Table of Contents

0. 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.
- 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 (1) 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 initilized 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 it's 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.
**NOTE: Each section that is *italicized* means it will not be included in every function

Function Format:
[ObjectName if it is a method].[function-name] ([function parameters])
Desc: [short description]
Parameters:

- [parameter-name] ([parameter type*], [if-optional] = [value-if-unset]):* [Description]
- *This will be repeated for each parameter*

Long Description: [this will be a longer description if necessary]
Example: [shows example of function being used, possibly with images]

1. 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.

1.1 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 here

1.2 Database PDB

These functions will use .PDB files downloaded from the protein databank 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 write **PDB_UI()** into command line or IDE.

Tutorial:

First, store a PDB file in the same path with the python code and call this function with an appropriate IDE (here take 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 and press return to continue. See screenshot below for details.

```
Enter pdb file name: 1utc.pdb
```

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.

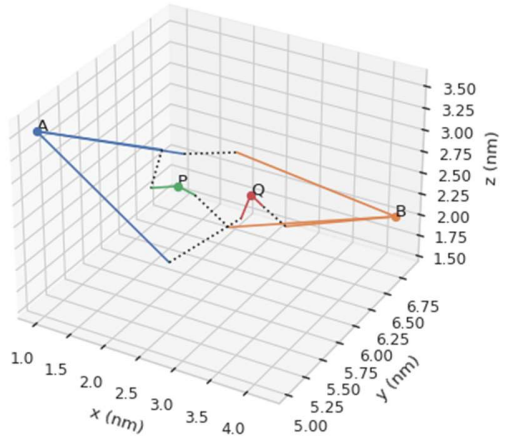
```
Finish reading pdb file
4 chain(s) in total: ['A', 'B', 'P', 'Q']
Each of them has [2754, 2763, 66, 66] atoms.
Center of mass of  A is: [0.975, 5.018, 3.545]
Center of mass of  B is: [4.214, 6.886, 2.019]
Center of mass of  P is: [2.635, 5.307, 3.115]
Center of mass of  Q is: [2.555, 6.419, 2.222]
Interaction site of A & B is: [1.160, 5.167, 3.336] and [4.030, 6.671, 2.111] distance between interaction sites is: 3.463 nm
Interaction site of A & P is: [1.118, 5.144, 3.415] and [2.705, 5.350, 3.071] distance between interaction sites is: 1.637 nm
Interaction site of A & Q is: [1.202, 5.405, 3.244] and [2.486, 6.372, 2.253] distance between interaction sites is: 1.888 nm
Interaction site of B & P is: [3.990, 6.526, 2.340] and [2.705, 5.350, 3.071] distance between interaction sites is: 1.889 nm
Interaction site of B & Q is: [4.066, 6.742, 2.123] and [2.486, 6.372, 2.253] distance between interaction sites is: 1.628 nm
Interaction site of P & Q is: [2.705, 5.350, 3.071] and [2.486, 6.372, 2.253] distance between interaction sites is: 1.327 nm
```

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.

```
Would you like to change the distance between interaction site (Type 'yes' or 'no'): yes
Which distance would you like to change (please enter an integer no greater than 6 or enter 0 to set all distance to a specific number): 0
Please enter new distance: 1.5
New interaction site of A & B is: [1.973, 5.593, 2.989] and [3.216, 6.244, 2.458] distance between interaction sites is: 1.500
New interaction site of A & P is: [1.184, 5.153, 3.400] and [2.638, 5.342, 3.085] distance between interaction sites is: 1.500
New interaction site of A & Q is: [1.334, 5.504, 3.142] and [2.354, 6.272, 2.355] distance between interaction sites is: 1.500
New interaction site of B & P is: [3.858, 6.405, 2.415] and [2.837, 5.471, 2.996] distance between interaction sites is: 1.500
New interaction site of B & Q is: [4.004, 6.727, 2.129] and [2.548, 6.386, 2.248] distance between interaction sites is: 1.500
New interaction site of P & Q is: [2.719, 5.283, 3.124] and [2.472, 6.438, 2.199] distance between interaction sites is: 1.500
Would you like to change the distance between interaction site (Type 'yes' or 'no'): yes
Which distance would you like to change (please enter an integer no greater than 6 or enter 0 to set all distance to a specific number): 1
Please enter new distance: 3.5
New interaction site of A & B is: [1.145, 5.159, 3.342] and [4.045, 6.678, 2.105] distance between interaction sites is: 3.500
Would you like to change the distance between interaction site (Type 'yes' or 'no'): no
Calculation is completed.
```

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.

```
Display a 3D plot of the protein complex? (Type 'yes' or 'no'): yes
```



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.

```
Do you want each chain to be centered at center of mass? (Type 'yes' or 'no'): yes
```

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

IMPORTANT: All future functions are methods of the ProteinComplex object, and are the sub-functions of the UI function mentioned above.

1.2.2 MAIN OBJECT - Read in PDB File to create protein complex object (PLEASE READ!!)

ProteinComplex(FileName, ChainsIncluded, MaxBoundLength, SymmetryApplied)
Description: Reads in a database PDB file and finds information important for NERDSS inputs. 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)
  Description: 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.

Note:

- Calc_angle() is a very powerful function that will result in some functions being unable to be run, and others to be run. Here is the list of both. All others can be run whenever.
- Can NOT be run after calc_angle(): filter, change_sigma
- Must be run after calc_angle(): write_input, norm_COM

Example:

```
Clathrin = ioNERDSS.ProteinComplex(FileName=”ioNERDSS\Test\database.pdb”,
ChainsIncluded=['A’,’ B’,’ P’,’ Q’]
>>> Creates a Protein Complex object called Clathrin that holds the data in the database file
```

Long Description: 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.

1.2.3 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 String elements): The desired name of chains that users intend to examine.

Example:
Clathrin.filter(ChainsIncluded=['A',' B'])
>>> Edits clathrin so it only included the A and B chain

1.2.4 MODIFY - Change the distance between 2 binding sites

ProteinComplex.change_sigma(ChangeSigma, SiteList, NewSigma)
Description: Changes the value of sigma, the distance between two binding interfaces. 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])
>>> Chanes distance between first and second chain to 1.01 nanometers

Long 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.

1.2.5 MODIFY - Calculate the angles between pairs of interfaces

ProteinComplex.calc_angle()
Description: Calculates the 5 associating angles of each pair of interfaces. After this runs, it is ready to be output. Will result in some functions being runable, and other functions being unrunable (on this object), be careful!
Parameters:

- None

Example:

Clathrin.calc_angle()
>>> Finds the 5 associating angles for each interface. Ready to be output now.

Long 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 'real_PDB_UI', the separated function does not support manual inputs. The returns will contain all the information for further analysis.

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. 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.

Long 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.

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. Must be run after calc_angle().
Parameters:

- None

Example:
Clathrin.write_input()
>>> Creates new .inp and .mol files ready to be input into NERDSS

Long 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 same directory as the Jupyter Notebook file once the function finish running.

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.
Parameters:

- None

Example:

```
Clathrin.write_PDB()
>>> Creates new .pdb files to be compared with original inputted .pdb file
```

Long 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. Besides, the unit for the coordinates in PDB file is in Angstrom but not nm, so the value will be 10 times larger than that in NERDSS input files.

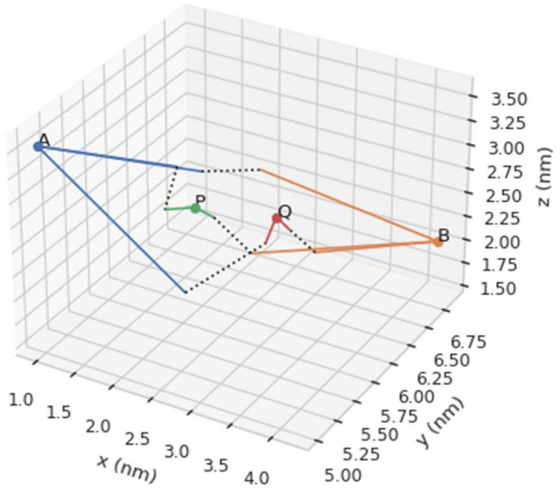
1.2.9 OUTPUT - Create a 3D plot of each inputted chain

ProteinComplex.plot_3D ()
Description: Generates a 3D plot indicating the spatial geometry of each simplified chain.
Parameters:

- None

Example:

```
Clathrin.plot_3D()
```



Long 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.

1.3 Platonic Solid Self-assembly Input File Writing

Platonic solid self-assembly include 10 models, so that 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): It is 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): It is 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: regularizing symmetric protein cages made of repeated subunits recorded in .pdb files to create one set of NERDSS input parameters

Parameters:

- PathName(string): 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(a 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 inputted, it will assume that the .pdb file only contain one kind of proteins.

Example:

```
repeated_protein_subunit_regularization(PathName = " ioNERDSS\Test\7as1_sites.pdb ")
```

```
Sphere center position [x,y,z] and radius [R] are, respectively:
 [ 16.32971401  16.33159668 -31.7905926   49.33703142]


Chains: ['A' 'B' 'C' 'D' 'I' 'N' 'E' 'J' 'O' 'F' 'K' 'P' 'G' 'L' 'Q' 'H' 'M' 'R']
Interfaces at: [x, y, z]
[[-1.01439254e+00  5.36390848e-01 -1.44560290e-17]
 [-4.41526634e-01 -1.07912306e+00 -3.01507070e-01]
 [-1.50446802e+00  5.78488504e-02 -1.39608625e+00]
 [ 1.98336126e+00 -1.37515022e+00  9.01916567e-01]
 [ 2.32282393e+00  7.12145237e-01  1.10701159e+00]]


With normal vector (1,0,0),
The binding parameters for the interfaces in chains ['A' 'B' 'C' 'D' 'I' 'N' 'E' 'J' 'O' 'F' 'K' 'P' 'G' 'L' 'Q' 'H' 'M' 'R'] are:
 [ sigma,      theta1,     theta2,     phi1,       phi2,       omega]
[[ 0.79468511  2.60111286  1.71772077  2.2458037  -3.10896605  0.45614352]
 [ 0.7291351   2.18563697  2.17197659 -1.80265058 -1.85761042 -2.25272529]
 [ 0.79468511  1.71772077  2.60111286 -3.10896605  2.2458037   0.45614352]
 [ 0.84923009  2.60534104  1.63218867 -0.00938971  0.45644332 -0.22648114]
 [ 0.86962093  1.59428579  2.63806666  0.46755012 -0.00886112 -0.22754121]]
```

Long 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.

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

gui()

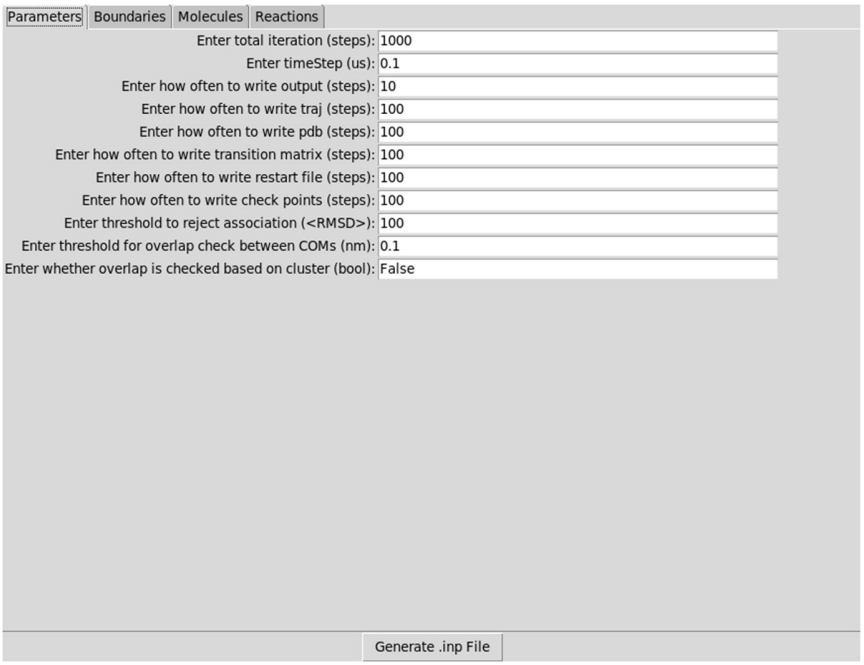
Description: using user inputted values to create .inp and .mol NERDSS input files

Parameters:

- None

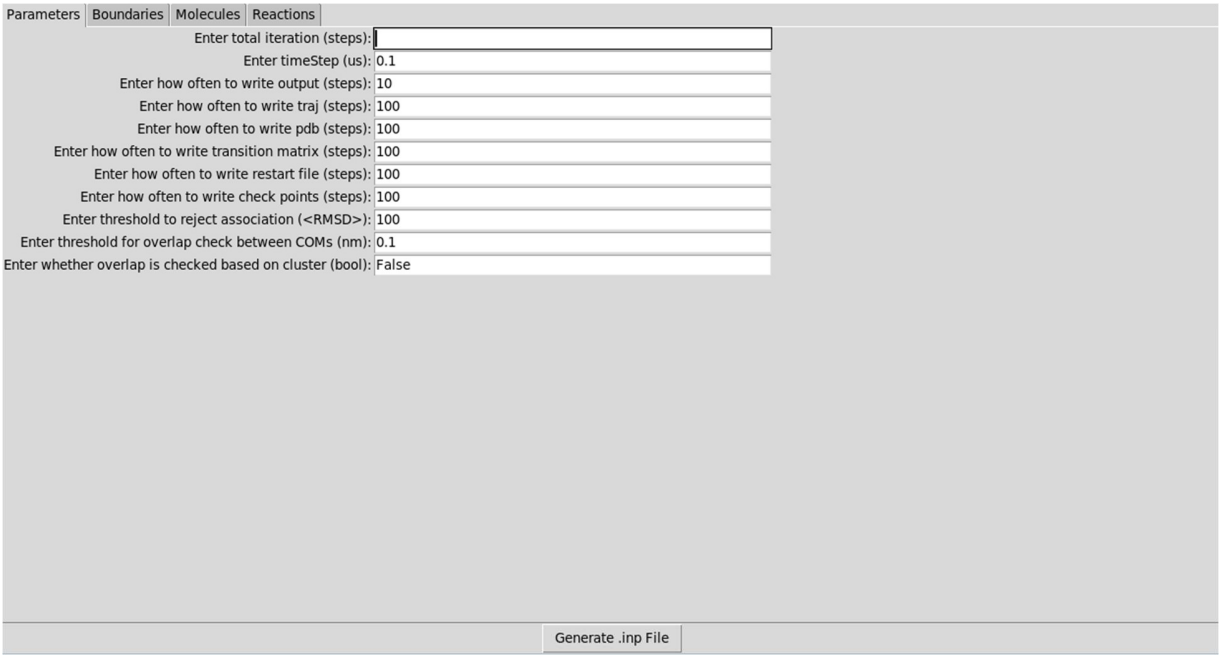
Example:

```
import ioNERDSS as io
io.gui()
```



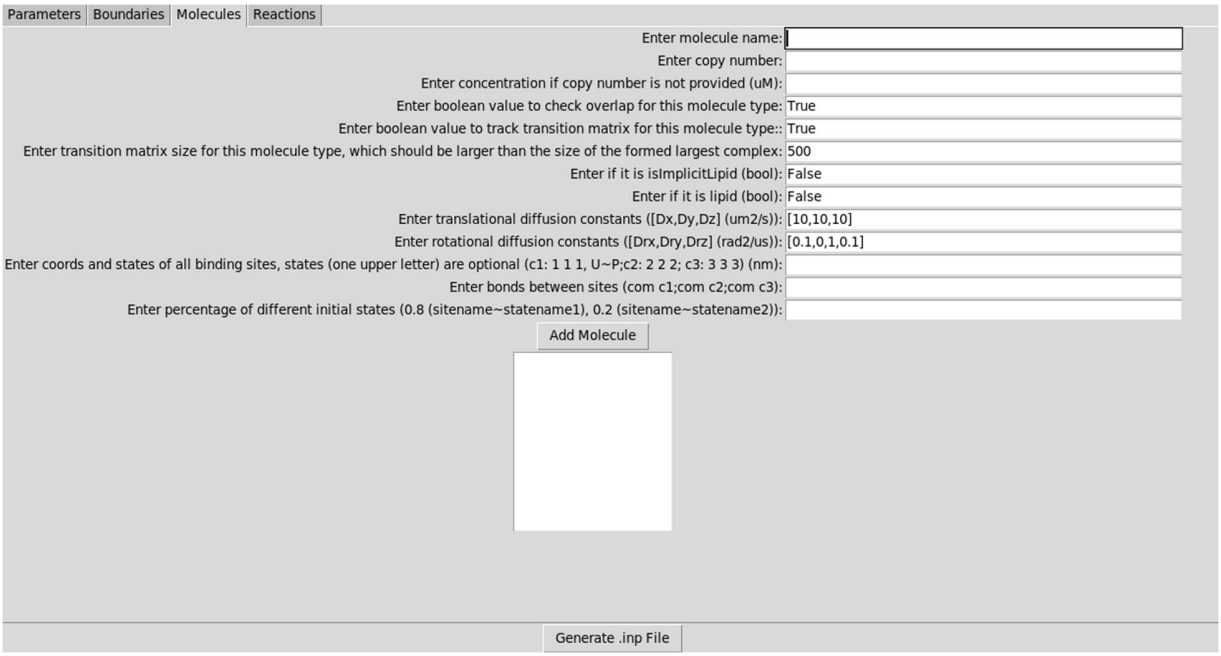
Long Description:
Graphic interface to generate .inp and .mol NERDSS input files.

General tutorial:



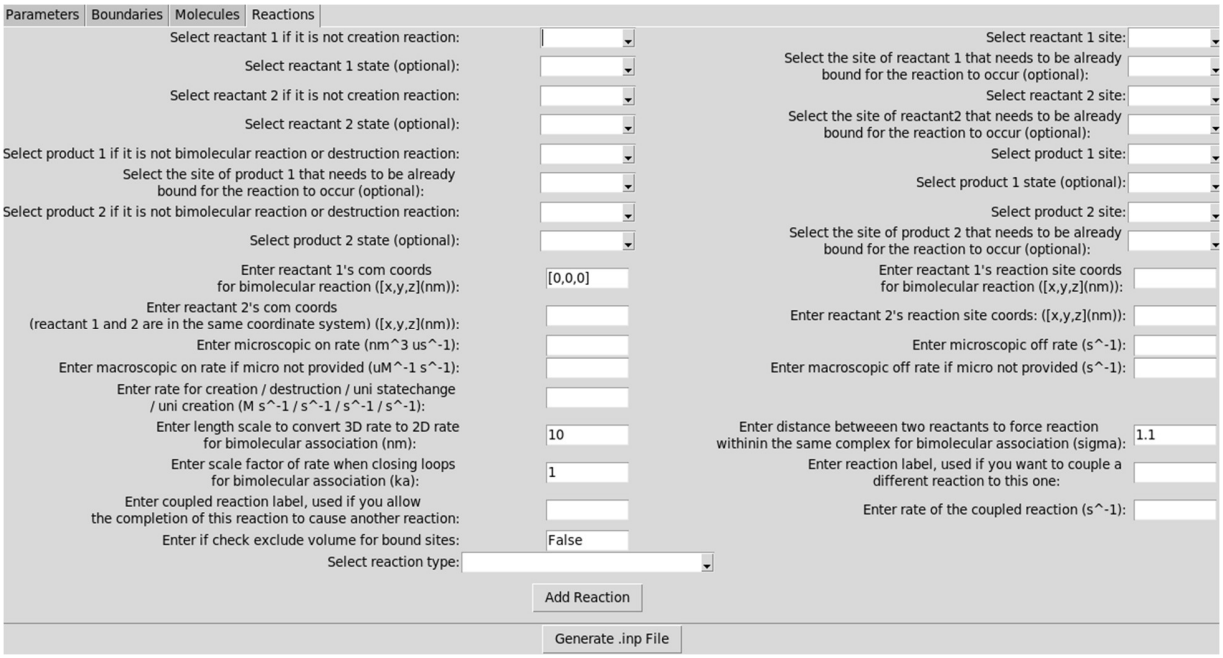
In the parameters tab, enter the intended parameters for the simulation. For a more detailed description of what these parameters do, please refer to the NERDSS_USER_GUIDE_MASTER available in the github NERDSS repository.

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.



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.



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 dataframes, 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 initilization functions and functions included in both objects.

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

2.1.1 MAIN OBJECT – The Main Single-component Histogram Object (PLEASE READ!!)

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

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

Parameters:

- FileName (String)It is 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): It is the number of the total input file. If multiple files are provided, their names should obey the naming rule listed below.
- InitialTime (Float): It is the initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
- FinalTime (Float): It is the final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
- SpeciesName (String): It is 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 (PLEASE READ!!)

MultiHistogram(FileName, FileNum, InitialTime, FinalTime, SpeciesName)
Description: An object that holds all of the data from a multi species histogram file, and allows for easy interpretation of the data

Parameters:

- FileName (String)It is 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): It is the number of the total input file. If multiple files are provided, their names should obey the naming rule listed below.

- InitialTime (Float): It is the initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
- FinalTime (Float): It is 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

Single/MultiHistogram.hist_to_df (FileNum, SaveCsv)
Description: Converts the raw .dat file to a data frame in python pandas package.
Parameters:

- OpName (str, optional = "histogram"): The name of the output .csv file. If SaveCsv is false, this parameter is irrelelvent, however 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
```

Long 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.

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

Single/MultiHistogram.hist_to_csv (FileNum)
Description: This function enables users to convert the raw .dat file to a .csv file. 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")
>>>
```

Time(s):	A: 1.	A: 1. B: 1.	A: 1. B: 2.	A: 1. B: 3.	A: 1. B: 4.	A: 1. B: 5.	A: 1. B: 6.	A: 10. B: 1	A: 2.	A: 2. B: 1.
0	100	0	0	0	0	0	0	0	0	0
0.001	86	4	1	0	0	0	0	0	2	1
0.002	76	3	1	1	0	0	0	0	5	3
0.003	67	5	2	1	0	0	0	0	5	3
0.004	61	4	2	1	0	0	0	0	6	1
0.005	56	3	2	1	0	0	0	0	7	2
0.006	51	6	2	1	0	0	0	0	7	3
0.007	43	5	2	1	0	0	0	0	7	4

Long Description: This function enables users to convert the raw .dat file to a .csv file 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.

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

Single/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 period.
Parameters:

- SpeciesName (Str, Optional-ish): 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:

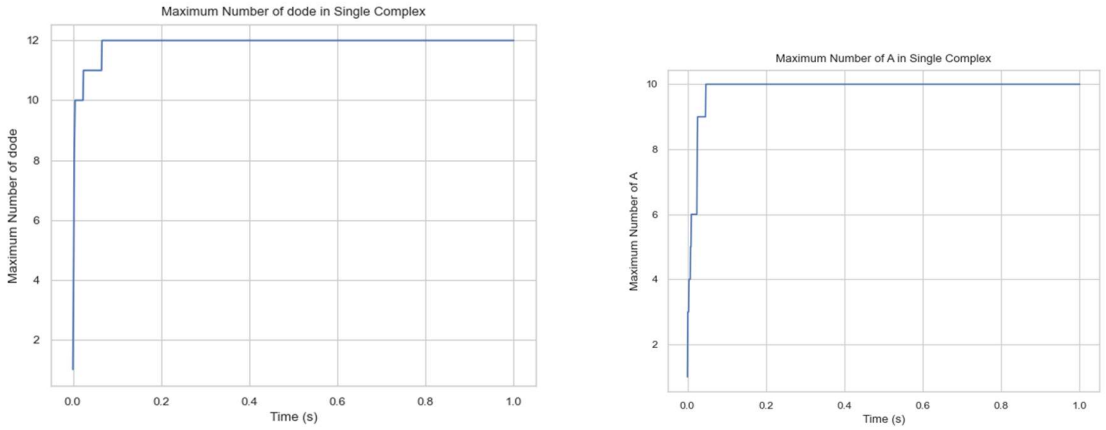
- time_stamps: the time stamps included in the line graph
- max_cmplx_size: list of the max complex size at each time stamp
- std: standard deviation of each max_cmplx_size

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)
>>>
```



Long Description: Will create a histogram where the X-axis is time, and Y-axis is the largest # of the tracked protein in a single complex

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

Single/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 period.
Parameters:

- SpeciesName (Str, Optional-ish): 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

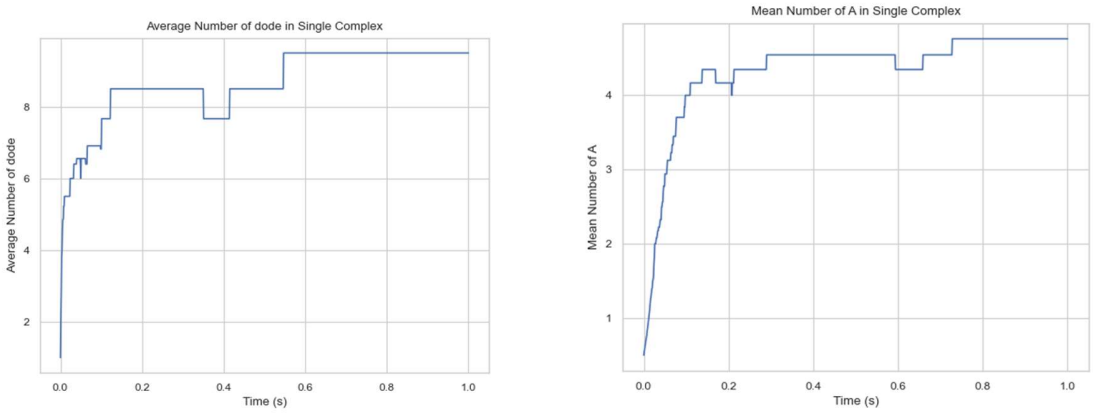
- SaveVars (Bool, Optional = False): Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

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:
t time_stamps, mean_cmpx_size, std =
est_histogram.line_mean_complex_size(SpeciesName='A',ShowFig = True, SaveFig =
False,SaveVars=False)
>>>
```



Long Description: Will create a histogram where the X-axis is time, and Y-axis is the average # of the tracked protein in a single complex

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
Parameters:

- BarSize (Int, Optional = 1): It is 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'.

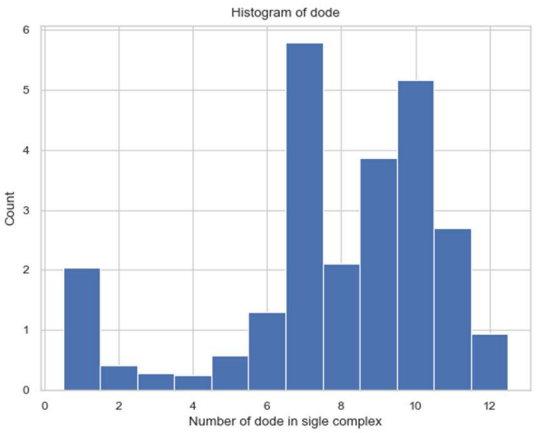
Naming Rule: If single file is provided, the input file should be named as its original name ('histogram_complexes_time.dat'); if multiple files are provided, the name of input file should also include serial number as 'histogram_complexes_time_X.dat' where X = 1,2,3,4,5…

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)
>>>
```



Long Description: Will create a histogram where the X-axis is each complex species type (each size), and Y-axis is the average count over the time period (Initial to Final)

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

SingleHistogram.hist_3d_complex_count (TimeBins, xBarSize, ShowFig, SaveFig, SaveVars)
Description: Generates a 3D histogram representing the number of each type of complex over time.

Parameters:

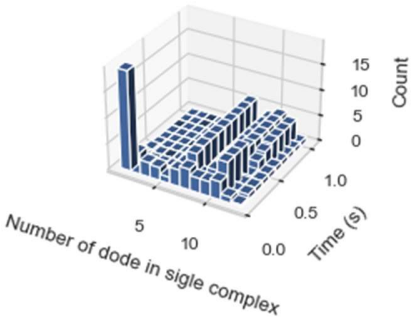
- TimeBins (Int): It is the number of bins that users want to divide the selected time period into. The value should be a positive integer.
- xBarSize (Int, Optional = 1): It is the size of each data bar in x-dimension. The x-axis will be separated evenly according to this number and the count of each 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)
>>>
```



Long 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.

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

SingleHistogram.heatmap_complex_count (TimeBins, xBarSize, ShowFig, ShowMean, ShowStd, SaveFig, SaveVars)
Description: Generates a 2D histogram of numerical distribution of different complex sizes vs. time. corresponding time period is reached.
Parameters:

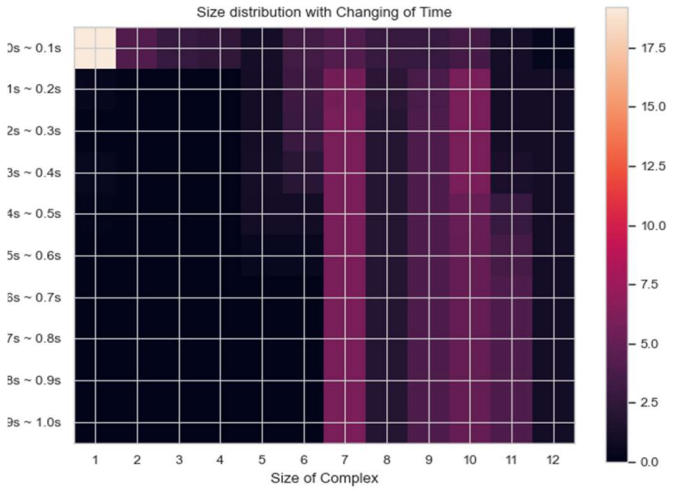
- TimeBins (Int): It is the number of bins that users want to divide the selected time period into. The value should be a positive integer.
- xBarSize (Int, Optional = 1): It is the size of each data bar in x-dimension. The x-axis will be separated evenly according to this number and the count of each 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)
>>>
```



Long 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 period. The color in each box indicates the number of corresponding N-mers when corresponding time period is reached.

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: Generates 2D histogram of total count of monomers in different complex species over time.
Parameters:

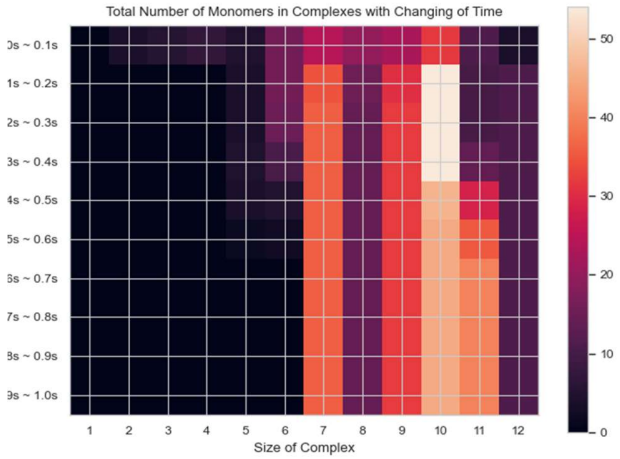
- TimeBins (Int): It is the number of bins that users want to divide the selected time period into. The value should be a positive integer.
- xBarSize (Int, Optional = 1): It is the size of each data bar in x-dimension. The x-axis will be separated evenly according to this number and the count of each 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_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)
>>>
```



Long 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 period. The color in each box indicates the total number of corresponding monomers in N-mers when corresponding time period is reached.

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.
Parameters:

- TimeBins (Int): It is the number of bins that users want to divide the selected time period into. The value should be a positive integer.
- xBarSize (Int, Optional = 1): It is 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’.

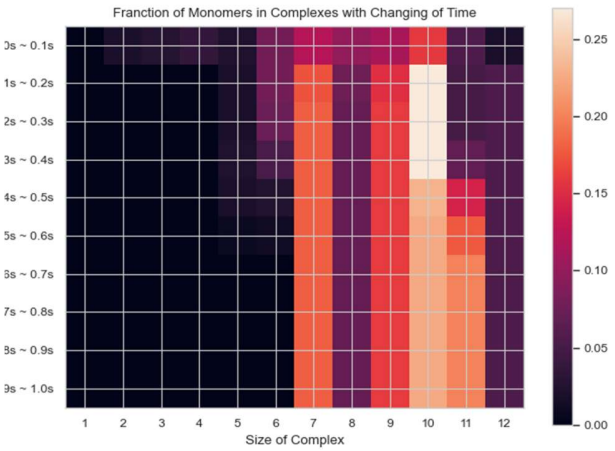
Returns:

- 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)
>>>
```



Long Description: 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 period. The color in each box indicates the fraction of monomers forming corresponding N-mers when corresponding time period is reached.

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: Creates a general histogram of total size of complex or selected species inside each complex for a multi-species system.
Parameters:

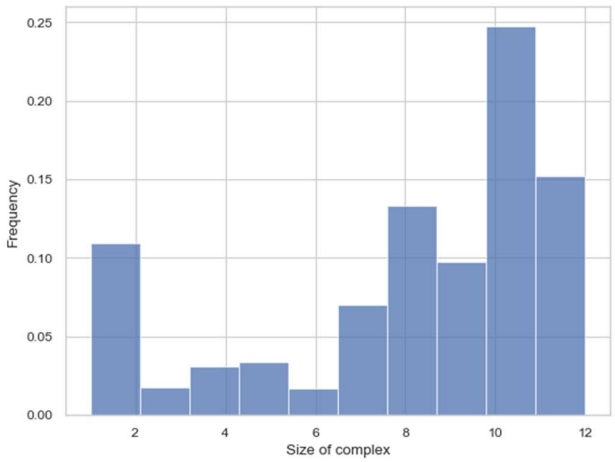
- BinNums: (Int, Optional = 10): Number of bins in the histogram.
- ExcludeSize (Int, Optional = 0): In the generated plot, the number of monomers in the complex that are no larger than this number will be excluded and will not be considered into the average calculation.
- ShowFig (Bool, Optional = True): 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)
>>>
```



Long 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.

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: Plots general histogram of total size of selected species for a multi-species system. Each bar is split into three stacked bars which represent the size distribution of another selected species compared to a desired input.
Parameters:

- xAxis (String): It 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): It indicates the name of the species that users want to separate by size.

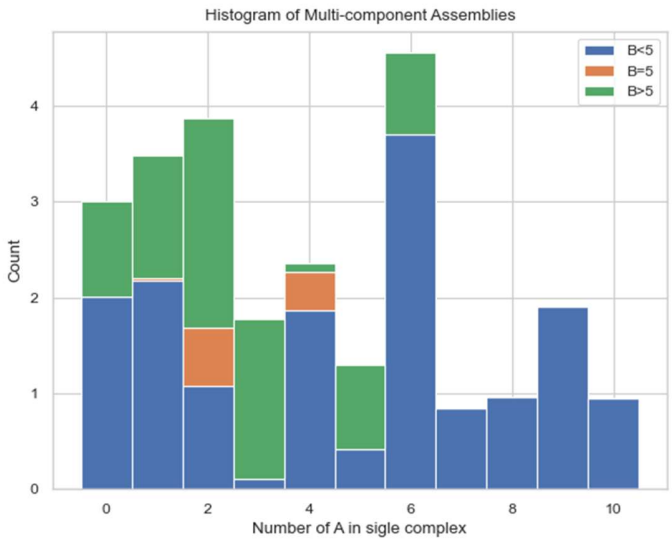
- DivideSize (Int): This is the value that separates the size of the dissociate complex, for example, if DivideSize = 5, that means the dissociate events are classified as ‘DivideSpecies size < 5’, ‘DivideSpecies size = 5’ and ‘DivideSpecies size > 5’.
- BarSize (Int, Optional = 1): It is size of each 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)
>>>
```



Long 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.

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: Creates a heatmap during a certain time period representing the distribution of size of selected species.
Parameters:

- xAxis (String): It 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): It 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): It is 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): It is 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’.

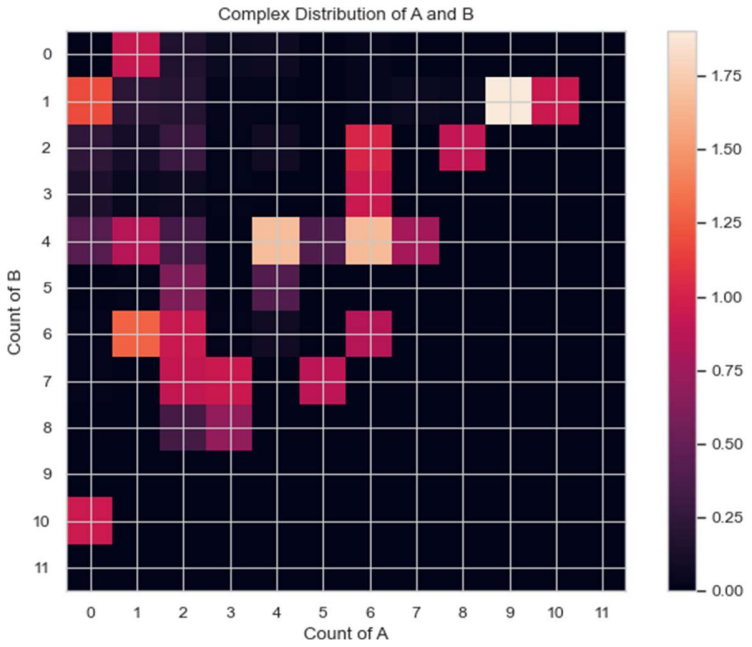
Returns:

- 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)
```

>>>



Long Description: This function enables users to generate a heatmap during a certain time period 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.

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

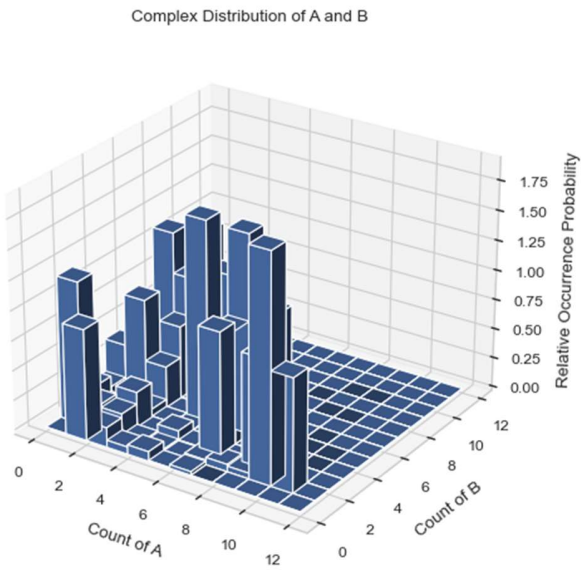
MultiHistogram.hist_3D_complex_dist(xAxis, yAxis, xBarSize, yBarSize, ShowFig, SaveFig, SaveVars)
Description: Generates a 3D histogram during a certain time period representing the distribution of size of selected species.
Parameters:

- xAxis (String): It 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): It 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): It is 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): It is 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.
- 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)



Long Description: Generates a 3D histogram during a certain time period 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.

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 (str): 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.

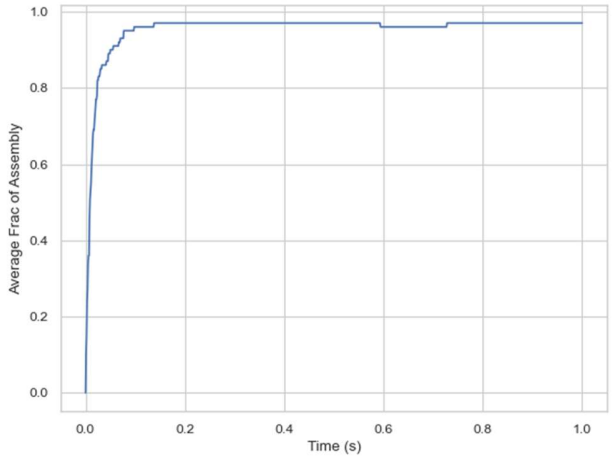
- SaveVars (Bool, Optional = False): Whether the values will be saved in text / .csv files. Will be put into a folder called ‘vars’.

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 selected time period among different size of complexes.

Parameters:

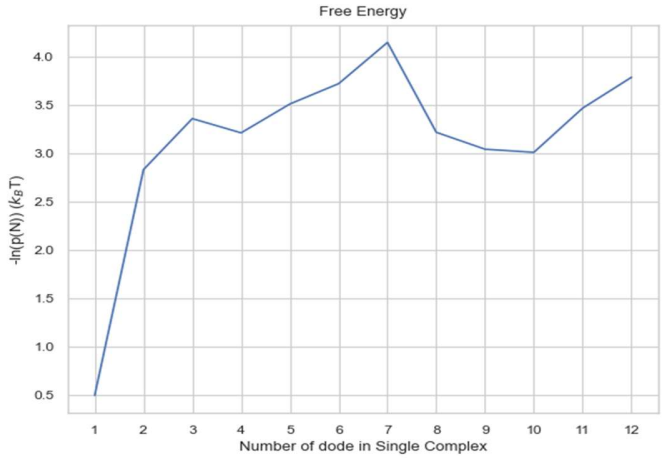
- FileName (String): It is the path to the '.dat' file, which is usually named as 'transition_matrix_time.dat', representing the histogram data to be analyzed.
    - If there are multiple, the files should be named [name]_1.dat, [name]_2.dat, ect. And the FileName should be just [name].dat
    - Example: transition_1.dat, transition_2.dat …. FileName = transition.dat
- FileNum (Int): It is the number of the total input file. If multiple files are provided, their names should obey the naming rule.
- InitialTime (Float): It is the initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
- FinalTime (Float): It is the final time in **seconds** that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
- SpeciesName (String): It is 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 = 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_energy: the –ln(p(N)) in K_bT of each complex
- std: the std 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=Fals
e,SaveVars=True)
>>>
```



Long Description: The plot indicates the change in free energy in selected time period 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.

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. 'Symmetric' in the function name means that for the associate reaction, both sizes of complexes are counted as associating events symmetrically.

Parameters:

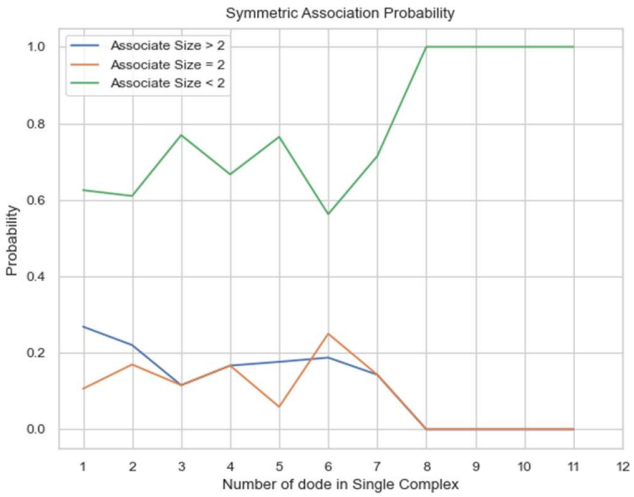
- FileName (String): It is the path to the '.dat' file, which is usually named as 'histogram_complexes_time.dat', representing the histogram data to be analyzed.
    - If there are multiple, the files should be named [name]_1.dat, [name]_2.dat, ect. And the FileName should be just [name].dat
    - Example: transition_1.dat, transition_2.dat …. FileName = transition.dat
- FileNum (Int): It is the number of the total input file. If multiple files are provided, their names should obey the naming rule.
- InitialTime (Float): It is the initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
- FinalTime (Float): It is the final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
- SpeciesName (String): It is 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): This is 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 std 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)
>>>
```



Long 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.

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. '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.

Parameters:

- FileName (String): It is the path to the '.dat' file, which is usually named as 'histogram_complexes_time.dat', representing the histogram data to be analyzed.

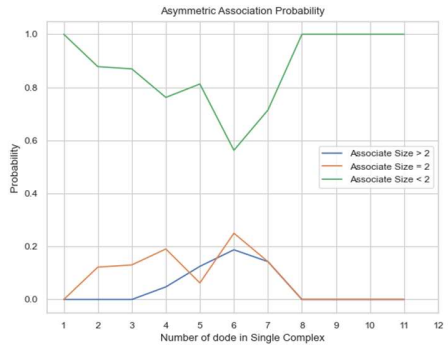
    - If there are multiple, the files should be named [name]_1.dat, [name]_2.dat, ect. And the FileName should be just [name].dat
    - Example: transition_1.dat, transition_2.dat …. FileName = transition.dat
- FileNum (Int): It is the number of the total input file. If multiple files are provided, their names should obey the naming rule.
- InitialTime (Float): It is the initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
- FinalTime (Float): It is the final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
- SpeciesName (String): It is 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): This is 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 std of mean probability

Example:

```
cmplx_size, mean_associate_probability, std =
ioNERDSS.associate_prob_asymmetric(FileName="ioNERDSSPyPi\TestingFunctions\\transitio
n_matrix_time.dat",FileNum=1,InitialTime=0,FinalTime=1,SpeciesName="dode",DivideSize=2,
ShowFig=True,SaveFig=False,SaveVars=True)
>>>
```



Long 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.

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. 'Symmetric' in the function name means that for the dissociate reaction, both sizes of complexes are counted as dissociating events.
symmetricallyParameters:

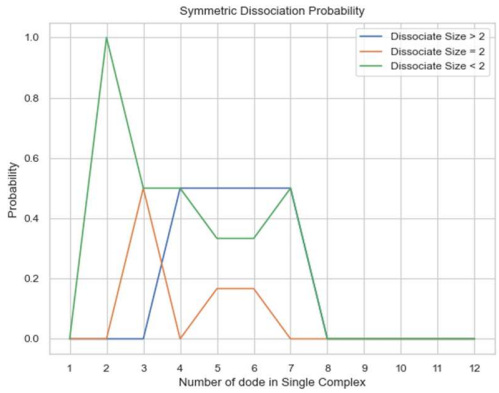
- FileName (String): It is the path to the '.dat' file, which is usually named as 'histogram_complexes_time.dat', representing the histogram data to be analyzed.
    - If there are multiple, the files should be named [name]_1.dat, [name]_2.dat, ect. And the FileName should be just [name].dat
    - Example: transition_1.dat, transition_2.dat …. FileName = transition.dat
- FileNum (Int): It is the number of the total input file. If multiple files are provided, their names should obey the naming rule.
- InitialTime (Float): It is the initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
- FinalTime (Float): It is the final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
- SpeciesName (String): It is 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): This is 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 std of mean probability

Example:

```
cmplx_size, mean_dissociate_probability, std =
ioNERDSS.dissociate_prob_symmetric(FileName="ioNERDSSPyPi\TestingFunctions\\transitio
n_matrix_time.dat",FileNum=1,InitialTime=0,FinalTime=1,SpeciesName="dode",DivideSize=2,
ShowFig=True,SaveFig=False,SaveVars=True)
>>>
```



Long 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.

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. '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.
Parameters:

- FileName (String): It is 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: transition_1.dat, transition_2.dat …. FileName = transition.dat

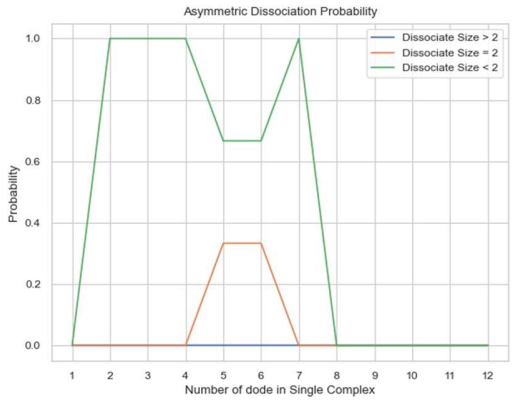
- FileNum (Int): It is the number of the total input file. If multiple files are provided, their names should obey the naming rule.
- InitialTime (Float): It is the initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
- FinalTime (Float): It is the final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
- SpeciesName (String): It is 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): This is 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\\transiti
on_matrix_time.dat",FileNum=1,InitialTime=0,FinalTime=1,SpeciesName="dode",DivideSize=
2,ShowFig=True,SaveFig=False,SaveVars=True)
>>>
```



Long 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.

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.

Parameters:

- FileName (String): It is 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: transition_1.dat, transition_2.dat …. FileName = transition.dat
- FileNum (Int): It is the number of the total input file. If multiple files are provided, their names should obey the naming rule.
- InitialTime (Float): It is the initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
- FinalTime (Float): It is the final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
- SpeciesName (String): It is 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'.

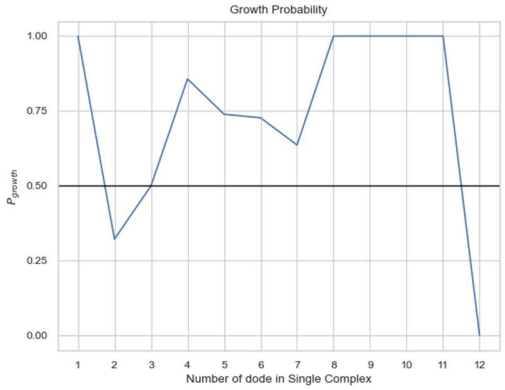
Returns:

- cmplx_size: list of each complex size
- mean_dissociate_probability: the probability of a certain size of complex becoming another smaller size
- std: the std 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=Fal
se,SaveVars=True)
>>>
```



Long 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.

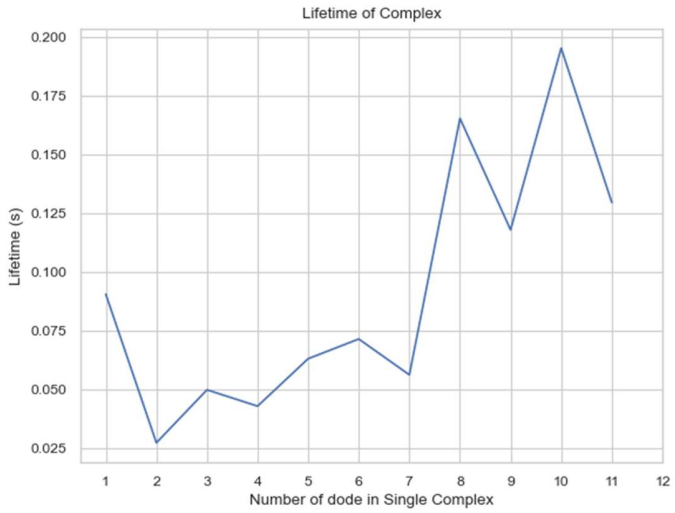
2.4.7 LINE PLOT – Average lifetime for each complex type:

complex_lifetime(FileName, FileNum, InitialTime, FinalTime, SpeciesName, ShowFig, SaveFig, SaveVars)
Description: This line plot indicates the average lifetime for each complex size.
Parameters:

- FileName (String): It is 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: transition_1.dat, transition_2.dat …. FileName = transition.dat
- FileNum (Int): It is the number of the total input file. If multiple files are provided, their names should obey the naming rule.
- InitialTime (Float): It is the initial time in **seconds** that will be examined. Must be smaller than FinalTime and bigger than start time in the file.
- FinalTime (Float): It is the final time in seconds that will be examined. Must be bigger than InitialTime and smaller than max time in the file.
- SpeciesName (String): It is 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)
>>>
```



Long 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.

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.
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
...
```

Additional Info: 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.

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.
Parameters:

- FileName (String): It is the path to the .xyz file, which is usually names as 'trajectory.xyz'.
- LitNum (Int, Optional = -1)
  Description: It is 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 specificied trajectory data included

Example:

```
ioNERDSS. xyz_to_csv(FileName="ioNERDSSPyPi\TestingFunctions\\trajectory.xyz",
LitNum=-1)
>>>
```

literation	name	x	y	z
0	ap	87.42062	-270.109	-203.662
0	ap	88.08153	-271.052	-205.297
0	ap	86.75972	-269.166	-202.027
0	ap	-58.6471	277.5285	-353.236
0	ap	-57.8368	278.0403	-354.991
0	ap	-59.4575	277.0167	-351.481
0	ap	321.4152	-267.935	338.5269
0	ap	320.3727	-269.137	339.7386
0	ap	322.4577	-266.733	337.3153
0	ap	74.40736	51.46147	-242.958
0	ap	74.10961	53.13188	-244.017

Long 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.

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): It is the path to the .xyz file, which is usually names as 'trajectory.xyz'.
- LitNum (Int, Optional = -1)
  Description: It is 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 of 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: racks the COM coordinate changing of one or more proteins.
Parameters:

- FileName (String): It is the path to the .xyz file, which is usually names as 'trajectory.xyz'.
- SiteNum (Int): This is 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): This is 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]]]
```

Long 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.

2.7 Analyzing .pdb files

2.7.1 LINE PLOT- Auto correlation function for complexes

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))

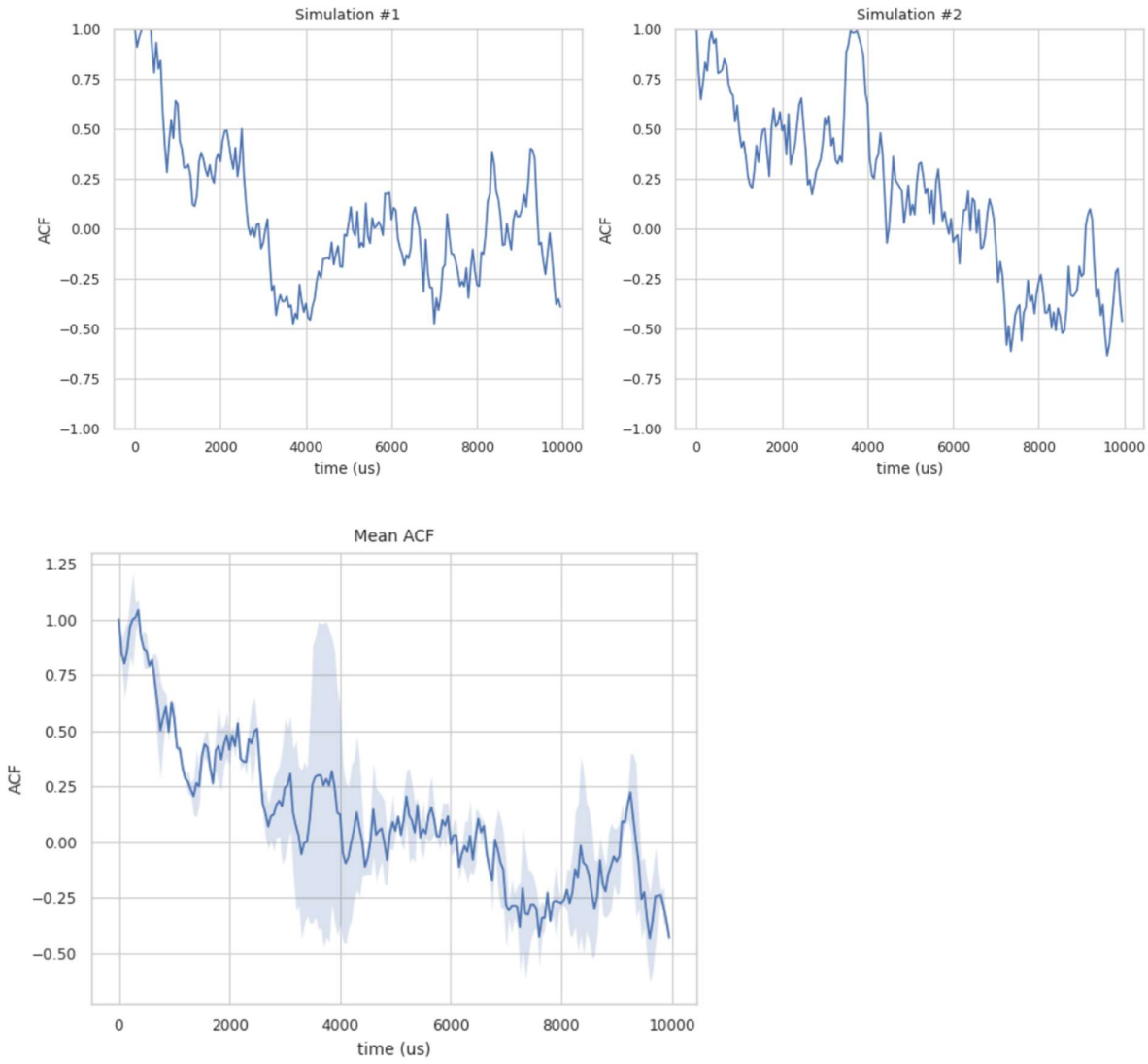
acf_coord(PDBDirectory, mol_list, sim_num, time_step, show_fig, save_fig)
Description: Calculates the mean acf of protein complexes stored in a series of NERDSS generated PDB files.

Parameters:
- PDBDirectory (String): The name of the directory where all PDB files are stored
- mol_list(List of String): the name of the molecules intended to be evaluated
- sim_num (int, optional = 1): number of repeated simulations intended to be evaluated
- time_step (int, optional = 1): time steps of the NERDSS simulation in micro-seconds. If not inputted, 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)



Long Description:
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.)
…/

2.8 Gag Sphere

2.8.1 Reshape Gag

reshape_gag(PathName)
Description: reshape and regularize experimentally measured gag lattice structure

Parameters:

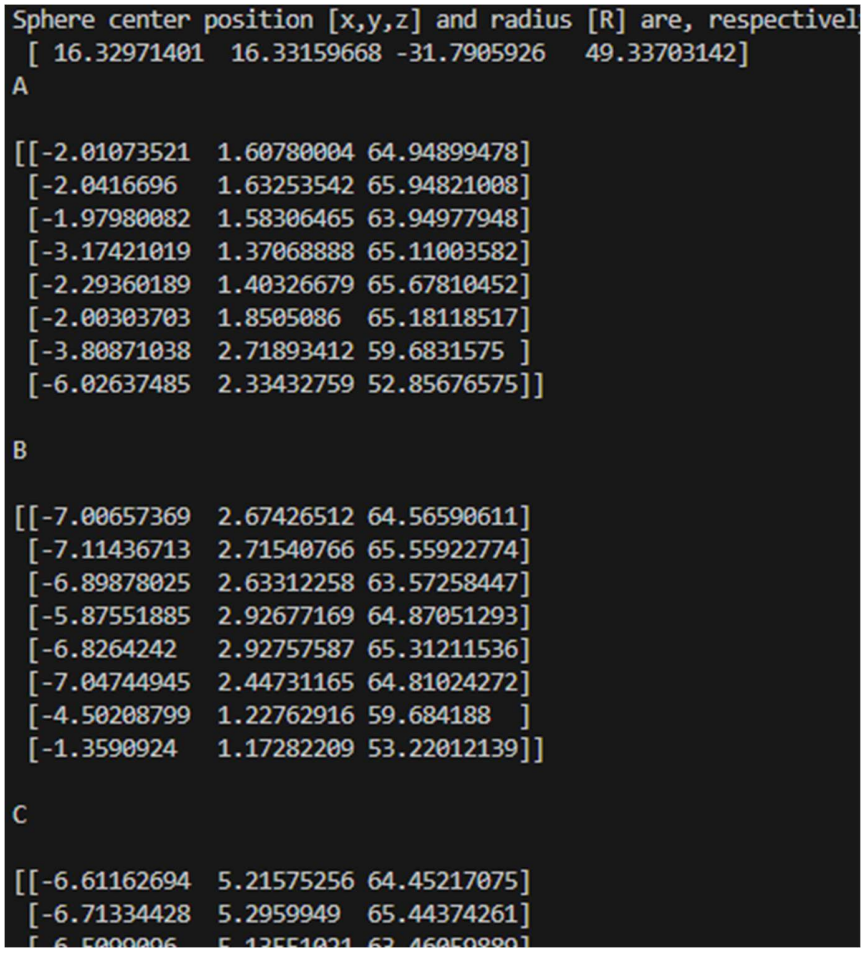
- PathName(String) : the path of the .pdb file

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"))
```



Long description: To construct a model of the gag monomer, experimentally measured gag lattice structures must be regularized to eliminate thermal fluctuations and other experimental errors. This function does this regularization job.

2.8.2 Sphere Regularization Index

sphere_regularization_index(PathName, IterNum, TimeStep, ComplexNum, SpeciesName)
Description: Calculating the sphere regularization index of a complex yielded from NERDSS

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 intended to be analyzed

Returns:

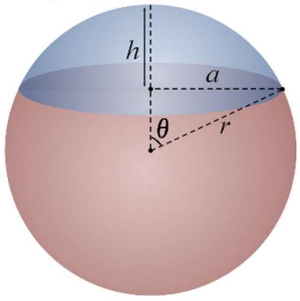
- max_complex_size_return (list): the complex size of each complex evaluated (1 indicates monomer, etc)

- 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")

```
The total number of complexes is 1715
Complex Size: 6.000000
Theta of the sphere cap: 0.928128
R of the fitted circle: 4.500332
Sphere center coord:  [360.08775134 334.06316669 324.12423557]
Sphere cap COM:  [360.151, 334.6261666666667, 324.17900000000003]
Regularixation index:  0.0
--------------------------------------------------------------------------------
The total number of complexes is 1715
Complex Size: 6.000000
Theta of the sphere cap: 0.577679
R of the fitted circle: 7.071632
Sphere center coord:  [232.6418975   25.71270218 430.04258583]
Sphere cap COM:  [231.44450000000003, 20.360500000000002, 430.05566666666664]
Regularixation index:  0.0
--------------------------------------------------------------------------------
The total number of complexes is 1715
Complex Size: 5.000000
Theta of the sphere cap: 0.294724
R of the fitted circle: 12.523278
Sphere center coord:  [227.1519892   48.0934494   53.14100005]
Sphere cap COM:  [220.01, 39.382999999999996, 49.8514]
Regularixation index:  0.2
----------------------------------End-----------------------------------------
```

Long Description:



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 reality. 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 example 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.

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: merging NERDSS output files from restart simulations

Parameters:

- None

Returns:

- None

Long Description: Arrange 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 run the function. (Note: First, run the function in the directory containing 1/, 2/, … Second, assign names the restart#### directories according to the order of their first recorded iterations, ie, the restart simulation data that begins at time=0.5 should be stored in for instance restart0001 and the restart simulation data that begins at time=1.0 should be stored in for instance restart0003 but not the other way around). 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 of its restart simulations. All restart#### directories will be removed after merging.

3.1.2 Merging NERDSS output files

merge_files(destination_file, source_file, file_type)
Description: merging NERDSS output files.

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 intended 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

Long Description: If the parameter of file_type is not inputted, the function will simply concatenate the source_file to the end of the destination file. If one of the given file types is inputted, 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 kept 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.