

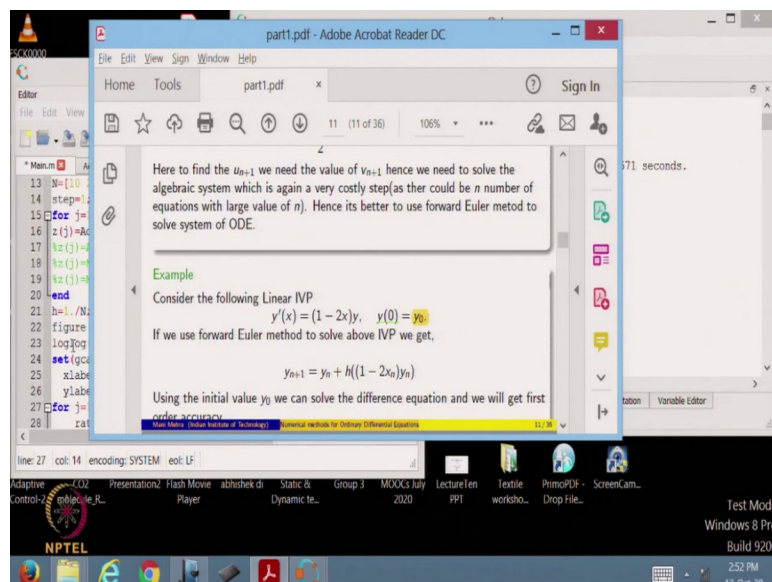
**Scientific Computing using Matlab**  
**Professor. Vivek Aggarwal & Professor. Mani Mehra**  
**Department of Mathematics**  
**Indian Institute of Technology, Delhi**  
**Lecture No. 64**  
**Matlab/Octave Code for Initial Value Problems**

So, welcome to all of you in the next class of this course, as you have already seen, how to go ahead for numerical methods for initial value problems. In today's lecture we will implement those numerical methods or difference schemes to solve initial value problems with the help of Octave software. Though you can also solve those problems in Matlab software I mean that the codes which I will be giving you here can be implemented in Matlab software also.

But Matlab is a commercial software and Octave is a freely available software that you, you must be knowing because you have handled with Octave in the previous part of this course already. So, that is why some of you are having Matlab. You can run those codes in Matlab otherwise you can download the octave and you can run those codes because Octave is freely available software.

Though in Matlab software you have some flexibility but it is still the code which I am giving you can be done in Matlab simultaneously in Octave as well. So, if you remember from our previous lectures, we have taken one example, which is this.

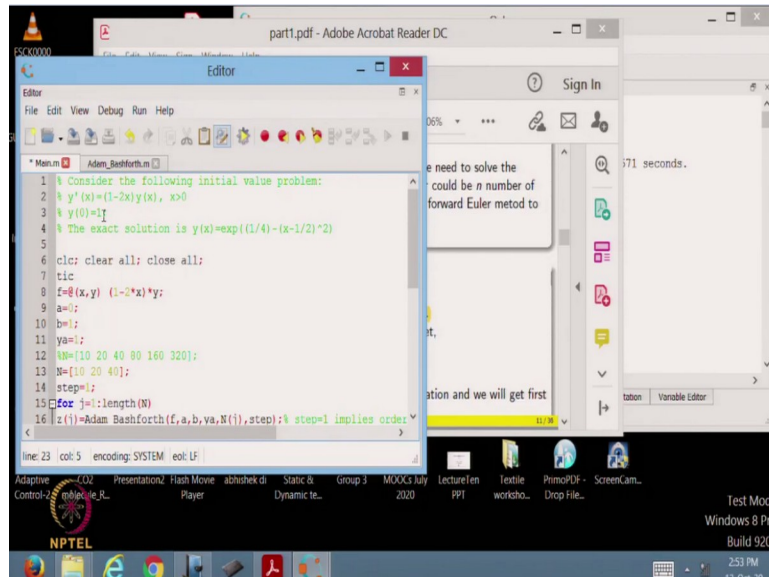
(Refer Slide Time: 01:43)



So, we have that consider the following linear Initial Value Problem

$y' = (1 - 2x)y, y(0) = y_0$ . So, in today's lecture, we will focus on the following example of Initial Value Problem and we will try to solve this Initial Value Problem with different numerical methods which we have already studied in previous lectures.

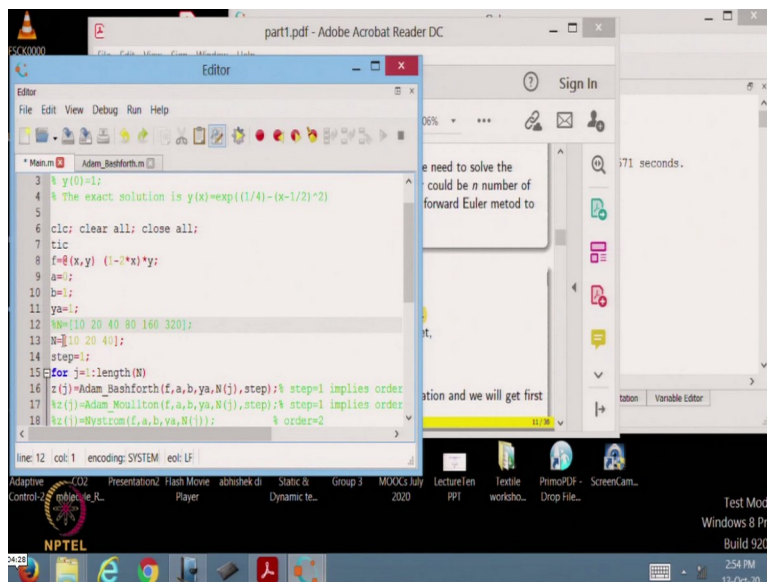
(Refer Slide Time: 02:14)



```
1 % Consider the following initial value problem:
2 % y'(x)=(1-2x)y(x), x>0
3 % y(0)=1
4 % The exact solution is y(x)=exp((1/4)-(x-1/2)^2)
5
6 clc; clear all; close all;
7 tic
8 f=@(x,y) (1-2*x)*y;
9 a=0;
10 b=1;
11 ya=1;
12 N=[10 20 40 80 160 320];
13 N=[10 20 40];
14 step=1;
15 for j=1:length(N)
16     z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=2
17 end
```

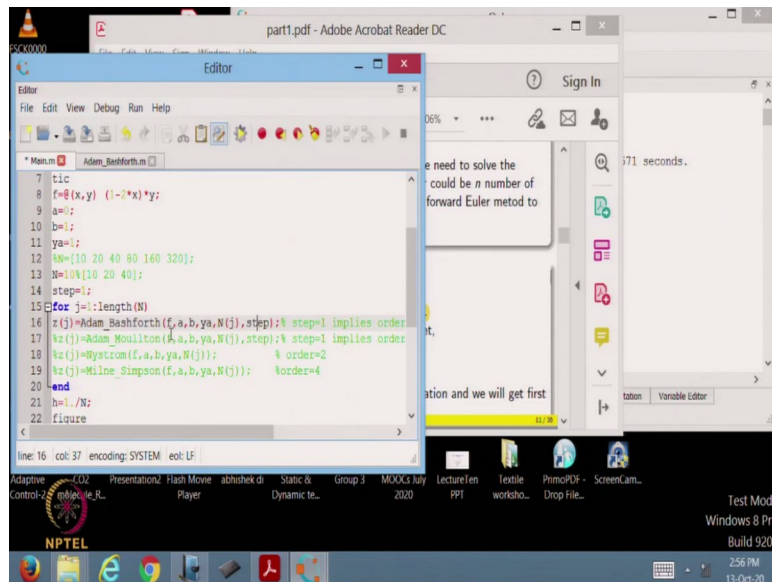
So, let us, I already know. I already have a code with me, I will try to explain each line of this code to you, so that you can type on your own code and you can go ahead with it. So, initially let me start with the comment which I have written which is very easy. So, this is a differential equation with the same example, I have typed in the comment part of this code and there it was an arbitrary initial condition  $y_0$  now, because to solve it numerically, we have to work out for a particular value of this initial condition which I have considered is 1.

(Refer Slide Time: 02:57)



So, after that `clc` clear all close all you are all you must be familiar with all those commands in earlier part of this course. So, I am not going to explain all those commands again. And then I am defining the function in the following way  $f$  is equal to which you can see from here,  $f = @(x, y)(1 - 2 * x) * y$ ; This is the way I can define the function. And then the initial point  $a$  is 0,  $b$  is 1,  $ya$  is 1, these are some variables  $a$ ,  $b$  and  $ya$ , which I am choosing in the following way. So, and after that, I am choosing a sample point  $N$ , which is  $N = [10, 20, 40]$ . For a time being initially I can take only 10. And then another variable which I am considering is  $step$ , which is equal to 1.

(Refer Slide Time: 4:07)



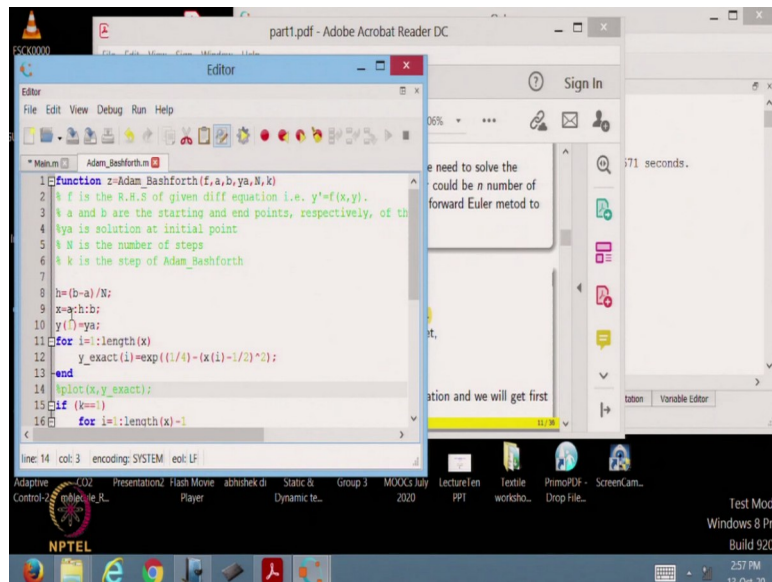
And then after that we are using the loop, for loop you must already be familiar with these concepts. That is why I am not again re explaining those concepts for  $j$  is equal to 1 into length of  $N$  length of  $N$  I am using because later on I will be working out with the array of  $N$  but right now I have just considered one single value of 10 so length of  $N$  does not make any really big concept here.

And then whatever algorithm of Adam Bashforth method with different order those algorithms I have implemented in the separate functions whose name is Adam Bashforth. So, that is a very good programming practice that whatever you should decompose your program into a smaller component and for each component you can write down your function. So, Adam Bashforth is a name of your functions and the argument of this function which you can observe from here is  $f$  which is a function  $a$   $b$   $ya$   $N(j)$  and a step.

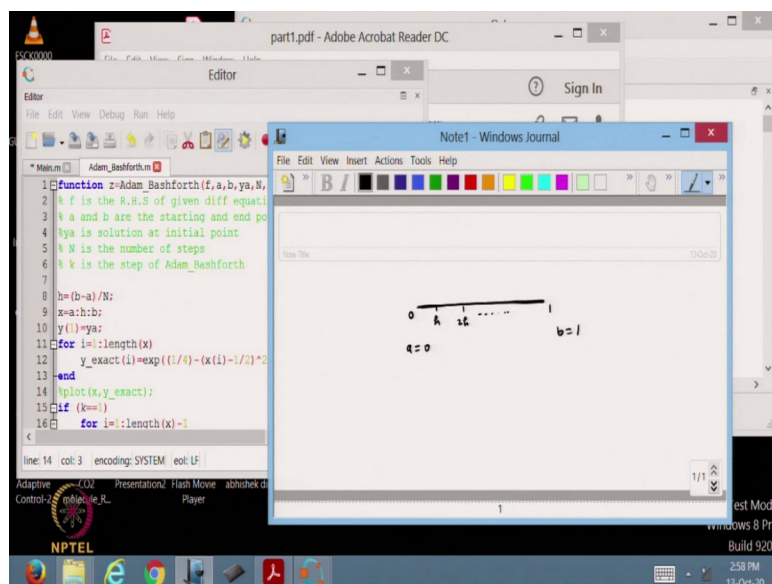
I will explain to you the meanings of each argument later on and the meaning of  $f$  is clear to you,  $a$  is an initial point  $b$  is the point where you wanted to compute your solutions. And  $ya$  is also final time and  $Nj$  is the total number of a grid points mesh size which I was based on and you can also define  $h$ . Basically,  $h = 1/N$ ,  $N$  step. So, step basically means which method you wanted to choose under the Adam Bashforth category.



(Refer Slide Time: 06:08)



```
1 function z=Adam_Bashforth(f,a,b,ya,N,k)
2 % f is the R.H.S of given diff equation i.e. y'=f(x,y).
3 % a and b are the starting and end points, respectively, of the
4 % ya is solution at initial point
5 % N is the number of steps
6 % k is the step of Adam_Bashforth
7
8 h=(b-a)/N;
9 x=a:h:b;
10 y(1)=ya;
11 for i=1:length(x)
12     y_exact(i)=exp((1/4)-(x(i)-1/2)*2);
13 end
14 plot(x,y_exact);
15 if (k==1)
16     for i=1:length(x)-1
```



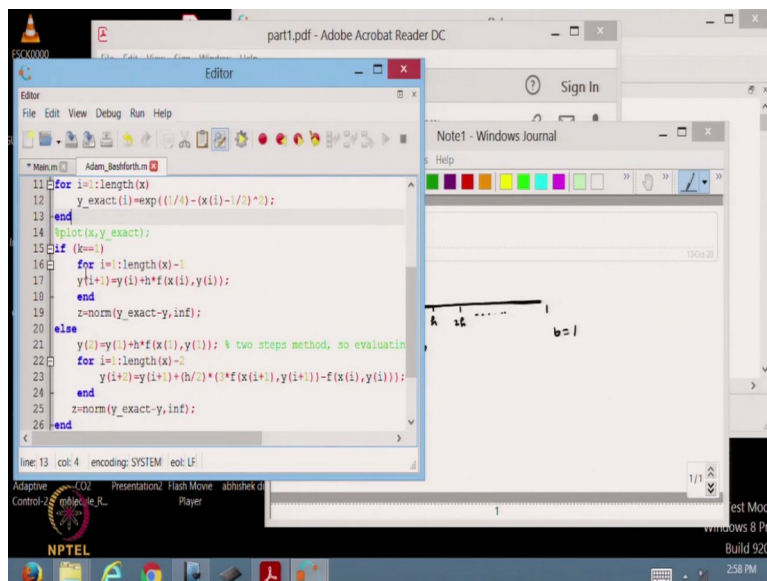
So, accordingly I have implemented this algorithm in Adam Bashforth function which I am showing you now. So, again the prototype of this function is this which you can observe from here function z is equal to Adam Bashforth and arguments are f a b ya N and k. So, here again I have commented each thing that f is the right hand side of a given differential equation, a and b are the starting and end points, ya is the solution at initial points. ya is the solution at the initial point and N is the number of step, k is the step of Adam Bashforth.

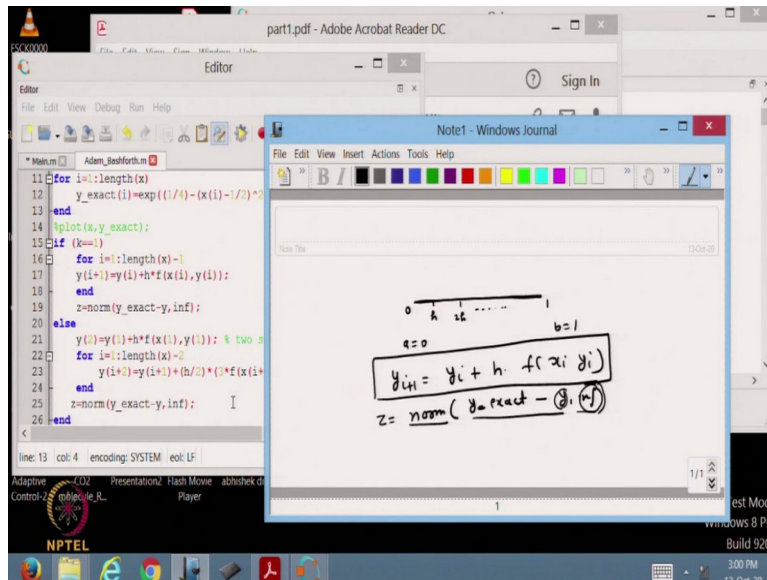
So, here I am explaining you the meaning of each argument which we are passing through Adam

Bashforth in the main program and then we are defining 
$$h = \frac{b - a}{N}$$
 and  $x = a:h:b$ ; basically if you see we are taking this mesh from 0 to 1, so, a is 0 and b is 1. So, h to 2h this way we can keep going. This is the mesh we are choosing to compute the solution of an Initial Value Problem.

So, after that again we are taking the concept of a loop for  $i = 1:\text{length}(x)$  and then we are sampling that exact solution which we already know will be in the following form. So, this is the exact solution of our Initial Value Problem. We are also implementing an exact solution, so that we can compute the error.

(Refer Slide Time: 08:17)

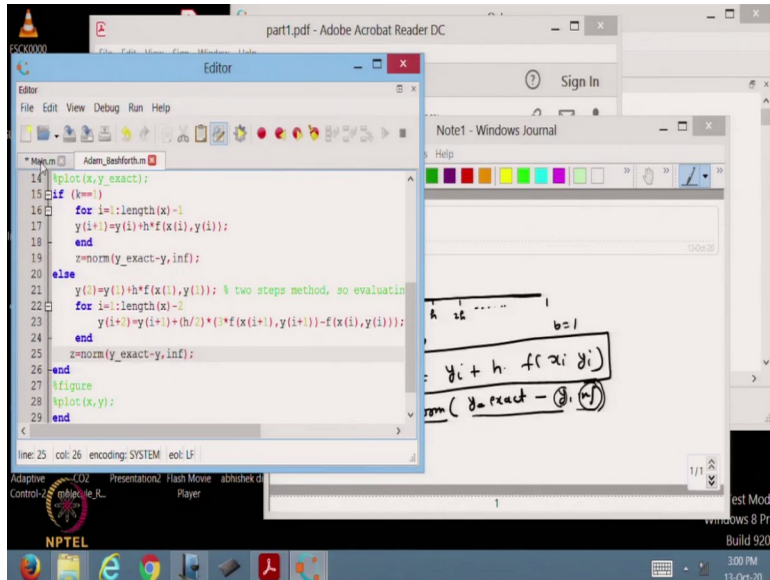




And then we have chosen the ifelse command to categorize based on different steps. So, if  $k = 1$  so, basically a step is it means a step is equal to 1, then as you with the help of again for loop we are implementing the same algorithm which we already know  $y_{i+1} = y_i + h f(x_i, y_i)$ . So, this is the algorithm of Euler methods which we already know. So, in this Adam\_Bashforth.m file you can see how I have implemented this algorithm.

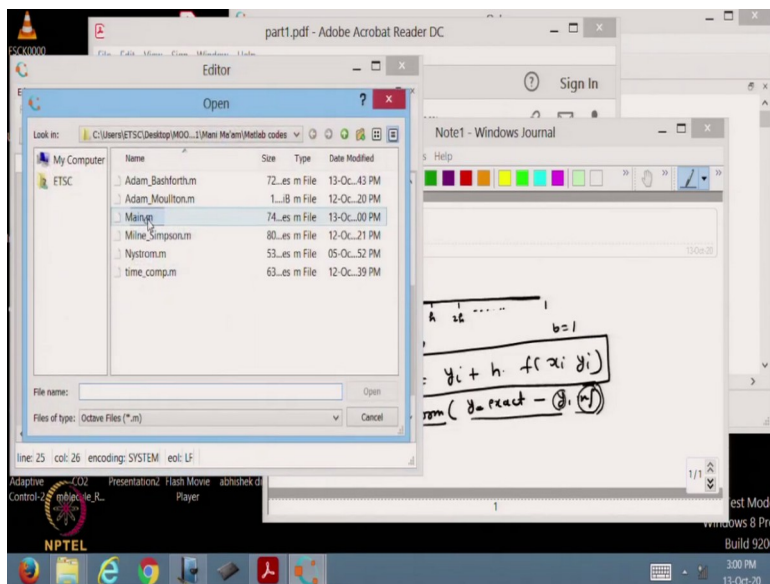
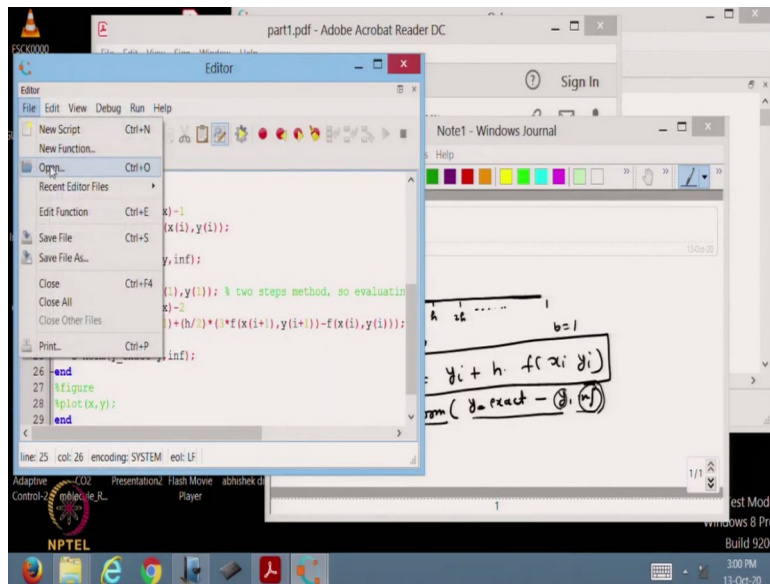
And then what we are doing we are also computing norm of infinity norm, infinity norm. So, norm is an inbuilt function in Octave or Matlab which computes the norm. Of course, default is 2 norm but here I am explicitly after comma I am explicitly mentioning inf So, this will be an infinity norm. So, y\_exact is the exact solution and y is a numerical solution. So, we are basically computing the norm of the infinity norm of the error or you can say maximum norm of the error.

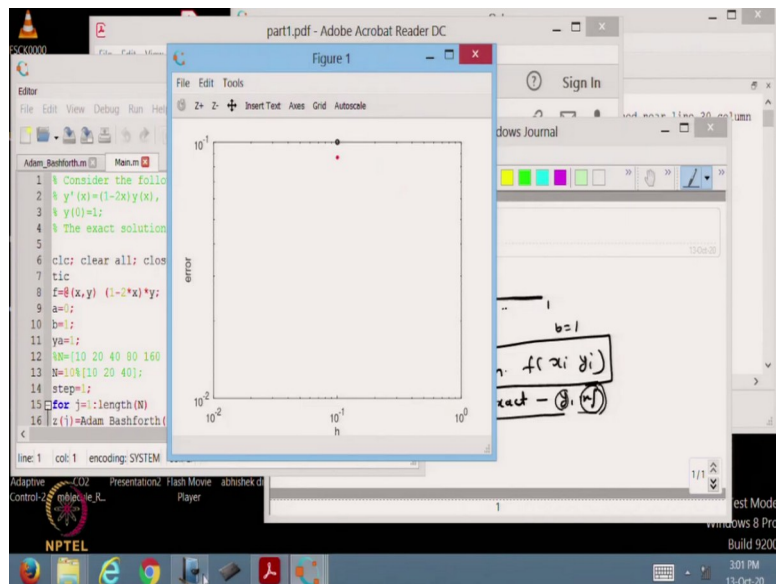
(Refer Slide Time: 10:03)

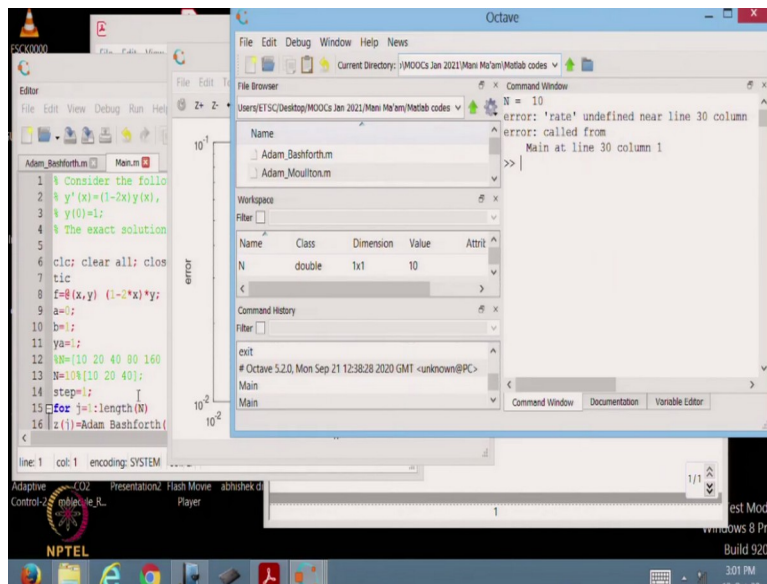


So, that's it. So, just by computing the error we can find out whether our difference scheme is correct or not we are going in the right direction or not. So, for that reason I am running my program.

(Refer Slide Time: 10:24)



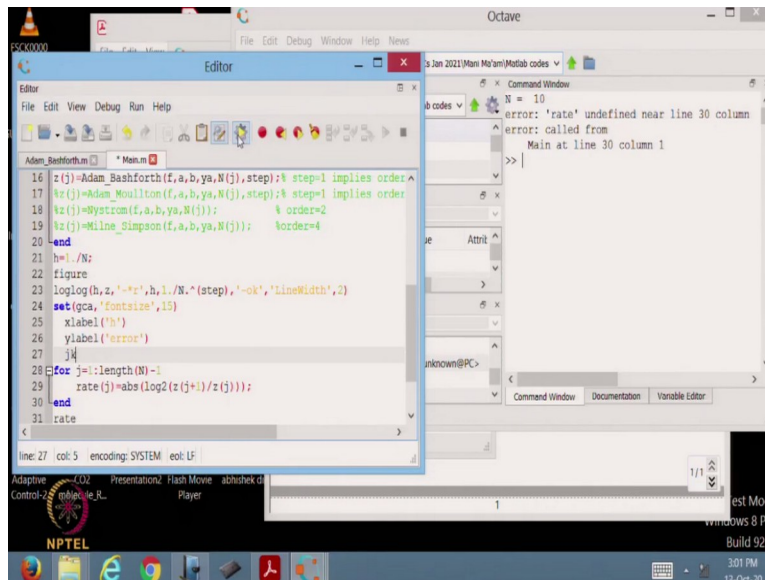




So, if you see here what is the error rate undefined.



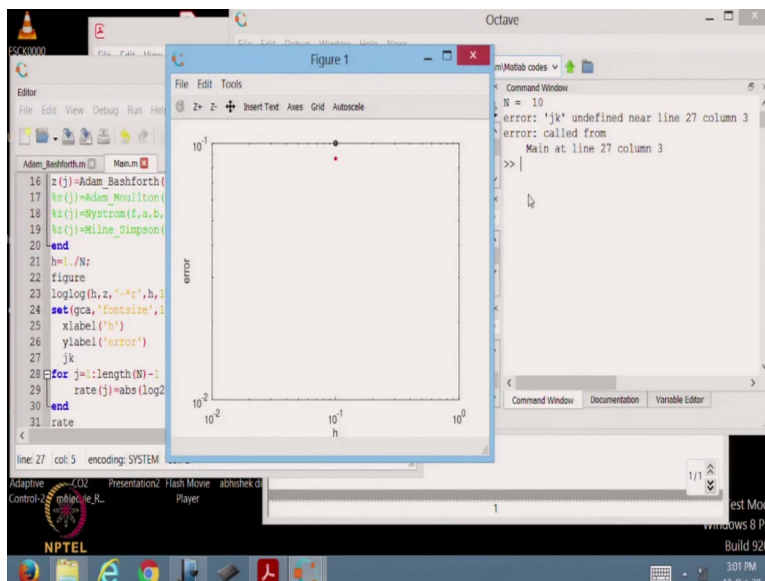
(Refer Slide Time: 10:44)



The screenshot shows the Octave IDE with a script in the Editor window. The script defines a function `z(j)` that calls `Adam_Bashforth`, `Adam_Moulton`, `Wystrom`, and `Wline_Simpson` with various parameters. It then calculates `h=1./N;`, creates a `figure`, and uses `loglog` to plot `h` versus `error`. The `error` variable is calculated as `abs(log2(z(j+1)/z(j)))`. The script ends with `rate`. The Command Window shows an error: `error: 'rate' undefined near line 30 column 1`. The error message indicates that the variable `rate` is undefined at the point where it is used in the script.

So, basically I will tell you what the problem is. So, I want my code to be run only this till this point that is why I have just typed some arbitrary number. So, that error should come naturally and code should stop.

(Refer Slide Time: 11:09)



So, error jk has come.

(Refer Slide Time: 11:16)

The screenshot shows the Octave editor window with a script file named 'Main.m'. The script contains the following code:

```
13 N=10; [10 20 40];
14 step=1;
15 for j=1:length(N)
16     z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order
17     %z(j)=Adam_Moulton(f,a,b,ya,N(j),step); % step=1 implies order
18     %z(j)=Mystrom(f,a,b,ya,N(j)); % order=2
19     %z(j)=Mline_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-xr',h,1./N,'(step)','-ok','LineWidth',2)
24 set(gca,'FontSize',15)
25 xlabel('h')
26 ylabel('error')
27 jk
28 for j=1:length(N)-1
```

The Command Window displays the following error message:

```
N = 10
error: 'jk' undefined near line 27 column 3
error: called from
Main at line 27 column 3
>> |
```

The screenshot shows the Octave editor window with the same script file 'Main.m'. The script is now corrected and includes the following code:

```
13 N=10; [10 20 40];
14 step=1;
15 for j=1:length(N)
16     z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order
17     %z(j)=Adam_Moulton(f,a,b,ya,N(j),step); % step=1 implies order
18     %z(j)=Mystrom(f,a,b,ya,N(j)); % order=2
19     %z(j)=Mline_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-xr',h,1./N,'(step)','-ok','LineWidth',2)
24 set(gca,'FontSize',15)
25 xlabel('h')
26 ylabel('error')
27 jk
28 for j=1:length(N)-1
```

The Command Window displays the following output:

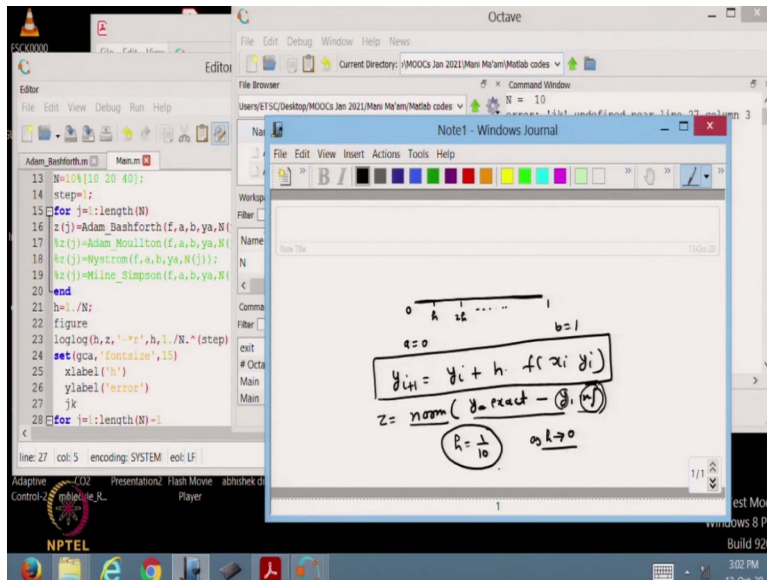
```
N = 10
error: 'jk' undefined near line 27 column 3
error: called from
Main at line 27 column 3
>> z
z = 0.087182
>>
```

The Workspace window shows the following variables:

Name	Class	Dimension	Value	Attrib
N	double	1x1	10	

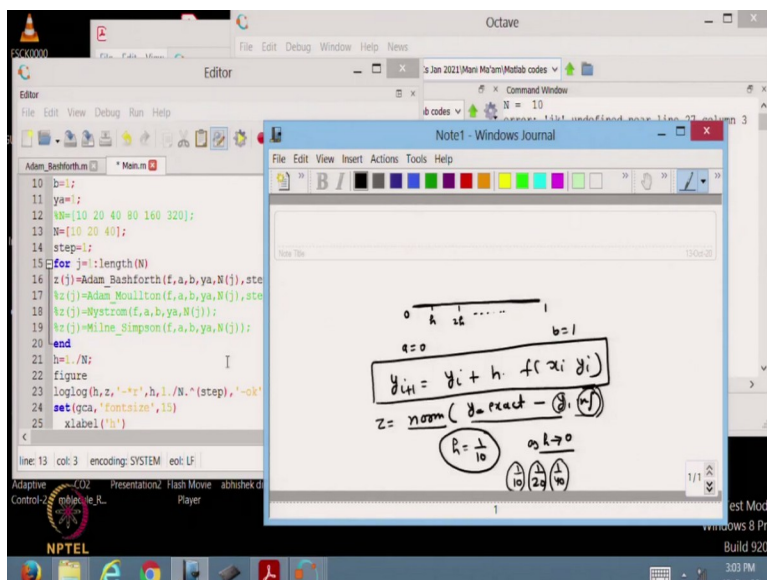
The Command History window shows the following commands:

```
exit
# Octave 5.2.0, Mon Sep 21 12:38:28 2020 GMT -unknown@PC>
Main
Main
```

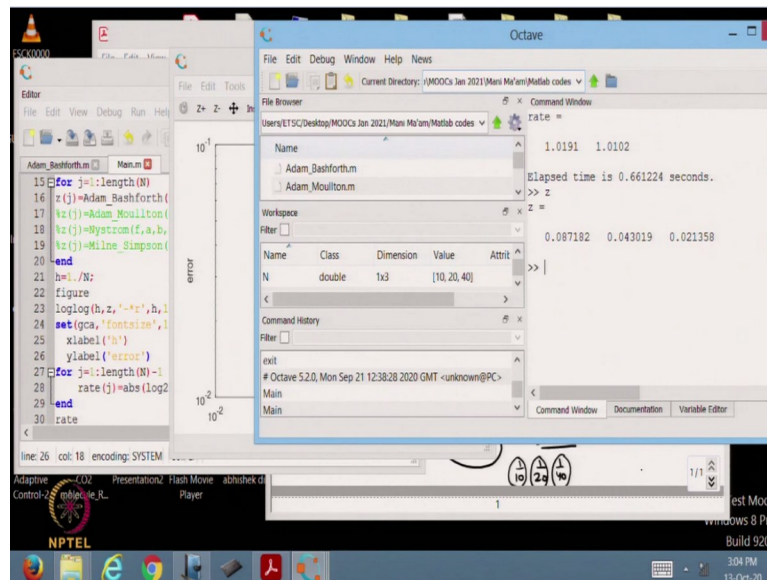


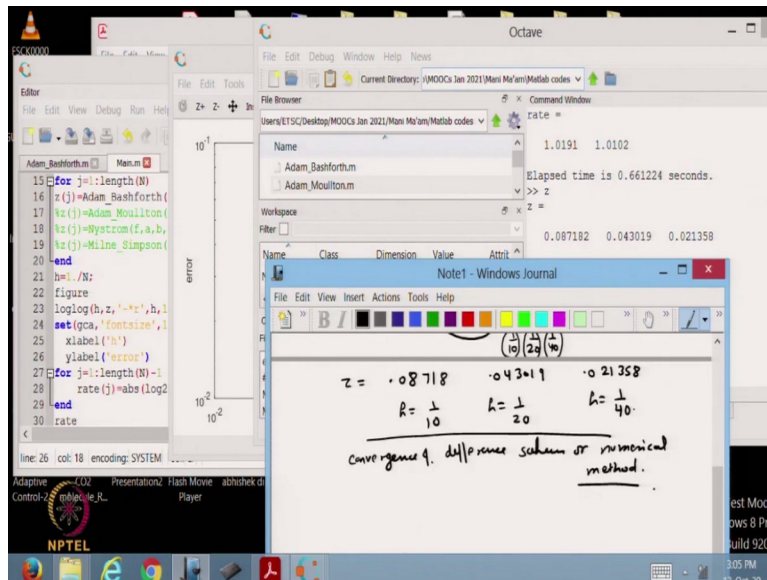
So, now and if you see what is the output of this code, Adam Bashforth output is in the form of  $z$  so, basically  $z$  is that is the error which if I type it here I should see what is the error. So, you see the error is 0.08 and in our opinion this is the error corresponding to  $N = 10$  or what you can say corresponds to  $h = 1/10$ . So, how we have studied the convergence of a difference scheme that  $h \rightarrow 0$  numerical solution should converge to exact solution or basically error should converge to 0, that is what we have already seen in our previous lectures. So, for that reason, I will also show you how error behaves with respect to reducing  $h$ . So, if I am reducing  $h$  it means I have to increase  $N$  that is the reason that I was taking an array here.

(Refer Slide Time: 12:33)



(Refer Slide Time: 13:07)



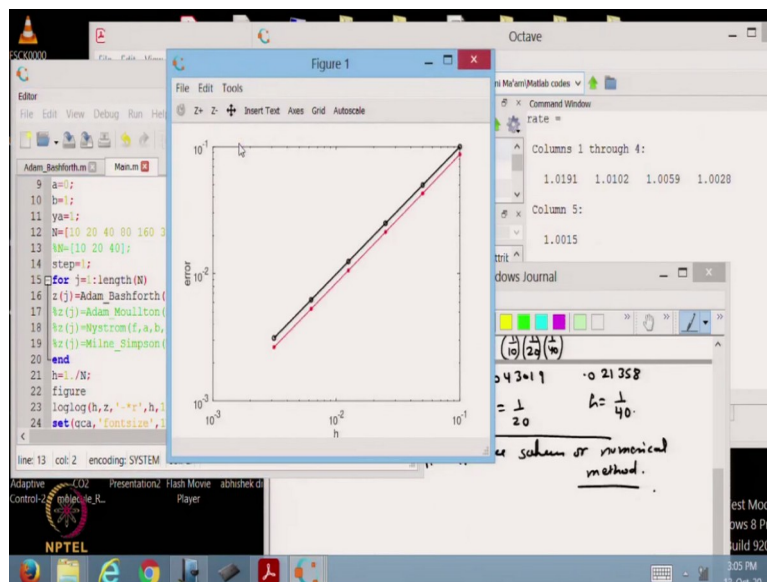
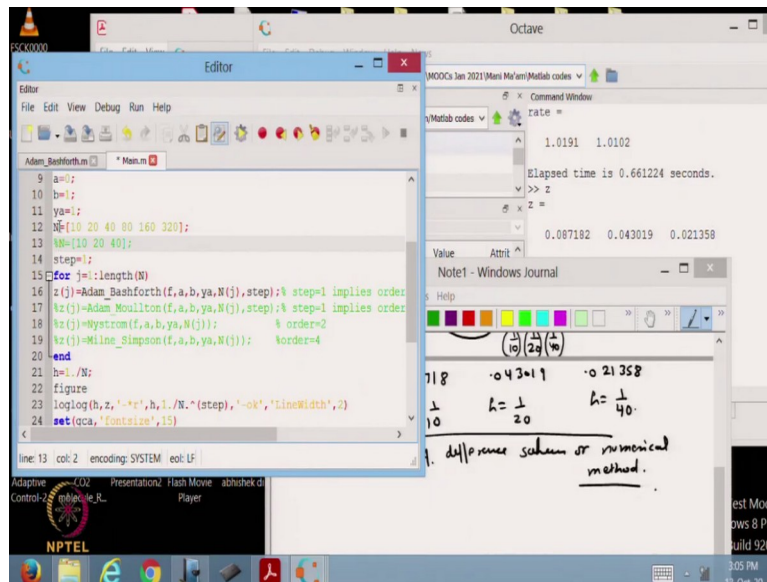


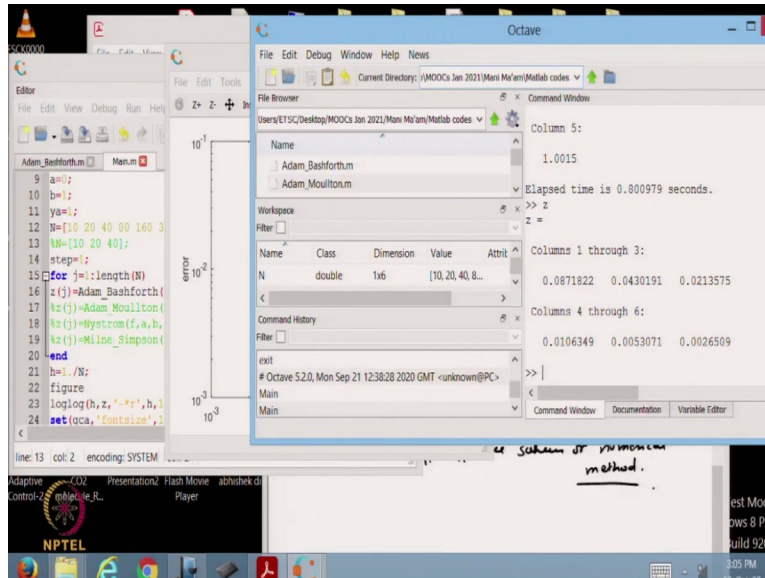
So, now the error will also be in the vector  $z$ . So, error is also a stored corresponds to  $h = 1/10$  corresponds to  $h = 1/20$  and corresponds to  $h = 1/40$ , which you can see from here. So, as expected,  $h = 1/10$ , which I can write down here. So, the value which I am getting this 0.08718 corresponds to  $h = 1/10$ , 0.043019 corresponds to  $h = 1/20$ , 0.021358 corresponds to  $h = 1/40$ .

So as expected, error is decreasing, error is decreasing with respect to  $h \rightarrow 0$  that is what we call it as a convergence of a difference scheme or numerical method, convergence of difference scheme or numerical method. So, we have verified that the forward Euler method is converging by taking one particular example of an Initial Value Problem.



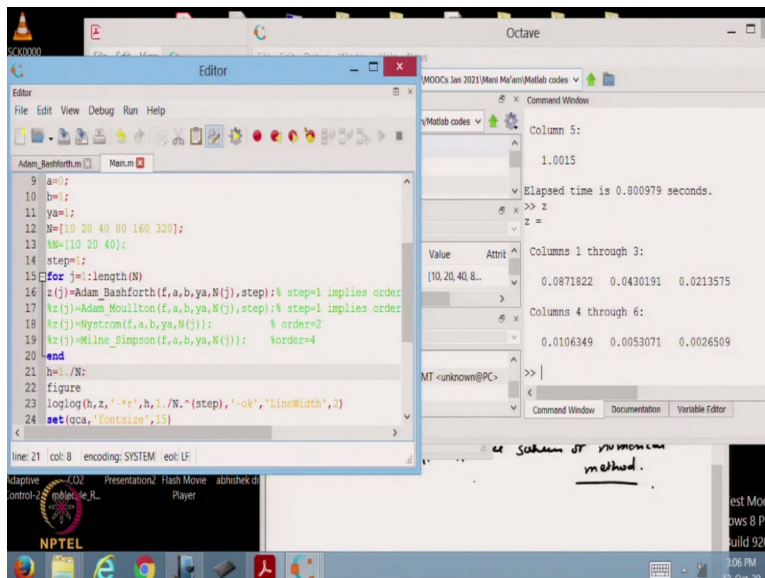
(Refer Slide Time: 14:44)



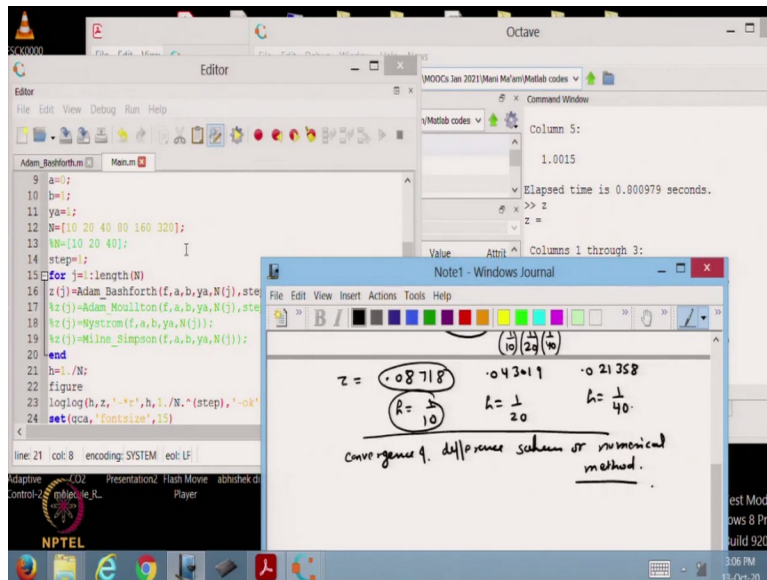


In fact, if you want to see you can also see by taking this N, so here in fact, error will be for error not will be error should be further reduced. So that is what you can type z here and you can see.

(Refer Slide Time: 15:16)

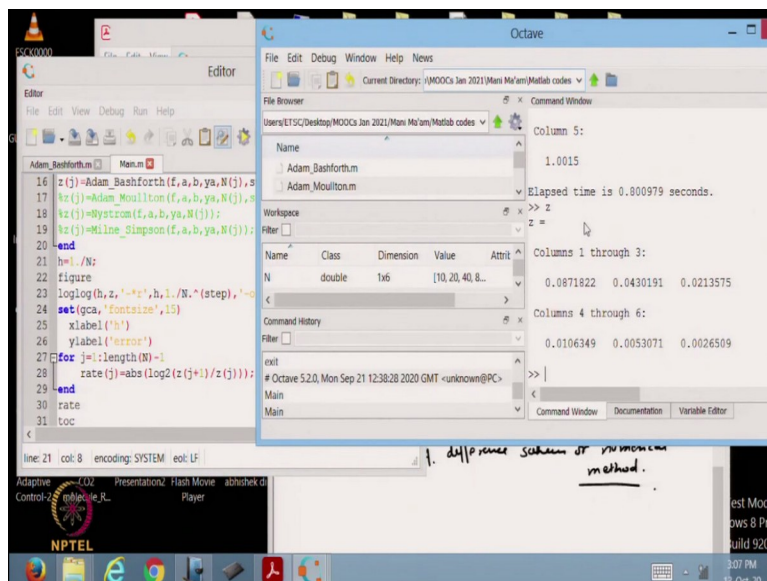


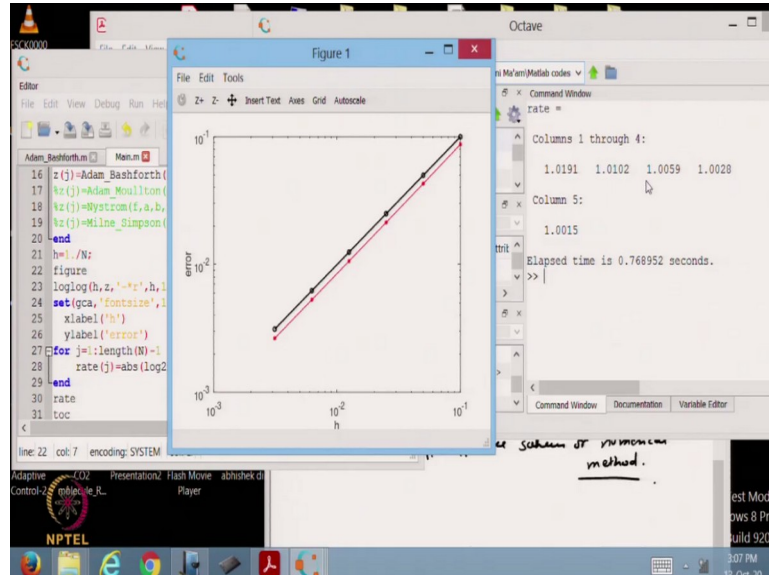
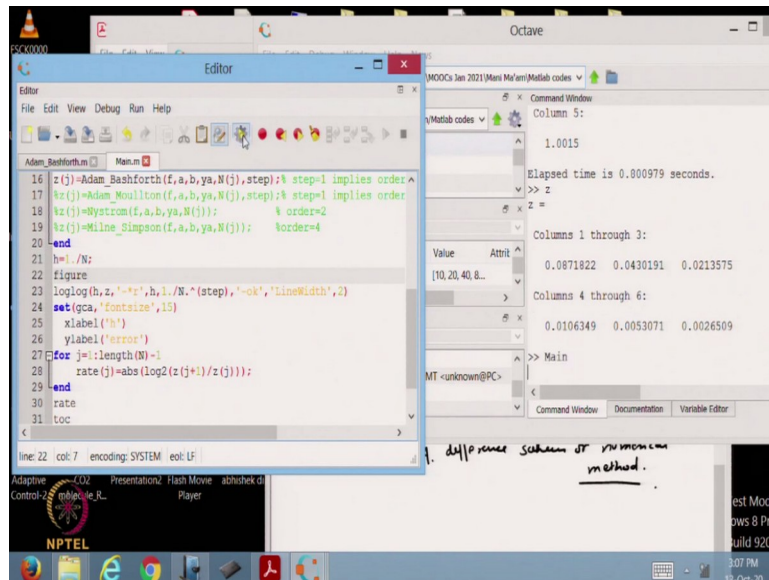




So, if you want to decrease the error further, you have to increase and you can go beyond 320 that is up to you, but I have just given you the flavor how you can verify that whether your differences scheme is converging or not with the help of one particular example, which we have already discussed in our lecture class. So, now, because just by computing error it corresponds to  $1h$ , we do not guarantee that our differences scheme is converging, converging means as  $h \rightarrow 0$  error should tend to 0 and that we can observe provided we have worked out for different values of  $h$  or  $N$ , that is what we have done with the help of following examples.

(Refer Slide Time: 16:08)





And further, basically, if we want to compute the rate also, that also these are just some commands I have given to put the x label inviolable in the finger and then finally, with the help of a loop, we have also computed the rate which you can observe from here. So, basically, if you see the rate when I run this code, you will see the rate here, it is fast. So, what it means is that the forward Euler method is first order accurate, first order accurate, that is what we have done with  $\text{abs}(\log_2(z(j+1)/z(j)))$ .

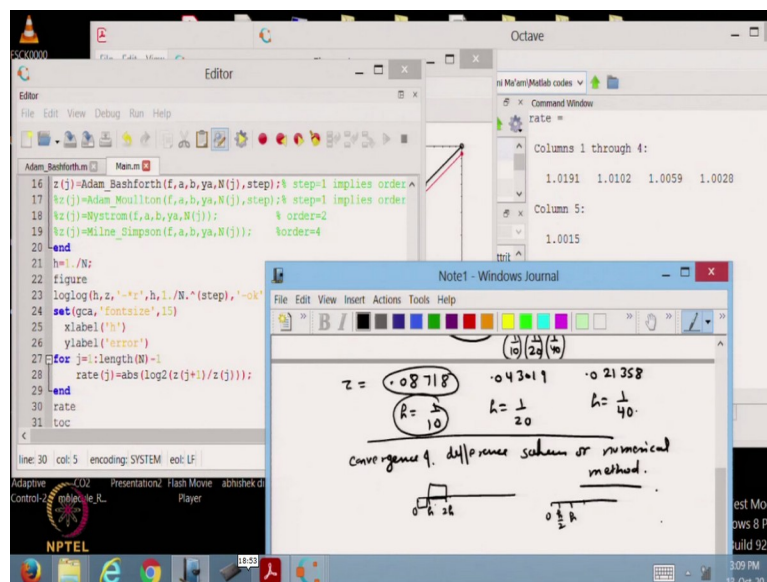
So, basically we are dividing the error at two different  $h$  and then we are taking the log on both sides. So that is how we will get to compute the rate formula. So, rate is first order accurate, that is what we have proved theoretically also earlier. And this is we have plotted error versus in  $y$

axis we are plotting the error and enact axis we are plotting h. So, as from the figure also you can observe the same thing which we were earlier looking at the command window that as  $h \rightarrow 0$  so, basically we are moving to this side, our error is tending to 0.

So, if you want to drop your error further you have to workout for a smaller h, and N will be increased. So, then computation will also be increased. So that you have to decide for yourself that you want how much error you want, how much accuracy you want.

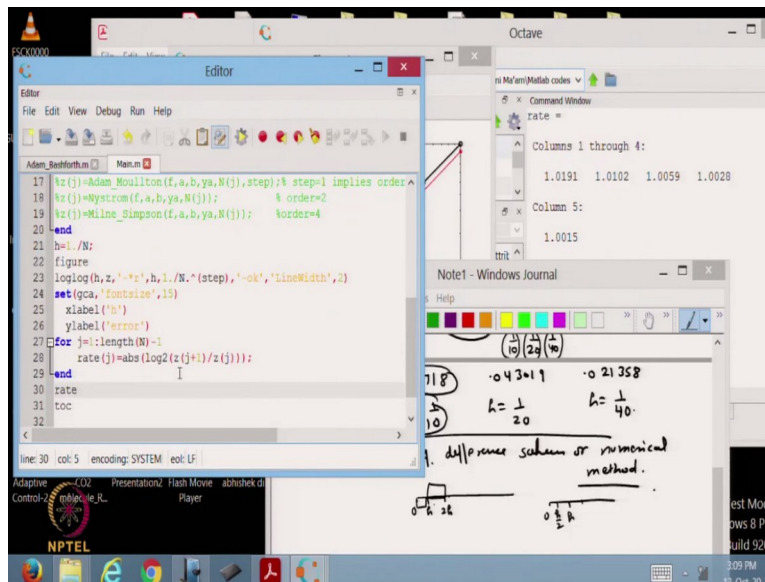
So, rate formula is also given in the following way which you can observe from here  $\text{rate}(j) = \frac{\log(z(j+1)/z(j))}{\log(2)}$ . z is that is the error at j+1 and means at the second.

(Refer Slide Time: 18:24)



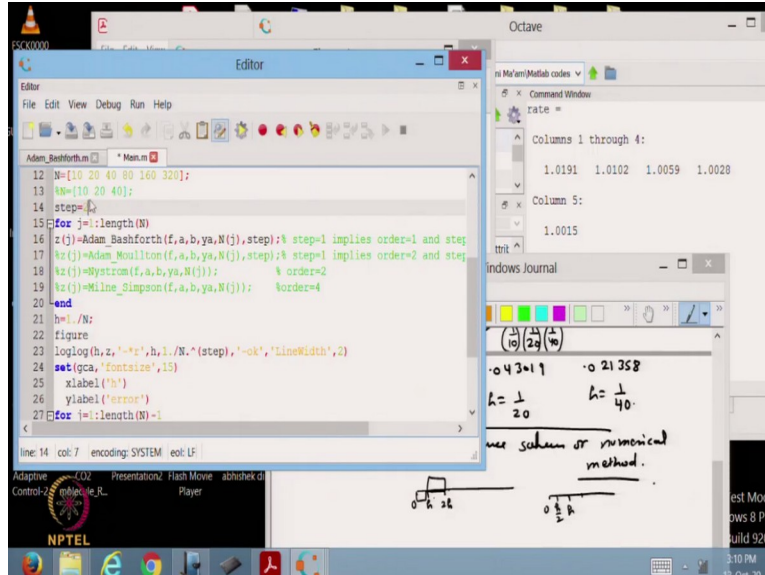
I mean to say that, when I am going from h and then 2h, so, we are basically dividing the error from this to this, this is h and then we are working with 0 to h/2, h when we are working with half.

(Refer Slide Time: 18:47)



Because we have saved the value of z corresponds to j for each h. So that is why here we are writing  $z(j+1)/z(j)$ .

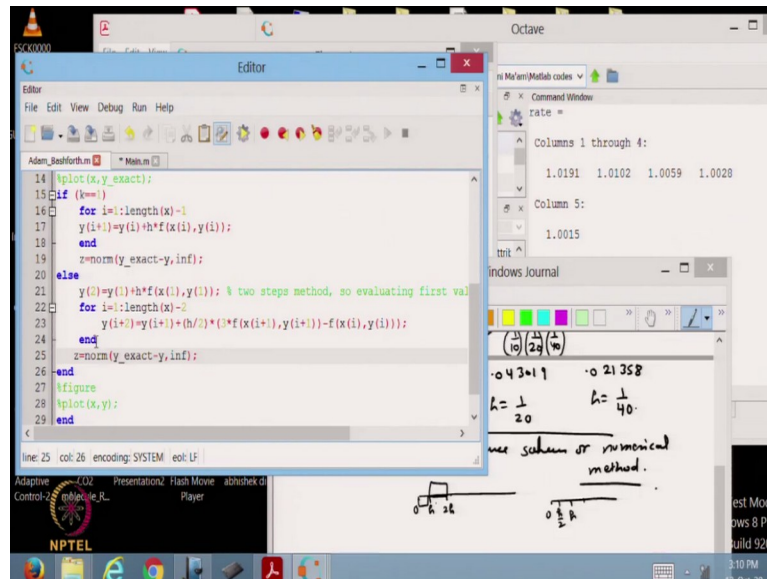
(Refer Slide Time: 19:12)



So now, if you remember, forward Euler was the first example or first category of a Adam Bashforth method, which corresponds to step is a first step method. And the order was also first order accurate. So we have verified this. Now, let us work out for a second category of a Adam Bashforth formula, which corresponds to step = 2.

Because it was a Two step method, and why I am choosing this step, correspondingly, I have implemented different algorithms for the step 1 is and for is step 2 in the functions, Adam Bashforth that is why I am taking this variable the step = 1 to choose forward Euler, step = 2 to choose another difference scheme which we will see in Adam Bashforth, because we are not calling it that by no that was just called as AB 2 method.

(Refer Slide Time: 20:22)



So, let me type out a step = 2 and then if you see if  $k = 1$ , so, the second choice if  $k = 1$  second choice  $k = 2$ .



(Refer Slide Time: 20:36)

Octave Editor window showing MATLAB code for solving a differential equation using the Runge-Kutta method. The code includes a function definition and a main script that calls the function.

```

14 function [y, x] = rk4(f, x0, y0, x_end, h);
15 if (k==1)
16     for i=1:length(x)-1
17         y(i+1)=y(i)+h*f(x(i),y(i));
18     end
19     z=norm(y_exact-y,inf);
20 else
21     y(i)=y(i)+h*f(x(i),y(i)); % two steps
22     for i=1:length(x)-2
23         y(i+2)=y(i)+(h/2)*(3*f(x(i+1),y(i+1))+f(x(i),y(i)));
24     end
25     z=norm(y_exact-y,inf);
26 end
27 figure
28 plot(x,y);
29 end

```

Windows Journal window showing handwritten calculations for the Runge-Kutta coefficients and a diagram of the method's steps.

Handwritten calculations:

$$z = 0.8718, \quad 0.4319, \quad 0.21358$$

$$h = \frac{1}{10}, \quad h = \frac{1}{20}, \quad h = \frac{1}{40}$$

Convergence of difference solution or numerical method.

Diagram showing the steps of the Runge-Kutta method:

$$y_{i+1} = y_i + h \left( \frac{1}{4}k_1 + \frac{3}{4}k_2 \right)$$

Adobe Acrobat Reader DC window showing a PDF document. The document contains text about the Runge-Kutta method, including the scheme definition and a diagram of the method's steps.

Then the scheme will be

$$y_{n+1} = y_n + h y'_n + \frac{h^2}{2} (y''_n - y'_n)$$

It is a two step explicit scheme of order 2.

Implicit

If we want to evaluate  $y(x)$  at  $x = x_n$  then we will write

Windows Journal window showing handwritten calculations for the Runge-Kutta coefficients and a diagram of the method's steps.

Handwritten calculations:

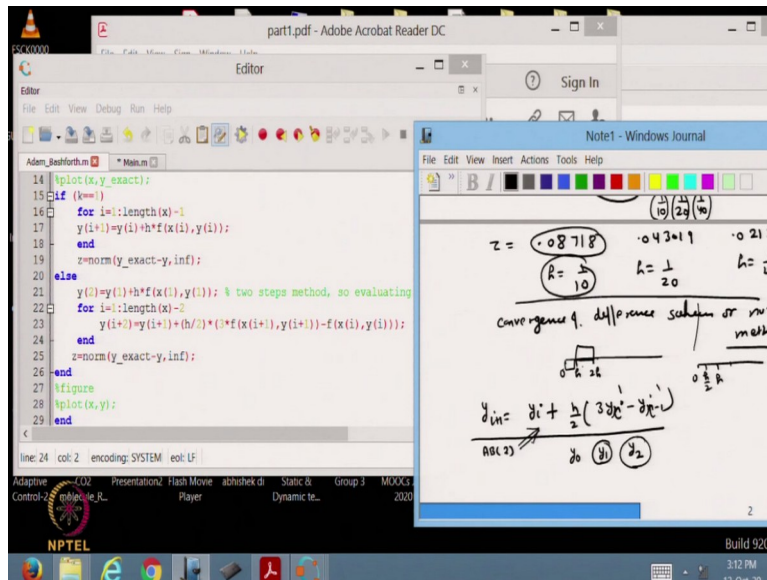
$$z = 0.8718, \quad 0.4319, \quad 0.21358$$

$$h = \frac{1}{10}, \quad h = \frac{1}{20}, \quad h = \frac{1}{40}$$

Convergence of difference solution or numerical method.

Diagram showing the steps of the Runge-Kutta method:

$$y_{i+1} = y_i + h \left( \frac{1}{4}k_1 + \frac{3}{4}k_2 \right)$$



So, for that reason, our difference scheme if you remember, let me type here also what was that or I can show it in our lectures also, this was the schemes So,

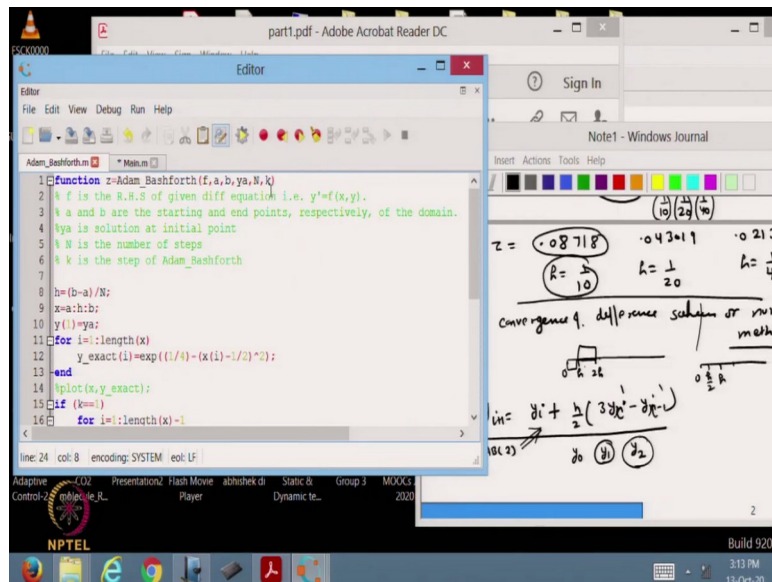
$$y_{i+n} = y_i + \frac{h}{2}(3y'_n - y'_{n-1})$$
 . This is the difference scheme AB(2). Of course, it is a Two step method. That is why I am choosing it to step = 2.

And one thing about this is that it is a multi step method, here I have written N it should also be i and it should also be i. It is a multi step method. So, that is why if I am starting with  $y_0$ , first I have to compute  $y_1$  with the help of some other single step method and then I can compute  $y_2$  with the help of this method, that is what I have implemented this numerically also. So initially from  $y_1$ , I am going to  $y_2$  with the help of the forward Euler method itself and from  $y_1$ , I am going to  $y_2$  with the help of AB(2).

The difference is the scheme of this AB(2) is given by the following which I have implemented here.  $y(i+2) = y(i+1) + (h/2) * (3 * f(x(i+1), y(i+1)) - f(x(i), y(i)))$ .

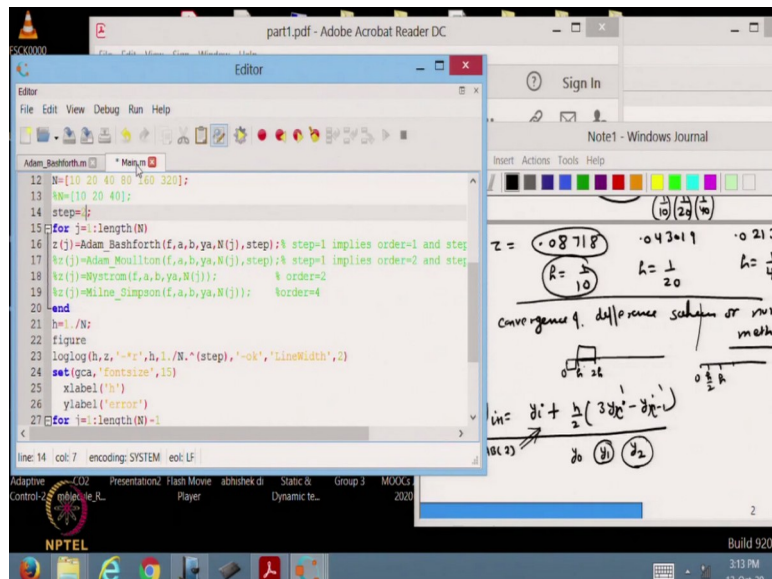


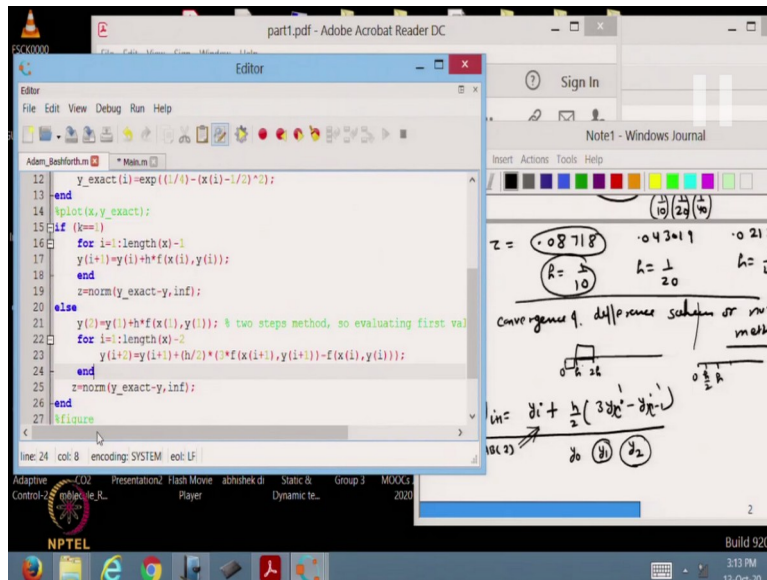
(Refer Slide Time: 22:32)



So, if I will give the value of step which is basically k here, so, that is just a different name.

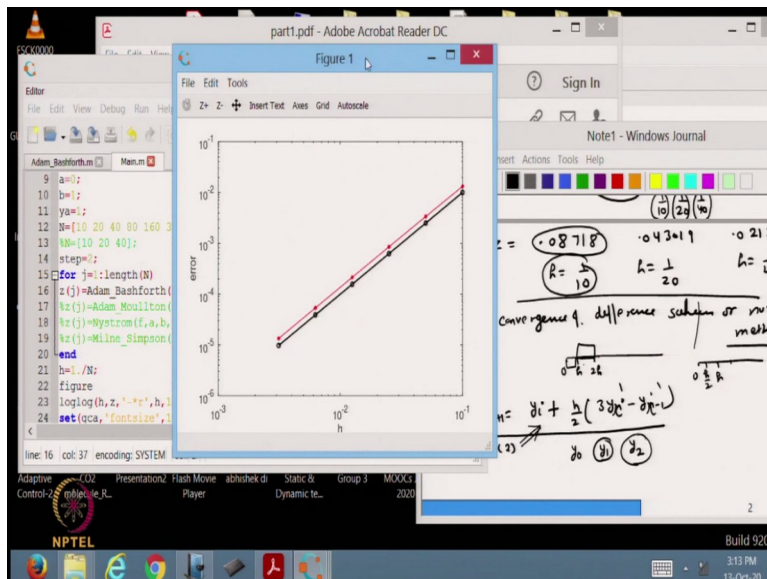
(Refer Slide Time: 22:42)

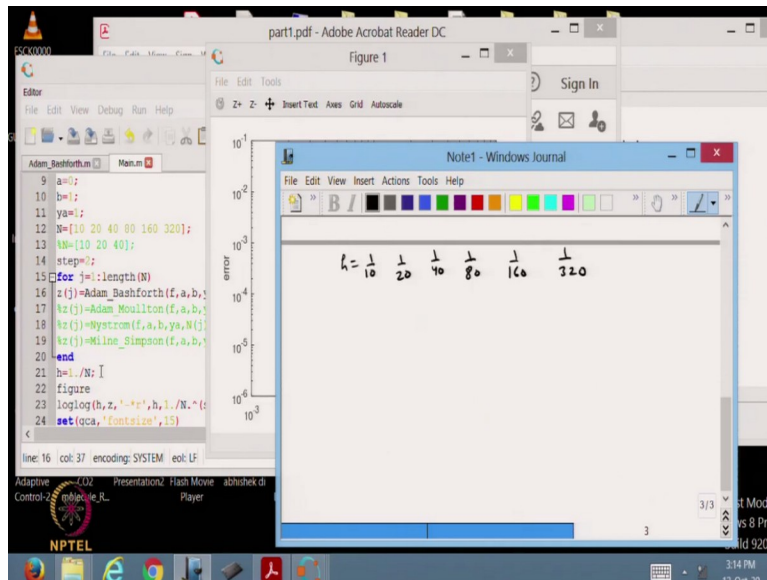




So, the value which I will pass with the help of mean in each step will be taken by the variable k and if I will give that as a k= 2. So, these codes which, which algorithm which I have written in else part will get executed, which you can observe now. Because I am running a mean program with the variable step = 2.

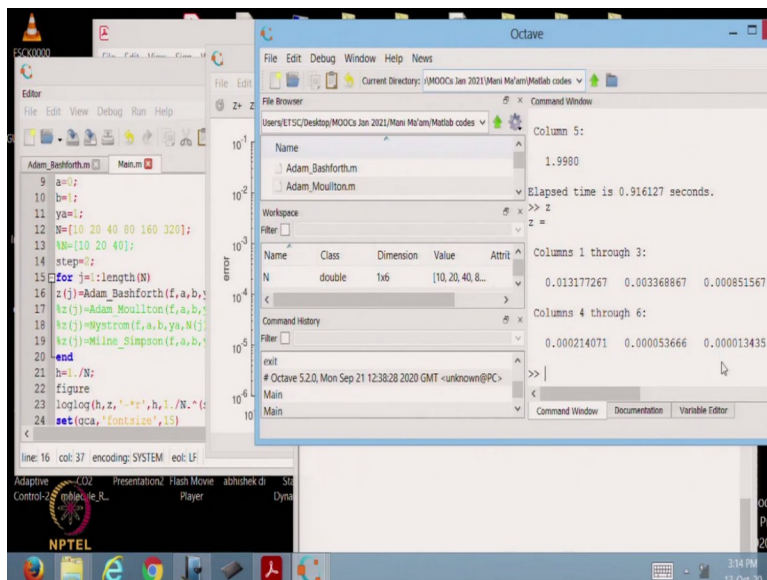
(Refer Slide Time: 23:10)

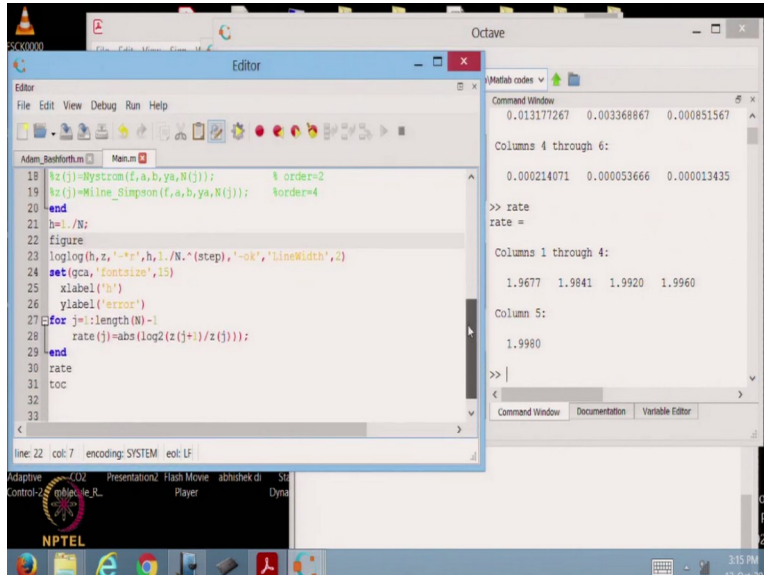
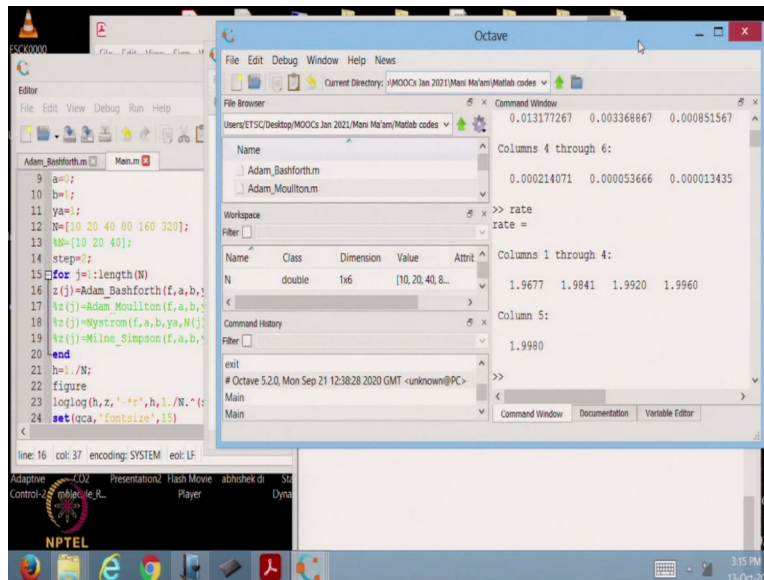




So, that is again I have worked out for different  $h$ ,  $h = 1/10, 1/20, 1/40, 1/80, 1/160, 1/320$ , because you can see the array of  $N$  correspondingly  $h$  is basically  $1/N$  which we can observe from the program.

(Refer Slide Time: 23:49)





So, what will be the error which you can also see from here after typing `z`. So, you can see how much accuracy we could achieve by increasing  $N$  or decreasing  $h$ . So, that is called the convergence of AB(2) which is an Adams Bashforth method of order 2. Order 2 also we can verify numerically just by looking at the rate which we are also computing in the following program. So, the rate is also close to 2 as expected. Rate is also close to 2 as expected which we are computing here.

But the only thing is we AB(2) is a Two step method. So, we have, it is not a self starting method because from going from  $y_0$  to  $y_1$  I have to use some other single step method. And for that

reason, I am choosing the forward Euler method here locally forward Euler is  $O(h^2)$ , that is why total error will be  $O(h^2)$  as well. Because when you are working with a multi step method, you have to be really careful how you make it because these methods are not self starting.

So, you have to be very careful in choosing the method when you are going for some initial approximation like how you are going from  $y_0$  to  $y_1$  and then if it is more than 2 steps then you have to compute  $y_2$  also because you can start from  $y_3$  that is the nature of a multi-step algorithm etcetera.

(Refer Slide Time: 25:57)

```

11 ya=1;
12 N=[10 20 40 80 160 320];
13 hN=[10 20 40];
14 step=2;
15 for j=1:length(N)
16     z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17     kz(j)=Adams_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18     kz(j)=Runge_Kutta(f,a,b,ya,N(j)); % order=4
19     kz(j)=Milne_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-s',h,1./N.^(step),'-o','LineWidth',2)
24 set(gca,'FontSize',15)
25 xlabel('h')
26 ylabel('error')
27 for j=1:length(N)-1

```

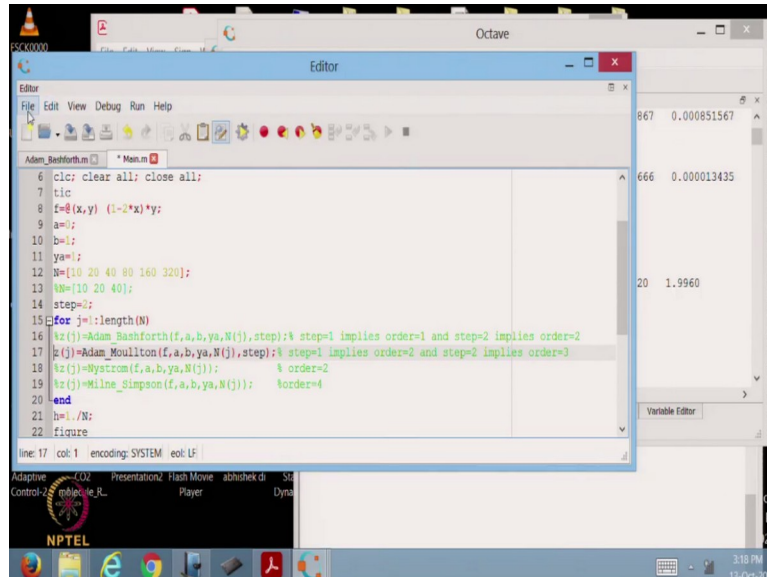
So, of course, I have implemented just two different algorithms. Adam Bashforth corresponds to step = 1 and corresponds to step = 2, corresponds to step = 1 it is an order 1 method corresponds to step = 2 it is order 2 method. But, if you want you can also implement other high order methods under this category, but as far as this code is concerned I have just implemented for k = 1 and k = 2. So, it should be very much clear to you.

Because you cannot give arbitrary value of a step here is step = 3 then also AB(2) will run because that thing I have given in the else part here if you see so, it will run for any value of k, but the algorithm which we are implementing for any value of k is just AB(2). So, you have to modify this code if you want high order differences schemes under Adam Bashforth category.



So, basically we have seen one method of order 1 and another method of order 2 which was a Two step method and how to implement that Two step method that also we have seen.

(Refer Slide Time: 27:28)



```
6 clc; clear all; close all;
7 tic
8 f=@(x,y) (1-x)*y;
9 a=0;
10 b=1;
11 ya=1;
12 N=[10 20 40 80 160 320];
13 h0=[10 20 40];
14 step=2;
15 for j=1:length(N)
16     hz(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17     hz(j)=Adam_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18     hz(j)=Rungutta(f,a,b,ya,N(j)); % order=2
19     hz(j)=Milne_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
```

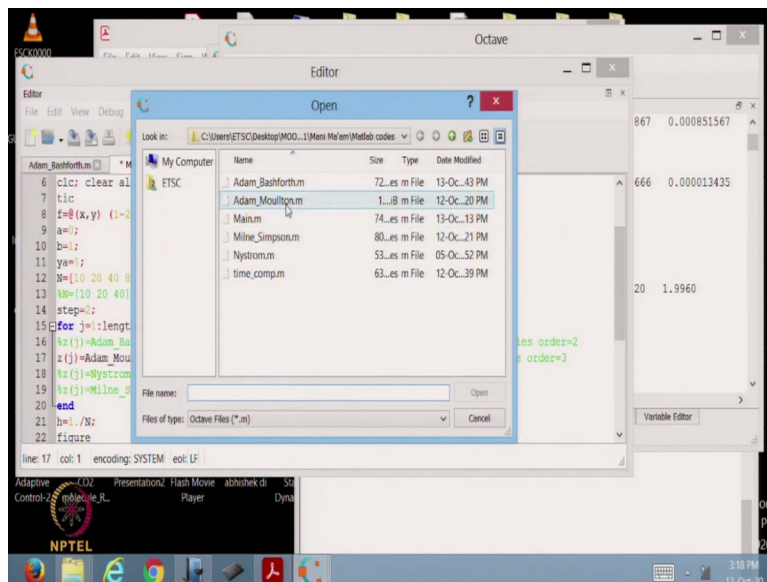
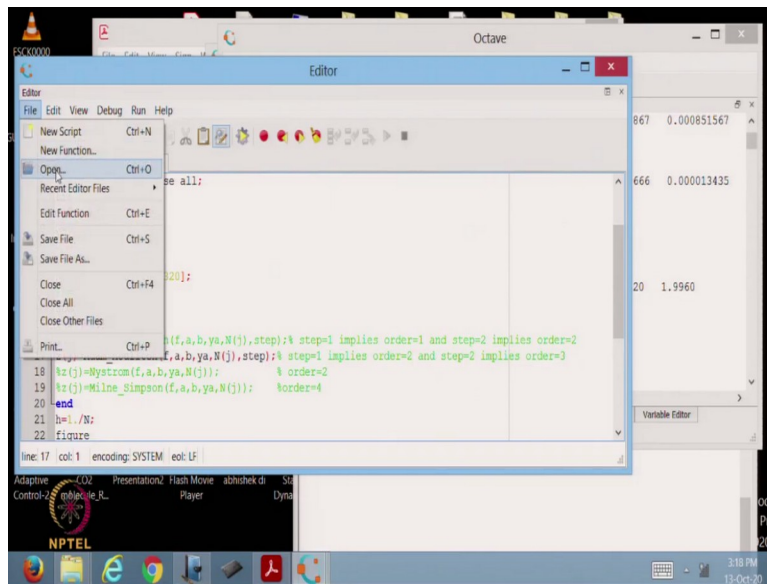
Variable Editor

Variable	Value
hz	0.000851567
hz	0.000013435
hz	1.9960

Now, we will be looking at Adam Moulton's formula or Adam Moulton's methods So, for that reason I am commenting on Adam Bashforth and I am opening the comment part of Adams Moulton. So, here again, I have written the different name of the function which is Adam Moulton; the argument of this function again remains the same as I have already discussed it an Adam Bashforth and the output is also error which is stored in the variable z.



(Refer Slide Time: 28:04)



(Refer Slide Time: 28:09)

```

1 function z=Adam_Moulton(f,a,b,ya,N,k)
2 %syms w;
3 %f is the R.H.S of given diff equation i.e. y'=f(x,y).
4 %a and b are the starting and end points, respectively, of the domain.
5 %ya is solution at initial point
6 %N is the number of steps
7 %k is the step of Adam_Moulton
8 h=(b-a)/N;
9 x=a:h:b;
10 y(1)=ya;
11 for i=1:length(x)
12     y_exact(i)=exp((1/4)-(x(i)-1/2)^2);
13 end
14 if (k==1)
15     for i=1:length(x)-1
16         y(i+1)=(1/(1-(h/2)*(1-2*x(i+1))))*(y(i)+(h/2)*f(x(i),y(i))); % implicit method
17         legm=(h/2)*f(x(i+1),y)+f(x(i),y(i))-y(i);
18     end
19 else
20     z=norm(y_exact-y,inf);
21 end
22 %y(2)=y(1)+h*f(x(1),y(1)); using TS(1)
23 %y(2)=y(1)+h*f(x(1),y(1))+h^2/2*(-1*y(1)+(1-2*x(1))^2)*y(1); using TS(2)

```

Variable Editor

Variable	Value
x	0.000851567
y	0.000013435
y_exact	1.9960

```

7 %k is the step of Adam_Moulton
8 h=(b-a)/N;
9 x=a:h:b;
10 y(1)=ya;
11 for i=1:length(x)
12     y_exact(i)=exp((1/4)-(x(i)-1/2)^2);
13 end
14 if (k==1)
15     for i=1:length(x)-1
16         y(i+1)=(1/(1-(h/2)*(1-2*x(i+1))))*(y(i)+(h/2)*f(x(i),y(i))); % implicit method
17         legm=(h/2)*f(x(i+1),y)+f(x(i),y(i))-y(i);
18         y(i+1)=solve(eqn,w);
19     end
20 else
21     z=norm(y_exact-y,inf);
22 end
23 %y(2)=y(1)+h*f(x(1),y(1)); using TS(1)
24 %y(2)=y(1)+h*f(x(1),y(1))+h^2/2*(-1*y(1)+(1-2*x(1))^2)*y(1); using TS(2)

```

Variable Editor

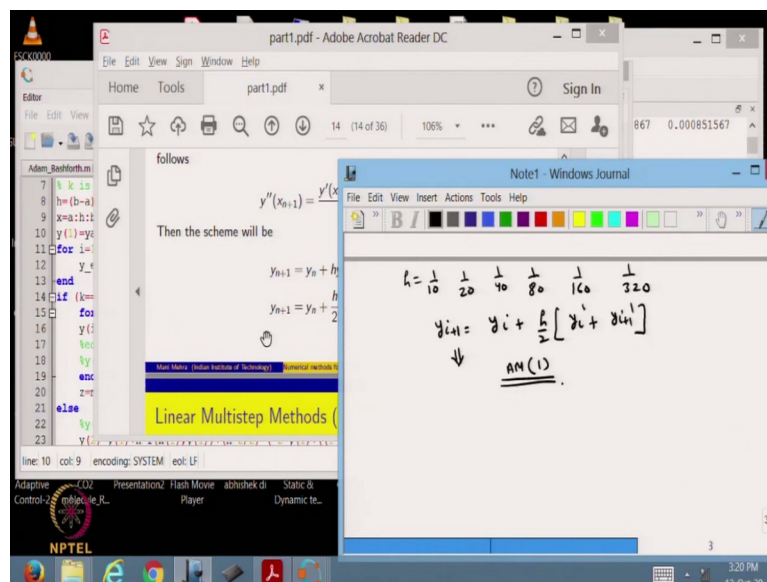
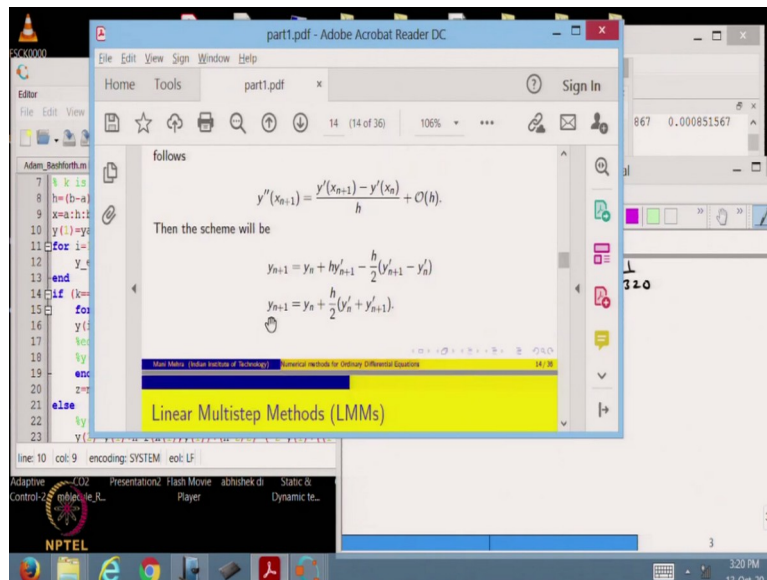
Variable	Value
x	0.000851567
y	0.000013435
y_exact	1.9960

So, now, I am opening the function Adams Moulton's again what are the meanings of these arguments that also I have explained in the comment sections. Of course, since w you can ignore because since does not since is the only thing which I have observed, it does not work in the older version of the Octave, I do not know whether it runs in the latest version of Octave I have not tried, but other things are Okay.

f is the right hand side of a given differential equation a and b are the starting and end points ya is the solution at initial points which we were choosing 1, again N is the number of steps and k is that step off again here it should be not k is the step of Adams Moulton. h is again defined in the

same way,  $x$  is also the array, and  $y1$  is  $y_a$  again,  $y$  exact will remain the same, because we are solving the same Initial Value Problem, I do not need to repeat that stuff again. And corresponds to  $step = 1$  which is  $k = 1$  corresponds to the Trapezoidal method.

(Refer Slide Time: 29:37)



So, the algorithm of the Trapezoidal method we already know. I can also show you in our lecture

part, so basically,  $y_{i+1} = y_i + \frac{h}{2} (y'_i + y'_{i+1})$ . So, which is also called the Trapezoidal method or AM(1). AM(1) first method of this category. It is again a single step method, but if

you compare this with the other methods, this is an implicit method. This is the implicit method that I have already explained in our lecture classes, why it is an implicit method?

(Refer Slide Time: 30:42)

```

9 x=a:h:b;
10 y(1)=ya;
11 for i=1:length(x)
12     y_exact(i)=exp((1/4)-(x(i)-1/2)^2);
13 end
14 if (k==1)
15     for i=1:length(x)-1
16         y(i+1)=(1/(1-(h/2)*(1-2*x(i+1))))*(y(i)+(h/2)*f(x(i),y(i))); % implicit method
17         eqn=-h/2*(f(x(i+1),y(i+1))-f(x(i),y(i)))-y(i);
18         [y(i+1)]=solve(eqn,w);
19     end
20     z=norm(y_exact-y,inf);
21 else
22     [y(2)]=y(1)+h*f(x(1),y(1)); using TS(1)
23     [y(2)]=y(1)+h*f(x(1),y(1))+(h^2/2)*(-2*y(1)+(1-2*x(1))^2)*y(1); using TS(2)
24     [y(2)]=y(1)+h*f(x(1),y(1))+(h^2/2)*(-2*y(1)+(1-2*x(1))^2)*y(1)+...
25         +(h^3/6)*(-6*(1-2*x(1))*y(1)+(1-2*x(1))^3*y(1)); % using TS(3)

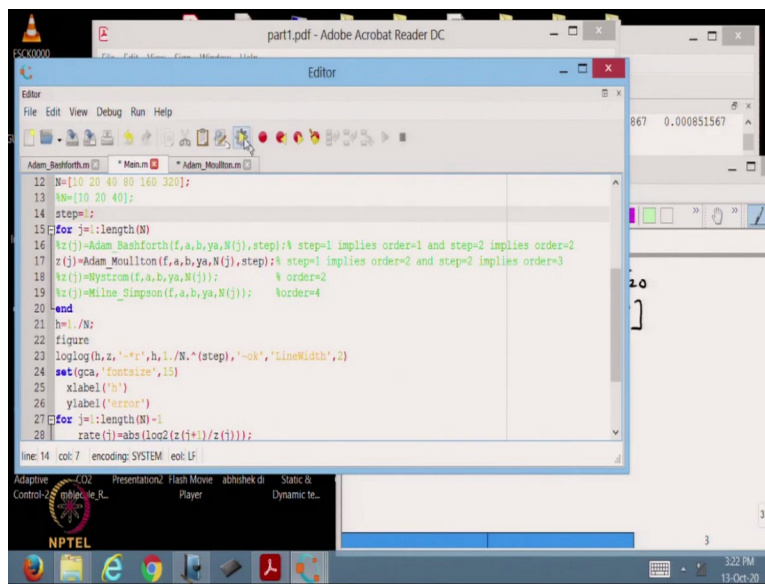
```

So, for  $k = 1$  again the same algorithm I have implemented here, which you can observe from here

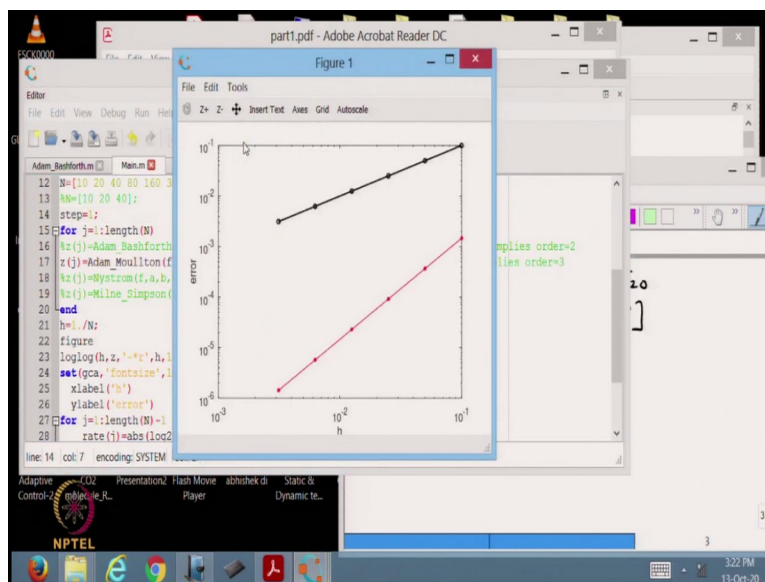
$$y_{i+1} = (1/(1 - (h/2) * (1 - 2 * x(i + 1)))) * (y(i) + (h/2) * f(x(i), y(i)));$$

then again, the same algorithm which I have written here, I have been implemented here. And then we are computing that error again there will be a difference of exact solution and numerical solution we are computing the norm infinity norm basically.

(Refer Slide Time: 31:25)

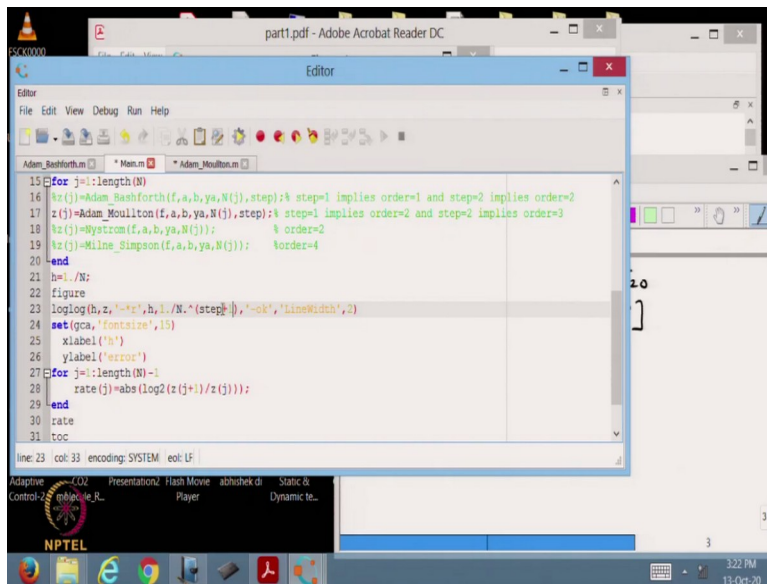


```
12 N=[10 20 40 80 160 320];
13 h=1./N;
14 step=1;
15 for j=1:length(N)
16     %z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17     %z(j)=Adam_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18     %z(j)=Symstr(f,a,b,ya,N(j)); % order=2
19     %z(j)=Milne_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-r',h,1./N,'-b',h,1./N,'LineWidth',2)
24 set(gca,'fontsize',15)
25 xlabel('h')
26 ylabel('error')
27 for j=1:length(N)-1
28     rate(j)=abs(log2(z(j+1)/z(j)));
```



So, now, if we want to see the output, we have to come to the main, so, again here I have to write a step = 1, and now I have to run the code. So, let us see how it behaves.

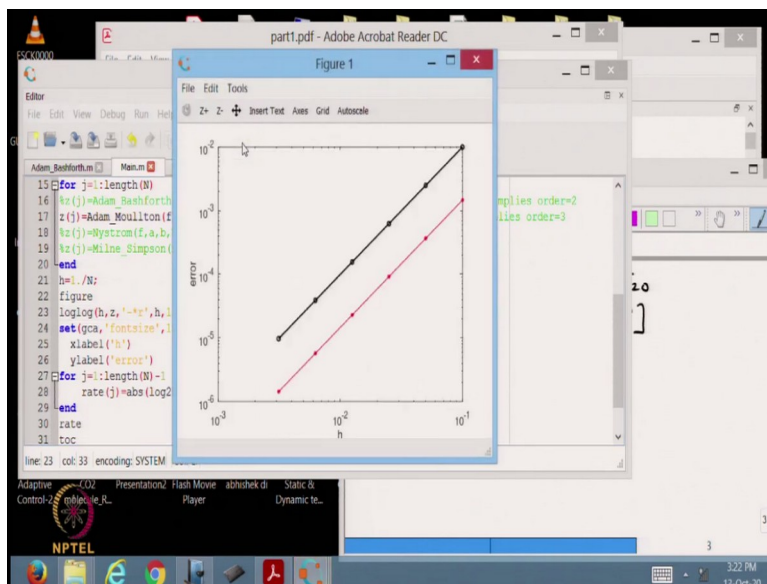
(Refer Slide Time: 31:45)



The image shows a MATLAB Editor window with the following code:

```
15 for j=1:length(N)
16     z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17     z(j)=Adam_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18     z(j)=Wystrom(f,a,b,ya,N(j)); % order=2
19     z(j)=Milne_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-o',h,1./N.*(step-1),'-o','LineWidth',2)
24 set(gca,'fontSize',15)
25 xlabel('h')
26 ylabel('error')
27 for j=1:length(N)-1
28     rate(j)=abs(log2(z(j+1)/z(j)));
29 end
30 rate
31 toc
```

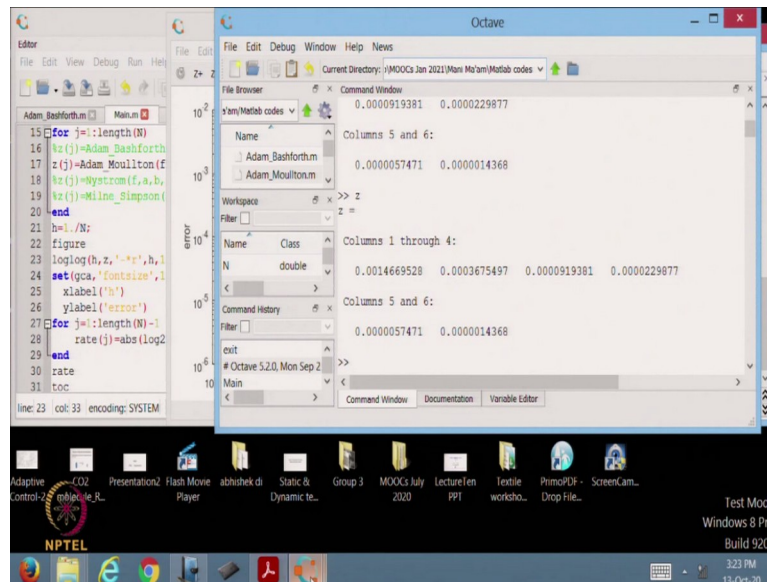
The code implements four numerical methods: Adam-Bashforth, Adam-Moulton, Wystrom, and Milne-Simpson. It then plots the error versus step size  $h$  on a log-log scale. The plot shows that the error decreases as  $h$  decreases, with the slope indicating the order of the method.



Of course, corresponds, so, what I am.

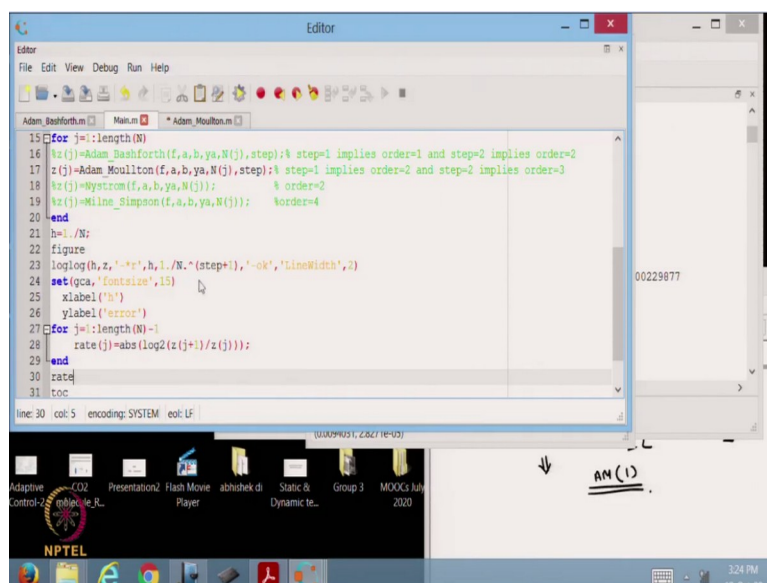


(Refer Slide Time: 32:03)



So, here again let me type out the error  $z$  error is stored in function  $z$ , so, let me typed out that you can observe that  $z$  is so, you can see so, again the  $z$  stores the error corresponds to different value of  $h$ , which I have already mentioned here in the these are the values of  $z$  corresponds to which errors are here. So, again you can verify that error is tending to 0 as  $h \rightarrow 0$  which you can observe from the graph also. So, basically in the last figure also I forgot to mention what this black line represents and what this red line represents.

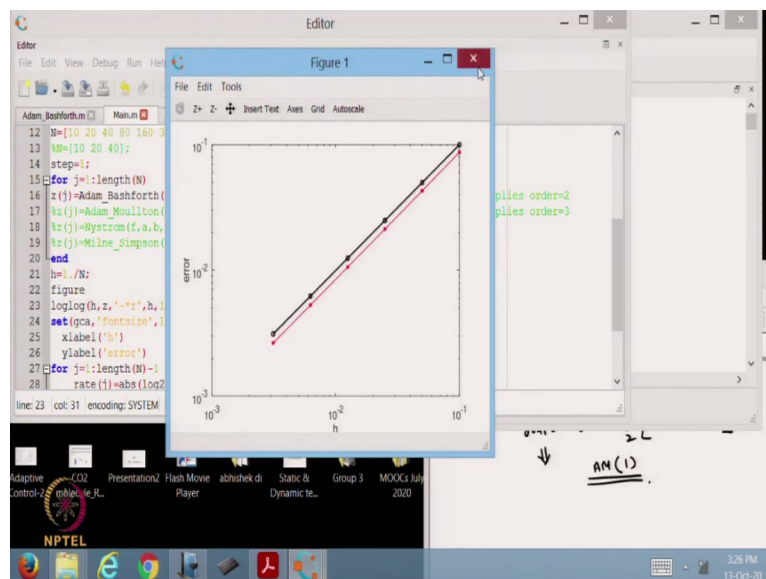
(Refer Slide Time: 32:59)



So, as you can see that here we are in the log plot red line is basically the error actual error and a here with the black is this is the order we are plotting. So, this you can see from here, red is the actual error, red is the actual error and this is basically  $O(h^2)$  line in case of a second order method or  $O(h)$  line in case of a first order method. Similarly, you can superimpose this line depending on the order of the method which you can see from here, we are plotting  $\log(h, z)$  and then with the red lines and then  $h$  again this is basically  $O(h^2)$  That is why when I am taking a step = 1.

In the case of an Adams Moulton this corresponds to order is equal to 2. So, that is why I am plotting the line  $O(h^2)$ . So, both are parallel, it verifies that our method is second order accurate. Similarly, as I told you, I forgot to mention in the case of Adam Bashforth which I can show it again. So just by commenting this line and uncommenting the previous one and here because it is a step = 1 in case of Adam Bashforth it is a first order method. So the line should be parallel to the line  $h$ .

(Refer Slide Time: 34:54)



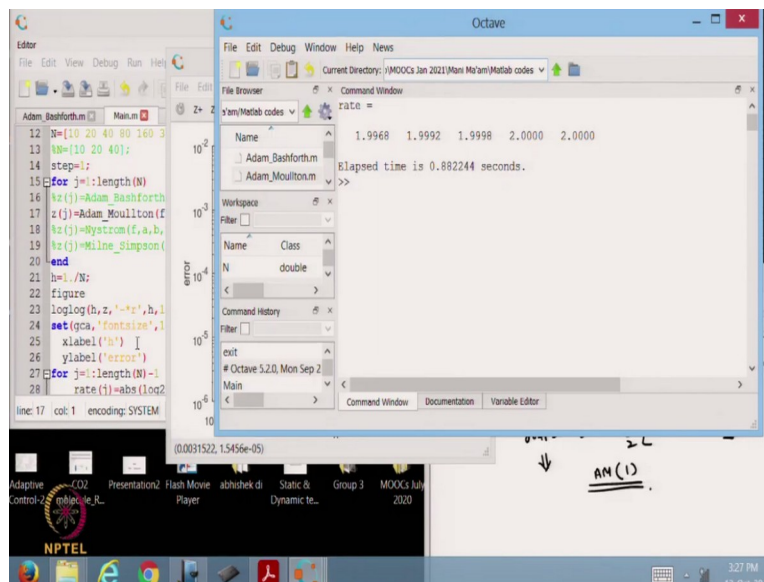
So, that is what you can see.

(Refer Slide Time: 34:58)

```

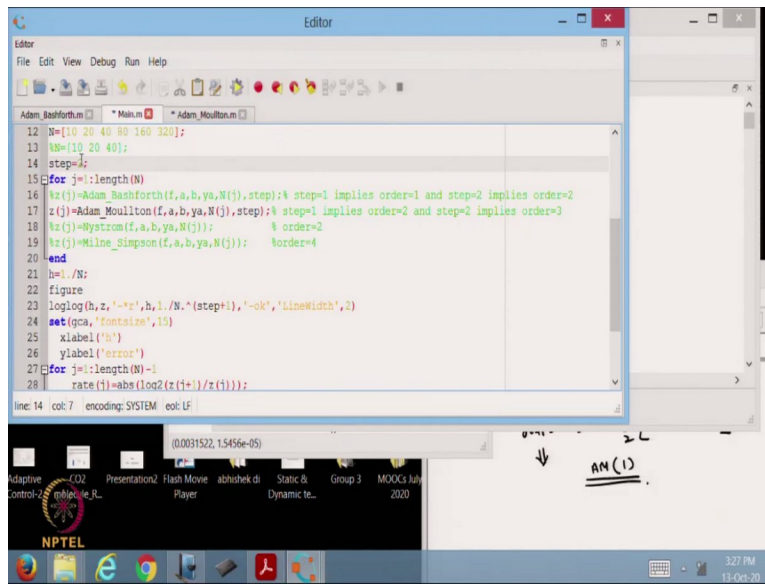
12 N=[10 20 40 80 160 320];
13 h0=[10 20 40];
14 step=1;
15 for j=1:length(N)
16     z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17     z(j)=Adam_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18     z(j)=Wylstrom(f,a,b,ya,N(j)); % order=2
19     z(j)=Milne_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-r',h,1./N.*(step+1),'-ok','LineWidth',2)
24 set(gca,'FontSize',15)
25 xlabel('h')
26 ylabel('error')
27 for j=length(N)-1
28     rate(j)=abs(log2(z(j+1)/z(j)));

```



And in fact, that is why here I have returned  $step+1$  when I was choosing the Adams Moulton formula. So, because in case of a first order line it should be order of  $h$  in case of a second order line which it should be  $O(h^2)$ . So, both the lines should be parallel, that is how we can verify the order of the method also numerically. So, this is the right way to observe graphically. Otherwise, if you want you can also see by typing  $z$

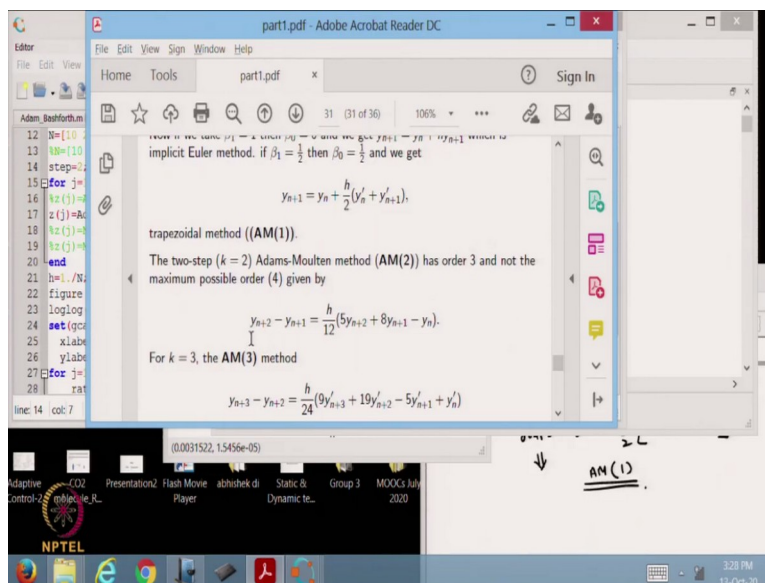
(Refer Slide Time: 35:42)



The MATLAB Editor window displays the following code:

```
12 N=[10 20 40 80 160 320];
13 h0=[10 20 40];
14 step=1;
15 for j=1:length(N)
16     %z(j)=Adams_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17     z(j)=Adams_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18     %z(j)=Rungstron(f,a,b,ya,N(j)); % order=2
19     %z(j)=Milne_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-xr',h,1./N.*(step+1),'-ok','LineWidth',2)
24 set(gca,'FontSize',15)
25 xlabel('h')
26 ylabel('error')
27 for j=1:length(N)-1
28     rate(j)=abs(log2(z(j+1)/z(j)));
```

The plot area shows a log-log graph with a title bar indicating coordinates (0.0031522, 1.5456e-05). The x-axis is labeled 'h' and the y-axis is labeled 'error'. The plot shows a series of points connected by lines, with a downward arrow pointing to the label AM(1).



The Adobe Acrobat Reader DC window displays a PDF document titled 'part1.pdf'. The document content includes:

implicit Euler method. if  $\beta_1 = \frac{1}{2}$  then  $\beta_0 = \frac{1}{2}$  and we get

$$y_{n+1} = y_n + \frac{h}{2}(y'_n + y'_{n+1}),$$

trapezoidal method ((AM(1))).

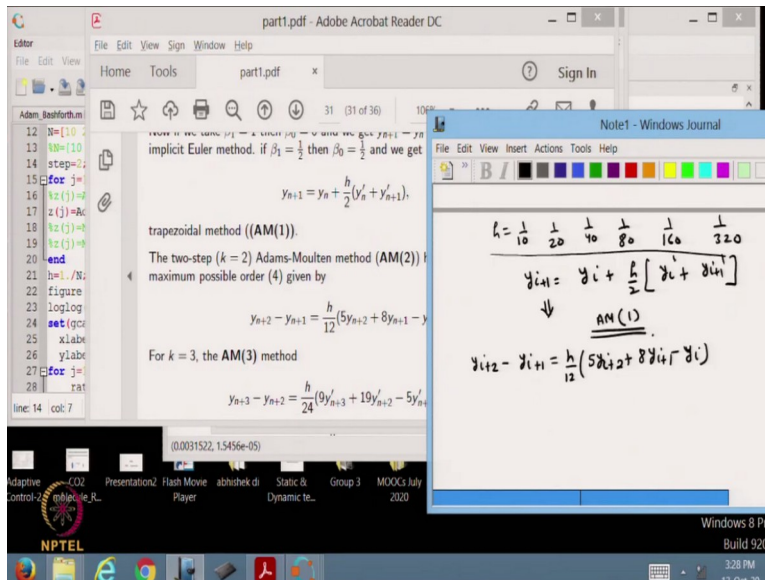
The two-step ( $k=2$ ) Adams-Moulton method (AM(2)) has order 3 and not the maximum possible order (4) given by

$$y_{n+2} - y_{n+1} = \frac{h}{12}(5y'_{n+2} + 8y'_{n+1} - y'_n).$$

For  $k=3$ , the AM(3) method

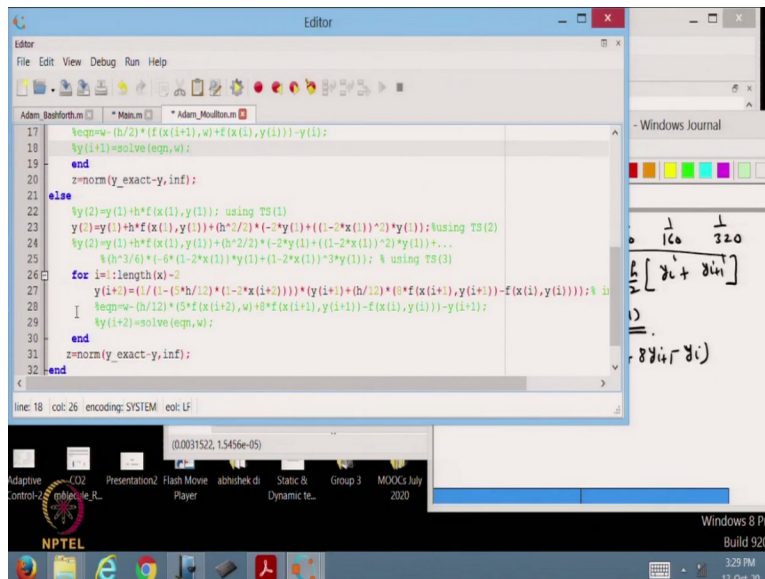
$$y_{n+3} - y_{n+2} = \frac{h}{24}(9y'_{n+3} + 19y'_{n+2} - 5y'_{n+1} + y'_n)$$

The plot area shows a log-log graph with a title bar indicating coordinates (0.0031522, 1.5456e-05). The x-axis is labeled 'h' and the y-axis is labeled 'error'. The plot shows a series of points connected by lines, with a downward arrow pointing to the label AM(1).



And then let me change this a step size to 2 and then basically if you remember which algorithm we will be implementing because that is we can see from here yes Adams Moulton family so, the AM(1) is this which is a Trapezoidal which we have already seen and AM(2) is this basically which let me also write down here - $y_i$ .

(Refer Slide Time: 36:50)



So, now, you can check that whether we have implemented the same differences scheme called again in this case I have implemented two differences scheme corresponds to  $k$  is equal to 1 and corresponds to  $k$  is equal to 2. If you want it to go for higher or other method you can modify these codes. So,  $y_2$  is basically this this is the scheme's.



(Refer Slide Time: 37:17)

```

17 legm=(h/2)*f(x(i+1),w)+f(x(i),y(i))-y(i);
18 %y(i+1)=solve(eqn,w);
19 end
20 z=norm(y_exact-y,inf);
21 else
22 %y(2)=y(1)+h*f(x(1),y(1)); using TS(1)
23 y(2)=y(1)+h*f(x(1),y(1))+h^2/2*(-2*y(1)+(1-2*x(1))^2)*y(1); %using TS(2)
24 %y(2)=y(1)+h*f(x(1),y(1))+h^2/2*(-2*y(1)+(1-2*x(1))^2)*y(1)+...
25 %h^3/6*(-6*(1-2*x(1))*y(1)+(1-2*x(1))^3*y(1)); % using TS(3)
26 for i=length(x)-2
27 y(i+2)=(1/(1-(5*h/12)*(1-2*x(i+2))))*(y(i+1)+(h/12)*(8*f(x(i+1),y(i+1))-f(x(i),y(i))-y(i+1))); % i
28 legm=(h/12)*(5*f(x(i+2),w)+8*f(x(i+1),y(i+1))-f(x(i),y(i))-y(i+1)); % i
29 %y(i+2)=solve(eqn,w);
30 end
31 z=norm(y_exact-y,inf);
32 end

```

$$h = \frac{1}{10}, \frac{1}{20}, \frac{1}{40}, \frac{1}{80}, \frac{1}{160}, \frac{1}{320}$$

$$y_{i+1} = y_i + \frac{h}{2} [y'_i + y''_i]$$

$$\downarrow$$

$$y_{i+2} - y_{i+1} = \frac{h}{12} (5y_{i+2} + 8y_{i+1} - y_i)$$

$$y(i) \rightarrow y'_i, y''_i, y'''_i \text{ onwards}$$

$$\frac{d}{dx} \rightarrow \frac{d}{dt} \rightarrow \frac{d}{dt}$$

So, again if you observe this method, this is again

$$y_{i+2} - y_{i+1} = \frac{h}{2} (5y_{i+2} + 8y_{i+1} - y_i)$$

So, this is again not a single method.

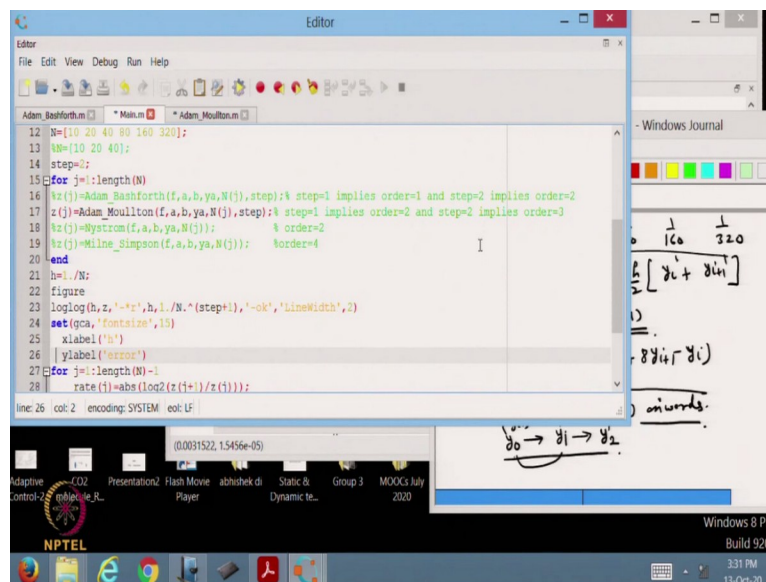
This is also a Two step method. So, first we have to compute  $y_2$ , so, that  $y_2$  we are predicting with the help of TS 2 method Taylor series method of order 2 that we have also mentioned in the commented part that how we are computing  $y_2$  using TS 2 method. So, we have implemented the



algorithm for TS 2 as well to predict  $y_2$  because  $y_2$  corresponds to basically  $y_1$  if we start from  $y_0$  initial conditions if we think in that way.

And then once  $y_1$  is known to us we can compute  $y_2$  but actually indexing in Matlab does not start from 0 that is why  $y_0$  corresponds to  $y_1$  and  $y_1$  corresponds to  $y_2$ . And from here onwards I think it will start  $y_3$  onwards. So, this is again, I have made it clear how we have gone from  $y_0$  to  $y_1$  with the help of TS 2 method. And now the reason should also be clear to you why we are choosing TS 2 method not to forward Euler methods. Because, so that the total order of the method should be consistent. If I am choosing a low order method to predict  $y_1$ , that error will dominate in the final part as well. So that is why to have a consistency in the order of the method I am choosing it with TS 2.

(Refer Slide Time: 39:29)

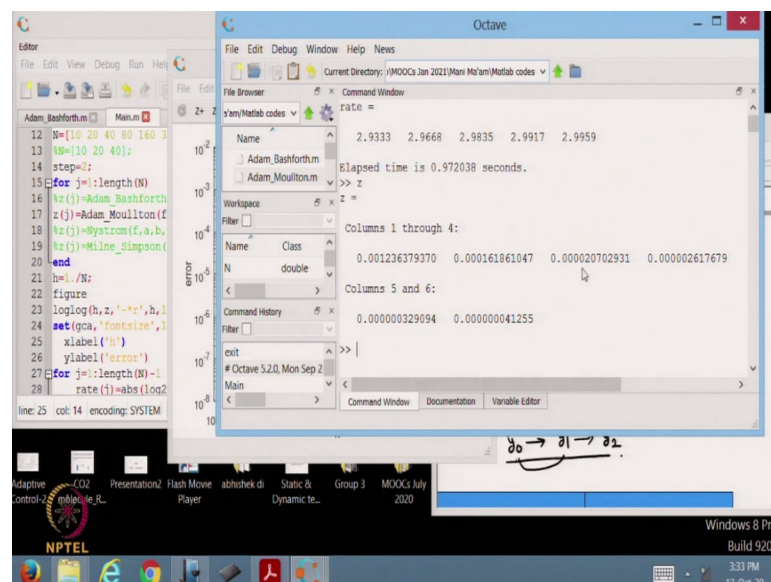
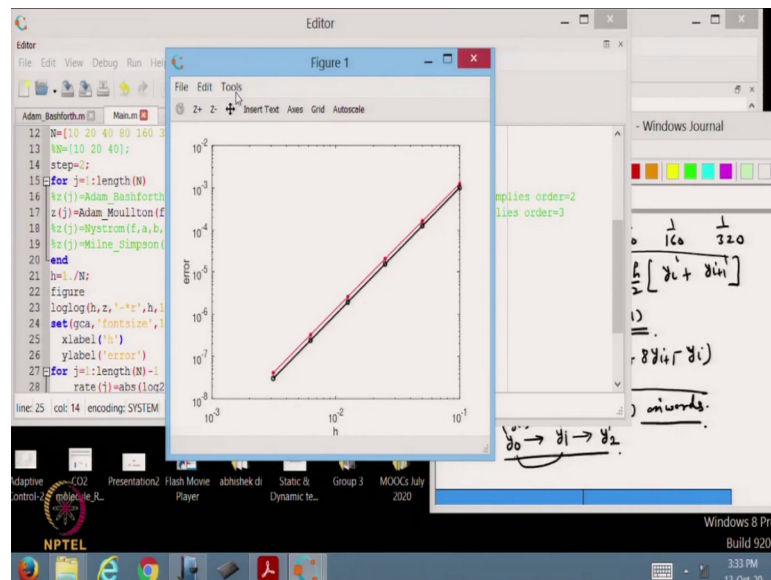


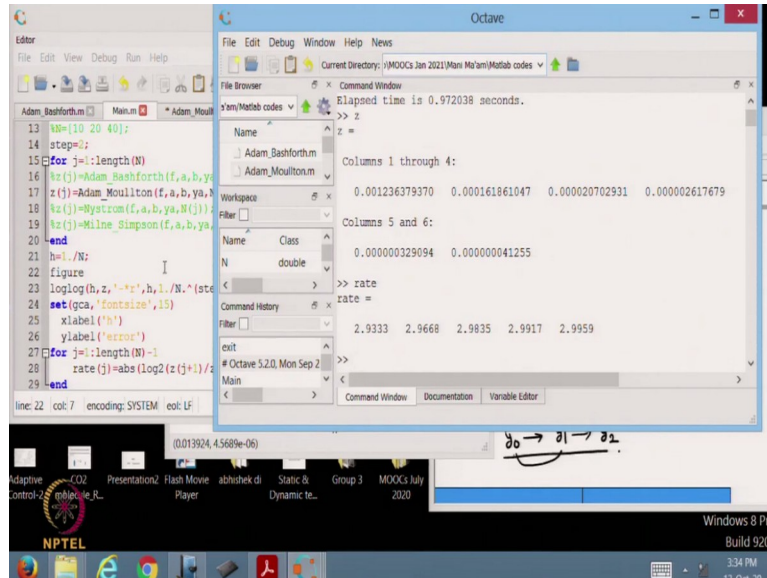
So because as you can see, step 2 implies order 3, that is why we are choosing TS 2 otherwise there will be inconsistency if we choose low order method and that error will be dominated then there is no point because if there is a low order accurate method in predicting the value of  $y_1$  and that error will dominate in rest of the computations. So, there is no advantage if we are choosing a higher order method for subsequent values. So, that is why initially I also said that in multi step methods you have to be very careful how you choose the initial approximation.

Of course, that initial approximation should also be with the consistent method and that is why in the previous when I was taking a step 1 I have chosen Two step method we have seen in case of

a Adam Bashforth and then we have seen forward Euler and now we are predicting with the TS 2. Why there is a difference in choosing forward Euler in case of Adam Bashforth of step 2, and in case of Adam Moulton of step 2, because this is order 2 method and this is order 3 method. So, now, if I then correspond to step = 2, of course, here what is the expected order is 3, that is what we are also plotting is step+1 is step= 2, so, both the lines should be parallel that is as expected.

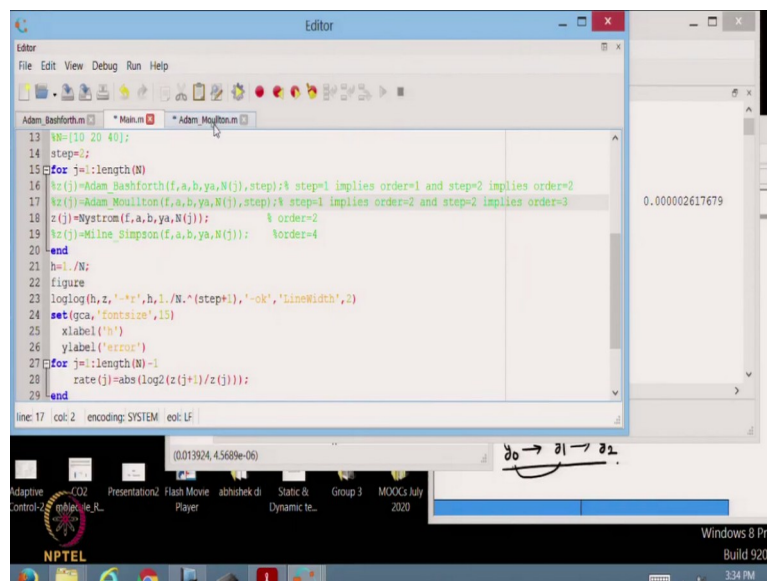
(Refer Slide Time: 41:16)





So that is what you can see here, you can if you want you can also type  $z$  error is tending to 0 as  $h \rightarrow 0$ . If you want to see the or you want to observe the same thing graphically, the graph is also in front of you again. The red line is the actual error and black line corresponds to the line or  $h^3$  because this is an order 3 method. So that is what we are doing here, if you wanted to see the rate, rate also which is close to 3. So, in this case also we have verify that our method is third order accurate.

(Refer Slide Time: 42:14)



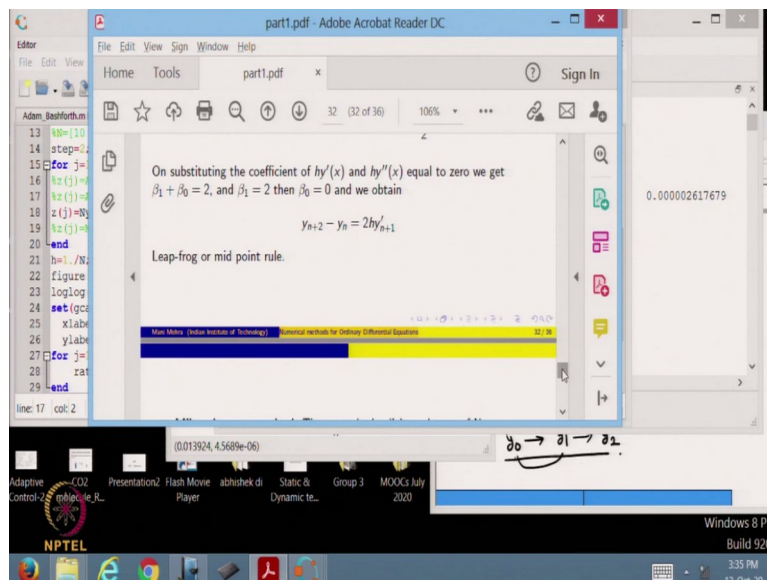
Now, we are going to the third category of the method which we have learned in previous lectures. I am commenting on Adam Bashforth, Adam Moulton's because I am explaining to you all these different categories with the help of the same main program, that is why I am doing the same thing.

(Refer Slide Time: 42:48)

```

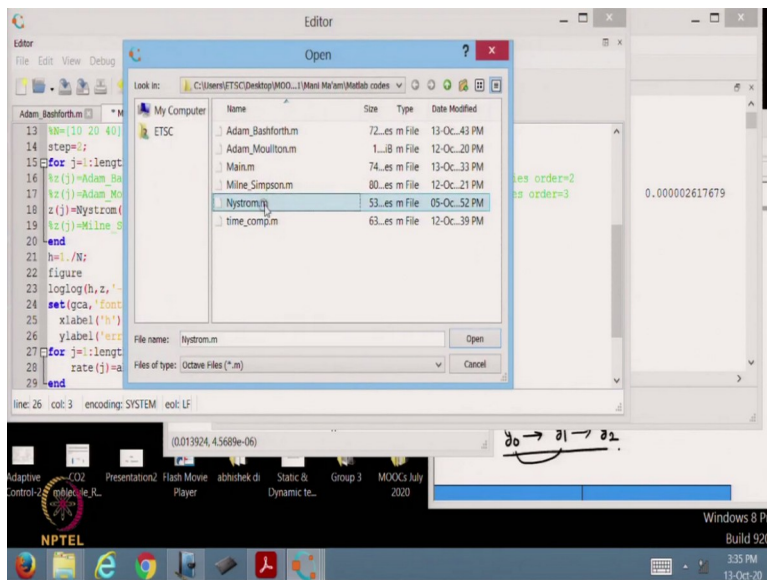
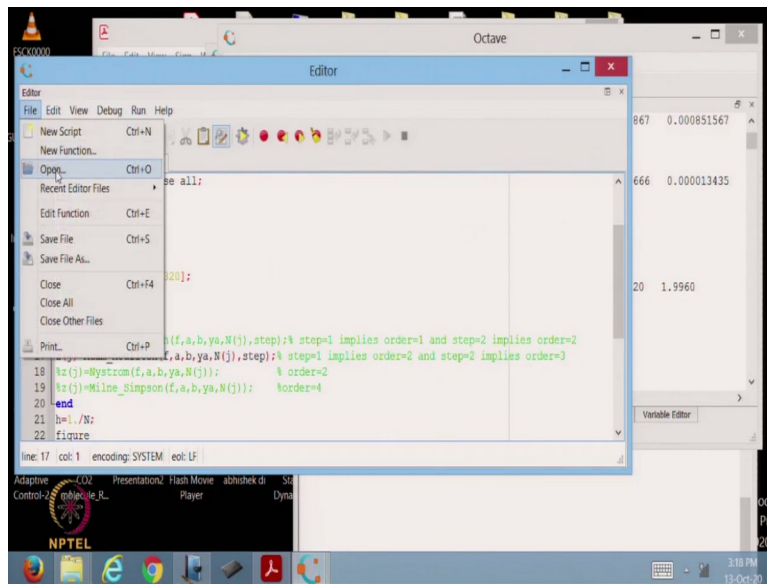
13 N=[10 20 40];
14 step=2;
15 for j=1:length(N)
16     z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17     z(j)=Adam_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18     z(j)=Nystrom(f,a,b,ya,N(j)); % order=2
19     z(j)=Milne_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-r',h,1./N.^(step),'-ok','LineWidth',2)
24 set(gca,'FontSize',15)
25 xlabel('h')
26 ylabel('error')
27 for j=1:length(N)-1
28     rate(j)=abs(log2(z(j+1)/z(j)));
29 end

```



So, now, Nystrom method, in Nystrom method, if you remember what we have learned in the lecture class. So, we have seen only one category that was Leap Frog or Midpoint and the same category we are implementing here, if you wanted to see high order methods under this category, you can again modify these codes and implement yourself.

(Refer Slide Time: 43:11)





```

1 function z=Nystrom(f,a,b,ya,N)
2 % f is the R.H.S of given diff equation i.e. y'=f(x,y).
3 % a and b are the starting and end points, respectively, of the domain.
4 % ya is solution at initial point
5 % N is the number of steps
6 % k is the step of Adam_Moulton
7
8 h=(b-a)/N;
9 x=a:h:b;
10 y(1)=ya;
11 y(2)=y(1)+h*f(x(1),y(1)); % two steps method, so evaluating first value using Euler's method.
12 for i=1:length(x)-2
13     y_exact(i)=exp((1/4)-(x(i)-1/2)^2);
14 end
15 for i=1:length(x)-2
16     y(i+2)=y(i)+2*h*f(x(i+1),y(i+1));
17 end

```

```

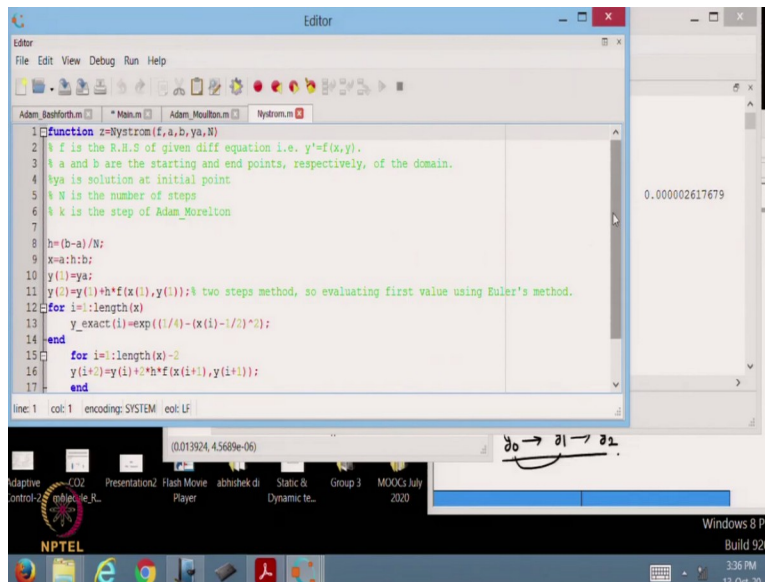
13 %N=[10 20 40];
14 step=1;
15 for j=1:length(N)
16     z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17     z(j)=Adam_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18     z(j)=Nystrom(f,a,b,ya,N(j)); % order=2
19     z(j)=MlineSimpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-r',h,1./N.^ (step+1),'-ok','LineWidth',2)
24 set(gca,'FontSize',15)
25 xlabel('h')
26 ylabel('error')
27 for j=1:length(N)-1
28     rate(j)=abs(log2(z(j+1)/z(j)));
29 end

```

So for that reason, let me also open this m file corresponding to the Nystrom method which we have made. So again, because in this case, I am implementing just one difference scheme. So that is why we are not taking argument as a step because we are not working out with different steps. We are working with just LeapFrog. That is what that argument is missing here, step which is intentionally.



(Refer Slide Time: 43:51)



```
1 function z=Nystrom(f,a,b,ya,N)
2 % f is the R.H.S of given diff equation i.e. y'=f(x,y).
3 % a and b are the starting and end points, respectively, of the domain.
4 % ya is solution at initial point
5 % N is the number of steps
6 % k is the step of Adam_Moulton
7
8 h=(b-a)/N;
9 x=a:h:b;
10 y(1)=ya;
11 y(2)=y(1)+h*f(x(1),y(1)); % two steps method, so evaluating first value using Euler's method.
12 for i=1:length(x)-2
13     y_exact(i)=exp((1/4)-(x(i)-1/2)*2);
14 end
15 for i=1:length(x)-2
16     y(i+2)=y(i)+h*f(x(i+1),y(i+1));
17 end
```

0.000002617679

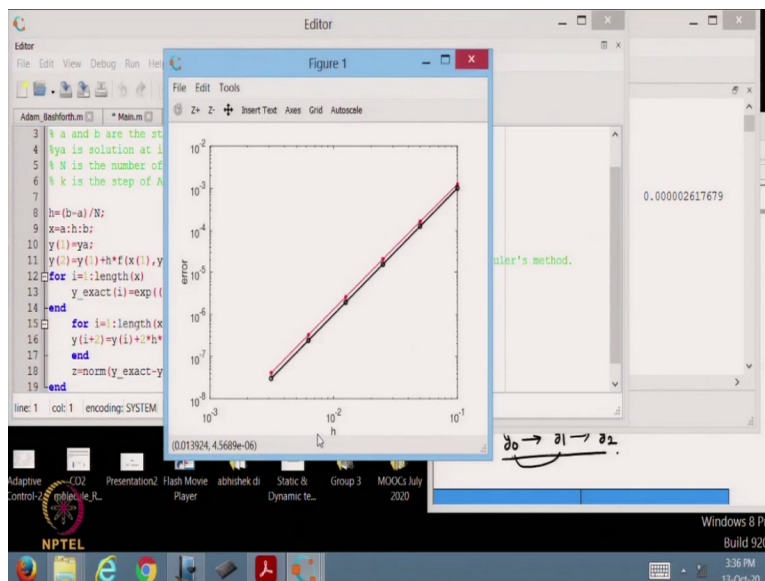
(0.013924, 4.5689e-06)

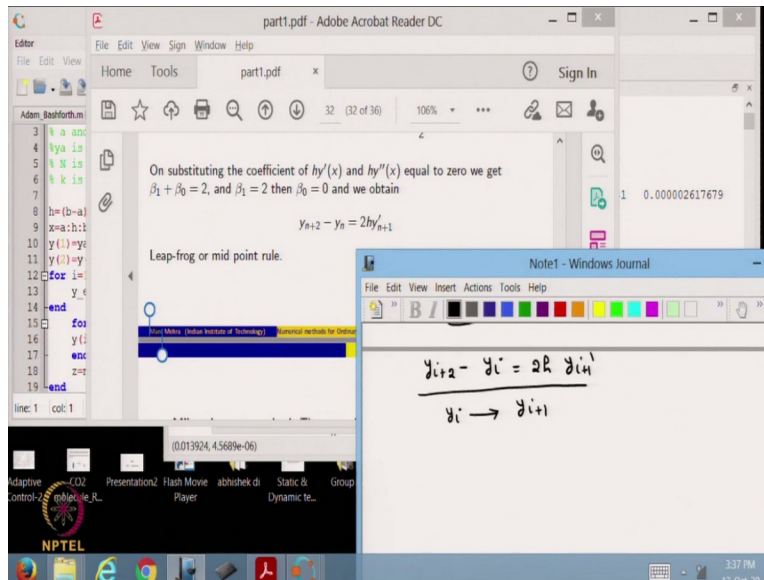
NPTEL

Windows 8 Pro  
Build 9200  
3:36 PM  
13-Oct-20

And now if I see the function name Nystrom which is here. So again, the function prototype is in front of you and all the meaning of all the arguments seem, except the step which is not here and the reason also I have explained to you just now.

(Refer Slide Time: 44:11)

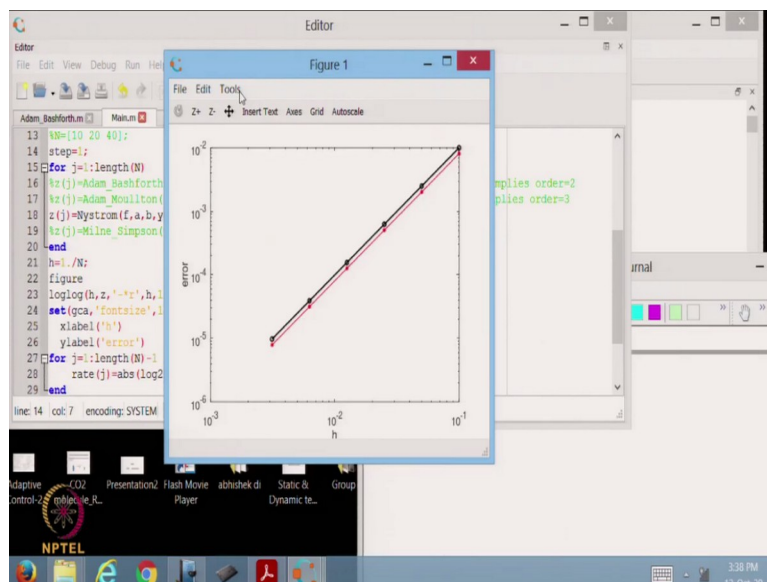
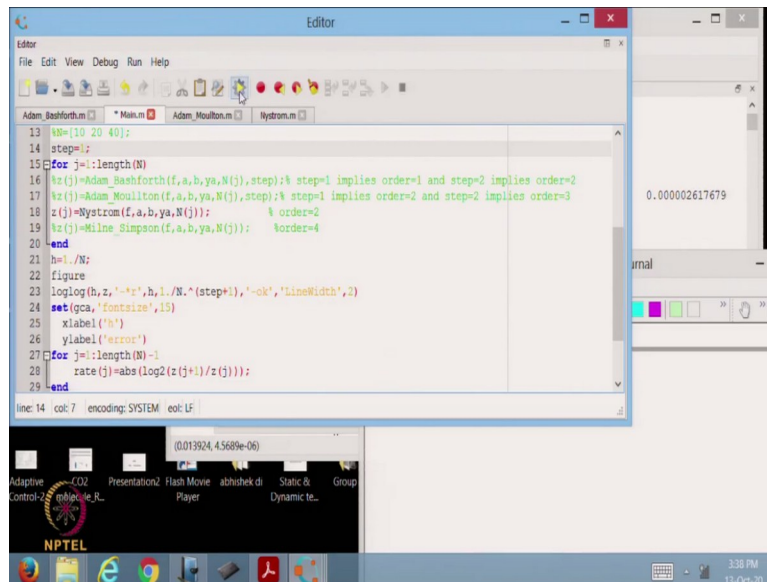




So, again, you can see the algorithm which we have implemented is  $y_{i+2} - y_i = 2hy'_{i+1}$ . So, again, this is a Two step method, I have to choose some other method to go, go from  $y_i$  to  $y_{i+1}$ . So, for that reason again because the final order of the method is 2, and the local truncation error in case of a forward Euler is also 2 that is why in this case also I am choosing forward Euler which you can observe from here.  $y_1, y_2$  is predicted by using Euler method everything is commented.

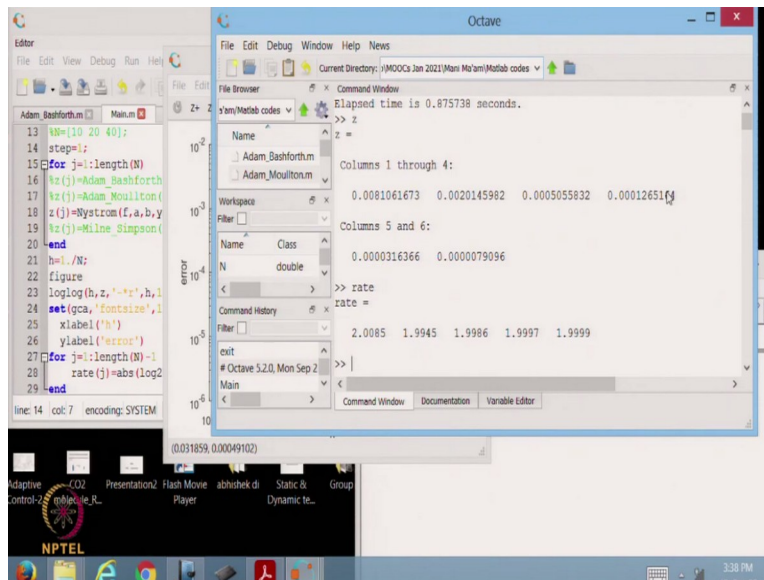
And then the final differences scheme which we have implemented is here which you can observe from here and again we are computing the error and then this error is as an output which we will get in the main function we are basically interested in error, but if some of you wanted to see how the numerical solution and exact solution look like that also I will explain you later how you can plot those exact solution and numerical solution basically both of them should be consistent.

(Refer Slide Time: 45:45)



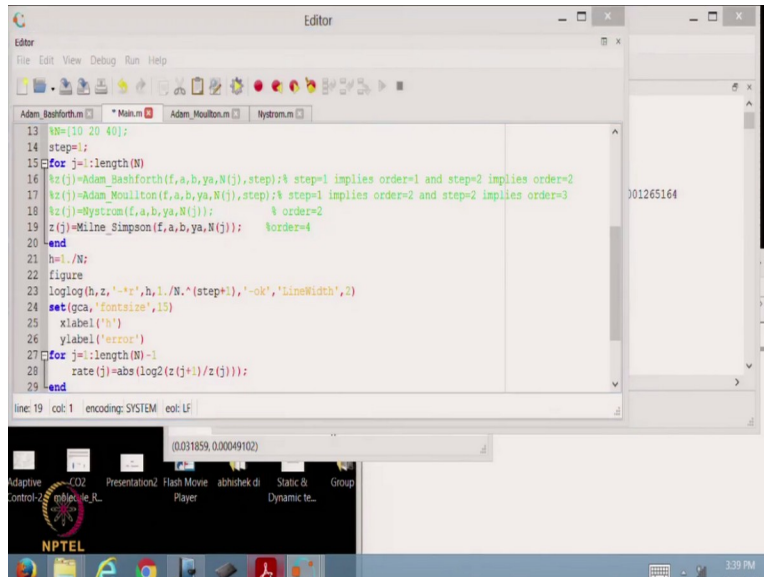
So, this is I can press 1 though it is irrelevant corresponds to Nystrom, but why I mentioned modifying this step is equal to 1 just so, that or at  $h^2$  line should be plotted parallel to the actual error because the order of the method is 2 that is what we have seen.

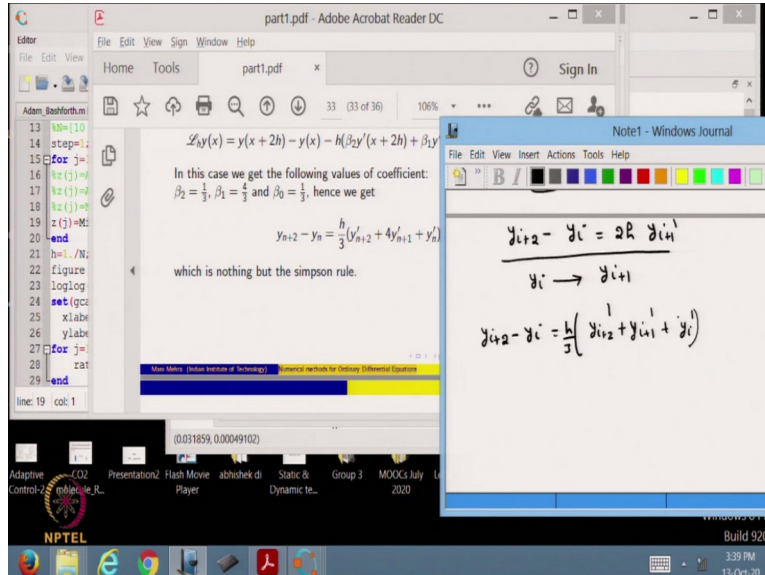
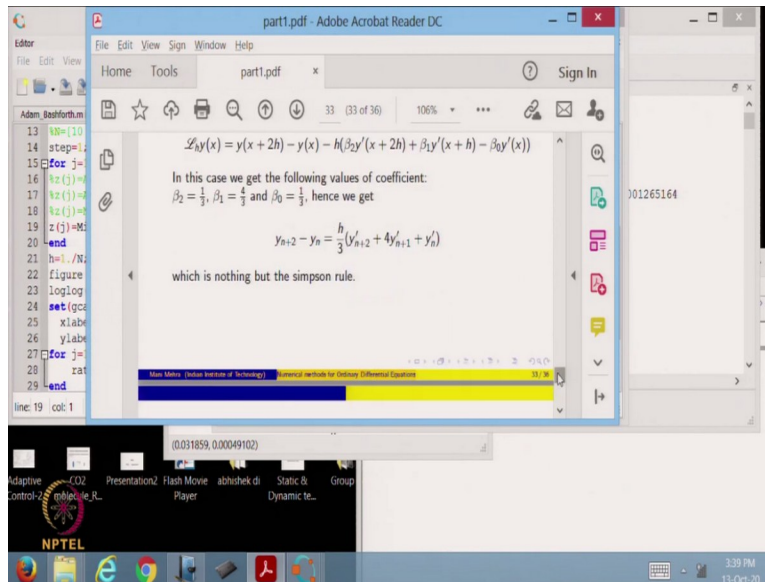
(Refer Slide Time: 46:18)



So, again if you want you can also see the error, error is tending to 0 as  $h \rightarrow 0$  if you want it to observe the rate which should be close to 2 as expected.

(Refer Slide Time: 46:36)



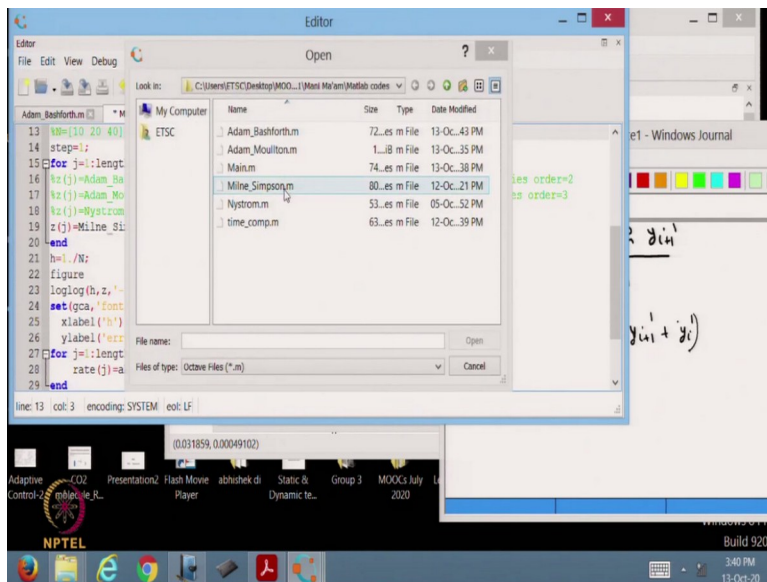
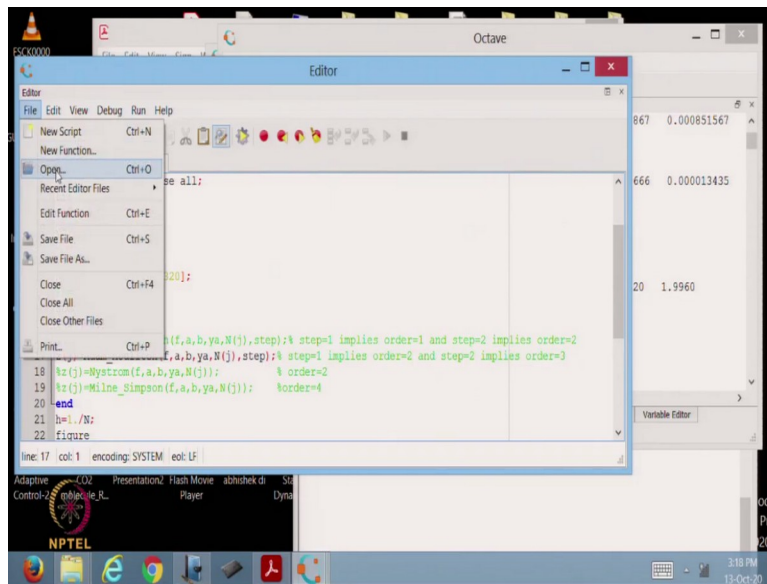


So, now, I am commenting on this because in this category as I have already said, We have worked out only one difference scheme. Milne's Simpsons formula, so, we can check in the slides corresponds to Milne's Simpsons formula what we have done basically we have implemented for  $k = 2$  we have implemented this differences scheme which I am writing again

here  $y_{i+2} - y_i = \frac{h}{3}(y'_{i+1} + y'_{i+1} + y'_i)$ . So, which is nothing but the Simpsons rules.

So, again here that part I have written in the comment that this will correspond to order is equal to 4.

(Refer Slide Time: 47:52)





```

7 h=(b-a)/N;
8 x=a:h:b;
9 y(1)=ya;
10 for i=1:length(x)
11     y_exact(i)=exp((i/4)-(x(i)-1/2)*2);
12 end
13 %y(2)=y(1)+h*f(x(1),y(1)); %using TS(1)
14 %y(2)=y(1)+h*f(x(1),y(1))+(h^2/2)*(-2*y(1)+(1-2*x(1))^2)*y(1); %using TS(2)
15 y(1)=y(1)+h*f(x(1),y(1))+(h^2/2)*(-2*y(1)+(1-2*x(1))^2)*y(1)+...
16     (h^3/6)*(-6*(1-2*x(1))*y(1)+(1-2*x(1))^3)*y(1); % using TS(3)
17 for i=1:length(x)-2
18     y(i+2)=(1/(1-(h/3)*(1-2*x(i+2))))*(y(i)+(h/3)*(1-f(x(i+1),y(i+1))+f(x(i),y(i)))));
19     eqn=w-(h/3)*(f(x(i+2),w)+4*f(x(i+1),y(i+1))+f(x(i),y(i))-y(i));
20     %y(i+2)=solve(eqn,w);
21 end
22 z=norm(y_exact-y,inf);
23 end

```

So, now, let me open the file Milne Simpsons. So, here also you can see we have implemented that algorithm here. An Initial Value you can predict with TS 1 TS 2 TS 3, but that you have to decide which you should choose, so that the global error of the method remains consistent. The order of the method which we expect from Simpsons is order 4 that is why we have to choose TS 3. If you want the previous one, but then that error will be dominated.

(Refer Slide Time: 48:51)

```

13 %N=[10 20 40];
14 step=1;
15 for j=1:length(N)
16     %z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17     %z(j)=Adam_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18     %z(j)=Rustrom(f,a,b,ya,N(j)); % order=2
19     z(j)=Milne_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-r',h,1./N.*(step+1),'-ok','LineWidth',2)
24 set(gca,'FontSize',15)
25 xlabel('h')
26 ylabel('error')
27 for j=length(N)-1
28     rate(j)=abs(log2(z(j+1)/z(j)));
29 end

```

So, again we can again in this method because we have worked out only with one difference scheme is step is irrelevant.

(Refer Slide Time: 49:04)

```

8 x=a:h:b;
9 y(1)=ya;
10 for i=1:length(x)
11     y_exact(i)=exp((1/4)-(x(i)-1/2)^2);
12 end
13 %y(2)=y(1)+h*f(x(1),y(1)); %using TS(1)
14 %y(2)=y(1)+h*f(x(1),y(1))+h^2/2*(-2*y(1)+(1-2*x(1))^2)*y(1); %using TS(2)
15 y(2)=y(1)+h*f(x(1),y(1))+h^2/2*(-2*y(1)+(1-2*x(1))^2)*y(1)+...
16     (h^3/6)*(-6*(1-2*x(1))*y(1)+(1-2*x(1))^2*y(1)); % using TS(3)
17 for i=1:length(x)-2
18     y(i+2)=(1/(1-(h/3)*(1-2*x(i+2))))*(y(i)+(h/3)*(4*f(x(i+1),y(i+1))+f(x(i),y(i)))));
19     %eqn=-h/3*(f(x(i+2),w)+4*f(x(i+1),y(i+1))+f(x(i),y(i)))-y(i));
20     %y(i+2)=solve(eqn,w);
21 end
22 z=norm(y_exact-y,inf);
23 end
24

```

And now, if I open Milne's Simpsons method here also a step is not taken as an argument because of the same reason because we have implemented only one difference scheme.

(Refer Slide Time: 49:22)

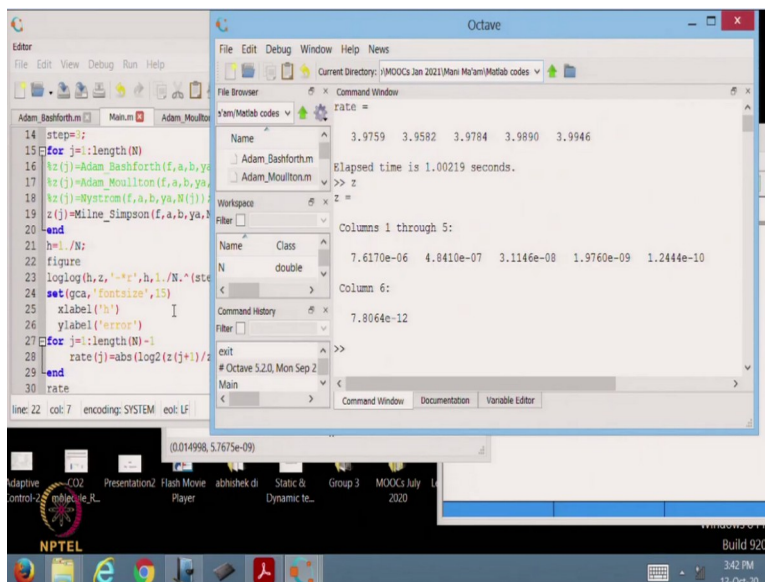
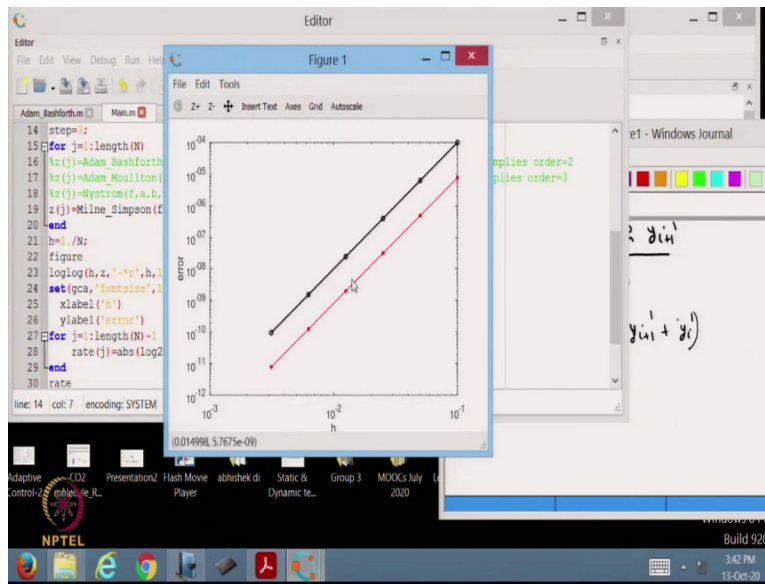
```

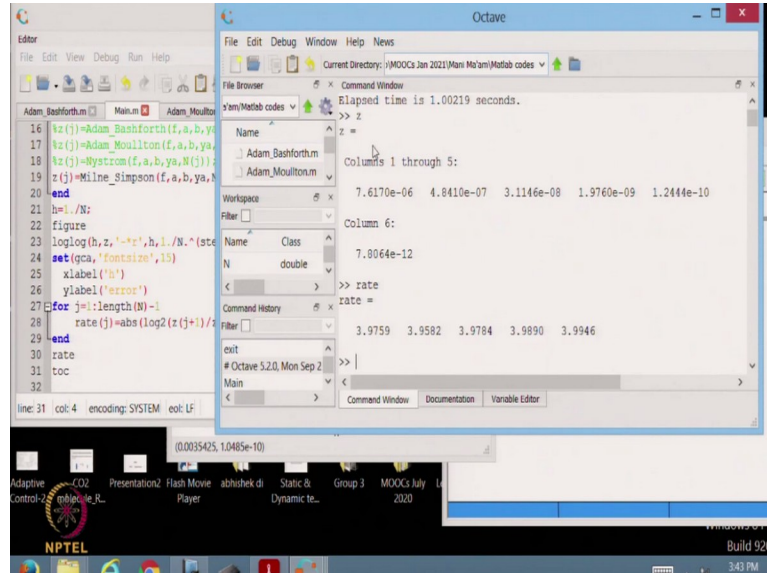
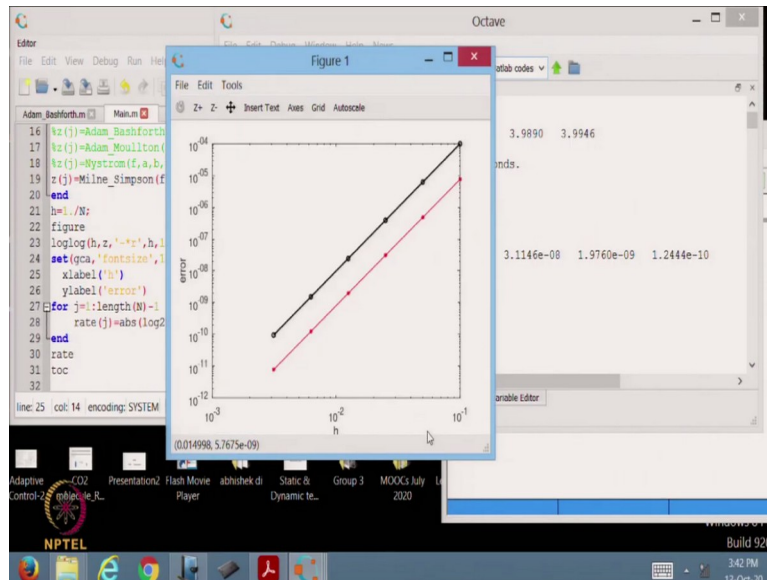
14 step=1;
15 for j=1:length(N)
16     %z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17     %z(j)=Adam_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18     %z(j)=Nystrom(f,a,b,ya,N(j)); % order=2
19     z(j)=Milne_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-o',h,1./N,'-o','LineWidth',2)
24 set(gca,'FontSize',15)
25 xlabel('h')
26 ylabel('error')
27 for j=1:length(N)-1
28     rate(j)=abs(log2(z(j+1)/z(j)));
29 end
30 rate

```

So, here also if you want I can order 4 so, here basically I should type it 3. So, that  $h^4$  line should be plotted of course, step is not as an argument in Milne's Simpsons and as well like Nystrom.

(Refer Slide Time: 49:44)





So, here also you can see, if you want you can separately typed out  $z$  very good as this is you can see how fast error is tending to 0 because this is the highest order method which we have implemented so far. Earlier methods were either 1 2 or 3 this is the highest order method which we have implemented and the result of that highest order is also visible to you error is tending to 0 very fast. So, here also  $z$  that also you can observe graphically that error is of course, when  $h \rightarrow 0$  and the maximum error which we achieve by taking  $h$  is roughly order of  $10^{-11}$  which we can observe from here also and rate should also be close to 4. That is what we have seen the comment part and here also by typing which is also here.

So, that is the analysis of every defensive scheme under different categories Adam Bashforth, Adam Moulton's, Nystrom, Milne's Simpsonn, we have done we have verified numerically the order of each differences scheme So, if we have determined the order also we have also seen the how this differences schemes are converging with respect to one particular example.

(Refer Slide Time: 51:34)

```

1 % Consider the following initial value problem:
2 % y'(x)=(1-2*x)y(x), x>0
3 % y(0)=1;
4 % The exact solution is y(x)=exp((1/4)-(x-1/2)*2)
5
6 clc; clear all; close all;
7 tic
8 f=@(x,y) (1-2*x)*y;
9 a=0;
10 b=1;
11 ya=1;
12 N=[0 20 40 80 160 320];
13 h=[10 20 40];
14 step=1;
15 for j=1:length(N)
16     %z(j)=Adam_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17     %z(j)=Adam_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3

```

```

17     %eqm=- (h/2)*f(x(i+1),w)+f(x(i),y(i))-y(i);
18     %y(i+1)=solve(eqm,w);
19     end
20     z=norm(y_exact-y,inf);
21     else
22         %y(2)=y(1)+h*f(x(1),y(1)); using TS(1)
23         %y(2)=y(1)+h*f(x(1),y(1))+ (h^2/2)*f'(x(1),y(1))-y(1); using TS(2)
24         %y(2)=y(1)+h*f(x(1),y(1))+ (h^2/2)*f'(x(1),y(1))+ (h^3/6)*f''(x(1),y(1))-y(1); using TS(3)
25         % (h^3/6)*(-6*(1-2*x(1))*y(1)+(1-2*x(1))^2*y(1)); % using TS(3)
26         for i=1:length(x)-2
27             %y(i+2)=(1/(1-(5*h/12)*(1-2*x(i+2))))*(y(i+1)+(h/12)*(5*f(x(i+1),y(i+1))-f(x(i),y(i))))); % using TS(4)
28             %eqm=- (h/12)*(5*f(x(i+2),w)+5*f(x(i+1),y(i+1))-f(x(i),y(i))-y(i+1)));
29             %y(i+2)=solve(eqm,w);
30         end
31     z=norm(y_exact-y,inf);
32 end

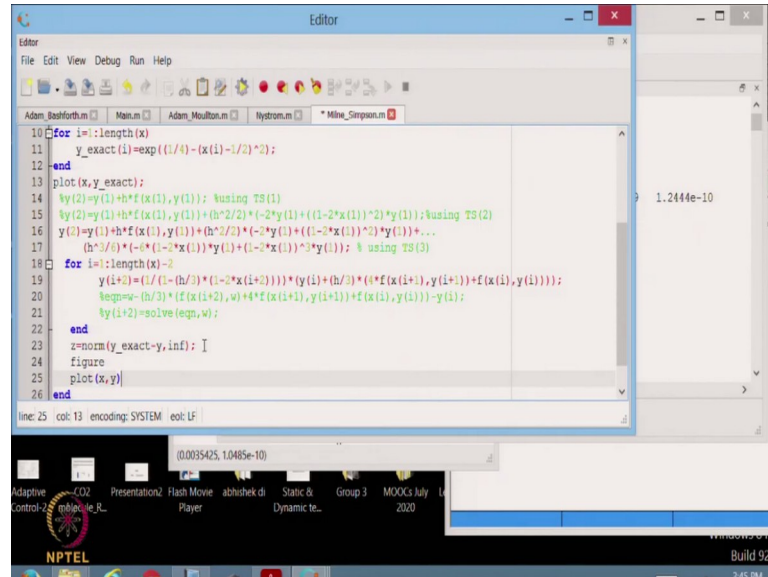
```

Of course, if you wanted to work out with different examples, you have to change this right hand side correspondingly you have to change in the functions also because in functions also, sometimes to predict TS 1 TS 2, we have also computed the derivative of the right hand side. So,



correspondingly you have to change in all those four routines. Now, if you wanted to see how this exact solution and numerical solution look like.

(Refer Slide Time: 52:12)



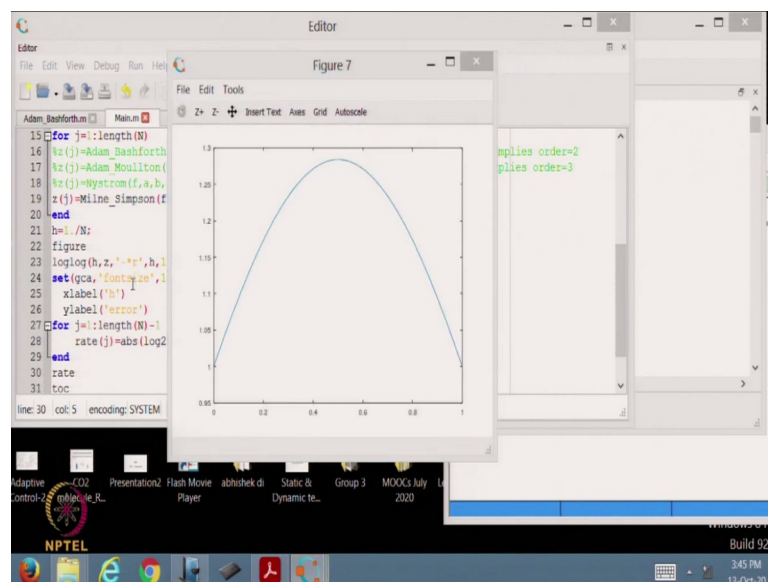
```

10 for i=1:length(x)
11     y_exact(i)=exp((1/4)-(x(i)-1/2)^2);
12 end
13 plot(x,y_exact);
14 %y(2)=y(1)+h*f(x(1),y(1)); %using TS(1)
15 %y(2)=y(1)+h*f(x(1),y(1))+ (h^2/2)*(-2*y(1)+(1-2*x(1))^2)*y(1); %using TS(2)
16 y(2)=y(1)+h*f(x(1),y(1))+ (h^2/2)*(-2*y(1)+(1-2*x(1))^2)*y(1)+...
17     (h^3/6)*(-6*(1-2*x(1))*y(1)+(1-2*x(1))^3*y(1)); % using TS(3)
18 for i=1:length(x)-2
19     y(i+2)=(1/(1-(h/3)*(1-2*x(i+2))))*(y(i)+(h/3)*(4*f(x(i+1),y(i+1))+f(x(i),y(i)))));
20     %eqn=- (h/3)*(f(x(i+2),w)+4*f(x(i+1),y(i+1))+f(x(i),y(i)))-y(i));
21     %y(i+2)=solve(eqn,w);
22 end
23 z=norm(y_exact-y,inf);
24 figure
25 plot(x,y)
26 end
  
```

line: 25 col: 13 encoding: SYSTEM eol: LF

So, in that case, what I will suggest you do is type this command. So, basically I am plotting the exact solution and let me also plot down the numerical solution here load x into y.

(Refer Slide Time: 52:56)





```

7 tic
8 f=@(x,y) (1-2*x)*y;
9 a=0;
10 b=1;
11 yap=1;
12 N=[10 20 40 80 160 320];
13 h=10/20/40;
14 step=1;
15 for j=length(N)
16 %z(i)=Adams_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17 %z(i)=Adams_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18 %z(i)=Mystrum(f,a,b,ya,N(j)); % order=2
19 z(j)=Milne_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-r',h,1./N,'(step+1)', '-ok', 'LineWidth',2)

```

So, so now I can run. I will also see the exact numerical solutions, why these different figures are coming because I am choosing different N. So let us work out for a particular case where I am achieving the maximum error. So, this is basically 320. And I have chosen only so rate will not make any sense and your code will stop here.

(Refer Slide Time: 53:30)

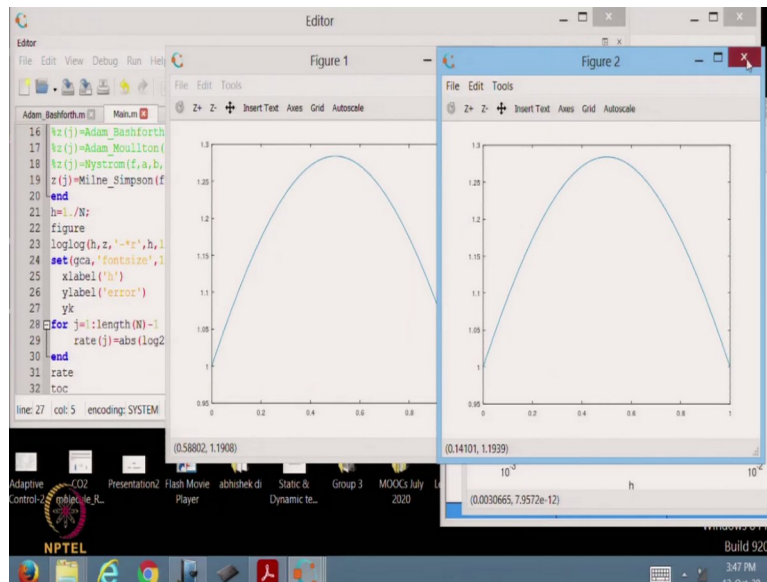
```

16 %z(i)=Adams_Bashforth(f,a,b,ya,N(j),step); % step=1 implies order=1 and step=2 implies order=2
17 %z(i)=Adams_Moulton(f,a,b,ya,N(j),step); % step=1 implies order=2 and step=2 implies order=3
18 %z(i)=Mystrum(f,a,b,ya,N(j)); % order=2
19 z(j)=Milne_Simpson(f,a,b,ya,N(j)); % order=4
20 end
21 h=1./N;
22 figure
23 loglog(h,z,'-r',h,1./N,'(step+1)', '-ok', 'LineWidth',2)
24 set(gca,'FontSize',15)
25 xlabel('h')
26 ylabel('error')
27 yk
28 for j=length(N)-1
29 rate(j)=abs(log2(z(j+1)/z(j)));
30 end
31 rate
32 toc

```

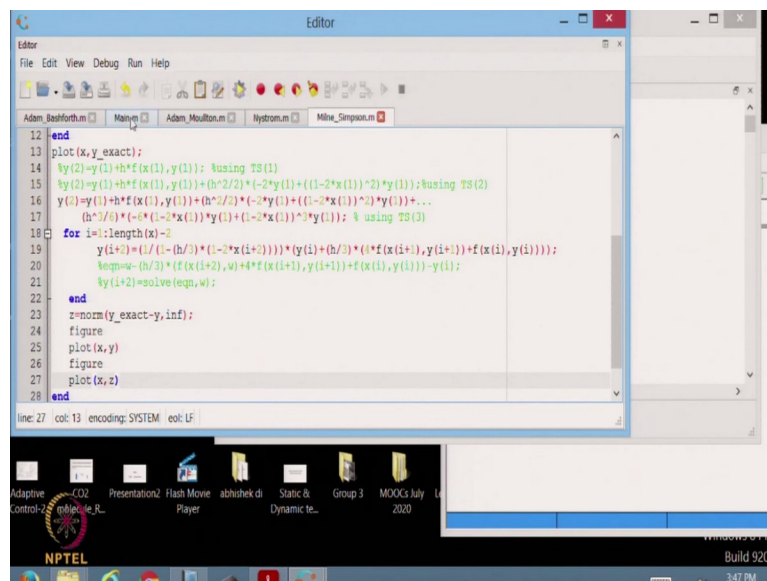
Again, some arbitrary thing I am writing so that my code should stop here.

(Refer Slide Time: 53:38)



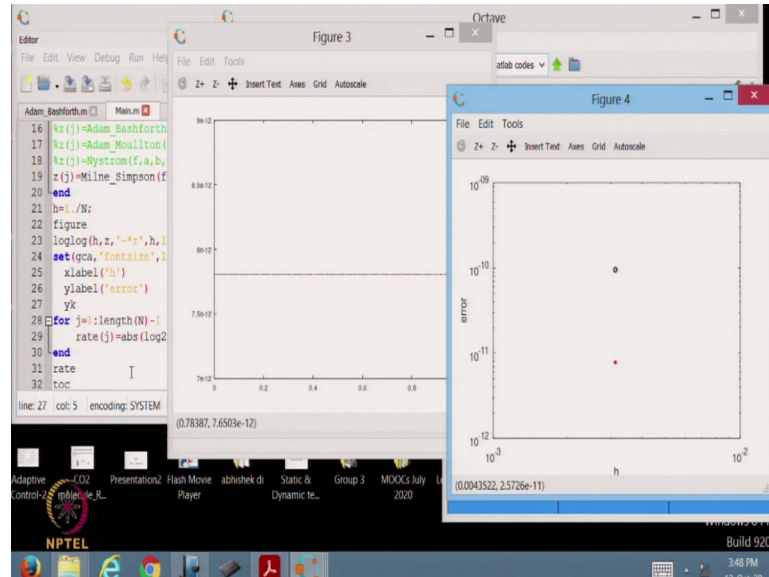
So of course, this is only one particular  $h$ ,  $h$ , that is why you do not see the line which we were seeing earlier, because it corresponds to only one particular  $h$ . So, this is figure 1 is the exact solution and figure 2 is the numerical solution. So, this is just to look at the shape of how this numerical solution is behaving, both of them are consistent. Anyway, we can observe from this figure and very fine details if you want to observe the numerical solution and exact solution, so, for that reason, you have to plot the error also. So, that error also if you want, you can plot it.

(Refer Slide Time: 54:35)



You do not compute the norm of the error instead you plot the error. So instead, here or here itself in separate figures we can plot the error  $x$  into  $z$ .

(Refer Slide Time: 55:08)



So, let me run so that you can see the magnitude of the error which is roughly the same because the function is quite smooth. So, if you wanted to observe the behavior of the error and with respect to  $01$  that also you can check by plotting error, because here we are taking the norm of the error corresponding to one  $h$ .

So, I am closing now. So, in today's lecture, we have seen how to implement explicit schemes, implicit schemes as well as Two step methods for a Linear Problem. So, thank you very much.