

Physics through Computational Thinking
Dr. Auditya Sharma & Dr. Ambar Jain
Departments of Physics
Indian Institute of Science Education and Research, Bhopal
Lecture 25
Introduction to Euler's Method for Solving Differential Equation

Welcome back to Physics through Computational Thinking. In this video we are going to learn about Euler's method of solving ordinary differential equations. So, let us go ahead and get started. Remember when we were discussing about harmonic oscillators and simple harmonic oscillator, and then damped harmonic oscillator, anharmonic oscillator, we ran into equations which we could not solve analytically.

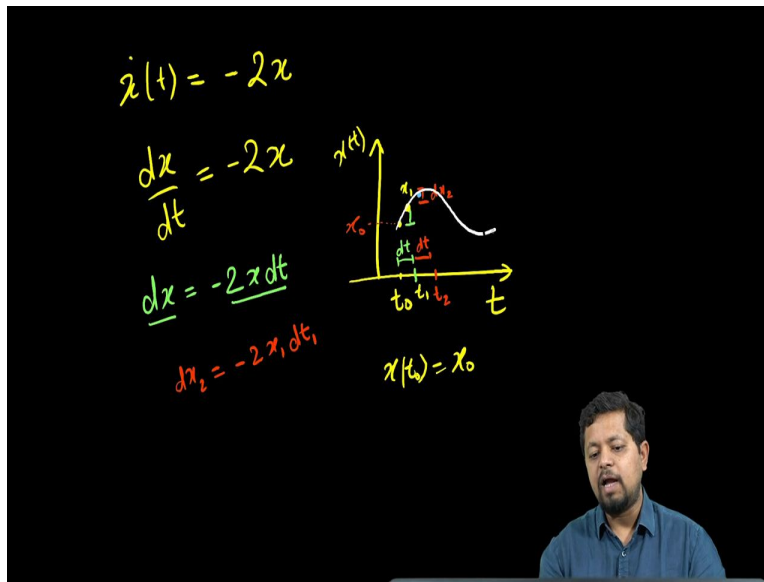
In such a case you will have to resort to a computational technique to solve for a differential equation such as Euler's method. So, Euler's method is one of the simplest methods of solving differential equations, or ordinary differential equations, and we are going to learn about the numerical method called Euler's method in this video.

(Refer Slide Time: 1:09)

Euler's Method

- It's not possible to solve all problems involving differential equations analytically. So we will learn to solve the initial value problems numerically through *Euler's method*, one of the first and simplest methods of this variety.
- Let's say your differential equation is given by
$$\dot{x} = f(x, t) \quad (1)$$
- Differentially, you can write this equation as (which is the inspiration of the Euler's method)
$$dx = f(x, t) dt \quad (2)$$
- We are interested in finding the solution of this equation between times t_i and t_f .
- In order to solve it numerically we will discretize the time into a grid N intervals and $N + 1$ time instants. $t_i = t_0$ being initial time and $t_f = t_N$ being the final time.
$$t_0, t_1, t_2, \dots, t_N \quad (3)$$
- Step size h is defined as,
$$h = \Delta t = t_n - t_{n-1} = (t_f - t_i) / N \quad (4)$$
- According to Euler's method
$$\begin{aligned} x_0 &= x_i \\ x_{n+1} &= x_n + h f(t_n, x_n) \end{aligned} \quad (5)$$
- This successively determines all the x_n .

00:02 of 5



So let us go ahead and get started. Euler's method aims to solve a differential equation of the sort $\dot{x} = f(x, t)$. So, let us consider this simple example. $f(x, t)$, where $f(x, t)$ can be any function. To give you a simple example $\dot{x}(t)$ can be simply some function of x and t , in this case, only a function of x , let us say minus $-2x$.

In order to solve this equation, the first thing that I want to do is I want to write this as dx/dt and then I want to write this as equal to $-2x$ and then usually what I do is, when I am solving this analytically, I will bring everything of x on one side and t on another side and go on but when I want to do numerically, what I want to do is I want to start from some initial position.

Let us say I have got a time axis on the x axis and I have got $x(t)$ on this axis and to start with at $t = t_0$, I know my initial value is over here, that is $x(t = 0)$ is given to me as some value x_0 , which in this case is over here, this is x_0 . Now, I want to find out what happens to x at time dt later. So, in order to find that out, all I need to do is write dx as $-2x dt$ and I can go ahead and on this graph, I can find my next point by calculating what is $-2x dt$.

That means in time d , in time Δt , that is if this is time dt ; I want to find out how much is dx , and dx is $-2x dt$. So, if I calculate that dx , that is going to be some height, let us say this much.

So, at time dt later, that is a $t_0 + dt$, my new point would have shifted over here. Then, I will repeat that process and let me call this point as t_1 , at t_1 this is my value of x , which is x_1 and then I am going to go ahead and point t_2 .

At point t_2 which is again, some dt distance away, I want to find out how much x has changed so I will go ahead and calculate dx again at point t_1 and for that, I will use so $dx_2 = -2 x_1 dt_1$; dt does not change dt is always same. So, when I calculate this quantity, I get dx_2 , let us say this is my dx_2 and if I know dx_2 , I can find out this and keep on doing that.

And that process, what will end up happening is I generate a solution for my differential equation, some curve which is represented by this white line. So, what I am doing is: I am solving a differential equation, step by step for small time dt at a time and as a consequence, I am generating the entire solution. This is the essence of the Euler's method. Let us understand this method.

(Refer Slide Time: 4:48)

Euler's Method

- It's not possible to solve all problems involving differential equations analytically. So we will learn to solve the initial value problems numerically through *Euler's method*, one of the first and simplest methods of this variety.
- Let's say your differential equation is given by

$$\dot{x}(t) = f(x, t) \quad (1)$$
- Differentially, you can write this equation as (which is the inspiration of the Euler's method)

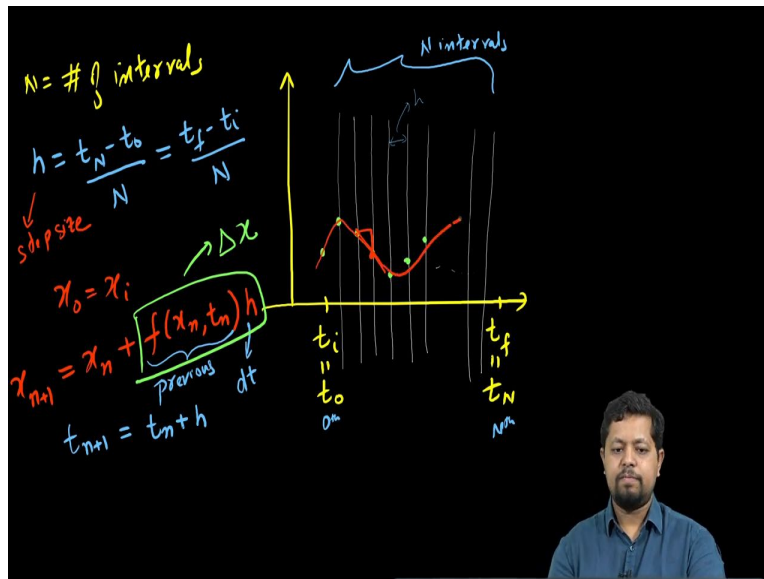
$$dx = f(x, t) dt \quad (2)$$
- We are interested in finding the solution of this equation between times t_i and t_f .
- In order to solve it numerically we will discretize the time into a grid N intervals and $N + 1$ time instants. $t_i = t_0$ being initial time and $t_f = t_N$ being the final time.

$$t_0, t_1, t_2, \dots, t_N \quad (3)$$
- Step size h is defined as,

$$h = \Delta t = t_n - t_{n-1} = (t_f - t_i) / N \quad (4)$$
- According to Euler's method

$$\begin{aligned} x_0 &= x_i \\ x_{n+1} &= x_n + h f(x_n, t_n) \end{aligned} \quad (5)$$
- This successively determines all the x_n .

Slide 2 of 5



Let us understand the systematics of this method. So, in order to implement Euler's method or what we do is we write first dx as $f(x, t) dt$. Where $f(x)$ is a derivative dx/dt . Now, what we do is we take the time slice, suppose we want to solve a differential equation from time t_0 to t_f or some t_i to t_f . Let me go ahead and do that.

Let me take my time slice, this is t_i , which are also called as t_0 and let us say this is t_f , which are also called as t_N , where N is the number of time intervals in which I want to do the calculation. So, what I am going to do is I am going to cut this graph into very thin time slices, all of equal size like that. Each of this time slices will have some width.

This width I am going to call as h , that is that is $h = t_N - t_0$ or $(t_f - t_i)/N$. So, the total number of intervals I have over here is N intervals and total number of time instants I have is $N+1$, t_0 being 0th time and since, this being the N th time instance, total $N+1$ time instances between that I have got N intervals and I can also write this as $t_f - t_i$, final time minus initial time divided by the number of intervals.

So, h is my step size, I can also call h as step size. Once I got step size, I can go ahead and once I have calculated my h like this, I can go ahead and write $x_0 = x_i$ and after that any x_{n+1} is

simply calculated as $f(x_n, t_n) * h$; h is nothing but dt or the step size and I will evaluate the function at its previous value at the previous point. So, this determines my new x_n and similarly, my new t_n or $t_{n+1} = t_n + h$.

Alright, so this is the essence of the Euler's method. This is how the Euler's method is implemented. So, now we will go ahead and implement. I am sorry, this is the increment, so let me correct this, this is the increment in. So, x_{n+1} is x_n plus the increment in x , which is the change in x , this is the Δx at that instant. So, I am adding to x_n , the change at that instant there is a Δx , and that will generate me a solution.

So, my solution will start from some initial value over here, next instant it will jump over here and so on, depending on the derivatives, f is the derivative, as you would have noticed that this f is derivative, so f is telling me the slope at this point. So, what I am doing really here is, at any given point, I am finding out what is the slope of the function, once I know the slope of the function I move dt distance along that slope and that gives me the Δx , this is the dt along the slope, and that gives me the height.

So, that determines my new point, and so on and when I join all these points through that generates me a smooth curve, so this is the idea behind the Euler's method. Let us go ahead and learn how to implement it on a computer.

(Refer Slide Time: 9:23)

For Loop

- We will use the **For** loop to implement Euler's method in Mathematica. For loop has the following structure


```
For[initialization, condition, increment, body]
```
- This means execute the body of the **For** loop starting from initialization until the condition holds to be true, executing the increment after executing the body.
- Here is an example of **For** loop to produce a list of squares.


```
array1 = {};
For[n = 1, n <= 10, n++, AppendTo[array1, n2]]
array1
{1, 4, 9, 16, 25, 36, 49, 64, 81, 100}
```
- As you are familiar, same can also be accomplished with **Table**

```
array2 = Table[n2, {n, 1, 10}]
{1, 4, 9, 16, 25, 36, 49, 64, 81, 100}
```
- Typically **Table** is faster and more efficient than **For**. We should use **For** only as a last resort. In implementation of Euler method, **Table** won't quite work as we need to access iteration for the current one.

In order to implement this on a computer, what we will do is we will use the For loop. Remember the structure of the For loop. For loop had 4 arguments inside the square brackets. The first argument was initialization, second was condition. Third was increment and 4th was body. We will use the For loop to do this iterative procedure of implementing the Euler's method. Note that we discussed that For loop is less efficient compared to a table command.

And we use For Loop when decision for this instance, or this iteration is based on what happened in the last iteration or, you know, we need the information of the last iteration to determine the value of this iteration. That is when we use the for loop, while table is something that applies on an array and something that we will try to apply on an array in one go. That is where we will try to use the table command.

So, over here, we need to know where the initial x was? Where the last x_n was? In order to determine the next x_{n+1} , therefore, we are going to use the For loop. Let us go ahead and learn how to implement this Euler's method using the For loop.

(Refer Slide Time: 10:36)

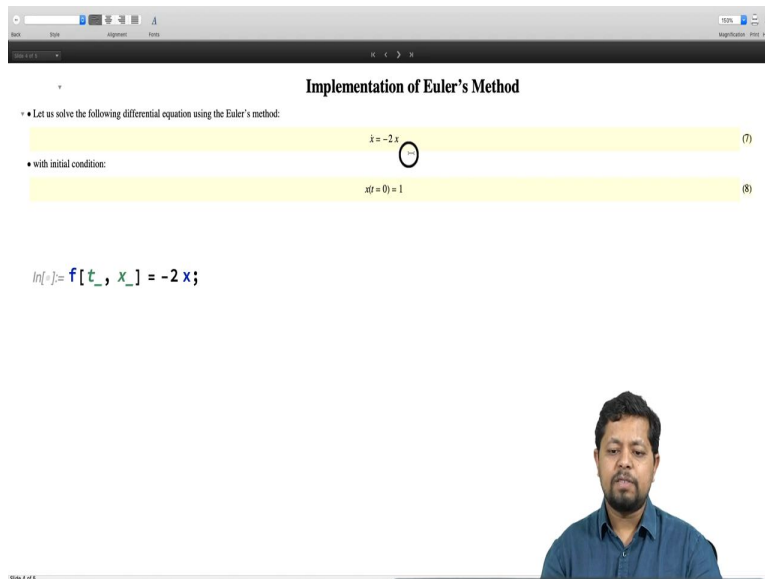
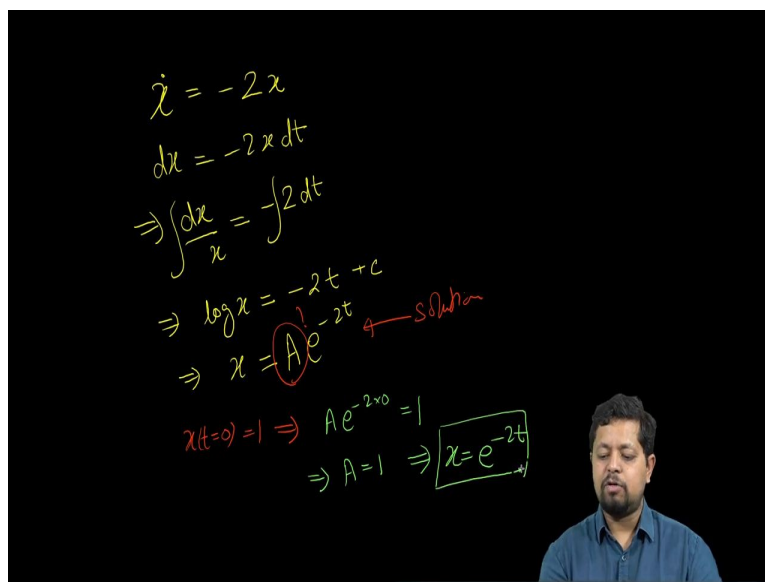
Implementation of Euler's Method

- Let us solve the following differential equation using the Euler's method:

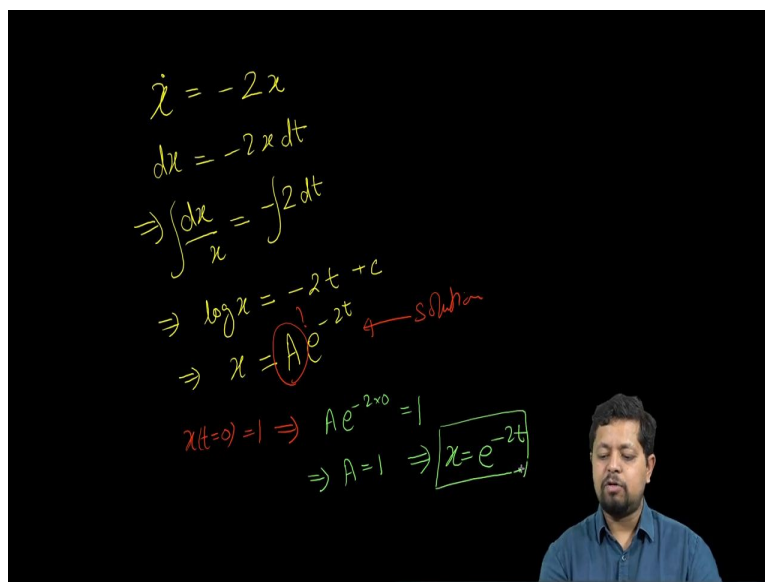
$$\dot{x} = -2x \quad (7)$$
- with initial condition:

$$x(t=0) = 1 \quad (8)$$

`ln[]:= f[t_, x_] = -2 x;`

$\dot{x} = -2x$
 $dx = -2x dt$
 $\Rightarrow \int \frac{dx}{x} = \int -2 dt$
 $\Rightarrow \log x = -2t + c$
 $\Rightarrow x = A e^{-2t}$ ← solution
 $x(t=0) = 1 \Rightarrow A e^{-2 \cdot 0} = 1$
 $\Rightarrow A = 1 \Rightarrow x = e^{-2t}$



For this purpose, let us go ahead and take this example $\dot{x} = -2x$, and the initial condition is $x(t=0) = 1$. First, let us go ahead and solve this equation analytically. So, the equation that I want to solve is $\dot{x} = -2x$, solving analytically I get $dx = -2x dt$ and I can write this as $dx/2x$, or rather $dx/x - 2 dt$. Integrating both sides, I get $\log x = -2t$ and I can write this as plus a constant and I can write this as $x = e^{-2t}$ times the constant log.

When I take the exponential e^c becomes a constant, which I am going to call A, and this is my solution, there is my solution for the equation. I need to determine what is A. To determine A, I

am going to use initial condition. Initial condition given to me is $x(t = 0) = 1$. So, that implies that $A * e^{-2*0} = 1$.

$e^0 = 1$, therefore, $A = 1$. Therefore, my solution is simply $x = e^{-2t}$. Good now that we have found the analytical solution, we will use this result to compare with our numerical solution. So, let us go ahead and implement this numerically using the For loop.

(Refer Slide Time: 12:37)

```
In[15]:= f[t_, x_] = -2 x
Out[15]=
-2 x

Initialization/Define

In[29]:= xi = 1;
          ti = 0;
          tf = 10;
          nMax = 20;
          h = (tf - ti) / nMax;
          structure of the solution: datalist = {{t0, x0}, {t1, x1}, {t2, x2} ...}

          datalist = {};
          xprev = xi;
          tprev = ti;
          datalist = Append[datalist, {tprev, xprev}]
```

```
In[34]:= datalist = {};
          xprev = xi;
          tprev = ti;
          datalist = Append[datalist, {tprev, xprev}]

Out[37]=
{{0, 1}}

For[n = 1, (*initialization*)
     n <= nMax, (*condition*)
     n++, (*increment*)
     xnext = xprev + h f[tprev, xprev]; (*body*)
     tnext = tprev + h;
     datalist = Append[datalist, {tnext, xnext}];
     xprev = xnext;
     tprev = tnext;
]

Out[38]=
```


So, here is my function. So, I will define a function as a function f , with 2 arguments, one argument being t the other being x and in order to define a function, I will use underscore so that so Mathematica understands or Wolfram language understand that t is a variable, not a particular value, x is a variable not a particular value and I say $f(t,x) = -2x$. In this case, let me remove the semicolon and execute it.

So, $f(t,x)$ is defined as $-2x$, I can go ahead and call, I can go ahead and give some value of t and x and find out whether this is working properly or not. It is clearly independent of t and you can verify that. So, that is my $f(t,x)$. Now, I am going to go ahead and define x initial, I will call that as x_i . So, $x_i = 1$ for me. Similarly, $t_i = 0$. As start, let us assume that we are going to solve the equation from $t=0$.

So, to some final time, let us say $t=10$. So my final time $t_f = 10$. Now, h is going to be $(t_f - t_i)/nMax$. Let us let us do that. Let us go ahead and also define $nMax$. $nMax$ is the number of steps that we want to use. Let us say, we have to start with this start with 20 steps. With 20 steps, let me go ahead and call $nMax$ over here. Let me go ahead and execute this statement, notice that I have defined all of these things in a single cell.

This is my initialization of the problem. This initialization of the problem, I am going to define a single cell, I do not have to write separate lines. Just this makes the code more compact, and it is easier to work with and read. So, let me go ahead and execute this to find out what is my h for $nMax$ has been defined, so the capital M, so I can do that. There we go. So, this gives me $h = 1/2$. Alright, Mathematica is a symbolic language.

So, it will give you a $\frac{1}{2}$ result 0.5. We will see what is the impact of that in a moment? So, let me also go ahead and suppress it with a semicolon. So, this is my, you know, initialization or definition of the problem. So, let me go ahead and just mark it up, this is initialization, slash define. Alright, let us go ahead now and define our For loop. But before I write for loop, I should decide how I want my solution to look like what do I want my solution for.

So, what I really want is my solution to be of the form it should be list of lists. So, I should have the structure or the solution of the form (t_0, x_0) . So, I want my solution to be in the form, (t_0, x_0) , (t_1, x_1) , (t_2, x_2) , and so on. This is how I want my solution to look like. So, if I want, if I want my solution to look like, I am going to call this as datalist, ok, this is my solution.

So, this I am going to call this datalist. So, I want to, all I know is I that I know t_0 and I know x_0 and I know t_f and I know the differential equation $\dot{x} = -2x$, I want to generate a set of points at each interval. So, distance between t_0 and t_1 is h , distance between t_1 and t_2 is h and so on. So, I want to generate this set of points (t_n, x_n) at each interval. In order to do that, I should go ahead and define my empty array data list.

So, let me go ahead and define this as empty array, ok. Now, I want datalist, I want to take this datalist and I want to prepend into this datalist. Sorry, append to this datalist or add to this datalist the first value, that is the first value x_{prev} and t_{prev} , for that I need to define what is x_{prev} and t_{prev} . So, x_{prev} , x_{prev} is simply x_i and t_{prev} is simply t_i . So, this is my previous step, which is which is what I already know and to this I want to add t_{prev} and x_{prev} .

So, let us go ahead and execute that and see. So, there we go, we get my the first element of datalist as (0,1). So, we can go ahead and check that out. So, the first element of datalist is (0,1) and that is all that is there. Now, I want to go ahead and calculate the next step, and next step and I want to do that using a For loop. So, let me go ahead and this point, start writing my For loop. So, I want my For loop to be 4 things, and I will enter each of the things in a different line.

So, first line is initialization. So, I will say let me use the counter n , so will start from $n = 0$ and I will say that is my initialization and all the arguments should be comma separated. So, then I want $n \leq nMax$ which is the number of steps I want to do, rather I should start from $n = 1$. So, $n \leq nMax$ and I want to increment n , to increment n I will simply use $n++$.

And now here I want to write the body of the For loop. So, this is the initialization part of the For loop. Notice the combination star and round bracket is used for commenting in Mathematica. So,

anything typed inside a pair of star and round bracket will be ignored. This is condition the For loop is going to work as long as this condition holds. This is incrementing step and then finally, the body appears over here. So, what we want to do in the body is we want to take, we want to find x_{next} and t_{next} .

So, $t_{\text{next}} = t_{\text{prev}} + h$, that is straightforward and $x_{\text{next}} = x_{\text{prev}} + h * f(t_{\text{prev}}, x_{\text{prev}})$. Once I have that, so this is the body starts, I will mark this up as body, the body is from here to the end. So, this is my x_{prev} , t , x_{next} and t_{next} now the what I want to do is I want to push this x_{next} and t_{next} inside the datalist, I will say datalist equal to append, append is going to add the next point into the datalist and over here I will say t_{prev} , t_{next} and x_{next} that is my append statement.

This is going to add 1 more line, 1 more set of points to datalist and now I want to update x_{prev} and t_{prev} once this is done, x_{prev} will become x_{next} , and t_{prev} should become t_{next} , so that we are preparing ourselves for the next round. Very well so you see that my body contains multiple statements and those statements are separated by semicolon. The arguments or the For loop there are 4 arguments of the for loop initialization, condition, increment and the body, those 4 arguments are separated by comma.

I pressed enter after each comma so that all these lines are separated from each other so that you can read more easily otherwise, pressing the enter is not required, I can go ahead and combine all of this in one line, it will be perfectly fine. Well it will just become a nightmare for you to read and understand. So, therefore I am putting each 1 of these statements in a new line.

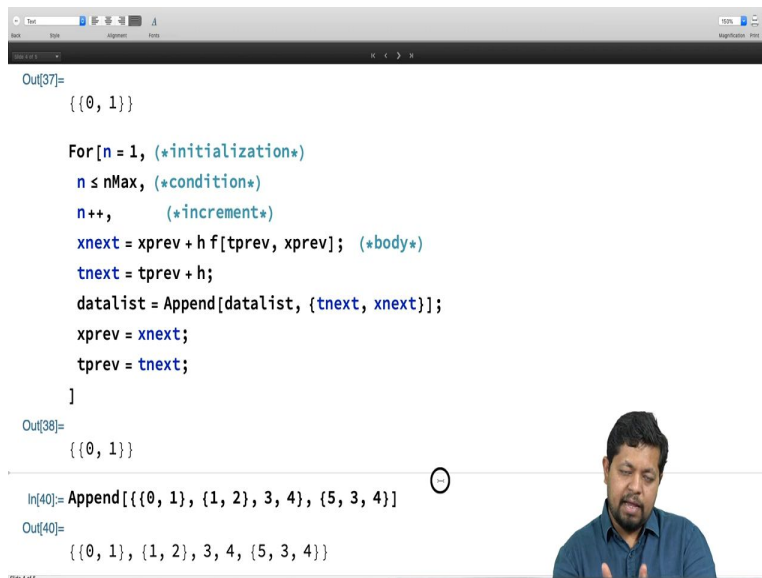
So, as a single argument of the For loop can contain multiple statements, those statements are separated by a semicolon and arguments for the for loop are separated by a comma and this is a structure which is common across Mathematica, this is not particular to the For loop. This is pretty much true for any function. Arguments are separated by commas and within a single argument you can have multiple statements separated by semicolons.

The last argument which is not, which does not contain the semicolon usually is passed on as the value for that argument. In this case, all of these statements are separated by a semicolon. So, the evaluation of this entire block is going to be null. But that does not matter to us because what we are interested in calculating is the datalist. Okay so looks like we are ready. So, let us just quickly review what we, what we have done and cross check if we have not missed anything.

So, to start from here, we initialize datalist as empty, we identified, we started out initializing x_{prev} and t_{prev} as x_i and t_i the initial points to start with, then we append it to the datalist, the t_{prev} and x_{prev} values. Once we have done that, we came down to the For loop, we decided to run For loop for from $n = 1$ to $nMax$, because we have got $nMax$ steps.

So, every time we are going to increment n by 1, and in each step, when we do that, we are going to calculate $t_{next} = t_{prev} + h$ increment time by 1 step and calculate $x_{next} = x_{prev} + h * f(t_{prev}, x_{prev})$ and that gives me the Δx value which is this. Adding it to x_{prev} gives me x_{next} , I want to insert x_{next} and t_{next} into the datalist. So, I am using the append, to append t_{next} , and x_{next} into datalist.

(Refer Slide Time: 25:25)



```

Out[37]=
{{0, 1}}

For[n = 1, (*initialization*)
n ≤ nMax, (*condition*)
n++, (*increment*)
xnext = xprev + h f[tprev, xprev]; (*body*)
tnext = tprev + h;
datalist = Append[datalist, {tnext, xnext}];
xprev = xnext;
tprev = tnext;
]

Out[38]=
{{0, 1}}

In[40]= Append[{{0, 1}, {1, 2}, 3, 4}, {5, 3, 4}]
Out[40]=
{{0, 1}, {1, 2}, 3, 4, {5, 3, 4}}

```

If we have not discussed append earlier, this is what append do, let me quickly show it to you. If I start with a list 1, 2, 3, 4 and I ask it to append 5 to this, the output will be a list in which 5 is added to the end. In the case that we are talking, we have got pairs of values like this. So, we have got a list of list. The inner list contains 2 values. Let us say something like this and to that we want to add append something else.

It does not really matter as Mathematica is can does not necessarily need same kinds of values in a list, it can take arbitrary expressions inside a list. That means all the elements of the list should not have the same properties. For example, in this case, this is my, this is my outer list from here to there, it has got 5 elements, first one is a pair. The second one is a pair.

Sorry, 4 elements first one was a pair, second one was a pair and these are single numbers to this are appending 5, 3, 4, 5, 3, 4 is a three tuple, this three tuple, this three tuple will be appended as an independent element to the list such as over here. So, now this list has 5 elements. This is the first element, the second element, this is a third element, this is the 4th element, and this is the fifth element.

Now, this was an arbitrary example to just demonstrate to you what does append do. For our purposes, all elements are of the same variety, there are 2 tuples. The first element of the 2 tuple is time. The second element of the tuple is the x variable and we are appending them in pairs, such as over here. So, my datalist will be a list of 2 tuples.

(Refer Slide Time: 27:29)

```
In[41]:= For[n = 1, (*initialization*)
  n < nMax, (*condition*)
  n++, (*increment*)
  xnext = xprev + h f[tprev, xprev]; (*body*)
  tnext = tprev + h;
  datalist = Append[datalist, {tnext, xnext}];
  xprev = xnext;
  tprev = tnext;
]

In[42]:= datalist
Out[42]=
{ {0, 1}, {1/2, 0}, {1, 0}, {3/2, 0}, {2, 0}, {5/2, 0}, {3, 0},
  {7/2, 0}, {4, 0}, {9/2, 0}, {5, 0}, {11/2, 0}, {6, 0}, {13/2, 0},
  {7, 0}, {15/2, 0}, {8, 0}, {17/2, 0}, {9, 0}, {19/2, 0}, {10, 0} }
```

Let us go ahead and execute this. So, I am going to execute it. Now I want to see what is the outcome of this? So, let me go ahead and print the datalist and there we go, we have got pairs of points. The nice thing about Mathematica here, Wolfram language here is that it is going to calculate them as symbolic expressions or fractions rather than necessarily numbers.

But there is a disadvantage of that usually such a process becomes slower. We will see that in a moment. We will compare that in a moment. But this is my datalist and you see at some point, since the solution, as we saw was exponentially decaying function, at some point, actually right over here it became 0 and then it always remains 0, ok. That is quite accidental; actually we can go ahead and try something else.

(Refer Slide Time: 28:25)

```
In[53]:= dataList
Out[53]=
```

$$\left\{ \left(0, 1 \right), \left(\frac{1}{10}, \frac{4}{5} \right), \left(\frac{1}{5}, \frac{16}{25} \right), \left(\frac{3}{10}, \frac{64}{125} \right), \left(\frac{2}{5}, \frac{256}{625} \right), \left(\frac{1}{2}, \frac{1024}{3125} \right), \right.$$
$$\left. \left(\frac{3}{5}, \frac{4096}{15625} \right), \left(\frac{7}{10}, \frac{16384}{78125} \right), \left(\frac{4}{5}, \frac{65536}{390625} \right), \left(\frac{9}{10}, \frac{262144}{1953125} \right), \left(1, \frac{1048576}{9765625} \right), \right.$$
$$\left. \left(\frac{11}{10}, \frac{4194304}{48828125} \right), \left(\frac{6}{5}, \frac{16777216}{244140625} \right), \left(\frac{13}{10}, \frac{67108864}{1220703125} \right), \left(\frac{7}{5}, \frac{268435456}{6103515625} \right), \right.$$
$$\left. \left(\frac{3}{2}, \frac{1073741824}{30517578125} \right), \left(\frac{8}{5}, \frac{4294967296}{152587890625} \right), \left(\frac{17}{10}, \frac{17179869184}{762939453125} \right), \right.$$
$$\left. \left(\frac{9}{5}, \frac{68719476736}{3814697265625} \right), \left(\frac{19}{10}, \frac{274877906944}{19073486328125} \right), \left(2, \frac{1099511627776}{95367431640625} \right), \right.$$
$$\left. \left(\frac{21}{10}, \frac{4398046511104}{476837158203125} \right), \left(\frac{11}{5}, \frac{17592186044416}{2384185791015625} \right), \left(\frac{23}{10}, \frac{70368744177664}{1192092895078125} \right), \right.$$
$$\left. \left(\frac{12}{5}, \frac{281474976710656}{59604644775390625} \right), \left(\frac{5}{2}, \frac{112509906842624}{298023223876953125} \right), \right.$$
$$\left. \left(\frac{13}{5}, \frac{4503599627370496}{1490116119384765625} \right), \left(\frac{27}{10}, \frac{18014398509481984}{7450580596923828125} \right), \right.$$
$$\left. \left(\frac{14}{5}, \frac{72057594037927936}{37252902984619140625} \right), \left(\frac{29}{10}, \frac{288230376151711744}{186264514923095703125} \right), \right.$$
$$\left. \left(3, \frac{1152921504606846976}{931322574615478515625} \right), \left(\frac{31}{10}, \frac{4611686018427387904}{4656612873077392578125} \right), \right.$$

```
In[54]:= dataList
Out[54]=
```

$$\left(\frac{10}{5}, \frac{476837158203125}{2384185791015625} \right), \left(\frac{10}{10}, \frac{1192092895078125}{1192092895078125} \right), \left. \left(\frac{12}{5}, \frac{281474976710656}{59604644775390625} \right), \left(\frac{5}{2}, \frac{112509906842624}{298023223876953125} \right), \right.$$
$$\left. \left(\frac{13}{5}, \frac{4503599627370496}{1490116119384765625} \right), \left(\frac{27}{10}, \frac{18014398509481984}{7450580596923828125} \right), \right.$$
$$\left. \left(\frac{14}{5}, \frac{72057594037927936}{37252902984619140625} \right), \left(\frac{29}{10}, \frac{288230376151711744}{186264514923095703125} \right), \right.$$
$$\left. \left(\frac{3}{5}, \frac{1152921504606846976}{931322574615478515625} \right), \left(\frac{31}{10}, \frac{4611686018427387904}{4656612873077392578125} \right), \right.$$
$$\left. \left(\frac{16}{5}, \frac{18446744073709551616}{23283064365386962890625} \right), \left(\frac{33}{10}, \frac{73786976294838206464}{116415321826934814453125} \right), \right.$$
$$\left. \left(\frac{17}{5}, \frac{295147905179352825856}{582076609134674072265625} \right), \left(\frac{7}{2}, \frac{1180591620717411303424}{2910383045673370361328125} \right), \right.$$
$$\left. \left(\frac{18}{5}, \frac{4722366482869645213696}{14551915228366851806640625} \right), \left(\frac{37}{10}, \frac{18889465931478580854784}{72759576141834259033203125} \right), \right.$$
$$\left. \left(\frac{19}{5}, \frac{7557863725914323419136}{363797880709171295166015625} \right), \left(\frac{39}{10}, \frac{30223145490365723676544}{1818989403545858078125} \right), \right.$$
$$\left. \left(4, \frac{1208925819614629174706176}{9094947017729282379150390625} \right), \left(\frac{41}{10}, \frac{4835703278458424704}{454747350886464151953125} \right), \right.$$
$$\left. \left(\frac{21}{5}, \frac{19342813113834066795298816}{227373675443232059478759765625} \right), \right.$$
$$\left. \left(\frac{43}{10}, \frac{77371252455336267181195264}{112606277316160207303700890625} \right), \right.$$

```

29 83 076 749 736 557 242 056 487 941 267 521 536
5 34 694 469 519 536 141 888 238 489 627 838 134 765 625
59 332 306 998 946 228 968 225 951 765 070 086 144
10 173 472 347 597 680 709 441 192 448 139 190 673 828 125
6 1 329 227 995 784 915 872 903 807 060 280 344 576
61 5 316 911 983 139 663 491 615 228 241 121 378 304
10 4 336 808 689 942 017 736 029 811 203 479 766 845 703 125
31 21 267 647 932 558 653 966 460 912 964 485 513 216
5 21 684 043 449 710 088 680 149 056 017 398 834 228 515 625
63 85 070 591 730 234 615 865 843 651 857 942 052 864
10 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125
32 340 282 366 920 938 463 463 374 607 431 768 211 456
5 542 101 086 242 752 217 003 726 400 434 970 855 712 890 625
13 1 361 129 467 683 753 853 853 498 429 727 072 845 824
2 2 710 505 431 213 761 085 018 632 002 174 854 278 564 453 125
33 5 444 517 870 735 015 415 413 993 718 908 291 383 296
5 13 552 527 156 068 805 425 093 160 010 874 271 392 822 265 625
67 21 778 071 482 940 061 661 655 974 875 633 165 533 184
10 67 762 635 780 344 027 125 465 800 054 371 356 964 111 328 125
34 87 112 285 931 760 246 646 623 899 502 532 662 132 736
5 338 813 178 901 720 135 627 329 000 271 856 784 820 556 640

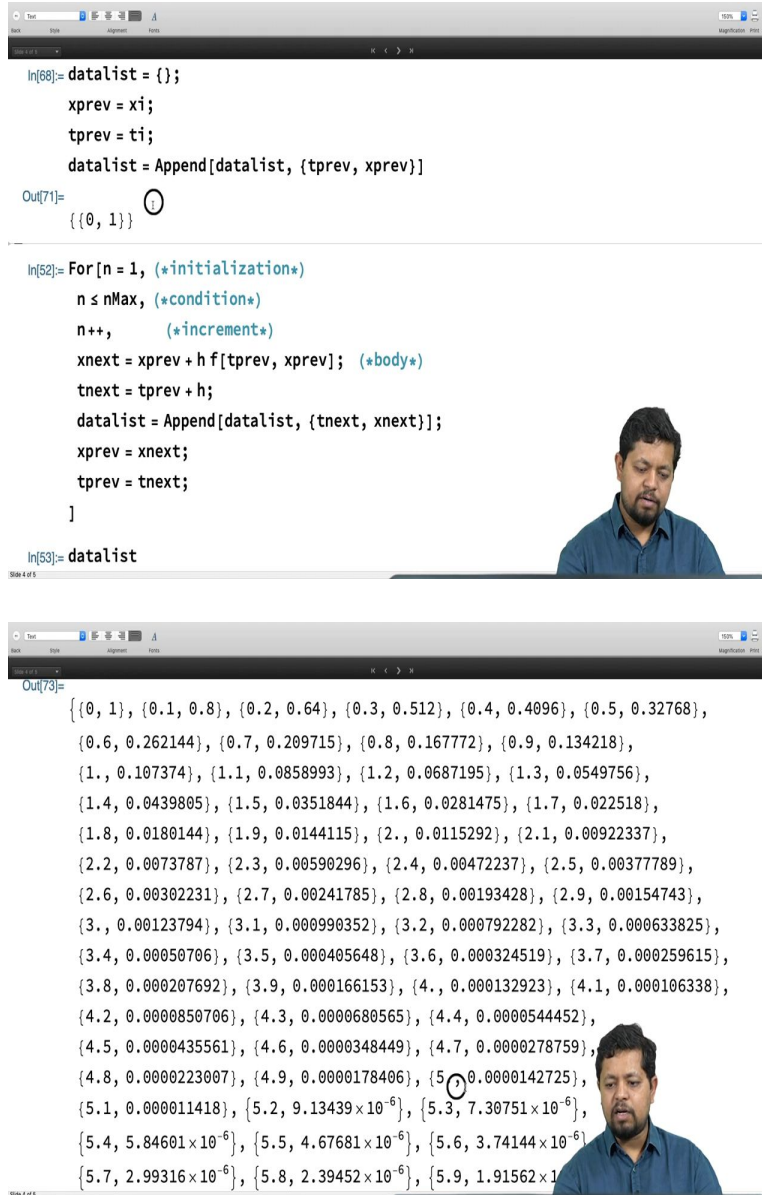
```



Now we can go ahead and make maybe n equal to 100 and then initialize this again, and then run the For loop again, and then print the datalist again and there you go. You see a long list of numbers over here, ok. This becomes very difficult to read and that is what I was saying and then also actually takes us to take a lot of computation time. So, there are 100 numbers over here. These distractions are small numbers, but they are much more difficult to read. So, what we will do is we will make sure that this is done numerically.

So, to make that simply, what we will do is we will simply put a $//N$ over here. So, that itself is evaluated numerically rather than as a exact expression. So, $h = 0.1$. The moment h becomes 0.1 every time a decimal number enters in the calculation over here and as a consequence, all these numbers will get evaluated, numerical values which will become much-much more readable.

(Refer Slide Time: 29:23)



```
In[68]:= datalist = {};  
xprev = xi;  
tprev = ti;  
datalist = Append[datalist, {tprev, xprev}]  
  
Out[71]=  
{0, 1}
```

```
In[52]:= For[n = 1, (*initialization*)  
n <= nMax, (*condition*)  
n++, (*increment*)  
xnext = xprev + h f[tprev, xprev]; (*body*)  
tnext = tprev + h;  
datalist = Append[datalist, {tnext, xnext}];  
xprev = xnext;  
tprev = tnext;  
]  
  
In[53]:= datalist
```

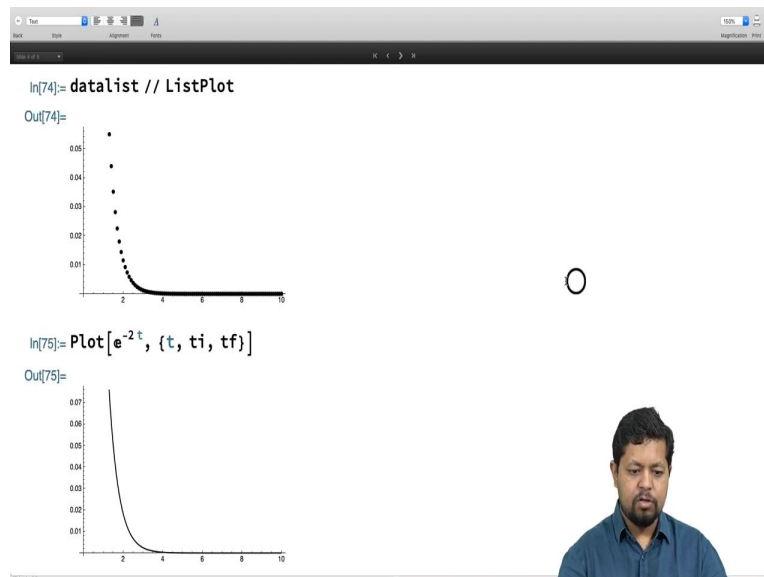
```
Out[73]=  
{0, 1}, {0.1, 0.8}, {0.2, 0.64}, {0.3, 0.512}, {0.4, 0.4096}, {0.5, 0.32768},  
{0.6, 0.262144}, {0.7, 0.209715}, {0.8, 0.167772}, {0.9, 0.134218},  
{1., 0.107374}, {1.1, 0.0858993}, {1.2, 0.0687195}, {1.3, 0.0549756},  
{1.4, 0.0439805}, {1.5, 0.0351844}, {1.6, 0.0281475}, {1.7, 0.022518},  
{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 × 10-6}
```

So, let us go ahead and do that. So, let us go ahead and execute this statement again. Initialize this part again, if you do not remember the data list has not been set to empty now means attempt to initialize and running the for loop will simply add more and more values to the previous data list. So, that is the reason why I need to run this set of statements again, that is why I am pressing shift plus enter over here.

Now at the for loop, when I run it again, I am going to get this datalist and now you can see it is much more readable. At $t = 0$, I have $x = 1$ which is my initial condition, at $t = 0.1$, x is 0.8, $t = 0.2$ it is 0.64 and so on and so forth dropping down and very quickly at about 3 or 4, it becomes a very small number dies out, this is what you expected. For this equation, remember this equation was $\dot{x} = -2x$, the solution we found was $x = e^{-2t}$. So, it is exponentially decaying solution.

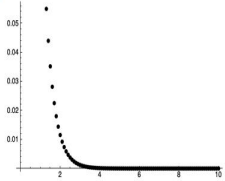
For exponentially decaying solution, we expect x to become smaller and smaller, converging to 0 and that is exactly what is happening over here. That is what you expected. So, let us go ahead and plot this plot this and find out if this is exactly the solution that I wanted is exponentially decaying as I wanted or it is something that is decaying, but not exactly the solution that I wanted, ok? So, what we will want to do is we are going to do a list plot.

(Refer Slide Time: 30:41)



```

n++, (*increment*)
xnext = xprev + h f[tprev, xprev]; (*body*)
tnext = tprev + h;
datalist = Append[datalist, {tnext, xnext}];
xprev = xnext;
tprev = tnext;
]

In[74]:= datalist // ListPlot
Out[74]=

Plot[e-2t, {t}]

```

Slide 4 of 5

```

ti = 0;
tf = 10;
nMax = 200;
h = (tf - ti) / nMax // N
Out[83]=
0.05

```

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

```

In[68]:= datalist = {};
xprev = xi;
tprev = ti;
datalist = Append[datalist, {tprev, xprev}]
Out[71]=
{{0, 1}}

```

```

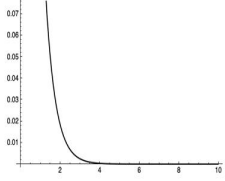
In[72]:= For[n = 1, (*initialization*)
n ≤ nMax, (*condition*)

```

Slide 4 of 5

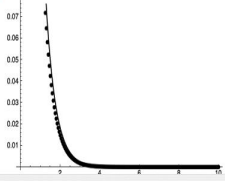
```
In[90]:= solnplot = Plot[e-2t, {t, ti, tf}]
```

Out[90]=




```
In[91]:= Show[solnplot, dataplot]
```

Out[91]=



Slide 4 of 5



```
Initialization/Define
```

```
In[92]:= xi = 1;  
ti = 0;  
tf = 10;  
nMax = 500;  
h = (tf - ti) // N  
nMax
```


Out[96]=
0.02

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

```
In[84]:= datalist = {};  
xprev = xi;  
tprev = ti;  
datalist = Append[datalist, {tprev, xprev}]
```

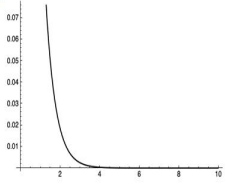
Out[87]=
{{0, 1}}

Slide 4 of 5



```
soInPlot = Plot[e^-t, {t, ti, tf}]
```

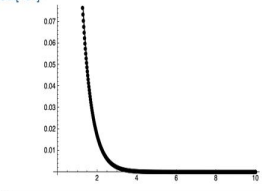
Out[103]=




In[104]=

```
Show[soInPlot, dataplot]
```

Out[104]=



Slide 4 of 5



```
tf = 10;  
nMax = 10 000;  
h = (tf - ti) / nMax
```

Out[109]=

0.001

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


```
In[97]= datalist = {};  
xprev = xi;  
tprev = ti;  
datalist = Append[datalist, {tprev, xprev}]
```

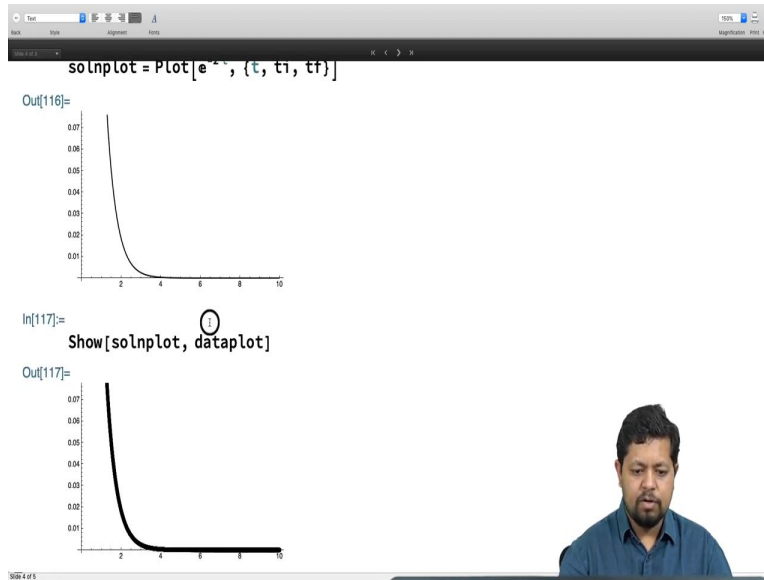
Out[100]=

```
{{0, 1}}
```

```
In[101]= For[n = 1, (*initialization*)  
n <= nMax, (*condition*)  
n++, (*increment*)
```

Slide 4 of 5





We will enclose this inside a list plot, you can also do a postfix so for example, we could do that, that does look like a you know, a decaying function, but is it the solution that we are looking for, so we have to compare this with our solution. So, let us go ahead and plot our solution. Our solution was e^{-2t} and we want it from $t = 0$ to t_f , which was, t_f was 10.

So, it means put t_f over here. This is t_f , t_i to t_f . That is my solution. Let me go and put them on top of each other so that you can compare and do that, I will use the show command. I will say. For that I will define them as something. So, I will call this as you know data plot. This is my data plot. I will give them here is my solution plot, I will put, solution plot and data plot on this, show them together. You see, they are quite close, but not right on top of each other.

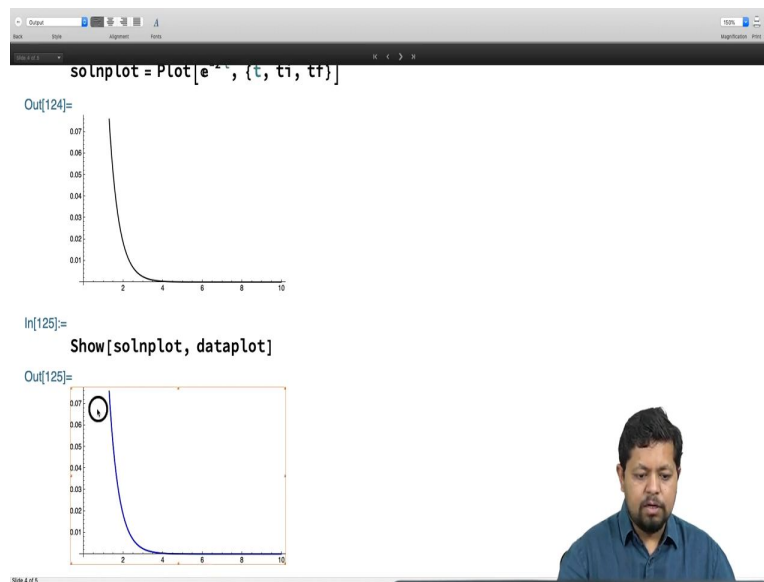
That is quite interesting. So, we got really close to the solution that we expected. So, this is my the dots are like computed value and this is the solid curve is my analytical value, and they are very close to each other, but they do not quite overlap. The reason for that has to do with the fact that the step sizes are probably too large; h is 0.1, which is probably too large for this thing, I want to make x smaller so maybe I want to make it 200 but I make it 200, h becomes 0.05.

So, let us go ahead and rerun it. Rerun the for loop again. Make the data plot again, ok, it becomes more denser, make the solution plot and show them together and that got a little bit

better, they came closer. So, which means that I need to make my h further smaller to get an even better agreement. So, let me go ahead and do that. So, let me make it 500. So, that makes h 0.02 and then we will go ahead and do this, execute this for loop, make the data plot, make the solution plot and show them on top of each other now they are in almost perfect agreement.

So, now you see that step size is extremely important. If your step size is too large, you might get roughly the shape of the solution but not you will learn exactly the solution. So, you need to choose your h appropriately. The same time so you can say you know I can go ahead and make h very-very small by choosing large number of steps. So, for example, I can go ahead and make it 10,000 but I do tend to do make it 10,000 because very small and I can go ahead and you know, do this and produce a really dense plot and I will get a perfect solution, ok.

(Refer Slide Time: 34:25)



```

ti = 0;
tf = 10;
nMax = 500;
h = (tf - ti) / nMax // N
Out[130]=
0.02

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

In[110]:=
datalist = {};
xprev = xi;
tprev = ti;
datalist = Append[datalist, {tprev, xprev}]

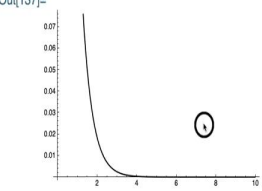
Out[113]=
{{0, 1}}

In[114]:=
For[n = 1, (*initialization*)

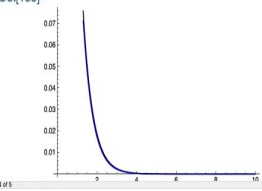
```

```

solnplot = Plot[e^-2 t, {t, ti, tf}]

Out[137]=


In[138]:=
Show[solnplot, dataplot]

Out[138]=


```

Over here you see almost a perfect solution in order to see the perfectness of that solution. I will join the points here. I want to actually remove the points, so I should do pointSize as, that's a wrong option I think. I need to see what is the correct option? I think it is plot marker, plot markers as none should do that. There we go. So, that is my, let me maybe make the plotStyle as blue. Now I will go ahead and put this on top of that and you see they are right on top of each other.

In fact, they are right on top of each other and for a very small h , in this case, we see we have produced a perfect solution. But the cost of that is, our datalist is very long. If this is a more complicated function to evaluate, the time it is going to take is much, much larger. So, therefore, it is not very-very beneficial to actually have such a small value of h or such a large number of step sizes. To find a balance between the accuracy that we want and the number of steps to do the calculation that is required.

For simple problems like this, it does not really matter, you can choose $nMax$ to be very large and h to be very small. But in order to optimize, we may probably want to, you know, decide to stick to some number between 500 or 1000, I thought 500 was pretty decent. So, let us go back to 500 and do this again. There we go. So, 500 we see it is quite decent, the black and blue curves are sitting almost on top of each other and the computational time involved is also very-very less.

(Refer Slide Time: 37:31)



```
dataList = Append[dataList, {tprev, xprev}];

Out[147]=
{{0, 1}}


In[148]:=
For[n = 1, (*initialization*)
  n <= nMax, (*condition*)
  n++, (*increment*)
  xnext = xprev + h f[tprev, xprev]; (*body*)
  tnext = tprev + h;
  dataList = Append[dataList, {tnext, xnext}];
  xprev = xnext;
  tprev = tnext;
] // Timing

Out[148]=
{0.001905, Null}


In[136]:=
dataplot = ListPlot[dataList, Joined -> True, PlotMarkers -> None];
```

The screenshot shows a Mathematica notebook interface. The code defines a list of points and iteratively updates it using a numerical method. The output shows the initial point {0, 1} and the execution time for the loop, which is 0.001905 seconds. A small video inset of a man is visible in the bottom right corner of the notebook window.

```
ti = 0;
tf = 10;
nMax = 10000;
h = (tf - ti) / nMax // N
Out[153]=
0.001
structure of the solution: datalist = {{t0, x0}, {t1, x1}, {t2, x2} ...}
In[144]:=
datalist = {};
xprev = xi;
tprev = ti;
datalist = Append[datalist, {tprev, xprev}]
Out[147]=
{{0, 1}}
In[148]:=
For[n = 1, (*initialization*)
```



```
tprev = ti;
datalist = Append[datalist, {tprev, xprev}]
Out[157]=
{{0, 1}}
In[158]:=
For[n = 1, (*initialization*)
n <= nMax, (*condition*)
n++, (*increment*)
xnext = xprev + h f[tprev, xprev]; (*body*)
tnext = tprev + h;
datalist = Append[datalist, {tnext, xnext}];
xprev = xnext;
tprev = tnext;
] // Timing
Out[158]=
{0.243042, Null}
```



```

In[159]:=
  xi = 1;
  ti = 0;
  tf = 10;
  nMax = 500;
  h = (tf - ti) // N
Out[163]=
  0.02

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

In[154]:=
  datalist = {};
  xprev = xi;
  tprev = ti;
  datalist = Append[datalist, {tprev, xprev}]

Out[157]=
  {{0, 1}}

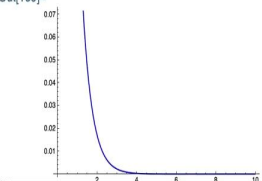
```

```

  xnext = xprev + h f[tprev, xprev]; (*body*)
  tnext = tprev + h;
  datalist = Append[datalist, {tnext, xnext}];
  xprev = xnext;
  tprev = tnext;
] // Timing

Out[168]=
  {0.001803, Null}

In[169]:=
  dataplot = ListPlot[datalist, Joined -> True, PlotMarkers -> None, PlotStyle -> Blue]

Out[169]=


```

If you do not believe me, you can go in and you can go out and try this out, you can check the computation time. So, let us say with 500 points, the computation time involved is in the For loop. So, at the end of the For loop, I will apply timing to it and calculate the time. So, with 500 points, it took 0.0019 seconds, that is 0.002 seconds to calculate 500 steps for this function. Now, if I make it 10,000 that will be much larger than that.

So, let us go ahead and check that out. What happens when I make it very small? How does the computation time change? You see, it is from 0.002 seconds it has become 0.2 seconds, became

almost 100 times as much for doing this computation. So, if you are solving a much more difficult and challenging problem, the time cost becomes 100 times more that means that much more computational power, you require that much more hardware you require and that is something that is not going to be feasible for all the problems that you are going to solve.

So, therefore, one has to think of optimizing the algorithm that you are using for the accuracy that you want. So in this case, the accuracy that we want, $nMax$ equal to 500 is quite decent. So, we will stick to that value. As we go along in this course, we will explore more methods of advanced methods, improved methods, which will take the small value, reasonably large value of h , but still give me smaller accuracy.

But let us go ahead and do a little bit more, you know, learn how to make this code more efficient. So, let us quickly review what we have done. We initialize the problem by giving the x initial, t initial, t final calculate gave decided some value $nMax$, calculate h , then we initialized datalist, previous value of x , previous value of time, and then we appended x previous, t previous datalist and then we ran the, executed the For loop.

So, this was our code. But I can actually make my code much more efficient and much more compact. So, let us go ahead and see how to do that. So, let me leave this part as it is, which is my definition part, what I will do is I will get rid of all of this by simply putting that inside the For loop over here. In fact, I do not even need n . Notice that I am not using n anywhere inside the body of the For loop, n is just being used as the counter.

(Refer Slide Time: 39:49)

```

0.02
structure of the solution: datalist = {{t0, x0}, {t1, x1}, {t2, x2} ...}

For[datalist = {{ti, xi}};
  xprev = xi;
  tprev = ti, (*initialization*)
  Length[datalist] < nMax + 1, (*condition*)
  n++, (*increment*)
  xnext = xprev + h f[tprev, xprev]; (*body*)
  tnext = tprev + h;
  datalist = Append[datalist, {tnext, xnext}];
  xprev = xnext;
  tprev = tnext;
] // Timing

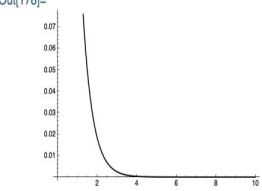
Out[168]=
{0.001803, Null}

In[169]=
datalist = ListPlot[datalist, Joined -> True, PlotMarkers ->

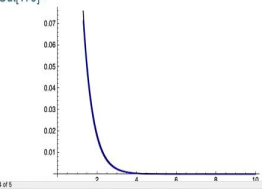
```

```

In[177]=
solnplot = Plot[e^-2 t, {t, ti, tf}]

Out[178]=


In[179]=
Show[solnplot, dataplot]

Out[179]=


```

So, I can actually do that by simply rather than using n for initialization, I can go ahead and initialize data list right over here as the previous value, so let me go ahead and decide, define the previous value. So, let me go ahead and the previous value is t_i and x_i . So, this is my datalist and since I said I can enter multiple statements at the same argument. So, in the initialization argument, I will enter multiple arguments by separating them with semicolons and for that purpose, I will also say x previous as x_i .

So, this is my initialization of x previous, and t previous as t_i . So, all of this is initialization for me, that takes care of all these 4 lines and I can go ahead and delete them. They are not required for me, ok, great. So, now my new initialization contains x previous, t previous and datalist initialization all together in 1 statement. I do not really require this either, all I need to do is check the length of the datalist.

So, say the length of datalist should be less than equal to $nMax + 1$. It already has the initial point t_0, x_0 or t_i, x_i and total number of instances are going to have been inside the datalist is $nMax + 1$ because $nMax$ is the number of steps. So, then I do not need this increment $n++$, in fact, I can copy this line. I can go ahead and do this in increment, increment is always executed after the body, so I can remove these 2 lines and put them in the increment.

There we go. So, $x_{prev} = x_{next}$ and $t_{prev} = t_{next}$ becomes my increment and my body becomes calculation of x_{next} and t_{next} and I can also go ahead and remove this because this is the increment step. This is not really a calculation, I am just incrementing my datalist. So, I can go ahead and also that in the increment.

So, now my initialization contains initialization of datalists and x_{prev} and t_{prev} . In the condition checking I simply check whether my data list has contains $nMax + 1$ instances or not, if it is less than that continue to run. When the body is executed it calculates x_{next} and t_{next} from x_{prev} and t_{prev} .

Once that has been calculated I increment my x_{prev} , t_{prev} and the datalist, so I can organize my for loop up in that particular way, and let us go ahead and check that this is actually doing the same job. So, let me go ahead and start from this definition part. I will go ahead and exclude the For loop make the data plot, it is the same data plot. Of course, the same result.

(Refer Slide Time: 43:28)

```

tf = 10;
nMax = 500;
h = (tf - ti) / nMax // N
Out[188]=
0.02

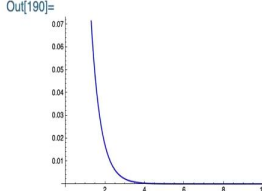
structure of the solution: datalist = {{t0, x0}, {t1, x1}, {t2, x2} ...}

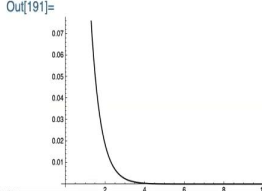
In[189]:=
For[datalist = {{ti, xi}}; prev = {ti, xi};, (*initialization*)
Length[datalist] < nMax + 1, (*condition*)
prev = next;
datalist = Append[datalist, next];, (*increment*)
next = prev + {h, h f @@ prev}; (*body*)
] // Timing
Out[189]=
{0.002478, Null}

In[177]:=
dataplot = ListPlot[datalist, Joined -> True, PlotMarkers -> None, PlotStyle -> Blue]

```

```

In[190]:=
dataplot = ListPlot[datalist, Joined -> True, PlotMarkers -> None, PlotStyle -> Blue]
Out[190]=


In[191]:=
solnplot = Plot[e^-2 t, {t, ti, tf}]
Out[191]=


```

I can go ahead and make my For loop even more compact. In fact, what I will do is rather than writing x_{prev} and t_{prev} separately, I will just call them as one item that is previous and simply go ahead and define it as a pair of 2 numbers, t_i and x_i . Once I define my previous like that, my updation also is becomes previous is equal to the next.

So it let me write it as next and this should be my next I can calculate next also in 1 go by saying next is equal to next is a set of 2 items. So, that is $prev + h$ and $h*f(prev)$. So for that I will take

$f@@prev$, note the use of at at operator. f acting on previous is going to remember the previous is a 2 tuple, that is what it is intended, a previous is (1,2).

$f@@prev$ is going to or let me call it $g@@prev$ will be $g(1,2)$, what we want to do is we want to take our function f and apply it to the arguments 1 and 2, and now evaluate to some number. All right, so what is this statement doing is it is calculating, I do not need this statement anymore now. So, you see my For loop becomes much more compact now.

I am calculating the next point equal to previous point plus the change in the previous point or the increment in the previous point. So, previous point is a list of 2 items. To that I am adding another list or 2 items and in Wolfram language or Mathematica, the 2 lists can be added 1 item by item. Therefore, h will be added to the first component of previous which is the time value and $h*f$ applied to previous will be calculated and added to the second argument of previous thus giving me next.

And once I open next in the increment part, I am going to set previous equal to next and append to datalist next and that makes my For loop very-very compact. So, let me go ahead and initialize this again. Run the For loop again and calculate this again and you see the same results. So, during the intended task, and so what we have accomplished by doing this as I made my code much more compact.

Notice that it scrolled back and forth up and down again and again to see my code before but now my entire code fits in in 1 line, we can in fact, go ahead and make this code even more compact and add a few more lines. But we will take this up and the next time when we go ahead and improve our Euler's method to Runge-Kutta second order or improved Euler method. So, we will see you next time.