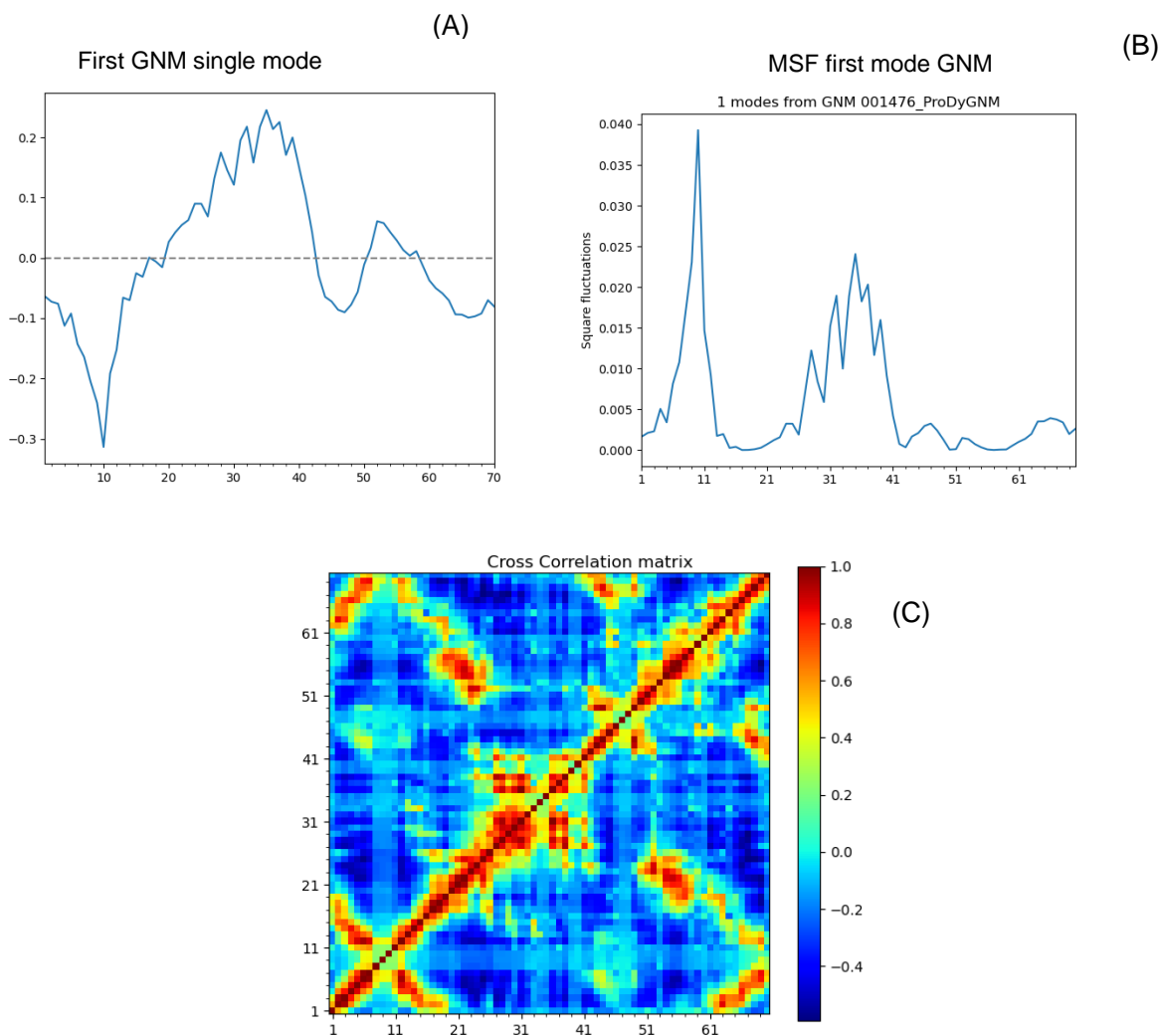


FIGURE 4.9: Results Scipion vs ProDy analysis of Ubiquitin.

First GNM single mode (A), Square Fluctuation mode 1 (B) and Cross-correlation matrix represented as a heat-map (C)

4.4 Analysis of the results of the dynamical domain decomposition

We can end this chapter by detailing the results obtained for the dynamical domain decomposition of the two molecules (AMPA and NMDA). In order to appreciate the different results of this kind of analysis, simulations will be done for 20 and for 5 modes.

The different colour groups that will be drawn represents sectors of the molecule that may have similar dynamics. The bigger the number of modes used, the more groups of colours we will have and the more difficult will be to simplify the dynamic analysis of the molecule. [17]

For all the results we can observe that the number of modes in dynamical domain reflects the number of sectors of the molecules that would have approximately the same dynamics, so this means that the number of colours in the figure will be the same as the number of modes analysed. In our case for the 5 modes we should get 5 different colours (dynamical domains) and for the 20 modes we should have 20 different colours. The results are showed below in figure 4.10 and 4.11:

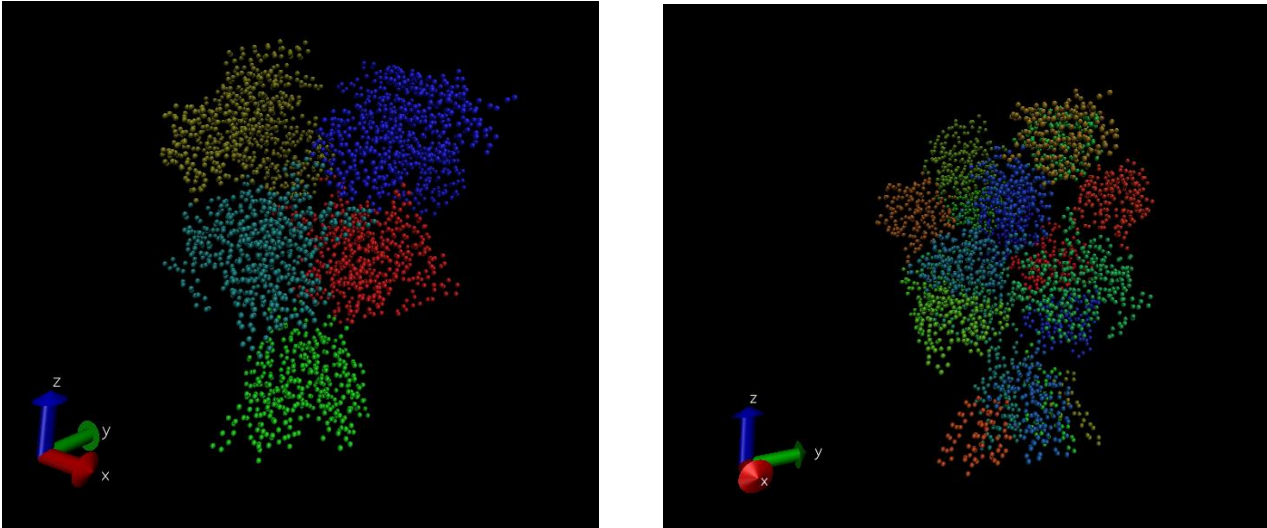


FIGURE 4.10: VMD results of the dynamical domains for 5 (left) and 20 modes (right) for NMDAR

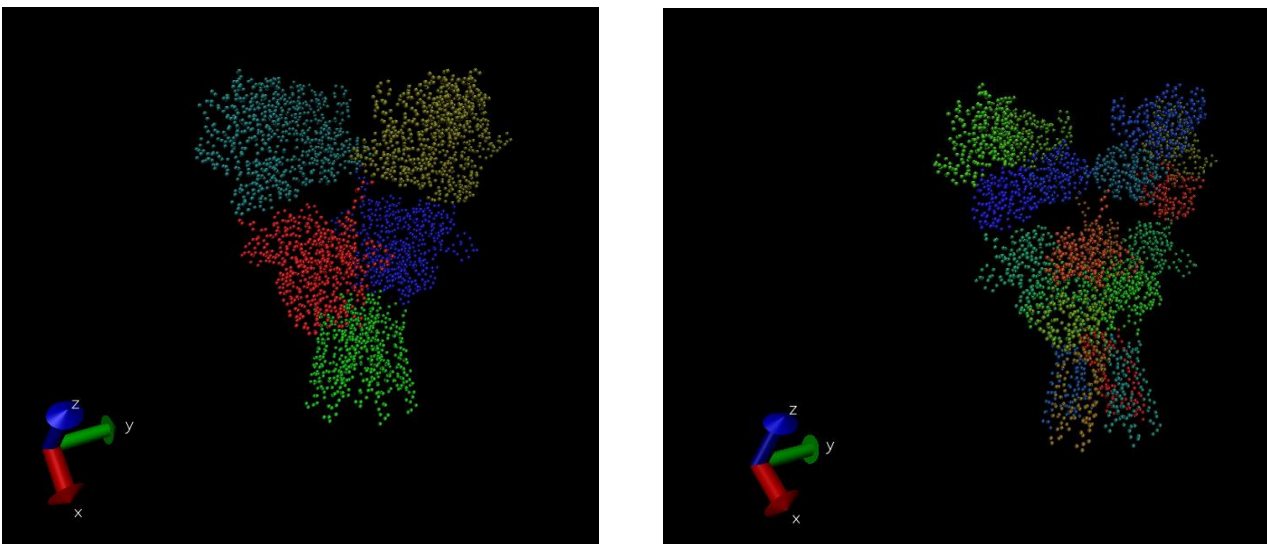


FIGURE 4.11: VMD results of the dynamical domains for 5 (left) and 20 modes (right) for AMPAR

5. Conclusions and future work

5.1 Conclusions

In this section we will go through the proposed objectives in the chapter 1 and check whether they have been achieved. As the way the project was tested was by simulating a case of use by replicating some existing studies and checking whether an average user with limited notions in programming, would be able to accomplish their study necessities, we can assure that the main objective of the project, that I made a friendly environment of the ProDy protocols using the Scipion engine, has been successfully achieved.

Of course, the other two main objectives have been also completed and, linking them with the previous one, we can assert that almost any user would be able to make a very complete GNM analysis using this tool thanks to all the implementations done.

While making the study of the actual state of the art, I have assimilated the main concepts relating, not only to the GNM analysis, but also to other Normal Modes Models such as ANM, and of course the analysis of the dynamical domain decomposition. Thanks to all this acquired knowledge I was able to make a deeper analysis on the necessities of the code, so it would solve as much of the problems as possible that the final user may receive.

The understanding of new tools for me like GitHub, it is also a good point to mention. The collaborative environment that this kind of workspaces offer has been a very important part of this project and it has meant an efficient way to solve the problems with the code and the potential errors.

The complete development of this project was made in a parallel way of working. This means that the tests were being done at the same time as the code was being developed in order to be able to correct the possible mistakes and to get feedback from every function developed. This has let me adapt the necessities that were appearing in some certain stages of development by going back to previous codes developed to solve the particular need.

Concluding, under my point of view, this project is to be considered a very complete and solid work, were the need for adaptation and the ability to understand how biological structures work have been determinant for its success. The understanding of all this background is due to the acquired knowledge of algebra and calculus during the degree that have let me make a quick analysis on how the Network Models are conceived.

Subjects like Network analysing I and II or Object Oriented Programming have given to me the knowledge to be able to complete the coding in Python of this project. The understanding of the C language was also helpful but not crucial. Thanks to this project I was given a first contact with bioinformatics and computational biophysics and I have been shown the multiples possibilities that this field of the technology offers.

5.2 Future work

Regarding the future work for this project, we could have the following possibilities in order to complete the and improve the implementations done:

- **The development of a formal programmed test:** It is very important for this kind of projects to make sure that everything works correctly. An important implementation that needs to be done in the future is the coding of some formal test that checks that every module have been installed correctly. Ultimately, to make a code able to do what we have done qualitatively, but automated. Some examples of this exist for other parts such as ANM analysis.
- **Complete the compare modes pipeline:** An interesting implementation that we mentioned above would be to make the compare modes protocol able to compare different sets of atoms. For example to make the comparison of the mentioned study [27] between the AMPA and the NMDA GNM modes or normal modes. This should be achievable with multiple existing protocols but they will likely need further development to work properly.
- **New protocols from ProDy:** A good improvement that should be done is the inclusion of functionalities from ProDy inside Scipion as we did for the GNM and the dynamical domain decomposition but for new ones. ProDy has hundreds of functions for the calculations and analysis of the dynamics of proteins. A good example could be the implementation of the calculation of the Markovian hitting time and the perturbation response scanning (PRS), two methods that use ANM or GNM results to probe signal propagation through these networks inside the scipion-em-prody package. It is good to have multiple tools for similar analyses in the same framework for comparison and consensus. Other tasks related to these could also be included such as averaging results over dynamical domains to understand general properties of larger protein complexes as done in a recent study [17]. This is a kind of result that could be interesting to have in future updates of this project.

Bibliography

- [1] H. Li, Y. Chang, J. Lee, I. Bahar and L. Yang, "DynOmics: dynamics of structural proteome and beyond.," *Nucleic Acids*, vol. 45, no. 1, pp. 374-380, 2017.
- [2] Scipion, "Installing Scipion," [Online]. Available: <https://scipion-em.github.io/docs/docs/scipion-modes/how-to-install.html>. [Accessed 1 Junio 2022].
- [3] J. d. I. Rosa-Trevín, "Scipion: A software framework toward integration, reproducibility," *Journal of Structural Biology*, 20 April 2016.
- [4] Centro Nacional de Biotecnología, "Xmipp," [Online]. Available: <http://xmipp.i2pc.es/>. [Accessed 19 June 2022].
- [5] E. Pettersen, T. Goddard, C. Huang, E. Meng, G. Couch, T. Croll, J. Morris and T. Ferrin, "UCSF Chimera, an extensible molecular modeling system," *Protein Sci.*, 30 January 2021. [Online]. Available: <https://pubmed.ncbi.nlm.nih.gov/32881101/>. [Accessed 19 June 2022].
- [6] "GitHub," [Online]. Available: <https://github.com/scipion-em/scipion-em-continuousflex>. [Accessed 19 June 2022].
- [7] Prody, "ProDy, Protein Dynamics & sequence Analysis," [Online]. Available: <http://prody.csb.pitt.edu/>. [Accessed 1 June 2022].
- [8] W. Humphrey, K. Schulten and A. Dalke, "VMD: visual molecular dynamics," *Journal of molecular graphics*, vol. 14, no. 1, pp. 33-8, 1996.
- [9] A. B. e. al., "roDy: protein dynamics inferred from theory and experiments," *Bioinformatics (Oxford, England)*, vol. 27, no. 11, pp. 1575-7, 2011.
- [10] A. e. a. Bakan, "Evol and ProDy for bridging protein sequence evolution and structural dynamics," *Bioinformatics (Oxford, England)*, vol. 30, no. 18, pp. 2681-3., 2014.
- [11] J. M. Krieger and S. Zang, "ProDy 2.0: increased scale and scope after 10 years of protein dynamics modelling with Python," *international society for computational biology*, vol. 37, no. 20, p. 3657-3659, 5 april 2021.
- [12] I. Bahar, T. Lezon, L. Yang and E. Eyal, "Global dynamics of proteins: bridging between structure and function.," *Annual review of biophysics*, vol. 39, p. 23.42, 9 June 2010.
- [13] "ProDy Tutorial: Gaussian Network Models," [Online]. Available: http://prody.csb.pitt.edu/_static/ipynb/workshop2021/prody_gnm.html. [Accessed 2 June 2022].
- [14] B. Erman, "The Gaussian Network Model: Precise Predictions of Residue Fluctuations and Application to Binding Problems," *Biophysical Journal*, vol. 91, no. 10, pp. 3589-3599, 2006.
- [15] L. T. B. A. S. I. Ivet Bahar, "Normal mode analysis of biomolecular structures: functional mechanisms of membrane proteins.," *Chem Rev*, vol. 110, no. 3, pp. 1463-1497, 2010.
- [16] N. Sauerwald, S. Zhang, C. Kingsford and I. Bahar, "Chromosomal dynamics predicted by an elastic network model explains genome-wide accessibility and long-range couplings," *Nucleic Acids Research*, vol. 45, no. 7, pp. 3663-3673, 2017.
- [17] Y. Zhang, J. M. Krieger, K. Mikulska-Ruminska, B. Kaynak, C. O. Sanchez Sorzano, J. Carazo, J.

- King and I. Bahar, "State-dependent sequential allostery exhibited by chaperonin TRiC/CCT revealed by network analysis of Cryo-EM maps," *Prog Biophys Mol Biol*, vol. 160, pp. 104-120, 2021.
- [18] Microsoft, "docs.microsoft.com," [Online]. Available: <https://docs.microsoft.com/es-es/windows/wsl/install>. [Accessed 6 June 2022].
- [19] Mobatek, "MobaXterm, Enhanced terminal for Windows with X11 server, tabbed SSH client, network tools and much more," [Online]. Available: <https://mobaxterm.mobatek.net/>. [Accessed 19 June 2022].
- [20] "ipython," [Online]. Available: <https://ipython.org/>. [Accessed 19 June 2022].
- [21] "GitHub," [Online]. Available: <https://github.com/scipion-em/scipion-em-prody>. [Accessed 19 June 2022].
- [22] RicardoSerr, "GitHub," [Online]. Available: <https://github.com/RicardoSerr/ProDy>. [Accessed 19 June 2022].
- [23] R. Serrano, "GitHub," Implementation of the RMSF calculation for modes, 2022. [Online]. Available: [com/prody/ProDy/commit/7bdde4c423b1b108c9ca29f046e2501f12b4770d](https://github.com/prody/ProDy/commit/7bdde4c423b1b108c9ca29f046e2501f12b4770d). [Accessed 19 June 2022].
- [24] R. Serrano, "ShowAtomicLines is fixed for multiple chains," GitHub, [Online]. Available: <https://github.com/prody/ProDy/commit/3614c31cfa3bab9b792137a2fd8bfdca62439cd>. [Accessed 19 June 2022].
- [25] R. Serrano and J. M. Krieger, "scipion-em/scipion-em-prody commits by RicardoSerr," GitHub, 2022. [Online]. Available: <https://github.com/scipion-em/scipion-em-prody/commits?author=RicardoSerr>. [Accessed 19 June 2022].
- [26] R. Serrano and J. M. Krieger, "prody/ProDy commits by RicardoSerr," GitHub, 2022. [Online]. Available: <https://github.com/prody/ProDy/commits?author=RicardoSerr>. [Accessed 19 June 2022].
- [27] J. M. Krieger, A. Dutta, J. Y. Lee, j. Garcia-Nafria, i. H. Greger and I. Bahar, "Cooperative Dynamics of Intact AMPA and NMDA Glutamate Receptors: Similarities and Subfamily-Specific Differences," *CellPress*, vol. 23, no. 9, pp. 1692-1704, 2015.
- [28] ProDy, "Gaussian Network Model," [Online]. Available: http://prody.csb.pitt.edu/tutorials/enm_analysis/gnm.html. [Accessed 13 June 2022].
- [29] L. Ahyoung, J.-T. Hwang and K. Changwon, "Ubiquitin and Ubiquitin-like Proteins in Cancer, Neurodegenerative Disorders, and Heart Diseases," *International journal of molecular sciences*, vol. 23, no. 9, 2022.
- [30] ProDy, "Selections," [Online]. Available: <http://prody.csb.pitt.edu/manual/reference/atomic/select.html#selections>. [Accessed 13 June 2022].
- [31] "Prody; calcGNMDomains," [Online]. Available: http://prody.csb.pitt.edu/_modules/prody/chromatin/cluster.html#calcGNMDomains. [Accessed 2 June 2022].

