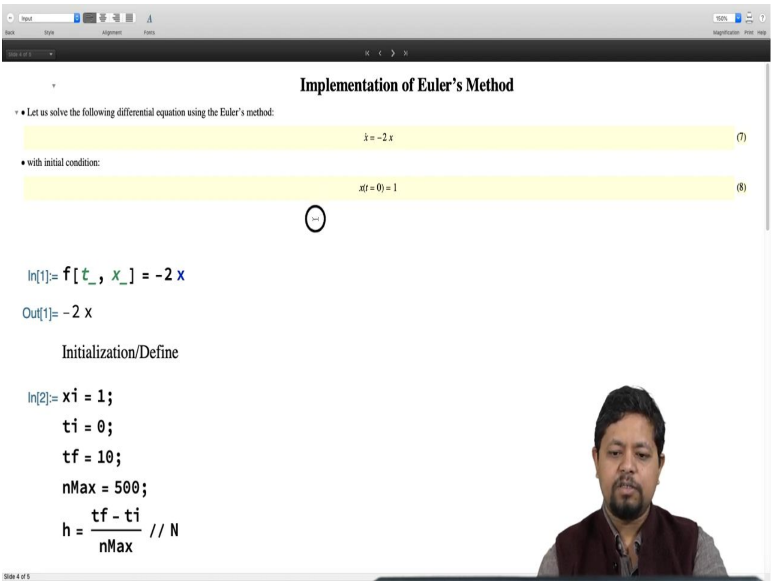
Physics through Computational Thinking
Professor Ambar Jain
Department of Physics
Indian Institute of Science Education and Research, Bhopal
Lecture 27
Writing Euler's Method as a Custom Function

(Refer Slide Time: 00:27)



Welcome back, last time we learned how to implement Eulers method, we wrote down our code using the for loop, let us quickly review that, let us take a differential equation $\dot{x}=-2x$, we want to find x as a function of time and we use initial condition that $x(t=0)=1$.

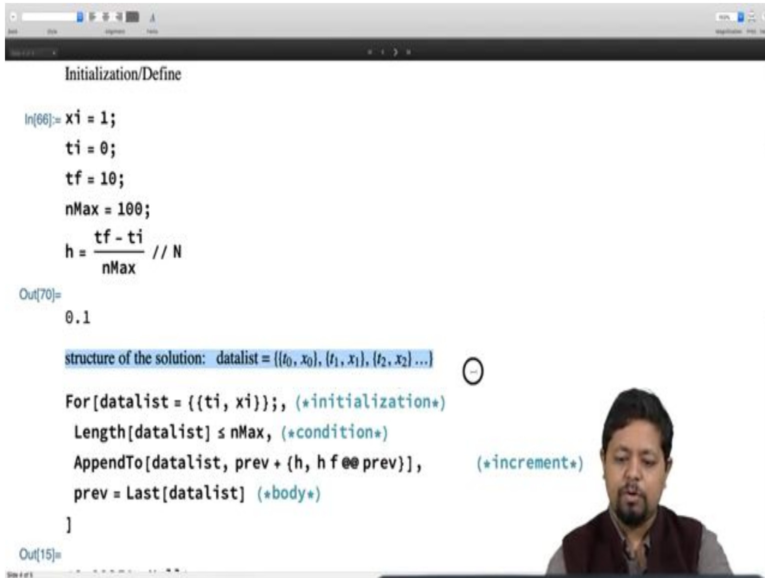
So therefore we define the derivative function that is $\dot{x}=f$ as $f(t,x)=-2x$, f in general can be a function of both t and x in this case special case it is only a function of x, so we defined $f(t,x)=-2x$.

(Refer Slide Time: 01:02)

```
Out[1]= -2 x

Initialization/Define

In[66]:= xi = 1;
ti = 0;
tf = 10;
nMax = 100;
h = (tf - ti)/nMax // N
Out[70]=
0.1

structure of the solution:  datalist = {{t0, x0}, {t1, x1}, {t2, x2} ...}

For[datalist = {{ti, xi}};, (*initialization*)
 Length[datalist] ≤ nMax, (*condition*)
 AppendTo[datalist, prev + {h, h f @@ prev}],     (*increment*)
 prev = Last[datalist] (*body*)
```

Then we initialize the certain variables number code x initial, xi = 1, t initial, ti = 0, t final, tf = 10 and we decided nMax equal to some number of steps, let us start out by let us say 100 steps and then we calculate $h = \frac{tf - ti}{nMax}$, h is the step size, we evaluated this we got our step size $h = 0.1$ this is the only statement we are printing the others are suppressed.

(Refer Slide Time: 01:35)



$$h = \frac{tf - ti}{nMax} \; // \; N$$

Out[70]=

0.1

structure of the solution: datalist = $\{\{t_0, x_0\}, \{t_1, x_1\}, \{t_2, x_2\} \ldots\}$

```
For[datalist = {{ti, xi}};, (*initialization*)
 Length[datalist] ≤ nMax + 1, (*condition*)
 AppendTo[datalist, prev + {h, h f @@ prev}],        (*increment*)
 prev = Last[datalist] (*body*)
]
```

Out[15]=

{0.00251, Null}

In[72]:= h f @@ {0.1, 2}

Out[72]=

-0.4

In[11]:= datalist

Out[74]=

{{0, 1}, {0.1, 0.8}, {0.2, 0.64}, {0.3, 0.512}, {0.4, 0.4096}, {0.5, 0.32768},
{0.6, 0.262144}, {0.7, 0.209715}, {0.8, 0.167772}, {0.9, 0.134218},
{1., 0.107374}, {1.1, 0.0858993}, {1.2, 0.0687195}, {1.3, 0.0549756},
{1.4, 0.0439805}, {1.5, 0.0351844}, {1.6, 0.0281475}, {1.7, 0.022518},
{1.8, 0.0180144}, {1.9, 0.0144115}, {2., 0.0115292}, {2.1, 0.00922337},
{2.2, 0.0073787}, {2.3, 0.00590296}, {2.4, 0.00472237}, {2.5, 0.00377789},
{2.6, 0.00302231}, {2.7, 0.00241785}, {2.8, 0.00193428}, {2.9, 0.00154743},
{3., 0.00123794}, {3.1, 0.000990352}, {3.2, 0.000792282}, {3.3, 0.000633825},
{3.4, 0.00050706}, {3.5, 0.000405648}, {3.6, 0.000324519}, {3.7, 0.000259615},
{3.8, 0.000207692}, {3.9, 0.000166153}, {4., 0.000132923}, {4.1, 0.000106338},
{4.2, 0.0000850706}, {4.3, 0.0000680565}, {4.4, 0.0000544452},
{4.5, 0.0000435561}, {4.6, 0.0000348449}, {4.7, 0.0000278759},
{4.8, 0.0000223007}, {4.9, 0.0000178406}, {5., 0.0000142725},
{5.1, 0.000011418}, {5.2, 9.13439×10^{-6}}, {5.3, 7.30751×10^{-6}},
{5.4, 5.84601×10^{-6}}, {5.5, 4.67681×10^{-6}}, {5.6, 3.74144×10^{-6}
{5.7, 2.99316×10^{-6}}, {5.8, 2.39452×10^{-6}}, {5.9, 1.91562 ×

Then we wanted to generate a solution in the structure of set of points, t0, x0, t1 x1, t2 x2 and so on. So in order to do that, we wrote a for loop and this is the condensed version of the for loop that we came down to starting for a from, from a longer version, so what did we do in this for loop?

We initialize data list, the first element of data list is ti, xi, so we created a data list and inserted a first element in ti xi, this was the initialization step and there was only initialization step that was required. Then we went ahead and put the condition that I want data list size of data list to be less than equal to nMax. Because they are nMax number of steps, so we said in fact, it should be nMax number of steps.

So size of data list should be nMax plus 1, so let me add 1 over here, because the size is already 1 and the, there are nMax steps, so total nMax plus 1 the size will become nMax plus 1. So, length of data list is less than equal to nMax plus 1 that is our condition to stop the for loop, as long as this condition is true for loop will continue to execute the body.

The next statement is the increment, but increment is always done after the body is executed. So, in the body, we only wrote down one statement that find the previous step previous step is the last element of data list, in this case when the for loop being last element of data list is just initial value ti xi and after the body is executed the for loop we are going to increment.

Then in the increment step, we increment to data list, the next step and the next step is evaluated as previous plus the time component is h in the time step we add an h in x we step h times f acting on previous that is a derivative f is the derivative f acting on previous to quickly remind you.

Remember the previous is a set of points, for something for times let us say 0.1 and something for x let us say 2 and we are doing f at at, so f is send the 2 arguments in that as the first element of the list and the second element of the list, the first element becomes time and the second element becomes the x and f is past these two arguments and f evaluates to the value remember f was minus 2x so $-2*2=-4$.

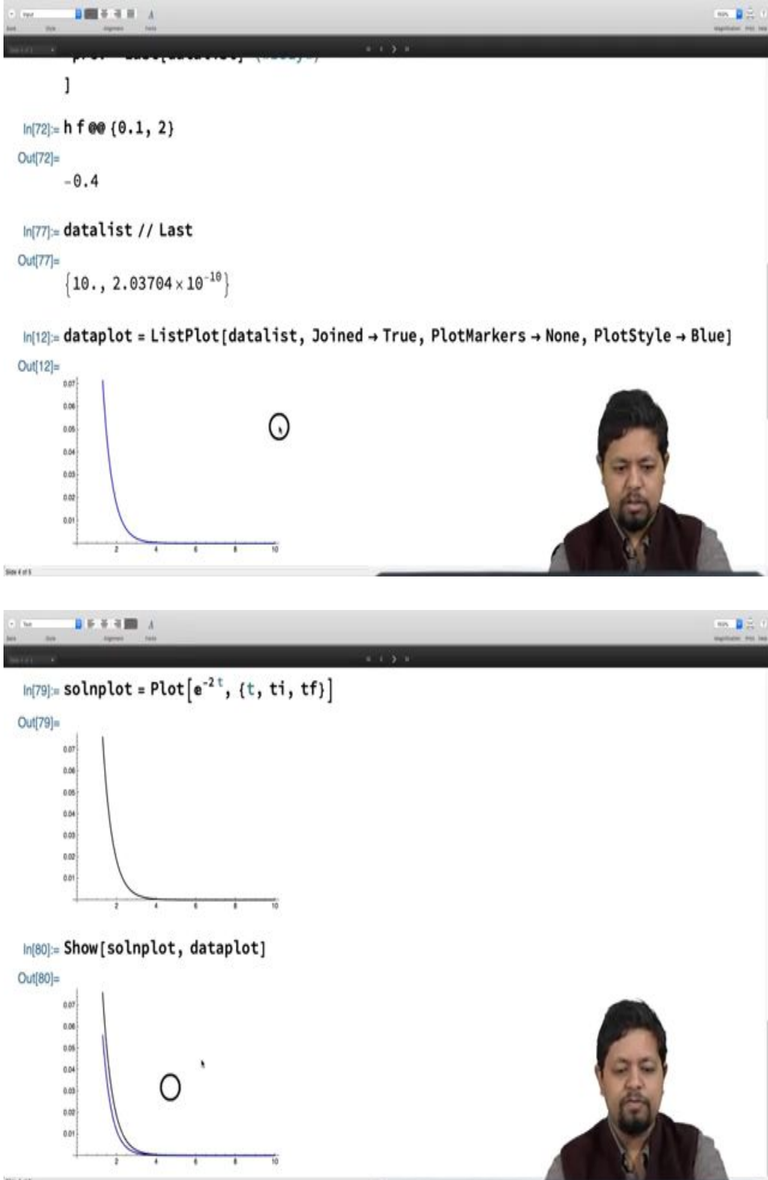
So that is what this does multiplying it with h, h in this case is multiplying this with h gives me the step size and x, the increment that is to be made in x. And adding that to previous, adding h, h * f at a previous to h increments both the time and the x simultaneously. So, we have done quite a bit in this increment statement, we have calculated the next step, added it to the previous and upended that entire thing to the data list.

So, now this becomes the last element of data list, next time when the body is executed previously becomes the last, so the the the next step of this time become the previous for the next time. And in this fashion for loop continues till nMax plus 1 steps and we are going to execute the for loop now this is done. And if you want you can print the data list, this is my data list, let us go ahead and check the last element of data list, so last element of data list is 10.1.

(Refer Slide Time: 05:29)

So I must have made a logical mistake because I wanted to calculate to final time tf equal to 10. So I think this has to be nMax, let us go ahead and do this again and check last element of data is that is correct it is now turning out to be 10, so you think about this counting, I will move forward and we will go ahead and make a plot.

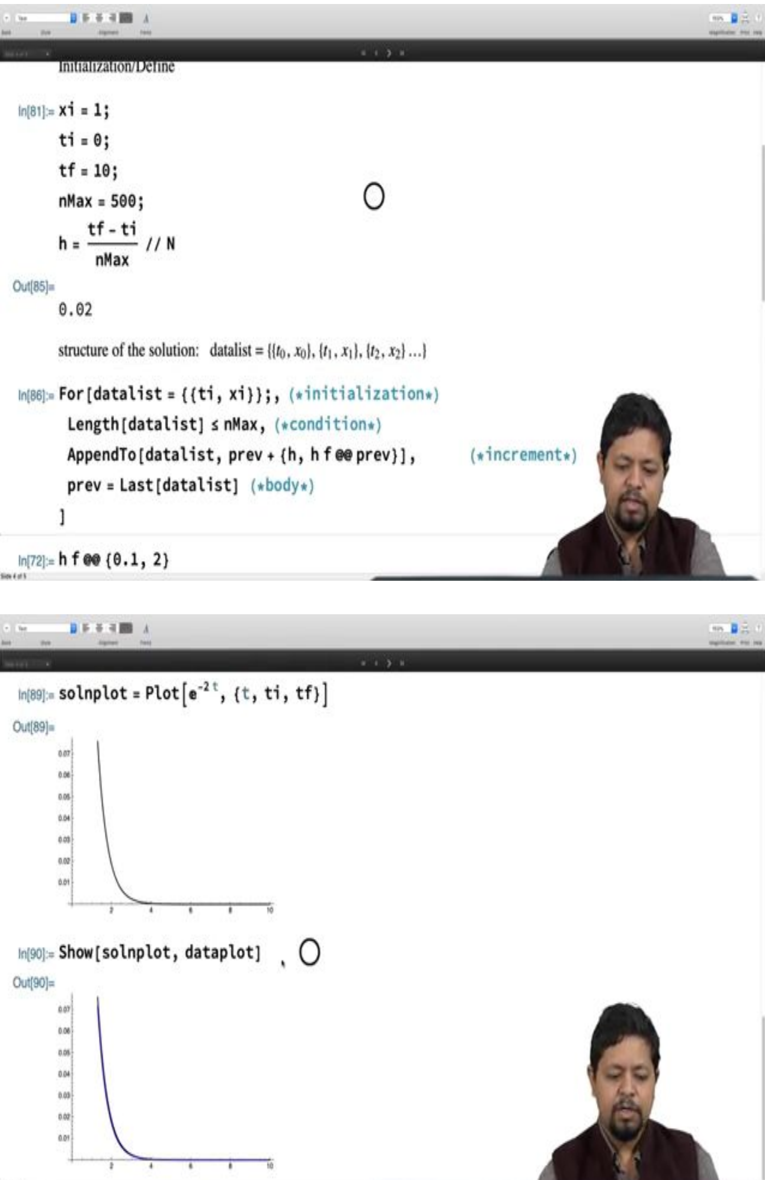
(Refer Slide Time: 05:47)



So, I make a data plot for this I get this plot I make the solution plot for the function the expected solution e^{-2t} and I get this result. I can show them on top of each other and you see there is some difference, so we are getting this shape of the solution but not quite right the solution, the reason for that is the step size is too large at this point.

The black is the solution expected solution, other differential equation and blue is the numerical solution that we have obtained.

(Refer Slide Time: 06:26)



So let us go ahead and make the h smaller and we have done this last time remember when we made h for the smaller down to the let us say nMax = 500 makes h = 0.02 and then we executed for loop and then the statement I do not require so delete it.

And when I did this you saw that there was a significant improvement in the solution. But you see still that there is a minor difference.

(Refer Slide Time: 06:58)

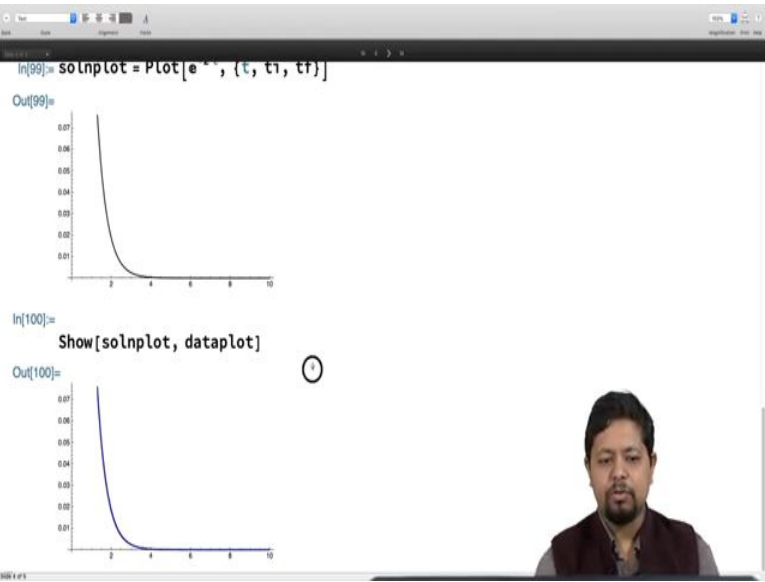
```
ti = 0;
tf = 10;
nMax = 1000;
h = (tf - ti)/nMax // N
Out[95]=
0.01
```

structure of the solution: datalist = $\{\{t_0, x_0\}, \{t_1, x_1\}, \{t_2, x_2\} \ldots\}$

```
In[96]:= For[datalist = {{ti, xi}};, (*initialization*)
  Length[datalist] ≤ nMax, (*condition*)
  AppendTo[datalist, prev + {h, h f @@ prev}],        (*increment*)
  prev = Last[datalist] (*body*)
 ]

In[87]:= datalist // Last
Out[87]=
{10., 1.36652 × 10^-9}
```

structure of the solution: datalist = $\{\{t_0, x_0\}, \{t_1, x_1\}, \{t_2, x_2\} \ldots\}$

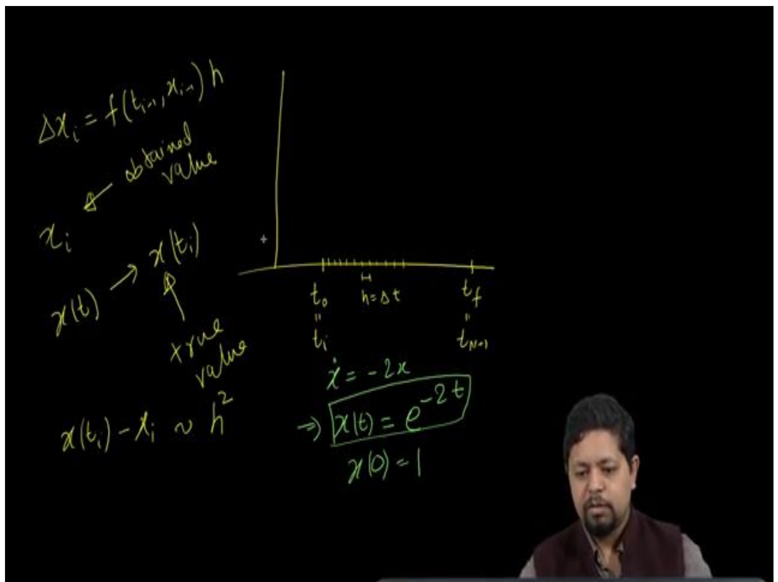
```
In[96]:= For[datalist = {{ti, xi}};, (*initialization*)
  Length[datalist] ≤ nMax, (*condition*)
  AppendTo[datalist, prev + {h, h f @@ prev}],        (*increment*)
  prev = Last[datalist] (*body*)
 ]

In[97]:= datalist // Last
Out[97]=
{10., 1.68297 × 10^-9}

In[88]:= dataplot = ListPlot[datalist, Joined → True, PlotMarkers → None, P...le → Blue]
Out[88]=
```



And in order to improve that you may probably want to go to higher number of nMax maybe 1000 and we can try this again to even get higher accuracy. You see that there was some small change in this number that happened is the last value of the data list, so let us go ahead and do this again, there was some very small change, now let us go ahead and do this now again.

Now you see there is a perfect agreement. So, in order to get this perfect agreement, we had to go down to a very small value of h, h = 0.01, let us go ahead now and understand what is the impact of h on the quality of the solution. So, without proof I would like to tell you a little bit about you know importance of h when you are doing Eulers method.

(Refer Slide Time: 07:49)



Remember that we wanted to find the solution from time from t naught to some time tf or this was ti and this was tn plus 1 and we divided this time axis into small, small, small, small tiny bits, each of this was Δt or step size h and we use Eulers method to do the integration that means Δxi was calculated as f evaluated at ti - 1 xi - 1 * Δt or h I should write h, because that is what we have been calling so this is h.

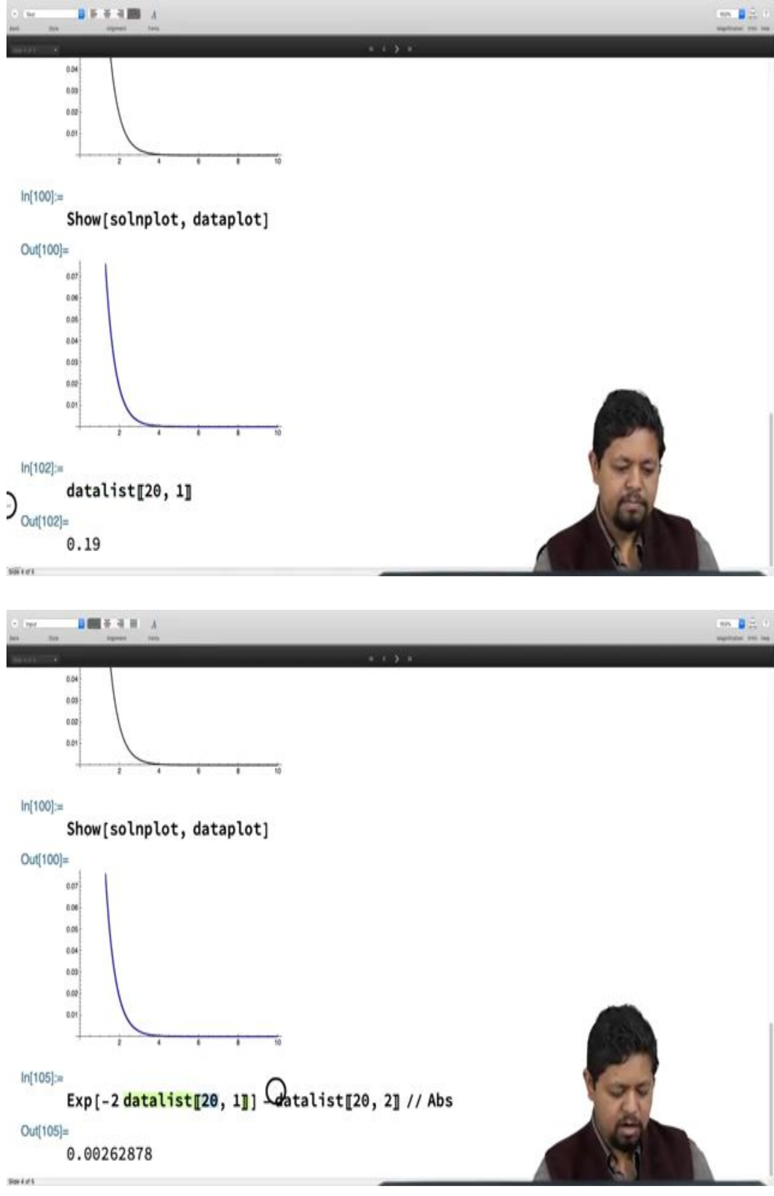
Now, the question is what is the size of the error? What is the size of the error that is so we are getting xi's by doing this process by building up steps, we are getting an xi and we have got a solution from the integration that is we have obtained solution x as a function of t, so from this we can obtain what is x(ti).

This is the true value of x from the solution and this is the numerically obtain value of xi. Since we are making increments of step size h, So, our accuracy will be h, therefore the error will be order h^2 . So, I expect that x(ti) - xi this should be order h^2 . So, let us go ahead and validate this numerically.

Now, the problem that is at hand is $\dot{x}=-2x$ the solution of this is $x(t)=e^{-2t}$ ofcourse we have used initial condition x(0) = 1. So, therefore I have obtained I know the solution, so I will

substitute ti in this and that will be my true value of the of the solution and the expected value something that I have obtained numerically. So, that is what I want to do.

(Refer Slide Time: 10:40)

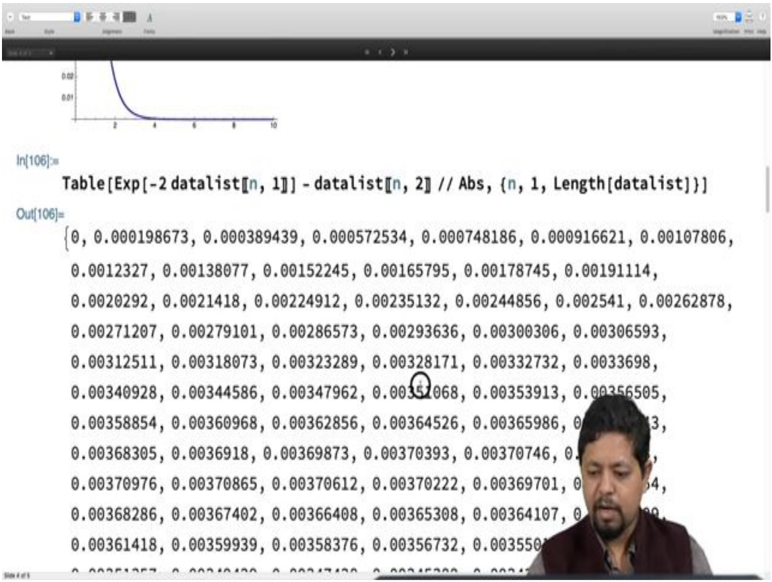


So, let me go ahead and try to do that really quickly and what I will do is, I will add over here, I will take data list, I will take a certain point in data list, let us say the the the 20th element in data list is has t equal to 0.19 and the x value is 0.68. So, now I will take the first element of this 20th element that is 0.19 and I will calculate my true value by multiplying minus 2 to this that is e to power minus 2t.

And that is 0.68 to this I will subtract the obtained value which is the data list, same thing 20 and the second element of this. So, let us just first compare them, so in order to compare them, I am going to put them in a list. So, the first item is the true value and the second item in this list is the calculated value or numerical value. So, we see this is a pretty close but there is some difference between the two and I want to understand what is the difference.

So, therefore in order to understand that difference, I will take a difference of these two numbers and I will calculate its absolute value, so I will do a postfix Abs, Abs is for absolute value, it is going to put a mod on that and I get a number 0.002. Now, you see that this 0.002 is for 20th point, so what I am going to do is now I want to make this arbitrary.

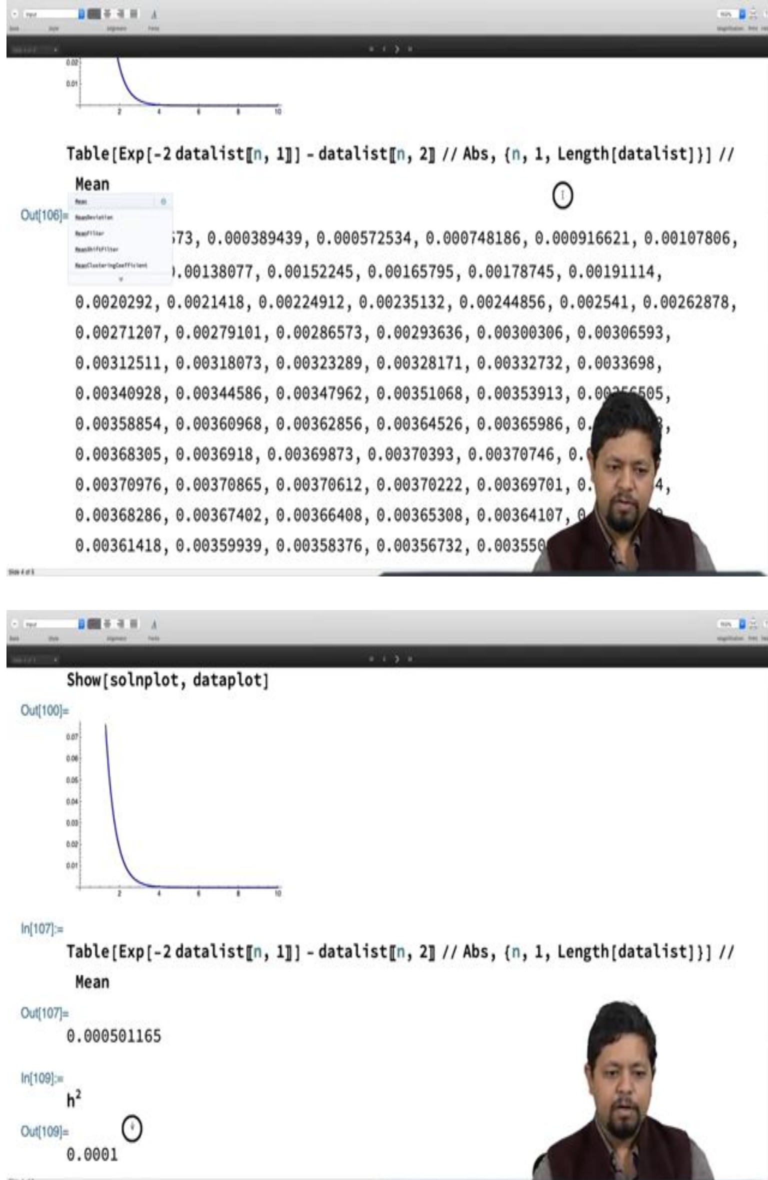
(Refer Slide Time: 12:32)



So let me call this as n and call this n and I will put this inside the table and I will run n from 1 to length of a data list. So, length of data list and they will give me all the values of this difference, the absolute values.

And here is my entire list and what I really want to do now at this point is, you see all these numbers are comparable see the this transferred from point some really small values but slowly builds up to 0.001 then 0.002 and then eventually it remains of that order, so I want to get a fair estimate of you know of the of the size of the error, so what I will do is I will calculate the mean of this whole thing.

(Refer Slide Time: 13:23)



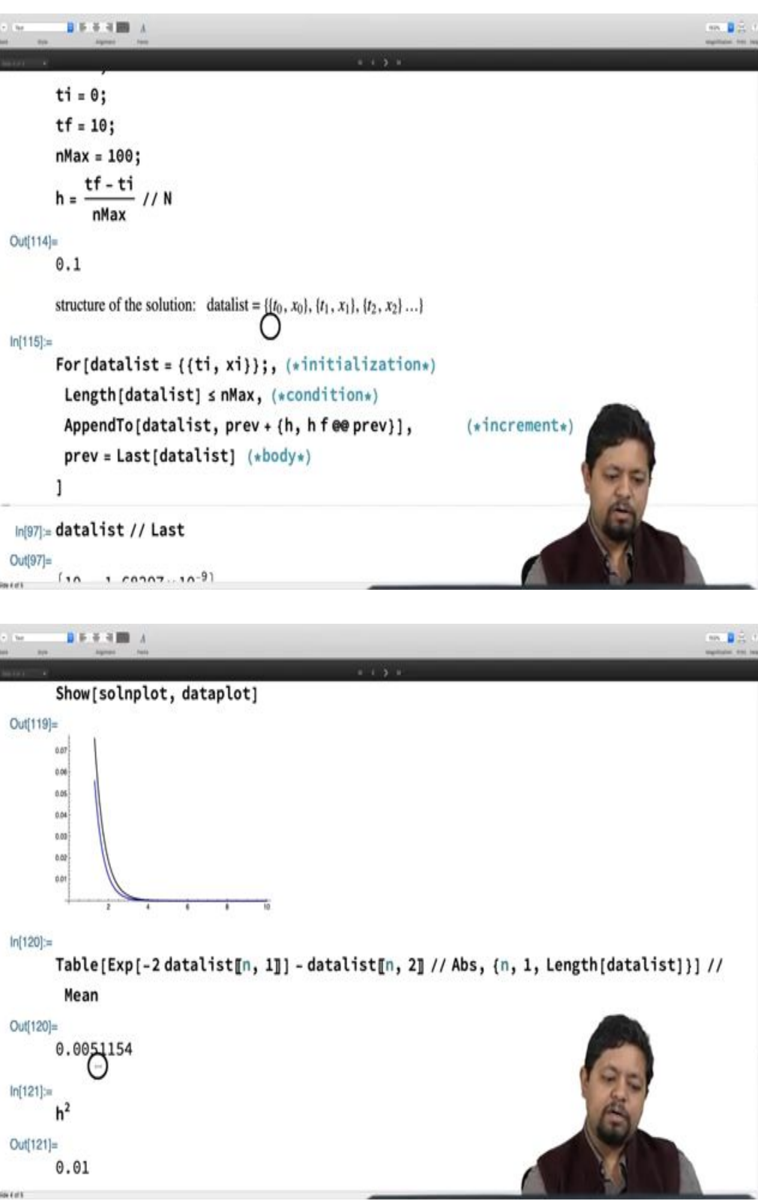
So this data list, that I have obtained, this data I want to calculate the mean of that, an average. So, I will apply postfix mean function on that, mean function will give me mean of the all of these values and there we go.

The mean that we obtain is 0.0005 and let us go ahead and compare this with h. So, the h value, so the h value I have is 0.01, I set the error average is going to be order h square, so and you see that it is of that order. In fact that is 5 times higher is very specific to the problem the the the

point and the point over here is that the views the Eulers method we moved in step sizes of h, Eulers method tells us that errors are of the order of h^2 .

We are not giving a proof of that, we are just giving you that this is the case with the Euler method within error of order h^2 . So, therefore we calculate the average error from all the calculation we had and we found the average error was 0.0005 and when we calculate h^2 we found that number was comparable.

(Refer Slide Time: 15:00)



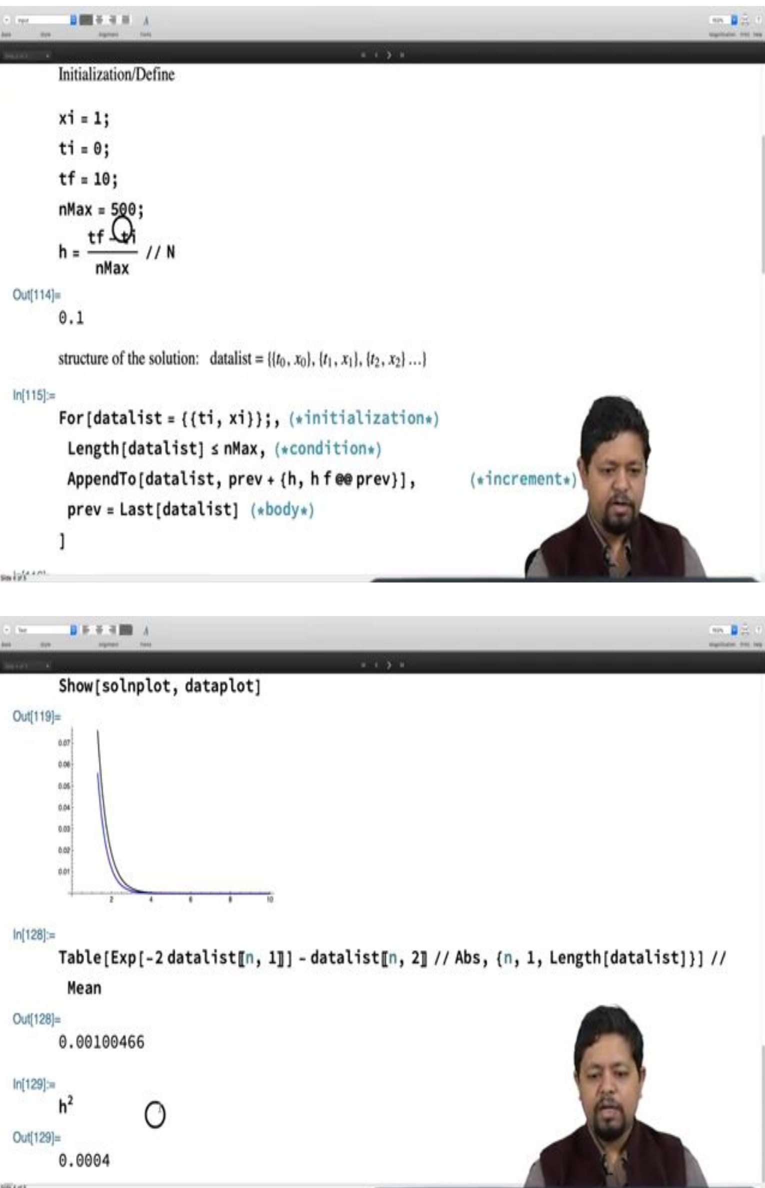
In order to verify this, let us go ahead and re-run this entire thing and in this time we are going to change a nMax and that is going to change h now my h is 0.1, therefore my error should become larger it should become of order 0.01, so let us go ahead and find that out, so I will execute this for loop again.

And I can do the data plot again and you will visually see the difference and you can from here you can estimate the difference visually, if you move in a line vertically up you do see that the error is of the order of 0.01 approximately, if I move along this line up here, the value here is 0.01 on the blue curve and on the black curve it is about 0.02 maybe slightly lesser, but then we are making a ballpark estimate.

So, the error is about the difference between them is about 0.01, but let us go ahead and that is for 1.0 which we want to calculate mean for this entire range. So, let us go ahead and execute this statement again, which should give me the mean and we do see that it turns for of 0.005 and h square is 0.01 so it is about half of that.

So, we found that in this case expected size is 0.01 multiplied by an order one number can happen, it is not going to be exactly h^2 is going to be order 1 number times h^2 in this case that order one number turned out to be 1.5. We can go ahead and and re-evaluate this for different value of h and I will invite you to play around with this.

(Refer Slide Time: 16:26)



So let us go ahead and make it 500 and run this again and we do not need to execute these statements and you can go ahead and execute this one over here and execute h^2 and you see that again we find that the mean error that we found in this is comparable to order h^2. So, that validates the point that when we are using Eulers method, the error is order h^2.

Let us go ahead and improve this you see we have been going back and forth, back and forth again and again to execute all of this, but if you want to do a thorough study, you may want to construct one single function which you can do all of that. So, what we will do now is, we will

turn this for loop into a function that is our Eulers method into a function and for this we will use module construct.

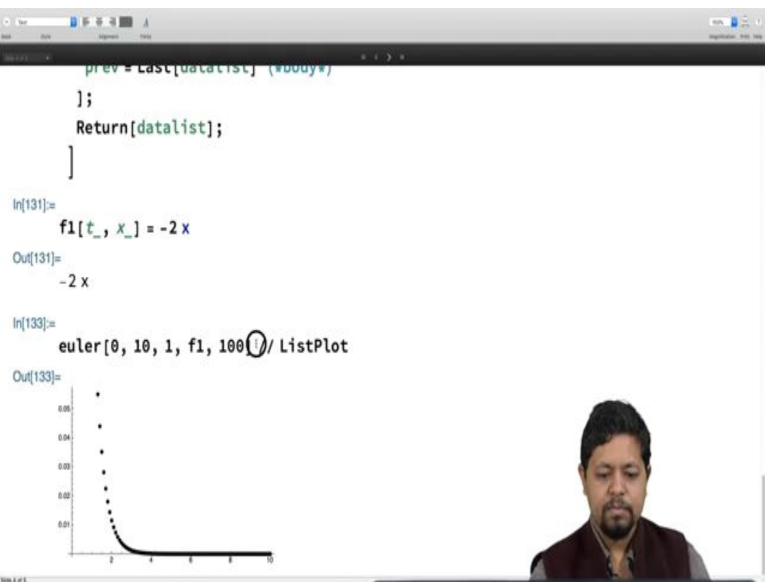
If you have not watched my video about module construct that is the technical prelim 5 please go ahead and watch that, because in this now we are going to use the module function to construct, we are going to turn this into a function, so let us go ahead and do that.

(Refer Slide Time: 17:42)

Euler's Method as a function

```
In[130]:=
euler[ti_, tf_, xi_, func_, nMax_] :=
 Module[{h, datalist, prev},   (*local variables*)
  h = (tf - ti)/nMax // N; (*body*)
  For[datalist = {{ti, xi}};, (*initialization*)
   Length[datalist] ≤ nMax, (*condition*)
   AppendTo[datalist, prev + {h, h func @@ prev}],     (*increment*)
   prev = Last[datalist] (*body*)
  ];
  Return[datalist];
 ]

f1[t_, x_] := -2
```

```
  Return[datalist];
 ]

In[131]:=
f1[t_, x_] = -2 x
Out[131]=
-2 x

In[132]:=
euler[0, 10, 1, f1, 100]
Out[132]=
{{0, 1}, {0.1, 0.8}, {0.2, 0.64}, {0.3, 0.512}, {0.4, 0.4096}, {0.5, 0.32768},
 {0.6, 0.262144}, {0.7, 0.209715}, {0.8, 0.167772}, {0.9, 0.134218},
 {1., 0.107374}, {1.1, 0.0858993}, {1.2, 0.0687195}, {1.3, 0.054
 {1.4, 0.0439805}, {1.5, 0.0351844}, {1.6, 0.0281475}, {1.7, 0.
 {1.8, 0.0180144}, {1.9, 0.0144115}, {2., 0.0115292}, {2.1, 0.0
 {2.2, 0.0073787}, {2.3, 0.00590296}, {2.4, 0.00472237}, {2.5, 0 789},
 {2.6, 0.00302231}, {2.7, 0.00241785}, {2.8, 0.00193428}, {2
 {3., 0.00123794}, {3.1, 0.000990352}, {3.2, 0.000792282
```



So, I am going to add a title over here. So, Eulers method as a function, I am going to call that function as Euler the name of my function will be euler. All the custom functions or custom variables that I defined I always start the small letter such as small e over here, because block letters are used for defining built-in mathematica functions, I do not want to run into any conflict, so I always define my own functions with small letters. And that is the policy I will use over here.

So, I want to define a Euler function and to this Euler function, I want to take some inputs such and my inputs in this case are going to be the initial time ti, the final time tf, initial value xi and I want to give the function that is the derivative function that is $\dot{x} = f(x)$, so this is my input, let us see what else do we required.

So, here is what our inputs were for the for loop xi, ti, tf, nMax, so, nMax we want also as a input which tells me how many steps I want in order to do my evaluation, you can use nMax as a input or you can use h as an input, but I usually like nMax as a input because I fix my final time, I do not want to you change my final time by defining nMax and h, I fixed my final time, I decide nMax and from that I calculate h rather than using h to calculate nMax.

As you can also take another approach if you want and I want to do a SetDelayed over here so colon equal to and I want to define my own functions on going to use the module construct. And

this module construct I am going to have two parts, one is the local variables and others is the body, so I will write the local variables and different statements of body in different lines.

So, these are my local variables, so let me for illustration let me put that comment over here, this is local variables which are only visible inside module function and in the body this is going to be my body there we go and the first thing I want to do in the body is I want to calculate h, so I want to define h as a local variable.

Remember that there was one h already defined which is a global variable, now now I want to do to define h which is a local variable. So, I will put an h inside the first argument and that becomes a local variable you see there is a colour change and now this h is going to be $(tf-ti)/nMax$ and we will do a //N, so that it is evaluated to approximate numbers so that the computation is fast.

And there is my body and then I will go ahead and write other statements of the body and this point I want my for loop, so I will just go ahead and copy this for loop and paste over there, now you see that this for loop also has some variables inside it. And I will have to declare those variables as local.

So, that they are not affected by the global values, I will go ahead and call data list also as a local variable and you see the moment I do that there is a colour change over here and then there is a previous and f and clearly this f has to be the local variable function that I am passing and I wanted to define one more local variable that is previous.

And now I think this is done, I will end this for loop with a semicolon because I do not want any output from that, I just want to run that for loop and then I am going to return data list, the local variable data list that has been part of this module I want to calculate that data list, append information to that data list, I want to return the data list.

And this way I have constructed a function Euler which takes the input ti, tf, xi the function that I want to pass on, nMax. So, now I can go ahead and execute this and you see when I execute this nothing happens because this is just the definition made, a SetDelayed definition that I am going to execute this when I call the Euler function with certain inputs.

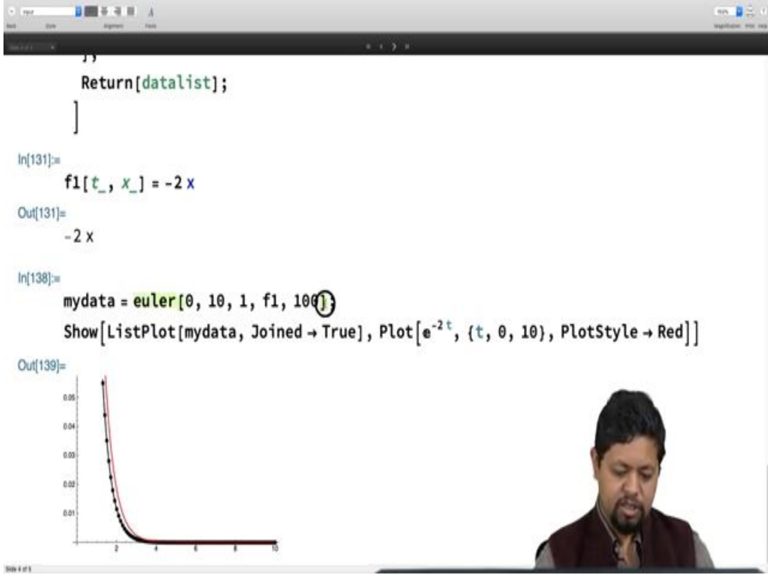
So, until unless I call Euler function nothing is going to happen and then I am going to call Euler function I am going to call it with certain ti, tf, xi function nMax. Alright so, let us go ahead and do that.

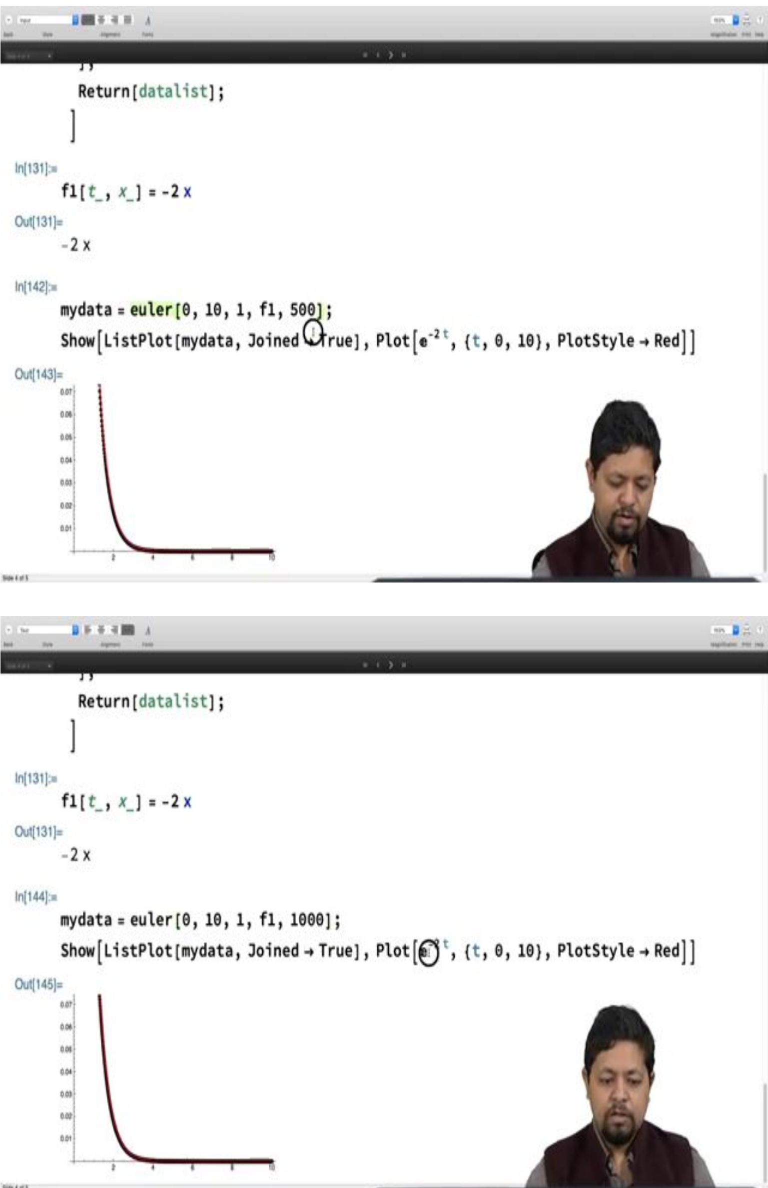
So, for this purpose I will need to define a function so let me call this ff this time, let me call this $f1(t,x)=-2x$ this does not really matter I can make a SetDelayed definition or a set definition let me make a set definition, there we go.

Now I want to call my Euler function, so I want to call my Euler function with the inputs so my ti was 0, tf was 10, xi was 1, that is at t equal to 0, x was 1 to the function I just want to pass the name of the function f1, note that over here I do not need to pass the argument because arguments are being applied over here.

Here I just need to pass function of the symbol, the function of the symbol is the argument for this, so I just need to pass f1 and this f1 needs to be defined, that we have already done, and then I want to decide nMax equal to a 100 points let us say and I want to execute this the output of this should be database and that is what I got. I can go ahead and do a, you know list plot on that. And when I apply a list plot to that is what I get it.

(Refer Slide Time: 24:30)





Let us go ahead and do this, I am going to call this my data equal to Euler, so that is one statement, in the next statement I will do a list plot this list plot I will say my data and then joined is true, let us go ahead and execute that, this is what I get one plot.

And I want to compare this now with with the solution, so the solution was I want to enclose all of this inside the show function, the solution was e^{-2t} and t goes from 0 to 10, there we go I can change the colours make this red so that we can differentiate, so red is my expected solution and this is my calculated solution.

And I want to improve this so I can just go ahead and change nMax and now you see this has been defined as a function so I can go ahead and simply call the function improve it, I can further improve it by changing nMax, there we go and if you want to further improve it, maybe I can go 1000, there we go. So, this Euler function all we have defined as a function and the single call I am able to do all these 5 or 6 different steps that we were doing before.

Let us go ahead and do one more thing, we are going to define an error function, error function to calculate the error that we calculate over here, this was the error that I was calculating, mean absolute error, so in order to do that, I am going to define a function that is an error function.

(Refer Slide Time: 26:35)

```
In[150]:=
error[data_, solfunc_] := Module[{tlist, xlist, xtrue},
  tlist = data[[;; , 1]];
  xlist = data[[;; , 2]];
  xtrue = solfunc /@ tlist;
  Return[xtrue];
 ]

soln[t_] = e^-2 t;
error[mydata, soln]
Out[149]=
{1, 0.98, 0.9604, 0.941192, 0.922368, 0.903921, 0.885842, 0.8681 0763,
 0.833748, 0.817073, 0.800731, 0.784717, 0.769022, 0.753642, 0.
 0.723798, 0.709322, 0.695135, 0.681233, 0.667608, 0.654256, 0.6
 0.628347, 0.61578, 0.603465, 0.591395, 0.579568, 0.567976, 0
 0.545484, 0.534575, 0.523883, 0.513405, 0.503137, 0.493
```

```
In[151]:=
soln[t_] = e^-2 t;
error[mydata, soln]
Out[152]=
{1, 0.980199, 0.960789, 0.941765, 0.923116, 0.904837, 0.88692, 0.869358,
 0.852144, 0.83527, 0.818731, 0.802519, 0.786628, 0.771052, 0.755784,
 0.740818, 0.726149, 0.71177, 0.697676, 0.683861, 0.67032, 0.657047, 0.644036,
 0.631284, 0.618783, 0.606531, 0.594521, 0.582748, 0.571209, 0.559898,
 0.548812, 0.537944, 0.527292, 0.516851, 0.506617, 0.496585, 0.486752,
 0.477114, 0.467666, 0.458406, 0.449329, 0.440432, 0.431711, 0.423162,
 0.414783, 0.40657, 0.398519, 0.390628, 0.382893, 0.375311, 0.367879,
 0.360595, 0.353455, 0.346456, 0.339596, 0.332871, 0.32628, 0.3
 0.313486, 0.307279, 0.301194, 0.29523, 0.289384, 0.283654, 0.
 0.272532, 0.267135, 0.261846, 0.256661, 0.251579, 0.246597, 0
 0.236928, 0.232236, 0.227638, 0.22313, 0.218712, 0.214381, 0.2
 0.205975, 0.201897, 0.197899, 0.19398, 0.190139, 0.186374
 0.179066, 0.17552, 0.172045, 0.168638, 0.165299, 0.162
```

So, I will define another function I will call this function as error and it is going to have 2 arguments, the one argument will be the data that I am going to pass through it, the data will be in the same structure as my data list and I am going to pass it another function which is the solution function.

So, I will call it as solution function which is the function of t, so then I can find the difference and compare, I am going to write this as a module again, so what I want to do here is again define a set of local variables and then the body in the next line.

(Refer Slide Time: 31:23)

So I want to define over here, what I want to do is I want to calculate only define variables, tlist and xlist, so for tlist, I will extract from my data the first component of it, so I will take all the rows, so double semicolon means all the rows of it and the first element or in each row, that is that is all the times.

And I will get the x list and that will be in order to obain the error element, I press escape, press 2 square brackets, press escape again that becomes a double square bracket and then I do the same thing to close it and that becomes that that can be used to determine the array elements, I

think we have done this before but this is just to remind you again we will use the double semicolon to get all the rows and the second element for this.

Let us, see if this is what is going to give me expected results, so let me just go ahead and return, return t list, I am not done with the function yet, I am just building the functions so I want to test it out step by step.

So, it is going to return me tlist, let me go ahead and print the tlist now to see this is what tlist was supposed to be, I want to call error function with the data mydata and some solution function which have been not defined, so I will just go ahead and pass f1 to it, does not really matter.

Alright, so this is it has taken mydata and extracted the time part of it and I can go ahead and check for xlist also, so if I make it xlist just take in the data and extracted the xlist, the x component of the data, alright that is what was intended.

Now I what I want to do is I want to calculate another quantity called xtrue and this is going to be my solution function, solution function which is a function of only t, I want to apply to each value, so I am going to use the map operator / add of t list. So, I am going to apply solution function at each element of tlist will give me a list of true values of x that is expected.

I want to define xtrue as a local variable, go ahead, let us go ahead and check that I am getting some expected result, so let us go ahead and return xtrue from here and let me in order to pass this time I cannot pass f1. Because f1 is the derivative function not the solution function, it has 2 arguments, now I am actually using solution function and it is and the argument solution function is a single element each element of t list.

So I should define that, so my solution function which is just a function of t only is e^{-2t} , so that is my solution function and I want to pass this is an argument to error function, let us go ahead and do that and there we go. So, that gives me exponential e^{-2t} applied on the times.

Alright so what now I want to do is xlist is the calculated values, xtrue is the true values of x and what I really want is take a difference of the two, so when I take a difference of two arrays I am going to get difference of array of difference of elements, so I can go and directly to that I can go

ahead and directly do xtrue - xlist and to this I can apply postfix absolute value and postfix mean and that will give me there the mean value.

Now, I can do this very quickly with experience, but if you are not entirely sure you can go ahead and do this step by step, make the checks in between that are necessary for you to do, just like I printed t list or return tlist and return xlist return xtrue, to verify that I was getting some expected results.

You can make stronger verifications by evaluating some of the numbers on a calculator or separately over here and comparing them. But if you experienced you can actually go ahead and directly to this thing and let us go ahead and do that. So, this is my definition, so now I am going to get now my error function going to give me mean absolute error returned and let us go ahead and check that.

So, I am going to get a single number and just to you know verify that from our older code that when I had 1000 points this is the error that I got, so let us go back to our old code and make this as 1000 code just to verify that our error function is actually giving what we did before. So, we can go ahead and calculate the error from the statement over here, there we go those points 0.005 and over here is again 0.005 very good.

So, now what we have done? We have got two functions one is the error function and the other one is the Euler function, we will define two functions over here and we will keep on using these functions again and again for testing the calculations we are making. So, let us go ahead and go ahead and combine this, so I will just copy this and put this side over here.

(Refer Slide Time: 33:48)

```
]

f1[t_, x_] = -2 x;
soln[t_] = Exp[-2 t];
mydata = euler[0, 10, 1, f1, 1000];
error = [mydata, soln]
Show[ListPlot[mydata, Joined → True], Plot[soln[t], {t, 0, 10}, PlotStyle → Red]]
```

Out[145]=

In[154]:=

```
soln[t_] = e^(-2 t);
error[mydata, soln]
```

```
  tlist = data[[;; , 1]];
  xlist = data[[;; , 2]];
  xtrue = solfunc /@ tlist;
  Return[xtrue - xlist // Abs // Mean];
 ]

f1[t_, x_] = -2 x;
soln[t_] = Exp[-2 t];
mydata = euler[0, 10, 1, f1, 1000];
error[mydata, soln]
Show[ListPlot[mydata, Joined → True], Plot[soln[t], {t, 0, 10}, PlotStyle → Red]]
```

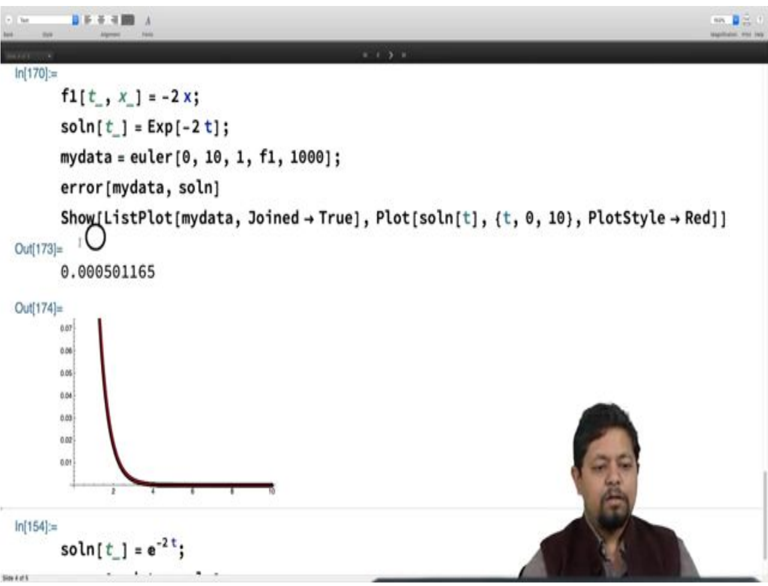
Syntax: "error =" cannot be followed by "[mydata, soln]".

In[154]:=

```
soln[t_] = e^(-2 t);
error[mydata, soln]
```

Out[155]=

```
0.000501165
```



Now I can go ahead and combine the statement also over here. So, I will define f1(t,x) that is my derivative function as -2x the solution of this equation is a solution function which is a function of time only and I am going to define that as e^{-2t} now I will call Euler function to calculate the solution and then I am going to call the error function to go ahead and calculate the errors so for this I will pass it arguments my data and solution that is my error let me not put a semicolon because I want to print it.

And then over here I am going to do a show of listplot of my data and this is the solution function, so let me replace this e^{-2t} by solution function of t and this is my ti, ti from 0 to tf. And you go and execute all these 5 statements in a single go, seems like this, the error is the function, so it has should have argument of the typo there and I want to go and execute now and there we go. So, we have got the error over here and the comparison of the two plots.

(Refer Slide Time: 35:47)

In[175]:=

```
f1[t_, x_] = -2 x;
soln[t_] = Exp[-2 t];
mydata = euler[0, 10, 1, f1, 100];
error[mydata, soln]
Show[ListPlot[mydata, Joined → True], Plot[soln[t], {t, 0, 10}, PlotStyle → Red]]
```

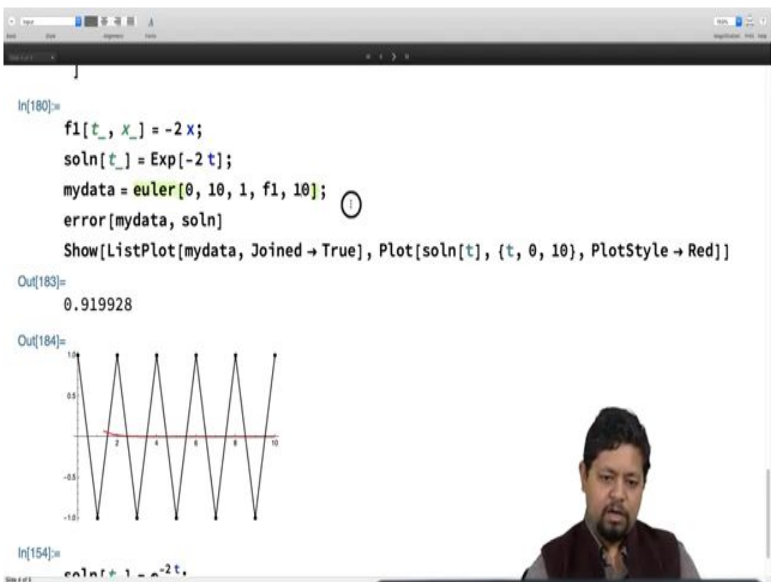
Out[178]=

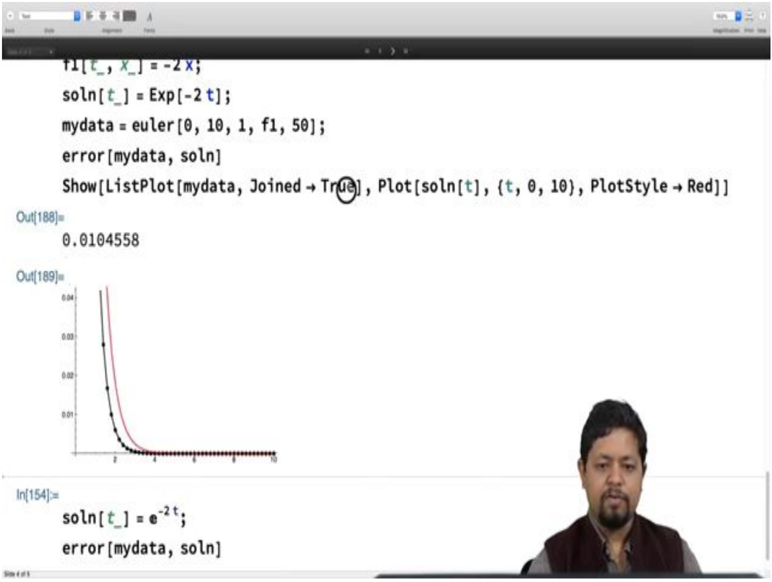
0.0051154

Out[179]=

In[154]:=

```
soln[t_] = e^-2 t;
error[mydata, soln]
```





And now you can go ahead and play around with this, you can make 100, the error changes and you see the comparison over here also changes I can make it even worse make it 10 and you see a really get a bad result, my error is order 1 and I am getting a really oscillatory kind of solution which is completely unexpected for this.

Ofcourse that would happen if we make nMAx very small and h very large. So, in order to get some reasonable value we should make it larger 50 you see that the error is 0.01 and the difference in the curve is visible over here. Now, we can go ahead and play this around with for some more functions, some more other known functions.

(Refer Slide Time: 36:32)

In[190]:=

```
f1[t_, x_] = Cos[t];
soln[t_] = Sin[t];
mydata = euler[0, 10, 0, f1, 50];
error[mydata, soln]
Show[ListPlot[mydata, Joined → True], Plot[soln[t], {t, 0, 10}, PlotStyle → Red]]
```

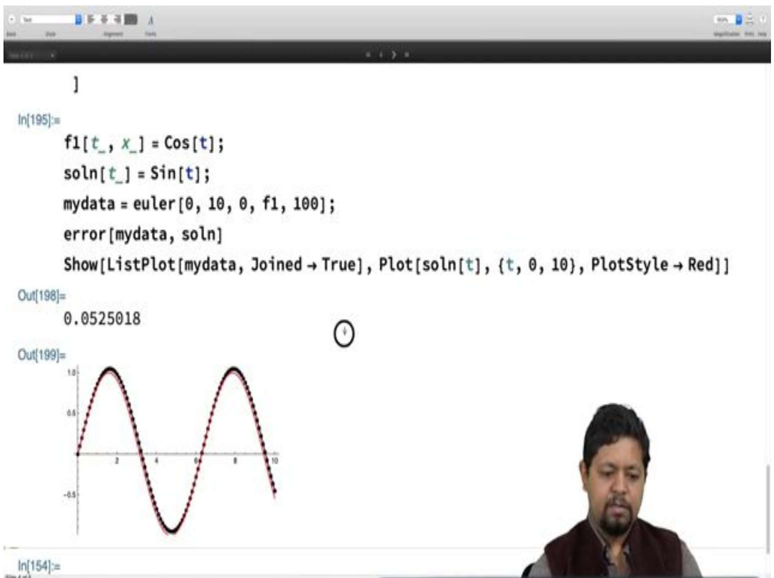
Out[193]=

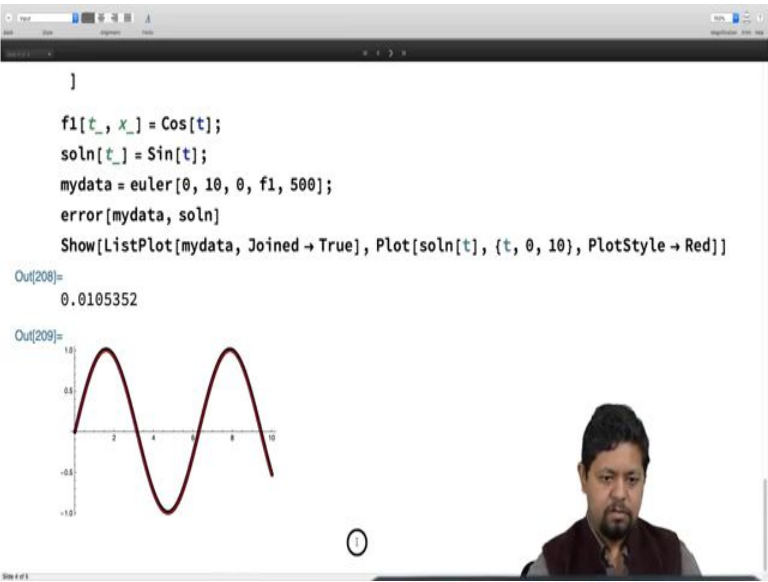
```
0.104576
```

Out[194]=

In[154]:=

```
soln[t_] = e^(-2 t);
```





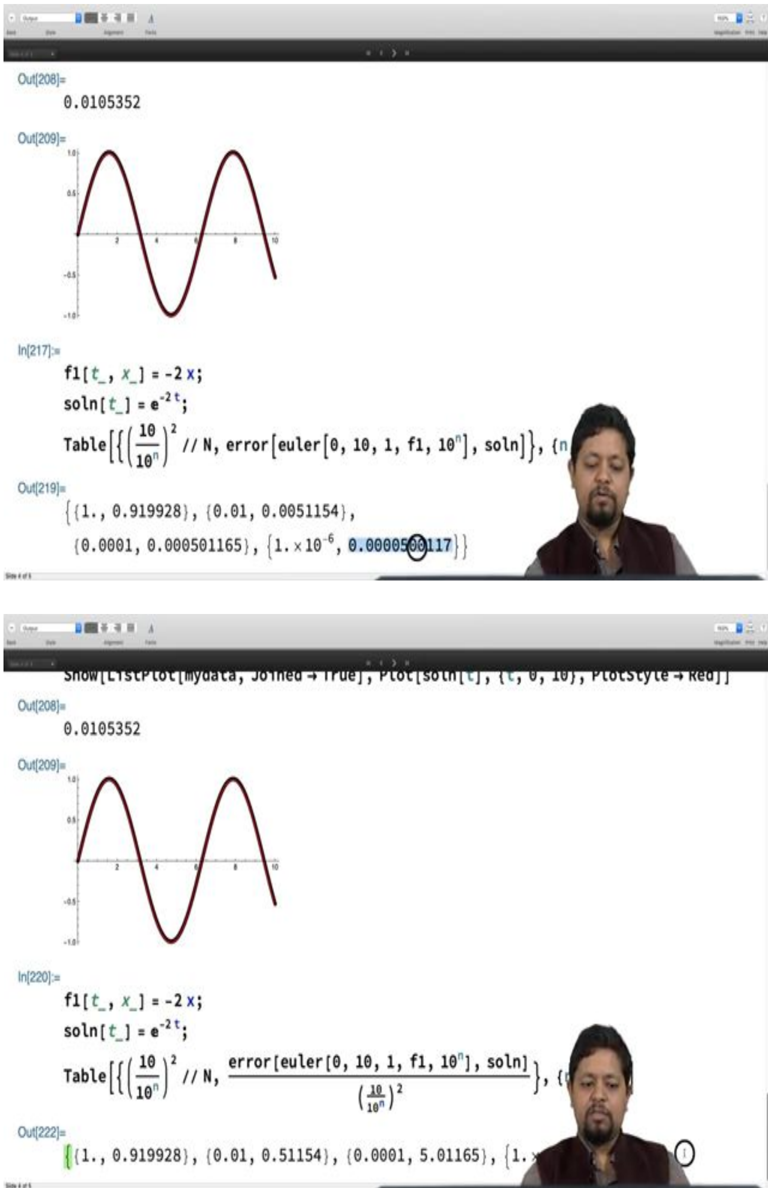
Let us try f of t x as let us say this Cos(t), you know the solution of that, solution of that is Sin(t), so if x0 is Cos(t) the solution x(t) Sin(t) given that the initial condition is at t = 0, x = 0. So, that is my initial condition at t = 0, x = 0 and I am calculating to final time tf, let us go ahead and do this with 50 points and I want to let us go ahead and execute that There we go.

We get an error of 0.1, mean absolute error of 0.1 and the red curve is the true solution, black dots, join together is the calculated solution and you say that they are they are pretty close, but they do not agree with each other very well, ofcourse we can improve it by increasing the number of points or the number of steps in the calculation and that makes that makes things look a little bit better.

I can improve it further again there we go it becomes even better and improve it further increasing more points to get the perfect agreement. So, this is close to perfect agreement, now you see that the the the error has become 0.01. And we should probably print also the h value or rather we can do as we can print error divided by h square.

Alright, let us we will we will do that later in a bit, so there we go, we have got, now what we will do is we are going to go ahead and compare this error I do not need this we will go ahead and create a list of errors.

(Refer Slide Time: 38:48)



So, let us go ahead and go back to to so we will what I want to do is I want to compare error for different values of h, so my h is (tf – ti)/nMax, in this case ti = 0, tf = 10, nMax = 100, so if nMax is 100 so tf over nMax is h and to this I want to compare, to this I want to compare the error.

So, error of my data but my data is obtained by Euler function, so let me go and call this Euler function. The second argument is the solution function, let me go ahead and put the solution

function as a second argument and this nMax is 100 let us go ahead and calculate that and we get 1/10.05.

So you see this is comparable, actually I want to compare this with the h square, so this is h I want to compare with h^2 , so those two numbers are the same order they are comparable. And now I want to go ahead and change nMax to 200 over here to see if this comparison works, okay we get that, now let us go ahead and make this arbitrary.

So, let us go ahead and go back to our older function $f1(t,x)=-2x$ because we have straight values for that a while ago so let us do that again and solution function for this was function of t only and that was e^{-2t} that is my solution function and initial value of x was 1, so let us go ahead and do that and now I want to make this automated.

So I want to define this as 10^n and I want to very n, so wrap this inside the table and I will go ahead and say I want to change n from 1 to 4. So, 10 to power 1, so nMax is 10 to power 1 then next time 10 to power 2, then next 10 to power 3 and 10 to power 4. So, let us go ahead and do that we are going to get 4 sets of numbers there we go.

So, the first time we get h^2 to be order 1 and we get error also to be close to 1, next time we get h^2 to be order 0.01 and we get that error of the same order, next time we get error h^2 to be order 0.0001, 10^{-4} and that is the same size as the error and next time we will get a h as 10^{-6} and again error is order of 10^{-5} . So, you see that the error does gets squared.

In fact if you want you can divide this by h^2 that is I want to divide this by this to see that we get an order one number. Now, what I am doing is I am dividing the error that I am obtaining by h^2 and when I execute that you see that I am getting about an order one number, but for very small h's you see that the error is even larger it is 50 times larger. So, at some point the error is not h^2 to become significantly larger.

Alright so this was understanding the Eulers method, writing the Eulers method as a function and understanding the mean absolute error that we get from Eulers function. So, go ahead and try out

a few more examples of this and you will see that will get somewhat similar results, you will find that errors that you obtain will be typically of order h^2 , sometimes slightly more, sometimes slightly less depending on the function of the case you are working. But you see the precision of Euler's method is only order h^2 .

Now, that means that if I want to get an accuracy of order h^2 suppose I want to get an accuracy of 10^{-4} that means h^2 is order 10^{-4} that means h has to be 10^{-2} . So, in order to make h equal 10^{-2} I have to appropriately choose nMax that is (tf - ti)/nMax has to be 10^{-2} , so therefore I have to appropriately choose nMax.

Now, this makes the Eulers method not very useful because if I want to the accuracy of 10^{-6} or even 10^{-8} , I need to make h smaller and smaller that means number of steps going to become larger and larger. So, this is the limitation of the Eulers method, because the order is order of h^2 I need to make h smaller and smaller to get higher accuracy.

So, therefore next time what we will do is we will look at Improved Eulers method and Fourth Order Runge-Kutta method. So, that we can get higher accuracy with less number of step size. So, next time we will review these methods and we will see how they improve upon the Eulers method that we have worked out so far.

In the meantime go ahead and try out some examples with this method write your Eulers function, try different cases where you know the analytical solution and see how playing around with h, nMax ti and tf all other parameters, how does the solution compared with the exact known solution, we will see you next time.