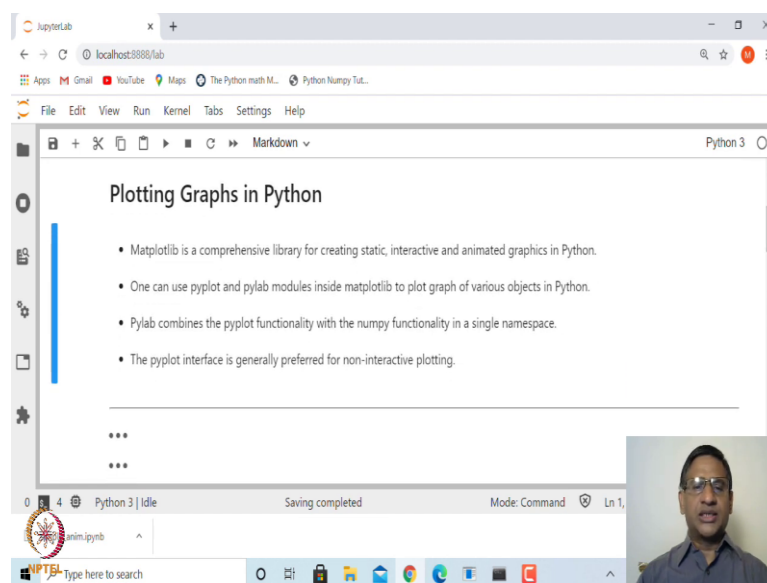


Computational Mathematics with SageMath
Prof. Ajit Kumar
Department of Mathematics
Institute of Chemical Technology, Mumbai

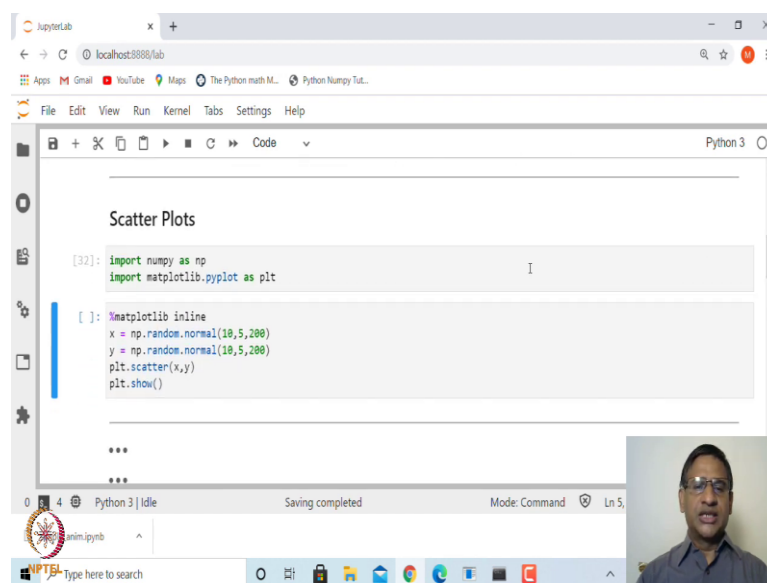
Lecture – 11
Python Graphics using Matplotlib

(Refer Slide Time: 00:34)



Welcome to the 10th lecture on Computational Mathematics with SageMath. In this lecture we shall look at plotting graphs in Python. We will be looking at briefly because exhaustive plot of two dimension and three dimension objects we shall look at in SageMath.

(Refer Slide Time: 01:27)



So, first of all, let us see how does one plot graphs in Python. Plotting graphs in Python is done through a module known as `matplotlib`, which is a comprehensive library for creating static interactive and animated graphics in Python. One can use `pyplot` and `pylab` modules inside `matplotlib` to plot graphs of various objects inside Python.

PyLab actually is a combination of pyplot functionality and NumPy functionality in a single namespace. We will be using pyplot mostly. So, let us look at first how to start plotting set of points in Python what we call as scattered plot. So, first of all let us import NumPy and also import pyplot lib from that pyplot as plt ok.

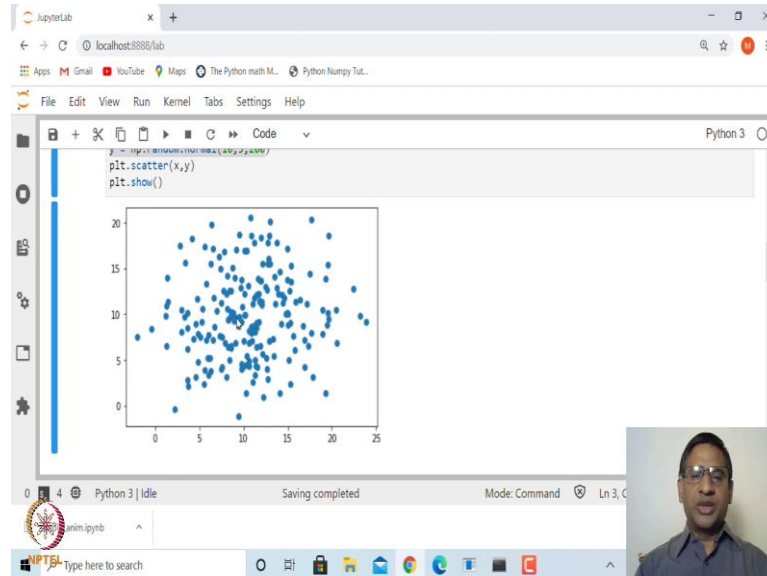
So, we are importing pyplot from matplotlib and we are naming it as plt. So, let us run this. Next, in order to plot set of points, let us generate random 200 points, which are normally distributed with mean 10 and standard deviation 5. So both x and y are 200 random points which are normally distributed with mean 10 and standard deviation 5.

And that can be obtained from random library inside NumPy and use normal function ok? Now, this percentage matplotlib space inline what it, what it does, does, is that it will plot the graphs inside this window itself.

Of course, this is more useful when you use the I-Python from command line, ok, because in that case plotting maybe outside your editor ok? So, after you have created x and y scattered 200 points then we shall use inside pyplot. There is a function called

scatter. So, simply say scatter x comma y and then ask plt to be shown, that is, plt dot show.

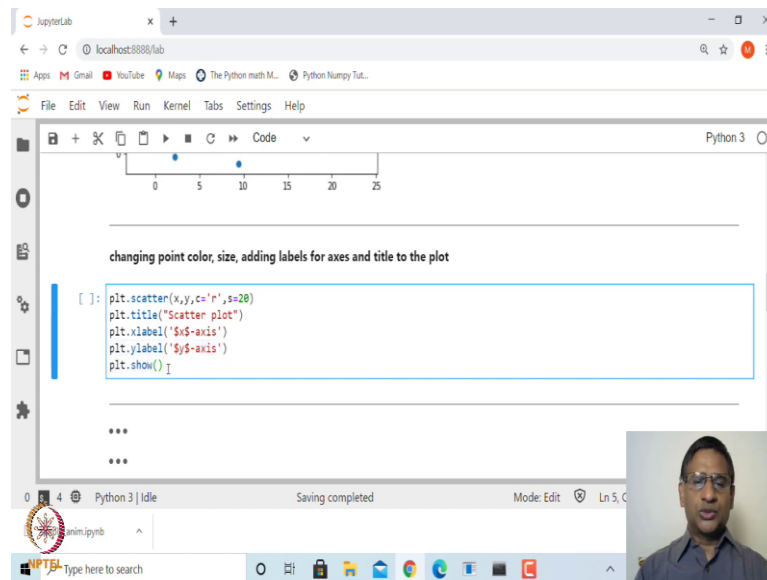
(Refer Slide Time: 03:32)



So, let me run this. When you run this, you get output which is 200 random points which are normally distributed with mean 10 and standard deviation 5. If you want to increase the number of points you can simply increase this 200 to 500 or 1000 and so on.

So, here what is it doing? It is taking set of point (x_i, y_i) , where x is coming from this array and y is coming from this array, right? Now, suppose you want to change various things of this plot, for example, you want to make this color, let us say red, you want to give title to x-axis, title to y-axis we can also give title to this graph.

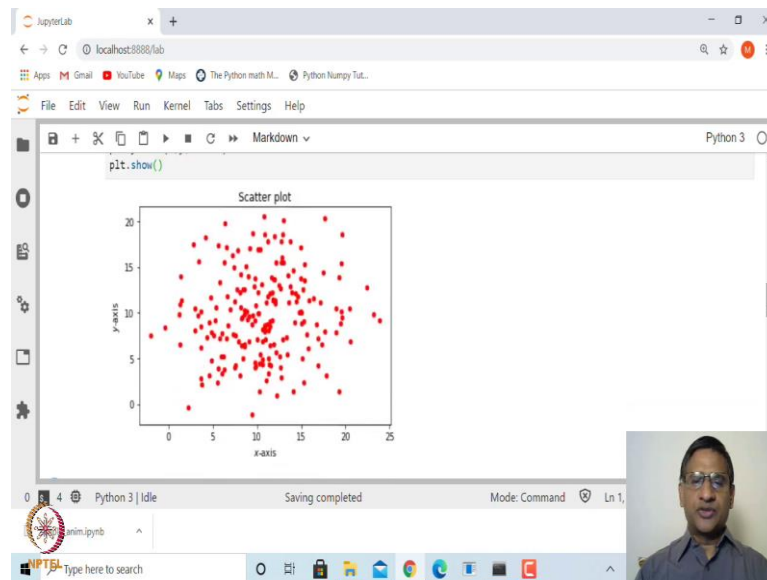
(Refer Slide Time: 04:30)



So, all these things can be done very easily. Let us see some of it. So, for example, if you want to change the color you can simply say `c` equal to single quote `r`. This is color, `c` for color or you could even simply say `r` that will also work and this `s` equal to 20 is the point size point size and then `plt dot title` will add title to the plot, `plt dot x label` as inside this single quote you write `x-axis`.

Here we have written inside dollar because then in that case it will convert `x` into italic, mathematical `x`. So, this is a latex command for typing `x`. Similarly, add label to `y-axis` and then ask it to show.

(Refer Slide Time: 05:24)

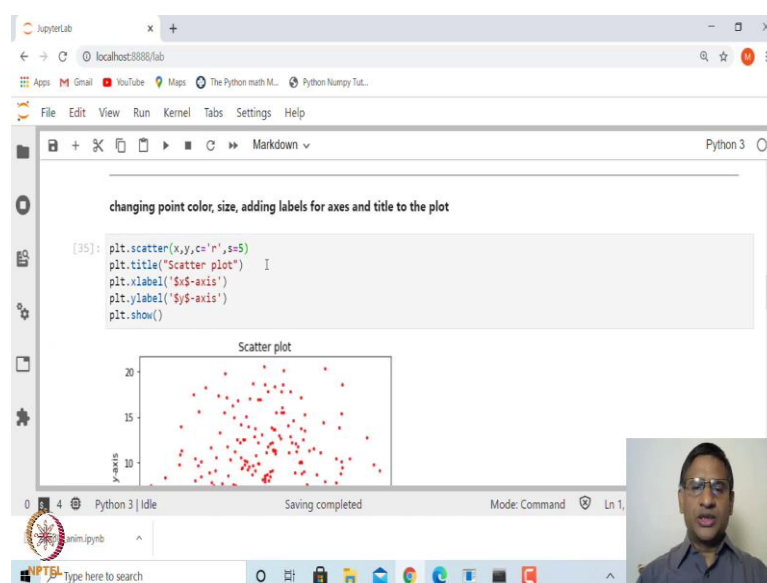


So, let me run this. This is what you get.

So, it has added scatter plot as a title; x-axis has been given label x-axis, y-axis is given label y-axis and you can see here this x is not usual x, it is somewhat italic. So, if I want to do various things to this graph we can simply add all these things using plt.

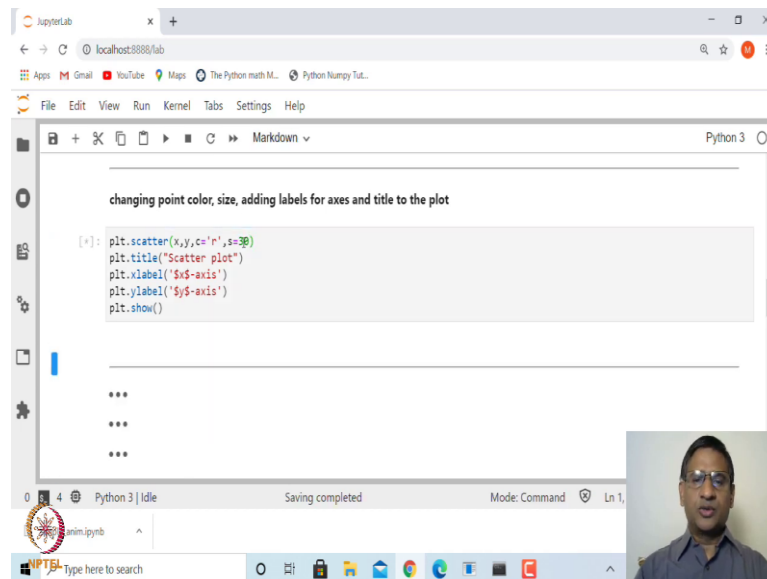
So, you can take help on plt dot scatter and you can see what are the other options available ok? Let me change this point size.

(Refer Slide Time: 06:06)

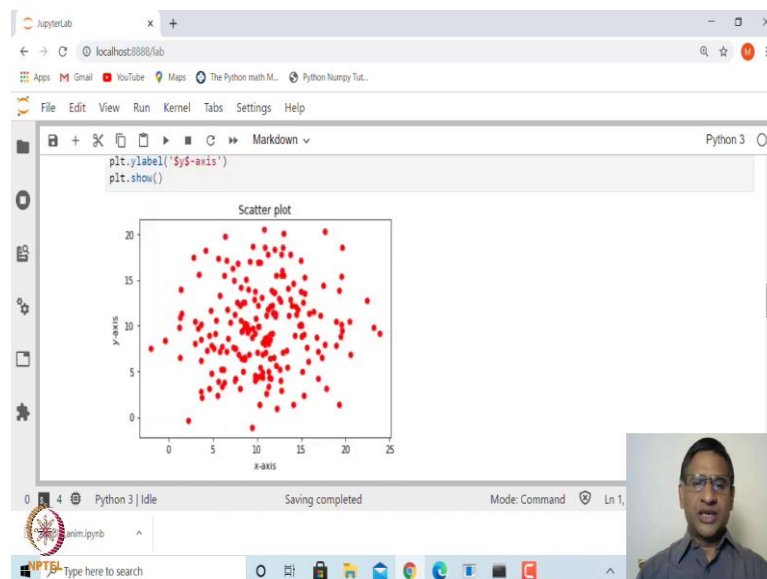


So, instead of 20 let us make it 5, then you will get smaller point size.

(Refer Slide Time: 06:11)



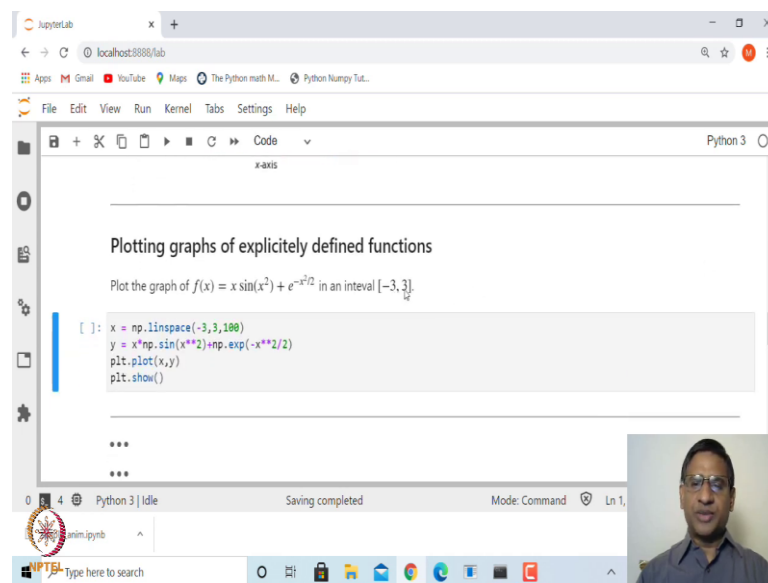
(Refer Slide Time: 06:13)



Or instead of 20, if I make it 30 you will get larger point size.

So, all these things can be done very easily. Next let us look at how to plot graph of a function f of x or graph of function which is explicitly defined in x .

(Refer Slide Time: 06:28)

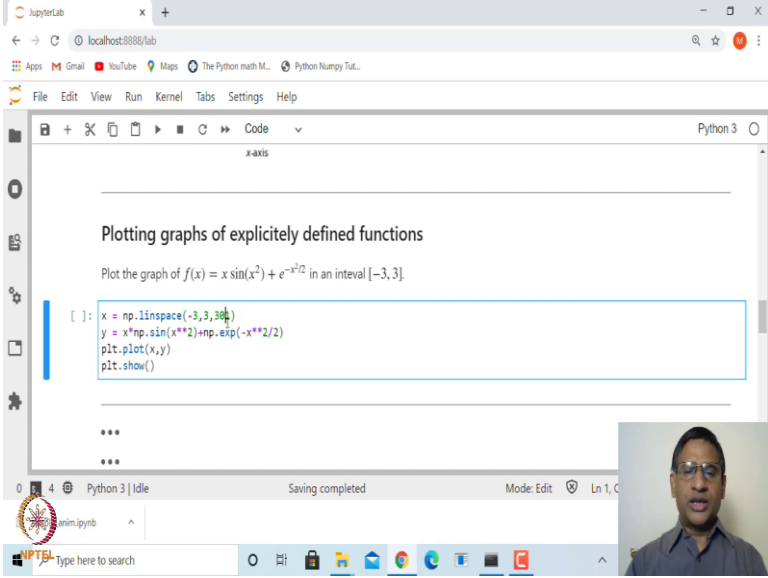


So, let us look at, we want to plot graph of a function x into $\sin x$ square plus exponential of minus x square by 2 in an interval from minus 3 to 3.

So, how does one plot graphs? So, first, you need to generate set of points in the interval minus 3 to 3, and then for each of this point, you evaluate what is the value of the function y which is equal to f of x . So, if the point is x_i then y_i will be f of x_i and then you plot x_i, y_i the set of points and then join them by freehand line drawing ok?

So, how do we do that? So, first of all, we need to generate the set of points from minus 3 to 3. So that we can do using `linspace` from NumPy library that we have seen already. So, from minus 3 to 3 let us generate 100 points. Let me make it more points. Suppose we want to let us say 300 points.

(Refer Slide Time: 07:32)



The screenshot shows a JupyterLab window with a browser address bar at localhost:8888/lab. The interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a toolbar. The main area contains a code cell with the following text:

Plotting graphs of explicitly defined functions

Plot the graph of $f(x) = x \sin(x^2) + e^{-x^2/2}$ in an interval $[-3, 3]$.

```
[ ]: x = np.linspace(-3,3,300)
y = x*np.sin(x**2)+np.exp(-x**2/2)
plt.plot(x,y)
plt.show()
```

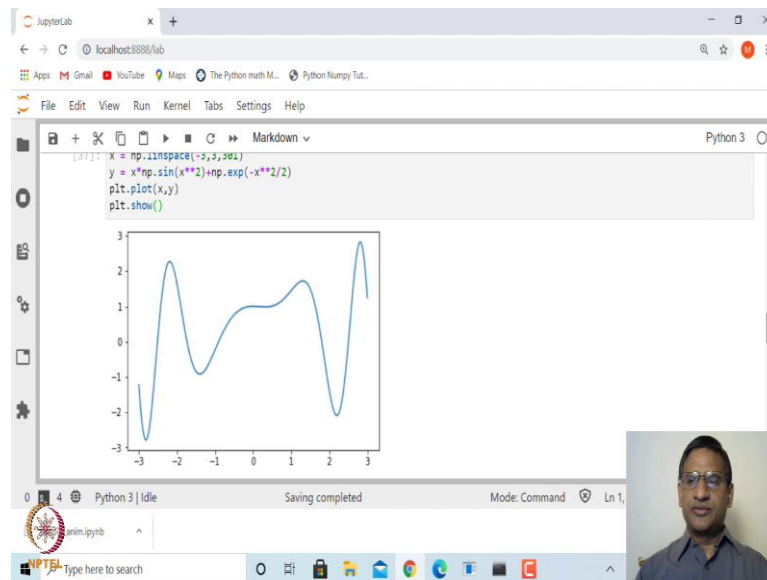
Below the code cell, there are three dots indicating the output area. At the bottom of the JupyterLab window, a status bar shows 'Python 3 | Idle', 'Saving completed', and 'Mode: Edit'. A small video feed of a person is visible in the bottom right corner of the JupyterLab window.

So, 300 points or let me make it 301; that means, it will divide this entire interval from minus 3 to 3 into 300 equal sub-intervals. Therefore, there will be 301 points between minus 3 and 3 both included.

Now, y is equal to x times since we want for each xi $f(x_i)$. Therefore, you, you should be using sin and exponential from NumPy because num sin cosine etcetera are exponential functions this operates on any NumPy array ok?

So, y is equal to f of x or is same as saying x into np dot sin x square plus np into np dot exp minus x square by 2. And then you simply say plt dot plot x comma y and then ask it to show it plt dot show ok?

(Refer Slide Time: 08:33)

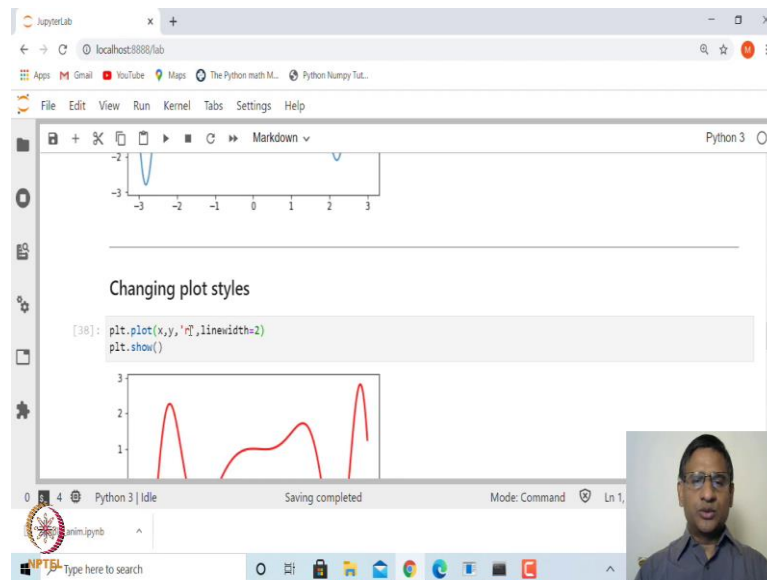


So, this is the graph of $f(x)$ is equal to $x \sin x^2 + e^{-x^2/2}$. This seems fairly complicated graph ok?

Next if you want to do any annotation to this graph, for example, if you want to change the color of this graph, you want to add x-axis label, y-axis label, or title, or you want to change the style of this plot. For example, we can plot in dotted lines, we can plot in dot dot dot lines, all these options are available.

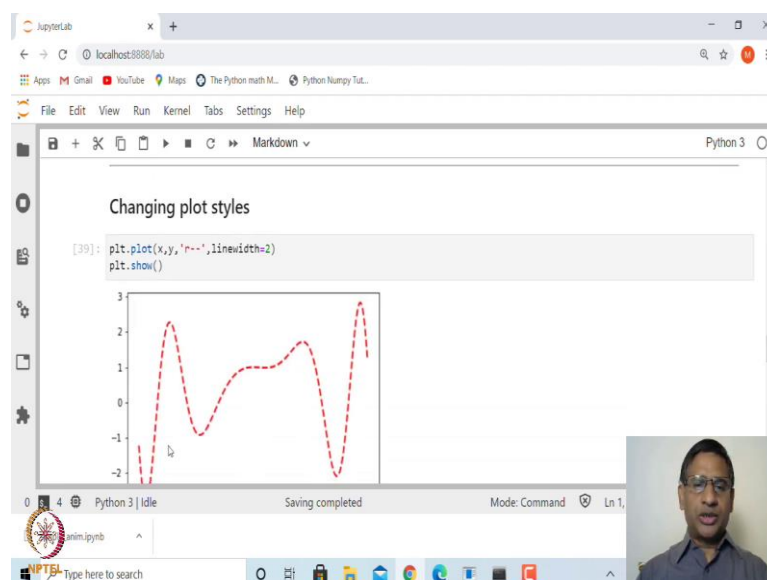
So, again you should look at help on plot from pyplot that is since we have imported pyplot as plt. You can simply look at help on plt dot plot and then go through this help manual and see what are the other options which are available in order to make changes to this graph; so, let us explore some of the options that plot provides.

(Refer Slide Time: 09:38)



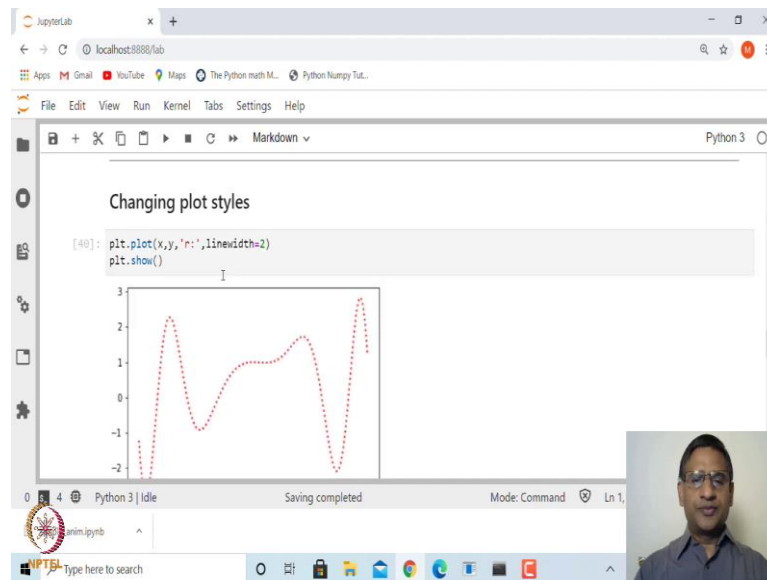
So, first, for example, changing plot style. So, if I want to change the color we have seen that we can simply say c is equal to red or simply say r. And then let us say we want to plot the change the line width. So, by default, the line width will be 1 and if I want to make it line width equal to 2, then you can see this line has become thicker and it has become red color.

(Refer Slide Time: 10:10)



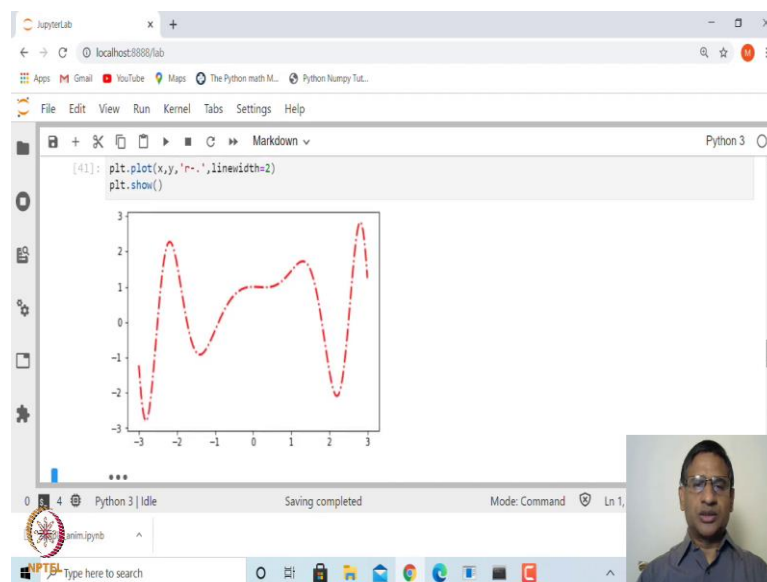
Suppose I want to make it red, but dotted lines, so, then you can simply say this, this will plot.

(Refer Slide Time: 10:29)



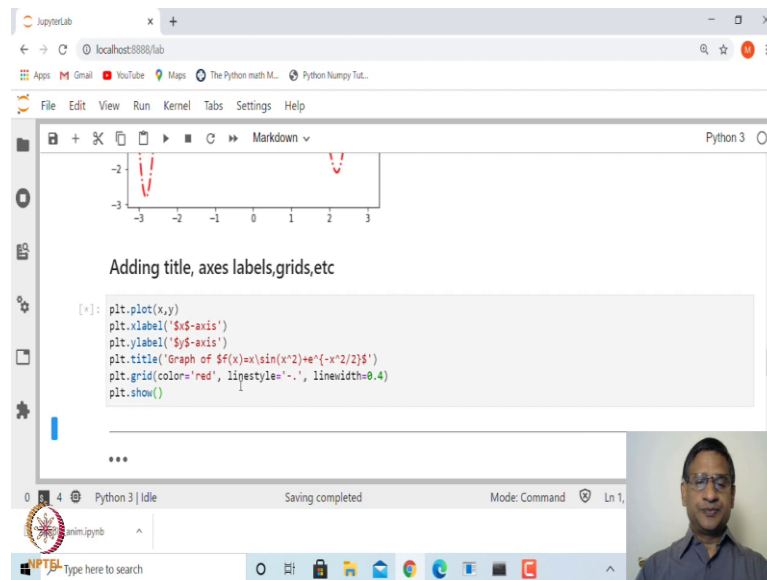
And similarly if you want, let us say, dotted line you simply say colon, then it will plot as dot dot dot. If you want to plot using plus or using some other symbols, all these there are several options available. So, you can just explore.

(Refer Slide Time: 10:45)



If I say hyphen, and then a dot, then it will plot hyphen followed by dot, hyphen followed by dot, ok, that is what you can see. So, many other things you can explore.

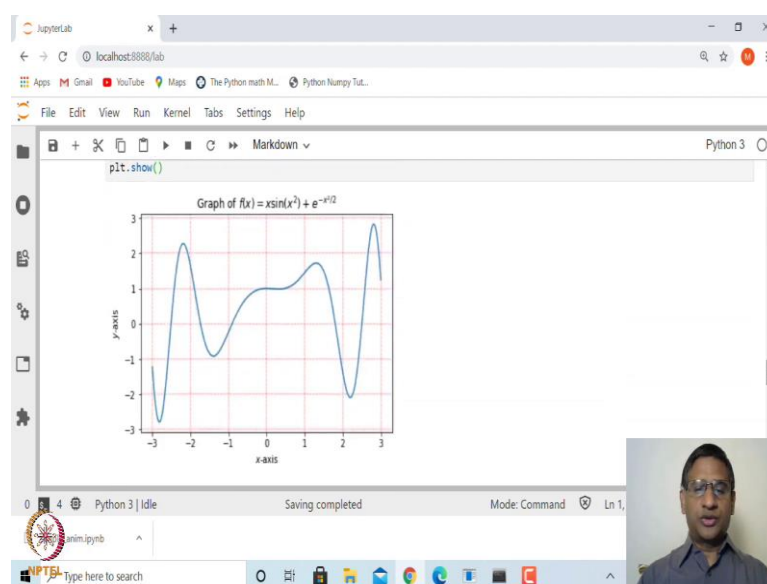
(Refer Slide Time: 10:59)



Similarly, if you want to add, let us say, axes label. Let x-axis, x-axis as a x-axis, y-axis is.

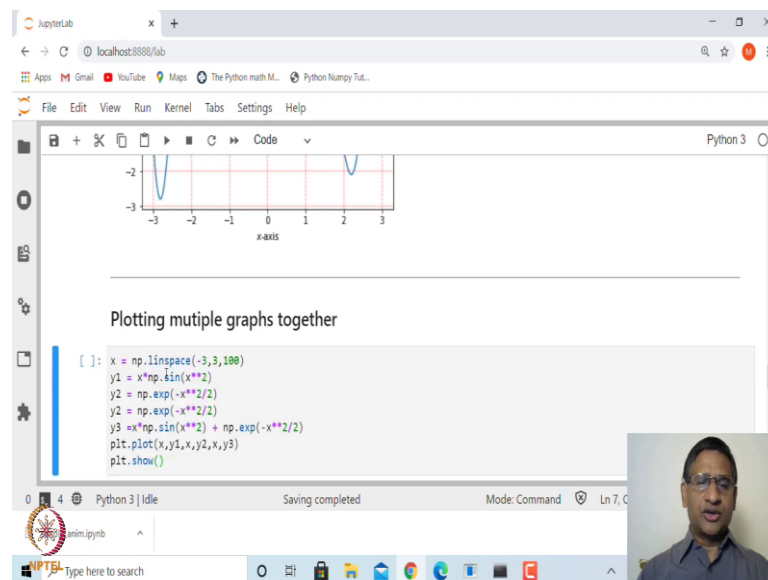
So, you can use x label, y label and then you can put a title to this graph and then you can also add grid lines using this function plt dot grid and you can mention what is the color of the grid, line style what should be the line style, what should be the line width. So, all these options are available, ok?

(Refer Slide Time: 11:28)



So, this is what you see. Now, you can see here x label, y label and this is the title of the graph and these are the grid lines. Again grid line also, there are several options.

You can make these grids finer, you can mention what are the points at which you want grid lines on x-axis and y-axis, all these options are available inside `pyplot` dot `plot` ok. (Refer Slide Time: 11:56)

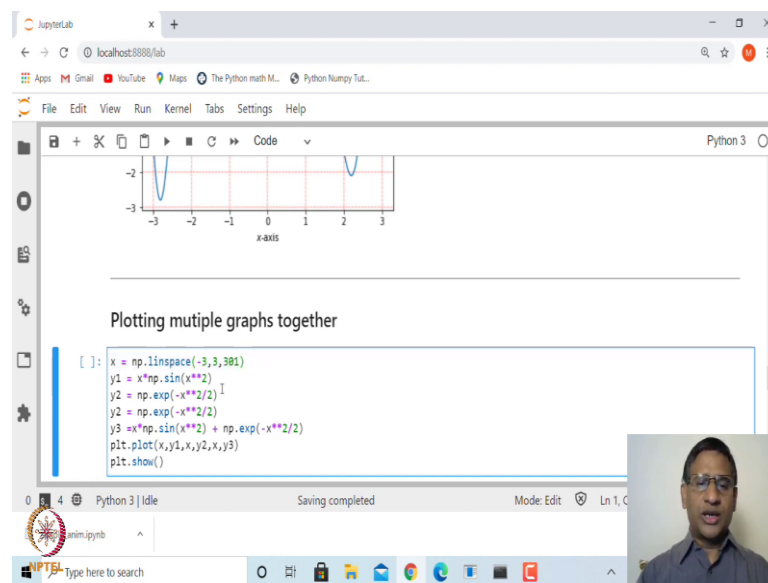


Next, suppose you want to plot multiple graphs together.

So, for example, here we have plotted graph of x into $\sin x$ square plus e to the power minus x cube. Now, suppose we wanted to plot this as a graph of x into $\sin x$ square and another graph as x into e to the power minus x square and then add these two together.

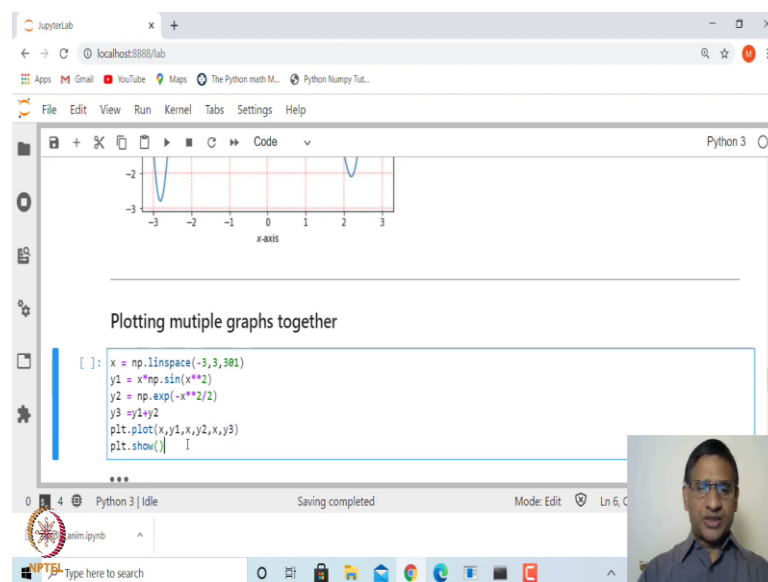
So that means, we want to plot three graphs together. One is for x into $\sin x$ square another one e to the power minus x square by 2 and the third one is f of x which is sum of these two. So, how do we do that? This is again fairly simple. So, we have already generated set of points x between minus 3 and 3, this we generated 301. So, let us make it 301.

(Refer Slide Time: 12:40)



And then let me call the first function as y_1 which is x into $\sin x$ square. y_2 as second function which is e to the power minus x square.

(Refer Slide Time: 12:59)

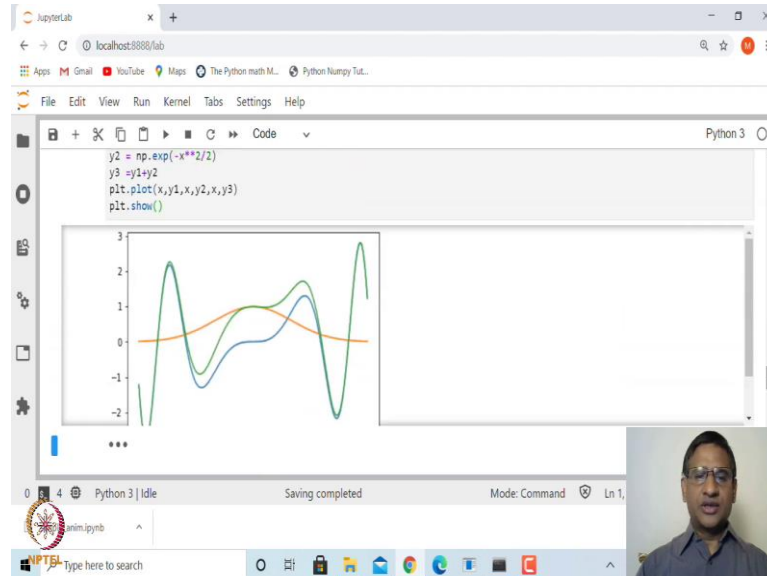


And the third function is, third function is, let us say, y_3 which is sum of these two functions. I can simply write y_1 plus y_2 .

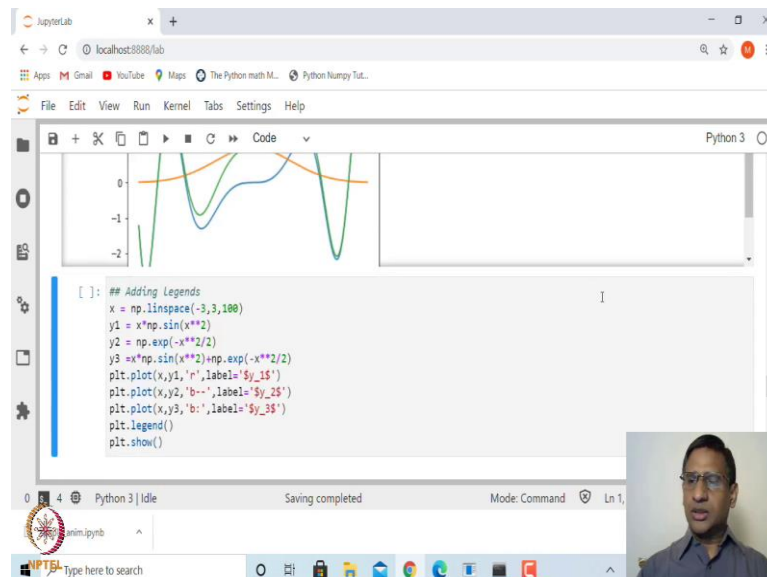
So, let me write that y_1 plus y_2 . So, y_1 and y_2 are two arrays for each x_i , y_1 is, y_1 is going to be set of points which is x_i into \sin of x_i square right? Now, when we want to plot this, all these things together you can simply again use `py plt dot plot` and you

mention x comma y1 that is the first graph, x comma y2 that is second graph and x comma y3 which is a third graph.

(Refer Slide Time: 13:42)



So, let us execute this. When we run this with this is what we get. This is the, the graph and you can see this by default it has put graphs in some default color. If you want you can mention the color here also ok? (Refer Slide Time: 13:59)



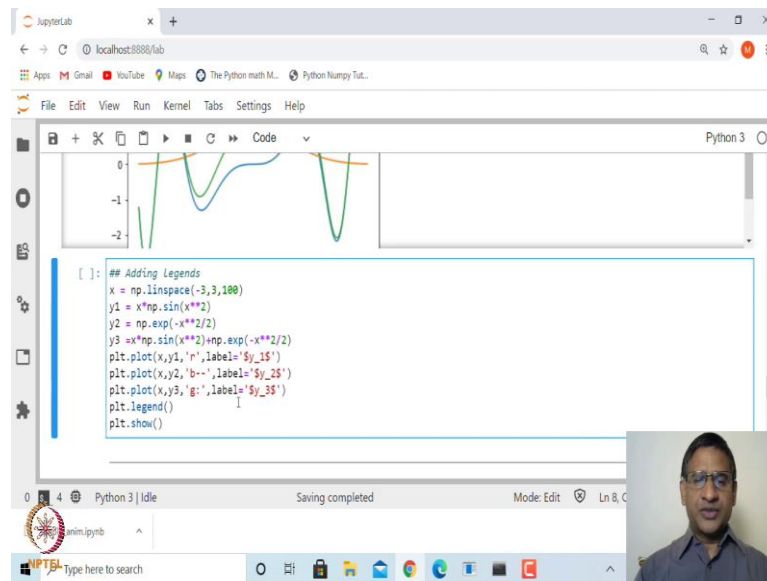
Suppose we want to let us say add.

So, for example, here it has plotted it does not know which graph is which right? So, what we can do is we can add legend. So, legend is basically it will add label to each of

this graph and then it will show which color graph is which graph of which function right?

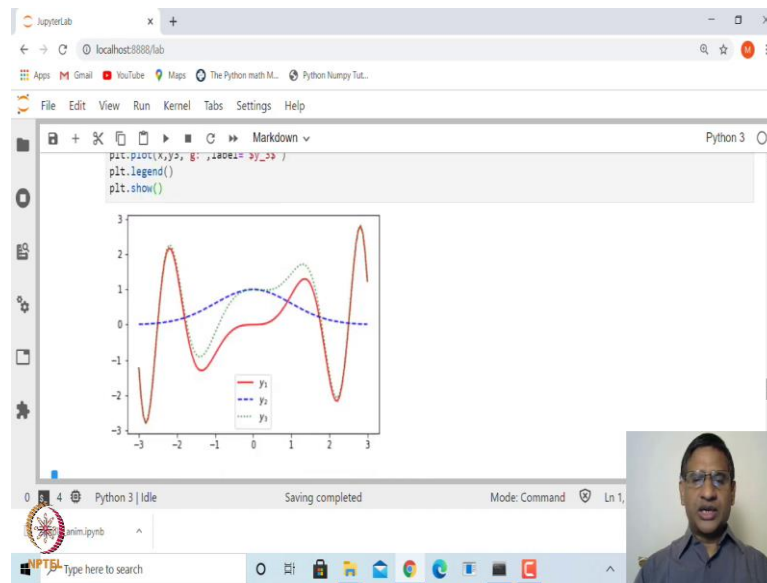
So, again let us generate y_1 , y_2 , y_3 in the same way as we have done before. First, let us plot a graph of y_1 in red color and give a label y_1 . Similarly plot graph of y_2 and give in blue color and with dash dash dash and then give this label as y_2 and similarly plot graph of y_3 and that is in let us, let us put this in green color and let us give the label y_3 .

(Refer Slide Time: 14:50)



And then you simply say `plt dot legend empty round brackets` and then ask it to show, it will plot the graphs.

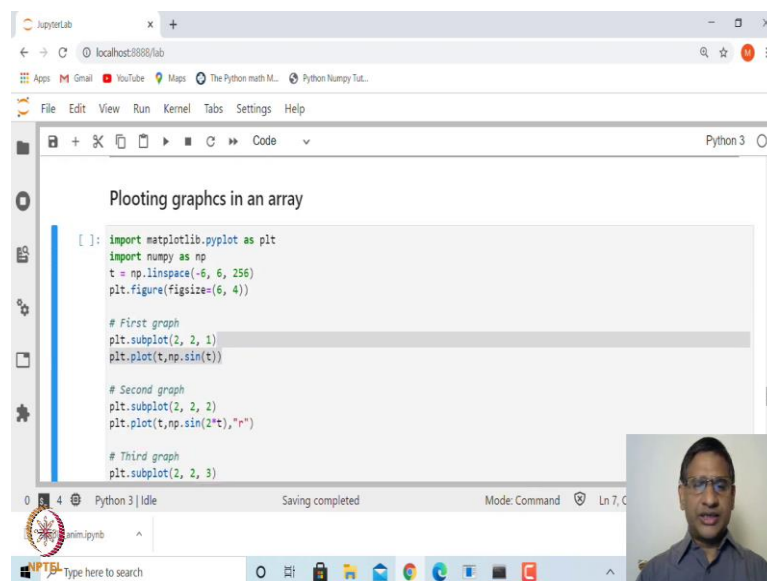
(Refer Slide Time: 15:03)



Now you can see here this blue graph is graph of y1 sorry blue graph is graph of y2, red is graph of y1 and dot dot dot this is dotted graph is a graph of y3.

And this this legend is plot is written in this place, but you have an option to put it wherever you want. So, that you can, you can look at help on plt dot legend and you will see how to place this legend location at various points right?

(Refer Slide Time: 15:41)

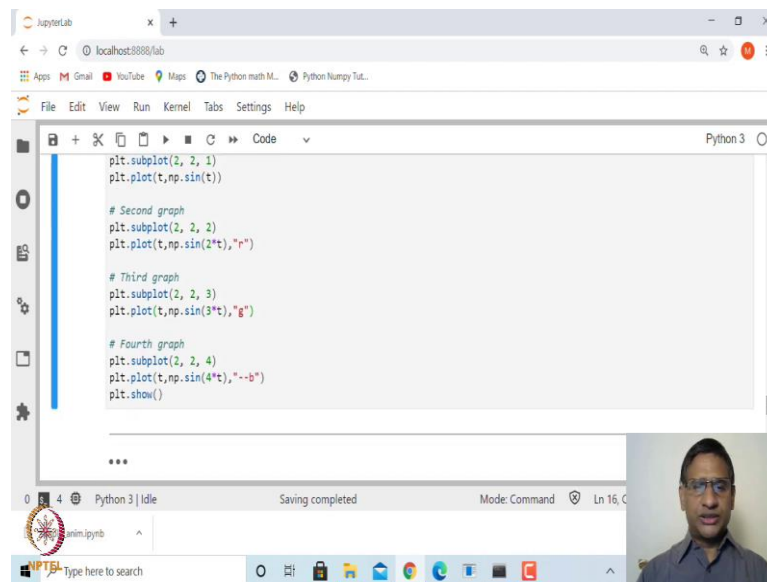


Next, let us look at, suppose you want to plot several graphs together and here we have plotted all of them in the same window, but suppose you want to plot these side by side.

Let us say we have four graphs and we want to plot first graph then on the right hand side second graph below the first graph as graph of third one and below the second graph, graph of fourth one. That is you want to plot this in array, 2 cross 2 array, or if you have 6 graphs you can plot in 3 cross 2 array or 2 cross 3 array.

So, this can also be done. So, how do we do that?

(Refer Slide Time: 16:28)



```
plt.subplot(2, 2, 1)
plt.plot(t, np.sin(t))

# Second graph
plt.subplot(2, 2, 2)
plt.plot(t, np.sin(2*t), "r")

# Third graph
plt.subplot(2, 2, 3)
plt.plot(t, np.sin(3*t), "g")

# Fourth graph
plt.subplot(2, 2, 4)
plt.plot(t, np.sin(4*t), "--b")
plt.show()
```

So, that is achieved by what is known as subplot, subplot. So, first, let us look at suppose we want to plot graph of $\sin x$ and then $\sin 2x$ and then $\sin 3x$ and $\sin 4x$ in a 2 cross 2 array, 2 by 2 array. So, first again let us import pyplot and NumPy.

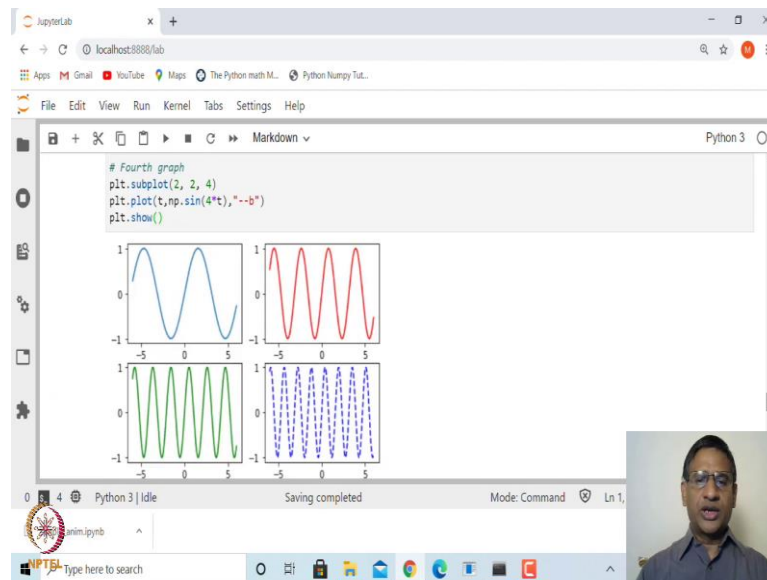
Though we have already done, so, there is, there is no point in importing it again, but in case you are doing it afresh, you should always import these two. And then t , let us say, t is a set of points from minus 6 to 6 and there are 256 points. If you want you can make it more and then create a figure of size 6 by 4, figure of size 6 by 4.

You can increase this size, you can make it bigger, you can make it smaller, and then first graph is first graph, that is spelling mistake here. So, first graph is graph of $\sin t$. So, again; so and that how do we add? So, in first place that is what at (1,1) place you are saying this is 2 cross 2 and then this is the first graph.

So, it has created 2 by 2 array, and that it is adding first inside first let us say, box. So, it will create a box of, of 2 by 2, 4 boxes of 2 cross 2. The first box will have plot of \sin function and then next we will say, in second graph that is again this here say sub-plot 2 comma, 2 comma 2, that is the second graph.

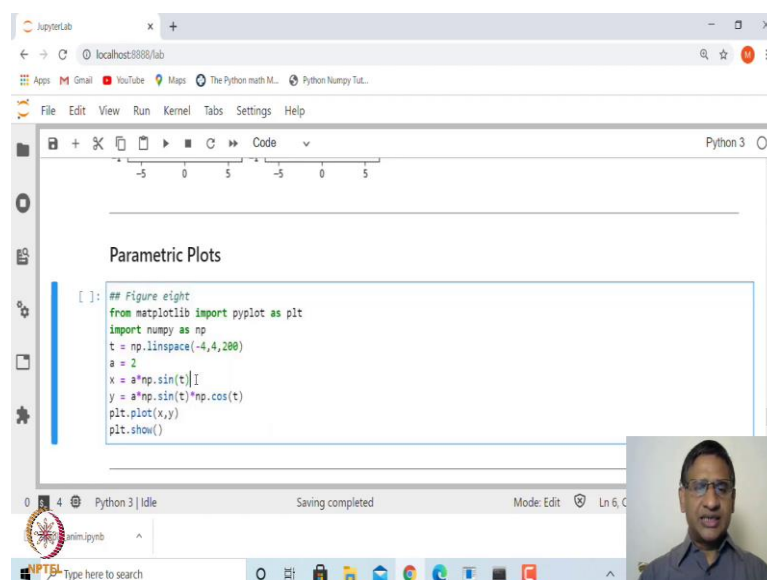
And this is 2 by 2, and this is graph of \sin of $2t$ in red color and this is the third graph which is a `plt dot sub-plot 2 comma 2 comma 3`, that is the third graph and so on. And this is the $\sin 3t$ which is in green color and the next one is the fourth graph which is `plt subplot 2 comma 2 comma 4` and this is $\sin 4t$ and this is plotted in blue color and with dash dash, ok?

(Refer Slide Time: 18:41)



So, let us plot this. So, when you plot this, again you can see here 4 graphs are plotted together. This is $\sin x$, this is $\sin^2 x$, $2x$, this is $\sin 3x$ and this is $\sin 4x$. You can have many more graphs together. In fact, it also you can plot, plot inside a plot. Suppose this is a big plot, inside this, I want a small plot that options are also available. So, you can go through some of these options ok?

(Refer Slide Time: 19:10)

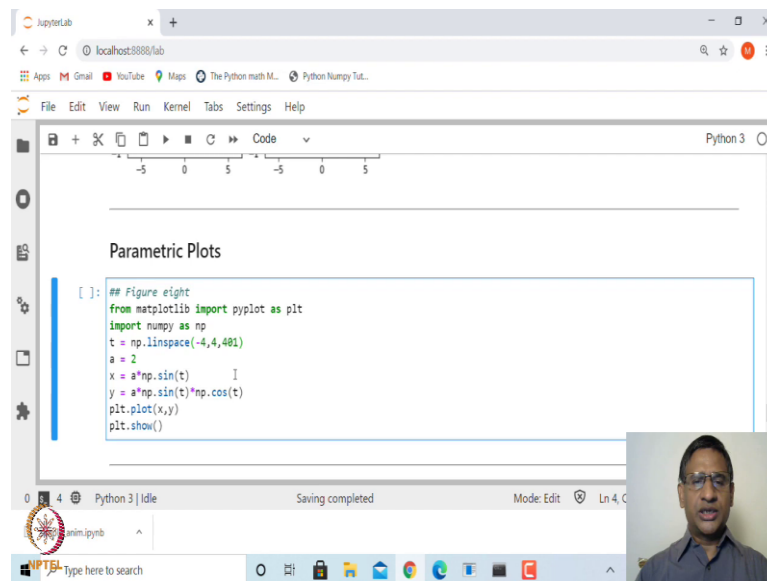


Next, let us look at how to plot graph of a curve which are, which is defined by parameter parametric course. So, x-coordinate is some, defined as f of t , y-coordinate is

defined as g of t , and then you want to plot x comma, (x,y) from let us say t in some interval a to b .

So, let us look at we want to plot a graph of a function whose x -coordinate is given by $\sin t$ and y -coordinate is given by $\sin t$ into $\cos t$ ok? So, and we want to plot this graph for t from minus 4 to 4 and we are dividing this into 200 points.

(Refer Slide Time: 19:55)



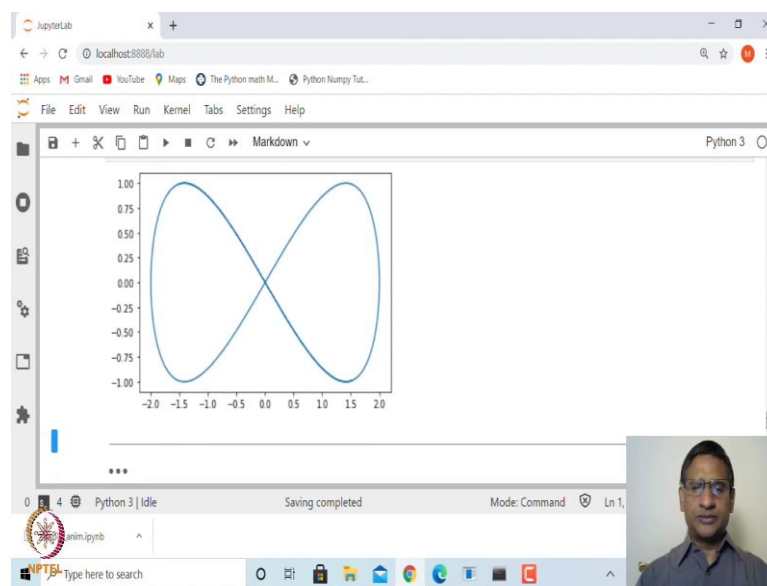
```
Parametric Plots

[ ]: ## Figure eight
from matplotlib import pyplot as plt
import numpy as np
t = np.linspace(-4,4,401)
a = 2
x = a*np.sin(t)
y = a*np.sin(t)*np.cos(t)
plt.plot(x,y)
plt.show()
```

So, if you want, you can make it, let us say, 400 points. Let me make it 401. So, there are 400. This minus 4 to 4, it is divided into 400 equal intervals and here a is taken as 2; so, a into $\sin t$. So, $2 \sin t$. x-coordinate is $2 \sin t$, y-coordinate is $2 \sin t$ into $\cos t$ which is actually $\sin 2t$.

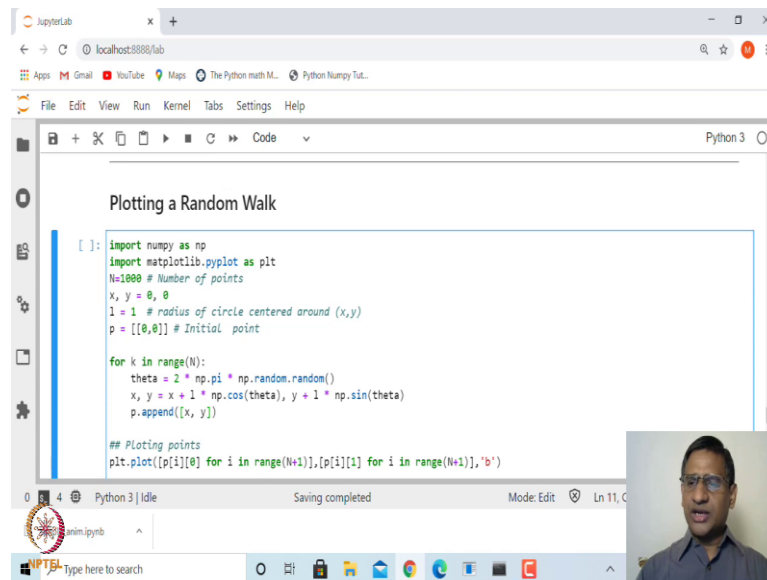
So, you can simply say $\sin 2t$ also and then again simply say `plt dot plot` and then `plt dot show`. So, it, earlier also when we have plotted graph of f of (x,y) , actually we did generate x and y -coordinates and then we plotted using `plot` function inside `plt` right.

(Refer Slide Time: 20:35)



So, this is what you get. This is a graph of a parametric curve whose x-coordinate is $2 \sin t$ and y-coordinate is $2 \sin t \cos t$ and this is, this graph is called figure 8 graph. So, that is how. So, this is a parametric plot ok?

(Refer Slide Time: 20:57)



```
Plotting a Random Walk

[ ]: import numpy as np
import matplotlib.pyplot as plt
N=1000 # Number of points
x, y = 0, 0
l = 1 # radius of circle centered around (x,y)
p = [[0,0]] # Initial point

for k in range(N):
    theta = 2 * np.pi * np.random.random()
    x, y = x + l * np.cos(theta), y + l * np.sin(theta)
    p.append([x, y])

## Plotting points
plt.plot([p[i][0] for i in range(N+1)], [p[i][1] for i in range(N+1)], 'b')
```

Next, let us look at, suppose we want to plot, let us say, random walk. So, in this case, what do we do? Suppose we start with, from 0,0 and take a random walk and let us say of unit length. So, and then, random walk would be in any direction and then generate the next point.

Next point again in, take random walk from the first point which is, from first point which you have generated; again inside unit circle, inside the unit circle, that is of unit length and keep on doing this and then let us try to plot all these points and see how it looks like.

So, again, how do we do that? This is fairly simple. So, what we are doing is, we are taking, let us say 1000 points. So, again, first of all you, you import NumPy and pyplot and then the 1000 points and starting the coordinate of is 0,0; x comma y, comma, and we are this is the, the radius or the step length by which you are taking a walk.

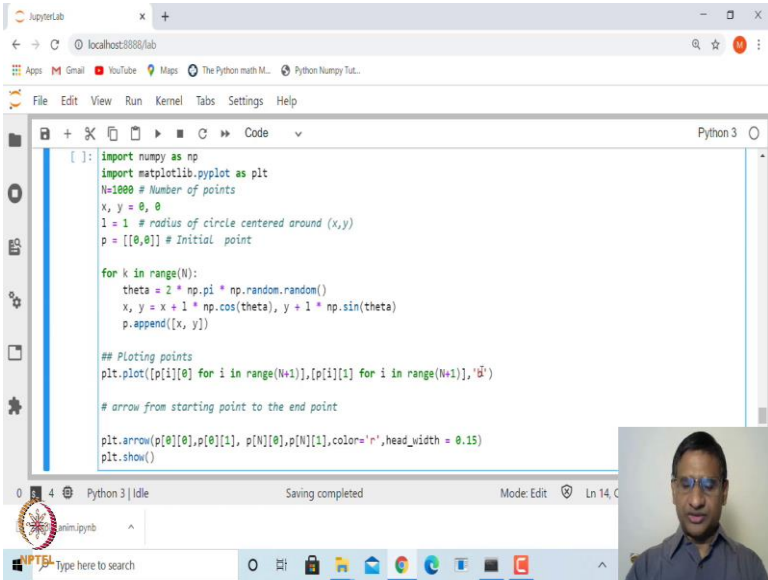
So, this is unit step length and, but it will be in any direction. So, we will point, generate a point on a circle of unit length. And then this is the, we want to store all these points generated 1000 points generated in a list.

So, we have initialized this list as origin which is 0 comma 0 and then we will keep on adding these points inside p. And how do we do that? So, first, generate a random angle, right? So, this could be between anything between 0 to 2 pi.

So, generate a, a random number between 0 to 1 and multiply this by 2 pi. So, it will go from 0 to 2, 2 pi, any point between the, any angle will be from 0 to 2 pi; and this will be random and then generate two points x and y.

So, x point will be you add x plus the r cos theta; r here is 1 and y plus r cos theta, y-coordinate is y plus r sin theta. So, you know that any point on a circle of radius r is given by, centered at origin, is given by (r cos theta, r sin theta), but if it is centered at, let us say (x,y), then the x-coordinate will be x plus r cos theta, y-coordinate will be y plus r sin theta. And then you append this to p; p dot append x comma y.

(Refer Slide Time: 23:34)



```
[ ]: import numpy as np
import matplotlib.pyplot as plt
N=1000 # Number of points
x, y = 0, 0
l = 1 # radius of circle centered around (x,y)
p = [[0,0]] # Initial point

for k in range(N):
    theta = 2 * np.pi * np.random.random()
    x, y = x + l * np.cos(theta), y + l * np.sin(theta)
    p.append([x, y])

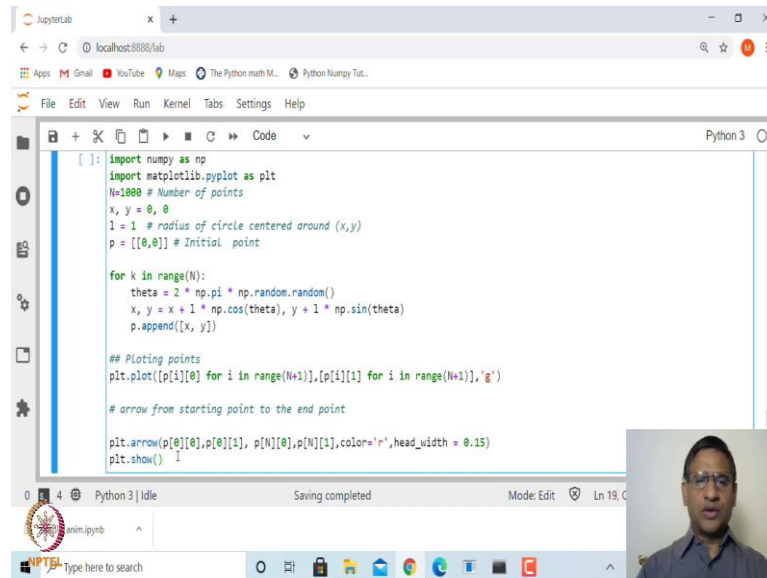
## Plotting points
plt.plot([p[i][0] for i in range(N+1)], [p[i][1] for i in range(N+1)], 'b')

# arrow from starting point to the end point
plt.arrow(p[0][0], p[0][1], p[N][0], p[N][1], color='r', head_width = 0.15)
plt.show()
```

So, we have generated these 1000 random points, actually it will be 1001, we are starting with origin, and then we want to plot these points. So, how do we plot these points? So, when you are plotting these points, we have to plot a, actually a line starting from 0 to first point (x1,y1), then from (x1,y1) to (x2,y2). From (x2,y2) to (x3,y3), and so on, and this can be again done by using py dot plot.

So, we are taking the, this is the x-coordinate of the first point and then join with the. So, the, these are the set of first coordinates of all the points. These are second coordinate of all these points, and then put in, the points are in blue in color. This, this, this will be, will be, actually it will be plotting a line, and blue in color.

(Refer Slide Time: 24:29)



```
[ ]: import numpy as np
import matplotlib.pyplot as plt
N=1000 # Number of points
x, y = 0, 0
l = 1 # radius of circle centered around (x,y)
p = [[0,0]] # Initial point

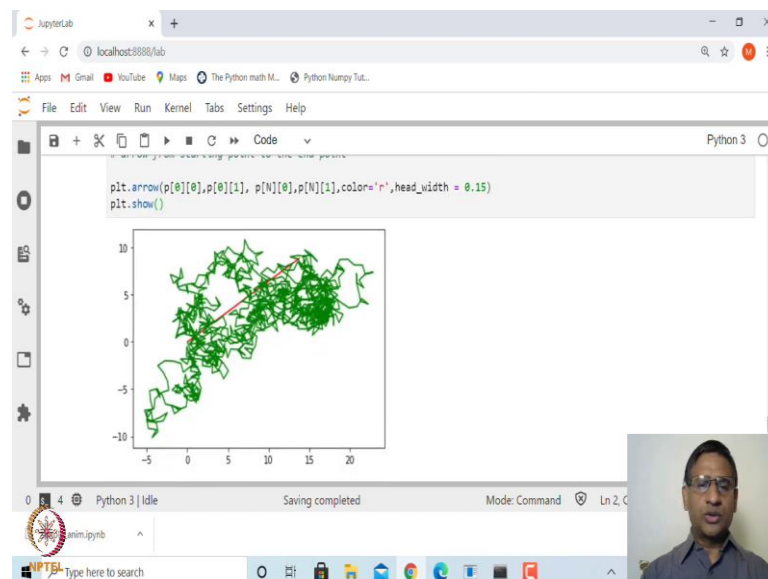
for k in range(N):
    theta = 2 * np.pi * np.random.random()
    x, y = x + l * np.cos(theta), y + l * np.sin(theta)
    p.append([x, y])

## Plotting points
plt.plot([p[i][0] for i in range(N+1)], [p[i][1] for i in range(N+1)], 'g')

# arrow from starting point to the end point
plt.arrow(p[0][0], p[0][1], p[N][0], p[N][1], color='r', head_width = 0.15)
plt.show()
```

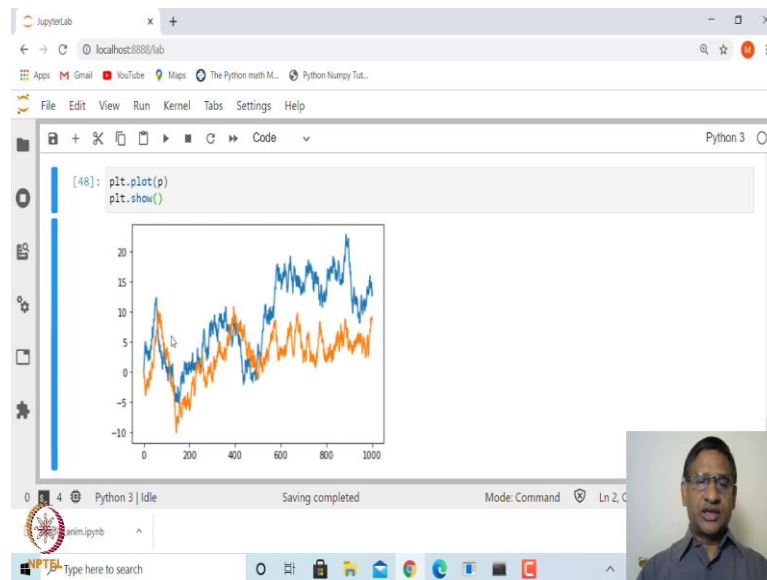
So, let me put that is fine. This is, or let me put green in color. And then you plot the arrow from starting point, that is origin, to the last point, that is the (pN0, pN1), ok, that is the arrow. So, a plt inside the plt inside pyplot you also have a function called arrow to plot arrow.

(Refer Slide Time: 24:53)



Now let me run this, ok? So, this is what you get. So, we had started with origin here. Then it takes 1000 steps, random steps, and this is where it reaches actually, right? So, this is we are just making use of all these things in this random walk, right?

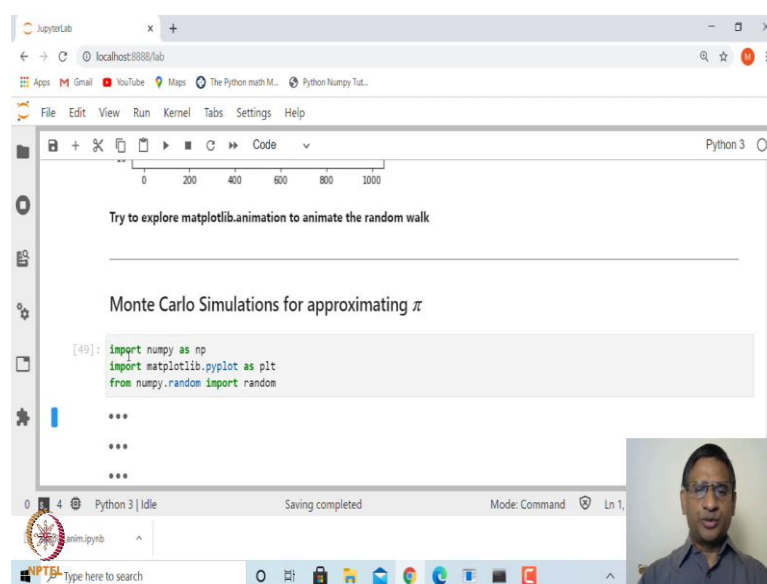
(Refer Slide Time: 25:14)



You can of course, suppose if you have generated only set of all the points; if you plot plotted just a set of all the points in, in this p, the graph will look like this. So, here 1,1 is for x-coordinate.

This is x-coordinate, how it varies, and the yellow the, the orange one is the other one. So, one will be x-coordinate, other one will be y-coordinate. So, this is some kind of time series plot actually, ok?

(Refer Slide Time: 25:48)

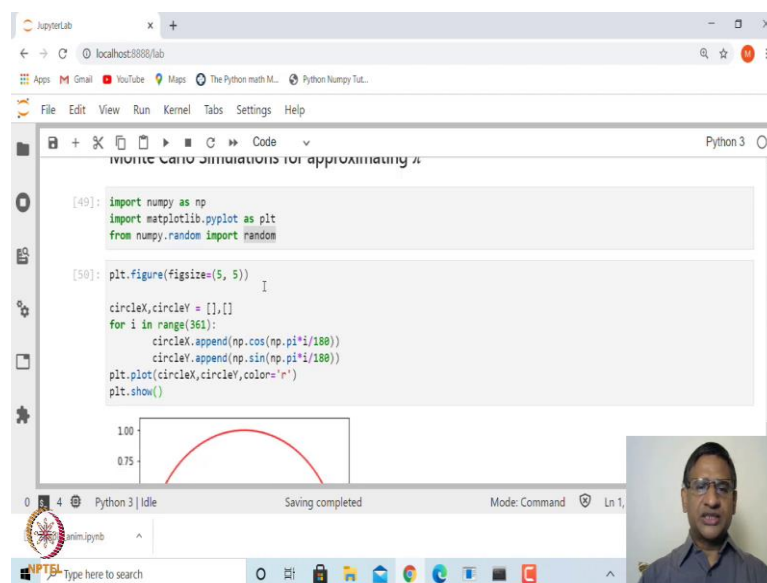


Now you can try to explore, there is a, also an option to do animation.

So, if you want, you can even animate this random walk from matplotlib; there is a function, there is a library, or module called animation. So, you can look at that and then see how we can animate this random walk.

Lastly, let us look at one more graph. We have already seen how to approximate pi using Monte Carlo simulation. Now, there we, we simply generated the points, and we counted how many points are there inside unit circle and how many are inside, outside this unit circle, inside unit square.

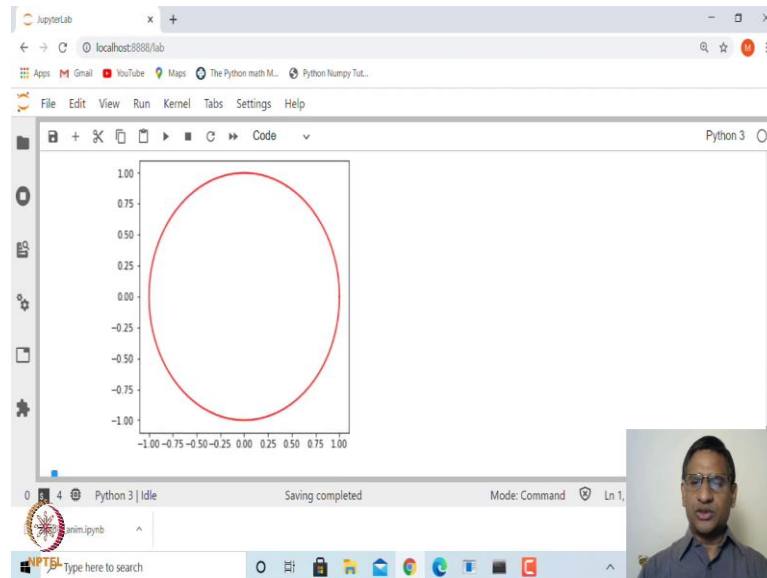
Of course, we, we worked with only in first quadrant, but we can also work inside the all the coordinates, got it? So, what we will do? We will generate a; let me do it step by step. So, first let us import all these things which are necessary, that is NumPy, then random function from NumPy random, and then pyplot from matplotlib. (Refer Slide Time: 27:02)



And then first let us generate a square. So, square or first let us generate a circle. So, circle, we want to generate a circle. So, of course, there is a circle function inside the, the pyplot, but here we have just generated a set of points about 361 points and then all these points are equally placed.

And then add this inside x-coordinate and y-coordinate of the circle and then plot this x-coordinate and y-coordinate of this circle and let me see.

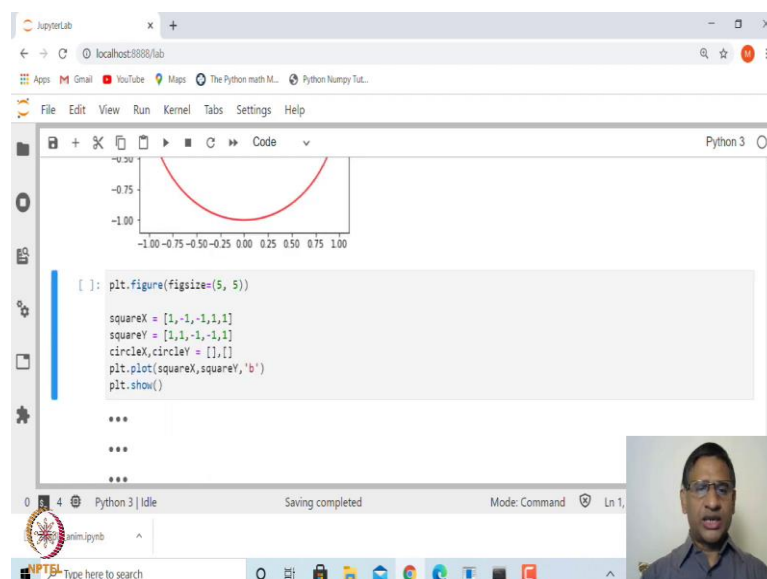
(Refer Slide Time: 27:41).



So, this is a unit circle. Of course, here we have mentioned the figure size to be 5 by 5 that is why you see here, a unit circle.

In case your figure size is 5 cross 6, you may look, you may see this as not as a circle, but as an ellipse because that is the aspect ratio of the, the axes, ok? Next, we can generate similarly a unit square.

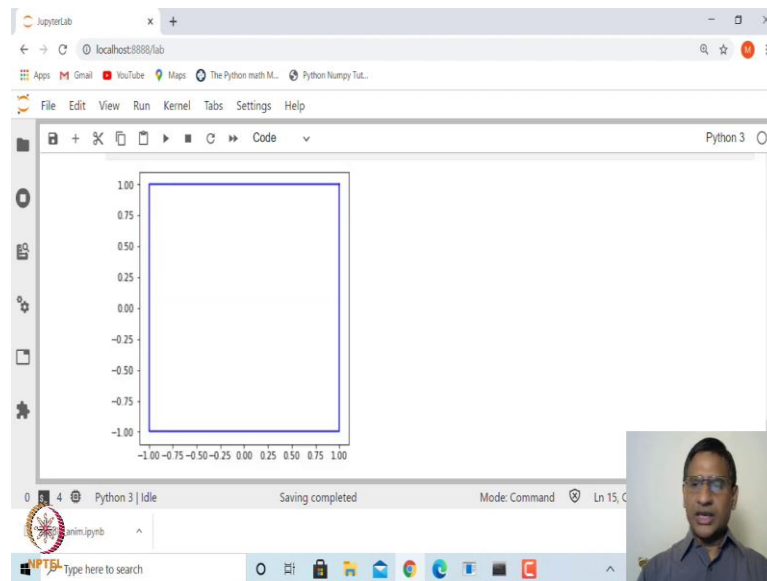
(Refer Slide Time: 28:05)



So, we are writing the first x-coordinate of this unit square and y, y-coordinate of this unit square.

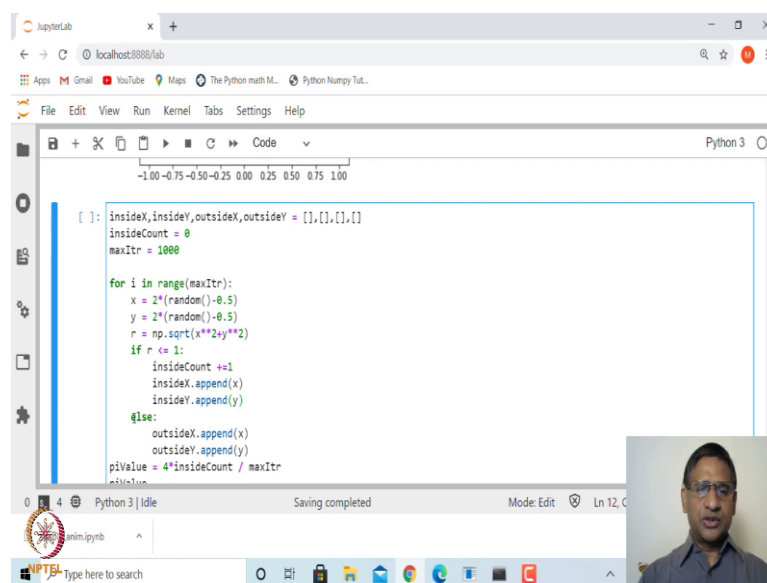
So, of course, it is (1, minus 1, minus 1, 1, 1), because you have to start with one point, and get back to that, that point. So, it is starting at 1 comma 1 ok?

(Refer Slide Time: 28:27)



So, let us plot this. This is blue in color. So, this is a unit scale. Now, inside this we will generate all these points. So, how do we generate these points?

(Refer Slide Time: 28:33)



So, we want to plot a circle inside this unit square and then generate some, let us say 1000 random points between minus 1 and 1. x-coordinate and y-coordinate lying between minus 1 and 1, and then count which are the points which are inside the circle, which are the points which are outside the circle. So, the point (x_i, y_i) will be inside the circle, if $x_i^2 + y_i^2$ is less than or equal to 1, otherwise, it will be outside the circle.

So, in case it is inside the circle, you increase count by 1, so, right. So, what we are doing? We are saying what are the x-coordinate of inside points, y-coordinate of inside points, and then outside points x-coordinate, outside point y-coordinates.

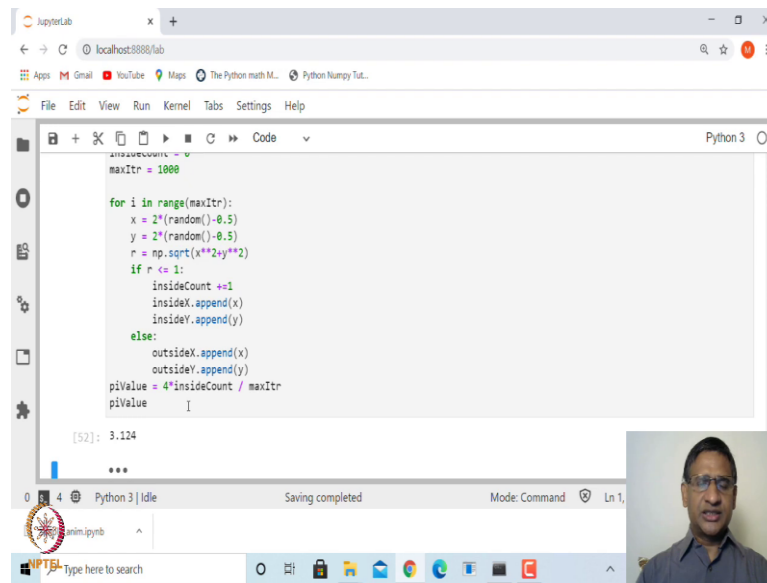
So, to begin with, they are all empty list, and then inside count, count how many are inside? 0 and then maximum number of points you want to generate is 1000. And then so what we are doing? For each i from 1 to 1000 then you generate x-coordinate.

So, this is random, it will generate a random number between 0 and 1 and then subtract 0.5 from this. So, this will lie between minus 0.5 to 0.5 and then multiply by 2. So, that this, this will lie between minus 1 and 1, similarly y-coordinate. And r is the, the, calculate r as the distance from the origin of x_i, y_i , (x_i, y_i) that is $x_i^2 + y_i^2$ square root.

And in case r is less than or equal to 1, then it lies inside then you increment the inside count by 1 and append the inside x and inside y as x and y . Otherwise, you append to the outside x and y -coordinates, ok, and once you have done this, and then let us count how many are inside and how many are outside.

So, the ratio of inside count divided by the total number of points and multiply by 4, that is approximately the value of π , that is quite easy. We looked at this in, in the last class also, in the previous class, ok?

(Refer Slide Time: 30:50)



```
from random import random
from math import sqrt
insideCount = 0
maxItr = 1000

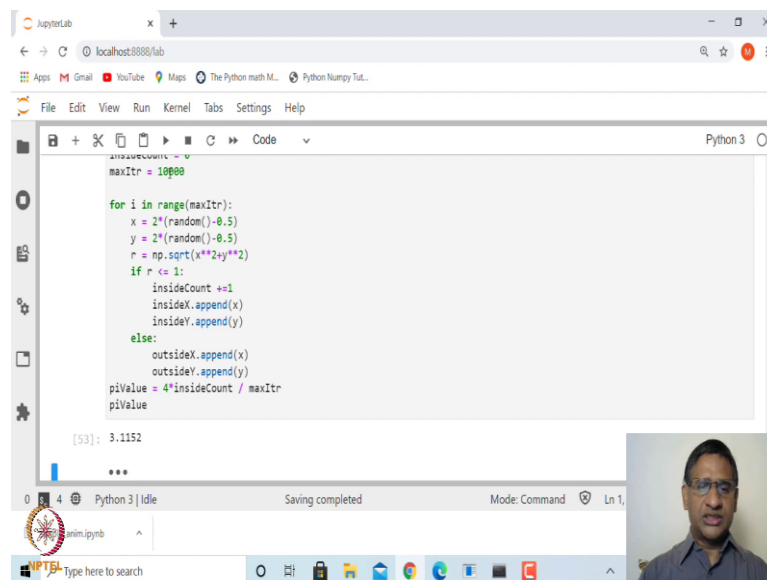
for i in range(maxItr):
    x = 2*(random()-0.5)
    y = 2*(random()-0.5)
    r = np.sqrt(x**2+y**2)
    if r <= 1:
        insideCount +=1
        insideX.append(x)
        insideY.append(y)
    else:
        outsideX.append(x)
        outsideY.append(y)

piValue = 4*insideCount / maxItr
piValue
```

[52]: 3.124

So, let us generate this. So, in this case, when we have 1000 points, the approximate value is 3.124. Of course, this is random, so, it may vary.

(Refer Slide Time: 31:00)



```
from random import random
from math import sqrt
insideCount = 0
maxItr = 10000

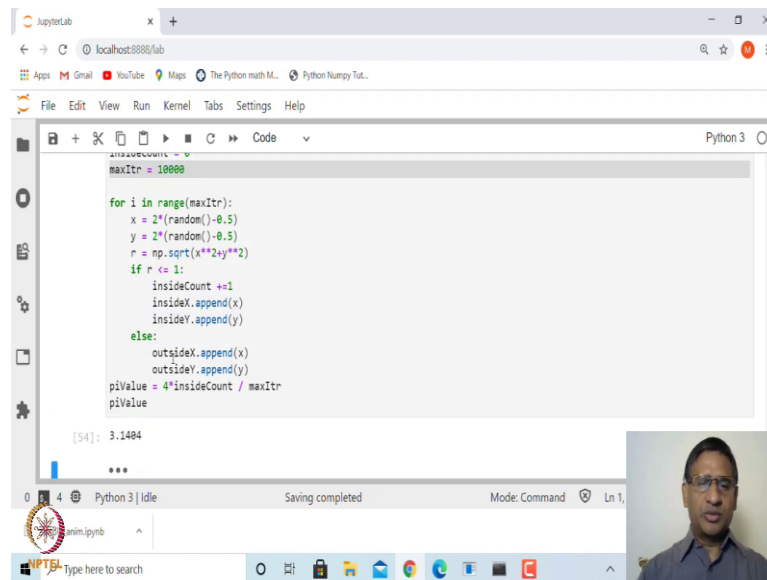
for i in range(maxItr):
    x = 2*(random()-0.5)
    y = 2*(random()-0.5)
    r = np.sqrt(x**2+y**2)
    if r <= 1:
        insideCount +=1
        insideX.append(x)
        insideY.append(y)
    else:
        outsideX.append(x)
        outsideY.append(y)

piValue = 4*insideCount / maxItr
piValue
```

[53]: 3.1152

So, instead of 1000 point, if I say 10000 points then it is 3.11.

(Refer Slide Time: 31:05)



The screenshot shows a JupyterLab window with a Python 3 kernel. The code in the cell is as follows:

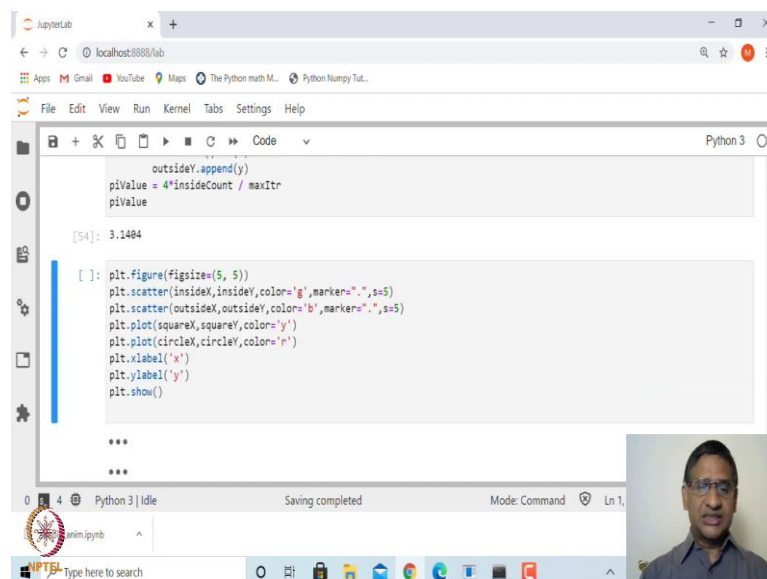
```
insideCount = 0
maxItr = 10000

for i in range(maxItr):
    x = 2*(random()-0.5)
    y = 2*(random()-0.5)
    r = np.sqrt(x**2+y**2)
    if r <= 1:
        insideCount +=1
        insideX.append(x)
        insideY.append(y)
    else:
        outsideX.append(x)
        outsideY.append(y)

piValue = 4*insideCount / maxItr
piValue
```

The output of the cell is `[54]: 3.1404`. The interface includes a file explorer on the left, a top menu bar, and a bottom status bar.

If I run once more it may be something else, ok? So, that is how you can; now, now we want to plot these, these points also. So, this we did earlier, but now extra thing we can even plot. (Refer Slide Time: 31:17)



The screenshot shows the same JupyterLab window with the following code added to the cell:

```
insideCount = 0
maxItr = 10000

for i in range(maxItr):
    x = 2*(random()-0.5)
    y = 2*(random()-0.5)
    r = np.sqrt(x**2+y**2)
    if r <= 1:
        insideCount +=1
        insideX.append(x)
        insideY.append(y)
    else:
        outsideX.append(x)
        outsideY.append(y)

piValue = 4*insideCount / maxItr
piValue

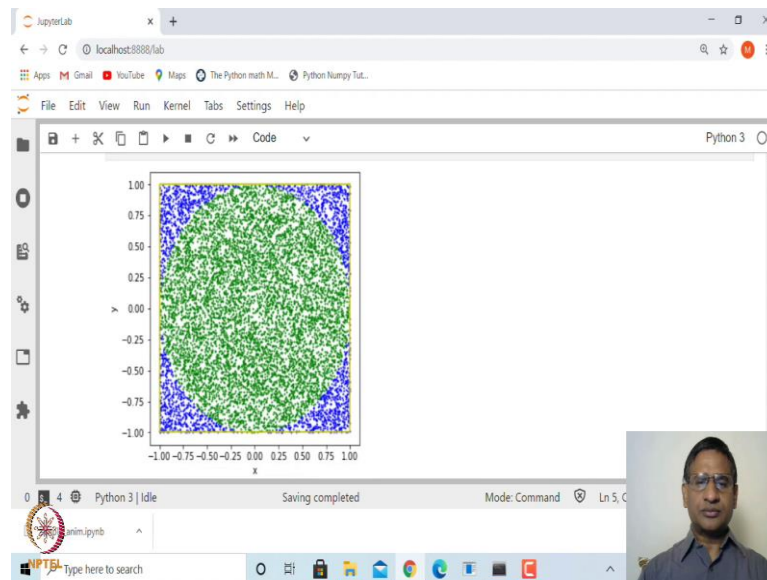
[54]: 3.1404

[ ]: plt.figure(figsize=(5, 5))
plt.scatter(insideX, insideY, color='g', marker='.', s=5)
plt.scatter(outsideX, outsideY, color='b', marker='.', s=5)
plt.plot(squareX, squareY, color='y')
plt.plot(circleX, circleY, color='r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

The output of the cell is `[]: 3.1404`. The interface includes a file explorer on the left, a top menu bar, and a bottom status bar.

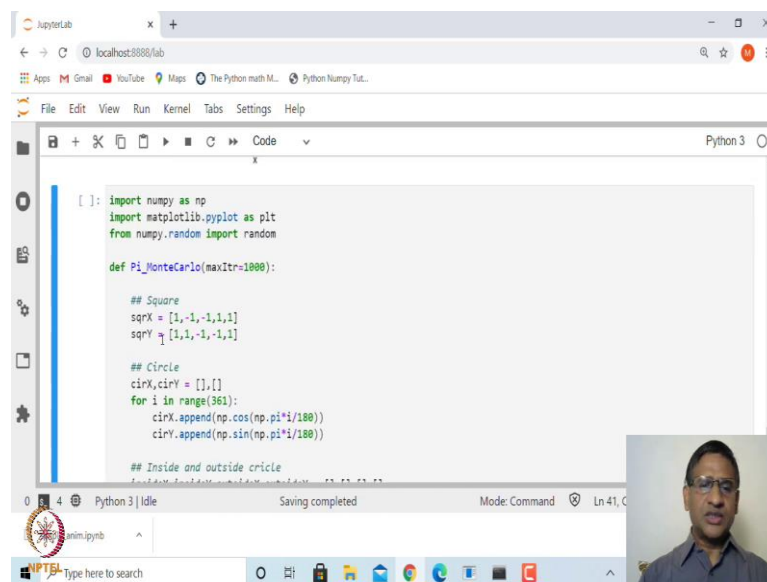
So, how do we plot this? This is again very easy. So, first you will generate a figure size let us a 5 by 5. Plot these all these scattered points inside x-coordinate of inside points, y-coordinate of inside point. Let us say green in color and the point size is, let us say 5. Similarly, plot all those points which are outside in blue in color and plot the square, plot the circle and then put, let us say, label x-axis label, y-axis label, and then ask it to show.

(Refer Slide Time: 31:54)



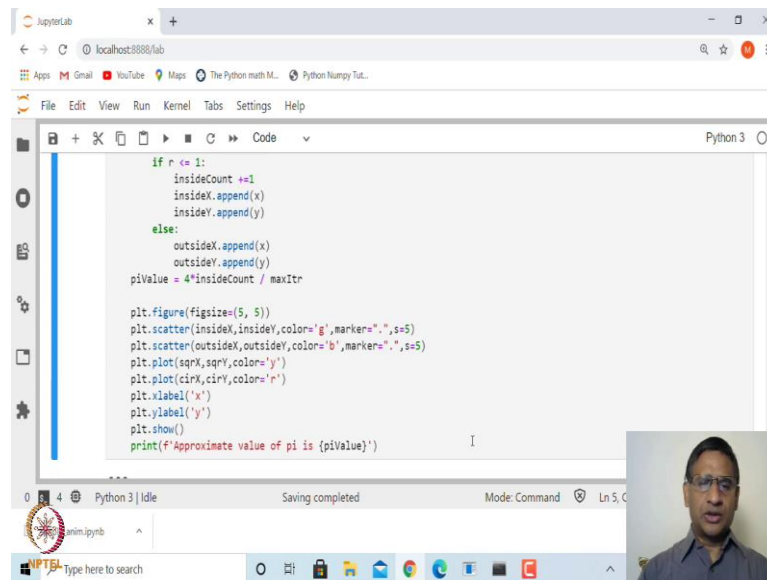
So, that is. So, these are the 10000 points. Green are inside the circle, the blue are, are outside the circles and then we are taking the ratio. Of course, you can make a user-defined function for this.

(Refer Slide Time: 32:08)

A screenshot of a JupyterLab interface. The main window displays a Python code cell. The code defines a function 'Pi_MonteCarlo' that approximates pi using a Monte Carlo simulation. The code includes imports for numpy and matplotlib, and a loop that generates random points and checks if they are inside a unit circle. The JupyterLab interface includes a file browser on the left, a code editor at the top, and a terminal at the bottom. A small video feed of a person is visible in the bottom right corner.

So, let us see how do we; how do we do that. So, we are making a user-defined function called pi underscore MonteCarlo. Monte Carlo simulation of approximating pi, and then by default, let us say we choose 10000, 1000 points.

(Refer Slide Time: 32:38)



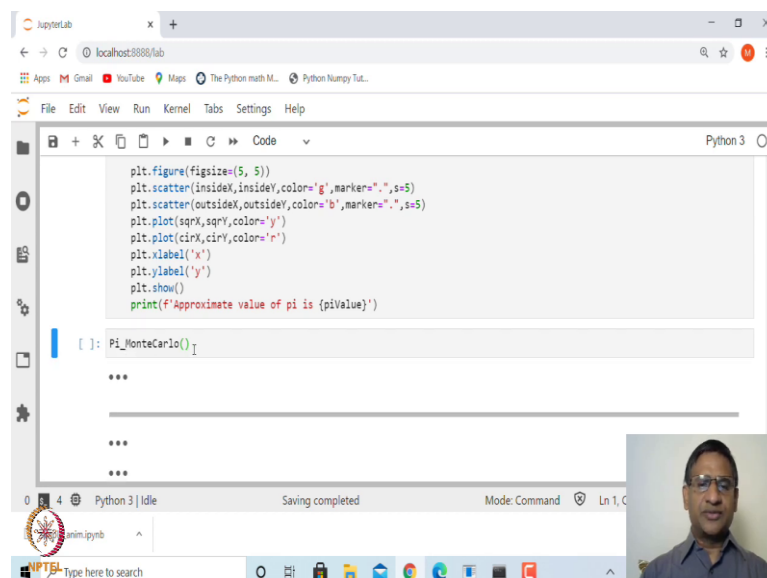
```
if r <= 1:
    insideCount +=1
    insideX.append(x)
    insideY.append(y)
else:
    outsideX.append(x)
    outsideY.append(y)
piValue = 4*insideCount / maxIter

plt.figure(figsize=(5, 5))
plt.scatter(insideX, insideY, color='g', marker='.', s=5)
plt.scatter(outsideX, outsideY, color='b', marker='.', s=5)
plt.plot(sqrtX, sqrtY, color='y')
plt.plot(cirX, cirY, color='r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
print(f'Approximate value of pi is {piValue}')
```

And all we, I have done here is, whatever we have generated all these steps of generating circle, generating unit square and then generating all these random points, counting how many are inside, how many are outside and then plot. All these things are put inside this body of the function and at the end we are also printing, printing what is approximation of pi.

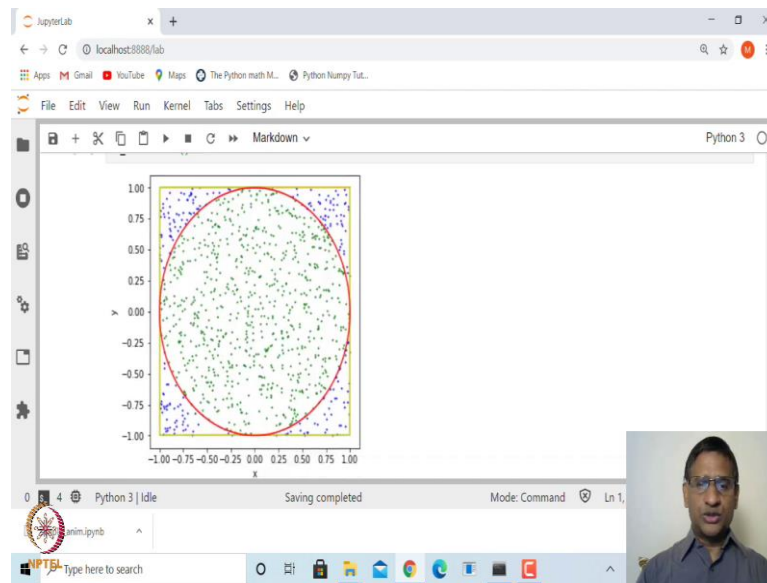
So, that is, that is, that will create. So, if you just look at this particular code, it is nothing but we have just copied all these things inside the body of the function and you can just.

(Refer Slide Time: 33:04)



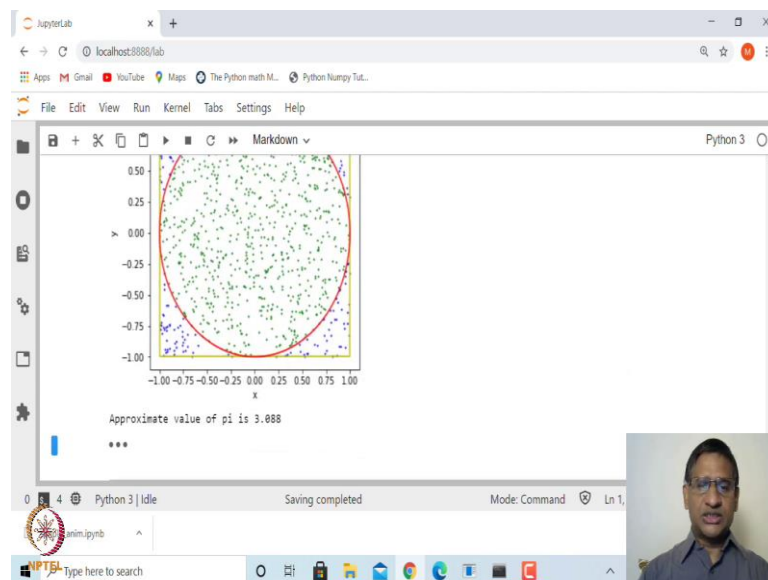
```
def Pi_MonteCarlo():
    ...
    ...
    ...
    plt.figure(figsize=(5, 5))
    plt.scatter(insideX, insideY, color='g', marker='.', s=5)
    plt.scatter(outsideX, outsideY, color='b', marker='.', s=5)
    plt.plot(sqrtX, sqrtY, color='y')
    plt.plot(cirX, cirY, color='r')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
    print(f'Approximate value of pi is {piValue}')
```

(Refer Slide Time: 33:09)



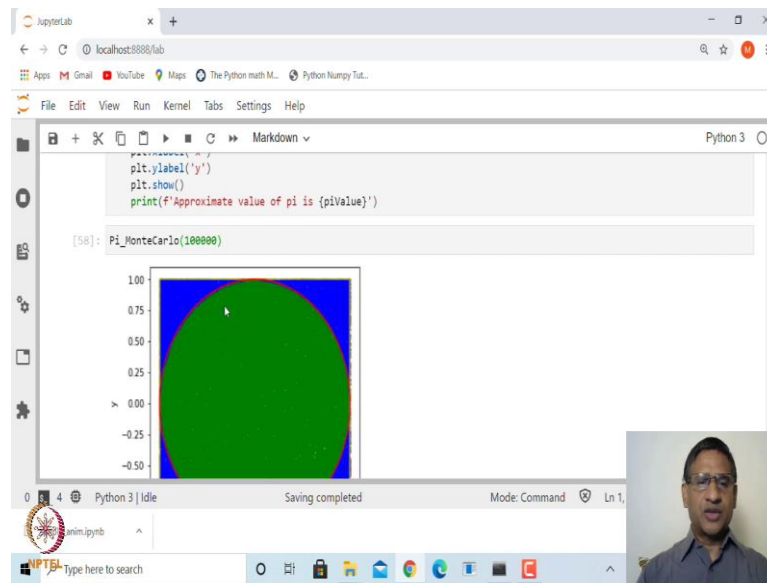
When you run this now, let us call this say pi underscore Monte Carlo, by default it will generate 10000 points, right?

(Refer Slide Time: 33:12)



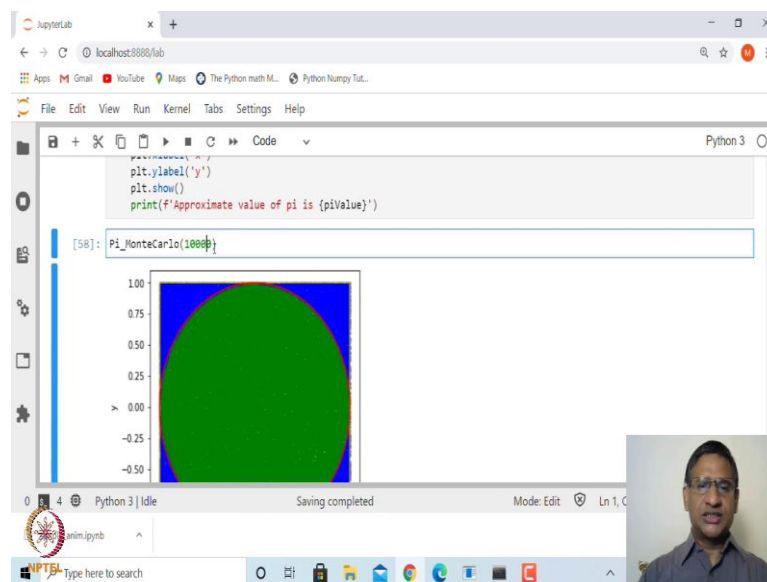
So, this is, sorry 1000 points and the approximate value of pi in this case, is 3.088.

(Refer Slide Time: 33:21)

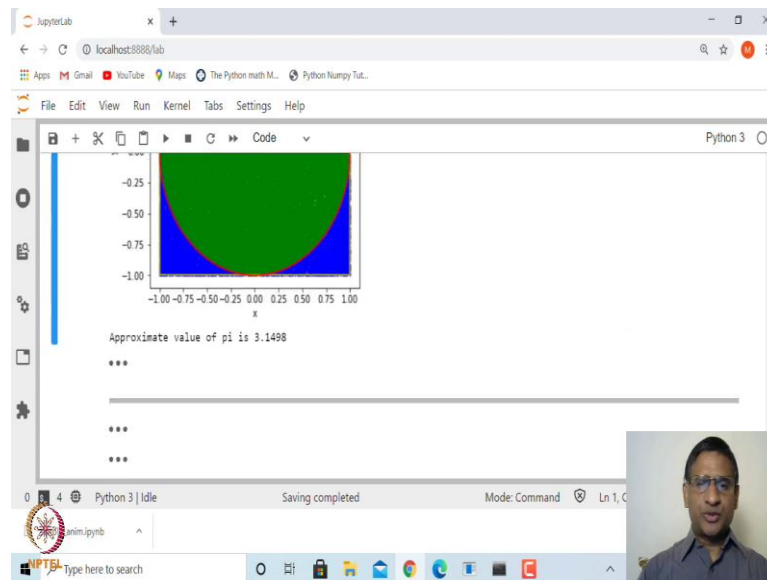


If I increase this, instead of, let us say, call, give a big, big number. So, this is 100000 points, and it may take a little while, but you can see here now you can hardly see the points.

(Refer Slide Time: 33:39)

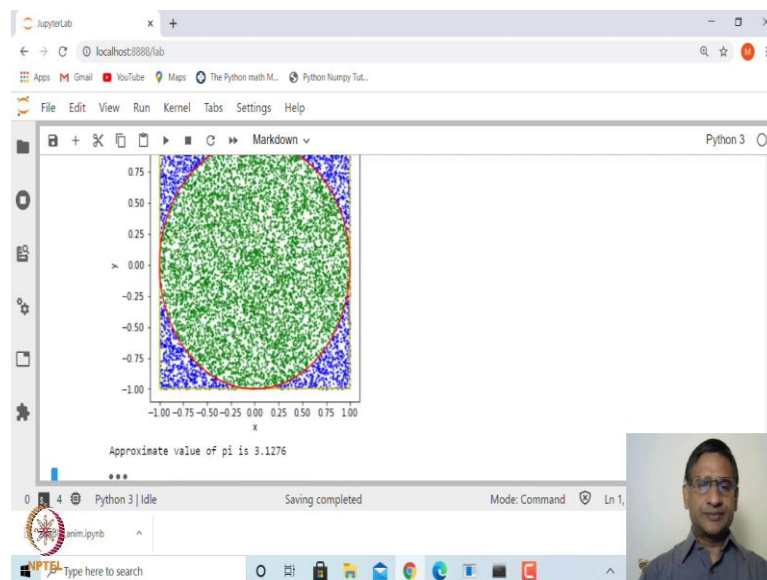


(Refer Slide Time: 33:41)



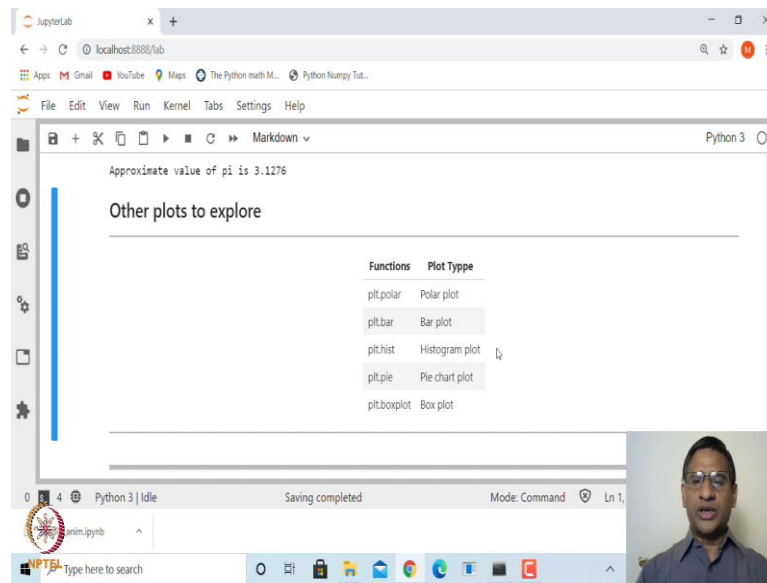
Let me reduce the point size, but in this case what is the approximation? Approximation is 3.14 so, correct up to two decimal places in this case.

(Refer Slide Time: 33:49)



And let us say if I want 10000 points this is how it looks like, ok? So, right.

(Refer Slide Time: 33:55)



The screenshot displays a JupyterLab environment. The top browser window shows the URL `localhost:8888/lab`. The notebook interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a toolbar. The main content area is a Markdown cell with the text "Approximate value of pi is 3.1276" and a section titled "Other plots to explore". Below this title is a table:

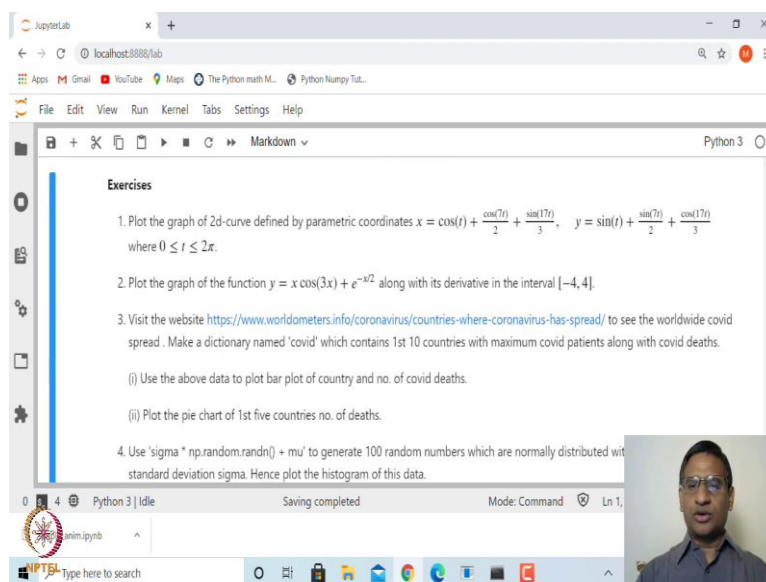
Functions	Plot Type
<code>plt.polar</code>	Polar plot
<code>plt.bar</code>	Bar plot
<code>plt.hist</code>	Histogram plot
<code>plt.pie</code>	Pie chart plot
<code>plt.boxplot</code>	Box plot

The bottom status bar indicates "Python 3 | Idle", "Saving completed", "Mode: Command", and "Ln 1,". A small video feed of a person is visible in the bottom right corner.

So, you can, there are other plots which are available inside the plt inside pyplot. So, for example, there is a pyplot dot polar to plot polar graph or polar functions defined in polar coordinates.

If you want to plot bar plots then you can look at pyplot dot bar, pyplot dot hist will give you histogram plot, pyplot dot pie will give you a pie chart plot, pyplot dot box plot will give you box plot. So, those who, who has, who knows bit of a statistics etcetera, you can look at all these plots, and explore, right?. So, it also has option to plot graphs in three dimensions, etcetera, but all these things we will be also doing in SageMath. There it is much better and much more convenient, ok?

(Refer Slide Time: 34:51)



So, at the end, let me leave you with few exercises. First is to plot parametric curve whose x-coordinate is given by this, y-coordinate is given by this and t varies between 0 to 2 pi. The next is, plot graph of a function which is, y is equal to x into sin 3 x plus e to the power minus x by 2 along with its derivative in the interval minus 4 to 4. So, you have to calculate the derivative and define the function product.

Next is, look at this website, which is actually a website which gives you the corona count worldwide. So, from this, you can look at the, the data of 1st 10 countries with maximum number of covid patients along with the covid death, and make a dictionary out of this data. So, you can go to this website and define a dictionary. Let us call this dictionary as covid.

So, this is the, the, the key, and its value can be another dictionary with two fields. Two keys: one is the total number of patients, and the third, the second one, is the number of deaths. And from this dictionary now extract this data and try to plot a bar plot of number of deaths of all these countries.

You can also plot, for example, pie chart of 1st five countries, again by looking at the number of deaths. You can plot the other things, for example, density plot and other things, etcetera. Next problem is, again let us generate a normally distributed points with mean μ and a standard deviation σ , using this, this command, but we have also seen random dot normal that can also be used.

So, generate 100 random numbers which are normally distributed with mean μ and standard deviation σ , and plot its histogram, and maybe also density plot, ok; but, there are other more, actually libraries which can make the Python plot or Python graphics much better. One of them is called seaborn.

So, you can explore seaborn also, ok? So, let me stop here. So, thank you very much, and in the next class, we should look at another two important libraries, which are called scipy and sympy for scientific computing and also symbolic computing. That again we will look at very briefly, because all these things we will be doing in, in SageMath.

Thank you very much.