NMA-Task Documentation

Michael Glenister

Jul 05, 2018

Getting started

CHAPTER 1

Introduction

MODE-TASK is a collection of tools for analysing normal modes and performing principal component analysis on biological assemblies.

Novel coarse graining techniques allow for analysis of very large assemblies without the need for high performance computing clusters and workstations.

1.1 PyMOL Plugin

A PyMOL plugin for MODE-TASK has been made available here.

1.2 Cite this project

Caroline Ross, Bilal Nizami, Michael Glenister, Olivier Sheik Amamuddy, Ali Rana Atilgan, Canan Atilgan, Özlem Tastan Bishop; MODE-TASK: Large-scale protein motion tools, Bioinformatics, May 2018 , https://doi.org/10.1093/bioinformatics/bty427

1.3 Contributing

To contribute to the documentation please follow this guide.

To contribute to the source code, submit a pull request to our Github repository.

1.4 License

The project is licensed under GNU GPL 3.0

CHAPTER 2

Installation

2.1 Platform compatibility

A Linux-like operating system is recommended. However, MODE-TASK is compatible with most platforms which are able to run Python 2.7 or Python 3.6.

A compiler is required to compile C++ from source, in this instance we use g++.

2.2 Download the project

MODE-TASK can be cloned from its GitHub repository

```
git clone https://github.com/RUBi-ZA/MODE-TASK.git
cd MODE-TASK
```

2.3 Installing dependencies

Ubuntu:

Python 2.7 with pip and virtualenv

```
sudo apt-get update
sudo apt-get install python-pip virtualenv virtualenvwrapper
source venv/bin/activate
pip install -r requirements.txt
```

Python 3.6 with pip and virtualenv

```
sudo apt-get update
sudo apt-get install python3-pip virtualenv virtualenvwrapper
source venv/bin/activate
pip3 install -r requirements.txt
```

Conda

```
conda create -n mode_task
source activate mode_task
conda install -c conda-forge numpy
conda install -c conda-forge cython
conda install -c omnia mdtraj
conda install -c conda-forge scipy
conda install -c conda-forge pandas
conda install -c conda-forge sklearn-contrib-lightning
conda install -c conda-forge matplotlib
```

To install conda follow their documentation

OSX:

```
brew update
brew install python gcc
pip install virtualenv virtualenvwrapper
```

Windows:

Enable Windows Subsystem for Linux (WSL) by following these instructions.

Install the system dependencies as with Ubuntu above.

CHAPTER 3

Normal Mode Analysis (NMA)

NMA analyses the oscillations of a structure. For proteins, it is useful for studying the large amplitude motions for a selected conformation. The main assumption is that the motions are harmonic. Thus, each normal mode, which is a concerted motion of many atoms, acts as a simple harmonic oscillator, and it is independent of all the other normal modes.

For a harmonic oscillator with a mass m supported on a spring with force constant k, the potential energy of the system, $V = kx^2$, for an extension x leads to the restoring force,

$$\mathbf{F} = -\frac{dV}{dx} = -kx$$

By substituting this Hooke's Law force into Newton's Law, $\mathbf{F} = m\mathbf{a}$ leads to the differential equation,

$$-kx = m\frac{d^2x}{dt^2}$$

The solution is,

$$-kx = -4\pi^2 v^2 mx$$

with v being the frequency of the vibration.

In three dimensions and for a set of N atoms, one has the corresponding generalized equation,

$$-\mathbf{HX} = -4\pi^2 v^2 \mathbf{X}$$

where $\mathbf{H}$ is the $3N$x$3N$ Hessian symmetric matrix of force constants and $\mathbf{X}$ is the $3N$x1 vector of the positions of the atoms. The solutions of the equation may be cast in the form of an eigenvalue decomposition of $\mathbf{H}$ where the eigenvectors are the mass weighted displacements of the normal coordinates; i.e. the independent vibrational motions of the collection of atoms. The corresponding eigenvalues are the negative of the squared normal mode frequencies of each mode. $\mathbf{H}$ has exactly six zero eigenvalues for the translations and rotations of the molecule in three dimensional space.

In NMA of proteins, it is central to obtain a good representation of the protein for a proper analysis of the available modes of motion. One approach would be to obtain the Hessian matrix from the second derivatives of the energy for the conformation of interest following a very stringent minimization of the molecule. The latter is important as NMA

is based on the harmonic assumption which is only a valid approximation at the bottom of the potential energy minima. For the complex potentials representing interactions in proteins in the water environment, these second derivatives may be obtained through numerical methods which are extremely costly.

Alternatively, $\mathbf{H}$ may be approximated by the variance-covariance matrix of the protein, obtained from a molecular dynamics (MD) simulation of suitable length. This is relevant as it may be shown through employing statistical mechanics that the average displacements of nodes from a mean structure for an atom $i, \Delta\mathbf{R}_i$ are related to those of all other atoms through the relationship,

$$< \Delta\mathbf{R}_i \cdot \Delta\mathbf{R}_j >= 3k_BT\mathbf{H}^{-1}$$

It is important to select the length of the MD simulation for this procedure such that the molecule samples only the state of interest,[1] since when more than a single potential energy well along the conformational space of the molecule is sampled, the harmonic assumption would again fail.

As a third alternative for obtaining the Hessian, one may make use of the elastic network property of a folded protein. Here, the assumption is that, once a protein folds to a functionally relevant conformation, its total energy is represented by a simple harmonic potential such that residues are connected to their nearest neighbors via elastic springs. By further employing the assumption that the spring constants, γ, are equivalent in all pairs of interactions, one arrives at the anisotropic network model (ANM).[2]

$\mathbf{H}$ is thus composed of NxN super-elements, i.e.,

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{1,1} & \mathbf{H}_{1,2} & \cdots & \mathbf{H}_{1,\mathbf{N}} \\ \mathbf{H}_{2,1} & & & \mathbf{H}_{2,\mathbf{N}} \\ \vdots & & & \vdots \\ \mathbf{H}_{\mathbf{N},1} & & & \mathbf{H}_{\mathbf{N},\mathbf{N}} \end{bmatrix}$$

where each super-element $\mathbf{H_{ij}}$ is a 3x3 matrix that holds the anisotropic information regarding the orientation of nodes i, j:

$$\mathbf{H_{ij}} = \begin{bmatrix} \partial^2V/\partial X_i\partial X_j & \partial^2V/\partial X_i\partial Y_j & \partial^2V/\partial X_i\partial Z_j \\ \partial^2V/\partial Y_i\partial X_j & \partial^2V/\partial Y_i\partial Y_j & \partial^2V/\partial Y_i\partial Z_j \\ \partial^2V/\partial Z_i\partial X_j & \partial^2V/\partial Z_i\partial Y_j & \partial^2V/\partial Z_i\partial Z_j \end{bmatrix}$$

Denoting the separation between nodes i and j by S_{ij}, the elements of the off-diagonal super-elements $\mathbf{H_{ij}}$ are given by:

$$\partial^2V/\partial X_i\partial Y_j = -\gamma(X_j - X_i)(Y_j - Y_i)/S_{ij}^2$$

and those of the diagonal super-elements $\mathbf{H_{ij}}$ are obtained via,

$$\partial^2V/\partial X_i^2 = \gamma\sum_j(X_j - X_i)^2/S_{ij}^2 \quad \text{for the diagonal terms}$$

$$\partial^2V/\partial X_i\partial Y_j = \gamma\sum_j(X_j - X_i)(Y_j - Y_i)/S_{ij}^2 \quad \text{for the off-diagonal terms.}$$

In this representation of the protein, the structure is coarse-grained at the level of a residue, usually through the coordinates of $\mathbf{C}_\alpha$ or $\mathbf{C}_\beta$ atoms obtained from the protein data bank. Moreover, pairs of nodes are assumed to interact if they are within a pre-selected cut-off distance, r_c. For large structures such as viruses, further coarse graining may be shown to well describe the most collective modes of motion.

The selection of r_c has been the cause of much research. While it is clear that there is a lower bound where the condition of six zero eigenvalues of $\mathbf{H}$ should be satisfied, distances in the range 10-25 Å have been employed in the

[1] C Atilgan, OB Okan, AR Atilgan, "Network-based Models as Tools Hinting at Non-evident Protein Functionality," Annual Review of Biophysics, 41, 205-225 (2012).

[2] AR Atilgan, SR Durell, RL Jernigan, MC Demirel, O Keskin, I Bahar, "Anisotropy of Fluctuation Dynamics of Proteins with an Elastic Network Model," Biophysical Journal, 80, 505-515.

literature. It has been shown by a systematic study of increasing r_c that the collective modes of motion do not change beyond a certain value for proteins; i.e. selection of too large r_c is safer than a too small value.[3] The reason for this has been explained by expressing $\mathbf{H}$ as the sum of an essential and a trivial part. The essential part of $\mathbf{H}$ includes all the local information necessary for the correct representation of the modes. On the other hand, due to the symmetries in a protein originating from the orientational order of closely packed spheres, the effects from the long range neighbors cancel out.

References

[3] C Atilgan, OB Okan, AR Atilgan, "Orientational Order Governs Collectivity of Folded Proteins," Proteins: Structure, Function, Bioinformatics, 78, 3363-3375 (2010).

CHAPTER 4

Principal Component Analysis (PCA)

A molecular dynamics (MD) simulation of a protein provides the positional movements of each atom with respect to a fixed reference frame at a given time. The mean squared positional fluctuations (variances) of each atom are readily calculated once the total simulation and sampling times are set. Sufficiency of both total observation period and the sampling rate are crucial in collecting the data so as to identify biologically relevant motions. Let us monitor the variance of each residue's $\mathbf{C}_\alpha$ or $\mathbf{C}_\beta$ atom during a MD simulation of a protein. Suppose that these variances do not change significantly in time, like a stationary process. This suggests that within the period of observation we have recorded the motion about one (native) conformation. Though constant in time for a given residue, the variances do change from one residue to another. It is important to distinguish the differences between the variances of different parts of the protein and to explain the root cause of these differences; e.g. while loop or disordered regions exhibit high values, relatively rigid parts, such as helices or sheets display lower variances.

PCA[4] operates on the variance-covariance matrix, $\mathbf{C}$, of the protein, obtained from a MD simulation of any length; thus, the observed process need not be stationary. It is useful in distinguishing the different parts of the energy landscape sampled during the MD simulation. To obtain $\mathbf{C}$, first the protein coordinates are superimposed on a reference structure, usually the initial coordinates, or the average coordinates. The displacement vector for each residue (described by the $\mathbf{C}_\alpha$ or $\mathbf{C}_\beta$ coordinates of the residue i) at a time point t, $\Delta\mathbf{R}_i(t)$ is obtained. For a set of M recorded coordinates, these are organized in the trajectory fluctuation matrix of order $3N\text{x}M$:

$$\Delta\mathbf{R} = \begin{bmatrix} \Delta\mathbf{R}_1(t_1) & \Delta\mathbf{R}_1(t_2) & \cdot & \Delta\mathbf{R}_1(t_M) \\ \Delta\mathbf{R}_2(t_1) & \Delta\mathbf{R}_2(t_2) & \cdot & \Delta\mathbf{R}_2(t_M) \\ \Delta\mathbf{R}_3(t_1) & \Delta\mathbf{R}_3(t_2) & \cdot & \Delta\mathbf{R}_3(t_M) \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \Delta\mathbf{R}_n(t_1) & \Delta\mathbf{R}_n(t_2) & & \Delta\mathbf{R}_n(t_M) \end{bmatrix}$$

The $3N\text{x}3N$ $\mathbf{C}$ matrix is then obtained via the operation,

$$\mathbf{C} = \Delta\mathbf{R}\Delta\mathbf{R}^{\mathbf{T}}$$

If a single energy well along the potential energy surface of a protein is sampled, then $\mathbf{C}$ approximates the inverse Hessian, $\mathbf{H}^{-1}$, as the harmonic approximation applies in this case (see NMA for details). However, if different parts

[4] A Amadei, ABM Linssen, HJC Berendsen, "Essential Dynamics of Proteins," Proteins: Structure, Function and Genetics, 17, 412-425 (1993).

of the landscape are sampled, the decomposition of $\mathbf{C}$ will carry information on all the regions entered during the simulation. Thus, the diagonalization,

$$\mathbf{C} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathbf{T}}$$

yields the eigenvectors and the corresponding eigenvalues of the $\mathbf{C}$ matrix. $\mathbf{\Lambda}$ is the $3N$x$3N$ diagonal matrix holding the eigenvalues λ_i with six zero values corresponding to the translations and rotations of the molecule. The i^{th} row of the $\mathbf{U}$ matrix holding the eigenvector corresponding to the i^{th} eigenvalue. The trajectory $\Delta\mathbf{R}$ may be projected onto the eigenvectors to obtain the principal components, q_i, which are the rows of the $3N$xM $\mathbf{Q}$ matrix.

$$\mathbf{Q} = \mathbf{U}\Delta\mathbf{R}$$

Since a few principal components usually carry the largest amount of information of the trajectory, the different regions of the conformational space will manifest as more than one blob in a plot of q_i versus q_j where i and j are small. Furthermore, the size of the blobs in the plots will provide information on the width of the potential wells sampled. Finally, the time points when passage between different wells occur may be pinpointed by this method. The different implementations of the construction of the $\mathbf{C}$ matrix and the various ways of decomposing it have been discussed in detail in the literature,[5] and implemented in MODE-TASK.

References

[5] CC David, DJ Jacobs, "Principal component analysis: a method for determining the essential dynamics of proteins," Methods in Molecular Biology, 1084, 193-226 (2014).

CHAPTER 5

Kernel PCA (kPCA)

Standard PCA assumes that input data are linearly related. In cases where variables are not intrinsically linearly related the user has option to perform nonlinear generalization of PCA such as Kernel PCA. It is an extention of normal PCA, where different kernel fucntions (such as linear, RBF, polynomial, and cosine) are used to perform non-linear dimensional reduction.

N points are linearly non-separable in dimension ($d < N$). But there can be a hyperplane dividing them in a higher dimension given N points, (Xi) it can be maped to an N-dimensional space with

$$\Phi(\mathbf{Xi}) \text{ where } \Phi : \mathbf{R^d} -> \mathbf{R^n}$$

Φ is an arbitrary choosen fucntion.

MODE-TASK includes the tools to perform the Kernel PCA on MD trajectory, and offers the user different choices for the kernel. In Kernel PCA the input trajectory is first raised to a higher dimension by a kernel function and then PCA is performed on the elevated data. One should use Kernel PCA with caution as it is difficult to interpret the results since the input trajectory is mapped to a different feature space than conformational space. Nevertheless, Kernel PCA could be an invaluable tool in studying structural mechanisms behind protein dynamics in cases where conventional PCA is not helpful.

CHAPTER 6

Incremental PCA

A major bottleneck in the speed of PCA calculation is the availability of computer memory during the loading of a MD trajectory. IPCA is a memory efficient variant of PCA, where only most substantial singular vectors are used to project the input data to a lower dimension. The IPCA algorithm uses a batch data loading approach and the incremental storage of various variables, thus achieving higher memory efficiency. MODE-TASK has implemented IPCA on MD trajectory, available through scikit-learn Python library on the original algorithm[6][7].

References

[6] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. J. Mach. Learn. Res. 2011;12:2825–30.

[7] Ross DA, Lim J, Lin R-S, Yang M-H. Incremental Learning for Robust Visual Tracking. Int. J. Comput. Vis. 2008;77:125–41.

CHAPTER 7

Multi-dimensional scaling (MDS)

MDS is a technique of dimensionality reduction, where measure of dissimilarity in a dataset is used. It places each input point in N-dimensional space while trying to preserve the original distance matrix as much as possible. MODE-TASK implements metric and nonmetric types of MDS for MD trajectory by using the scikit-learn library[6]. The Euclidean distance between internal coordinates and pairwise RMSD between the MD frames is used as dissimilarity measures in MODE-TASK.

CHAPTER 8

t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is another dimensionality reduction method for data of high dimensions[8]. t-SNE has been implemented for protein MD trajectories in MODE-TASK. Like MDS, the Euclidean distance between internal coordinates of a protein structure and pairwise RMSD between a set of atoms are used as measures of dissimilarity.

References

[8] Van der Maaten L, Hinton G. Visualizing Data using t-SNE. J. Mach. Learn. Res. 2008;9:2579–605.

CHAPTER 9

NMA Scripts

9.1 Coarse grain

Takes a protein structure or biological assembly and coarse grains to select a set amount of CB atoms **Command:**

```
coarseGrain.py <options> --pdbFile <pdb file> --atomType <string>
```

Inputs:

Input (**required*)	Input type	Flag	Description
PDB file *	File	`--pdb`	PDB structure to coarse grain. Can also accept biological assembly (.pdb1)
Atom type *	String	`--atomType`	Specify the type of atom to be selected in CG models. Only CA or CB accepted.
Coarse grain level	Comma Separted String	`--cg`	Level/Levels by which to coarse grain protein. Lower is less coarse grained **E.g –cg 4** OR E.g –cg 1,3,5 Default: 4
Starting atom	Integer	`--startingAtom`	Residue number of the starting atom. Default: 1
Output file	File	`--output`	Specify a name for the PDB output file. Default: ComplexCG.pdb

Outputs:

Output	Description
PDB file/s	Coarse grained protein/s or macromolecule/s

9.2 ANM

Constructs an elastic network model of a protein complex and solves for the eigenvalues and eigenvectors of the system.

Compile:

```
g++ -I cpp/src/ ANM.cpp -o ANM
```

Command:

```
ANM <options> --pdb <pdb file> --atomType <atom type>
```

Inputs:

Input (*required*)	Input type	Flag	Description
PDB file *	File	`--pdb`	PDB input file
Atom type *	String	`--atomType`	Specify the type of atom to be selected in CG models. Only CA or CB accepted.
Cutoff	Integer	`--cutoff`	Cuttoff radius in Å. Default: 15

Outputs:

Output	Description
W matrix	Text file of $3N$ eigenvalues
VT matrix	Text file of $3N$x$3N$ eigenvectors. Printed in rows
U matrix	Text file of $3N$x$3N$ eigenvectors. Printed in columns

9.3 Mean square fluctuation

Calculates and returns the diagonals of the correlation matrix for a given set of modes.

The user can also compare the msf between two protein complexes. Let's say that the user has performed NMA on two coarse grained models of the same protein, and now wants to compare if the additional coarse graining decreased the accuracy. If we obtain the same mean square fluctuations for each residue, then in each model we can say that the results are comparable regardless of the coarse graining level. Obviously, we must compare only the residues that are common in each model. Hence, we specify common residues.

Command:

```
meanSquareFluctuations.py <options> --pdb <PDB file> --wMatrix <text file> --vtMatrix
↪<text file> --atomType <string>
```

Inputs:

Input (**required*)	Input type	Flag	Description
PDB file *	File	`--pdb`	PDB input file
W matrix file *	File	`--wMatrix`	W values from ANM script for PDB
VT matrix file *	File	`--vtMatrix`	VT values from ANM script for Comparison PDB
Atom type *	String	`--atomType`	Specify the type of atom to be selected in CG models. Only CA or CB accepted.
Comparison PDB	File	`--pdbC`	When assigned, calculates mean square fluctuations based on common residues between the two systems.
W matrix file for pdbC	File	`--wMatrixC`	When assigned, calculates W values from ANM for Comparison PDB
VT matrix file for pdbC	File	`--vtMatrixC`	When assigned, calculates VT values from ANM for Comparison PDB
Selected modes	String OR Colon Separated String OR Comma Separated String	`--modes`	MSFs will be calculated over specified modes. Options: 1) Single mode E.g –modes 7; 2) A range E.g –modes 7:20; 3) A list E.g –modes 8,9,11 If unspecified MSFs will be calculated for the first twenty slowest modes (7:27)

Outputs:

Output	Description
The following are generated for the PDB and Comparison PDB (if pdbC was assigned)	
MSF text file	MSF for all residues, calculated over all modes
MSF modes text file	MSF for all residues, calculated for a specific mode range
Common residue MSF text file	MSF for all common residues, calculated over all modes
Common residue MSF modes text file	MSF for all common residues, calculated over a specific mode range

9.4 Assembly Covariance

Calculates and plots Covariance matrices

The user can compare the Covariance between different regions in the biological assembly, or can calculate the Covariance across the full assembly complex. The user also has the option to perform the calculation over a specified list of modes or a mode range. The function also has a zoom option that allows the user to create a Covariance plot for a particular chain within a particular asymmetric unit.

Command:

```
assemblyCovariance.py <options> --pdb <PDB file> --wMatrix <text file> --vtMatrix
↪<text file> --atomType <string>
```

Inputs:

Input (**required*)	Input type	Flag	Description
PDB file *	File	`--pdb`	PDB input file
W matrix file *	File	`--wMatrix`	W values from ANM script for PDB
VT matrix file *	File	`--vtMatrix`	VT values from ANM script for Comparison PDB
Atom type *	String	`--atomType`	Specify the type of atom to be selected in CG models. Only CA or CB accepted.
Selected modes	String OR Colon Separated String OR Comma Separated String	`--modes`	Covariance will be calculated over specified modes Options: 1) All modes E.g –modes all; 2) Single mode E.g –modes 7; 3) A range E.g –modes 7:20; 4) A list E.g –modes 8,9,11 If unspecified, Covariance will be calculated for all modes.
Asymmetric Units	String OR Comma Separated String	`--aUnits`	Covariance will be calculated and plotted for specified asymmetric units Options: 1) Single unit E.g –aUnits 5; 2. A list of units E.g –aUnits 1,3 If unspecified, Covariance will be calculated for the first asymmetric unit in the assembly.
Zoom	Comma Separated String	`--zoom`	If specified, Covariance will be calculated and plotted for a specified chain in a specified unit. Only format accepts is: [Unit,Chain] **E.g –zoom 1,2** OR E.g –zoom 1,B **(Chain specifier must match** chain label in PDB file) The above calculates the covariance for the second chain in the first asymmetric unit.

Outputs:

Output	Description
Covariance Plots	Covariance Matrices plotted as a Linear Segmented Color map
Matrix text files	Covariance Matrices printed in .txt format

9.5 Conformation mode

Identifies modes responsible for the conformational change of a molecule.

Command:

```
conformationMode.py <options> --pdbConf <PDB file> --pdbANM <PDB file> --vtMatrix
↪<text file> --atomType <string>
```

Inputs:

Input (*required*)	Input type	Flag	Description
Unaligned PDB file *	File	`--pdbConf`	PDB file of the conformational change
PDB *	File	`--pdbANM`	PDB file that was used to run ANM
VT matrix file *	File	`--vtMatrix`	Eigenvetors obtained from ANM script
Atom type *	String	`--atomType`	Specify the type of atom to be selected in CG models. Only CA or CB accepted.
Output file	File	`--output`	Specify a name for the output file. Default: ModesOfConfChange.txt

Outputs:

Output	Description
Conformation file	Text file with the overlap and correlation of each mode

9.6 Combination mode

Calculates the combined overlap and correlation for specified set of modes to a known conformational change. This script also calculates the overlap and correlation per chain in each asymmetric unit for the specified modes. This allows the user to determine which parts of the complex, in each mode, contribute the most to the overall conformational change.

Command:

```
combinationMode.py <options> --pdbConf <PDB file> --pdbANM <PDB file> --vtMatrix
↪<text file> --modes <comma separated string> --atomType <string>
```

Inputs:

Input (*required*)	Input type	Flag	Description
Unaligned PDB file *	File	`--pdbConf`	PDB file of the conformational change
PDB *	File	`--pdbANM`	PDB file that was used to run ANM
VT matrix file *	File	`--vtMatrix`	Eigenvetors obtained from ANM script
Modes *	Integer	`--modes`	Calculate the overlap for a combination of specific modes. Numbers are separated by commas: 1,5,7
Atom type *	String	`--atomType`	Specify the type of atom to be selected in CG models. Only CA or CB accepted.
Output file	File	`--output`	Specify a name for the output file. Default: ModeSpecificConfChange.txt

Outputs:

Output	Description
Combination file	Text file with the overlap and correlation of each mode as well as the combined overlap and correlation for the modes specified
Break down per unit file	Text file with the overlap and correlation calculated for each chain in each asymmetric unit in the complex. Calculations are performed for each specified mode.

9.7 Mode visualisation

Generates a set of frames, where eigenvectors are plotted as a set of unit vectors multiplied by an increasing factor in each frame. Vectors are also plotted as arrows that can be viewed in the tool VMD

Command:

```
visualiseVector.py <options> --pdb <PDB file> --vtMatrix <text file> --mode <int> --
↪atomType <string> --direction <int>
```

Inputs:

Input (**required*)	Input type	Flag	Description
Coarse grained PDB file *	File	`--pdb`	Coarse grained PDB input file
Mode index value *	Ingeter	`--mode`	Value specifying the index of the mode
VT matrix file *	File	`--vtMatrix`	VT values from ANM script
Atom type *	String	`--atomType`	Specify the type of atom to be selected in CG models. Only CA or CB accepted.
Direction	Boolean integer (1 or -1)	`--direction`	Direction of overlap correction. Default = 1
Arrow head	float	`--head`	Radius of cone that forms the head of each vector arrow
Arrow tail	float	`--tail`	Radius of cylinder that forms the tail of each vector arrow
Arrow length	float	`--arrowLength`	Specify a factor by which to increase or decrease the length of each arrow E.g –arrowLength 2 doubles the default length and –arrowLength 0.5 halves the default length
Colours	Comma Separted String	`--colourByChain`	Colour the vectors arrows of each chain. E.g for a two chain protein –colourByChain blue,red will colour the arrows of Chain A as blue and Chain B as red
Asymmetric Units	String OR Comma Separated String	`--aUnits`	Vector frames and arrows will be plotted for specified asymmetric units Options: 1) Single unit E.g –aUnits 5 2. A list of units E.g –aUnits 1,3
Chain	String	`--chain`	Draws arrows only for the specified chain. This option only accepts a single chain

Outputs:

Outputs are generated in output/VISUALISE directory by default.

Output	Description
PDB file	Output PDB to be opened in VMD
Arrows file	Tcl script that can be copied into the VMD TK console

CHAPTER 10

PCA Scripts

Principal component analysis (PCA) is a useful statistical technique that has found applications in detection of correlated motion in MD data. Protein dynamics is manifested as a change in molecular structure, or conformation over a timescale. PCA extracts most important motions from a protein's MD trajectory using a covariance/correlation matrix (C-matrix) constructed from atomic coordinates. Different types of coordinate systems (Cartesian or internal coordinates) can be employed to define atomic movement in each time frame of a trajectory. Modes describing the protein motions can be constructed by diagonalizing the C-matrix. It leads to a complete set of orthonormal (orthogonal and normalized) collective modes (eigenvectors) with eigenvalues (variance) that characterize the protein dynamics. The largest eigenvalues represent the most collective spatial motion. When the original mean centered data (MD trajectory) is projected on the eigenvectors, the results are called Principal Components (PC). Diagonalization of the C-matrix can be done by Eigenvalue decomposition (EVD) or Singular value decomposition (SVD), with the latter being computationally efficient.

As stated earlier, different representations of protein conformations can be used. One can choose Cartesian coordinates or internal coordinates such as the pairwise distance between atoms, 1-3 angle, torsional angles (Φ or Ψ). Since decomposition of a C-matrix is memory intensive and very often the program will run out of memory, often a coarse graining is required such as selecting CA atoms. The user can select the subset of atoms from the trajectory for the analysis such as CA, backbone atoms or all protein's atoms. It is highly recommended that the user should strip the water from the trajectory before hand, as it would result in faster loading and alleviate the memory issues.

PCA uses linear transformation which may not be sufficient in cases where variables are non-linearly related. Thus, the user has the option to perform Nonlinear generalization of PCA such as Kernel PCA (kPCA). Caution should be given while interpreting the kPCA results since it is mapped to a feature space which is inherently different than conformational space. Nevertheless, kPCA is useful in understanding the protein's functions in terms of its conformational dynamics.

General Usage:

To perform PCA on a protein's MD trajectory we need a sufficiently sampled MD trajectory and a corresponding topology file. This can be achieved by running the following command.

Command: `pca.py -t <MD trajectory> -p <topology file>`

To see the all the available options run the following command: `pca.py -h`

Inputs:

Input (**required*)	Input type	Flag	Description
Trajectory file *	File	`-t`	MD trajectory input file (.xtc, .mdcrd etc.)
Topology file *	File	`-p`	Topology file (.gro, .pdb etc)
Output directory	String	`-out`	Name of the output directory. Default is out, suffixed by trajectory name
Atom group	String	`-ag`	Group of atoms for PCA. Default is CA atoms. Other options are: all = all atoms, backbone = backbone atoms, CA = C-alpha atoms, protein = protein atoms
Reference structure	File	`-r`	Reference structure for RMSD. Default: First frame of MD trajectory
PCA method	String	`-pt`	PCA method. Default is svd (Single Value Decomposition) PCA. Options are: evd, kpca, svd, ipca. If svd is selected, additional arguments can be passed by flag -st. If KernelPCA is selected kernel type can also be defined by flag -k
Number of components	Int	`-nc`	Number of components to keep in a PCA object. Default: All the components will be kept.
Kernel Type	String	`-kt`	Type of kernel for KernalPCA. Default is linear. Options are: linear, poly, rbf, sigmoid, cosine, precomputed
SVD solver type	String	`-st`	Type of svd_solver for SVD (Single Value Decomposition) PCA. Default is auto. Options are: auto, full, arpack, randomized

Outputs:

Output	Description
PC plots	2D Plot of first 3 PCs. It is grace formatted text file
PC plots (.png)	2D Plot of first 3 PCs. Same as above, but points are color coded according to MD time
Scree plot	Scree plot of contribution of first 100 modes (eigenvectors)
RMSD plot	RMSD of selected atoms over the MD time
RMSD Modes	Plot of contribution of each resdiues towards the first 3 modes (eigenvectors)

Besides the above-mentioned plots, it also prints useful information on the terminal such as, information about the trajectory, Kaiser-Meyer-Olkein (KMO) index of the trajectory, and cosine contents of the first few PCs. KMO value range from 0 to 1, 1 indicating that the MD has been sampled sufficiently. The cosine content of PCA projections can be used as an indicator if a simulation is converged. Squared cosine value should be more than 0.5.

Specific Examples:

10.1 PCA on Cartesian coordinates

Given a trajectory called `trajectory.xtc` and a topology file called `complex.pdb`, the following command is used:

```
pca.py -t trajectory.xtc -p complex.pdb
```

This will perform the singular value decomposition (SVD) based PCA on CA atoms by default. To use other methods, see the following examples.

10.1.1 SVD PCA

To perform SVD PCA on CA atoms of a MD trajectory

Command: `pca.py -t trajectory.xtc -p complex.pdb -ag CA -pt svd`

To perform the SVD PCA on backbone atoms

Command: `pca.py -t trajectory.xtc -p complex.pdb -ag backbone -pt svd`

10.1.2 Kernel PCA

To perform the Kernel PCA with linear kernel

Command: `pca.py -t trajectory.xtc -p complex.pdb -ag CA -pt kpca -kt linear`

To perform the Kernel PCA with rbf kernel

Command: `pca.py -t trajectory.xtc -p complex.pdb -ag CA -pt kpca -kt rbf`

10.1.3 Incremental PCA

Incremental PCA (IPCA) is a variant of usual PCA, which uses low-rank approximation of the input MD trajectory. It uses the amount of memory to store the input trajectory which is independent of trajectory size. IPCA is very useful in case the size of trajectory is larger than that may be handled by the available computer memory.

Command: `pca.py -t trajectory.xtc -p complex.pdb -ag CA -pt ipca`

10.1.4 Eigenvalue decomposition (EVD) PCA

To perform the PCA by eigenvalue decomposition

Command: `pca.py -t trajectory.xtc -p complex.pdb -ag CA -pt evd`

Detailed usage:

Run the following command to see the detailed usage and other options: `pca.py -h`

10.2 PCA on internal coordinates

Users can also perform the PCA on internal coordinates of a MD trajectory. Options are available for different types of internal coordinates such as: *pairwise distance between atoms*, *1-3 angle between backbone atoms*, *psi angle*, and *phi angle*.

General Usage:

Command: `internal_pca.py -t <MD trajectory> -p <topology file>`

Inputs:

Input (**required*)	Input type	Flag	Description
Trajectory file *	File	`-t`	MD trajectory input file (.xtc, .mdcrd, etc.)
Topology file *	File	`-p`	Topology file (.gro, .pdb, etc)
Output directory	String	`-out`	Name of the output directory. Default is out, suffixed by trajectory name
Atom group	String	`-ag`	Group of atom for PCA. Default is CA atoms. Other options are: all = all atoms, backbone = backbone atoms, CA = C-alpha atoms, protein = protein atoms
Coordinate Type	String	`-ct`	Internal cordinate type. Options are: distance, angles, phi, and psi

Outputs:

Output	Description
PC plots	2D Plot of first 3 PCs. It is a grace formatted text file
PC plots (.png)	2D Plot of first 3 PCs. Same as above, but points are color coded according to MD time
Scree plot	Scree plot of the contribution of the first 100 modes (eigenvectors)

Specific Examples:

PCA on pairwise distance between CA atoms:

To perform the PCA on pairwise distance between CA atoms of an MD trajectory `trajectory.xtc` and a topology file `complex.pdb`

Command: `internal_pca.py -t trajectory.xtc -p complex.pdb -ag CA -ct distance`

PCA on psi angles:

Command: `internal_pca.py -t trajectory.xtc -p complex.pdb -ct psi`

Detailed usage:

Run the following command to see the detailed usage and other options: `internal_pca.py -h`

10.3 MDS (Multi-dimensional scaling) on MD trajectory

MDS is a tool to visualize the similarity or dissimilarity in a dataset. Two types of dissimilarity measures can be used in the case of a MD trajectory. The first is Euclidean distance between internal coordinates of a protein structure, the second is pairwise RMSD between a set of atoms over the frames of a MD trajectory.

General Usage:

command: `mds.py -t <MD trajectory> -p <topology file>`

Inputs:

Input (**required*)	Input type	Flag	Description
Trajectory file *	File	`-t`	MD trajectory input file (.xtc, .mdcrd, etc.)
Topology file *	File	`-p`	Topology file (.gro, .pdb, etc)
Output directory	String	`-out`	Name of the output directory. Default is out, suffixed by trajectory name
Atom group	String	`-ag`	Group of atoms for MDS. Default is CA atoms. Other options are: all = all atoms, backbone = backbone atoms, CA = C-alpha atoms, protein = protein atoms
MDS type	String	`-mt`	Type of MDS. Options are nm = non-metric, metric = metric
Dissimilarity type	String	`-dt`	Type of dissimilarity matrix to use. euc = Euclidean distance between internal coordinates, rmsd = pairwise RMSD. Default is rmsd
Coordinate type	String	`-ct`	Internal coordinate type. Default is pairwise distance. Only used if Dissimilarity type is Euclidean
Atom indices	String	`-ai`	Group of atoms for pairwise distance. Default is CA atoms. Other options are: all = all atoms,backbone = backbone atoms, alpha = C-alpha atoms,heavy = all non-hydrogen atoms, minimal = CA, CB, C, N, O atoms

Outputs:

Output	Description
PC plots	2D Plot of the first 3 PCs. It is a grace formatted text file
PC plots (.png)	2D Plot of the first 3 PCs. Same as above, but points are color coded according to MD time

Specific Examples:

MDS on pairwise RMSD:

To perform MDS on the pairwise RMSD between CA atoms

Command: `mds.py -t trajectory.xtc -p complex.pdb -dt rmsd -ag CA`

MDS on internal coordinates:

To perform MDS on the pairwise distance between CA atoms

Command: `mds.py -t trajectory.xtc -p complex.pdb -dt euc -ag CA`

Detailed usage:

Run the following command to see the detailed usage and other options: `mds.py -h`

10.4 t-SNE on MD trajectory

t-distributed Stochastic Neighbor Embedding (t-SNE) is a tool for dimensionality reduction. It is a variant of stochastic neighbor embedding technique. t-SNE uses a measure of dissimilarity, which, in the case of MD trajectory, may be

the Euclidean distance between internal coordinates or pairwise RMSD.

General Usage:

Command: `tsne.py -t <MD trajectory> -p <topology file>`

Inputs:

Input (**required*)	Input type	Flag	Description
Trajectory file *	File	`-t`	MD trajectory input file (.xtc, .mdcrd, etc.)
Topology file *	File	`-p`	Topology file (.gro, .pdb, etc)
Output directory	String	`-out`	Name of the output directory. Default is out, suffixed by trajectory name
Atom group	String	`-ag`	Group of atoms for t-SNE. Default is CA atoms. Other options are: all = all atoms, backbone = backbone atoms, CA = C-alpha atoms, protein = protein atoms
Coordinate type	String	`-ct`	Internal coordinates type. Default is pairwise distance . Only used if Dissimilarity type is Euclidean
Dissimilarity type	String	`-dt`	Type of dissimilarity matrix to use. euc = Euclidean distance between internal coordinates, rmsd = pairwise RMSD. Default is rmsd
Atom indices	String	`-ai`	Group of atoms for pairwise distance. Default is CA atoms. Other options are: all = all atoms, backbone = backbone atoms, alpha = C-alpha atoms, heavy = all non-hydrogen atoms, minimal = CA, CB, C, N, O atoms
PERPLEXITY	Float	`-pr`	[t-SNE parameters] The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms Default is 30
LEARNING_RATE	Float	`-lr`	[t-SNE parameters] The learning rate for t-SNE. Default is 200
N_ITER	Int	`-ni`	[t-SNE parameters] Number of iteration to run. Default is 300

Outputs:

Output	Description
PC plots	2D Plot of the first 3 PCs. It is grace formatted text file
PC plots (.png)	2D Plot of the first 3 PCs. Same as above, but point are color coded according to MD time

Specific Example:

t-SNE on CA atoms: To perform t-SNE using the pairwise RMSD of CA atoms as index of dissimilarity.

command: `tsne.py -t trajectory.xtc -p complex.pdb -ag CA -dt rmsd`

To perform t-SNE using the Euclidean space between pairwise distance of CA atoms as index of dissimilarity.

command: `tsne.py -t trajectory.xtc -p complex.pdb -ag CA -dt euc -ai alpha`

Detailed usage:

Run the following command to see the detailed usage and other options: `tsne.py -h`

CHAPTER 11

NMA Tutorial

Enterovirus 71 (EV-71) is a human pathogen that predominantly infects small children. The capsid is icoshedral and contains 60 protomer units. In a **mature capsid** the protomers are assembled as a set of 12 pentamers. Each protomer contains a single copy of the proteins VP1-VP4. During infection, the virus capsid expands to release its RNA into the host cell. This expanded capsid is known as the **A-particle.**

11.1 Aim

In this tutorial we will apply the ANM model to a single pentamer of the mature EV-71 capsid. We aim to identify the normal modes that contribute to the conformational changes within a pentamer during capsid expansion.

11.2 Create a working directory

First create a directory for all the MODE-TASK scripts using the Linux command:

```
mkdir ModeTask
```

Copy the entire contents of the MODE-TASK Scripts into the MODE-TASK directory.

Within this directory create a folder called **Tutorial:**

```
cd ModeTask
mkdir Tutorial
```

We will run all scripts from the ModeTask directory.

11.3 Preparation of structure of the mature capsid

1. Download the 3VBS biological assembly (3VBS.pdb1) of the **mature EV-71 capsid** from the PDB.

2. Open 3VBS.pdb1 in PyMOL.
3. Use the **split_states 3VBS** command to visualise the full capsid.
4. Save the capsid: File – Save Molecule – Select the first five states. Save as EV71_Pentamer.pdb into the **ModeTask/Tutorial directory.**

Each protomer has four subunits: VP1-VP4. VP4 is an internal capsid protein.

- Number of residues per protomer = 842
- Number of residues per pentamer = 4210

The estimated run time to perfom ANM on a complex of 4210 residues, using **Mode Task** is 25 hours.

For the sake of this tutorial we will use the **coarseGrain.py** script to construct a lower resolution pentamer.

11.4 Preparation of the structure of the A-particle capsid

1. Download the 4N43 biological assembly (4N43.pdb1) of the **A-partcile EV-71 capsid** from the PDB.
2. Open 4N43.pdb1 in PyMOL.
3. Use the **split_states 4N43** command to visualise the full capsid.
4. Save the capsid: File – Save Molecule – Select the first five states. Save as Apart_Pentamer.pdb into the **ModeTask/Tutorial directory.**

11.5 Coarse grain

The MODE-TASK package is designed to analyse both single proteins and larger macromolecules such as a virus capsid. The ANM.cpp script contructs an elastic network model on all CA or CB atoms in a given PDB file. This is ideal for smaller protein complexes. For larger protein complexes, the coarseGrained.py script can be used to construct an additional coarse grained PDB file.

1. Create a two models of the EV71 Pentamer complex with additional coarse graining set at levels 3 and 4 of selected CB atoms:

```
coarseGrain.py --pdb Tutorial/EV71_Pentamer.pdb --cg 3,4 --startingAtom 1 --
↪output EV71_CG3.pdb --outdir Tutorial --atomType CB
```

The input paramaters include:

- pdb: This is the pdb structure that you wish to coarse grain
- cg: This specifies the levels of coarse graining. To select fewer atoms increase the level
- starting atom: This specifies the first residue to be selected in the complex
- output: The filename of the coarse grained pdb file
- outdir: The directory in which to save the coarse grained pdb file

Output:

1. EV71_CG3.pdb and EV71_CG4.pdb : Two separate coarse grained pdb files that have the coordinates of selected CB atoms from residues that are equally distributed across the complex. As an example, EV71_CG3.pdb is shown in the figure below.
2. Command line output

```
============================================================
Started at: 2017-12-12 11:34:36.399300
------------------------------------------------------------
SUMMARY OF COARSE GRAINING PERFORMED AT LEVEL 3
No. atoms selected per unit: 122 from 842 original residues
No. atoms selected per macromolecule: 610 from 4210 orignal residues
------------------------------------------------------------
------------------------------------------------------------
SUMMARY OF COARSE GRAINING PERFORMED AT LEVEL 4
No. atoms selected per unit: 54 from 842 original residues
No. atoms selected per macromolecule: 270 from 4210 orignal residues
------------------------------------------------------------
Completed at: 2017-12-12 11:34:36.541637
- Total time: 0:00:00
```

Note that, the same set of 122 atoms from each protomer were selected for CG3, likewise, the same set of 54 atoms from each protomer were selected for CG4 – thus the symmetry of the pentamer is retained.

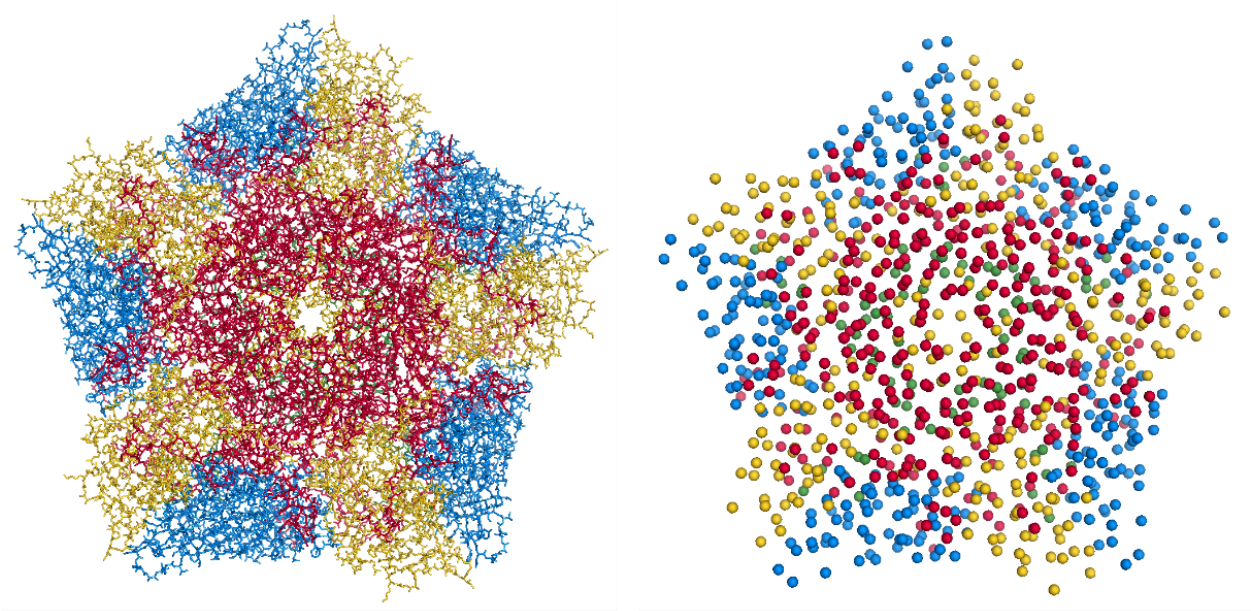

Fig. 1: Left) Crystal structure of the EV71 Pentamer (3VBS). Right) EV71_CG3.pdb contains 610 CB atoms from 4210 total residues.

11.6 Mode decomposition

The ANM.cpp script accepts a PDB file and a cutoff distance. The script constructs the Hessian matrix connecting all CB atoms within the specific cutoff radius. The script then performs singular value decomposition to return the eigenvalues and eigenvectors of the Hessian matrix.

Input parameters:

- pdb: path to PDB file

- cutoff: cutoff radius in A. The script will construct an elastic network model by connecting all atoms that interact within the cutoff distance (default = 15Å)
- outdir: folder in which output is saved

Output:

W_values.txt: A list of 3N eigenvalues of the system. Eigenvalues are ordered from slowest to fastest.

VT_values.txt: A 3Nx3N list of the eigenvectors for each mode. Eigenvectors are printed as a set of rows.

U_values.txt: A 3Nx3N list of the eigenvectors for each mode. Eigenvectors are printed as a set of columns.

1. Compile the ANM.cpp script

The ANM.cpp script requires classes of the AlgLib library. These classes can be found in the cpp/src folder in the GitHub Directory. The path to these classes must be specified in the compile command using the -I parameter:

```
g++ -I cpp/src/ ANM.cpp -o ANM
```

In this tutorial, we will perform a comparative analysis between the normal modes of the EV71_CG3.pdb and EV71_CG4.pdb

2. Run ./ANM to analyse EV71_CG4.pdb with a cutoff of 24Å

```
./ANM --pdb Tutorial/EV71_CG4.pdb  --outdir Tutorial --atomType CB --cutoff␣
↪24
```

Example of the command line output:

```
Started at: 2017-08-22 11:55:33
Starting Decomposition
Completed at: 2017-08-22 11:55:47
- Total time: 0:00:13
```

3. Run ./ANM to analyse EV71_CG3.pdb

3.1) First make a sub-directory to avoid overwriting of your previous ANM output:

```
mkdir Tutorial/CG3
```

3.2)

```
./ANM --pdb Tutorial/EV71_CG3.pdb --outdir Tutorial/CG3 --atomType CB --cutoff 24
```

Example of command line output:

```
Started at: 2017-08-22 11:56:42
Starting Decomposition
Completed at: 2017-08-22 11:59:14
- Total time: 0:02:0-704
```

11.7 Identification of modes that contribute to the conformational change

We have performed ANM on two separate pentamer complexes. From each model, we have obtained a set of eigenvalues and eigenvectors corresponding to each normal mode:

1. EV71_CG4.pdb, total non-trivial modes = 804

2. EV71_CG3.pdb, total non-trivial modes = 1824

For each model we will now identify the modes that contribute to the conformational change of a pentamer during capsid expansion.

We will then compare the modes from the respective models and determine if the additional coarse graining affected the ability to capture such modes.

To determine if our modes overlap with the direction of the conformational change, we must first determine the conformational change between the crystal structures of the **mature** and **A-particle pentamer.** The **conformationMode.py** scripts take two UNALIGNED pdb files and the set of all eigenvectors determined for the complex. The script aligns the structures, calculates the known conformational change and then identifies which modes contribute to the change.

Prepare the A-particle pentamer in PyMOL, using the biological assembly: 4n43.pdb1

11.8 Conformation mode

1. Compute the overlap between all modes of the EV71_CG4 model:

```
conformationMode.py --pdbANM Tutorial/EV71_CG4.pdb --vtMatrix Tutorial/VT_
↪values.txt  --pdbConf Tutorial/Apart_Pentamer.pdb --outdir Tutorial/ --
↪atomType CB
```

Input paramters:

–pdbANM: This is the PDB file that you use to run ANM. Do not use the aligned file here

–vtMatrix: The eigenvalues obtained from ANM of the EV71_CG4 model

–pdbConf: This is the pdb file of the conformational change. In this case, the pentamer of the A-particle (The –pdbANM and –pdbConf must NOT BE ALIGNED)

Output:

A text file with the overlap and correlation of each mode to the conformational change. The modes are ordered by the absolute value of their overlap.

2. Compute overlap between all modes of the EV71_CG3 model (Remember to specify the correct directory):

```
conformationMode.py --pdbANM Tutorial/EV71_CG3.pdb --vtMatrix Tutorial/CG3/
↪VT_values.txt  --pdbConf Tutorial/Apart_Pentamer.pdb --outdir Tutorial/CG3␣
↪--atomType CB
```

Top output from conformationalMode.py of EV71_CG4:

```
MODE           Overlap             Correlation

Mode: 9         0.759547056636      0.502678274421
Mode: 37        0.274882204134      0.0404194084198
Mode: 36       -0.266695656516      0.116161361929
Mode: 23        0.260184892921      0.0752811758038
Mode: 608       0.224274263942      0.0255344947974
Mode: 189      -0.208122679764      0.143874874887
Mode: 355       0.165654954812      0.0535734675763
Mode: 56        0.14539061536       0.11985698672
Mode: 387      -0.137880035134      0.245587436772
Mode: 307      -0.130040876389      0.145317107434
```

Top output from conformationalMode.py of EV71_CG3:

```
MODE           Overlap             Correlation

Mode: 9       -0.663942246191      0.236900852193
Mode: 30      -0.235871923574      0.192794743468
Mode: 56       0.159507003696      0.083164362262
Mode: 101      0.157155354273      0.272502734273
Mode: 172      0.156716125374      0.275230637373
Mode: 166     -0.153026188385      0.332283689479
Mode: 189     -0.147803049356      0.372767489438
Mode: 38      -0.13204901279       0.196369524407
Mode: 423     -0.131685652034      0.334715006091
Mode: 76      -0.129977918229      0.296798866026
```

In addition, the command line output will specify the precise atoms over which the calculations were performed. (Of course, this will correspond to all atoms that are present in both conformations). The RMSD between the two structures will also be specified:

```
Started at: 2017-12-12 12:50:48.922586

*****************************************************************
WARNING!!!:
Not all chains from PDB files were selected
Suggested: Chain IDs do not match between PDB Files

*****************************************************************
Correlations calculated across 465 common residues (93 per 5 asymmetric␣
↪units).
Breakdown per chain:

A: 32 residues per asymmetric unit
Residues selected include: 74 79 92 98 101 105 108 112 122 139 142 148 155␣
↪158 161 171
                           175 180 189 198 203 213 216 224 240 253 265 269␣
↪273 282
                           290 293

B: 29 residues per asymmetric unit
Residues selected include: 17 37 44 58 65 76 79 83 90 108 115 128 134 141␣
↪151 155 180
                           186 189 202 208 219 222 227 231 234 241 245 249

C: 32 residues per asymmetric unit
Residues selected include: 2 7 12 15 18 28 32 36 40 65 78 82 86 92 98 104␣
↪112 133 139
                           147 152 158 169 174 202 205 209 214 219 222 229␣
↪233

*****************************************************************


RMSD between the two conformations = 3.95802072351

Completed at: 2017-12-12 12:50:49.269902
- Total time: 0:00:00
```

11.9 Combination mode

This option allows to calculate the overlap and correlation to a conformational change, over a combination of modes. In this example, we will use the EV71_CG3 Model and perform the calculation over the modes 9 and 30.

> combinationMode.py –pdbANM Tutorial/EV71_CG3.pdb –vtMatrix Tutorial/CG3/VT_values.txt –pdb-Conf Tutorial/Apart_Pentamer.pdb –modes 9,30 –outdir Tutorial/CG3 –atomType CB

Output from combinationMode.py

The command line output is the same as described for conformationMode.py

The script will also print out two text files:

1. A file that specifies that calculated overlap and correlation over the full model:

```
MODE           Overlap              Correlation

Mode: 9        -0.663942246191      0.236900852193
Mode: 30       -0.235871923574      0.192794743468

****************************************************************
Combined Overlap = 0.616937749679
Combined Correlation = 0.219893695954
****************************************************************
```

2. A file that gives a breakdown of the calculated overlap and correlation per chain in each asymmetric unit of the model. This is very useful for identifying which regions of the complex contribute the most to the conformational change for a given mode:

```
================================================================
================================================================

ASYMMETRIC UNIT: 1
CHAIN: A
MODE           Overlap              Correlation

Mode: 9        -0.677454134085      0.101259205597
Mode: 30       -0.396594527376      0.601345215538

Combined Overlap = 0.620398046618
Combined Correlation = 0.337867917512
----------------------------------------------------------------
CHAIN: B
MODE           Overlap              Correlation

Mode: 9        -0.717931968623      0.491498558701
Mode: 30       -0.348260895864      0.249005547277

Combined Overlap = 0.679846136775
Combined Correlation = 0.321369216974
----------------------------------------------------------------
CHAIN: C
MODE           Overlap              Correlation

Mode: 9       -0.637082761027       0.198091140187
Mode: 30       0.0309855898365      0.149051660589

Combined Overlap = 0.532447057412
```

(continues on next page)

(continued from previous page)

```
Combined Correlation = 0.14767859844
-----------------------------------------------------------------

=================================================================
=================================================================

ASYMMETRIC UNIT: 2
CHAIN: A
MODE            Overlap              Correlation

Mode: 9         -0.677486033685      0.101126894833
Mode: 30        -0.396528584512      0.601655942534

Combined Overlap = 0.620396963618
Combined Correlation = 0.337655761311
-----------------------------------------------------------------
CHAIN: B
MODE            Overlap              Correlation

Mode: 9         -0.717946715867     0.491379282027
Mode: 30        -0.34820663545      0.249321165251

Combined Overlap = 0.679888476475
Combined Correlation = 0.321447980441
-----------------------------------------------------------------
CHAIN: C
MODE            Overlap              Correlation

Mode: 9        -0.637045607049       0.19801176313
Mode: 30        0.0310759318839      0.149266120068

Combined Overlap = 0.53259259653
Combined Correlation = 0.147730501227
-----------------------------------------------------------------

=================================================================
=================================================================

ASYMMETRIC UNIT: 3
.
.
.
ASYMMETRIC UNIT: 4
.
.
.
ASYMMETRIC UNIT: 5
```

11.10 Mode visualisation

From each model we have identified which mode overlaps the most with the direction of the conformational change. We can now project these vectors onto the respective models using the **visualiseVector.py** script and then visualise them as a set of frames in VMD:

1) Standard visualisation This option uses the default settings: Radius of arrow head = 2.20 Radius of arrow tail = 0.80

Arrow are coloured by chain in ascending order of PDB file according to the list:

In a biological assembly, respective chains from each asymmetric unit are presented in the same colour. The script can handle 20 non-identical changes, after which all arrows will be coloured black by default

1.1) Visualise eigenvectors for mode 9 of the CG4 model. Note this overlap is positive, thus the vectors act in the direction to conformational change. Therefore we can specify the direction as 1 (or rely on the default setting of direction = 1) when visualising the vectors:

> visualiseVector.py –pdb Tutorial/EV71_CG4.pdb –vtMatrix Tutorial/VT_values.txt –mode 9 –atomType CB –direction 1 –outdir Tutorial OR visualiseVector.py –pdb Tutorial/EV71_CG4.pdb –vtMatrix Tutorial/VT_values.txt –mode 9 –atomType CB –outdir Tutorial

1.2) Visualise eigenvectors for mode 9 of the CG3 model. Note this overlap is negative, thus the vectors act in the opposite direction to conformational change. Therefore we must specify the direction as -1 when visualising the vectors:

> visualiseVector.py –pdb Tutorial/CG3/EV71_CG3.pdb –vtMatrix Tutorial/CG3/VT_values.txt –mode 9 –atomType CB –direction -1 –outdir Tutorial/CG3

Output from visualiseVector.py

The script will produce a folder named VISUALISE. For every mode that you give to **visualiseVector.py** two files will be produced:

1. A VISUAL PDB file. This can be opened in VMD and visualised as a set of 50 frames.
2. A VISUAL_ARROWS text file. This file contains a Tcl script that can be copied into the VMD TK console. The script plots a set of arrows indicating the direction of each atom.

Visualising the results in VMD

1. Open VMD.
2. To load the VISUAL_9.pdb file click the following tabs: `File >> New Molecule >> Browse >> Select VISUAL_9.pdb.`
3. The VISUAL_9.pdb file contains a set of 50 frames of the eigenvectors of mode 9. This can be visualised as a movie by clicking on the Play button. The frame set can also be coloured to the user's desire using the options under the `Graphics >> Representations`
4. The VISUAL_ARROWS text file contains a script that can be copied and pasted straight into the Tk Console in VMD: `Extensions >> Tk Console`
5. To obtain a clearer observation, change the background to white: `Graphics >> Colors >> Under Categories select Display >> Under Names select Background >> Under Colors select White`
6. To obtain only the arrows, delete all frames of the VISUAL_9.pdb molecules: `Right click on the number of frames >> Delete frames >> Delete frames 0 to 49`

7) Alternatively you can plot the arrows onto the original PDB (uncoarse grained) PDB file and visualise it in cartoon format: Load EV71_Pentamer.pdb into VMD >>''Graphics >> Representations >> Drawing method >> NewCartoon'' >> copy and paste the VISUAL_ARROWS text file into the Tk Console. To improve clarity under the `NewCartoon` options select: `Material >> Transparent Spline Style >> B-Spline`

8. **To colour tha protein complex by chain:** `Graphics >> Colours >> Under Categories select Chain >> Under Name select A >> Under Colours select Red` To match the arrows colours as: Chain A = Red Chain B = Blue Chain C = Orche Chain D = Purple Finally instruct VMD to colour by chain `Graphics >> Representations >> Coloring Method >> Chain`

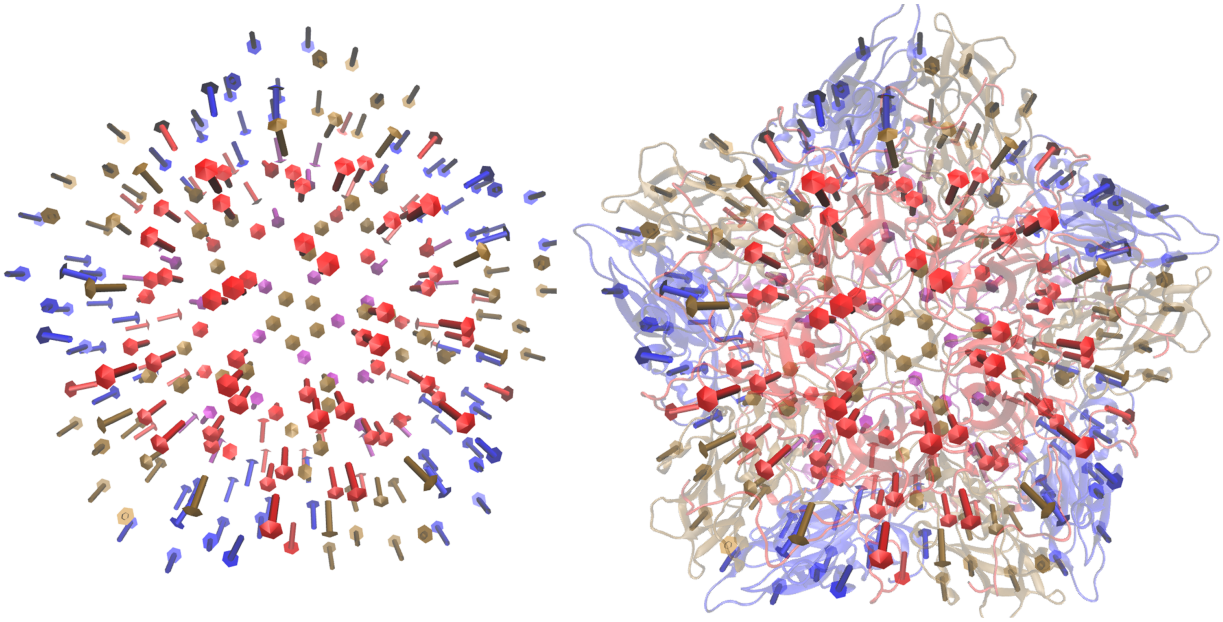

Fig. 2: Fig: Visualisation in VMD. Left) Only arrows depicted Right) Arrows plotted onto cartoon depiction of pentamer

2. Additional options for visualisation

Here you have the options to: 2.1) Change the thickness and length of the arrows 2.2) Specify the colours of the arrows for each change 2.3) Visualise the motion and draw arrows for a single or specified set of asymmetric units 2.4) Draw arrows for a single chain

We will demonstrate each of the above options using the EV71_CG4 model.

2.1) Change the thickness and length of the arrows Here we will increase the thickness of the arrow head to 3.0, increase the thickness of the arrow tail to 1.5 and the increase the length pf each arrow by a factor of 2

> visualiseVector.py –pdb Tutorial/EV71_CG4.pdb –vtMatrix Tutorial/VT_values.txt –mode 9 –atomType CB –outdir Tutorial –head 3.0 –tail 1.5 –arrowLength 2

2.2) specify the colours of the arrows for each change

Here we will colour the arrows as follows: Chain A = Yellow Chain B = Blue Chain C = Pink Chain D = Green

> visualiseVector.py –pdb Tutorial/EV71_CG4.pdb –vtMatrix Tutorial/VT_values.txt –mode 9 –atomType CB –outdir Tutorial –colourByChain yellow,blue,pink,green

2.3) Visualise the motion and draw arrows for a single or specified set of asymmetric units

Here we will visualise the motion of asymmetric units 1 and 3.

> visualiseVector.py –pdb Tutorial/EV71_CG4.pdb –vtMatrix Tutorial/VT_values.txt –mode 9 –atomType CB –outdir Tutorial –aUnits 1,3

The motion will be captured in the frame set: VISUAL_AUNITS_9.pdb in the Tutorial folder, and can be played in VMD.

2.4) Draw arrows for a single chain

Here we will draw arrows only for A chain of asymmetric unit 1 of the EV71_CG4 pentamer, in colour gray

> visualiseVector.py –pdb Tutorial/EV71_CG4.pdb –vtMatrix Tutorial/VT_values.txt –mode 9 –atomType CB –outdir Tutorial –aUnits 1 –chain A –colourByChain gray

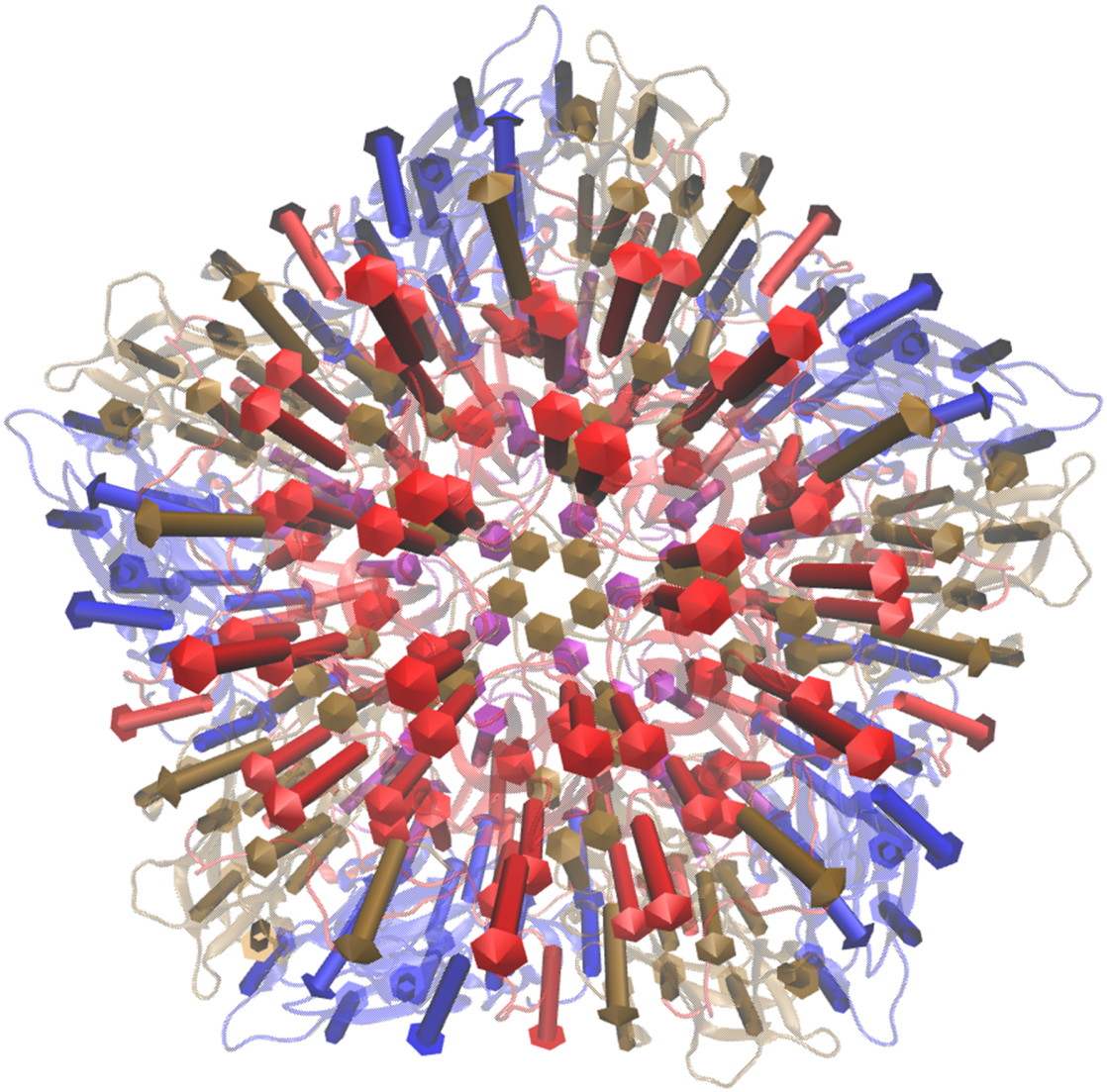

Fig. 3: Fig: Visualisation in VMD after increasing arrow sizes

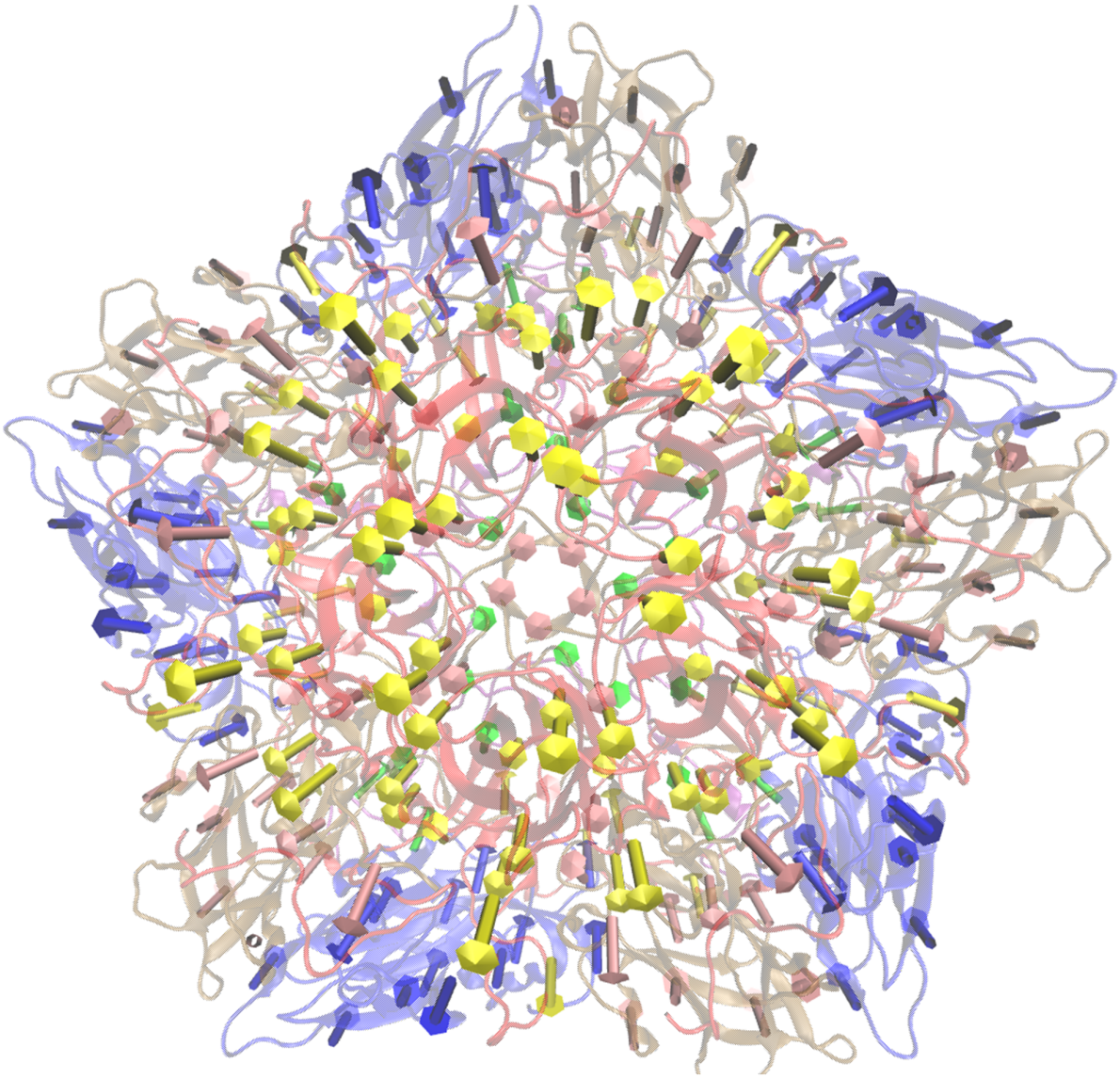

Fig. 4: Fig: Visualisation in VMD with arrows coloured as specified by user

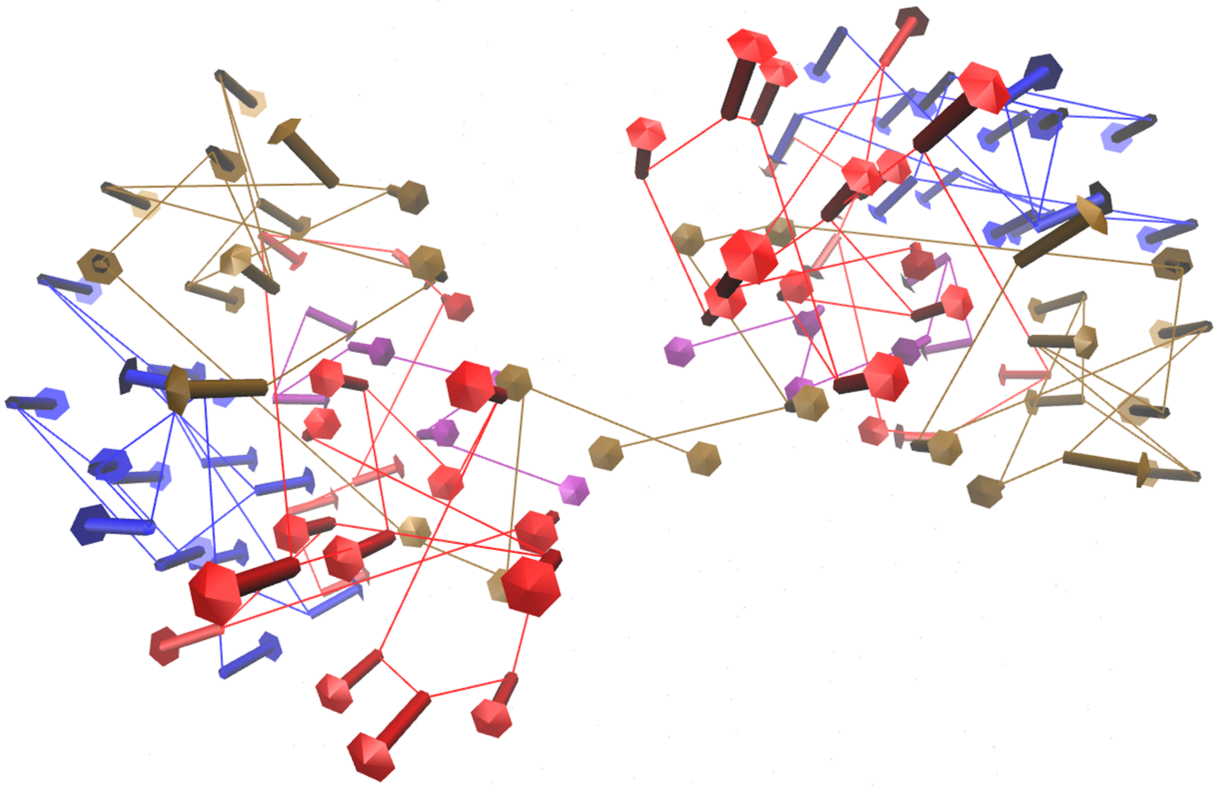

Fig. 5: Fig: Vectors arrows for asymmetric units 1 and 3 of the pentamer

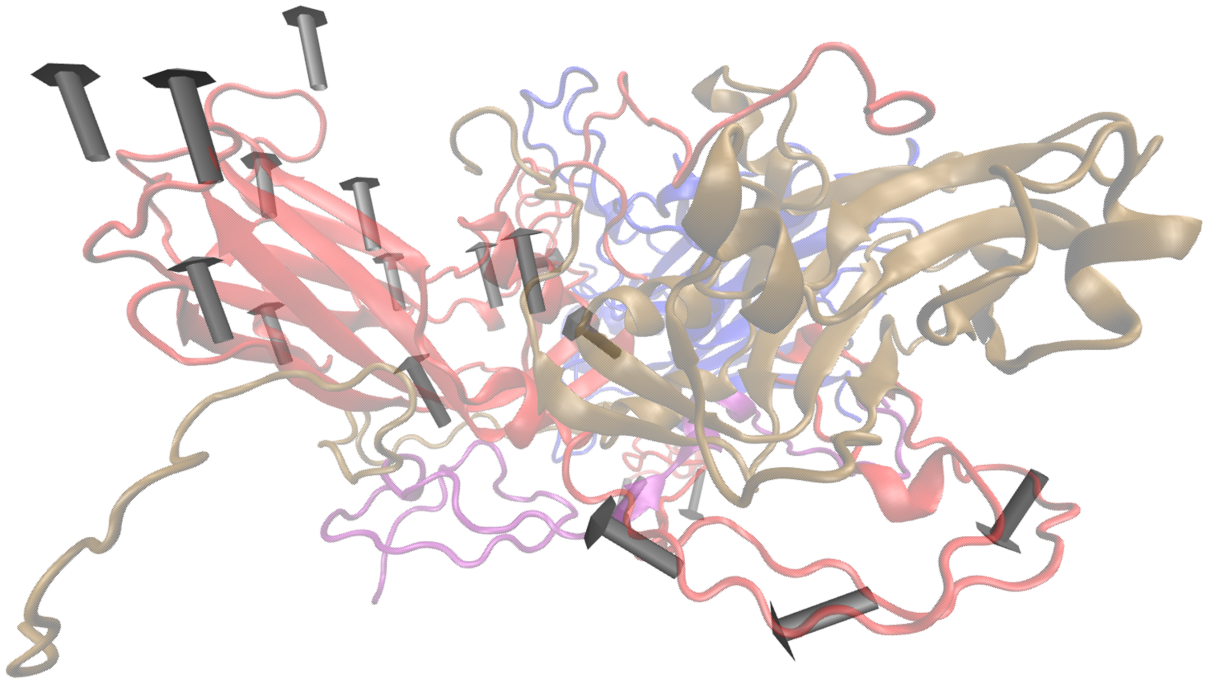

Fig. 6: Fig: Vectors arrows for Chain A of asymmetric units 1 in colour gray

11.11 Mean square fluctuation (MSF)

Next, we will use the meanSquareFluctuations.py script to calculate the MSF of the CB atoms. The scripts allows you to calculate:

1. the MSFs, calculated over all modes
2. the MSFs of the CB atoms for a specific mode, or a specific range of modes.

The script also allows for comparison of MSF obtained from modes of different models. We can use the –pdbConf2 parameter to send the script a second PDB model. The script will then calculate the MSF of atoms corresponding to residues that are common between both models.

In this tutorial, we will analyse and compare the MSF between EV71_CG4 and EV71_CG3. This will give an indication as to whether or not the higher coarse grained model is also suitable to study the virus.

1. We will compare the MSFs between the two models for a) all modes, and b) mode 9

```
meanSquareFluctuation.py --pdb Tutorial/EV71_CG3.pdb --wMatrix Tutorial/CG3/
↪W_values.txt --vtMatrix Tutorial/CG3/VT_values.txt --pdbConf2 Tutorial/
↪EV71_CG4.pdb --wMatrixC Tutorial/W_values.txt        --vtMatrixC Tutorial/
↪VT_values.txt --modes 9 --outdir Tutorial/ --atomType CB
```

Output for Model CG3:

1) PDB1_msf.txt: Text file of the overall MSFs values for all residues of CG3

2) PDB1__msfSpecificModes.txt: MSFs for all residues for mode 9 of CG3

3) PDB1CommonResidues_msf.txt: Overal MSFs for residues (of CG3) common to CG3 and CG4

4) PDB1_CommonResidues_msfSpecificModes.txt: MSFs for residues (of CG3) common to CG3 and CG4 calculated for mode 9

Output for Model CG4:

1) PDBCompare_msf.txt:: Text file of the overall MSFs values for all residues of CG4

2) PDBCompare__msfSpecificModes.txt: MSFs for all residues for mode 9 of CG4

3) PDBCompareCommonResidues_msf.txt: overal MSFs for residues (of CG4) common to CG4 and CG3.

4) PDBCompare_CommonResidues_msfSpecificModes.txt: MSFs for residues (of CG4) common to CG4 and CG3 calculated for mode 9

11.12 Assembly Covariance

Now, we will use the assemblyCovariance.py script to calculate to plot various covariance matrices of the complex. For this example we will use the EV71_CG3 Model.

1. First, we will plot the overall covariance for the full model, as calculated over all modes:

   assemblyCovariance.py –pdb Tutorial/EV71_CG3.pdb –wMatrix Tutorial/CG3/W_values.txt –vtMatrix Tutorial/CG3/VT_values.txt –modes all –outdir Tutorial/CG3/ –atomType CB

The above function will produce a plot corresponding to the full model, AND as a default a second plot that zooms into the first asymmetric unit will also be produced

2. Now we will use the additional options to calculate the covariance for mode 7 only (the first non-trivial mode). We will also plot the covariance between the asymmetric units 1 and 3, and then zoom into chain A of the first asymmetric unit. We have also adjusted the values of the axes to increase sensitivity for a single mode.

   assemblyCovariance.py –pdb Tutorial/EV71_CG3.pdb –wMatrix Tutorial/CG3/W_values.txt –vtMatrix Tutorial/CG3/VT_values.txt –modes 7 –aUnits 1,3 –zoom 1,A –outdir Tutorial/CG3/M7 –atomType CB –vmin -0.005 –vmax 0.005

The above function will produce a plot corresponding to the full model for mode 7, a second plot that zooms into covariance between the first and third asymmetric units, and a third plot for the covariance of Chain A and Unit 1.

For each of the steps above, the script also outputs each covariance matrix in txt file format.

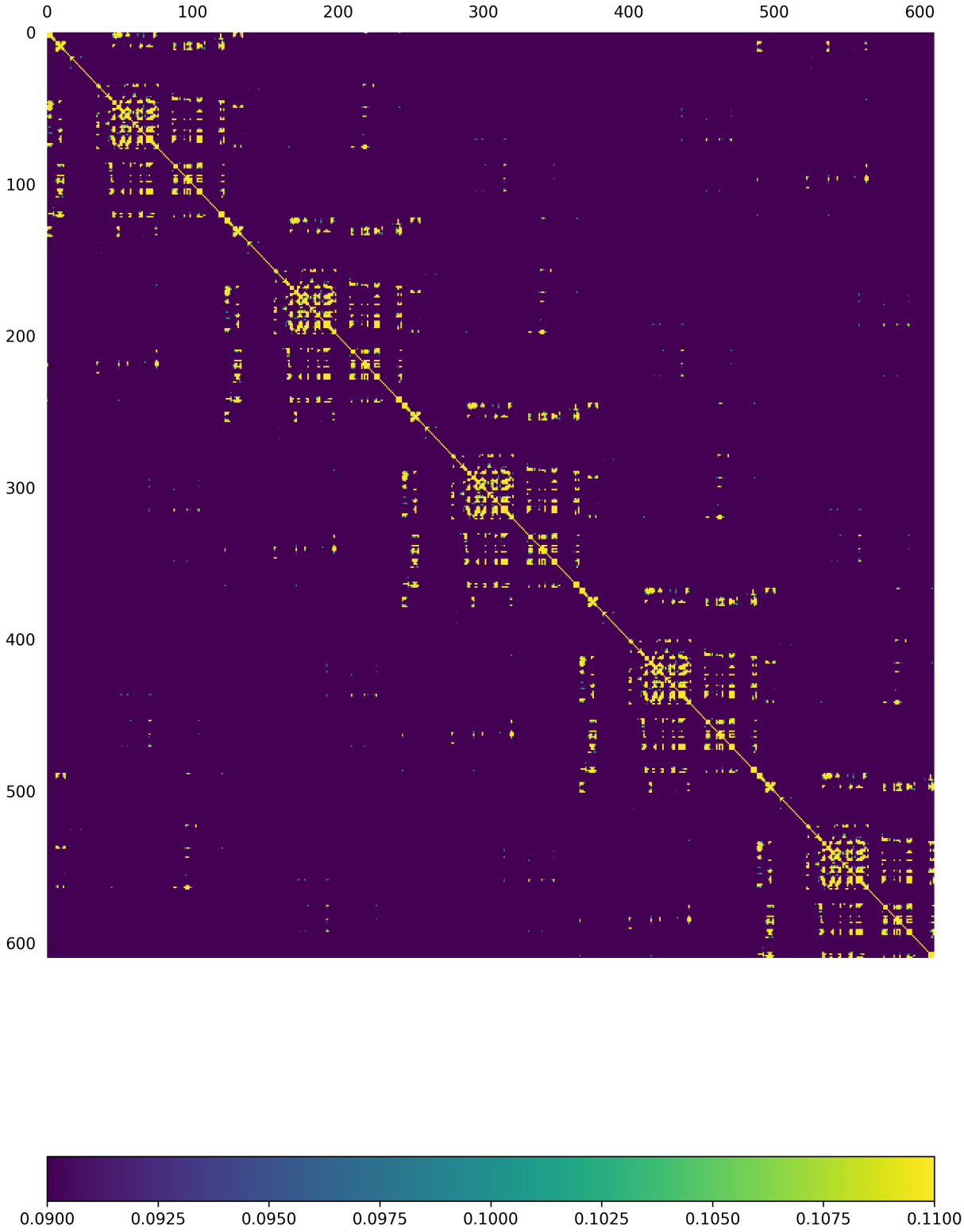

Fig. 7: Fig: Overall covariance matrix for the full EV71_CG3 Model

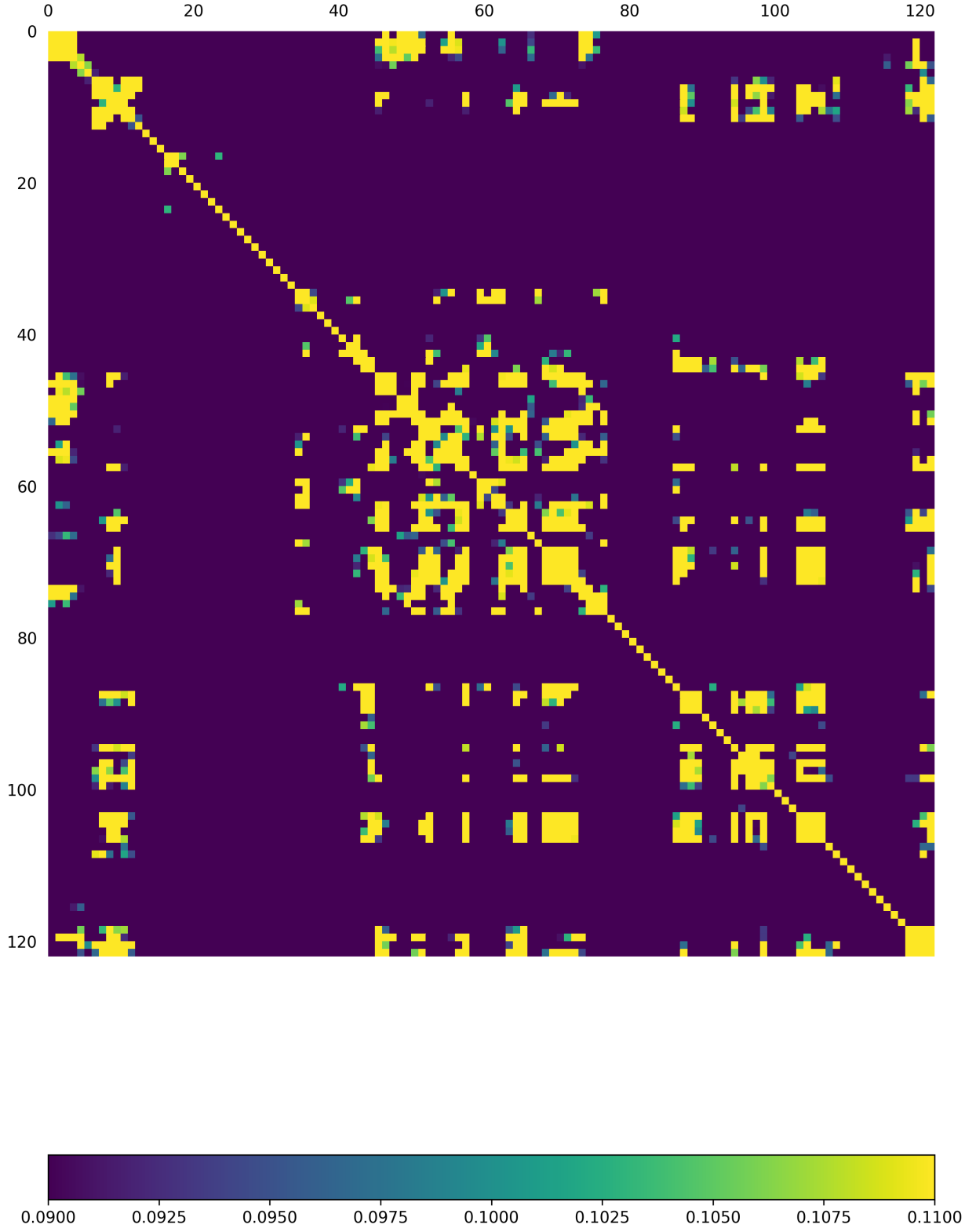

Fig. 8: Fig: Overall covariance matrix for a single protomer within the EV71_CG3 Model

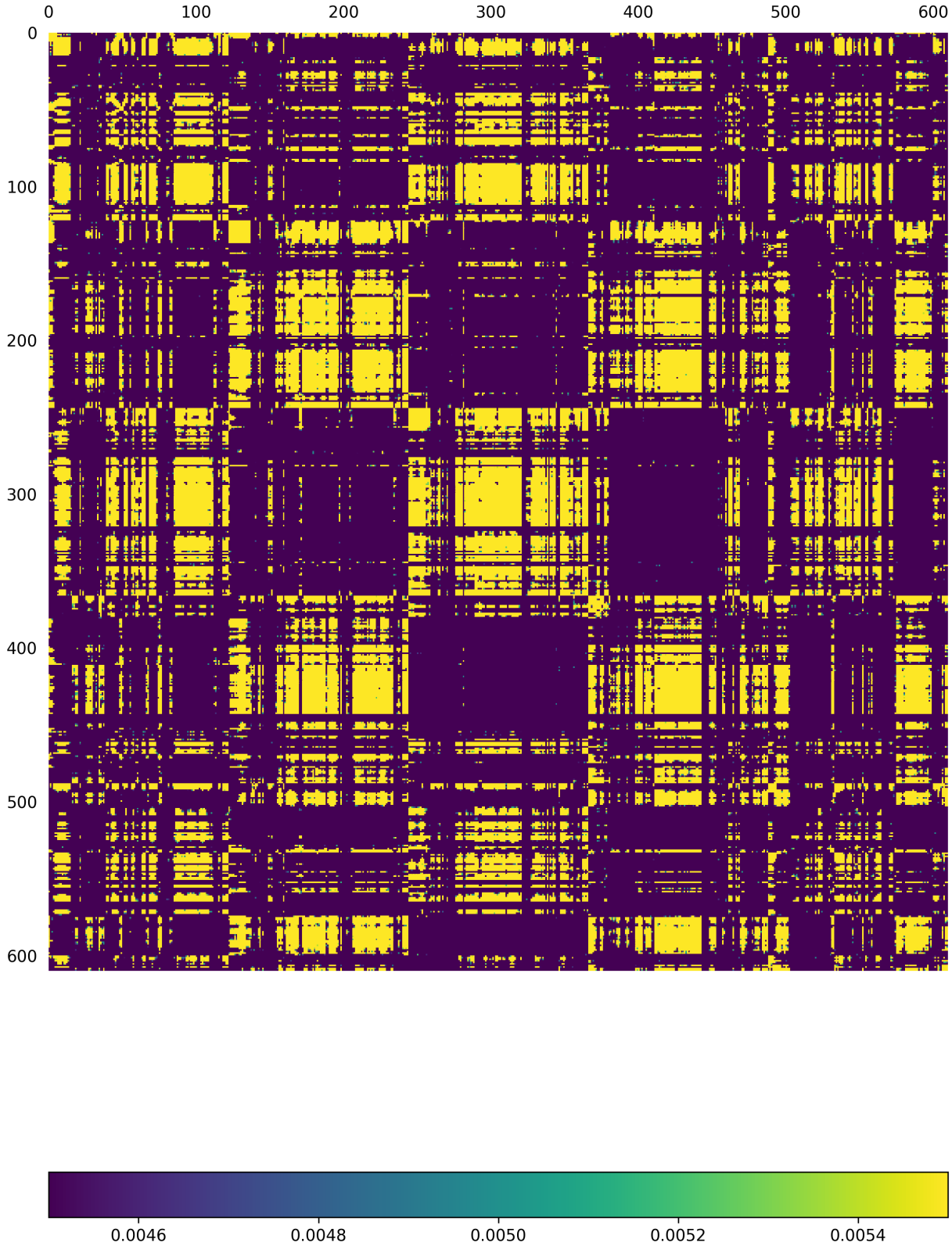

Fig. 9: Fig: Covariance matrix for the full EV71_CG3 Model calculated over Mode 7

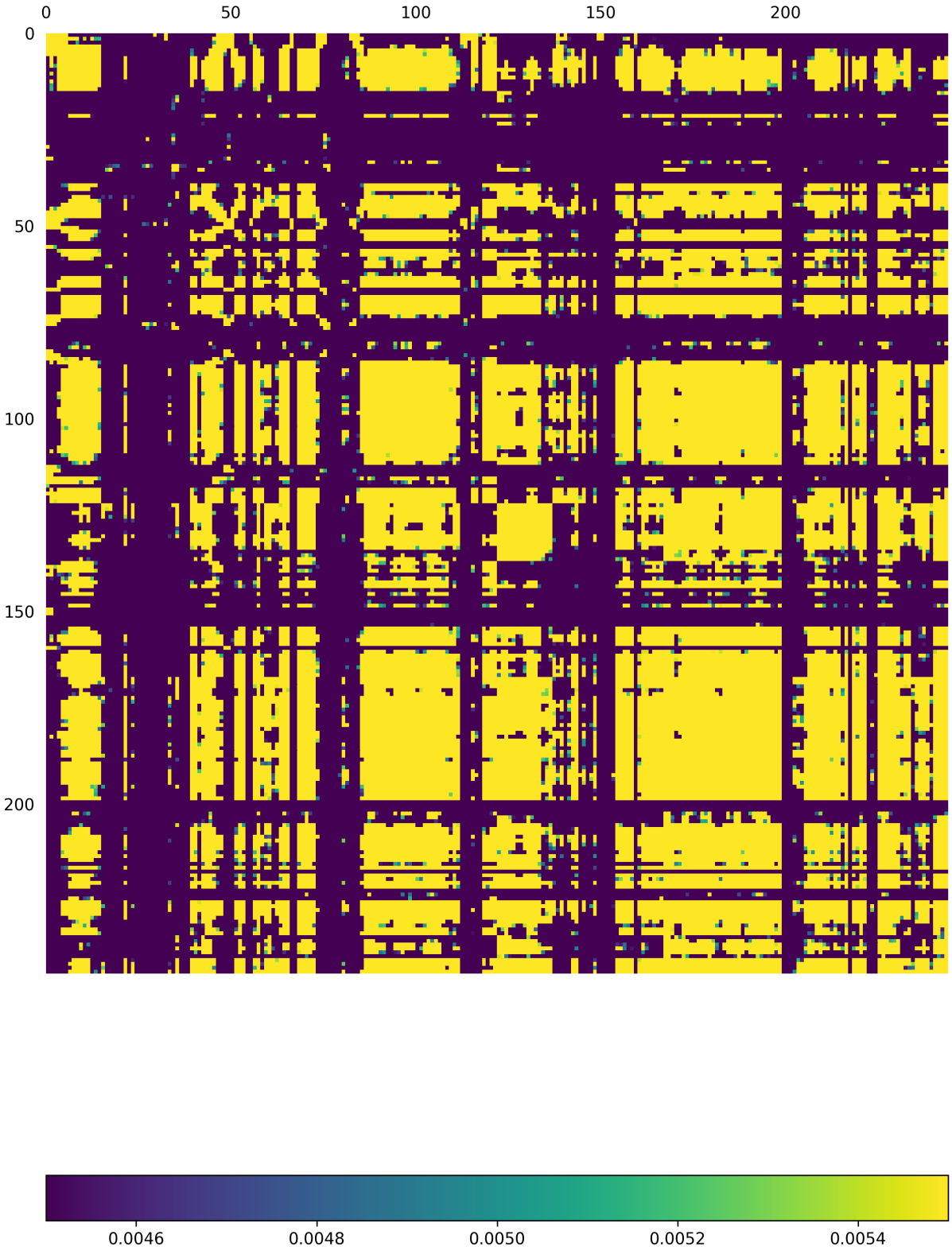

Fig. 10: Fig: Covariance matrix for the asymmetric units 1 and 3 of the EV71_CG3 Model calculated over Mode 7

Fig. 11: Fig: Covariance matrix for Chain A in asymmetric units 1 the EV71_CG3 Model calculated over Mode 7

CHAPTER 12

PCA Tutorial

12.1 PCA of a MD trajectory

In this tutorial, we will be performing PCA on a MD trajectory of protein. Before doing the PCA, we need to prepare the trajectory which includes removing periodicity and removing water molecules. Most of the MD packages have options to do this. We will be using GROMACS in this tutorial. We will be using .xtc format for trajectory and .pdb for topology file. Any other common trajectory format should also work with MODE-TASK.

1. Preparation of trajectory

1.1. Remove periodicity

```
gmx_mpi trjconv -s md_01.tpr -f md_01.xtc -o md_01_noPBC.xtc -pbc mol -ur␣
↪compact
```

select system to apply it.

1.2. Remove water

```
gmx_mpi trjconv -s md_01.tpr -f md_01_noPBC.xtc -o md_01_noWAT.xtc -n index
```

and select protein

2. Create a working directory

First, create a directory for all the MODE-TASK scripts using the Linux command:

```
mkdir ModeTask
```

Copy the entire contents of the MODE-TASK scripts into the MODE-TASK directory.

Within this directory create a folder called Tutorial:

```
cd ModeTask
mkdir Tutorial
```

We will run all scripts from the MODE-TASK directory. Move the trajectory (md_01_noWAT.xtc) and topology file (complex.pdb) into the Tutorial directory.

3. Running PCA

MODE-TASK includes tools to perform PCA on Cartesian coordinates as well as internal coordinates. It also allows users to run different variants of PCA on a single MD trajectory.

3.1. PCA on Cartesian coordinates

Run the following command to perform the singular value decomposition (SVD) PCA on CA atoms.

```
pca.py -t Tutorial/md_01_noWAT.xtc -p Tutorial/complex.pdb -ag CA -pt svd
```

Output:

(a)- Various output files are written to the out_md_01_noWAT.xtc directory. 2D Plot of first 3 PCs, Scree plot, RMSD plot, and RMSD Modes plot. For details about these output files, please refer to the MODE-TASK documentation.

(b)- Command line output: Following output is redirected to command line.

```
Results will be written in out_md_01_noWAT.xtc
Reading trajectory Tutorial/md_01_noWAT.xtc ...
No reference structure given, RMSD will be computed to the first frame in␣
↪the trajectory
Trajectory info:
Total 101 frames read from Tutorial/md_01_noWAT.xtc
MD time is from 199000.0 to 200000.0 ps
13244 atoms and 861 residues in the trajectory
Atom group selected for PCA: CA
Total 860 CA atoms selected for analysis
KMO for input trajectory is 5.25051335835e-06
RMSD written to rmsd.agr
Performing SVD (Single Value Decomposition) PCA with 'auto' svd_solver
Trace of the covariance matrix is:  4.9427056479
cosine content of first PC= 0.777934456531
cosine content of second PC= 0.643848137376
cosine content of 3rd PC= 0.70061477062
cosine content of 4th PC= 0.530112237076
```

3.2. Visualizing the results

2D Plot of the first 3 PCs in grace and png format is written. In order to open the .agr file with xmgrace run the following command.

```
xmgrace out_pca_test_trj.xtc/pca_projection1_2.agr
```

You can also visualize the .png format figure plot by opening it with your favorite picture visualizer. In the same way, open the rmsd.agr and pca_variance.agr.

3.3. PCA on internal coordinates

One can also do PCA on internal coordinates of a MD trajectory. Options are available for different types of internal coordinates such as, pairwise distance between atoms, 1-3 angle between backbone atoms, phi angle, and psi angle. Run the following command to do PCA on pairwise distance between CA atoms.

```
internal_pca.py -t Tutorial/md_01_noWAT.xtc -p Tutorial/complex.pdb -ag CA -
↪ct distance
```

Run the following command to do PCA on backbone psi angles.

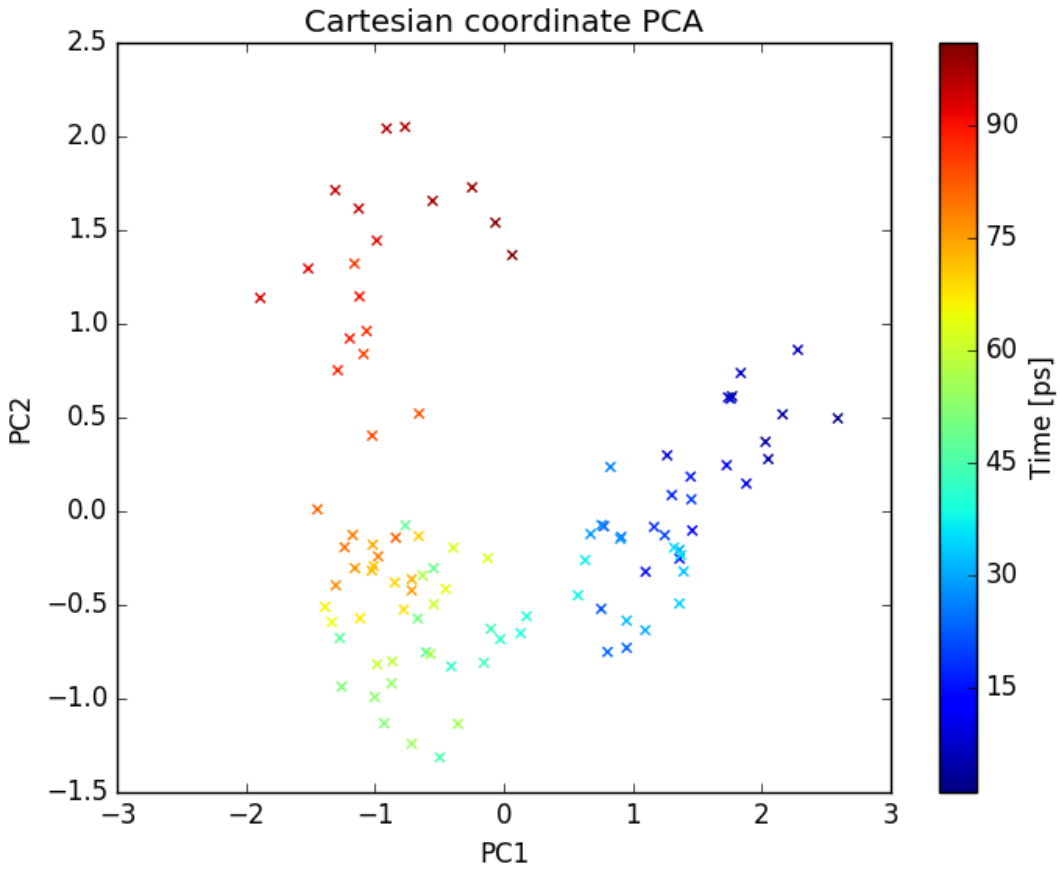

Fig. 1: plot of PC1 and PC2

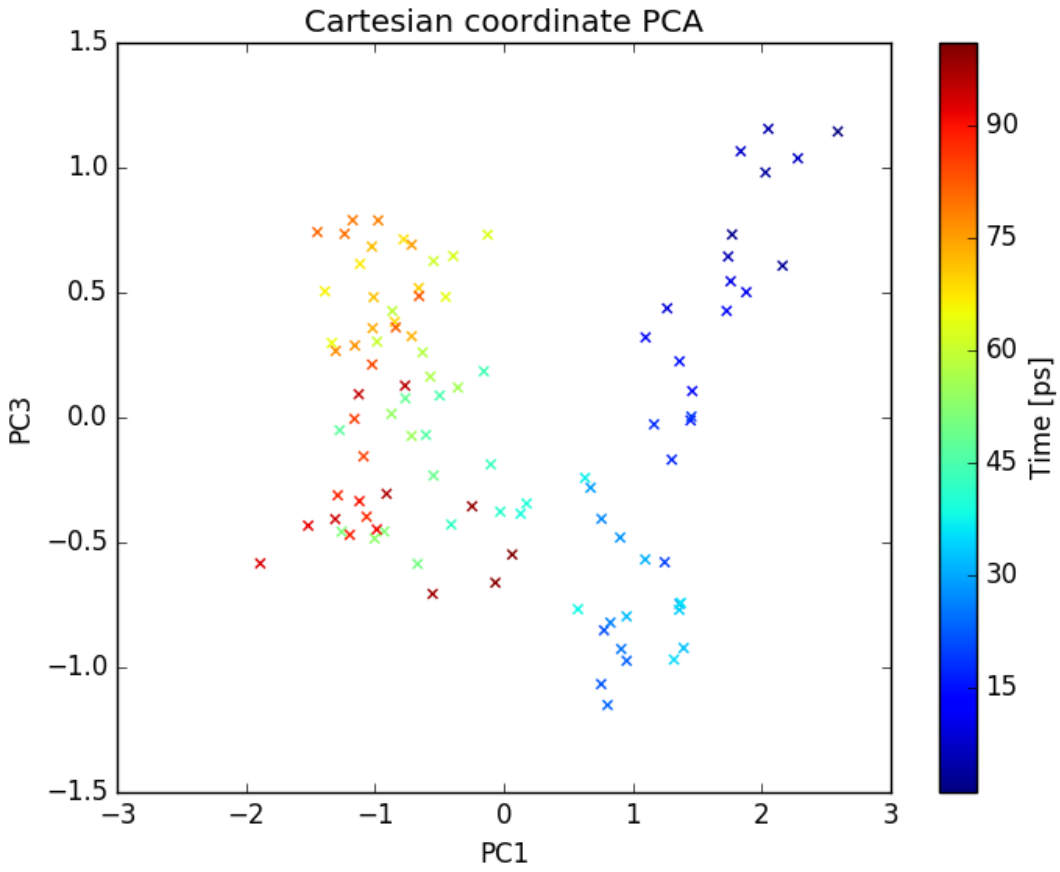

Fig. 2: plot of PC1 and PC3

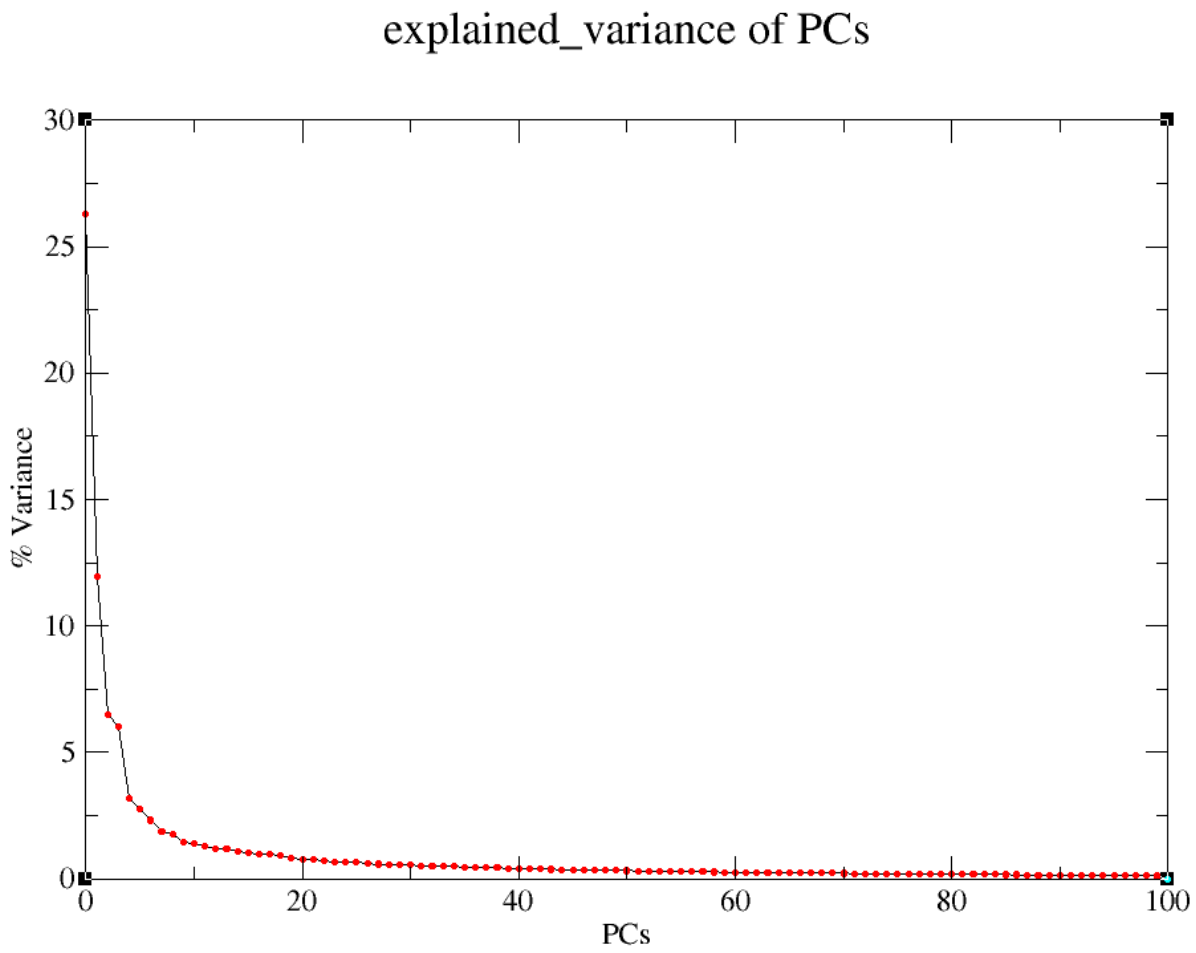

Fig. 3: Explained variance of PCs

```
internal_pca.py -t Tutorial/md_01_noWAT.xtc -p Tutorial/complex.pdb -ag CA -
↪ct psi
```

Output files include 2D plot of first 3 PCs and Scree plot, which can be visualized using xmgrace as described earlier.

12.2 MDS (Multi-dimensional scaling) on a MD trajectory

To perform the MDS on pairwise RMSD between C-alpha atoms, run the following command.

```
mds.py -t Tutorial/md_01_noWAT.xtc -p Tutorial/complex.pdb -ag CA -dt rmsd
```

Output files include 2D plot of first three PCs which can be visualized using xmgrace as described earlier.

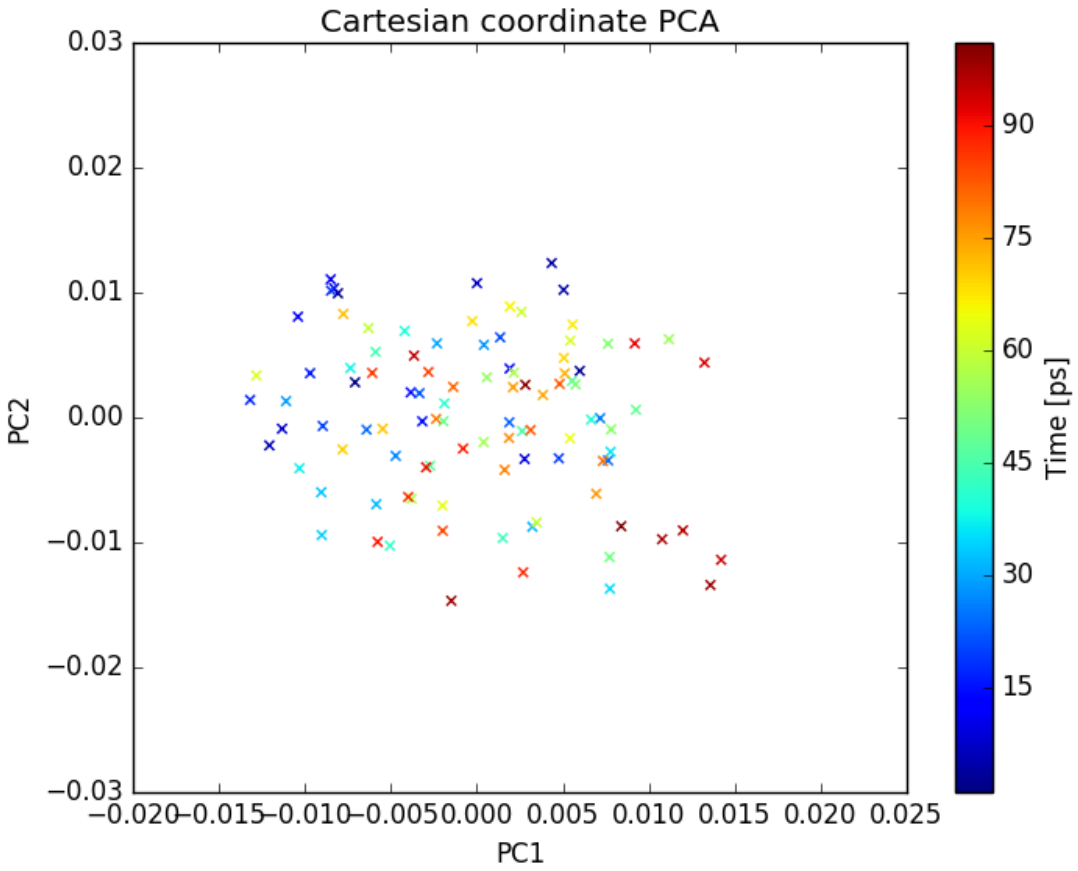

Fig. 4: plot of PC1 and PC2

12.3 t-SNE on a MD trajectory

Run the following command to perform t-SNE using pairwise RMSD of CA atoms as the index of dissimilarity.

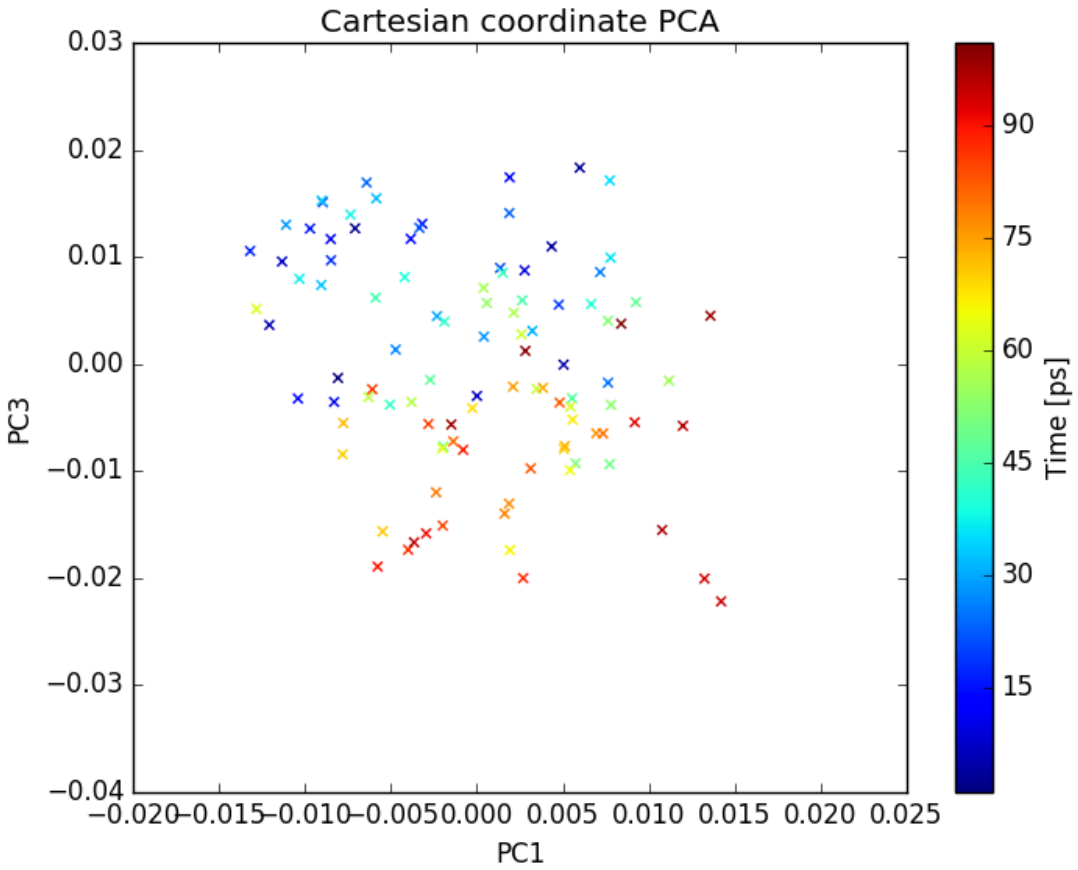

Fig. 5: plot of PC1 and PC3

```
tsne.py -t Tutorial/md_01_noWAT.xtc -p Tutorial/complex.pdb -ag CA -dt rmsd
```

Output files include 2D plot of the first 3 PCs, which can be visualize using xmgrace as described earlier.

Note: The t-SNE algorithm is non-linear and highly flexible, which makes it difficult to interpret the results. Different set of parameters gives very different output. Users are required to try different set of values for "perplexity" , "learning rates", and "number of iteration". A useful discussion covering these issues can be found here https://distill.pub/2016/misread-tsne/

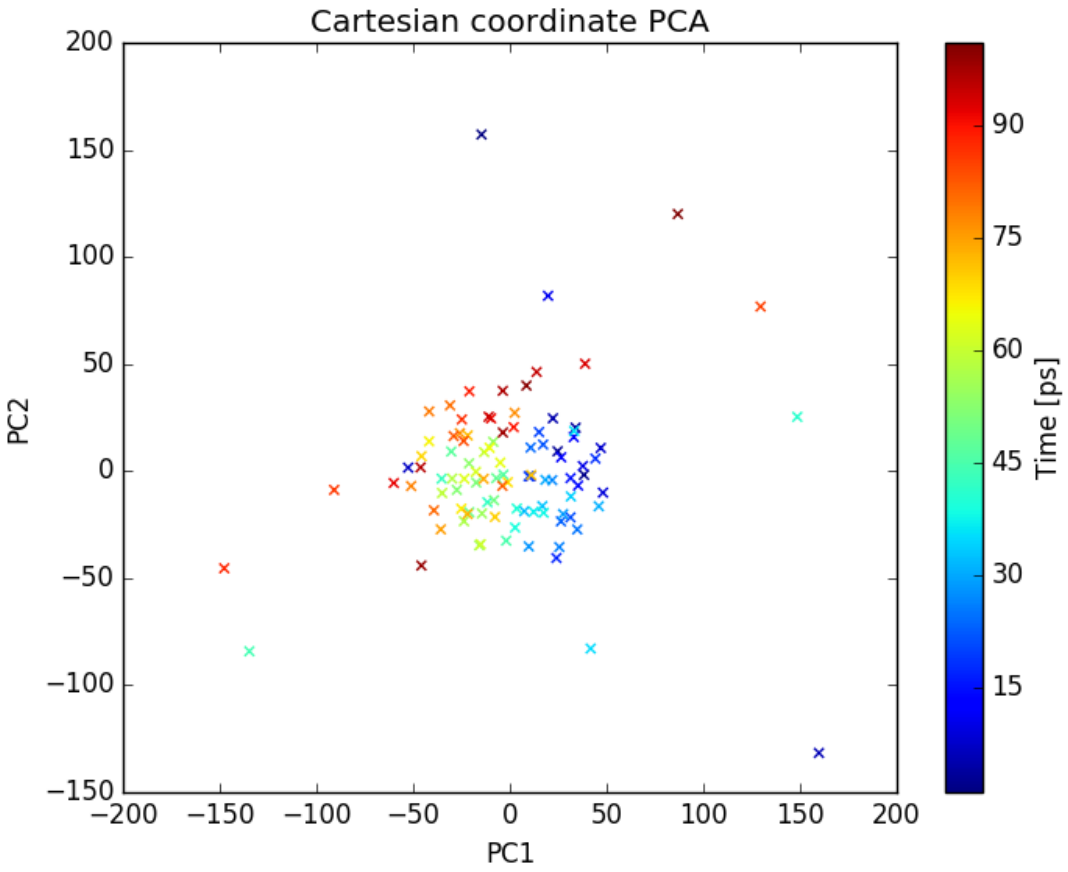

Fig. 6: plot of PC1 and PC2

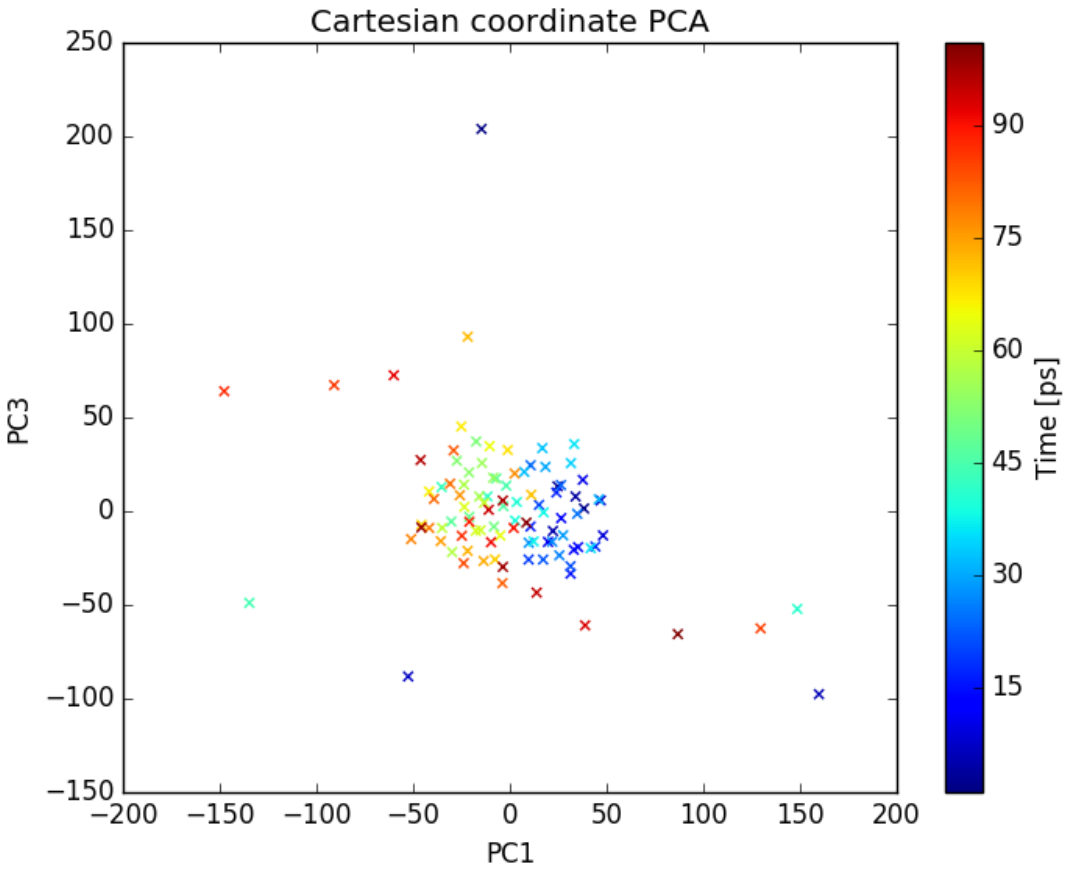

Fig. 7: plot of PC1 and PC3

CHAPTER 13

pyMODE-TASK- PyMOL plugin

Kindly see the pyMODE-TASK documentaion for further details.