**Note 1**

* In this plot, we will analyze the trajectories generated by our Brownian simulation code to calculate the diffusion constant using three different approaches based on the theoretical knowledge we introduced previously.
* First, we will use the mean-squared displacement and velocity auto-correlation function obtained from an equilibrium simulation.
* Then, we will perform a non-equilibrium simulation with an external drift force exerted on the Brownian particles and compute the diffusion constant from the average drift velocity.

**Note 2**

* Let us perform a simulation to produce the trajectory data for the motion of 1000 non-interacting Brownian particles.
* As always, we begin by importing the usual numerical and plotting libraries.
* The code we use here, is exactly the same as the one we used in the previous lesson.
* We solve for the motion of 1000 Brownian particles and save the trajectory data for their position, velocity, and random forces at each time step.

**Note 3**

* We first calculate the diffusion constant via the mean square displacement using Eq.(H1).
* Note that since all the particles are initially at the origin, the displacement is simply given by the particle position.
* In the code shown here, we define an array msd, of size 'nums', which gives the mean-squared displacement at each time step, averaged over all the particles.
* The diffusion constant can then be obtained by integrating the msd in time, as shown in Eq.(H2).
* We perform this integration using numpy's built in 'trapz' function, which evaluates the integral through the trapezoidal rule.
* The numerical value of the diffusion constant is 0.93, which is in good agreement with the theoretical prediction of 1.0.
* Finally, we have plotted the mean-square displacement as a function of time, and compared it with the theoretical value of 6Dt, which gives a straight line of slope '6 D'.
* While the curves seem to be slightly different from each other, what is important here is the fact that the slope of the curves is the same, this means that the diffusion constant is the same.

**Note 4**

* Next, we calculate the diffusion constant via the Green-Kubo formula using Eq.(H3).
* In the code example, we again use the built-in 'correlate' function to compute the velocity correlation function.
* To obtain the appropriate average, we sum over the correlations for the x, y, and z components for all particles.
* The diffusion constant is then obtained by integrating this correlation function in time and dividing by three.
* The evaluated value of the diffusion constant is now 0.94, which is again in good agreement with the theoretical prediction.
* In the figure, we have plotted both the correlation function computed from the simulation, as well as the theoretical curve.
* The agreement between the two is excellent.

**Note 5**

* The third method we use to evaluate the diffusion constant is to apply an external drift force.
* In this example, we will add a constant external force in the x-direction to all the particles.
* By measuring the response of the system, in particular the average velocity along the x-direction, we will be able to compute the diffusion constant.
* The code we use here is exactly the same as before, the only difference is the addition of the external force 'F0'.
* The force is defined in lines 8 and 14, and used in line 22 to update the particle velocities (in addition to the friction and random forces).

**Note 6**

* We next calculate the diffusion constant under an external drift force using Eq.(H4).
* As shown in the equation, the diffusion constant is proportional to the average velocity along the x-direction (the direction along which the external force is applied).
* To evaluate the averages we make use of the built-in 'average' function.
* Here, we should be careful to specify the appropriate axis along which the average should be calculated.
* Vs, which contains the velocities of all particles for all times, is a 3D array, where the first axis refers to the time values, the second to the particle, and the third to the spatial dimension.
* Since we want an average over particles, we would specify 'axis=1'.
* This returns a 2D array 'Vsa', where the first axis refers to time, and the second to the spatial dimension.
* Finally, we compute the time-averaged x velocity by calling the 'average' function on 'Vsa[:,0]'. Note that we have selected only the x-component, so this is a 1D array and we do not need to explicitly specify the axis.
* The evaluated value of the diffusion constant is 0.98, which again is in excellent agreement with the theoretical prediction.
* In the figure, we have plotted the time evolution of the average particle velocity as a function of time.
* Note that the y and z velocities are fluctuating around zero, while the x component of the velocity fluctuates around a non-zero value determined by the diffusion constant.