**Note 1**

* In this lesson, we will try to simulate the dynamics of a Brownian particle undergoing random thermal fluctuations, as dictated by the Langevin Equation.
* Today, we will write a python code that is as simple as possible, just enough to produce the time evolution of the particle.

**Note 2**

* In the previous lesson, we learned how to properly discretize the Langevin equation to perform a numerical integration of the equations of motion for a Brownian particle.
* In particular, we discussed that a naive application of Euler's method is not suitable. We must treat the random force term with special care.
* This is done by considering the cumulative impulse felt by the particle as a Wiener process.
* Thus, the time evolution of the position and velocity of a given particle can be updated through Eqs. (F5) and (F9). Where Ri and Vi denote the position and velocity at the i-th time step, and Δt is the time interval used for the integration.
* The effect of the random fluctuations is included as the update of the Wiener process over this time interval, ΔWi.
* This random process is described by a Gaussian random variable with zero mean and delta correlations in time and space, as given by Eqs. (F10) and (F11).
* Finally, the only remaining information we need to start the simulation is to specify the initial conditions, that is, the position and velocity of the particles at time zero defined in Eq.(F12).

**Note 3**

* So now, let us use Python to write a program for performing simulations of Brownian particles.
* As always, we begin by importing the necessary numerical and plotting libraries.
* In this plot, since we want to draw the trajectories of Brownian particles, which are evolving in 3D, we will use matplotlib with nbagg option and mplot3d module.
* The other libraries and commands have already been introduced and used before.

**Note 4**

* Here we specify the parameters of our simulation.
* We will work in 3 dimensions, x, y, and z, and consider the dynamics of one hundred non-interacting Brownian particles.
* We set the number of simulations steps to 1024, with a time step of Δt=0.05 in simulation units.
* You can change the number of steps, but for later purposes, please keep it as some power-of-two.
* Without loss of generality, we can set the friction constant ζ, particle mass m, and thermal energy KBT equal to unity.
* For simplicity, we compute one additional constant, "std" which gives the amplitude of the variance of the cumulative impulse, as defined in Eq. (F11).
* After initializing the random number generator, we create and initialize all the arrays needed for the simulation.
* R, V, W are the current position, velocity, and random forcing term at the current time step.
* Rs, Vs, Ws are the positions, velocities, and random forcing terms for the whole trajectory to be used for the data analysis.
* Time is just an array with the time values for each step

**Note 5**

* The main part of the present code example is shown here.
* To perform the simulation, we repeatedly solve for the position, and velocities, according to equations (F5) and (F9).
* This is achieved by using a for loop from i=0 to nums-1.
* For each iteration of the for loop, we use the current position, velocity and random force to update the particle configuration.
* Notice that the random forces, for each particle and spatial direction, are drawn from a Gaussian distribution with standard deviation given by "std" using “randn” function.
* After computing the current position and velocity using Eqs.(F5) and (F9), we make sure to save the values in the arrays Rs, Vs, and Ws to accumulate all past values for the data analysis we will perform later.
* Please run this code example.
* While it shows no output, the present simulation of 100 Brownian particles for 1024 steps will finish within a second or at most in a few seconds.

**Note 6**

* Let us now visualize the simulation data just obtained.
* By definition, the particles all exhibit Brownian motion along the x, y, and z directions, independently.
* To easily visualize this motion, we first Plot the trajectories of all the 100 particles in the x-y plane.
* Let us run the code example shown here.
* While the motions along the z-axis are not visible in this figure, you may examine them by plotting trajectories in the x-z or y-z planes as well.
* Notice how the density of the trajectories shows spherical symmetry, and is highest at the origin, where the particles were initially located.

**Note 7**

* In quite many situations, it is useful to visualize stereoscopic information in quasi 3D space on a computer screen.
* The bottleneck is that such 3D graphics usually require more detailed knowledge and computer programming skills.
* The Jupyter notebook, however, allows us to perform this task in an extremely easy way.
* This is a code example to plot the trajectories of all the 100 particles in 3D space.
* As you may notice, the difference from the previous 2D plot is rather small.
* In the 2nd line, only the use of the fig.add\_subplot command with the projection='3d' keyword creates an additional axis to the standard 2D axes.
* And the ax.view\_init command on the 10th line defines the initial view point of the camera, but the remaining part is the same as for creating the 2D plot.
* If you run this example, you will see the full 3D trajectories in the figure.
* By clicking and dragging on the graph, you can pan and rotate the camera to look at the trajectories from different angles.
* Through a careful examination of this 3D plot, you should be convinced about the spherical symmetry of the resulting trajectories centered at the origin.