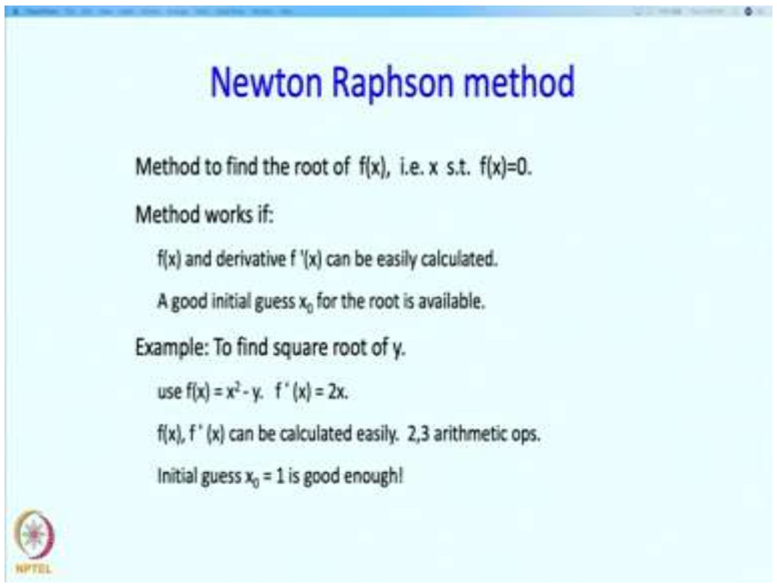
An Introduction to Programming through C++

Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 8 Part – 4
Computing Mathematical Functions
Newton Raphson Method

In the last segment we discussed the bisection methods for finding roots. This is a very simple method. In this segment we are going to discuss Newton Raphson which is perhaps the most sophisticated among the methods that we have discussed in this lecture sequence.

(Refer Slide Time: 0:38)



Newton Raphson method

Method to find the root of f(x), i.e. x s.t. f(x)=0.

Method works if:

- f(x) and derivative f '(x) can be easily calculated.
- A good initial guess x_0 for the root is available.

Example: To find square root of y.

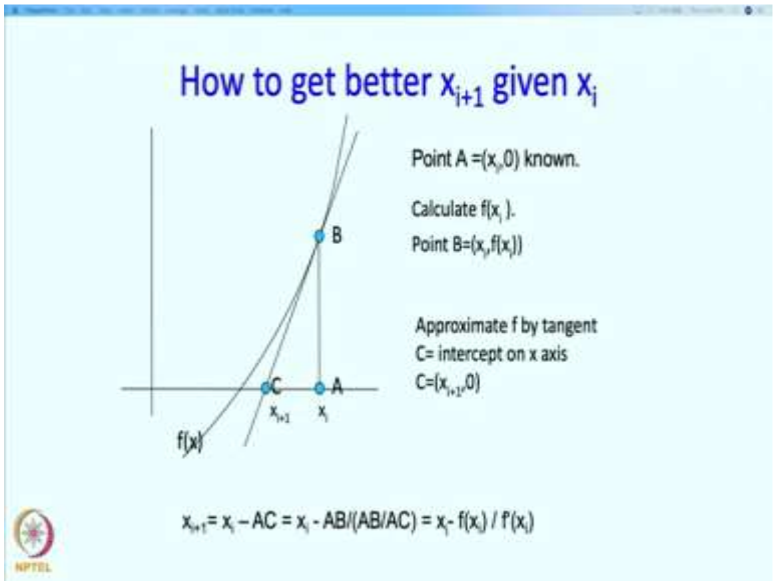
- use $f(x) = x^2 - y$. $f'(x) = 2x$.
- f(x), f '(x) can be calculated easily. 2,3 arithmetic ops.
- Initial guess $x_0 = 1$ is good enough!

So the Newton Raphson method, it is again a method for finding the root of some function f(x), that is finding x such that f(x) equals 0. This method is going to work if the following conditions are met. We should be able to evaluate f(x) and its derivative f'(x). And we should have available a good initial guess. So sometimes this condition is not quite necessary, but it definitely helps. So we should, if we are close to the root then this method will quickly take us even closer very close. But if we are far then this method occasionally may wander around too much.

So again we are going to use this method to find the square root of y. So as before we are going to say that our function is f(x) is x^2-y. Now, in this case we want a derivative as well and we might as well write this down immediately, so f'(x) is to 2x. So notice that given an x value, we can evaluate f(x) as well as f'(x) quite easily in 2, 3 arithmetic operations.

We also need an initial guess and for this square root finding problem it turns out that x0 equal to 1 is a good guess. Actually for this problem the guess requirement is easily met. I think essentially (any) any number will be ok.

(Refer Slide Time: 2:24)



How to get better x_{i+1} given x_i

Point A =$(x_i,0)$ known.

Calculate $f(x_i)$.

Point B=$(x_i,f(x_i))$

Approximate f by tangent

C= intercept on x axis

C=$(x_{i+1},0)$

$x_{i+1} = x_i - AC = x_i - AB/(AB/AC) = x_i - f(x_i) / f'(x_i)$

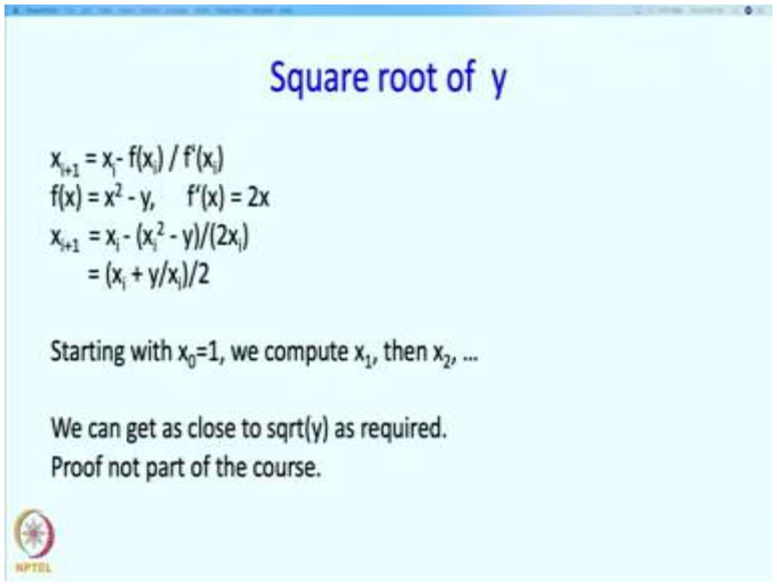
So this is again and iterative method and the key step over here is to get a better approximation of the root than the one that has been given to us. So let us say we have been given xi, which is our current approximation to the root. So now, I am going to tell you how I can (get) how we can get a better approximation and that better approximation we are going to call xi+1. So initially we know this point A. So A is xi, 0, x coordinate is xi and y coordinate is 0. It is a point on the x-axis. Next we are going to calculate f(xi). So f(xi) is just f(xi) is just the function value at this point or it is this height upto this point. So in fact it is exactly this point. The B point is the point over here. Because its x coordinate is the same as that of A and the height is f(xi). So we have so we can determine the point B if we know xi because we can get the coordinates of point B.

Next step is we are going to approximate this function by a tangent at this point. So we are going to approximate it by a tangent and we'll ask where does the tangent meet the x-axis. So let us say C is the intercept of the tangent on x-axis. So C is going to be our new guess. So its coordinate is going to be (xi+1, 0). I have not told you exactly what that value is but that is what it is going to be. And of course the Y coordinate is going to be 0. So all that remains now is that I have told you how to geometrically get to C. But I have not told you how to algebraically get to C. So that is all that remains. So you can observe that xi plus 1 is this distance from origin to C. So it is this distance origin to A minus AC. So that is what this first equality case mean, xi+1=xi-AC. Now I am going to write is AC in this funny form. I am going to write it as AB upon AC. Of course if you simplify this you will get AC. But then why did I write it in this funny form. Well I did that because I know what the value of AB is for one thing. What is the value of AB? Well it is just f(xi) so I am going to put f(xi) instead of this AB over here. What is AB upon AC? AB upon AC is this height upon this and that is simply the slope of this tangent or it is the value of the derivative at point xi. So that is what I have written down over here. So AB upon AC is f'(xi).

So what have we got? We now have a formula for xi plus 1, given in terms of xi, f(xi) and f'(xi). So xi+1=xi-f(xi)/f'(xi). So this is the formula which we are going to use to get to a better approximation. And in this picture you can see that C is likely to be a much much better approximation. In fact, if the tangent over if the curve over here is almost a straight line. Then we would get to the root at one shot. But of course it will not be a straight line in general, but you can see that we are likely to get closer to the root.

Alright, so let us now use this formula xi+1=xi-f(xi)/f'(xi) to find the square root of a number. So square root of a number y. So that is our formula and we want to find a square root of y.

(Refer Slide Time: 6:59)



Square root of y

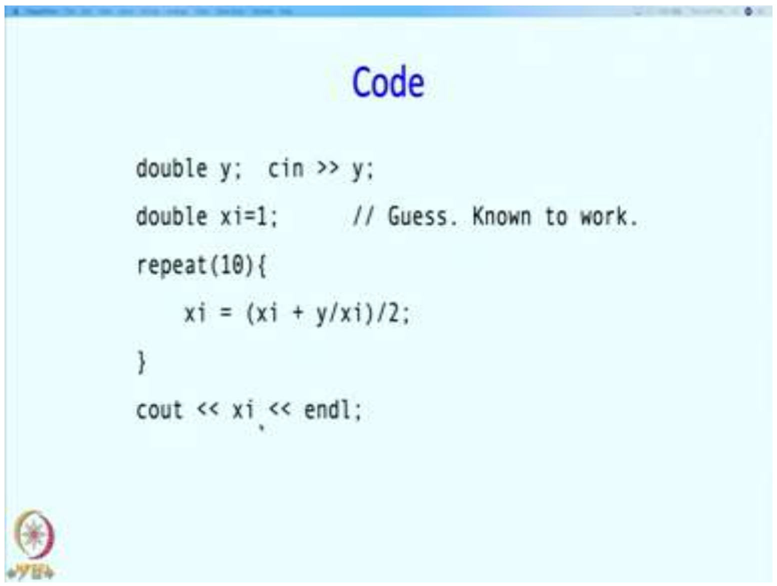
$x_{i+1} = x_i - f(x_i) / f'(x_i)$
$f(x) = x^2 - y, \quad f'(x) = 2x$
$x_{i+1} = x_i - (x_i^2 - y)/(2x_i)$
$= (x_i + y/x_i)/2$

Starting with $x_0=1$, we compute x_1, then x_2, ...

We can get as close to sqrt(y) as required.
Proof not part of the course.

So for that we said that our function f(x) is x^2-y and we said that its derivative is 2x. So all that remains is now to substitute these quantities into this expression. So what do we get for f(xi)? xi^2-y and for f'(x) we get 2x. So we get xi+1=xi-xi^2-y/2xi. And this if you substitute is going to turn out to this, why? So xi squared upon 2xi is going to give me xi upon 2. So that will take off half of xi from this so I am going to get an xi but notice that there is a upon 2 over here. And what about this (term) this side? So here I am going to get a y, but there is an upon 2xi over here. But this y on the this minus sign and this minus sign will cancel each other out. So I am going to get y upon xi with that 2. So xi+1 is just going to be (xi+y/xi)/2. So our basic iteration is quite simple. So starting with x0 we are going to use this to compute x1, then compute x2 and so on. And it turns out that if we keep on doing this we can get as close to square root of y as required. The proof is a little bit involved but calculus a little bit of Calculus will help. Certainly first year calculus of Science and Engineering should be adequate to prove this. But anyway it is not a part of the programming course, the proof is not the part of programming course.

(Refer Slide Time: 8:54)

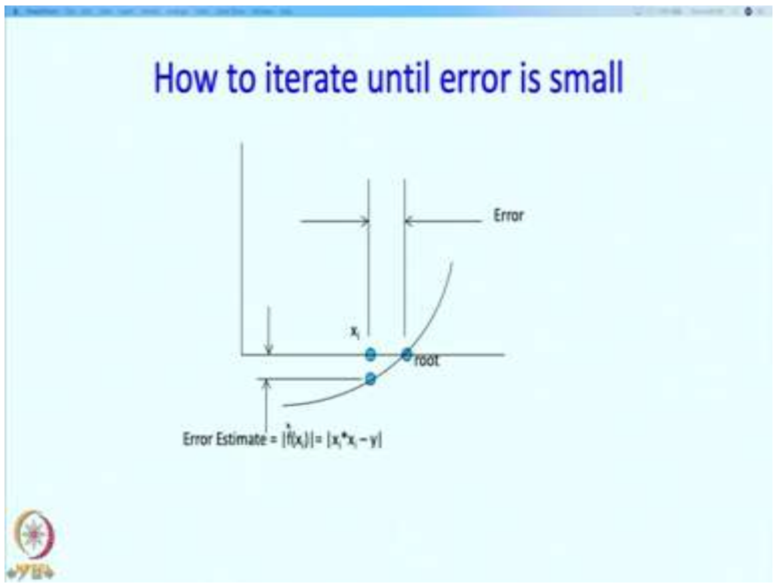


Code

```
double y;  cin >> y;
double xi=1;      // Guess. Known to work.
repeat(10){
    xi = (xi + y/xi)/2;
}
cout << xi << endl;
```

So the code is very simple so we are going to have y in which we are going to read the value whose square root we want to calculate. Then our xi is going to be kept in a variable xi which we are going to start off with 1. This is a guess and as I said for this square root almost anything works and we are going to stick with x because this does work. And then first I am going to repeat 10 times. xi equal to our basic iteration. So this is the old value of xi that we are going to use. We are going to calculate the new value of xi. And Right away we are going to put it into xi. So we are just going to do this 10 times. And at the end we are going to print out the result. So this is this is fairly straight forward.

(Refer Slide Time: 10:03)



How to iterate until error is small

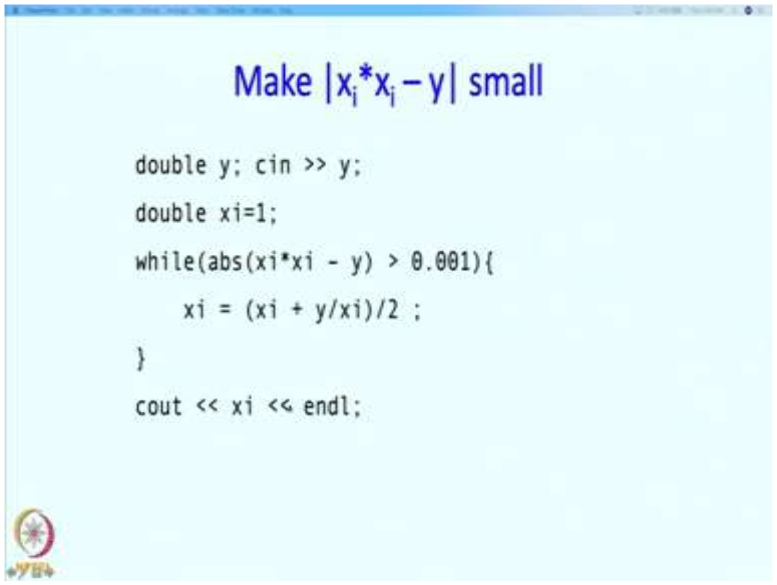
Now we have said that we are going to do to this 10 times. So if we are going to do this 10 times, we do not really know how much the error is going to be. Ideally we would like to say that look do this until the error is small. Let us see if you can get not exactly that condition but some similar conditions.

So here what I have done is I have plotted my function f(x). So this curved line is my function f(x). What is that function? xi squared minus y, that is my function. This curved line. Now my current xi is this. The actual xi that I want the actual root that I want is this. So I want this distance because that is my error. Or I want an estimate of this distance. I can never get the exact error because if I got the exact error I would just add it to my current estimate and then I will get the exact answer. So that is that is not we are not going to be so lucky.

So, but if we get an upper bound on the error that is also fine. But here even determining an upper bound on distance is a little tricky at least not immediately obvious. Instead of that what we can see is that this distance is relatively easy to calculate, well almost. So how much is this distance? So, so this function is xi squared minus y. So it is the distance between the square of my current guess and the root. So it is sort of an error it is sort of the vertical distance from this point to this curve. What I really want is the horizontal distance from this point to this curve. Whereas, this value is the vertical distance from this point to this curve.

So I can do that, I can use that if the slope is not too bad and in the case of the square root it is ok I guess. So if the slope is not too bad. Then I can in fact get a good estimate from this.

(Refer Slide Time: 12:06)

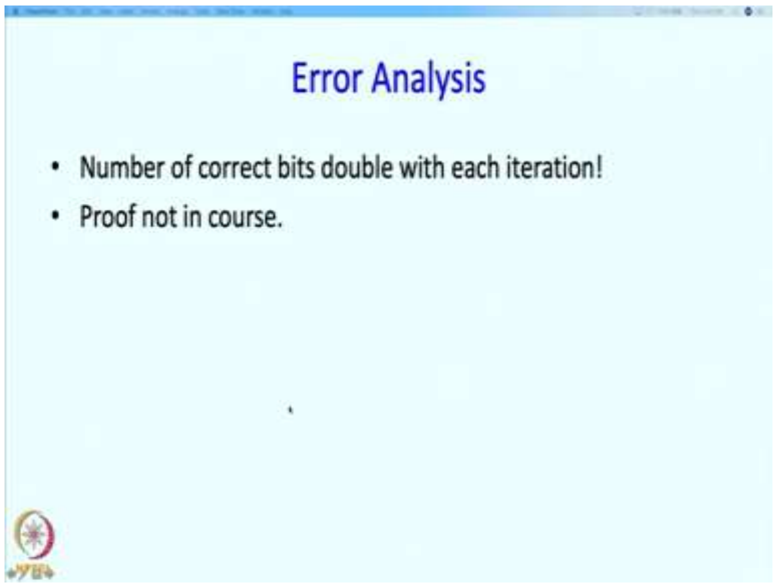


Make $|x_i*x_i - y|$ small

```
double y; cin >> y;
double xi=1;
while(abs(xi*xi - y) > 0.001){
    xi = (xi + y/xi)/2 ;
}
cout << xi << endl;
```

So here is how my function might change. So my program will again ask for y, it will again set xi to 1. But now instead of running for 10 iterations it is going to say, okay have I come close to this curve, has my xi come close to this curve along the y axis, in the y direction. I really want in the X direction, but have I come close in the Y direction and if it is if I am not come close and whatever error we want we can put over here then we are going to iterate again and again. Until this condition becomes false. At the end of it we know that our vertical error has become smaller than this. And even though we want the horizontal error to become small that is ok, at least we are making some effort. But not but you have to remember that this is not the actual error this is the some kind of a very heuristic estimate of what the actual error might be. So in any case at the end of it we are going to put out our current our current xi as our answer.

(Refer Slide Time: 13:16)

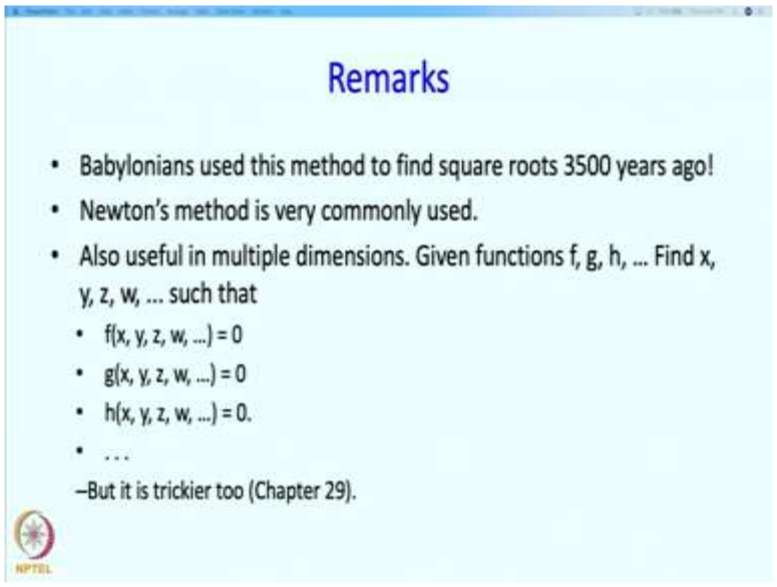


Error Analysis

- Number of correct bits double with each iteration!
- Proof not in course.

Now, the error analysis can be done and in fact it turns out that the number of correct bits essentially doubles with each iteration that is a remarkable result. It is not its proof is not easy. But with a little bit of effort and you should be able to understand it, it is given in several places. But let me just tell you what that result is saying. So it is saying that suppose you start off with 1 bit correct, then if you run it once you will get 2 bits if you run it twice you will get 4 bits, if you run it three times we will get 8 bits, if you run it 4 times you will get 16 bits, if you run it 5 times you will get 32 bits, if you run it 6 times you will get 64 bits correct. So basically it says that you have to run this just about six times only 6 iterations have to happen. So that is that is really fast convergence to the correct answer. It is not exactly doubling, so it maybe it will take 7 or 8, but not much more than that.

(Refer Slide Time: 14:34)



Remarks

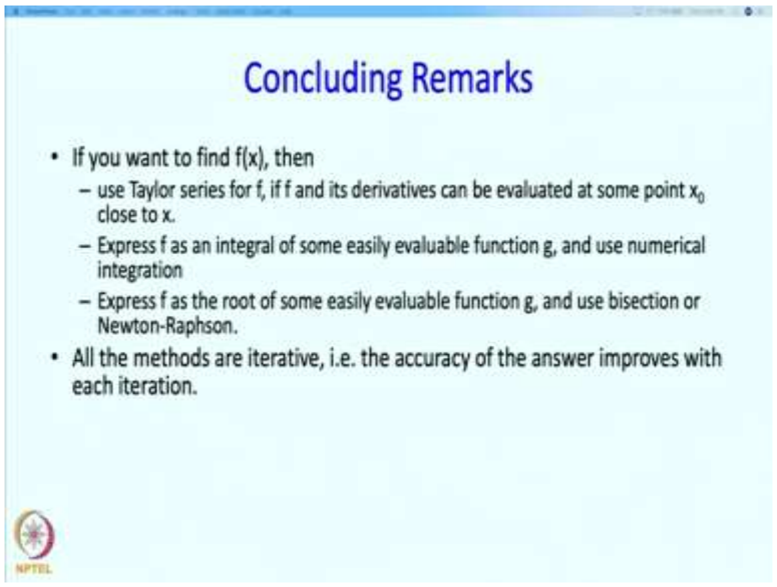
- Babylonians used this method to find square roots 3500 years ago!
- Newton's method is very commonly used.
- Also useful in multiple dimensions. Given functions f, g, h, ... Find x, y, z, w, ... such that
  - f(x, y, z, w, ...) = 0
  - g(x, y, z, w, ...) = 0
  - h(x, y, z, w, ...) = 0.
  - . . .

  –But it is trickier too (Chapter 29).

So let me make some remarks on this, some interesting points. So one interesting point is that Babylonians used this method to find square root 3500 years ago! They did not, they did not know this whole tangency, and they do not that does not seem that they knew that they could be, this could be used to find the roots of any function for which we can evaluate f and f'. So for that for that realization we have to wait for Newton and Raphson. But 3500 years ago getting square roots as accurately as you want is an amazing achievement. Now, I should mention that Newton's method is very commonly used in many many places. And it is also useful in multiple dimensions. So, what is the multiple dimensions problem? We are given several functions of several variables. So we want to find values to x, y, z, w. One set of values to x, y, z, w such that this equation is satisfied, this equation is satisfied, this equation is satisfied. So x, y, z, w the set the set of numbers defines a point in high dimensional space. And what we want is a single point in this high dimensional space at which all of these functions f, g, h get take the value 0.

So essentially we want to find a point in high dimensional space which is the simultaneous route of all these functions f, g, h. So you can use Newton's method to solve such problems however, solving such problems is very tricky. So you have to use Newton's method very carefully and sometimes it just does not work. But anyway Newton's method is a great method, very very useful and it is also remarkable that one part of it or aspect of it was known to Babylonians 3500 years ago.

(Refer Slide Time: 17:11)



Concluding Remarks

- If you want to find f(x), then
  - use Taylor series for f, if f and its derivatives can be evaluated at some point x_0 close to x.
  - Express f as an integral of some easily evaluable function g, and use numerical integration
  - Express f as the root of some easily evaluable function g, and use bisection or Newton-Raphson.
- All the methods are iterative, i.e. the accuracy of the answer improves with each iteration.

So that concludes this lecture sequence, let me just make some final remarks. So in general what we talked about here or sort of the major problem that we talked about here is to evaluate f(x), where f(x) is some interesting mathematical function, but which is not easy to calculate directly. So if you want to do that then you can use Taylor series. If f and its derivatives can be evaluated at some point x0 of your choosing. And that point should be close to x or x should be in that radius of convergence of that point x0 for that function f.

Then we also said that we can express f as an integral of some easily evaluable function, not easily integrable, easily evaluable function. Then we can numerically integrate this function g, and we can calculate f. We can also calculate f we can express it as the root of some easily evaluable function g, and then we can use the bisection method or Newton Raphson method.

We also said and it is worth definitely keeping an important thing in mind that all these methods are iterative. So the accuracy of the answer improves with each iteration. So we do not get the exact answer, but we can get answers which are as close to the actual answer as we want and since we are representing numbers only to a small number of bits or say a large number of bits whatever it is still we do not really care for answers which are more precise than what is possible in that many bits. And therefore, these methods are actually quite good enough for using on a computer. We do not really have to worry about deriving some kind of exact methods because there is no such exact thing on a computer anyway, at least if we use the standard data types.

Now, we also said that ideas of ideas if these chapters are very fundamental. And in fact, if you do numerical calculation you will see them again and again and even in the book they appear in two places chapter 19 and chapter 29. And I would also like to say that the ideas might be mathematically involved, the programs are actually quite simple. Once you understand what is to be done, the programs are quite simple. And they essentially have one loop which could be a for, which you could use as a for loop or you could use, even the repeat loop sometimes. But certainly a while loop or a for loop and I suggest you get practice in writing all these things using for loops and while loops.

And that was also one of the motivations we have had in doing all this to get practice and learn programming and get practice in writing loops, writing simple loops with a small number of variables. So that concludes this lecture sequence. Thank you.