**Note 1**

* So far we have successfully performed simulations for independent Brownian particles and confirmed that the simulation results are consistent with the corresponding theoretical predictions.
* In this lesson, we modify our simulations to calculate the diffusive dynamics of interacting Brownian particles for which fully analytical predictions are not available.
* Theoretical calculations based on such models, despite their apparent simplicity, can be carried out only by the use of computer simulations.

**Note 2**

* Because we already have a simulation code for independent Brownian particles, modifying it for interacting Brownian particles is rather simple.
* There are two main changes that we must implement, namely the inter-particle forces and the use of periodic boundary conditions.
* The former is obvious, if we want to consider interacting particles, we need to specify and calculate their interactions.
* The latter, while not strictly necessary, is often used in order to approximate infinite systems and avoid boundary effects in our results.
* The idea of periodic boundaries is quite simple; imagine that your finite-sized simulation box (colored in orange in the figure) is periodically repeated throughout space.
* The unit cell is usually a cubic box, which can be used to generate a perfect three-dimensional tiling that fills all of space, as illustrated in the figure.
* The orange unit cell, is surrounded by infinitely many blue periodic copies of itself.
* Note that all the boxes are exactly the same, which one you call the unit cell does not matter.
* In addition, since they are all the same, we only need to consider what happens in one of them.
* We just have to be careful to properly handle particle exchanges between cells.
* When a particle, shown here in blue as an example, passes through one of the boundaries of the unit cell, it enters the neighboring cell.
* This means that it re-appears on the opposite side of the unit cell with the same velocity.

**Note 3**

* To introduce inter-particle interactions, a well defined potential function must be defined for the system.
* Here we assume a purely repulsive soft-core potential to represent the excluded volume of each Brownian particle with a diameter equal to σ.
* The total potential energy of the total system is defined by Eq.(I1), where the sum is taken over particle pairs separated by a distance less than 2 σ only.
* The inter-particle force acting on the i-th particle is then given by Eq.(I2).

**Note 4**

* As always, we begin by importing the necessary numerical and plotting libraries.
* Compared to the previous code example, we import the additional module newaxis from the numpy library.
* The use of this newaxis module allows us to compute inter-particle interactions efficiently in Python by avoiding unnecessary for loops.

**Note 5**

* Compared to the previous code example, a few modifications have been done for the init function.
* The array F, which will be used to store the current values of the inter-particle force, is added to the list of global variables and initialized by setting all the elements to zero.
* Because particles should not overlap with each other due to the inter-particle interaction, we prepare the initial particle configuration using uniform random numbers by calling a function initconf that will be defined later.
* The remaining parts of the initialization code are unchanged.

**Note 6**

* In the animate function, we compute the inter-particle force F on all particles by calling the particleforce function in the 5-th line.
* This function will be defined later.
* In the 6-th line, we modified the velocity update operation to include the inter-particle force F just computed above.
* In the 14th and 15th lines, the object particles, which has to be drawn in the animation, is updated to use the current particle positions under periodic boundary conditions.
* This is done by calling the pbc function on the x, y, and z particle positions.
* You might wonder why we don't use the pbc function directly when updating the positions, in line 7 of the code.
* The reason for this is to simplify the analysis that we will perform later.
* In particular, calculating the mean-squared displacements becomes more difficult if we use the periodically wrapped positions.
* As written, our code keeps track of the particles that were initially in the center unit cell.
* While the positions stored in R and Rs might go outside of the center cell, if we know the position of one of the copies, we know the position of all of the copies.
* We just have to be careful when computing interactions between particles, to make sure that we properly take this into account.

**Note 7**

* To modify our previous code for the present case of interacting Brownian particles, we have defined 5 new functions as shown here.
* The pbc function applies the periodic boundary conditions to a vector 'r', assuming the size of the unit cell is 'box'.
* If any of the components of 'r' lie outside the unit cell, they are wrapped back to be inside the proper domain.
* As written here, the function works if r is a vector or a matrix (vector of vectors).
* The distance function computes the distance vector from r1 to r2.
* Given python's advanced array capabilities, the same function can be used to compute the distance between two particles, or between one particle and a set of particles.
* In the first case r1 and r2 are both 3d vectors, and the result is the vector r12.
* In the second case r1 is a 3d vector and r2 would be a n×3 matrix, and the result is a matrix, with row 'i' containing the distance vector between particle 1 and particle 'i'.
* If you want to write efficient python code, you should learn how these array broadcasting rules work.
* More information can be found online at the SciPy.org website [[https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html](https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html" \t "_blank)].
* The fij function together with the particleforce function computes the inter-particle force F acting on each particle by examining all nump^2 particle pairs.
* The initconf function is used to create a random particle configuration using uniform random numbers.
* To avoid the occurrence of particle overlap, the distance between all existing particles is examined. If an overlap exists, the particle position is rejected and a new random position is tested.

**Note 8**

* Here we set the system parameters and initialize all the particle variables.
* Compared to the previous code example, we introduced three new parameters sig, eps, and vf.
* sig and eps are the units of length and energy appearing in Eq.(I1), which we set to be both equal to one.
* vf represents the volume fraction of the particles =πσ3/6Vcell.
* Using the parameters defined here, we perform a 4096 step simulation for 100 interacting Brownian particles of diameter one, in a cubic box of side length 37 σ, which gives rise to a particle volume fraction of 0.001.
* This means that only 0.1% of the space in the unit cell is occupied by particles.

**Note 9**

* Now we can run the simulation, visualize the results, and store the trajectory data.
* On a standard pc it may take a few minutes until the simulation finishes, but please be patient to complete it.
* The trajectory data generated here is needed in the next step.

**Note 10**

* After your simulation ends successfully, let us calculate the mean square displacement which we already learned about in the previous lesson.
* By running the code example, you will find the mean square displacement as a function of time plotted in blue.
* The red line represents the theoretical prediction for non-interacting independent Brownian particles, which agrees fairly well with the present simulation results for interacting Brownian particles at the very dilute volume fraction of 0.001.
* The agreement is not perfect. As you might have noticed we were forced to run our simulation with 10 times fewer particles as before to get the simulation to finish in a reasonable time.
* Thus, our statistics are drastically reduced. The reason for this increased cost is the very expensive force calculation, which scales as N^2 (N the number of particles).
* For large systems composed of more than several hundred particles, you should not use the naive algorithm we have used here. Instead, you should use something like a link-list to only evaluate pairs that are close to each other.
* Finally, you could improve the calculation of the averages by allowing for different time origins (here we have only considered t=0), since any time can be taken as the origin for steady processes.
* This is outside the scope of our current lecture.
* By repeating the same simulation but with different values of the volume fraction, you can examine the density dependence of the diffusion constant.
* You will then find that the diffusion constant decreases rapidly with increasing volume fraction, going to zero as the volume fraction approaches a finite value.
* This drastic slowing down in particle dynamics is referred to as the glass transition, the true nature of which is not yet understood and is listed as one of the major unsolved problems in physics.