In this tutorial we will go over some of the functions of the NCIfolding pipeline. We will explain the different steps it follows, its requirements to run and the files it intermediate and output files it generates.

To carry out this tutorial we will utilize a short simulation we have carried out on chignolin with the sequence TYR-TYR-ASP-PRO-GLU-THR-GLY THR-TRP-TYR. We employed the CHARMM22* forcefield, TIP3P water model, Nose-Hoover thermostat set to 300K and Parrinello-Rahman pressure coupling. We utilized

Our analysis will require: the MD to analyze, a file with its topology, a reference native structure and a settings file to guide our pipeline, we make all these files available with this tutorial.

- traj1_cln025.xtc (our simulation)
- first_frame.pdb (a frame to provide topology)
- charm22star_chignolin.pdb (native structure)
- relevant_data.txt (setting for the pipeline)

The different scripts we will introduce in this tutorial are all accessible to anyone and are exhaustively commented to help users in the case they need to modify any part of the pipeline to adopt it to their needs or goals. Additionally, some already modified scripts are included to exemplify this and simplify the tutorial.

1. Native contact analysis

Firstly, we want to differentiate the state of the protein in each frame, to both characterize the transitions and study independently the different ensembles (Folded, Transition and Unfolded). To that end, we utilize the fraction of native contact as described by Lindorff-Larsen et al. in their Science 2011 article.

Considering this approach exclusively considers contacts between residues seven or more positions apart in the protein sequence, it will exclusively discern when the two ends of the chignolin are close together, as in the native state, or apart.

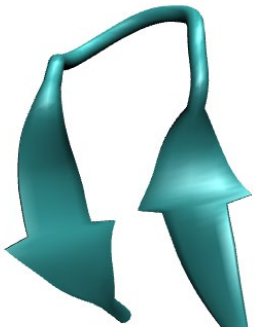

Figure 1. Alphafold generated chignoling native structure

However, as we will see, this can be enough to characterize folding and unfolding events in our system.

In every directory we work with this pipeline we will have a relevant_data.txt file which feeds the python script necessary data for that step. In this first step, for the Q analysis the relevant_data.txt contains the following data:

```
[Settings]
header = cln025
trajectory_directory = .
topology_file = first_frame.pdb
native_state_file = charmm22star_chignolin.pdb
waters_in_traj = False
```

The header is a string that can be changed freely and will be at the start of all the files generated in this pipeline to discern them from others.

The waters_in_traj states if we want to include protein-water interactions, in this case we set it as false since we will be working exclusively on protein-protein interactions.

The topology_file refers to the file used to define the structure in the simulation.

The native_state_file contains the native structure to compare analyzed structures.

The trajectory_directory refers to where the trajectory file or files are. This parameter is included because this pipeline was developed to analyze the ultra-long trajectories by Lindorff-Larsen et al., which are divided in numerous dcd files. The script as is prepared can track the Q values along the whole trajectory and extract transitions even if they are spread along multiple dcd files.

However, this is a different case, with only one trajectory and in xtc format. The original script works under the assumption that simulation fragments present a number in their name indexing them and loops over them. Instead, we will utilize a simplified version of this script to analyze only one xtc file (Q_analysis_1xtc.py).

Hence, to start the analysis we move to the NCIfolding_tutorial directory and run:

```
python3  Q_analysis_1xtc.py
```

Running it may take a minute, but it should produce:

- cln025_frame_vs_q_scatter_plot.png
- cln025_Q_analysis.txt
- transition_pathways_for_cln025 (plain text file)

The image should look something like this:

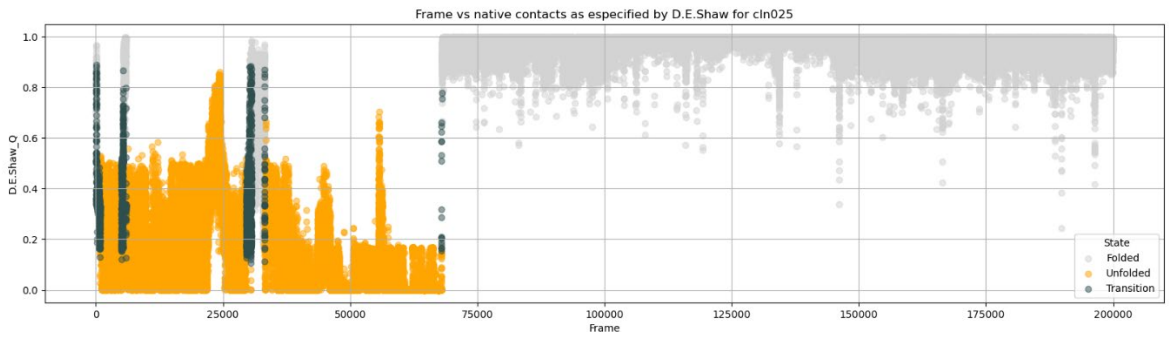

Figure 2. Frame vs native contact fraction (Q)

This shows the protein starting unfolded and adopting a folded conformation briefly at the very start of the simulation. Two similar events take place later with the protein adopting a native-like structure transiently. After this the protein is folded and does not unfold again.

We can also see various events in which the protein adopts intermediate values but recrosses back to its original state, avoiding a transition. The cln025_Q_analysis.txt that the following script utilizes shows further details into the data for each frame:

Frame	D.E.Shaw_Q	State	Traj_File	Frame_in_file
0	0.45367634	Folded	.\traj1_cln025.xtc	0
1	0.86228746	Folded	.\traj1_cln025.xtc	1
2	0.81724733	Folded	.\traj1_cln025.xtc	2
…				
53	0.96307206	Folded	.\traj1_cln025.xtc	53
54	0.7784545	Transition	.\traj1_cln025.xtc	54
55	0.8156615	Transition	.\traj1_cln025.xtc	55
…				
809	0.16554129	Transition	.\traj1_cln025.xtc	809
810	0.054167792	Unfolded	.\traj1_cln025.xtc	810
811	0.16738538	Unfolded	.\traj1_cln025.xtc	811

While the simulation started in an intermediate state, then it turned to folded, and as such the script deemed the first frames as also part of the folded ensemble, only characterizing as transitions pathways from folded (Q>0.9) to unfolded (Q<0.1) or vice versa.

2. Ensemble analysis

Now that we have not only the Q values, but also assigned configuration to each frame, we can sample the entire simulation to prepare ensembles:

```
python3 extract_ensembles.py
```

This will generate three directories:

- cln025_Ensemble_Folded
- cln025_Ensemble_Transition
- cln025_Ensemble_Unfolded

And in each of these we will find 100 random frames as an xtc with the first frame as a pdb to use as topology. And a relevant_data.txt file for subsequent steps.

The studied ensemble and number can be easily modified, by changing `ensemble_list = ['Folded', 'Transition', 'Unfolded']` and `n_snapshots = 100` at the start of the script.

Using the folded ensemble as example, the first frame among the selected frames to generate cln025_ensemble_Folded.xtc is cln025_fr83830.pdb (meaning frame 83830)

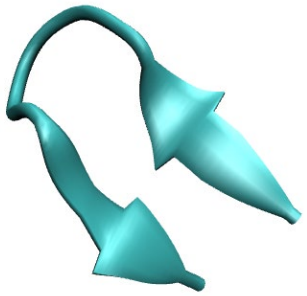

Figure 3. Frame 83830 in our example simulation (Folded)

The relevant_data.txt file in this directory is set so from this directory we can directly run:

```
python3 prepare_ncis_by_residue.py
```

This will generate a NCI_data directory with the xyz for each residue and the NCIPLOT inputs for all their possible combinations for each of the frames, e.g.:

cln025_ensemble_Folded_fr000_1TYR.xyz

cln025_ensemble_Folded_fr000_10TYR.xyz

cln025_ensemble_Folded_fr000_1TYR_vs_10TYR.inp

The NCIPLOT utilizes the keywords set in the previous script, which when used as default are:

```
2
cln025_ensemble_Folded_fr000_1TYR.xyz
cln025_ensemble_Folded_fr000_10TYR.xyz
OUTPUT 1
COARSE
INCREMENTS 0.1 0.1 0.1
CUTPLOT 0.05 0.5
CUTOFFS 0.5 1.0
INTERMOLECULAR
INTERCUT 0.85 0.75
INTEGRATE
ONAME cln025_ensemble_Folded_fr000_1TYR_vs_10TYR
RANGE 3
-0.2 -0.02
-0.02 0.02
0.02 0.2
```

Additionally, the script produces a make_ncis.sh that will run all NCIPLOT calculations as long as the program is already installed in your machine and its executable (nciplot) in your path.

```
./make_ncis.sh
```

This might take a couple of hours depending on the computing power of your machine. And it will generate a out and dat file for each calculation, with only the former being used for our ends.

Inside output files like cln025_ensemble_Folded_fr000_1TYR_vs_10TYR.out, we will find the density integrals our pipeline is based around:

…

```
            RANGE INTEGRATION DATA
-----------------------------------------------------------------------


-----------------------------------------------------------------------
 Interval        :      -0.20000000      -0.02000000


-----------------------------------------------------------------------
 Integration  over the volumes of rho^n
-----------------------------------------------------------------------
 n=1.0           :       0.27287409
 n=1.5           :       0.05137665
 n=2.0           :       0.00991772
```

…

This pipeline exclusively uses n=1.0 data, but there could be application to other generated results.

Following that we can compile the NCIPLOT outputs from the Folded Ensemble directory by running:

```
python3 join_nci_densities.py
```

This will generate a NCI_csvs directory with a csv file for each considered residue pair. For example cln025_ensemble_Folded_1TYR_vs_10TYR_densities.csv:

```
Frame,Attractive n^1,VdW n^1,Repulsive n^1,SG Attractive n^1,SG VdW n^1,SG Repulsive n^1
0,0.27287409,0.40678404,0.03131867,0.24833599399999995,0.558442107,0.041769139
1,0.02219579,0.48942367,0.0,0.24833599399999995,0.558442107,0.041769139
2,0.33255184,0.33718876,0.08499736,0.24833599399999995,0.558442107,0.041769139
```

…

```
98,0.43340383,0.58989452,0.11966645,0.354315794,0.5173432269999999,0.0948537279999999
99,0.32071008,0.249616,0.05895085,0.354315794,0.5173432269999999,0.09485372799999998
```

For each frame it contains the integral densities (attracttive, van der Waals and repulsive) for this pair as well as the smoothed integrals (using Savitsky-Golay for a 10 frame window).

From this point we can use:

```
python3 square_graphs_from_csvs.py
```

Generating the Square_Graphs directory with:

- cln025_ensemble_Folded_Interactions_Folded_ensemble.csv
- cln025_ensemble_Folded_Interaction_Matrix_of_Folded_ensemble.png

The csv file contains the average van der Waals and Attractive minus Repulsive density integrals:

```
Residue1,Residue2,Frame range,VdW,Att - Rep
1TYR,10TYR,None - None,0.4757784544,0.2226418728
1TYR,4PRO,None - None,0.0050812613,0.0
1TYR,5GLU,None - None,3.0689999999999998e-06,0.0
```

…

Since we are not considering specific trajectory segments, but an average of the whole ensemble, in this case Frame range states none to none.

The resulting contact maps for the three different ensembles are:

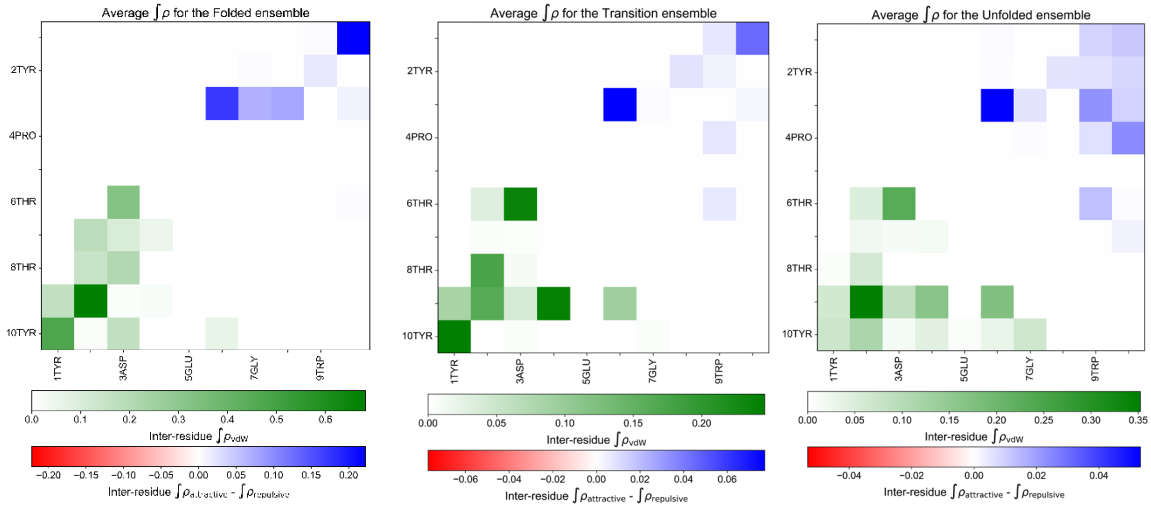

Figure 4. NCI contact maps for the Folded, Transition and Unfolded ensembles

Consider that the colorbar limits are set independently unless we set them ourselves to match with:

```
#global_max_att_minus_rep = 0.19
#global_max_vdw = 0.49
```

For the purposes of this tutorial feel free to peruse the code and test changing parts of the code, such as the number of snapshots to extract or the colorbar limits. We have tried to exhaustively comment the code so that is easy to read and modify for the porpoises of any potential user.

Knowing that our simulation has 135267 folded, 2115 transition and 62619 unfolded frames based on our Q analysis, we can take 66 folded, 1 transition and 33 unfolded data points to generate the number of h bonds (Baker-Hubbard) vs attractive density graph. With the following script in the starting directory:

```
python3 H-bonds_vs_attractive_density_scatter_plot_only_ensembles.py
```

Originally this script also used the data from transitions, but in this case we have simplified the code so it can run without having to run all the transition calculations. However in all cases we have to set the ensemble_weights.

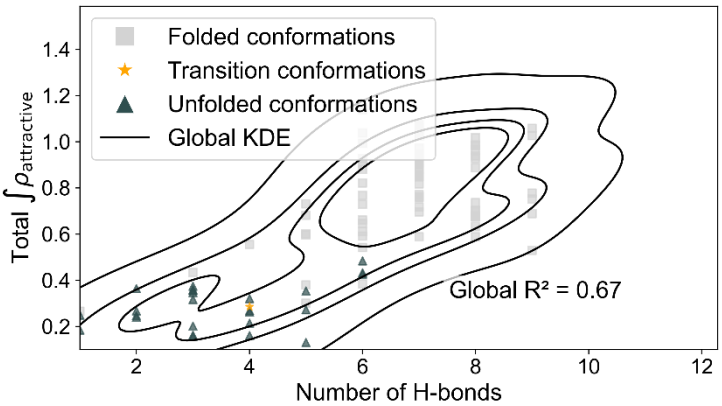

Figure 5 Number of H-bonds vs the sum of attractive density for sampled geometries from each ensemble

3. Transition analysis

With the Q analysis ready, we can also extract every characterized transition for their detailed analysis. The extract_transitions.py script would usually go over the Q analysis results finding the data corresponding to transitions from one or multiple dcd files and making a separate copy ready for its analysis. Since in this case we are working with an xtc file we will use a slightly modified version of this script:

```
python3  extract_transitions_1xtc.py
```

This script generates a separate directory for each transition with the trajectory, topology and its relevant_data.txt file.

A summary of the transitions can be found in the plain text file transition_pathways_for_cln025:

```
Transition cln025_unfolding_event_1:
   Start Frame: -16 (.\traj1_cln025.xtc, Frame in File: 0)
   End Frame: 879 (.\traj1_cln025.xtc, Frame in File: 879)

Transition cln025_folding_event_1:
   Start Frame: 4983 (.\traj1_cln025.xtc, Frame in File: 4983)
   End Frame: 5413 (.\traj1_cln025.xtc, Frame in File: 5413)
```

…

```
Transition cln025_folding_event_4:
   Start Frame: 67967 (.\traj1_cln025.xtc, Frame in File: 67967)
   End Frame: 68005 (.\traj1_cln025.xtc, Frame in File: 68005)
```

Our methodology selects a sampling step that results in the transition being represented by ~100 snapshots. Using that same sampling step it takes 10 additional frames (set as extra_frames in the script) at the start and end of the transition to better represent the transition and reduce the loss of relevant data due to the 10-frame smoothing window. This has caused the start of the very early transition to go into negative frames, in this case we will just start at 0.

Taking as an example that first unfolding event in directory cln025_folding_event_1 we will find:

- cln025_folding_event_1.xtc
- cln025_folding_event_1_frame0.pdb
- relevant_data.txt

The relevant_data.txt for this script is:

```
[Settings]
header = cln025_folding_event_1
topology_file = ./cln025_folding_event_1_frame0.pdb
native_state_file = ../charmm22star_chignolin.pdb
trajectory_file = ./cln025_folding_event_1.xtc
Start Frame: 4983
End Frame: 5413
Sampling step: 3
```

This file contains the starting and ending frame for the transition (plus extra frames at start and end) we have defined. The sampling step shows that we will take one in every 3 frames to go over the transition meaning that the 431 frames of the complete trajectory will be represented by 144 frames we will study with NCIPLOT.

Similar to the previous case we perform the individual calculations by:

```
python3 prepare_ncis_by_residue.py

cd NCI_data

./make_ncis.sh

cd ..

python3 join_nci_densities.py
```

As a note if the last script outputs an error like:

```
Error in file ./NCI_data/cln025_folding_event_1_fr5250_3ASP_vs_10TYR.out
```

This is indicative of that individual calculation ending abruptly and not producing results to parse. In that case we can repeat that calculation by going into NCI_data:

```
nciplot cln025_folding_event_1_fr5250_3ASP_vs_10TYR.inp >
cln025_folding_event_1_fr5250_3ASP_vs_10TYR.out
```

When correctly run will generate an equivalent NCI_csvs. With these files we can check the progress of the interaction densities during the folding event. A good way of visualizing it can be:

```
python3 distances_vs_NCIdata.py
```

The are two ways to use this script, either setting a csv to check in target_filename in line 13 to check that pair or setting it to None to see all.

This way we can generate images as the following in the NCI_csvs directory:

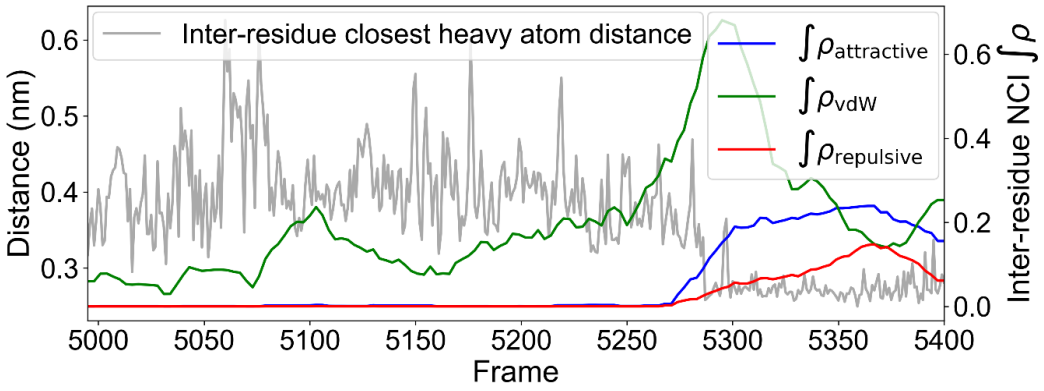

Figure 6 Interaction densities and distances between 1TYR and 10TYR for folding event 1

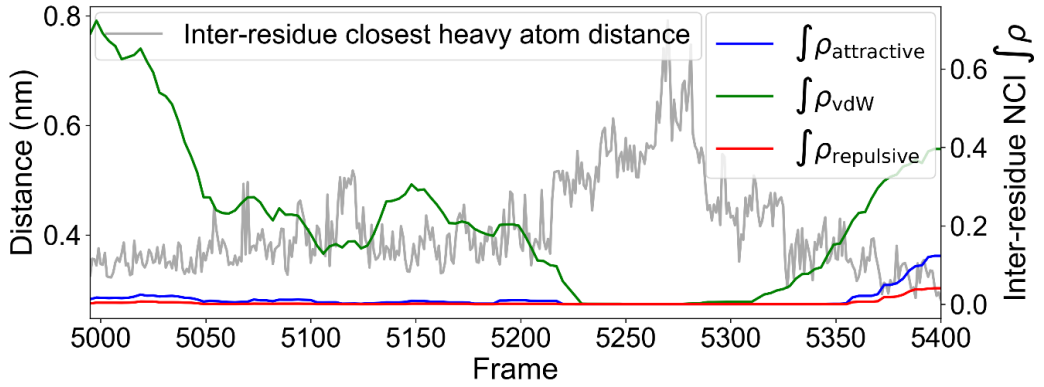

Figure 7 Interaction densities and distances between 2TYR and 9TRP for folding event 1

We can also run:

```
python3 square_graphs_from_csvs.py
```

Generating square graphs of the contacts in different sections of this transition event:

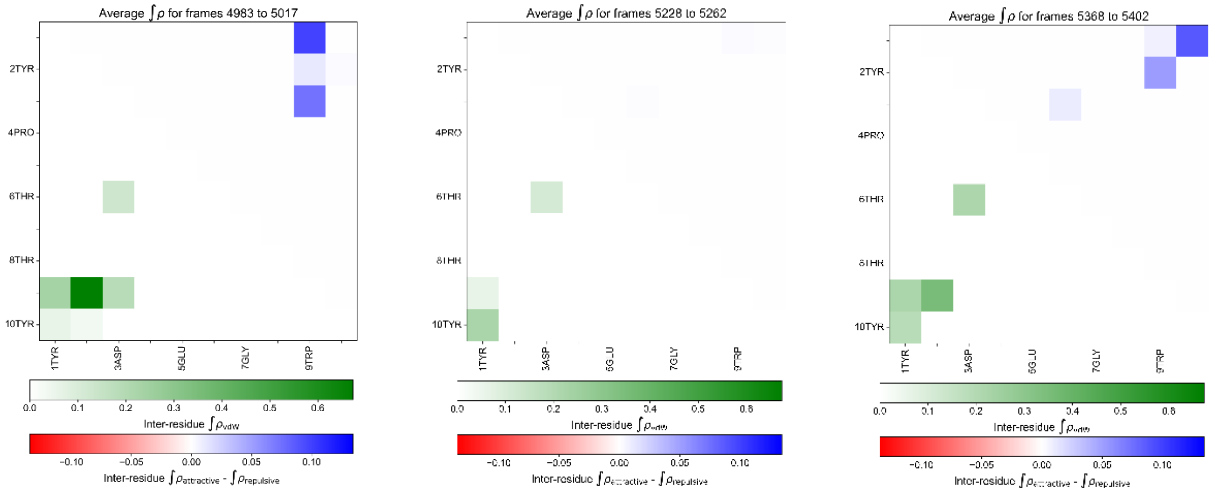

Figure 8. NCI contact maps for different points during the folding event

4. Clustering pair contacts

With the csv for the folding event generated we can turn to the clustering analysis. We run the script from the starting directory, and it goes over folding/unfolding directories to parse the csv data and cluster it. At this point we can either delete the 7 transitions we have not calculated the interaction densities for, or calculate the rest.

To cluster the residue pairs based on the csv data we run:

```
python3 Agglomerative_tsMSM_separate_traj_clustering.py
```

The script should output the following lines:

```
.\cln025_folding_event_1\NCI_csvs  event length: 144 NCI data length: 144
.\folding event 1  :  (28, 3, 144)
distance matrices (28, 28)
```

For each directory it will print the number of sampled frames and the NCI data points (as a sanity check). In the next line it prints (the number of studied residue pairs, the different types of densities we are separately considering, the number of analyzed snapshots). Lastly it prints the size of the distance matrix used for the clustering. Since we are clustering 28 different residue pairs (separated by more than 3 residues) this matrix is 28x28 and it is obtained as the average of the distance matrix obtained for each individual transition.

The output is generated in Clustering_results in the subdirectory

Agglomerative_tsMSM_separate_event_Clustering_cost1_window0p5:

It plots the contents of the clusters using from 2 up to n_clusters (variable we set).

E.g.: Agglomerative_tsMSM_separate_event_clustering_n2_cost1_window0p5.png

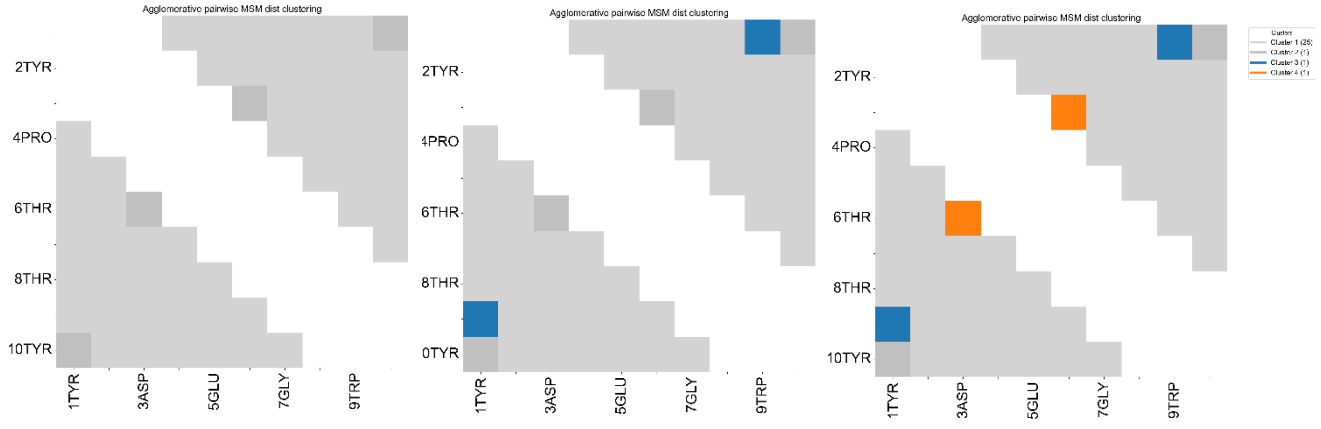

Figure 9. Residue pair clustering using 2, 3 and 4 clusters for the first folding event

Additionally, the script generates time series of plots of densities and native contact fractions for every event (in this case only one) in separate directories such as

TimeSeries_Plots_n3_agglomerative_ts_pairwise_separate_event, containing:

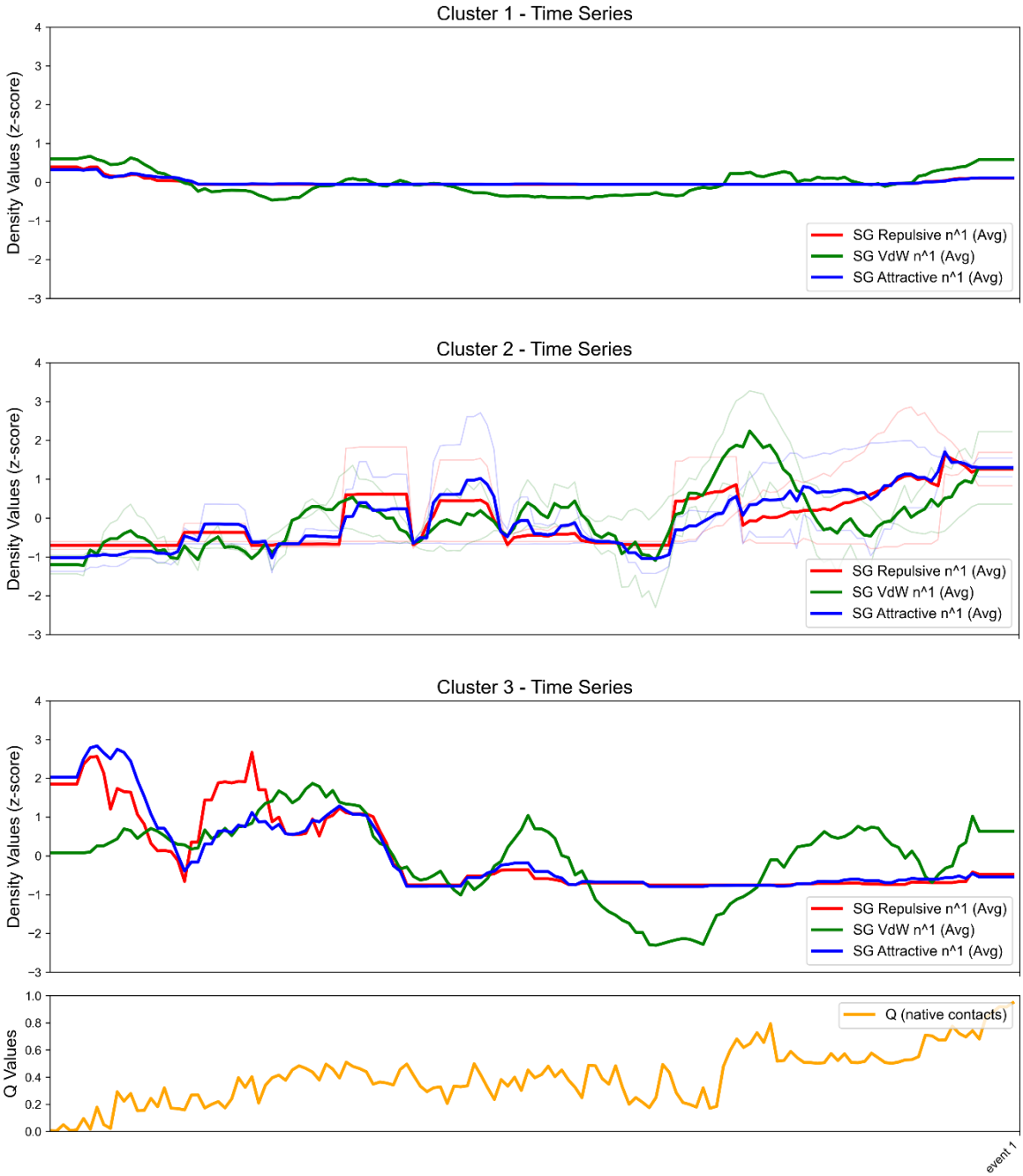

Figure 10. Cropped time series plots for every cluster when using three

It plots the average density for every cluster and the individual densities for every residue pair outside of Cluster 1, which we define as noise.

Additionally, the script generates a series of test to check the quality of clustering so we can check on how well defined the clusters are and the ideality of the methodology or number of clusters used.

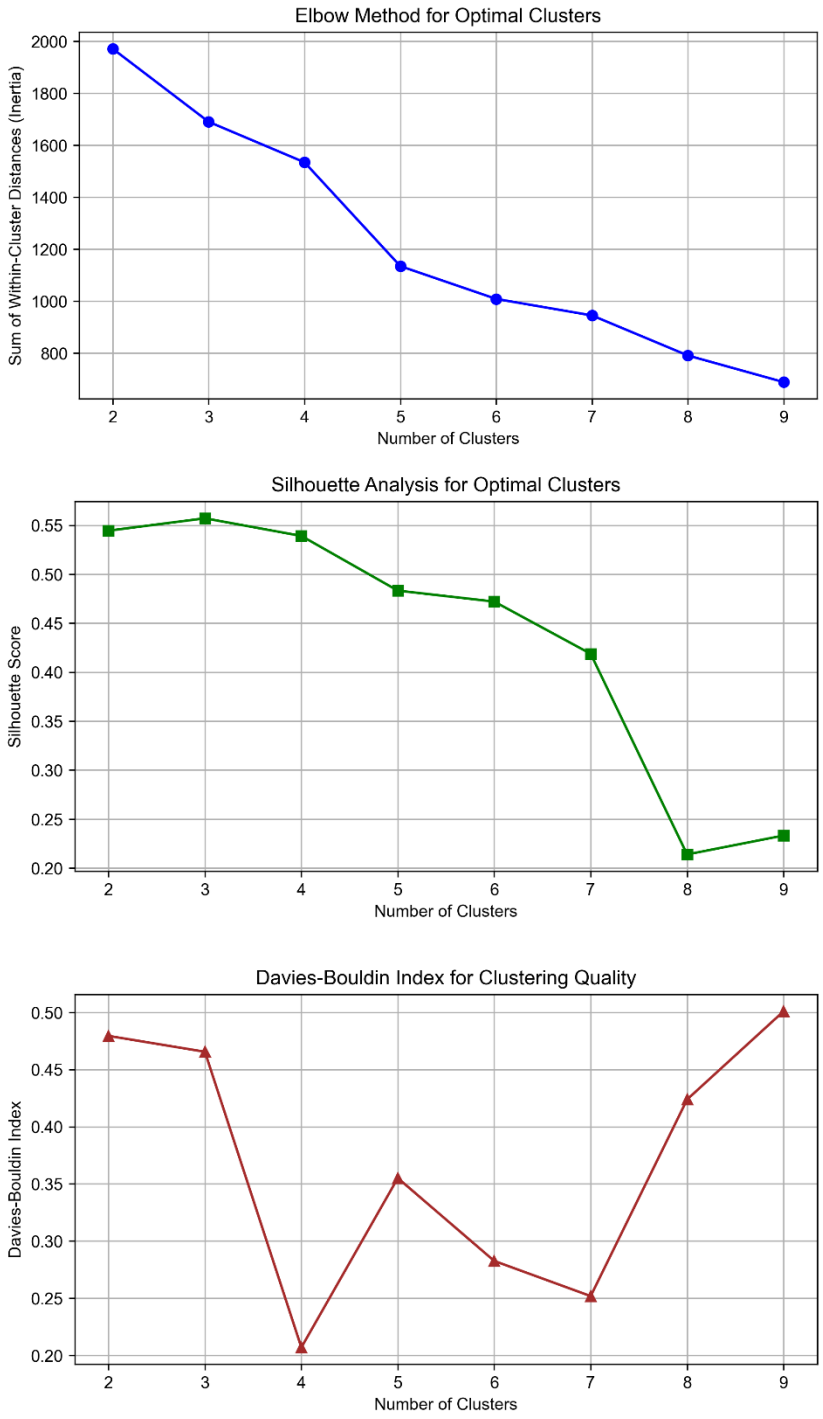

Figure 11. Elbow method, silhouette analysis and Davies-Bouldin index for every tested number of clusters

Keep in mind these are just some of the simplest indicators of cluster quality. They may not be the most suited for your system or goals. Testing other analyses and supporting your conclusions on other available information on the system is a valuable resource.

Another tool the script provides is the PCA and tSNE analyses:

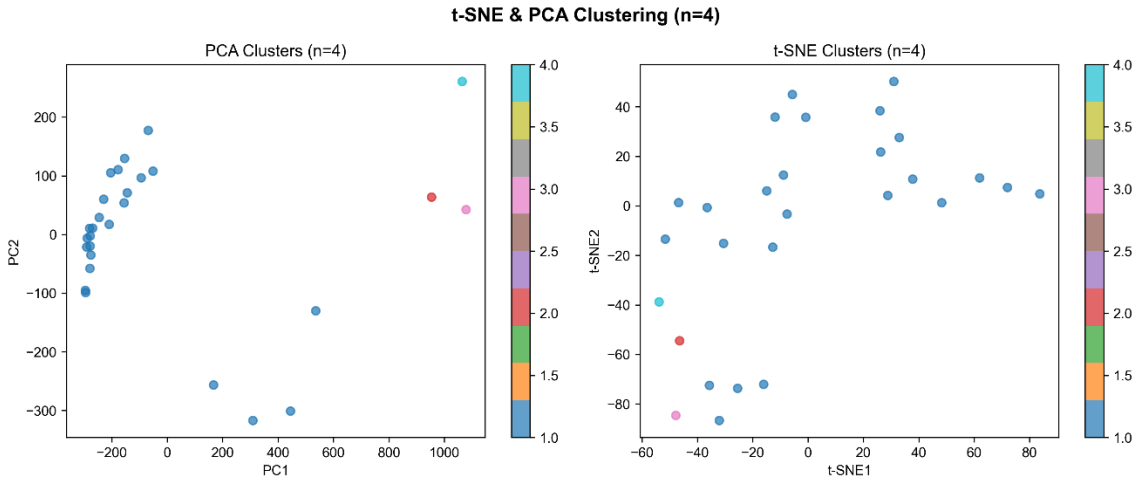

Figure 12. PCA and tSNE analyses using 4 clusters

Lastly, it generates a plain text file showing the residue pairs in each cluster, in this case res_pairs_tsmsm_penalty1_window0p5.txt

```
Results for n=2
Cluster 2: [[0, 9], [2, 5]]

Results for n=3
Cluster 2: [[0, 9], [2, 5]]
Cluster 3: [[0, 8]]

Results for n=4
Cluster 2: [[0, 9]]
Cluster 3: [[0, 8]]
Cluster 4: [[2, 5]]
```

…

Last thing to mention is that at the start of the script we set the parameters relevant to the clustering so they can be easily changed to test using different ones or with a little more work if they require different variables a different clustering methodology.

```
n_clusters = 9 # maximum number of clusters to try
msm_penalty = 1
msm_window = 0.5 # meaning 50% of the trajectory is considered for time warping
```

Changing the two last variables will generate a new subdirectory with a different name to easily test and compare the result for other values. As an example, if we

increased the penalty used to 2.0, then the new results would be generated in the agglomerative_tsMSM_separate_event_Clustering_cost2_window0p5 subdirectory.

Having defined the residue pair clusters, we could now turn to clustering transitions based on the cumulative densities in each of these clusters (minus the noise one). For that objective we will utilize the trajectory_clustering_DTW.py script. However, first we must include in it the cluster definitions we have selected in lines 20-46.

Using the data from res_pairs_tsmsm_penalty1_window0p5.txt we will modify those lines in trajectory_clustering_DTW.py:

```
# Parameteres used for DTW clustering
n_clusters_range = range(2, 7)
bounding_window = 0.5

# Density data used for the clustering
density_types = ['SG Attractive n^1', 'SG VdW n^1', 'SG Repulsive n^1']

# Define residue pairs of each structural elements to track; these lines were taken from a res_pairs_tsmsm_xxx.txt
structural_elements = {
    "Cluster 2": [[0, 9]],
    "Cluster 3": [[0, 8]],
    "Cluster 4": [[2, 5]],
    }

# Define colors for structural elements, we match them to the colors assigned by the residue pair clustering
structural_colors = {
    "Cluster 2": "blue",
    "Cluster 3": "green",
    "Cluster 4": "orange",
    }

title_colors = {
    'SG Attractive n^1': 'blue',
    'SG VdW n^1': 'green',
    'SG Repulsive n^1': 'red'}
```

We have utilized these modifications on the script to cluster the 8 transitions in our test case trajectory, obtaining the following results:

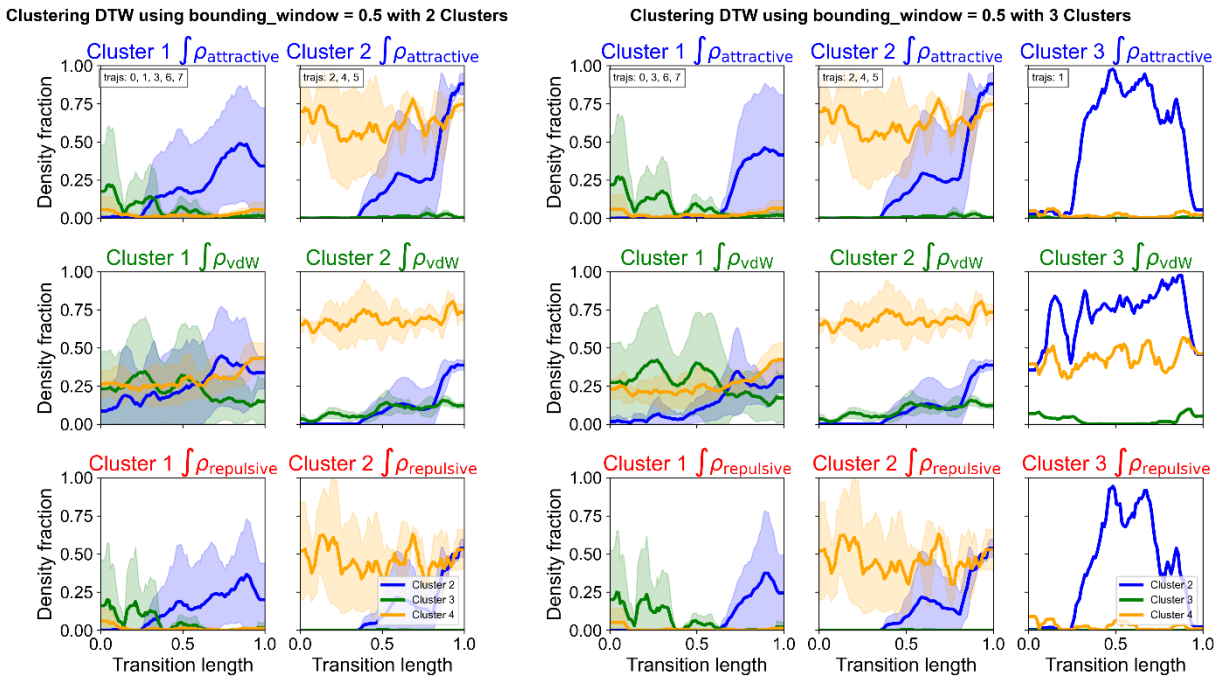

Figure 13. Trajectory clustering (for n_cluster = 2 and 3) based on the interaction densities of the selected residue pair clusters

Additionally, this script generates the same cluster quality analyses (elbow, silhouette, Davis-Bouldin, PCA and tSNE).

We hope this tutorial helps you in case any errors or questions appear in the utilization of this pipeline. The pipeline has been designed so it allows for easy modifications to allow for tests and fit the requirements of different systems with minimal modifications. All the code is exhaustively commented on aiding in its comprehension and on making changing it as straightforward as possible.

We must remark that the example we have worked with is a considerably short simulation, meaning is likely a faulty representation of its behavior where we have applied the same methodologies that yielded accurate results for a larger more structured protein. This analysis is only intended as an example of the different steps and capabilities of the pipeline and likely the results we have obtained are not accurate nor valuable.