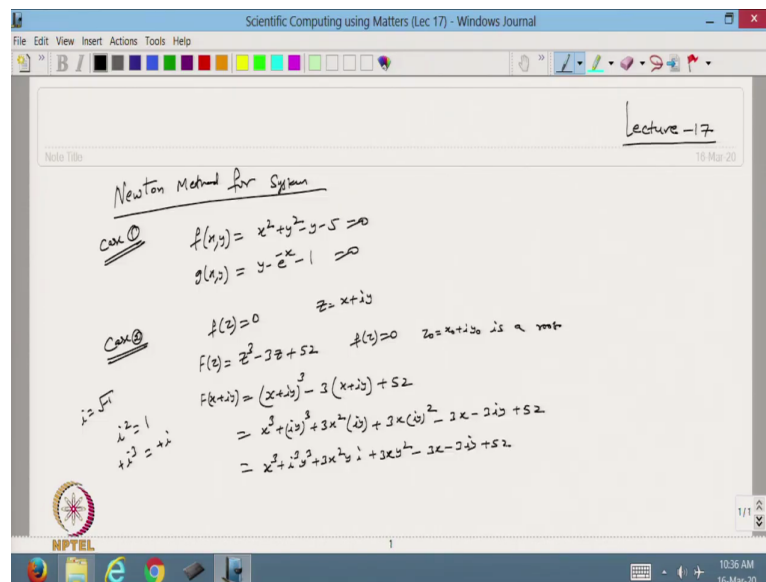


Scientific Computing Using MATLAB
Professor Vivek Aggarwal & Professor Mani Mehra
Department of Mathematics
Indian Institute of Technology, Delhi & Delhi Technological University, Delhi
Module 04
Lecture 17
MATLAB Code for Newton Method for Solving System of Equations

Hello viewers, welcome back to lecture number 17. So, today we are going to discuss lecture 17. So, in the previous lecture, we discussed how we can apply the Newton-Raphson method for solving a system of equations. So, today we will do the example based on that one and then we will do the MATLAB coding.

(Refer Slide Time: 00:44)



So, let us do one example here using the Newton method for the system. So, we are talking about here the 2-dimensional system like suppose I have $f(x, y)$.

$$f(x, y) = x^2 + y^2 - y - 5$$

$$g(x, y) = y - e^{-x} - 1$$

I want to find the roots of this equation. So, this is the case 1 I can take. So, this we will solve using the MATLAB code.

Now, I take the case 2, that how we can find the complex root. So, suppose I have a function in z , $F(z) = 0$ where z is a complex number $x + iy$. So, in this case, let us take the example that So, suppose z naught is equal to x naught plus iy naught is the root; is a root of this equation.

$$F(z) = z^3 - 3z + 52 = 0$$

So, you know that this is in the complex form and in fact it is an analytic function. So, I can split this function as a function of $x + iy$. So, this z is there so I should write this as the capital F . So, it is capital F .

$$\begin{aligned} F(z) &= (x + iy)^3 - 3(x + iy) + 52 \\ &= (x^3 - 3xy^2 - 3x + 52) + i(3x^2y - y^3 - 3y) \end{aligned}$$

I can collect the real terms and the imaginary terms.

(Refer Slide Time: 04:29)

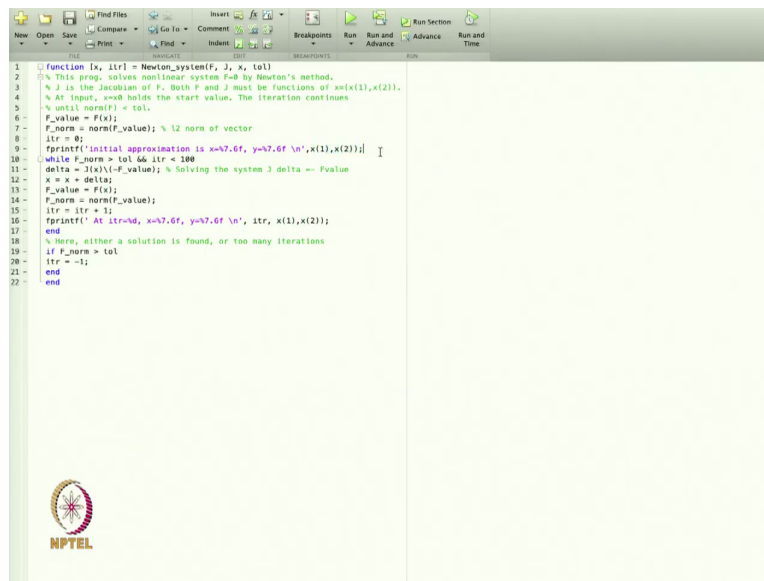
$$f(x, y) = (x^3 - 3xy^2 - 3x + 52)$$

$$g(x, y) = (3x^2y - y^3 - 3y)$$

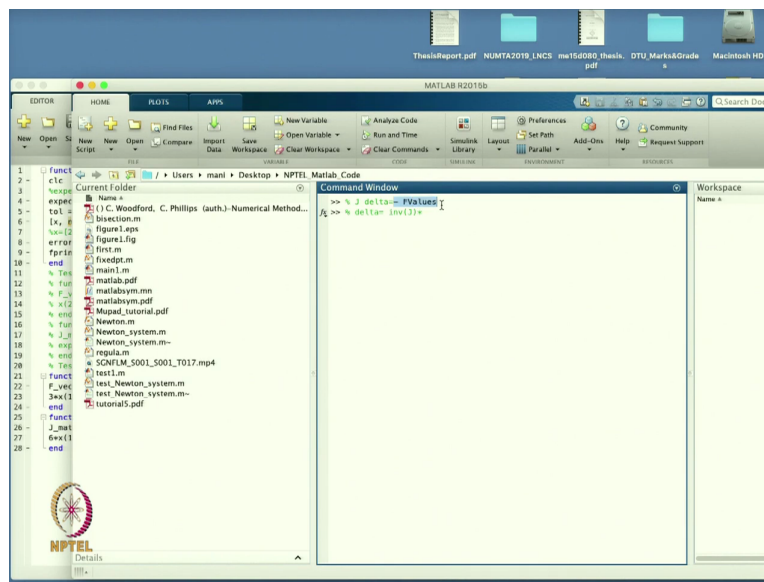
So that will be the real part and the imaginary part. So, from here I can write my $F(z)$ is equal to this.

So, then we can say, so whatever the value of x and y I am getting from here, I can write that z is equal to that $x_0 + iy_0$ is a root of equation number, I can take this equation number 1. So, by this way, we can find the root of the equation. So, let us go to the MATLAB code. Let us MATLAB code. So today, let us start with the MATLAB code.

(Refer Slide Time: 07:51)



```
1 function [x, itr] = Newton_system(F, J, x, tol)
2 % This prog. solves nonlinear system F=0 by Newton's method.
3 % J is the Jacobian of F. Both F and J must be functions of x=[x(1),x(2)].
4 % At input, x=0 holds the start value. The iteration continues
5 % until norm(F) < tol.
6 F_value = F(x);
7 F_norm = norm(F_value); % l2 norm of vector
8 itr = 0;
9 fprintf('Initial approximation is x=7.6f, y=7.6f \n', x(1), x(2));
10 while F_norm > tol && itr < 100
11 delta = -J(x)\F_value; % Solving the system J delta = -F_value
12 x = x + delta;
13 F_value = F(x);
14 F_norm = norm(F_value);
15 itr = itr + 1;
16 fprintf('At itr=%d, x=7.6f, y=7.6f \n', itr, x(1), x(2));
17 end
18 % Here, either a solution is found, or too many iterations
19 if F_norm > tol
20 itr = -1;
21 end
22 end
```



So, today I have already made the code for you because now we are familiar with how to write a code. So, let us start with the code I have made. So, this is the code we have I have started with. So that is the function value. So, this is the function and it gives you the x and iteration. Now, I keep the name Newton_system.

So, this capital F is the function that is whatever we are going to define. J is the Jacobian that we already know how to because in this case I have to find the Jacobian also.

Where $x = [x(1) \ x(2)]$ and this is the tolerance we are given.

So, this program solves nonlinear system $F=0$ by Newton method. J is Jacobian; Jacobian we know that it is a partial derivative with respect to the variable x and y. So, that is the Jacobian.

So, this is my $x = [x(1) \ x(2)]$. And that $x = x_0$ is the starting value, the initial value what we are going to start with.

So, now from here I write that $F(x)$. So, whatever the function I am getting I put the value of x and I get the value of F , F_value . Then, in this case because it is a vector valued function, so we have to find the norm of that one; instead of absolute value, we are finding the norm. So, l2 norm if we take, so this will be the l2 norm. So, that will be the norm.

```
function [x, itr] = Newton_system(F, J, x, tol)
```

```
% This prog. solves nonlinear system  $F=0$  by Newton's method.
```

```
% J is the Jacobian of F. Both F and J must be functions of  $x=(x(1),x(2))$ .
```

```
% At input,  $x=x_0$  holds the start value. The iteration continues
```

```
% until  $\text{norm}(F) < \text{tol}$ .
```

```
F_value = F(x);
```

```
F_norm = norm(F_value); % l2 norm of vector
```

```
itr = 0;
```

```
fprintf('initial approximation is x=%7.6f, y=%7.6f\n', x(1), x(2));
```

```
while F_norm > tol && itr < 100
```

```
delta = J(x)\(-F_value); % Solving the system  $J \delta = -F\_value$ 
```

```
x = x + delta;
```

```
F_value = F(x);
```

```
F_norm = norm(F_value);
```

```
itr = itr + 1;
```

```
fprintf(' At itr=%d, x=%7.6f, y=%7.6f\n', itr, x(1), x(2));
```

```
end
```

```
% Here, either a solution is found, or too many iterations
```

```
if F_norm > tol
```



```
itr = -1;
```

```
end
```

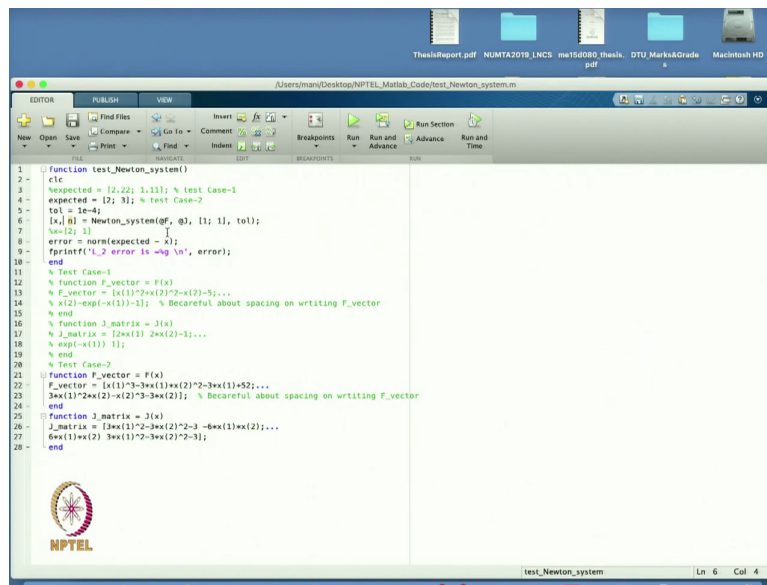
```
end
```

Now, I will put the value of delta in the initial approximation and my approximation x will be improved and then I will find the value of F because this is a function we are passing. So, whatever the new x I am getting here, I will get the value of the function here. So, that is the F value and the same one it is the norm of that function.

Now, I will find the iteration, increase the iteration by 1 and then I will be making the print of that one that what is the root of this finding the root at each iteration. So, this will be written here and then it will do the end of this while loop. And now, if it is here that here either a solution is found or too many iterations.

So, if F_norm is greater than tolerance, iteration will be minus 1 and then this is the end of the function. So, this function I have started with the name Newton_system. So, this name and the file name will be the same. So, this is my function we are calling. Now I have to write the main program. So, the main program is written like this.

(Refer Slide Time: 13:21)



So here because we are dealing with the system of equations, so I am writing this function in the form of a function value. So, I am writing: function test_Newton_system and no argument. It means it is, it will work as a script file. So, in this case, I will start with the name of this function that it test_Newton_system. So, I will start with the clc whatever is written there. Then, I will give the expected value; expected root. So, whatever the root I am going to expect, so I will write here then I will pass the tolerance. And then this is the function I am calling from here the Newton_system what just we have defined.

```
function test_Newton_system()

clc

%expected = [2.22; 1.11]; % test Case-1

expected = [2; 3]; % test Case-2

tol = 1e-4;

[x, n] = Newton_system(@F, @J, [1; .5], tol);

%x=[2; 1]

error = norm(expected - x);

fprintf('L_2 error is =%g \n', error);

end
```

```

%Test Case-1

% function F_vector = F(x)

% F_vector = [x(1)^2+x(2)^2-x(2)-5;...
% x(2)-exp(-x(1))-1]; % Be careful about spacing on writing F_vector

% end

% function J_matrix = J(x)

% J_matrix = [2*x(1) 2*x(2)-1;...
% exp(-x(1)) 1];

% end

% Test Case-2

function F_vector = F(x)

F_vector = [x(1)^3-3*x(1)*x(2)^2-3*x(1)+52;...
3*x(1)^2*x(2)-x(2)^3-3*x(2)]; %Be Careful about spacing on writing
F_vector

end

function J_matrix = J(x)

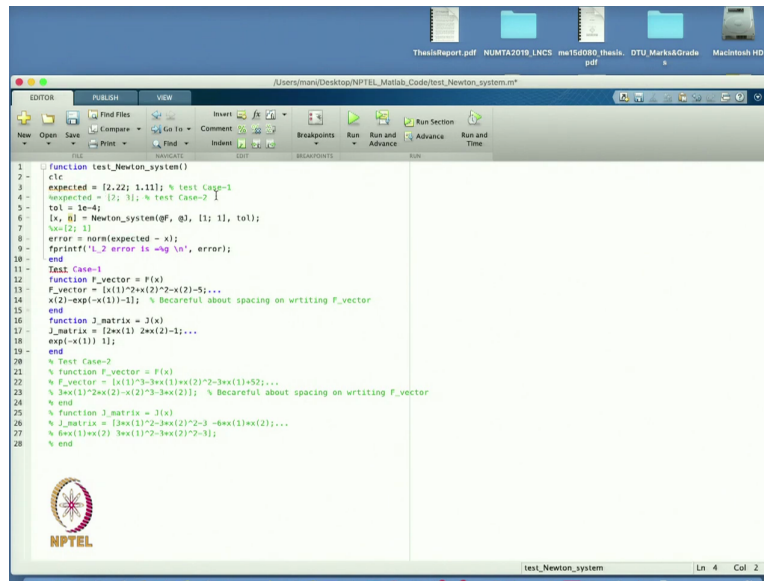
J_matrix = [3*x(1)^2-3*x(2)^2-3 -6*x(1)*x(2);...
6*x(1)*x(2) 3*x(1)^2-3*x(2)^2-3];

end

```

So, this is a function passing at the rate (@). So the unanimous function I am passing. So, this is the F, that is the Jacobian. This is the initial approximation that was the x here, and that is the tolerance we are passing. And then the error, I am finding the norm. So, norm is l2 norm. Expected value, whatever the root is there, minus whatever the approximation we are getting. So, I am finding that error and here I am printing the error. So, this is a function we are finding here.

(Refer Slide Time: 15:24)



So now, I will start with my case 1 and case 2. So, let us start with so I will just be starting with the case 1. So, this is I am going to start with. Now my expected value will be 2.22 and 1.11 for the test case 1; so, whatever the test case 1 I have taken, okay. So, in this case, I am starting with the initial 1, 1 and tolerance I am taking 10^{-4} .

Now I will start with the function. So, my function was $x^2 + y^2 - y - 5$. So, I am finding writing a function that is a vector value.

```
function F_vector = F(x)

F_vector = [x(1)^2+x(2)^2-x(2)-5;...

x(2)-exp(-x(1))-1];

end
```

And the second value is, here I am writing the semicolon means I am writing a column vector and these 3 dots are that if the vector is, if the value of the function is very large then sometime, we have to write the value in the next in the line. So, to further continue the line we put the 3 dots. So, it means that the next line will be the continuation of this one.

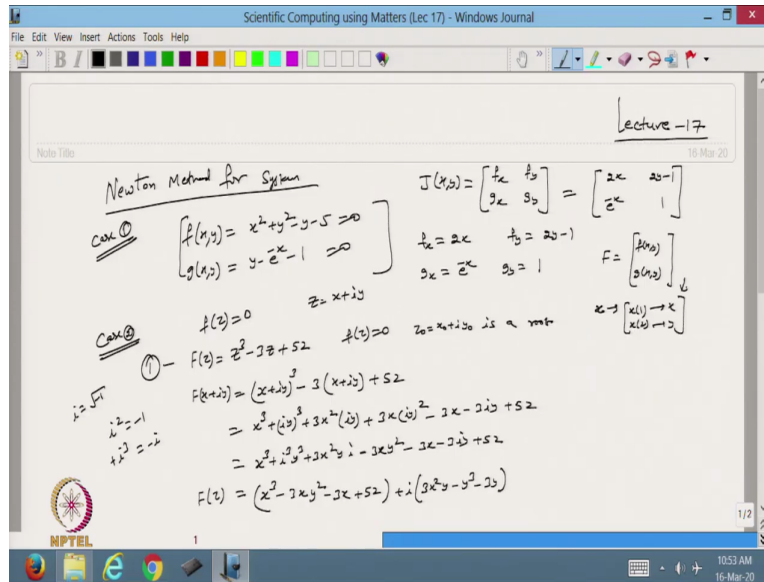
So, this is my first function and the second function $g(x, y)$ was $y - e^{-x} - 1$. So that is the function value we have defined. So, we have defined a column vector with the $F(x, y)$ here and $g(x, y)$ here. Now, this is the first function we have defined. Now I have defined the Jacobian. So, the J_matrix I am defining. So, from here I am taking the value of x . So, $J(x)$ will be defined here. So, the J_matrix is equal to $J(x)$. So, J_matrix will be what? I am taking the partial derivative of this function.

function J_matrix = J(x)

J_matrix = [2*x(1) 2*x(2)-1;...

exp(-x(1)) 1];

(Refer Slide Time: 17:31)



So, x square so let us say from here. So, I am taking the Jacobian now. So, Jacobian basically is this one.

$$J = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}$$

$$J = \begin{bmatrix} 2x & 2y - 1 \\ e^{-x} & 1 \end{bmatrix}$$

So this is the Jacobian.

I am going to pass for this function. So, this is my J (x) and F_vector I have defined like this one: f (x, y) and g (x, y) and this x in the code I have defined is a vector. So, x (1) and x (2). So, basically this is my x and this is my y because in the MATLAB code, we have to define the multivalued vector in this form or multivalued variable in this form: x and y (in the vector form). So, that is what we are going to define.

(Refer Slide Time: 19:16)

Handwritten work on a digital whiteboard:

$$F(z) = (x^3 - 3xy^2 - 3x + 52) + i(3x^2y - y^3 - 3y)$$

$$F(z) = f(x,y) + i g(x,y)$$

Let $z = x + iy$ is a root of eq (1)

Let us go to the Matlab Code.

$$F(x,y) = \begin{bmatrix} f(x,y) \\ g(x,y) \end{bmatrix}$$

$$J(x,y) = \begin{bmatrix} 3x^2 - 3y^2 - 3 & -6xy \\ 6xy & 3x^2 - 3y^2 - 3 \end{bmatrix}$$

The same way for this function. Now, I will define my F (z).

$$f(x, y) = (x^3 - 3xy^2 - 3x + 52)$$

$$g(x, y) = (3x^2y - y^3 - 3y) \quad \text{and}$$

$$J(x) = \begin{bmatrix} 3x^2 - 3y^2 - 3 & -6xy \\ 6xy & 3x^2 - 3y^2 - 3 \end{bmatrix}$$

(Refer Slide Time: 20:35)

MATLAB script editor showing the following code:

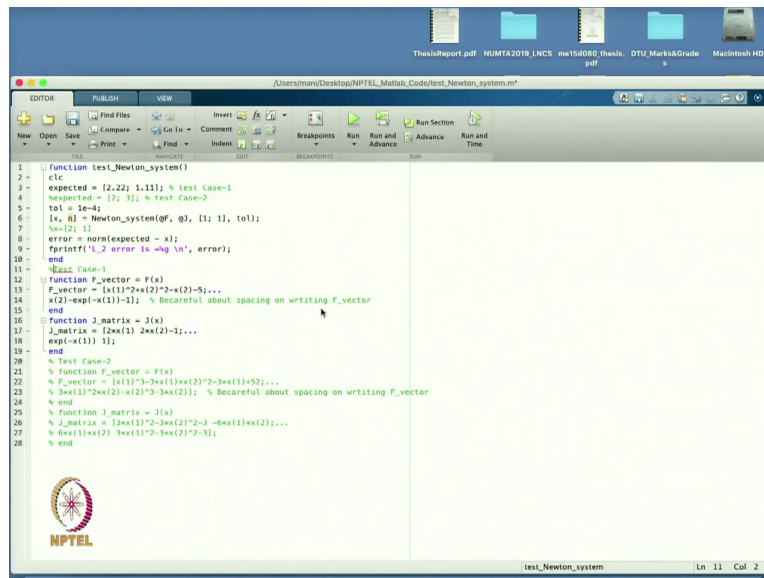
```

1 function test_Newton_system()
2   clc
3   %expected = [2.22; 1.11]; % Test Case-1
4   expected = [2; 3]; % Test Case-2
5   tol = 1e-4;
6   [x, y] = Newton_system(@F, @J, [1; 1], tol);
7   %x(2), y(2)
8   error = norm(expected - x);
9   fprintf('1_2 error is %g \n', error);
10  end
11
12 % Test Case-1
13 % function F_vector = F(x)
14 % F_vector = [x(1)^3-3*x(1)*x(2)^2-3*x(1)+52;...
15 % x(2)-exp(-x(1))-1]; % Be careful about spacing on writing F_vector
16 % end
17 % function J_matrix = J(x)
18 % J_matrix = [3*x(1)^2-3*x(2)^2-3 -6*x(1)*x(2);...
19 % 6*x(1)*x(2) 3*x(1)^2-3*x(2)^2-3];
20 % end
21
22 % Test Case-2
23 % function F_vector = F(x)
24 % F_vector = [x(1)^3-3*x(1)*x(2)^2-3*x(1)+52;...
25 % 3*x(1)^2-3*x(2)^2-3-6*x(1)*x(2)]; % Be careful about spacing on writing F_vector
26 % end
27 % function J_matrix = J(x)
28 % J_matrix = [3*x(1)^2-3*x(2)^2-3 -6*x(1)*x(2);...
29 % 6*x(1)*x(2) 3*x(1)^2-3*x(2)^2-3];
30 % end

```

So, this is the way we have defined. So, let us write my $J(x)$ here. So, now my $J(x)$ is defined. So, as in the previous one, we defined the $J(x)$. So, $J(x)$ was, so it was $2x$ then $2y-1$ e^{-x} and 1 . So, that is my Jacobian. So, now from here I will get the value of x . From here, I will pass the value of F_vector . So, F_vector is equal to $f(x)$ and J matrix is equal to $g(x)$. And these vectors, these values of the vectors will be passed to this function. So, at the rate ($@$) F and at the rate ($@$) means this is a function value. Now, let us run this one.

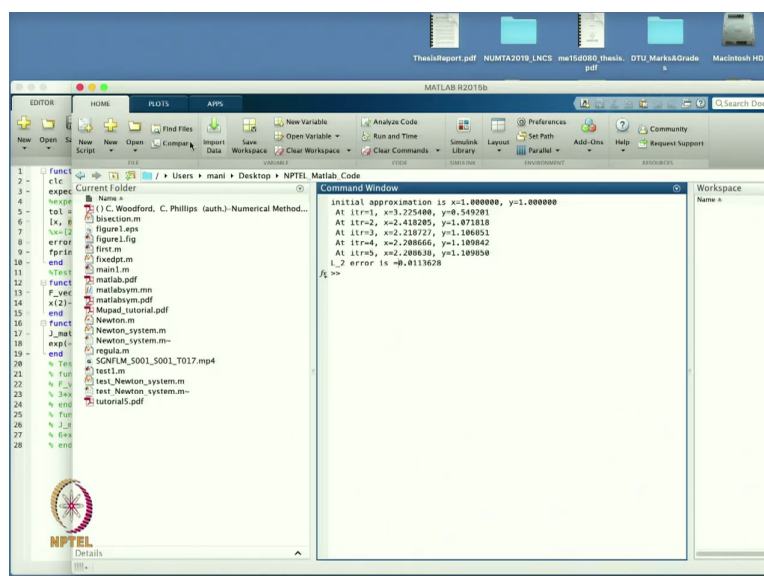
(Refer Slide Time: 21:28)



```

1 function test_Newton_system()
2
3 c1c
4 expected = [2.22; 1.11]; % test Case-1
5 % expected = [2; 3]; % test Case-2
6 tol = 1e-4;
7 [x, l1] = Newton_system(@f, @J, l1, tol);
8 % x=[2; 3]
9 error = norm(expected - x);
10 fprintf('l2 error is %g\n', error);
11 end
12 % Test Case-1
13 function F_vector = f(x)
14 F_vector = [x(1)^2*x(2)^2-x(2)-5;...
15 x(2)-exp(-x(1))-1]; % Be careful about spacing on wrtiting F_vector
16 end
17 function J_matrix = J(x)
18 J_matrix = [2*x(1) 2*x(2)-1;...
19 exp(-x(1)) 1];
20 end
21 % Test Case-2
22 function F_vector = f(x)
23 F_vector = [x(1)^3-3*x(1)*x(2)^2-3*x(1)+52;...
24 3*x(1)^2*x(2)-x(2)^3-3*x(2)]; % Be careful about spacing on wrtiting F_vector
25 end
26 function J_matrix = J(x)
27 J_matrix = [3*x(1)^2-3*x(2)^2-3 -6*x(1)*x(2);...
28 6*x(1)*x(2) 3*x(1)^2-3*x(2)^2-3];
29 end

```



```

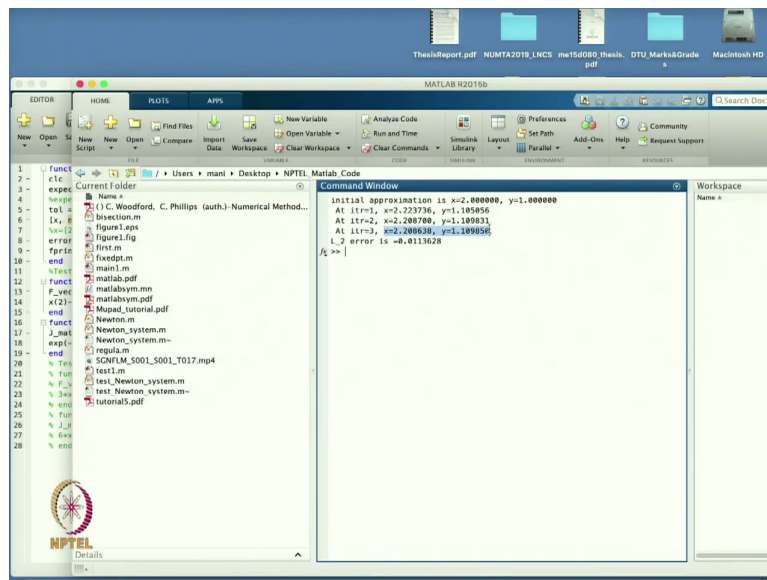
Initial approximation is x=1.000000, y=1.000000
At itr=1, x=5.225488, y=0.549201
At itr=2, x=2.418285, y=1.071818
At itr=3, x=2.218727, y=1.106851
At itr=4, x=2.208666, y=1.109842
At itr=5, x=2.208639, y=1.109838
l2 error is 0.0113628
f2 >>

```

So, let us so there is some end term. So, this is we have to write function end, function end and then function end. This is the test case so we have to write like this one. Now, find the value of the function. So, this is so I started with $1, 1$ and after 5 iterations, that is my solution. So, it is 2.20 and 1.10 . And my error, $l2$ error is giving is finding here 0.011 . So, this

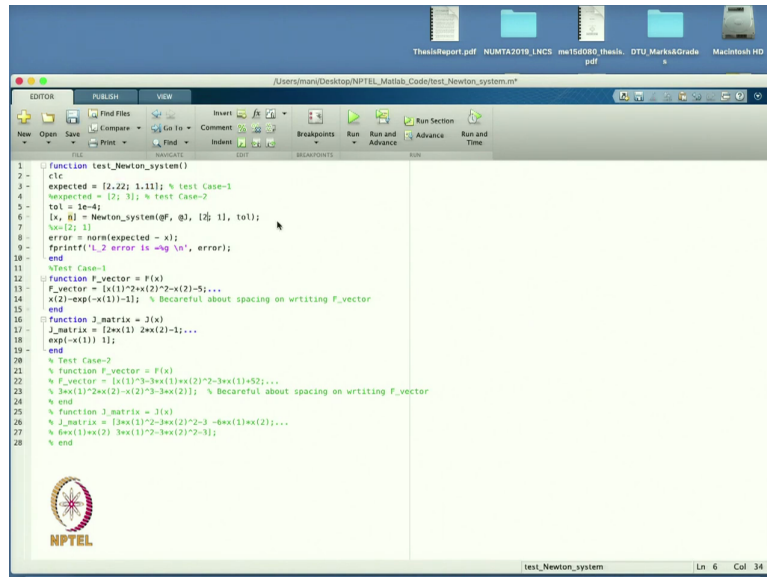
is the value we are getting because here the expected value is also not known. It is just the approximation value of the roots. So, that is why whatever the value we are getting is the value corresponding to, the error corresponding to this expected value.

(Refer Slide Time: 22:33)



The screenshot shows the MATLAB R2016b Command Window. The output of the function is as follows:

```
Initial approximation is x=2.000000, y=1.000000
At itr=1, x=2.223736, y=1.109896
At itr=2, x=2.288780, y=1.109831
At itr=3, x=2.288836, y=1.109836
L2 error is -0.4113628
```



The screenshot shows the MATLAB R2016b Editor with the code for the Newton's method function. The code is as follows:

```
function test_Newton_system()
    c1c
    expected = [2.22; 1.11]; % test Case-1
    unexpected = [2; 1]; % test Case-2
    tol = 1e-4;
    [x, B] = Newton_system(x0, [2; 1], tol);
    error = norm(expected - x);
    fprintf('L2 error is %g\n', error);
end

% Test Case-1
function F_vector = F(x)
    F_vector = [x(1)^2*x(2)^2-x(2)-5;...
                x(2)-exp(-x(1))-1]; % Be careful about spacing on writing F_vector
end

function J_matrix = J(x)
    J_matrix = [2*x(1) 2*x(2)-1;...
                exp(-x(1)) 1];
end

% Test Case-2
function F_vector = F(x)
    F_vector = [x(1)^3-3*x(1)*x(2)^2-3*x(1)+52;...
                3*x(1)^2*x(2)-x(2)^3-3*x(2)]; % Be careful about spacing on writing F_vector
end

function J_matrix = J(x)
    J_matrix = [3*x(1)^2-3*x(2)^2-3 -6*x(1)*x(2);...
                6*x(1)*x(2) 3*x(1)^2-3*x(2)^2-3];
end
```


The screenshot shows the MATLAB R2015b interface. The Command Window displays the following output:

```

Initial approximation is x=8.000000, y=1.000000
At itr=1, x=-4.000000, y=6.000000
At itr=2, x=-3.036150, y=2.973789
At itr=3, x=-2.088849, y=2.097484
At itr=4, x=-1.289620, y=2.621344
At itr=5, x=-0.764780, y=2.725211
At itr=6, x=-0.569884, y=2.778997
At itr=7, x=-0.545771, y=2.725463
At itr=8, x=-0.545467, y=2.725414
L-2 error is +3.28271

```

The screenshot shows the MATLAB R2015b interface with the script `test_Newton_system.m` open in the Editor. The script contains the following code:

```

1 function test_Newton_system()
2
3 expected = [2.22; 1.11]; % test Case-1
4 expected = [2; 1]; % test Case-2
5 tol = 1e-4;
6 [x, y] = Newton_system(x0, y0, [0; 1], tol);
7
8 error = norm(expected - x);
9 fprintf('L-2 error is %g\n', error);
10
11 %Test Case-1
12 function F_vector = F(x)
13 F_vector = [x(1)^2*x(2)^2-x(2)-5;...
14             x(2)-exp(-x(1))-1]; % Be careful about spacing on writing F_vector
15 end
16 function J_matrix = J(x)
17 J_matrix = [2*x(1) 2*x(2)-1;...
18             exp(-x(1)) 1];
19 end
20
21 % Test Case-2
22 function F_vector = F(x)
23 F_vector = [x(1)^3-3*x(1)*x(2)^2-3*x(1)+52;...
24             3*x(1)^2*x(2)-x(2)^3-3*x(2)]; % Be careful about spacing on writing F_vector
25 end
26 function J_matrix = J(x)
27 J_matrix = [3*x(1)^2-3*x(2)^2-3 -4*x(1)*x(2);...
28             6*x(1)*x(2) 3*x(1)^2-3*x(2)^2-3];
29 end

```

Now, we can even change the initial condition and then let us run this one. So, in this case, you can see that even with the three iterations we are getting the solution and the error is this one. So based on that, whatever the value you are going to take, let us see with the 0, 1. So, see, in this case, we have to take 8 iterations and the results are giving you the different roots.

So, it is giving the x is equal to this value and y is equal to this value. So, in this case, my root is not getting the same one because you have to tell in the initial that around which you are going to find the root. So, in this case, I am going to find the root near to 2.22 and 1.11. So that is why I have to start with this value.

(Refer Slide Time: 23:28)

The top screenshot shows the MATLAB script 'test_Newton_system.m' in the Editor. The script defines two test cases. Test Case-1 uses initial conditions $x_0 = [1.5; 1]$ and a tolerance of $1e-4$. Test Case-2 uses initial conditions $x_0 = [1.5; 1]$ and a tolerance of $1e-4$. The script defines the function $F(x)$ and the Jacobian matrix $J(x)$.

```

1 function test_Newton_system()
2     clc
3     expected = [2.22; 1.11]; % test Case-1
4     unexpected = [2; 1]; % test Case-2
5     tol = 1e-4;
6     [x, it] = Newton_system(@F, @J, [1.5; 1], tol);
7     %x[2; 1]
8     error = norm(expected - x);
9     fprintf('L2 error is %g\n', error);
10 end
11 %Test Case-1
12 function F_vector = F(x)
13     F_vector = [x(1)^2*x(2)^2-x(2)-5;...
14               x(2)-exp(-x(1))-1]; % Be careful about spacing on writing F_vector
15 end
16 function J_matrix = J(x)
17     J_matrix = [2*x(1)*x(2)+x(2)-1;...
18               -exp(-x(1))];
19 end
20 %Test Case-2
21 function F_vector = F(x)
22     F_vector = [x(1)^3-3*x(1)*x(2)^2-3*x(1)+52;...
23               x(1)^2*x(2)-x(2)^3-3*x(2)]; % Be careful about spacing on writing F_vector
24 end
25 function J_matrix = J(x)
26     J_matrix = [3*x(1)^2+3*x(2)^2-3 -6*x(1)*x(2);...
27               2*x(1)*x(2) 3*x(1)^2-3*x(2)^2-3];
28 end

```

The bottom screenshot shows the MATLAB R2015b interface. The script is loaded in the Editor. The Command Window displays the iterative results of the Newton-Raphson method for Test Case-1. The results show convergence to the expected root $[2.22; 1.11]$ after 8 iterations.

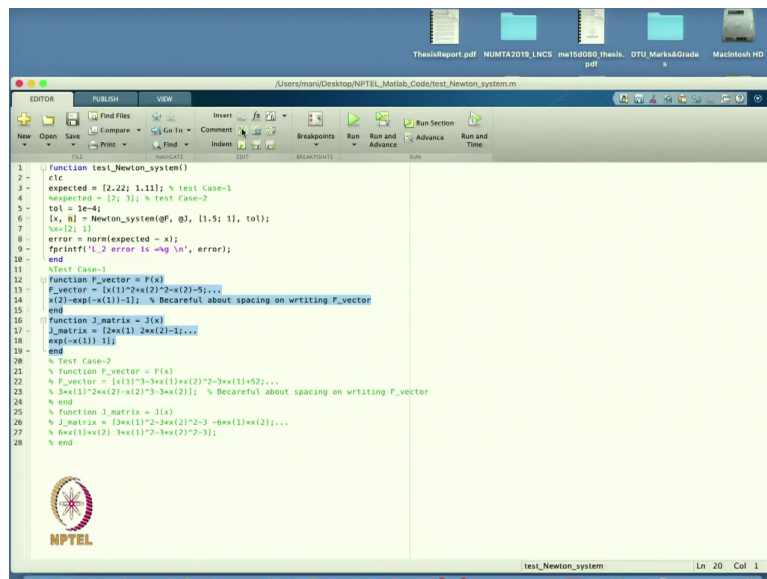
```

Initial approximation is x=0.000000, y=1.000000
At itr=1, x=-4.000000, y=6.000000
At itr=2, x=-3.036150, y=2.973789
At itr=3, x=-2.000000, y=2.007664
At itr=4, x=-1.289620, y=2.621344
At itr=5, x=-0.764789, y=2.725211
At itr=6, x=-0.569884, y=2.728807
At itr=7, x=-0.545771, y=2.725403
At itr=8, x=-0.545467, y=2.725414
L2 error is =3.20271

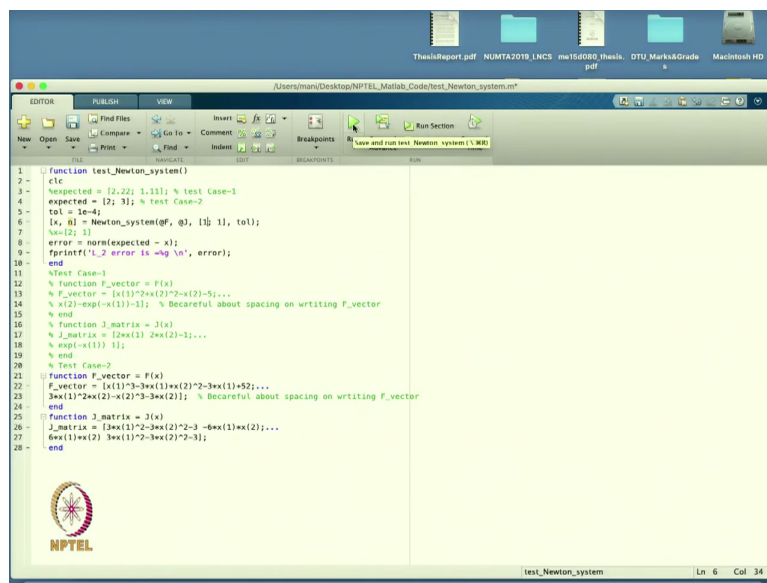
```

Or maybe I can start with 1.5. And then let us see what will happen? Now we are adding convergence to this root. So, it is based on which roots you want because it may have more than one root and you do not know which roots you are going to find. So, in this case, I am going to find the root here. So, I take this initial condition and based on this initial condition I am my route is converging to 2.20 and 1.10. So, this is what we expected. So that is the way we can solve the test case 1.

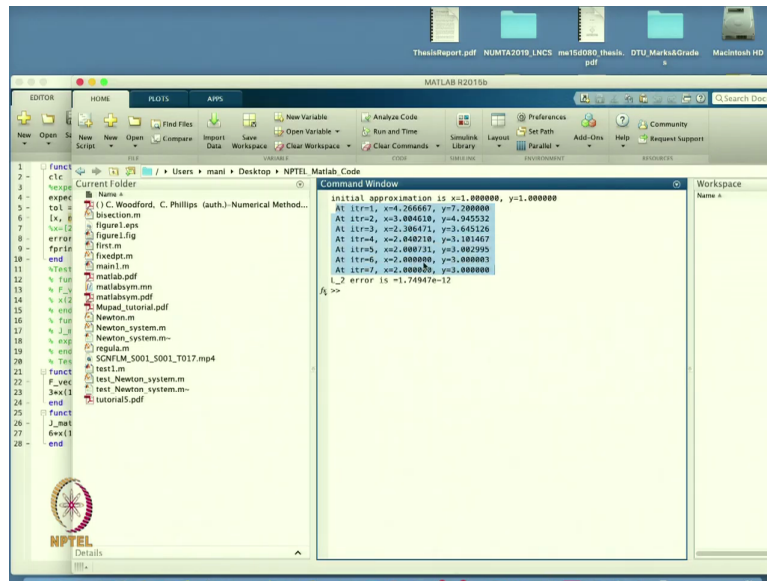
(Refer Slide Time: 24:08)



```
1 function test_Newton_system()
2   clc
3   expected = [2.22; 1.11]; % test Case-1
4   expected = [2; 3]; % test Case-2
5   tol = 1e-4;
6   [x, it] = Newton_system(@F, @J, [1; 5], 1, tol);
7   %x(2) 1
8   error = norm(expected - x);
9   fprintf('1.2 error is %g \n', error);
10  end
11  %Test Case-1
12  % function F_vector = F(x)
13  F_vector = [x(1)^2*x(2)^2-x(2)-5;...
14             x(2)-exp(-x(1))-1]; % Decareful about spacing on wrtiting F_vector
15  % end
16  % function J_matrix = J(x)
17  J_matrix = [2*x(1)*2*x(2)-1;...
18             exp(-x(1)) 1];
19  % end
20  % Test Case-2
21  % function F_vector = F(x)
22  F_vector = [x(1)^3-3*x(1)*x(2)^2-3*x(1)+52;...
23             3*x(1)^2*x(2)-x(2)^3-3*x(2)]; % Decareful about spacing on wrtiting F_vector
24  % end
25  % function J_matrix = J(x)
26  J_matrix = [3*x(1)^2-3*x(2)^2-3 -6*x(1)*x(2);...
27             6*x(1)*x(2) 3*x(1)^2-3*x(2)^2-3];
28  % end
```



```
1 function test_Newton_system()
2   clc
3   expected = [2.22; 1.11]; % test Case-1
4   expected = [2; 3]; % test Case-2
5   tol = 1e-4;
6   [x, it] = Newton_system(@F, @J, [1; 5], 1, tol);
7   %x(2) 1
8   error = norm(expected - x);
9   fprintf('1.2 error is %g \n', error);
10  end
11  %Test Case-1
12  % function F_vector = F(x)
13  F_vector = [x(1)^2*x(2)^2-x(2)-5;...
14             x(2)-exp(-x(1))-1]; % Decareful about spacing on wrtiting F_vector
15  % end
16  % function J_matrix = J(x)
17  J_matrix = [2*x(1)*2*x(2)-1;...
18             exp(-x(1)) 1];
19  % end
20  % Test Case-2
21  % function F_vector = F(x)
22  F_vector = [x(1)^3-3*x(1)*x(2)^2-3*x(1)+52;...
23             3*x(1)^2*x(2)-x(2)^3-3*x(2)]; % Decareful about spacing on wrtiting F_vector
24  % end
25  % function J_matrix = J(x)
26  J_matrix = [3*x(1)^2-3*x(2)^2-3 -6*x(1)*x(2);...
27             6*x(1)*x(2) 3*x(1)^2-3*x(2)^2-3];
28  % end
```



Now I will do the test case 2. So, I will comment on this one, and we will start with this. So, let us do the test case 2. So, in this case, I will change my expected value. So, in this case, the complex functions whatever we have defined, so this is the function we have defined: The F_vector is $x^3 - 3xy^2 - 3x + 52$. So, that was my $f(x, y)$ and the second was $3x^2y - y^3 - 3y$.

So, that was my $g(x, y)$; the other function. And the Jacobian we have defined.

$$J(x, y) = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}$$

So, Jacobian we are getting and I am expecting this value as 2 and 3 as a root of this. So, we will start with this same initial condition, and I run this one and let us see.

So, based on this one initial approximation (1, 1) after the seventh iteration, we are getting my root 2, 3. And because in this case, we already know the exact value of the root that is 2 and 3. So you can see that l2 error is very small; 10^{-12} . So, in this case, my root is almost after 7 iteration, our root is very close to the exact value. It is converging very fast. So maybe I can take some other condition.

(Refer Slide Time: 26:00)

```

1 function test_Newton_system()
2     c1c
3     %expected = [2.22; 1.11]; % test Case-1
4     %expected = [2; 3]; % test Case-2
5     tol = 1e-4;
6     [x, l2] = Newton_system(F, @, [], 0; 11, tol);
7     %x=[2; 3]
8     error = norm(expected - x);
9     fprintf('l2 error is %g\n', error);
10    end
11    %Test Case-1
12    % function F_vector = F(x)
13    % F_vector = [x(1)^2+x(2)^2-x(2)-5;...
14    % x(2)-exp(-x(1))-1]; % Decareful about spacing on wrtting F_vector
15    % end
16    % function J_matrix = J(x)
17    % J_matrix = [2*x(1) 2*x(2)-1;...
18    % exp(-x(1)) 1];
19    % end
20    % Test Case-2
21    % function F_vector = F(x)
22    % F_vector = [x(1)^3-3*x(1)*x(2)^2-3*x(1)+52;...
23    % 3*x(1)^2*x(2)-x(2)^3-3*x(2)]; % Decareful about spacing on wrtting F_vector
24    % end
25    % function J_matrix = J(x)
26    % J_matrix = [3*x(1)^2-3*x(2)^2-3 -6*x(1)*x(2);...
27    % 6*x(1)*x(2) 3*x(1)^2-3*x(2)^2-3];
28    % end

```

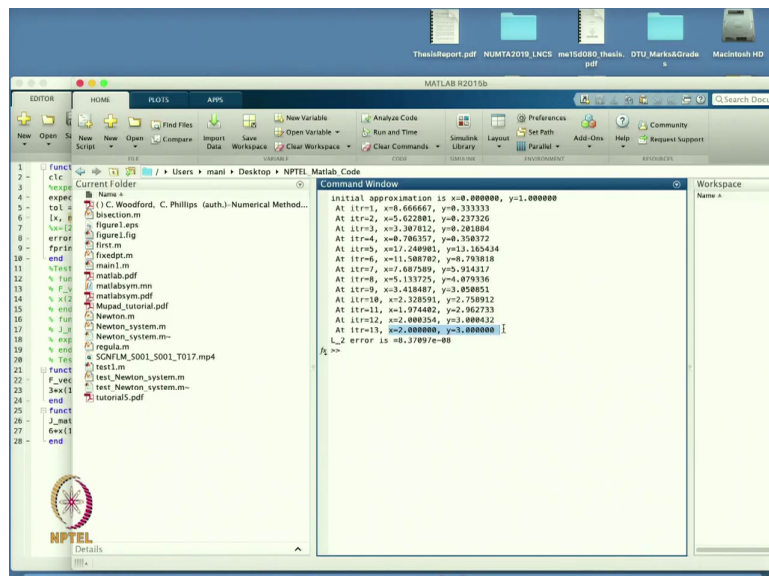
```

Initial approximation is x=0.000000, y=1.000000
At itr=1, x=6.66667, y=0.333333
At itr=2, x=5.622881, y=0.237326
At itr=3, x=3.387812, y=0.281884
At itr=4, x=0.706357, y=0.358372
At itr=5, x=17.248901, y=13.185434
At itr=6, x=11.588782, y=0.793818
At itr=7, x=1.687269, y=0.914317
At itr=8, x=5.133725, y=4.079336
At itr=9, x=3.418487, y=3.058851
At itr=10, x=2.128591, y=2.758912
At itr=11, x=1.074402, y=2.962733
At itr=12, x=2.000354, y=3.000423
At itr=13, x=2.000000, y=3.000000
l2 error is 0.37897e-08

```

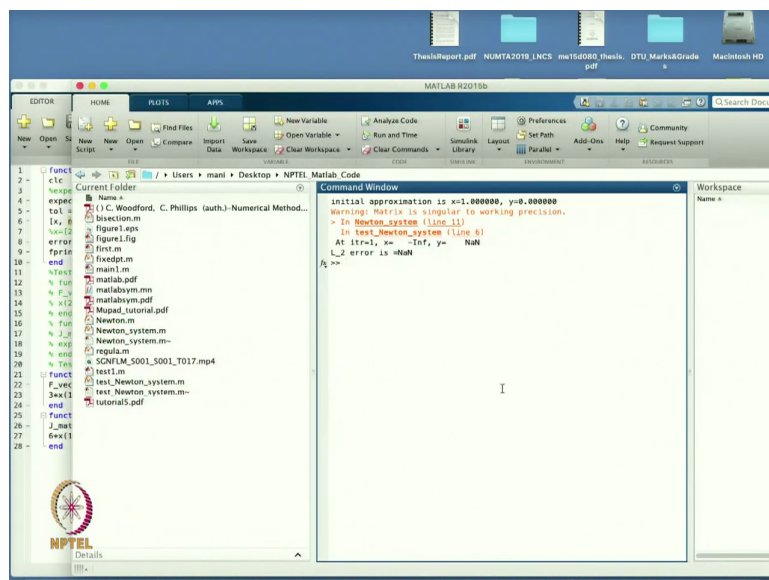
Maybe I start with 0 and 1 and let us see whether we are heading toward the same root or the different root. So, from here, you can see that we started with this one, and in this case, the iterations are large, 13 but after the 13 iterations, I am heading toward the root. So, this is the approximation of the root and l2 error is 10^{-8} . So that is the value of the root of the function $F(z)$ equals 0.

(Refer Slide Time: 26:41)



The screenshot shows the MATLAB R2015b environment. The Command Window displays the results of a Newton-Raphson iteration process. The initial approximation is $x=0.000000$, $y=1.000000$. The iterations proceed until the error is $0.27897e-08$.

```
Initial approximation is x=0.000000, y=1.000000
At itr=1, x=0.666667, y=0.333333
At itr=2, x=0.622081, y=0.237326
At itr=3, x=0.387812, y=0.281884
At itr=4, x=0.786357, y=0.358372
At itr=5, x=0.748981, y=0.162434
At itr=6, x=1.588782, y=0.793818
At itr=7, x=0.887589, y=0.914317
At itr=8, x=0.133725, y=0.879336
At itr=9, x=0.418487, y=0.858851
At itr=10, x=0.228391, y=0.758912
At itr=11, x=1.974482, y=0.962733
At itr=12, x=0.088354, y=0.088432
At itr=13, x=0.000000, y=0.000000
L_2 error is 0.27897e-08
```



The screenshot shows the MATLAB R2015b environment. The Command Window displays a warning message: "Warning: Matrix is singular to working precision." This occurs during the Newton-Raphson iteration process. The error is NaN .

```
Initial approximation is x=1.000000, y=0.000000
Warning: Matrix is singular to working precision.
> In Newton_system (line 11)
> In test_Newton_system (line 6)
In test_Newton_system (line 6)
At itr=1, x= Inf, y= NaN
L_2 error is NaN
```



```

1 function test_Newton_system()
2 c1c
3 %expected = [2.22; 1.11]; % Test Case-1
4 %expected = [2; 5]; % Test Case-2
5 tol = 1e-4;
6 [x, y] = Newton_system(x0, y0, 11, .50, tol);
7 %x=[2; 1]
8 error = norm(expected - x);
9 fprintf('L_2 error is %g\n', error);
10 end
11 %Test Case-1
12 % function f_vector = f(x)
13 % f_vector = [x(1)^2+x(2)^2-x(2)-5;...
14 % x(2)-exp(-x(1))-1]; % Be careful about spacing on writing f_vector
15 % end
16 % function J_matrix = J(x)
17 % J_matrix = [2*x(1) 2*x(2)-1;...
18 % exp(-x(1)) 1];
19 % end
20 % Test Case-2
21 function f_vector = f(x)
22 f_vector = [x(1)^3-3*x(1)*x(2)^2-3*x(1)*x(2);...
23 3*x(1)^2*x(2)-x(2)^2-3*x(2)]; % Be careful about spacing on writing f_vector
24 end
25 function J_matrix = J(x)
26 J_matrix = [3*x(1)^2-3*x(2)^2-3 -6*x(1)*x(2);...
27 6*x(1)*x(2) 3*x(1)^2-3*x(2)^2-3];
28 end

```

```

Initial approximation is x=1.000000, y=0.500000
At itr=1, x=4.981961, y=15.941176
At itr=2, x=3.331878, y=10.024148
At itr=3, x=2.302659, y=7.104194
At itr=4, x=1.841839, y=4.832317
At itr=5, x=1.750147, y=3.511395
At itr=6, x=1.926186, y=3.021962
At itr=7, x=1.999883, y=2.998481
At itr=8, x=2.000000, y=3.000001
L_2 error is <0.02614e+07

```

Even though I can change this value and I can take this initial approximation. So, we cannot give the value 0 here. So maybe I should give it 0.5 and let us see. Now it is okay. And after taking this value, my iteration is after 8 iteration I am getting this value. So, by this way I can solve different different equations.

Or maybe this is for 2-dimensional we have taken. Similarly, we can define for 3-dimensional. So, in that case, my function here in this column vector will be another. So, this is the first vector first value, second value, the third value we can define. And the same way that Jacobian will be changed.

Because if I take three functions, then the size of these Jacobian is a 3 by 3 matrix. So in this way, we can find the roots of a system of equations. It may be 2-dimensional, 3-dimensional using the Newton system method. So, we just have to change the value of the function here. And by doing this program, you can find the root of the equation.

So, this is, we are able to find the root of the equation using the Newton-Raphson method for the system of equations. So that is the end of this unit. From the next lecture, we are going to start with another unit that is how to deal with the linear systems and eigenvalue problems. So, thanks for watching this lecture. Thanks very much.