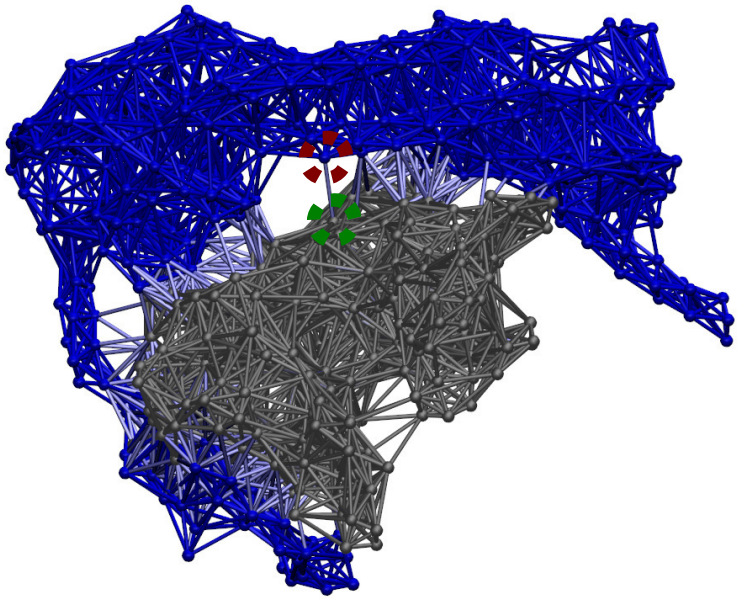
cNMA Usage Documentation

Tomasz Oliwa, Haoran Chen
cnma.shenlab@gmail.com

September 14, 2016

Abstract

This is the work in progress documentation for the cNMA set of scripts. It containts usage hints for a user points of view.



Contents

1 cNMA Prerequisites

The prerequisites for cNMA are:

- Python >= 2.7
- ProDy >= 1.5.1 `http://prody.csb.pitt.edu/`
- NumPy (tested on 1.8.0), SciPy (tested on 0.12.1 and 0.13.3)
- Matplotlib

1.1 Anaconda - A collection of prerequisites

The aforementioned (except ProDy) were installed on the cluster in userspace through the Anaconda Python package `http://continuum.io/downloads`. At the time of writing, the Anaconda package for 64 Bit machines with Python 2.7 version is ''Anaconda2-4.1.1-Linux-x86_64.sh'' and can be installed through the following commands:

```
$ chmod +x Anaconda2-4.1.1-Linux-x86_64.sh
$ ./Anaconda2-4.1.1-Linux-x86_64.sh
```

The will create a about 1.1 GB large directory "anaconda2" in the home directory " /", containing the binaries, packages and environments of Python and the libaries. To then run a Python program with the Python interpreter from Anaconda, the path to it should be given in the commandline, such as:

```
$ ~/anaconda2/bin/python x.py
```

In this manual, it is assumed that the cNMA and other Python programs are run with the prerequisites installed, and a Python 2.7 interpreter is called either through the Anaconda path or by having it installed otherwise.

Anaconda is fairly large and installs numerous packages. To reduce the needed installation space, the package "Miniconda" can be used instead. It is available from `http://conda.pydata.org/miniconda.html#miniconda`. Miniconda contains a version of Python and the conda installer, through which specific pre-requisites (such as NumPy) can be obtained. Commandline examples to do so can be found on the Miniconda website.

ProDy itself can be also installed once Anaconda has finished the installation process. Anaconda provides "pip", a package manager for Python to do so as follows:

```
$ ~/anaconda2/bin/pip install ProDy
```

The pip installation of ProDy will be placed inside the anaconda2 directory.

For Python 3.5 users, the only difference step for the procedure above is that the Anaconda version should be ‘‘Anaconda3-4.1.1-MacOSX-x86_64.sh’’.

1.2 Download of cNMA

cNMA is available on Github at: `https://github.com/Shen-Lab/cNMA`.

cNMA can be obtained by using “Git” through:

```
$ git clone https://github.com/Shen-Lab/cNMA.git
```

The commandline will promt for the users github username and password.

cNMA can be also downloaded through a web browser by clicking “Clone or download ” and then “Download ZIP”at `https://github.com/Shen-Lab/cNMA`.

Note that if using “Download ZIP” to obtain cNMA, the name of unzipped folder will be cNMA-master.

1.3 Modification to Prerequisites

Modifications to the ProDy prerequisite have been made and can be found in the folder “cNMA/Manual/Modified/”. The modifications are:

- ProDy:
    1. Add method matchTNMAChains in prody.proteins.compare
       This method defines chain matching based on Bio.pairwise2 from Biopython and forces it. The chain matching method was too deeply rooted in ProDy to create a new one in the NMAUnified programs, hence a method “matchTNMAChains” was created based on the ProDy method “matchChains” and put into prody.proteins.compare (in the file compare.py)
    2. Bugfix to pass the argument zeros in the method calcANM in prody.dynamics.anm, to calculate trivial normal modes (in the file anm.py)
    3. Addon with the zeros parameter to write trivial normal modes directly with the writeNMD method in prody.dynamics.nmdfile. This addon is not necessary to run the cNMA data collection, as these perform the writing of trivial normal modes into the NMD

files via shell scripting and temp files. However, this addon is useful for direct experimentation with ProDy and normal modes (in the file nmdfile.py) to be able to visualize modes directly after calculating them.

The files are located in the `cNMA/Manual/Documentation/modified/` subfolder of this manual and should be copied over the original ProDy files with the same name.

A script called "replacePrerequisites.sh" is provided along with the modified files, which will place them in their right position. To run it, the current work directory should be changed into its folder, the script made executable and then finally run, this is done by the following three commands:

```
$ cd cNMA/Manual/Documentation/modified/
$ chmod +x replacePrerequisites.sh
$ ./replacePrerequisites.sh
```

The contents of "replacePrerequisites.sh" itself is:

```
#!/bin/bash
cp compare.py ~/anaconda2/lib/python2.7/site-packages/prody/proteins/
cp anm.py ~/anaconda2/lib/python2.7/site-packages/prody/dynamics/
cp nmdfile.py ~/anaconda2/lib/python2.7/site-packages/prody/dynamics/

rm -f ~/anaconda2/lib/python2.7/site-packages/prody/proteins/compare.
    pyc
rm -f ~/anaconda2/lib/python2.7/site-packages/prody/dynamics/anm.pyc
rm -f ~/anaconda2/lib/python2.7/site-packages/prody/dynamics/nmdfile.
    pyc
```

If Anaconda is installed in a different path then specified by the script above, this path needs to be located to replace the modified .py files.

For instance, on dao the userspace path is `~/anaconda2/lib/python2.7/site-packages/prody/` and on the local Fedora machine it is `/usr/lib64/python2.7/site-packages/prody/`.

The corresponding *.pyc also need to be deleted so that Python will guarantee to compile the .py file anew. A .pyc file is usually made when the Python .py file is imported by some other script and contains the compiled byte-code of the source file.

2 Data collection

Usage overview over how to generate the cNMA results.

2.1 NMAUnified

The main program for the cNMA is calculation is "NMAUnified.py". Its input are a configuration class describing the experiment and two proteins in PDB format. Its output is written in a folder and contains the experimental results such as RMSD reduction and NMD files.

2.1.1 Input

Running it without arguments prints a help text with usage and argument descriptors:

```
$ python NMAUnified.py
```

Ultimatively, NMAUnified expects two arguments in the following order:

1. config, the Configurations class
2. protein1_A, the Protein 1 in first conformational state
3. protein2_A, the Protein 2 in first conformational state

In config, the parameters of the experiments are set, each parameter is documented via a comment. Examples of parameters are the output paths, the kind of NMA performed or the RMSD fitting precision.

An example of how to run the program with a profiler attached to it is:

```
$ python NMAUnified.py --profile ~/cNMA/Manual/Example/Example1/Input/
    Configuration.py ~/cNMA/Manual/Example/Example1/Input/1A2K_r_u.pdb
     ~/cNMA/Manual/Example/Example1/Input/1A2K_l_u.pdb
```

The optional parameters, such as "–profile" for NMAUnified are:

- -h, –help show this help message and exit
- –profiler Run the program with a profiler attached to it, that writes runtime information after a successful run.
- –outputPath OUTPUTPATH Directly specify the output path of the program, ignoring the output path in the Configuration
- –affirmProteinNames AFFIRMPROTEINNAMES If the arguments for protein1_A, protein2_A, do not follow the xxxx_r/l_u/b.pdb naming scheme of the Protein-Protein Docking Benchmark 4.0, provide "receptor" to tell the program that protein1 is a receptor (and protein2 therefore a ligand), or provide "ligand" to state the opposite. The titles of the proteins will be adjusted accordingly.

2.1.2 Output

The output will be written outputPath+experimentName.
The variables outputPath and experimentNamePrefix are set in the Configuration file, the experimentName is determined by the filename of the protein inputs. An example from a Configuration.py file can looks as follows:

```
# Path to the output
self.outputPath = "/home/usersname/Documents/cNMAoutput/"

# Experiment name prefix to be used to create the results output folder
self.experimentNamePrefix = "results"
```

2.2 Examples

2.2.1 Example 1

Two pdb files for cNMA can be found in `cNMA/Manual/Example/Example1/Input/`. One is 1A2K_r_u.pdb as unbound receptor, another is 1A2K_l_u.pdb as unbound ligand. After running the shell script "run_example1.sh" in `cNMA/Manual/Example/Example1/`, the output is in the folder `cNMA/Manual/Example/Example1/Result/`. The expected output is in the folder `cNMA/Manual/Example/Example1/Output/` for comparison. Note that in "run_example1.sh", the default path for input files is `~cNMA/Manual/Example/Example1/Input`.

2.2.2 Example 2

One pdb file 1A2K_r_u.pdb for canonical NMA can be found in `cNMA/Manual/Example/Example2/Result/`. After running the shell script "run_example2.sh" in `cNMA/Manual/Example/Example2/`, the output is in the folder `cNMA/Manual/Example/Example2/Result/`. The expected output is in the folder `cNMA/Manual/Example/Example2/Output/` for comparison. Note that in "run_example2.sh", the default path for input files is `~cNMA/Manual/Example/Example2/Input`.

2.2.3 Example 3

Four pdb files for cNMA with posterior analysis can be found in `cNMA/Manual/Example/Example3/Result/`. They are 1A2K_r_u.pdb for unbound receptor, 1A2K_l_u.pdb for unbound ligand, 1A2K_r_b.pdb for bound receptor, and 1A2K_l_b.pdb for bound ligand. After running the shell script

"run_example3.sh" in `cNMA/Manual/Example/Example3/`, the output is in the folder `cNMA/Manual/Example/Example3/Result/`. The expected output is in the folder `cNMA/Manual/Example/Example3/Output/` for comparison. Note that in "run_example3.sh", the default path for input files is`~cNMA/Manual/Example/Example3/Input`.

2.2.4 Configuration

The Configurations file given to NMAUnified determines the experiment setup. In the following, at first a parameter dictionary for the configuration file is given, followed by a listing of example configuration files accompaning data gathering files. Note that the output path in configuration is based on the current directory. It could be modified to absolute path where output need to be in.

2.2.5 Configuration Dictionary

The parameters (here in **boldface**) are present in a configuration file. After each parameter, an explanation of it is given here, followed by an example of a set parameter. Parameters with star are for posterior analysis in Example 3.

- **self.outputPath**: The path to the output.
  Example: self.outputPath=''/home/usersname/Documents/cNMAoutput/''
- **self.experimentNamePrefix**: Experiment name prefix to be used to create the results output folder.
  Example: self.experimentNamePrefix = "result"
- **self.investigationsOn**: Data collection on individual proteins or for a complex. Possible values: "Individual", "Complex".
  Example: self.investigationsOn=''Individual''
- **self.measuresOnWhole**: Data collection on the whole protein/complex if true, else on the interface. Possible values: "True", "False".
  Example: self.measuresOnWhole=True
- **self.calculateZeroEigvalModes**: Calculate zero eigenvalue modes.
  Example: self.calculateZeroEigvalModes=True

- **self.align***: Align styles of unbound to bound. The setting of the aforementioned **measuresOnWhole** will furthermore determine if the alignment basis is the whole or the interface. Possible values: "alpha": align protein1 and protein2 independently, "beta" : align protein 1 and rigid bodily "drag" protein 2 along, "complexOnComplex": align complex on complex, "2bcomplex": first align alpha whole to create the complex, then align the complex again based on complexOnComplex, "2bindividual": first align alpha whole to create the complex, then align the complex again based on beta of the reference segment for whole or interface
  Example: self.align=''2bindividual''

- **self.complexRMSDreduction***: The setup of the normal mode array for data gathering, for **self.investigationsOn** set to "Individual", choose "2k", for investigations on "Complex", choose "2k" for a cNMA setup to be further specified in the parameter **self.whichCustomHC**, or choose "1k1k" or "1k1k6" for a conventional NMA setup, the latter enabling six rigid body modes for data gathering RMSD reduction purposes.
  Example: self.complexRMSDreduction=''1k1k6''

- **self.whichCustomHC**: HC setup, for **self.investigationsOn** set to "Individual", choose "HC_U1". For **self.investigationsOn** set to "Complex", choose "HC_U1" for allowing intermolecular interactions, "HC_0" for not allowing intermolecular interactions, "HC_06" for "HC_0" and 6 trival modes of the ligand as first modes to be used for RMSD reduction calculations, "HC_U1_1k1k" to calculate modes from "HC_U1", then split them into 1k1k component vectors.
  Example: self.whichCustomHC=''HC_U1''

- **self.filterPDB**: When parsing PDB files, filter by ProDys "protein" selection? Without this filtering (set as "None"), mismatches have occurred.
  Example: self.filterPDB=''protein''

- **self.whatAtomsToMatch***: What atoms are subject to the matching of chains, Possible values: "calpha", "bb" or "all".
  Example self.whatAtomsToMatch=''bb''

- **self.customH**: Create the modified Hessians or keep conventional/-canonical Hessians. If set to "False", only canonical NMA will be performed, if set to "True", the potential U1 will be used to involve

intermolecular interactions.
Example: self.customH=True

- **self.customHRdistance**: Cut-off distance D for intermolecular springs.
  Example: self.customHRdistance=8.0

- **self.customForceConstant** Force constant gamma for intermolecular springs.
  Example: self.customForceConstant=0.25

- **self.whichCustomHIndividual**: Method to obtain the normal modes for the RMSD reduction on **self.investigationsOn** set to "Individual", the valid settings are: "HC_subvector": subvector method, "submatrix" : submatrix method, "canonical": use normal modes from the canonical individual protein.
  Example: self.whichCustomHIndividual=''submatrix''

- **self.projectHessian**: Use a transformation/projection technique on the Hessian (projection matrix treadment 8.27 NMA book or Non-Eckart body Frame Fuchigami U matrix) is "True", else do not use such a technique.
  Example: self.projectHessian=True

- **self.projectionStyle**: If **self.projectHessian** is "True", define the style of the transformation/projection. Valid settings are "full": project away from protein1 part, "fullComplex": project away from the whole complex, "fixedDomainFrame": use the U transformation matrix to setup the receptor as fixed domain.
  Example: self.projectionStyle=''full''

- **self.rescaleEigenvalues**: Use a re-ranking of eigenvalues, known as the $lambda^R$ approach. If set to "True", eigenvalues are re-ranked via $lambda^R$, if set to "False", no re-ranking takes place.
  Example: self.rescaleEigenvalues=False

- **self.floatingPointThreshold**: Small value to consider skipping the protein investigation if the initial RMSD is not bigger than it.
  Example: self.floatingPointThreshold=0.000000000001

- **self.stopRMSDReductionAt***: Set the number of maximal normal modes, at which the RMSD reduction based on the Swarmdock betas approach is stopped. To not have any limit/stop, set this to a high

value.
Example: self.stopRMSDReductionAt=400

- **self.maxModesToCalculate**: Upper limit for mode calculation, set to very high number (1000000) to calculate 3n-6 modes. For **self.investigationsOn** set to "Individual", this is the number of modes (excluding trivial modes) to be calculated. For **self.investigationsOn** set to "Complex", this is the value k, meaning that 2k modes (excluding trivial modes) will be obtained for the complex.
  Example: self.maxModesToCalculate=400

- **self.precisionBetaFitting***: Precision for RMSD beta fitting.
  Example: self.precisionBetaFitting=1e-6

- **self.maxIterBetas***: How many iterations are to be allowed for the betas fitter (Swarmdock approach).
  Example: self.maxIterBetas=60000

Note that **self.customHRdistance**, **self.customForceConstant**, and **self.maxModesToCalculate** are crucial parameters for NMA. In configuration file, key words have been set for these parameters for convenience.

3 Data Dictionary

The outputed files (here in **boldface**) inside the path outputPath+experimentNamePrefix+experimentName are as follows. Files with star are exclusively generated by posterior analysis in Example 3 with bound structure data provided.

- **1A2K_r_uanms.nmd.tar.bz2**: The generated nmd files to be viewed by VMD.

- **Configuration.py**: The configuration file used for this experiment, the name will differ depending on the values specified in experimentNamePrefix variable in the Configuration file.

- **combined.txt**: A textfile combining all .txt files columnwise. Note: The generation of this file relies on the "columns" and "paste" bash shell scripts, which gives currently incorrect column indention on dao (which has older versions of these commands), but works OK on a up to date Linux machine.

- **eigenvaluesReference.txt**: The eigenvalues of the protein under investigation, protein1_A.
- **eigenvectorsReference.txt**: The eigenvectors of the protein under investigation, protein1_A.
- **numberOfModes.txt**: Number of modes for the protein1_A.
- **RMSDPrediction.txt**: The predicted RMSD for protein1_A. Model was trained by kernel ridge regression for 3,370 inputs within 10Åcomplex RMSD.
- **pdbName.txt**: Name of the input protein.
- **pdbs.tar.bz2**: All pdbs of the proteins, the constructed complexes with segments R* and L* and fixed chain names, and the chainmatched pdbs are included. Furthermore, there is a folder deformationsnapshots which includes a pdb output after every application of a RMSD reduction via an increasing number of modes.
- **profiler.pro.tar.bz2**: Output of the python profiler, if the NMAUnified program was run with the –profile flag. To visualize, use the visualizePstats.sh in helperScrips or run:

```
# $ run gprof2dot -f pstats profiler.pro | dot -Tpng -o gprof2dot_output.png
## OR with the RunSnakeRun visualizer
# $ runsnake profiler.pro
```

- **profiler.txt**: Text representation generated from the python profiler, includes the cumulative time of the most computational taxing methods during the experiment run.
- **collectivityArrayInterface.txt***: Array of mode collectivities on the interface atoms sorted in ascending order of the eigenvalue, includes trivial normal modes.
- **correlationArrayWhole.txt***: Array of mode correlations towards the true deformation vector sorted in ascending order of the eigenvalue, includes trivial normal modes.
- **correlationArrayInterface.txt***: Array of mode correlations of interface atoms towards the true interface deformation vector sorted in ascending order of the eigenvalue, includes trivial normal modes.

- **cumulOverlapWholePrody.txt***: The cumulative overlap as generated by ProDys calcCumulOverlap function
- **cumulOverlapWholePrody.txt***: The cumulative overlap for the interface as generated by ProDys calcCumulOverlap function
- **eigenvaluesComplex.txt***: The eigenvalues of the complex created by protein1_A and protein2_A
- **I_rms_after_align.txt***: The I_RMSD of the unbound to bound complex interfaces after superposition
- **L_RMSD_unbound_to_superposed_bound.txt***: For investigations on individual proteins, the content equals RMSD_unbound_to_superposed_bound.txt or RMSD_interface.txt (depending on which RMSD reduction has been calculated)
- **L_RMSReductions.txt***: For investigations on individual proteins, the content equals RMSDReductionsWhole.txt or RMSDReductionsInterface.txt (depending on which RMSD reduction has been calculated)
- **L_rms.txt***: The L_RMSD of unbound to bound ligand of the complex, after superposition of unbound receptor to bound receptor parts of the complex
- **numberOfModesComplex.txt***: Number of normal modes calculated on the complex
- **overlapArrayWhole.txt***: Array of mode correlations towards the true deformation vector sorted in ascending order of the eigenvalue, includes trivial normal modes.
- **collectivityArrayWhole.txt***: Array of mode collectivities sorted in ascending order of the eigenvalue, includes trivial normal modes.
- **overlapTApproxWhole.txt***: Overlaps of the approximated deformation vector on the whole protein by the Swarmdock approach, can be used as true cumulative Overlap
- **overlapTApproxInterface.txt***: Overlaps of the approximated deformation vector on the interface by the Swarmdock approach, can be used as true cumulative Overlap

- **RMSDReductionsWhole.txt***: The RMSD reductions via the Swarmdock approach on the whole protein.
- **RMSD_interface.txt**: The initial RMSD on the interface of the proteins
- **RMSDReductionsInterface.txt***: The RMSD reductions via the Swarmdock approach on the interface.
- **RMSD_unbound_to_superposed_bound.txt***: The initial RMSD of the whole proteins
- **singleModeOverlapsFromSuperset.txt***: Overlaps of the modes that the superset given to the Swarmdock RMSD reduction algorithm is comprised of.
- **stepPointsReductionWhole.txt***: A list of the number of normal modes which have been successively used for the Swarmdock based RMSD reduction. They can be viewed as corresponding to x-axis entries in RMSD reduction curves, where the y-axis would be RMSDReductionsWhole.txt
- **zeroEigvecsComplex.dat***: Temporary file used for the bash script inside NMAUnified to include the trivial normal modes into the nmd files.
- **zeroEigvecsProtein1.dat***: Temporary file used for the bash script inside NMAUnified to include the trivial normal modes into the nmd files.
- **zeroEigvecsProtein2.dat***: Temporary file used for the bash script inside NMAUnified to include the trivial normal modes into the nmd files.