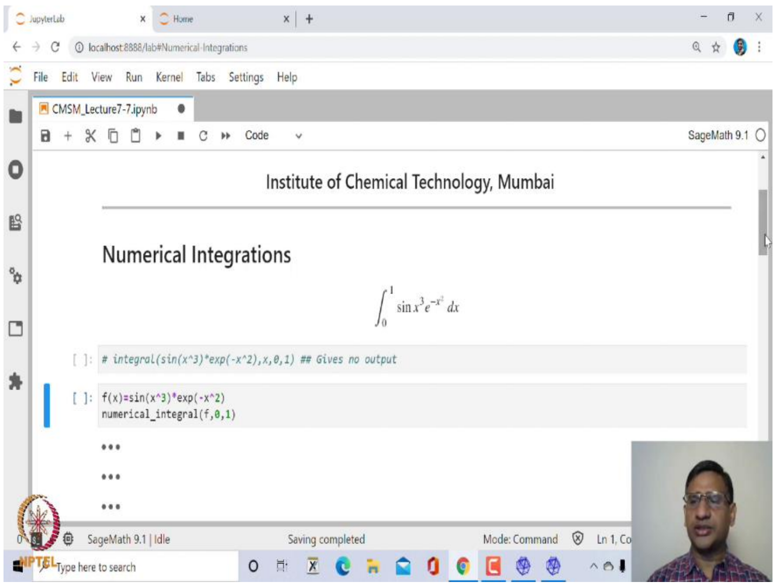
Computational Mathematics with SageMath

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

Lecture - 48
Numerical Integration in SageMath

(Refer Slide Time: 00:15)



Institute of Chemical Technology, Mumbai

Numerical Integrations

$$\int_0^1 \sin x^3 e^{-x^2}\, dx$$

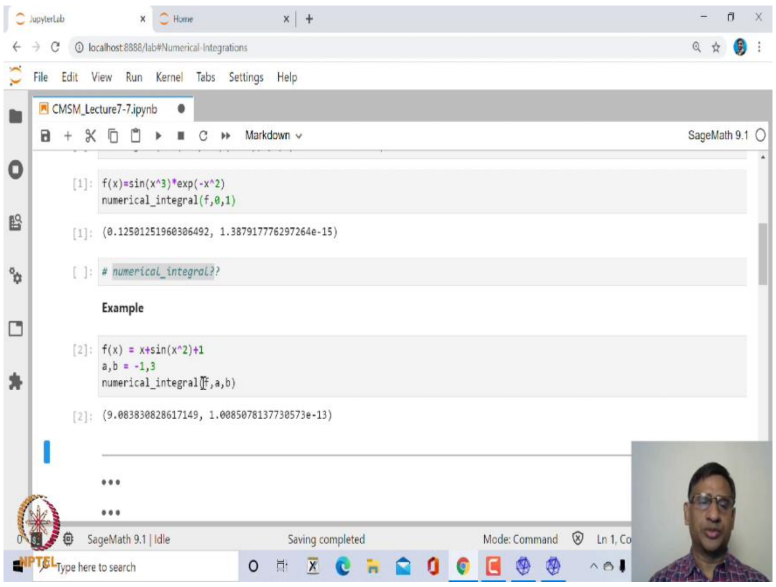
```
[ ]: # integral(sin(x^3)*exp(-x^2),x,0,1) ## Gives no output
```

```
[ ]: f(x)=sin(x^3)*exp(-x^2)
     numerical_integral(f,0,1)
```

Welcome to the 48th lecture on Computational Mathematics with SageMath. In this lecture, let us explore some methods of Numerical Integrations using SageMath. So, we have already seen, how to find integral of a function. So, let us say, for example, if you look at integral of sin x cube into e to the power minus x square from 0 to 1.

If you try to evaluate this integral using this integral function in sin, it will actually give you no output, it will just return the same command which you are giving, because, it is unable to, Sage is unable to compute this integral symbolically or explicitly.So, what you can do is, you can use numerical integration. We have already seen this, whenever you, you are, Sage is unable to find integral, definite integral explicitly, then we can use numerical integral. So, Sage already has inbuilt function to find numerical integral.

(Refer Slide Time: 01:24)



```
[1]: f(x)=sin(x^3)*exp(-x^2)
     numerical_integral(f,0,1)

[1]: (0.12501251960306492, 1.387917776297264e-15)

[ ]: # numerical_integral??
```

Example

```
[2]: f(x) = x+sin(x^2)+1
     a,b = -1,3
     numerical_integral(f,a,b)

[2]: (9.083830828617149, 1.0085078137730573e-13)
```

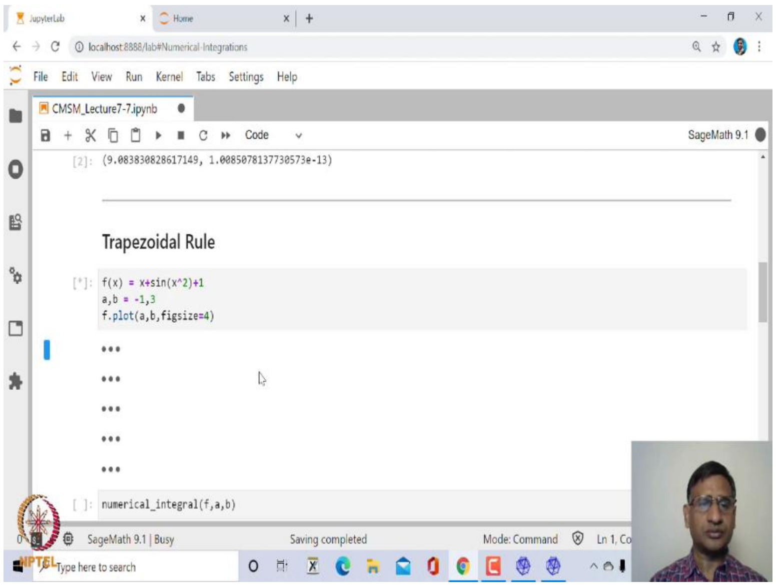
So, let us write this, so in, in this case, integral of sin x cube into e to the power minus x square, from 0 to 1 the value is this, this is the integral, and this is the error term. So, numerical integral also gives you error term, error when this integral is computed.

Now, of course, you can take help on this inbuilt function numerical integral, go through that document, and then see how this function is created in SageMath, right? So, let us take an example, another example, suppose we want to find integral of x plus sin x square plus 1 between minus 1 and 3.

So, of course, Sage has inbuilt function, using that we can compute the integral, in this case, integral is 9.08383 and so on, with an error or tolerance limit as 10 to, of the order 10 to the power minus 13.

Now, let us try to create our own user-defined functions using 2 methods, Gauss, one method is Trapezoidal method, that is Trapezoidal rule, and the other one is Simpsons one-third rule.

(Refer Slide Time: 02:56)



```
[2]: (9.083830828617149, 1.0085078137730573e-13)
```

Trapezoidal Rule

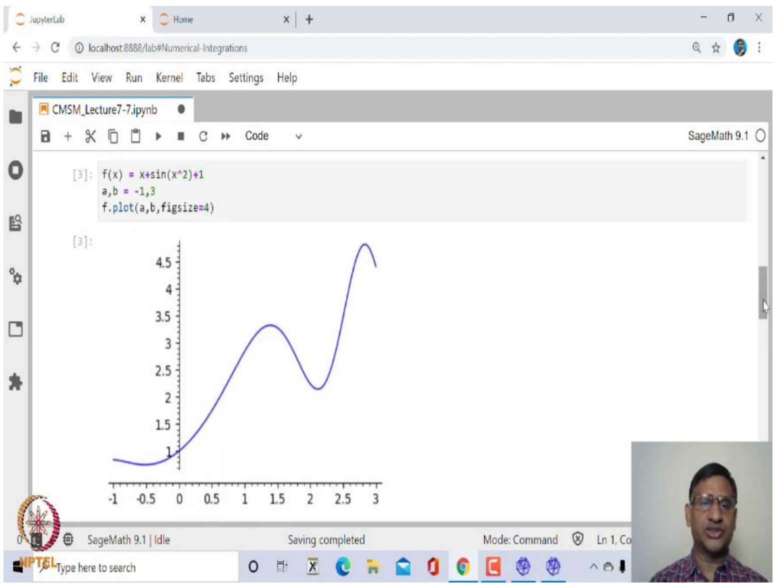
```
[*]: f(x) = x+sin(x^2)+1
     a,b = -1,3
     f.plot(a,b,figsize=4)
```

```
[ ]: numerical_integral(f,a,b)
```

So, how do we do that? So, in case, the idea of finding this numerical integral is to approximate this function integrand using interpolating polynomial. So, we can use interpolating polynomial of degree 1, that is linear function, or quadratic interpolating polynomial, or cubic interpolating polynomial, and one can also use Spline interpolating polynomial.

So, that is the basic idea. Trapezoidal rule uses linear interpolating polynomial, whereas Simpsons rules uses, Simpsons one-third rule uses quadratic interpolating polynomial, and Simpsons three-eighth uses cubic interpolating polynomial. So, let us start with Trapezoidal rule.

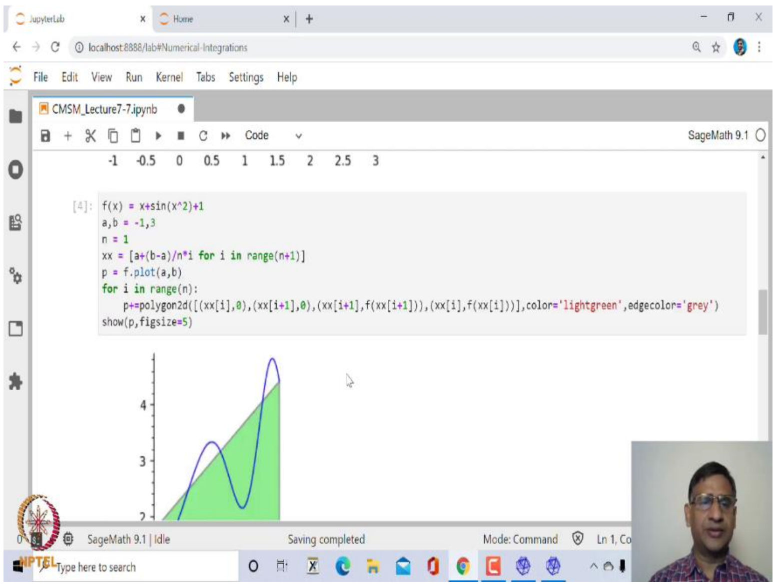
(Refer Slide Time: 03:46)



```
[3]: f(x) = x+sin(x^2)+1
     a,b = -1,3
     f.plot(a,b,figsize=4)
```

So, what is it? So, this is the, the, the function, the graph of this function. You want to find the area under this curve between x equal to minus 1 and x equal to 3. So, what do we do? We, we approximate this, the curve by means of a straight line, that is linear interpolating polynomial.

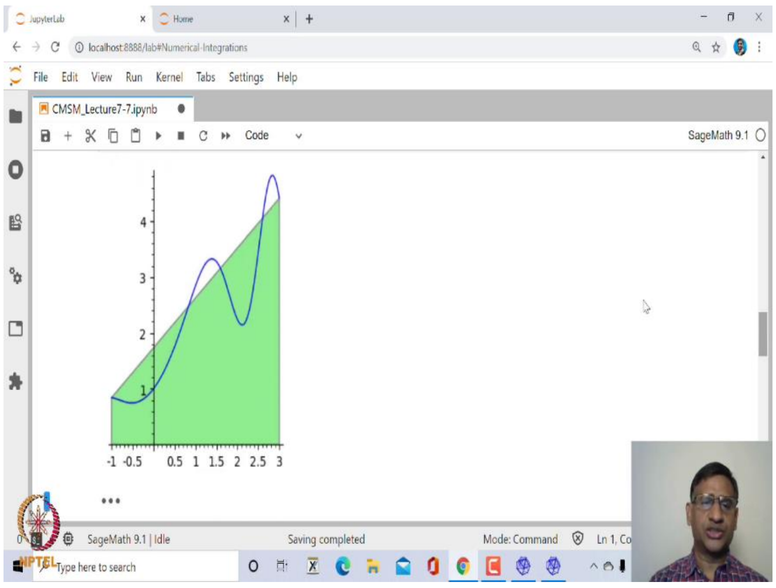
(Refer Slide Time: 04:09)



```
[4]: f(x) = x+sin(x^2)+1
     a,b = -1,3
     n = 1
     xx = [a+(b-a)/n*i for i in range(n+1)]
     p = f.plot(a,b)
     for i in range(n):
         p+=polygon2d([(xx[i],0),(xx[i+1],0),(xx[i+1],f(xx[i+1])),(xx[i],f(xx[i]))],color='lightgreen',edgecolor='grey')
     show(p,figsize=5)
```

So, when you, when you do that, when you, when you do that, then this is what you will get, right?

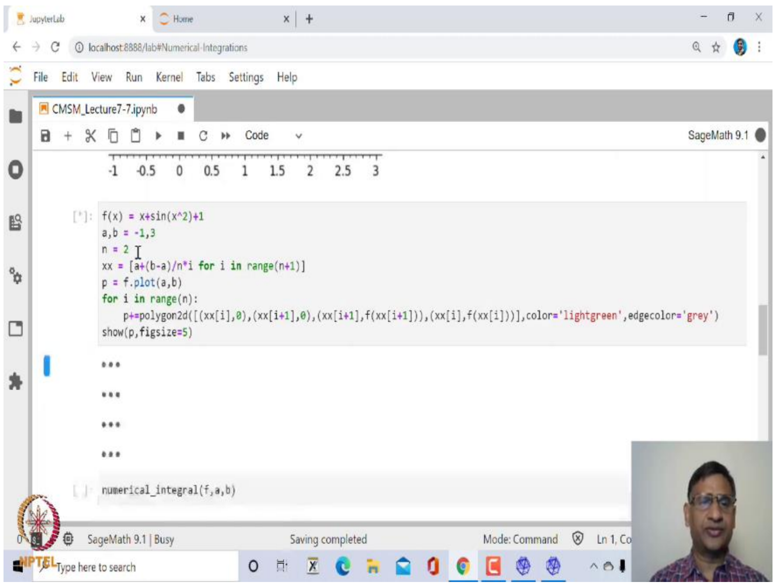
(Refer Slide Time: 04:14)



So, this is the, the, the original curve, this is the straight line passing through these initial point, let us say (a,f(a)) and (b,f(b)). However, if you, if you integrate this, you can see that this is the extra portion, this is extra portion, this is extra, this is extra portion, this is extra portion, so a lot of error is there.

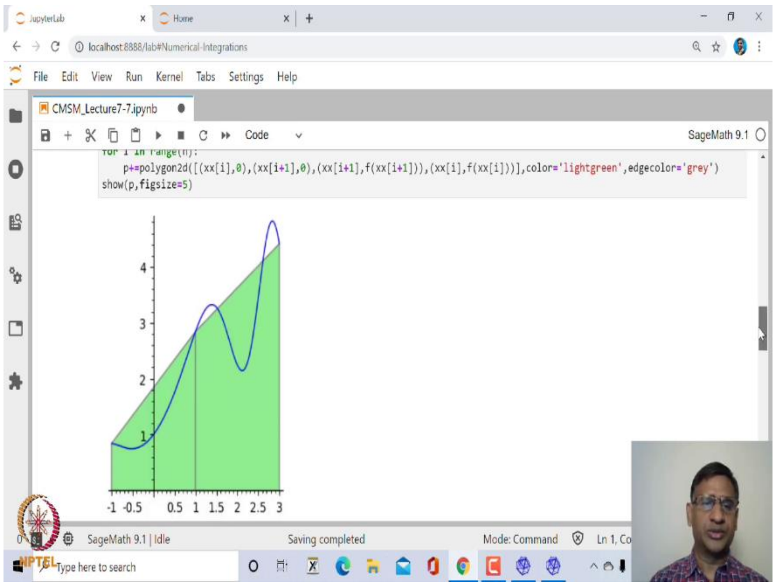
Now, instead of the whole interval, approximating this function in the whole interval by one single linear interpolating polynomial, now, let us divide this interval into 2 equal parts.

(Refer Slide Time: 04:57)



```
[*]: f(x) = x+sin(x^2)+1
     a,b = -1,3
     n = 2
     xx = [a+(b-a)/n*i for i in range(n+1)]
     p = f.plot(a,b)
     for i in range(n):
         p+=polygon2d([(xx[i],0),(xx[i+1],0),(xx[i+1],f(xx[i+1])),(xx[i],f(xx[i]))],color='lightgreen',edgecolor='grey')
     show(p,figsize=5)

     ...

     ...

     ...

     ...

[ ]: numerical_integral(f,a,b)
```

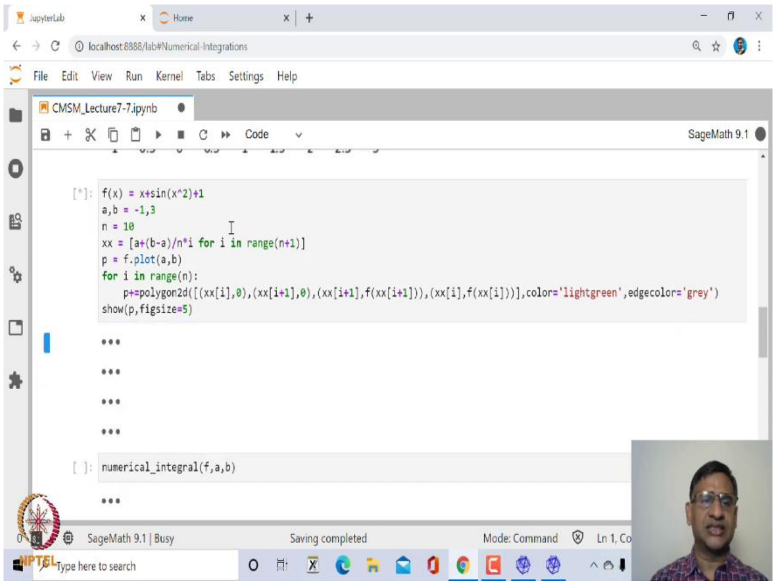
(Refer Slide Time: 04:58)



```
for i in range(n):
    p+=polygon2d([(xx[i],0),(xx[i+1],0),(xx[i+1],f(xx[i+1])),(xx[i],f(xx[i]))],color='lightgreen',edgecolor='grey')
show(p,figsize=5)
```

So, if I divide into 2 equal parts and then see what happens. Then the, the value of the integral, you are splitting into this part, and then this part so, you will see that the error would have reduced.

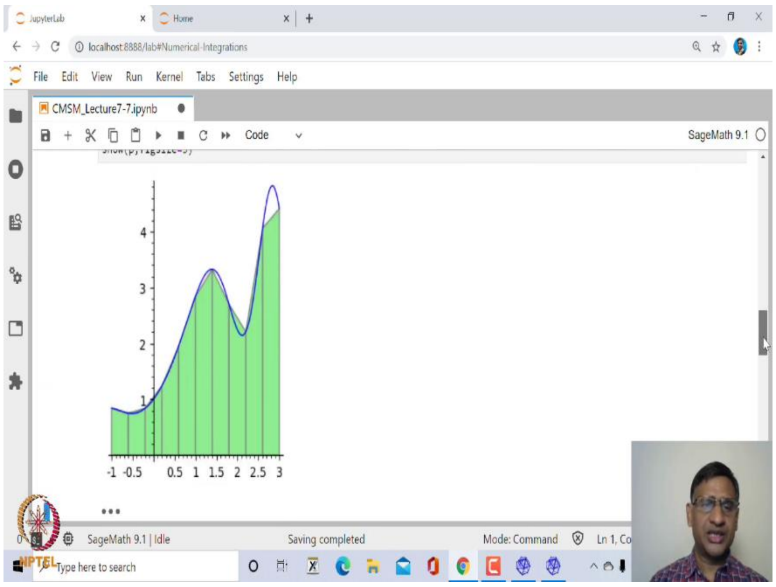
(Refer Slide Time: 05:15)



```
[*]: f(x) = x+sin(x^2)+1
     a,b = -1,3
     n = 10
     xx = [a+(b-a)/n*i for i in range(n+1)]
     p = f.plot(a,b)
     for i in range(n):
         p+=polygon2d([(xx[i],0),(xx[i+1],0),(xx[i+1],f(xx[i+1])),(xx[i],f(xx[i]))],color='lightgreen',edgecolor='grey')
     show(p,figsize=5)
```

```
[ ]: numerical_integral(f,a,b)
```

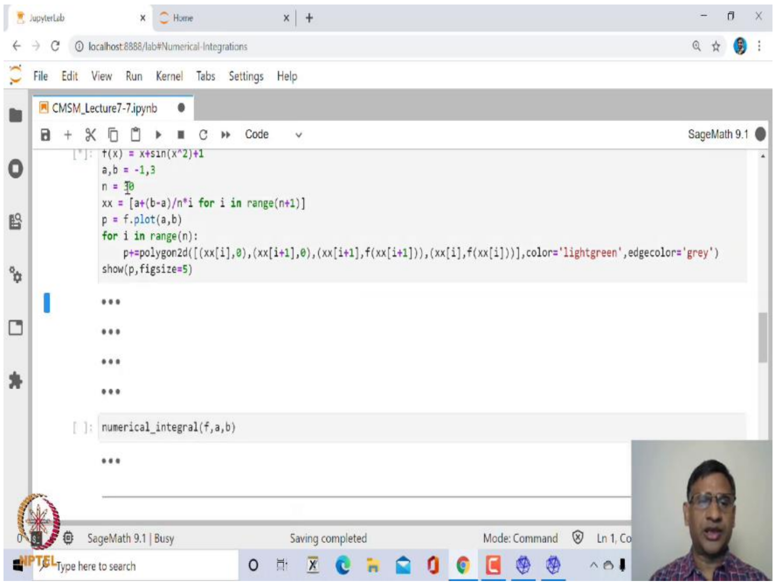
(Refer Slide Time: 05:16)



And instead of 2 parts, suppose if I take 10 parts, you divide this interval into 10 equal parts, and then in each sub-interval, you approximate the, the function by linear interpolating polynomial, and then join the end points by a straight line.

And then you can see here, this, this area of this, some of this trapezium, that is, these are all trapezium, that is why it is called Trapezoidal method, is much closer to the, the actual area between this curve.

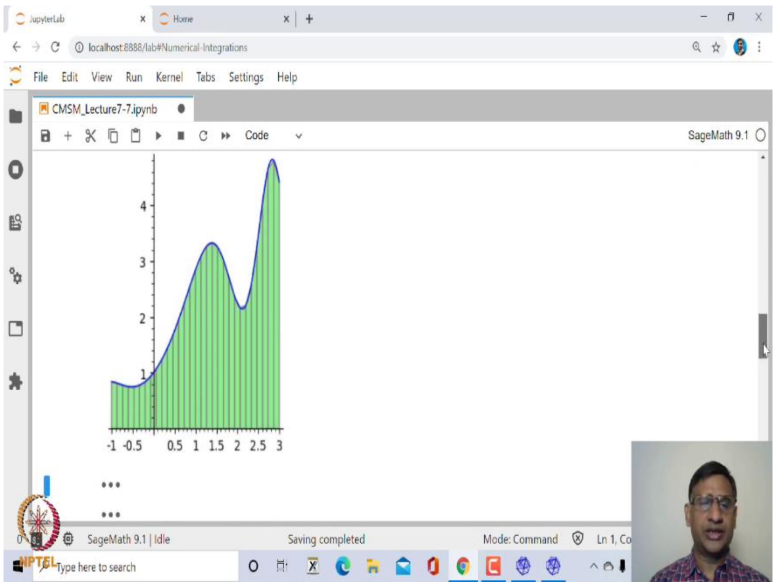
(Refer Slide Time: 05:47)



```
f(x) = x+sin(x^2)+1
a,b = -1,3
n = 30
xx = [a+(b-a)/n*i for i in range(n+1)]
p = f.plot(a,b)
for i in range(n):
    p+=polygon2d([(xx[i],0),(xx[i+1],0),(xx[i+1],f(xx[i+1])),(xx[i],f(xx[i]))],color='lightgreen',edgecolor='grey')
show(p,figsize=5)
```

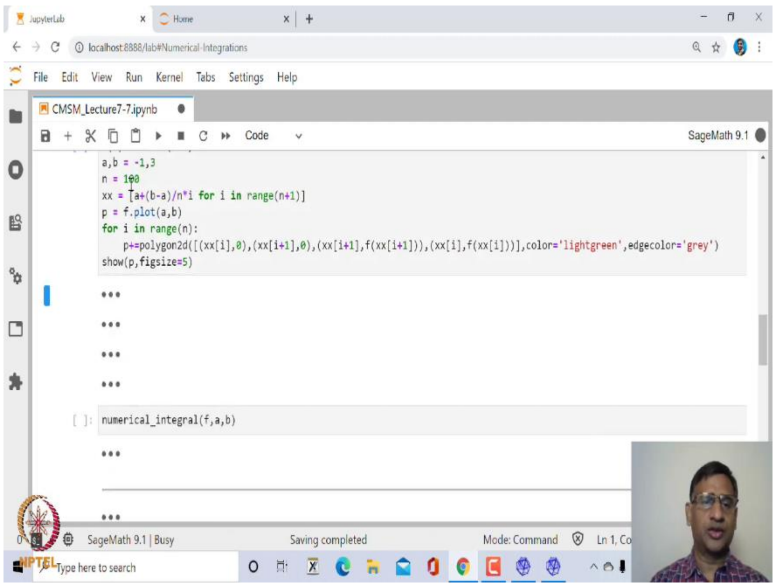
```
numerical_integral(f,a,b)
```

(Refer Slide Time: 05:48)



Instead of 10, let us make it 30, and then you will see that it is much closer, right?

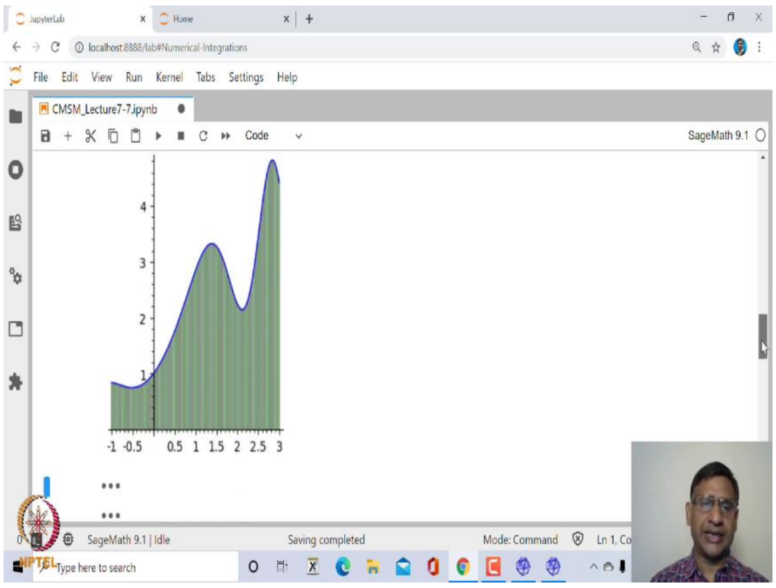
(Refer Slide Time: 05:56)



```
a,b = -1,3
n = 100
xx = [a+(b-a)/n*i for i in range(n+1)]
p = f.plot(a,b)
for i in range(n):
    p+=polygon2d([(xx[i],0),(xx[i+1],0),(xx[i+1],f(xx[i+1])),(xx[i],f(xx[i]))],color='lightgreen',edgecolor='grey')
show(p,figsize=5)
```

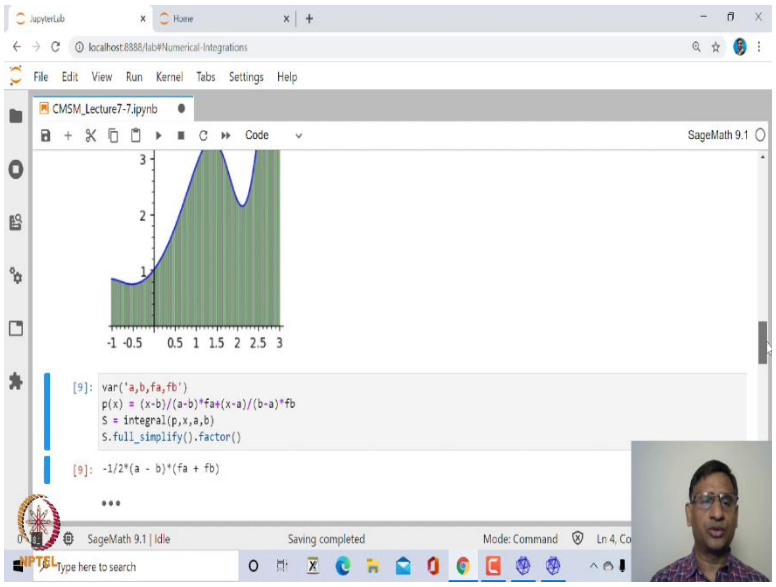
```
[ ]: numerical_integral(f,a,b)
```

(Refer Slide Time: 05:57)



Instead of 30, if you make it, for example 100, then you will see that the actual area is very close to the approximate area.

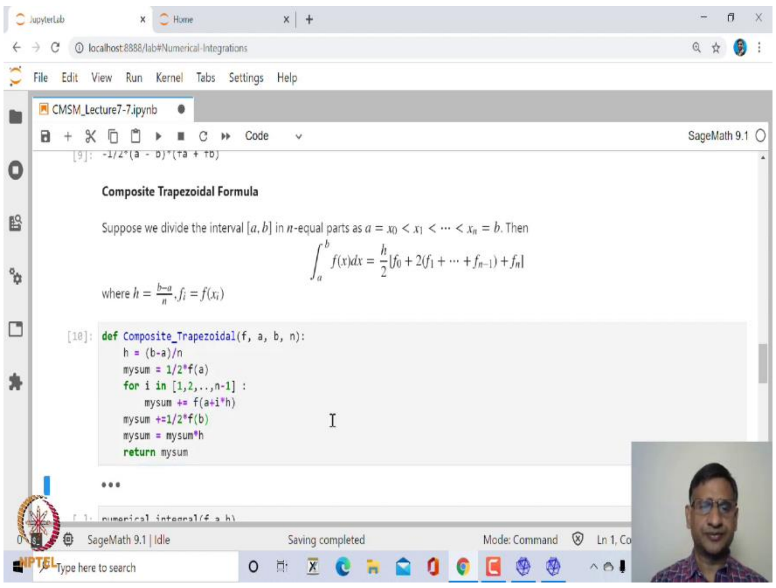
(Refer Slide Time: 06:03)



```
[9]: var('a,b,fa,fb')
     p(x) = (x-b)/(a-b)*fa+(x-a)/(b-a)*fb
     S = integral(p,x,a,b)
     S.full_simplify().factor()

[9]: -1/2*(a - b)*(fa + fb)
```

Now, let us look at what happens. So, suppose we, we obtain this interpolating polynomial, we fit this interpolating polynomial to this points (a, fa), (b, fb). So, fa and fb, I am just defining as variables, then, px is a polynomial which is equal to x minus b upon a minus b into fa plus x minus a divided by b minus a upon fb. So, this is actually a Lagrange interpolating polynomial passing through 2 points (a,fa), (b,fb).And then, if you integrate this and then ask it to simplify the integral, in this case, so, actually in this case, integral will, is going to be half into the, half into the base that is b minus a, into the sum of this parallel height, that is fa plus fb.

(Refer Slide Time: 07:27)



Composite Trapezoidal Formula

Suppose we divide the interval $[a,b]$ in n-equal parts as $a = x_0 < x_1 < \cdots < x_n = b$. Then

$$\int_a^b f(x)dx = \frac{h}{2}[f_0 + 2(f_1 + \cdots + f_{n-1}) + f_n]$$

where $h = \frac{b-a}{n}, f_i = f(x_i)$

```
[10]: def Composite_Trapezoidal(f, a, b, n):
          h = (b-a)/n
          mysum = 1/2*f(a)
          for i in [1,2,..,n-1] :
              mysum += f(a+i*h)
          mysum +=1/2*f(b)
          mysum = mysum*h
          return mysum
```

So, it will be half times b minus a into fa plus fb ,right, that is what it would have reported, yes? This is minus half into a minus b, a minus b, if you multiply, push it inside, it will b minus a into fa plus fb. So, that is the area using single, single interval.

If you divide this interval into, into n equal, sub-intervals so, if you divide this interval [a,b] into n equal parts, let us say, first part is x 1, x 0 to x 1, second part is x 1 to x 2, and so on, last part is x n minus 1 to x n. Then the sum of the, the area of this is going to be equal to sum of all these areas. So, first part is a, h is the length of this sub-interval, each of this sub-interval, so h by 2 into f 0 plus f 1.

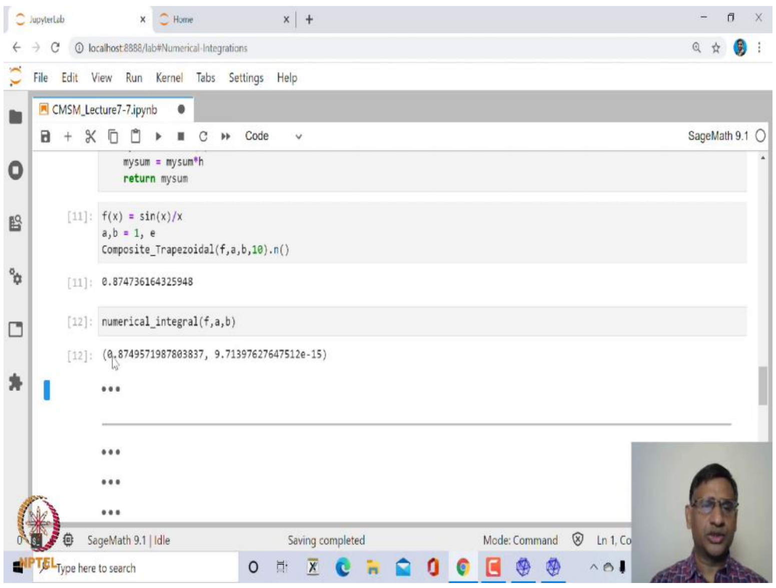
Next one will be h by 2 into f 1 plus f 2, and so on. Last one will be h by 2 into f n minus 1 plus f n, where f n, f i is f at x I, h is the b minus a upon n; and if you add all these things, the, except the initial f 0 and f n, all will come twice. So, this is what is called composite Trapezoidal rule of numerical integration.

So, let us create user-defined function for this, it is quite simple. So, I am giving the name composite Trapezoidal, you input f, the function, the endpoints a and b, and the number of sub-intervals. Number of intervals into which you want to divide this, right, so, h is equal to b minus a upon n, let us create this, this sum, I will call that as mysum.

So, initially, mysum is h by 2 into f 0, f 0 is f at a, and then for the remaining, it is add all these f 1, f 2, and f n. So, that is what is added here. And then, at the end, you add half of fn. So, here, instead of half of fi for i between 1 and n minus 1 we are just taking fi.

So, this is f 0 by a into h by 2, the remaining is f1 by f1 times h, f2 times h, and fn minus 1 times h, and the last one is fn. So, and then let us return this, let us run this.

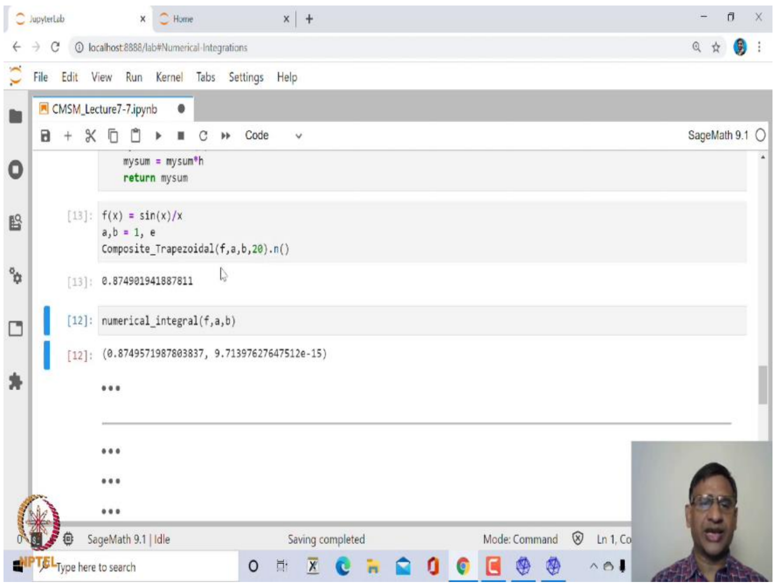
(Refer Slide Time: 09:30)



```
    mysum = mysum*h
    return mysum

[11]: f(x) = sin(x)/x
      a,b = 1, e
      Composite_Trapezoidal(f,a,b,10).n()

[11]: 0.874736164325948

[12]: numerical_integral(f,a,b)

[12]: (0.8749571987803837, 9.71397627647512e-15)
```

And let us now call this function and find integral of sin x upon x between 1 and the Euler number e using composited, composite Trapezoidal rule. And let us take 10 intervals where, and in case you obtain this integral using inbuilt function, this is what you get. So, this is 0.874957, actual value, and this case 0.8747 so, it is not very far, even 10 intervals has, has found using Trapezoidal rule, quite close to the actual value.

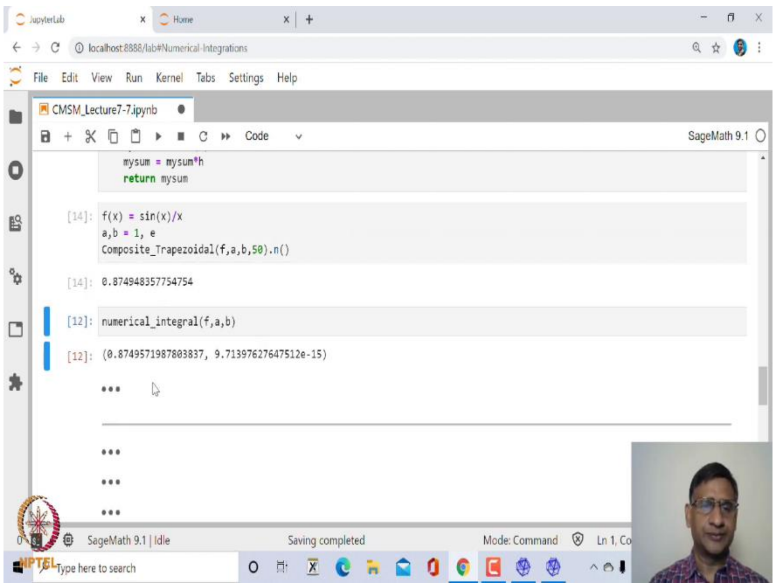
(Refer Slide Time: 10:12)



```
    mysum = mysum*h
    return mysum

[13]: f(x) = sin(x)/x
      a,b = 1, e
      Composite_Trapezoidal(f,a,b,20).n()

[13]: 0.874901941887811

[12]: numerical_integral(f,a,b)

[12]: (0.8749571987803837, 9.71397627647512e-15)
```

So, instead of 10, if I make it, let, for example 20, then this will be much closer to this actual value, or if I make it 50, these two will be much more closer, right?

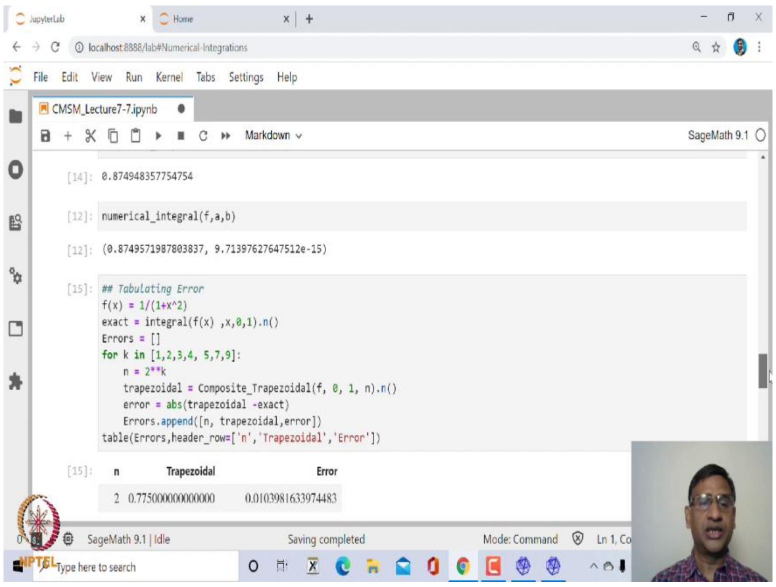
(Refer Slide Time: 10:18)



```
    mysum = mysum*h
    return mysum
```

```
[14]: f(x) = sin(x)/x
      a,b = 1, e
      Composite_Trapezoidal(f,a,b,50).n()

[14]: 0.874948357754754

[12]: numerical_integral(f,a,b)

[12]: (0.8749571987803837, 9.71397627647512e-15)
```

(Refer Slide Time: 10:22)



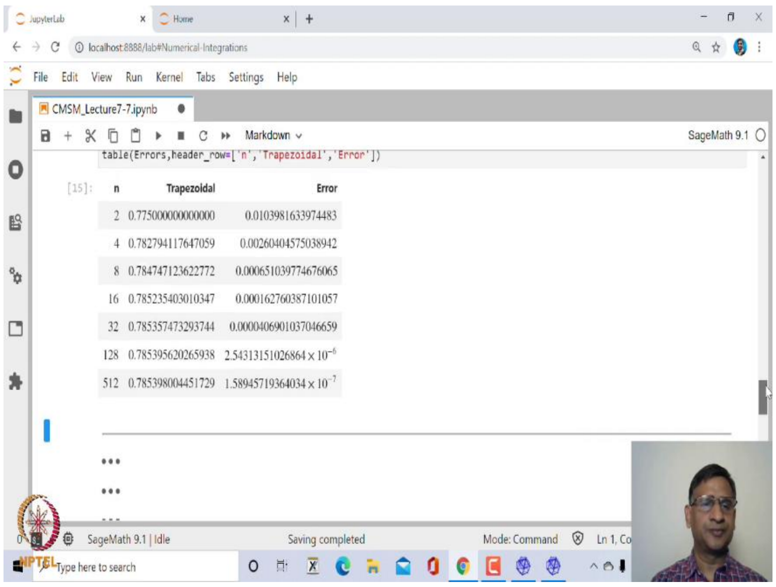
```
[14]: 0.874948357754754

[12]: numerical_integral(f,a,b)

[12]: (0.8749571987803837, 9.71397627647512e-15)

[15]: ## Tabulating Error
      f(x) = 1/(1+x^2)
      exact = integral(f(x) ,x,0,1).n()
      Errors = []
      for k in [1,2,3,4, 5,7,9]:
          n = 2**k
          trapezoidal = Composite_Trapezoidal(f, 0, 1, n).n()
          error = abs(trapezoidal -exact)
          Errors.append([n, trapezoidal,error])
      table(Errors,header_row=['n','Trapezoidal','Error'])
```

n	Trapezoidal	Error
2	0.775000000000000	0.0103981633974483

You can even tabulate what is the error term in each of this iteration. So, for example, we are taking the n, the number of interval as 2 to the power 1, 2 to the power 2, 2 to the power 3, 2 to the power 4, 2 to the power 5, 2 to the power 7, 2 to the power 9.

And find out the value of the integral using Trapezoidal rule and find out the error term, which is the absolute value of the, the integral computed using Trapezoidal rule and the exact integral. Exact integral, in this case, is the numerical integral of fx from 0 to 1, or you can write numerical underscore integral, both will work.

(Refer Slide Time: 10:58)



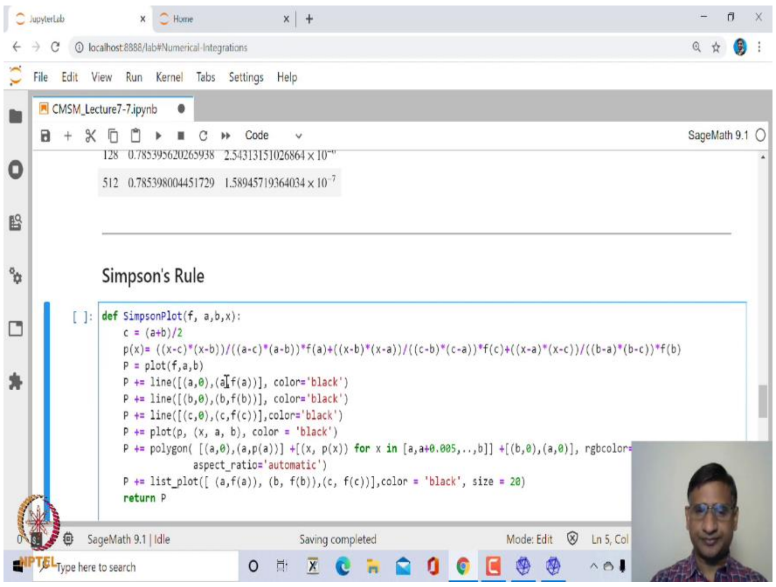
```
table(Errors,header_row=['n','Trapezoidal','Error'])
```

```
[15]:   n         Trapezoidal                    Error
        2   0.775000000000000      0.0103981633974483
        4   0.782794117647059      0.00260404575038942
        8   0.784747123622772      0.000651039774676065
       16   0.785235403010347      0.000162760387101057
       32   0.785357473293744      0.0000406901037046659
      128   0.785395620265938   2.54313151026864 × 10^-6
      512   0.785398004451729   1.58945719364034 × 10^-7
```

So, let me tabulate this error. So, you can see here, for n equal to 2 the error is quite large, for n equal to 4 it has reduced, and, and so on. For n is equal to 128, error is of the order 10 to the power minus 6, and for 512 error is 10 to the power minus 7.

So, it, the error is decreasing. Of course, this, the order of this error is not very high. So, this Trapezoidal rule, it is very simple rule of finding numerical integral, but in this case, error is, the, the convergence is a bit slow, right?

(Refer Slide Time: 11:37)



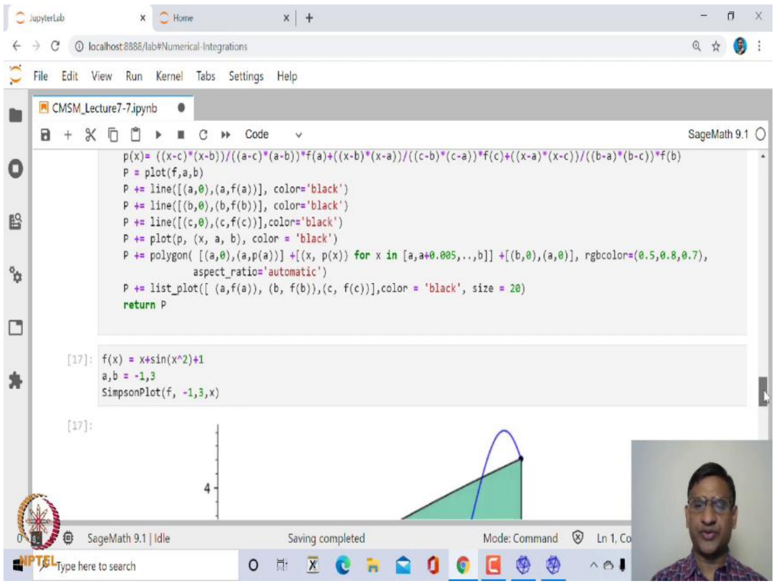
```
128   0.785395620265938   2.54313151026864 × 10^-6
512   0.785398004451729   1.58945719364034 × 10^-7
```

Simpson's Rule

```
[ ]: def SimpsonPlot(f, a,b,x):
         c = (a+b)/2
         p(x)= ((x-c)*(x-b))/((a-c)*(a-b))*f(a)+((x-b)*(x-a))/((c-b)*(c-a))*f(c)+((x-a)*(x-c))/((b-a)*(b-c))*f(b)
         P = plot(f,a,b)
         P += line([(a,0),(a,f(a))], color='black')
         P += line([(b,0),(b,f(b))], color='black')
         P += line([(c,0),(c,f(c))],color='black')
         P += plot(p, (x, a, b), color = 'black')
         P += polygon( [(a,0),(a,p(a))] +[(x, p(x)) for x in [a,a+0.005,..,b]] +[(b,0),(a,0)], rgbcolor=
                   aspect_ratio='automatic')
         P += list_plot([ (a,f(a)), (b, f(b)),(c, f(c))],color = 'black', size = 20)
         return P
```

So, now let us look at Simpsons one-third rule, and in this case, we approximate the the function by quadratic interpolating polynomial.

(Refer Slide Time: 11:52)

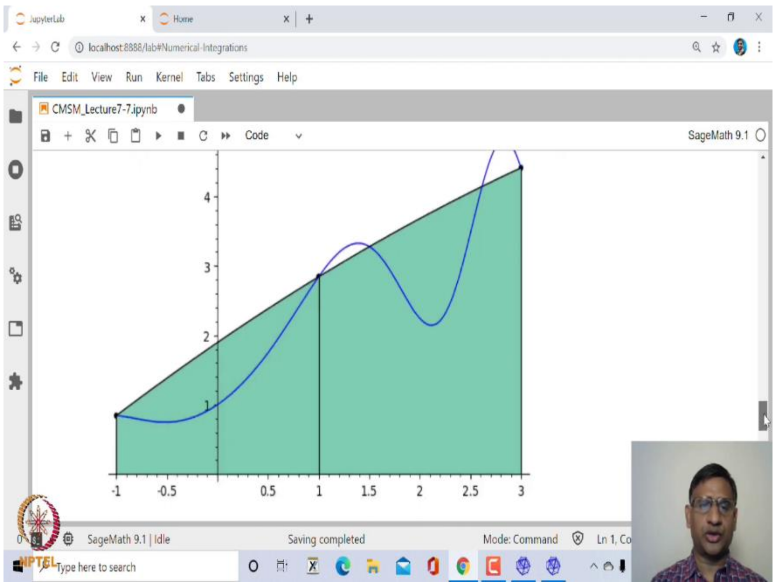


```
p(x)= ((x-c)*(x-b))/((a-c)*(a-b))*f(a)+((x-b)*(x-a))/((c-b)*(c-a))*f(c)+((x-a)*(x-c))/((b-a)*(b-c))*f(b)
P = plot(f,a,b)
P += line([(a,0),(a,f(a))], color='black')
P += line([(b,0),(b,f(b))], color='black')
P += line([(c,0),(c,f(c))],color='black')
P += plot(p, (x, a, b), color = 'black')
P += polygon( [(a,0),(a,p(a))] +[(x, p(x)) for x in [a,a+0.005,..,b]] +[(b,0),(a,0)], rgbcolor=(0.5,0.8,0.7),
          aspect_ratio='automatic')
P += list_plot([ (a,f(a)), (b, f(b)),(c, f(c))],color = 'black', size = 20)
return P
```

```
[17]: f(x) = x+sin(x^2)+1
      a,b = -1,3
      SimpsonPlot(f, -1,3,x)
```

So, let us, let me, let me look at what is this interpolating polynomial.

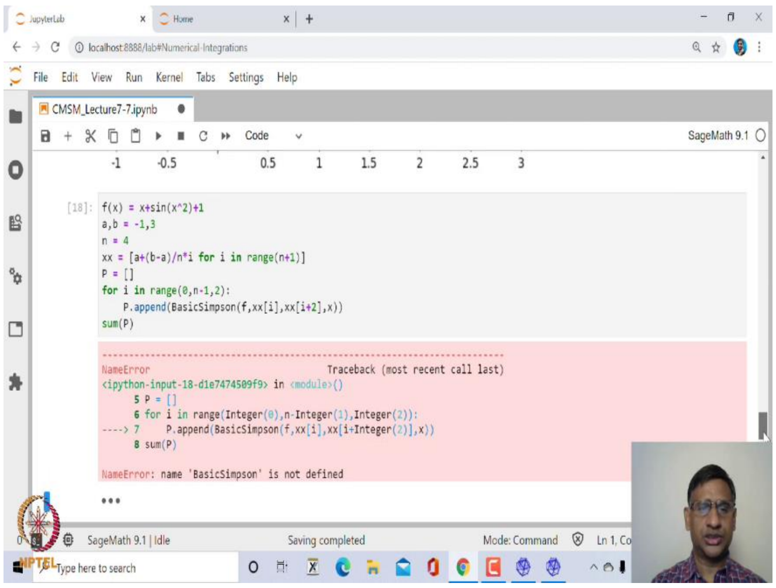
(Refer Slide Time: 11:58)



So, this is what you have, if you divide this in, so, if I want to fit interpolating, quadratic interpolating polynomial, we need 3 points. So, what do we do? With a comma a, between a and b, we divide this into 2 equal parts, and take these three points, this, this and these three points, let me reduce the figure size so that it will be ok, this is alright. So, and then we fit into Lagrange interpolating polynomial, quadratic interpolating polynomial to these three points. In this case, what we have done? This is the Lagrange interpolating polynomial fitted to the points (a,fa), (b,fb) and (c,fc), where c is the midpoint of a and b.

So, this is quite simple Sage routine to plot the, the graph of the function along with the fitted interpolating polynomial, quadratic interpolating polynomial, and it is shading the area under this interpolating polynomial, right?

(Refer Slide Time: 13:05)

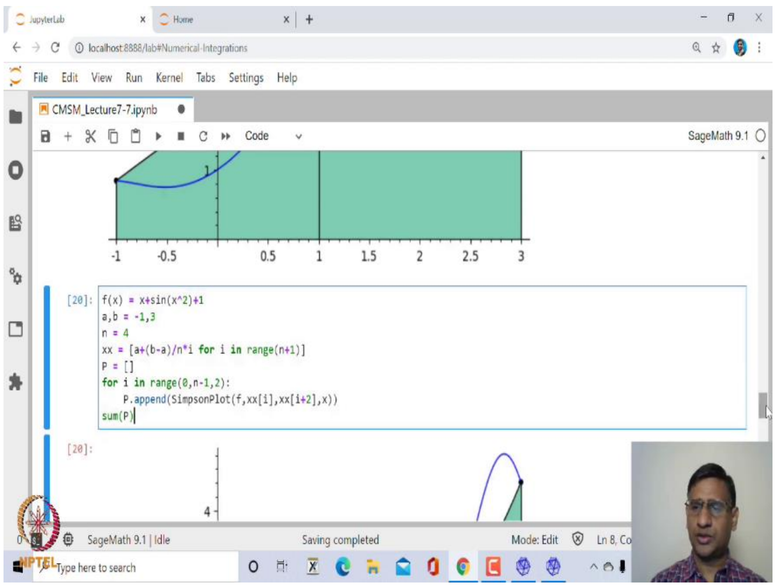


```
[18]: f(x) = x+sin(x^2)+1
      a,b = -1,3
      n = 4
      xx = [a+(b-a)/n*i for i in range(n+1)]
      P = []
      for i in range(0,n-1,2):
          P.append(BasicSimpson(f,xx[i],xx[i+2],x))
      sum(P)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-d1e7474509f9> in <module>()
      5 P = []
      6 for i in range(Integer(0),n-Integer(1),Integer(2)):
----> 7     P.append(BasicSimpson(f,xx[i],xx[i+Integer(2)],x))
      8 sum(P)

NameError: name 'BasicSimpson' is not defined
```

Now, let us look at, we increase this number of sub-intervals. So, suppose we increase, instead of dividing into 2 equal parts, let us divide into 4 equal parts. Of course, in this case, number of parts will be required to be even. So, in this case, when you try to, just a second, basic Simpson, Simpson plot, I have called, yeah.

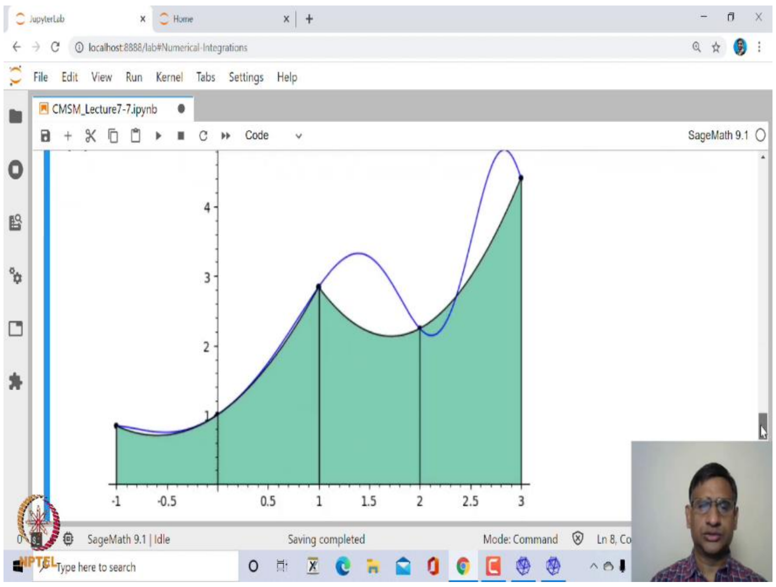
(Refer Slide Time: 13:34)



```
[20]: f(x) = x+sin(x^2)+1
      a,b = -1,3
      n = 4
      xx = [a+(b-a)/n*i for i in range(n+1)]
      P = []
      for i in range(0,n-1,2):
          P.append(SimpsonPlot(f,xx[i],xx[i+2],x))
      sum(P)
```

localhost:8888/lab#Numerical-Integrations
File Edit View Run Kernel Tabs Settings Help
CMSM_Lecture7-7.ipynb
Code
SageMath 9.1
SageMath 9.1 | Idle
Saving completed
Mode: Edit

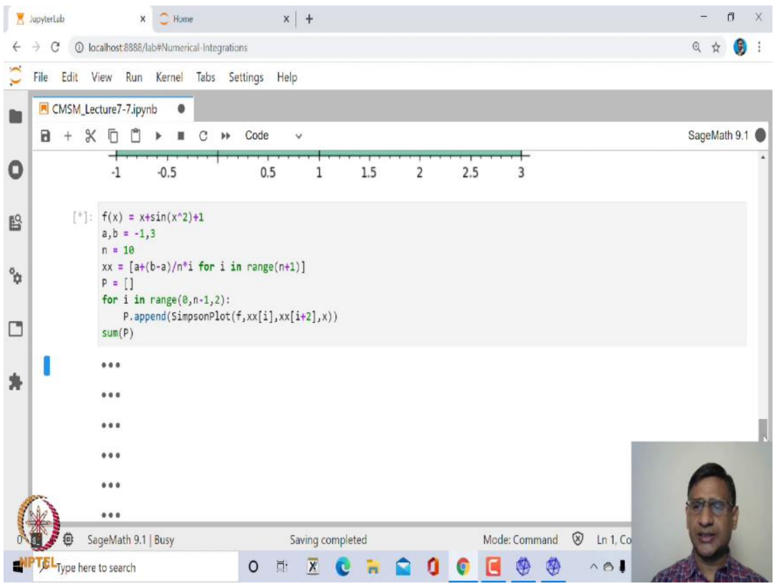
(Refer Slide Time: 13:46)



localhost:8888/lab#Numerical-Integrations
File Edit View Run Kernel Tabs Settings Help
CMSM_Lecture7-7.ipynb
SageMath 9.1
SageMath 9.1 | Idle
Saving completed
Mode: Command

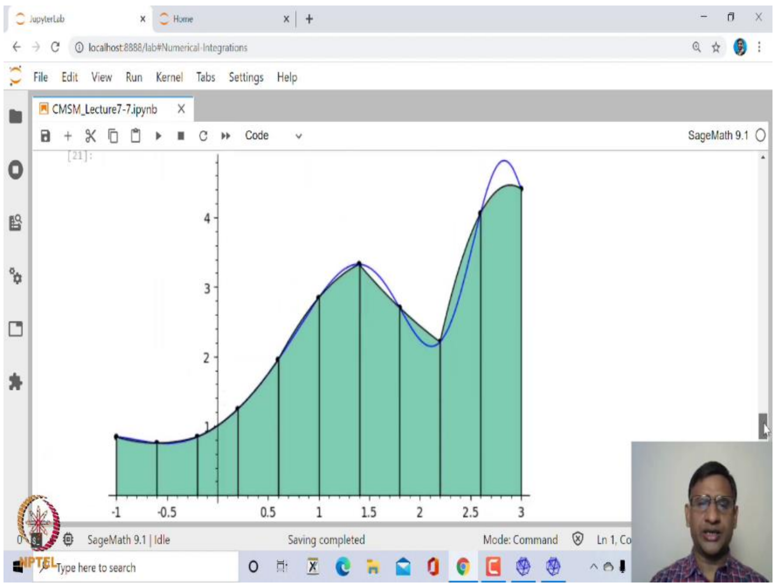
So, Simpson plot, what happened? Simpson plot, yeah. So, in this case now, you can see here, we divided this interval into 4 equal parts. So, this is the first two parts. So, we are fitting interpolating polynomial between these three points this, this and this, and then next, we are fitting interpolating polynomial between these three points, and so on.

(Refer Slide Time: 14:11)



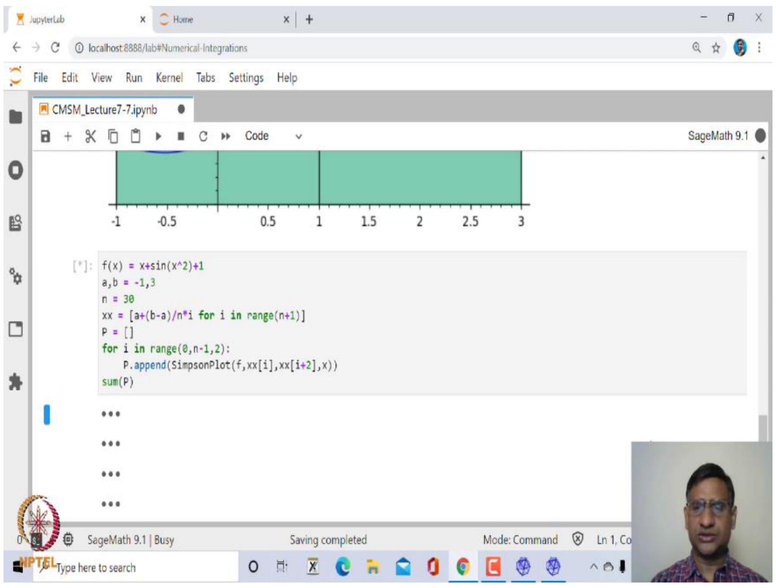
```
[*]: f(x) = x+sin(x^2)+1
     a,b = -1,3
     n = 10
     xx = [a+(b-a)/n*i for i in range(n+1)]
     P = []
     for i in range(0,n-1,2):
         P.append(SimpsonPlot(f,xx[i],xx[i+2],x))
     sum(P)
```

(Refer Slide Time: 14:14)



So, now instead of 4, let us make it 10 equal parts, and in this case, now you can see here, the value of the integral, actual integral and approximate integral will become much closer.

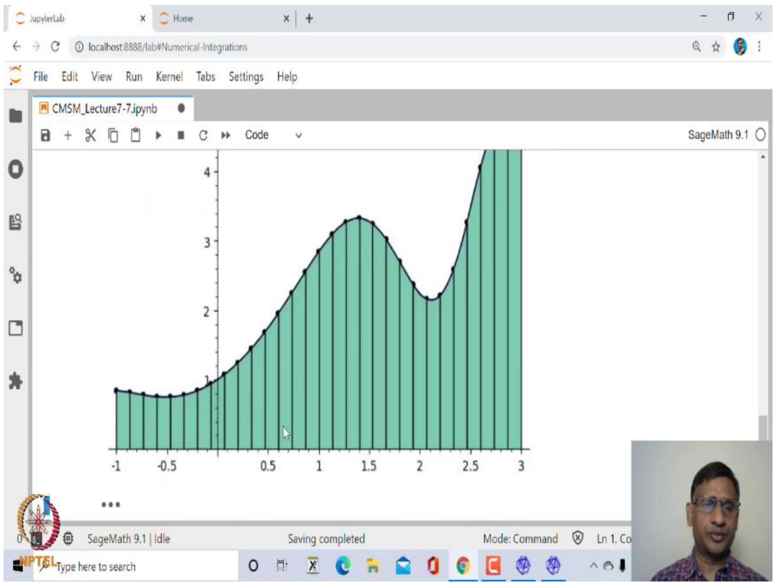
(Refer Slide Time: 14:28)



```
[*]: f(x) = x+sin(x^2)+1
     a,b = -1,3
     n = 30
     xx = [a+(b-a)/n*i for i in range(n+1)]
     P = []
     for i in range(0,n-1,2):
         P.append(SimpsonPlot(f,xx[i],xx[i+2],x))
     sum(P)
```

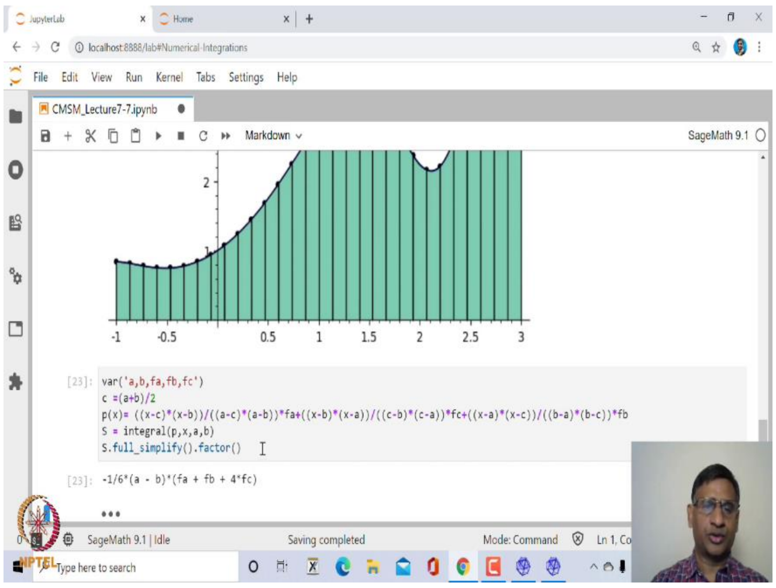
And instead of 10 parts, let us make it 30 parts. So, we divide into 30 equal parts, and yeah.

(Refer Slide Time: 14:34)



So, what you have to do? When you are dividing into 30 equal parts, you take first two parts and in that you fit a quadratic interpolating polynomial, take the next part, fit the inter, quadratic interpolating polynomial, and so on.

(Refer Slide Time: 14:53)

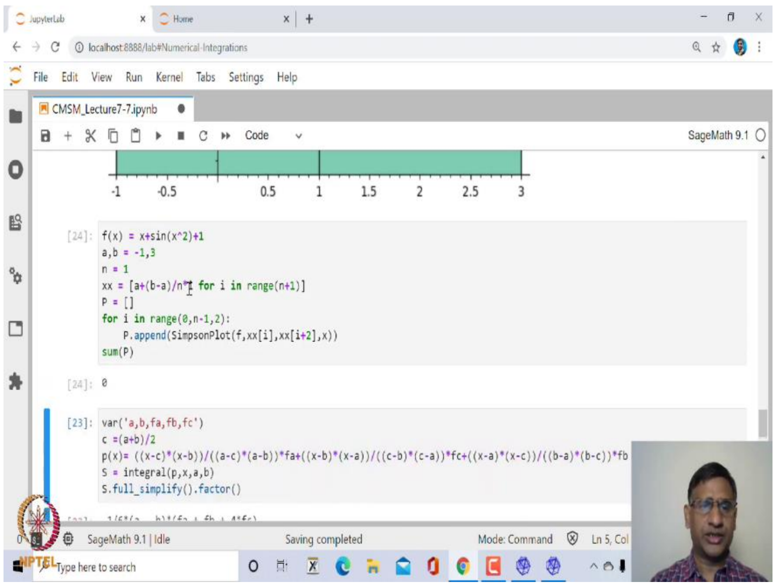


```
[23]: var('a,b,fa,fb,fc')
      c =(a+b)/2
      p(x)= ((x-c)*(x-b))/((a-c)*(a-b))*fa+((x-b)*(x-a))/((c-b)*(c-a))*fc+((x-a)*(x-c))/((b-a)*(b-c))*fb
      S = integral(p,x,a,b)
      S.full_simplify().factor()

[23]: -1/6*(a - b)*(fa + fb + 4*fc)
```

So, now let us look at what happens to this integral. Suppose, so, first approximation was, if you divide this interval [a,b] into 3, 2 equal parts, [a,b], [a,b], and c is midpoint of [a,b], and then when we fit this, this polynomial and try to find out.

So, this is the polynomial, quadratic interpolating polynomial, and then integrate this between a and b, and then if you try to simplify, this gives you minus 1 by 6 into a minus b, which you can thought of as b minus a divided by 6 into fa plus fb plus 4 times fc.

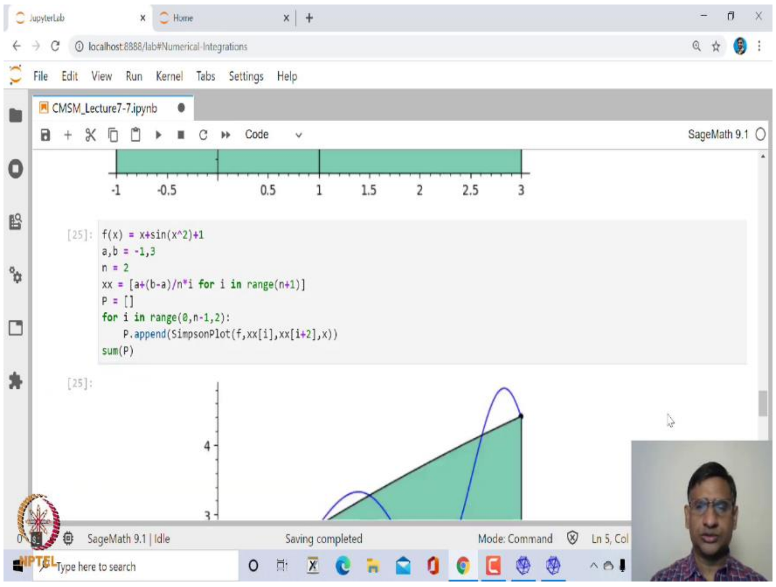
(Refer Slide Time: 15:44)



```
[24]: f(x) = x+sin(x^2)+1
      a,b = -1,3
      n = 1
      xx = [a+(b-a)/n*i for i in range(n+1)]
      P = []
      for i in range(0,n-1,2):
          P.append(SimpsonPlot(f,xx[i],xx[i+2],x))
      sum(P)

[24]: 0

[23]: var('a,b,fa,fb,fc')
      c =(a+b)/2
      p(x)= ((x-c)*(x-b))/((a-c)*(a-b))*fa+((x-b)*(x-a))/((c-b)*(c-a))*fc+((x-a)*(x-c))/((b-a)*(b-c))*fb
      S = integral(p,x,a,b)
      S.full_simplify().factor()
```

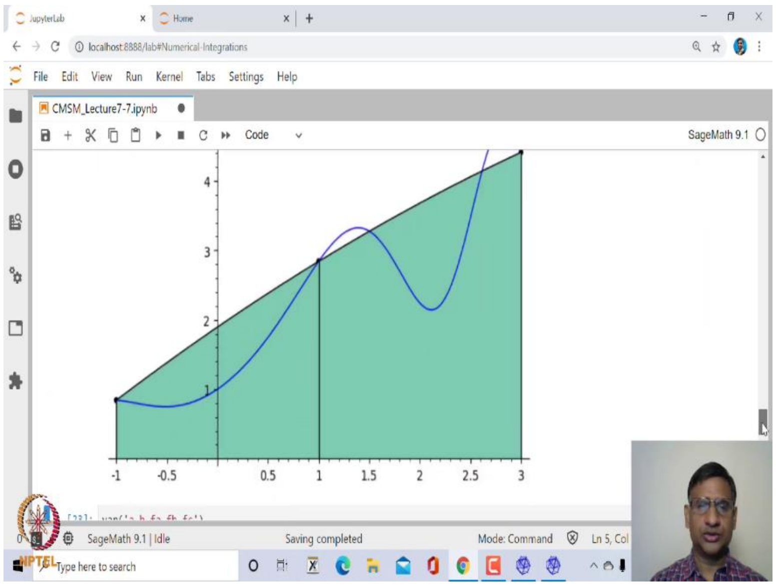
So, 4 times fc, so what is it? Let me, let me, let me make it 1 here, so, that, ok.

(Refer Slide Time: 15:50)



```
[25]: f(x) = x+sin(x^2)+1
      a,b = -1,3
      n = 2
      xx = [a+(b-a)/n*i for i in range(n+1)]
      P = []
      for i in range(0,n-1,2):
          P.append(SimpsonPlot(f,xx[i],xx[i+2],x))
      sum(P)
```

(Refer Slide Time: 15:53)

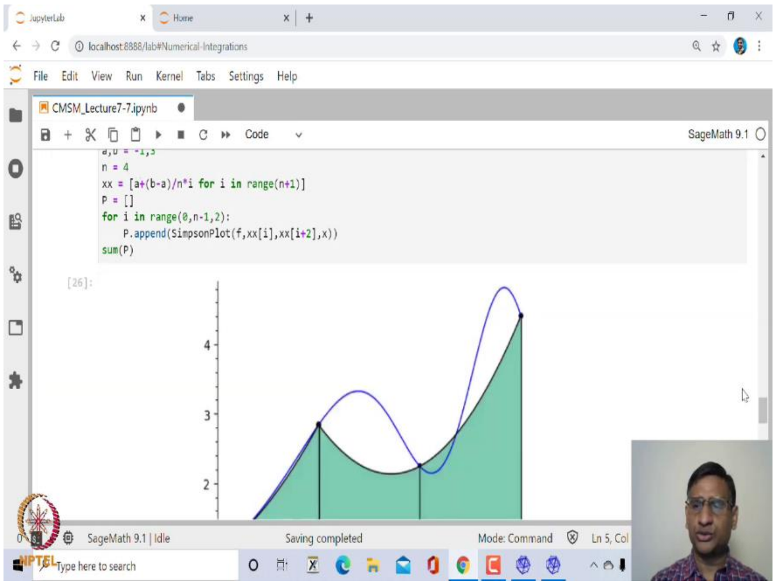


So, yeah no, this is 2. So, this is what you have. So, this area under this fitted cubic you are plotting, and this is your fa, this is fc, this is fb, this height is fb, and what we have obtained? This, in this case, area is given by, when you approximate this function fx by quadratic interpolating polynomial.

The area under this quadratic interpolating polynomial between a and b is given by 1 upon 6.

So, now 1 upon 6 in this case the, the length of each interval is half b minus a. So, b minus a upon 6 can be thought of as h by 3, where h is the length of this each sub-interval. that is b minus a by 2, in this case, plus fa plus fb plus 4 times fc.

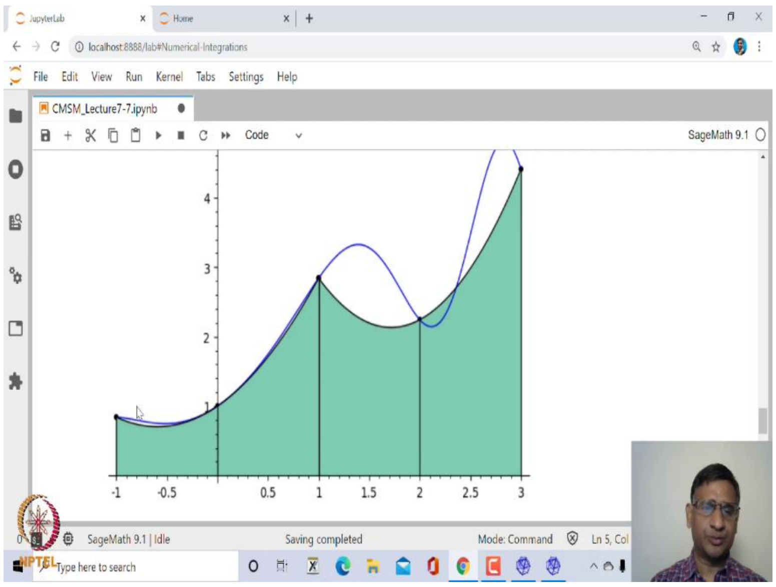
(Refer Slide Time: 16:54)



```
a,b = -1,3
n = 4
xx = [a+(b-a)/n*i for i in range(n+1)]
P = []
for i in range(0,n-1,2):
    P.append(SimpsonPlot(f,xx[i],xx[i+2],x))
sum(P)
```

Now, when you, when you, when you add these, now when you, for example, divide this into 4 equal parts.

(Refer Slide Time: 16:57)

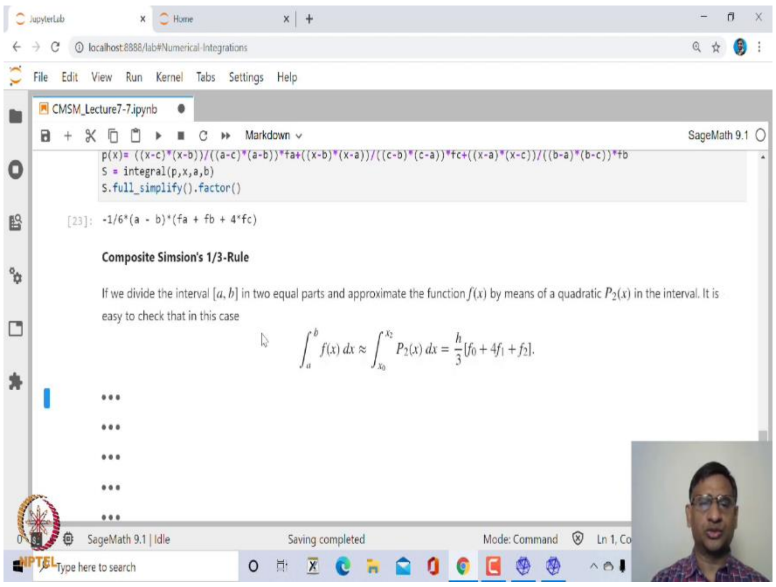


Now, what you have to do is, you need to add this portion and plus this portion.

So, what will this, this portion, the area under this portion is going to be? h by 3 times f, f at a, or let us say f at x 0 plus f at x 2 plus 4 times f at x 1. The next one will be f at h by 4, h by 3 times f at x 1, sorry, f at x 2 plus f at 4 times f at x 2 plus f at x 4 f at x 3 in this case.

So, this is x 0, x 1, x 2, x 3, x 4. So, this will be h by 4 times f0 plus 4 times f at 1 plus f at 2; next one will be f at h 0 by 3, h by 3 into f 0 plus 4 times f2, not f 0, it is f 2, f2, this is f3, and this is f 4, right?

(Refer Slide Time: 18:12)



```
p(x)= ((x-c)*(x-b))/((a-c)*(a-b))*fa+((x-b)*(x-a))/((c-b)*(c-a))*fc+((x-a)*(x-c))/((b-a)*(b-c))*fb
S = integral(p,x,a,b)
S.full_simplify().factor()
```

```
[23]: -1/6*(a - b)*(fa + fb + 4*fc)
```

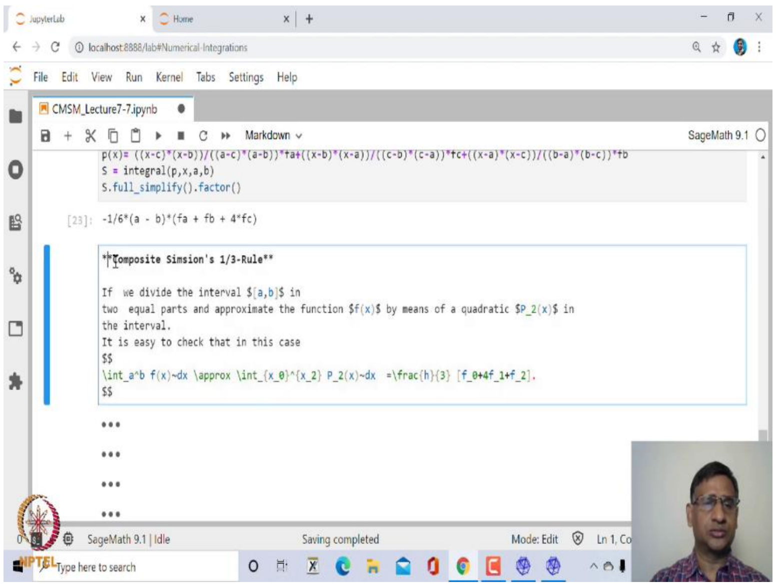
Composite Simsion's 1/3-Rule

If we divide the interval $[a, b]$ in two equal parts and approximate the function $f(x)$ by means of a quadratic $P_2(x)$ in the interval. It is easy to check that in this case

$$\int_a^b f(x)\,dx \approx \int_{x_0}^{x_2} P_2(x)\,dx = \frac{h}{3}[f_0 + 4f_1 + f_2].$$

So, if you add all these things together you get what is called Composite Simpsons rule.

(Refer Slide Time: 18:18)



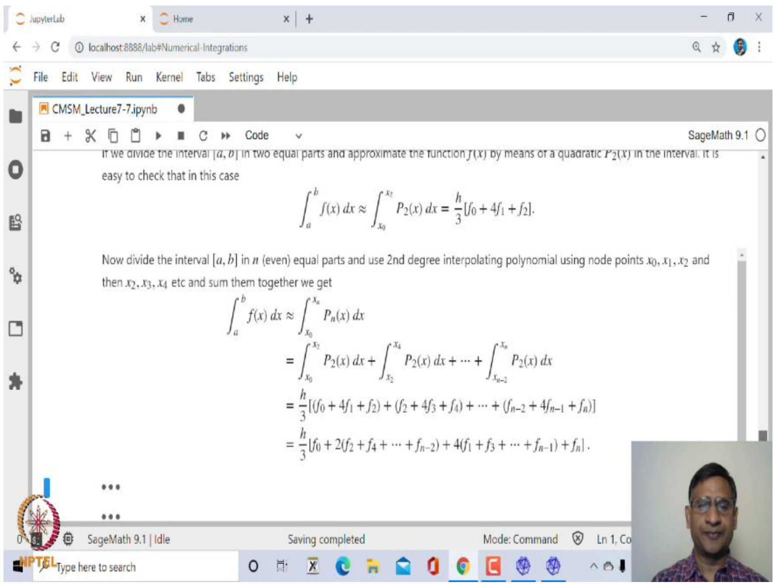
```
p(x)= ((x-c)*(x-b))/((a-c)*(a-b))*fa+((x-b)*(x-a))/((c-b)*(c-a))*fc+((x-a)*(x-c))/((b-a)*(b-c))*fb
S = integral(p,x,a,b)
S.full_simplify().factor()
```

```
[23]: -1/6*(a - b)*(fa + fb + 4*fc)
```

```
**Composite Simsion's 1/3-Rule**

If  we divide the interval $[a,b]$ in
two  equal parts and approximate the function $f(x)$ by means of a quadratic $P_2(x)$ in
the interval.
It is easy to check that in this case
$$
\int_a^b f(x)~dx \approx \int_{x_0}^{x_2} P_2(x)~dx  =\frac{h}{3} [f_0+4f_1+f_2].
$$
```

So, composite Simpsons rule. So, integral of fx from a to b, if you are dividing this interval into n equal parts, this is equal to x 0 to x 2 if you are dividing into 2 equal parts, x 0 to x 2, this is equal to h by 3 into f 0 plus 4 times f 1 plus f2.

(Refer Slide Time: 18:40)



If we divide the interval $[a,b]$ in two equal parts and approximate the function $f(x)$ by means of a quadratic $P_2(x)$ in the interval. It is easy to check that in this case

$$\int_a^b f(x)\,dx \approx \int_{x_0}^{x_2} P_2(x)\,dx = \frac{h}{3}[f_0+4f_1+f_2].$$

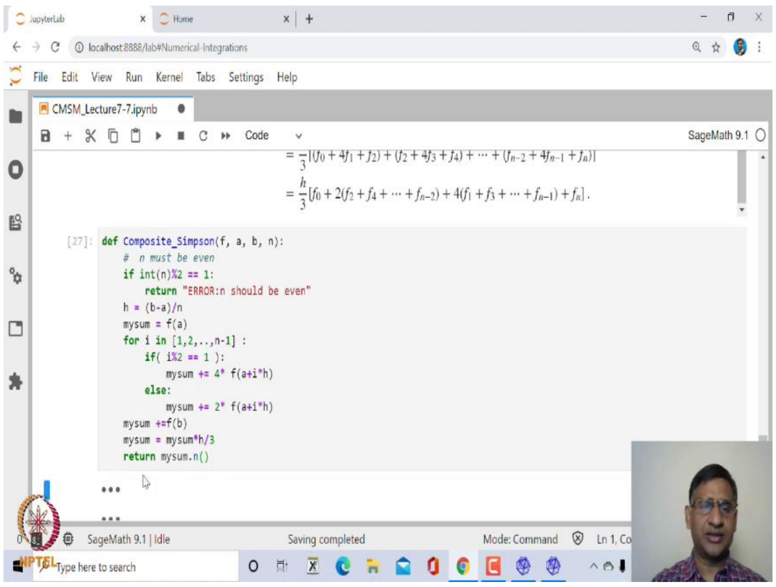
Now divide the interval $[a,b]$ in n (even) equal parts and use 2nd degree interpolating polynomial using node points x_0, x_1, x_2 and then x_2, x_3, x_4 etc and sum them together we get

$$\begin{aligned}\int_a^b f(x)\,dx &\approx \int_{x_0}^{x_n} P_n(x)\,dx \\ &= \int_{x_0}^{x_2} P_2(x)\,dx + \int_{x_2}^{x_4} P_2(x)\,dx + \cdots + \int_{x_{n-2}}^{x_n} P_2(x)\,dx \\ &= \frac{h}{3}[(f_0+4f_1+f_2)+(f_2+4f_3+f_4)+\cdots+(f_{n-2}+4f_{n-1}+f_n)] \\ &= \frac{h}{3}[f_0+2(f_2+f_4+\cdots+f_{n-2})+4(f_1+f_3+\cdots+f_{n-1})+f_n].\end{aligned}$$

And if you, if you generalize this, that is, when you add this, divide this interval into n equal parts, then how will you, what will be the integral, approximate integral? You integrate this approximate integral, approximate interpolating polynomial from x 0 to x 2, then from x 2 to x 4, and so on, x n minus 2 to x n, add all these things.

So, then this gives you a formula h by 3 into f 0 plus 2 times all the even f i's plus 4 times all odd f i's and then last one is fn alone. So, this is what is called Simpsons one-third rule, or composite Simpson's one-third rule.

(Refer Slide Time: 19:26)



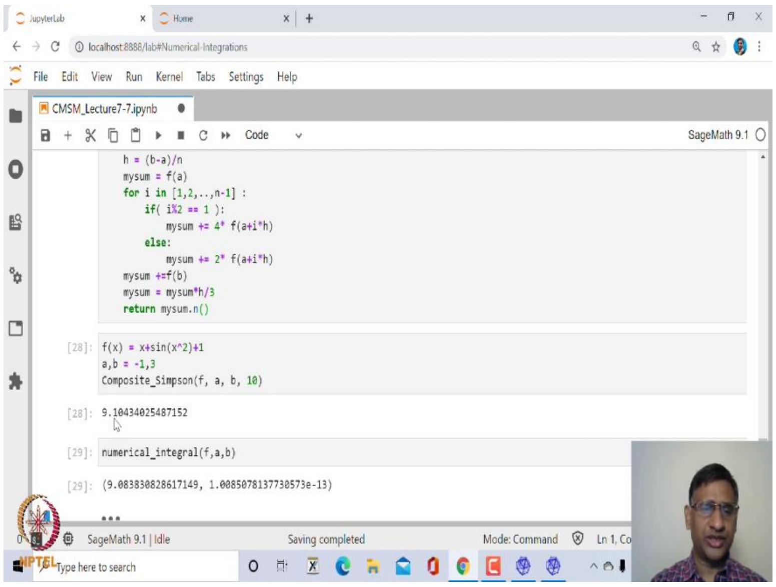
$$= \frac{h}{3}[(f_0 + 4f_1 + f_2) + (f_2 + 4f_3 + f_4) + \cdots + (f_{n-2} + 4f_{n-1} + f_n)]$$

$$= \frac{h}{3}[f_0 + 2(f_2 + f_4 + \cdots + f_{n-2}) + 4(f_1 + f_3 + \cdots + f_{n-1}) + f_n].$$

```
[27]: def Composite_Simpson(f, a, b, n):
          #  n must be even
          if int(n)%2 == 1:
              return "ERROR:n should be even"
          h = (b-a)/n
          mysum = f(a)
          for i in [1,2,..,n-1] :
              if( i%2 == 1 ):
                  mysum += 4* f(a+i*h)
              else:
                  mysum += 2* f(a+i*h)
          mysum +=f(b)
          mysum = mysum*h/3
          return mysum.n()
```

So, now let us write a Sage routine for this, again it is quite simple. First of all, this number of interval has to be even, so in case it is odd, you just throw an, an error saying that n should be even, and h is b minus a by n, and then declare mysum as fa which is f 0 here, and then add whenever i is divisible by by 2, you add 2 times this, 2 times, if it is not divisible by 2, then add 4 times, and then last one, you add f a, and then whole thing you divide by, multiply by h by 3, and return the numerical value of this.

(Refer Slide Time: 20:09)

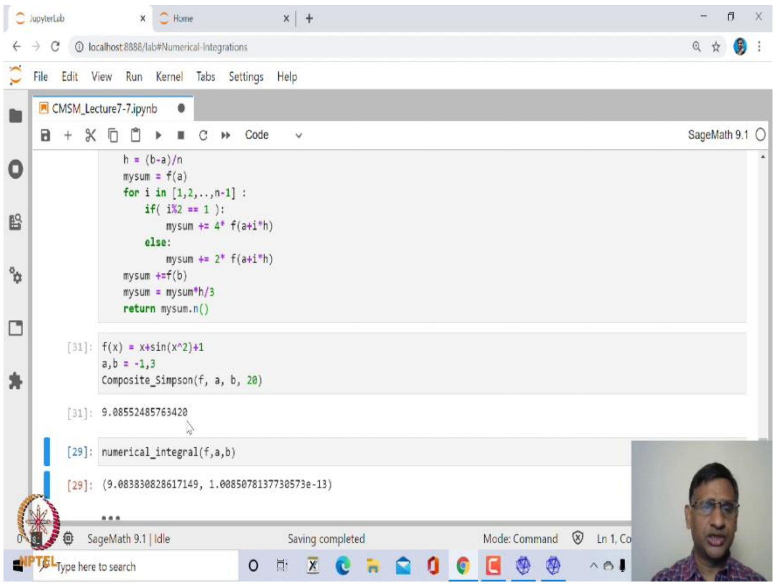


```
    h = (b-a)/n
    mysum = f(a)
    for i in [1,2,..,n-1] :
        if( i%2 == 1 ):
            mysum += 4* f(a+i*h)
        else:
            mysum += 2* f(a+i*h)
    mysum +=f(b)
    mysum = mysum*h/3
    return mysum.n()
```

```
[28]: f(x) = x+sin(x^2)+1
      a,b = -1,3
      Composite_Simpson(f, a, b, 10)

[28]: 9.10434025487152

[29]: numerical_integral(f,a,b)

[29]: (9.083830828617149, 1.0085078137730573e-13)
```

So, now, let us call this function, and in this case, about 10 interval it has got this value, and actual interval in integral is 9.0838 and you can see here.

(Refer Slide Time: 20:27)

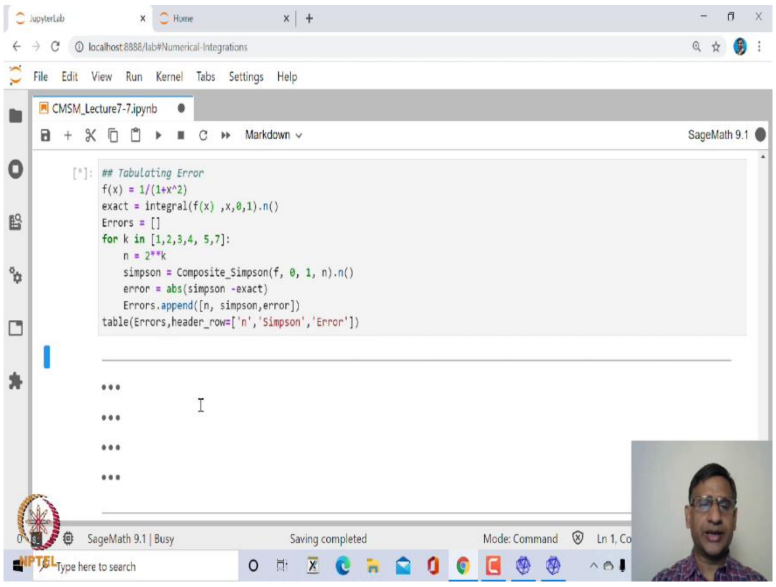


```
    h = (b-a)/n
    mysum = f(a)
    for i in [1,2,..,n-1] :
        if( i%2 == 1 ):
            mysum += 4* f(a+i*h)
        else:
            mysum += 2* f(a+i*h)
    mysum +=f(b)
    mysum = mysum*h/3
    return mysum.n()
```

```
[31]: f(x) = x+sin(x^2)+1
      a,b = -1,3
      Composite_Simpson(f, a, b, 20)

[31]: 9.08552485763420

[29]: numerical_integral(f,a,b)

[29]: (9.083830828617149, 1.0085078137730573e-13)
```

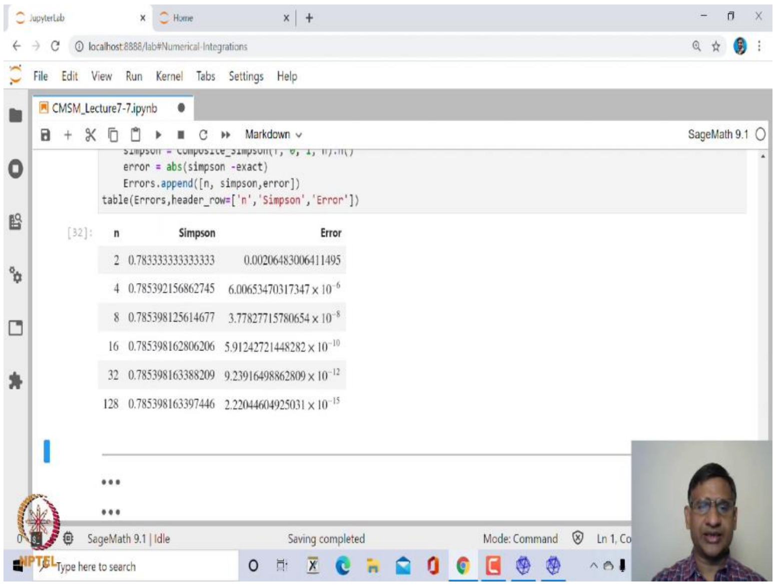
So in case now, instead of 10, if I give, sorry 200 is quite large, 20 then, you can see here ,these 2 integral are also much closer. And again you can, you can tabulate the error term.

(Refer Slide Time: 20:41)



```
[*]: ## Tabulating Error
     f(x) = 1/(1+x^2)
     exact = integral(f(x) ,x,0,1).n()
     Errors = []
     for k in [1,2,3,4, 5,7]:
         n = 2**k
         simpson = Composite_Simpson(f, 0, 1, n).n()
         error = abs(simpson -exact)
         Errors.append([n, simpson,error])
     table(Errors,header_row=['n','Simpson','Error'])
```

(Refer Slide Time: 20:42)

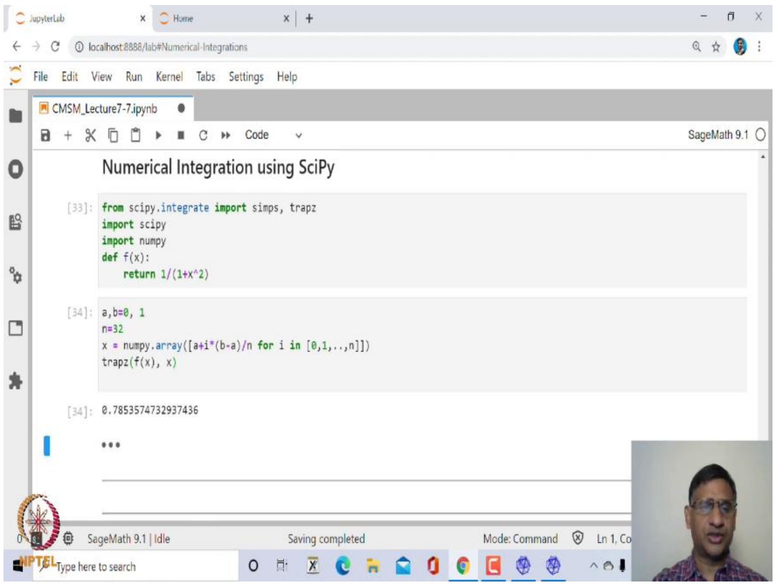


```
         error = abs(simpson -exact)
         Errors.append([n, simpson,error])
     table(Errors,header_row=['n','Simpson','Error'])
```

```
[32]:   n            Simpson                         Error
        2  0.783333333333333           0.00206483006411495
        4  0.785392156862745   6.00653470317347 × 10^-6
        8  0.785398125614677   3.77827715780654 × 10^-8
       16  0.785398162806206   5.91242721448282 × 10^-10
       32  0.785398163388209   9.23916498862809 × 10^-12
      128  0.785398163397446   2.22044604925031 × 10^-15
```

And one can show that the error term in this case is much more, much smaller. So, Simpsons three, one-third rule will be actually much better than the Trapezoidal rule. So, you can see here, in, in, in, in terms of four interval itself, the, the error is 10 to the power minus 6, and it is increasing quite rapid, decreasing quite rapidly, right?

(Refer Slide Time: 21:14)



Numerical Integration using SciPy

```
[33]: from scipy.integrate import simps, trapz
      import scipy
      import numpy
      def f(x):
          return 1/(1+x^2)
```

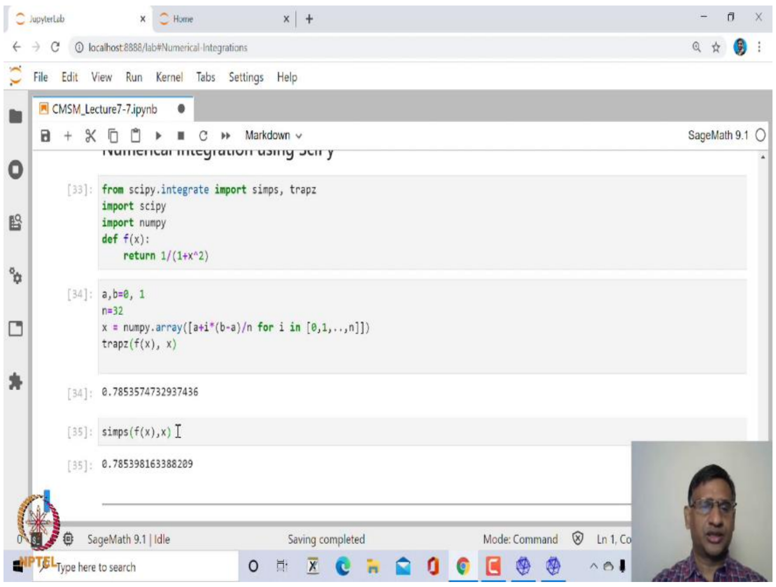
```
[34]: a,b=0, 1
      n=32
      x = numpy.array([a+i*(b-a)/n for i in [0,1,..,n]])
      trapz(f(x), x)
```

```
[34]: 0.7853574732937436
```

So, similarly you could use SciPy in order to compute numerical integral. So, how do we do that? Inside the SciPy there is an integrate function, integrate module, and inside that you have options, simps for Simpsons and trapz for Trapezoidal rule. So, let us look at, I have this function fx equal to 1 upon 1 plus x square.

And if you want to integrate this; let us say take n equal to 32, then what do we do? First, we need to find out what are these x values and then fit, call this trapeze, trapz fx over these points x. So, this is what we get using Trapezoidal rule, and using inbuilt scipy function trap, trapz.

(Refer Slide Time: 22:07)

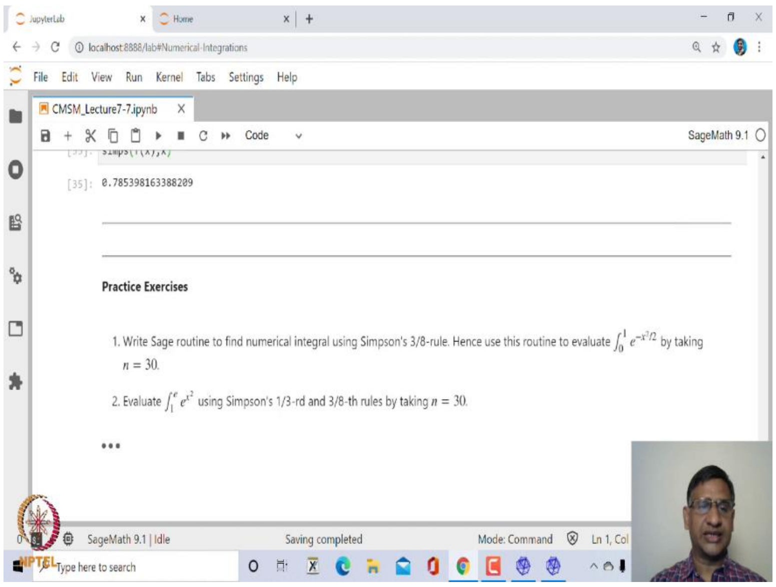


```
[33]: from scipy.integrate import simps, trapz
      import scipy
      import numpy
      def f(x):
          return 1/(1+x^2)

[34]: a,b=0, 1
      n=32
      x = numpy.array([a+i*(b-a)/n for i in [0,1,..,n]])
      trapz(f(x), x)

[34]: 0.7853574732937436

[35]: simps(f(x),x)

[35]: 0.785398163388209
```

Similarly, you can use simps this, right? So, you can use either scipy or you can use, you can create your own function. So, there are many other methods of finding numerical integral and, for example, you can, you can use, instead of 3, one-third rule you can, Simpsons 1 3rd rule, you could use Simpsons three-eighth rule which uses a cubic polynomial to approximate the function in a given interval.

So, in that case you will need 3, 4 points in order to fit a cubic interpolating polynomial.

(Refer Slide Time: 22:55)



```
[35]: 0.785398163388209
```

Practice Exercises

1. Write Sage routine to find numerical integral using Simpson's 3/8-rule. Hence use this routine to evaluate $\int_0^1 e^{-x^2/2}$ by taking $n = 30$.
2. Evaluate $\int_1^e e^{x^2}$ using Simpson's 1/3-rd and 3/8-th rules by taking $n = 30$.

So, I, I, I will leave that as an exercise . Write Sage routine to find numerical integral using Simpsons three-eighth rule, and hence use this routine to evaluate this integral by taking n equal to 30. It, in this case, the n has to be divisible by 3.

And similarly, evaluate this integral from 0 to e of e to the power x square using Simpsons three-eighth rule and Simpsons one-third rule. You could also again tabulate the error term, right? There are many other methods actually, for example, Gauss quadrature formula and other things, etcetera.

But, we will not look at all these methods. I am sure you should be able to write Sage routine for any of this numerical integral method which you have you have learnt. So, let me stop here.