

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

Lecture - 47
Interpolations in SageMath

(Refer Slide Time: 00:15)

Interpolations

Lagrange Interpolation

Let $\{(x_i, y_i)\}_{i=0}^n$ be $n+1$ points where $y_i = f(x_i)$ and x_i may not be equally placed. We want to fit a polynomial $P_n(x)$ of degree $\leq n$ passing through above points.

We can write

$$P_n(x) = l_0(x)y_0 + l_1(x)y_1 + \dots + l_n(x)y_n,$$

where $l_i(x)$ are polynomials of degree n . Since $P_n(x_i) = y_i$ for all i it is easy to check that

$$l_i(x_j) = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j. \end{cases}$$

Since $l_i(x) = 0$ for x_0, x_1, \dots, x_n we can write $l_i(x) = A(x-x_0)(x-x_1)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)$. It is easy to find the constant A and show that

$$l_i(x) = \frac{(x-x_0)(x-x_1)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)(x_i-x_1)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)}$$

Substituting $l_i(x)$ in the above equation we get the Lagrange interpolation polynomial.

Welcome to the 47th lecture on Computational Mathematics with SageMath. In this lecture, we shall explore some methods of Interpolations. So, let us start with Lagrange interpolations. So, what is Lagrange interpolations method? So, in case you are given, let us say, set of $n+1$ points in the plane, where y_i is, let us say, value of a function f of x_i , and this x_i may or may not be equally placed.

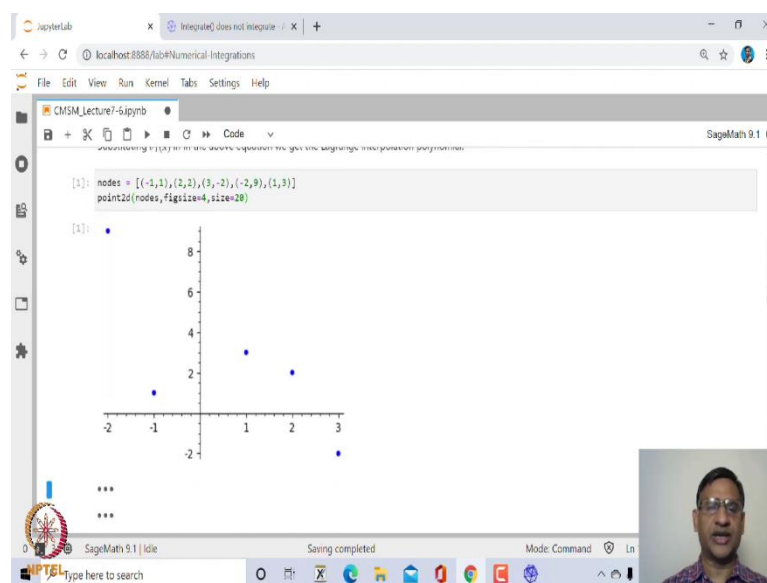
Then we want to, what we want? We, one can show that then there is just a unique n th degree polynomial passing through these $n+1$ points. And this polynomial P_n , let us say, call this polynomial P_n , is given by $l_0(x)y_0 + l_1(x)y_1 + \dots + l_n(x)y_n$, where $l_i(x)$ will be 0 if i is not equal to j , and is equal to 1 if i is equal to j .

So, one can show that, or one can find that l_i is given by this formula, which is product of $(x-x_0)*(x-x_1)*\dots*(x-x_{i-1})*(x-x_{i+1})*\dots*(x-x_n)$. So, numerator is product of x minus x

j , i not equal to j , and denominator is $(x_i - x_0) \cdot (x_i - x_1) \cdot \dots \cdot (x_i - x_n)$. So, the product will be x_i minus x_j , when i is not equal to j .

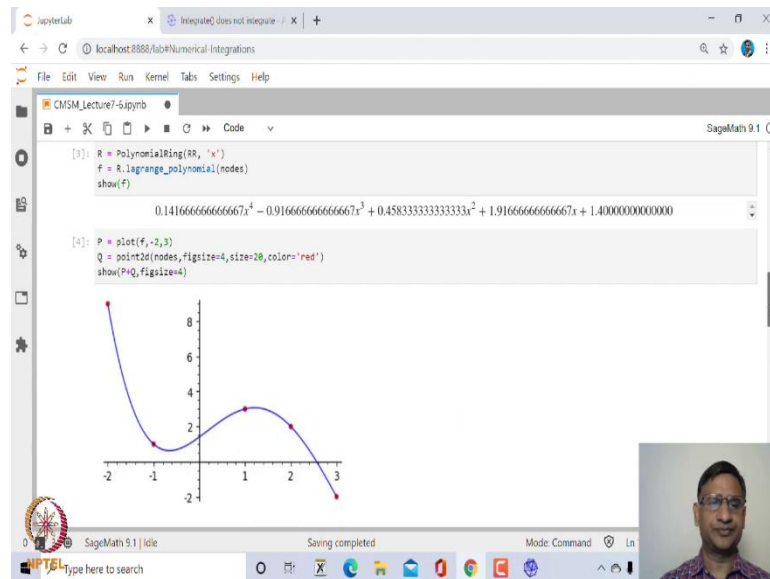
So, that is L_i . So, this is what is, any interpolating polynomial like this is known as Lagrange interpolating polynomial. So, let us see how we can create this Lagrange interpolating polynomial; it should be quite easy to write a program, Sage program that finds this Lagrange interpolating polynomial.

(Refer Slide Time: 02:22)



However, we will use inbuilt methods in Sage. So, let us start with, we have these five points minus $(1, 1)$, $(2, 2)$, $(3, -2)$, $(-2, 9)$ and $(1, 3)$, and let us plot these points. So, these are the set of five points, to these five points, we want to fit a Lagrange interpolating polynomial. So, here since there are five points, the Lagrange interpolating polynomial will be of degree 4.

(Refer Slide Time: 03:02)



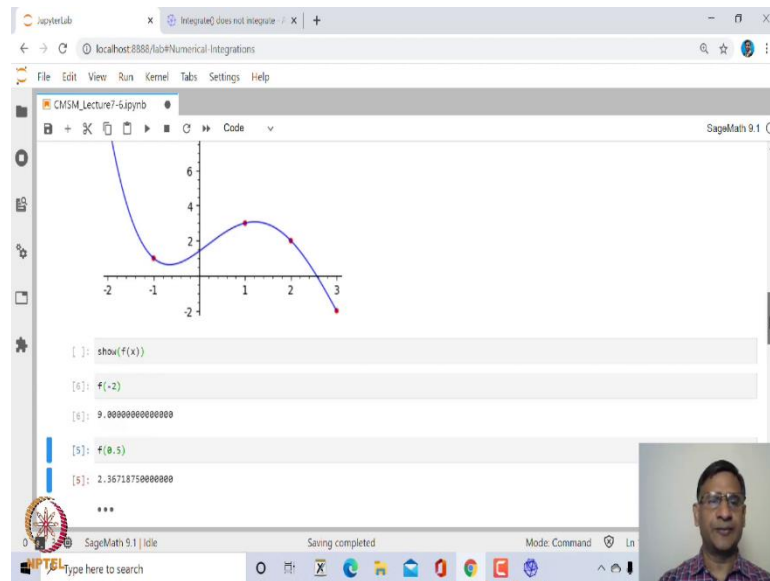
Now, how do we do that? So, first define R to be PolynomialRing over this x, PolynomialRing over R, and with symbol x. This is what is generally, we call R x. And then the Sage has inbuilt method over this R, called Lagrange, Lagrange underscore polynomial, and that you fit it to these nodes, which are these five points.

Now, if you look at what is this; what is this f; let me ask you to show what is this f, this is the interpolating polynomial you have got, these are the coefficients. So, this is coefficient of x power 4, this is coefficient of x power cube, and so on, so that, it has inbuilt method to find interpolating polynomial.

Now, let us plot the graph of this interpolating polynomial along with these set of points. So, we simply add this plot f between minus 2 and 3. So, you can see here this, right, the x coordinate of these points lies between minus 2 and 3. So, let us plot this, and we will plot points in red color, and the curve in blue color. So, that is the Lagrange interpolation polynomial along with the set of points.

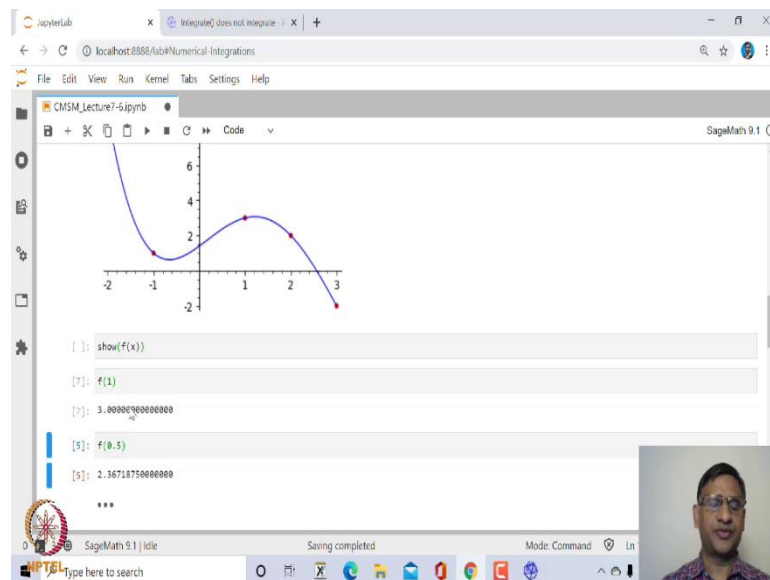
And you can extend this to any finitely many points. Of course, if you have large number of points, then it will take lot more time, right, ok? So, this we have already done.

(Refer Slide Time: 04:37)



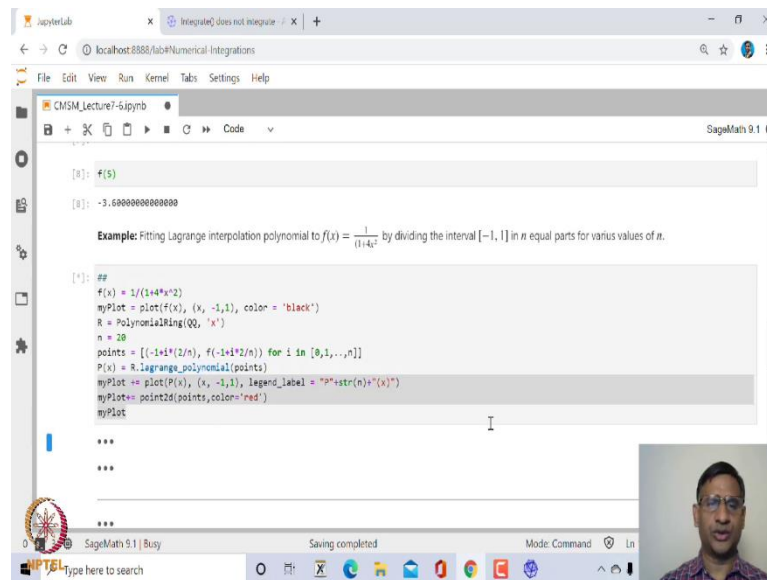
Next, you can evaluate value of f at any point, intermediate point between minus 3 and 2. So, f at 0.5 is this; if you evaluate, let us say, for example, f at node point. So, for example, f at minus 2, f at minus 2 is 9. And what was the f at minus 2? That is also 9, minus 2 is 9; f at 1 is 3.

(Refer Slide Time: 05:15)



So, for example, if I say f at 1 is equal to, f at 1, then it will be 3. So, this polynomial passes through all these points. So that means, if I, the p , if I call this polynomial as f , then f of x is y i, right?

(Refer Slide Time: 05:39)



```
[1]: f(5)
[2]: -3.600000000000000

Example: Fitting Lagrange interpolation polynomial to  $f(x) = \frac{1}{(1+4x^2)}$  by dividing the interval  $[-1, 1]$  in  $n$  equal parts for various values of  $n$ .

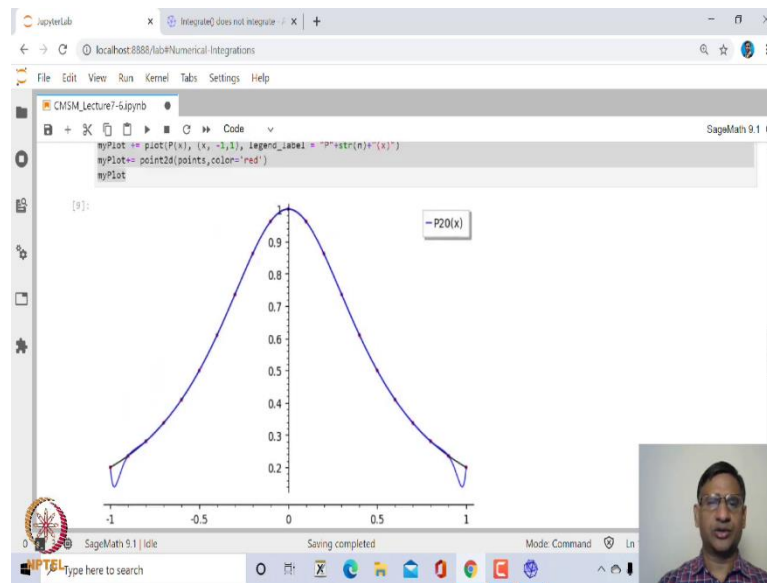
[3]: ##
f(x) = 1/(1+4*x^2)
myPlot = plot(f(x), (x, -1,1), color = 'black')
R = PolynomialRing(QQ, 'x')
n = 20
points = [(-1+i*(2/n), f(-1+i*(2/n))) for i in [0,1,...,n]]
P(x) = R.Lagrange_polynomial(points)
myPlot += plot(P(x), (x, -1,1), legend_label = "P"+str(n)+"(x)")
myPlot += point2d(points,color='red')
myPlot

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

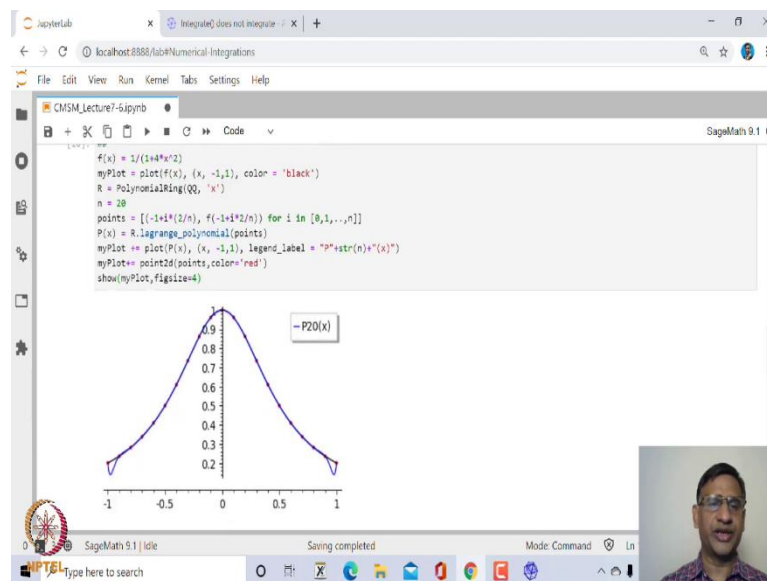
And similarly, you can, you can even find out f , the value of f at, let us say for example, 5, it will give you, of course the question of accuracy, the, it may not be all that accurate, ok? Similarly, let us look at, suppose we have a function $f(x)$ is equal to 1 upon 1 plus 4 x square, and let us say, divide this into n equal parts for various values of n , and let us find out Lagrange interpolating polynomial to this.

So, how do we do that? We can take $f(x)$ equal to 1 upon 1 plus 4 x square and then let us plot this function first, and then again define R to be polynomial ring over QQ , or over RR . And let us take, we have 20 points. So, plot these 20 points x_i and y_i ; y_i is f at x_i , and then fit, find out this polynomial which is P_x , I am calling this as P_x , and then plot the points along with the given set of polynomials. So, let us see how it looks like. Yeah, so, this is how it looks like.

(Refer Slide Time: 06:45)



(Refer Slide Time: 06:52)

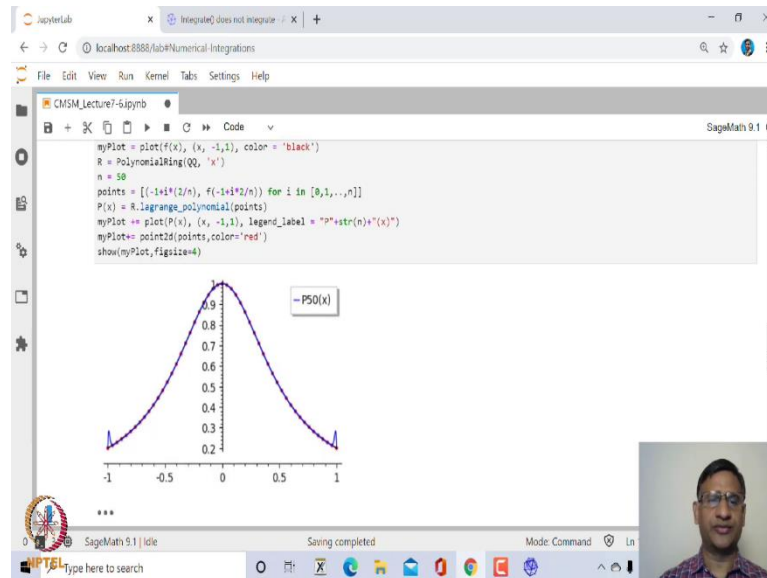


Let me, let me reduce the figure size, show this with fig size is equal to, let us say 4. So, this is what you get, this is a, so, you can see here, I mean this fitted polynomial would be quite close to the actual graph of the function.

So, in this case, these are the set of points and the fitted polynomial. You could also plot graph of this, the function f of x . So, let me, let me plot, let us say, yeah, this is already plotted, this is in black color, and this fitted polynomial in, in blue color and you can see

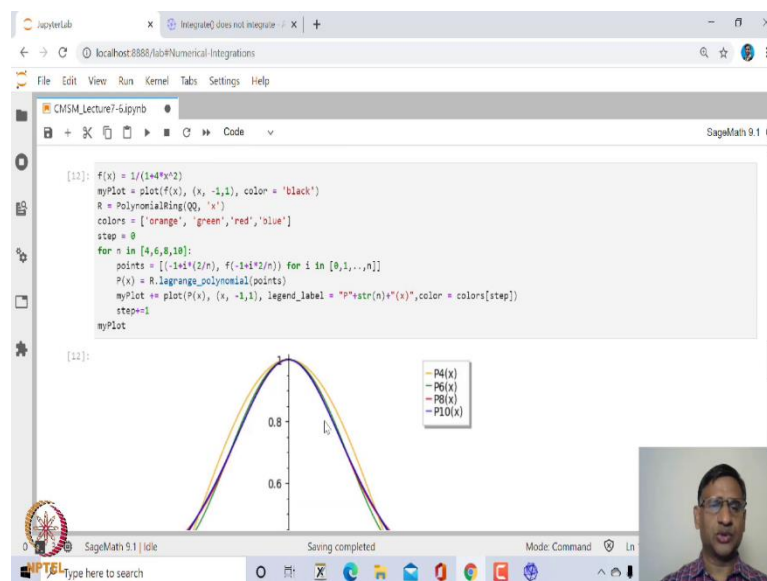
here, both are almost close to each other, except somewhere here, which is not all that close.

(Refer Slide Time: 07:47)



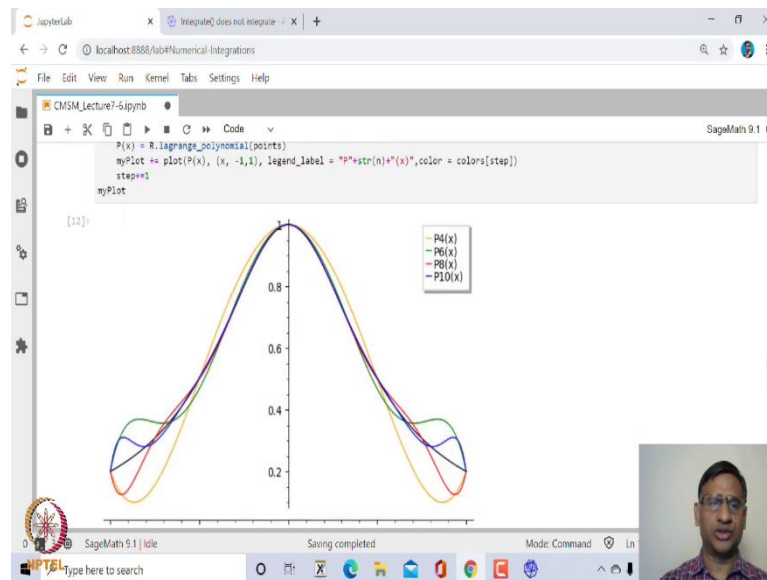
So, if you increase the number of points, for example, if I increase instead of 20, let us make it to 50, and then it will be much more closer actually, right? So, one can find out what is the error term.

(Refer Slide Time: 07:56)



You could also create, for example, plot several interpolating polynomials together.

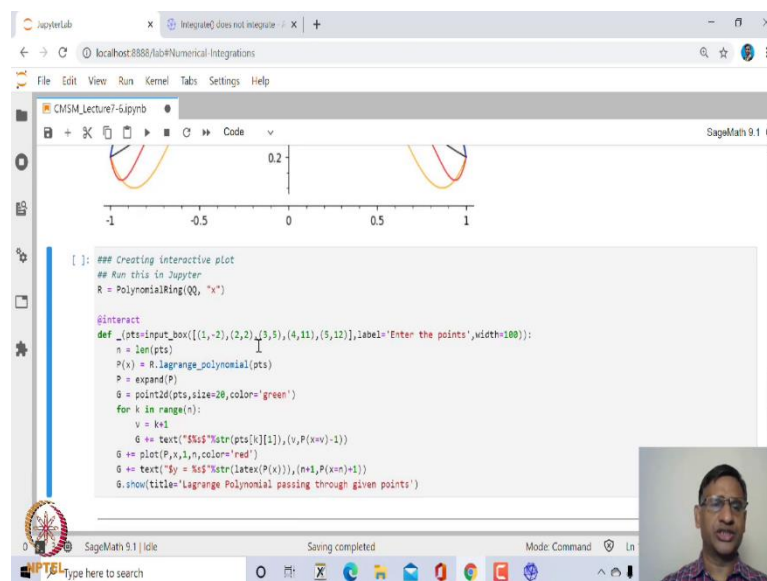
(Refer Slide Time: 08:18)



So, this is just, we are plotting inter, interpolating polynomial by dividing this interval minus 1 to 1 in 4 parts, in 6 parts, in 8 parts, and in 10 parts.

So, all these together you can also plot. So, this, this is just to demonstrate that in case you increase the degree of this interpolating polynomial, then this approximation is much better than, much better, or it is very close to the actual graph.

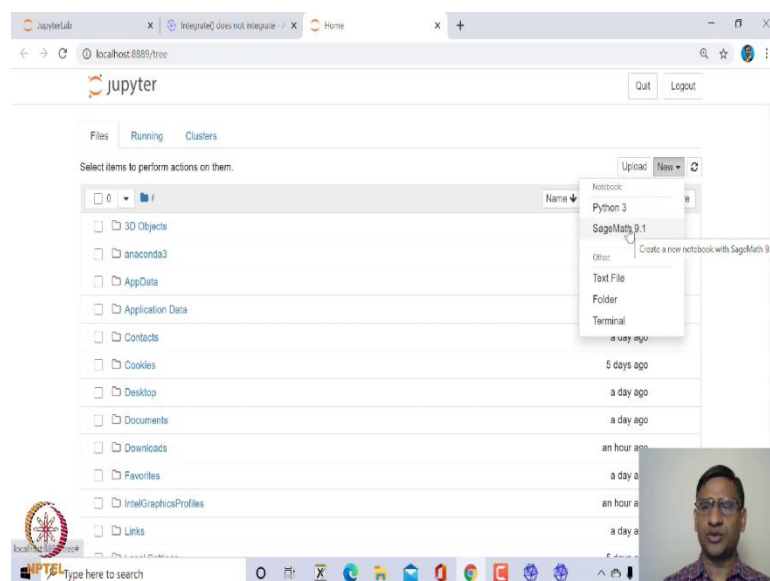
(Refer Slide Time: 08:36)



So, what I will do? I will run this in Jupyter. So, let me go to my desktop, and let me start Sage notebook.

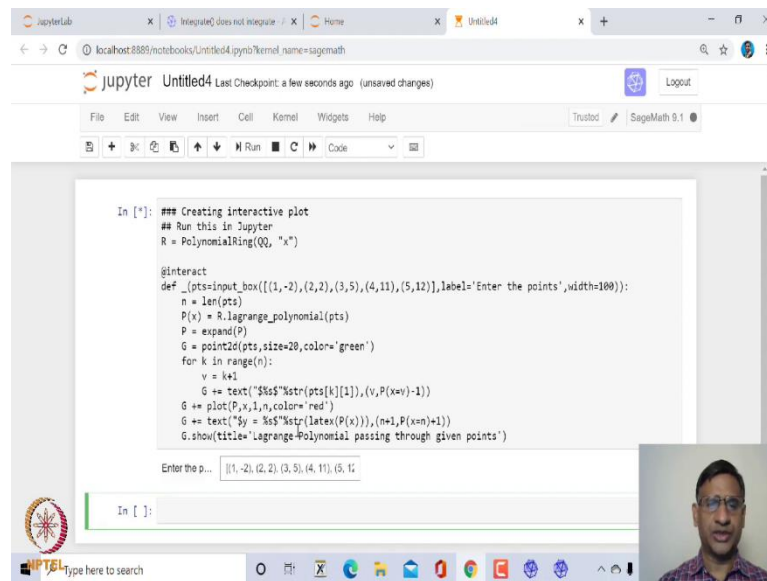
[illegible]

(Refer Slide Time: 09:17)



Now, let me start a new Notebook, that is new SageMath kernel.

(Refer Slide Time: 09:23)

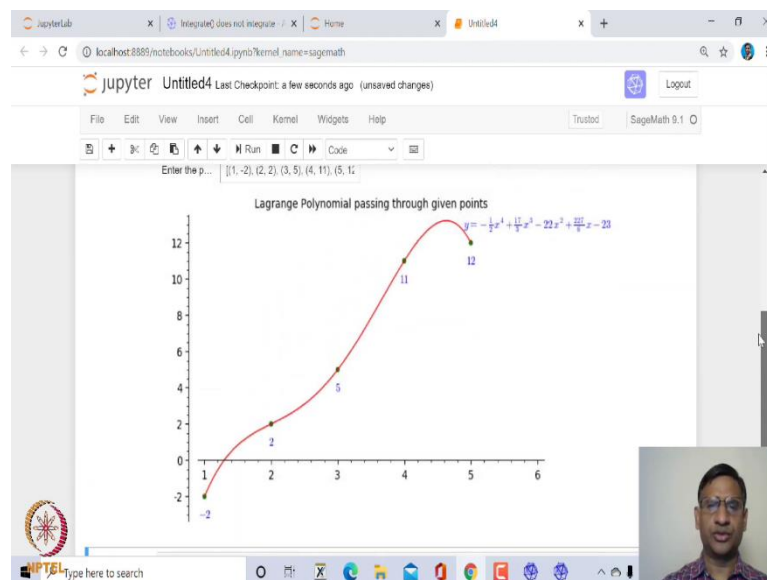


```
In [*]: ### Creating interactive plot
## Run this in Jupyter
R = PolynomialRing(QQ, "x")

@interact
def _(pts=input_box([(1,-2),(2,2),(3,5),(4,11),(5,12)],label='Enter the points',width=100)):
    n = len(pts)
    P(x) = R.lagrange_polynomial(pts)
    P = expand(P)
    G = point2d(pts,size=20,color='green')
    for k in range(n):
        v = k+1
        G += text("$x$"+str(pts[k][1]),(v,P(x*v)-1))
    G += plot(P,x,1,n,color='red')
    G += text("$y = %s$" % latex(P(x)),(n+1,P(x*n)+1))
    G.show(title='Lagrange Polynomial passing through given points')
```

And let me paste it here, and let us execute this, let us execute this. It is taking time, yet it should come, right?

(Refer Slide Time: 09:44)



So, this is what you get, and it should also give you, just one second, let me run once again. So, this gives you the set of points along with the polynomial which it has found out. And you can change these points.

(Refer Slide Time: 10:01)

```
G.show(title="Lagrange Polynomial passing through given points")

Enter the p... [1, 1], [2, 1], (3, 5), (4, 11), (5, 12)

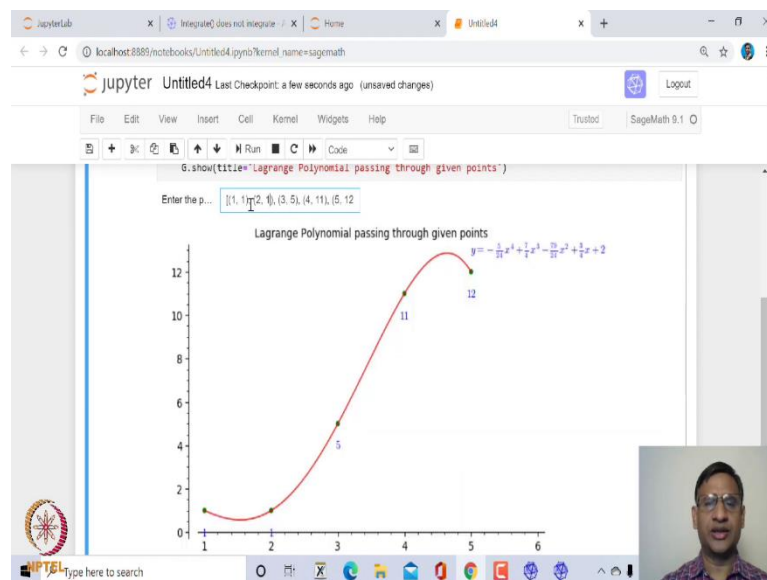
.....
Traceback (most recent call last)
/opt/sagemath-9.1/local/lib/python3.7/site-packages/ipywidgets/widgets/interaction.py in update(self, *args)
    249         value = widget.get_interact_value()
    250         self.kwargs[widget._kwarg] = value
--> 251         self.result = self.f(**self.kwargs)
    252         show_inline_matplotlib_plots()
    253         if self.auto_display and self.result is not None:

<ipython-input-2-8c321661de75> in _in_(pts)
     6 def _in_(pts=input_box([Integer(1), Integer(2), Integer(2), Integer(2), Integer(3), Integer(5)), Integer(4), Integer(11)), Integer(5), Integer(12)), label='Enter the points', width=Integer(180))):
     7     n = len(pts)
----> 8     _texp = var("x"); P = symbolic_expression(R.lagrange_polynomial(pts)).function(x)
     9     P = expand(P)
    10     G = point2d(pts, size=Integer(20), color='green')

/opt/sagemath-9.1/local/lib/python3.7/site-packages/sage/rings/polynomial/polynomial_ring.py:ange_polynomial(self, points, algorithm, previous_row)
    2262         return self.zero()
    2263
```

So, for example, we have taken 1, minus 2; I can take 1, for example, 1, 1; 2, let us say 1. And then run, automatically this will change.

(Refer Slide Time: 10:10)

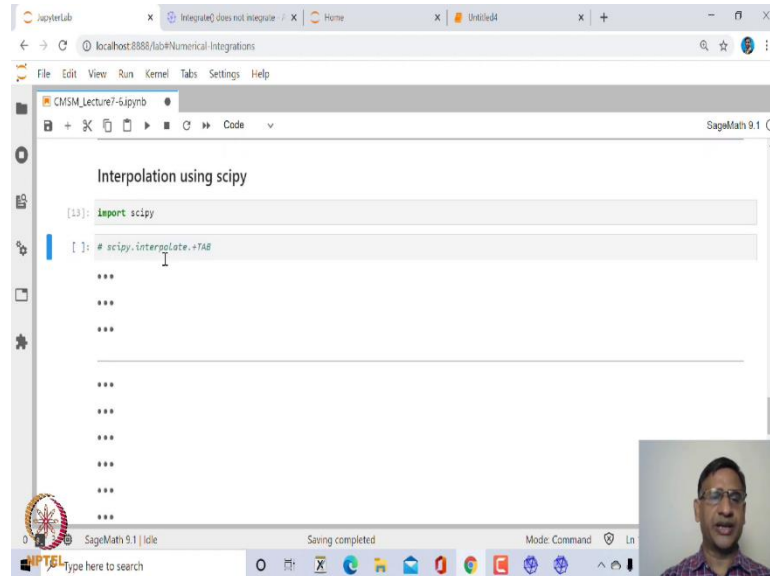


So, this is how one creates, this is what is called as the Sage interact.

So, first, you are giving the set of points, that is, some default points, and this is the label, and then we are saying what is the width of this, and then all you need to do is, just add these commands which we used for plotting interpolating, Lagrange interpreting polynomial along with the set of points, that is all is required.

So, you can, you can create this for actually, this is what is called interactive graph, right?
Let me go back to the Sage Jupyter lab.

(Refer Slide Time: 10:55)



```
Interpolation using scipy

[13]: import scipy

[ ]: # scipy.interpolate.*TAB

***

***

***

***

***

***

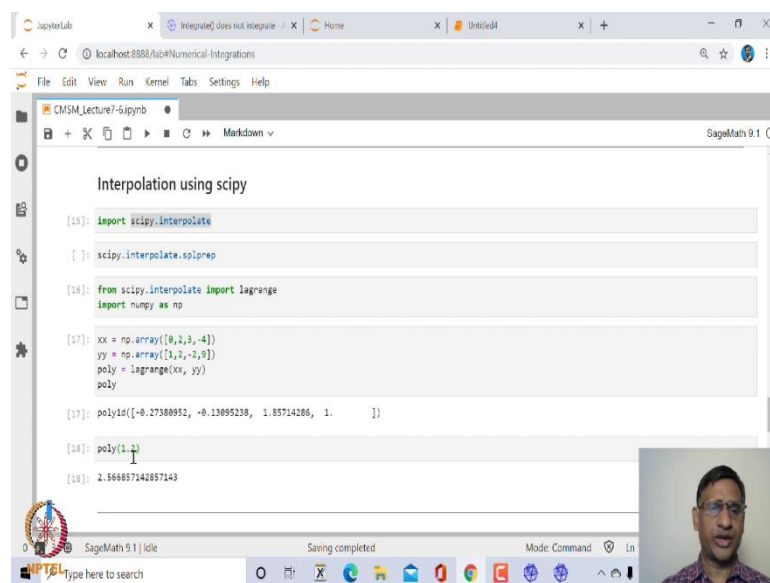
***

***

***
```

Once, next, yeah, let us look at, Sage has also inbuilt function to, you can also use scipy in order to find interpolating polynomial. So, how do we, how do we do that? So, let us first import scipy, and once you have imported scipy, then inside scipy, there is a, there is a, there is a module called interpolate.

(Refer Slide Time: 11:17)



```
Interpolation using scipy

[15]: import scipy.interpolate

[ ]: scipy.interpolate.splprep

[16]: from scipy.interpolate import lagrange
import numpy as np

[17]: xx = np.array([0,2,3,-4])
yy = np.array([1,2,-1,9])
poly = lagrange(xx, yy)
poly

[17]: poly([+0.27388952, +0.13695238, 1.85714286, 1.])

[18]: poly(1.2)

[18]: 2.566887142857143
```

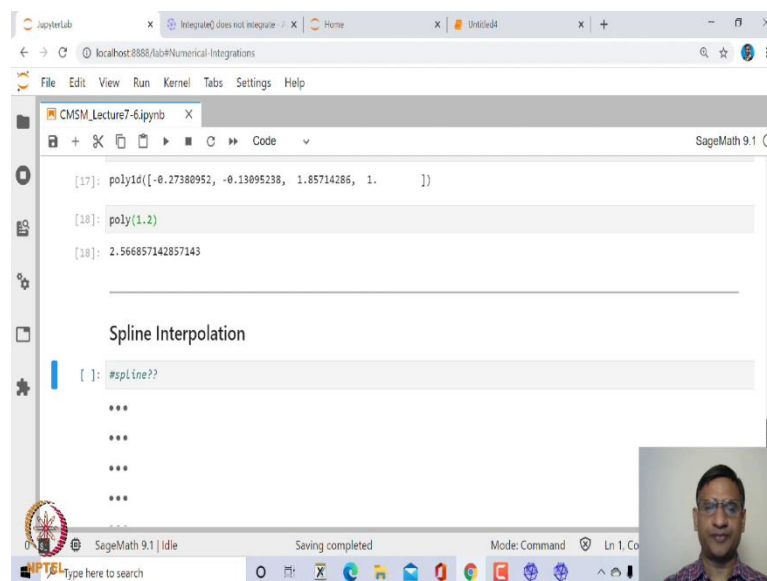
So, if I say Sage scipy dot interpolate, and then dot, and then press tab, let us run this, and then press tab. You will see all the several, just one second, scipy dot interpolate, inter, oh there is inter p dot, just one second. Now, we have to import, let us import interpolate, scipy dot interpolate, ok? Now, you can ask for scipy, scipy dot interpolate dot tab; then you will see the several, there are several methods, including one which is Lagrange.

So, this is Lagrange interpolation. It also has spline interpolation, and there are many other things. So, you can explore this and it also has b spline, right? So, how do we make use of this Lagrange interpolation from scipy? So, let us import numpy as np and from, in Sage, from scipy interpolate import lagrange. And then how do we create?

Suppose you have these points xx, the x coordinate of the set of points four points is what I have taken here, yy is the y coordinates of this set of points, and just say lagrange(xx,yy), and then it will give you the polynomial. Of course, in this case, it gives you the coefficients of the Lagrange interpolating polynomial.

And again you can find out what is the value of Lagrange interpolating polynomial at any point, let us say, for example, 1.2, right? So, this is how you can use Lagrange interpolating polynomial, either using this method Lagrange underscore polynomial, or you can use scipy, right?

(Refer Slide Time: 13:46)



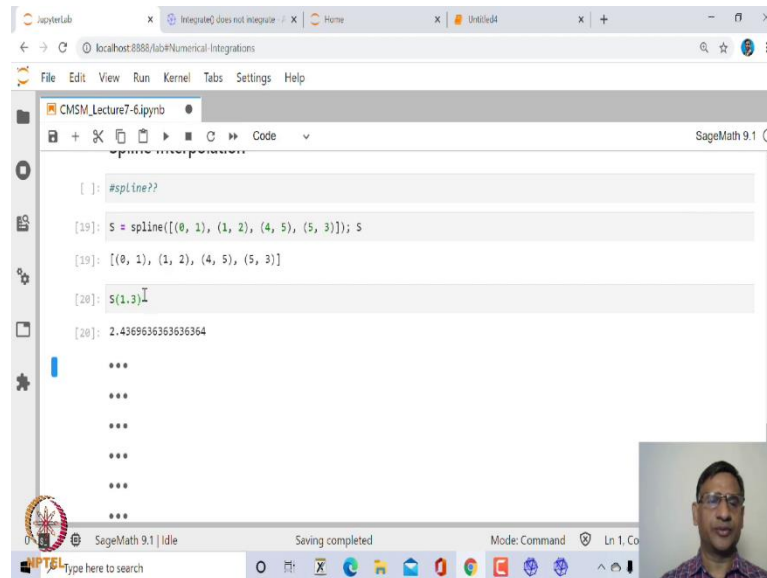
```
[17]: poly1d([-0.27380952, -0.13095238, 1.85714286, 1.])
[18]: poly(1.2)
[18]: 2.566857142857143
```

Spline Interpolation

```
[ ]: #spline??
***
***
***
***
```

Now, let us look at spline interpolation. So, spline interpolation is actually very powerful tool and it is used very heavily in many engineering branches, whenever you want to approximate some curve, then we use spline interpolation. So, Sage has inbuilt function called spline. So, you can, you can take help on spline, and then see what are the facilities available inside this document, right?

(Refer Slide Time: 14:23)



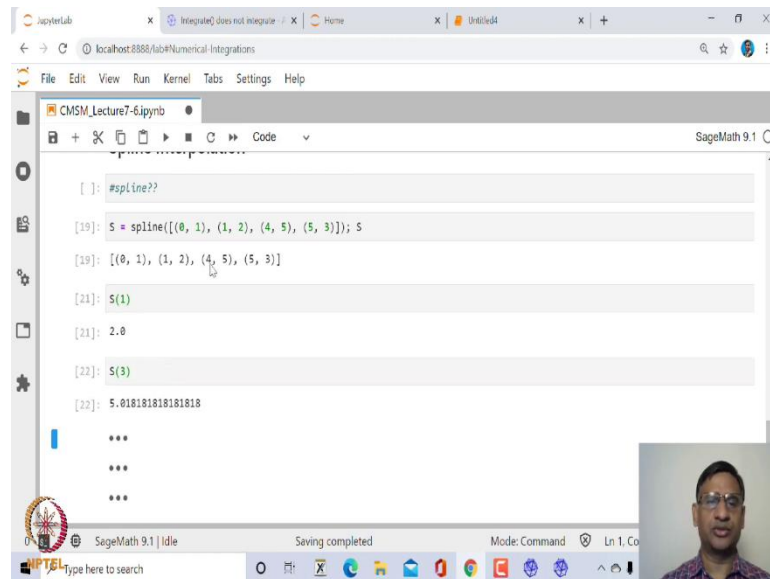
The screenshot shows a SageMath 9.1 JupyterLab window with the following code and output:

```
[ ]: #spline??  
[19]: S = spline([(0, 1), (1, 2), (4, 5), (5, 3)]); S  
[19]: [(0, 1), (1, 2), (4, 5), (5, 3)]  
[20]: S(1.3)  
[20]: 2.4369636363636364  
...  
...  
...  
...  
...  
...
```

The output shows the list of points $[(0, 1), (1, 2), (4, 5), (5, 3)]$ and the value of the spline at $x = 1.3$, which is 2.4369636363636364 . The interface also shows a file explorer on the left with a file named 'CMSM_Lecture7-6ipynb' and a video feed of the presenter in the bottom right corner.

So, let us create spline. So, first of all, you take, define the set of points. So, here the points we are taking as $(0, 1)$, $(1, 2)$, $(4, 5)$, $(5, 3)$, and we simply say spline over this set of points, list of points, and then let us see what is S . It gives you S as it has created the spline that fits to this set of points, and it says that you can, this spline is fitted over the set of points. Now, you can find out what is the value of the S at any point, intermediate point.

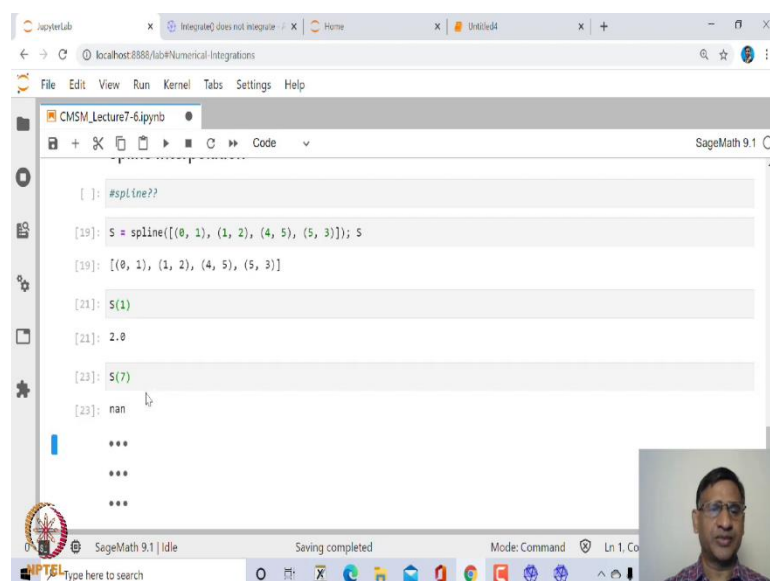
(Refer Slide Time: 15:06)



```
[ ]: #spline??  
[19]: S = spline([(0, 1), (1, 2), (4, 5), (5, 3)]); S  
[19]: [(0, 1), (1, 2), (4, 5), (5, 3)]  
[21]: S(1)  
[21]: 2.0  
[22]: S(3)  
[22]: 5.018181818181818  
***  
***  
***
```

So, suppose if I say S of 1, it should give me 2; if I say S of any value between 1, 0 and 5, it should give me. So, for example, if I say S of, say 3, then it is this value, 3 is not part of any of this x, value of the nodes.

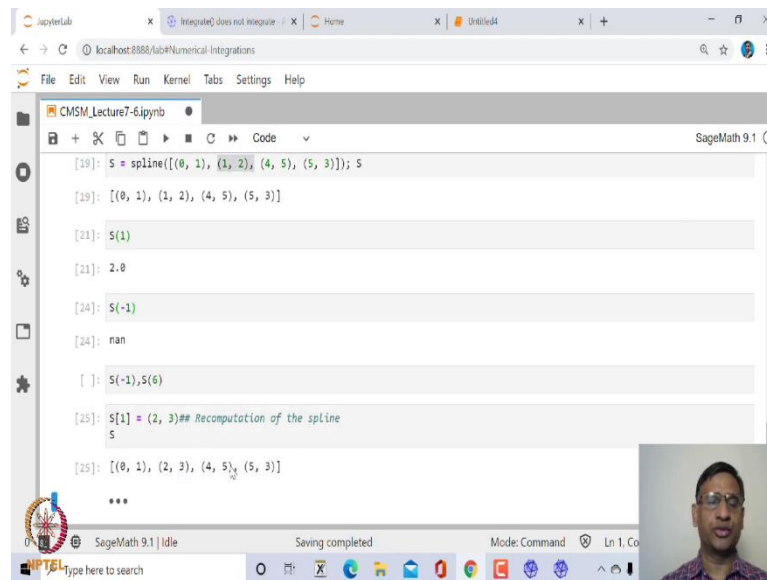
(Refer Slide Time: 15:27)



```
[ ]: #spline??  
[19]: S = spline([(0, 1), (1, 2), (4, 5), (5, 3)]); S  
[19]: [(0, 1), (1, 2), (4, 5), (5, 3)]  
[21]: S(1)  
[21]: 2.0  
[23]: S(7)  
[23]: nan  
***  
***  
***
```

But if I say S of, let us say 7, then it will give you none value; if I say anything beyond these x values. So, S of, let us say, minus 1, it will also give you none value.

(Refer Slide Time: 15:37)



```
[19]: S = spline([(0, 1), (1, 2), (4, 5), (5, 3)]); S
[19]: [(0, 1), (1, 2), (4, 5), (5, 3)]

[21]: S(1)
[21]: 2.0

[24]: S(-1)
[24]: nan

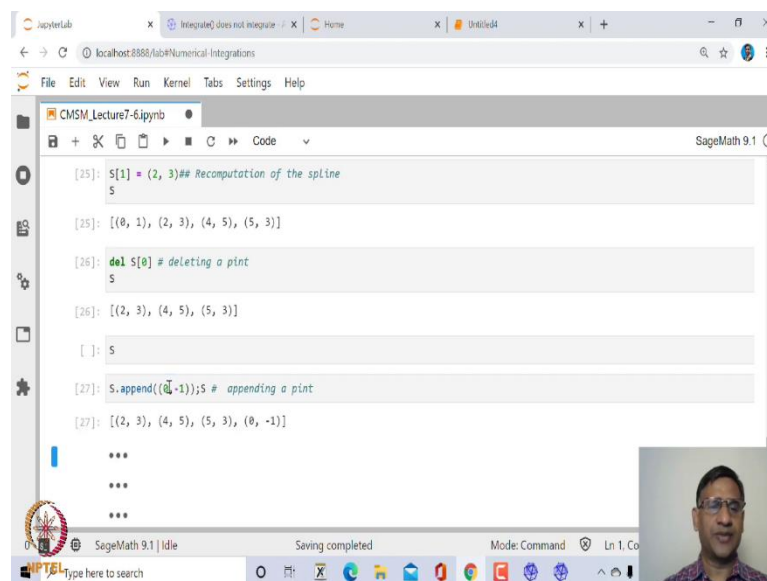
[ ]: S(-1), S(6)

[25]: S[1] = (2, 3) ## Recomputation of the spline
[25]: [(0, 1), (2, 3), (4, 5), (5, 3)]

***
```

So, this, unlike Lagrange interpolating polynomial, this spline will fit only within these set of points, not beyond this. Yeah, so, this I already explained. You could even change the value of one of these nodes after you have fitted this spline. So, for example, S of 1, S of 1 will be this 1, 2 was S of 1; that is the second point, in this case, was (1,2). Now, suppose I change that to (2,3), and then you can again run, it will recompute this spline and gives you the value, right? So, you do not have to do this procedure again; you simply just reassign one of this node, it will compute.

(Refer Slide Time: 16:32)



```
[25]: S[1] = (2, 3) ## Recomputation of the spline
[25]: [(0, 1), (2, 3), (4, 5), (5, 3)]

[26]: del S[0] # deleting a point
[26]: [(2, 3), (4, 5), (5, 3)]

[ ]: S

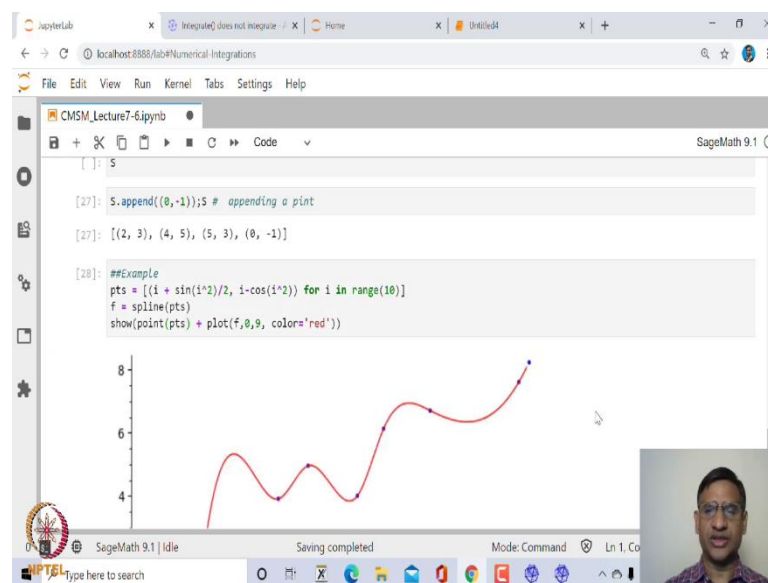
[27]: S.append([(0, -1)]); S # appending a point
[27]: [(2, 3), (4, 5), (5, 3), (0, -1)]

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


Similarly, you could, you could even delete one of these nodes. So, suppose I, if we delete, delete the first point; what was the first point, $(0, 1)$. Suppose we delete this first point and then see what is S ; you will see that now first point is not there.

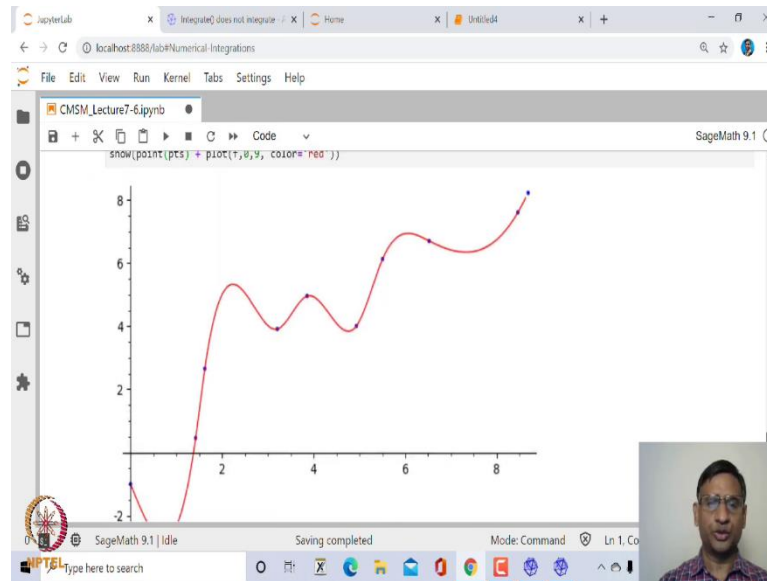
You could even, you could even append a point; now, let us say, append first point 0, instead of $(0,1)$, let us say $(0,-1)$ and then. Now, of course, it will append at the end, not at the beginning; but it does not matter whether, this order does not matter, you can still, you will be able to fit this spline.

(Refer Slide Time: 17:13)

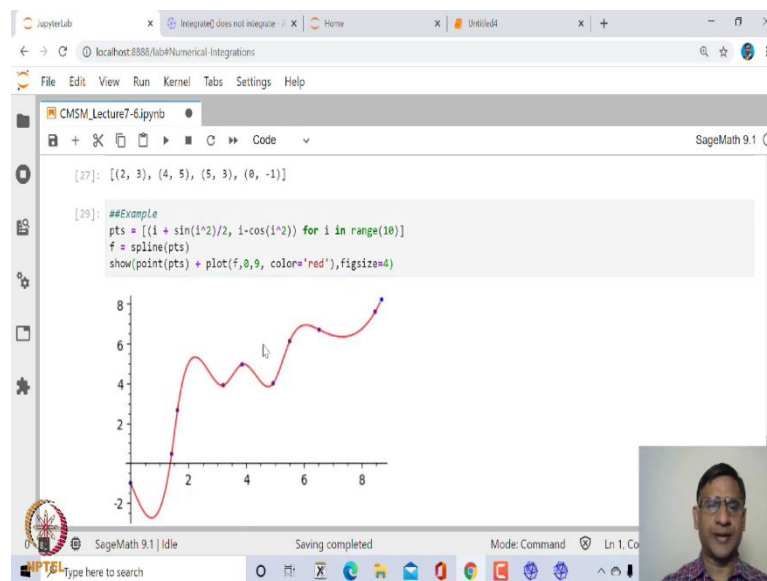


You could, now let us plot the graph of set of points; the points we are taking as i plus $\sin i$ square divided by 2, and the second coordinate is i minus $\cos i$ square by 2, i is varying between 0 to 9. And when you fit this spline to this set of points, and let us plot this graph.

(Refer Slide Time: 17:41)

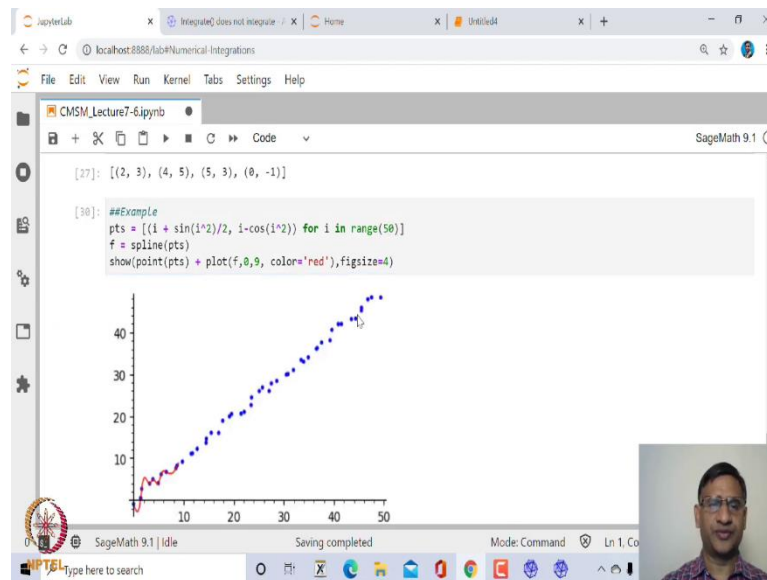


(Refer Slide Time: 17:52)



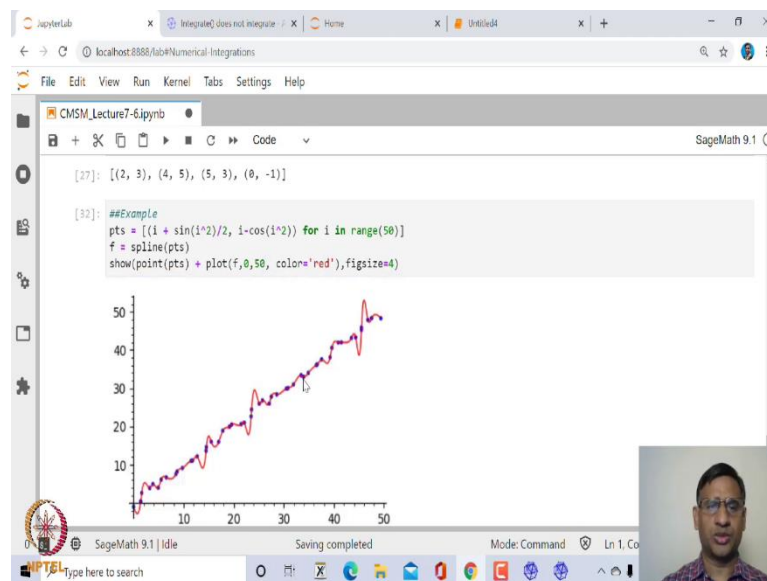
So, you can see here, this is actually very very smooth. So, fitting spline is, let me, let me reduce the figure size, fig size is equal to let us say 4; yeah, if you increase the number of points, for example, instead of 10, let us make it 50 and then this is, this is.

(Refer Slide Time: 18:03)



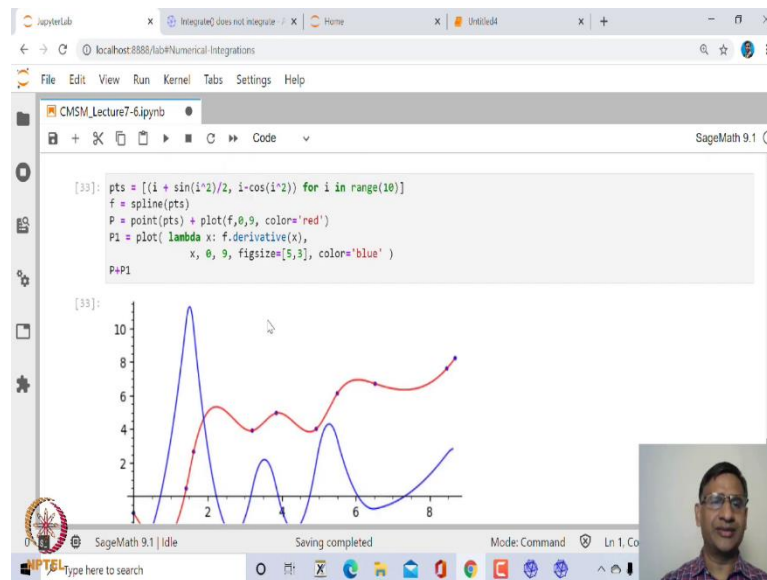
So, in this case, these set of points has gone out.

(Refer Slide Time: 18:15)



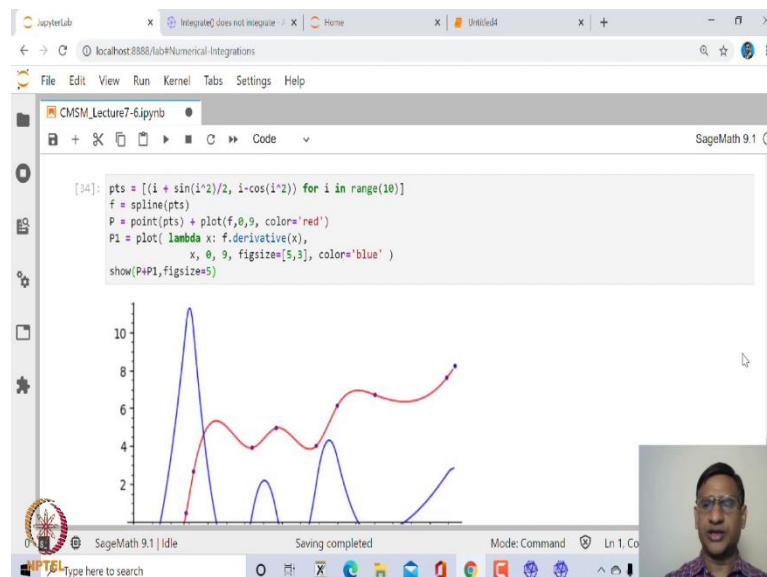
So, I should say here n; oh no, this is, this is 50, right? So, that is how you can fit.

(Refer Slide Time: 18:29)



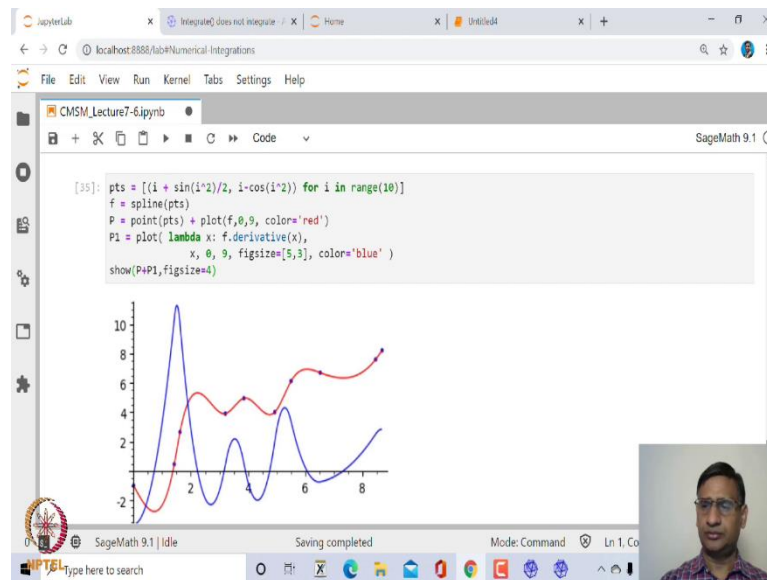
Similarly, let us take; you could, once you have fitted this spline to given set of points, that spline you can even find the derivative, you can find derivative. So, for example, let us look at the same spline and then find out derivative of this f ; f is the spline fitted to this set of points, and we can find out the derivative of this, and then plot the graph.

(Refer Slide Time: 18:59)



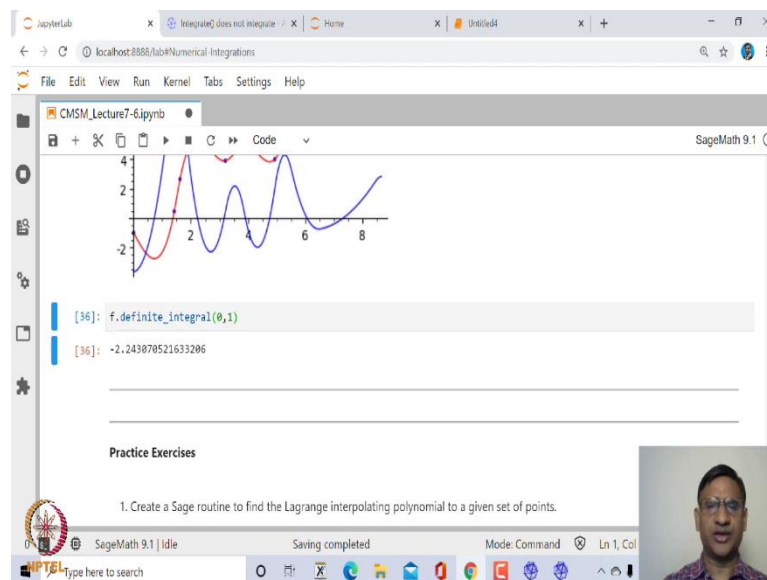
So, let me again reduce the figure size, figsize, figsize is equal to let us say 5, in this case, right?

(Refer Slide Time: 19:10)



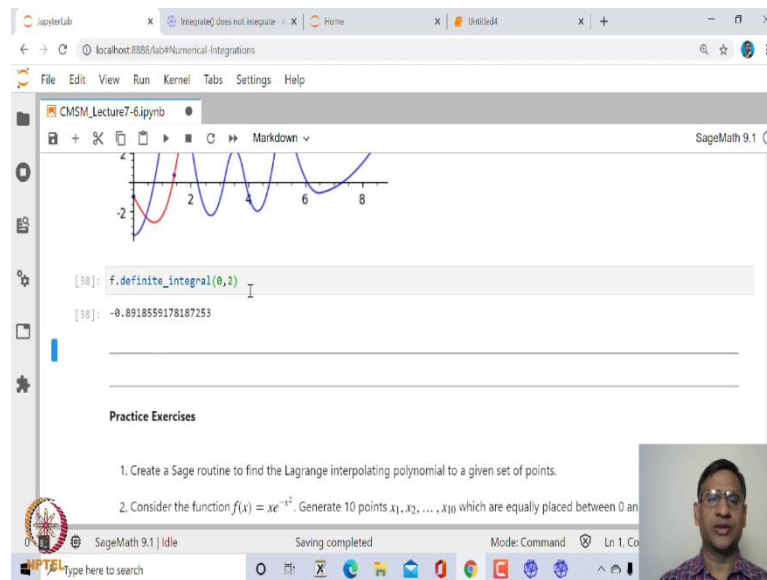
So, no, 5 is slightly big. So, let me make it 4, you can, right? So, this fitted spline, this is actually a differentiable function.

(Refer Slide Time: 19:28)



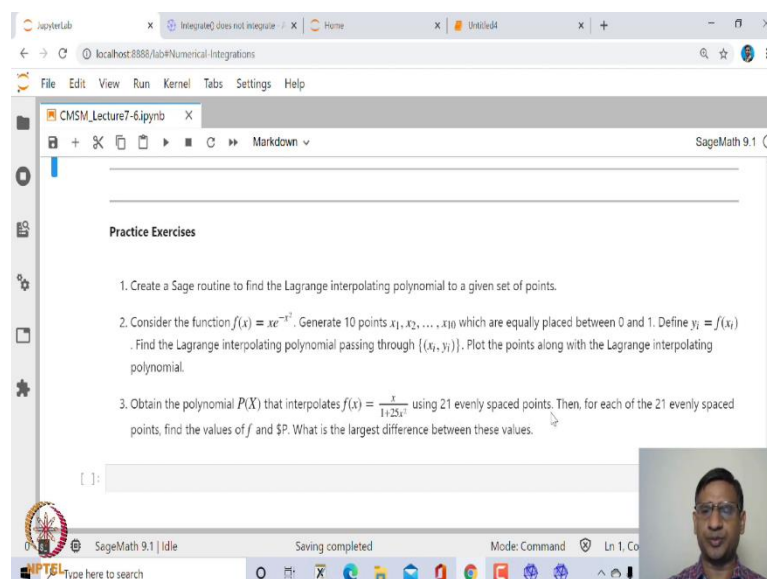
Similarly, you can, you can even integrate. So, you can find out f dot definite integral of this between 0 and 1. So, you can find different, integral of this between 0 and 1, or 0 and whatever is the number, for example, we have taken here 10.

(Refer Slide Time: 19:53)



So, we can ask 9, sorry 9 is I think beyond. Yeah so, within that limit, it will give you the definite integral, ok? So, you could also explore, for example, I would expect you, in case you have learned other interpolation method like Newton's method, which uses equally placed points in an interval; then you could actually create your own Sage subroutine for Newton's backward and forward interpolation, similarly for Newton's divided difference, right, ok?

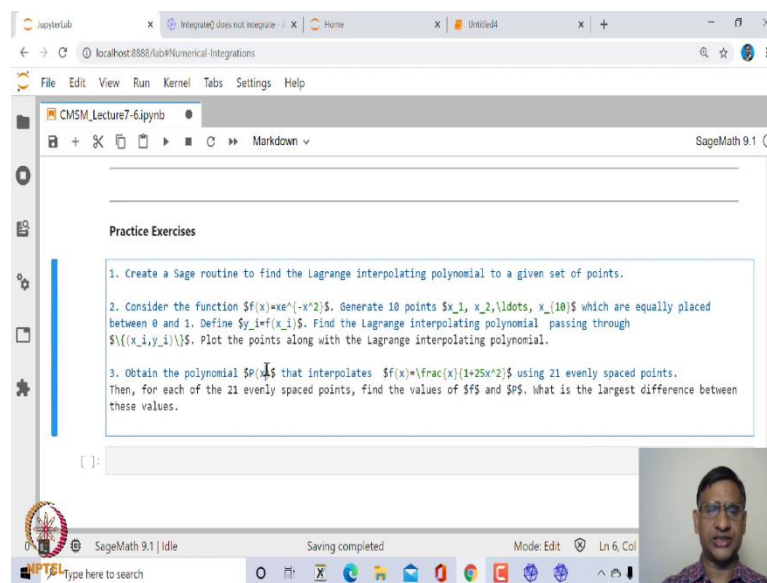
(Refer Slide Time: 20:36)



So, let me leave you few simple exercises. The first is, create Sage routine to find Lagrange interpolating polynomial to a given set of points, we have used inbuilt methods. Next one is consider a function $f(x)$ equal to x into e to the power minus x square. Generate 10 points, random points x_1, x_2, \dots, x_n which are equally placed, between 0 and 1, and define y equal to $f(x_i)$.

So, you have 10 points (x_i, y_i) ; now find the Lagrange interpolating polynomial passing through this, and then again plot these points along with this. So, I expect you to do this for your user-defined function that you have created for Lagrange interpolating polynomial. And the last one is, obtain the polynomial P_x which interpolates this using 21 evenly spaced points, and then for each of these 21 evenly placed points, find the value of f , and the value of P , and value of P and also tabulate the difference between the value of f and the value of P .

(Refer Slide Time: 21:33)

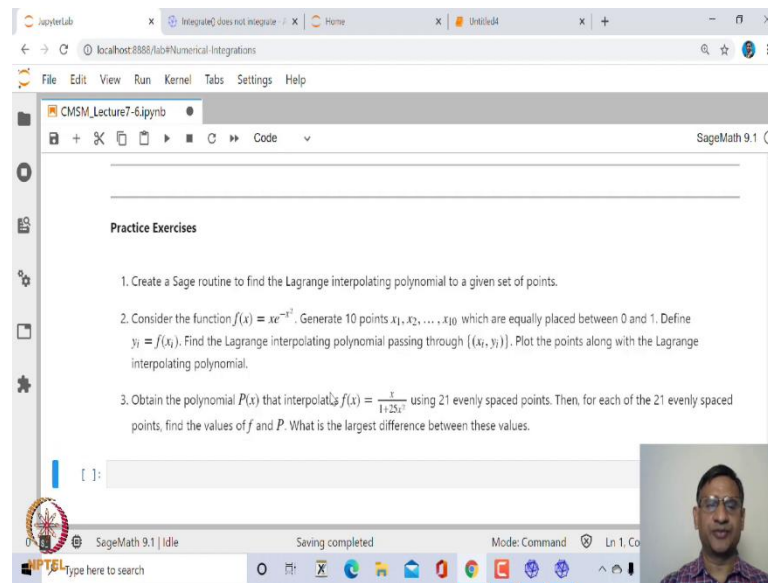


The screenshot shows a JupyterLab window with a SageMath 9.1 notebook titled 'CMSM_Lecture7-61pynb'. The notebook contains three practice exercises:

1. Create a Sage routine to find the Lagrange interpolating polynomial to a given set of points.
2. Consider the function $f(x) = xe^{-x^2}$. Generate 10 points x_1, x_2, \dots, x_{10} which are equally placed between 0 and 1. Define $y_i = f(x_i)$. Find the Lagrange interpolating polynomial passing through $\{(x_i, y_i)\}$. Plot the points along with the Lagrange interpolating polynomial.
3. Obtain the polynomial $P(x)$ that interpolates $f(x) = \frac{x}{1+25x^2}$ using 21 evenly spaced points. Then, for each of the 21 evenly spaced points, find the values of f and P . What is the largest difference between these values.

The interface includes a file explorer on the left, a top menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help), and a bottom status bar showing 'SageMath 9.1 | Idle' and 'Mode: Edit'.

(Refer Slide Time: 21:38)



Let this, let me put this as small x, right? So, that is the, these are the three exercises; you could use here spline interpolation or Lagrange interpolation, and in both the cases, you can, you can, you can compare, right? So, this, these are some simple exercises you could try.

So, thank you very much.