Tutorial 1: Calculating Nanoparticle Properties

Subhamoy Mahajan

January 24, 2021

This tutorial demonstrates the use of **NPanalysis** module for calculation of nanoparticle properties. The tutorial will focus on analysis performed in Mahajan and Tang. The codes are written for aggregation of DNAs and polyethylenimines (PEIs), where only PEIs and DNAs can bind together; A PEI cannot bind with another PEI, a DNA cannot bind with another DNA. The same codes can be implemented to study two-component aggregation, where molecules of different kinds bind with each other. The code assumes the structure file begins with DNA molecules and then followed by PEI molecules. All DNA and PEI molecules should be identical.

1. Calculating the minimum distance between all DNA-PEI pairs

To determine the DNAs and PEIs present in different nanoparticles, we must first know which DNA-PEI pairs are bound to each other. To determine this we can calculate the minimum distance between each pair of DNA and PEI from the trajectory files provided,: `md_1.xtc` and `md_1.tpr`. To reduce the file size `md_1.xtc` and `md_1.tpr` only contains DNA, PEI, and ions (simulation was performed with Martini polarizable water).

The minimum distance is calculated using the Gromacs function *mindist*. This is achieved by the script `calc_mindist.sh`.

```
Tut1$ bash calc_mindist.sh
```

The script was written for Compute Canada's Cedar cluster. Some minor changes might be necessary on a different computing cluster. When run with 48 processors (a whole node) the calculation takes less than 3 minutes. It creates a index file `index_mol.ndx`, directory `mindist`, and files `mindist/mindist[d]-[p].xvg`, where `[d]` and `[p]` are the IDs of DNA and PEI respectively. The `mindist/mindist*.xvg` files were calculated using the `-pbc` option to calculate the minimum distance considering the periodic boundary conditions.

The mindist directory can be compressed using,

```
Tut1$ tar -czvf mindist_d_p.tar.gz mindist
```

The next steps **2-7** is achieved using the command,

```
Tut1$ python calc_NPprop.py
```

In the python script,

```python
import sys
sys.path.insert(0,'../../')
import NPanalysis as NPa
```

is used to import the `NP_analysis.py` file stored in '`../../`'. Every python script using `NP_analysis` must use the command,

```python
NPa.import_constants('.')
```

to import constants. The string '`.`' denotes the location of `constants.py`, which stores all important constants of the aggregation simulation. Value of all constants must be provided even though all constants may not be used.

2. Calculating a connection matrix between DNAs and PEIs

To determine which DNAs and PEIs are part of the same cluster (nanoparticle), we need the information of DNAs and PEIs that are bound to each other in the form of a connection matrix for all the timesteps. At every timestep, if the minimum distance between a DNA-PEI pair is less than the contact distance (*contact_dist* in `constants.py`), then they are considered to be bound to each other.

To perform this calculation total number of timesteps in the simulation is needed. This can be determined from any `mindist/mindist*.xvg` file. We determine the total number of timesteps from `mindist/mindist0-0.xvg`, using the command,

```
Tut1$ grep -v '@' mindist/mindist0-0.xvg | grep -v '#' | wc -l
```

The command calculates the number of lines in `mindist/mindist0-0.xvg` excluding lines beginning with '#' and '@'. For this tutorial, the total number of timesteps is found to be 251. Therefore, the connection matrix is determined using the line,

```
NPa.conv_mindist2connected(251,mindist_loc='mindist/')
```

The default value of *out_mindist='mindists'* and *out_connected='connected.pickle'* ' is used for calculating the connection matrix. As a result, the connection matrix and minimum distance between DNA-PEI pair are stored as a pickled file `connected.pickle` and `mindists.pickle` respectively.

From the connection matrix (`connected.pickle`), the number of PEIs in different roles, average number of PEIs between bridged DNA pairs and total number of bridged DNA pairs can be calculated (leaving other properties for future Tutorials).

3. Calculating number of PEIs in different roles

Based on Mahajan and Tang, PEI plays three major roles: free in the solution, peripheral of a nanoparticle, and bridging DNAs in a nanoparticle. More formally, free PEIs are not bound to any DNAs, peripheral PEIs are only bound to one DNA, and bridging PEIs are bound to more than one DNA. The complete trajectory has 251 timesteps, and the number of PEIs in each role is averaged over 50 timesteps, resulting in 5 values. This is achieved using the line,

```
NPa.get_PEI_roles(50,251)
```

Default values for arguments were used: *connected_pickle* = *'connected.pickle'*, and *outname* = *'PEI_roles.dat'*.

4. Caculation average number of bridging PEI between a pair of bridged DNAs and total number of bridged DNA pairs

The total number of bridging PEIs is divided by the total number of bridged DNA pairs to determine the average bridging PEI between a bridged DNA pair. Similar to PEI roles, average was performed over 50 timesteps resulting in 5 values. This is achieved using the line,

```
NPa.get_PEI_roles2(50,251)
```

Default values for arguments were used: *connected_pickle* = *'connected.pickle'* , and *outname*=*'PEI_roles.dat'*.

5. Calculating clusters (nanoparticles) of DNAs and PEIs

To determine clusters (nanoparticles) in each timestep from the connection matrix, mathematical graphs are used. Iterating over all DNA-PEI pairs, if the DNA-PEI pair is bound to each other, an edge between two new nodes *d[i]* and *p[j]* is created where *[i]* and *[j]* are DNA and PEI IDs respectively. All IDs start from zero. For a particular DNA, a node *d[i]* is created even if there are no PEIs bound to it. However, no nodes are created for free PEIs. This creates multiple disjoint mathematical graphs, which are essentially different clusters (nanoparticles). This is achieved using the function *get_pdgraph()*.

To convert the disjoint graphs into list of DNA and PEI IDs for each cluster at a given timestep, the function *get_cluster()* used. It utilizes *node_connected_components* in the *networkx* python module, to find all the nodes that are directly or indirectly connected with a given node. Using this, all the nodes directly or indirectly connected with the node *'d0'* is chosen to be the first cluster. For each DNA with ID greater than 0, the DNA presence of a DNA in a previous cluster is checked. If the DNA is not present in a previous cluster, then a new cluster is created with all the nodes directly or indirectly connected with it.

The function *gen_cluster()*, first calculates all disjoint graphs in each timestep using *get_pdgraph()*. Second, it converts them into list of DNAs and PEIs using *get_cluster()*. Third, it removes the characters 'd' and 'p', and converts the string of DNA and PEI IDs into list of integers and sorts them. Finally it stores all the clusters generated at different timesteps into a single pickled file. So, calculation of clusters can be achieved with a single line command,

```
NPa.gen_clusters(251)
```

Default values of arguments were used: *connected_pickle='connected.pickle'*, and *headername = 'cluster'*.

Several nanoparticle properties can be calculated from clusters data. Few of these properties are shown in this tutorial: average size of a nanoparticles, average number of nanoparticles and nanoparticle charge as a function of nanoparticle size, hydrodynamic radius, and radius of gyraiton (leaving other properties for future Tutorials).

The function *w2f_cluster()* is used to write the clusters data at different time steps in separate files. This is required for the calculation of radius of gyration and hydrodynamic radius, when using a C-code (See Section 8).

```
NPa.w2f_cluster(outheader='cluster/cluster')
```

Default value of the argument was used: *cluster_pickle='cluster.pickle'*.

6. Calculating average nanoparticle size

The function *gen_avgsize()* is used to calculate two types of average nanoparticle size based on the number of DNAs present in it: number average and weight average. The number average size of the nanoparticle calculates the average number of DNAs in each nanoparticle. This is calculated using the equation,

$$number\ average\ size = \frac{1}{N_{NP}} \sum_{i=1}^{N_{NP}} D_i = \frac{N_{DNA}}{N_{NP}}$$

Where, D_i is number of DNAs in i^{th} nanoparticle, N_{NP} is the number of nanoparticles, and N_{DNA} is the number of DNAs in the simulation at a given timestep. The weight average size is the weighted average of the number of DNAs in a nanoparticle, where the weights are equal to the number of DNAs in each nanoparticle. This is evaluated using,

$$weight\ average\ size = \frac{1}{N_{DNA}} \sum_{i=1}^{N_{NP}} D_i^2$$

The function *gen_avgsize()* also averages the number and weight average size over multiple timesteps. The average is performed over 50 timesteps. This results in ignoring the last timestep and generating 5 time-averaged number and weight average sizes.

```
NPa.gen_avgsize(50,'avigsize.dat',251,0.002)
```

The default value of arguments were used: *cluster_pickle='cluster.pickle'*.

7. Calculating number of nanoparticles and charge as a function of the size

The number of nanoparticles and nanoparticle charge is calculated as a function of number average size. The average is performed over timesteps and different nanoparticles having the same number average size. For performing time average, the total number of timesteps is divided into bins. In this tutorial 5 bins are used, i.e., average over 0-50, 50-100, 100-150, 150-200, 200-250 timesteps.

```
NPa.gen_ncNP_s(251,5,'num_charge_NPs_size.dat')
```

The default value of arguments were used: *cluster_pickle* = *'cluster.pickle'*. The function *gen_ncNP_s*, stands for generation of number and charge of NPs as a function of size.

To calculate average number of nanoparticles and nanoparticle charge over a specified time range, the function *run_ncNP_s()* can be used.

8. Calculating hydrodynamic radius and radius of gyration

Calculation of geometric size of nanoparticles can be a critical property for several nanoparticles. One of the most widely used quantity is simulations is the radius of gyration. Hydrodynamic radius is most widely used in experiments due to their ease of measurement. However, the use of hydrodynamic radius is limited in simulations due to its computationally expensive calculation ($O(N^2)$ complexity). As of this date, Gromacs can calculate radius of gyraiton but not hydrodynamic radius.

The calculation of radius of gyration in Gromacs fails for two-component nanoparticle for the following reasons:

1. Molecules may be broken over the simulation box

2. Even if molecules are made whole with *gmx trjconv -pbc whole*, the relative location of molecules may not be consistent with the connection matrix

3. When molecules are clustered together using *gmx trjconv -pbc cluster*, in few cases it is not consistent with the connection matrix. This is because it iteratively moves molecules to bring it closer to the center of mass of nanoparticle, and as a result two bound molecules might be far apart.

4. A dynamically evolving nanoparticle may change the number of molecules in it, making it harder to evaluate radius of gyration over time using Gromacs.

`NP_analysis.py` aims to solve all these issues. Moreover, it can calculate hydrodynamic radius. The first DNA in the cluster is kept in its place. All PEIs bound the the first DNA is moved across periodic boundaries such that the minimum distance between the DNA and PEI is lowest. Then all DNAs bound to the PEIs are moved acorss periodic boundaries such that the minimum distance is the lowest. This is done sequentially untill all PEIs and DNAs in the cluster is moved appropriately. Finally the whole nanoparticle is moved across periodic boundaries such that the geometric center lies within the main simulation box.

In the current version of the radius of gyration and hydrodynamic radius of a nanoparticle at a given time is calculated using the following formulas.

$$\frac{1}{R_{hyd}} = \frac{1}{N^2}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\frac{1}{r_{ij}}$$

$$R_g = \sqrt{\frac{\sum_i^N m_i\left(\mathbf{r_i} - \mathbf{r_{com}}\right)^2}{\sum_i^N m_i}}$$

$$\mathbf{r_{com}} = \frac{\sum_{i=1}^N m_i\mathbf{r_i}}{\sum_{i=1}^N m_i}$$

Where, r_{ij} is the distance between i^{th} and j^{th} particle in the nanoparticle, $\mathbf{r_i}$ is position vector of i^{th} particle in the nanoparticle, $\mathbf{r_{com}}$ is the position vector of the center of mass of the nanoparticle, m_i is the mass of i^{th} particle in the nanoparticle, and N is the number of particles in the nanoparticle.

To correct the position of molecules in the trajectory, first the trajectory is separated into `*.gro` files, while making the molecules whole,

```
Tut1$ mkdir -p GROs
Tut1$ echo "0" | gmx trjconv -f md_1.xtc -s md_1.tpr -pbc whole -o
    GROs/DP.gro -sep
```

The script `get_massinfo.sh` is used to extract the mass of atoms from `md_1.tpr` file into `mass.dat`.

```
Tut1$ bash get_massinfo.sh
```

The files `GROs/DP*.gro` are correctly moved to satisfy the connection matrix using the commands,

```
Tut1$ mkdir -p New_GROs
Tut1$ python gen_Rh.py
```

The function, *update_gros()* in `gen_Rh.py` modifies `GROs/DP*.gro` files into `New_GROs/New*.gro`,

```
NPa.update_gros(251,inname='GROs/DP',outname='New_GROs/New',move_method='
    mindist')
```

Default values for the arguments were used: *cluster_pickle* = *'cluster.pickle'*, and *connected_pickle* = *'connected.pickle'*.

The function *calc_Rh_Rg()* can be used to calculate the hydrodynamic radius and radius of gyration but is very slow. To enable faster calculations a C-code `calc_Rh_Rg.c` is provided, which is about 100 times faster. To calculate the radii using C, use the following commands,

```
Tut1$ gcc ../../calc_Rh_Rg.c -lm -o ../../calc_Rh_Rg
Tut1$ ../../calc_Rh_Rg -i New_GROs/New -oRh Rh.dat -oRg Rg.dat -m
    mass.dat -clust cluster/cluster -cons constants.py -times 251
```

For the C-code name of corrected `*.gro` files (without [number].gro) should be provided after -*i*, output file for hydrodynamic radius after -*oRh*, output file for radius of gyration after -*oRg*, name of the file containing mass information after -*m*, name of cluster files (without [number].dat) should be provided after -*clust*, the constants,py file after -*cons*, and total number of timesteps after -*times*.

9. Issues and Concerns

If you find a issue with the code or tutorial please report an issue in github or email to

'subhamoygithub AT gmail DOT com'