

Problem Solving through Programming in C
Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

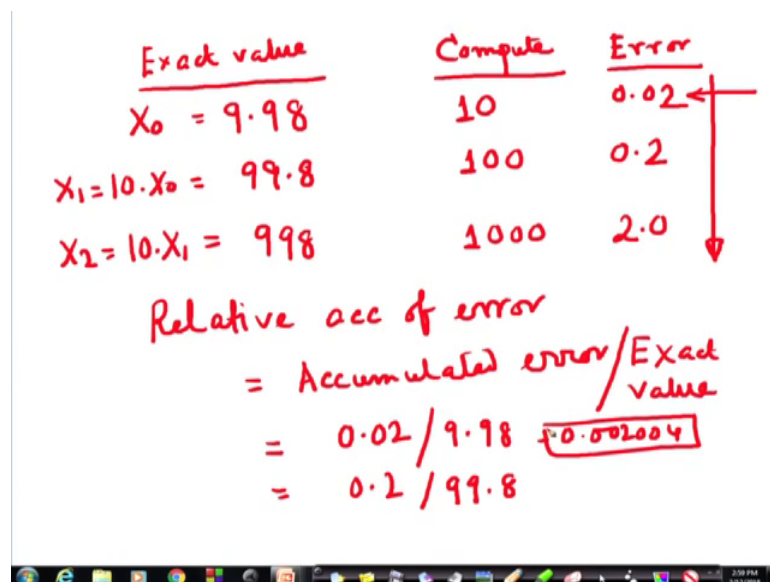
Lecture-50
Bisection Method

We were talking about errors and round off errors and percentage errors. Now, I will briefly show you that if we commit an error how that error continuously gets accumulated and ultimately has a much larger effect.

(Refer Slide Time: 00:46)

<u>Exact value</u>	<u>Compute</u>	<u>Error</u>
$X_0 = 9.98$	10	0.02
$X_1 = 10 \cdot X_0 = 99.8$	100	0.2
$X_2 = 10 \cdot X_1 = 998$	1000	2.0

Relative acc of error
= Accumulated error / Exact value
= $0.02 / 9.98$ 0.002004
= $0.2 / 99.8$



For example, suppose the exact value of some variable the first instance is 10 all right. And suppose no sorry suppose the exact value is 9.9 8 and we compute 10 all right. So, the inherent error first the error actually is 0.0 2 right.

Now, as we go on iterating suppose x_1 is 10 times X_0 , then the exact value should be 99.8, but here we will get 100 ok, I will say suppose it is 100. So, you can see that the error has increased to how much 0.2. Now if the next iteration X_2 is again 10 times X_1 then it will be 998 whereas, the computed value will be 1000. So, the error is becoming 2.

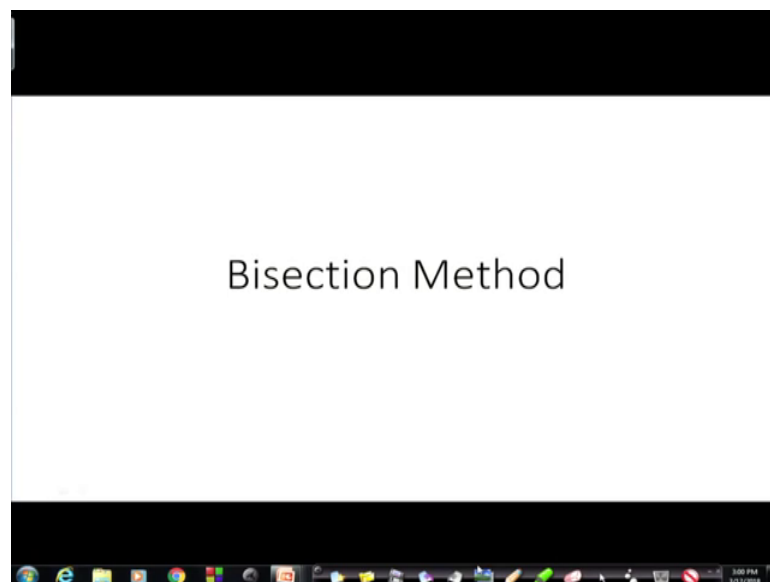
So, you see how with if we start with an inherent error how that error accumulates over time? Ok. So, we can say a very important term is relative accumulation of error, which is accumulated error accumulated error divided by exact value for that iteration.

For example in the first iteration it was the error accumulated error was 0.0 2 divided by the exact value which was 9.9 8, but later on it became 0.2 divided by 99.8. So, this is this was 0.0 0 2 0 0 4, but whatever that is that is not that important I do not want to confuse with you with this.

Now there are some cases where this accumulation of error actually goes on increasing ok. If the rate of accumulation error decreases if the rate of accumulation of error decreases or if the rate of accumulated error increases, but the rate of relative error decreases, then we call it a stable algorithm.

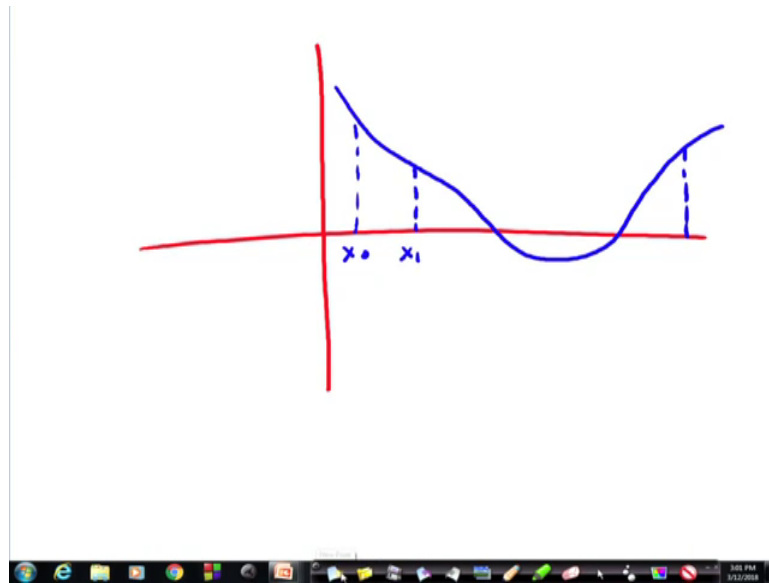
However, I am not going into the details and the intricacies of this, but this is just to give you an idea how the error propagates through iterations. And so, we must be very conscious about the rate of increase of this error all right.

(Refer Slide Time: 04:50)



With these words we move to the algorithm which we are planning to discuss in this lecture that is a bisection method.

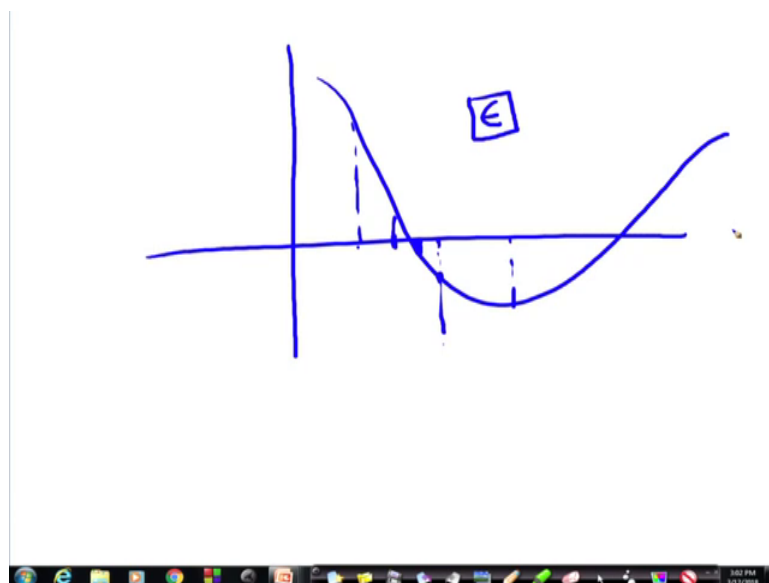
(Refer Slide Time: 05:10)



I have already told you that the bisection method is given a particular function some function, on this x and y axis if I have some function that moves, in this way then we start with any two points any two points arbitrary points here and maybe here, now these two points will not do.

Because if I select these two points then both of them are positive so, that if I had selected this point for example, these two points x_0 and x_1 that would not have served my purpose because I do not know where the root is there or not.

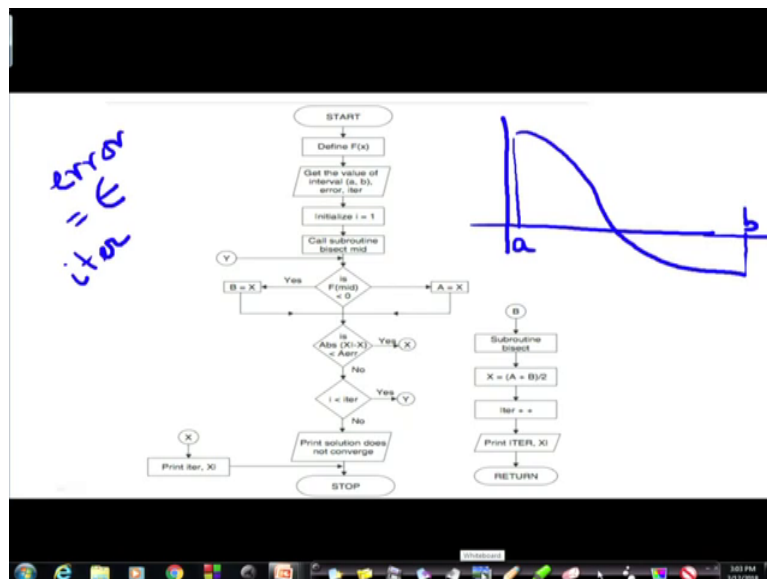
(Refer Slide Time: 05:52)



So, I would rather select two points, which are of opposite signs and therefore, I know that somewhere in between the root lies. So, I will take the midpoint of this somehow here if this point is negative then I will keep the positive fixed and I will find out the value of y. So, these two are opposite signs. So, the root must be somewhere here.

So, in that way I come to this one and find out the value of the root here in that way I go on dividing it till I come very close to the root as is being shown here very close to the root. Now how close that will depend on my decision that is the basic approach of bisection method.

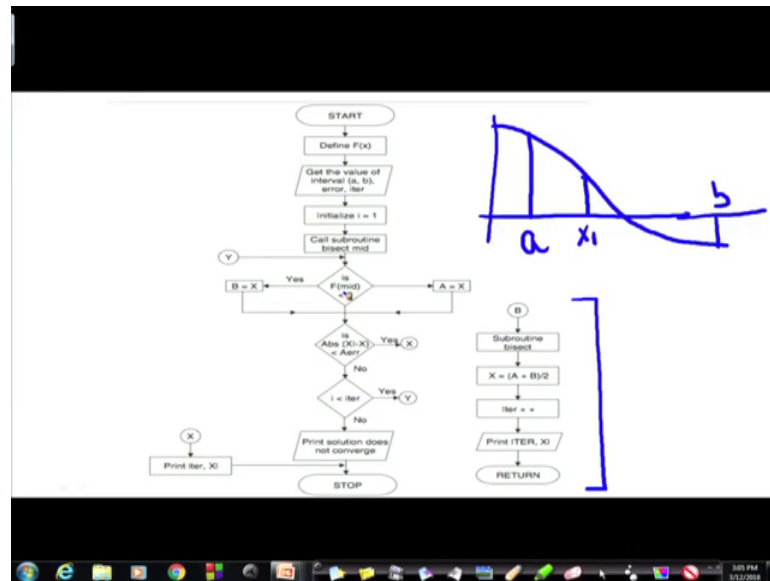
(Refer Slide Time: 06:51)



So, with that let us try to have a look at the algorithm, therefore, first we start we define the function we define the function $F(x)$ and get the value of the interval $A B$ that is there is a function. So, here there is a function and the function can be long enough. So, I take the limits that I have to find out the root within this interval a and b and I find out how much error is required. So, how much error is acceptable.

So, this error is the allowed error is the epsilon that I was talking of and also the number of iterations the number of iterations, because it may be that in some case I am not finding the root, because I am going on looking at say for example, this sort of scenario and my a is here my b is here.

(Refer Slide Time: 08:00)

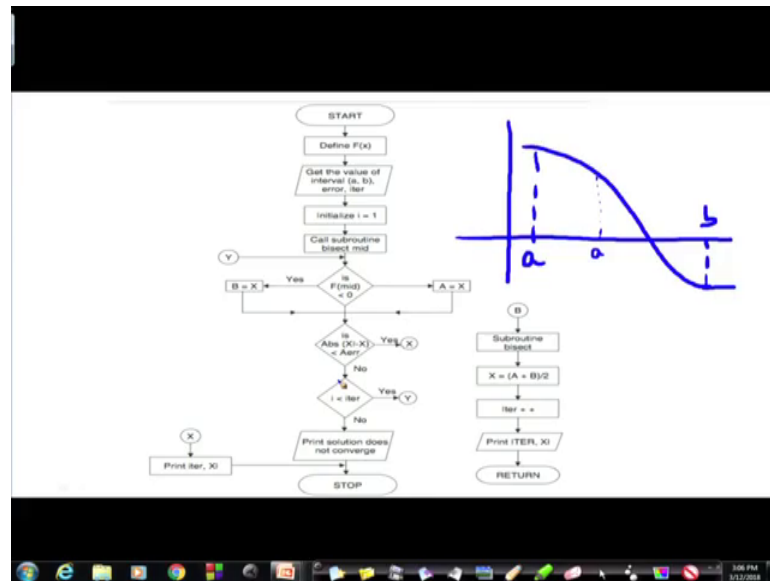


Then; obviously, the root does not lie between this. So, I will go on doing this bisection and again doing this bisection how long will they go on, but still I am not I will not find I mean it may be that I will not find the root therefore, there is a maximum limit that is kept number of iterations.

Now I initialize I to be 1 some I just some index to be 1 and then I call a subroutine or a function bisect the midpoint; that means. So, here is a sub subroutine bisect what it does here is a subroutine you see what it does or a function? What does it do here, it finds out between A and B the midpoint and increments the iteration and prints the value of X 1 X 1 is a middle point.

So, if my function was like this and this was a this was b, then the root mass lies somewhere here I find out the midpoint of this. So, this becomes x 1 to be there is a next 1 and how many what is the iteration? Next is F; that means $F_{mid} < 0$.

(Refer Slide Time: 09:46)



So, there can be 2 things all right here I select this to be my b and this to be my a . Now; obviously, if I take the midpoint then the midpoint will be somewhere here, a and b where of different signs. If F_{mid} is not less than 0, then a should be x ; that means, I will now move it on this side and try to find out this should be this should be the next a and between these 2 I will have to find out. And every time I am trying to find out whatever value of $F X$ that I compute is it less than the absolute error if yes then x that is I am getting my solution otherwise I am going on doing this.

So, this is a flow chart of the whole thing, but I think you will be more interested in looking at the algorithm and let us look at the algorithm for a second and the program here is algorithm you see this is much more understandable to you.

(Refer Slide Time: 11:07)

```
graph TD; 1[1.Start] --> 2[2.Read x1, x2, e]; 2 --> 3[3.Compute: f1 = f(x1) and f2 = f(x2)]; 3 --> 4{4.If (f1*f2) > 0, then display initial guesses are wrong and goto (11). Otherwise continue.}; 4 --> 5[5.x = (x1 + x2)/2]; 4 --> 11[11.Stop]; 5 --> 6{6.If ( | (x1 - x2)/x | < e ), then display x and goto (11).}; 6 --> 11; 6 --> 7[7.Else, f = f(x)]; 7 --> 8{8.If ((f*f1) > 0), then x1 = x and f1 = f.}; 8 --> 9{9.Else, x2 = x and f2 = f.}; 8 --> 10[10.Goto (5).]; 9 --> 10; 10 --> 5;
```

I start I read x_1 , x_2 and the error here x_1 and x_2 are the initial guesses all right. Here is my thing I have taken this is x_2 this is x_1 is absolute error; that means, how much error is permissible, compute f_1 that is $f(x_1)$ compute for this function this value and if 2 compute this value all right. If f_1 and f_2 the product of these two is greater than 0 greater than 0; that means, my initial guesses are wrong, because both of them are positive, then I can do many things I will instead of going to 11 my initial guesses are wrong I will again ask for new guess all right.

So, I take a new guess and I find that that is less than less than 0 then I take in this step x_1 plus x_2 mid of that. So, suppose mid of that is this 1 and that is becoming x , if x_1 minus x_2 by x actually here it should be if I think it is wrong here it should be if $f(x_1)$, minus $f(x_2)$ please read this as $f(x_1)$ now the error, $f(x_1)$ minus $f(x_2)$ divided by x is less than e then display x ; that means, if my error between these 2 between these 2 points the value difference is less is 0.0 0 0 2 and that is I just wore that I can assume as 0 then I will display this particular value of x .

So, right now it is not the case otherwise I will make this f to be $f(x)$ I take this all right. Now between these two I again divide I come here and in this way I go on alright. So, you will be able to write the program as the program runs. So, just to show you I am sure you can write the program yourself how can you translate this in the form of a code.

(Refer Slide Time: 14:16)

```
C Program for Bisection Method Source Code

#include<stdio.h>
#include<math.h>
float fun (float x)
{
    return (x*x*x - 4*x - 9);
}
void bisection (float *x,float a, float b, int *itr)
/* this function performs and prints the result
of one iteration */
{
    *x=(a+b)/2;
    ++(*itr);
    printf("iteration no. %3d X = %7.5f\n", *itr,
*x);
}

void main ()
{
    int itr = 0, maxitr;
    float x, a, b, allerr, x1;
    printf("\nEnter the values of a, b, allowed error and maximum
iterations:\n");
    scanf("%f %f %d", &a, &b, &allerr, &maxitr);
    bisection (&x, a, b, &itr);
    do
    {
        if ((fun(a)*fun(x) < 0)
        b=x;
        else
        a=x;
        bisection (&x1, a, b, &itr);
        if (fabs(x1-x) < allerr)
        {
            printf("After %d iterations, root = %6.4f\n", itr, x1);
            return 0;
        }
        x=x1;
    }
    while (itr < maxitr);
    printf("The solution does not converge or iterations not sufficient");
    return 1;
}
```

$x^3 - 4x - 9$

So, here you see let us try to understand this code it is a C program for the bisection method, I have included a `stdio.h` `math.h` and there is some function, because I have to find the root or of a particular polynomial.

So, for example, here it is given this is a polynomial. So, what is this polynomial it is x^3 minus $4x$ minus 9 . So, that value of fx has to be continued computed. So, the function is the `fun` is the name of the function, then there is another function `bisection` this function performs and prints the result of one iteration.

So, it is a plus b by 2 ; now in an earlier lecture we had talked about this what is this because here when I am calling this function, I am calling by reference how? I am calling I am just passing the address x and whatever I do here once again is a part of revision you can see if I come here `float *x`; that means, what x is there some variable and I have just passed the address of that.

(Refer Slide Time: 15:44)

```
C Program for Bisection Method Source Code

#include<stdio.h>
#include<math.h>
float fun (float x)
{
    return (x*x*x - 4*x - 9);
}
void bisection (float *x, float a, float b, int *itr)
/* this function performs and prints the result
of one iteration */
{
    *x=(a+b)/2;
    ++(*itr);
    printf("iteration no. %3d X = %7.5f\n", *itr,
*x);
}

void main ()
{
    int itr = 0, maxitr;
    float x, a, b, allerr, x1;
    printf("\nEnter the values of a, b, allowed error and maximum
iterations:\n");
    scanf("%f %f %f %d", &a, &b, &allerr, &maxitr);
    bisection (&x, a, b, &itr);
    do
    {
        if ((fun(a)*fun(x) < 0)
        b=x;
        else
        a=x;
        bisection (&x1, a, b, &itr);
        if (fabs(x1-x) < allerr)
        {
            printf("After %d iterations, root = %6.4f\n", itr, x1);
            return 0;
        }
        x=x1;
    }
    while (itr < maxitr);
    printf("The solution does not converge or iterations not sufficient");
    return 1;
}
```

So, star x is the content of this say 50 all right. So, I passed it on and in the main function I am passing on the address of x ok. This we have seen earlier. So, I am taking the midpoint and incrementing the iteration; iteration is also a call by reference. And float a float b are two points in between which have been passed on.

Now, what is being done in the main function? In the main function I am setting the iteration to be 0 and here I am saying how many iterations are permitted?

(Refer Slide Time: 16:34)

```
C Program for Bisection Method Source Code

#include<stdio.h>
#include<math.h>
float fun (float x)
{
    return (x*x*x - 4*x - 9);
}
void bisection (float *x, float a, float b, int *itr)
/* this function performs and prints the result
of one iteration */
{
    *x=(a+b)/2;
    ++(*itr);
    printf("iteration no. %3d X = %7.5f\n", *itr,
*x);
}

void main ()
{
    int itr = 0, maxitr;
    float x, a, b, allerr, x1;
    printf("\nEnter the values of a, b, allowed error and maximum
iterations:\n");
    scanf("%f %f %f %d", &a, &b, &allerr, &maxitr);
    bisection (&x, a, b, &itr);
    do
    {
        if ((fun(a)*fun(x) < 0)
        b=x;
        else
        a=x;
        bisection (&x1, a, b, &itr);
        if (fabs(x1-x) < allerr)
        {
            printf("After %d iterations, root = %6.4f\n", itr, x1);
            return 0;
        }
        x=x1;
    }
    while (itr < maxitr);
    printf("The solution does not converge or iterations not sufficient");
    return 1;
}
```

Maximum number of iterations x , a , b allowed error how much error is allowed and sum x 1 value is given into the values of a b allowed error and maximum iterations. So, all these I read the range a and b between, which points I have to do a and b and how much is the allowed error and what is the maximum number of iteration? Then with this I call bisection what do I do bisection and x ; that means, this will be give me the midpoint I will call bisection here a b iteration.

So, a is being passed here, b is being passed here, and the number of iterations is being passed here. Now here I am finding the midpoint and that midpoint is being returned here is common right, and then here at this point I find out I call the function that is I am computing the polynomial.

(Refer Slide Time: 17:54)

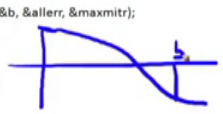
```

C Program for Bisection Method Source Code

#include<stdio.h>
#include<math.h>
float fun (float x)
{
    return (x*x*x - 4*x - 9);
}
void bisection (float *x, float a, float b, int *itr)
/* this function performs and prints the result
of one iteration */
{
    *x=(a+b)/2;
    ++(*itr);
    printf("Iteration no. %3d X = %7.5f\n", *itr,
*x);
}

void main ()
{
    int itr = 0, maxitr;
    float x, a, b, allerr, x1;
    printf("\nEnter the values of a, b, allowed error and maximum
iterations:\n");
    scanf("%f %f %d", &a, &b, &allerr, &maxitr);
    bisection (&x, a, b, &itr);
    do
    {
        if ((fun(a)*fun(x) < 0))
            b=x;
        else
            a=x;
        bisection (&x1, a, b, &itr);
        if ((fabs(x1-x) < allerr)
        {
            printf("After %d iterations, root = %6.4f\n", itr, x1);
            return 0;
        }
        x=x1;
    }
    while (itr < maxitr);
    printf("The solution does not converge or iterations not sufficient");
    return 1;
}

```



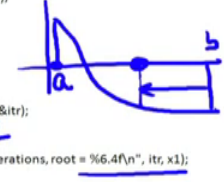
If the polynomial at a value of the polynomial at a , and the volume value of the polynomial at x there is a midpoint is less than 0. So, what happened this was my scenario here was a and here was b , now let me draw it.

(Refer Slide Time: 18:30)

```
C Program for Bisection Method Source Code

#include<stdio.h>
#include<math.h>
float fun (float x)
{
    return (x*x*x - 4*x - 9);
}
void bisection (float *x, float a, float b, int *itr)
/* this function performs and prints the result
of one iteration */
{
    *x=(a+b)/2;
    ++(*itr);
    printf("iteration no. %3d X = %7.5f\n", *itr,
*x);
}

void main ()
{
    int itr = 0, maxitr;
    float x, a, b, allerr, x1;
    printf("\nEnter the values of a, b, allowed error and maximum
iterations:\n");
    scanf("%f %f %d", &a, &b, &allerr, &maxitr);
    bisection (&x, a, b, &itr);
    do
    {
        if (fun(a)*fun(x) < 0)
            b=x;
        else
            a=x;
        bisection (&x1, a, b, &itr);
        if (fabs(x1-x) < allerr)
        {
            printf("After %d iterations, root = %6.4f\n", itr, x1);
            return 0;
        }
        x=x1;
    }
    while (itr < maxitr);
    printf("The solution does not converge or iterations not sufficient");
    return 1;
}
```

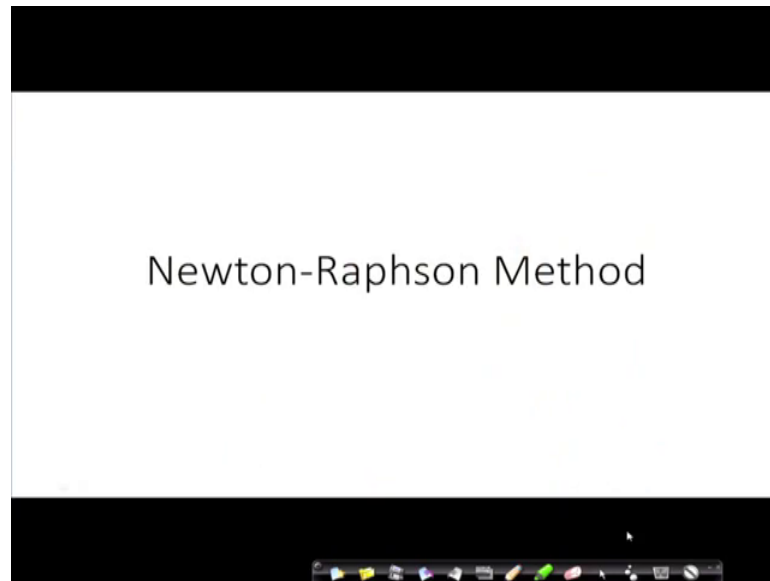


So, here was b and here was a. So, now, what I do I got the meet point somewhere here, then the value of the function at this point and the value of the function at this point a negative. Therefore, I move this b to x all right x is becoming b and I do the same thing.

Otherwise if it was on the other side I would have made x to be a alright this clear and then again I call bisection. After calling bisection I find if the absolute error of x_1 minus x is less than the allowed error, then I will print the root.

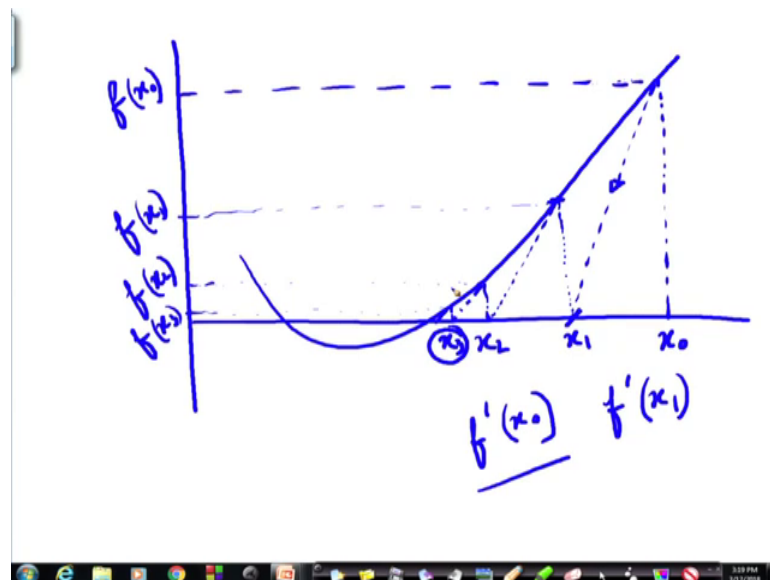
So, this will go on while iter this is under this do while this will go on until I exceed the maximum iteration. So, this is how I code and all of you should be able to practice this yourself ok. Next we will move to another algorithm another algorithm, which is that another method, which is known as the Newton-Raphson method.

(Refer Slide Time: 20:18)



This method adopts a different approach to find the root of the function let us try to understand this briefly.

(Refer Slide Time: 20:40)



So, I have got a function like this; now Newton-Raphson method, what it does at it starts at some x_0 and the corresponding value of the function at x_0 is $f(x_0)$, now what it does it finds out the tangent at this point tangent to this point. So, what would that tangent be that tangent is nothing, but $f'(x_0)$ ok, because we know that is f' prime means $\frac{d f}{d x}$.

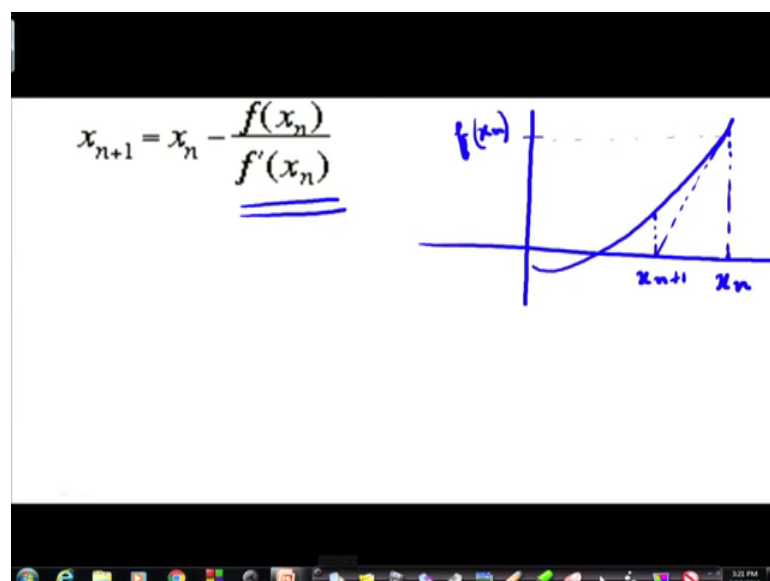
So, I draw the tangent here the tangent intersects the x axis at some point. Now I take the x_1 this let this be the value x_1 I drew the tangent and got the value x_1 ok. I come here and from here, since the value here $f(x_1)$ at every stage have to check whether the value is close to 0 or not; obviously, this is not the case.

Therefore, I draw another tangent from here and what is this tangent this tangent is $f'(x_1)$ dashed x_1 derivative at this point. So, this becomes x_2 , now I find out $f(x_2)$ now I again compare whether $f(x_2)$ to is very close to 0 or not still it is not the case. So, I draw a tangent from here to this I am sorry I should have made it dotted as I was doing at all I draw a tangent at this point.

So, this is x_3 and I find out if x_3 . Suppose this value $f(x_3)$ is very close to 0 suppose this is within my allowed error, then x_3 is the root otherwise if it was not there from here again I would have to draw a tangent in this way it goes all right.

So, this is the essence of Newton-Raphson's method. So, what are you doing here we are taking a function starting with a point and finding a tangent to that curve to the function at that point and see where that far that tangent intersects the x axis from there I find out $f(x_2)$ and then I go on doing this. So, I think this geometric geometrical exposition will be very helpful to you. So, next let us try to see how Newton-Raphson method works.

(Refer Slide Time: 24:46)



So, at every stage x_{n+1} is $x_n - f(x_n) / f'(x_n)$ why because of the simple reason that I had this curve I had this curve and this was my x_n all right from there I drew the tangent. So, this was $f(x_n)$ and I drew the tangent here, then if I divide this and my subtracted from here I will get this x_{n+1} here.

So, you see it is coming in the other way alright because at every stage I am computing this next here x_n is the current known value of x $f(x_n)$ represents the value of the function, $f'(x_n)$ is the derivative of the slope at that point x_{n+1} represents the next x value that you are trying to find.

(Refer Slide Time: 25:53)

Here, x_n is the current known x -value, $f(x_n)$ represents the value of the function at x_n , and $f'(x_n)$ is the derivative (slope) at x_n . x_{n+1} represents the next x -value that you are trying to find. Essentially, $f'(x)$, the derivative represents $f(x)/dx$ ($dx = \Delta x$). Therefore, the term $f(x)/f'(x)$ represents a value of dx .

$f(x) = x^2 - 4$
 $f'(x) = 2x$
 $x_0 = 6$

$\frac{f(x)}{f'(x)} = \frac{f(x)}{f(x)/\Delta x} = \Delta x$

n	x_n	$f(x_n)$	$f'(x_n)$	x_{n+1}	dx
0	$x_0 = 6$	$f(x_0) = 32$	$f'(x_0) = 12$	$x_1 = 3.33$	
1	$x_1 = 3.33$	$f(x_1) = 7.09$	$f'(x_1) = 6.66$	$x_2 = 2.27$	$dx = 1.06$
2	$x_2 = 2.27$	$f(x_2) = 1.15$	$f'(x_2) = 4.54$	$x_3 = 2.01$	$dx = .26$
3	$x_3 = 2.01$	$f(x_3) = 0.04$	$f'(x_3) = 4.02$	$x_4 = 2.00$	$dx = 0.01$

So, this expression is coming from the fact that $f(x) / f'(x)$, where dx is Δx , therefore, the term $f(x) / f'(x)$ takes is actually the value of dx how much the dx value sorry the x value; that means, how much I should count down.

So, you can see from this expression $f(x) / f'(x)$ is $f(x) / (f(x) / \Delta x)$; that means, the Δx actually this is the Δx part and; that means, how I am shifting this x . So, the x was here and I am shifting it by Δx and coming here again shifting it by Δx coming here like that I am going all right.

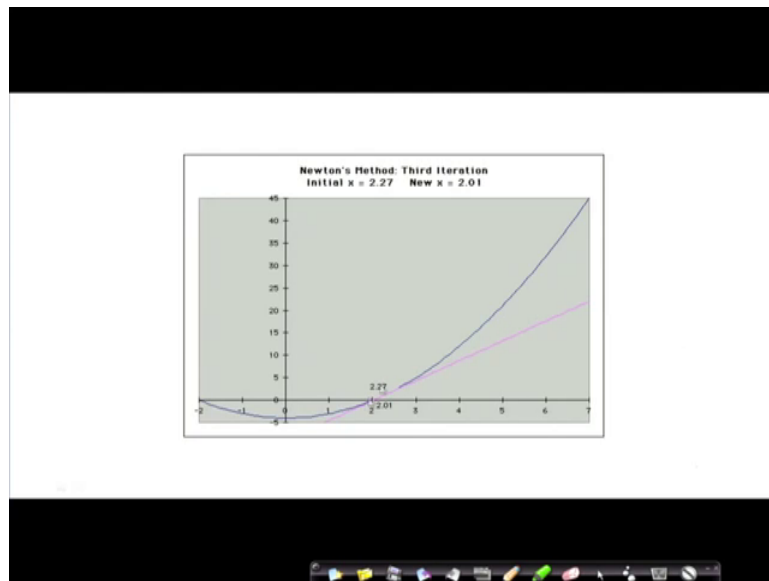
So, suppose $f(x)$ let us take an example here suppose $f(x)$ was $x^2 - 4$, then $f'(x)$ is; obviously, $2x$ and x_0 was 6 suppose I assumed x_0 to be 6 . So, here is how it goes first iteration x_0 is 6 $f(x_0)$ here $f(x_0)$ is supposed 32 , because $6^2 - 4$. If

prime x is what if prime x is $2x$ that is 12 if prime x is 12, then x plus $1 \times n$ sorry I am sorry this is $x \times n$ and the next value will be next value will be $x \times n$; that means, 6 minus 32 by 12. So, whatever that is I subtract and I get they 3.3 3 next iteration I come to 3.3 3.

So, it was something like this that I started with 6 and then I moved to 3.3 3. So, you see it is converging very fast 3.3 3, then at 3.3 3 the value of $f(x)$ is 7.0 9 here if you compute this the derivative will be twice of this that is 6.6 6 derivative is $2 \times 6.6 6$ if I subtract this 6.6 6 divided by 7.0 9 subtract it from 3.3 3 is 2.2 7. So, my dx is 1.0 6 I go on like this and ultimately I come to a dx of 0.0 I assume.

So, next time it is 2.2 7 I start with that again find the next value to be 2 2 point instead of 2.2 7 it will be 2.0 1 and with 2.0 1 I compute and gradually you see the dx is coming down as the dx is coming down; that means, I am approaching the actual root. So, this is Newton-Raphson method and we can very easily code it.

(Refer Slide Time: 29:48)

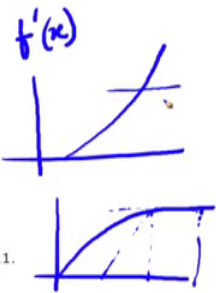


So, here is an example now you can see this it starts with 6 goes to 3.3 3 then from the 3 here is a little animation 3.3 3; I am coming to 2.2.7 then from 2.2.7, I am coming here and gradually the error is that it is not increasing. So, it is very much converging. So, I get the solution with 2.0 1 and I get the solution.

(Refer Slide Time: 30:18)

Newton Raphson Method Algorithm:

1. Start
2. Read x , e , n , d
* x is the initial guess
 e is the absolute error i.e the desired degree of accuracy
 n is for operating loop
 d is for checking slope*
3. Do for $i = 1$ to n in step of 2
4. $f = f(x)$
5. $f1 = f'(x)$
6. If ($|f1| < d$), then display too small slope and goto 11.
7. $x1 = x - f/f1$
8. If ($|(x1 - x)/x1| < e$), the display the root as $x1$ and goto 11.
9. $x = x1$ and end loop
10. Display method does not converge due to oscillation.
11. Stop



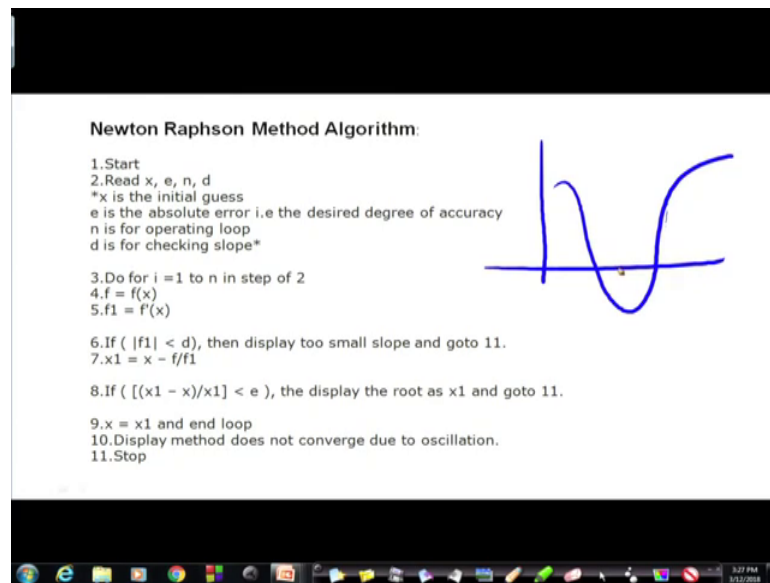
The image shows a slide with the Newton-Raphson Method Algorithm. The algorithm is listed in 11 steps. To the right of the text are two hand-drawn graphs. The top graph shows a curve labeled f(x) and a tangent line at a point, with the derivative f'(x) labeled. The bottom graph shows a curve and a tangent line, with a dashed line indicating the slope of the tangent line.

So, quickly the algorithm will look like this again I will read x the a maximum error allowed number of iterations and d d is the checking for checking the slope, here are the comments x is the initial guess absolutely that is e n is the number of iterations.

So, do in a loop i to n in steps of 2 $f x$ f is equal to $f x$ and then $f 1$ is f prime x . Now these are 2 functions, which you have to write and if the now why is why am I keeping this check? Why am I keeping this check? $f 1$ which is a slope if f dashed x is too small; that means, what that the slope is nearly horizontal; that means, I am not going to get any suppose something is something like this. If I come to this point and try to find a slope of this the slope will be very horizontal.

So, this is not a good choice in that case I have to reduce it and come to a point, where I can find a slope all right. Now, in that way I go on and find the whether the it is coming to the close to the root and go on. Now, if it goes on the iteration goes on say for example, it is possible that I am missing the loop coming close to that say a curve like this.

(Refer Slide Time: 32:05)



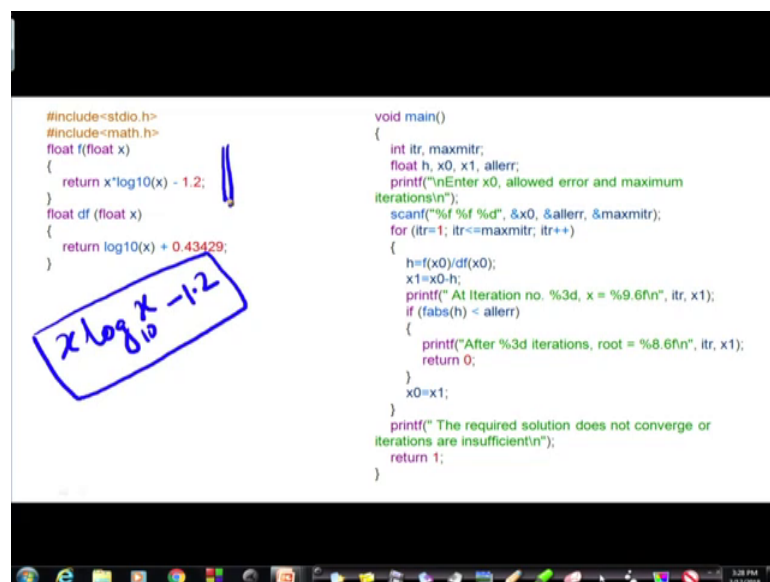
Newton Raphson Method Algorithm:

- 1.Start
- 2.Read x , e , n , d
* x is the initial guess
 e is the absolute error i.e the desired degree of accuracy
 n is for operating loop
 d is for checking slope*
- 3.Do for $i = 1$ to n in step of 2
4. $f = f(x)$
5. $f1 = f'(x)$
- 6.If ($|f1| < d$), then display too small slope and goto 11.
7. $x1 = x - f/f1$
- 8.If ($|(x1 - x)/x1| < e$), the display the root as $x1$ and goto 11.
9. $x = x1$ and end loop
- 10.Display method does not converge due to oscillation.
- 11.Stop

The slide also features a hand-drawn blue graph of a function with a root marked on the x-axis.

The slope of the curve was such that I was trying to come here and somehow I miss the root I go to another point. So, that is another special case I need not bother you with that right now. So, let us have a quick look at the program the program will be again.

(Refer Slide Time: 32:28)



```
#include<stdio.h>
#include<math.h>
float f(float x)
{
    return x*log10(x) - 1.2;
}
float df (float x)
{
    return log10(x) + 0.43429;
}

void main()
{
    int itr, maxitr;
    float h, x0, x1, allerr;
    printf("\nEnter x0, allowed error and maximum iterations\n");
    scanf("%f %f %d", &x0, &allerr, &maxitr);
    for (itr=1; itr<=maxitr; itr++)
    {
        h=f(x0)/df(x0);
        x1=x0-h;
        printf(" At iteration no. %3d, x = %9.6f\n", itr, x1);
        if (fabs(h) < allerr)
        {
            printf("After %3d iterations, root = %8.6f\n", itr, x1);
            return 0;
        }
        x0=x1;
    }
    printf(" The required solution does not converge or iterations are insufficient\n");
    return 1;
}
```

Handwritten notes in blue ink include a box around the function $x \log_{10} x - 1.2$ and a vertical line next to the derivative function definition.

So, here we are trying to find out the root of a function $x \log x$, the function is $x \log x$ to the base 10 minus 1.2. So, that is the function that function is embodied in another C function. Now $d f$ is returning $d f$ is nothing, but f dashed x .

(Refer Slide Time: 33:12)

```
#include<stdio.h>
#include<math.h>
float f(float x)
{
    return x*log10(x) - 1.2;
}
float df(float x)
{
    return log10(x) + 0.43429;
}

void main()
{
    int itr, maxitr;
    float h, x0, x1, allerr;
    printf("\nEnter x0, allowed error and maximum
    iterations\n");
    scanf("%f %f %d", &x0, &allerr, &maxitr);
    for (itr=1; itr<=maxitr; itr++)
    {
        h=f(x0)/df(x0);
        x1=x0-h;
        printf(" At Iteration no. %3d, x = %9.6fn", itr, x1);
        if (fabs(h) < allerr)
        {
            printf("After %3d iterations, root = %8.6fn", itr, x1);
            return 0;
        }
        x0=x1;
    }
    printf(" The required solution does not converge or
    iterations are insufficient\n");
    return 1;
}
```

f(x)
f'(x)

So, if this function is given I also keep f dashed x written. So, this is $f(x)$ this is $f'(x)$. Now I know I have already pre coded them and that will return me the value for different values of x .

So, now again I read as scan, f i read the initial x_0 the allowed error the maximum iteration. Now, then in this loop what I do I find $f(x)$ by $f'(x)$ and that is h how much I should reduce, how much I should change the initial value? The initial value was x_0 with which I started I subtract that and come to the next point.

And if the absolute error is less than absolute value at that point if absolute at that point is less than error then that is a solution. Otherwise, I will go up and repeat this. Now if I go on and ultimately if I overshoot the maximum iteration, then I can say that the required solution does not converge or the iterations are inefficient.

So, Newton-Raphson usually gives us a very fast way of finding the root, but sometimes it does not converge and that is one problem of that; however, there are many other sophisticated ways of finding roots just to summarize I would like to say that what we have learnt in the past couple of lectures is that one of the major technological requirements a computational requirements are finding roots of polynomials for many solutions for many engineering solutions, I have to solve equations.

For that there are many methods we have just gone through to simpler methods one is the bisection method and the other one that we saw just now is the Newton-Raphson method. Next we look at something else called interpolation and other things.

Thank you.