Table of Contents

[0. Introduction to ioNERDSS 3](#_Toc171937261)

[Description 3](#_Toc171937262)

[Section Descriptions 3](#_Toc171937263)

[Methods VS Functions 4](#_Toc171937264)

[Function Formatting: 4](#_Toc171937265)

[1. Creating NERDSS Inputs 5](#_Toc171937266)

[1.1 Structure of the NERDSS input files: 5](#_Toc171937267)

[1.2 Database PDB 5](#_Toc171937268)

[1.2.1 PDB\_UI (): 5](#_Toc171937269)

[1.2.2 MAIN OBJECT - Read in PDB File to create protein complex object: 7](#_Toc171937270)

[1.2.3 MODIFY - Calculate the angles between pairs of interfaces: 8](#_Toc171937271)

[1.2.4 MODIFY - Filter the protein complex object, so it only includes the chains you want: 8](#_Toc171937272)

[1.2.6 MODIFY - Normalize the COM of each chain: 9](#_Toc171937273)

[1.2.8 OUTPUT - Writes a new PDB file based on chain info: 10](#_Toc171937274)

[1.3 Platonic Solid Self-assembly Input File Writing: 12](#_Toc171937275)

[1.4 Repeated Protein Subunit Regularization: 13](#_Toc171937276)

[1.5 GUI – graphic interface to create .inp and .mol: 14](#_Toc171937277)

[2. Analyzing NERDSS Outputs: 17](#_Toc171937278)

[2.1 Analyzing Histogram Files (General Functions): 17](#_Toc171937279)

[2.1.1 MAIN OBJECT – The Main Single-component Histogram Object 17](#_Toc171937280)

[2.1.2 MAIN OBJECT – The Main Multi-Component Histogram Object 19](#_Toc171937281)

[2.1.3 DATAFRAME – stores count of each complex type for each timestep 20](#_Toc171937282)

[2.1.4 CSV – stores count of each complex type for each timestep 21](#_Toc171937283)

[2.1.5 LINE GRAPH – max count of protein species in a single complex at a time 22](#_Toc171937284)

[2.1.6 LINE GRAPH – mean count of protein species in a single complex at a time 24](#_Toc171937285)

[2.2 Analyzing Single-Component Histogram Files 26](#_Toc171937286)

[2.2.1 HISTOGRAM – average number of each complex species size: 26](#_Toc171937287)

[2.2.2 3D HISTOGRAM – relative occurrence of each species over time: 28](#_Toc171937288)

[2.2.4 HEATMAP - total count of monomers in each complex size vs. time: 32](#_Toc171937289)

[2.3 Analyzing Multi-Component Histogram Files 36](#_Toc171937290)

[2.3.1 HISTOGRAM – Frequency of each complex size: 36](#_Toc171937291)

[2.3.2 STACKED HISTOGRAM – Counts of complex species with certain protein compositions: 38](#_Toc171937292)

[2.3.3 HEATMAP – Average count of each complex composition over simulation time: 40](#_Toc171937293)

[2.3.4 3D HISTOGRAM - Average count of each complex composition over simulation time: 42](#_Toc171937294)

[2.3.5 LINE GRAPH – Fraction of monomers assembled over time: 44](#_Toc171937295)

[2.4 Analyzing Transition Matrix Files 45](#_Toc171937296)

[2.4.1 LINE PLOT - calculates change in free energy among different sizes of complexes: 45](#_Toc171937297)

[2.4.2 LINE PLOT – symmetric probability of association between complex sizes: 47](#_Toc171937298)

[2.4.3 LINE PLOT - asymmetric probability of association between complex sizes: 49](#_Toc171937299)

[2.4.4 LINE PLOT - symmetric probability of dissociation between complex sizes: 51](#_Toc171937300)

[2.4.5 LINE PLOT - asymmetric probability of dissociation between complex sizes: 53](#_Toc171937301)

[2.4.6 LINE PLOT – Growth probability for each complex size: 55](#_Toc171937302)

[2.4.7 LINE PLOT – Average lifetime for each complex type: 57](#_Toc171937303)

[2.5 Locating position for certain size of complexes by PDB/restart file 59](#_Toc171937304)

[2.5.1 By PDB file: 59](#_Toc171937305)

[2.5.2 By ‘restart.dat’ file: 60](#_Toc171937306)

[2.6 Analyzing .xyz files 61](#_Toc171937307)

[2.6.1 CSV – creates spreadsheet of protein locations 61](#_Toc171937308)

[2.6.2 DATAFRAME – creates dataframe of protein locations 62](#_Toc171937309)

[2.6.3 MATRIX - tracks the trajectory of specific protein(s) 63](#_Toc171937310)

[2.7 Analyzing .pdb files 64](#_Toc171937311)

[2.7.1 LINE PLOT- Auto correlation function for complexes 64](#_Toc171937312)

[2.8 Gag Sphere 67](#_Toc171937313)

[2.8.1 Reshape Gag 67](#_Toc171937314)

[2.8.2 Sphere Regularization Index 68](#_Toc171937315)

[3. Merging results from restart simulations 70](#_Toc171937316)

[3.1.1 Merging NERDSS restart simulations output files 70](#_Toc171937317)

[3.1.2 Merging NERDSS output files 71](#_Toc171937318)

# Introduction to ioNERDSS

### Description

ioNERDSS is a python library to create input files that are executable by the NERDSS simulation software and to analyze outputs generated by simulation trajectories. Input files can be generated from structures of macromolecular complexes, such as defined in Protein Data Bank (PDB) files or based on the idealized geometries of Platonic solids. The package is designed to improve the usability and quantitative interpretation of NERDSS simulations.

### Section Descriptions

This user guide is separated into 2 main sections.

**Creating NERDSS Inputs**: Automatically creates executable NERDSS inputs for you!

* Database PDB: This will create NERDSS inputs based on a Protein Databank (.pdb) file downloaded from the [RSCB PDB](https://www.rcsb.org/).
* Platonic Solid: This will create NERDSS input files based on the type of solid it will create when fully assembled. There are 2 options for each of the 5 platonic solids.

**Analyzing NERDSS Outputs:** Creates graphs, spreadsheets, and analyzed datasets from NERDSS outputs

* Histogram Section: Analyzes and outputs data from a histogram.dat file.
  + General: Functions that can be run on multi or single species histogram files
    - Single-Species = 1 sub-protein type.
    - Multi-Species = 2+ sub-protein types
  + Single Species Histograms: Functions specifically for the single species object.
  + Multi Species Histograms: Functions specifically for the multi species object
* Complex Location: Reads in PDB + restart or input files to determine location of certain complex sizes
* Transition Matrix: Reads in the transition matrix file and create a variety of outputs
* XYZ: Reads in .xyz files which report coordinates for specific timestamps, and create a variety of outputs

### Methods VS Functions

All functions included are either normal functions that can be run on their own or (2) methods of objects

Normal Functions: These can be run on their own, and each time they run it is completely disconnected from any previous function calls.

* The formatting for a function in these docs will just be it standing alone (ex: create\_pdf() )

Methods: Methods are functions that are connected to an object and the data stored in an object. This means after an object is initialized all methods run will store outputs in the object (as well as outputting them).

* The formatting for a method in these docs will always include the name of its parent object (ex: MultiHistogram.draw\_line() )

### Function Formatting:

Each sub-section will have a variety of useful functions listed. This is what each section means

\*NOTE: Each section that has [] around it means that it will be replaced by something in the actual description. These words inside of the brackets describe what that will be.

Function Format:

[**ObjectName**(if it is a method)]*.*[function-name] ([function parameters])

Description:

[Description of function.]

Important Notes:

[Important notes for the function.]

Parameters:

* **[parameter-name**]: ([*parameter type], [if-optional] = [value-if-unset])*
  + [Description of parameter.]
* This is repeated for each parameter.

Returns:

* [name of the return of the function]: Description of return.

Example:

[An example of the function being used, with images if necessary.]

# Creating NERDSS Inputs

This section describes how to automatically create inputs for NERDSS for two types of models, either a Platonic solid or a PDB structure.

## Structure of the NERDSS input files:

## 

NERDSS requires two input files to simulate a model, a parameter file (parms.inp) and a molecule structure file for each species in the system (SPECIES1.mol, SPECIES2.mol, etc.)

Why It Is Useful: These files often hold a lot of data that needs to be copied and pasted, and data that requires lots of math. This automates that process.

INP Files: The main input file for NERDSS that includes many important parameters.

* Includes: timesteps, dimensions, included molecules, and reactions

MOL Files: Each file stores information about each included molecule

* Includes: Name, Center of Mass, rotation, binding sites, and molecule type

For more information, read the [NERDSS user guide](https://github.com/mjohn218/NERDSS/blob/master/NERDSS_USER_GUIDE.pdf) here

## 1.2 Database PDB

These functions will use .PDB files downloaded from the [protein databank](https://www.rcsb.org/) to create a NERDSS input.

### 1.2.1 PDB\_UI ():

Description: This function will read in the PDB file and create a NERDSS input with a user-friendly UI. This cannot be run in a Jupyter Notebook, as it requires a command line to output text and get user input.

\* There are no inputs in the original function, you just need to open python in the command line, import ioNERDSS, then write **PDB\_UI()** into command line or an integrated development environment (IDE).

**Tutorial:**

First, store a PDB file in the working directory and call this function with an appropriate IDE (VSCode for example). The interface will require the user to input the name of the desired PDB file. Type in the full name of the file (along with the file path if not saved in working directory) and press return to continue. See screenshot below for details.

A screen shot of a computer

Description automatically generated

Once the file name is input, the code will read the desired information inside this PDB file and show some basic parameters on the interface (this will take a while), including the name of each chain, size of each chain, the coordinate of each COM and each pair of interfaces.

A picture containing table

Description automatically generated

Among all pairs of interfaces, you are asked if you want to change the distance between interfaces (AKA sigma), if you enter ‘yes’, you can change any of the distance shown above or change all distances to the same value; if you enter ‘no’, the distances will not be changed.

Graphical user interface, text

Description automatically generated with medium confidence

You can then choose to whether display the protein complex so far in a 3D plot. If you type “yes”, a 3D plot will be displayed showing the interfaces on all chains with updated changes of the distances between interfaces.



A graph of a graph with lines and numbers

Description automatically generated

At last, you are asked if you want each chain to be centered at COM. If you write ‘yes’, the COM coordinate will be normalized as (0,0,0) and the corresponding coordinates of all interfaced will be all changed accordingly in the final output; if you write ‘no’, the coordinates for all COM and interfaces will stay the same as the original ones.



The code will then automatically quit and the corresponding input (multiple .mol files and single .inp file) will be found in the working directory.

### 1.2.2 MAIN OBJECT - Read in PDB File to create protein complex object:

**ProteinComplex**(FileName, ChainsIncluded, MaxBoundLength, SymmetryApplied)

Description:

Reads in a database PDB file and finds information important for NERDSS inputs. This function will extract the coordinate information stored inside a real PDB file and calculate the COM of each unique chain, as well as recognize the binding information between each pair of chains (all atoms of different unique chains that are closer than 3.0 angstroms are considered as bound), including whether two chains are bound and the coordinates of each binding interface. All the information will be printed on the screen and the returns will contain all the information for further analysis. All below functions are methods of this.

Parameters:

* **FileName:** *string*
  + The full path of the desired PDB file or name of the file if in same directory.
* **ChainsIncluded:** *list, optional*
  + A list of which chains you want to be included. Must be more than 2.
* **MaxBoundLength:** *float, optional, default = 0.3*
  + Atoms on different chains that are lower than MaxBoundLength nm apart are considered as bound.
* **SymmetryApplied:** *boolean, optional, default = false*
  + If the inputted .pdb file is converted from .cif files that records symmetry information, please set this parameter to True to have to program recognize chain names correctly.

Example:

Clathrin = ioNERDSS.ProteinComplex(FileName=”ioNERDSS\Test\database.pdb”, ChainsIncluded=['A’,’ B’,’ P’,’ Q’], MaxBoundLength=0.3, SymmetryApplied=False)

>>> *Creates a Protein Complex object called Clathrin that holds the data in the database file*

### 1.2.3 MODIFY - Calculate the angles between pairs of interfaces:

**ProteinComplex.calc\_angle**()

Description:

Calculates the 5 associating angles of each pair of interfaces. The default normal vector will be assigned as (0, 0, 1). If the co-linear issue occurs, the system will use (0, 1, 0) instead to resolve co-linear issue. The calculated 5 angles will be shown on the screen automatically. If user intends to manually input the normal vector, please refer to function ‘PDB\_UI’, the separated function does not support manual inputs. The returns will contain all the information for further analysis. After it runs, it is ready to be output. Will result in some functions being runnable, and other functions not runnable (on this object), so be careful!

Important Notes:

* Calc\_angle() is a very powerful function that will result in some functions being able to run, and others unable to run. Here is the list of both. All other functions can be run as normal.
* Can not be run after calc\_angle(): filter, change\_sigma
* Must be run after calc\_angle(): write\_input, norm\_COM

Parameters:

* None

Example:

Clathrin.calc\_angle()

>>> *Finds the 5 associating angles for each interface. Ready to be output now.*

### 1.2.4 MODIFY - Filter the protein complex object, so it only includes the chains you want:

**ProteinComplex.filter**(ChainList)

Description:

This function will filter the desired chain according to the input list of chains. Must be run before calc\_angle().

Parameters:

* **ChainList:** *list with str elements*
  + The desired name of chains that users intend to examine.

Example:

Clathrin.filter(ChainsIncluded=['A’,’ B’])

>>> *Edits clathrin so it only includes the A and B chain*

1.2.5 MODIFY - Change the distance between 2 binding sites:

**ProteinComplex.change\_sigma**(ChangeSigma, SiteList, NewSigma)

Description:

This function allows users to change the value of sigma (the distance between two binding interfaces). The new sigma value and the corresponding coordinates of interfaces will be shown on the screen and the returns will contain all the information for further analysis. **Note:** Cannot be run after calc\_angle().

Parameters:

* **ChangeSigma** (*bool, optional = False*): Whether the sigma values will change or stay the same
* **SiteList** (*list with int elements, optional*): It consists of the serial numbers of the pair of interfaces for which the user needs to modify the sigma value. The serial number is determined by the pairing sequence shown by the initialization function. If the serial number is 0, it means to change all pairs of interfaces into the same sigma value.
* **NewSigma** (*list with float elements, optional*): It consists of the actual sigma value that users desire to change, according to the sequence of input ‘SiteList’.

Example:

Clathrin.change\_sigma(ChangeSigma = True, SiteList = [1,2], NewSigma = [1.01, 0.80])

>>> *Changes distance between first and second chain to 1.01 nanometers*

### 1.2.6 MODIFY - Normalize the COM of each chain:

**ProteinComplex.norm\_COM**()

Description:

Normalizes the COM of each chain as (0, 0, 0). The interface of each chain will be subtracted by the COM coordinates accordingly. Once the calculation is completed, there will be a message shown on the screen. The returns will contain all the information for further analysis. **Note:** Must be run after calc\_angle().

Parameters:

* None

Example:

Clathrin.norm\_COM()

>>> *Center of Mass and all binding locations are now set around 0,0,0.*

1.2.7 OUTPUT - Writes new NERDSS input files based on chain info:

**ProteinComplex.write\_input**()

Description:

This function will write ‘.inp’ and ‘.mol’ files according to all the calculations and modifications above. Multiple ‘.mol’ file and a ‘.inp’ file can be found in the working directory once the function has finished running. **Note:** Must be run after calc\_angle().

Parameters:

* None

Example:

Clathrin.write\_input()

>>> *Creates new .inp and .mol files ready to be input into NERDSS*

### 1.2.8 OUTPUT - Writes a new PDB file based on chain info:

**ProteinComplex.write\_PDB**()

Description:

This function will generate a PDB file that only contains the calculated COMs and reaction interfaces for visualization and comparison with the original PDB file. The input will be the returns of the previous function. The unit for the coordinates in the PDB file is in angstroms not nanometers, so the value will be 10 times larger than that in NERDSS input files.

Parameters:

* None

Example:

Clathrin.write\_PDB()

>>> *Creates new .pdb files to be compared with original inputted .pdb file*

1.2.9 OUTPUT - Create a 3D plot of each inputted chain:

**ProteinComplex.plot\_3D**()

Description:

This function will generate a 3D plot indicating the spatial geometry of each simplified chain. The solid lines of different colors are connecting the COM with interfaces within each chain; the black dotted line is connecting each pair of interfaces, and the COMs are shown as solid points with their names above. To interact with the plot, other IDEs rather than Jupyter Notebook (such as VSCode) are recommended.

Parameters:

* None

Example:

Clathrin.plot\_3D()

A graph of a graph with lines and numbers

Description automatically generated

## 1.3 Platonic Solid Self-assembly Input File Writing:

Platonic solid self-assembly includes 10 models, so 10 separate functions are needed. The names of the functions are given in the following table:

|  |  |  |
| --- | --- | --- |
| Platonic Solid | Center-of-Mass Position | Name of Function |
| Tetrahedron (4-face) | Each Face | tetr\_face (radius, sigma) |
| Tetrahedron (4-face) | Each Vertex | tetr\_vert (radius, sigma) |
| Cube (6-face) | Each Face | cube\_face (radius, sigma) |
| Cube (6-face) | Each Vertex | cube\_vert (radius, sigma) |
| Octahedron (8-face) | Each Face | octa\_face (radius, sigma) |
| Octahedron (8-face) | Each Vertex | octa\_vert (radius, sigma) |
| Dodecahedron (12-face) | Each Face | dode\_face (radius, sigma) |
| Dodecahedron (12-face) | Each Vertex | dode\_vert (radius, sigma) |
| Icosahedron (20-face) | Each Face | icos\_face (radius, sigma) |
| Icosahedron (20-face) | Each Vertex | icos\_vert (radius, sigma) |

Description:

Generates NERDSS input files (.inp and .mol files) for Platonic solid self-assembly system.

Parameters:

* **radius:** *float*
  + The radius of the Platonic solid in **nm**, which is defined by the distance from the center of Platonic solid to each vertex.
* **sigma:** *float*
  + The distance of each interface when a reaction takes place in **nm**.

Example:

tetr\_face(radius=10, sigma=1)

>>> *Creates parms.inp and .mol files for the self-assembly system for a tetrahedron with COM in the face*

## 1.4 Repeated Protein Subunit Regularization:

**repeated\_protein\_subunit\_regularization**(PathName, UniqueChainList)

Description:

The program regularizes symmetric protein cages made of repeated subunits recorded in .pdb files and accounts for experimental errors to create one set of NERDSS input parameters. The first part of the output gives the center and radius of the sphere of the best fit. The second part of the output gives the input parameters needed to create the .mol file. The COM is by default at [0,0,0]. In the example, there are five interfaces each with their coordinates [x,y,z] listed on each row. The third part of the output gives the binding parameters needed to create the .parms file. In the example, the binding parameters corresponding to the first site is listed on the first row [sigma, theta1, theta2, phi1, phi2, omega], and those corresponding to the second site is listed on the second row, and so on.

Parameters:

* **PathName:** *str*
  + The pathname to **the .pdb file that only contains the calculated COMs and reaction interfaces** generated from original PDB files downloaded from the RCSB Protein Data Bank. For instructions of how to do generate such a file see 1.2.8 OUTPUT - Writes a new PDB file based on chain info.
* **UniqueChainList:** *list of lists, optional*
  + A list to group chains that should be identical, ie, if the .pdb file records a cage made of clathrin light chains A, B, C and clathrin heavy chains D, E, F, input [[“A”, “B”, “C”],[“D”, “E”, “F”]] for UniqueChainList. If not input, it will assume that the .pdb file only contain one kind of proteins.

Example:

repeated\_protein\_subunit\_regularization(PathName = " ioNERDSS\Test\7asl\_sites.pdb ")

A screen shot of a computer

Description automatically generated

## 1.5 GUI – graphic interface to create .inp and .mol:

**gui**()

Description: Graphic interface to generate .inp and .mol NERDSS input files using user inputted values.

Parameters:

* None

Example:

import ioNERDSS as io

io.gui()

A screenshot of a computer

Description automatically generated

**\*\*\*Making an actual tutorial for GUI soon**

**Tutorial:**

A screenshot of a computer

Description automatically generated

In the parameters tab, enter the parameters for the simulation. For a more detailed description of what these parameters do, please refer to the [NERDSS user guide](https://github.com/mjohn218/NERDSS/blob/master/NERDSS_USER_GUIDE.pdf).

A screenshot of a computer

Description automatically generated

In the Boundaries tab, set up the shape of the space in which all molecules are confined during the simulation. Users can choose either a rectangular or spherical shape and define its size.

A screenshot of a computer

Description automatically generated

In the Molecules tab, enter information for each molecule that will participate in the simulation. Click “Add Molecule” when finished inputting parameters for one molecule. Repeat this process to incorporate another molecule.

IONERDSS also supports automatically generating .mol files from PDB files. Please refer to section 1.2.1 in this user guide for more details.

A screenshot of a computer

Description automatically generated

In the Reaction tab, enter intended reactions during the simulation. NERDSS supports biomolecular associations, zeroth order creation, destructions, Michaelis-Menten reactions, unimolecular creation, bimolecular state change, and state change. Please refer to the NERDSS\_USER\_GUIDE\_MASTER for more information. Click “Add Reaction” when finished. Repeat this process to enter different reactions.

When finished entering all parameters, click “Generate .inp File” to generate a parm.inp and .mol files in the current directory.

# 2. Analyzing NERDSS Outputs:

Creates graphs, spreadsheets, pandas data frames, 3D models, and more from the NERDSS outputs!

## 2.1 Analyzing Histogram Files (General Functions):

Histogram.dat files are outputs from NERDSS that holds the count of each complex size/type at every time step.

This section includes the initialization functions and functions included in both objects.

\*NOTE: All histogram functions are methods of either the SingleHistogram object or MultiHistogram object. The initialization functions and functions included in both objects are in the general section.

### 2.1.1 MAIN OBJECT – The Main Single-component Histogram Object

**SingleHistogram**(FileName, FileNum, InitialTime, FinalTime, SpeciesName)

Description:

An object that holds all the data from a single species histogram file, and allows for easy interpretation of the data

Parameters:

* **FileName:** *str*
  + The path to the ‘.dat’ file, which is usually named as ‘histogram\_complexes\_time.dat’, representing the histogram data to be analyzed.
  + If there are multiple, the files should be named [name]\_1.dat, [name]\_2.dat, ect. And the FileName should be just [name].dat
  + Example: histogram\_1.dat, histogram\_2.dat …. FileName = histogram.dat
* **FileNum:** *int*
  + The number of the total input file. If multiple files are provided, their names should obey the naming rule listed below.
* **InitialTime:** *float*
  + The initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
* **FinalTime:** *float*
  + The final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
* **SpeciesName:** *str*
  + The name of species that users want to examine, which should also be identical with the name written in the input (.inp and .mol) files.

Functions:

* Every function in the general / SingleHistogram section is a method of this object.

Returns:

* A new instance of the SingleHistogram object.

Example:

test\_histogram = ioNERDSS.SingleHistogram(FileName = "ioNERDSSPyPi\TestingFunctions\histogram\_single\_component.dat", FileNum = 1, InitialTime = 0.0, FinalTime = 1.00, SpeciesName = 'dode')

>>> *Initializes the test\_histogram SingleHistogram object for use in all future functions*

### 2.1.2 MAIN OBJECT – The Main Multi-Component Histogram Object

**MultiHistogram**(FileName, FileNum, InitialTime, FinalTime, SpeciesName)

Description:

An object that holds all the data from a multi species histogram file, and allows for easy interpretation of the data

Parameters:

* **FileName:** *str*
  + The path to the ‘.dat’ file, which is usually named as ‘histogram\_complexes\_time.dat’, representing the histogram data to be analyzed.
  + If there are multiple, the files should be named [name]\_1.dat, [name]\_2.dat, ect. And the FileName should be just [name].dat
  + Example: histogram\_1.dat, histogram\_2.dat …. FileName = histogram.dat
* **FileNum:** *int*
  + The number of the total input file. If multiple files are provided, their names should obey the naming rule listed below.
* **InitialTime:** *float*
  + The initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
* **FinalTime:** *float*
  + The final time in **seconds** that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
* **SpeciesList:** *list*
  + The names of the species that are in the multi-histogram file, which should also be identical with the name written in the input (.inp and .mol) files.

Functions:

* Every function in the general / MultiHistogram section is a method of this object!

Returns:

* A new instance of the MultiHistogram object.

Example:

test\_histogram = ioNERDSS. MultiHistogram(FileName = "ioNERDSSPyPi\TestingFunctions\histogram\_multi\_component.dat", FileNum = 1, InitialTime = 0.0, FinalTime = 1.00, SpeciesList = [‘A’,’B’])

>>> *Initializes the test\_histogram SingleHistogram object for use in all future functions*

### 2.1.3 DATAFRAME – stores count of each complex type for each timestep

**SingleHistogram.hist\_to\_df**(FileNum, SaveCsv)

**MultiHistogram.hist\_to\_df**(FileNum, SaveCsv)

Description:

This function enables users to convert the raw .dat file to a data frame in python pandas package for multi-species system. Each column in the data frame includes the simulation time and selected occurrences of species during the simulation; each row is separated by a different simulation time.

Parameters:

* **OpName:** *str, optional = “histogram”*
  + The name of the output .csv file. If SaveCsv is false, this parameter is irrelevant, it will not break if you input a name.
* **SaveCsv:** *bool, optional = True*
  + Whether the corresponding .csv file will also be saved as ‘histogram.csv’

Returns:

* hist\_dataframe: a panda dataframe that holds the count of each complex at each timestep.

Example:

hist\_dataframe

= test\_histogram.hist\_to\_df(SaveCsv = False, FileNum = -1)

>>>

Time(s): A: 1. A: 1. B: 1. A: 1. B: 2. A: 1. B: 3. A: 1. B: 4. ... B: 2. B: 3. B: 4. B: 5. B: 6.

0 0.000 100 0 0 0 0 ... 0 0 0 0 0

1 0.001 86 4 1 0 0 ... 4 0 0 0 0

2 0.002 76 3 1 1 0 ... 5 0 0 0 0

### 2.1.4 CSV – stores count of each complex type for each timestep

**SingleHistogram.hist\_to\_csv**(FileNum)

**MultiHistogram.hist\_to\_csv**(FileNum)

Description:

This function enables users to convert the raw .dat file to a .csv file for a system. Each column in the data frame includes the simulation time and selected occurrences of species during the simulation; each row is separated by a different simulation time. If given multiple histograms, it will calculate the average between them.

Parameters:

* **OpName:** *str, optional = “histogram”*
  + The name of the output .csv file

Returns:

* csv: returns a .csv file that holds the count of each complex at each timestep.

Example:

test\_histogram.hist\_to\_csv(OpName=”histogram”)

>>>

A table with numbers and letters

Description automatically generated

### 2.1.5 LINE GRAPH – max count of protein species in a single complex at a time

**SingleHistogram.line\_max\_complex\_size**(SpeciesName, ShowFig, SaveFig, SaveVars)

**MultiHistogram.line\_max\_complex\_size**(SpeciesName, ShowFig, SaveFig, SaveVars)

Description:

Creates a plot indicating maximum number of a specific protein species in single complex molecule during a certain time. The X-axis is time, and Y-axis is the largest # of the tracked protein in a single complex

Parameters:

* **SpeciesName:** *str, optional for single histogram*
  + Required if a multi-histogram file is input. It is the protein species that will be tracked. If ‘tot’ is entered, it will be all species.
* **ExcludeSize:** *int, optional*
  + Monomers in a complex that is smaller or equal to this number will not be included.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

* time\_stamps: the time stamps included in the line graph.
* max\_cmplx\_size: list of the max complex size at each time stamp.
* std: standard deviation of each max\_cmplx\_size.

Example:

time\_stamps, max\_cmpx\_size, std =

test\_histogram. line\_max\_complex\_size(ShowFig = True, SaveFig = False,SaveVars=False)

Or Multi-hist:

time\_stamps, max\_cmpx\_size, std =

test\_histogram. line\_max\_complex\_size(SpeciesName=’A’,ShowFig = True, SaveFig = False,SaveVars=False)

>>>

A graph with blue lines

Description automatically generatedA graph with a line

Description automatically generated

### 2.1.6 LINE GRAPH – mean count of protein species in a single complex at a time

**SingleHistogram.line\_mean\_complex\_size**(SpeciesName, ShowFig, SaveFig, SaveVars)

**MultiHistogram.line\_mean\_complex\_size**(SpeciesName, ShowFig, SaveFig, SaveVars)

Description:

This function enables users to obtain a plot indicating mean number of a specific protein species in single complex molecule during a certain time. The X-axis is time, and Y-axis is the average # of the tracked protein in a single complex.

Parameters:

* **SpeciesName:** *str, optional for single histogram*
  + Required if a multi-histogram file is input. It is the protein species that will be tracked.
* **ExcludeSize:** *int, optional*
  + Monomers in a complex that is smaller or equal to this number will not be included.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

* time\_stamps: the time stamps included in the line graph.
* mean\_cmplx\_size: list of the mean complex size at each time stamp.
* std: standard deviation of each max\_cmplx\_size.

Example:

time\_stamps, mean\_cmpx\_size, std =

test\_histogram.line\_mean\_complex\_size(ShowFig = True, SaveFig = False, SaveVars=False)

Or multi-hist:

time\_stamps, mean\_cmpx\_size, std =

est\_histogram.line\_mean\_complex\_size(SpeciesName=’A’,ShowFig = True, SaveFig = False,SaveVars=False)

>>>

A graph with blue line

Description automatically generatedA graph with a line drawn on it

Description automatically generated

## 2.2 Analyzing Single-Component Histogram Files

Includes methods that are methods of the SingleHistogram object.

### 2.2.1 HISTOGRAM – average number of each complex species size:

**SingleHistogram.hist\_complex\_count**(BarSize, ShowFig, SaveFig,SaveVars)

Description:

Creates histogram of the average number of each type/size of complex species. The X-axis is each complex species type (each size), and Y-axis is the average count over a time frame (initial to final). If a single file is provided, the input file should be named as its original name (‘histogram\_complexes\_time.dat’); if multiple files are provided, the name of input file should also include serial number as ‘histogram\_complexes\_time\_X.dat’ where X = 1,2,3,4,5…

Parameters:

* **BarSize:** *int, optional = 1*
  + The size of each data bar in x-dimension. The x-axis will be separated evenly according to this number and the count of each size range will be summed up and shown together.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

* cmplx\_sizes: each complex size.
* cmplx\_count: average count of each complex size.
* std: standard deviation of each cmplx\_count.

Example:

Cmplx\_sizes, cmplx\_count, std =

test\_histogram. hist\_complex\_count(BarSize = 1, ShowFig = True, SaveFig = False,SaveVars=False)

>>>

A graph with blue bars

Description automatically generated

### 2.2.2 3D HISTOGRAM – relative occurrence of each species over time:

**SingleHistogram.hist\_3d\_complex\_count**(TimeBins, xBarSize, ShowFig, SaveFig, SaveVars)

Description:

This function enables users to generate a 3D histogram representing the number of monomers in a single complex as simulation time develops. The x-axis is the number of monomers, y-axis is the averaged time and z-axis is the relative occurrence probabilities.

Parameters:

* **TimeBins:** *int*
  + The number of bins that users want to divide the selected time into. The value should be a positive integer.
* **xBarSize:** *int, optional = 1*
  + The size of each data bar in x-dimension. The x-axis will be separated evenly according to this number and the count of each size range will be summed up and shown together.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

* cmplx\_sizes: each complex size.
* time\_bins: each time bin included.
* cmplx\_count: average count of each complex size.

Example:

cmplx\_sizes time\_bins, cmplx\_count, std=

test\_histogram.hist\_3d\_complex\_count(TimeBins=10, xBarSize = 1, ShowFig = True, SaveFig = False,SaveVars=False)

>>>

A graph of a bar graph

Description automatically generated

2.2.3 HEATMAP – average number of complexes at each time interval:

**SingleHistogram.heatmap\_complex\_count**(TimeBins, xBarSize, ShowFig, ShowMean, ShowStd, SaveFig, SaveVars)

Description:

This function enables users to generate 2D histogram of numerical distribution of different N-mers vs. time. The x-axis is the distribution of number of monomers in single complex and y-axis is the time. The color in each box indicates the number of corresponding N-mers when corresponding time is reached.

Parameters:

* **TimeBins:** *int*
  + The number of bins that users want to divide the selected time into. The value should be a positive integer.
* **xBarSize:** *int, optional = 1*
  + The size of each data bar in x-dimension. The x-axis will be separated evenly according to this number and the count of each size range will be summed up and shown together.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **ShowMean:** *bool, optional = False*
  + Whether the corresponding mean value will be shown in the center of each box
* **ShowStd:** *bool, optional = False*
  + Whether the corresponding standard deviation value will be shown in the center of each box
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be saved as a .png.
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

* cmplx\_sizes: each complex size.
* time\_bins: each time bin included.
* cmplx\_count: average count of each complex size.
* std: standard deviation of each cmplx\_count.

Example:

time\_stamps, time\_bins, mean\_cmpx\_size, std =

test\_histogram. heatmap\_complex\_count(TimeBins = 10, xBarSize = 1, ShowFig = True, SaveFig = False, ShowMean=False, ShowStd=False,SaveVars=False)

>>>



### 2.2.4 HEATMAP - total count of monomers in each complex size vs. time:

**SingleHistogram.heatmap\_monomer\_count**(TimeBins, xBarSize, ShowFig, ShowMean, ShowStd, SaveFig, SaveVars)

Description:

This function enables users to generate 2D histogram of total count of monomers in different N-mers vs. time. The x-axis is the number of monomers in single complex and y-axis is the time. The color in each box indicates the total number of corresponding monomers in N-mers when corresponding time is reached.

Parameters:

* **TimeBins:** *int*
  + The number of bins that users want to divide the selected time frame into. The value should be a positive integer.
* **xBarSize:** *int, optional = 1*
  + The size of each data bar in x-dimension. The x-axis will be separated evenly according to this number and the count of each size range will be summed up and shown together.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **ShowMean:** *bool, optional = False*
  + Whether the corresponding mean value will be shown in the center of each box
* **ShowStd:** *bool, optional = False*
  + Whether the corresponding standard deviation value will be shown in the center of each box
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be saved as a png file.
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

* cmplx\_sizes: each complex size.
* time\_bins: each time bin included.
* mono\_count: number of monomers in each complex.
* std: standard deviation of each cmplx\_count.

Example:

cmplx\_sizes time\_bins, mono\_count, std =

test\_function.heatmap\_monomer\_count(Timebins = 10, xBarSize = 1, ShowFig = True, SaveFig = False, ShowMean=False, ShowStd=False,SaveVars=False)

>>>

A graph of a number of monomers

Description automatically generated

2.2.5 HEATMAP - fractions of original monomers in each complex species vs. time:

**SingleHistogram.heatmap\_monomer\_fraction**(TimeBins, xBarSize, ShowFig, ShowMean, ShowStd, SaveFig, SaveVars)

Description:

Generates 2D histogram of fraction of original monomers in different complex species over time. This function enables users to generate 2D histogram of fractions of monomers forming different N-mers vs. time. The x-axis is the number of monomers in single complex and y-axis is the time frame. The color in each box indicates the fraction of monomers forming corresponding N-mers when corresponding time is reached.

Parameters:

* **TimeBins:** *int*
  + The number of bins that users want to divide the selected time frame into. The value should be a positive integer.
* **xBarSize:** *int, optional = 1*
  + The size of each data bar in x-dimension. The x-axis will be separated evenly according to this number and the count of each size range will be summed up and shown together.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **ShowMean:** *bool, optional = False*
  + Whether the corresponding mean value will be shown in the center of each box
* **ShowStd:** *bool, optional = False*
  + Whether the corresponding standard deviation value will be shown in the center of each box
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be saved as a .png.
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

* cmplx\_sizes: each complex size.
* time\_bins: each time bin included.
* mono\_fractions: % of monomers in each complex.
* std: standard deviation of each cmplx\_count.

Example:

cmplx\_sizes time\_bins, mono\_fractions, std =

test\_function.heatmap\_monomer\_fraction(TimeBins = 10, xBarSize = 1, ShowFig = True, SaveFig = False, ShowMean=False,ShowStd=False,SaveVars=False)

>>>

A graph of different colors

Description automatically generated with medium confidence

## 2.3 Analyzing Multi-Component Histogram Files

Includes methods that are methods of the MultiHistogram object.

### 2.3.1 HISTOGRAM – Frequency of each complex size:

**MultiHistogram.hist\_complex\_count**(ExcludeSize, ShowFig, SaveFig, SaveVars)

Description:

This function enables users to plot general histogram of total size of complex or selected species inside each complex for a multi-species system. It will also analyze multiple input files and show the result along with error bar. The x-axis is the size of selected species or total number of monomers, and the y-axis is the average number of counts for corresponding size.

Parameters:

* **BinNums:** *int, optional = 10*
  + Number of bins in the histogram.
* **ExcludeSize:** *int, optional = 0*
  + In the generated plot, the number of monomers in the complex that are no larger than this number will be excluded and will not be considered into the average calculation.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

* cmplx\_sizes: each complex size.
* cmplx\_count: frequency of each complex.
* std: standard deviation of each cmplx\_count.

Example:

cmplx\_sizes, cmplx\_count, std =

test\_histogram.hist\_complex\_count(BinNums = 10, ShowFig = True, SaveFig = False, SaveVars = False)

>>>

A graph of a graph

Description automatically generated with medium confidence

### 2.3.2 STACKED HISTOGRAM – Counts of complex species with certain protein compositions:

**MultiHistogram.stack\_hist\_complex\_count**(xAxis, DivideSpecies, DivideSize, BarSize, ExcludeSize, ShowFig, SaveFig, SaveVars)

Description:

This function enables users to plot general histogram of total size of complex or selected species for a multi-species system. Each bar is split by three stacked bars which represent the size distribution of another selected species compared to a desired input. It will also analyze multiple input files and show the result along with error bar. The x-axis is the size of selected species or total number of monomers, and the y-axis is the average number of counts for corresponding size.

Parameters:

* **xAxis:** *string*
  + Indicates the species shown on the x-axis. Either input a species, so the x-axis will only show the number of selected components. Else, input ‘tot’, so the x-axis will count all species in a single complex.
* **DivideSpecies:** *string*
  + Indicates the name of the species that users want to separate by size.
* **DivideSize:** *int*
  + The value that separates the size of the dissociate complex, for example, if DivideSize = 5, that means the dissociate events are classified as ‘DivideSpecies size < 5’, ‘DivideSpecies size = 5’ and ‘DivideSpecies size > 5’.
* **BarSize:** *int, optional = 1*
  + The size of each data bar in x-dimension. The x-axis will be separated evenly according to this number and the count of each size range will be sum up and shown together.
* **ExcludeSize:** *int, optional = 0*
  + In the generated plot, the number of monomers in the complex that are no larger than this number will be excluded and will not be considered into the average calculation.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

* x\_mono\_count: count of the x monomer in a complex.
* cmplx\_count: frequency of each complex. List1 = below division, list2 = equal division, list3 = above division.
* std: standard deviation of each cmplx\_count.

Example:

x\_mono\_count, cmplx\_count, std =

test\_histogram. multi\_hist\_stacked(xAxis = "A", DivideSpecies = "B", DivideSize = 5, BarSize = 1, ShowFig = True, SaveFig = False, SaveVars = False)

>>>



### 2.3.3 HEATMAP – Average count of each complex composition over simulation time:

**MultiHistogram.heatmap\_ complex\_dist**(xAxis, yAxis, xBarSize, yBarSize, ShowFig, ShowMean, ShowStd, SaveFig, SaveVars)

Description:

This function enables users to generate a heatmap during a certain time frame representing the distribution of size of selected species. The x and y axis are both desired individual components and the color of each square represents the relative occurrence probability of complex of corresponding size.

Parameters:

* **xAxis:** *string*
  + Indicates the species shown on the x-axis. If xAxis is included inside SpeciesList, the x-axis will only show the number of selected components.
* **yAxis:** *string*
  + Indicates the species shown on the x-axis. If yAxis is included inside SpeciesList, the x-axis will only show the number of selected components.
* **xBarSize:** *int, optional = 1*
  + The size of each data bar in x-dimension. The x-axis will be separated evenly according to this number and the count of each size range will be sum up and shown together.
* **yBarSize:** *int, optional = 1*
  + The size of each data bar in y-dimension. The y-axis will be separated evenly according to this number and the count of each size range will be sum up and shown together.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **ShowMean:** *bool, optional = False*
  + Whether the corresponding mean value will be shown in the center of each box
* **ShowStd:** *bool, optional = False*
  + Whether the corresponding standard deviation value will be shown in the center of each box
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be saved as a .png.
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

* x\_mono\_count: count of the x monomer in a complex.
* y\_mono\_count: count of the y monomer in a complex.
* cmplx\_count: frequency of each complex.
* std: standard deviation of each cmplx\_count.

Example:

x\_mono\_count, y\_mono\_count, cmplx\_count, std =

test\_histogram.heatmap\_complex\_dist(xAxis=’A’, yAxis=’B’, xBarSize=1, yBarSize=1, ShowFig = True, ShowMean=False, ShowStd=False, SaveFig = False, SaveVars = False)

>>>



### 2.3.4 3D HISTOGRAM - Average count of each complex composition over simulation time:

**MultiHistogram.hist\_3D\_complex\_dist**(xAxis, yAxis, xBarSize, yBarSize, ShowFig, SaveFig, SaveVars)

Description:

Generates a 3D histogram during a certain time frame representing the distribution of size of selected species. The x and y axis are both desired individual components and the height of each column represents the relative occurrence probability of complex of corresponding size.

Parameters:

* **xAxis:** *string*
  + Indicates the species shown on the x-axis. If xAxis is included inside SpeciesList, the x-axis will only show the number of selected components.
* **yAxis:** *string*
  + Indicates the species shown on the x-axis. If yAxis is included inside SpeciesList, the x-axis will only show the number of selected components.
* **xBarSize:** *int, optional = 1*
  + The size of each data bar in x-dimension. The x-axis will be separated evenly according to this number and the count of each size range will be sum up and shown together.
* **yBarSize:** *int, optional = 1*
  + The size of each data bar in y-dimension. The y-axis will be separated evenly according to this number and the count of each size range will be sum up and shown together.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be saved as a png file.
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

* x\_mono\_count: count of the x monomer in a complex.
* y\_mono\_count: count of the y monomer in a complex.
* cmplx\_count: frequency of each complex.

Example:

x\_mono\_cout, y\_mono\_count, cmplx\_count =

test\_histogram.hist\_3D\_complex\_dist(xAxis=’A’, yAxis=’B’, xBarSize=1, yBarSize=1, ShowFig = True, SaveFig = False, SaveVars = False)

A graph of a bar graph

Description automatically generated

### 2.3.5 LINE GRAPH – Fraction of monomers assembled over time:

**SingleHistogram.frac\_of\_assemble**(Mol, Threshold, ShowFig, SaveFig, SaveVars)

Description:

Determines fraction of monomers in a complex of a certain size at each time stamp.

Parameters:

* **Mol:** *string*
  + Which molecule will be shown
* **Threshold:** *int, optional = 2*
  + The minimum size to be considered assembled
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be saved as a .png.
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

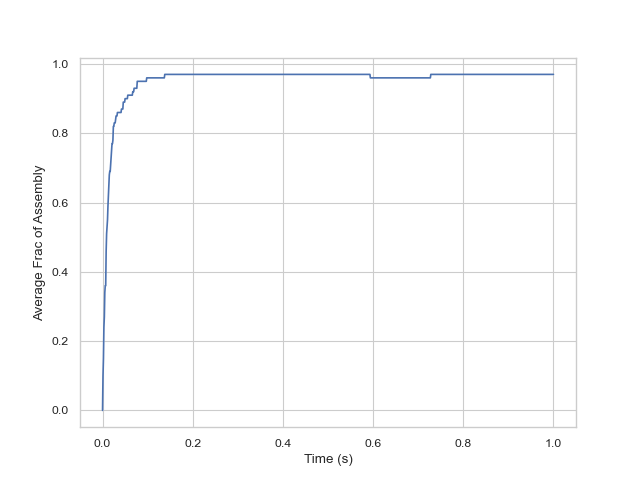
* time\_bins: each time bin included.
* frac\_assembled\_monos: % of monomers in a complex > threshold.
* std: standard deviation of each frac\_assembled\_monos.

Example:

time\_bins, frac\_assembled\_monos, std =

test\_function.frac\_of\_assemble(Mol=”B”,Threshold=2,ShowFig = True, SaveFig = False, ShowMean=False,ShowStd=False, SaveVars=False)

>>>



## 2.4 Analyzing Transition Matrix Files

### 2.4.1 LINE PLOT - calculates change in free energy among different sizes of complexes:

**free\_energy**(FileName, FileNum, InitialTime, FinalTime, SpeciesName, ShowFig, SaveFig, SaveVars)

Description:

The plot indicates the change in free energy in a selected time frame among different size of complexes. The x-axis is the size of complex and the y-axis is the free energy calculated as in the unit of , where the refers to the probability of occurrence of the number of times N-mer is counted (including association and dissociation). If multiple input files are given, the output plot will be the average value of all files and an error bar will also be included.

Parameters:

* **FileName:** *string*
  + The path to the ‘.dat’ file, which is usually named as ‘transition\_matrix\_time.dat’, representing the histogram data to be analyzed. If there are multiple, the files should be named [name]\_1.dat, [name]\_2.dat, etc., and the FileName should be just [name].dat
    - Example: transition\_1.dat, transition\_2.dat …. FileName = transition.dat
* **FileNum:** *int*
  + The number of the total input file. If multiple files are provided, their names should obey the naming rule.
* **InitialTime:** *float*
  + The initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
* **FinalTime:** *float*
  + The final time in **seconds** that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
* **SpeciesName:** *string*
  + The name of species that users want to examine, which should also be identical with the name written in the input (.inp and .mol) files.
* **DivideSize:** *int, optional = 2*
  + Value that distinguishes the size of the associate complex.
* **ShowFig:** *bool, optional = Tru*e
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

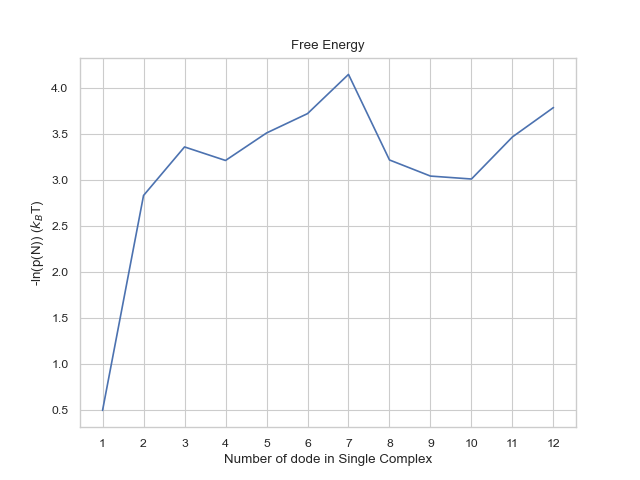
* cmplx\_size: list of each complex size.
* mean\_energy: the –ln(p(N)) in KbT of each complex.
* std: the standard deviation of mean energy.

Example:

cmplx\_sizes, mean\_energy, std =

ioNERDSS.free\_energy(FileName="ioNERDSSPyPi\TestingFunctions\\transition\_matrix\_time.dat",FileNum=1,InitialTime=0,FinalTime=1,SpeciesName="dode",ShowFig=True,SaveFig=False,SaveVars=True)

>>>



### 2.4.2 LINE PLOT – symmetric probability of association between complex sizes:

**associate\_prob\_symmetric**(FileName, FileNum, InitialTime, FinalTime, SpeciesName, DivideSize, ShowFig, SaveFig, SaveVars)

Description:

This line plot represents the probability of association between complexes of different sizes and other complexes of different sizes. The x-axis is the size of the complex and y-axis is the associate probability. Three lines will exist in the line graph, representing associating to complexes of sizes less than, equal to, or greater than the specified size, respectively. 'Symmetric' in the function name means that for the associate reaction, both sizes of complexes are counted as associating events symmetrically, for example, if an associate event occurs where a trimer associates to a tetramer as a heptamer, then this event is counted twice, which are trimer associates to tetramer and tetramer associates to trimer. If multiple input files are given, the output plot will be the average value of all files and an error bar will also be included.

Parameters:

* **FileName:** *string* 
  + The path to the ‘.dat’ file, which should be ‘histogram\_complexes\_time.dat’, representing the histogram data to be analyzed. If there are multiple, the files should be named [name]\_1.dat, [name]\_2.dat, etc., and the FileName should be [name].dat.
    - Example: transition\_1.dat, transition\_2.dat … FileName = transition.dat
* **FileNum:** *int* 
  + The number of the total input file. If multiple files are provided, their names should obey the naming rule.
* **InitialTime:** *float* 
  + The initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
* **FinalTime:** *float*
  + The final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
* **SpeciesName:** *string* 
  + The name of species that users want to examine, which should also be identical with the name written in the input (.inp and .mol) files.
* **DivideSize:** *int, optional = 2*
  + The value that distinguishes the size of the associate complex, for example, if DivideSize = 2, that means the associate events are classified as ‘associate size < 2’, ‘associate size = 2’ and ‘associate size > 2’.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

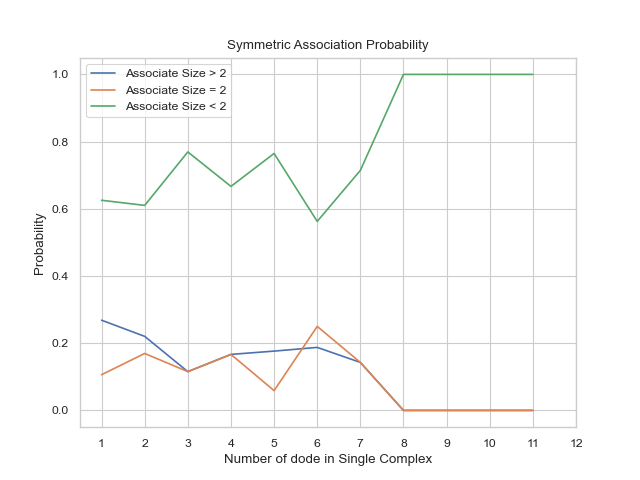
* cmplx\_size: list of each complex size.
* mean\_associate\_probability: the probability of a certain size of complex becoming another larger size.
* std: the standard deviation of mean probability.

Example:

cmplx\_size, mean\_associate\_probability, std =

ioNERDSS.associate\_prob\_symmetric(FileName="ioNERDSSPyPi\TestingFunctions\\transition\_matrix\_time.dat",FileNum=1,InitialTime=0,FinalTime=1,SpeciesName="dode",DivideSize=2,ShowFig=True,SaveFig=False,SaveVars=True)

>>>



### 2.4.3 LINE PLOT - asymmetric probability of association between complex sizes:

**associate\_prob\_asymmetric**(FileName, FileNum, InitialTime, FinalTime, SpeciesName, DivideSize, ShowFig, SaveFig, SaveVars)

Description:

This line plot represents the probability of association between complexes of different sizes and other complexes of different sizes. The x-axis is the size of the complex and y-axis is the associate probability. Three lines will exist in the line graph, representing associating to complexes of sizes less than, equal to, or greater than the specified size, respectively. 'Asymmetric' in the function name means that for the associate reaction, only the complexes of smaller size associating to the larger one is counted as associate event asymmetrically, for example, if an associating event occurs where a trimer associates to a tetramer as a heptamer, then this event is counted only once, which is a trimer associates to tetramer. If multiple input files are given, the output plot will be the average value of all files and an error bar will also be included.

Parameters:

* **FileName:** *string*
  + The path to the ‘.dat’ file, which is usually named as ‘transition\_matrix\_time.dat’, representing the histogram data to be analyzed. If there are multiple, the files should be named [name]\_1.dat, [name]\_2.dat, etc., and the FileName should be just [name].dat
    - Example: transition\_1.dat, transition\_2.dat …. FileName = transition.dat
* **FileNum:** *int*
  + The number of the total input file. If multiple files are provided, their names should obey the naming rule.
* **InitialTime:** *float* 
  + The initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
* **FinalTime:** *float*
  + The final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
* **SpeciesName:** *string* 
  + The name of species that users want to examine, which should also be identical with the name written in the input (.inp and .mol) files.
* **DivideSize:** *int, optional = 2*
  + The value that distinguishes the size of the associate complex, for example, if DivideSize = 2, that means the associate events are classified as ‘associate size < 2’, ‘associate size = 2’ and ‘associate size > 2’.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

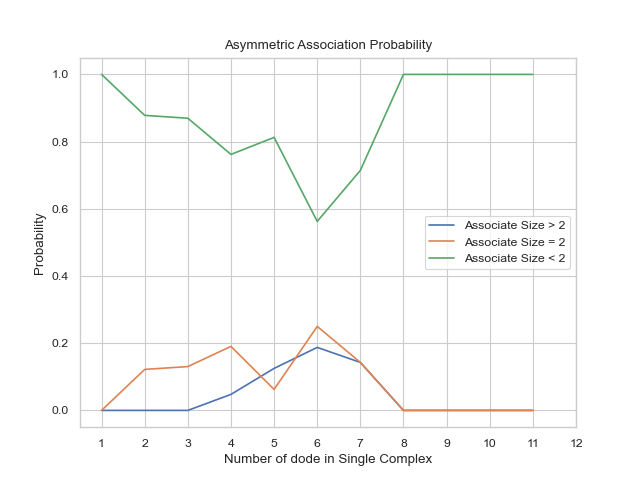
* cmplx\_size: list of each complex size.
* mean\_associate\_probability: the probability of a certain size of complex becoming another larger size.
* std: the standard deviation of mean probability.

Example:

cmplx\_size, mean\_associate\_probability, std =

ioNERDSS.associate\_prob\_asymmetric(FileName="ioNERDSSPyPi\TestingFunctions\\transition\_matrix\_time.dat",FileNum=1,InitialTime=0,FinalTime=1,SpeciesName="dode",DivideSize=2,ShowFig=True,SaveFig=False,SaveVars=True)

>>>



### 2.4.4 LINE PLOT - symmetric probability of dissociation between complex sizes:

**dissociate\_prob\_symmetric**(FileName, FileNum, InitialTime, FinalTime, SpeciesName, DivideSize, ShowFig, SaveFig, SaveVars)

Description:

This line plot represents the probability of dissociation of complexes of different sizes into other complexes of different sizes. The x-axis is the size of the complex and y-axis is the dissociate probability. Three lines will exist in the line graph, representing dissociating to complexes of sizes less than, equal to, or greater than the specified size, respectively. 'Symmetric' in the function name means that for the dissociate reaction, both sizes of complexes are counted as dissociating events symmetrically, for example, if an dissociate event occurs where a heptamer dissociates into a tetramer and a trimer, then this event is counted twice, which are heptamer dissociates to tetramer and heptamer dissociates to trimer. If multiple input files are given, the output plot will be the average value of all files and an error bar will also be included.

Parameters:

* **FileName:** *string*
  + The path to the ‘.dat’ file, which is usually named as ‘transition\_matrix\_time.dat’, representing the histogram data to be analyzed. If there are multiple, the files should be named [name]\_1.dat, [name]\_2.dat, etc., and the FileName should be just [name].dat
    - Example: transition\_1.dat, transition\_2.dat …. FileName = transition.dat
* **FileNum:** *int*
  + The number of the total input file. If multiple files are provided, their names should obey the naming rule.
* **InitialTime:** *float*
  + The initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
* **FinalTime:** *float*
  + The final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
* **SpeciesName:** *string*
  + The name of species that users want to examine, which should also be identical with the name written in the input (.inp and .mol) files.
* **DivideSize:** *int, optional = 2*
  + The value that distinguishes the size of the dissociate complex, for example, if DivideSize = 2, that means the associate events are classified as ‘dissociate size < 2’, ‘dissociate size = 2’ and ‘dissociate size > 2’.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

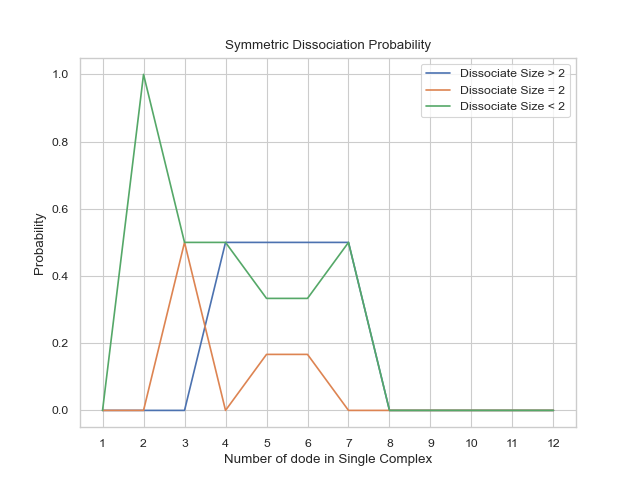
* cmplx\_size: list of each complex size.
* mean\_dissociate\_probability: the probability of a certain size of complex becoming another smaller size.
* std: the standard deviation of mean probability.

Example:

cmplx\_size, mean\_dissociate\_probability, std =

ioNERDSS.dissociate\_prob\_symmetric(FileName="ioNERDSSPyPi\TestingFunctions\\transition\_matrix\_time.dat",FileNum=1,InitialTime=0,FinalTime=1,SpeciesName="dode",DivideSize=2,ShowFig=True,SaveFig=False,SaveVars=True)

>>>



### 2.4.5 LINE PLOT - asymmetric probability of dissociation between complex sizes:

**dissociate\_prob\_asymmetric**(FileName, FileNum, InitialTime, FinalTime, SpeciesName, DivideSize, ShowFig, SaveFig, SaveVars)

Description:

This line plot represents the probability of dissociation of complexes of different sizes into other complexes of different sizes. The x-axis is the size of the complex and y-axis is the dissociate probability. Three lines will exist in the line graph, representing dissociating to complexes of sizes less than, equal to, or greater than the specified size, respectively. 'Asymmetric' in the function name means that for the dissociate reaction, only the complexes of smaller size dissociating from the original one is counted as dissociate event asymmetrically, for example, if an dissociate event occurs where a heptamer dissociates into a tetramer and a trimer, then this event is counted only once, which is heptamer dissociates to trimer. If multiple input files are given, the output plot will be the average value of all files and an error bar will also be included.

Parameters:

* **FileName:** *string*
  + The path to the ‘.dat’ file, which is usually named as ‘transition\_matrix\_time.dat’, representing the histogram data to be analyzed. If there are multiple, the files should be named [name]\_1.dat, [name]\_2.dat, etc., and the FileName should be just [name].dat
    - Example: transition\_1.dat, transition\_2.dat …. FileName = transition.dat
* **FileNum:** *int*
  + The number of the total input file. If multiple files are provided, their names should obey the naming rule.
* **InitialTime:** *float*
  + The initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
* **FinalTime:** *float*
  + The final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
* **SpeciesName** *string*
  + The name of species that users want to examine, which should also be identical with the name written in the input (.inp and .mol) files.
* **DivideSize:** *int, optional = 2*
  + The value that distinguishes the size of the associate complex, for example, if DivideSize = 2, that means the associate events are classified as ‘associate size < 2’, ‘associate size = 2’ and ‘associate size > 2’.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** bool, optional = False
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

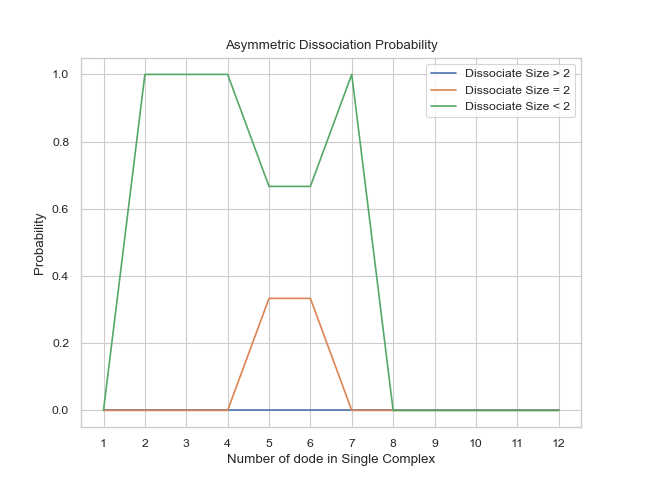
* cmplx\_size: list of each complex size.
* mean\_dissociate\_probability: the probability of a certain size of complex becoming another smaller size.
* std: the std of mean probability.

Example:

cmplx\_size, mean\_dissociate\_probability, std =

ioNERDSS.dissociate\_prob\_asymmetric(FileName="ioNERDSSPyPi\TestingFunctions\\transition\_matrix\_time.dat",FileNum=1,InitialTime=0,FinalTime=1,SpeciesName="dode",DivideSize=2,ShowFig=True,SaveFig=False,SaveVars=True)

>>>



### 2.4.6 LINE PLOT – Growth probability for each complex size:

**growth\_prob**(FileName, FileNum, InitialTime, FinalTime, SpeciesName, ShowFig, SaveFig, SaveVars)

Description:

This line plot indicates the probability of growth in size for different sizes of complexes. The x-axis is the size of complexes, and the y-axis is the growth probability. If multiple input files are given, the output plot will be the average value of all files and an error bar will also be included.

Parameters:

* **FileName:** *string*
  + The path to the ‘.dat’ file, which is usually named as ‘transition\_matrix\_time.dat’, representing the histogram data to be analyzed. If there are multiple, the files should be named [name]\_1.dat, [name]\_2.dat, etc., and the FileName should be just [name].dat
    - Example: transition\_1.dat, transition\_2.dat …. FileName = transition.dat
* **FileNum:** *int* 
  + The number of the total input file. If multiple files are provided, their names should obey the naming rule.
* **InitialTime:** *float* 
  + The initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
* **FinalTime:** *float*
  + The final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
* **SpeciesName:** *string*
  + The name of species that users want to examine, which should also be identical with the name written in the input (.inp and .mol) files.
* **ShowFig:** *bool, optional = True*
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Returns:

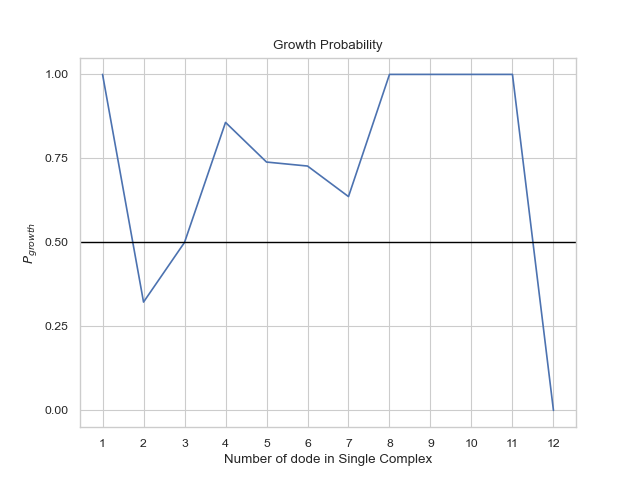
* cmplx\_size: list of each complex size.
* mean\_dissociate\_probability: the probability of a certain size of complex becoming another smaller size.
* std: the standard deviation of mean probability.

Example:

cmplx\_size, mean\_dissociate\_probability, std =

IoNERDSS.growth\_prob(FileName="ioNERDSSPyPi\TestingFunctions\\transition\_matrix\_time.dat",FileNum=1,InitialTime=0,FinalTime=1,SpeciesName="dode",ShowFig=True,SaveFig=False,SaveVars=True)

>>>



### 2.4.7 LINE PLOT – Average lifetime for each complex type:

**complex\_lifetime**(FileName, FileNum, InitialTime, FinalTime, SpeciesName, ShowFig, SaveFig, SaveVars)

Description:

This line plot indicates the lifetime for different sizes of complexes. The x-axis is the size of complexes, and the y-axis is its average lifetime in unit of second. If multiple input files are given, the output plot will be the average value of all files and an error bar will also be included.

Parameters:

* **FileName:** *string*
  + The path to the ‘.dat’ file, which is usually named as ‘transition\_matrix\_time.dat’, representing the histogram data to be analyzed. If there are multiple, the files should be named [name]\_1.dat, [name]\_2.dat, etc., and the FileName should be just [name].dat
    - Example: transition\_1.dat, transition\_2.dat …. FileName = transition.dat
* **FileNum:** *int*
  + The number of the total input file. If multiple files are provided, their names should obey the naming rule.
* **InitialTime:** *float*
  + The initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
* **FinalTime:** *float*
  + The final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
* **SpeciesName:** *string*
  + The name of species that users want to examine, which should also be identical with the name written in the input (.inp and .mol) files.
* **ShowFig:** *bool, optional = True* 
  + Whether the plot will be shown.
* **SaveFig:** *bool, optional = False*
  + Whether the plot will be saved as a ‘.png’ file
* **SaveVars:** *bool, optional = False*
  + Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

Example:

IoNERDSS. complex\_lifetime(FileName = "ioNERDSSPyPi\TestingFunctions\\transition\_matrix\_time.dat", FileNum = 1, InitialTime = 0.0, FinalTime = 1.00, SpeciesName = 'dode', ShowFig = True, SaveFig = False, SaveVars = False)

>>>



## 2.5 Locating position for certain size of complexes by PDB/restart file

### 2.5.1 By PDB file:

**locate\_pos\_no\_restart**(FileNamePdb, NumDict, FileNameInp, BufferRatio)

Description:

This function enables users to locate specific complexes of certain size from a PDB file after simulation. The result will be output as a new file named “output\_file.pdb” containing only the desired complex.

Parameters:

* **FileNamePdb:** *str* 
  + The path to the PDB file, which is usually the last frame of the simulation.
* **NumDict:** *dictionary* 
  + A dictionary that holds the requested number of protein types in a complex
* **FileNameInp:** *str*
  + The path to the '.inp' file, which usually stores the reaction information.
* **BufferRatio:** *float, optional = 0.01*
  + The buffer ratio used to determine whether two reaction interfaces can be considered as bonded.
* **OpName:** str, optional = “output\_file”
  + The name of the outputted file.

Returns:

* .pdb file with only the selected complexes

Example:

IoNERDSS. locate\_pos\_no\_restart(FileNamePdb = "ioNERDSSPyPi\TestingFunctions\\nerdss\_output.pdb", NumDict={"dod":9}, FileNameInp="ioNERDSSPyPi\TestingFunctions\parm.inp", OpName = “output”)

>>> *Output\_file.pdb that includes only proteins in complexes of the selected size*

...

ATOM 19 COM dod 3 301.720 116.470 306.361 0 0CL

ATOM 20 lg1 dod 3 315.636 126.000 315.231 0 0CL

ATOM 21 lg2 dod 3 312.386 125.024 293.086 0 0CL

....

### 2.5.2 By ‘restart.dat’ file:

**locate\_pos\_restart**(FileNamePdb, NumDict, FileNameRestart)

Description:

This function enables users to locate specific complexes of certain size from a PDB file along with ‘restart.dat’ file after simulation. The result will be output as a separated file named “output\_file.pdb” containing only the desired complex.

Important Note:

The advantage of reading the 'restart.dat' file is that the file directly stores the binding information of each complex in the system and can be used directly, so the function runs faster; however, the function is not universal, if the 'restart.dat ' file's write logic changes, then this function will no longer work.

Parameters:

* **FileNamePdb:** *str*
  + The path to the PDB file, which is usually the last frame of simulation.
* **NumDict:** *dictionary*
  + A dictionary that holds the requested number of protein types in a complex
* **FileNameRestart:** *str* 
  + The path to the 'restart.dat' file. Defaults to 'restart.dat'.
* **OpName:** *str, optional = “output\_file”*
  + The name of the outputted file.

Example:

IoNERDSS.locate\_pos\_restart(FileNamePdb = "ioNERDSSPyPi\TestingFunctions\\nerdss\_output.pdb", NumDict={"dod":9}, FileNameRestart="ioNERDSSPyPi\TestingFunctions\restart.dat", OpName = “output”)

>>> *Output\_file.pdb that includes only proteins in complexes of the selected size*

...

ATOM 19 COM dod 3 301.720 116.470 306.361 0 0CL

ATOM 20 lg1 dod 3 315.636 126.000 315.231 0 0CL

ATOM 21 lg2 dod 3 312.386 125.024 293.086 0 0CL

ATOM 22 lg3 dod 3 294.395 112.226 289.287 0 0CL

...

## 2.6 Analyzing .xyz files

.xyz files hold the location of every protein at specific times. (Does not necessarily include every timestamp, more to compare a couple of timestamps).

### 2.6.1 CSV – creates spreadsheet of protein locations

**xyz\_to\_csv**(FileName, LitNum)

Description:

This function enables users to convert the output .xyz file by NERDSS simulation into a .csv file of a specific or entire time frame. The generated csv file will contain 5 columns, including number of literation, species name, x, y, and z coordinates.

Parameters:

* **FileName:** *string*
  + The path to the .xyz file, which is usually named as ‘trajectory.xyz’.
* **LitNum:** *int, optional = -1*
  + The number of literation user desire to examine. If the input is -1, the function will extract the entire literation.
* **OpName:** *str, optional = “output\_file”*
  + The name of the outputted file.

Returns:

* Csv: .csv file with the specified trajectory data included.

Example:

IoNERDSS. xyz\_to\_csv(FileName="ioNERDSSPyPi\TestingFunctions\\trajectory.xyz", LitNum=-1)

>>>



### 2.6.2 DATAFRAME – creates dataframe of protein locations

**xyz\_to\_df**(FileName, LitNum, SaveCsv):

Description:

This function enables users to convert the output .xyz file by NERDSS simulation into a pandas.DataFrame of a specific or entire time frame. The generated csv file will contain 5 columns, including number of literation, species name, x, y and z coordinates.

Parameters:

* **FileName:** *string*
  + The path to the .xyz file, which is usually names as ‘trajectory.xyz’.
* **LitNum:** *int, optional = -1*
  + The number of literation user desire to examine. If the input is -1, the function will extract the entire literation.
* **SaveCsv:** *bool, optional = True*
  + Whether the corresponding .csv file will be saved

Returns:

* Trajectory dataframe: holds all the specificed trajectory information.

Example:

traj\_df =

IoNERDSS. xyz\_to\_df(FileName="ioNERDSSPyPi\TestingFunctions\\trajectory.xyz", LitNum=-1, SaveCsv = False)

>>> literation name x y z

0 0 ap 87.420620 -270.109172 -203.661987

1 0 ap 88.081526 -271.052470 -205.297038

2 0 ap 86.759715 -269.165874 -202.026936

3 0 ap -58.647113 277.528515 -353.236112

...

### 2.6.3 MATRIX - tracks the trajectory of specific protein(s)

**traj\_track**(FileName, SiteNum, MolIndex)

Description:

This function enables users to track the COM coordinate changing of one or more molecule. The return will be a 2D matrix with the size of the number of literation times the number of desired molecules.

Parameters:

* **FileName:** *string* 
  + The path to the .xyz file, which is usually names as ‘trajectory.xyz’.
* **SiteNum:** *int* 
  + The total number of COM and interfaces of a single molecule. For example, if a molecule possesses 1 COM and 5 interfaces, the SiteNum value should be 6.
* **MolIndex:** *list with int elements*
  + The index of molecule users desired to track. The number in the list should be no smaller than 1.
* **SaveVars:** *bool, optional = False* 
  + Whether the outputs are saved in a file

Returns:

* Trajectory: holds the COM coordinates of 1+ proteins at different time stamps.

Example:

trajectory =

IoNERDSS. traj\_track(FileName="ioNERDSSPyPi\TestingFunctions\\trajectory.xyz", SiteNum=3, MolIndex = [1,4,10])

>>>

[[[87.42062, -270.109172, -203.661987], [40.873538, 168.96348, -497.993163]],

[[74.407358, 51.461467, -242.958456], [187.824563, 325.913499, -497.993163]],

[[20.608487, 330.919045, -182.061499], [-27.367719, 330.945162, -497.993163]]]

## 2.7 Analyzing .pdb files

### 2.7.1 LINE PLOT- Auto correlation function for complexes

**acf\_coord**(PDBDirectory, mol\_list, sim\_num, time\_step, show\_fig, save\_fig)

Description:

Calculates the mean acf of protein complexes stored in a series of NERDSS generated PDB files. If pdb files from multiple simulations are intended to be evaluated, the NERDSS generated PDB files should be organized in the following way before running the function:

“Inputted directory name to the function”/

1/

…/

PDB/

(NERDSS generated PDB files from simulation #1; 0.pdb, 500.pdb, … etc.)

…/

2/

…/

PDB/

(NERDSS generated PDB files from simulation #2; 0.pdb, 500.pdb, … etc.)

…/

3/

…/

PDB/

(NERDSS generated PDB files from simulation #3; 0.pdb, 500.pdb, … etc.)

…/

…/

n/

…/

PDB/

(NERDSS generated PDB files from simulation #n; 0.pdb, 500.pdb, … etc.)

…/

Important Note:

Given a series of .pdb files generated during NERDSS simulation, the function calculates the auto correlation function (acf) of the system. The acf describes the correlation of a signal with a delayed copy of itself as a function of delay. In our case, it is the correlation of a complex’s position with its initial position as a function of time, calculated as the inner product between the initial position vector of a complex and its current position vector divided by that squared magnitude of its initial position vector:

acf(t) = dot(r(0), r(t))/dot(r(0), r(0))

Parameters:

* **PDBDirectory:** *string* 
  + The name of the directory where all PDB files are stored.
* **mol\_list:** *list of strings*
  + The name of the molecules intended to be evaluated.
* **sim\_num:** *int, optional = 1*
  + The number of repeated simulations intended to be evaluated.
* **time\_step:** *int, optional = 1*
  + The time steps of the NERDSS simulation in micro-seconds. If not input, the default is 1 micro-seconds.
* **show\_fig:** *bool, optional = True*
  + Whether generated plots will be shown.
* **save\_fig:** *bool, optional = False*
  + Whether generated plots will be saved.

Returns:

* average\_time\_array: array of iterations when mean of acf over all repeated simulations are calculated.
* average\_acf\_array: array of mean of calculated acf over all repeated simulations.
* std\_acf\_array: array of standard deviation of calculated acf over all repeated simulations.

Example:

IoNERDSS.acf\_coord(PDBDirectory = “unimportant/TestingFunctions/testing\_PDBs”, mol\_list = [“A”], sim\_num = 2, time\_step = 0.1, show\_fig = True, save\_fig = False)

A graph of blue lines

Description automatically generated with medium confidence

A graph showing the value of a stock market

Description automatically generated

## 2.8 Gag Sphere

### 2.8.1 Reshape Gag

**reshape\_gag**(PathName)

Description:

Constructs a model of the gag monomer, experimentally measured gag lattice structures must be regularized to eliminate thermal fluctuations and other experimental errors. This function reshapes and regularizes experimentally measured gag lattice structures.

Parameters:

* **PathName:** *string*
  + The path of the .pdb file.

Returns:

* finalPositionsVec(list 144 x 3): the coordinate of the COM and 5 interfaces for each of the 18-gag monomer.

Example:

reshape\_gag(“unimportant/TestingFunctions/7asl\_sites.pdb”))

A screenshot of a computer

Description automatically generated

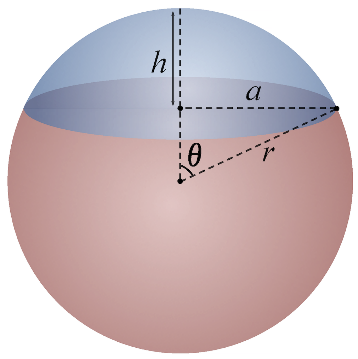
### 2.8.2 Sphere Regularization Index

**sphere\_regularization\_index**(PathName, IterNum, TimeStep, ComplexNum, SpeciesName)

Description:

Calculates the sphere regularization index of a complex yielded from NERDSS.

Important Note:



In the sphere assembled by gag proteins, each gag monomer can be viewed as a sphere cap contributing a portion of the sphere. Ideally, the gag monomers would self-organize themselves exactly on the surface of the ideal sphere and become “spherical caps” that perfectly comprise of the sphere. However, this assembly may not be perfect in real life. The theta of each gag sphere cap can be either greater than or less than the ideal theta of the sphere. In this case, the sphere regularization index is calculated as the ratio of gag monomers that have their theta less than or equal to the ideal theta in an assembled complex. This function calculated the regularization index of the complexes formed in the simulation.

The function calculates the sphere regularization index of the first, second, and third largest gag complex at iteration 200000. It requires the histogram\_complexes\_time.dat, restart####.dat, ####.pdb from the NERDSS simulation in this arrangement (#### being the iteration number). The regularization index is calculated as the ratio of gag monomers that have their cap theta less than or equal to the ideal theta of the gag sphere in an assembled complex.

“PathName”/

…/

histogram\_complexes\_time.dat

PDB/

####.pdb

RESTARTS/

restart####.dat

…/

…

Which is the default arrangement NERDSS output after a simulation is run.

Parameters:

* **PathName:** *string*
  + The path of the histogram file, PDB, and restart file.
* **IterNum:** *int*
  + The iteration number of the simulation intended to be evaluated.
* **TimeStep:** *float*
  + The time step of the NERDSS simulation, in micro-seconds.
* **ComplexNum:** *int*
  + The number of complexes to be analyzed, starting from the largest complex size.
* **SpeciesName:** *string, optional = “gag”*
  + The name of the species to be analyzed.

Returns:

* max\_complex\_size\_return (list): the complex size of each complex evaluated (1 indicates monomer, etc).
* theta\_ideal\_return (list): the ideal spherical angle of each complex.
* sphere\_radius\_return (list): the sphere radius of each complex.
* complex\_COM\_return (list): the center of mass of each complex.
* regularization\_index\_return (list): the regularization index of each complex.

Example:

IoNERDSS.sphere\_regularization\_index(PathName = “unimportant/TestingFunctions/gagsphere”, IterNum = 200000, TimeStep = 0.1, ComplexNum = 3, SpeciesName = “gag”)

A screenshot of a computer

Description automatically generated

# 3. Merging results from restart simulations

Merging files and results from multiple restart simulations.

### 3.1.1 Merging NERDSS restart simulations output files

**merge\_simulation\_results**()

Description:

Arranges simulations and restart simulations in the following way:

1/

PDB/

RESTARTS/

bound\_pair\_time.dat

copy\_number\_time.dat

event\_counters\_time.dat

histogram\_complexes\_time.dat

transition\_matrix\_time.dat

restart0001/

PDB/

RESTARTS/

bound\_pair\_time.dat

copy\_number\_time.dat

event\_counters\_time.dat

histogram\_complexes\_time.dat

transition\_matrix\_time.dat

restart0002/

PDB/

RESTARTS/

bound\_pair\_time.dat

copy\_number\_time.dat

event\_counters\_time.dat

histogram\_complexes\_time.dat

transition\_matrix\_time.dat

restart0003/

…

2/

…

Then runs the function. The function will merge the bound\_pair\_time.dat, copy\_number\_time.dat, event\_counters\_time.dat, histogram\_complexes\_time.dat, transition\_matrix\_time.dat as well as the PBD directory and the RESTART directory of the original simulation and all its restart simulations. All restart#### directories will be removed after merging.

Important Note:

First, run the function in the directory containing 1/, 2/, …etc. Second, assign names to the restart#### directories according to the order of their first recorded iterations, i.e. the restart simulation data that begins at time=0.5, so it should be stored in restart0001 and the restart simulation data that begins at time=1.0 should be restart0003 but not the other way around.

Parameters:

* None

Returns:

* None

### 3.1.2 Merging NERDSS output files

**merge\_files**(destination\_file, source\_file, file\_type)

Description:

Merges NERDSS output files. If the parameter of file\_type is not input, the function will simply concatenate the source\_file to the end of the destination file. If one of the given file types is input, the function will correctly keep the iterations in the destination file that happened before the first recorded iteration in the source file and concatenate the data from the source file to the end of the kept iterations in the destination file. For instance, the destination file contains iterations from time=0.0 to time=1.0, and the source file contains iterations from time=0.5 to time=100.0. Calling this function and correctly inputting the filetypes will keep the iterations from time=0.0 to the last iteration before time=0.5 in the destination file and concatenate iterations from time=0.5 to time=100 to the destination file.

Parameters:

* **destination\_file:** *string*
  + The path of the destination file.
* **source\_file:** *string* 
  + The path of the source file.
* **file\_type:** *string*
  + The file type of the files to be merged.
    - Options: “histogram” (histogram\_complexes\_time.dat), “event” (event\_counters\_time.dat), “bound” (cound\_pair\_time.dat), “copy” (copy\_numbers\_time.dat), “transition” (transition\_matrix\_time).

Returns:

* None