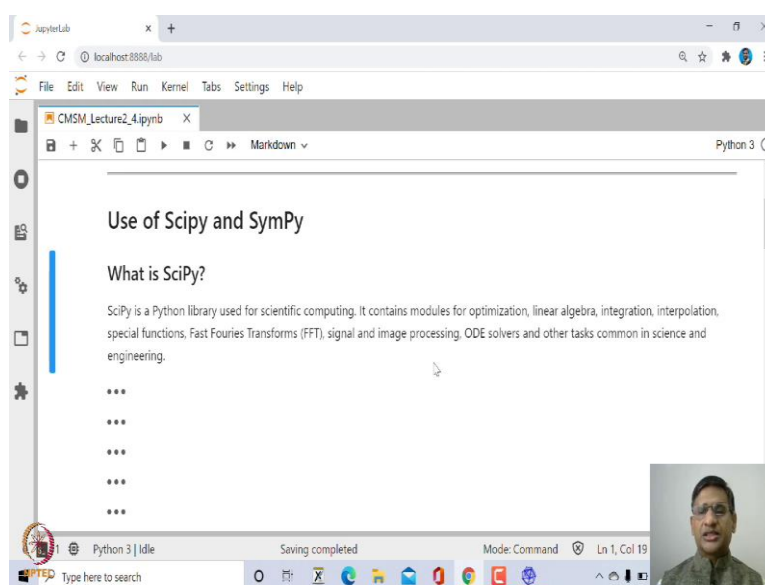


Computational Mathematics with SageMath
Prof. Ajit Kumar
Department of Mathematics
Institute of Chemical Technology, Mumbai

Lecture – 12
Use of SciPy and SymPy in Python

Welcome to the 11th lecture on Computational Mathematics with SageMath. In this lecture, we will explore two python libraries namely SciPy, and SymPy.

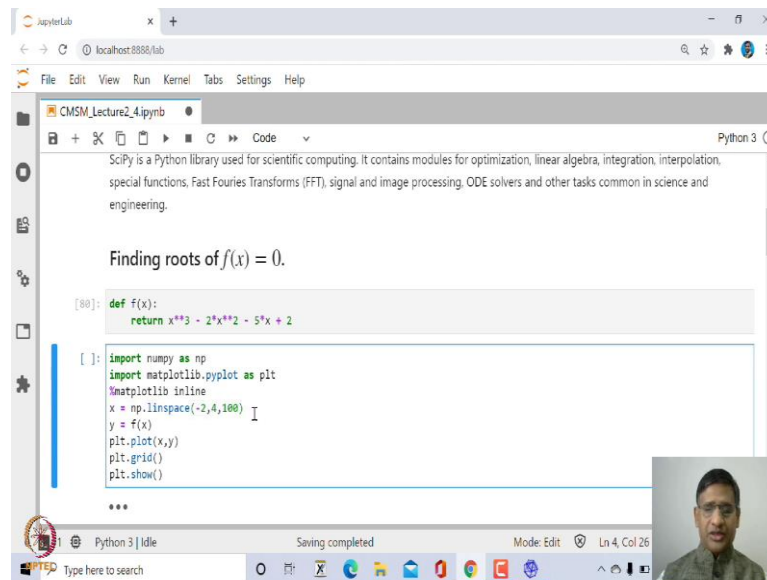
(Refer Slide Time: 00:27)



These two libraries are very important, and useful when we want to use Python in order to explore mathematical concepts. So, let us look at what is SciPy?

SciPy is a Python library that uses scientific computing. It contains modules for optimization, for linear algebra, for interpolation, integration, special functions, fast fourier transforms, signal, and image processing, ODE solver, and similar other common tasks in science, and engineering. So, let us make use of SciPy for doing some numerical computations.

(Refer Slide Time: 01:23)



The image shows a JupyterLab window with a file named 'CMSM_Lecture2_4.ipynb'. The code in the cell is as follows:

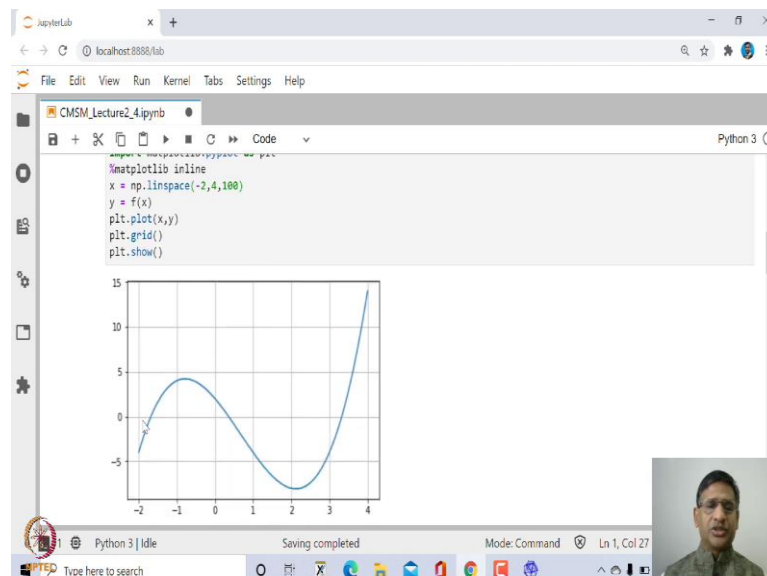
```
def f(x):  
    return x**3 - 2*x**2 - 5*x + 2  
  
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline  
x = np.linspace(-2,4,100)  
y = f(x)  
plt.plot(x,y)  
plt.grid()  
plt.show()
```

The code defines a function $f(x) = x^3 - 2x^2 - 5x + 2$ and plots it using Matplotlib. The x-axis ranges from -2 to 4 with 100 points. The y-axis ranges from -5 to 15. The plot shows a cubic curve with three real roots.

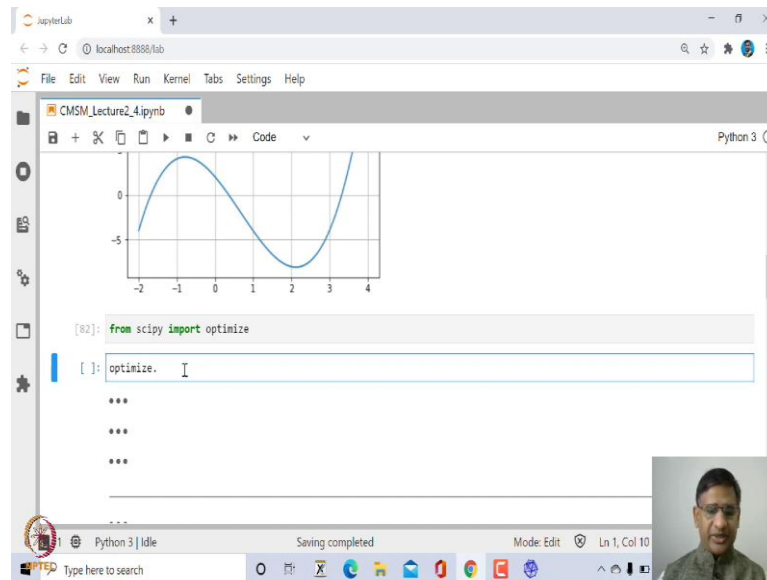
So, suppose we want to find a root of this equation $f(x)$ equal to 0 or zero of $f(x)$. Let us begin with an example; suppose we have a function $f(x)$ which is x cube minus $2x$ square minus $5x$ plus 2 . Since this is a quadratic, this will have a real zero.

So, let us run this, and then let us first plot graph of this function. So, we will use pyplot from matplotlib, and let us plot its graph between minus 2, and 4. So, first, we will generate the x values between minus 2 and 4; let us say, there are a 100 points, and then let us plot its graph.

(Refer Slide Time: 02:19)

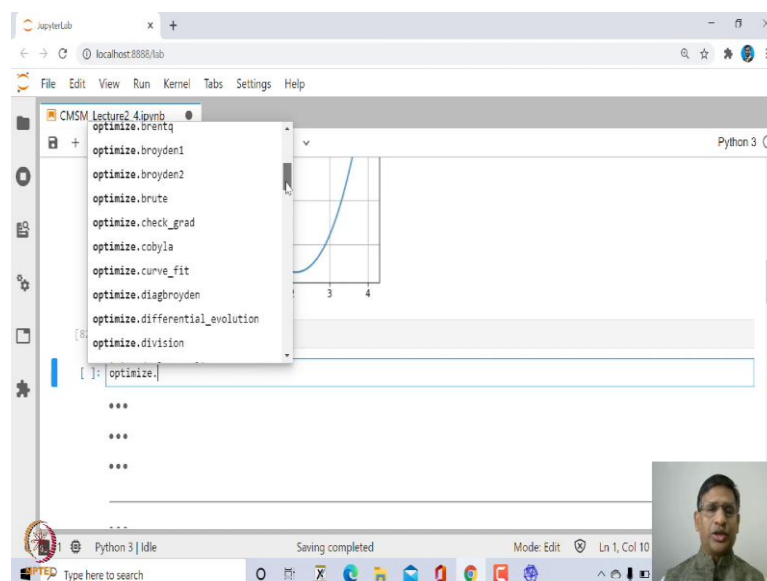


So, this is how the graph looks, and see. We have put grids, so that we can locate its roots. So, there is a root between minus 2 and 1 somewhere here; there is also a root between 0 and 1, and there is a root between 3 and 4. So, we want to locate these roots. (Refer Slide Time: 02:43)



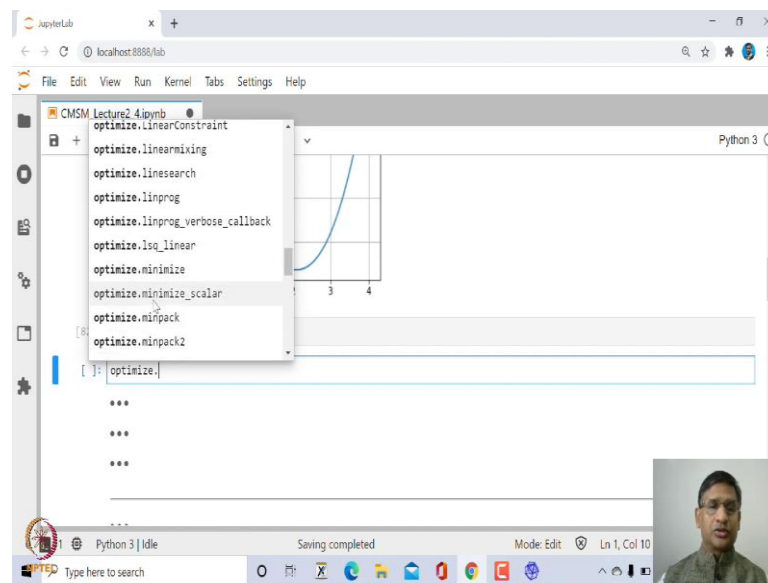
So, how do we do that? So, inside SciPy, there is another module called optimize, and inside this optimize, you can find several functions related to finding zeros, also related to finding optimization or maximum/minimum of a, a function of 1 and 2 variables, and using different methods. So, let us first import this optimize. If you try to look at help on this optimize.

(Refer Slide Time: 03:27)

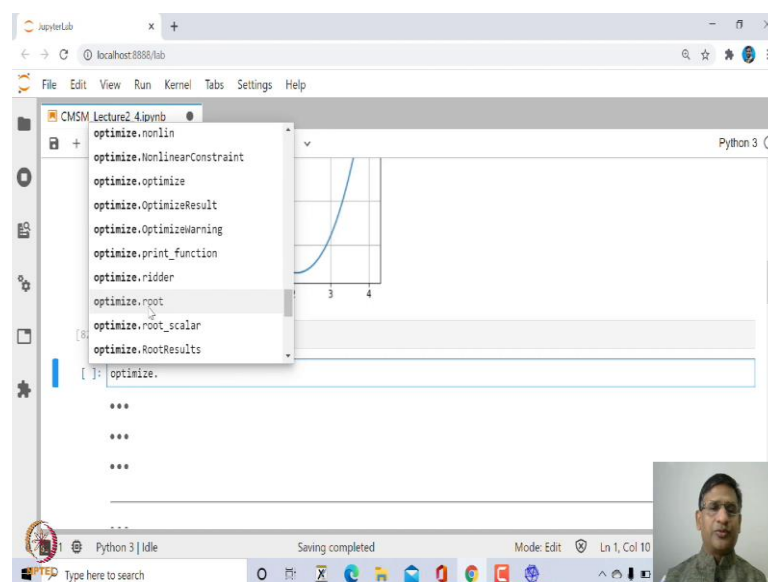


So, let us use optimize dot tab, let us press tab, and then you can see here there are several options available. In case you have gone through a course on optimization, you would be able to recognize some of the methods. So, here you have also, you have curve, curve fitting curves.

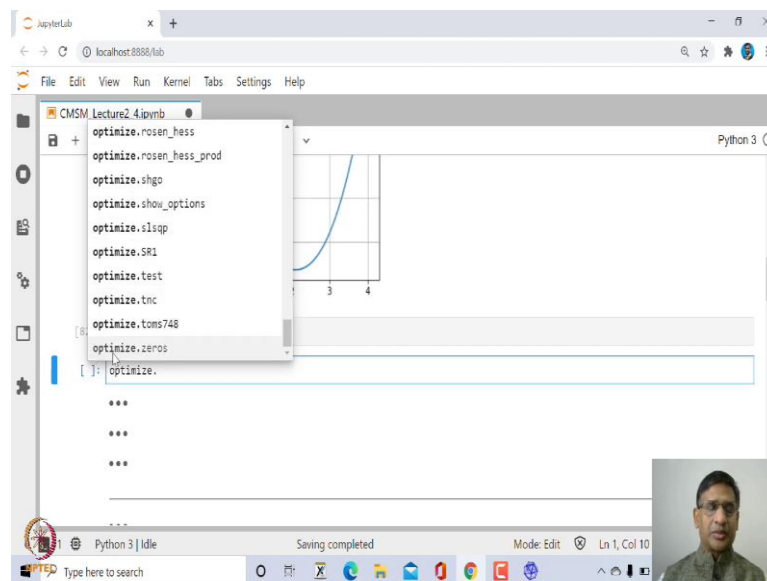
(Refer Slide Time: 03:51)



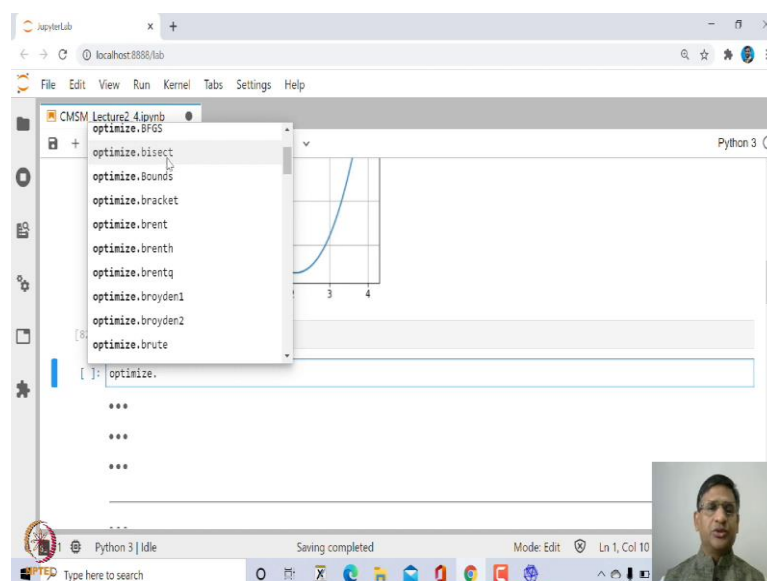
And there are several methods, including finding minimum/maximum; you can, you can also use it for solving linear programming problem. (Refer Slide Time: 03:59)



And we also have a method to find roots; this can find root of one variable function, and also multi-variable function. (Refer Slide Time: 04:09)

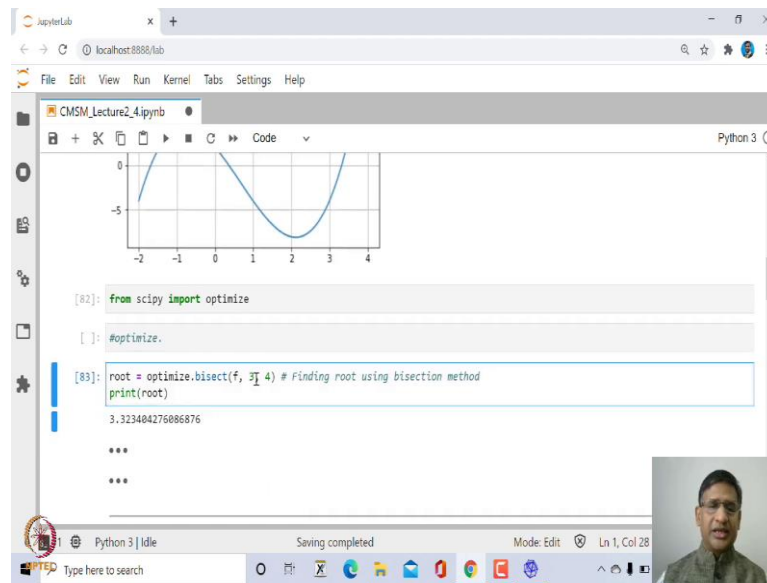


But in this case, you have also a zeros which can find zeros; but there is a function. (Refer Slide Time: 04:21)



Let us see, there is one function called bisect, which will use bisection method.

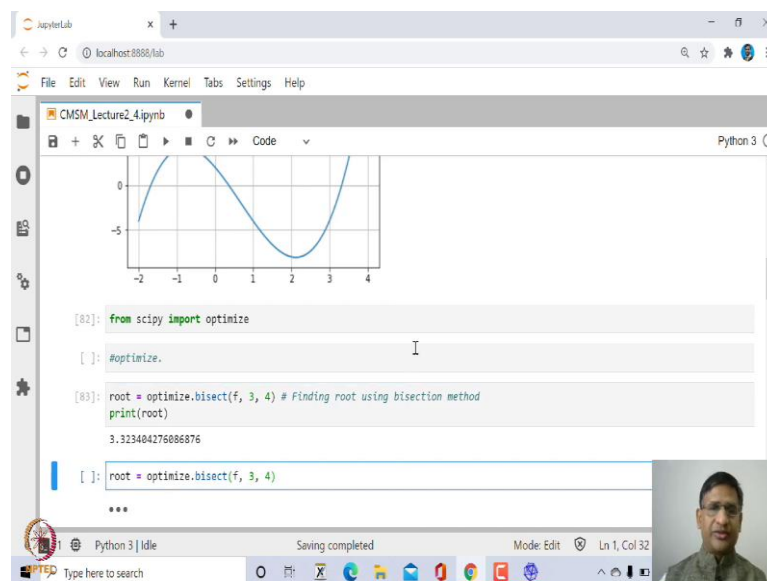
(Refer Slide Time: 04:29)



So, let us make use of this bisect. So, let us say, optimize dot bisect f, and you need to mention the interval in which we want to find a root, so that is 3 comma 4.

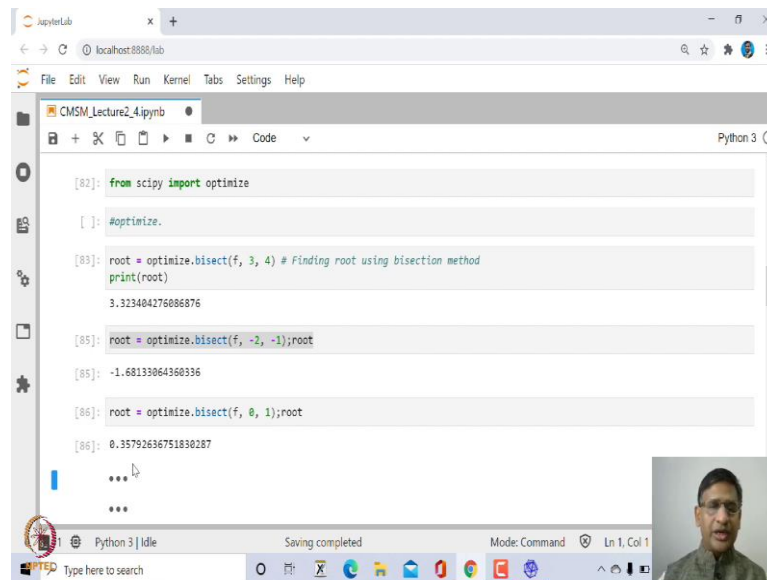
So, it will find a root between 3 and 4. So, let us run this. So, the root, in this case, is 3.3223404, and so on, right? So, in case we, we give an interval, let us say, some other interval.

(Refer Slide Time: 05:03)



Say for example

(Refer Slide Time: 05:07)



The image shows a JupyterLab window with a file named 'CMSM_Lecture2_4.ipynb'. The code in the notebook is as follows:

```
[82]: from scipy import optimize
[ ]: #optimize.

[83]: root = optimize.bisect(f, 3, 4) # Finding root using bisection method
      print(root)
      3.323404276086876

[85]: root = optimize.bisect(f, -2, -1);root
[85]: -1.68133064360336

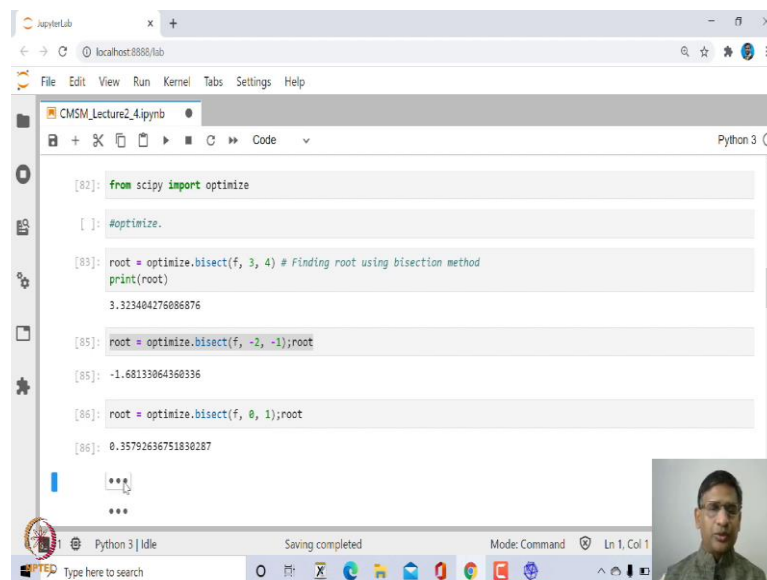
[86]: root = optimize.bisect(f, 0, 1);root
[86]: 0.35792636751830287

...
...
```

The bottom of the window shows a Windows taskbar with various icons and a search bar. A small video feed of a man is visible in the bottom right corner.

let us mention the interval which is, which is minus 2 to minus 1; then it will find a root in this interval. So, let us call for this root, yes.

(Refer Slide Time: 05:25)



The image shows a JupyterLab window with a file named 'CMSM_Lecture2_4.ipynb'. The code in the notebook is as follows:

```
[82]: from scipy import optimize
[ ]: #optimize.

[83]: root = optimize.bisect(f, 3, 4) # Finding root using bisection method
      print(root)
      3.323404276086876

[85]: root = optimize.bisect(f, -2, -1);root
[85]: -1.68133064360336

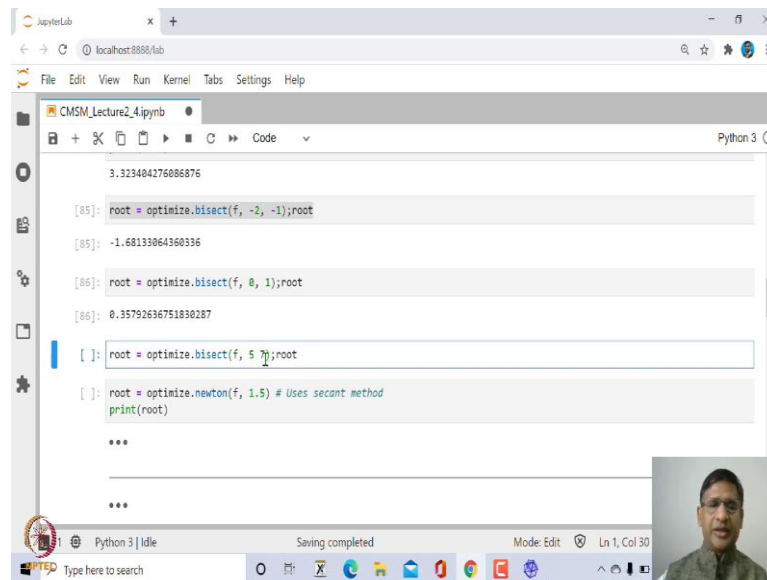
[86]: root = optimize.bisect(f, 0, 1);root
[86]: 0.35792636751830287

...
...
```

The bottom of the window shows a Windows taskbar with various icons and a search bar. A small video feed of a man is visible in the bottom right corner.

And if I, if you, let us say, interval between 0 and 1; then you will get a root in, in the, in the interval 0 to 1. However, if you give an interval in which it does not have a root.

(Refer Slide Time: 05:45)



The screenshot shows a JupyterLab window with a file named 'CMSM_Lecture2_4.ipynb'. The code in the cell is as follows:

```
3.323404276086876

[85]: root = optimize.bisect(f, -2, -1);root
[85]: -1.68133064360336

[86]: root = optimize.bisect(f, 0, 1);root
[86]: 0.35792636751830287

[ ]: root = optimize.bisect(f, 5, 7);root

[ ]: root = optimize.newton(f, 1.5) # Uses secant method
print(root)

...

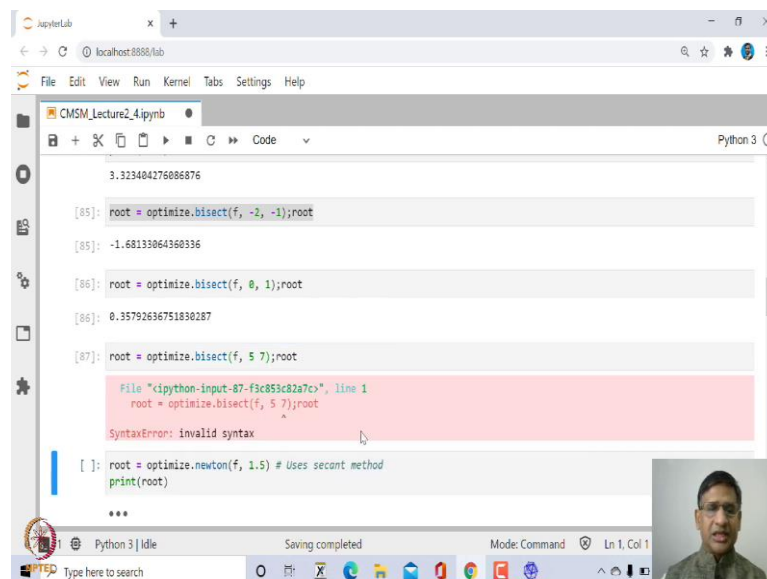
...

```

The interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help), a toolbar, and a status bar at the bottom indicating 'Python 3 | idle' and 'Saving completed'.

So, for example, let us say, give a between 4, and or let us say 5, and 7.

(Refer Slide Time: 05:49)



The screenshot shows the same JupyterLab window, but now a syntax error has occurred. The error message is displayed in a red box:

```
File "c:\python-input-87-f3c853c82a7c", line 1
root = optimize.bisect(f, 5, 7);root
SyntaxError: invalid syntax

```

The code cell now shows the error on line 1, which corresponds to the line `root = optimize.bisect(f, 5, 7);root`. The status bar at the bottom now shows 'Mode: Command'.

And then if you run this, then 5 comma 7; then it will give you value error.

(Refer Slide Time: 05:53)

The screenshot shows a JupyterLab window with a file named 'CMSM_Lecture2_4.ipynb'. The code in the cell is as follows:

```
[86]: root = optimize.bisect(f, 0, 1);root
[86]: 0.35792636751830287

[88]: root = optimize.bisect(f, 5, 7);root

ValueError                                Traceback (most recent call last)
<ipython-input-88-ba055a025680> in <module>()
----> 1 root = optimize.bisect(f, 5, 7);root

/opt/sagemath-9.1/local/lib/python3.7/site-packages/scipy/optimize/zeros.py in bisect(f, a, b, args, xtol, rtol, maxite
r, full_output, disp)
   529     if rtol < _rtol:
   530         raise ValueError("rtol too small (%g < %g)" % (rtol, _rtol))
--> 531     r = _zeros_bisect(f, a, b, xtol, rtol, maxiter, args, full_output, disp)
   532     return results_c(full_output, r)
   533

ValueError: f(a) and f(b) must have different signs

[ ]: root = optimize.newton(f, 1.5) # Uses secant method
```

The error message indicates that the function f must have different signs at the endpoints a and b for the bisection method to work. The user is running the code in a JupyterLab environment, and the error is displayed in a red box. The user is also running a Newton-Raphson method using `optimize.newton`.

What does it say? It says that $f(a)$, and $f(b)$ must have different signs. In this case, between 5 and 7, the function is positive, therefore it does not have a root. So, ok, that is how you can make use of bisection in order to find a root using bisection method.

(Refer Slide Time: 06:23)

The screenshot shows a JupyterLab window with a file named 'CMSM_Lecture2_4.ipynb'. The code cell contains the following Python code:

```

529 if rtol < _rtol:
530     raise ValueError("rtol too small (%g < %g)" % (rtol, _rtol))
--> 531 r = _zeros_bisect(f, a, b, xtol, rtol, maxiter, args, full_output, disp)
532 return results_c(full_output, r)
533
ValueError: f(a) and f(b) must have different signs

```

Below the error message, the code cell shows the execution of a Newton-Raphson optimization:

```

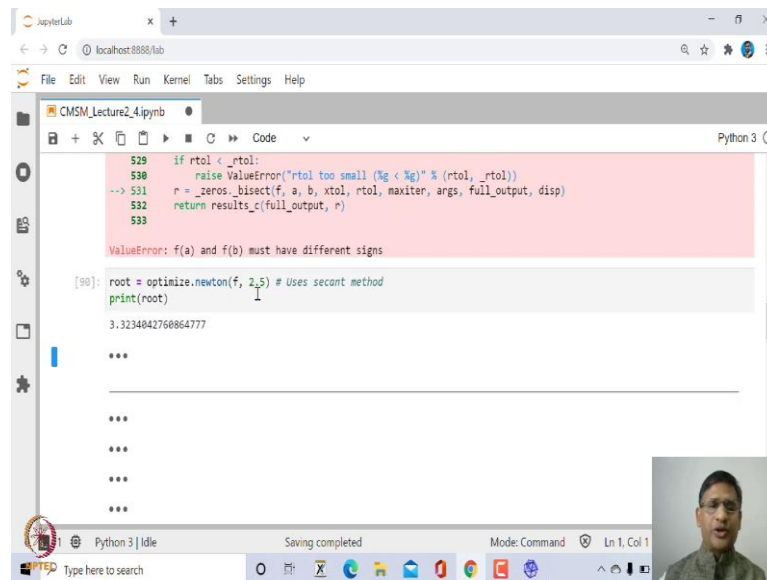
[]: root = optimize.newton(f, 1.5) # Uses secant method
print(root)

```

The output of the cell shows a series of empty lines, indicating that the optimization did not converge within the specified parameters.

But you can also find root using secant method. So, inside optimize, there is a function called Newton, and in case you supply only the function value, and initial guess, it will use secant method in order to find its root.

(Refer Slide Time: 06:35)



The screenshot shows a JupyterLab window with a code cell. The code defines a function `_zeros_bisect` and calls `optimize.newton(f, 2.5)`. A red error message is displayed: `ValueError: f(a) and f(b) must have different signs`. Below the error, the output of the `print(root)` statement is shown as `3.323464276064777`.

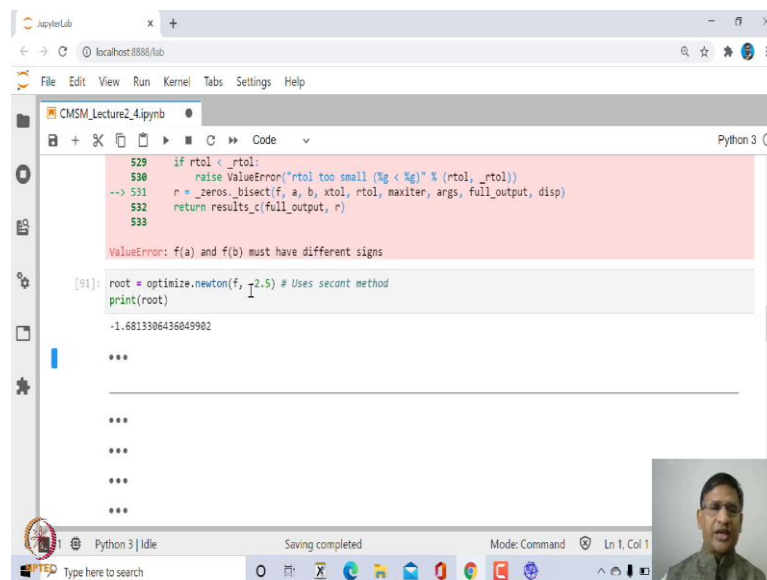
```
529 if rtol < _rtol:
530     raise ValueError("rtol too small (%g < %g)" % (rtol, _rtol))
--> 531 r = _zeros_bisect(f, a, b, xtol, rtol, maxiter, args, full_output, disp)
532 return results_c(full_output, r)
533

ValueError: f(a) and f(b) must have different signs

[90]: root = optimize.newton(f, 2.5) # Uses secant method
      print(root)
      3.323464276064777
      ...
      ...
      ...
      ...
      ...
```

And you can give different guess value. For example, if I say, 2.5 or if I say, minus 2.5, it will find the negative root.

(Refer Slide Time: 06:39)



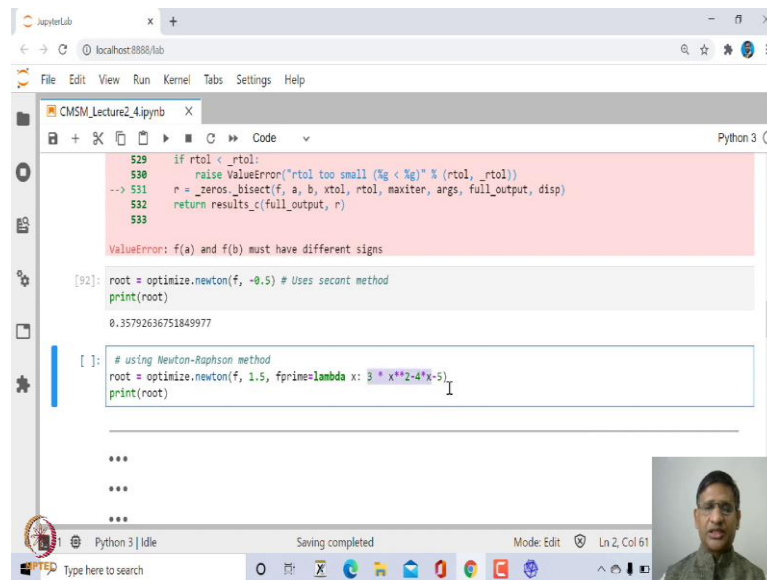
The screenshot shows a JupyterLab window with a code cell. The code is identical to the previous one, but the guess value in `optimize.newton(f, -2.5)` is -2.5. The output of the `print(root)` statement is shown as `-1.6813306436049902`.

```
529 if rtol < _rtol:
530     raise ValueError("rtol too small (%g < %g)" % (rtol, _rtol))
--> 531 r = _zeros_bisect(f, a, b, xtol, rtol, maxiter, args, full_output, disp)
532 return results_c(full_output, r)
533

ValueError: f(a) and f(b) must have different signs

[91]: root = optimize.newton(f, -2.5) # Uses secant method
      print(root)
      -1.6813306436049902
      ...
      ...
      ...
      ...
      ...
```

(Refer Slide Time: 06:43)



```
529 if rtol < _rtol:
530     raise ValueError("rtol too small (%g < %g)" % (rtol, _rtol))
--> 531 r = _zeros_bisect(f, a, b, xtol, rtol, maxiter, args, full_output, disp)
532 return results_c(full_output, r)
533
ValueError: f(a) and f(b) must have different signs

[92]: root = optimize.newton(f, -0.5) # Uses secant method
print(root)
0.35792636751849977

[ ]: # using Newton-Raphson method
root = optimize.newton(f, 1.5, fprime=lambda x: 3 * x**2-4*x-5)
print(root)

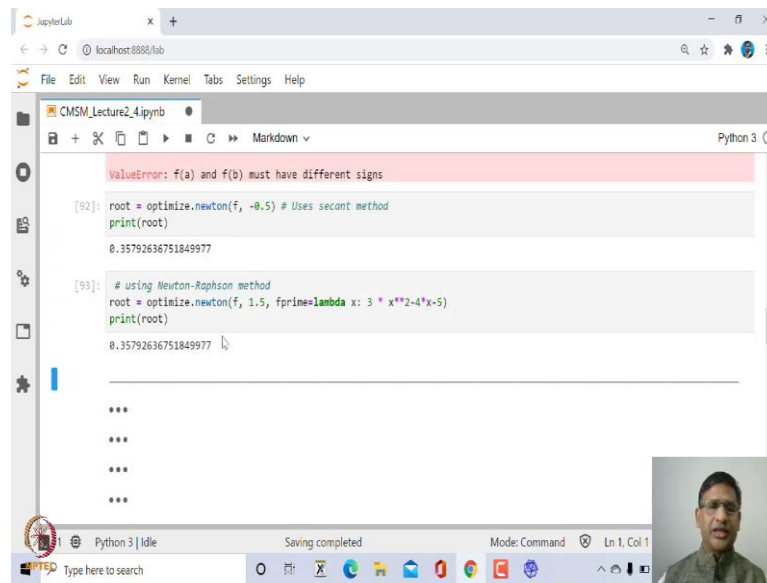
***
***
***
```

Or if I say, let us say, 0 point, negative 0.5, it finds a positive root. So, this, by giving different initial guess values, you can locate different roots, right? In case you supply derivative of this function inside Newton, if you also supply derivative of the function f prime.

So, in this case, we are supplying derivative of the function which is f prime is equal to, we are using lambda notation, lambda x , and this, the output is $3x^2 - 4x - 5$, which is derivative of this function. So, function was $x^3 - 2x^2 - 5x + 2$.

So, its derivative will be $3x^2 - 4x - 5$. So, that is what we are supplying.

(Refer Slide Time: 07:31)



The screenshot shows a JupyterLab window with a file named 'CMSM_Lecture2_4.ipynb'. A red error message is displayed: 'ValueError: f(a) and f(b) must have different signs'. Below the error, two code cells are visible. The first cell (index 92) uses the 'optimize.newton' function with an initial guess of -0.5 and prints the root, which is 0.35792636751849977. The second cell (index 93) uses the 'optimize.newton' function with an initial guess of 1.5 and a derivative function 'fprime' defined as $3 * x^2 - 4 * x - 5$. It also prints the root, which is the same value. The bottom status bar shows 'Python 3 | Idle' and 'Saving completed'.

```
ValueError: f(a) and f(b) must have different signs

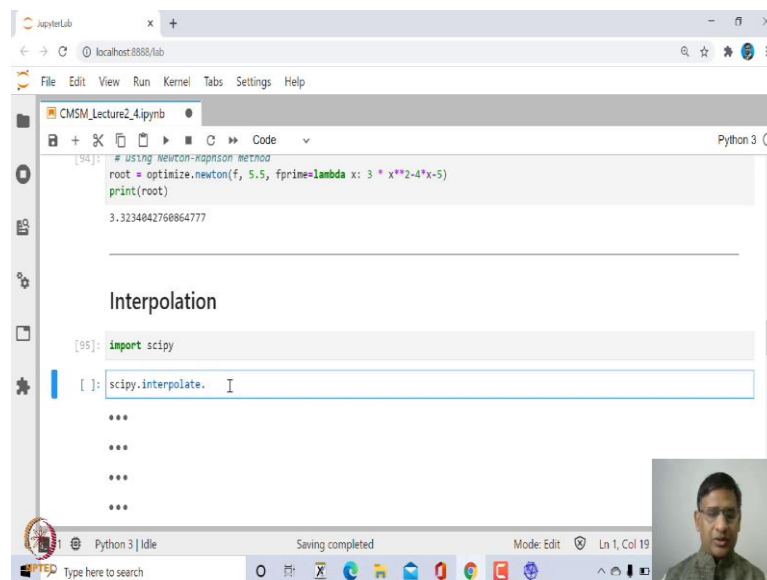
[92]: root = optimize.newton(f, -0.5) # Uses secant method
      print(root)
      0.35792636751849977

[93]: # using Newton-Raphson method
      root = optimize.newton(f, 1.5, fprime=lambda x: 3 * x**2-4*x-5)
      print(root)
      0.35792636751849977

***
***
***
***
```

So, in case we supply derivative of this function, then it will find also a root; but it will use now Newton Raphson method, right?

(Refer Slide Time: 07:45)



The screenshot shows the same JupyterLab window. The first code cell (index 94) uses 'optimize.newton' with an initial guess of 5.5 and the same derivative function, printing the root 3.3234042760864777. Below this, a section titled 'Interpolation' is shown. The second code cell (index 95) starts with 'import scipy'. The third code cell (index 96) starts with 'scipy.interpolate.' and has a cursor at the end of the line. The bottom status bar shows 'Python 3 | Idle' and 'Saving completed'.

```
[94]: # using Newton-Raphson method
      root = optimize.newton(f, 5.5, fprime=lambda x: 3 * x**2-4*x-5)
      print(root)
      3.3234042760864777

Interpolation

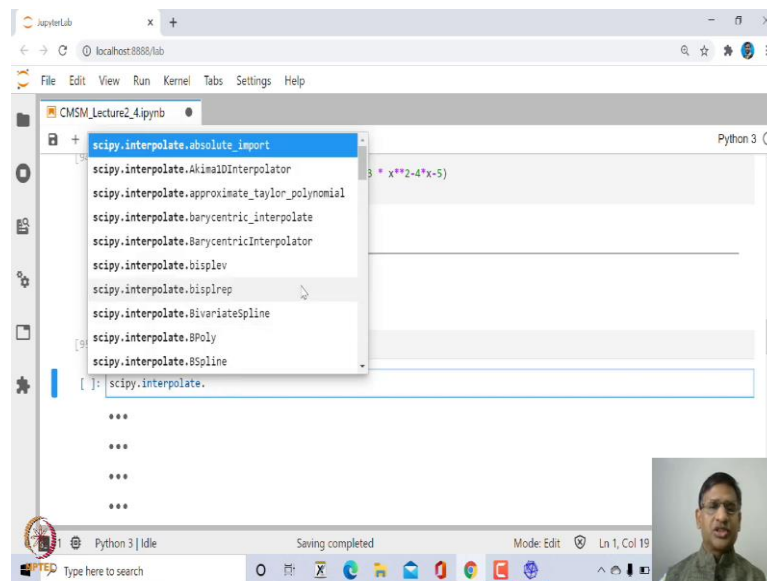
[95]: import scipy

[ ]: scipy.interpolate.
***
***
***
***
```

You can, you can start with different values, for example, if I say, 5.5; then it will find the root between 3 and 4, and so on, ok?

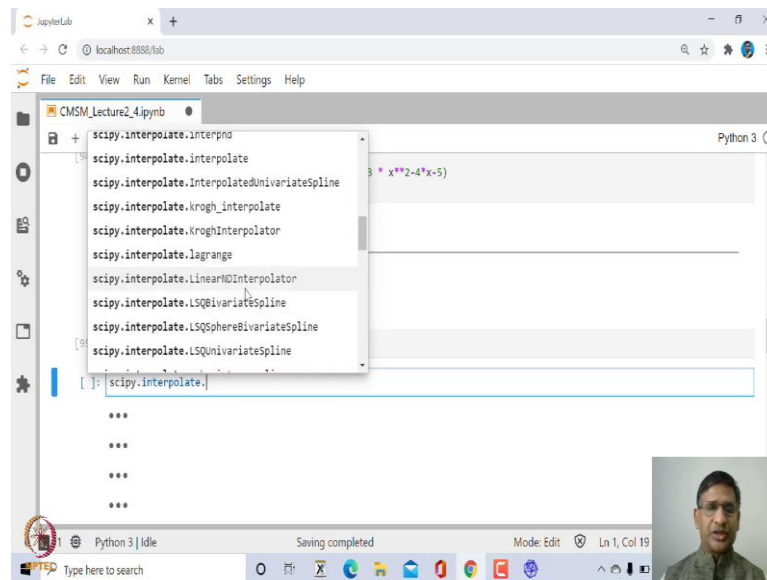
So, next, let us look at how we can do interpolation using a SciPy? So, let us start first importing this SciPy module, and inside this SciPy module, there is a, there is a module called interpolate.

(Refer Slide Time: 08:15)

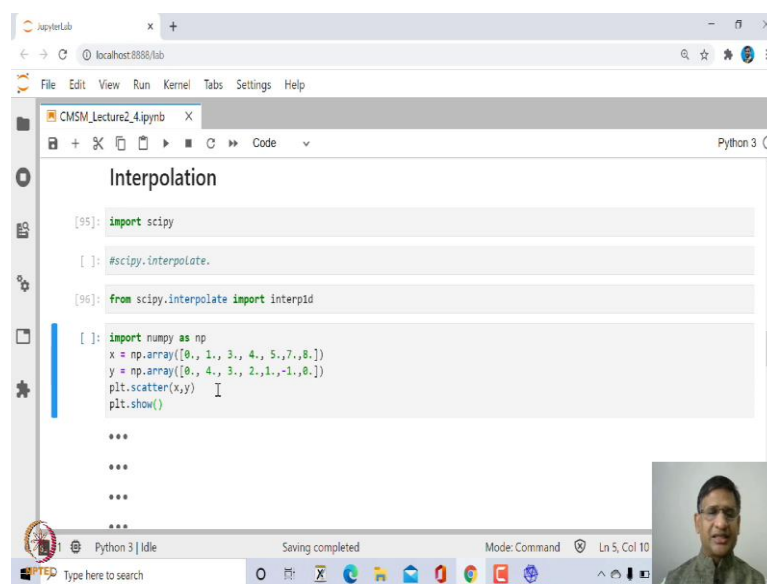


So, if you try to look at some functions that is there, some of the methods which is there inside interpolate module; you will see that, it has an interpolation of one dimension, of two dimension, it also has Lagrange interpolation, and various other, including multi-dimensional, right?

(Refer Slide Time: 08:31)



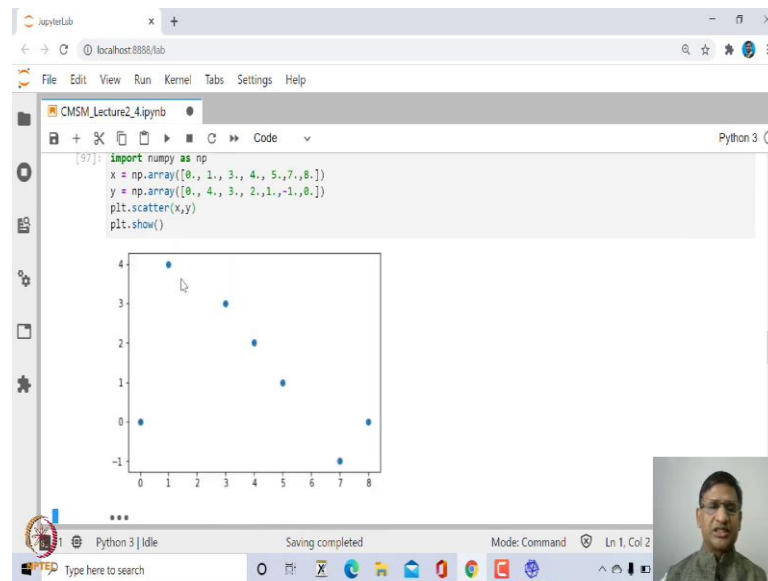
(Refer Slide Time: 08:37)



So, you can explore this more; in case you have learnt interpolations in more detail, you would be able to make use of this, ok? So, let us look at a simple example. So, from SciPy interpolate, let us import `interp1d` that is one-dimensional interpolation. Next, let us look at a set of points.

So, we are taking set of points x comma (x_i, y_i) , and x coordinates are 0, 1, 3, 4, 5, 7, 8; y coordinates are 0, 4, 3, 2, 1, minus 1, 0.

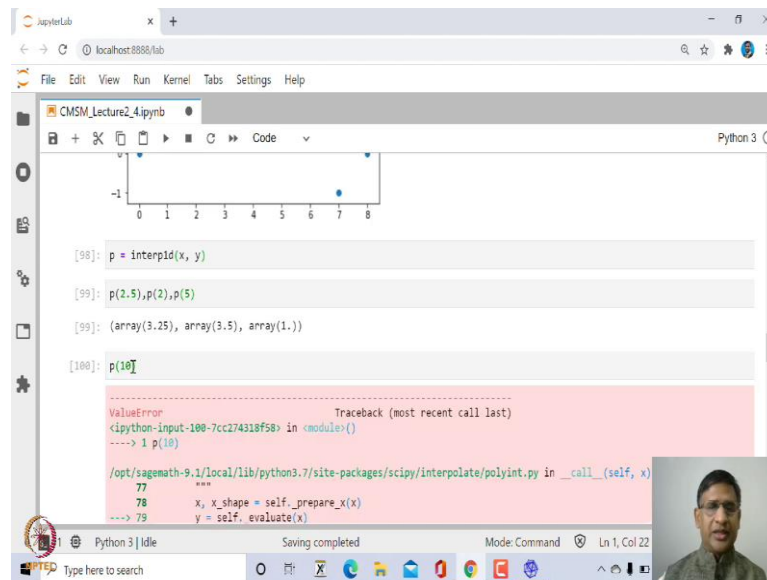
(Refer Slide Time: 09:17)



And let us plot graph of these functions, let us plot these points, scattered points. So, this is how it looks like. Now, inside this, we want to fit a one-dimensional interpolate, interpolating polynomial.

So, it will actually be a set of straight line, something like joining these points. So, let us see.

(Refer Slide Time: 09:35)



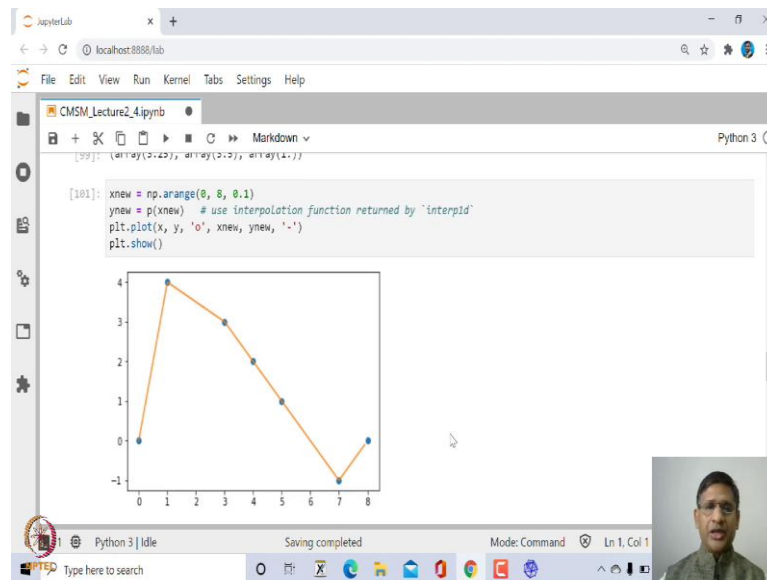
So, what we will do? We will say, `interp1d`, and give them, mention the x-coordinates, and y-coordinates.

So, and we store this in `p`. Now, you can look at what should be `p`. So, for example, if I evaluate value of `p` at different points, let us just say, `p` at 2.5, `p` at 2, `p` at 5. So, 2 and 5 are in the, in the x-coordinates; 2 is not there, but 5 is there.

So, at 5, the value is 1, and at 2, the value is 3.5, at 2.5 the value is 3.25, and it gives you a value as an array, one-dimensional array, right? So, you can evaluate, for example, if I say, what is the value of `p` at let us say, 10; then it gives you error, because this 10 is out of this range, x range, in this case, is between 0 to 8.

So, anything which is outside this will not be able to evaluate, right? So, let me delete this, right?

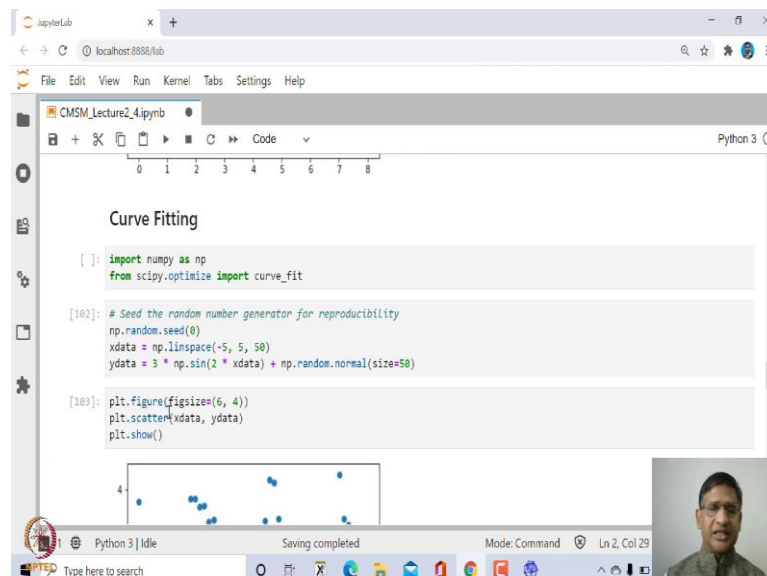
(Refer Slide Time: 10:45)



And you can also now plot its graph, the fitted interpolating polynomial of dimension one. So, let us give x values between 0 to 8, of a step length 0.1; y value is p at new, the xnew that is the x values, and then let us plot its graph along with the points, and the line joining these.

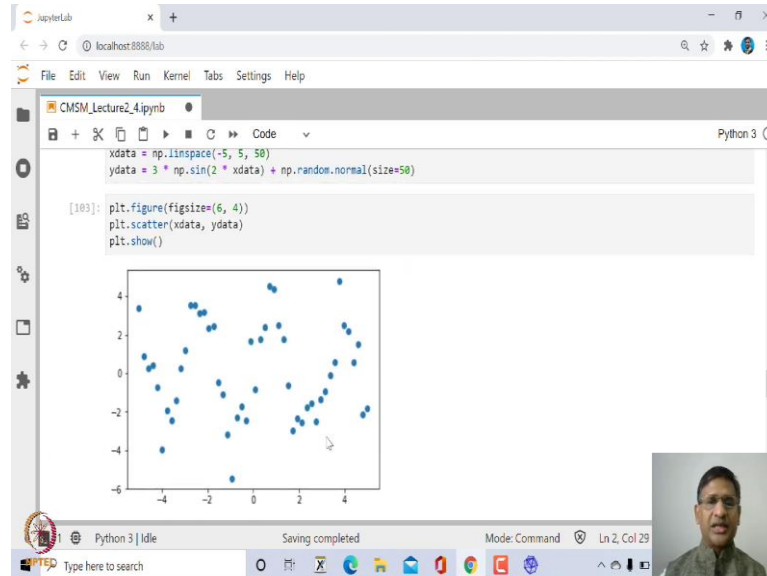
So, this is how the interpolating polynomial looks like. You can take higher a degree interpolating polynomial, and then try to plot its graph, and then a set of points. We will, we will do more examples of this kind using SageMath, right?

(Refer Slide Time: 11:33)



Now, let us look at another example of curve fitting. So, suppose we have set of points; let us look at a set of points.

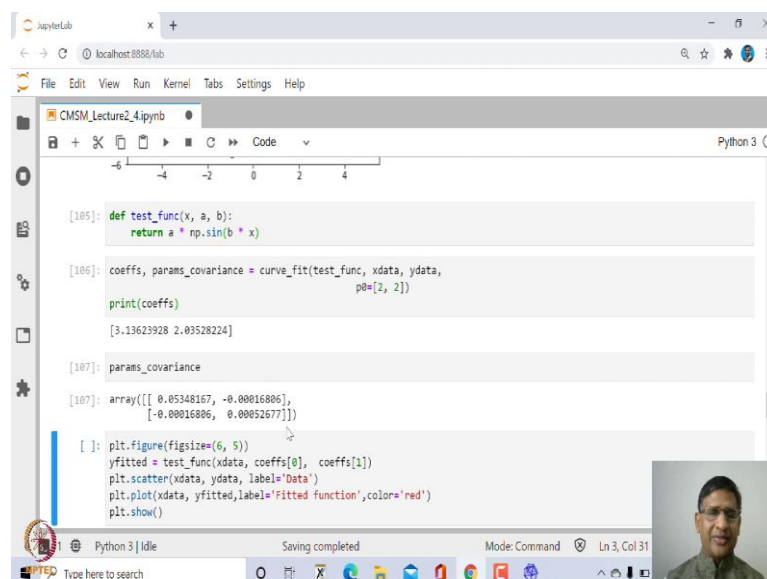
(Refer Slide Time: 11:47)



These are the set of points, some random points, about 50 random points, and inside this to, this set of points we want to fit, let us say, some curve.

So, first, let us import curve underscore fit from SciPy optimize.

(Refer Slide Time: 12:09)



And then let us create the function which we want to fit. So, in this case, let us say, we want to fit a function which is $\sin b x$ into $a \sin b x$.

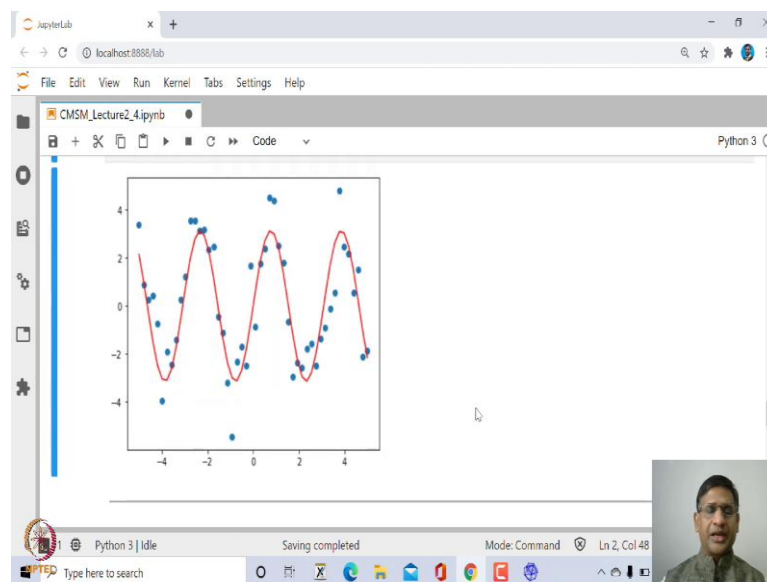
So, we want, need to find what should be value of a , and b which best fits in this set of points. So, first we will define this test function or you can call this as a model, which you want to fit, and then how do we make use of this.

So, use `curve_fit`, and give the test function, and then the x values, and y values of the data, and you also need to give the initial point; since it is a numeric, numerical method, you need to mention what are the initial guess values for these parameters a and b .

So, it returns two things; one, it will return approximate value of these parameters a and b , and it will also return covariance matrix of this parameter a and b , some kind of error errors, right? So, let us run this, and it finds the value of a as 3.1362, value of b as 2.035.

And you can also print the covariance matrix; so this is the covariance matrix of a and b , and then you can plot the set of points along with this fitted curve.

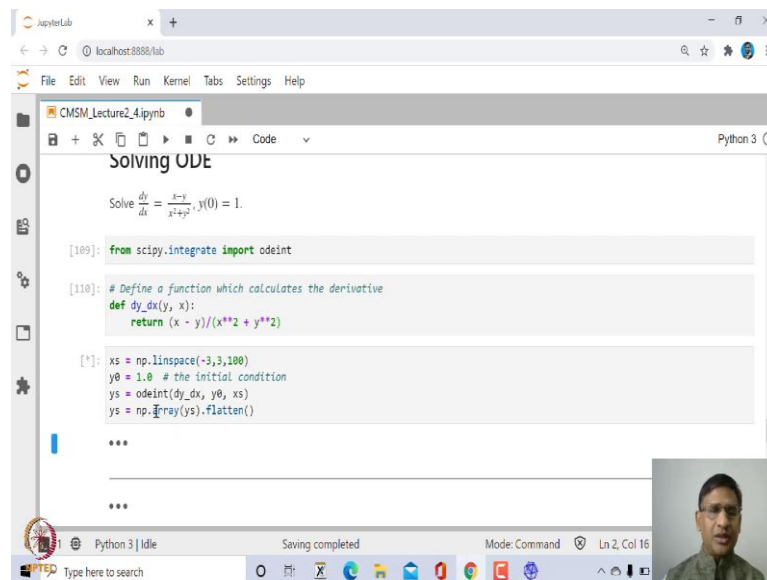
(Refer Slide Time: 13:37)



So, this is how you can do. So, you get, you are just using y fitted is equal to test function, and x data, y data, and the coefficient; you are mentioning is a coefficient 0 which is 'a' value, coefficient 1 which is 'b' value, and then scatter plot, and then this color is red, right?

So, this is how you can fit a curve. Instead of a into b's into sin b x; you could use for example, quadratic or you can use any other function. Of course, depending upon the complexity of the function, the error may vary, right?

(Refer Slide Time: 14:17)



```
Solving ODE

Solve  $\frac{dy}{dx} = \frac{x-y}{x^2+y^2}, y(0) = 1.$ 

[109]: from scipy.integrate import odeint

[110]: # Define a function which calculates the derivative
def dy_dx(y, x):
    return (x - y)/(x**2 + y**2)

[*]: xs = np.linspace(-3,3,100)
y0 = 1.0 # the initial condition
ys = odeint(dy_dx, y0, xs)
ys = np.array(ys).flatten()

***

***
```

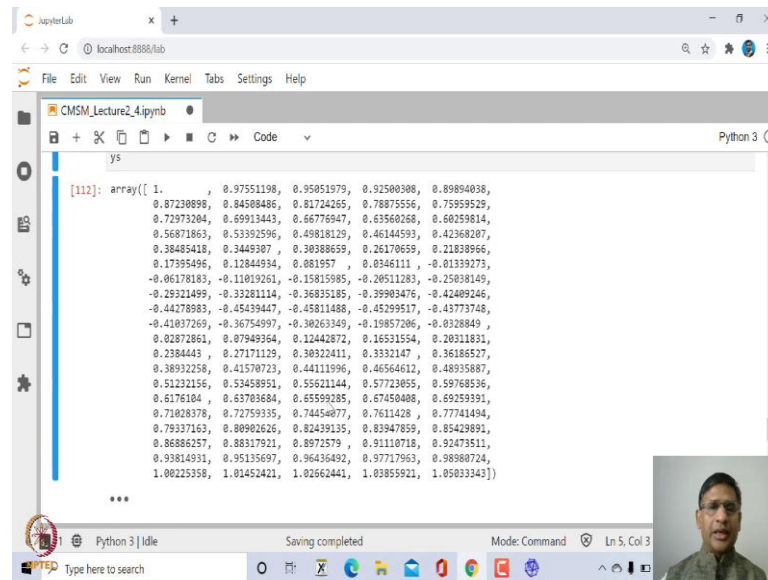
Similarly, let us look at a small example of solving initial value, a first-order differential equation using a SciPy. So, let us take an example. We want to solve dy by dx is equal to x minus y upon x square plus y square, y at 0 is equal to 1. So, how do we do that? Inside, again SciPy integrate, there is a module called ode int stands for initial value Ordinary Differential Equation.

So, let us import this module, and then first let us define the right-hand side of this dy by dx as a function, which I am calling as dy underscore dx. So, y comma x, and it returns x minus y upon x square plus y square. Once we have defined this function; then all we need to do is, we first need to create the set of points x.

So, we, we want to, let us say, create from minus 3 to 3, a 100 points between minus 3 and 3, and y 0 which is y0, which is y at x equal to 0, this is the initial condition, and

then let us call this function ode int; this you need to give the right-hand side of this dy/dx , and then initial guess values, and the points at which it wants to evaluate, those are the x values between minus 3, and 3, and this is what you get, it returns the approximate value of y.

(Refer Slide Time: 15:57)



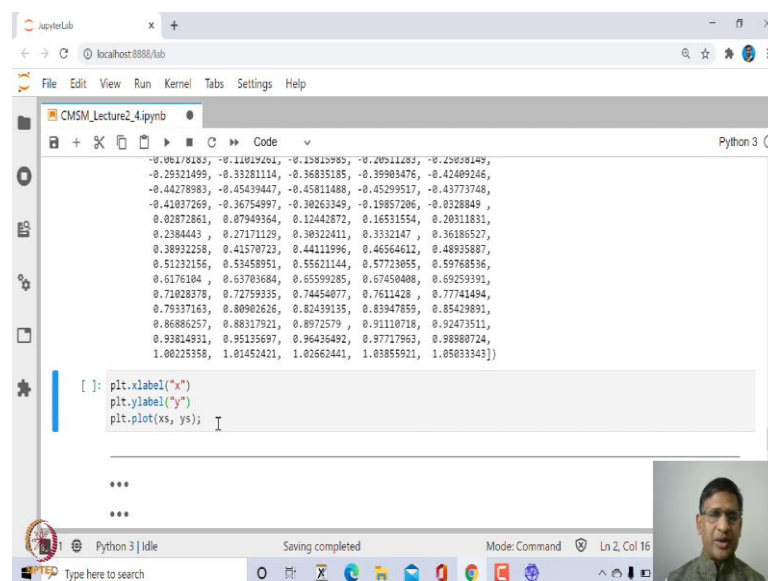
```

[112]: array([[ 1.          ,  0.97551198,  0.95851979,  0.92500308,  0.89894038,
  0.87230898,  0.84580486,  0.81724265,  0.78775556,  0.75959529,
  0.72973284,  0.69913443,  0.66776947,  0.63560268,  0.60259814,
  0.56871863,  0.53392596,  0.49818129,  0.46144593,  0.42368287,
  0.38485418,  0.3445987 ,  0.30388659,  0.26170659,  0.21838966,
  0.1795496 ,  0.12844934,  0.081957 ,  0.0346111 , -0.01339273,
 -0.06178183, -0.11019261, -0.15815985, -0.20511283, -0.25038149,
 -0.29321499, -0.33281114, -0.36835185, -0.39903476, -0.42489246,
 -0.44278983, -0.45439447, -0.45811488, -0.45299517, -0.43773748,
 -0.41837269, -0.36754997, -0.30263349, -0.19857206, -0.0328849 ,
  0.02872861,  0.07949364,  0.12442872,  0.16531554,  0.20311831,
  0.2384443 ,  0.27171129,  0.30322411,  0.3332147 ,  0.36186527,
  0.38932258,  0.41570723,  0.44111996,  0.46564612,  0.48935887,
  0.51232156,  0.53458951,  0.55621144,  0.57723055,  0.59768536,
  0.6176104 ,  0.63783684,  0.65599285,  0.67450408,  0.69259391,
  0.71028378,  0.72759335,  0.74454077,  0.7611428 ,  0.77741494,
  0.79337163,  0.80902626,  0.82439135,  0.83947859,  0.85429891,
  0.86886257,  0.88317921,  0.8972579 ,  0.91118718,  0.92473511,
  0.93814931,  0.95135697,  0.96436492,  0.97717963,  0.98980724,
  1.00225358,  1.01452421,  1.02662441,  1.03855921,  1.05033343]])

```

And let us, let us look at what are these values y's. So, this is a, this is the y values. So, at minus 3, the y value is 1, and so on, right?

(Refer Slide Time: 16:19)



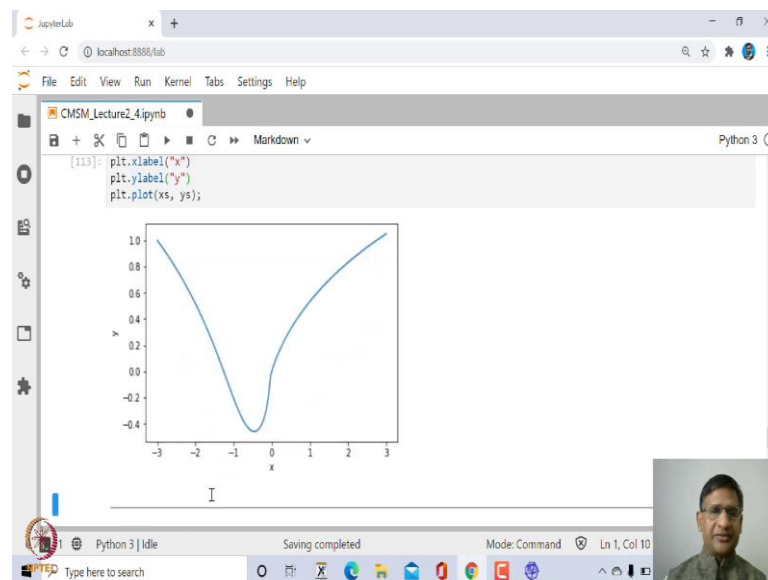
```

[ ]: plt.xlabel("x")
plt.ylabel("y")
plt.plot(xs, ys)

```

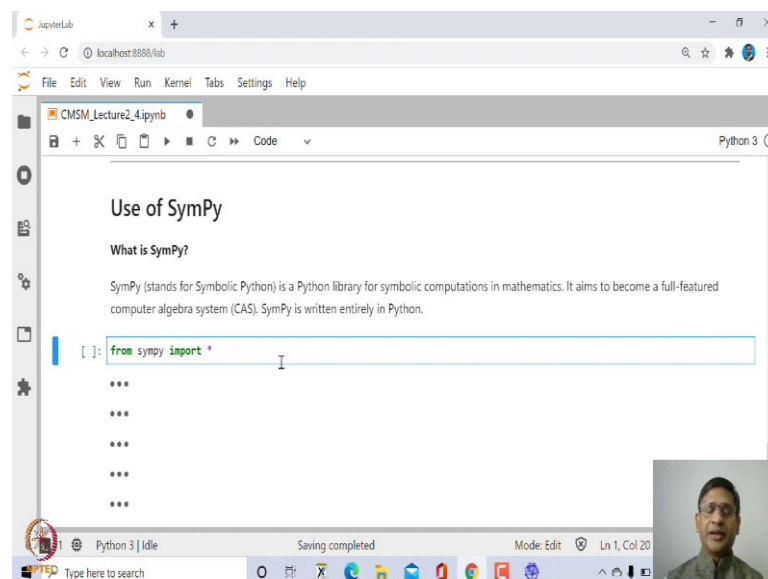
And next let us look at, let us try to plot graph of this solution curve. So, we can say, plt dot plot x label is equal to x, plt dot plot y label is equal to y, and then plt dot plot xs comma ys. What is xs? Xs is the points between minus 3 and 3, 100 points, and ys is the approximate value of the, the function. So, let us plot its graph.

(Refer Slide Time: 16:41)



So, you can see here this is how the solution curves, solution curve looks like, right?

(Refer Slide Time: 16:51)



And next, let us look at how we can make use of SymPy. So, as I, as I said, the SciPy has many more facilities in order to do numerical computations.

So, you can take help on this SciPy or you can go through help document from its official website, and see what are the other scientific computations or numerical computations which you come across in numerical analysis can be explored using SciPy.

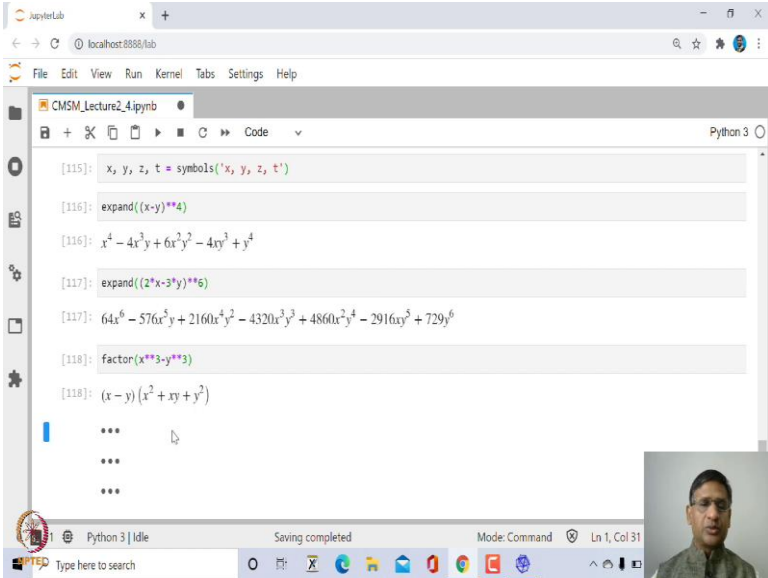
So, let us look at now briefly some facilities from SymPy. What is SymPy? So, SymPy actually stands for symbolic Python, and it is a Python library for symbolic manipulations in mathematics. In mathematics, we keep doing symbolic manipulations; we generally play with symbols.

And this SymPy actually aims to become full-featured computer algebra system, and it actually, it is written completely in Python, and SageMath also uses SymPy for many of the symbolic computations behind the scene, right?

So, first let us import SymPy, all the functions inside SymPy; though I have mentioned that this is usually not a good idea to import all the functions from a module, because there could be thousands of functions, and then in that case all these functions will be sitting in the memory, it may slow down your computer.

But in any case, however, let us just import all these functions from SymPy. You can take help on SymPy, and go through the functions which are available.

(Refer Slide Time: 18:31)



The screenshot shows a JupyterLab window with a file named 'CMSM_Lecture2_4.ipynb'. The code cell contains the following Python code:

```
[115]: x, y, z, t = symbols('x, y, z, t')
[116]: expand((x-y)**4)
[116]: x4 - 4x3y + 6x2y2 - 4xy3 + y4
[117]: expand((2*x-3*y)**6)
[117]: 64x6 - 576x5y + 2160x4y2 - 4320x3y3 + 4860x2y4 - 2916xy5 + 729y6
[118]: factor(x**3-y**3)
[118]: (x-y)(x2+xy+y2)
***
***
***
```

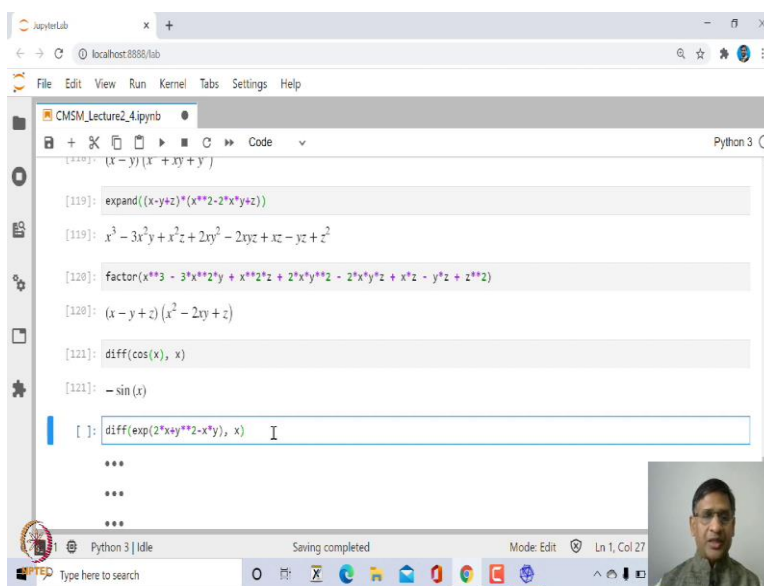
The output of the code is displayed in a separate cell, showing the expanded and factored expressions. The JupyterLab interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a status bar at the bottom indicating 'Python 3 | Idle' and 'Mode: Command'.

Now, let us create x, y, z, t as symbols. So, you can use symbols. So, that is, the symbols is a function inside SymPy, and x, y, z you need to mention inside single quote or double

quote, separated by comma. You need not to give comma, even without comma it will also work. So, let us run this, and let us expand x minus y to the power 4.

So, this is binomial expansion of x minus y to the power 4; this is what you get x power 4, minus 4 x cube y, plus 6 x square y square, minus 4 x y cube, and so on. Similarly, you can expand 2 x minus 3 y to the power 6, this is what you get; you can also factor, factorize, let us say, x cube minus y cube. So, this is factor of x cube minus y cube, is x minus y into x square plus x y plus y square.

(Refer Slide Time: 19:29)



The screenshot shows a JupyterLab window with a Python 3 kernel. The notebook file is named 'CMSM_Lecture2_4.ipynb'. The code cell contains the following:

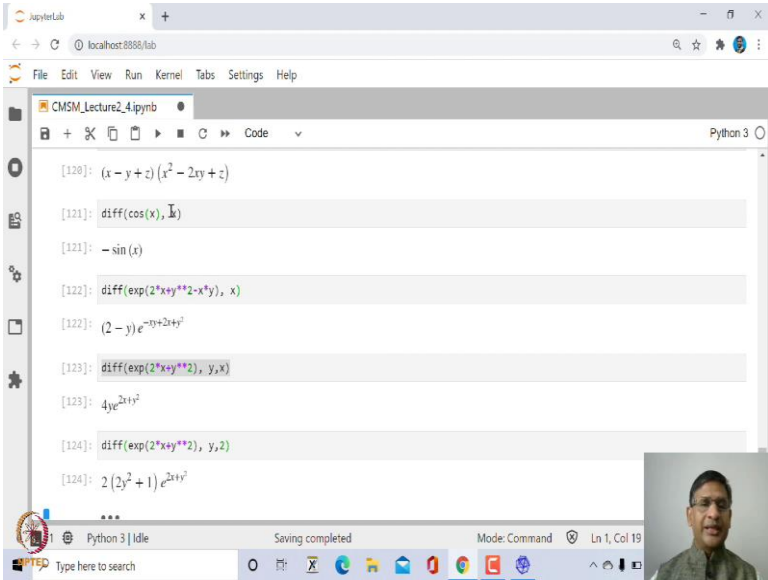
```
(x-y)*(x+xy+y)  
  
[119]: expand((x-y+z)*(x**2-2*x*y+z))  
[119]: x3 - 3x2y + x2z + 2xy2 - 2xyz + xz - yz + z2  
  
[120]: factor(x**3 - 3*x**2*y + x**2*z + 2*x*y**2 - 2*x*y*z + x*z - y*z + z**2)  
[120]: (x - y + z) (x2 - 2yz + z)  
  
[121]: diff(cos(x), x)  
[121]: -sin(x)  
  
[ ]: diff(exp(2*x*y**2-x*y), x)  
***  
***  
***
```

The output of the last cell is three asterisks, indicating it was not executed. The bottom status bar shows 'Python 3 | Idle', 'Saving completed', and 'Mode: Edit | Ln 1, Col 27'. A small video feed of a man is visible in the bottom right corner.

Or you can even expand some product of two polynomials in x , y , z . So, this is what you get, and if you try to factorize this, if you try to factorize this; then you should get the original factors, which is x minus y plus z into x square minus $2xy$ plus z , right?

So, you can also find derivative of a function. So, if you want to find derivative of $\cos x$; so you just use `diff cos x`, with respect to x . If you find a, if you have a function of two variables, let us say, exponential of $2x$ plus y square minus xy , and if you want to find the partiality of this function with respect to x ; again you can make use of same `diff` from SymPy, right?

(Refer Slide Time: 20:19)



```
[120]: (x - y + z) (x2 - 2xy + z)

[121]: diff(cos(x), x)
[121]: -sin(x)

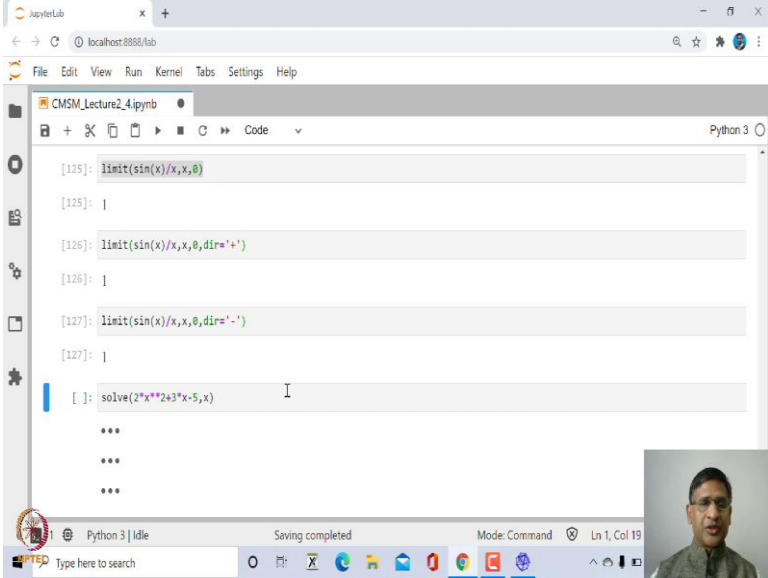
[122]: diff(exp(2*x*y**2-x*y), x)
[122]: (2 - y) e-y+2xy2

[123]: diff(exp(2*x*y**2), y,x)
[123]: 4ye2xy2

[124]: diff(exp(2*x*y**2), y,2)
[124]: 2 (2y2 + 1) e2xy2
```

And this, this gives you first-order partiality of this function with respect to x. You can find mixed second-order partial derivative. So, for example, first differentiate with respect to y, and then differentiate with respect to x, this is mixed second-order partial derivatives.

Or if you want to find second-order partial derivative with respect to y twice; then you can just say, y comma 2, this is second-order partial derivative of this function with respect to y, twice. So, you can, you can see here diff is for finding derivative, and it can be used for finding derivative of one variable function as well as multivariable function. (Refer Slide Time: 21:01)



The screenshot shows a JupyterLab window with a browser address bar at localhost:8888/lab. The notebook has a file named CMSM_Lecture2_4.ipynb. The code cells contain the following:

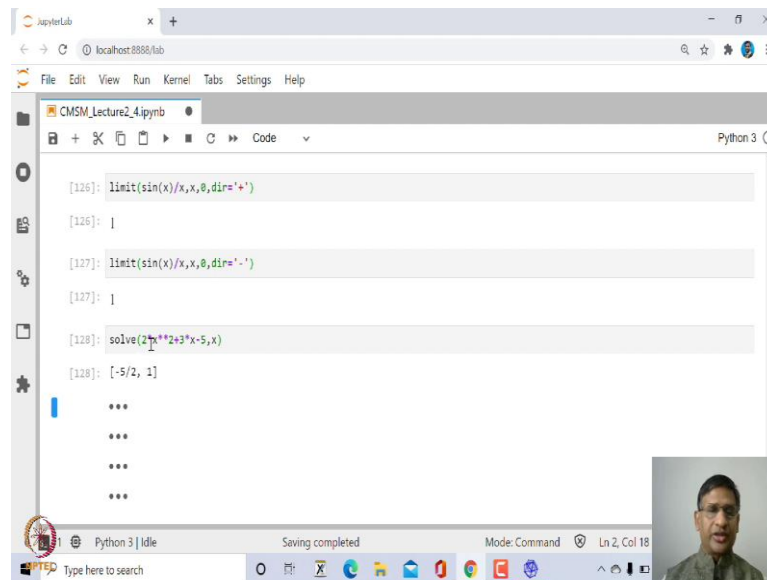
```
[125]: limit(sin(x)/x,x,0)
[125]: 1
[126]: limit(sin(x)/x,x,0,dir='+')
[126]: 1
[127]: limit(sin(x)/x,x,0,dir='-')
[127]: 1
[ ]: solve(2*x**2+3*x-5,x)
***
***
***
```

The bottom status bar shows 'Python 3 | Idle', 'Saving completed', 'Mode: Command', and 'Ln 1, Col 19'. A small video inset of a man is visible in the bottom right corner.

You can also find limit of a function. So, for example, limit of sin x by x with respect to x, and at x equal to 0 is 1; this is quite popular, and this is quite useful, and similarly, you can find left-hand derivative, and right-hand derivative. So, if you want to find right-hand derivative, inside this limit; you use an option dir dir equal to plus sign inside single quote.

Or you can find left-hand derivative; so instead of plus, if you say, minus, it will give you left-hand derivative. You can even find root of f(x) equal to 0. Of course, it may not give you root for everything, but wherever it is possible, it will solve.

(Refer Slide Time: 21:53)



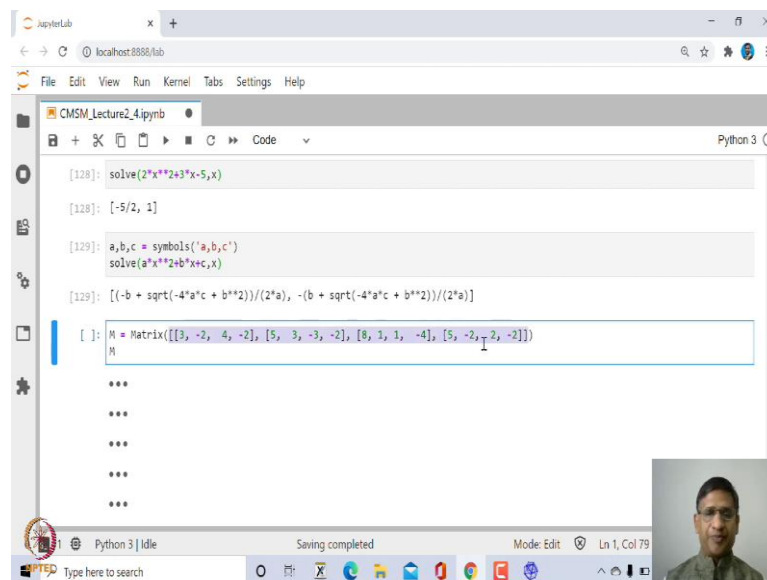
The screenshot shows a JupyterLab window with a file named 'CMSM_Lecture2_4.ipynb'. The code cell contains the following Python code:

```
[126]: limit(sin(x)/x,x,0,dir='+')
[126]: 1
[127]: limit(sin(x)/x,x,0,dir='-')
[127]: 1
[128]: solve(2*x**2+3*x-5,x)
[128]: [-5/2, 1]
```

The output shows the results of the limit and solve functions. The bottom status bar indicates 'Python 3 | Idle' and 'Mode: Command'.

So, this uses symbolic manipulations. So, the roots of this equation $2x^2 + 3x - 5 = 0$ are minus 5 by 2, and 1.

(Refer Slide Time: 22:03)



The screenshot shows a JupyterLab window with a file named 'CMSM_Lecture2_4.ipynb'. The code cell contains the following Python code:

```
[128]: solve(2*x**2+3*x-5,x)
[128]: [-5/2, 1]
[129]: a,b,c = symbols('a,b,c')
[129]: solve(a*x**2+b*x+c,x)
[129]: [(-b + sqrt(-4*a*c + b**2))/(2*a), -(b + sqrt(-4*a*c + b**2))/(2*a)]
[ ]: M = Matrix([[3, -2, 4, -2], [5, 3, -3, -2], [8, 1, 1, -4], [5, -2, 2, -2]])
[ ]: M
```

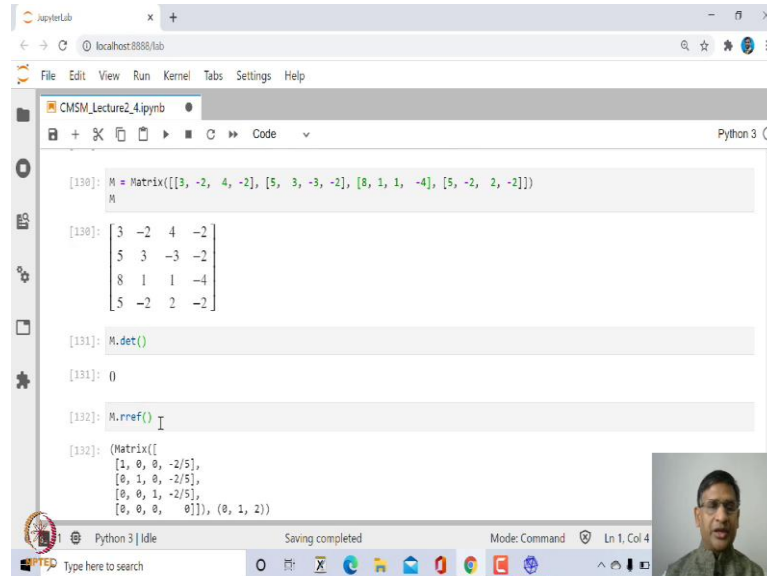
The output shows the results of the solve functions and the definition of the matrix M. The bottom status bar indicates 'Python 3 | Idle' and 'Mode: Edit'.

You can, you can even find the roots of a quadratic $ax^2 + bx + c$, where a, b, c are symbols. So, that it will give you roots as $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

So, that is what you get, right? You can define the matrix. So, inside SymPy; there is a function called a capital 'M a t r i x', Matrix, and define the matrix by giving its rows in

inside square bracket. So, this is very similar to what we have done in case of a NumPy array.

(Refer Slide Time: 22:37)



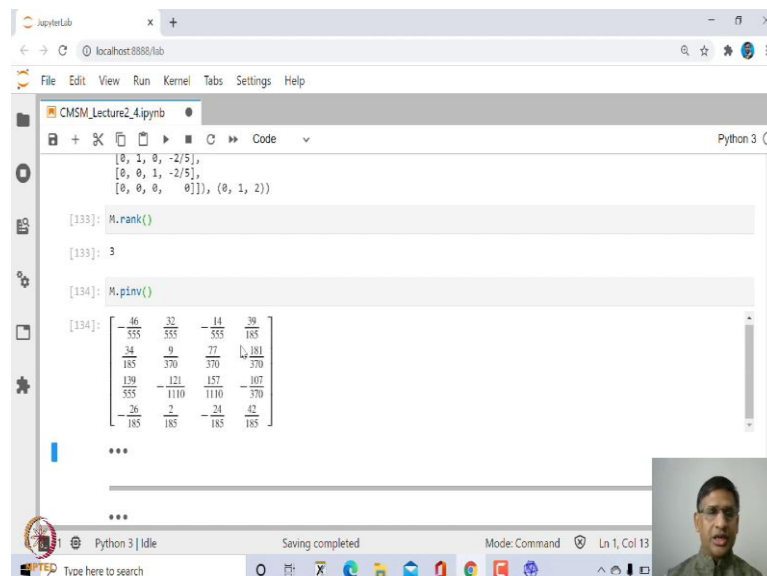
The JupyterLab interface shows a Python 3 kernel with the following code and output:

```
[190]: M = Matrix([[3, -2, 4, -2], [5, 3, -3, -2], [8, 1, 1, -4], [5, -2, 2, -2]])
M
[190]: 
$$\begin{bmatrix} 3 & -2 & 4 & -2 \\ 5 & 3 & -3 & -2 \\ 8 & 1 & 1 & -4 \\ 5 & -2 & 2 & -2 \end{bmatrix}$$

[191]: M.det()
[191]: 0
[192]: M.rref()
[192]: (Matrix([
[1, 0, 0, -2/5],
[0, 1, 0, -2/5],
[0, 0, 1, -2/5],
[0, 0, 0, 0]]), (0, 1, 2))
```

So, let us run this, and you can find various concepts related to this matrix. So, for example, you can find determinant of this matrix using det function; you can find reduced row echelon forms form of M. So, it, it has three non zero rows, and the last row in this case is 0, it gives you the pivot elements 0, 1, 2; that means first column, second column, and third column are pivot columns, right?

(Refer Slide Time: 23:11)



The JupyterLab interface shows the following code and output:

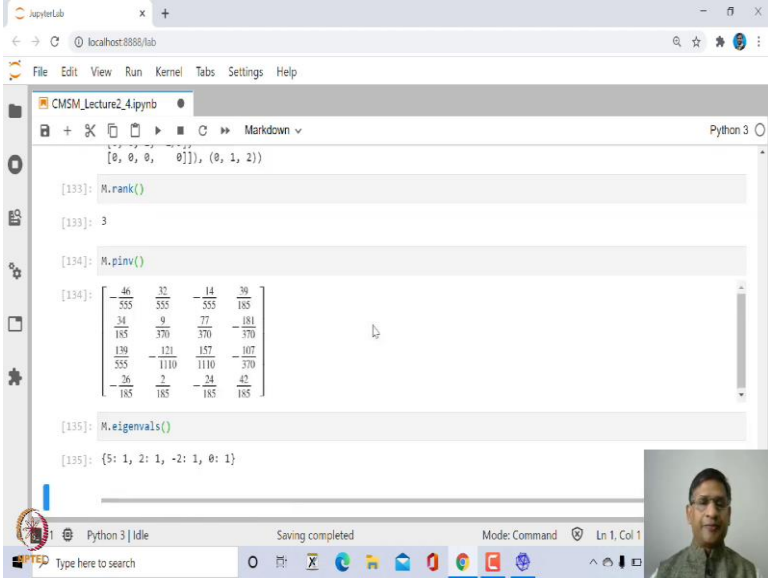
```
[193]: M.rank()
[193]: 3
[194]: M.pinv()
[194]: 
$$\begin{bmatrix} -46 & 32 & -14 & 39 \\ 535 & 535 & -535 & 185 \\ 24 & -9 & 77 & 181 \\ 185 & 570 & 370 & 370 \\ 139 & -121 & 357 & -107 \\ 535 & -1110 & 1110 & -370 \\ -26 & 2 & -24 & 42 \\ 185 & 185 & 185 & 185 \end{bmatrix}$$

...
...
...
```

You can even find rank of this matrix, and you can see here this rref of M has 3 non-zero rows; so that means rank of this M is 3. You can also find in, since this matrix has rank 3 which is strictly less than the dimension of this matrix which is 4, it will not be invertible; but what you can do is, you can find generalized inverse.

So, pinv is called generalized inverse of this matrix. So, in case you have a square matrix, or a matrix which is not invertible; you can find its generalized inverse. We will look at this concept as an application to singular value decomposition, right?

(Refer Slide Time: 23:49)



The screenshot shows a JupyterLab window with a file named 'CMSM_Lecture2_4.ipynb'. The code cell contains the following Python code and its output:

```
[0, 0, 0, 0]], (0, 1, 2))

[133]: M.rank()
[133]: 3

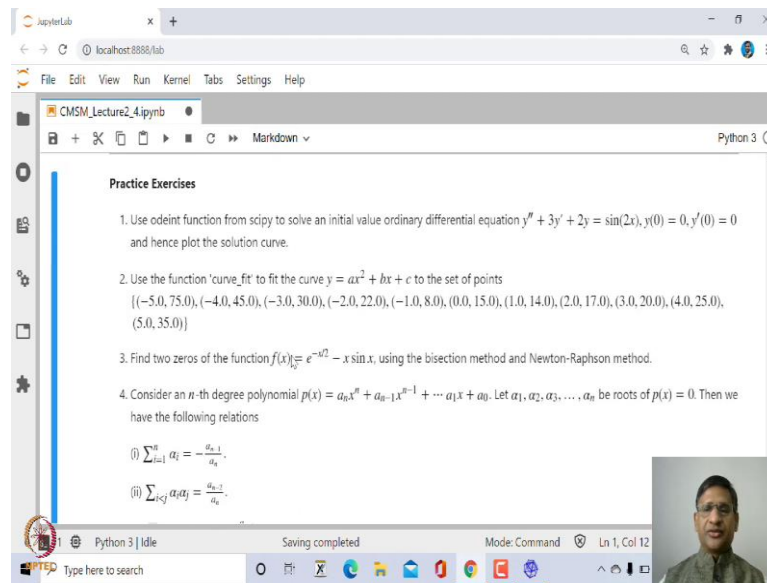
[134]: M.pinv()
[134]: 
$$\begin{bmatrix} -\frac{46}{555} & \frac{32}{555} & -\frac{14}{555} & \frac{39}{185} \\ \frac{34}{185} & -\frac{9}{370} & \frac{77}{370} & -\frac{181}{370} \\ \frac{139}{555} & -\frac{121}{1110} & \frac{157}{1110} & -\frac{107}{370} \\ -\frac{26}{185} & \frac{2}{185} & -\frac{24}{185} & \frac{43}{185} \end{bmatrix}$$


[135]: M.eigenvals()
[135]: {5: 1, 2: 1, -2: 1, 0: 1}
```

The output of `M.pinv()` is a 4x4 matrix of fractions. The output of `M.eigenvals()` is a dictionary showing eigenvalues and their multiplicities: 5 (multiplicity 1), 2 (multiplicity 1), -2 (multiplicity 1), and 0 (multiplicity 1).

You can also find eigenvalues of this matrix. So, in this case, there are now 4 eigenvalues; one is 5 of multiplicity 1, 2 of multiplicity 1, minus 2 of multiplicity 1, and 0 of multiplicity 1. So, you can explore various concepts related to matrix computation, and inside SymPy.

(Refer Slide Time: 24:17)



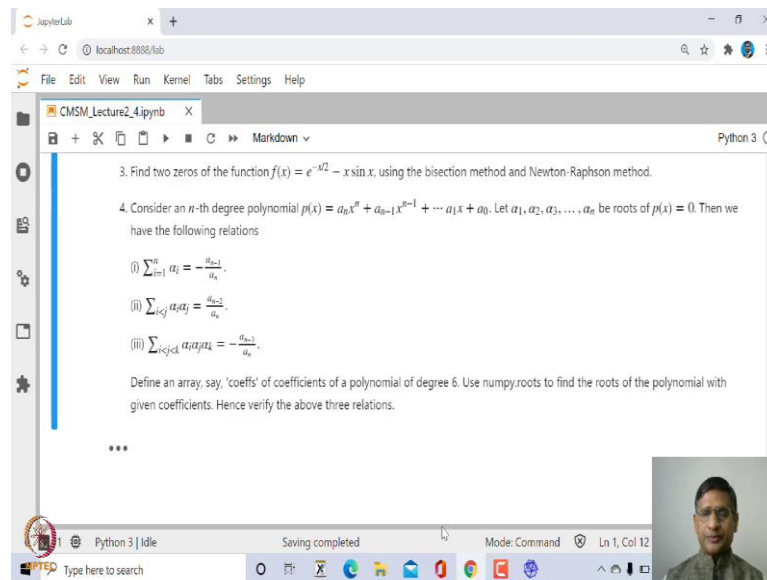
Now, let me leave you with a few simple exercises. So, 1st one is to solve this initial value problem, second-order initial value problem using, using SciPy ODE solver, and the 2nd one is to fit quadratic x square plus $b x$ plus c to this set of points. The 3rd exercise is to find zeros of this function using bisection method and Newton Raphson method.

And the 4th exercise is, if you look at a polynomial $p(x)$ which is a $n x$ to the power n , plus a n minus $1 x$ to the power n minus 1 , plus, dot dot dot, a $1 x$ plus a 0 , where, and let us assume that $\alpha_1, \alpha_2, \dots, \alpha_n$, are roots of this $p x$ equal to 0 ; then we, we, we have relationship between the coefficients the, the roots, and the coefficients.

So, let us look at some three properties. So, first property is to, to, it says that, sum of the roots is nothing but minus a , and minus 1 upon a n , that is the coefficient of x to the power n minus 1 , divided by coefficient of x to the power n .

And the second one is, sum of the product of the roots is nothing, but a n minus 2 divided by a n , and the sum of the products of three different roots is given by minus a n minus 3 upon a n .

(Refer Slide Time: 25:35)



So, your, your task is to take some arbitrary polynomial, and find its roots.

So, inside, inside SymPy inside NumPy, there is a function called roots. So, that will give you roots of all these polynomial, and then try to write a small Python code in order to find the sum of the roots, product of two roots, and product of three roots, and then verify these relations. One can extend these to product of four root, products of five roots, and product of n roots also, ok? So, let me stop here.

Thank you very much.