MATLAB Programming for Numerical Computation

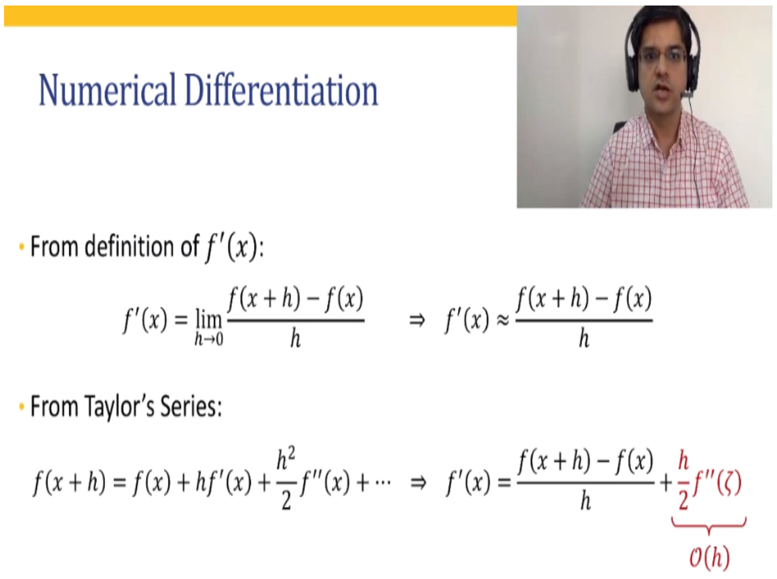
Dr Niket Kaisare
Department Of Chemical Engineering
Indian Institute of Technology- Madras

Module No. #02
Lecture No. #2.3
Errors and Approximations – Round off errors and Iterative methods

Hello and welcome to MATLAB programming for numerical computations. We are in module 2 errors and approximations. In lecture 2. 1 we introduced ourselves to errors and approximations and how they arise in numerical computations. Lecture 2. 2 we considered truncation errors. We have taken the example of Maclaurin series expansion of e ^ a and showed how including the additional terms form the infinite series improves the accuracy of the approximate method that we used.

In this method, in this lecture 2. 3 we are going to consider another type of errors which is known as round off errors and we will see the tradeoff between round off and truncation errors. There after we will talk about another type of method known as iterative method.

(Refer Slide Time: 01:01)



Numerical Differentiation

- From definition of $f'(x)$:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \quad \Rightarrow \quad f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

- From Taylor's Series:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \cdots \quad \Rightarrow \quad f'(x) = \frac{f(x+h) - f(x)}{h} + \underbrace{\frac{h}{2} f''(\zeta)}_{O(h)}$$

So, we will take the example from numerical differentiation. So instead of limit h tends to 0 in the finite precision machine, we can calculate f `(x) approximately as f (x + h) - f (x), the whole thing divided by h. This also can be derived from Taylor's series. This is something that you would have done in twelfth grade or your first year math class. We are not going to bother ourselves with how to derive these equations.

We are going to use these equations in MATLAB. We are going to just remember what this particular term means, order of accuracy when it comes to numerical methods okay. So let us go to MATLAB and do this numerical derivative problem.

(Video Starts 01:52) Okay so the example that we will solve it in today's lecture is going to be tan inverse of x we want to find d by dx of tan inverse of x at x = 1 okay. So let us look at what the command in MATLAB is to get tan inverse and that command is atan. So if we do atan of 1, we will get the result as pi by 4 and that is what we get when we put atan of 1. Let us check this by typing pi by 4 and as you can see atan of 1 is indeed pi by 4 okay. (Video Ends 02:30)

So now let us look at what we need to do. We need to use this particular approximate method in order to get the derivative. Let us define our a. (Video Starts: 02:40) Let us clear all, let us define our a =1. Our true value of the differential trueVal is going to be equal to 1 divided by 1 + a square okay. Remember the brackets. So the trueVal = 0.5. So, the d / dx of $\tan^{-1}(x)$ at x = 1 is going to be equal to 0.5. Now we want to compute this numerically and the way we will compute this numerically is, we will first define our h.

Let us start with h = 10 ^ -2, approxVal is going to be atan x + h rather atan a + h - f (h), which is atan of a now the whole thing in brackets divided by h that is going to be our approxVal, the error is going to be nothing but abs value of trueVal minus approxVal and the error is 0.0025.

Let us do this again. For a different value of h and that value of h is instead of 10 ^- 2, we will take that value as 10 ^ -3 okay. We will again calculate our approxVal in the same manner and we will calculate our error also in the same manner. So, what we see is when we decrease our h by a factor of 10, our error is also decreasing from 2*10 ^ - 3 to 2 *10 ^ - 4 which is again a factor of 10.

Let us decrease our h further. So we take the h as 10 ^ -4. approxVal calculate in the same manner, error we calculate in the same manner. What we see is when we further decrease our h by a factor of 10, the error also further decreases approximately by a factor of 10. (Video Ends 05:02)

Recall that this is the consequence of the fact that the method is h ^ 1 accurate which means that when we decrease the h by a factor of 10, our error decreases by a factor of 10. When we decrease h by a factor of 100, our error decreases by a factor of 100 so on and so forth. That is the meaning of the order of accuracy of any numerical method.

(Video Starts 05:31) Okay so what we have done so far is, made all these computations on the command prompt. Recall what we had said in the lecture on MATLAB functions and scripts. MATLAB script is nothing but running all this commands one after the other in a file okay. So what we can do is, we can copy and paste all of these in a file and that file will be a MATLAB script.

That will run our commands in a sequential manner. So, let us edit the file will call it new numDeriv to calculate numerical derivative. So let us go here h = 0. 0001. I am just pressing the up arrow key. When I keep press f arrow key, will go over the previous commands that were there in MATLAB. Sorry, okay so these are the numerical derivatives for h = 0.0001.

What we had also done is calculated this for 2 other values of h and we will do that here again. We just copy pasted these things. This for h = 10 ^ -2, h = 10 ^ -3 and h = 10 ^ -4 okay. And we will save this, will clear, clear all the variables and clc will clean up this screen for us okay.

So, now what i will do is, run this code numerical derivative a numDerive okay. Now when I run this code, I expect an error. I will take up a couple of seconds, you can pause this video right now and can you spot, what error we are going to get when we run this code okay. I will show you what happens when we run this code and what error we are going to get.
So, let me click on run. So when I run this, I get an error. It says undefined function or variable a. We did not get this error earlier, we got this error now. Why this is happening? What does this

error mean? What does error means is that the value of a has not been defined. Now why do you thing we got this error but we did not get this error earlier.

The reason why we got this error is because just before typing numDerive we had typed clear all which meant that all the variables that were there in MATLAB's memory has been erased. As a result of this, we were getting this error. We will cycle through that we had input earlier and we will see that when we started this particular module, we had a command a = 1. We also had a command trueVal = 1 / a^2, 1 / (1+a^2). We forgot to add this.

When we wrote this code, so let us go and type this in also and we will also have to type the trueVal okay. So, now when we run this code, we should not get any error okay. So, remember what we ended up doing and this is one of the more common errors that I will found. At least people new to MATLAB main. So, sometimes we forget that some of the variables have not been defined.

This is what how one should read, whenever we get an error, this states the first line after the error states that what the error actually is. The error is that the variable a is not defined. What we realized then is that earlier, all over codes they were running. The reason why they were running but they stop running now is because we had given the clear all command.

Then we realized that will be started this MATLAB session, we had given a = 1 and as well as commutated trueVal value. So, those when we insert in our MATLAB script, we have to run this. We are going to get the result as required. So, let me run this and we will get this result as required.

Now what I want to do is, why do I need to give this repeatedly in this particular manner, we want to now put it in a for loop and the for loop will become for i equal to say 2 to 4. In this case h is nothing but 10 ^ - i, approxVal, I will just copy and paste it over here and error I will just copy and paste it over here okay and I will say end, I will delete this, saved this.

When I run, I am going to get the same results again. Let me run this again for you numDeriv. And I am getting the same results as I had been getting before okay. So, next, what I want to do is, I want to see how this error actually varies for a fairly a large number of step sizes that we have taken. So instead of i going from 2 to 4, let me say i goes from say 2 to 7.

Let us say for example, okay what I am going to do also is, I am going to save my err and h in 2 vectors which I will call as hAll i -1 = h, errAll i - 1 = err. So these are just for storing the results. And plot error verses step wise, we want to do a loglog plot of hAll in the x axis, errAll in the y axis okay. And that is what we want to do and we want to see that this is a linear straight line okay. So now let us run this and we get a straight line as seen. What here, the reason why we get the straight lining is because. (Video Ends 13:06)

We are f `x. The error term is of the order of h which basically means when we plot the error against h on a loglog scale, it is going to be a straight line with slope of 1.The straight line, the slope is going to be the same as the order of the accuracy of the numerical scheme that we are going to use okay. Next what we will do is.

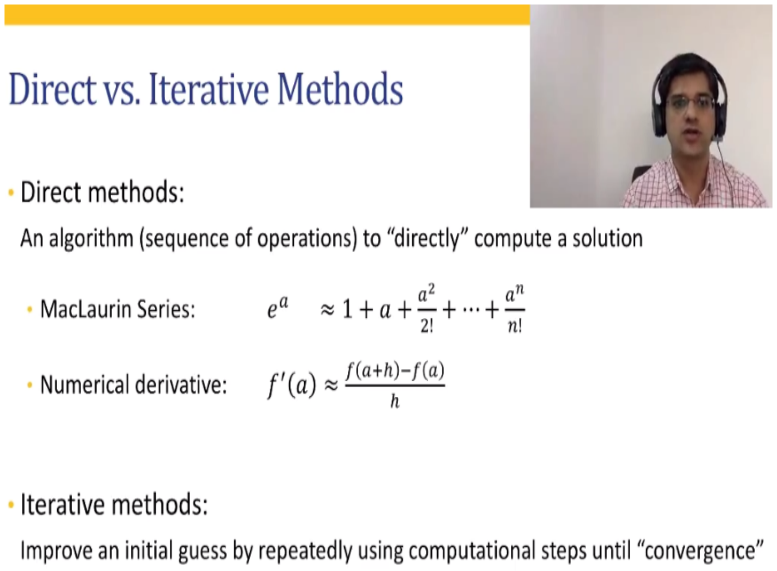
(Video Starts 13:33) That is instead of carrying this out until 7, let us do this for a larger number of terms. But will do it in steps of 2. Let us say and let us go to from 10 ^ -2 to 10 ^ – 14. I will just make small changes in this code over here. Clear all clc, close all and let me run the numDeriv code okay. And this is the result that we get.

As you see over here what we got when we decrease our h from 10 ^ - 2 to 10 ^ - 7 or in this case of put in ^ -8. We get the steady decrease in the overall error as the steps wise decreases. But after a certain point when we decrease the steps wise, error does not decrease any more. But the error starts to increase.

The reason why this happens is because of round off errors, remember what we said in the first lecture is that there is a machine precision; there is a least count for that machine at a certain point. The round off error that dominates and beyond that point the error increases. So, when we are going to do numerical derivates. We are going to start facing the issues of tradeoff between round off and truncation errors and there is a steps size which is going to best for numerical derivates.

This is something that we are going to cover in the next module. In module 3, on numerical derivatives and numerical integration. Here I wanted to demonstrate to you that there is something called truncation error, there is something called round off error and then that might be tradeoffs between these 2 errors which will result in some unique properties of some of the numerical methods. (Video Ends 15:34)

Okay so to go and recap what we did in numerical derivatives f `x, we calculated approximately as f (x) +h – f(x) divided by h. We saw that the truncation error decreases with h truncation error on a loglog scale, decreases linearly with h with a slope of 1. The machine precision determines how small h we can use. We cannot use h to be really small because of machine precision. The round off errors starts to accumulate. What we have covered so far is.
(Refer Slide Time 16:09)



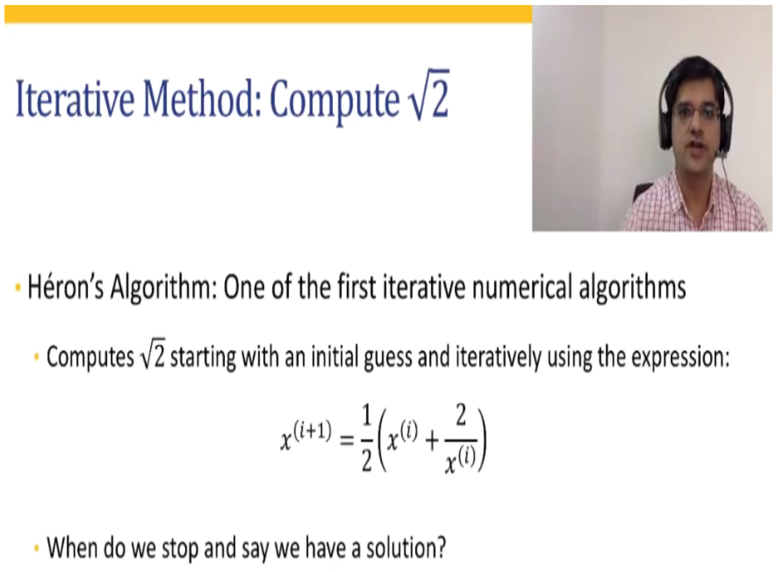
Direct vs. Iterative Methods

- Direct methods:
  An algorithm (sequence of operations) to "directly" compute a solution
  - MacLaurin Series: $e^a \approx 1 + a + \frac{a^2}{2!} + \cdots + \frac{a^n}{n!}$
  - Numerical derivative: $f'(a) \approx \frac{f(a+h) - f(a)}{h}$
- Iterative methods:
  Improve an initial guess by repeatedly using computational steps until "convergence"

What is known as direct methods? The first direct method was Maclaurin series expansion of e ^ a which was covered in the previous lecture. In this lecture we covered another direct method which was computation of numerical derivates. Direct method is nothing but an algorithm which directly computes a solution through a sequence of operations, a sequence of equations that we keep executing.

But we execute them once through and we are done with it that is how we are going to compute the solution using direct method. The second type of method that we are going to consider in this lecture is what is known as iterative methods. Iterative methods we will start with certain initial guess and we will use certain steps repeatedly or iteratively to improve that initial guess until we reach a convergence.

(Refer Slide Time: 17:05)



Iterative Method: Compute $\sqrt{2}$

- Héron's Algorithm: One of the first iterative numerical algorithms
  - Computes $\sqrt{2}$ starting with an initial guess and iteratively using the expression:

$$x^{(i+1)} = \frac{1}{2}\left(x^{(i)} + \frac{2}{x^{(i)}}\right)$$

  - When do we stop and say we have a solution?

Let us take an example of iterative method. As I mentioned before that example of initial method that we are going to take is what is known as Heron's algorithm. Heron's algorithm is arguably one of the first iterative numerical algorithms that were known to math. This was discovered in around in 17th century BC in Alexandria.

It computes square root of 2 using an initial guess. Let us say the initial guess was 0.5. We use that initial guess and we iteratively use this equation. So x = 1/2 (x + 2) / x is going to us the next solution and we will keep going with this iterations till we had convinced that we are reached the solution. So, let us go to MATLAB and do this okay.

(Video Starts 17:59) Will start with an initial value of x = 0.5 and we were going to iteratively use this equation which is the new guess of x is half of current value + 2 by current value. So x or let me just say xNew = 1/2 multiplied by x+ 2 / x okay. That is going to be xNew now because this is an iterative method.

We do not need to call it with another name, we can just call it with the same name as before in this particular example. So x is going to be average of x and 2 /x okay. So next value is 2.25. The value of that is 1.56, the value of that is 1.4142 okay. And if we repeat it multiple times, we are not going to get any more changes. So this is how an iterative method works.

We start with an initial guess and we iterate on that initial guess till the solution reaches a desired accuracy. Let us put this all in MATLAB, edit heron Algorithm, calculate square root of 2 using heron Algorithm. So let us have our initial guess as x = 0.5 say for i equal to let us say we expect of to 25 iterations okay. xNew is going to be equal to half multiplied by x +2 / x. Our error is nothing but abs of x -xNew and our x = xNew okay. So, let us type this of.

Maybe we should not do it by 25. We know that convergence very quickly. So let us just do it for 10 iterations and let us run this okay. So what we start off with the error was, 1.75 that is the error between or the difference between xNew and xhold is 1.75 once 0.68, 1, 0.14, 0.0007, 2 *10 ^ - 4 so on and so forth.

So this is probably where we can say that we need to stop and that is based on a tolerance value. So, let us go and change this a little bit. Let us say atol, the absolute tolerance value is 1.0 * e- 5 okay and instead of for, we will have a while loop. While err is greater than atol, we will keep repeating this these steps. So we will stop these steps when the error falls below the tolerance value, we also need to initialize the error. So we will just call the error, will just initialize this as 5 okay.

If we do not initialize this error okay, so, let us clear all heron Algorithms okay. So, as you can see we started with large enough error and we stop at resulting value here okay. And this is where we have use the while loop in order to compute the Heron's algorithm. If instead of x equal to say 0. 5, if we were to give a value of x = 0.1, let us see what happens.

Clear all Heron algorithm and again we converge to the desired solution in certain number of steps. As you can see over here, if the farther we are away from the true solution, greater the number of steps we are going to rewind okay. (Video Ends 22:38)

So, with that I come to the end this lecture. What we have discussed over here is, we look at direct method and then we look at iterative method and we look all the criteria when do we stop and say we have a solution. We will stop and say we have a solution is when the difference xNew and xold the absolute value of difference falls below a certain tolerance threshold.

What we covered in this lecture was a direct method to compute numerical derivative and saw the tradeoff between truncation and round off errors and then we saw an iterative method in Heron's algorithm okay. That completes our lecture 2 .3 and I will see you in the next lecture. Thank you and good bye.