MATLAB Programming for Numerical Computation

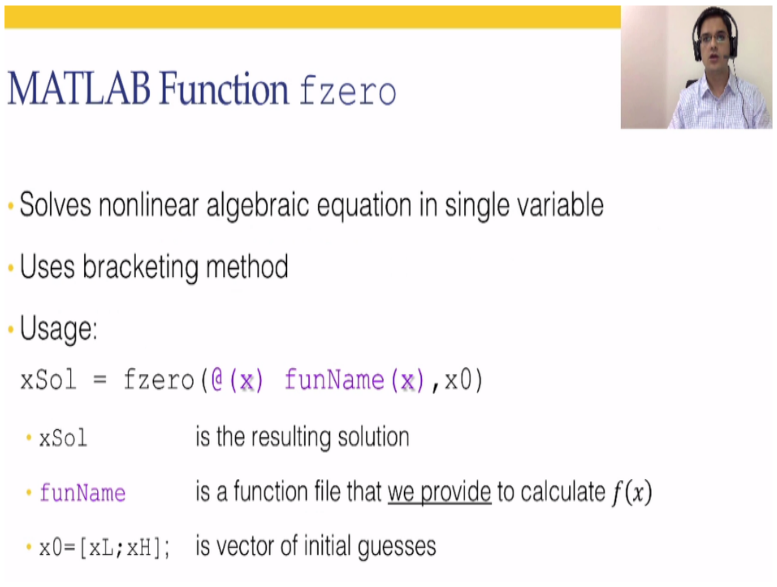
Dr.Niket Kaisare
Department of Chemical Engineering
Indian Institute of Technology, Madras

Module No. #05
Lecture No. #5.2
Non Linear Algebraic Equations – Using MATLAB function fzero

Hello and welcome to MATLAB programming for numerical computations. We are in week number 5 lecture 5.2. In this week we are covering solving nonlinear algebraic equations and in today's lecture we are going to focus on using MATLAB function fzero. The MATLAB function fzero is a method to solve a linear, a single nonlinear equation in a single unknown.

In the previous lecture we have seen the bisection method which was a method to compute the solution to f(x) = 0, given 2 initial guesses. The 2 initial guesses lie on either side of the solution. fzero is a MATLAB program that uses a techniques. Similar to bisection rule it uses bisection and a couple of other techniques depending on the problem statement.

So we are going to do 2 things in today's lecture, first see how to use the function fzero, second we are going to modify the bisection rule that we made the program in the previous lecture in order to do and understand bisection rule and understand numerical equation solving in a better way okay.
(Refer Slide Time: 01:22)



MATLAB Function fzero

- Solves nonlinear algebraic equation in single variable
- Uses bracketing method
- Usage:

```
xSol = fzero(@(x) funName(x),x0)
```

xSol	is the resulting solution
funName	is a function file that <u>we provide</u> to calculate $f(x)$
x0=[xL;xH];	is vector of initial guesses

So let us look at the MATLAB function fzero. We are going to use fzero in order to solve the same equation. That we did in the previous lecture that equation was 2-x+ln(x) okay. Now if we are going to do help fzero, we will find out how to use fzero. Some of the most important things I have listed over here.

The usage of fzero is going to be as follows. The solution xSol that is the output from fzero = fzero followed by the name of the function in the x variable followed by the initial guesses xSol is the resulting solution, funName is the name of the function that returns the value of f(x) given the variable x okay.

There are different ways in MATLAB in order to pass on the name of the function to a calling function. We are going to use basically this @ func @ representation using what is known as anonymized functions representation okay.

So this is the method that we are going to use. as seen over here the method is as shown is @ followed by in brackets the name of the variables that you need to solve for, followed by a space, followed by the name of the function in which you will use to get the value f(x) = 0 that you want to solve so funName is going to return f(x) given the function given the values of variable x.

Now funName itself can have other parameter as well in which case we will have the other parameter listed over here. This x and this x should be the same okay. That is the variable that fzero is going to solve for. When we use this MATLAB function fzero okay. There are various, as I said there are other ways of using functions as well and the most common way is to use @ funName.

We are not going to use that particular method for because that is somewhat of an older method and this method of using functions. Although a little less simpler in the beginning is actually a very powerful method okay. So let us go on in MATLAB and use function fzero in order to solve f(x) equal to 0.
(Video Starts: 03:53) So edit, let us say fun4Bisec okay. So we are going to use the function and the name of the function I am just going to call it fun4Bisec okay. So function fval = fun4Bisec(x). That is the name of our function okay. Keep in mind the name of the function that we have right over here should be the same this name has to be the same as the name of the file.

The file name is what the, what MATLAB recognizes the function as, so as far as we are concerned the name of the function that we write over here and the name of the file always needs to be the same okay. And the function that we have our f(x) is nothing but fval = 2-x+log (x) and end. End is optional so as a good programming practice we are going to include the end also over here okay.

So now let us say what the fun4Bisec is going to be for x=1, so f(x) = 1, so we will say x = 1 and f is just 2-x+log (x) okay. And for x = 1, 2-x+log (x) that value is also equal to 1 let us call that function fun4Bisec using x and that should return the value of 1 okay. Let us say we were to recall it using 2. That should return f (2). If we call it using 3, it will return f(3). If we call it using 4, it will return f of 4 okay.

If you recall from the previous lecture, we had given our initial guesses our xl we had given as = 1. Our xu we had given equal to 4. The reason why we had done that was xl for @ xl f(1) was equal to 1 @ xu f(4) was equal to -0.6137 which are of opposite signs if you recall in bisection method, we needed to give xl and xu of opposite signs and thats what is we need to do in fzero as well.

So now that we have created that function okay. We are going to say xSol = fzero okay. @x okay. x is the variable that we want to solve for, so @ x spacebar the name of the function. The name of the function is fun4Bisecx okay. So, f(x) is the name of that is the function. So, f(x) is exactly what we give over here okay.

This is nothing but our f(x). Just before f(x) we need to give with an inside the bracket, the name of the variables that we are going to solve for. And our initial guesses are 1 and 4. We will separate them with either a comma or a semicolon okay. And we press enter and we will get xSol.

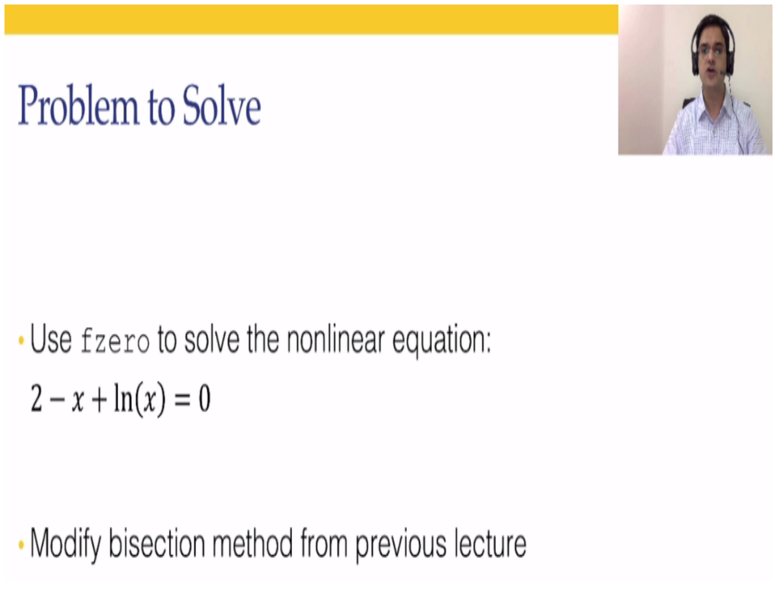
If you recall from the lecture 5.1, 3.4162 indeed was the solution that we got in the previous lecture okay. Let us go over again okay in the syntax of fzero. fzero is going to be xSol, the solution is going to be fzero in brackets, the name of the function, name of function of variable x, the initial guesses. Our initial guesses were 1 and 4 okay.

Now before the name of the function on x what we had was, the variable that we need to solve for @ x. So @ in bracket the variable we need to solve for followed by name of the function file. The function file was fun4Bisec. I will later type fun4 and click and press tab over here and MATLAB will auto fill this for me based on the function names that match with fun4 okay.

So this is really this is really the main part of fzero. The main part of fzero is how to pass on that function. And what are or how to give the initial guesses okay. We press semicolon at the end of it and we will get the solution without a going on the screen okay and we get xSol okay. Let us say we were to give this instead of 1 and 4, let us say we were to give this as 2 and 4 again. 2 and 4 bracket.

The solution and to bracket the same solution and if we were to press enter, we are going to get the same solution again. If you see on to your left hand side, our value of xSol has not really changed. Our xSol is the same as 3.1462 okay. Now instead if we were to give our initial guesses as 0 and 1 okay, now this is a problem. The problem is because log (0) cannot be obtained. So let us change this from 0.

Let us change to a very small number say 1e-5 and let us see what we get okay. And we give this and now we are getting the other solution. The other solution was 0.1586 and the first solution was 3.1462. Recall what we did in the previous lecture. We had shown that the curve intersects the x axis at 2 different points. We have now found out how to get both the solutions using fzero okay. (Video Ends: 10:34)
So let us go back to power point okay.
(Refer Slide Time: 10:37)



Problem to Solve

- Use `fzero` to solve the nonlinear equation:
  $2 - x + \ln(x) = 0$
- Modify bisection method from previous lecture

What the problem that we just solved was, to use fzero to solve the nonlinear equation 2-x+ln(x), we have created a function called fun4Bisec and use that function and passed it on to fzero. Now what we are going to do is the bisection method from the previous lecture. We are going to modify it okay. Instead of hard coding our f(x) inside our bisection method, we want to use the function that we just created for fzero and we are going to use that function in our bisection method.

(Video Starts: 11:10) So let us go back to MATLAB and do that okay. So again just to recap what our aim is, let me bring up our let us clear and clc clear all. Let us edit our bisection rule that we created in the previous lecture and we are going to modify that okay. In what we had done was this f(x), we had directly hard coded it in the overall running program.

Use a new method in which we are going to use this function fun4Bisec that we recently created in order to solve using the bisection rule okay. And this change is going to be fairly simple and everywhere. Where we are going to calculate function? You will, the function we are going to replace it with the function call itself. So fun4Bisec (xl) okay.

That is all. Remember what we did a few minutes earlier right. We said fun4Bisec okay. And fun4Bisec (4) gave nothing but the value of the function, fun4Bisec (1) gave nothing but the value of the function. If we were to say x= 4 and we were to call fun4Bisec with x, is going to give the value the function f(x). So all we going to do is, change this to fun4Bisec.

So let us just change this to fun4Bisec and fNew is also going to be fun4Bisec fNew sorry, fNew is going to be fun4Bisec, xNew, fun4Bisec. And that is all the changes that we need to do is nothing else that is changed. Let us save this and solve it in MATLAB okay. And the way we going to do is bisecRule, enter. And let us see our xNew and xNew is 3.1462. Now let us change our initial guesses instead of 1 and 4, let us change them to 0 or rather 1e-5 and 1.

Let us run this fun4Bisec sorry, let us run this bisecRule and see what the solution is going to be. xNew is going to be 0.1586 as we had obtained using our fzero method as well okay. And finally what we are going to do is, we are going to make one more change in our MATLAB code and we are going to have a stopping criteria based on an error criteria okay. So error tolerance are, let us call this as tolX okay.

Error in x is 1e-6. So if the error falls below or if x new-xl falls below 1e-6, we are going to stop this solution technique and that is what we are going to do over here. If abs of err is going to be less than tolX okay, then break. This is something that we have seen earlier also end .So we are going to break end less than 25 iterations.

So let us actually change the maxIter to 100 okay. So that we give enough number of iterations for the bisection rule in order for it converges. Let us give our initial guesses as before are 1 e-5 and 1. Let us save this and run bisecRule. Let us clear the screen clc and bisecRule okay. And our xNew in this case is 0.1586 okay.

And our and if we check our error, our error is 9*10 ^ -7 which is less than 10 ^ -6 okay. So now let us do one final thing. Change the initial guesses back to 1 and 4 and see how this runs okay. And run this bisecRule and press enter and our xNew is going to be the solution 3.1462.
That is the other solution over here. Our error is going to be 7*10 ^ -7. So our solution 3.1462 has converged to the error tolerance that we needed. And if we type i, we will know that we have converged in 23 iterations okay. (Video Ends: 16:33)

So what we have done. To recap what have done in this lecture, we have done 2 things in the lecture. The first thing that we did was to use fzero in order to solve the nonlinear equations 2-x+ln(x), the second thing that we did was 2 make our bisection rule better with 2 different things. One is to make it use the function fun4Bisec instead of hard coding it inside the script file itself

And the second change with it was that we stopped when as particular stopping criteria was met when the tolerance value was less than the desired accuracy. At that time we stopped the execution of the loop rather than letting the loop continue for a large number of iterations okay. So with that I come to the end of this lecture.

So thank you for listening to this lecture and in the next lecture we are going to cover the next set of numerical techniques for nonlinear algebraic equations. Specifically, we are going to start with fixed point iterations which is an open method and not a bracketing method. Thank you and see you in the next lecture.