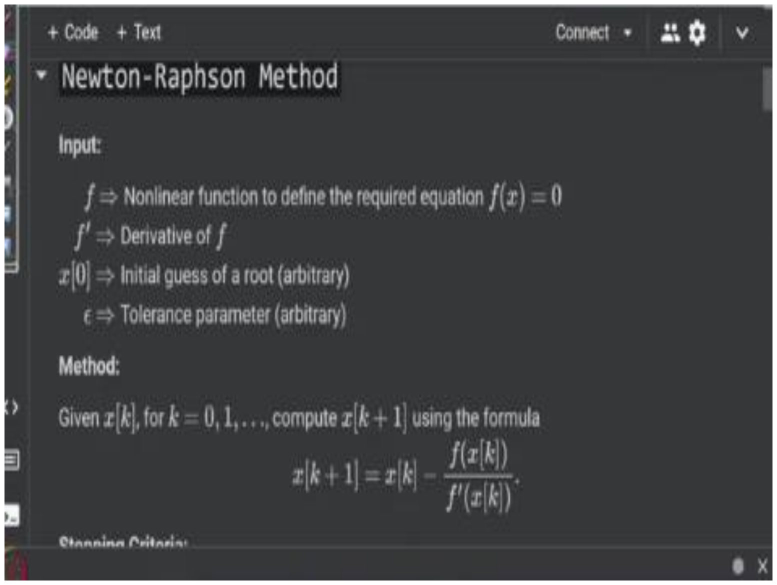
Numerical Analysis
Prof. S. Baskar
Department of Mathematics
Indian Institute of Technology, Bombay

Lecture - 37
Nonlinear Equations: Implementation of Newton-Raphson's Method as Python Code

We have completed iterative methods for nonlinear equations. In this lecture we will try to implement Newton-Raphson method. Let us start our discussion with developing a python code for Newton-Raphson method.

(Refer Slide Time: 00:35)



Newton-Raphson Method

Input:

$f \Rightarrow$ Nonlinear function to define the required equation $f(x) = 0$
$f' \Rightarrow$ Derivative of f
$x[0] \Rightarrow$ Initial guess of a root (arbitrary)
$\epsilon \Rightarrow$ Tolerance parameter (arbitrary)

Method:

Given $x[k]$, for $k = 0, 1, \ldots$, compute $x[k+1]$ using the formula

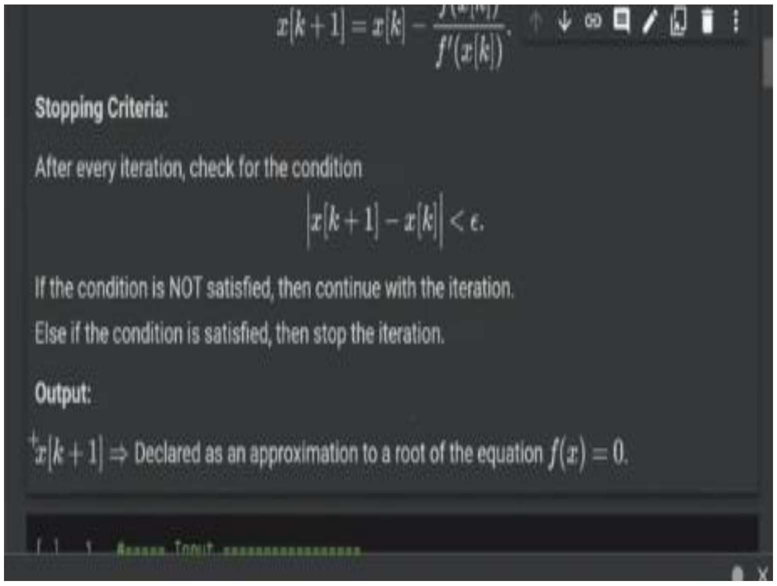
$$x[k+1] = x[k] - \frac{f(x[k])}{f'(x[k])}.$$

Stopping Criteria:

Let us recall the Newton-Raphson method along with what are all the inputs we need and also what we want to take as an output from the code. We need the following inputs. First of all, we need the function *f* which defines the required non-linear equation $f(x) = 0$. And if you recall the formula for Newton-Raphson method also includes the derivative of the function *f*. Therefore, we have to take f' also as the input.

These two are taken as exact expressions and then we need an initial guess for the root of our equation that is taken as x_0. And then once we start generating the iterative sequence, we have to stop the iteration at some point. For that we will take the tolerance parameter ϵ and we will try to check the Cauchy criteria to stop the iteration. Let us quickly recall the Newton-Raphson method.

(Refer Slide Time: 01:53)



$$x[k+1] = x[k] - \frac{f(x[k])}{f'(x[k])}.$$

Stopping Criteria:

After every iteration, check for the condition

$$\left|x[k+1] - x[k]\right| < \epsilon.$$

If the condition is NOT satisfied, then continue with the iteration.

Else if the condition is satisfied, then stop the iteration.

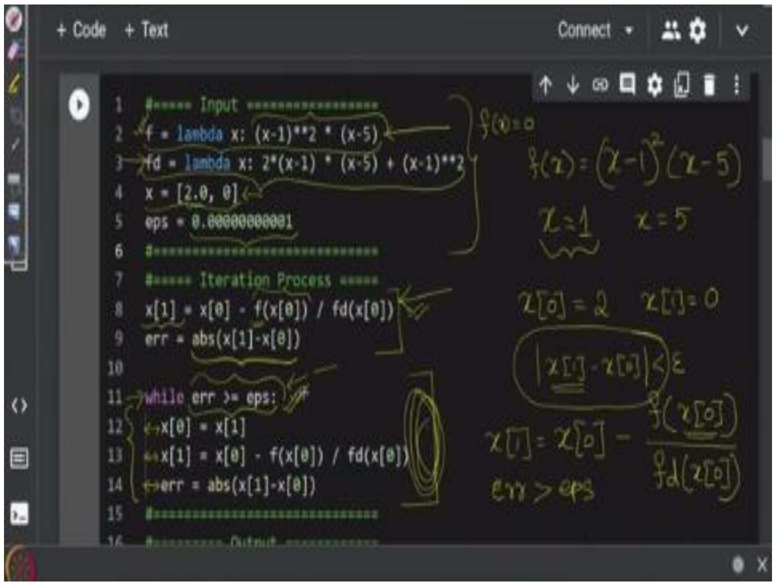
Output:

$x[k+1] \Rightarrow$ Declared as an approximation to a root of the equation $f(x) = 0$.

The Newton-Raphson method formula is given like this, remember we are given $x[0]$ and from there we will get $x[1]$ using this formula. And once you get $x[1]$, using $x[1]$ we will get $x[2]$ and so on. So, this is how the iteration should go. Remember x[0] is taken as the input that you will plug in here in the first iteration and then already, we have taken the expression for f and f'. Therefore $x[1]$ can be calculated using this expression and then using $x[1]$ you will get $x[2]$ and so on.

So, in that way we have to formulate a loop which will run for each iteration. Now where to break the loop and how to break the loop? For that we need the stopping criteria. For the stopping criteria we will use the Cauchy criteria and finally what is the output? The output should be the value of the iteration $x[k+1]$. I hope you understood the idea of how we have to go ahead in coding the Newton-Raphson method. Let us now see the code.

(Refer Slide Time: 03:36)



```
#===== Input =================
f = lambda x: (x-1)**2 * (x-5)
fd = lambda x: 2*(x-1) * (x-5) + (x-1)**2
x = [2.0, 0]
eps = 0.00000000001
#=============================
#===== Iteration Process =====
x[1] = x[0] - f(x[0]) / fd(x[0])
err = abs(x[1]-x[0])

while err >= eps:
    x[0] = x[1]
    x[1] = x[0] - f(x[0]) / fd(x[0])
    err = abs(x[1]-x[0])
#=============================
#========== Output ===========
```

Let us start with the input. We have to take the function f which defines the equation $f(x) = 0$. And that is defined here, you can see that I am taking $f(x) = (x-1)^2(x-5)$. From this itself we can clearly see what are all the roots of the equation $f(x) = 0$. The roots are $x = 1$ and $x = 5$. The next thing is, we have to also input the expression for the derivative of *f*, that I am taking in the variable fd. Remember these two are going to be considered as exact expressions.

By supplying the value of x, you can evaluate f(x) and fd(x). Then the next input is the array x has two components x[0]. Remember in python the index starts from 0, x[0] and that is assigned as two in this case and then x[1] is initialized as 0 here that is what this line means and then we have the epsilon which is the tolerance parameter. I have taken it to be a very small number so that I want a very good accuracy for my root.

So, these are all the inputs as we have seen previously. Now let us go into the main part of the code that is the iteration process. This is the main loop of my program but before getting into the program I am just doing one, iteration outside. Why, because I have to check x[1] – x[0]. So, for that I need the value x[1] that is why, once I am doing the iteration outside the loop and then I will start the loop by checking the stopping criteria.

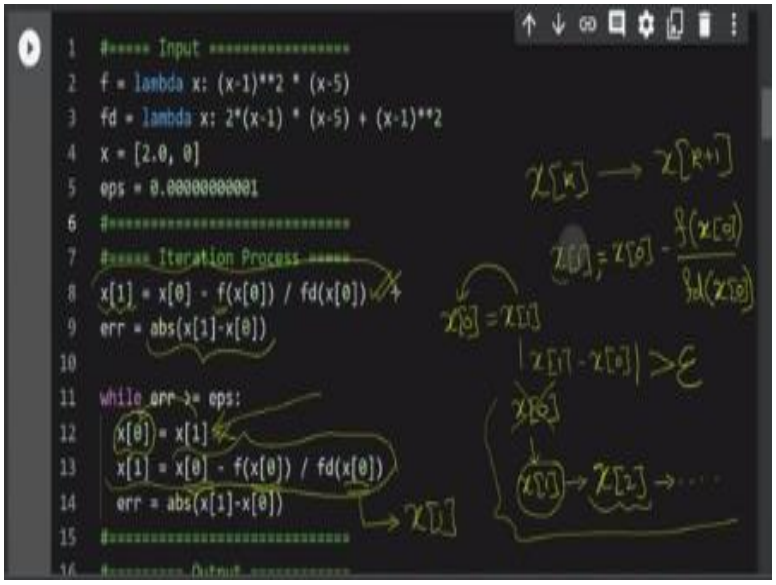
So, this is my stopping criteria. So, when I enter into the loop, I have to first have the value of this error. That is why, x[1] alone is computed outside the loop and now you can see how I am computing x[1] . Remember x[1] is nothing but x[0] – f(x[0]). Remember f is given by this expression and by sending the value of x[0], I can find the value of this expression by putting f(x[0]). That is what it means and then divided by f'(x[0]). But how do we have defined the expression for f'?

We have defined it by fd. Therefore, I will have fd(x[0]). So, that is precisely what I am writing in this expression. Once I compute the value of x[1] then I will compute the absolute value of the difference between x[1] and x[0]. This is precisely the absolute error which I want here because I want to check these stopping criteria whether it is less than epsilon or not. That is precisely checked in this expression.

So, I am initiating a while loop here and the while loop runs whenever this error is greater than epsilon, our loop will keep on running. When it becomes less than epsilon then the loop will break and we will come out of the loop. Now let us see what are we computing inside this loop as you can see there are 3 lines in the loop. Why are there only 3 lines? Because we have given this indent here, so whatever is given with indent, are considered to be the part of the loop.

So, we have given only these three lines with an indent. Therefore, these three lines are considered to be the part of the while loop and this will keep on going again and again till this condition is satisfied. Once this condition is violated, then it comes out of the loop. Let us try to understand what is happening inside this loop?

(Refer Slide Time: 09:29)



```
#===== Input =================
f = lambda x: (x-1)**2 * (x-5)
fd = lambda x: 2*(x-1) * (x-5) + (x-1)**2
x = [2.0, 0]
eps = 0.00000000001
#=============================
#===== Iteration Process =====
x[1] = x[0] - f(x[0]) / fd(x[0])
err = abs(x[1]-x[0])

while err >= eps:
  x[0] = x[1]
  x[1] = x[0] - f(x[0]) / fd(x[0])
  err = abs(x[1]-x[0])
#=============================
#========== Output ===========
```

Ideally, we should have many components of the array x because we have to take x[0] as an input then find x[1] and once you find x[1] you find x[2] and so on. So, we need many components of the array x but what we observe is, once you get x[1] to compute x[2], we only need x[1]. We do not need x[0] in order to compute x[2]. Therefore, why to waste the memory for so many components of x when we need only x[k] to compute x[k + 1].

That is why what I am doing here is, once I computed x[1] and also, I found this absolute error I do not need x[0] anymore. Therefore, I put the value of x[1] which is computed here into the variable x[0]. In that way I have made this memory for x[1] free now. The next iteration which is given like this is now computed and stored in x[1]. Therefore, I just compute x[2], which is actually x[1] – f(x[1]) divided by fd(x[1]).

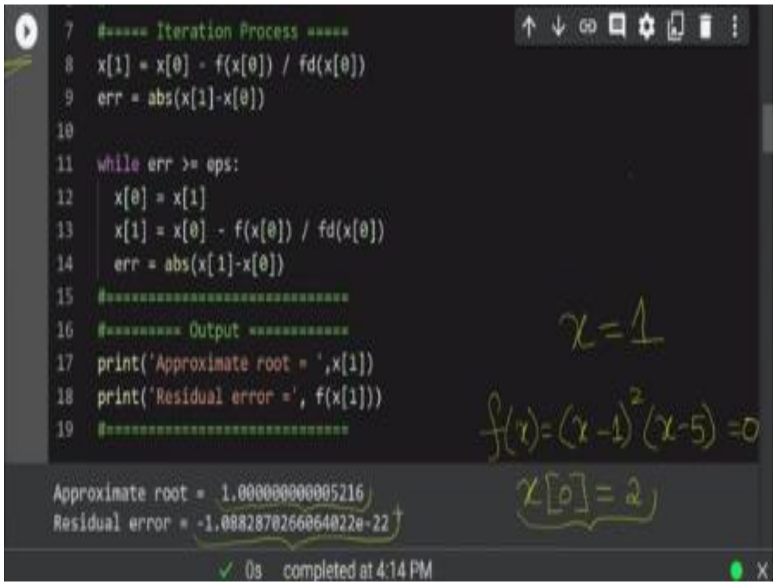
But what I did is, I took x[1] and put that value into x[0]. Therefore, I will simply write x[0] instead of x[1]. I hope you understood the logic here because I stored the value of x[1] in x[0], I am using this variable now x[0]. Remember this is holding the value of x[1] now because of this line, I do not need to create one more component for the array. So, what I am doing is I am simply using x[1] here. So, that is why I am using this expression.

So, this will overwrite the value that is already computed here. It does not matter that we do not need that value anymore that is why I am doing it. Once you have x[1], again you go to find the absolute error in x[1] when compared to x[2]. And now once you have the new value of the error again you come to check whether the stopping criteria is satisfied or not. If the stopping criteria is not satisfied, that is if the absolute error is still greater than or equal to epsilon.

Again, you will go now remember x[2] is actually stored in x[1] only. It is stored in x[1] that again you will rewrite with x[0]. And thereby x[1] is again made free and you will now store the x[3] value in x[1] by again doing this computation and then you will go to check the error, like that it will keep on going and then do this computation again and check this condition like this process will go. And this is the iterative process for the Newton-Raphson method.

And now we are into the output of the code. Let us see what we have to take as the output as far as the output is concerned.

(Refer Slide Time: 13:57)



```
#===== Iteration Process =====
x[1] = x[0] - f(x[0]) / fd(x[0])
err = abs(x[1]-x[0])

while err >= eps:
  x[0] = x[1]
  x[1] = x[0] - f(x[0]) / fd(x[0])
  err = abs(x[1]-x[0])
#============================
#========= Output ===========
print('Approximate root = ',x[1])
print('Residual error =', f(x[1]))
#============================
```

```
Approximate root =  1.000000000005216
Residual error = -1.0882870266064022e-22
```

We will take the approximate root of the equation as x[1] when the loop breaks, you have the recently calculated value stored in the variable x[1]. And that will be finally printed on the screen as the approximate root of our equation. I am also giving the residual error just for our information. Let us run this code to see what the output is. Remember we have the equation $f(x) = (x-1)^2(x-5)$ and that is equal to 0.

That is our equation and we started our initial guess as x[0] = 2. I am sorry this is actually x - 1. Let us see what is the output of this code. You can see that the approximate root is printed finally and that is given by approximately equal to 1. Therefore, when we start the iteration with the initial guess as 2 then the iteration seems to have converged to the root x = 1. That is what we are seeing. You can also see that the residual error is pretty small.

And this is what we would like to do in the Newton Raphson implementation. Thank you for your attention.