Manual for the codes

December 30, 2024

1 Instructions for running the code: bcd_hard_spheroids.c

Note: To understand the code please contact the author.
Email- vikkivarma16@gmail.com

This section provides instructions and details on configuring and running a BCD (Brownian Cluster Dynamics) simulation for hard spheroids in a cubic simulation box.

The detail of the structure and formulation is given in the reference, [4] and the findings from this code have been published in the reference [1, 4]. For the explaination and the meaning of the symbols used in this manual can also be found in the same reference [4].

The configuration parameters are written into a file named `Base_data.txt` through the input file named `runner_bhs.sh`, before executing the simulation binary which will be done by the same input (`.sh`) file. After execution of the binary compiled C code the file `Base_data.txt` will be deleted, after being read by the same code.

System Configuration

The simulation setup is defined via sequentially writing parameters by the runner `.sh` file, into the `Base_data.txt` file. Below are the parameters used in the `.sh` file, along with their descriptions:

For each type of particles its physical properties is defined in the `.sh` file in each row. For example aspect ratio in a system with two type of particle will be defined as

```
echo "50.0" >> Base_data.txt # variable of this type must be defined in rows
                             #   ....between this upper ....

echo "2.0" >> Base_data.txt  aspect ratio of type 1 particle
echo "1.0" >> Base_data.txt  aspect ratio of type 2 particle

echo "50.0" >> Base_data.txt # ... and this lower limit
```

In case if nothing is defined the default value will be considered. In case if the number of rows and hence the parameters defined for the particle properties increases than the required, then those values will be skipped. Same rule applies for all the similar variables defined below in the input file "`runner_bhs.sh`".

Specifying parameters in the runner file named `runner_bhs.sh`

1. **Simulation Start Condition:**

```
#simulation parameters

echo "0"  >  Base_data.txt  #0: to start from beginning
                          #...or 1: to start from a relaxed_system.txt file
```

`0` - Start simulation from scratch. `1` - Start from a relaxed configuration file named `relaxed_system.txt`, which is generated after each hundred physical time from the same code. It is often useful when we want to start the simulation from the point where it stopped.

2. **Simulation Type:**

```
echo "1" >> Base_data.txt  #finite volume fraction :1 single particle :0
```

This section switches the interactions between the particles on and off. When off, the particle will diffuse like a single particle. However, you would still need to consider a finite volume fraction ϕ, just to fill the box with the particles. Otherwise code will show an error.

3. **Number of particle:**

```
echo "1000" >> Base_data.txt  #number of particles
```

Specifies the number of particles (variable N used in the reference[2, 3] etc) in the system including all the type or components present in the system.

4. **Step Length:**

```
echo "0.01" >> Base_data.txt  #step length
```

This tells the constant step size s_T used in the BCD formalism. Bigger step size greter the physical time will be covered from the same number of simulation steps as described in the references.

5. **Simulation Steps:**

```
echo "500000000" >> Base_data.txt  #simulation steps
```

Total number of simulation steps (t_{sim}), where physical time will be $t_{phy} = t_{sim} \cdot s_T^2$.

6. **Probation Periods:**

```
echo "20" >> Base_data.txt  #probation time is the time the system is
                #...left to randomize after that the simulation will start
```

Number of probation periods, which is the physical time after which some of the calculation in the simulation code happening during the run phase, will take place. eg, diffusivity or some correlation function or whatever is present there inbuilt. However, the data output will be printed since the beginning of the code, so while analyzing the data (which is an output file named `0.txt, 1.txt, 2.txt` etc. where `0.txt` stores the data for the physical time 1 to 100 and `1.txt` stores the data for the physical time 101 to 200 ... etc.) keep in mind to consider the data only after a certain physical time.

7. **Volume Fraction:**

```
echo "0.04" >> Base_data.txt  #volume fraction
```

Total volume (ϕ)fraction calculated over all kinds of particles present in the system.

8. **Particle Types:**

```
echo "1" >> Base_data.txt  #kind of particles in the system e.g. 1 or 2
```

This specifies the type of species present in the system. It can be any system. For a monomeric or single component system, value 1, for binary system, value 2 and for ternary system value 3 and so on. However the type id assigned to the first kind of particle will be 0 second kind of particle will be 1 and so on.

The following lines assign the geometric properties of the spheroidal particles

9. **Constant Volume or Axis:**

```
#particle's shape and size

echo "50.0" >> Base_data.txt  #sco 0: constant volume 1: constant axis

echo "0" >> Base_data.txt

echo "50.0" >> Base_data.txt
```

This line specifies if the particles shape will be made spheroid, while keeping the volume of the particle constant and same as the volume of a sphere. Or the particles will be made spheroids while keeping its major axis constant and equal to the diameter of the spheres.

10. **Aspect Ratio:**

```
echo "50.0" >> Base_data.txt  #aspect ratio define for each
                              #...kind of particles default is 1.0

echo "1.0" >> Base_data.txt

echo "50.0" >> Base_data.txt
```

This line defines the shape of the particles. Aspect ratio (p) between the length of major (a) and minor (b) axis of the spheroidal particles (default is `1.0`, which is an sphere).

11. **Size Enhancement Parameter:**

```
echo "50.0" >> Base_data.txt  #size enhancement parameter default is 1

echo "1.0" >> Base_data.txt

echo "50.0" >> Base_data.txt
```

This parameter scales the size of the particles both for constant volume and constant axis length options. In this way, you can make your particle bigger and smaller for any aspect ratio in terms of its volume or size. As previous option restrict the particles to have either same volume or the same axis length in compare to the volume and diameter of a sphere, where all the length is scaled with the diameter of the spherical particles.

12. **Fraction Parameter:**

```
echo "50.0" >> Base_data.txt  #fraction of each kind of particles

echo "1" >> Base_data.txt

echo "50.0" >> Base_data.txt
```

Fraction of particles with a particular type, calculated over N. For example in two consecutive rows if the fraction is defined as 0.5, 0.6 for a system with two kinds of particle. If there is total 500 particles present in the system. Then, the fraction of particle type with type id 0 will have fraction 0.5 and number of particles as 250. And the second kind of particle, which is the particle with highest type id (which is1) will have fraction 1− total sum of all other particle-fraction with id lesser than highest particle type id. So in effective the fraction assigned to the last particle will be ignored and calculated using this method to keep the total sum of fraction 1. Therefore we will have fraction of type 2 particle with type id 1 as 0.5 and number of particles will be 250.

Note: You can not define the size of the simulation box. It will be calculated by using the number of particles and the volume fraction ϕ of particles.

Compilation and Execution section

The simulation code `bcd_hard_spheroids.c` is compiled and executed as follows:

```
$ gcc bcd_hard_spheroids.c
... -o bcd_hard_spheroids -lm
$ ./bcd_hard_spheroids.c
```

Some notes on the codes

Data output information

Code generates the data for each physical time which is given as equal to the $t_{sim} = 1/s_T^2$. For example if you are using $s_T = 0.005$ then your data generated on every $t_{sim} = 40000$ steps, which is equal to one physical time. For the data structure the reader can see the code itself where position stored as `ami` and orientation of the axes of the particles stored `li` and the diffusivity (translational in 3D: `ldcxyz[i][0], ldcxyz[i][1], ldcxyz[i][2]`, in perpendicula plain to the symmetry axis: `ldcxy[i][0], ldcxy[i][1]`, along the symmetry axis: `ldcz[i]`, and rotatioanl in 3d: `lom[i][0], lom[i][1], lom[i][2]`), is being printed in each simulation step. Data is stored from physical time $1 - 100$ in the file `0.txt` and from $101 - 200$ in the file `1.txt` and so on. The full configuration of the system is stored in the file *relaxed_system.txt* on each 100^{th} physical time steps. Which can be accessed and run instantly by switching the parameter "simulation start condition" on and making it 1. The further detail about the output structure is discussed in the next section.

Output Data file structure

The code prints the relevant quantities in the data file named **0.txt**, **1.txt**, ... and so on, where **0.txt** file stores the data for the physical time ranging between $1 - 100$ and **1.txt** stores the data for physical time $101 - 200$ and so on.

The data structure for each physical time is given as,

```
fprintf(cfu,"%lf\n", f[0]);  //  box length along x
fprintf(cfu,"%lf\n", f[1]);  //  box length along y
fprintf(cfu,"%lf\n", f[2]);  //  box length along z

for (i=0; i<N; i++)  \\ N number of particles
{
    fprintf(cfu,"%lf  %lf  %lf\n", ami[i][0], ami[i][1], ami[i][2]);
    \\ position of ith particle

    fprintf(cfu, "  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf\n",
    ...ldcxyz[i][0], ldcxyz[i][1], ldcxyz[i][2],
    \\ translational diffusivity in laboratory frame
    ...ldcxy[i][0], ldcxy[i][1],
    \\ diffusivity in perpendicular plain of the particle's symmetry
    \\... axis measured in the particle's frame
    ... ldcz[i],
    \\ diffusivity parallel to the particle's symmetry
    \\... axis measured in the particle's frame
    ....lom[i][0], lom[i][1], lom[i][2]);
    \\ rotational diffusivity measured in the laboratory frame




    fprintf(cfu,"%lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf\n",
    ... li[i][0], li[i][1], li[i][2],  li[i][3],
    ... li[i][4], li[i][5],  li[i][6], li[i][7], li[i][8]);
    \\ orientation of the ith spheroidal particle for all the three axes
}
```

Full configuration of the system

The full configuration of the system is printed after each hundred time steps in the file named **relaxed_system.txt**. Which stores the position and orientation of all the particles and also the nearest neighbor periodic boundary condition marking. This file can be directly accessed by the same code to run the simulation from the time specified in the same file (**relaxed_system.txt**). However, you will have to change the simulation start condition to the value 1. The section is given as below:

```
FILE* ku;
ku=fopen("Relaxed_system.txt","w");

fprintf(ku,"%d\n", sitt+1);  \\ physical time

fprintf(ku,"%lf\n", f[0]);  \\ box length along x
fprintf(ku,"%lf\n", f[1]);  \\ box length along y
```

```
fprintf(ku,"%lf\n", f[2]);  \\ box length along z

for (i=0; i<N; i++)
{
    fprintf(ku,"%lf  %lf  %lf\n", ami[i][0], ami[i][1], ami[i][2]);
    \\ position of ith particles

    fprintf(ku, "%d\n", wnrd[i]);
    \\ number of nearest neighbour of ith particle

    for (j=0; j<wnrd[i]; j++)
    {
        fprintf(ku,"   %d  %d", nrdi[i][j][0], nrdi[i][j][1]);
        \\ jth nearest neighbour of ith particle
        \\...and its periodic boundary mark
    }
    fprintf(ku, "\n");

    fprintf(ku,"%lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  \n",
    ...li[i][0], li[i][1], li[i][2], li[i][3], li[i][4],
    ...li[i][5], li[i][6], li[i][7], li[i][8]);
    \\ oritentation vector for all the three axis of ith particle
}

fclose(ku);
```

Miscellaneous output files

There are many other output file generated in the same code for the ease to access some of the quantities directly without any need to analyze the data. Following are the list of such files,

- **Compressed_system.txt**: This file contains the full configuration data after initializing the system (removing all the overlap of the particles), which is printed for the purpose of debugging.
- **Cord.vtk**: This file is printed for the purpose of visualization in paraview after each hundred time steps.
- **Diffusion_coefficient_el_l.txt**: This file prints the diffusivity averaged over all the particles after each 10 simulation time steps. The first column is the simulation time second column is the diffusivity in perpendicular plain of the symmetry axis in the particle's body frame. Third column is the diffusivity of parallel to the symmetry axis in the particle's body frame. Fourth column is the total diffusivity in the laboratory frame. Fifth column is the rotational diffusivity. The purpose of this file is to debug the code. The wrong value of diffusivity can indicate about something wrong happening in the simulation.
- **Initial_Cord.vtk**: This file can be used to visualize the particles just after the code is initialized (after removing all the overlapping of particles from the system).

2 Instructions for running the code: bcd_hard_spheroids_widoms.c

The given code does the same simulation as the code given in Sec. 1, except the code insert a Widom particle with the properties as same as assigned by the given widom particle id, to calculated the chemical potential μ. It creates satisfactory results for the low volume configuration and the result becomes poorer with introducing more anisotropy in the shape of the particles (which is done by changing the aspect ratio of the particles).

The configuration parameters are written into a file named `Base_data.txt` through the input file named `runner_bhsw.sh`, before executing the simulation binary which will be done by the same input (`.sh`) file. After execution of the binary compiled C code the file `Base_data.txt` will be deleted, after being read by the same code.

System Configuration

The simulation setup is defined via sequentially writing parameters by the runner `runner_bhsw.sh` file, into the `Base_data.txt` file. Most of the simulation parameters are the same as discussed in the Sec. 1). However, a few extra parameters are described below:

Specifying parameters in the runner file named `runner_bhsw.sh`

1. **Widom particle's id:**

```
#widom chemical potential parameters
echo "0" >> Base_data.txt  #particle type inserted as widom particle
```

This section of the input parameter specify the particle type id of the particle which will be inserted to compute the Widom's chemical potential.

Compilation and Execution section

The simulation code `bcd_hard_spheroids_widoms.c` is compiled and executed as follows:

```
$ gcc bcd_hard_spheroids_widoms.c
... -o bcd_hard_spheroids_widoms -lm
$ ./bcd_hard_spheroids_widoms.c
```

Output file and system data

All the specifications and descriptions of the output files and the system data is as same as discussed in the Sec. 1. Except the file generates the Widom's chemical potential stored in a file **Widoms_ratio.txt**. The output structure is given as:

```
fprintf(wu, "%d   %lf  %lf  %lf   %lf\n", sitt+1,
...widoms, twidoms,  widoms/twidoms, -log(widoms/ twidoms));
// sitt is the simulation time
```

```
// widoms is the total number of successfull insertion attemps
// twidoms is the total number of insertions
// -log(widoms/ twidoms) is the widoms chemical potential
```

3 Instructions for running the code: bcd_hard_spheroids_obstalce.c

This section provides instructions and details on configuring and running a BCD (Brownian Cluster Dynamics) simulation for hard spheroids in the presence of cylindrical obstacles, arranged in a periodic lattice.

The detail of the structure and formulation is given in the reference, [4] and the findings from this code have been published in the reference [1]. For the explanation and the meaning of the symbols used in this manual can also be found in the same reference [1].

The configuration parameters are written into a file named `Base_data.txt` through the input file named `runner_bhso.sh`, before executing the simulation binary which will be done by the same input (`.sh`) file. After execution of the binary compiled C code the file `Base_data.txt` will be deleted, after being read by the same code.

System Configuration

See the Sec. 1.

Specifying parameters in the runner file named `runner_bhso.sh`

The particle type id assigned to the obstacles is 0. And the first kind of particle type have particle type id as 1 and second kind have 2 and so on. Remember, in the absence of obstacles (Sec. 1) the particle type id assigned to the first kind of particle is 0 and so on. Data output file and diffusivity and these things printed should be carefully analyzed and used while keeping these things in mind. However, while defining the shape and size parameters for the particles in the input file. All the definition goes only to the particles not the obstacles. So the parameters are assigned to the particles in sequence where the first row is goes with particle type with type id 1 and so on.

All the parameters are as same as discussed in Sec. 1, except a few which is discussed as below:

1. **Radius of Obstacles:**

   ```
   echo "4.5" >> Base_data.txt #radius of the obstacles
   ```

   This line specifies the radius of the obstacles (r_o).

2. **Number Fraction of Obstacles:**

   ```
   echo "0.5" >> Base_data.txt #area fraction of the obstacles
   ```

   This line of parameter specify the area fraction of obstacles present in the system which is N_o per unit area of the simulation box.

Note: You can not define the number of particles. It will be defined by the obstacles configuration which will govern the simulation box size. Then the volume fraction ϕ of particles, will defined the number of particles corresponding to that simulation box.

Compilation and Execution section

The simulation code `bcd_hard_spheroids_obstacle.c` is compiled and executed as follows:

```
$ gcc bcd_hard_spheroids_obstacle.c
... -o bcd_hard_spheroids_obstacle -lm
$ ./bcd_hard_spheroids_obstacle.c
```

Some notes on the codes

Data output information

For the details, see the Sec 1.

It generates the data for all the particles including the obstacles, which in fact remains immobile and have particle type id set as 0. So the total number of particle generated in the system includes the number of obstacles while counting. And data like position and orientation of the obstacles is printed in the data output file.

Simulation box, number of particle and cylinder configuration

The code is not optimized for the performance therefore, for the spheroids number greater than 2500 the simulation time increases to some unfeasible value. Now, for a particular area fraction of cylindrical obstacles box size can not be reduced than a minimum value and therefore the number of particle. However, the code finds a trick. To reduce the number of particles, the code truncate the simulation volume of the system along the direction of the cylinder, while considering the finite size effect (so that the simulation artifacts appearing in structure and dynamics can be avoided). In other words, the size of the box can not be reduced to the length lesser than the four times of the size of the particle (particle grid number 4). This truncation implementation allow for almost all the explored obstacle density and radius configuration to be simulated but not all. Therefore, to generate the simulation box in certain cases, one need to calibrate the code section so that the correct truncated box can be generated. One need to check the terminal output, if the obstacle grid number reduced to *zero* the code will not run (where it should be minimum 1). Where obstacle grid number is calculated by dividing the length of the box with the size of the obstacle particle. The code section is given as below,

```
nt[0]=0;
it=0;
i=0;

while(it<500)
{
    i=i+1;
    nt[0]=i*i; // number of obstalces
    fe=pow(nt[0]*piee*robs*robs/obsd, 0.5); // box length
    it=(int)(((fe*fe*fe)-piee*fe*robs*robs*(float)nt[0])*fvf*6.0/piee);
    // total number of all kinds of particles for a given particle volume fraction
}
```

```
f[0]=fe;  // box length along x
f[1]=fe;  // box length along y
f[2]=fe;  // box length along z

bl[0]=ble;  // grid shell length for the particles
bl[1]=ble;  // grid shell length for the particles
bl[2]=ble;  // grid shell length for the particles

block[0]=(int)floor(f[0]/bl[0]);  // number of shells on the particle's grid
bl[0]=f[0]/(float)block[0];
f[0]=bl[0]*(float)block[0];

block[1]=(int)floor(f[1]/bl[1]); // number of shells on the particle's grid
bl[1]=f[1]/(float)block[1];
f[1]=bl[1]*(float)block[1];

block[2]=(int)floor(f[2]/bl[2]);  // number of shells on the particle's grid
bl[2]=f[2]/(float)block[2];
f[2]=bl[2]*(float)block[2];

if (it>600)
{
    tri=0;
    while(tri==0)
    {
        if(it<600 || block[1]<4)
        \\where 4 is the minimum number of particle grid...
        \\...calculated by box_length/size_of_the_particle
        {
            tri=1;
        }
        else
        {
            block[1]=block[1]-1;
            f[1]=bl[1]*block[1];
            it=(int)(((f[0]*f[1]*f[2])-piee*f[1]
            ...*robs*robs*(float)nt[0])*fvf*6.0/piee);
        }
    }
}

for (i=1; i<war  ; i++) \\ run over all the kinds of particle
                        \\ type id 0 is reserved for the obstacles itself
{
    nt[i]=(int)round((float)it*per[i]);
    //printf("%d  \n", nt[i]);
}
N=0;
for (i=0; i<war; i++) N=N+nt[i];
```

4 Instructions for running the code: bcd_hard_spheroids_obstacle_widoms.c

This section provides instructions and details on configuring and running a BCD (Brownian Cluster Dynamics) simulation for hard spheroids in the presence of cylindrical obstacles. The more detail is given in the Sec 1, 2 and 3.

All the structure of the code is same as the given for `bcd_hard_spheroids_obstacle.c`. However the code `bcd_hard_spheroids_obstacle_widom.c` for Hard Spheroids also does the widom particle insertion to calculate the Widom's chemical potential. Which for the sufficiently low volume fraction preduces very accurate result (on the calculation of the Widom's μ value) and for the single particle configuration (where just the cylinder particle interaction is implemented), almost exact result .

System Configuration

See the Sec. 1.

Specifying parameters in the runner file named `runner_bhsow.sh`

All the parameters are as same as discussed in the Sec. 1 and Sec. 3, however a few more parameters specified in the input file is described as:

1. **Widom's Particle ID:**

```
#widom chemical potential parameters

echo "1" >> Base_data.txt  #particle type id inserted as widom particle,
                #...remember the type id 0 is reserved for the obstacles
                    #...and the particle type id assignment starts from 1
```

Particle type id for Widom's test particle. (Note: Particle type id `0` is reserved for cylindrical obstacles.) See the Sec. 2.

Compilation and Execution section

The simulation code `bcd_hard_spheroids_ obstacle_widoms.c` is compiled and executed as follows:

```
$ gcc bcd_hard_spheroids_obstacle_widom.c
... -o bcd_hard_spheroids_obstacle_widom -lm
$ ./bcd_hard_spheroids_obstacle_widom.c
```

Output file

The description of output files are as same as discussed in the Sec. 1 and 2

5 Instructions for running the code: bcd_hard_patchy_spheroids.c

This section provides the details of the code corresponding to the sticky spheroids decorated with the patches. The code requires an extra file with the file name `Patch_vector.txt`. The results and framework generated from this code is discussed in the reference [3, 2].

System configuration

See the Sec. 1.

Specifying the parameters in the runner file named `runner_bhps.sh`

All the other specification of the system remains as same as specified in the section 1, except the option to run the single particle diffusion is not available, which serves no purpose as the code is aimed to study the self-assembly (which is not possible without assigning the inter-particle interaction and switching it on).

The patch decoration and specification is given as below:

1. **Total type of patches:**

```
echo "1" >> Base_data.txt    #kind of patches in the system e.g. 2
```

This section defines the type of patches present in the system which will later be used to assign the ids of the patches and their interaction parameters. Patch type id will be assigned as 0, 1, ... etc.

2. **Switching the interaction between the type of patches, on and off:**

```
#kind of interaction among patches...
#...0 to switch off and 1 to switch on

echo "50.0" >> Base_data.txt

echo "1" >> Base_data.txt  # between all patches with type id 0 and 0
echo "0" >> Base_data.txt  # between all patches with type id 0 and 1
echo "0" >> Base_data.txt  # between all patches with type id 0 and 2
echo "1" >> Base_data.txt  # between all patches with type id 1 and 1
echo "0" >> Base_data.txt  # between all patches with type id 1 and 2
echo "1" >> Base_data.txt  # between all patches with type id 2 and 2

echo "50.0" >> Base_data.txt
```

This section tells which patch will interact to which one. In case of the more specification given than the number of total configuration possible for a given type of patches the lines will be skipped. For example if there is only two types of patches defined then the first

line will be $0-0$, second line $0-1$ and the third line will specify the interaction between the patches with ids $1-1$, and the rest of the specifications will be ignored by the code automatically.

3. **Range of interaction between the type of patches:**

```
#Patchy potential

echo "50.0" >> Base_data.txt #epsi

echo "0.2" >> Base_data.txt  # between all patches with type id 0 and 0
echo "0.2" >> Base_data.txt  # between all patches with type id 0 and 1
echo "0.2" >> Base_data.txt  # between all patches with type id 1 and 1

echo "50.0" >> Base_data.txt
```

This section specify the range between the patches interaction where parameters are assigned to the pair of patches in the same fashion as it has been done in the last section.

4. **Strength of the interaction between the type of patches:**

```
echo "50.0" >> Base_data.txt #beta

echo "0.002" >> Base_data.txt  # between all patches with type id 0 and 0
echo "0.002" >> Base_data.txt  # between all patches with type id 0 and 1
echo "0.002" >> Base_data.txt  # between all patches with type id 1 and 1

echo "50.0" >> Base_data.txt
```

This section specify the probability of bond breaking, which is similar to the temperature of the system. The assignment of the values to the pairs done in the same manner as in the last section.

5. **Individual patches geometric properties: central cone angle:**

```
echo "50.0" >> Base_data.txt #omega

echo "0.95" >> Base_data.txt  # for patches with type id 0
echo "1.0" >> Base_data.txt   # for patches with type id 1

echo "50.0" >> Base_data.txt
```

This section specifies the cos(angle), formed between the line passing through the center of the annular width and the patch vector. Extra specifications given than it required for the specified type of patches, will be ignored. Less specifications will be set to some default value.

6. **Individual patches geometric properties: central cone width:**

```
echo "50.0" >> Base_data.txt #del_omega

echo "0.05" >> Base_data.txt  # for patches with type id 0
echo "0.06" >> Base_data.txt  # for patches with type id 1

echo "50.0" >> Base_data.txt
```

This section specifies the annular width of the annular patch. For example if you want to create a solid cone along the patch vector, then will specify the central cone angle as 0.95 and the annular width as 0.05. Now each particle with patch vector forming cos(theta) greater than $0.95 - 0.05$ and lesser than $0.95 + 0.05$ will make a bond. So, in this way with proper width and the orientation angle the cone can be made solid as well as annular depending upon the requirement.

7. **Number of patches assigned to the each particle types**

```
echo "50.0" >> Base_data.txt #number of patches on each particles types
                             #default value is 0
echo "2" >> Base_data.txt    # number of patches on
                             #...the paticles with type id 0
echo "6" >> Base_data.txt    # number of patches on
                             #...the paticles with type id 1
echo "50.0" >> Base_data.txt
```

This section tells the code, how much patches are associated with the particle having a particular partile type id. Maximum rows of parameters will be equal to the type of particles defined in the system, any extra lines will be skipped and with less lines the remaining particle types will be assigned as zero number of patches.

8. **Decoration of the patches over the particles**

```
echo "50.0" >> Base_data.txt #Patches on each particle type

echo "0 0" >> Base_data.txt         # decoration of patches
                                    #...over the particles with type id 0
echo "0 1 0 0 1 1" >> Base_data.txt # decoration of patches
                                    #...over the particles with type id 1
echo "50.0" >> Base_data.txt
```

The first line: `echo "0 0" >> Base_data.txt`, will add two patches to the particle type id 0. Where both the patches will have type id 0. Similarly the second particle will be decorated with the 6 patches with patch type 0, 1, 0, 0 ,1, 1.

In this section, we decorate the type of patches over the each kind of particles. Where we put the type of patches (the patch type id), in the same number and sequence as desired, such that the decorated patches will be assigned to the patch vector supplied by **Patch_vector.txt**. All the patch width and patch angle is defined with reference to the patch vector. Here any specification in number more than the assigned number of patches for each kinds of particles will cause the code to show the error. Therefore, sequence and sum of all the patches specified in this section must match the sequence and number of patch vector supplied by the file **Patch_vector.txt**.

9. **Compiling and running the code**

```
gcc bcd_hard_patchy_spheroids.c -o bcd_hard_patchy_spheroids -lm
./bcd_hard_patchy_spheroids
```

This section compile and run the code, but one have to make sure that, there is the patch vector file named **Patch_vector.txt**, and the number of vectors defined in each row in the file matches with the number and sequence of the patches assigned in the decoration section.

Output Data file structure

The code prints the relevant quantities in the data file named **0.txt**, **1.txt**, ... and so on, where **0.txt** file stores the data for the physical time ranging between $1 - 100$ and **1.txt** stores the data for physical time $101 - 200$ and so on.

The data structure for each physical time is given as,

```
fprintf(cfu,"%lf\n", f[0]);  //  box length along x
fprintf(cfu,"%lf\n", f[1]);  //  box length along y
fprintf(cfu,"%lf\n", f[2]);  //  box length along z
fprintf(cfu,"%lf   %lf   %lf\n", 1.0, 0.0, 0.0); //  box vector
fprintf(cfu,"%lf   %lf   %lf\n", 0.0, 1.0, 0.0); //  box vector
fprintf(cfu,"%lf   %lf   %lf\n", 0.0, 0.0, 1.0); //  box vector

for (i=0; i<N; i++)  \\ N number of particles
{
    fprintf(cfu,"%lf  %lf  %lf\n", ami[i][0], ami[i][1], ami[i][2]);
    \\ position of ith particle

    fprintf(cfu, "%d\n", wbobb[i]);
    \\ bonded neighbours of ith particles

    for (j=0; j<wbobb[i]; j++)
    {
```

```
        fprintf(cfu,"   %d", bobb[i][j][0]);
        \\ jth neighbour of the ith particle
    }
    fprintf(cfu, "\n");

    rpid=pid[i];

    fprintf(cfu,"%lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf\n",
    ... li[i][0], li[i][1], li[i][2],  li[i][3],
    ... li[i][4], li[i][5],  li[i][6], li[i][7], li[i][8]);
    \\ orientation of the ith spheroidal particle for all the three axes
}
```

Full configuration of the system

The full configuration of the system is printed after each hundred time steps in the file named **relaxed_system.txt**. Which stores the position and orietation of all the particles and also the patch vector nearest neighbor periodic boundary condition marking and bonded particles and patches involved in the bonding. The structure is given as:

```
FILE* ku;
ku=fopen("Relaxed_system.txt","w");

fprintf(ku,"%d\n", sitt+1);  \\ physical time

fprintf(ku,"%lf\n", f[0]);  \\ box length along x
fprintf(ku,"%lf\n", f[1]);  \\ box length along y
fprintf(ku,"%lf\n", f[2]);  \\ box length along z
fprintf(ku,"%lf   %lf   %lf\n", 1.0, 0.0, 0.0);  \\ box vector
fprintf(ku,"%lf   %lf   %lf\n", 0.0, 1.0, 0.0);  \\ box vector
fprintf(ku,"%lf   %lf   %lf\n", 0.0, 0.0, 1.0);  \\ box vector

for (i=0; i<N; i++)
{
    fprintf(ku,"%lf  %lf  %lf\n", ami[i][0], ami[i][1], ami[i][2]);
    \\ position of ith particles

    fprintf(ku, "%d\n", wnrd[i]);
    \\ number of nearest neighbour of ith particle

    for (j=0; j<wnrd[i]; j++)
    {
        fprintf(ku,"   %d  %d", nrdi[i][j][0], nrdi[i][j][1]);
        \\ jth nearest neighbour and its periodic boundary mark
        \\ of the ith particle
    }
    fprintf(ku, "\n");

    fprintf(ku, "%d\n", wbobb[i]);
    \\ number of bonded neighbour of ith particle
```

```
    for (j=0; j<wbobb[i]; j++)
    {
        fprintf(ku,"   %d     %d     %d", bobb[i][j][0],
        ...bobb[i][j][1],  bobb[i][j][2]);
        \\ jth bonded neighbour of ith particles
        \\ and the patches involved in the bonding
    }
    fprintf(ku, "\n");

    for (j=0; j<npatch[pid[i]]; j++)
    {
        fprintf(ku, "%lf   %lf   %lf \n", paxe[i][j][0],
        ...paxe[i][j][1] , paxe[i][j][2] );
        \\ patch vector of the jth patch of the ith particle
    }

    fprintf(ku,"%lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  \n",
    ...li[i][0], li[i][1], li[i][2], li[i][3], li[i][4],
    ...li[i][5], li[i][6], li[i][7], li[i][8]);
    \\ oritentation vector for all the three axis of ith particle
}
fclose(ku);
```

Miscellaneous output files

There are many other output file generated in the same code for the ease to access some of the quantities directly without any need to analyze the data. Following are the list of such files,

- **Compressed_system.txt**: This file contains the full configuration data after initializing the system (removing all the overlap of the particles), which is printed for the purpose of debugging.
- **Cord.vtk**: This file is printed for the purpose of visualization in paraview after each hundred time steps.
- **Cord_bond.csv**: This file can be used in paraview to visualize the bond using the glymph cylinder after applying the calculator and table to point filter.
- **Cord_bond_type_k_k.csv**: This file is the bond configuration as the previous one but only for the bonds between the **k** patch type id.
- **Cord_symmetry_axes.csv**: This file indicates about the orientation of the particles.
- **Cord_vector_k.csv**: This file can be used to visualize the patch vector with patch type id **k**.
- **Diffusion_coefficient_el_l.txt**: This file prints the diffusivity averaged over all the particles after each 10 simulation time steps. The first column is the simulation time second column is the diffusivity in perpendicular plain of the symmetry axis in the particle's body frame. Third column is the diffusivity of parallel to the symmetry axis in the particle's body frame. Fourth column is the total diffusivity in the laboratory frame. Fifth column

is the rotational diffusivity. The purpose of this file is to debug the code. The wrong value of diffusivity can indicate about something wrong happening in the simulation.

- **Initial_Cord.vtk**: This file can be used to visualize the particles just after the code is initialized (after removing all the overlapping of particles from the system).
- **Resulted_patch_vector.txt**: This file is generated immediately after assigning the patch to each particles, which contains the orientation of all kinds of patches assigned to all kind of particles in the system.

6 Instructions for running the code: npt_hard_patchy_spheroids.c

This code can run the simulation in the constant pressure, number of particles and temperature ensemble. The detail of the algorithm and the result of the code is published in the given reference[3]. The code can run both the orthogonal box simulation as well as floppy box simulation. Where the simulation parameters is given through the input file **runner_nphps.sh**. This code requires a patch vector file named **Patch_vector.txt** (as described in the Sec. 5).

System Configuration

See the Sec. 1.

Specifying the parameters in the runner file named `runner_nphps.sh`

All the specification in the input file is same as described in the Sec. 5. However a few other specification is given as:

1. **Parameter for the box size and shape**

```
echo "1"  >>  Base_data.txt  #  0: cubic boxes  1: floppy box
```

   This tells the code to change the size or both the size and shape of the simulation box. In the former case, the box remains cubic, however in the later case it can take any shape.

2. **Specifying the pressure of the system**

```
echo "0.5" >> Base_data.txt #  pressure of the system
```

   This specify the pressure of the system in the unit $p\sigma^3/k_BT$. The minimum pressure should not violate the minimum limit of ϕ or maximum box size implemented in the code (which is $\phi \geq 0.004$).

   Note: This code does not require any volume fraction specification, as it fluctuates to keep the pressure constant.

Output file

Output file and the structure of the data files remains the same as discussed in the Sec. 5.

7 Instructions for running the code: `nvt_hard_patchy_spheroids.c`

This code can run the simulation in the constant volume, number of particles and temperature ensemble. The detail of the algorithm and the result of the code is published in the given reference[3]. All the specification is as same as in the Sec. 5, except this file simulate the particles with variable step length optimized to equilibrate the configuration faster. Therefore this code simulates just purely a Monte Carlo (NVT) ensemble. Movement of the particles follows the usual Monte Carlo acceptance condition.

System Configuration

See the Sec. 1 and 5.

Specifying the parameters in the runner file named `runner_nvhps.sh`

All the specification in the input file is same as described in the Sec. 5.

Output file

Output file and the structure of the data files remains the same as discussed in the Sec. 5.

8 Conclusion

This document outlines the configuration parameters and instructions to execute the BCD simulation as well as Monte Carlo simulation for hard as well as patchy spheroids. Ensure the parameters are set appropriately before running the simulation.

References

[1] Vikki Anand Varma and Sujin B Babu. Dimensional confinement and superdiffusive rotational motion of uniaxial colloids in the presence of cylindrical obstacles, 2024.

[2] Vikki Anand Varma, Simmie Jaglan, Mohd Yasir Khan, and Sujin B. Babu. Breaking the size constraint for nano cages using annular patchy particles. *Phys. Chem. Chem. Phys.*, 26:1385–1395, 2024.

[3] Vikki Anand Varma, Kritika, Jaskaran Singh, and Sujin B. Babu. Self assembly of patchy anisotropic particle forming free standing monolayer film. *Adv. Theory Simul.*, n/a(n/a):2200666, January 2023.

[4] Vikki Anand Varma, Isha Malhotra, and Sujin B. Babu. Enhancement in the diffusivity of brownian spheroids in the presence of spheres. *Phys. Rev. E*, 106:014602, Jul 2022.