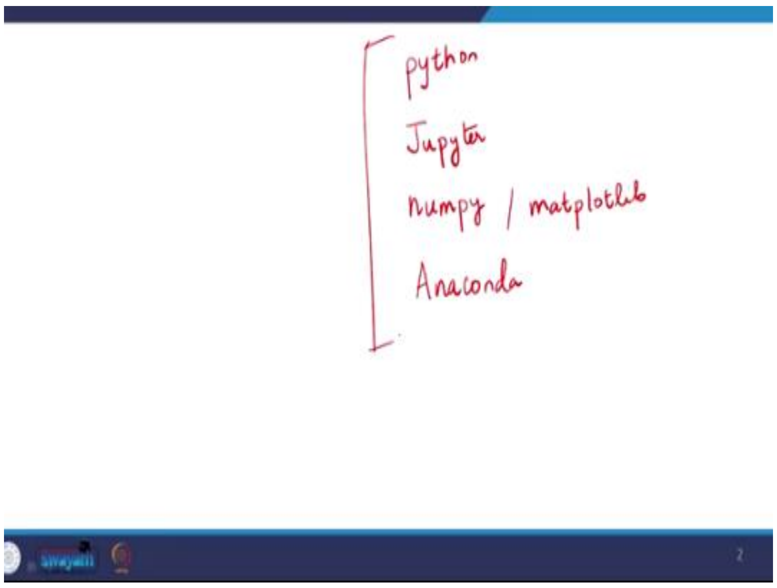
Advanced Thermodynamics and Molecular Simulations

Prof. Prateek Kumar Jha
Department of Chemical Engineering
Indian Institute of Technology, Roorkee

Lecture - 36
Numerical Implementation of Monte Carlo Simulation: Python Examples I

Hello all of you. So, in the last week, we have been discussing the basics of the Monte Carlo simulation method. So, in this week, we will discuss how do, we actually numerically implement that on a computer and we will first do the basic algorithms and then we will discuss some of the case studies where we can employ the Monte Carlo simulations.

(Refer Slide Time: 00:54)



python
Jupyter
numpy / matplotlib
Anaconda

So, before I proceed I want to mention here that all the programs that I will do will be done in the python language. If you look at the actual molecular simulation programs, they are mostly written in either C, C++ or FORTRAN but python is much easier to work with that is why I have chosen python. But keep in mind that if the efficiency of the program is not so good particularly for very large systems, then we typically have to move to C or C++ but for the purpose of demonstration I think python is much easier language.

So, if you are comfortable with python, it is great. If you are not very comfortable it is very easy to learn you can go to the python website and learn the basics of the python language. I will try to go through some of them in the course that I will discuss but clearly whenever you are in doubt, you can go to the python website and that is a great reference for learning python.

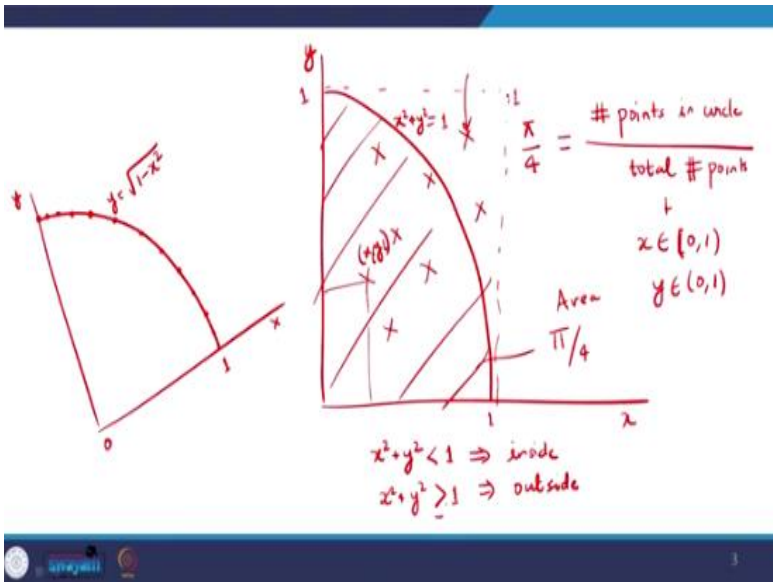
Now there are several development environments in python. The one I will use is Jupyter and just like python the Jupyter also is kind of open source, you can download for free and when you download it you typically download some kind of library of pythons in addition to the basic python code. The ones library that I would be using are numpy and matplotlib but there are a whole host of libraries like scipy and all that.

So, it is better to get an installation of python. One of the packages where python comes packaged with all these libraries is the Anaconda package you can download on any platform that you have a Windows, Mac or Linux and you should be able to install then the Jupyter notebook and typically when you open the Jupyter notebook it opens in a browser that does not to mean that it needs internet to run. It can be offline but it is just more comfortable to work in the browser. It does not have any separate editing thing, you can just operate in the browser.

So, all the examples I will show you will tend to show in a browser but keep in mind that the codes you can pretty much code in your computer. I will try to provide the codes later in the course. I will tell you where to get it from but for the time being I am pretty much giving you all the code or at least the code snippets that are necessary for whatever we will discuss.

So, with this background, let us first start with the first demonstration of the Monte Carlo scheme where we are not doing thermodynamics where we are interested in the computation of π. I already have demonstrated the algorithm or how can I use Monte Carlo for π computation.

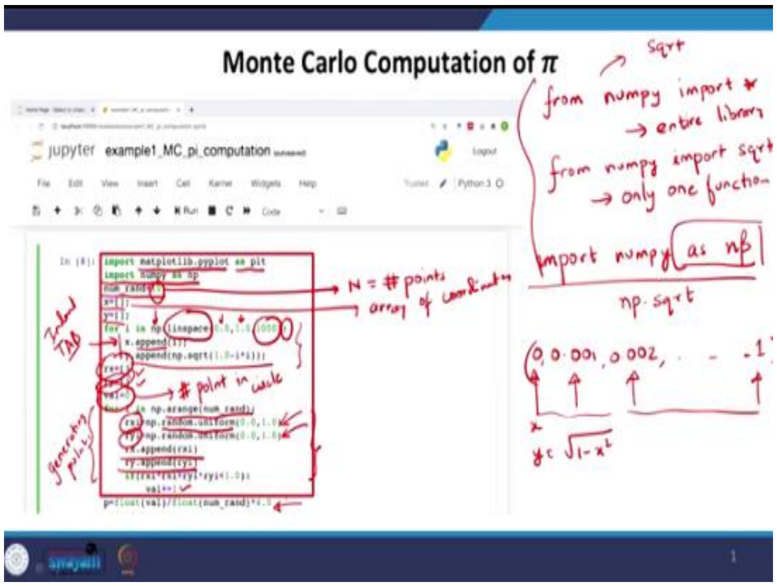
(Refer Slide Time: 04:06)



$y = \sqrt{1-x^2}$
$x^2+y^2=1$
$\frac{\pi}{4} = \frac{\#\text{ points in circle}}{\text{total }\#\text{ points}}$
$x \in [0,1)$
$y \in (0,1)$
Area $\pi/4$
(x,y)
$x^2+y^2<1 \Rightarrow$ inside
$x^2+y^2 \geq 1 \Rightarrow$ outside

Now I want to do the same thing on a computer. So, just to, quickly recall. The idea was I will look in a unit circle and one quadrant of that because four times that will be the area of the circle. So, the area of this thing is therefore π by 4, excuse my drawing. So, this is 1 here and 1 here. It is indeed a circle, the quadrant of that and the way the Monte Carlo should work is I imagine a square again of unit length and unit width and I generate points within that square and then the number of points that lie within the circle. So, the number of points in the circle divided by total number of points that I generate in the range x going from 0 to 1 and y going from 0 to 1 and that will give me that ratio of the areas that is π by 4 divided by 1 that is the area of the square and from therefore I can get the value of π.

(Refer Slide Time: 05:39)



Monte Carlo Computation of π

```
In [8]: import matplotlib.pyplot as plt
        import numpy as np
        num_rand=[illegible]
        x=[];
        y=[];
        for i in np.linspace(0.0,1.0,1000):
            x.append(i);
            y.append(np.sqrt(1.0-i*i));
        rx=[];
        ry=[];
        val=0
        for j in np.arange(num_rand):
            rxi=np.random.uniform(0.0,1.0);
            ryi=np.random.uniform(0.0,1.0);
            rx.append(rxi)
            ry.append(ryi)
            if(rxi*rxi+ryi*ryi<1.0):
                val+=1
        p=float(val)/float(num_rand)*4.0
```

Now let us see how it is done on a computer. So, essentially whenever I start a python code, I first import all the libraries that I need for the python code and there are several ways I can import the libraries. If I am using many functions from that library I can just import the entire library something like let us say I am using a numpy library. If I say from numpy import* I can import the entire library and of course as you may expect since that is ultimately going to be a burden on the RAM of the computer that is typically not a good idea if you are using only some command in the library.

So, you can also import a particular function let us say my code only needs the square root function I can say from numpy import sqrt and now the code can interpret sqrt because it is coming from the numpy library. So, only one function. We can add more functions by adding a comma, let us say sqrt comma some other function name and so on and then what we can also do is and that is necessary for when we are using the code with multiple libraries because sometimes what may happen is that the same function name is defined in two libraries that you are using. So, in that case there can be a possible conflict.

So, therefore instead of importing it as it is you can import as some particular name. So, let us say import numpy as some name I am giving, let us say np. So, now I am importing the entire numpy library, but now the function has to be called by something like np. sqrt, sqrt for square root.

In the earlier case I could have done simply sqrt and the code will interpret. So, by as and the name that I can give so if you give any name as I have chosen the name np. So, then it is basically imported as some kind of an object and the functions have to be called from that particular object. So, this is standard coding practice.

In most cases it does not make any difference how you are doing it but just as a good coding practice I will be using something like that many cases or something else that I will also see later. So, that is the first thing. So, every code does not really need libraries. I mean, like I want to do basic arithmetic operations even the basic version of python without any libraries can do that but if you need some sort of mathematical functions, it is typically essential to use numpy libraries. Similarly when you want to do some kind of matrix manipulations and all that sometimes you use also a scipy library and if you want to do some kind of plotting you need some plotting libraries and there are a whole variety of rich libraries.

And the good thing is that most of them are open source. So, we do not have to pay any money in doing that. So, with this particular idea if you look in this particular code, I am importing two libraries; a matplotlib dot pyplot that is for the plotting functions I will use in the code and I am importing as plt.

So, I will call the command from the library as plt dot whatever the function name is. Similarly I am importing numpy as np. So, I will call all the function from numpy library as np dot something. And then I define num rand is equal to 10, that is the number of points that I generate.

So, clearly you can use any number here the more numbers you put you may expect there will be more accuracy but at the same time it is going to take more time and but in this particular simple code you can pretty much do a very large number of points and it runs within a minute. So, then I need to store the coordinates of the points that I generate. So, in the example I was showing you all the points we will have some coordinates x y, let me call it something like x_i, y_i where i goes from 1 to n the number of points. So, I want to store those values because using that coordinates only I can find out whether the points will lie in the circle or outside the circle.

So, therefore I create an empty array to hold these x and y coordinate values. So, the array will contain the coordinates. So, one of the good things about python is that the flexibility with which you can use the arrays. So, when you are used to a C language you either define an array size as a-priori and if I want to increase the size in between the core it is quite complicated. You can of course use pointers and all that, that is somewhat better but of course more computationally more difficult to code but in python it is extremely easy on the behind of it may be doing something of that sort what C is doing. But when you are coding it you do not have to worry so much about all these things. It is much much easier in this particular language that is python.

And if you are also coming from the C background you may recall that you have to do many includes, hash include this, hash include that and so on that is not really needed in the python. It is much easier to compile and run. Of course, we are calling libraries but that is of course true so that is true also in the C, let us say I want some mathematical function I use the math dot x library. So, I have to include that and so this particular import functions are kind of analogous to that. But apart from that for the basic version of python you do not have to do any kind of inclusions and all that the compiler is much simpler to interpret both for the user and also on the behind effect.

So, then what I really want to do is I want to basically have some mechanism in which I can plot that circle. So, I want basically what I want from here is something that looks like the drawing that I have that is not necessarily required to do but just for the sake of plotting it will look nicer if something like this happens. So, I want to basically show an axis and show the circle and clearly we know that the circle is given as something like-

$$y = \sqrt{(1-x)^2}$$

So, in a computer the way to do it is I will compute y value for many points in the range 0 to 1 and basically plot that array. So, that is what I do here. So, I basically compute the y function as sqrt of 1 - i square where i is so this is the definition of a python array. So, what it tells me is i is going in the range 0 to 1 that is the range and I am generating thousand points between 0 to 1. There is no Monte Carlo here so far. I am just plotting the circle.

So, I am generating 1000 points between 0 to 1 and the points are uniformly located and this is done by the ‘linspace’ function that is present in the numpy library that I have called using np. And for i in that means I will start with the first element of that. So, ‘linspace’ will give me something like I want thousand points between 0 to 1. So, 0.001, 0.002 and so on until 1. That is what the array would be. And when I say for i in this it will in the first time it enters the loop it will start from i is equal to 0. Next time it will take i is equal to this, next time it will take i is equal to that and so on and for all these cases this becomes the x value, whatever I pick from here and the y value I get like 1 - x square, that is 1 minus i square.

And then now once I have done that so now I want to generate my points and to do that I again declare two arrays, rx and ry that will contain the x and y coordinates at the point. Before I get there keep in mind that how the ‘for’ loop has been defined here. So, I put a column after the ‘for’ loop that means that the loop has started and then this tab that I did or an indent I did is very important because as long as I am indented from the, for loop so this part is indented from the ‘for’ I am inside the ‘for’ loop. As soon as I go back to the same indent as the; for, let us say when I come to the r x it is in the same line as ‘for’ I am out of the loop.

So, it is much simpler than the C way of doing it of defining curly brackets and all that, you simply do a tab and shift tab to enter the loop and exit through and also there is an append function that is appending the value. So, I start with an empty array x and y and I am appending the new value of x as I am going in the loop and I am running the loop from 0 to 1 for thousand points. So, it is simply storing the thousand points in the array. So, at the end of the first ‘for’ loop you will have a thousand point array of x and y that represents the circle, the quadrant of the circle.

Now I am starting with the points themselves. So, I declare again array ‘rx’ and ‘ry’. This ‘val’ I have defined this will store the number of points in circle. So, before starting to generate the points I said that to 0 is to initialize it and now every time I find a new point in there I will simply increment it by one. It will be seen in a minute. So, now I go in the ‘for’ loop, that is where I am generating my points. And now I am generating something like a range of num rand. So, what it does is since num rand is 10 the number of points that I have started with what it will do is create an array from containing 10 elements from say 0 to 9. So, every time you enter the loop the i will be the element in the array. The first time it is 0, then 1, then 2 and the last will be n minus 1. So, i will be 9 this pretty much tells me that I will enter the loop 10 times or n times the number of points.

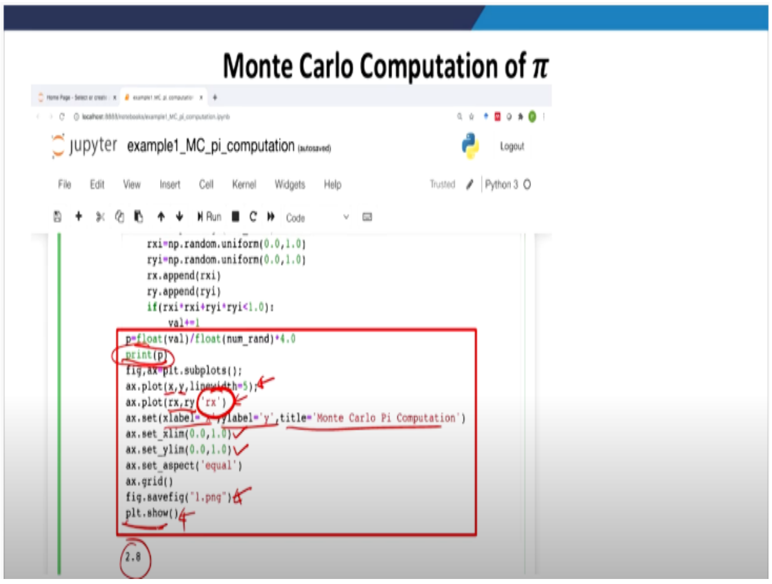
Now once I have done that then I want to uniformly generate points randomly, uniform distribution but randomly between 0 to 1 and that has been done by the random dot uniform function in the numpy library that I am calling as np. So, I generate a value of x and a value of y once in the loop and then since the loop is done 10 times or the number of point times this will be done for the number of points that I have started with.

So, I am saying that the x coordinate is rxi and y coordinate is ryi. And naturally I will append that into my rx and ry array that is going to contain the coordinates of my x and y points of the points that I am putting in. Now, I want to check whether the point lies in the circle or not and the way to check it is if $x^2 + y^2$ happens to be less than 1 then it is inside because the equation of a circle is $x^2 + y^2$ is equal to 1, it is a unit circle. If it is more than that then we are lying outside. So, outside will be $x^2 + y^2$ greater than 1.

We can say something like that with precise but friendly speaking we are doing in the range 0 to 1, so that does not really matter. So, then that is what I check for this is the condition where I am checking for whether the x square plus y square for the point is less than 1. So, x for a point is rx i, y for a point is ryi. The sum of the square is if it is less than 1 then I mean that I am inside the circle and I increment the value by 1. That is what I am doing here. If on the other hand, If it is greater than 1 that would have been the else condition. But I did not write it. Then the value will remain unchanged. So, either this condition is correct in that case I increment the value by 1 or the condition is wrong in case I do nothing because value remains same as earlier.

So, this loop will be run for the number of points that I have and then in the end I compute the number of points that are inside the circle that is given by val divided by the total number of points that is given by num rand and multiply by 4. And that will give me the value of π from the Monte Carlo code.

(Refer Slide Time: 21:21)



Monte Carlo Computation of π

jupyter example1_MC_pi_computation (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

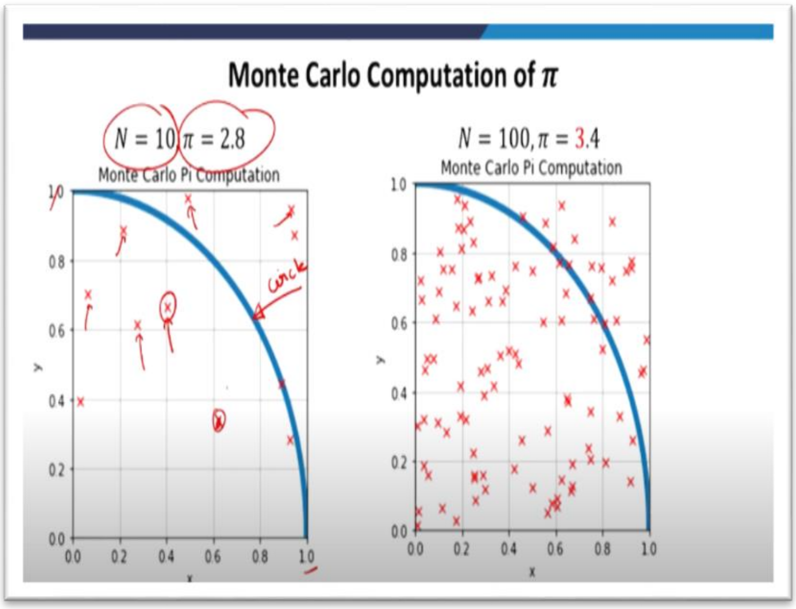
```
        rxi=np.random.uniform(0.0,1.0)
        ryi=np.random.uniform(0.0,1.0)
        rx.append(rxi)
        ry.append(ryi)
        if(rxi*rxi+ryi*ryi<1.0):
            val+=1
p=float(val)/float(num_rand)*4.0
print(p)
fig,ax=plt.subplots();
ax.plot(x,y,linewidth=5);
ax.plot(rx,ry,'rx')
ax.set(xlabel='x',ylabel='y',title='Monte Carlo Pi Computation')
ax.set_xlim(0.0,1.0)
ax.set_ylim(0.0,1.0)
ax.set_aspect('equal')
ax.grid()
fig.savefig("1.png")
plt.show()
```

```
2.8
```

So, with this particular idea I can then proceed to print the value of π and plot it. So, first of all, I am printing the value of π on the screen and that is the first thing that will print in here and then I want to also plot it. So, I want to plot first the x and y for the circle. So, it will show the circle nicely. Then I want to plot the points which are the arrays rx and ry. I want to plot in the range 0 to 1 in the x, 0 to 1 in the y we can set the aspect ratio to be equal and you can also have some title of the plot, some label of x, some level of y and you can beautify the plot the way you want to be.

The key point here is when I am plotting the rx and ry that is the points that I am randomly generating I am doing with something like a label rx right here. And that means it is r means red. It is going to be a red point and x means it will show like crosses on the screen. On the other hand when I am doing the plot for the circle I am not putting that thing that means it will simply plot a line. So, you will get a line for the circle and you will get crosses, red crosses for the points that you will generate in there and then you can also save it, give it some name, save it on the computer and you can also show it on the screen. So, you have to do the show command and then only it will show and also save the file there.

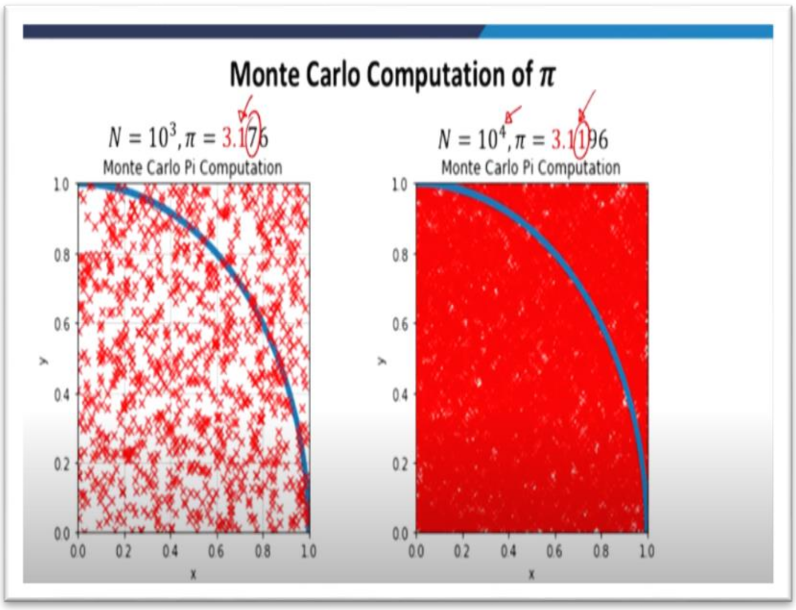
So, you can customize all these things and I am not going in details here pretty much this part of the code remains the same in many functions. Once you know how to plot it the same thing applies everywhere else simply the array definitions will change the x and y will change. So, once we have done that this is what we get-



So, we have plotted the circle, I am doing in the range 0 to 1 in x, 0 to 1 in the y. So, the x is pretty much defines the unit square that we have and these are the points that I am generating. And then I am simply finding the value of π by counting the number of points inside the circle dividing by total number of points and multiplying by 4. So, in this particular case if I run for n equal to 10, I get a π value of 2.8.

Now keep in mind that you will not get the same answer if you do it on your computer. Even if you run the same code twice you will not get the same answer because every time the code is running it is generating a new set of random numbers. It is a uniform generator so where you generate points will keep on changing as you do the code. So, every two runs of the code will not give you the same coordinates of the points and that is the whole idea of Monte Carlo that we can pretty much sample the entire reason. The points can be uniformly generated anywhere in there. So, you have equal probabilities to generate points here versus here or somewhere else in the system.

So, with this particular idea keep in mind that you will get a value of π and clearly it is not close to 3.14. It is quite far from there. If you run it again, you may get a value closer to 3.14 or slightly farther from this. We do not know we can keep on trying but it is still not too far because 2.8 is still very close to I mean, like itself close to 3.14. The error is still as you can say is around 10% from the value of point. But now if I increase the N to 100, so clearly we have more points and therefore we expect that we will have a more accurate value of π and that is kind of true. The first decimal seems right but we are still not there may be that if I run again I may get 3.15 if I am fortunate enough. But still I am not very close to the π value.

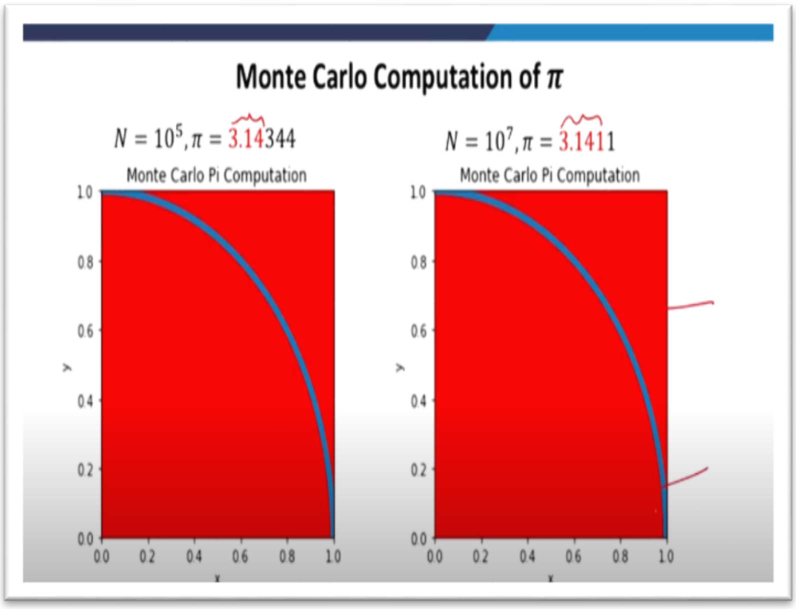

Monte Carlo Computation of π

$N = 10^3, \pi = 3.176$

$N = 10^4, \pi = 3.1196$

So, then I can do it for thousand and now you can see I get the first decimal. We are getting more points in there. And if I do 10^4 now the points are pretty much filling that entire square and then even my second decimal is kind of getting there. Not 3.14 but it is kind of 3.12 that is close to 3.14, if I compare to something like 3.17 here.

(Refer Slide Time: 26:27)


Monte Carlo Computation of π

$N = 10^5, \pi = 3.14344$

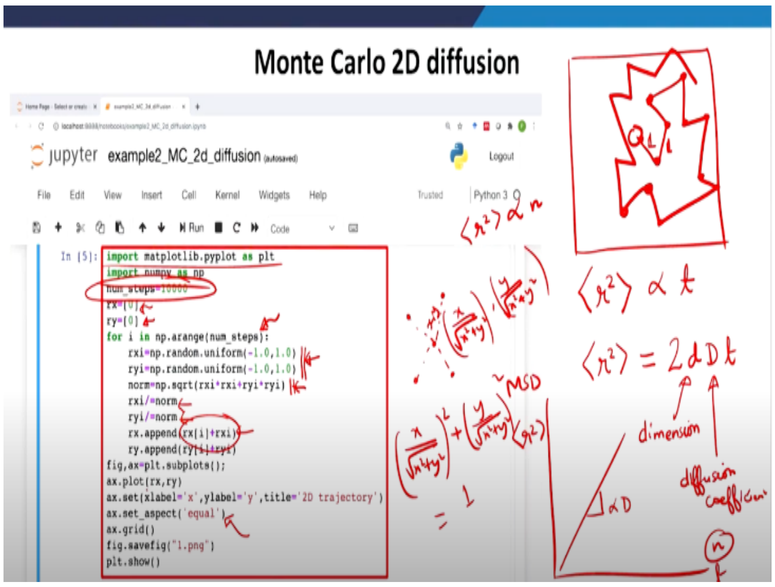
$N = 10^7, \pi = 3.1411$

I keep on doing it and this takes way less time to run because very simple code. For 10^5 points I get both the decimals, first and second decimal. And if I am doing it 10^7, then I am getting pretty much all the 3 decimals and you see now this entire region is now red and the reason is because I have generated way too many points. So, I have pretty much sampled the x and y space.

Now even though it seems like full. It is not still full because the markers I was drawing had a size and that is why it seems to fill the entire region. Markers were actually points on the diagram. Since I could not generate a point of 0 width that is why the markers have a finite width. So, it may seem like full but it is not quite full. But we have 10^7 points. That is the key idea and that is quite hues and then the accuracy is also quite reasonable and this pretty much tells me that Monte Carlo is an efficient way to sample the space. This is not a thermodynamic example but nonetheless you see the beauty of the Monte Carlo. I started with randomly generating points and as soon as the number of points started to increase I was able to sample the entire region from 0 to 1 in x and 0 to 1 in y. So, clearly the sampling part is quite efficient.

We do not expect that with just 10 points you can get the correct answer but with hundred we are quite reasonable we can say and with thousand or ten thousand we are even closer and beyond 10^5, we are pretty much getting the two decimals or three decimals correct. So, this is how a Monte Carlo scheme actually integrates or performs a numerical integration. And this is what I wanted to do to demonstrate on how we can use the Monte Carlo for numerical integration.

(Refer Slide Time: 28:42)



Monte Carlo 2D diffusion

jupyter example2_MC_2d_diffusion (autosaved)

```
In [5]: import matplotlib.pyplot as plt
        import numpy as np
        num_steps=10000
        rx=[0]
        ry=[0]
        for i in np.arange(num_steps):
            rxi=np.random.uniform(-1.0,1.0)
            ryi=np.random.uniform(-1.0,1.0)
            norm=np.sqrt(rxi*rxi+ryi*ryi)
            rxi/=norm
            ryi/=norm
            rx.append(rx[i]+rxi)
            ry.append(ry[i]+ryi)
        fig,ax=plt.subplots();
        ax.plot(rx,ry)
        ax.set(xlabel='x',ylabel='y',title='2D trajectory')
        ax.set_aspect('equal')
        ax.grid()
        fig.savefig("1.png")
        plt.show()
```

$\langle r^2 \rangle \propto n$

$\left(\frac{x}{\sqrt{x^2+y^2}}\right), \left(\frac{y}{\sqrt{x^2+y^2}}\right)$

$\left(\frac{x}{\sqrt{x^2+y^2}}\right)^2 + \left(\frac{y}{\sqrt{x^2+y^2}}\right)^2 = 1$

$\sim$ MSD

$\langle r^2 \rangle \propto t$

$\langle r^2 \rangle = 2dDt$

So, now I want to come to my next example and that is the example of a 2-dimensional diffusion. So, now we will do a diffusion process and diffusion is represented as a random walk. So, you already have discussed the drunkard walk we were doing a walk in one dimension. The drunkard who is fully unconscious kind of he has totally forgotten the memory and now he is making steps randomly to the left or to the right.

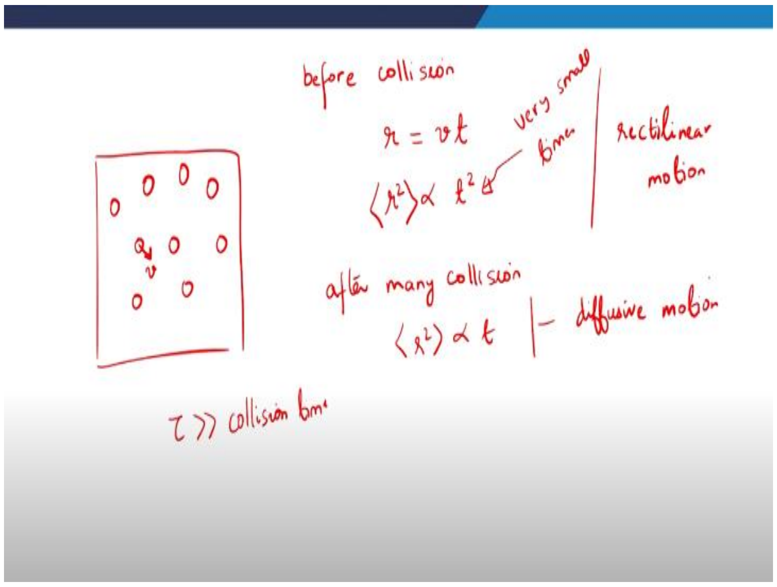
Now if you imagine the same situation in 2 dimensions, this is a random walk in 2 dimensions and that is actually representing some sort of a diffusion process or you can say this is how the molecules really move. The molecules of particularly the ideal gas you can see to move like that. So, every molecule is experiencing very frequent collisions and the collisions appear to be quite random because every other molecule can just come from anywhere with any particular momenta. So, the motions of the molecules are going to be quite random. So, I am doing it in two dimensions.

You can imagine this represents some sort of a surface diffusion but now after that we will also do the 3D case. So, what I am doing here essentially is I start with only one particle and I start moving it randomly. So, in one Monte Carlo step I make a movement that is of like a unit length. So, in this particular case I am not working in any kind of ideal units. I am working in some sort of dimensionless units. So, one is the step length and that is the unit of my system. So, I come here and from there again I take a random unit length step and I come somewhere else. Again I take a random step. I go somewhere else and I continue doing that. So, the step size has been fixed and this is in some sense representing the trajectory of a single molecule as it is moving in a gas that is what a diffusion process is.

So Einstein has shown that for diffusion processes there is a very beautiful relation called the Einstein relation that he developed before he got to relativity. He became famous for relativity but this was one of his first paper on the Brownian diffusion that explained the diffusion process that has been experimentally observed before him. So, what he said that the mean square displacement of a particle in the diffusion process will be proportional to time. In fact for a general walk in some D-dimensions it goes like 2d Dt where d is my dimension in this case it is 2 and D is my diffusion coefficient. This means that if I plot the mean square displacement, I will call it as MSD versus time, we expect that to be a line with a slope of that is proportional to the diffusion coefficient.

Now in the Monte Carlo simulation there is no time but you may imagine that the steps that I am making are of the unit length, that the steps is some measure of some sort of time step that I am making. It is not any kind of real time to be precise but that kind of represents something that will also happen in time because the particles are undergoing a random walk in reality as well. So, with that thing in mind that it is not very true dynamics, we are not doing molecular dynamic simulations. We can still say that if I do a random walk like using Monte Carlo scheme my mean square displacement is going to be proportional to the number of steps and the slope therefore when even if I replace t with n here that is going to give me some measure of the diffusion coefficient that is the idea that is always true whenever we have a diffusion process.

(Refer Slide Time: 33:55)



This is not completely true if we think about it because let us say we have a system containing molecule and for sake of argument let us say one of the molecules was moving with a velocity v. Now before it collides with any other particle it will keep on moving with that velocity v. So, before the collision the distance traveled will be actually something like v multiplied by t that will give me an r^2 that is proportional to t^2. So, we can say even on average my average r^2 will be going like t^2 as opposed to going like t that is what Einstein tells me. But if I am looking at time scales much larger than that I mean like within the time scale or time step that I am making or step I am making if many collisions have taken place then it is no longer true because if I think of like times τ much more than or even significantly more than the collision time then in that case in that τ period the molecule would have experienced many collisions and by virtue of which the velocity would have changed many times because of momentum transfers and this would have changed both its magnitude and its direction.

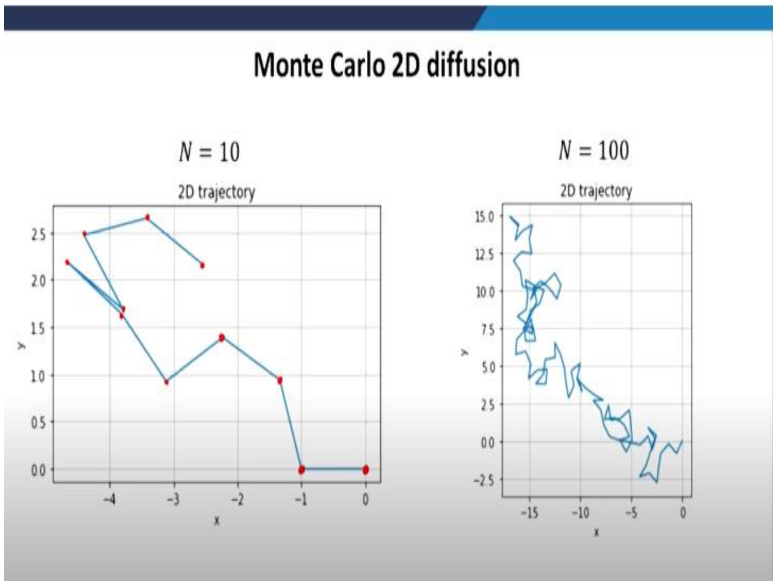
So, whatever we have said is actually true when we look in the limit of many collisions and only then we have r^2 proportional to t, it turns out that this regime is happening for very small times. The reason being that the particles are really close, the molecules are close. They will collide in very less time and therefore that regime is not extremely important. But it can be important in very specific cases. So, this is what is called a rectilinear motion and this is what is called a diffusive motion.

So, here I am trying to code that in here. So, let us go over this code query quickly. So, now the first two parts are the same as the earlier code. I am importing a plotting library, I am importing a numpy library.

Now we define the number of steps that is I will make ten thousand steps in this particular case I am showing and I again declare two areas to contain the coordinates after every step. And let us say if I start from the origin so in that case, I will put the initial value as 0, 0 to begin with. Now I go in the 'for' loop that pretty much runs this code num steps times there is the number of steps I am doing inside that I am generating first of all two numbers x and y between - 1 to + 1 and then I am dividing by square root of the x square plus y square of that, that will generate basically a step of length unity. So, let say, for example, if I am here, I can pretty much be anywhere on either side. So, I can go towards positive x, positive y, negative x, negative y, positive x negative y and whatever.

So, therefore what I am doing is I am generating that x, y in any particular direction. But I am dividing them by something like x square plus y square and if I do that then the step length that is the length of the x that is now normalized by this plus length of the y that is normalized at this. So, square of both of them added together, it is going to be something like that and that is clearly equal to 1 that means here that I am generating a step of length unity. But since I have generated x and y randomly that is going to be in any direction from the particular point. So, that is why I am dividing that by norm here and now I am appending not the x value I generate but the x value of the previous point plus this x value the y value of previous point plus this y value because I am making that step from the previous point to the new point. So, if you remember this is going to be a Markov process. So, the position at any point depends on the previous point. So, I simply add whatever step I have to the previous coordinates that we have. And now I go ahead and I want to plot the trajectory of the system. These are very standard plotting commands. I will do basically a plot of x and y coordinates as they are moving with time and what we do see is something like this.

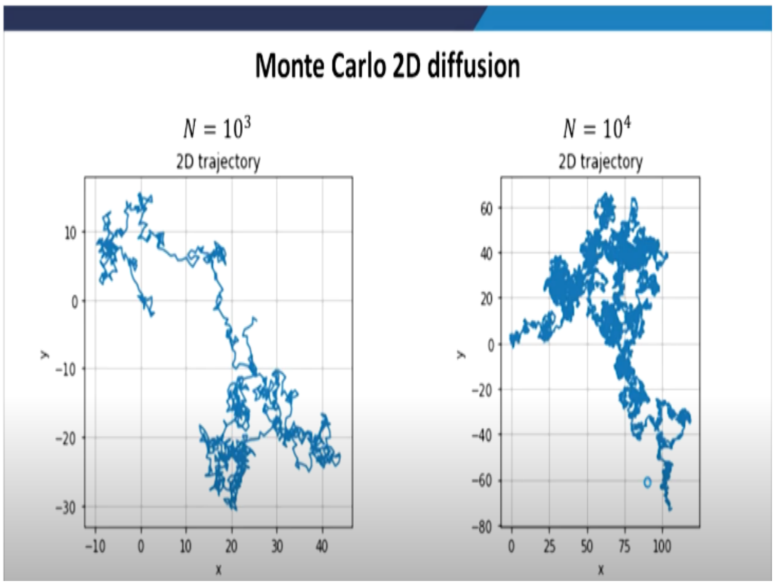
(Refer Slide Time: 39:58)



Monte Carlo 2D diffusion

So, I am starting from somewhere here that is my 0, 0 and I am making random steps and now you can see the steps every time are in very different directions. Again for 10 steps it may look like this. But if I run again it may look like something else. So, from anywhere I can go to pretty much in any direction and that is what is happening here. If I go to larger value of N trajectory may look something like this. Again, there is no directionality to what we are doing.

(Refer Slide Time: 40:32)



Monte Carlo 2D diffusion

And we can keep on thinking of even larger value of N and it is giving you a very interesting to look at trajectories of the diffusion process. So, with that point I want to stop here. In the next lecture we will discuss how to do the same in the three dimensions. And that is where I will also discuss how to implement the boundary conditions and then we do some more examples of Monte Carlo simulations.

So, with that I want to stop here. Thank you.