Network Analysis Manual

Overview

This analysis investigates how protein dimerization affects internal communication networks by comparing dynamic correlations between a GPCR in its monomeric and dimeric states. The workflow identifies how protein-protein interactions rewire allosteric pathways and alter conformational information flow within individual subunits.

Note: This code is based on tutorials provided by the **mdigest** package available at: https://github.com/fmaschietto/mdigest

System Requirements

The analysis requires the mdigest package with all correlation modules installed. You will need MD trajectory files for three systems: a monomer with its topology and trajectory files, and two separate files for dimer subunit A and dimer subunit B, each with their respective topology and trajectory data.

Workflow Steps

1. System Preparation

The first step involves setting up file paths for all three systems. You define the topology files (.gro format) and trajectory files (.xtc format) for the monomer, dimer subunit A, and dimer subunit B. The code then initializes separate MDS objects for each system to handle the molecular dynamics data independently.

```
python
# Define file paths
topology_M = 'mono_6cm4.gro'
trajectory_M = ['mono_6cm4.xtc']
# Similar for dimer A and B systems


# Initialize MDS objects for each system
mdsM = MDS()  # Monomer
mdsDA = MDS() # Dimer A
mdsDB = MDS() # Dimer B
```

Each system undergoes identical preparation steps. The trajectories are loaded into their respective MDS objects, then aligned using alpha-carbon atoms to remove translational and rotational motion. Protein selections are defined to focus the analysis on relevant atoms, and trajectories are strided to reduce computational load by analyzing every 5th frame while maintaining sufficient statistical sampling.

2. Multi-Modal Correlation Analysis

The analysis employs three complementary approaches to capture different aspects of protein dynamics and their correlations.

A. Dynamic Correlation Analysis

This method calculates correlations between alpha-carbon atom movements throughout the simulation. The analysis captures how residues move relative to each other in three-dimensional space, revealing coupled motions that indicate functional relationships.

```
python
dyncorrM = DynCorr(mdsM)
dyncorrM.parse_dynamics(scale=True, normalize=True,
                        LMI='gaussian', MI='knn_5_2', DCC=True, PCC=True)
```

The method applies multiple correlation metrics simultaneously. The Linear Mutual Information (LMI) with Gaussian assumption captures linear relationships, while the k-nearest neighbor Mutual Information (MI) approach detects non-linear correlations that traditional Pearson methods might miss.

B. Electrostatic Energy Correlation (Kabsch-Sander Analysis)

This approach analyzes correlations in backbone electrostatic interactions using the Kabsch-Sander method. It focuses on how hydrogen bonding patterns and electrostatic environments change in a correlated manner throughout the simulation.

```
python
KSM = KS_Energy(mdsM)
KSM.set_selection(['protein and backbone and name N', 'protein and backbone
and name O'])
KSM.KS_pipeline(MI='knn_5_2', covariance=True, correction=True)
```

The method selects backbone atoms involved in hydrogen bonding (nitrogen and oxygen atoms) and calculates how their electrostatic interaction energies fluctuate together. This provides insight into how secondary structure elements communicate through the hydrogen bonding network and how dimerization might alter these electrostatic communication pathways.

C. Dihedral Angle Correlation Analysis

This method examines correlations in backbone phi and psi dihedral angle fluctuations. Since these angles directly relate to local backbone conformation, their correlations reveal how conformational changes propagate through the protein structure.

```
python
dihdyncorrM = DihDynCorr(mdsM)
```

```
dihdyncorrM.parse_dih_dynamics(mean_center=True, LMI='gaussian',
MI='knn_5_2')
```

3. Save Data

All calculated correlation matrices and analysis objects are saved to disk for later use.

```
python
dyncorrM.save_class(file_name_root=cachedir + 'dyncorrM')
KSM.save_class(file_name_root=cachedir + 'KS_M', save_space=True)
```

4. Comparative Analysis

The core of the analysis involves systematic comparison between monomer and dimer states to identify how dimerization rewires internal communication networks.

Correlation Matrix Visualization

The code generates comprehensive heatmap visualizations that directly compare correlation patterns between different states. These matrices reveal the strength and sign of correlations between all residue pairs.

```
python
plots = Plots(dyncorrM_load, dyncorrDA_load, matrix_type='pcc',
              save=rd_A + '/mono-dima_ca_pcc')
plots.plot_gcc_per_replica()  # Creates 3-panel correlation matrix plot
```

The three-panel plot typically shows the monomer correlation matrix, the dimer correlation matrix, and their difference map. This visualization immediately highlights regions where correlations are strengthened, weakened, or reversed upon dimerization.

Network Centrality Analysis

Eigenvector centrality analysis identifies residues that are most important for network communication. The analysis computes how central each residue is to the overall correlation network and tracks how this centrality changes over time.

```
python
plots.plot_eigcent_per_replica()  # Creates centrality vs residue plot
```

This analysis reveals which residues serve as communication hubs and how dimerization affects their importance in the network. Residues with high centrality often correspond to functionally important sites such as allosteric regulators or active site residues.

5. Network Visualization

The final step transforms the calculated correlation matrices into interactive three-dimensional network visualizations using PyMOL. This process converts abstract numerical correlation data into intuitive visual representations that facilitate interpretation.

The visualization workflow begins by processing the correlation matrices for each system.

```
python
dist_M = np.mean([dyncorrM_load.distances_allreplicas['rep_%s' %i].copy() for i in range(1)], axis=0)
gcc_M = np.mean([dyncorrM_load.gcc_allreplicas['rep_%s' %i]['gcc_lmi'].copy() for i in range(1)], axis=0)
```

The ProcCorr class handles the conversion of correlation matrices into network-ready formats. This specialized class sources the molecular topology from the original MDS universe objects and generates representative PDB structures for visualization.

```
python
vizcorr = ProcCorr()
vizcorr.source_universe(mdsM.mda_u)
vizcorr.set_thresholds(prune_upon=np.asarray(dist_M.copy()), lower_thr=0, upper_thr=5.)
vizcorr.filter_by_distance(matrixtype='gcc_M', distmat=True)
```

The processed correlation data is normalized and converted into pandas DataFrames that contain all necessary information for network visualization. These DataFrames include residue identifiers, correlation strengths, and geometric information needed to draw connections in three-dimensional space. The final processed data is saved as pickle files for efficient loading into visualization software.

The PyMOL visualization system renders the correlation networks as colored connections between residue positions. Residues appear as nodes in the network, while correlations are represented as edges with colors and thicknesses that correspond to correlation strength and significance. This three-dimensional visualization approach provides immediate insight into how dimerization affects protein communication networks.