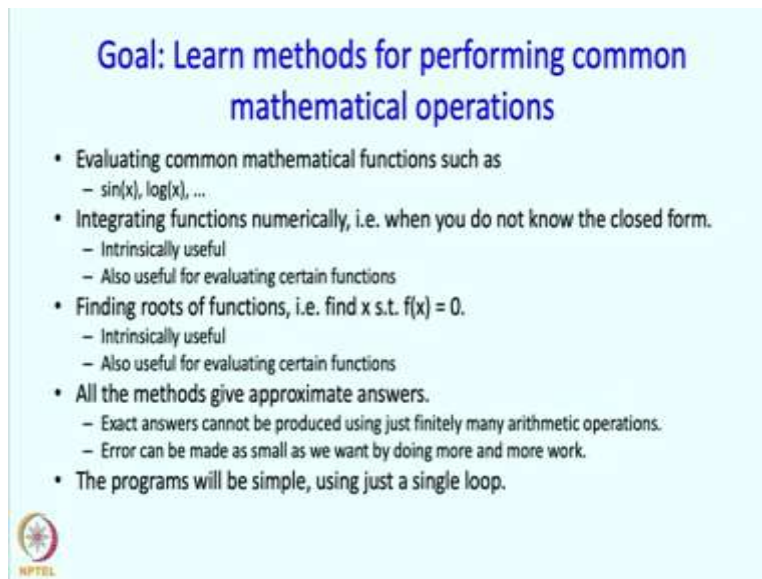


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 8 Part – 1
Computing Mathematical Functions

Computing common mathematical functions: Taylor series


Hello and welcome to the NPTEL course on an introduction to programming through C++. I am Abhiram Ranade, this is lecture sequence 3 of week 3. The reading for this is Chapter 8 of the textbook.

(Refer Slide Time: 0:35)



Goal: Learn methods for performing common mathematical operations

- Evaluating common mathematical functions such as
 - $\sin(x)$, $\log(x)$, ...
- Integrating functions numerically, i.e. when you do not know the closed form.
 - Intrinsically useful
 - Also useful for evaluating certain functions
- Finding roots of functions, i.e. find x s.t. $f(x) = 0$.
 - Intrinsically useful
 - Also useful for evaluating certain functions
- All the methods give approximate answers.
 - Exact answers cannot be produced using just finitely many arithmetic operations.
 - Error can be made as small as we want by doing more and more work.
- The programs will be simple, using just a single loop.



The goal of this lecture sequence is to learn methods for performing common mathematical operations. So we often need to evaluate mathematical functions such as $\sin x$, $\log x$ and we often also need to integrate functions. So these things can come up in Engineering or Science or even mathematics. And integration can of course be performed symbolically but sometimes it is not possible to get a closed form solution. In that case we can do numerical integration. So integration is useful intrinsically, but in addition we can also cast the problem of evaluating certain function as an integration problem. So we will see that soon.

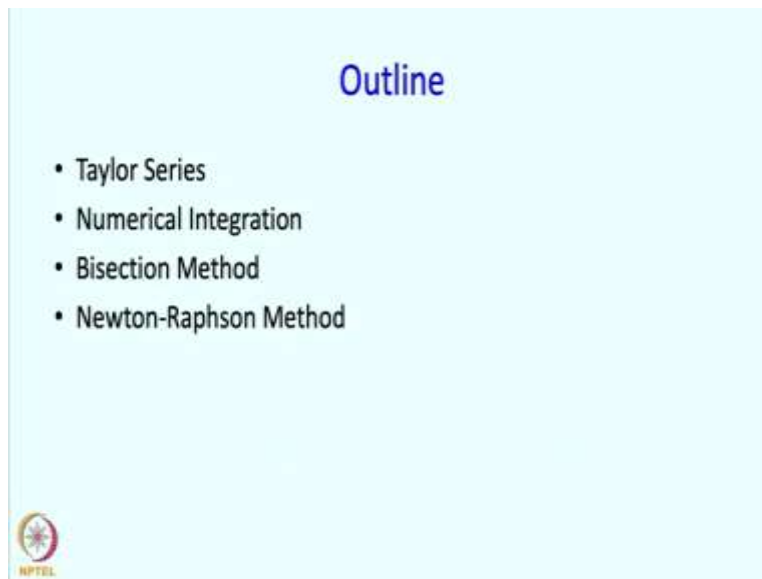
Finding roots is another important mathematical operation. And of course we have seen examples of such things, quadratic equations, solutions of simultaneous equations. So in general the goal here is to find an x such that $f(x)$ equal to 0 is satisfied. And of course we may have

several x that we might want to find simultaneously. But here let us consider we have an equation in 1 variable and we have to find a solution for that single variable. So this is an intrinsically useful problem, but in addition as we will see we can cast the problem of evaluating certain functions as root finding problem.

Now all these problems have methods which will give approximate answers. And in fact, exact answers are generally not possible for any of these problems. And by that I mean exact answers which use a certain finite number of arithmetic operations. However, the method that we describe will have the property that by doing more and more work we can reduce the error as much as we want. So we can make the error be as close to 0 as we want, but for that we might have to do more arithmetic operations. So there will be some kind of a tradeoff.

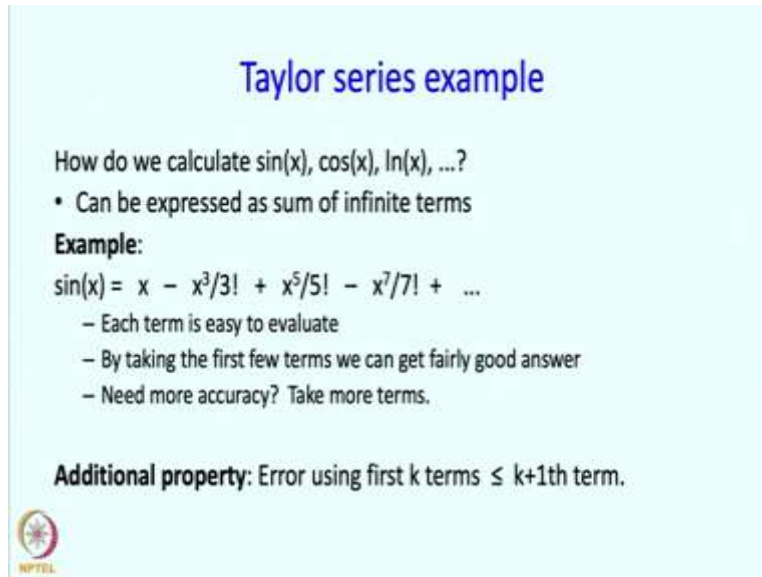
On the other hand, the programs that we will write for doing these computations will be extremely simple. They will just involve 1 loop and maybe 2, 3 variables. So in that sense they will turn out to be good problems for developing and practicing your programming skills.

(Refer Slide Time: 3:15)



So here is the outline of this lecture sequence. So first I am going to talk about methods based on Taylor series, then I will talk about numerical integration, then I will talk about a method called the bisection method and then a method called Newton-Raphson method.

(Refer Slide Time: 3:41)



The slide is titled "Taylor series example" in blue text. Below the title, it asks "How do we calculate $\sin(x)$, $\cos(x)$, $\ln(x)$, ...?". A bullet point states "• Can be expressed as sum of infinite terms". Under "Example:", the series for $\sin(x)$ is given as $\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \dots$. Three bullet points follow: "– Each term is easy to evaluate", "– By taking the first few terms we can get fairly good answer", and "– Need more accuracy? Take more terms.". At the bottom, it states "Additional property: Error using first k terms \leq k+1th term." and includes a small NPTEL logo in the bottom left corner.

So let me start with an example for a Taylor series. So the question over here is how do we calculate functions such as $\sin x$, $\cos x$, $\ln x$, $\tan x$ whatever square root x . Now, it turns out that these functions can be expressed as a sum of infinite terms. So let me give an example right away. So $\sin x$ can be written as $x - x^3/3! + x^5/5! - x^7/7! + \dots$, as you might guess $x^9/9! - x^{11}/11!$ and so on.

So as you can see the power of x increases by 2 as we go from the from one term to the next. And the factorial also increases by 2. So you can see that there is a very clear pattern. And the nice thing about this is that each term is easy to evaluate. So $x^5/5!$ can be evaluated by doing say 5 multiplications and 5 divisions. And another very important and nice thing is that even if we take the first few terms we do get a fairly good answer. And if we want a better accuracy we just have to take additional terms.

So this is very good, but in fact it has an even nicer property; which is that we can get a good estimate on how bad our error is. So suppose, we evaluate this sum to the first k terms. Then we can prove, we are not going to do it in that in this course, but it is possible to prove that the error is going to be smaller than the k plus 1th term. So if we use the k terms, then the next term actually just even tells us the bound on the error.

(Refer Slide Time: 6:09)

Let us compute $\sin(x) = x - x^3/3! + x^5/5! - \dots$

- In k^{th} iteration we will add k^{th} term to a running sum, initialized to 0.
- Let $t_k = k^{\text{th}}$ term of the series, $k=1, 2, 3, \dots$
- $t_1 = +x$, $t_2 = -x^3/3!$, $t_3 = +x^5/5!$, ...
- $t_k = (-1)^{k+1} x^{2k-1} / (2k-1)!$
- $t_1 = x$ is given as input.
- At the beginning of the k^{th} iteration: we will ensure we have t_k in variable T.
- At end of iteration k , we must have $S = \text{sum of first } k \text{ terms}$, $T = k^{\text{th}}$ term
- $t_{k+1} = (-1)^{k+2} x^{2k+1} / (2k+1)!$
- $t_{k+1} = t_k (-x^2) / ((2k)(2k+1))$
- In iteration k : add T to S. Then multiply T by $(-x^2) / ((2k)(2k+1))$.



$$t_1 = +x = \frac{x^1}{1!}$$
$$t_2 = -\frac{x^3}{3!}$$
$$t_3 = +\frac{x^5}{5!}$$

So let us try it out let us see how we can write a program to compute the value of $\sin x$. And you may observe perhaps and it is true the program is going to be similar to our calculation of e , the base of the natural logarithm that we did earlier. So let us compute $\sin x$ using this formula, this infinite sum. So our plan is going to be that in the k^{th} iteration we will add the k^{th} term of the series to a running sum which is initialized to 0. In some sense this is similar to the plan we made for calculating the value e , sometime ago. So, let us use t sub k to denote the k^{th} term of this series, for k equal to 1, 2, 3 and so on. So looking at this sum we can say that t_1 is equal to x or which we might in fact write as $x^1/1!$, t_2 is equal to $x^3/3!$, but there is a minus sign over here, t_3 is equal to $x^5/5!$ and I will just write a plus sign over here just to emphasize.

And so the point is that the first sign is plus, then there is a minus, then there is a plus and so on. So the plus and minus signs are going to alternate.

Alright, so these are the terms and now I can write this t_k as $-1^{(k+1)}$. Why? Why $k+1$? So, if I just wrote minus 1 to the power k then minus 1 to the power 1 would give me a negative number, so I will write minus 1 to the power k plus 1. And the minus 1 to the power increasing power will make sure that signs will alternate.

Then I have $x^{(2k-1)}$ let us check whether this is correct. So for k equal to 1 this gives 1 and then there is 1 factorial over here. So for k equal to 2, this $2k-1$ will give us 4 minus 1, 3 which is right upon 3 factorial and so on. So t_k is indeed this. So these are the terms that we are going to calculate in subsequent consecutive iterations. And we are going to add them to a running sum which we are going to initialize to 0 at the beginning. Then we know that t_1 is given to us as input. So t_1 is available to us at the beginning of the first iteration. So we would like t_2 to be available at the beginning of second iteration, and t_3 at the beginning of the third iteration and so on. So our plan is something like this at the beginning of the k th iteration, we will ensure somehow that we have t_k in variable capital T . We could have made a different plan, we could have said that we just want t_k minus 1 in the capital T at the beginning of the k th iteration. And in the k th iteration we will somehow calculate t_k . So that plan will also work, it will give rise to a different program. So here you are just going to stick to the plan that I have written down over here. Which is that at the beginning of the k th iteration we will ensure we have t_k invariable T .

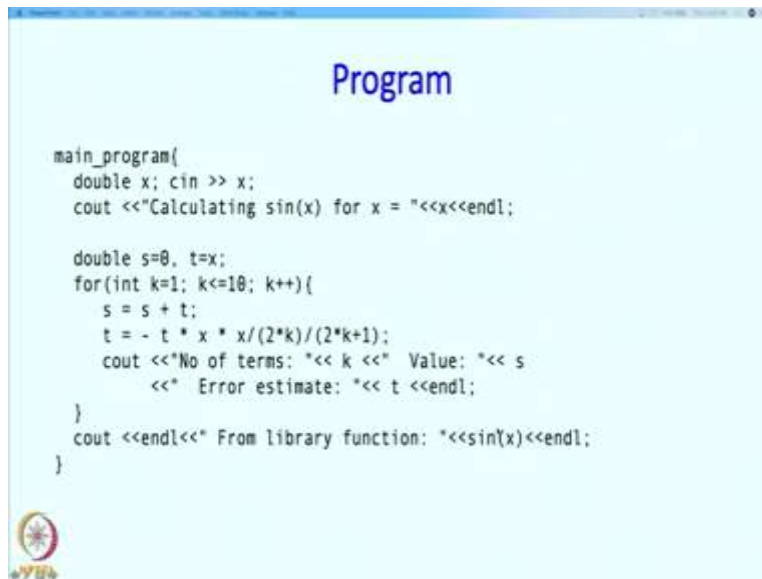
And then that means that at the end of iteration k what should we have? Well we have said at the top that S must equal the sum of the first k terms, so including t_k , and so t_k should get added to S . And for the k plus 1th iteration, we would like capital T to have t_{k+1} . So therefore, at the end of iteration k we must make capital T equal to the k plus 1th term or t_{k+1} . And what is this t_{k+1} , well I am going to substitute $k+1$ over here. So I am going to get a $k+2$ over here. And then this is going to be $2k-1$. So if I substitute k equal to $k+1$ over here, then I will get $2k+1$ over here. And then similarly, I will get $2k+1$ factorial over here.

So somehow, I have to get into variable capital T , this value, whereas originally in capital T I had this value. How do I do that? Well, if I observe these two values, I can see that they are not that different. So to what extent are they different? Well, here there is an extra multiplication by

minus 1. So I should multiply this term by minus 1 if I want to make look it like this. But that is not enough there is x to the power $2k$ minus 1 over here, and I want $2k$ plus 1 over here. So I should also multiply it by multiply it by x square. And I had $2k$ minus 1 factorial over here. So if I multiply it by $2k$ and also $2k$ plus 1 or rather it is in the denominator. So if I divide it by $2k$ and $2k$ plus 1 I will get this. Another way of stating the same thing is that t_k plus 1 is t_k times whatever factors I mentioned earlier. So I need a minus 1 and then I need an x squared and then I need to divide by $2K$ and $2K$ plus 1. So what needs to happen is that I already have t_k and that has to be modified a little by multiplying or dividing by these terms. Once I do that I will get t_k plus 1 in capital T as I want.

So in iteration T what is it that we are supposed to do? We are supposed to add the value that we have at the beginning in T into S. And then we are going to multiply T by this term over here. So T originally contain this. So when we multiply it by this term it will contain t_k plus 1, which is exactly what we wanted over here. So this allows us to write a program.

(Refer Slide Time: 12:23)



```
Program

main_program{
  double x; cin >> x;
  cout <<"Calculating sin(x) for x = "<<x<<endl;

  double s=0, t=x;
  for(int k=1; k<=10; k++){
    s = s + t;
    t = - t * x * x/(2*k)/(2*k+1);
    cout <<"No of terms: "<< k <<" Value: "<< s
        <<" Error estimate: "<< t <<endl;
  }
  cout <<endl<<" From library function: "<<sin(x)<<endl;
}
```

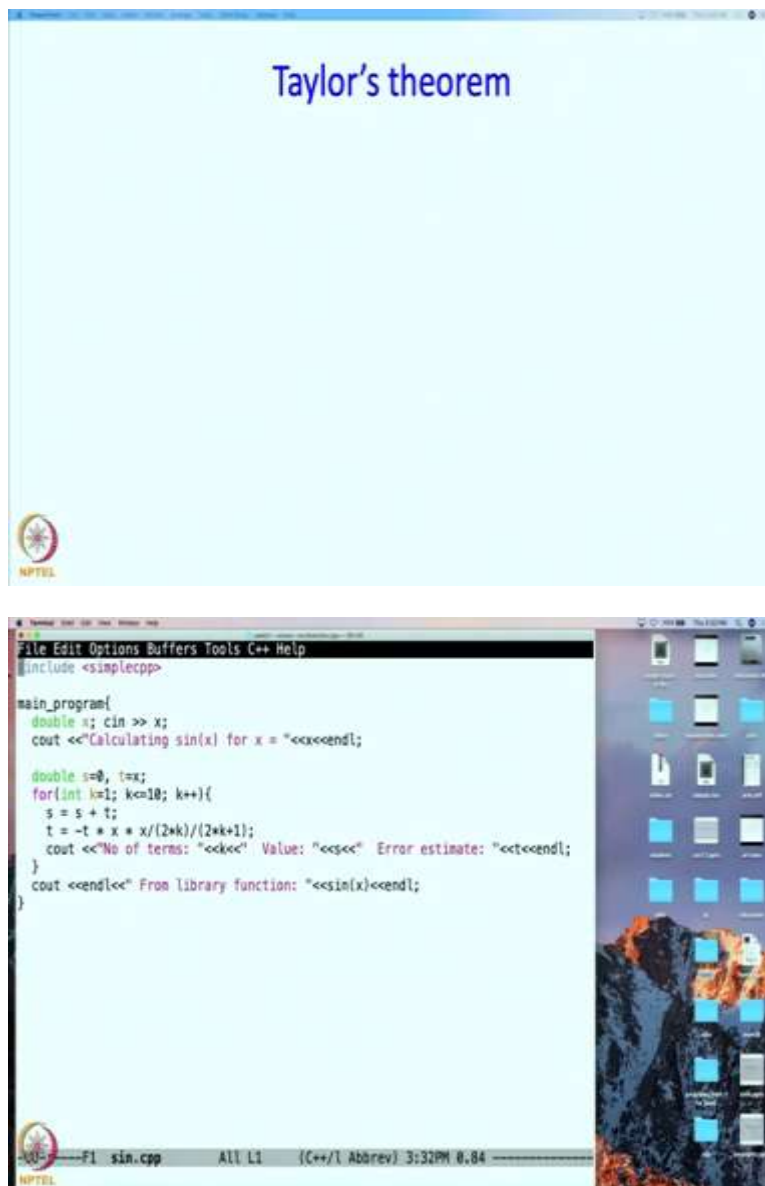
So we are ready to write a program. So we will declare a variable x and read it, this is the variable whose sin we want to calculate. So let us print out a message saying what exactly we are doing. And then we wanted a variable s in which we are going to accumulate the sum, and a variable t in which we are going to keep the successive terms of the series. So the first term is x , so which is how we initialize t . Then we are going to do this for 10 iterations. So we could have chosen different numbers, but 10 is as good as any. So in the first iteration we want sum to be initialized to 0 which we have done. And we want t to have the value of the first term of the series which also we have done. Then, inside the iteration we just said that we are going to add the term to the sum. So we write s equal to s plus t . And then we want to calculate the next value of the term. So the next value of the term we said can be obtained by multiplying the previous term by -1 and x square and then dividing by $2k$ and $2k+1$. So which is what we have done over here.

So at this point we have evaluated the sum to k terms. So we are going to print that. Just to see how the sum changes as the number of terms increases. So this is the number of terms k and we have evaluated the sum and that value is over here. That is not complete and we can also print out the error estimate. So as we said the error estimate is nothing but the new term value that is what happens to be the case for the $\sin x$ series so we are going to print that so when we run this program we will see how the sum is changing or how our calculation of $\sin x$ is progressing, but

we will also get an error estimate. So we will see that look the error in this current in this current value that we have computed is at most how much.

So that is the loop and then at the end what we are going to do this we are going to print out the value of $\sin x$ from the C++ library function. So there is a C++ library function called `sin`. And we apply it to x and we are going to get that value. And this is kind of the gold standard value or the value which we can regard as the correct value. And we will be able to check whether our value is the same or different from this value. So that is the end of the program.

(Refer Slide Time: 15:10)



The image consists of two parts. The top part is a slide with a light blue background and the title "Taylor's theorem" in blue text. The bottom part is a screenshot of a C++ code editor window. The code in the editor is as follows:

```
File Edit Options Buffers Tools C++ Help
#include <simplecpp>

main_program{
    double x; cin >> x;
    cout <<"Calculating sin(x) for x = "<<x<<endl;

    double s=0, t=x;
    for(int k=1; k<=10; k++){
        s = s + t;
        t = -t * x * x / (2*k) / (2*k+1);
        cout <<"No of terms: "<<k<<" Value: "<<s<<" Error estimate: "<<t<<endl;
    }
    cout <<endl<<" From library function: "<<sin(x)<<endl;
}
```

The screenshot also shows a Windows desktop with a mountain landscape wallpaper and several icons on the right side. The taskbar at the bottom displays the file name "sin.cpp", the current directory "All L1", and the system tray showing "(C++/1 Abbrev) 3:32PM 8.84".


```
1.41421 1.41421
1.41421 1.41421
1.41421 1.41421
1.41421 1.41421
1.41421 1.41421
1.41421 1.41421
~/Desktop/nptel/week3 : %
emacs -nw bisection.cpp

[1]+ Stopped emacs -nw bisection.cpp
~/Desktop/nptel/week3 : s++ sin.cpp
+ g++ sin.cpp -Wall -I/Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/si
mplecpp/include -I/opt/X11/include -L/opt/X11/lib -std=c++17
~/Desktop/nptel/week3 : ./a.out
0.1
Calculating sin(x) for x = 0.1
No of terms: 1 Value: 0.1 Error estimate: -0.000166667
No of terms: 2 Value: 0.0998333 Error estimate: 8.33333e-08
No of terms: 3 Value: 0.0998334 Error estimate: -1.98413e-11
No of terms: 4 Value: 0.0998334 Error estimate: 2.75573e-15
No of terms: 5 Value: 0.0998334 Error estimate: -2.58521e-19
No of terms: 6 Value: 0.0998334 Error estimate: 1.6859e-23
No of terms: 7 Value: 0.0998334 Error estimate: -7.64716e-28
No of terms: 8 Value: 0.0998334 Error estimate: 2.81146e-32
No of terms: 9 Value: 0.0998334 Error estimate: -8.22064e-37
No of terms: 10 Value: 0.0998334 Error estimate: 1.95729e-41
From library function: 0.0998334
~/Desktop/nptel/week3 : ||
```

So before we go to Taylor's theorem let us execute this program and see what happens. So this is our program, this is the program that we had typed. So let us try running it. So let me first compile it and let me run it. So let me give a value of sin so let us say I give the value 0.1. So let us see so what does it say? So for 0.1 the value calculated is 0.1 itself and the error estimate is already not so bad. The error is at this point just about 0.0001. So already our value is pretty good. But if we go to the next step, the error estimate goes down even drastically, very drastically and our value comes becomes something like 0.998. And then we can see from the third term after adding 3 terms our answer continues to remain the same basically. So very little improvement happens. So what has happened over here is that we have pretty much got the correct answer in just about 3 terms. And we can see that this is the answer that the library function is also going to give us. So in 3 terms we have got the correct answer.

(Refer Slide Time: 16:49)

```
0.1
Calculating sin(x) for x = 0.1
No of terms: 1 Value: 0.1 Error estimate: -0.000166667
No of terms: 2 Value: 0.0998333 Error estimate: 8.33333e-08
No of terms: 3 Value: 0.0998334 Error estimate: -1.98413e-11
No of terms: 4 Value: 0.0998334 Error estimate: 2.75573e-15
No of terms: 5 Value: 0.0998334 Error estimate: -2.58521e-19
No of terms: 6 Value: 0.0998334 Error estimate: 1.6059e-23
No of terms: 7 Value: 0.0998334 Error estimate: -7.64716e-28
No of terms: 8 Value: 0.0998334 Error estimate: 2.81146e-32
No of terms: 9 Value: 0.0998334 Error estimate: -8.22064e-37
No of terms: 10 Value: 0.0998334 Error estimate: 1.95729e-41

From library function: 0.0998334
~/Desktop/nptel/week3 : ./a.out
0.5
Calculating sin(x) for x = 0.5
No of terms: 1 Value: 0.5 Error estimate: -0.0208333
No of terms: 2 Value: 0.479167 Error estimate: 0.000260417
No of terms: 3 Value: 0.479427 Error estimate: -1.5501e-06
No of terms: 4 Value: 0.479426 Error estimate: 5.38229e-09
No of terms: 5 Value: 0.479426 Error estimate: -1.22325e-11
No of terms: 6 Value: 0.479426 Error estimate: 1.96033e-14
No of terms: 7 Value: 0.479426 Error estimate: -2.33373e-17
No of terms: 8 Value: 0.479426 Error estimate: 2.14497e-20
No of terms: 9 Value: 0.479426 Error estimate: -1.56796e-23
No of terms: 10 Value: 0.479426 Error estimate: 9.33311e-27

From library function: 0.479426
~/Desktop/nptel/week3 : ||
```

Let us just run it one more time, maybe for a different value so let us say 0.5. So here the error is a little bit worse after the first iteration and also the second iteration, but from the third iteration onwards the error seems to be really negligible. And from the fourth iteration the value is exactly the value that the library function gives. And of course the error is small. So this seems pretty good. This series is giving us quite good results.

(Refer Slide Time: 17:37)

Taylor's theorem

Taylor's theorem: Many interesting functions can be written as:

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)(x-x_0)^2}{2!} + \frac{f'''(x_0)(x-x_0)^3}{3!} + \dots$$

if x is close to x_0 , i.e. $|x - x_0| < \text{"radius of convergence"}$.

We can often choose x_0 such that it is possible to easily evaluate f and its derivatives at x_0 .


Example: $f(x) = \sin(x)$: Choose $x_0 = 0$.

$$f(0) = \sin(0) = 0, \quad f'(0) = \cos(0) = 1,$$
$$f''(0) = -\sin(0) = 0, \quad f'''(0) = -\cos(0) = -1$$

...

$$\text{So } \sin(x) = 0 + x + 0 - \frac{x^3}{3!} + 0 \dots$$

Radius of convergence happens to be infinite for $\sin(x)$ at $x_0 = 0$.



So now, let me talk in more general terms. So we just did something for sin, turns out we can do something similar for many other functions. So Taylor's theorem is what allows us to do this. And Taylor's theorem says that many interesting functions can be written in the following form. So $f(x)$ can be written as $f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)(x - x_0)^2}{2!} + \frac{f'''(x_0)(x - x_0)^3}{3!} + \dots$, so the derivative of the function evaluated at x_0 , times $x - x_0$, plus the second derivative of f evaluated at x_0 , times $(x - x_0)$ squared upon 2 factorial plus $f'''(x_0)$ times $(x - x_0)$ cubed upon 3 factorial and so on.

Now this equality holds assuming we add up infinite number of terms. If x is within some particular distance of x_0 , and this distance is called the radius of convergence. So this is some distance that we can estimate mathematically and so long as x is close to x_0 as compared to this distance, then this theorem, this expression is correct.



Now, why does this expression help us to evaluate $f(x)$? I mean $\sin x$? Why does it, why would it help us to evaluate $\sin x$? The point is that is you can often choose x_0 such that not only is it possible to evaluate $f(x_0)$, but also we can do it for $f'(x_0)$ and so on. f' , f'' , and f''' . So we can often choose x_0 such that it is possible to easily evaluate f and its derivatives of x_0 . So let us take the example of sin, so we will choose x_0 equal to 0. So then, what we know? So we know that $f(0)$ or $\sin(0)$ is 0. $f'(0)$, what is the derivative of $\sin x$? It is $\cos x$. So $\cos(0)$ is actually 1. So again we could evaluate the derivative of $f(x)$ at x_0 equal to 0. What about the second derivative? The second derivative of sin is $-\sin$ and again at 0 $-\sin$ is 0.

The third derivative is $-\cos$. But at 0 it is $-\cos$ of 0 is -1 . After this the pattern is actually going to repeat, the fourth derivative is going to be \cos and so it is going to be $0, 1, 0, -1, 0, 1, -1$ and so on. So where does that leave us for \sin ? So $\sin(x)$, if you substitute $x=0$ in this you are going to get $f(0)$, so you will get a 0 over here. Then over here you are going to substitute 0 , so $f'(0)$ is you are going to get a 1 over here. And this is 0 so you are going to get an x over here. So you got an x . Then for f'' , $f''(0)$ is 0 . So this term will go away, f''' is minus 1 , so we are going to get a minus term over here and there is going to be an x minus x^0 cube term. But x^0 is 0 so you are going to get x^3 upon 3 factorial. And as you can see the even terms are going away and the odd terms are coming out with alternating signs which is exactly what we started off with before. So what we did was exactly an application of Taylor's theorem. And it turns out that this will work no matter what value you substitute. So if you go to large enough terms this will get very very close and in the limits it will become exactly $f(x)$.

(Refer Slide Time: 21:37)

What we discussed

- Taylor series are available for many many functions.
 - Exercise: construct the series and write the program to calculate $\cos(x)$.
- Other infinite expressions have also been used for calculating math functions, e.g. infinite products, continued fractions.
 - See exercises in book.



Alright, so what have we discussed? We have discussed that Taylor series are available for many functions. And in fact I will ask you as an exercise to construct the series and write down the program to calculate $\cos(x)$. And Taylor series is just one infinite expression which has been used for calculating math functions. But people have written similar infinite expressions, say infinite products or continued fractions. And these can also be used to evaluate mathematical expressions. So these are discussed in the book and in the exercises in the book. So we will stop here.