PyEDA:
A Software to analyze Normal Modes in NMR and MD Structures using an Essential Dynamics approach.*

Joan Francesc Gilabert & David Mas - Prof:Javi Garcia & Baldo Oliva.

March 18, 2016

Abstract

In this project we have developed PyEDA, a python based software capable of analyzing Normal Modes from NMR structures and MD trajectories in pdb format. It uses an Essential Dynamics approach and it gives different outputs like a structure pdb file with the superimposed structures, the eigenvalues in increasing order, the fluctuations of the structure by residue and a final structure with the main motions of defined normal modes applied. It can be used as a python Package, with a tkinter interface and also as a CLI standalone program. The main dependencies of the software are BioPython[1], Tkinter, Matplotlib[2], NumPy and SciPy[3] packages.

Contents

*Introduction to Pythons & Structural Bioinformatics - M. Sc. in Bioinformatics for Health Sciences at Universitat Pompeu Fabra

1 Introduction to NMA and Essential Dynamics

Normal mode analysis or NMA are powerful and common methods used by the protein field to study the collective motions of biomolecules. Its arise comes from the fact that protein function, protein structure and protein dynamics are closely related. Knowing how a part of a protein is moving can help us determine weather a specific part of the structure is relevant in terms of functionality. Although it is not covered in this software, dynamics analysis can also help in the use of homology modeling to order to determine evolutionary patterns in highly conserved sequences.

There are 3 important approaches to perform a NMA:

- Standard Normal Mode Analysis
- Elastic Network Models
- Essential Dynamics

The presented software uses a Essential Dynamics approach but this tutorial is going to give a brief explanation of the basics of the other approaches.[4]

1.1 Standard Normal Mode Analysis

The Normal Mode Analysis are mostly used to probe large-scale and shape-changing motions in biological molecules. It has connections with experimental techniques such as infrared and Raman spectroscopy but its main use is to theoretically predict motion in proteins.

The most important Normal Modes are the ones that have greater fluctuation and lower frequency because as any other biological sequence they have emerged from evolutionary design rather than by chance. For example, a movement of a protein produced by chance will be not just in one direction but in many and therefore its fluctuation in one directory will be reduced by the fluctuation in the other directory. The movement of the protein to bind or interact with another protein will be seen in NMA as a unique Normal Mode with a great fluctuation.

In its purest form, NMA is an harmonic analysis and at the end it uses the same force fields that the ones used in the Molecular Dynamics Simulation. There are 3 steps in order to perform a NMA of a protein.

i In the first step a standard NMA minimize the conformational potential energy as a function of the atomic coordinates

ii Next, it will compute a Hessian Matrix using second derivatives of the potential energy for each coordinate

iii Finally, it will compute the diagonalization of the Hessian matrix obtaining eigenvalues and eigenvectors also named as "normal modes".

Each of these steps are very computationally demanding. Because of this complexity associated to these computations other more simple approaches have been used in order to minimize the computational cost of a NMA.

1.2 Elastic Network Models

The Elastic Network[5] approach is a big simplification of the problem and the results are not much different from the standard NMA(sNMA) considering the difference from an sNMA to reality. At the end this Elastic Network approach uses a protein model that is dramatically simplified. In this model the atoms are connected by a network of elastic connections. The advantages versus the traditional approach are the elimination of the energy minimization step and the reduced number of atoms considered. The first advantage is held because the approach already consider the elastic connections to be at their minimum length. The reduced number of atoms is due to the use of C_α only for the analysis as they are the points connected in the protein. For most of the functional movements and dynamics of proteins it is enough to consider these atoms as they represent the structure shape.[4]

1.3 Essential Dynamics

The Essential Dynamics approach[6] is a completely different computation. It is based on a principal components analysis (PCA). This radical change has its importance in the complexity of biomolecules. Proteins and in general all biomolecular systems have complex dynamics simulations that can be hard to analyze and to uncover functional mechanisms or motions. This disadvantage of the previous approaches is solved using Essential Dynamics as it only remark the most important modes. Interestingly, it has been found that most of protein dynamics can be described by a low number of degrees of freedom. Moreover this approach have the huge advantage that the dynamics can be inspected mode by mode and separately. Allowing to use only the most characteristic modes.

In contrast to NMA, PCA of a molecular dynamics simulation trajectory does not rest on the assumption of a harmonic potential. In fact, PCA can be used to study the degree of anharmonicity in the molecular dynamics of a simulated system. It was shown in proteins that in physiological temperatures especially the major modes are dominated by anharmonic fluctuation and therefore can be predicted using this approach[6].

Using Essential Dynamics, the modes are sorted according to variance rather than frequency. Nevertheless, the largest-amplitude modes of a PCA usually also represent the slowest dynamical transitions. Therefore the conclusions of an standard NMA analysis stands with this approach.[4]

1.3.1 Steps in a Essential Dynamics approach

The steps to perform this approach are the following:

i Superposition to a common reference structure. This step is crucial particularly in NMR structures.

ii Construction of a covariance matrix of positional fluctuations using atom coordinates

iii Diagonalitzation of the covariance matrix to obtain eigenvalues and eigenvectors representing the modes of motion

iv Projection of the original configurations onto a defined set of principal components to get principal coordinates.

The matrix construction is the key step. It is build first constructing a vector with all the 3 coordinates for each atom and for each model and then computing the means of each position for different models (for NMR structures) or time frames (for Molecular Dynamics). After these 2 small steps the C matrix is constructed as a co-variance matrix using the following equation,

$$C = \langle (x_i(k) - \langle x_i \rangle) \cdot (x_j(k) - \langle x_j \rangle) \rangle \tag{1}$$

where $x_i(k)$ and $x_j(k)$ are the atomic coordinates of the atoms i and j in configuration k[6,7].

In order to obtain the modes a diagonalization step for solving this must be held.

$$\Lambda = \mathbb{V}^T \mathbb{C} \mathbb{V} \tag{2}$$

where Λ is a diagonal matrix (containing the eigenvalues) and where $\mathbb{V}$ is the matrix formed by the eigenvectors, each one corresponding to the eigenvalues of Λ.

The eigenvectors are the representation of each normal mode. The eigenvalue that is associated with a eigenvector (or mode) indicates the relative contribution of that mode (or eigenvector) to the motion of the whole protein structure.[7]

Therefore, the normal modes can be sorted using the eigenvalues as representation of their contribution to the protein. Usually in decreasing order. For a structure of N atoms, $\mathbb{C}$ would be a 3N x 3N matrix. As a result of the diagonalitzation, $3N6$ eigenvectors with nonzero eigenvalues will be obtained. There should be 6 eigenvalues equal to 0 because the corresponding eigenvectors describe the overall rotation and translation that is eliminated in the superposition step. In practice, with full-protein structures and considering M as the number of models or frames, and $3N > M$, we will get at most $M - 1$ eigenvalues different from 0, since the covariance matrix will not have full rank. To sum up, in most of the analyses we would be able to obtain $M - 1$ nonzero eigenvalues which is not a big deal considering we are only interested in the major ones[4].

The next step is the projection of the principal components into principal coordinates. To do so, the following 2 equations have been used:

$$p_i(k) = \mu_i \cdot (x(k) - \langle x \rangle) \tag{3}$$

$$x'_i(k) = p_i(k) \cdot \mu_i + \langle x \rangle \tag{4}$$

In equation (3) we project the eigenvectors to the original coordinates to get the principal coordinates or p_i and then, in equation (4) we transform back these projections to Cartesian coordinates. This is an important step for visualization of the motions. In equation (3), the eigenvector is represented with the μ and the particular coordinate is represented with $x(k)$. In Equation (4), the representation stands.[4]

2 Practical Tutorial

2.1 Work-flow

The workflow is summarized in the following figure:

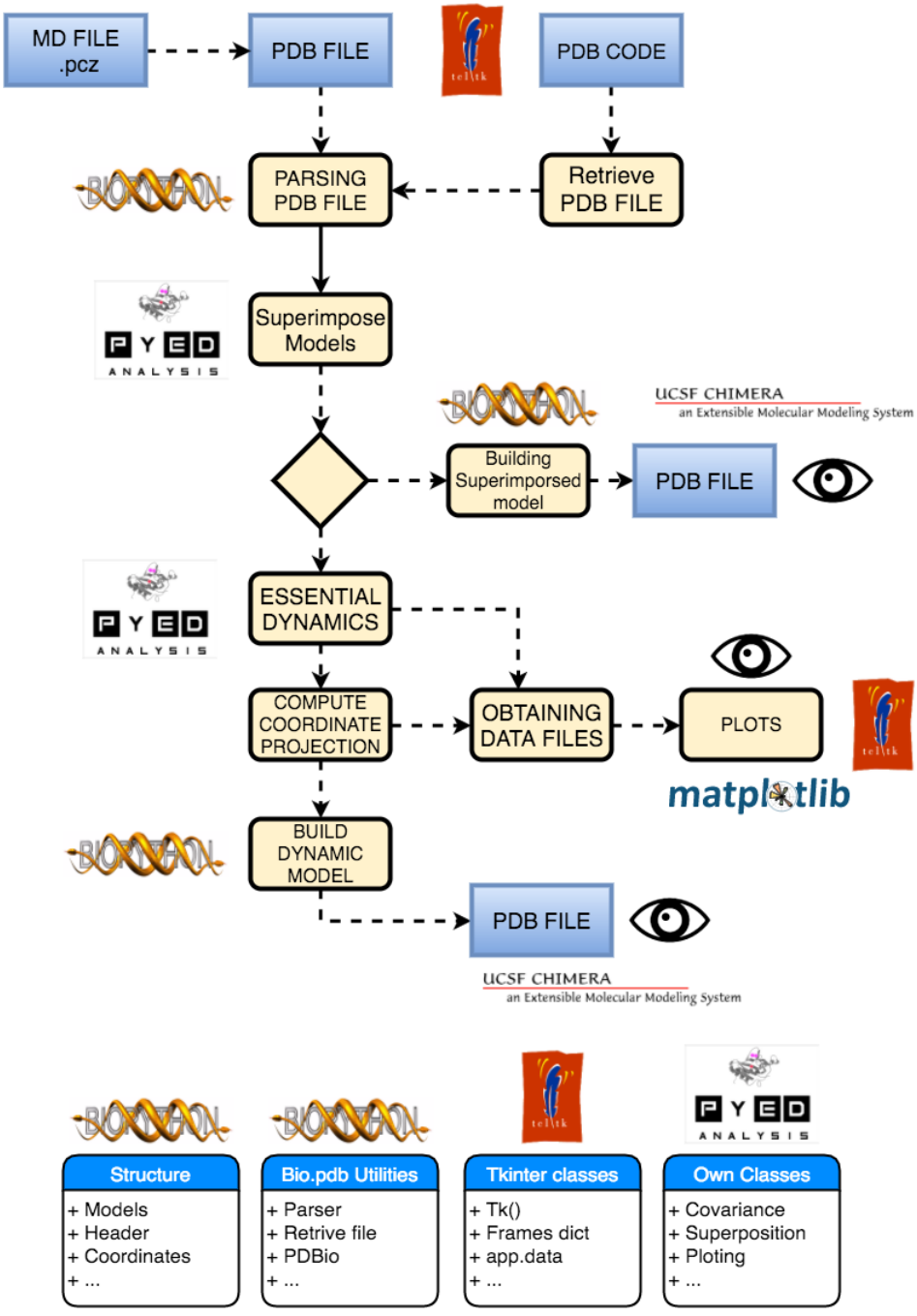


Figure 1: Entire Workflow for EDA

As seen in the figure 1, the user can introduce either a file or a pdb code. If the user introduce a pdb code the software retrieve the structure from pdb (if internet connection is available). The structure that must contain Models or Frames in order to analyze the

Normal modes. Then the Superimposition takes place and generates the first pdb file with the superimposed strucutres of the different modes. After that the Essential Dynamics step is performed obtaining the eigenvectors and eigenvalues that represent the normal modes. The regular plots obtained after this step (eigenvalues vs eigenvector index) can be obtain through a matplotlib integration in a Tkinter window. After that, the projection is performed and the projections plots (RMSD vs Residue Number) is sent to the same Tkinter window mentioned before. The final step is to generate a final pdb file containing the projected structure.

As you can see the main dependencies are included in the figure and represented by their logos when used. The Tkinter interface covers almost all steps from loading the data to the software to the plots in the visualization step. Moreover, BioPython[1] and NumPy and SciPy[3] packages are used almost everywhere. This Software is based in a *"do not reinvent the wheel"* approach. The generated pdb files can be visualized using either CHIMERA or PyMol softwares.

2.2 Step by Step guide

2.2.1 Checking Dependencies

Before starting the installation procedure we have to check whether or not we have the required packages installed. Open a shell terminal window and enter into the python3 console. Type the following to see if the packages are installed.

```
$ python3
>>> import numpy
>>> numpy.version.version
'1.10.2'
>>> import scipy
>>> scipy.version.version
'0.17.0'
>>> import tkinter
>>> tkinter.TkVersion
8.5
>>> tkinter.TclVersion
8.5
>>> import Bio.PDB
>>> import Bio
>>> print(Bio.__version__)
1.66
```

Note that it may work with older versions, these versions are the ones used to build the software. If some of this imports do not import the package you must installed before using our software.

2.2.2 Installing the Package

We provide a package with all the code by means of the python disutils distribution system. Once you have obtained the package you can install it with the command

```
sudo python3 setup.py install
```

Alternatively you can install it without the need of elevated permissions with

```
python3 setup.py install --user
```

This will install the package in your python path. Once installed, you can import PyEDA and all its modules:

- edanalysis: to write your own analysis
- helper_module
- interface: is the base of the graphical interface

2.2.3 Package contents

As mentioned before, the package has three main modules that can be imported. This can be helpful if you want to extend the analysis further, or focus on a certain aspect.

We will start describing the helper_module. As its own name indicates, this module contains several functions that are used by the program but are not closely related to the ED analysis. Examples of these functions are: merge_the_header, which is a function that allows to write a PDB including the header information, a feature which is missing in the biopython PDB module or pdb_code_check, which check if the code is a valid pdb code.

The edanalysis module contains most of the ED-specific code. It constains the ED-Analysis class, which serves as the backbone of our program. It has the following methods:

- __init__ - Set the basic attributes of the EDAnalysis
- get_id - Return the id of the object (as PDB code)
- get_mode - Return the mode of the object
- get_atom - Return the atom list chosen for the analysis
- superimpose_models - Superimpose two or more structures
- createcordsarray - Calculate the coordinates for the diferent models
- cal_cov - Calculate the covariance matrix
- plot_eig - Plot the values of the principal eigenvectors of the covariance matrix
- plot_eig_wosv - Same as plot_eig but adapted to some requirements of the Tkinter GUI
- is_NMR_struct - Check if the structure loaded is was obtained by NMR
- check_mode - Check if the mode selected is consisted with the data
- move_structure - Move the structure along a certain eigenvector
- RMSD_res_plot - Create a plot with the distance of the residues along some eigenvectors

and attributes:

- PDBid(private) - String with the PDB code of the structure anlyzed

- mode(private) - String with the mode of the anlysis (for now NMR or MD)
- atom(private) - List with the atoms to study
- structure - Bio.PDB.Structure object on which the analysis is performed
- reference - Bio.PDB.Structure object with a X-Ray structure that will serve as a reference model, this should only be used with the MD mode
- n - Number of models or trajectories (in NMR or MD modes,respectively)
- N - Number of coordinates of the analysis
- coords_array - nxN Numpy array with the N coordinates for the n models
- means - 1xN Numpy array with the means of the N coordinates over the n models
- C - NxN Numpy array that contains the covariance matrix
- eigvc - Numpy array with the eigenvectors of the covariance matrix
- eigvl - Numpy array with the eigenvalues of the covariance matrix

Additionally it contains a custom Exception class called WrongExceptionMode.

Finally, the interface module contains all the code relevant for the graphical interface. It contains basically a class for each window displayed in the interface and the main window which wraps them all up. Each class have its own methods and attributes that are important for the interface function. There are other methods and attributes, not shown, that are actually buttons, labels or entry boxes.

- EDA_app - It is the main window where all the other frames are projected. It contains the application data a a dictionary to switch easily between frames. It is inherited from tkinter.Tk.
  - app.data (attribute) it is a dictionary that contains the important data for the EDA module that needs to be passed from one frame to the other.
  - frames (attribute) it is also a dictionary that stores the frames of the application.
  - show_frame (method) It is the only method in this class and it is used to switch to a specific frame wherever frame the user is. It uses information from the frame attribute.
- StartPage - It is the first frame the users can see after the execution. It leads to the loading data frame or the "about us" page. It is inherited from tkinter.Frame
- initial_root - Acts as a loading data frame. Is where the user interacts with the program by entering the required data and options. It is inherited from tkinter.Frame
  - get_pdb (method) pass on the pdb code and move the user to the waiting_window
  - select_file (method) pass on the pdb filename and move the user to the waiting_window

- waiting_window - It is where the analysis is taking place. Incorporates the same steps that the PyEDA.py CLI except the projection step. After hitting the analysis button it redirects automatically to the plotting frame. It is inherited from tkinter.Frame
  - analysis (method) It contains all the computations needed to perform the EDA. It also generate the plots at the end. When the plots are generated move the app to the plot_window.
- plot_window - It shows buttons to the different ploting capabilities of the software. The plots are already computed and stored in the app data but here the canvas is set. It integrates matplotlib in the Tkinter window. It also contain an option to perform the projection. The user can select the eigenvector and the maximum span as integer values.
  - eigen_plot (method) It displays the plot eigenvalue vs eigenvector index.
  - RMSD_plot (method) It shows the RMSD vs Residue plot.
  - trajectory (method) It generates a trajectory file (.pdb) given the eigenvector index and the total span defined from the user
- About_EDA - It is a information page that contains an explanation of the software features.

2.2.4 Usage

As we mentioned before, our program can be used with a graphical interface or via the command line.

Graphical Usage The graphical usage is set by default. You only need to call the software via terminal like this.

```
$ python3 -m PyEDA
```

The best way to see how the interface works is by checking the video tutorial or trying it yourself. You can go to the video-tutorial following the link:

However, the different pages and windows are described below.

1. In the path to the results, the first window it appears is the Start page. The function of this page is just to improve the application status of this software. From here, we can go to the about EDA page and to the starting point of our analysis.
2. In the about EDA page we can see some information about the approach used in the software an what analyses it performs. You can go back to the starting page or go directly to the analysis.
3. Loading data page. In this winsow the user can load the data before starting with the analysis. There is 2 main options to load the data. You can either load a pdb file using the "Add a file button" (it have to be a standard pdb file) or you can introduce a pdb code in the entry box. After that you can press the "check code" button in order to check weather or not your code have 4 characters. If is a pdb-like code the box letters will be green and if there are less or more characters

it will change the color to red. This is not a definite checking step but is a proof of concept capability that can be improved in the future. Then we have the "get pdb" button to start the analysis with the pdb code. Right now, the user cannot import a file and a code at the same time. This page also contains the parameters for the analysis. The user can select weather the pdb code provided is from a NMR structure or from a Molecular Dynamics simulation and also with what atoms of the protein the user want to perform the analysis. Usually, the main movement can be detected only with the C_α atoms but in some proteins the user may need to use all the atoms to compute the Essential Dynamics Analysis. After the "Add file" button or the "get pdb" button the user is send to the waiting window.

4. Waiting window. In this page the user must hit the button "Analysis" to start the computations. Depending on the atoms selected and the size of the pdb file this could be a time consuming step but normally do not take longer than a few seconds. After the analysis are done and the plots are computed internally, the software bring the user to the plot window.

5. In this final window the user can choose which plot want to visualize. When clickin on a plot it enters in a matplotlib like environment where the user can move and interact with the graph and save it in a custom directory. The user can also generate as many trajectories as he wants. Basically to generate a trajectory you should introduce the eigenvector index which is going to be used to project the coordinates and then a full span, which will be the length of the trajectory. This trajectory generation is a time consuming task so if you are performing this analysis using the graphical interface the standard values are from 1 to 3.

....

We have experienced some troubles depending on the OS used in the PDBio class. In some cases the superimposed file "pdbcodealign.pdb" is saved with extra "END" lines that do not let the user directly see the pdb structure in PyMol. If it is your case, we provide you here a fast awk one-liner to solve the problem.

```
$ awk '$0 !~ /^END$/ {print $0} END {print "END"}' 1msfalign.pdb > corrected.pdb
```

CLI Usage The command line interface has been built using the argparse library, in figure 2 a complete description of all the options available can be seen. This description can be accessed with the commands:

```
$ python3 -m PyEDA -h
```

```
$ python3 -m PyEDA --help
```

To use the CLI interface the -ng option has to be specified, followed by any other options that might be relevant for the case to study.

```
usage: __main__.py [-h] [-ng] [-c CODE] [-i INFILE] [-m {NMR,MD}]
                   [-a {CA,Back,all}] [-d PATH] [-v] [--eigvl-num EIGVL]
                   [--time-max TIME] [--time-step STEP] [--eigvc-num EIGVC]

This program performs the essential dynamics(ED) analysis of a protein, given
a NMR solved structure or an MD trajectory. It provides two interfaces, a
graphical and a command line the default is the graphical, and the command
line can be used specifiyin the -ng option

optional arguments:
  -h, --help            show this help message and exit
  -ng                   Use the command line interface, if it is not specified
                        the graphical interface will be called
  -c CODE, --code CODE  Input pdb code for the protein you want to study
  -i INFILE, --input INFILE
                        Input pdb file with either NMR or MD data
  -m {NMR,MD}, --mode {NMR,MD}
                        Select wether you want to perform the ED analysis on
                        NMR or MD data
  -a {CA,Back,all}, --atoms {CA,Back,all}
                        Select wich atoms you want to perform the ED analysis,
                        CA means only Carbon alpha atoms, Back means backbone
                        atoms(CA, N, C, O) or all, this later option is
                        discouraged since it may be computationally very
                        expensive and take more time
  -d PATH, --directory PATH
                        Path to the directory where you want to work, if not
                        specified, current working directory will be used
  -v, -verbose          Verbose description of program actions
  --eigvl-num EIGVL     Number of eigenvalues to plot (default is 4)
  --time-max TIME       Maximum time during which the trajectories of a
                        certain eigenvector
  --time-step STEP      Time step for the generated trajectories
  --eigvc-num EIGVC     Number of eigenvector for which new trajectories will
                        be generated, default is 1 since it is a
                        computationally expensive step
```

Figure 2: Description of the command line usage

3 Analysis of the performance and results

3.1 Comparison of the results with Van Aalten 1998

In order to test the performance of our software we have used the analysis done by Van Aalten et al 1998[8] of a NMR structure of the Myb DNA-Binding Domain. We have been able to reproduce 2 important figures of the article.

In the article, they used an Essential Dynamics approach to analyze the Normal Modes of the NMR structure Mouse c-Myb R2R3 [PDB entry 1MSF[9]]. They have found large fluctuations of atoms that describe a hinge-bending motion between R2 and R3 repeats. They have observed that only with NMR data. After that, the authors have also used a Molecular Dynamics Simulation to predict and quantify the motion. Finally they have tested the effect of a proline to alanine mutation into the motion observed. The motion in the mutated protein was found to be enhanced.

The 1MSF structure is a NMR file with 25 Models of a binding protein to the a DNA chain. In the figure 3 we can see the first output of PyEDA, a superimposed structure of all the structural modes using PyMol[10].

Although this structure in particular is already superimposed in the pdb file we can see clearly how the different models are here superimposed. To check the superimposition capacity we have used a non superimposed pdb file(see subsection 3.2).

In the following output of the program, figure 4, we can see each eigenvalue of the different eigenvectors ranked by the contribution on the protein motion. The contribution is measured with the eigenvalue of each eigenvector. Here we can see the 24 eigenvectors expected considering that there where 25 models in the initial pdb file. This plots give us an idea of how each eigenvector is influencing the protein motion. The shape of the

Figure 3: 1MSF superimposed structure

plot is identical to the literature[8] so we can assume that the performance of the software is optimal. We can see how the most important part of the movement is kept by the first 5 or 10 eigenvectors. This is a typical behavior for proteins and it gives us the capability to distinguish between the evolutionary fixed moves and the random ones. Although this plot is important to understand the system, biologically it gives us information that in fact there is a evolutionary conserved move.

The net step of the program is to project the original structure with the trajectory of a given eigenvector. Therefore we can see the different structures generated by applying that eigenvector. In order to quantify this motions of the protein the most diferent 2 structures from the ones projected are superimposed in a sliding-window fashion with an window value of 15 residues. For each window a root mean sqare deviation (RMSD) of all C_α positions is computed. This can be seen in the figure 5 that is also almost identical to the literature figure[8]. This type of plot gives us important biological information and the ability to quantify the observed movement. As the authors stress on the paper[8] this shape shows a hinge-bending motion between residues 40 and 60 where the R2 and R3 repetitions end and start respectively.

The final output of the Program will give us a pdb file containing the projections of a defined eigenvector to the original structure stored as different models of the structure. This file, seen in platforms like PyMol[10] is useful to see the actual motion derived from the eigenvector. In this case, this projected file show clearly a hinge-bending motion as the authors state in the paper. A interesting thing to consider is that the hinge-bending motion will be only observed when considering the first and third eigenvectors. As it can be seen in 5 the other eigenvectors do not capture this motion. In the figure **??** we can see all the generated states in one image but the proper way to see the motion is using the PyMol commands,

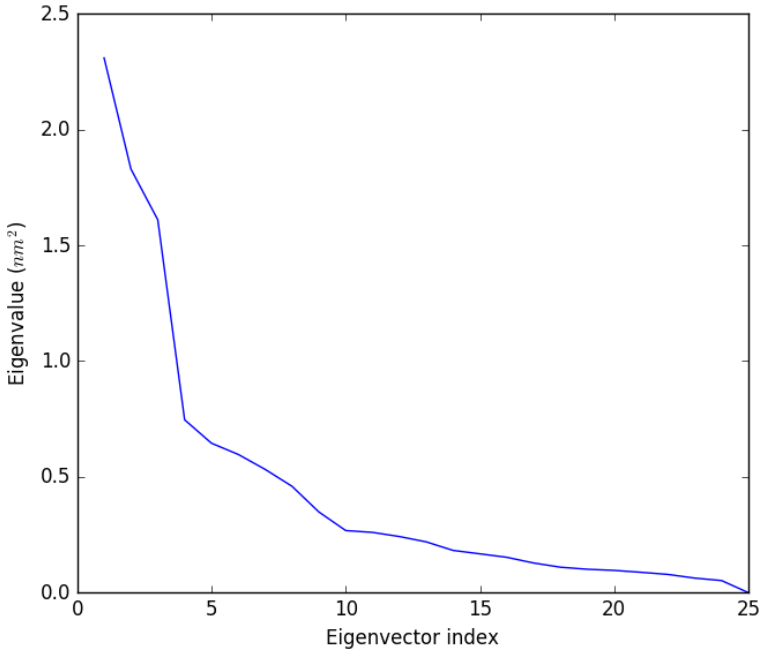


Figure 4: Eigenvalues against their corresponding eigenvector indexes

```
> hide everything
> show ribbon
> mplay
```

It is also possible to see how the residues between the 2 repetitions and the ends of the protein are that get most of the motion. As the motion is larger in the connection between both repetitions in the animation is easy to resemble the bending motion.

3.2 Testing superimposition with 1JOY structure

In order to test the first steps of the software analysis we have used the solution structure of the homodimeric domain of Envz protein from Escherichia coli [PDB entry 1JOY[11]]. This structure is also a NMR and as can be see in figure 7a the database pdbfile is not superimposed.

The superimposition is a critical step in our software, without one of this steps the actual protein movement could not be observed. The figure shows at the left panel the pdb original file and in the right panel the superimposition done by our software. It is quite clear the change between the 2 images and it is useful in terms of considering the importance of this step. Although the RMSD values are not a output of our software the image show how the superimposition strategy is optimal.

This structure is useful to see how protein with more than one chain can be also analyzed. This protein have a homodimeric structure formed by a 60 residues long chain. The motion of this NMR structure is localized on the N-terminal end which is bigger than the C-teminal end. As predicted, in the figure 8 we can see big fluctuation at the first residues and after the 60th residue that indicates the end of the first chain.

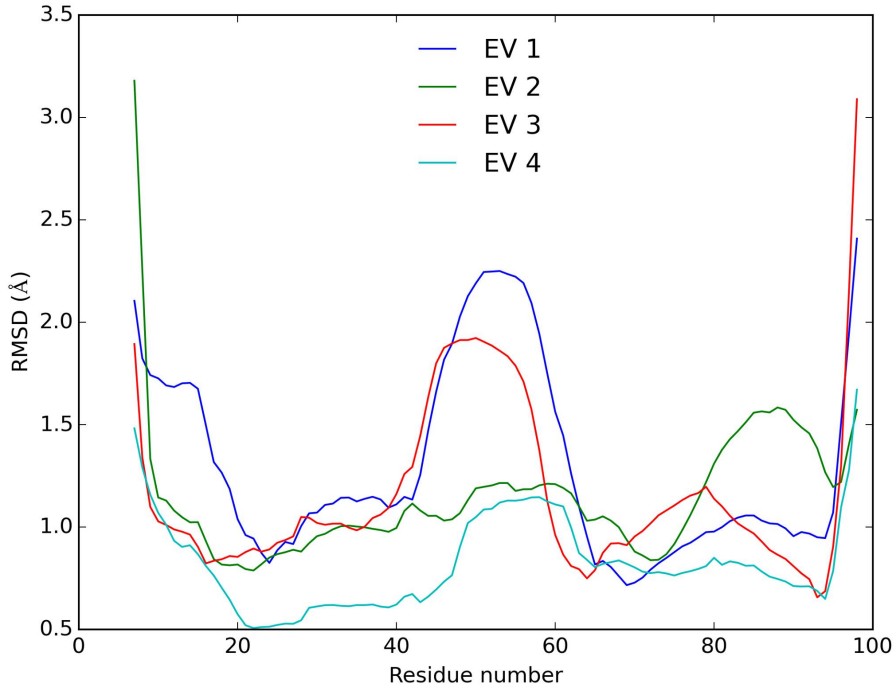


Figure 5: After the projections were computed for each eigenvector, the two structures with the minimum and maximum projections along that eigenvector were used for the superposition. The window size used are 15aa.

3.3 Conclusions and Caveats

PyEDA is a wrap up of the Essential Dynamics approach to get a Normal Mode Analysis of any NMR pdb structure and any MD pdb-like file. As conclusion we may stress that PyEDA is a promising and useful tool specially for the analysis of protein dynamics. It is a unified and complete analysis tool to get protein motion. It is designed to easily uncover and quantify the functional motions of NMR and MD structures. It can be used autonomously using the guided graphical interface or using the CLI which gives the user the possibility to automatize the process. It also have a easy and user-friendly installation process and it is based on the most used programming language, python.

The main improvement that we could implement in the future is a PyMol integration to the tkinter graphical interface. This feature have not been implemented because the actual version of PyMol have only a python2.7 integration meanwhile PyEDA is based on python3. Although a patch code exist to use PyMol with python3 we believe that it may be a stable version of the program in order to distribute PyEDA with the appropriate PyMol integration.

As a major caveats we must stress the large number of dependencies PyEDA have and a previously mentioned problem with the "END" keyword in the pdb file writting that we have not been able to find the source. PyEDA have dependencies on BioPython[1], Tkinter, Matplotlib[2], NumPy and SciPy[3] packages but if the user is a python experimented bioinformatician these packages should be installed and updated in their system. Moreover, to non-experimented python users these packages can also be easily installed using the pip3 module.

The "END" bug have only been detected in a mac OS meanwhile in fedora it works perfectly. These may be because of some Biopython dependencies. It should run okay in

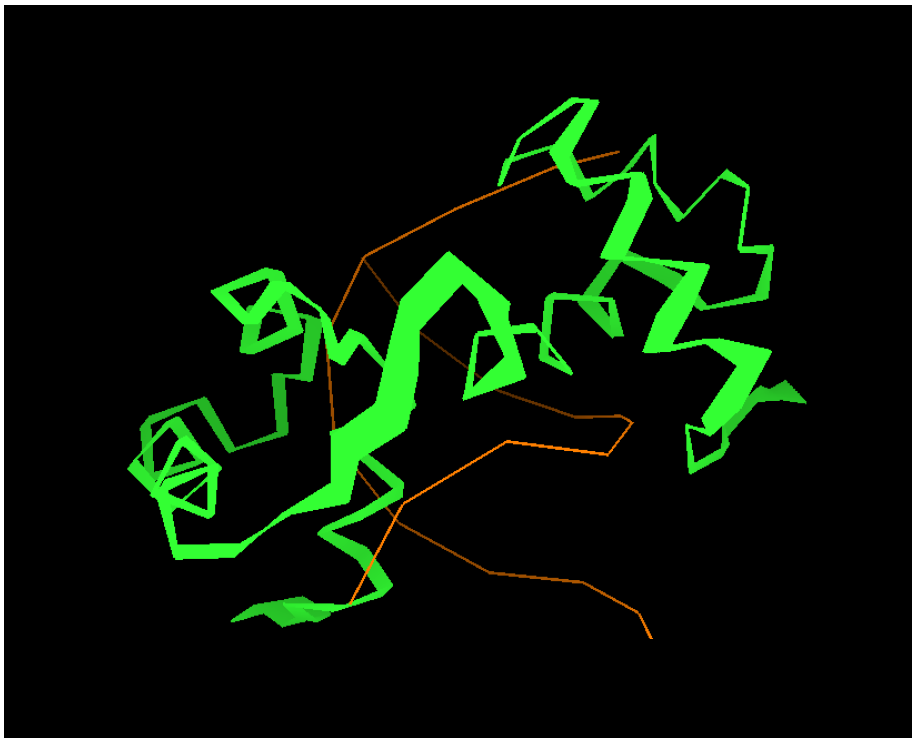
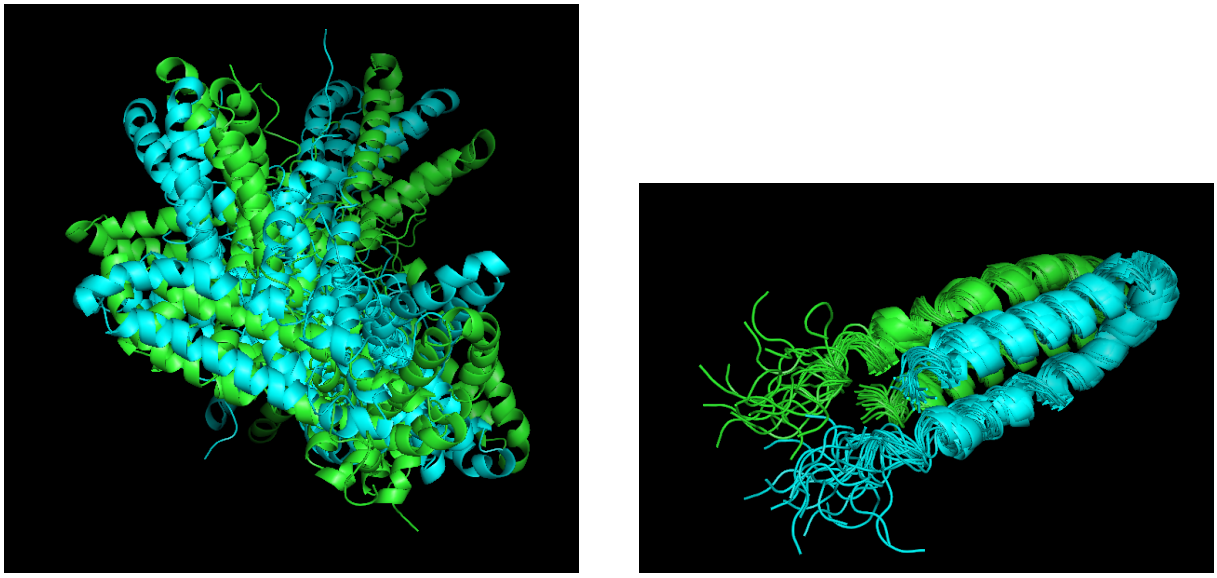


Figure 6: Projected models from 1MSF with total full-span of 3 and step of 0.1 using the first eigenvector

other machines but in case the bug is happening to you we propose an awk one-liner to easily correct this error (see section 2).



(a) 1joy from pdb (b) Superimposed 1joy

Figure 7: PyMol[10] visualitzation of the a) original 1joy structure directly retrieved from the pdb and the b) superimposed 1joy structure by PyEDA.

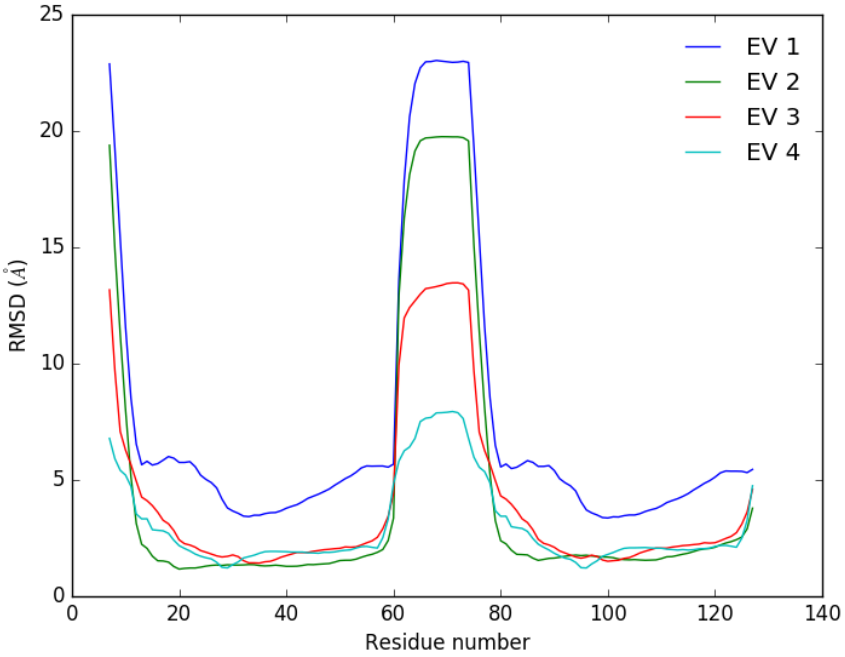


Figure 8: Plot of the most different projected structures. Window size used, 15aa.

List of Figures

References

1. Cock, P. J. A. *et al.* Biopython: Freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **25**, 1422–1423 (2009).

2. Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science and Engineering* **9**, 99–104 (2007).

3. van der Walt, S., Colbert, S. C. & Varoquaux, G. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering* **13**, 22–30 (2011).

4. Hayward, S. & De Groot, B. L. Normal modes and essential dynamics. In *Methods in Molecular Biology*, vol. 443, 89–106 (2008).

5. Tirion, M. M. Large Amplitude Elastic Motions in Proteins from a Single-Parameter, Atomic Analysis. *Physical Review Letters* **77**, 1905–1908 (1996).

6. Amadei, a. & Limddrn, A. Essential dynamics of Proteins. *Proteins: Structure, Function, and Genetics* **17**, 412–425 (1993).

7. Gargallo, R., Hünenberger, P. H., Avilés, F. X. & Oliva, B. Molecular dynamics simulation of highly charged proteins: comparison of the particle-particle particle-mesh and reaction field methods for the calculation of electrostatic interactions. *Protein science : a publication of the Protein Society* **12**, 2161–2172 (2003).

8. van Aalten, D. M., Grotewold, E. & Joshua-Tor, L. Essential dynamics from NMR clusters: dynamic properties of the Myb DNA-binding domain and a hinge-bending enhancing variant. *Methods (San Diego, Calif.)* **14**, 318–28 (1998).

9. Ogata, K. *et al.* Solution structure of a specific DNA complex of the Myb DNA-binding domain with cooperative recognition helices. *Cell* **79**, 639–648 (1994).

10. Schrödinger, LLC. The PyMOL Molecular Graphics Plugin, Version 1.8 (2015).

11. Tomomori, C. *et al.* Solution structure of the homodimeric core domain of Escherichia coli histidine kinase EnvZ. *Nature structural biology* **6**, 729–34 (1999).