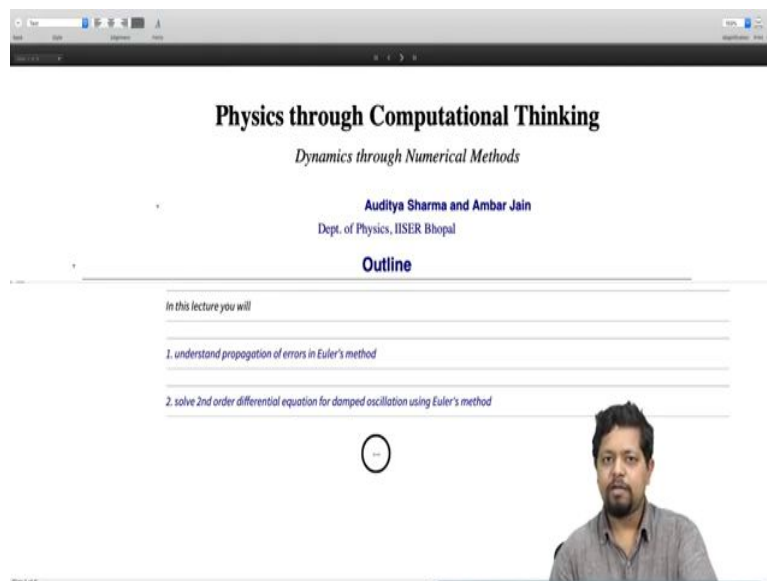


Physics through Computational Thinking
Professor Dr. Auditya Sharma
Dr. Ambar Jain
Department of Physics
Indian Institute of Science Education and Research, Bhopal
Lecture 28
Mean Global Error in Euler's method and Application of Euler's method

(Refer Slide Time: 00:26)



Welcome back, last time we learned how to implement Euler's method on the computer using Wolfram language on Mathematica and we learned how to solve problems using Euler's method, how to solve ordinary differential equations in one dimension using Euler's method. Today we are going to learn, how the error propagates in Euler's method that is how the step size matters in the precision of the calculation.

And we will also see how to solve multi-dimensional problems that is 2-dimensional problems or 3-dimensional problems using Euler's method and this will be our steps to build on to more rigorous, more complicated methods such as Runge Kutta second order, Runge Kutta fourth-order. So, let us go ahead and get started.

(Refer Slide Time: 01:13)

• For the differential equation of the type

$$\dot{x}(t) = f(t, x) \quad (1)$$

$$x(t_0) = x_0$$

• we implemented Euler's method using the `Module` function below

```
In[247]:=
euler[func_, xi_, ti_, tf_, nMax_] := Module[{h, datalist, prev},
  h = (tf - ti) / nMax // N;
  For[datalist = {{ti, xi}},
    Length[datalist] < nMax,
    AppendTo[datalist, prev + {h, h func @@ prev}],
    prev = Last[datalist];
  ];
  Return[datalist];
]
```

• where, the input arguments are:

- `func`: a function of $t, x, \dot{x} = \text{func}(t, x)$
- `ti`: initial time or start time for computation
- `tf`: final time or end time for computation
- `xi`: initial value of x at $t = t_0$
- `nMax`: number of time interval (or step size).

• The expected **local error** in computation is of the order of h^2 : $O(h^2)$

Slide 1 of 6

```
prev = Last[datalist];
];
Return[datalist];
]
```

• where, the input arguments are:

- `func`: a function of $t, x, \dot{x} = \text{func}(t, x)$
- `ti`: initial time or start time for computation
- `tf`: final time or end time for computation
- `xi`: initial value of x at $t = t_0$
- `nMax`: number of time interval (or step size).

• The expected **local error** in computation is of the order of h^2 : $O(h^2)$

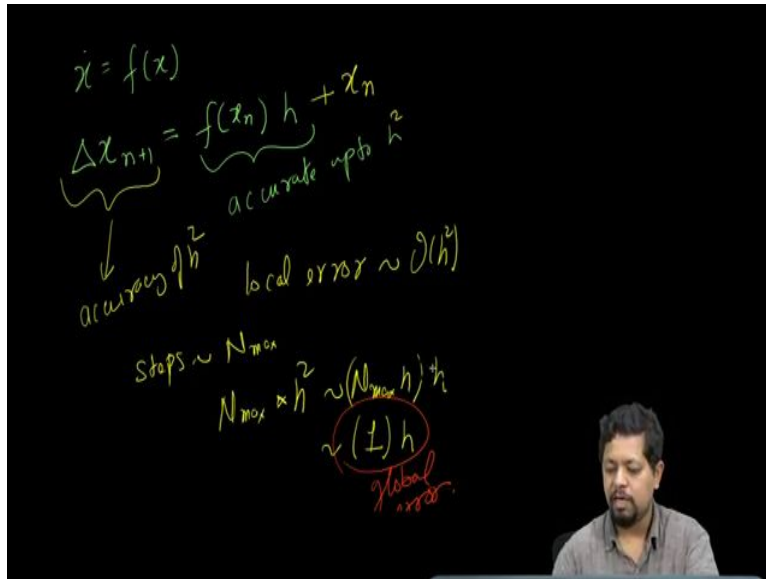
• The expected **global error** in the computation is of the order of h : $O(h)$

• We implemented the following `err` function to compute the mean global error given by equation

$$\text{err} = \frac{1}{N} \sum_{i=1}^N |x_i - f(t_i)| \quad (2)$$

```
err[dataset_, func_] := Module[{tlist, xlist, Fxlist},
  tlist = dataset[[;;, 1]]; (*Extract each time value*)
  xlist = dataset[[;;, 2]]; (*Extract each x value*)
  Fxlist = func /@ tlist;
  (*Apply func to each time value to get list of func[t, x]*)
  Return[Fxlist - EvFlist // Abs // Mean]
```

Slide 1 of 6



Just a quick review of Euler's method. We have a differential equation $\dot{x}=f(x, t)$ and some initial conditions $x(t_0)=x_0$ and this is how we implemented we used a for loop. We have used module construct to construct a single function, which has a bunch of inputs, one of the arguments or inputs of the function Euler, that we defined was function initial value of x, the initial value of time.

Final value of time up to which we want to evolve or we want to solve the differential equation and a number of steps which is in nMax. In the first we calculate, $h=(tf-ti)/nMax$. Then, we start over for loop and in each step, we calculate the next step for x, this is initialization step, we define the first element of data list which is the solution that we are looking for, so our solution is stored in the data list.

The condition is the length of the database should be less than nMax. We decided to, in the increment part, append data list with the previous plus the increment and this increment is both the increment in time and increment in x, f acting on the previous value is giving me the derivative which multiplied with h, gives me delta x.

So, h is the increment in time, $h * f(x)$ in t is an increment in x and together is added to the previous which is the two tuples. And, in the body we are just simply reading or updating the previous reading the last element of the data list. So, in that process for loop generates the

entire solution and at the end of the for loop, we return data list that ends the function. So, I will go ahead and execute it and that is my Euler function.

Since, Euler function, we are evaluating with size h that is we are solving a differential equation. $\dot{x} = f(x, t)$ and in that process, in the Euler's method we are calculating $\Delta x_{n+1} = f(x_n) * h$ or Δt . So, this is accurate up to h , which means that x and f , Δx is accurate or this is accurate to order h^2 because this is already order h .

So, Δx_n is correct up to the order h^2 . So, this is correct up to order h and the correction is order h^2 . So, I know Δx_n with an accuracy of its square. Therefore, a local error is of the order of h^2 . But that means the correction in Δx_{n+1} corresponding to x_n , oh sorry I am missing x_n over here. I should have a summation over x_n i.e., $\Delta x_{n+1} = f(x_n) * h + x_n$.

The determination Δx_{n+1} in comparison to x_n is accurate to order h^2 . The correction itself is of order h and the error in it is going to be at most order h^2 . Now, so at each step we are making a correction of order h^2 but we have to do of the order of N_{max} steps. So, by the time we reach the end of the solution, the number of steps involved is on the order of n_{max} .

So, therefore $N_{max} * h^2 \sim (N_{max} * h) * h$, $N_{max} * h$ is nothing but $(t_f - t_i)$ which is an order one number and this is h . So, therefore global error becomes order h . And, this is exactly what we are going to analyse now using numeric.

So, let us go back to our court, in order to find out the global error, what we are going to do is we are going to calculate the error using this formula, where we are going to take x_i , which is the x_i that we completed and $f(t_i)$, where $f(t_i)$ is the analytical solution at time t_i , So t_i is the expected analytical solution and x_i is something that we have completed.

We take the difference of the two, take the absolute value and sum over all i 's, from i equal to 1 to n and then we divide by n to calculate the mean absolute error or mean absolute deviation.

(Refer Slide Time: 06:13)

The slide contains the following text and code:

- Initial value or start time for computation
- Final time or end time for computation
- Initial value of x at $t = t_1$
- N : number of time interval (or step size).
- The expected local error in computation is of the order of h^2 , $O(h^2)$
- The expected global error in the computation is of the order of h , $O(h)$
- We implemented the following error function to compute the mean global error given by equation

$$err = \frac{1}{N} \sum_{i=1}^N |t_i - f(t_i)| \quad (2)$$

```
err[dataset_, func_] := Module[{tlist, xlist, Fxlist},
  tlist = dataset[[;;, 1]]; (*Extract each time value*)
  xlist = dataset[[;;, 2]]; (*Extract each x value*)
  Fxlist = func /@ tlist;
  (*Apply func to each time value to get list of func[t_i]*)
  Return[xlist - Fxlist // Abs // Mean];
]
```

• Calculate the mean global error for various value of h for fixed t_1 and t_f

```
in[ ] := f[t_, x_] = -x t;
ff[t_] = e^-t^2/2;
r, 5.0 err[Euler[f, 1.0, 5. 10^1, ff],
```

In order to do that, we will again use the module construct and we wrote down the error function, this is something we did last time, it is the quick review of that. We wrote down the error function, this is the error function.

In the error function, we again use a module construct `tlist`, `xlist` and `Fxlist` where `tlist`, we extracted the first element of the data set, data set is the argument that we are passing. This data set is the solution that we obtained through the Euler method that is this data list that the other method returned. We are passing that to the error function, this error function, the first element is the time component and the second element is the x component.

This stands for all the rows, so we extract with the time component, we extracted with the x component, then we calculated a list of true values of f by applying the function, that is this function which is the big F , the solution of the differential equation. We applied f at `tlist` to obtain the true values, then in order to find the mean absolute error we took the difference of `xlist` and `tlist` to apply the absolute function on it through the post-fixed application.

And, then calculate the mean again through post-fixed application and then return that to give me the mean absolute error.

(Refer Slide Time: 07:36)

```

err[dataset_, func_] := Module[{tlist, xlist, Fxlist},
  tlist = dataset[;;, 1]; (*Extract each time value*)
  xlist = dataset[;;, 2]; (*Extract each x value*)
  Fxlist = func /@ tlist;
  (*Apply func to each time value to get list of func[t;]*)
  Return[xlist - Fxlist // Abs // Mean];
]

```

• Calculate the mean global error for various value of h for fixed t_j and t_f

```

f[t_, x_] = -2 x;
ff[t_] = e-2t;
Table[{5.0/10n, err[euler[f, 1, 0, 5, 10n], ff]}, {n, 1, 4}]

```

Out[] = {{0.5, 0.0746396}, {0.05, 0.0639748}, {0.005, 0.0631697}, {0.0005, 0.0630899}}

```

In[ ] := err[euler[f, 1, 0, 5, 50], ff]
Out[ ] = 0.00648814

```

• We see that the global error scales at h, thus the error is considered as O(h).

• The expected global error in the computation is of the order of h: O(h)
 • We implemented the following err function to compute the mean global error given by equation

$$err = \frac{1}{N} \sum_{i=1}^N |y_i - f(t_i)| \quad (2)$$

```

err[dataset_, func_] := Module[{tlist, xlist, Fxlist},
  tlist = dataset[;;, 1]; (*Extract each time value*)
  xlist = dataset[;;, 2]; (*Extract each x value*)
  Fxlist = func /@ tlist;
  (*Apply func to each time value to get list of func[t;]*)
  Return[xlist - Fxlist // Abs // Mean];
]

```

• Calculate the mean global error for various value of h for fixed t_j and t_f

```

f[t_, x_] = -2 x;
ff[t_] = e-t2/2;
Table[{5.0/10n, err[euler[f, 1, 0, 5, 10n], ff]}, {n, 1, 4}]

```

Out[] = {{0.5, 0.0746396}, {0.05, 0.0639748}, {0.005, 0.0631697}, {0.0005, 0.0630899}}

So, let us say the function that we are trying , the function for which the solution that we are looking for is, let us go ahead and do what we have been doing so far. Let us start out with $-2x$

So, let us say our differential equation that is $\dot{x}=f(x)$, so this is little f(x,t) which is what I have defined over here as $-2x$. This ff is the function that I am going to pass to the error function, this is the solution of the differential equation and in this case, if $\dot{x}=-2x$ in the solution is e^{-2t} .

So let us go ahead and do that first and so this is my derivative \dot{x} , this is the solution, x is a function of t and here is the table that I have constructed. The two elements of this table, this is $t_f - t_i$ and I am talking t_f as 5 and t_i as 0.

So, $t_f - t_i$ is 5, $5/10^n$ which is N_{max} , to the error function I am passing 2 arguments, the Euler and the solution f , this is the Euler list, Euler function is going to return me data list, which is the solution for \dot{x} equal to this f . From $x(t_i) = 0$, $t_i = 5$. For $x_0 = 1$ and $N_{max} = 10^n$. So, this is going to, you know the first argument is going to return me data list.

That, the data list is passed to error function and then error function is also passed the solution, hence going to find the difference in the mean absolute error. I am dividing that mean absolute error that I am obtaining from the numerator over here by h . Let me, for now I am going ahead and remove that.

(Refer Slide Time: 09:27)

```
err[dataset_, func_] := Module[{tlist, xlist, Fxlist},
  tlist = dataset[;;, 1]; (*Extract each time value*)
  xlist = dataset[;;, 2]; (*Extract each x value*)
  Fxlist = func /@ tlist;
  (*Apply func to each time value to get list of func[t_i]*)
  Return[xlist - Fxlist // Abs // Mean];
]

```

• Calculate the mean global error for various value of h for fixed t_j and t_j

```
f[t_, x_] = -2 x;
ff[t_] = e-2t;
Table[{ $\frac{5.0}{10^n}$ , err[euler[f, 1, 0, 5, 10n], ff]}, {n, 1, 4}]
```

Out[248]= {{0.5, 0.0746396}, {0.05, 0.0639748}, {0.005, 0.0631697}, {0.0005, 0.0630899}}

```
In[249]:= err[euler[f, 1, 0, 5, 50], ff]
```

Out[249]= 0.00648814

• We see that the global error scales at h, thus the error is considered as O(h).

```
xlist = dataset[;;, 2]; (*Extract each x value*)
Fxlist = func /@ tlist;
(*Apply func to each time value to get list of func[t_i]*)
Return[xlist - Fxlist // Abs // Mean];
]

```

• Calculate the mean global error for various value of h for fixed t_j and t_j

```
In[248]=
f[t_, x_] = -2 x;
ff[t_] = e-2t;
Table[{ $\frac{5.0}{10^n}$ , err[euler[f, 1, 0, 5, 10n], ff]}, {n, 1, 4}]
```

Out[250]= {{0.5, err[{{0, 1}, {0.5, 0.}, {1., 0.}, {1.5, 0.}, {2., 0.}, {2.5, 0.}, {3., 0.}, {3.5, 0.}, {4., 0.}, {4.5, 0.}, {5., 0.}, {5.5, 0.}, {6., 0.}, {6.5, 0.}, {7., 0.}, {7.5, 0.}, {8., 0.}, {8.5, 0.}, {9., 0.}, {9.5, 0.}], {0.0005, err[{{0, 1}, {0.5, 0.}, {1., 0.}, {1.5, 0.}, {2., 0.}, {2.5, 0.}, {3., 0.}, {3.5, 0.}, {4., 0.}, {4.5, 0.}, {5., 0.}, {5.5, 0.}, {6., 0.}, {6.5, 0.}, {7., 0.}, {7.5, 0.}, {8., 0.}, {8.5, 0.}, {9., 0.}, {9.5, 0.}], ff}}

So we will just take the error, mean absolute error and we are going to execute it for n equal to 1 to 4. So, you are going to get 4 values, n equal to 1 to 4. So, n will be 10 that means N_{max} is 10 and when n is 4, N_{max} is 10000. So, for 10 steps, 100 steps, 1000 steps and 10000 steps we are going to execute this, so this is going to give me, oh I think I forgot to define error function as we go ahead and define that.

Okay, now I will do it again and this is going to give me the value of h. So, each of these pairs is giving me a value of h that I am using over here and the corresponding error and as

we see that as h decreases proportionately error also decreases, the mean absolute error also decreases. So, this is the global error that we are looking at.

(Refer Slide Time: 10:28)

```
tlist = dataset[;;, 1]; (*Extract each time value*)
xlist = dataset[;;, 2]; (*Extract each x value*)
Fylist = func /@ tlist;
(*Apply func to each time value to get list of func[t;]*)
Return[xlist - Fylist // Abs // Mean];
]
• Calculate the mean global error for various value of h for fixed ti and tf
In[255]:=
f[t_, x_] = -2 x;
ff[t_] = e-2t;
Table[{ $\frac{5.0}{10^n}$ ,  $\frac{\text{err}[\text{euler}[f, 1, 0, 5, 10^n], ff]}{10^n}$ }, {n, 1, 4}]
Out[257]=
{{0.5, 0.105809}, {0.05, 0.100622}, {0.005, 0.100018}, {0.0005, 0.09999569}}
In[ ]:= err[euler[f, 1, 0, 5, 50], ff]
Out[ ]:= 0.00648814
• We see that the global error scales at h, thus the error is considered as O(h).
```

And if I divide this global error by the value of h, which is this we copy that and paste it over here and when I execute that you see, you are going to get approximately a constant. So, for $h = 0.5$ we are giving mean absolute error by h as 0.1 or 0.05 also, mean absolute error by h as 0.1 and so on even for h equal to 0.0005, I am getting the same thing.

(Refer Slide Time: 10:57)

```

tlist = dataset[;;, 1]; (*Extract each time value*)
xlist = dataset[;;, 2]; (*Extract each x value*)
Fylist = func /@ tlist;
(*Apply func to each time value to get list of func[t;]*)
Return[xlist - Fylist // Abs // Mean];
]

```

• Calculate the mean global error for various value of h for fixed t_f and t_i

```

In[258]:=
f[t_, x_] = -2 x;
ff[t_] = e^-2t;
Table[{10.0/10^n, err[euler[f, 1, 0, 10, 10^n], ff]}, {n, 1, 4}]

```

```

Out[260]=
{{1., 0.919928}, {0.1, 0.051154}, {0.01, 0.0501165}, {0.001, 0.0500116}}

```

```

In[ ]:= err[euler[f, 1, 0, 5, 50], ff]
Out[ ]:= 0.00648814

```

• We see that the global error scales at h, thus the error is considered as O(h).

$\dot{x} = f(x)$
 $\Delta x_{n+1} = f(x_n)h + x_n$
 accuracy of h^2 local error $\sim O(h^2)$
 Steps $\sim N_{max}$
 $N_{max} \times h^2 \sim (N_{max} h) h$
 $\sim O(h)$
 Global error

You can go ahead and change for example I can change t_f to 10 if I change t_i to 10. This is $t_f - t_i$, so this also I should change to 10 and this also I should change to 10. And I can execute this again you will see that the first one, the h was too large, h was 1 because 10 over 10 is 1. So, that is something that is an outlier, so we leave this one out but apart from that, for small values of h when h is 0.1, 0.01 or 0.001 we see that mean absolute error divided by h is a constant. This is a numerical check of the fact that we laid out.

That, global error in Euler method scale-like order h and that is what we are seeing over here. We will go ahead and try out something else.

(Refer Slide Time: 11:56)

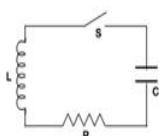
```
xlist = dataset[;;, 2]; (*Extract each x value*)
Fylist = func /@ tlist;
(*Apply func to each time value to get list of func[t;]*)
Return[xlist - Fylist // Abs // Mean];
]
• Calculate the mean global error for various value of h for fixed t0 and tf
In[261]:=
f[t_, x_] = -x t;
ff[t_] = e-t2/2;
Table[{10.0/10n, err[euler[f, 1, 0, 10, 10n], ff]}, {n, 1, 4}]
Out[263]=
{{1., 0.0491139}, {0.1, 0.0327624}, {0.01, 0.031662}, {0.001, 0.0316228}}
In[ ]:= err[euler[f, 1, 0, 5, 50], ff]
Out[ ]:= 0.00648814
• We see that the global error scales at h, thus the error is considered as O(h).
```

Let us go ahead and try out $\dot{x} = -xt$. The solution of that is Gaussian, so solution $e^{-t^2/2}$, you can go ahead and check that out and we are going to go ahead and execute this again and we see that again that pretty much for all values of h that we have taken over here, mean absolute error divided by h is a constant.

Therefore, the error is of the order of h or the global error scales like h. Or else you are welcome to go ahead and play with this little but more for various different functions that you can try out for various different \dot{x} s, that you can solve and find the analytical solution, feed that n, run this a few times to find out how in your examples that you take, how does the mean absolute error scale.

(Refer Slide Time: 12:53)

Reducing Higher Order ODE to a set of first order ODEs



• The equations we are usually interested in solving are equation of motion, which are usually second order differential equation, such as

$$\frac{d^2 Q}{dt^2} = -\frac{Q}{LC} - \frac{R}{L} \frac{dQ}{dt} \quad (3)$$

• associated with a suitable initial value condition like

$$\begin{aligned} Q(0) &= Q_0 \\ \dot{Q}(0) &= 0 \end{aligned} \quad (4)$$

• In order to apply a suitable numerical method, such as Euler's method we will turn this IVP into a set of coupled first order ODEs:

$$\begin{aligned} \frac{dQ}{dt} &= I \\ \frac{dI}{dt} &= -\frac{Q}{LC} - \frac{R}{L} I \end{aligned} \quad (5)$$

• Now this is set of two coupled ODEs of order 1, in terms of dynamical quantities Q and I , with initial condition:

$$\begin{aligned} Q(0) &= Q_0 \\ I(0) &= 0 \end{aligned} \quad (6)$$

• We need to solve them simultaneously using Euler's method.

Generalization

Alright, let us go ahead and work on how to reduce higher-order ODEs to a set of first-order ODEs because what we have learned in the RS method is that if you give me a first-order ODE, I can use Euler's method to solve that first order ODE. However, most of the time in physics we do not come across the first order of ODEs, we come across second order ordinary differential equations.

And, if you want to use Euler's method to solve the second-order of ODEs, then we need to reduce those second order ODEs, the first order ODEs. So that we can apply Euler's method.

In order to understand that let us go back and go to the familiar example of an LCR circuit and obtain a second order ODE.

If you go back and review the LCR circuit example, this is the differential equation we

obtained. $\frac{d^2Q}{dt^2} = -\frac{Q}{LC} - \frac{R}{L} \frac{dQ}{dt}$ For this differential equation, the initial conditions were

$Q(0) = Q_0 = 0$. Now, in order to apply Euler's method, this is the second-order differential equation, we are going to reduce it to the first-order differential equation by defining

auxiliary quantity, in this case, we will call it $I = \frac{dQ}{dt}$ which is also physical for I is the current.

$I = \frac{dQ}{dt}$, so we will use $I = \frac{dQ}{dt}$ and will replace $\frac{d^2Q}{dt^2}$ with $\frac{dI}{dt}$. So, therefore now we have

got two equations, $I = \frac{dQ}{dt}$ and $\frac{dI}{dt} = -\frac{Q}{LC} - \frac{R}{L} I$. So, this set of two equations is equivalent

to this second-order equation. Now, the interesting thing is that this set of the equation, these two equations they are both first-order in derivatives.

This equation is first order in time, it has only $\frac{dQ}{dt}$, this equation again is first order in time because it has got $\frac{dI}{dt}$, I itself is $\frac{dQ}{dt}$ but we are not considering I as $\frac{dQ}{dt}$ we are talking I as a independent variable, Q as an independent variable and we have got two coupled ordinary differential equations of the first order and the initial conditions also modify accordingly.

$Q(0)$ becomes Q_0 , $I(0)$ becomes 0, rather than writing $\dot{Q}=0$, $\dot{Q}=I$ so we say $I(t=0)=0$. So, we modify both the initial conditions and the set of differential equations reducing them down to the first-order ODEs. Now, I can go ahead and use Euler's method because I know that now in order to solve this equation with Euler's method, I can calculate dQ in steps by finding out $I * dt$.

Previous $I * dt$, I want to find I, all I have to do is use this equation which says $\frac{dI}{dt}$ is this right-hand side, so I calculate this right-hand side at previous time and multiply by dt and that will give me dI. So, I can calculate subsequently dQ steps and dI steps and keep on

incrementing them and by doing so I can implement the Euler's method to find out both I as a function time and Q as a function of time.

Now, that you have got a gist of it, you can go ahead and generalize it and here is how the generalization works.

(Refer Slide Time: 16:12)

The slide content is as follows:

- associated with a suitable initial value condition like $Q(0) = Q_0$ (4)
- In order to apply a suitable numerical method, such as Euler's method we will turn this IVP into a set of coupled first order ODEs:

$$\frac{dQ}{dt} = I$$

$$\frac{dI}{dt} = -\frac{Q}{LC} - \frac{R}{L}I$$
 (5)
- Now this is set of two coupled ODEs of order 1, in terms of dynamical quantities Q and I , with initial condition:

$$Q(0) = Q_0$$

$$I(0) = 0$$
 (6)
- We need to solve them simultaneously using Euler's method.
- Generalization**
- Any n^{th} order ODE can be converted to coupled n first order ODE. Let's say ODE has the form:

$$\text{ODE: } \frac{d^n x}{dt^n} = f(t, x, \dot{x}, \ddot{x}, \dots, x^{(n-1)})$$
 (7)
- then, we can define

$$y = x$$

$$z = \dot{x}$$

$$\vdots$$

$$w = \dot{z}$$

$$w = f(t, x, y, z, \dots, w)$$
 (8)
- which is a set of coupled ODEs

Let us consider a general n^{th} order ODE, for example, n^{th} order ODE $d^n x/dt^n$ is a function of f and is given by f which is a function of time x , \dot{x} , \ddot{x} and so on, $x^{(n-1)}$ means that $n - 1$ is derivative of x with respect to time. So, in general my n^{th} derivative can be written as a function of all the lower derivatives and the time.

Therefore, most generally I am to deal with such a differential equation and I am talking about ordinary differential equations and such a equation I can reduce to a set of ODEs by using the following trick for every quantity such as \dot{x} . Starting with \dot{x} , I will define \dot{x} as y and then I can take \ddot{x} as $d/dt \dot{x}$ which is y .

So, \dot{y} becomes z and then so on I can write as \dot{z} equal to w , or rather w is \dot{z} , should be other way around, $w = \dot{z}$ where w is the $n - 1$ derivative of s and then, my w which is the n^{th} derivative of x becomes a function of t x y z as w , etc. which is where I take t x y z after w , all of them as independent variables and I simultaneously, evolve them using this set of differential equations where this is the equation of motion, that was given to you.

And, these are the auxiliary equations that you wrote down in order to convert this equation into a set of the first-order ODE. Now, notice that all of these equations are first-order differential equations. So, now if we can go ahead and implement Euler's method on that.

(Refer Slide Time: 18:26)

Define and Translate LCR circuit eqn for Numerical Evaluation

- Remember the four steps of Computational Thinking: **Define, Translate, Compute and Interpret**
- In this problem we will define/identify and translate the problem into a form that it can be solved on the computer
- First step is to non-dimensionalize the equation of motion for LCR circuit, thus reducing the number of parameters in the problem
- Second step is to write the second order ODE into a set of first order ODEs (as shown in the previous section)

Problem: For the IVP given by equation below

$$\frac{d^2 Q}{dt^2} = -\frac{Q}{LC} - \frac{R}{L} \frac{dQ}{dt} \quad (9)$$

$$Q(0) = Q_0$$

$$Q'(0) = 0$$

(a) Non-dimensionalize the equation by choosing suitable scales for Q, I and t , expressing the equation in dimensionless quantities.

(b) How many free parameters are left in the equation after non-dimensionalization?

(c) Write the non-dimensional equation as a set of first order ODEs.

(d) Express the solution of this equation in terms of your dimensionless quantities:

$$Q(t) = Q_0 e^{-\frac{R}{2L}t} \cos\left(\sqrt{\frac{1}{LC} - \frac{R^2}{4L^2}}t\right) \quad (10)$$

Solution

Q scale: Q_0 (1)

t scale: L/R

I scale: $Q_0 R/L$

Slide 4 of 4 Make the transformation:

(d) Express the solution of this equation in terms of your dimensionless quantities:

$$Q(t) = Q_0 e^{-\frac{R}{2L}t} \cos\left(\sqrt{\frac{1}{LC} - \frac{R^2}{4L^2}}t\right) \quad (10)$$

Solution

Q scale: Q_0

t scale: L/R

I scale: $Q_0 R/L$

Making the transformation:

$$Q \rightarrow Q_0 Q$$

$$t \rightarrow \frac{L}{R} \tau$$

$$I \rightarrow \frac{Q_0 R}{L} I$$

we get

$$\frac{Q_0}{L/R} \frac{d^2 Q}{d\tau^2} = -\frac{Q_0 Q}{LC} - \frac{R}{L} \frac{Q_0}{L/R} \frac{dQ}{d\tau} \quad (11)$$

$$= \frac{d^2 Q}{d\tau^2} = -\frac{L}{R^2 C} Q - \frac{dQ}{d\tau} \quad (12)$$

In the equation above, Q and I are dimensionless. Therefore, $I = dQ/dt$ is also dimensionless. After non-dimensionalization, there is only one competing time scales in the problem.

Slide 4 of 4

So, let us go ahead and prepare this LCR circuit for numerical evaluation, there is one more step that we need to do before we can actually apply Euler's method. So, remember the 4 steps of our computational thinking approach, define, translate, compute and interpret. What we are going to do now is we will work on the first two steps, define and translate.

Then, compute and interpret something we will do on the computer using Euler's method and define and translate, we need to do on pen and paper. In order to translate our LCR circuit problem on to the computer.

So, remember define and translate we have to break into pieces and translate, we have to make the equations ready for evaluation on the computer or turn them from physics to a problem of mathematics which we can set up on the computer. So, we are going to do this, so this is the set of equations we have, first we are going to do is non-dimensionalize this equation by choosing suitable scales for Q , I and t .

We already know how to reduce them to the first-order ODEs. But, first, we are going to do is non-dimensionalize them, without non-dimensionalization we cannot turn them into a problem on computers, because computers only know numbers they do not understand physical dimensions. Then, we are going to check how many free parameters are left in the equation after equation after non-dimensionalization that is how many free quantities are there on whichever solution can depend.

Then, we are going to write the non-dimensionalize set of equations as the first set of ODEs. We just did this set of equations is first order ODEs. Now, once we have non-dimensionalize we will take the non-dimensionalize equations and write them as the first set of ODEs because those are the ones we are going to put on the computer. And, finally, you know the analytical solution, this is the analytical solution.

We are going to take this analytical solution and also known non-dimensionalize this one because that is what we are going to go and compare on the computer. So, let us go ahead and get started with that, so non-dimensionalization, the always the easiest way of doing that is find out the scales in the problem for various physical quantities, in this case, physical quantities are Q , I and t .

So, for Q the natural scale present is Q_0 , for t there two choices of scales either I can take \sqrt{LC} as a unit of time or I can take L/R as a unit of time. So, I will make a choice here, any of these choices are fine, you can go ahead and try the other choice. For the sake of this example, I will take L/R as the choice and now I need a scale, for current, I will take current is Q_0/t .

I have already chosen scale for Q_0 and scale for t , so therefore a natural choice for scale for current becomes $Q_0 R/L$. So, let me go ahead and substitute this in this set of equations and

when I do that rather than introducing new variables which are dimensionless, it is always easy to make the following replacement. Replace Q by $Q_0 Q$, where this Q was dimension for Q and after replacement, this Q becomes dimensionless Q and Q_0 contains all the dimensions.

The same thing for the time, the time here is dimensionless time, L/R contains the dimensions of time, this I is dimensionless I and $Q_0 R/L$ contains the dimensions of current. So, this Q , t and I are before the replacement that is in this equation, they are dimension full quantities. Q , t and I on the right-hand side of the arrows are dimensionless Q , t and I . So, I am going to go substitute Q , t and I with the right-hand side values in this equation.

When I do that I get this over here, I have pulled out dimensions of Q and t and as a consequence I am getting this factor out here, again in my differential equation I had got Q/LC . So for Q I am writing $Q_0 Q$ and that pulls me at the dimension of Q_0 out. Again, pulling out dimensions in dQ/dt dimensions of Q out of this and dimensions of dt out of this.

I get a factor of $Q_0 L/R$, R/L is a constant. LC is a constant, will leave them as it is and now if you simply find you will get this equation where most of the dimension full quantities will cancel out and I am only going to get one ratio $L/R^2 C$. This Q is dimensionless, this t is dimensionless, so $d^2 Q/dt^2$ is dimensionless. This dQ/dt is dimensionless because both Q and t are dimensionless.

This Q is dimensionless, therefore $L/R^2 C$ is also dimensionless and this is a dimensionless ratio. This is the only parameter in my problem that is left after the translation, so we are done, define and translate, these two steps. After translate, this is my maths problem, it is not complete ready for implementation for Euler's method. But, as far as the maths problem is concerned I must solve this math problem for a single parameter $L/R^2 C$.

So, every time $L/R^2 C$ changes, the solution changes, math solution changes. Taking that math solution to a physics solution after that only means I am setting the scale for physical quantities such as Q , t and I , which is determined by, $Q_0 L/R$ etc. But, as far as the problem

is concerned, the solution only depends on the ratio L/R^2C or the nature of the solution will depend on the ratio L/R^2C .

(Refer Slide Time: 23:57)

Making the transformation:

$$\begin{aligned} Q &\rightarrow Q_0 Q \\ t &\rightarrow \frac{L}{R} \tau \\ i &\rightarrow \frac{Q_0 R}{L} I \end{aligned} \quad (12)$$

we get

$$\begin{aligned} \frac{Q_0}{L/R^2} \frac{d^2 Q}{d\tau^2} &= \frac{Q_0 Q}{LC} - \frac{R}{L} \frac{Q_0}{L/R} \frac{dQ}{d\tau} \\ \Rightarrow \frac{d^2 Q}{d\tau^2} &= \frac{L}{R^2 C} Q - I \end{aligned} \quad (13)$$

In the equation above, Q and τ are dimensionless. Therefore, $I = dQ/d\tau$ is also dimensionless. After non-dimensionalization, there is only one free parameter $L/(R^2 C)$ which is the ratio of two competing time scales in the problem.

Writing as first order ODE we get

$$\begin{aligned} \frac{dQ}{d\tau} &= I \\ \frac{dI}{d\tau} &= -\frac{L}{R^2 C} Q - I \\ Q(0) &= 1 \\ I(0) &= 0 \end{aligned} \quad (14)$$

The solution, in terms of dimensionless Q and τ can be written as:

Very good, now we are going to play the same thing, we are going to reduce this equation to our first-order ODE, a set of the first-order ODE. So will define dQ/dt , the dimensionless charge divide by the rate of dimensional charge with dt , the derivative of dimensionless charge is with respect to time. As dimensionless current and $\frac{dI}{dt}$ as $\frac{-L}{R^2 C} Q - I$, where this dQ/dt was replaced by I .

And the initial conditions $Q_0(0)=1$, again Q is dimensionless, so $Q_0(0)=1$ and $I_0(0)=0$. So, now this is the math problem that is ready for me to be put on a computer. This is set of the first order of ODEs without any dimensions and the only parameter is $L/R^2 C$.

(Refer Slide Time: 24:52)

(a) Non-dimensionalize the equation by choosing suitable scales for Q , t and i , expressing the equation in dimensionless quantities.

(b) How many free parameters are left in the equation after non-dimensionalization?

(c) Write the non-dimensionalized equation as a set of first order ODEs.

(d) Express the solution of this equation in terms of your dimensionless quantities:

$$Q(0) = Q_0 e^{-\frac{R}{L} t} \cos \left(\sqrt{\frac{1}{LC} - \frac{R^2}{4L^2}} t \right) \quad (10)$$

Solution

Q scale: Q_0
 t scale: L/R
 i scale: $Q_0 R/L$ (11)

Making the transformation:

$$\begin{aligned} Q &\rightarrow Q_0 Q \\ t &\rightarrow \frac{L}{R} t \\ i &\rightarrow \frac{Q_0 R}{L} i \end{aligned} \quad (12)$$

we get

$$\frac{Q_0}{L/R^2} \frac{d^2 Q}{dt^2} = \frac{Q_0 Q}{LC} - \frac{R}{L} \frac{dQ}{dt} \quad (13)$$

$$\frac{d^2 Q}{dt^2} = \frac{L}{R^2 C} Q - \frac{dQ}{dt}$$

In the equation above, Q and t are dimensionless. Therefore, $t = dQ/dt$ is also dimensionless. After non-dimensionalization, there is only one free parameter $L/(R^2 C)$ which is the ratio of two competing time scales in the problem.

Writing as first order ODE we get

$$\begin{aligned} \frac{dQ}{dt} &= I \\ \frac{dI}{dt} &= -\frac{L}{R^2 C} Q - I \\ Q(0) &= 1 \\ I(0) &= 0 \end{aligned} \quad (14)$$

The solution, in terms of dimensionless Q and t can be written as:

$$Q(t) = e^{-t} \cos \left(\sqrt{\frac{L}{R^2 C} - \frac{1}{4}} t \right) \quad (15)$$

As expected, solution also depends only on one parameter: $L/(R^2 C)$ which should be greater than $\frac{1}{4}$ for the solution to be valid.

Now, if I look at the solution, the solution, the analytical solution that was given to me for this, which we worked out a couple of lectures ago. I again non-dimensionalize this when I non-dimensionalize this my solution reduces down to this equation, where this is dimensionless charge, this is simply an $e^{-t/2}$ and a cosine function of this argument.

Where, I have got the constant $\sqrt{L/R^2 C - 1/4}$. It is constant multiplying dimensionless time and this constant, inside this square root is also dimensionless. Now, my solution depends on

the ratio L/R^2C . As I saw it in the differential equation, the differential equation only dependent on the ratio L/R^2C

So does the solution already dependent on that and I can see that this solution is only valid as long as $L/R^2C > 1/4$. So, that is our domain of validity of the solution and this is the domain which will work and we will take this parameter L/R^2C as something. We can call it α and as long as $\alpha > 1/4$ we are going to have this solution as a valid solution. Alright, let us go ahead and work on how to implement it on the computer.

(Refer Slide Time: 26:08)

Solving Coupled ODEs using Euler's Method

- Lets take the following set of coupled ODEs, which is most general case for ODEs, as higher order ODEs can always be brought in similar form.

$$\begin{aligned} \dot{x} &= f(t, x, y, z) \\ \dot{y} &= g(t, x, y, z) \\ \dot{z} &= h(t, x, y, z) \end{aligned} \quad (16)$$
- Define a column vectors X and F as

$$X = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad F = \begin{pmatrix} f \\ g \\ h \end{pmatrix} \quad (17)$$
- Then the coupled ODEs can be written as

$$\dot{X} = F \quad (18)$$
- It is straightforward to generalize this to arbitrary dimensions as long as there is only one dynamical variable t .
- Euler's method is given by

$$\begin{aligned} X_0 &= X_{\text{initial}} \\ X_{n+1} &= X_n + hF(X_n) \end{aligned} \quad (19)$$
- Here is its implementation. Notice the use of **Through** function


```
Through({f, g, h}[x])
{f[x], g[x], h[x]}

Through({f, g, h}@x)
{f[x], g[x], h[x]}
```

So, before we solve that problem, let us understand the general framework of putting this on a computer. So, now in general I have got a set of coupled ODEs and in general those set of coupled ODEs can look like this.

Let us take work in 3 dimensions, \dot{x} is some function of t, x, y and z , \dot{y} is some function g t, x, y , and z , \dot{z} is some function of t, x, y , and z . We can express these equations better in the form of matrix equations by doing the following, we can define big X as a column vector of t, x, y and z and big F as the set of functions f, g and h and I have also appended or prepended this with 1.

So, f is 1, f , g and h now you see that my differential equation that is this set of equations over here, this set of coupled first-order ODEs simply can be written as a single matrix equation and this kind of setting up, work out really easily on a system like Mathematica which can directly work with matrices and lot of operations on matrices can easily be done. That is why we are setting this up in the form of a matrix equation.

You see that the first equation in the $\dot{x}=f$, $\dot{x}=f$ is a matrix equation, that is sort of 4 equations because x has 4 dimensions t , x , y , and z . The first equation is kind of a trivial one which is $dt/dt=1$, so the first equation is saying that time derivative of time with respect to time is simply 1.

So, that is the trivial equation and $\dot{x}=f$, $\dot{y}=g$ and $\dot{z}=h$ is the set of these three equations. Now, the initial condition also can be written in terms of x_0 , that is at $t=t_0$, I have got $x=x_0$, y_0 and z_0 . So, X_0 becomes my initial condition with $X_0=\{t_0, x_0, y_0, z_0\}$, set of 4 quantities at time $t=0$.

So, we are simply going to write X_0 as x initial which is the initial value of time x , y and z , the combination of all four of them. And, Euler's method simply becomes evaluating $X_{n+1}=X_n+hF(X_n)$.

So, this is a matrix equation again, X_{n+1} is a four tuple, X_n is a four tuple, h is a scalar number, the step size and $F(X_n)$ is a four tuple because F is a function which is a column vector of four functions where the first one is the identity function, so $F(X_n)$ is going to give me a vector or a four tuple, column vector of four items.

So, this is how I can go ahead and implement that, now you can see that the Euler's method code is pretty much going to be same, I am going to make minor modifications. The only minor modifications I have to make is to take care of doing this for all the functions at the same time, that is do it for the matrix as a single number. So, let us go ahead and see how to do that, for that I am going to use Through function.

(Refer Slide Time: 29:35)

```

• Here is its implementation. Notice the use of Through function
In[265]:= Through[{f, g, h}[x]]
Out[265]= {f[x], g[x], h[x]}

In[266]:= Through[{f, g, h}@x]
Out[266]= {f[x], g[x], h[x]}

In[267]:= Through[{f, g, h}@@{t, x, y, z}]
Out[267]= {f[t, x, y, z], g[t, x, y, z], h[t, x, y, z]}

eulerGen[Func_, X0_, tf_, nMax_] := Module[{h, datalist, prev, n},
  h = (tf - X0[[1]]) / nMax // N;
  For[datalist = {X0},
    Length[datalist] < nMax,

```

$$X = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad F = \begin{pmatrix} f \\ g \\ h \end{pmatrix} \quad (17)$$

• Then the coupled ODEs can be written as

$$\dot{X} = F \quad (18)$$

• It is straightforward to generalize this to arbitrary dimensions as long as their is only one dynamical variable t .

• Euler's method is given by

$$\begin{aligned} X_0 &= X_{\text{initial}} \\ X_{n+1} &= X_n + h F(X_n) \end{aligned} \quad (19)$$

• Here is its implementation. Notice the use of Through function

```

In[264]:= {f, g, h}[x]
Out[264]= {f, g, h}[x]

Through[{f, g, h}@x]
{f[x], g[x], h[x]}

Through[{f, g, h}@@{x, y}]
{f1[x, y], g[x, y], h[x, y]}

```

What Through function does is a notice from this example, when I take some function f, g, and h and apply it on x, that is without the Through function, lets do it is without the Through function and if I do this, you are simply going to get that back, but when I apply Through function on this, as I enclose this whole thing Through function. You see I get an array of f applied on x, g apparent, x and h apparent and that is exactly what I want.

I can also implement this through an @ operator or prefix application. So, f, g, h is set of functions that I am applying on x. When I do use a Through, I get f, g and h applied to all three of them and in return, I get a list of three functions and if my set of arguments were a

vector, as they happened to be in this case I can apply on a set of vector x, y or for that matter t, x, y and z which is my column vector.

I can take it and apply on that, in return I am going to get an array of functions $\{f[t, x, y, z], g[t, x, y, z], h[t, x, y, z]\}$.

(Refer Slide Time: 30:57)

```

{f[t, x, y, z], g[t, x, y, z], h[t, x, y, z]}
In[268]:= Through[{f, g, h}@{t, x, y, z}]
Out[268]:= {f[{t, x, y, z}], g[{t, x, y, z}], h[{t, x, y, z}]}

eulerGen[Func_, X0_, tf_, nMax_] := Module[{h, datalist, prev, next, rate},
  h = (tf - X0[[1]]) / nMax // N;
  For[datalist = {X0},
    Length[datalist] < nMax,
    AppendTo[datalist, next],
    prev = Last[datalist];
    rate = Through[Func @@ prev];
    next = prev + h rate;
  ];
  Return[datalist];
]

```

So, we can go ahead and use this through the construct, there are two ways of doing this, either I can use the @@ operator or alternately, I can also use the @ operator in the following sense. If I use the single @ the operator, I am going to get $\{f[t, x, y, z], g[t, x, y, z], h[t, x, y, z]\}$ and so on.

I can use either of these constructs and it depends on how you define your functions f, so let us work with the @@ example and you can modify the Euler general function that I am going to give you for this example. So, let us go ahead and work, redefine the Euler function, for the more general case of multiple dimensions. So, this case 3 dimensions x, y, and z add the time dimensions to it, it becomes 4 dimensions.

Because Euler general is going to be for 3 plus 1 dimension, one is a time dimension and everything else depends on time, so 3 other dimensions. Again, for this Euler general

function I have got arguments, the set of functions f, g, h etc. The initial point X_0 , this X_0 is going to contain also the initial point time t_0 . So, this contains t_0 , x_0 , y_0 and z_0 .

All the 4 items in it and the final time t_f up to which I want to evolve and N_{max} , N_{max} is the number of steps that I want to keep. So, you this, some modification over here t_0 is been absorbed into X_0 , as part of the initial condition X_0 . This is a set of functions, not a single function and I will show you, how we are going to implement that. Again, I have got a bunch of local variables in the module construct.

In order to make things little bit more readable, I have defined previous, next and rate. So, let us go ahead and understand this code. The first thing I am going to do is, I am going to calculate h, h is $t_f - t_i$, t_i is a first element of X_0 , that has been resolved over here, this is the first element of X_0 . So, $h = (t_f - X_0[1]) / N_{max}$, then I want to start my For loop and this is the part of my for loop.

In the for loop, again I have got the initialization, the condition, the increment and the body. So, the last 3 statements over here constitute the body and in the increment I am appending to the data list, the next element. The next element is calculated in the body, so let us go ahead and understand that how we are doing this, so first in the initialization X_0 is the set of 4 items t_0 , x_0 , y_0 and z_0 .

We are adding all of that into column vector, X_0 and that X_0 has been added to data list, this is initialisation of data list, ofcourse the condition is same as before, length of data list is less than N_{max} and increment, first we are going to execute the body and in the body, we are going to calculate the next and in the increment of the for loop we are going to append that next value to data list.

In order to calculate next, I need to first read previous, previous is read from the data list. So, I take previous, there is last element of data list. In the first run of the for loop, that will be simply X_0 because data list only contains one element X_0 . Then, I calculate the rate, the

rate is given by application of the function on the previous. So, each function that means, should be of the form, it should have 4 arguments.

Each function should be of the form it can take 4 arguments t, x, y and z or whatever the number of variables we have. The way this code is written it does not care about how many variables you have got, as long as the size of X_0 and the function is same, this will be fine. Because size of previous is same as the size of X_0 and function requires the number of argument that the function require should be the size of X_0 or the size of previous.

Once you, so this is going to pass each of the functions, the arguments through is going down make sure that each of the functions in my array are passed on the arguments in the previous and the rate is calculated. So, that means I calculate X_0 from here or \dot{X} from here and next is calculated by adding previous * h + h * the rate or h * \dot{X} .

So, this is instantaneous \dot{X} , at the previous point multiplied by h that gives me the step size. Adding that to previous gives me, the value of the next step, I execute this for loop, when the for loop ends I return the data list. Let us go ahead and run this function. Now that this definition has been made, let us go ahead and implement this for the problem that we are looking forth.

(Refer Slide Time: 36:10)

Application to 2nd Order ODEs: LCR circuit

• We want to solve the IVP:

$$\frac{d^2Q}{dt^2} + \frac{L}{R^2C} \frac{dQ}{dt} - Q = I$$

$$\frac{dI}{dt} = -\frac{L}{R^2C} Q - I$$

$$Q(0) = 1$$

$$I(0) = 0$$

$$X = \begin{pmatrix} Q \\ I \end{pmatrix} \quad \dot{X} = \begin{pmatrix} I \\ -\frac{L}{R^2C} Q - I \end{pmatrix}$$

• Implementation: Lets take the ratio $w = L / (R^2 C)$

```

w = 10;
beta = sqrt(w - 1/4);
2.0*pi
beta
id[t_, charge_, current_] = 1;
chargeDot[t_, charge_, current_] = current;
currentDot[t_, charge_, current_] = -w charge - current;

```

$$X = \begin{pmatrix} Q \\ I \end{pmatrix} \quad X = \begin{pmatrix} Q \\ -\frac{L}{R^2 C} Q - I \end{pmatrix}$$

• Implementation: Lets take the ratio $w = L/(R^2 C)$

```

In[270]:=
w = 10;
beta =  $\sqrt{w - \frac{1}{4}}$ ;
 $\frac{2.0 \pi}{beta}$ 
id[t_, charge_, current_] = 1;
chargeDot[t_, charge_, current_] = current;
currentDot[t_, charge_, current_] = -w charge - current;
initial = {0, 1, 0};

Out[272]=
2.01223

```

data = eulerGen[{id, chargeDot, currentDot}, initial, 10, 100]

The problem we were looking for is this set of equations for LCR circuit, the damped LCR circuit, damping is because of the resistor and oscillation is because of LC. So, my equation is $dQ/dt = I$, $dI/dt = -L/R^2 C$, $Q(0) = 1$ and $I(0) = 0$. It is better to actually write this equation as, so this is my $Q(0)$, this is the function of only I and this side, I is a function of Q and I, these are the only two variables.

So, you can think of Q as x and I as y, so this dx/dt and this is dy/dt and we have got $x(0)$ and $y(0)$. So, therefore in order to implement this what I am really doing is I am writing my $X = Q$, Q as a two tuple of Q and I and ofcourse I have to include time also.

So, I can go ahead and include that, so t, Q and I is my x and I am writing \dot{x} as, again this is one, this is simply I because $\dot{Q} = I$ and I is this, so this is my differential equation, written in the matrix form. This is my differential equation in the matrix form I want to solve this differential equation and in order to do that I am going to take this ratio $L/R^2 C$ as w.

So this is what I have defined as w and just for reference I have also defined β because $\beta = \sqrt{w - 1/4}$, this the part that goes into the analytical solution. So, I am going to take w as 10 which is much bigger than 1 by 4 so to start with. This is $2\pi/\beta$, β is the oscillation frequency in my analytical solutions of $2\pi/\beta$, will give me the time period.

Now, I want to define these 3 functions, this is my first function, this is my \dot{Q} and this is my \dot{I} , the third argument over here is I . So, in order to define that I am going to call the first one as identity, this one as identity, so this is defined over here. This is identity and this is simply one, but it has to have three arguments because my code wants each of these functions to have three arguments.

So, this one is a trivial one, you have to define and identity always in order to run the code, it should have three arguments time, charge and current and independent of those arguments always going to return 1, because dt/dt is always 1. Then, charge dot is \dot{Q} , it is again an argument of t charge and current but it is only a function of current, so therefore \dot{Q} is only equal to current and then finally current dot which is the principal part of the differential equation.

Again, it is a function of t, charge and current in general but in this scale there is $-L^2/C$ which is $wQ - I$. So, therefore I is the third equation in this matrix equation and finally the initial conditions are to set, the initial conditions are at $t=0$. I have got $Q=1$ and $I=0$.

So, this is my set of initial conditions, when I execute that, I suppress all the outputs using the semicolon over here. So, when I suppress the only output that I am printing over here is this. So, when I take $w = 10$, $\beta = \sqrt{w-1}/4$ and $2\pi/\beta$ is 2.01. So, this is my time period in the dimensionless units of time.

(Refer Slide Time: 40:25)

```


beta
id[t_, charge_, current_] = 1;
chargeDot[t_, charge_, current_] = current;
currentDot[t_, charge_, current_] = -w charge - current;
initial = {0, 1, 0};

Out[272]=
2.01223

data = eulerGen[{id, chargeDot, currentDot}, initial, 10, 100]

Show[ListPlot[data[[;;, 1 ;; 2]], Joined -> True, PlotMarkers -> None,
      PlotRange -> Full], Plot[e^{-t/2} Cos[beta t], {t, 0, 10}, PlotRange -> Full,
      PlotStyle -> Red]]

```



```

{6.7, -0.650421, -2.09869}, {6.8, -0.86029, -1.2384},
{6.9, -0.98413, -0.254268}, {7., -1.00956, 0.755288}, {7.1, -0.934028, 1.68932},
{7.2, -0.765096, 2.45441}, {7.3, -0.519655, 2.97407}, {7.4, -0.222248, 3.19632},
{7.5, 0.0973833, 3.09893}, {7.6, 0.407276, 2.69166}, {7.7, 0.676442, 2.01521},
{7.8, 0.877963, 1.13725}, {7.9, 0.991688, 0.145562}, {8., 1.00624, -0.860683},
{8.1, 0.920176, -1.78086}, {8.2, 0.74209, -2.52295}, {8.3, 0.489795, -3.01275},
{8.4, 0.188521, -3.20127}, {8.5, -0.131606, -3.06966},
{8.6, -0.438572, -2.63109}, {8.7, -0.701694, -1.92941},
{8.8, -0.894621, -1.03479}, {8.9, -0.9981, -0.0366867},
{9., -1.00177, 0.965082}, {9.1, -0.905261, 1.87034}, {9.2, -0.718226, 2.58857},
{9.3, -0.459369, 3.04794}, {9.4, -0.154576, 3.20251}, {9.5, 0.165676, 3.03684},
{9.6, 0.46936, 2.56748}, {9.7, 0.726107, 1.84137}, {9.8, 0.910245, 0.931126},
{9.9, 1.00336, -0.0722308}, {10., 0.996134, -1.06836}

Show[ListPlot[data[[;;, 1 ;; 2]], Joined -> True, PlotMarkers -> None,
      PlotRange -> Full], Plot[e^{-t/2} Cos[beta t], {t, 0, 10}, PlotRange -> Full,
      PlotStyle -> Red]]

```

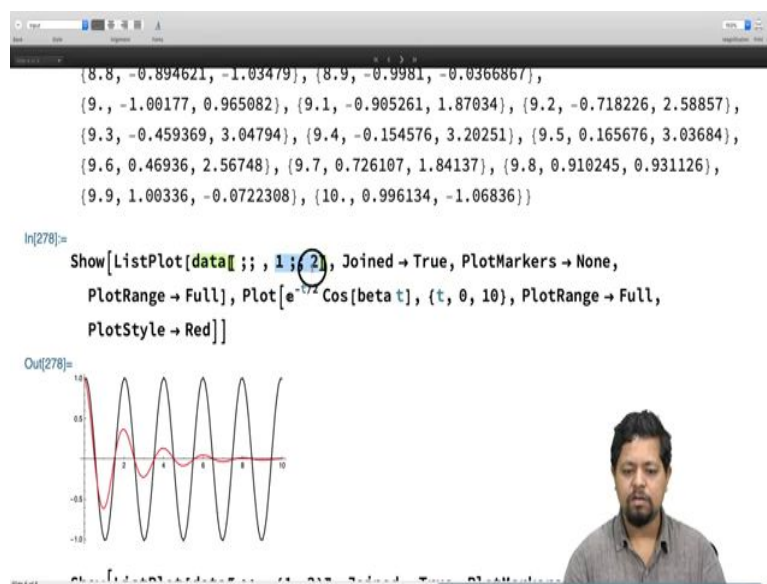
Now, I am going to go ahead and evaluate this, notice that for the Euler general function I am going to pass the first argument. First argument was supposed to be a set of functions. Set of functions which are defining x dot and those set of functions are id, charge dot and current dot. So, this is the set of three functions, id is 1, this is q dot and this is i dot.

Initial condition again, the size of initial condition is 3 and size of the function has to be also 3, if these two are not equally in some sort of error. Then, final time $t_f=10$ and number of steps, I am going to take as, let us say 100 to start out small. Let us go ahead and actually, go

ahead and execute this hundred I can print on screen and read so that may remove the semicolon.

I am going to execute this and there we go, this gives me the expected result. I see an array of set of 3 points, this is my initial condition, this was the initial value and then some changes happened to the initial value and then, some more changes and some more evaluation and so on. We get evaluation up to $t_f=10$, so this is the value of final charge and this is the value of final current. Second value is always the charge and third one is the current. So, let us go ahead and plot this.

(Refer Slide Time: 41:55)



The screenshot shows a Mathematica notebook with the following content:

9.	-1.00177	0.965082
9.1	-0.905261	1.87034
9.2	-0.718226	2.58857
9.3	-0.459369	3.04794
9.4	-0.154576	3.20251
9.5	0.165676	3.03684
9.6	0.46936	2.56748
9.7	0.726107	1.84137
9.8	0.910245	0.931126
9.9	1.00336	-0.0722308
10.	0.996134	-1.06836

In[278]=
`Show[ListPlot[data[[;;, 1 ;; 2]], Joined -> True, PlotMarkers -> None, PlotRange -> Full], Plot[e-t/2 Cos[beta t], {t, 0, 10}, PlotRange -> Full, PlotStyle -> Red]`

Out[278]=

In[280]=
`Show[ListPlot[data[[;;, 1 ;; 2]], Joined -> True, PlotMarkers -> All, PlotRange -> Full], Plot[e-t/2 Cos[beta t], {t, 0, 10}, PlotRange -> Full, PlotStyle -> Red]`

Out[280]=
 - ListPlot: $\begin{matrix} 0. & 1. & 0. \\ 0.1 & 1. & -1. \end{matrix}$ is not a list of numbers or pairs of numbers.
 - ListPlot: $\begin{matrix} 0. & 1. & 0. \\ 0.1 & 1. & -1. \end{matrix}$ is not a list of numbers or pairs of numbers.
 - ListPlot: $\begin{matrix} 0. & 1. & 0. \\ 0.1 & 1. & -1. \end{matrix}$ is not a list of numbers or pairs of numbers.
 - General: Further output of ListPlot::lgn will be suppressed during this calculation.
 - Show: Could not combine the graphics objects in
`Show[ListPlot[$\begin{matrix} 0 & 1 & 0 \\ 0.1 & 1. & -1. \end{matrix}$, Joined -> True, PlotMarkers -> All, PlotRange -> Full]`

In order to plot this here is my comparison, what I have done over here is that I am doing a list plot of the data that I have just calculated and in the data I am taking all the rows, but only the first and the second column that is I am taking the first column is time, the second column is current sorry charge.

So, I can go ahead and actually, in order to make it more readable I can do this as table form, so that you can see. So, the first column is time, second column is charge and the third column is current and I want to take all the rows, this is for all the rows. But, only the first and the second column, so that gives me Q as a function of time or Q versus time data, and I make a list plot of that. So, that gives me, this part gives me the list plot of that.

And, so let me put the plot markers back so that I can see and then I am comparing this with the analytical solution which is in the red. Analytical solution $e^{-t}/2$, the non-dimensional version of the analytical solution $\cos(\beta t)$, $\beta = \sqrt{w-1/4}$, w was a dimensionless ratio of L, R and C.

So, this is the analytical solution, I want to compare the analytical solution with the solution that I have got and see that this comparison is really bad. I think I have put a table form here.

(Refer Slide Time: 43:43)

```

chargeDot[t_, charge_, current_] = current;
currentDot[t_, charge_, current_] = -w charge - current;
initial = {0, 1, 0};

Out[272]=
2.01223

data = eulerGen[{id, chargeDot, currentDot}, initial, 10, 100];

Out[279]/TableForm=


| t   | charge    | current   |
|-----|-----------|-----------|
| 0   | 1         | 0         |
| 0.1 | 1.        | -1.       |
| 0.2 | 0.9       | -1.9      |
| 0.3 | 0.71      | -2.61     |
| 0.4 | 0.449     | -3.059    |
| 0.5 | 0.1431    | -3.2021   |
| 0.6 | -0.17711  | -3.02499  |
| 0.7 | -0.479609 | -2.54538  |
| 0.8 | -0.734147 | -1.81123  |
| 0.9 | -0.91527  | -0.895963 |
| 1.  | -1.00487  | 0.108903  |


```

```

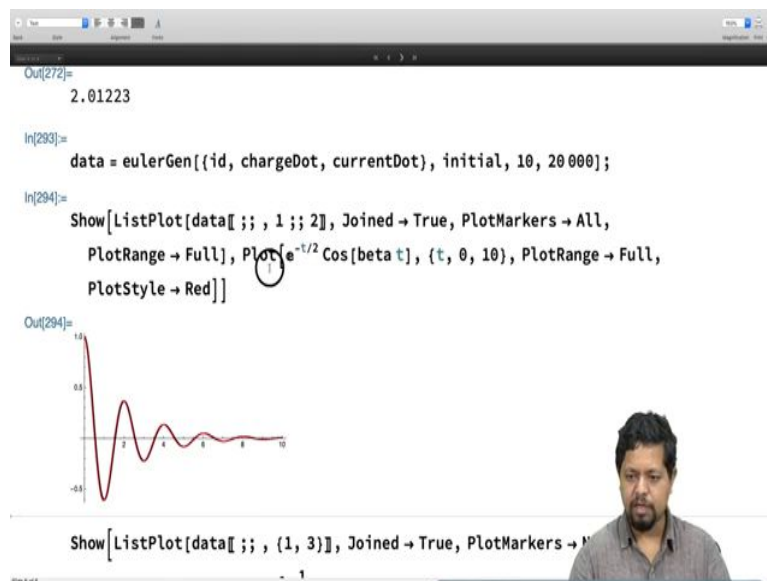
In[281]=
data = eulerGen[{id, chargeDot, currentDot}, initial, 10, 100];

In[282]=
Show[ListPlot[data[[;;, 1 ;; 2]], Joined -> True, PlotMarkers -> All,
PlotRange -> Full], Plot[e^{-t/2} Cos[beta t], {t, 0, 10}, PlotRange -> Full,
PlotStyle -> Red]]

Out[282]=


Show[ListPlot[data[[;;, {1, 3}]], Joined -> True, PlotMarkers -> None,
PlotRange -> Full], Plot[-1/2 e^{-t/2} Cos[t beta] - e^{-t/2} beta Sin[t beta], {t, 0, 10}, PlotRange -> Full, PlotStyle -> Red]]

```



So let me remove this table form because I want to take data as a list. Let me put a semicolon back again, now that you understand, so let me put tangled semicolon here. Now, I am going to do this comparison and there we go, so this is my analytical solution and it does not really work very well. It does not give me any comparison, any good comparison with the analytical solution. So, let me go ahead and see this if this is because of h being large.

Let me make it number steps as thousand, so h becomes small and compare that, that is a significant improvement. In order to see that we try it as 100 less, try 500 and it is in between. So 100 was really poor, in order to just verify that, this was 100, 100 was extremely poor, that means errors were large. 500 is getting better and 1000 is even better, this is thousand and finally 10000.

If I go ahead and do the 10000, you will see it is a fantastic agreement actually, not that great but we can actually go ahead and increase it to 20000 and you see from 10000 to 20000 is a slight amount of improvement. If you notice that, but not a significant amount of improvement, red and black colours are not exactly on top of each other, but very close to each other, it is quite decent. So, you can go ahead and play with values of beta.

(Refer Slide Time: 45:26)

The top screenshot shows the following code and output:

```

f(t) = 0
X = {t, Q, I}
X = {t, 1, 0}

```

Implementation: Lets take the ratio $w = L / (R^2 C)$

```

In[295]:=
w = 5;
beta = Sqrt[w - 1/4];
2.0 Pi
beta
id[t_, charge_, current_] = 1;
chargeDot[t_, charge_, current_] = current;
currentDot[t_, charge_, current_] = -w charge - current;
initial = {0, 1, 0};
Out[297]=
2.88292

```

The bottom screenshot shows the following code and output:

```

Out[297]=
2.88292
In[302]:=
data = eulerGen[{id, chargeDot, currentDot}, initial, 10, 20000];
In[303]:=
Show[ListPlot[data[[;;, 1 ;; 2]], Joined -> True, PlotMarkers -> All,
PlotRange -> Full], Plot[E^-t/2 Cos[beta t], {t, 0, 10}, PlotRange -> Full,
PlotStyle -> Red]]
Out[303]=

```

The plot shows a decaying oscillation of the current over time, with a red curve and a black curve. A circled 'I' is visible next to the plot.

So in order to place value of beta what we need to do is, or $L/R^2 C = w$. I can go ahead and make w even smaller, if I make w smaller the oscillation frequency becomes larger. Time period becomes 2.88, we can actually compare the time period also before the time period was 2 and that is about right from peak to peak the time period was 2.

And now it should be close to 3. So, let us go ahead and check that, there we go, it is our comparison. We changed the value of beta so the curves changed or we changed the value of

w so the curves changed and our time period from this peak to this peak is about 2.88 which is slightly less than 3 which we can see from the plot over here.

It is a good idea to actually check some of the basics while we are doing the computation so that we know that computer is doing what we intended to do, we have not made some sort of mistake.

(Refer Slide Time: 46:21)

The image shows two screenshots of a Mathematica notebook. The first screenshot displays the following code and output:

```

X =  $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$  X =  $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ 
Implementation: Lets take the ratio  $w = L / (R^2 C)$ 
w = 20;
beta =  $\sqrt{w - \frac{1}{4}}$ ;
 $\frac{2.0 \pi}{\text{beta}}$ 
id[t_, charge_, current_] = 1;
chargeDot[t_, charge_, current_] = current;
currentDot[t_, charge_, current_] = -w charge - current;
initial = {0, 1, 0};
Out[338]= 4.74964
In[350]= data = eulerGen[{id, chargeDot, currentDot}, initial, 10, 100

```

The second screenshot shows the following code and output:

```

initial = {0, 1, 0};
Out[353]= 1.41383
In[358]= data = eulerGen[{id, chargeDot, currentDot}, initial, 10, 10000];
In[359]= Show[ListPlot[data[[;;, 1 ;; 2]], Joined -> True, PlotMarkers -> None,
PlotRange -> Full], Plot[e^{-t/2} Cos[beta t], {t, 0, 10}, PlotRange -> Full,
PlotStyle -> Red]]
Out[359]=

```

The plot shows a decaying oscillation (red line) and a reference function $e^{-t/2} \cos(\beta t)$ (black line).

When w is 20, L/R^2C is 20. So, therefore this is going to, the R is going to be very-very small and as a consequence resistance small that means I should have lots of oscillations and you see the time period reduced down to 1.41.

Let us go ahead and do this for 10000 points which should be enough, the execution is done, so let us go ahead and compare and we see the comparison is very well. Let us try to re-obtain the limit when the resistance is very small.

(Refer Slide Time: 46:55)

$$X = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \quad X = \begin{pmatrix} 1 \\ 1 \\ -\frac{1}{R^2 C} Q - 1 \end{pmatrix}$$

• Implementation: Lets take the ratio $w = L / (R^2 C)$

```

w = 100;
beta = Sqrt[w - 1/4];
2.0 Pi
beta
id[t_, charge_, current_] = 1;
chargeDot[t_, charge_, current_] = current;
currentDot[t_, charge_, current_] = -w charge - current;
initial = {0, 1, 0};
Out[353]=
1.41383
In[358]:=
data = eulerGen[id, chargeDot, currentDot, initial, 10, 100];

```

0.629105

```

In[367]:=
data = eulerGen[{id, chargeDot, currentDot}, initial, 10, 10000];
In[368]:=
Show[ListPlot[data[[;;, 1 ;; 2]], Joined -> True, PlotMarkers -> None,
PlotRange -> Full], Plot[e^-t/2 Cos[beta t], {t, 0, 10}, PlotRange -> Full,
PlotStyle -> Red]]
Out[368]=

```

Show[ListPlot[data[[;;, {1, 3}]], Joined -> True, PlotMarkers -> None,
 PlotRange -> Full], Plot[-1/3 e^-t/2 Cos[t beta] - e^-t/2 beta Sin[t

Let us make this even larger, let us make w equal to 100 that is the case when resistance becomes small and we can go ahead and evaluate this and do a comparison. Now, you see as I make w larger and larger, the damping becomes much-much more slower and there are more oscillations that are happening.

That means by the time, amplitude decays to one third, we are going to cover more oscillations. We can go ahead and check that back.

(Refer Slide Time: 47:24)

• Implementation: Lets take the ratio $w = L / (R^2 C)$

```
In[369]:= w = 10;
```

$$\text{beta} = \sqrt{w - \frac{1}{4}};$$

```
2.0 \pi  
beta  
id[t_, charge_, current_] = 1;  
chargeDot[t_, charge_, current_] = current;  
currentDot[t_, charge_, current_] = -w charge - current;  
initial = {0, 1, 0};
```

```
Out[371]= 2.01223
```

```
In[367]:= data = eulerGen[{id, chargeDot, currentDot}, initial, 10, 10000];
```

```
In[368]:= Show[  
ListPlot[data[[;;, 1 ;; 2]], Joined -> True, PlotMarkers -> None,  
PlotRange -> Full], Plot[e^{-t/2} Cos[beta t], {t, 0, 10}, PlotRange -> Full,  
PlotStyle -> Red]
```

Out[377]=

```
Show[  
ListPlot[data[[;;, {1, 3}]], Joined -> True, PlotMarkers ->
```

When w was 10, for $w=10$ we can see that amplitude was 1 over here by the time it dropped down to one third which is about here. We have covered just about one oscillation.

(Refer Slide Time: 47:41)

The first screenshot shows the following code and output:

```

dI = R^2 C
Q(0) = 1
I(0) = 0
X = {I, Q}
X = {I, Q - I}

```

Implementation: Lets take the ratio $w = L / (R^2 C)$

```

In[378]:=
w = 100;
beta = Sqrt[w - 1/4];
2.0 Pi
beta
id[t_, charge_, current_] = 1;
chargeDot[t_, charge_, current_] = current;
currentDot[t_, charge_, current_] = -w charge - current;
initial = {0, 1, 0};
Out[380]=
0.629105

```

The second screenshot shows the following code and output:

```

Out[380]=
0.629105
In[385]:=
data = eulerGen[{id, chargeDot, currentDot}, initial, 10, 10000];
In[386]:=
Show[ListPlot[data[[;;, 1 ;; 2]], Joined -> True, PlotMarkers -> None,
PlotRange -> Full], Plot[E^-t/2 Cos[beta t], {t, 0, 10}, PlotRange -> Full,
PlotStyle -> Red]]
Out[386]=

```

The output shows a plot of the charge and current over time, with a red curve representing the theoretical solution and a blue curve representing the numerical solution. A small circle highlights a point on the red curve.

And as I increase w , make it larger, let us say make it 100 and there we go. By the time the amplitude drops to one third that is about here. We have covered about 3 or 4 oscillations. So, that means we are going more and more closer, the damping becomes less and less important and we are going more and more closer to the regime, where we are going to see the oscillations and not much of damping.

You see that at this turning points the agreement is not very well and it is because of the limitation of the Euler method. We can try to improve it by making this, increasing the h and let us see if that gives me any significant improvement. It does gives me little bit

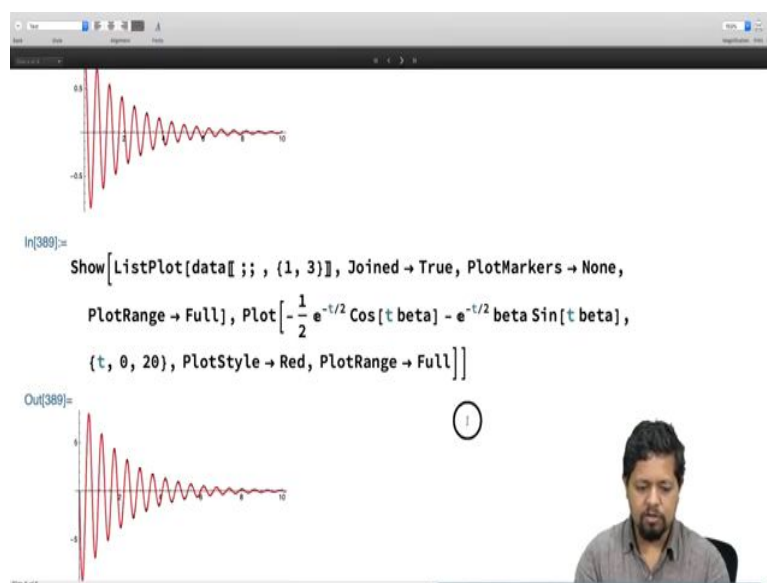
improvement but see to get that much of improvement which was about, you know fraction of maybe just about for 4 percent, for 5 percent improvement in this region over here.

I had to double the number of points, the computational cost is very high with the Euler method. At least that is what we noticed from this exercise, that the computational cost working with the Euler method is very high because local accuracies order h^2 and global accuracies order h , so I have to keep on, in order to improve the global accuracy of my solution I have to linearly scale down h that is, I have to, in order to improve the accuracy from point 1 to 0.01 that is by an order of magnitude, I have to take 10 times more points.

So, in order to improve accuracy by 1 order magnitude I have to take points 10 times more which is a huge computational cost. This is something we can improve by using better methods such as improved Euler's method or the second order Runge Kutta or even better fourth order Runge Kutta which is an accuracy of, local accuracy of h^6 and global accuracy of h^5 .

Before we do that, let us go ahead and quickly check. We did plot of current versus time, let us go ahead and do a plot of current versus, time versus current. This was time versus charge.

(Refer Slide Time: 49:50)



So in order to time versus current you need to extract the first element and third element of the data. So, this is all the rows of the data but the first and the third element, first being the time and third being the current.

So, if I do this and compare this with the current, you can work out the current equation for the current, I is a function of t from the analytical solution. You will find that this is the solution and you can do this comparison and you will see this is also a fair agreement.