MATLAB Programming for Numerical Computation

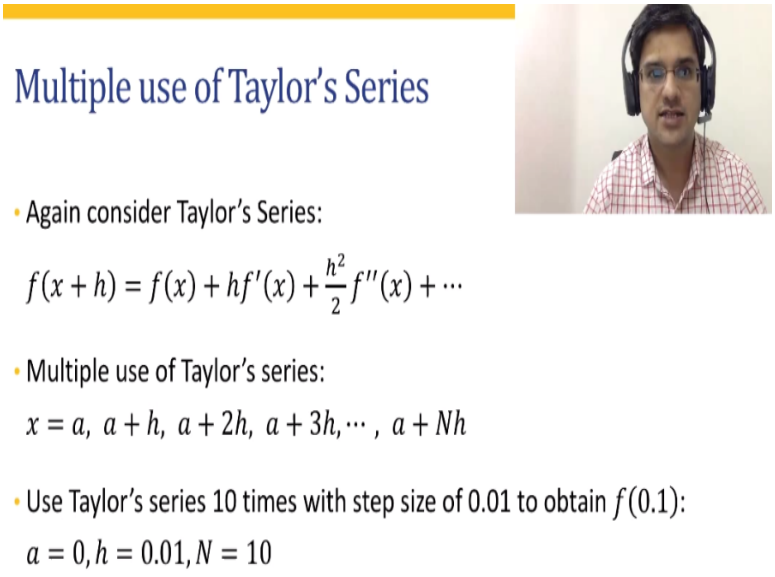
Dr. Niket Kaisare
Department of Chemical Engineering
Indian Institute of Technology, -Madras

Module No. #02
Lecture No. #2.4
Errors and Approximations – Step wise methods and error propagation

Hello and welcome to the MATLAB programming for numerical computations. We are in the last lecture of module 2. In this module we are covering errors and approximations in this lecture 2 .4. We are going to cover what is known as step-wise methods. The step wise methods we start with a certain initial condition.

Multiple steps go on from the initial condition and multiple steps go on from the initial condition to a final point. The errors that come in step-wise method accumulate from one step to another and we get a certain way in this error which prop again. This is what we are going to cover in today's lecture. This is what we are going to cover in today's lecture.

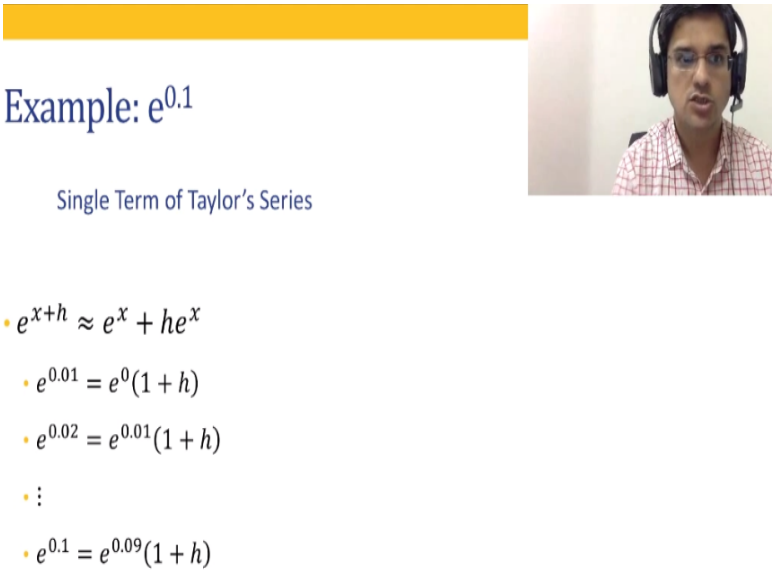
(Refer Slide Time: 00:53)



Let us consider the Taylor's series. Taylor's series itself in which lecture 2.2, we have use to derived the Lawrence series expansion of e ^ of 0.1, in that lecture we considered approximating e ^ 0.1 in a single step.

In this case we are going to take multiple steps that means we start with e ^ 0 then go to e ^ 0 + h, e ^ 2h and e ^ 3h and so on up to e ^ nh. We take multiple steps in order to go from 0 to 0.1 For example if we were to use the Taylor's series expansion 10 times starting with a = 0 and h = 0.01 what we are going to get is, as follows.

(Refer Slide Time: 01:43).



So e ^ 0.01 is going to be 1 multiplied by 1+ h. Why do we get that is because using the Taylor's series expansion e ^ x+ h is f (x) which is e ^ x + h multiplied by f'(x). Therefore e ^ 0.01 is 1 * l + h. e ^ 0.02 is this value, e ^ 0.01 * 1+ h, e ^ 0.03 is e ^ 0.02 * 1+h and so on so forth. So at each time the new expValue at the next point is going to be the expValue multiplied by 1+ h.

(Video Start: 02:33) Let us do this in MATLAB. The script called multiStepExp. Calculate e ^ 0.1 using multiple steps of Taylor's series okay. a = 0.1, h = 0.01, N, the number of steps are going to be nothing but a divided by h okay. And trueVal is nothing but exp of a. Problem set up multi-step computation okay.

We will have to use for loop, for i going from 1 to n because we are going to go from e ^ 0 to e ^ 0.1 in n steps. So therefore we have for i=1 to e expVal is equal to expVal multiplied by 1 + h. Remembered that is what said when we did in this MATLAB. At each time the new expVal is the previous expVal * 1+ h okay.

And write this end calculate error and display so, let say calculate true error and display err equal to abs value of trueVal -expVal. We run this we are likely to get in error. So just take a couple of error and just pause for a second. You can pause this video; you can look at the score and think about or write down why you think we are going to get an error okay.

So, let me save this and run this and let see what error we get. Okay we got an error. Can you think of what error it might be undefined function or variable expVal. Remember in the previous lecture that is the error we have obtained also why is because we have not initialized that. Let go back to the power point and see. We start with e ^ 0, e ^ 0 at anything to the power 0; we know is equal to 1.

So let start with expVal = 1. So that is our starting point and with this starting point, when we let this run, we are going to get our error is 5.5 * 10 ^ -4. That is what we get. Whenever h was 0.01, if we decrease the h to 0.001 and run this, the error is going to decrease the error goes 5.5*10 ^ -4 to 5.5 * 10 ^ -5.

So, it decreases by a factor of 10. If we were to change this loop that say h=0.1 and run this code, the error from this point has increased by a factor of 10. When we increase the step size by a factor of 10, the error increases by a factor of 10. When we decrease the step size by a factor of 10, the error decrease by a factor of 10. What do we mean by all this we mean is that the order of accuracy of this method is h ^ 1 that we have seen in the previous couple of lectures?

That is the definition that is how we understand the order of accuracy of this method. So if we want to retain of single term of Taylor's series, the order of accuracy is going to be equal to h ^ 1. What we are going to do is, we are going to convert this script that we have obtained into a function and that function we are going to call multiple times with different step sizes.

What I mean by that is this, is the function usually contains the core of the computations which is this part, the input and output are often place in a separate script. That is what we usually do in larger programs and that is what you are very likely to do in the most of the assignment submission that you are going to do of this particular course.

So, let us try and do that so what we are going to do is, we are going to modify multiStepExp into a function that function will take as inputs. It will take the value of a and the value of h that means does the final points and the step size going to take the order to x to reach the final point. And it will retain the expVal at the final point okay. And all the input and output things we are going to move that to a script. So, that is added edit called that as scriptExpResults okay.

To understand errors in multi-step method is this script to understand scripts and functions are written in MATLAB projects. What I mean by projects is. If you have a large project that you need to deliver then how we write scripts and functions. So we will as I said in lecture 1.1, in large projects we will have one single script that script is going to be the driver script and everything else will be put a function okay.

So the problem set up is going to be move from here to the script problem setup okay. Now this we are going to converted in to a function okay. Here the result that we want is expVal equal to multi step okay. The name of the function should be the same as the file name multiStepExp and input for this are the final point a and step size in which we are going to reach a, okay.

Now what we need over here is, to calculate n also okay. So n is going to be equal to a/ h as we have done earlier the output should not be moved over here. The output should be move to calling function or to be sorry, to be calling script. So we are move this calling script. And now the final thing that we need to do, we need to call this function that will compute the result for us the function is going to be expVal is going to be multiStepExp a, h and we do not need the computation of n over here.

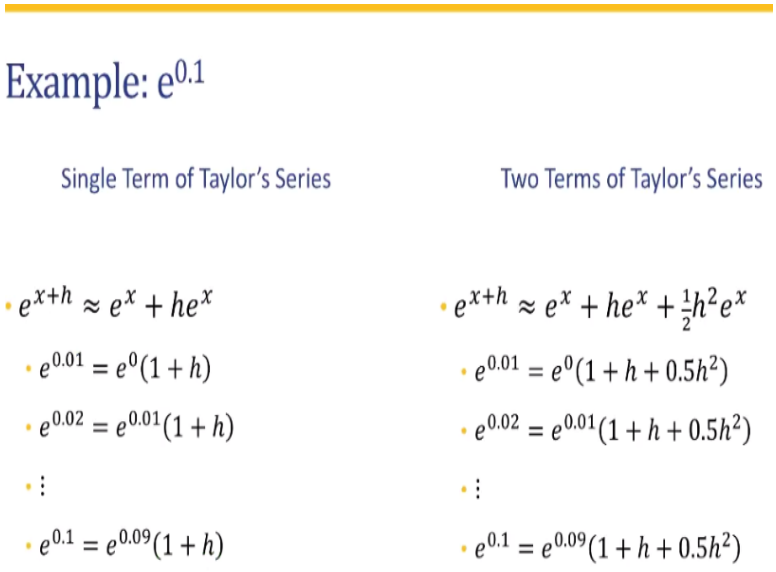
Because n is not use at all computing using multiStepExp method okay. So this is what we have, so, let save this okay. Run this script exp results with 0.1, we called 0.0052 as the error. If we were to change this, we will get this 5 * 10 ^ -4 and if we are going to change this further, we are again now going to get 5 * 10 ^ -5 exactly.

As we have got the 4. Okay so now you are going to do is use the script file in order to find out the overall error varies with step size. So we are going to run this for multiple step size is not just for a single step size for i=1, 2, 3, h is going to be equal to 10 ^ –i.

Remember this what we had done in the previous lecture also expVal is this guy and we will move this error also inside the loop, error = abs of trueVal - expVal that is also moved over here. I put a semi colon in front of it. So we need to display this okay and we need to store these values, h all, we want to store h, errAll, we use that to store our errors and instead of calculated true error and display we will say plot error verses step size okay.

And what I will do is loglog (hAll, errAll) as a blue color line okay. So let us see what happens we actually do not need h because this h is defined over here. So we can delete this. That is clear all and lets run this script, so this is how the error decreases with step size it is a linear line slope = 1.

(Refer Slide Time: 13:41)



Now instead of single term if we had two terms of Taylor's series expansion our results are going to change little bit. e to the power 0.01 is going to be the initial guy multiplied by 1+ h+ 1/2 h^2. This additional term comes in now, expVal come to be the old expVal again multiplied by 1 + h over here is going to be (1 + h) + 0.5 * h^2. In order to do this val, we need to do is to go to the

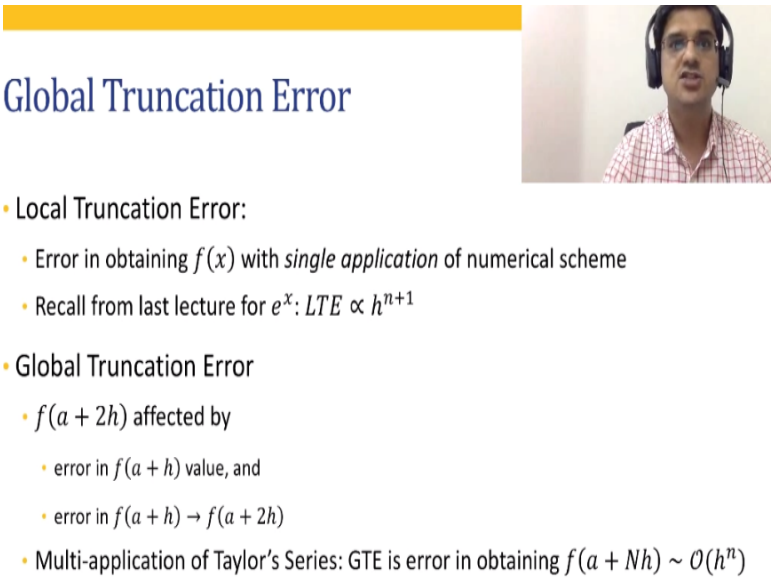
multiStepExp and at this + 0.5 * h^2 and save this okay. Do not forget this to save this because if you do not save it, we are going to get same results as we had before so now. We run this, we would have run this for 2 terms either series expansion the first order term as the second order term. What we want to do is if we want to dash red color line with square symbols, let us save this.

We need to do one more thing, we need to type hold on. When we type hold on earlier, lot will be held if we do not type, hold on the earlier plot will be over written in MATLAB okay. So let us and we got the end results and let us look at the error graph. So, when we see error graph this is what the results look like when we have 2 terms of the Taylor's series expansion retain okay.

So let actually look at these error values of errAll, just print errAll at each step decreases by 2 orders magnitude this is 1.7 in to 10 ^ - 4. It goes to 1.8 *10 ^ - 6 and it will go to 1 .8 * 10 ^ – 8. So let us try this errAll 3 is 1.8 *10 ^ -8.

What we see is every time we decrease the step size by a factor of 10, the error decreases by a factor of 100 which means that the method is h ^ 2 accurate, when we retain 2 terms in the Taylor's series. Let we summarize the results that we have gotten.

(Refer Slide Time: 16:10)



So far what we have seen in the multistep Taylor's series expansion is, it is not a local truncation error but that is global truncation error. The global truncation error comes in because f of a + 2h is affected by f of the error that was there f (a+ h) as well as it is affected by the error inherent in the single step

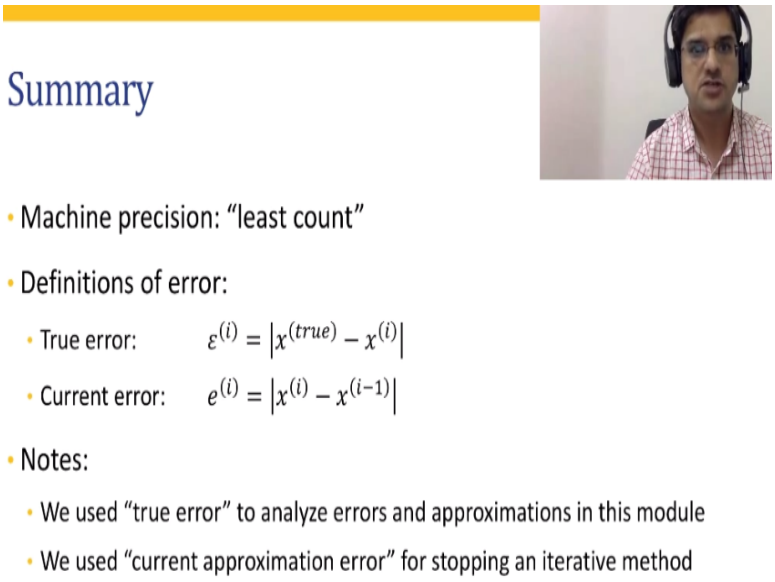
So there is a accumulation of error that happens when we from 0.01, 0.02, 0.03, so on so forth. The error keeps adding up at each and every distance and that why the global truncation error is usually greater than the local truncation but the way the global truncation error behaves is slightly different slightly different. That's the way the local truncation error behaves. The local truncation error if it is n+ 1 order accurate through the global truncation error, it would be the nth order angle okay.

If you did not really understand the graph this point I would suggest you do not worry about it. However I am presenting this to you because in the rest of the lecture. I am going to bring up the term local truncation error and global truncation error and I just want to put a contacts in the today's lecture okay.

So again to recap the main aspect of what we have learn in today's lecture is, how to go how to use multistep methods and how to converts step in to functions and run that functions multiple

times. So that we can get trends how which the error changes with the steps. With that I come to the end of lecture 2.4. So to summarize the module 2 what we started of it.

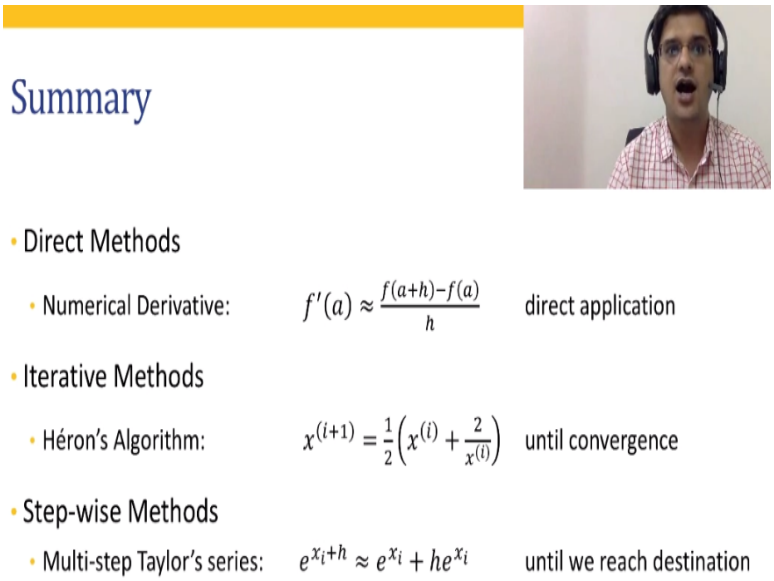
(Refer Slide Time: 18:01)



We talk about machine precision is nothing but least count of any computer we defined true errors. The true error is the difference between x (true) and x (i). The current case of the numerical techniques we use these true errors to analyze the errors and approximation in this particular module to understand what we need by truncation errors and round of errors and error propagation that the contact in which we use the term true error we use the term current approximation error.

In order to design when to stop the aggregative method, when ei fell below certain threshold, certain tolerance value as well we said that we exit the while loop, what we did in lecture 2.3.

(Refer Slide Time: 18:54)



Finally to summarize the various type of methods that we are going encounter in modules 3 to 8 are the direct methods or the direct methods we directly apply set of formulas or set of expressions that we have derived and obtained the results in a direct method. In an iterative method a set of formulae or set of expressions that we will do.

We apply this in a iterative manner again and again till the difference i plus 1 and xi falls below the certain threshold. And finally today's lecture we saw step wise method and we took example of multistep Taylor's series where we use the multistep Taylor's series again and again until we reach the final destination starting from an initial point. So with that I come to the end of lecture 2. 4 and indeed to the end of module 2.I will see you in the next module and thank you, bye

(Video Ends: 19:45)