**Note 1**

* In the previous plot, we wrote and used a very simple python code to simulate the motion of Brownian particles.
* Although the code is enough to produce trajectory data that can be used for later analysis, the strong graphic capability of the Jupyter notebook allows us to perform simulations with on-the-fly animations and also save them as movies quite easily.
* Today, I will show you how to take advantage of this graphics capability by modifying our previous simulation code to display the motions of Brownian particles in real time.

**Note 2**

* As always, we begin by importing the necessary numerical and plotting libraries.
* Compared to the previous code example without animation, we import two additional libraries, the “mlab” and “animation” modules from the “matplotlib” library.
* The roles of these new modules are briefly explained in comments after #, and more detailed information can be found in the official “matplotlib” website.

**Note 3**

* For this lesson, we will perform a simulation of Brownian particles and we wish to see how their positions evolve in time.
* In addition, we want to visualize the trajectory of one chosen particle as a line, to see how it moves in space.
* The easiest way to animate your data in python is to use the "FuncAnimation" function provided by matplotlib.
* To use this, we must define two basic functions that tell the library how to update and animate our data.
* The first of these functions is "init".
* As its name implies, it is used to initialize the animation.
* "init" will only be called once, at the beginning of the animation procedure.
* It should define the different objects or "artists" that will be drawn.
* Notice how we declare global variables explicitly in the function definition.
* This allows us to modify variables which are declared outside of the function.
* R, V, and W will contain the current position, velocity and cumulative impulse for each of the particles.
* Rs, Vs, and Ws are the corresponding values for all time steps.
* An array “time” will contain the time values.
* We initialize all the variables to zero from the 3rd to 9th lines.
* From line 9 to 11, we define three different objects to draw, "particles", "line", and "title".
* "particles" is used to display the particles as points in 3d space
* "line" is used to display the trajectory of a given particle
* "title" is used to display the current time
* Here, the particles and line data are created just as empty arrays and time is set as an empty string.
* These three objects will be modified later, when we call the "animate" function

**Note 4**

* The "animate" function is the main function used by "FuncAnimation".
* It is called at every time step, in order to update the figure and create a new animation frame.
* Thus, the animate procedure should be responsible for performing the integration of the Langevin equation in time.
* It updates the positions and velocities by propagating the solution to the Langevin equation over Δt.
* After the updated configuration is calculated from the 4th to 6th lines, we accumulate the new configuration in the trajectory variables Rs, Vs, and Ws from the 7th to 9th lines.
* Next, we update the objects in our animation.
* We set the title to display the current time at the 10th line.
* At the 11th line, we then set the line object, which displays the trajectory of a selected particle “n” to contain all the x, y, and z points until the current iteration step i.
* Finally, we set the current position of all the particles to be R.
* It is important that the “animate” function, as well as the “init” function, return the objects, in this case “particles”, “title”, and “line”, that are redrawn during the animation.

**Note 5**

* Here, we define the parameters of our simulations between lines 1 to 8.
* We will work in 3d, with 1000 particles at this time.
* We use a time step of 0.05 in simulation units and will simulate over a total of 1024 steps.
* We set the friction constant \zeta, mass “m”, and thermal energy “kB T” all equal to one.
* At the 8th line, we define the standard deviation “std” of the Wiener process in order to satisfy the fluctuation dissipation theorem.
* Finally, we create the necessary arrays. R, V, and W will store the current position, velocity, and cumulative impulse for each of the 1000 particles.
* Rs, Vs, and Ws will store the corresponding values for all 1024 time steps.
* and the time array will contain the time value for each step

**Note 6**

* Now we can run the simulation code and visualize the current particle positions.
* First, in the 1st line, we create a figure of size 10 inches by 10 inches, and we add an axis to this figure in the 2nd line .
* Note that when we draw objects, we draw them on the axis, not on the figure directly.
* Notice how the axis was explicitly set to be '3d'
* Next, we set the limits for each of the x, y, and z axis, as well as the labels from the 4th to 9th line.
* Using the view\_init command on the 10-th line, we specify the initial position of the camera.
* However, this is not fixed, as you are able to pan and zoom by clicking on the figure.
* The main code here is from the 11th to 13th line, where the empty objects or "artists", that will later be updated by the animate function, are created.
* These are particles, title, and line objects, which are all set to be empty.
* Notice however, that we specify how these objects will be drawn.
* That is, we can specify the line or marker type, as well as the color.
* These parameters will be used throughout the simulation, even though the underlying data will change as the particle positions change.
* Finally, we call the "FuncAnimation" function and specify where to draw, how to initialize, and how to update the animation.
* We must also specify how many frames, or steps to take.
* Let us now run the code. You will see an on-the-fly animation of the particles diffusing in space ...
* While all the particles are set to be at the origin at t=0, the particles start expanding radially outward.
* This is a very similar problem to how a drop of ink will diffuse through a container of water.