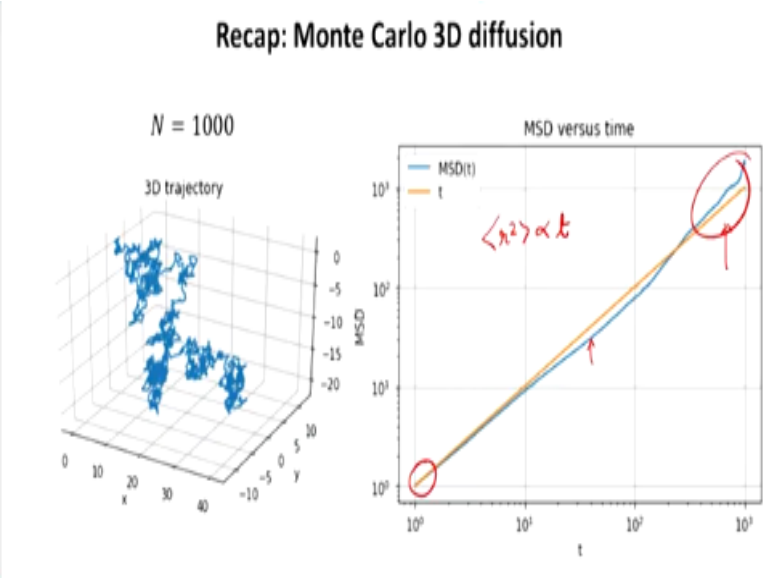
Advanced Thermodynamics and Molecular Simulations
Prof. Prateek Kumar Jha
Department of Chemical Engineering
Indian Institute of Technology, Roorkee

Lecture - 38
Numerical Implementation of Monte Carlo Simulation: Python Examples III

Hello all of you, in the last lecture we have been discussing some examples of Monte Carlo simulations, we discussed some simple examples like computation of π in the beginning and then we discussed how can we simulate the motion of one particle that is a diffusion in 2 dimensions and 3 dimensions. So, today I will take it further I will discuss things like boundary conditions that is the first thing that will come in the today's class and then we will go towards the discussion of the how do we simulate multi-particle system.

(Refer Slide Time: 01:05)



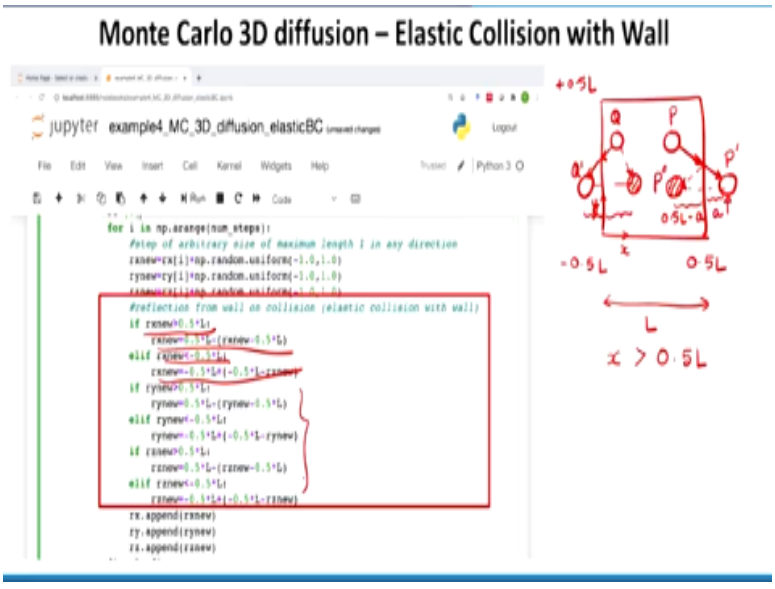
So, very quickly to recap the idea, I was simulating the 3D trajectory of a single particle that is undergoing a random walk where every step is of a constant length and we looked at the mean square displacement versus time and it should follow a linear relation with respect to time because the motion is diffusive this is what Einstein relation tells me and we discussed that there is some problem in the beginning and towards the end. In the beginning it is rectilinear motion that is of course not captured in the model that we did but it is present in a multi particle system but towards the end we have issues with averaging because we have lesser number of trajectories containing large number of steps.

So, I want to continue on that idea but, now I want to put the condition that the particle is actually bond bounded in a box so as to speak. So, there is a simulation box and the particle has to move within the simulation box and in that case we require some sort of a boundary condition. So, let us see how can we then implement the boundary conditions in there, and since I was telling you that ultimately you are simulating only a small part of the system because you are limited by the number of particles you can simulate of course this is one particle system but we are soon going towards a multi-particle system.

So, therefore, it is important that the physics is not altered because we are doing a smaller system and one of the essential part of that physics is the Einstein relation that the particles must undergo a diffusive motion that diffusion coefficient that relation of the mean square displacement time should not be affected at least very strongly, when we are doing any kind of a boundary condition if it is somewhat artificially imposed for example in the case of periodic boundary condition that we discussed.

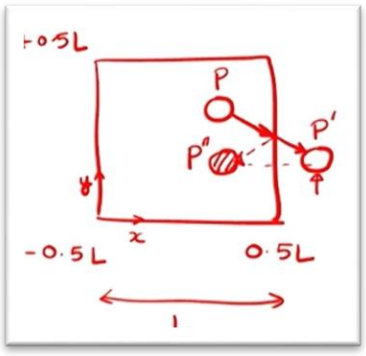
So, let us start with the simplest possible boundary condition and we see what is the problem with that so already said that in an actual physical system when we look at the part of the system that is going to be a small volume within the system and therefore for that small volume since it can exchange particles with the other volumes in the system there is no physical walls present, nonetheless you may imagine that there is a rigid wall fixed wall surrounding the system and once the particles hit there it comes down comes back just like an elastic rebound.

(Refer Slide Time: 04:03)



So, let us try to see how it works out and then, we see what the problem with doing that is. So, essentially what we are trying to simulate now is I am giving example in 2D but you can also think the same thing in 3D, I am doing the code in 3D but example is given in 2D. So, let us say for example a particle comes and the step is such that it wants to eventually come here. Now since it cannot really go there what it can do, it can somehow rebound to this particular position that is like an elastic kind of a rebound, and this is what I am trying to model in this particular part of the code where, I am trying to model the rebound effect inside there.

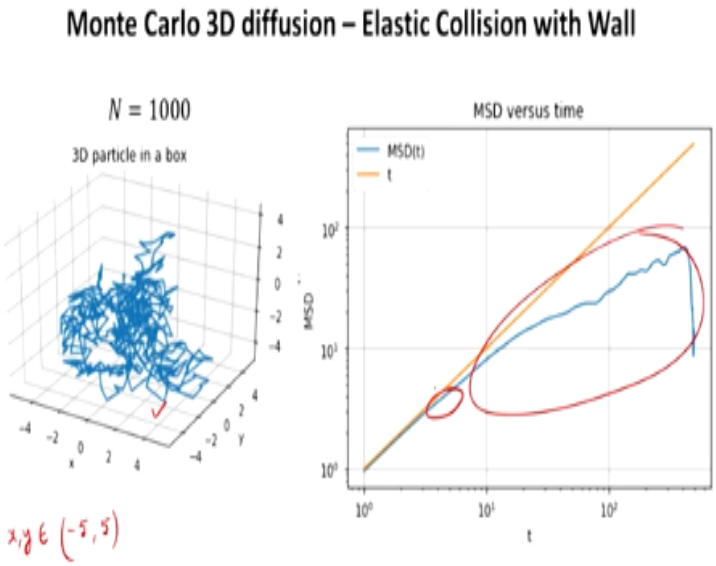
So, the key idea in this particular rebound that I have done is that let us say if I define my coordinates as x and y here, then this is where the particles should have gone so let us say if the particle is named P and this P prime is where the position should have been and P double prime is the position because of the wall that is the rebound. So, now you can notice that the y coordinate of the particle remains the same because in both these scenarios we are only exceeding the x coordinate beyond the bounds.



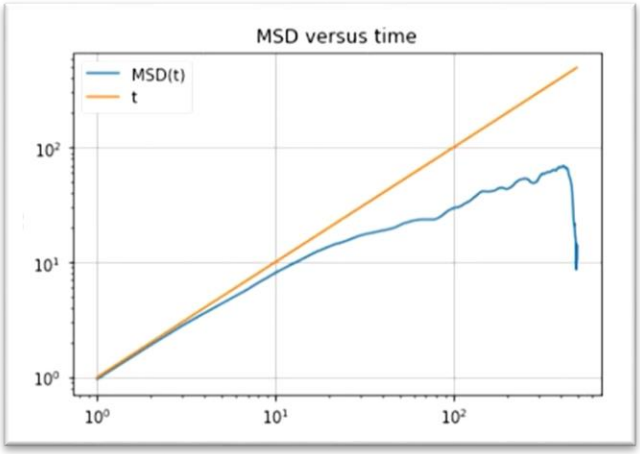
So, let us say the box was defined from minus 0.5 L in x to 0.5 L that gives me a box length L, and the same thing in y then whenever my x is more than 0.5 L in that case I have to find a new value that will be inside the box and how much of that will be that, that is pretty much how much I exceed that has to be subtracted from the wall boundary. So, let us say if I exceed a distance 'a' along the x direction then the new position will be something like what we could have went past that should be subtracted from the 0.5 L, so the new will be like 0.5 L minus of that particular distance and if x is within the bounds then we do not do that the same applies on the other side of the boundary, So, let us say for example if I start from the Q here and it wants to go to somewhere here this is my Q prime it cannot go there so it will come somewhere over here and that position will be basically added to the minus 0.5 L and this is what exactly what we are doing in this particular code.

So, in the previous code that we had discussed we were generating the new positions as the old position plus a random step size but now I am going to check for whether we are having a collision with the wall or not and this is precisely what we are doing in this particular code. So, if for example the X coordinate is more than 0.5 L then we basically do the subtraction and, if X is less than that then we do the operation that we suggested and the same thing is done for the Y and Z coordinates, if any of these coordinates are already within the bounds then we pretty much do nothing. We only change the coordinate when they exceed either 0.5 L or they are less than minus 0.5 L so this will be a modified code that will take care of the collision with the wall, and let us see what we get from here.

(Refer Slide Time: 08:47)



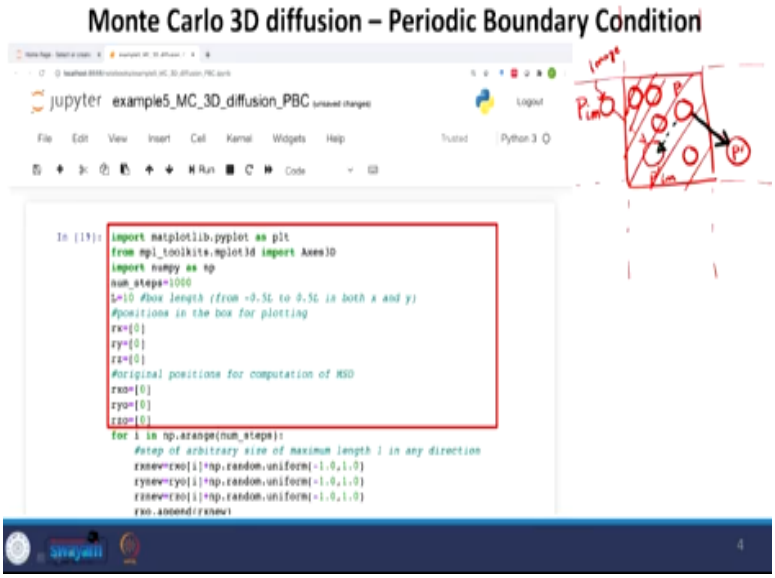
So, clearly now the trajectory is kind of bounded because my box size was 10, so x goes from minus 5 to 5 and y goes from minus 5 to 5 and clearly this is what you see here the trajectories are contained within the box so that part is okay.



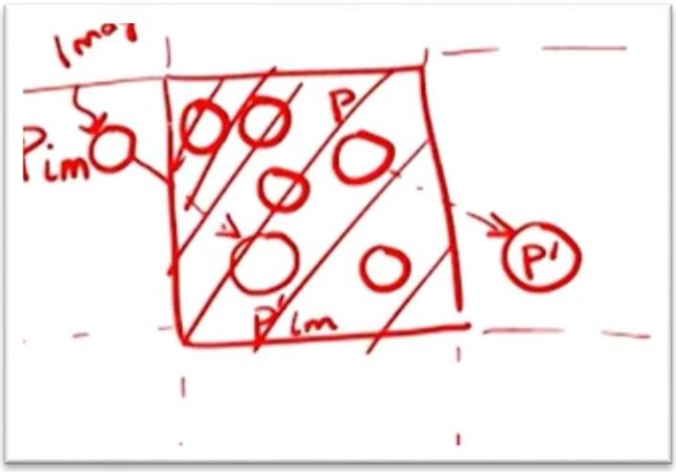
However if I look at the mean square displacement versus time it is no longer forming a linear relationship with time actually there is a very large reason actually even from very beginning there is starting off the deviation from the line and therefore we clearly know that this is probably not a good representation of the large system because ultimately the walls were artificially introduced because there was no wall in reality.

So, unless we want to simulate a system that indeed has a wall there is something wrong with the boundary condition that we are doing if it indeed has a wall it is okay, but if it does not have a wall then we have a problem. So, the next thing therefore we do is we resort to the use of periodic boundary condition. Now the same problem in periodic boundary condition will go like this, as I have been discussing earlier.

(Refer Slide Time: 10:09)



So, now let us say you had this particle P and it wants to come somewhere over here, So, now what I do is I let that particle go out but since there are images of the particle in the nearby systems or what we can say as replicas of systems that we have assumed around it we are simulating only the central box but we are assuming that the same configuration is repeated everywhere in the system. So, there would be then another particle that is right here that is my image, let me call it some image of P_{im}, now as P comes to P prime the P_{im} also does the same thing and it comes within the box.

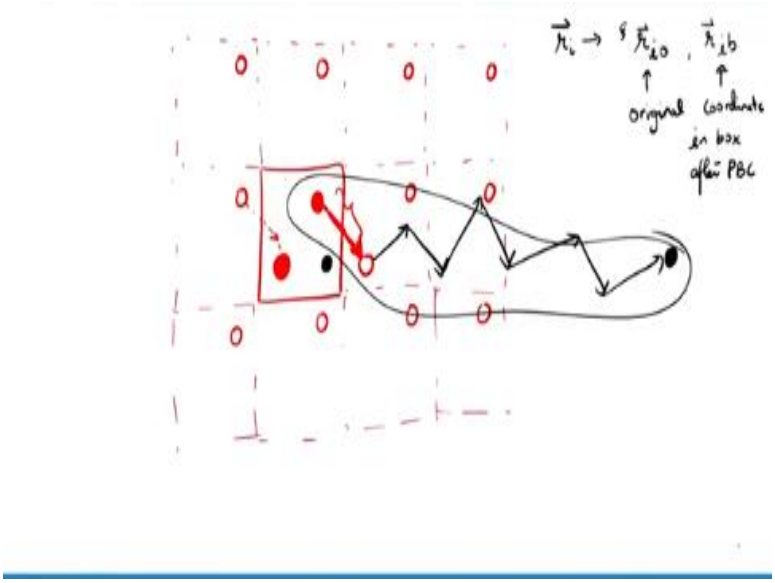


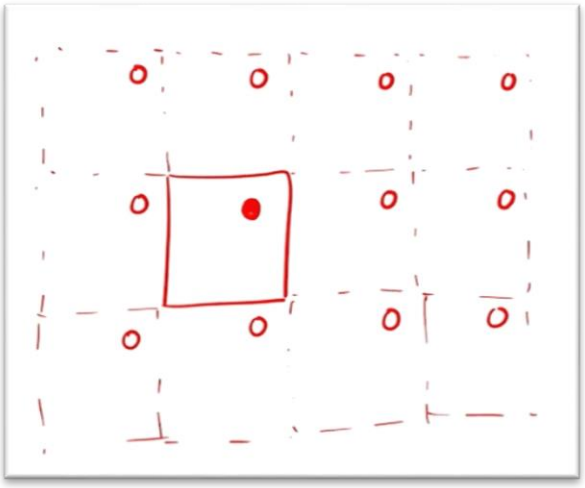
So, therefore, now within the new system after the movement there are still two party, there is still that particle is present except that now one of the images has come and substituted for that particular particle the number of particles remains the same one particle moved out so another particle came in from the opposite face and that is the whole idea of periodic boundary conditions for most of what we care we only consider the particles which are within the box at any given time and clearly these includes any of the images that moved in the box in lieu of the particle in the box going out. So, at any given time we only look at you can say the particle or its image that is present in the simulation box and that takes care of most of the stuff.

However if you think in terms of the diffusive motion this creates bit of a problem because the actual motion of P was this so within one Monte Carlo step this was the step in black that it actually took, but since we are only considering the images that are present inside the box, I would essentially find a step that is the dotted line here and that is clearly not the case, when we look at the actual motion of the particle. So, what we do is for the purpose of computation of the diffusion coefficients we track the original portion positions of the particle.

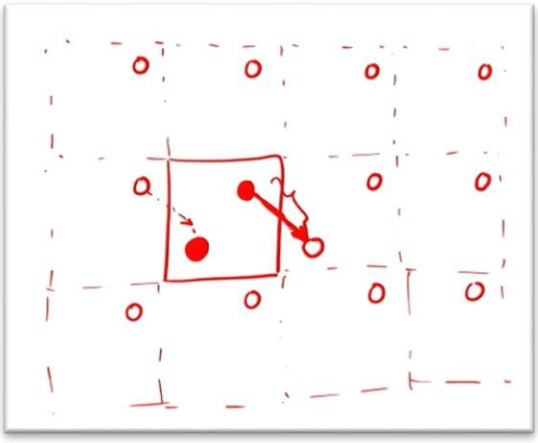
Just to take a longer example so let's say this is my simple simulation box and now we have images around it, so clearly I am considering only the box drawn in the bold lines here and let us say we are looking at a particle inside the box. So, clearly we are going to have its image in all the boxes but we are not simulating the images so as to speak we are simulating only the one that is present inside the central box.

(Refer Slide Time: 13:25)

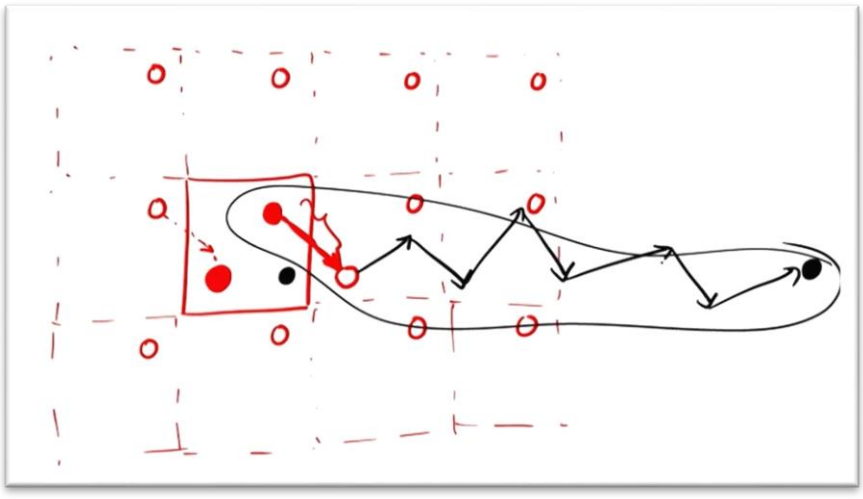




Now let us say this particle moves in here to a new position like here, then the image actually moves in to basically the same position here and now I will color that full because that is the position I am considering within the box. So, at any given time one particle left and another particle came in so there is no problem for the box but when we look at the actual motion we should have looked at this line as the actual motion that took place.



So, now, let us say this happens over several steps and the particle moved somewhere far off let us say the particle came here, now in after all these steps there will still be an image that will be somewhere here that will correspond to some other particle that has come from some other box, some other image box you can say in lieu of this particle that we lost. But if I want to look at the diffusion motion I should look at this original trajectory this is precisely what we do.



So, when we use a periodic boundary condition we save two coordinates, one is the original coordinates assuming that there was no periodic boundary condition so particle is moving, moving, moving. So, I just keep their original positions and then I also save a position that will lie inside the box and that would be of the image. So, I can define two coordinates of the particles let us say ri. So, for ri can define two coordinates one I can say r_{io} that is my original coordinate and then another one let me call it some r_{ib} that is the coordinate in box, after the application of the periodic boundary condition and this is what we have to do in the code.

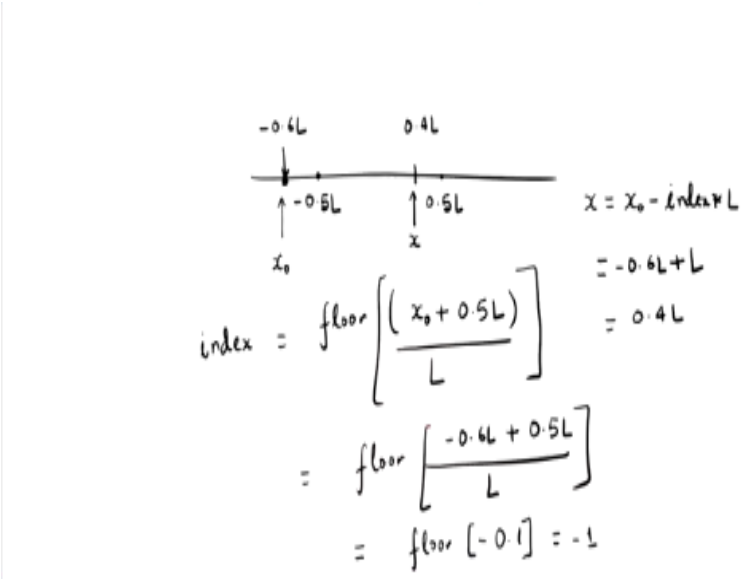
So, whether we really have to compute the position or do we only care about the distances so can we work in terms of the original coordinates only it turns out that we can apply some tricks and using only the original coordinates you can work but when you need the image coordinates it can be found as some transformation of the original coordinates that has to be possible. But in general you want to save probably both these coordinates or at least theoretically speaking you should have some idea about both the original and the coordinate that will lie inside the box. So, this is precisely what we will do in the code when we employ the periodic boundary condition.

So the initial part of the code remains pretty much same as earlier so we use some plotting libraries, some access libraries, we use numpy library, we are performing for 10,000 steps we have a box size of L equal to 10 and so on and now, I am defining two coordinates r x and r xo. So, r xo is the original coordinates that I will use for the computation of the mean square displacements and r x, r y, r z are the coordinates which I will use for showing where the particles are that is for the purpose of visualization or for anything else that we can see in the other examples how that can be useful. So, this is like coordinate after PBC correction.

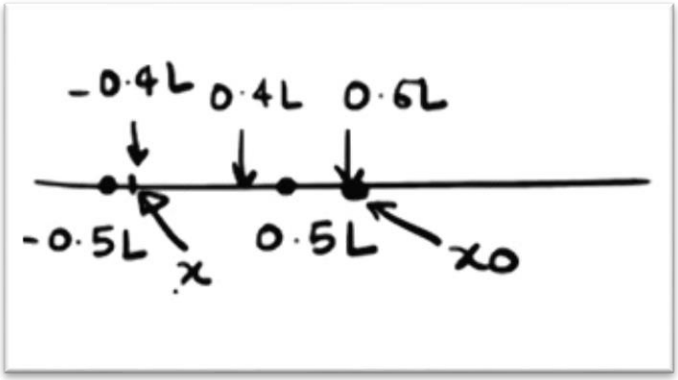
So, now, we are within our Monte Carlo code, so you may have noticed here and also in the previous example that I was not dividing the step with normalized the norm of it, that is I am not taking a constant step of certain thing that really makes no difference so as to speak in the diffusion process, but that is something that we can keep in mind if we need to keep a fixed step we can indeed do that but we are not doing in this example. So, we are simply taking a step that can be any value in x direction between minus 1 to 1 any value in y and any in z, If we require a constant step we can also do that but I am not doing in here. So, what we do is we find the new position of the particle in terms of the original position, so the original position has moved so, I find the new position in terms of the original position of the particles and then I append in the array for the original position that is r xo, r yo and r zo.

Now in the next step I will start performing the Monte Carlo simulation so this is what we are doing here so we are taking we are taking a step and we are appending in there that is what we have done earlier. Now this is the part where I am applying the correction for the periodic boundary condition and we play a very smart trick here what we say is we compute, so let us say I take a simpler example where I am looking only along the x direction.

(Refer Slide Time: 23:58)



So, let us say the particle was originally here at 0.4 L and now it has moved to 0.6 L. So, the original position is xo is this, now you will have an image inside the box and we can find the image easily in the example by simply subtracting L from 0.6 L. So, the image will lie one box before that is L length before this and that will be going to lie at minus 0.4 L and that becomes the x after the periodic correction. So, 0.6 L should be the original position after the movement and minus 0.4 L should be the position after the periodic boundary condition correction.



So, with this kind of an idea I can apply an operation like what I am doing here I can define a quantity called index that is defined as-

$$index = floor\left[\frac{x + 0.5L}{L}\right]$$

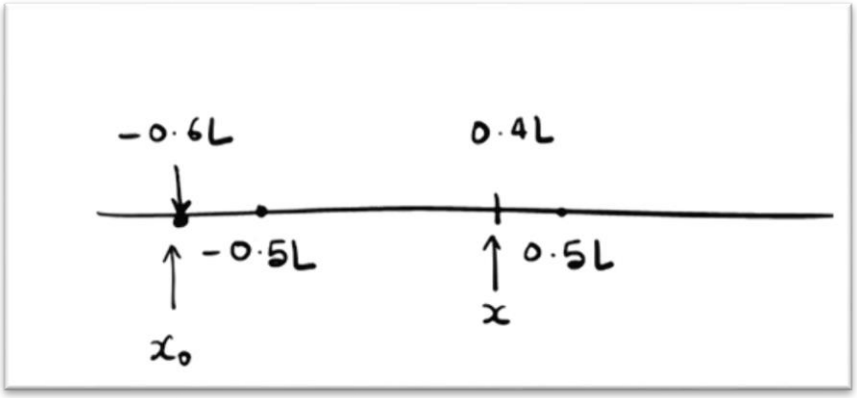
and what is that quantity is floor of so x in this case is so will put an xo here actually because we are doing it for the original coordinate. So, it is going to be-

$$index = floor\left[\frac{0.6L + 0.5L}{L}\right] = floor[1.1] = 1.0$$

and floor is an operation where we look at the integer just below whatever number we have. So, for 1.1 the floor is going to be 1, for 1.2 the floor is 1 for 1.9 also the floor is 1 only when we exceed 2 the floor is going to be 2. So, it is an integer just below the number that we are considering. So, this will give me 1 and then what I do is I define my new position as-

$$x = x_0 - index * L = 0.6L - L = -0.4L$$

This is the operation that we are we are doing, we can think of another example when we are moving on the other side. So, let us say for example again I am looking along the x coordinate and let us say after the movement the particle comes here, that is at minus 0.6 L. So, this becomes my x_0, the new position should lie between minus 0.5 L to 0.5 L after the periodic boundary condition correction. So, in this case if I add L to this, I get 0.4 L and this becomes my x now let us see, if we can get the same thing by the formula.



So, index is going to be-

$$index = floor\left[\frac{x_0 + 0.5L}{L}\right] = floor\left[\frac{-0.6L + 0.5L}{L}\right] = floor[-0.1L]$$

So, what is the integer below it so we are now in the negative domain so integer below less in value compared to minus 0.1 is going to be minus 1 and therefore my x is going to be-

$$x = x_0 - index * L$$

$$x = -0.6L + L = 0.4L$$

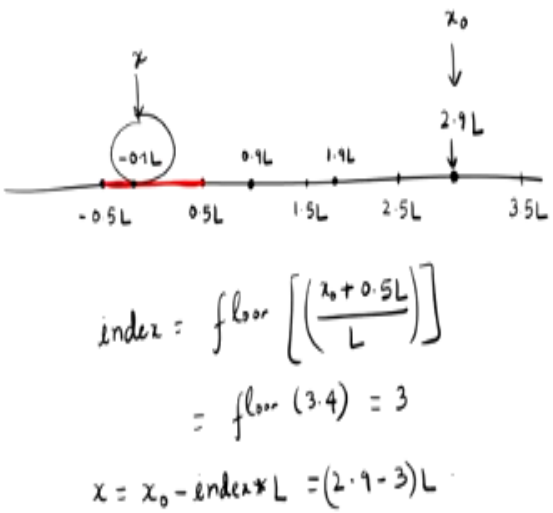
and this is what we wanted.

It turns out that in the both the examples that I have done here, I was only moving to the just immediately as adjacent box but if I look in course of several steps that is not necessarily true

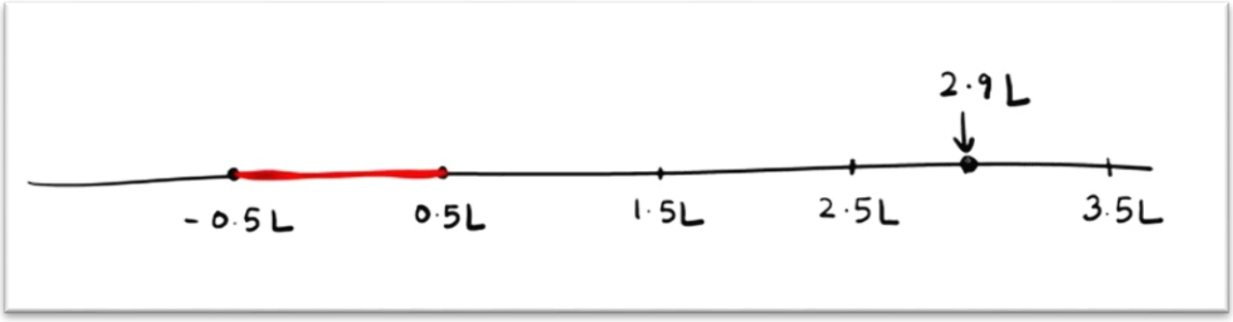
so I can move to several boxes to the right or several boxes to the left or actually a combination of that some boxes to left and some boxes to the top or bottom of the system. So, that particular formula, that we have used kind of takes care of that and let us see how it works. So, again I will look at example where I am looking only along the x coordinate the same will apply along the y and z coordinates as well.

So, let us say this is my original box let me color that and let us say the particle is somewhere here, after several steps so clearly in one step the particle cannot jump to several boxes to the right or left if that is happening you probably do not want that we will come to that point later. So, it should be like movement should be much smaller than the box size in practice I will come to the that point later, but after many, many steps it can get to any box to the right.

(Refer Slide Time: 26:34)



So, let us say for example this is the immediately next box then going from 0.5 L to 1.5 L and then we have another box 1.5 to 2.5 and the next box is 2.5 to 3.5 and let us say this is at something like 2.9 of L.



Now I apply the same formula here. So, I want to find the position that will lie inside the box it is very easy to find the position. So, if I move one box to the left of 2.9 L, I will come to 1.9 L if I move one more box to the left I will get like 0.9 L and if I move one more box to the left I will get at minus 0.1 L. So, all these points minus 0.1 L, 0.9 L, 1.9 L are the images of 2.9 L this is the one that is the x value after the correction and this is my x_0 value.

So, what is going to be the index? Index is going to be-

$$index = floor\left[\frac{x_0 + 0.5L}{L}\right] = floor[3.4] = 3$$

and therefore x is going to give me-

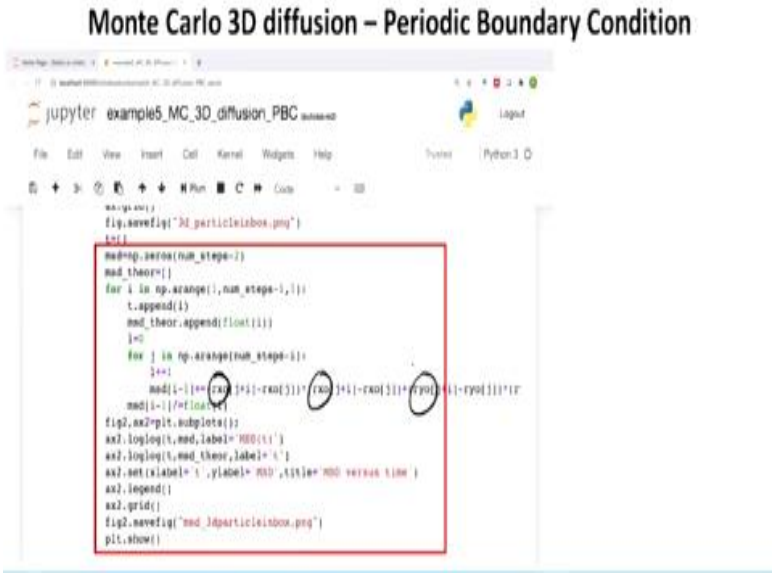
$$x = x_0 - index * L \ = (2.9 - 3)L = 0.1L$$

So, the formula that we have used is smart enough to do it for even jumps beyond one box and the same we can so when we are looking at on the other side of the box that is towards the left. The same will apply there as well I already showed you for movement to the next immediately next box but the same will apply also in this case, so these are the small tricks that will really help you make the codes efficient that is using appropriate arithmetic operations that can or numerical operations that can simplify the computation without adding too much effort on the computer I could have done that for example by using some kind of a switch statement or if statement or using something else that require more lines of code but in this particular case I found that this formula really works well even when we are going to many boxes to the right or many to the left.

So, this precisely what we are doing in the code here so I have to find now three indexes in X Y and Z directions and I will apply the correction and therefore I get a new set of coordinates

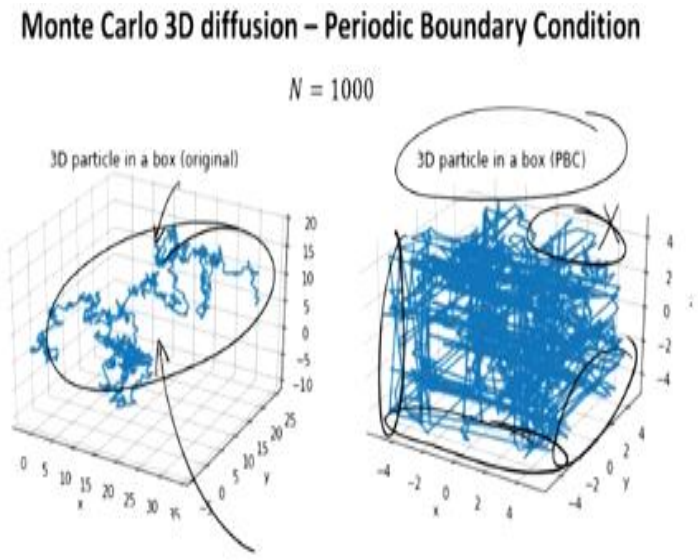
that will lie inside the box and this is what I will append in the array that is keeping the coordinates after the PBC correction or the coordinates within the box that I am simulating.

(Refer Slide Time: 31:01)

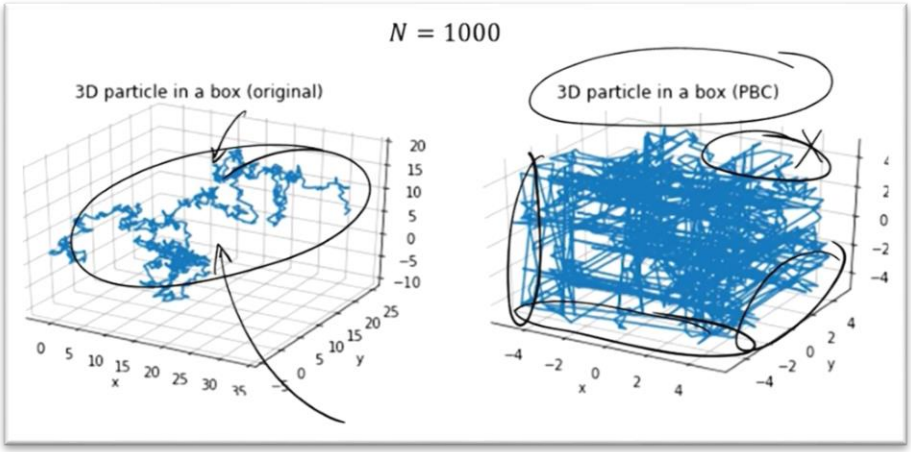


Now I go ahead and plot it so when I plot the trajectory I plot in terms of the coordinates after the collection, so I will only see particle moving within the box that I am simulating but when I compute the mean square displacement I will use the original coordinates. So, this is what we are doing here so in this case actually I am if you look at it I am plotting both, so I am plotting the original one and I am also plotting after the correction and in the next step I find the mean square displacement the code remains pretty much same as the earlier example except that we are doing it over the original coordinates as opposed to the coordinates inside the box.

(Refer Slide Time: 31:53)

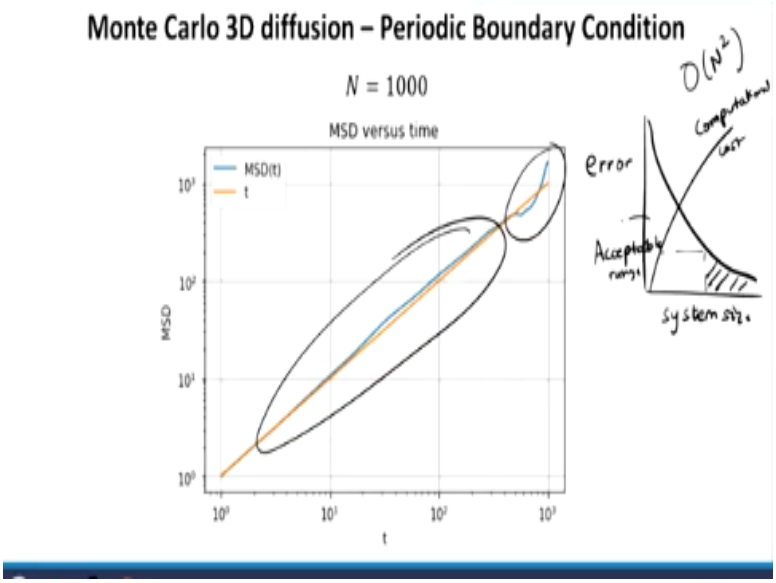


And now you see, so I am showing here the original coordinates so now clearly the original coordinates are not bounded the box size is like - 5 to + 5 and in the original coordinates the particle is going from something like up to 35 in the x direction and like 25 in the y direction the particle is moving too far from the original box but after the PBC correction we are looking at the coordinates within the box and now you can see there are these large lines which are coming because of the periodic boundary condition, so the original trajectory was this but since some other particle came inside the box when one particle left and that was the image of the particle that left it is clear that the trajectory of that particle image is not the same as the trajectory of the particle, so for the purpose of MST we should therefore use this and not this but for the purpose of showing, where the particles are in the box this is probably a good representation.



And we can compute the mean square displacement with time and it is perfectly I would say very close to linear line there are still some problems out there but it is much better than the fixed boundary condition problem.

(Refer Slide Time: 33:13)



So, having said this there is a small point here that we have to keep in mind and that is the approximation of the periodic boundary condition will improve as we are going to larger and larger systems and therefore, we should make sure that the mean square isplacement for example the diffusion coefficient if that is the property we are interested in is good for the system size that we are simulating good meaning that it is showing a linear relationship if that is not happening then there is not a problem with periodic boundary condition as such the problem could be that the system size is too small if I increase the system size then the error will reduce.

In other words the error of the periodicity because, periodicity anyway is an artificial assumption. I was telling you that essentially physically speaking it comes down to the fact that there is a repetition of the system like a crystal and that clearly is not the case when the number of particles are small that is really a very poor approximation, only when the system is very large and the number of particle in system is very large that becomes a reasonable approximation.

So, the error will reduce as the system size increase. and therefore we should try to seek some kind of system size that is giving me the error within some acceptable range that we can tolerate let us say for example it is if I look at diffusion coefficient if it is say within 5% of the value that I will get without the periodic boundary condition or for an actual experimental system that is probably good I mean that number is something that we have to choose but the choice of that number is also dictated by the fact that my computational cost is going to increase

substantially as we are simulating more and more particles in the system so under the same condition if I am looking at a multi particle system the number of particle is also going to increase for one particle system it does not matter, what is the box size but when we are doing the multi particle system then in that case as I increase the system size and I want to simulate the same concentration of particles then the number of particles has to increase and I was telling you that the force computation goes like N square, so if I double the system size my computation will be like four times if I assume that the force computation is the main bottleneck here.

So, clearly even though the error will reduce by using larger box size and you will be tempted to think that I should make the system size as large as possible that is limited by the computational cost and therefore you always have to make some sort of a compromise in terms of how much computation you can afford and how much error you can tolerate, in most cases you have to keep slightly larger estimates of the tolerance limit the acceptable range for the Monte Carlo simulation to work with the computational efficiency that you desire.

So, that becomes like I mean physically speaking there is no problem we can keep on increasing system size and all that in fact for infinite system size it becomes like the system without any boundaries. But can we simulate that and the answer is no, so, we therefore have to make a compromise between the errors and the computational cost.

So, with this particular example I want to conclude the discussion today in the next lecture we will see how we extend the idea of the one particle simulation for the case of multiple particles.

Thank you.