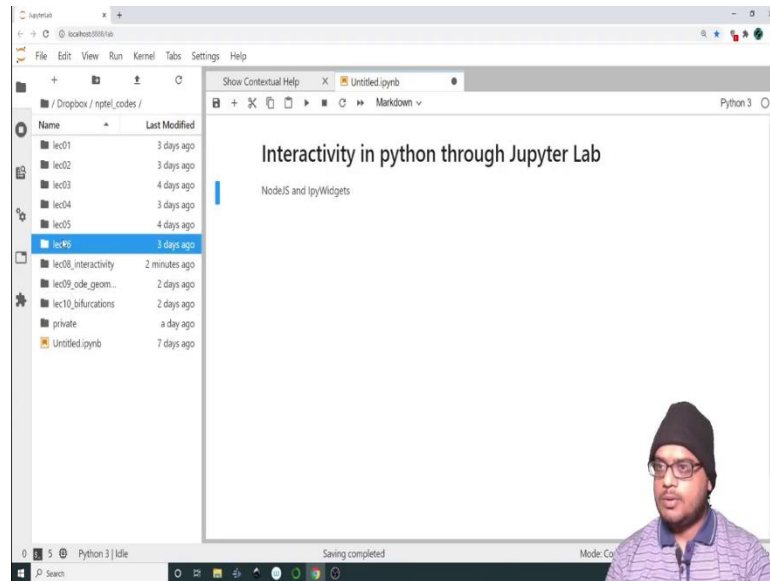


Tools in Scientific Computing
Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture – 08
Interactivity with Python – Ipywidgets

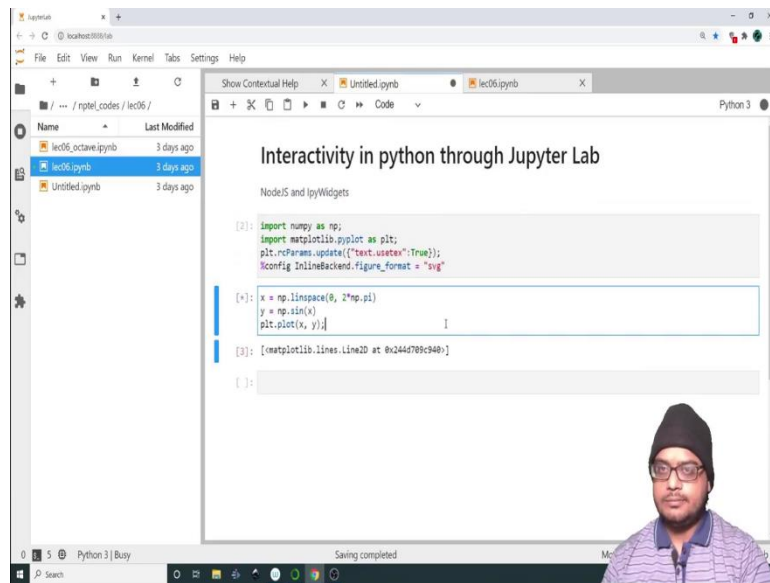
(Refer Slide Time: 00:27)



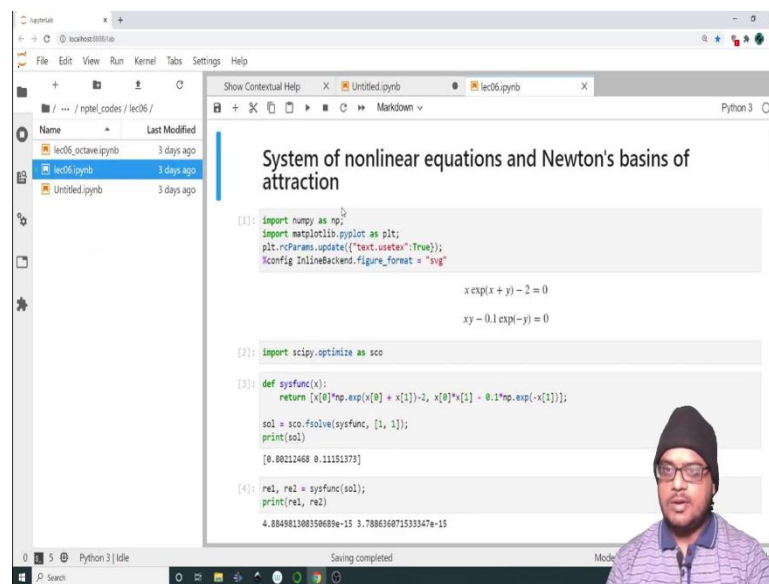
Hey guys it is me again. In this video we are going to look at some Interactivity in Python enabled through the IPython console in Jupyter Lab. So, in particular you need to install NodeJS and IpyWidgets and these are all cross platforms so, you can install them on any operating system that you are.

In fact, whatever we are doing in this particular course should be possible in all operating systems. So, once you have installed NodeJS and IpyWidgets you can make use of such things.

(Refer Slide Time: 01:00)

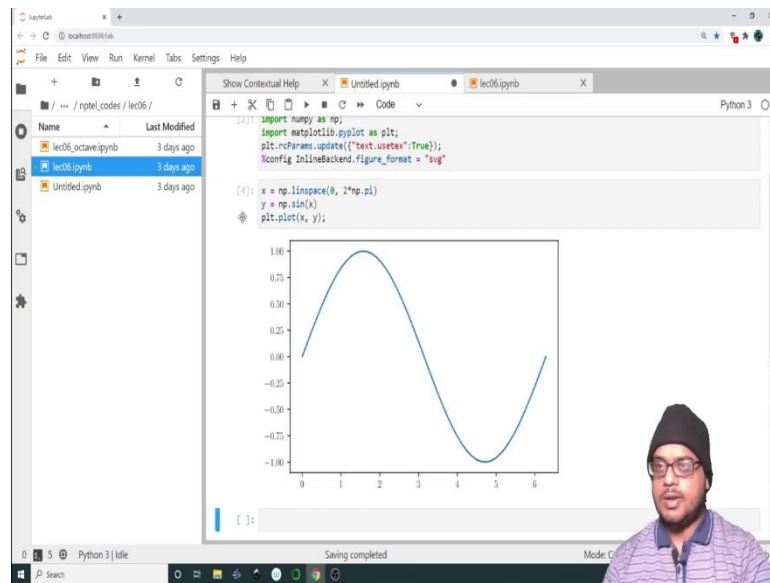


(Refer Slide Time: 01:03)



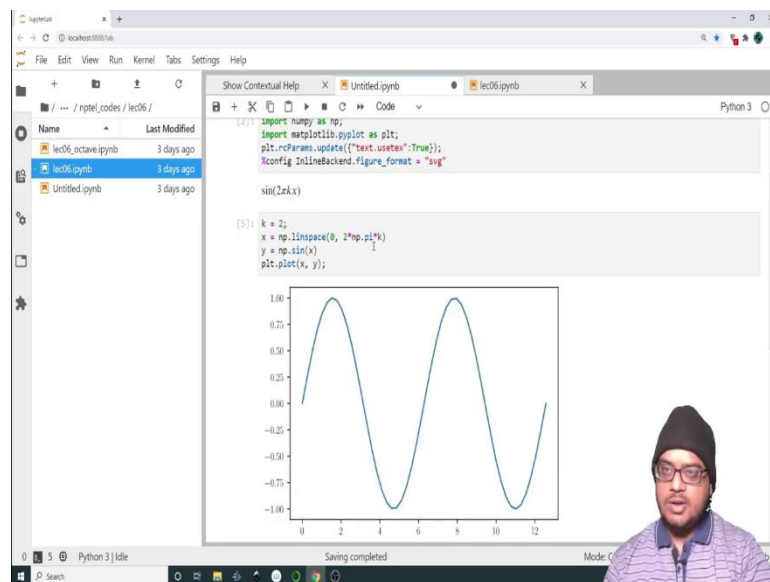
So, let me open whatever we had done in lecture 6 because we will need these things. So, first thing is first. Let me copy this because we are going to need all of these. So, we are going to import numpy and matplotlib. We are going to update the plot parameters and export format. So, once this is done we can declare. Suppose let us do a very simple example; x be np dot linspace 0 to 2 times pi and y be sin of x ok. So, let us do a plot of this, very simple ok.

(Refer Slide Time: 02:01)



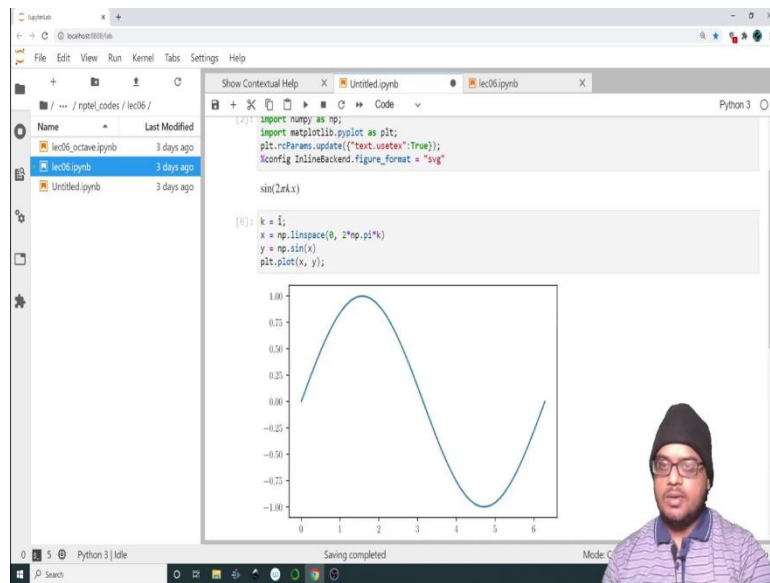
So, we have a plot like this. So, now, let me in fact, pass a wave number to this ok. So, wave number we can pass it like this.

(Refer Slide Time: 02:21)



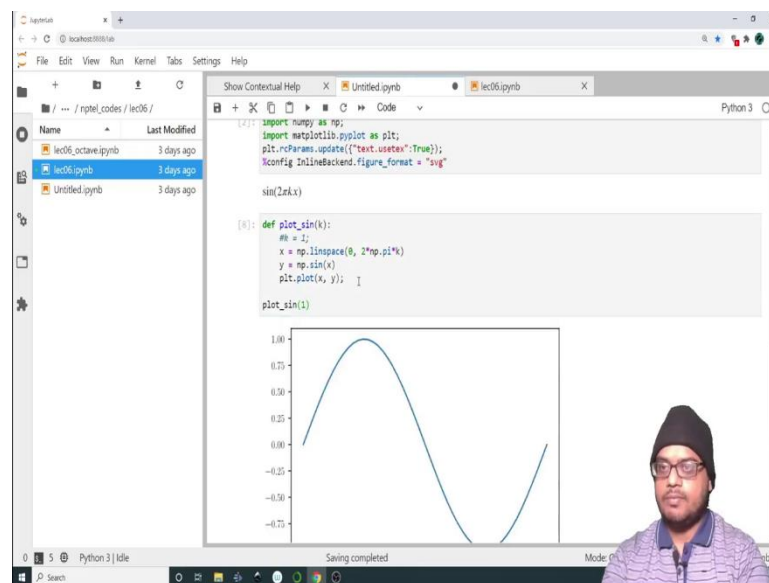
So, 2π times k and then I must define what k is. So, let k be equal to 2 ok. So, basically what we are doing is trying to plot \sin of $2\pi k x$, this is what we are trying to plot ok. So, because k is 2 we have 2 waves in the 2π domain.

(Refer Slide Time: 02:50)



If I make it 1 then I have only this. So, now, suppose I want to interact with this plot, suppose I want to change the value of k through a slider that would be very useful for us to see how the wave changes as k changes ok. So, in that case the first thing that we need to do is to convert whatever we have written over here which is the core of the code into a function. So, let me do that.

(Refer Slide Time: 03:19)

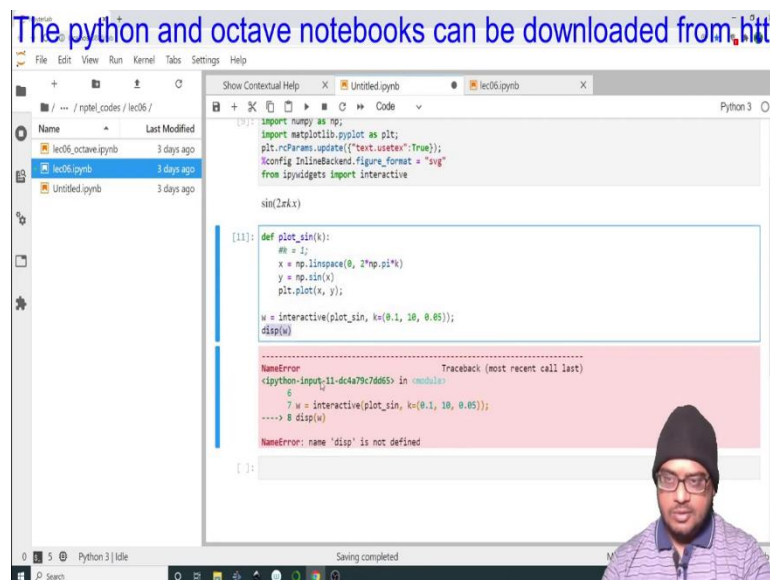


So, def plot sin and let it take only an input k . So, because we are taking an input k we can eliminate this particular line. Let me comment it at this point of time ok. So, I have

commented this line and because we are defining a function we need to indent whatever we want inside the function with respect to this ok. So, everything is indented whatever is inside this is therefore, inside this function. So, let us make a function call. Let me run this. So, it running this simply loads the function it does not run anything per say.

So, let me do this. So, plot sin 1. So, it passes this parameter one into the function and makes the plot because plotting is a part is executed inside the function. The function has no return value which is fine. It does not have to have a return value. Function can simply do something ok. So, now, once we have this we can do the following. Let us import one more thing.

(Refer Slide Time: 04:30)



The screenshot shows a Jupyter Notebook window with the following code in a cell:

```
import numpy as np;
import matplotlib.pyplot as plt;
plt.rcParams.update({'text.usetex':True});
%config InlineBackend.figure_format = 'svg'
from ipywidgets import interactive

sin(2*pi*x)

[11]: def plot_sin(k):
      # = 1;
      x = np.linspace(0, 2*np.pi*k)
      y = np.sin(x)
      plt.plot(x, y);

      w = interactive(plot_sin, k=(0.1, 10, 0.05));
      disp(w)
```

Below the code, a red error message is displayed:

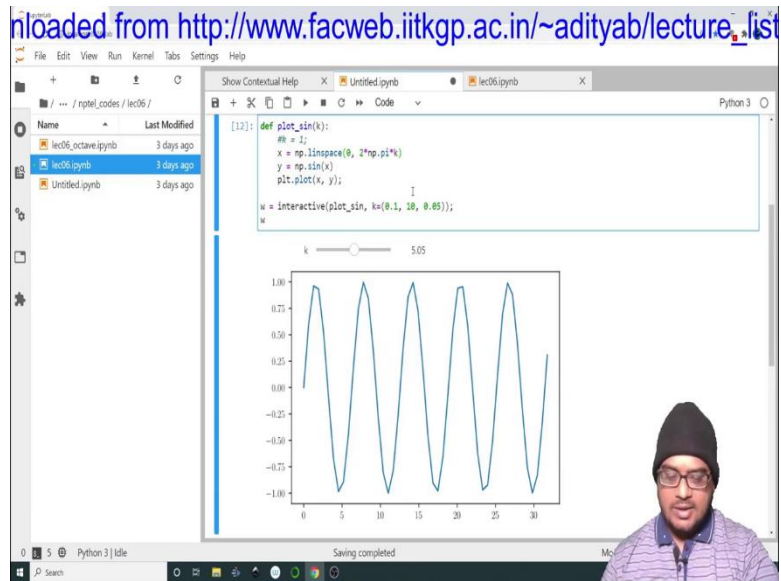
```
NameError                                Traceback (most recent call last)
<ipython-input-11-dc4a79c7d665> in <module>
      6
----> 8 disp(w)
NameError: name 'disp' is not defined
```

The notebook interface also shows a file explorer on the left with files like 'lec06_octave.ipynb', 'lec06.ipynb', and 'Untitled.ipynb'. A video feed of a person is visible in the bottom right corner.

So, from ipywidgets import interactive; so, interactive is the sub mode is the module inside ipywidgets which will help us to make the interactive plots. So, now that we have defined this let me define an object w and it will be interactive. So, now, I will pass the name of the function which I want to make interactive. So, it will be plot sin. I will just pass the function handle.

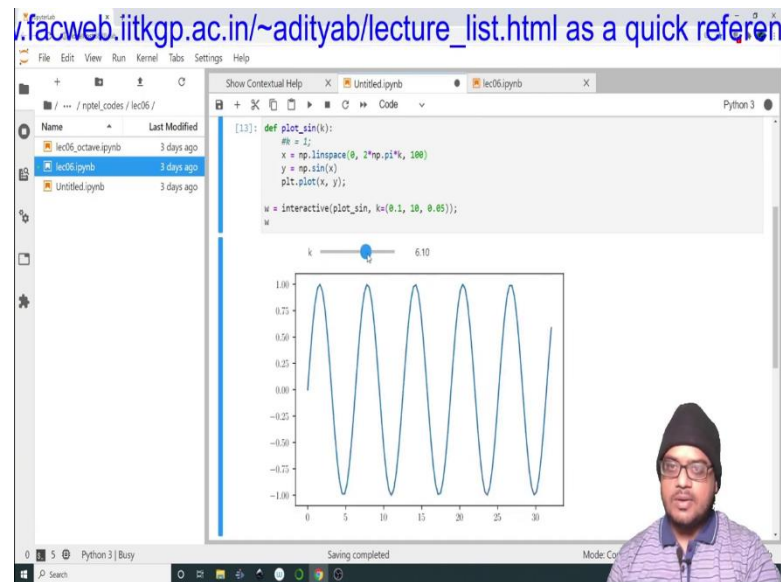
This is what is passing the function handle and I will tell that k is the parameter which should be interactive. So, after this I will tell that k can take a value from 0.1 to 10 and it should do so, in steps of 0.5. After this is done I must output w. So, if I just press w and enter it will sort of show what w is or I can also do this w; disp is not there.

(Refer Slide Time: 05:36)



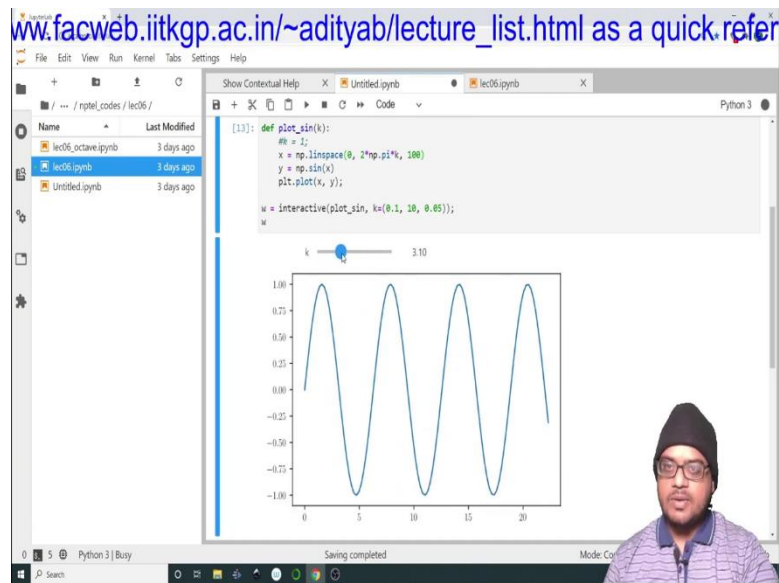
So, let me just show what w is ok. So, it makes this and we see a bunch of jagged lines because we have not yet increase the resolution of the x axis.

(Refer Slide Time: 05:50)



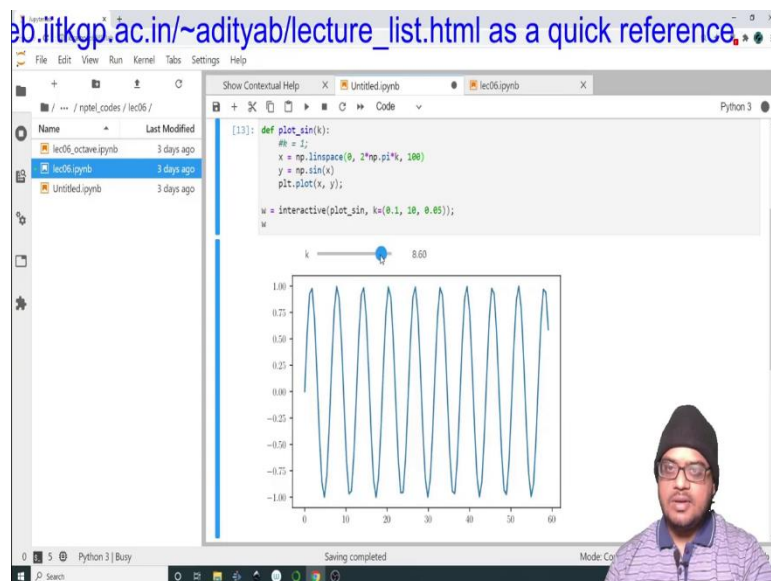
Let me make it one 100 that will make things much smoother.

(Refer Slide Time: 05:54)



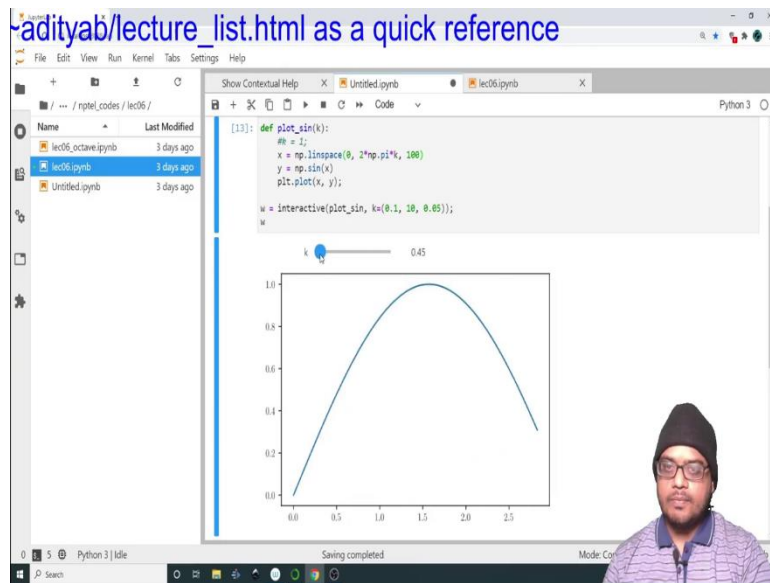
So, now, when I slide around this, I can see how the plot looks like ok.

(Refer Slide Time: 05:57)

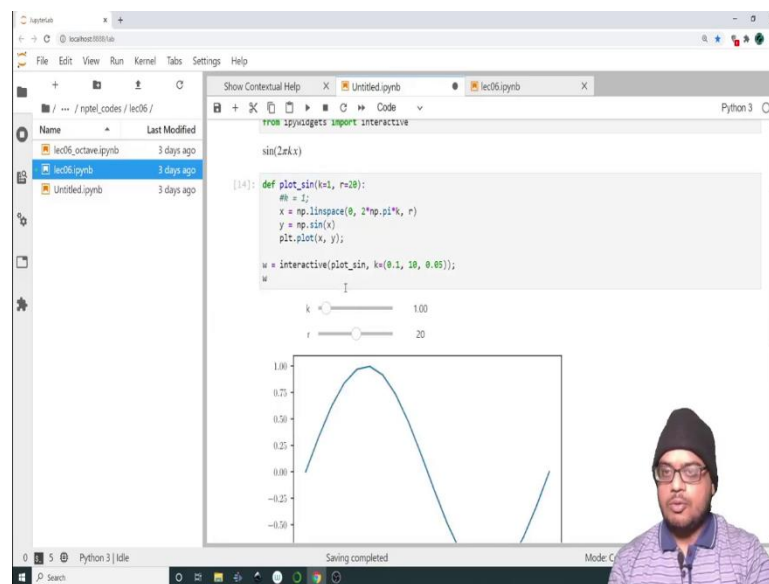


This is how I can study this particular plot in a very easy manner.

(Refer Slide Time: 06:02)



(Refer Slide Time: 06:17)

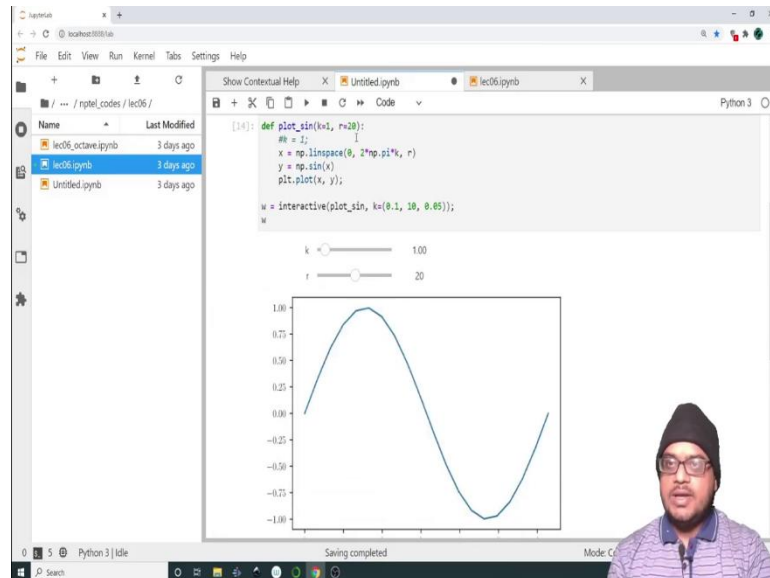


Now, suppose I want to pass this resolution also. So, let me call this r . I want to pass the resolution of the linspace. Let me create this function. So, now, we have a function which takes the wavelength or the wave number and the resolution of the line. Now, I have to now pass.

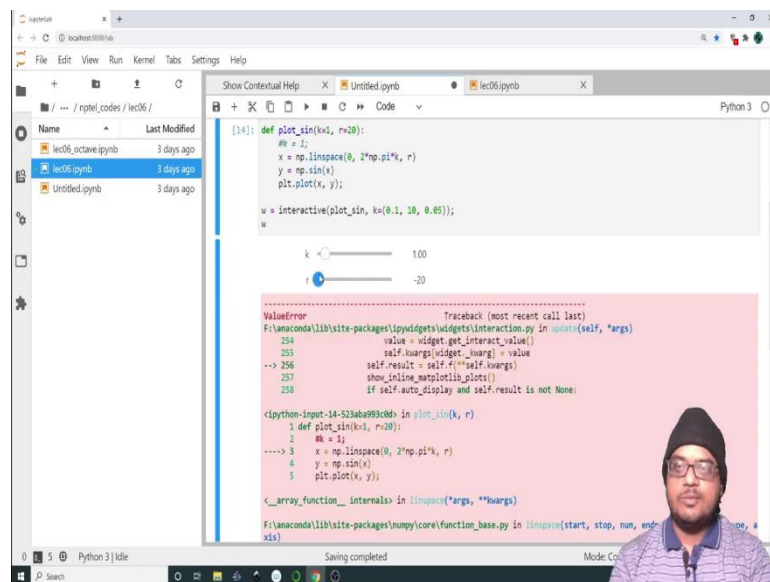
So, I can set some default parameters over here. I can set k equal to 1 and r equal to 2 for example, not 2, but 20. So, defining the function in this fashion even if you do not pass any parameters it will call the function with this default value.

So, these are the default values with which the function will be called. Now, what I can do is. So, I can run this straight away, no problem.

(Refer Slide Time: 07:03)

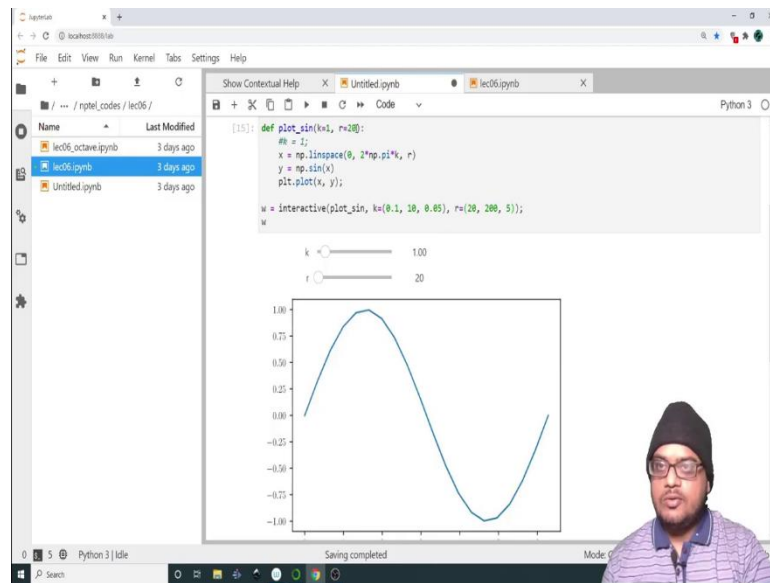


(Refer Slide Time: 07:08)



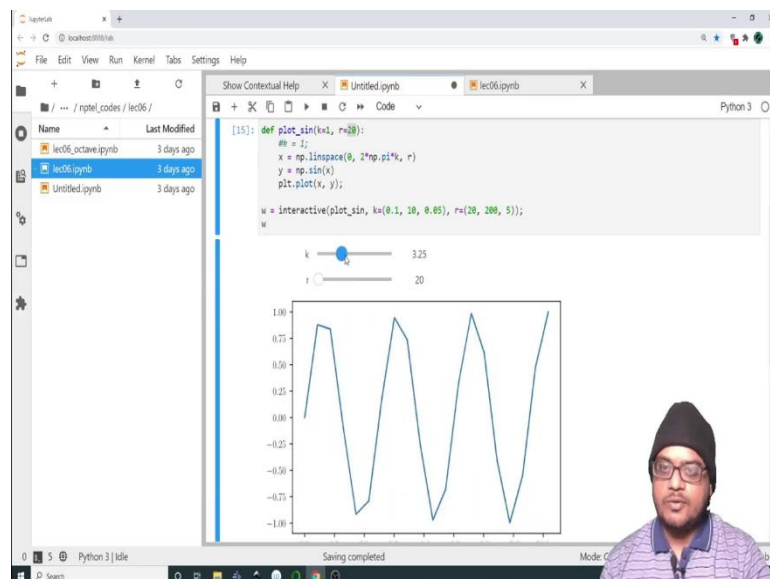
It will create for me an automatic slider for r in steps of 1 ok. It will create for me an automatic slider and I do not want that linspace should get something which is less than 2. So, now, see what happens when I put it to minus 20? Obviously, there is an error because a linspace cannot be created with minus 20 number of points.

(Refer Slide Time: 07:32)



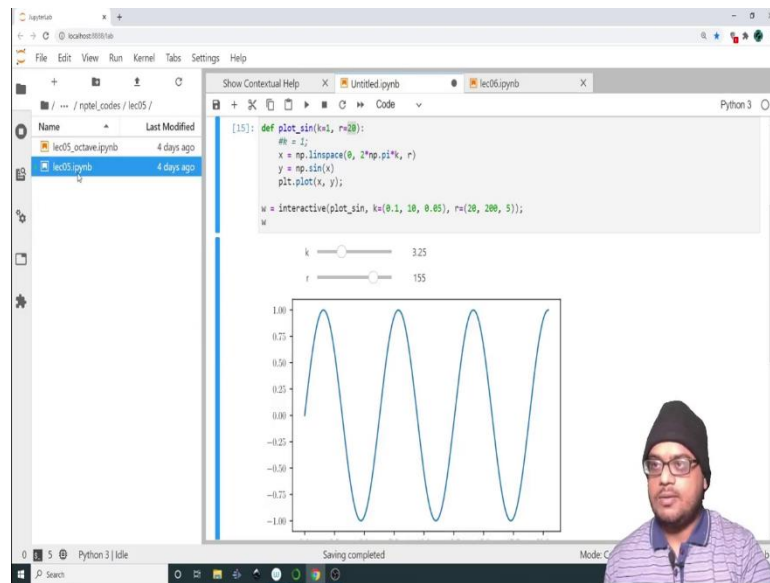
So, I must ensure that r is the slider for r is going to vary from 20 all the way to 200 and in steps of 5. Let me run this. So, by default the values are kept at 1 and 20, the slider values are kept at 1 and 20. Now, I can change the resolution. So, it becomes much smoother, the plot becomes much smoother. As r is reduced the plot becomes more jagged alright.

(Refer Slide Time: 08:03)



Let me increase k. So, now, the plot looks very jagged.

(Refer Slide Time: 08:07)



When I increase the resolution the plot smoothens out. So, this is the kind of behavior that we expect, everything looks fine ok. Let me in fact, open lecture 5 as well.

(Refer Slide Time: 08:19)

The screenshot shows a Jupyter Notebook interface displaying a slide titled "Fixed point iterations and cobwebs". The slide content includes:

Fixed point iterations and cobwebs

In this lecture, we will be looking at solving nonlinear equations. In particular, we will be looking at a very simple idea - fixed point iterations. While fixed point iterations have far reaching consequences for understanding various aspects of nonlinear dynamics as well, it lends an insight into the nature of convergence of roots and the dependency of the convergence behaviour depending on the functional choice.

```
[2]: import numpy as np;  
import matplotlib.pyplot as plt;  
plt.rcParams.update({"text.usetex": True});  
%config InlineBackend.figure_format = "svg"
```

Fixed point iterations in 1D

Let us consider the nonlinear equation

$$\exp(x) - 3x^2 = 0,$$

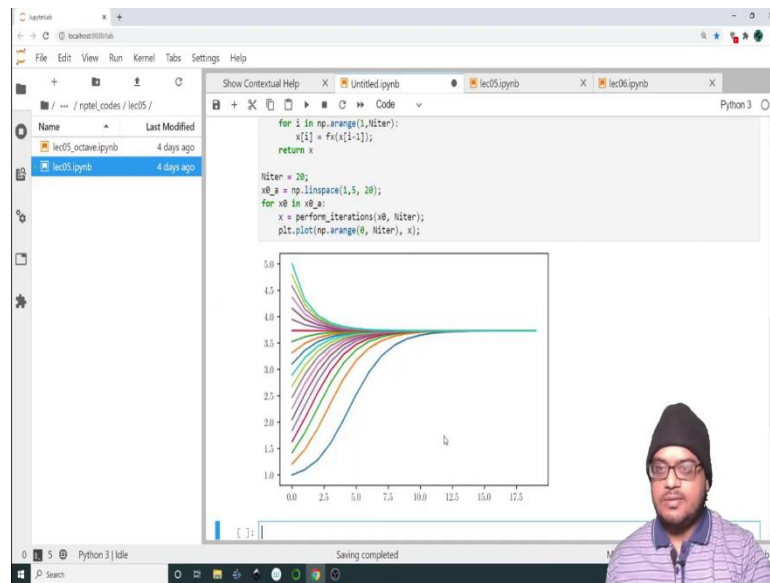
whose roots we are interested in. In order to apply fixed point iterations, we will first cast it in the form of $x = g(x)$. We can have multiple possibilities to do so. Let's consider the first possibility as shown below.

We rearrange the equation to write

$$x = \ln 3 + 2 \ln x.$$

In case we had the root, the left hand side and right hand side of the equations would match exactly. However, if we have a value different from the root, then the left hand side and right hand side would not match. In that case we can use the right hand side as a new value for x and repeat the process.

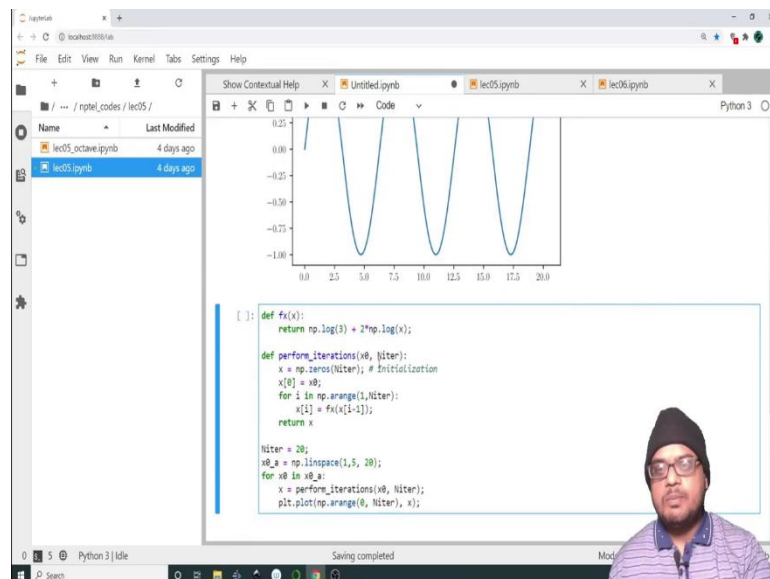
(Refer Slide Time: 08:26)



We had done a bunch of fixed point iterations and we had made this plot where different initial conditions we were trying to see that depending on what initial guess we have, how does the curve converge to the fixed point. So, this is the solution 3.7 something, if I remember correctly.

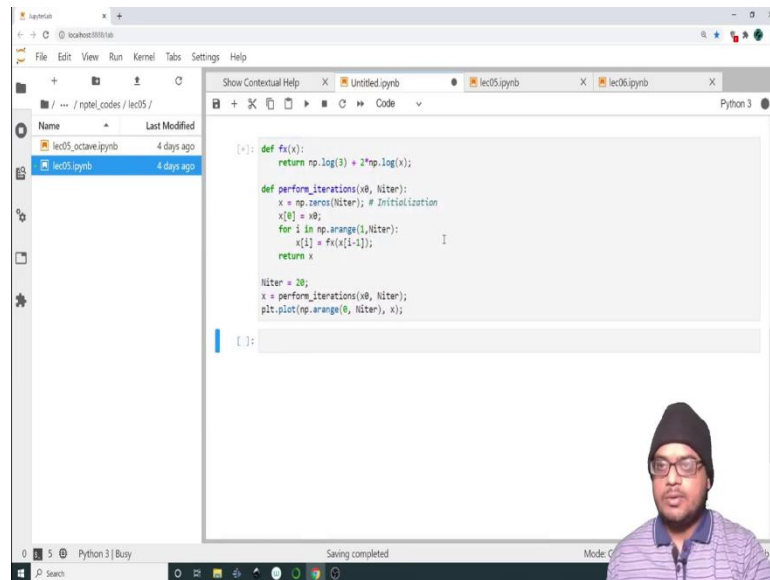
So, now, I am more interested in. So, over here we had swept over a bunch of initial values, but suppose you do not want to do that. You do not want to sweep over a bunch of initial values, you want to select a new guess point and then see how it goes.

(Refer Slide Time: 09:09)



So, let us do this. Let us take this entire snippet. Let us take it to this particular file. Let us try to make that particular plot interactive. So, first things first we need to remove this loop. See when I run this it will show a bunch of values because we ran the guess values in a loop.

(Refer Slide Time: 09:24)



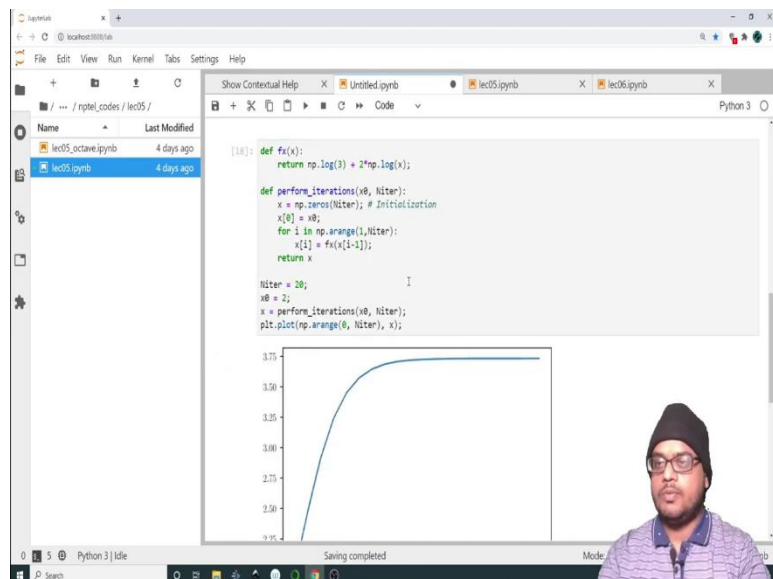
```
[*]: def fx(x):
      return np.log(3) + 2*np.log(x);

def perform_iterations(x0, Niter):
    x = np.zeros(Niter); # Initialization
    x[0] = x0;
    for i in np.arange(1,Niter):
        x[i] = fx(x[i-1]);
    return x

Niter = 20;
x = perform_iterations(x0, Niter);
plt.plot(np.arange(0, Niter), x);
```

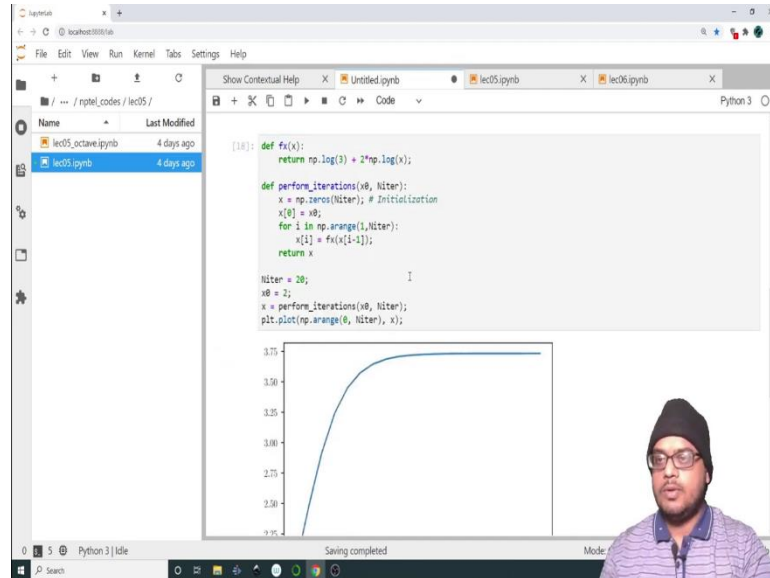
So, we do not need to run. We do not need a loop counter for anything. We just need this. So, this bit of code should be sufficient ok. So, x naught is like the guess, perform iterations is the function over here and that function calls f x which is over here.

(Refer Slide Time: 09:53)



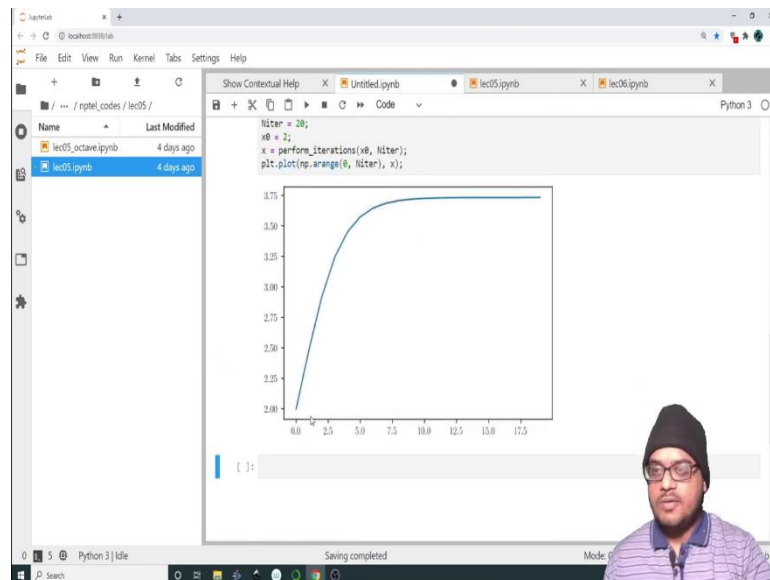
So, all of this code is self contained, but obviously, I have not ok x naught was there from the previous value.

(Refer Slide Time: 09:59)



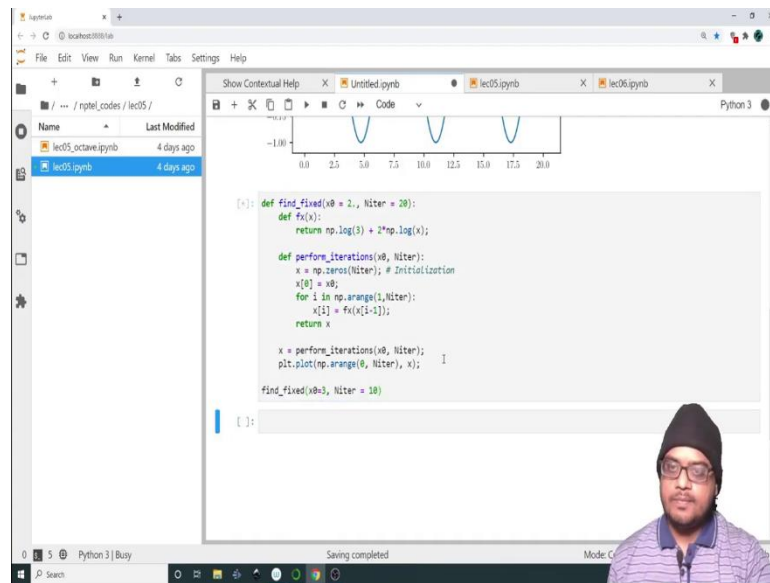
So, let x naught be equal to 2 ok.

(Refer Slide Time: 10:03)



So, if x naught starts at 2, it goes over here no problem alright. So, let us now try to wrap all of these into a single function because we want to interact with this. So, let me remove Niter and x naught ok. So, we will put all of this inside a function.

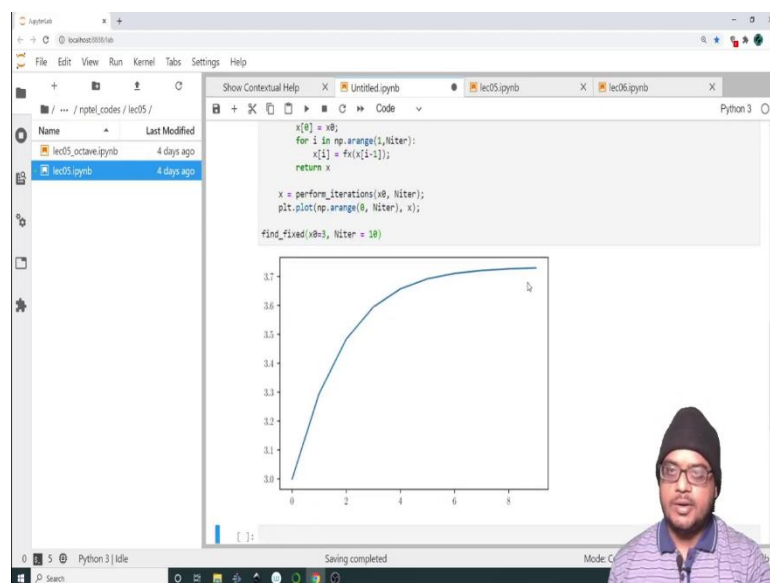
(Refer Slide Time: 10:26)



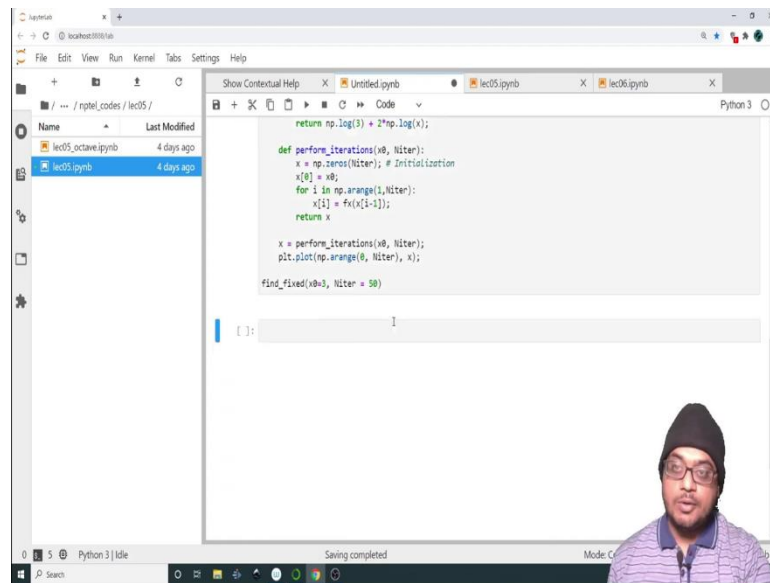
So, def find fixed and the inputs will be x naught equal to say 2 and Niter will be equal to say 20. So, now all of these is inside a function and the default values if you do not pass these things it will be 2 and 20 alright. So, with this in mind let us now; let us now just call the function and see what happens.

So, find fixed and let us say x naught equal to 3 and Niter equal to 10. So, this is what you get. Let me do it for x naught equal to 50, Niter equal to 50.

(Refer Slide Time: 11:10)



(Refer Slide Time: 11:14)



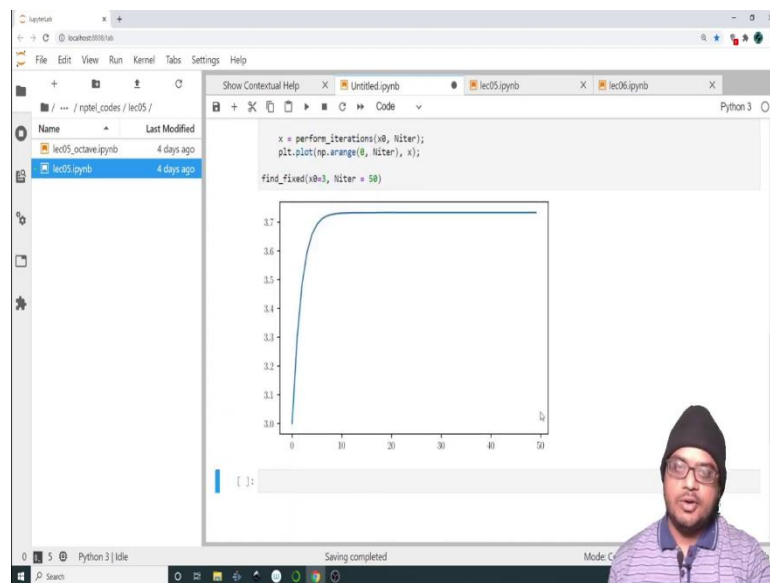
```
return np.log(3) + 2*np.log(x);

def perform_iterations(x0, Niter):
    x = np.zeros(Niter); # Initialization
    x[0] = x0;
    for i in np.arange(1, Niter):
        x[i] = fx(x[i-1]);
    return x

x = perform_iterations(x0, Niter);
plt.plot(np.arange(0, Niter), x);

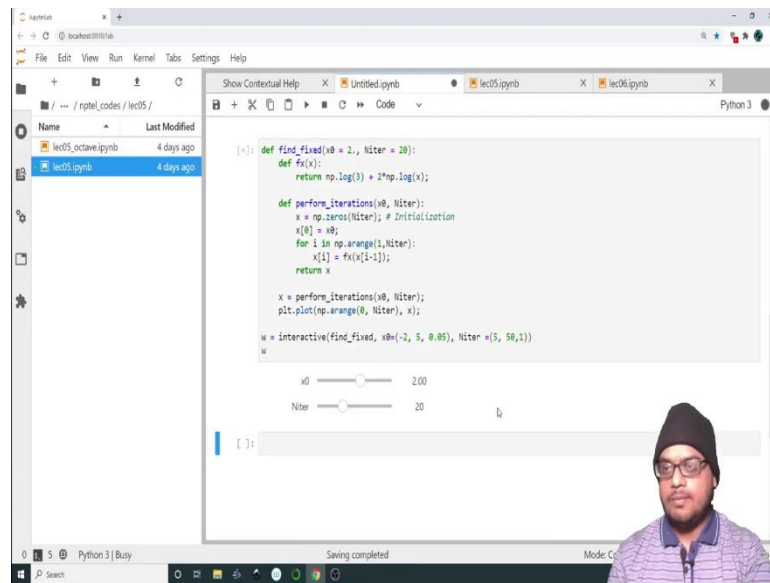
find_fixed(x0=3, Niter = 50)
```

(Refer Slide Time: 11:16)



So, this is what you get. You get 50 iterations ok. So, basically this function works and now we want to interact with it.

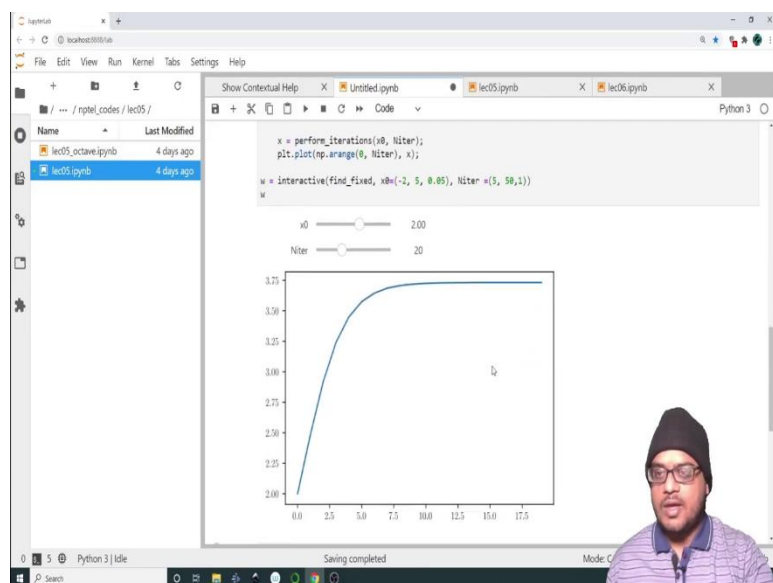
(Refer Slide Time: 11:24)



So, we will declare an object w and it will be interactive. We will pass the function handle and we will say that x naught can vary from; x naught can vary from minus 2 all the way to 5 in steps of 0.05 and Niter can vary from 5 all the way to 50 in steps of 1 that is fine.

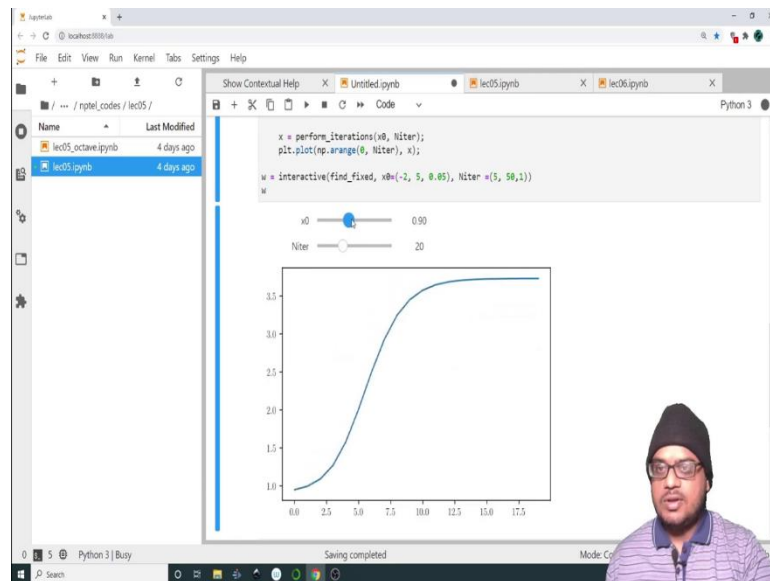
I mean this one is redundant. If you do not pass the step it will counted as 1, but for completeness let me pass 1. Lastly we have to display that object. So, w is actually like a widget which runs inside your browser. So, then you have to display that widget as well.

(Refer Slide Time: 12:13)

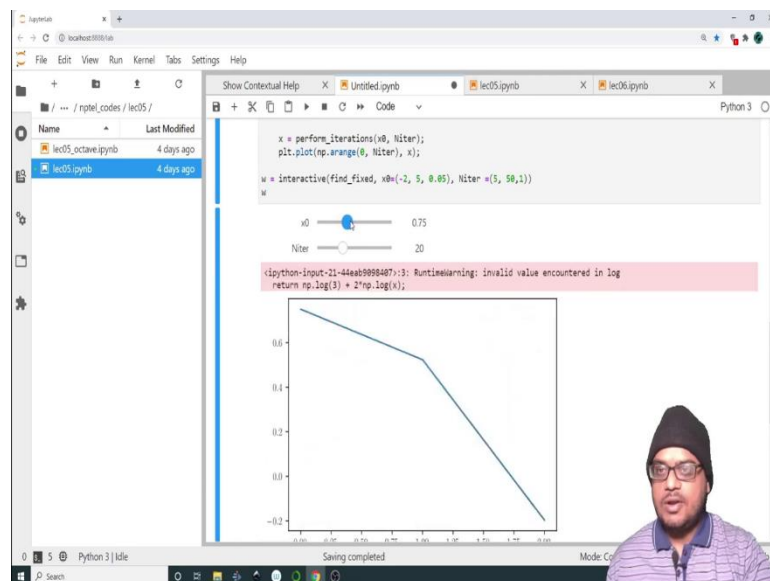


So, when I run this is what we have.

(Refer Slide Time: 12:17)

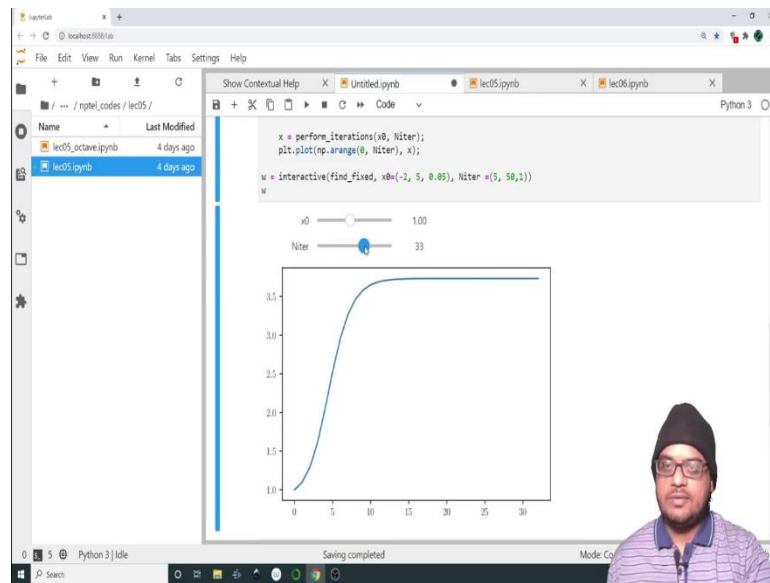


(Refer Slide Time: 12:18)

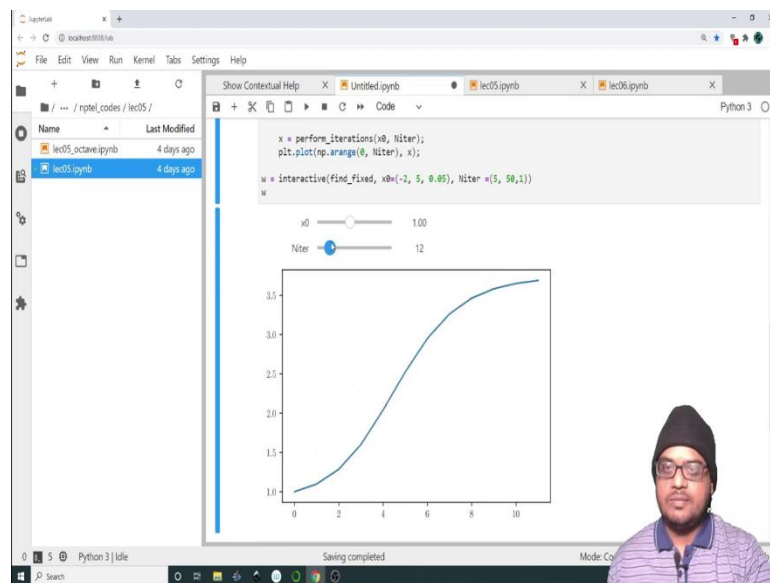


So, when I change the initial value we get a runtime error because the iterations would not converge if you have any guess value ok.

(Refer Slide Time: 12:29)

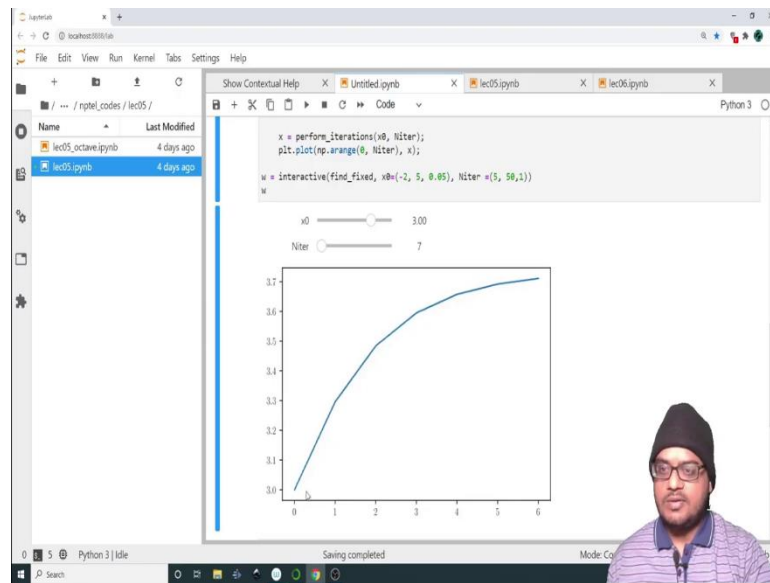


(Refer Slide Time: 12:32)



So, you can see how the shape changes when you change the number of iterations what happens ok. So, sometimes if you choose very odd values it breaks everything. So, you need to rerun it.

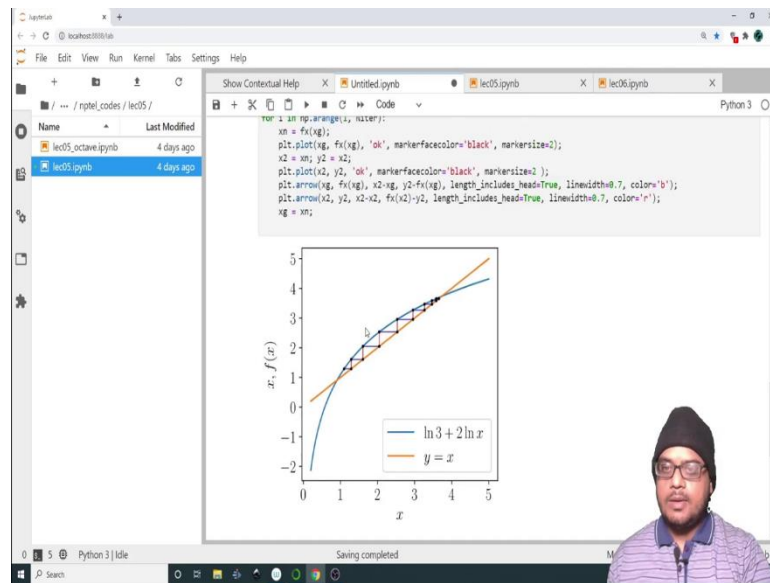
(Refer Slide Time: 13:01)



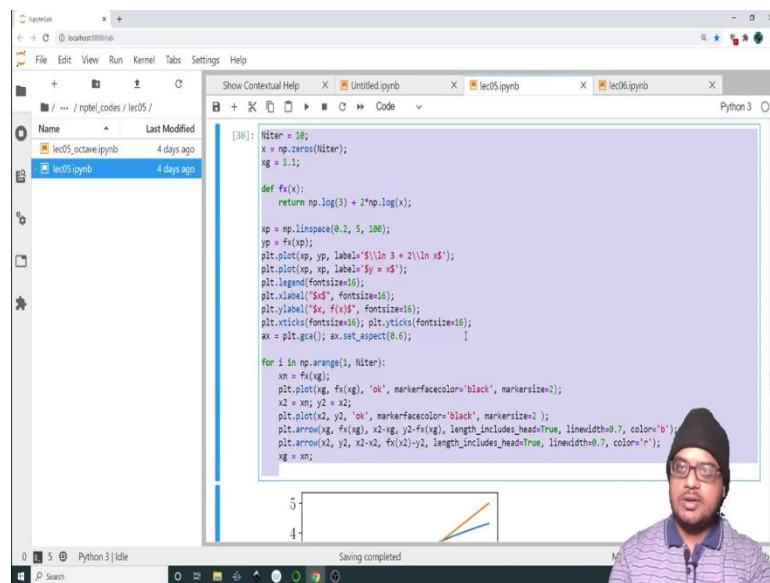
So, very few number of iterations it has not yet converged ok. When you change the initial guess this is how it looks ok. It starts at 3 and take 7 iterations starting from 0 of course. When you increase the number of iterations this is what you get ok. So, this is a very nice way of quickly discerning what is going on in your problem.

If you want to analyze a bunch of parameter space of course, you can avoid doing all these, you can make functions, you can loop over all the initial conditions, but this gives you a very raw feel of what is going on. It gives you can play around with the values ok. Let us go back to our old file and let us see whether we can wrap something else something interesting ok.

(Refer Slide Time: 14:00)

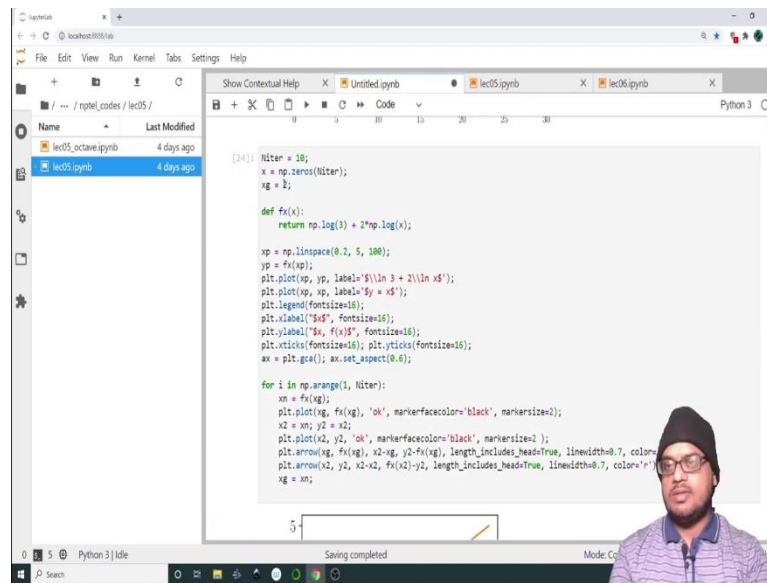


(Refer Slide Time: 14:08)



So, this was the case where the iterations were converging at this point. Let us take this bit of code and let us try to wrap it around another function. So, let me run this. Let us see whether it. So, we have this plot no problem. Now, let us see how the initial condition converges to that particular solution ok. So, number of iterations are fixed x guess. So, this is the parameter that we want to vary and see.

(Refer Slide Time: 14:36)



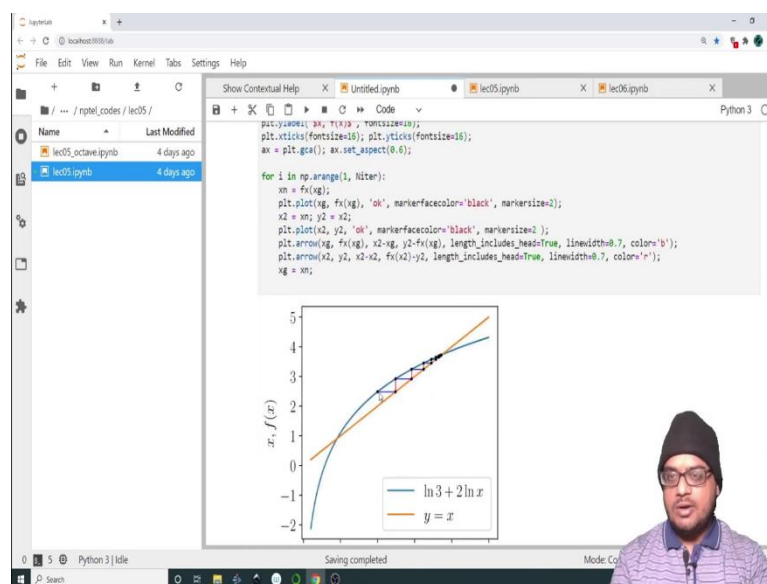
```
[24]: Niter = 10;
x = np.zeros(Niter);
xg = 1;

def fx(x):
    return np.log(3) + 2*np.log(x);

xp = np.linspace(0.2, 5, 100);
yp = fx(xp);
plt.plot(xp, yp, label='ln 3 + 2 ln x');
plt.plot(xp, xp, label='y = x');
plt.legend(fontsize=16);
plt.xlabel('x', fontsize=16);
plt.ylabel('f(x)', fontsize=16);
plt.xticks(fontsize=16); plt.yticks(fontsize=16);
ax = plt.gca(); ax.set_aspect(0.6);

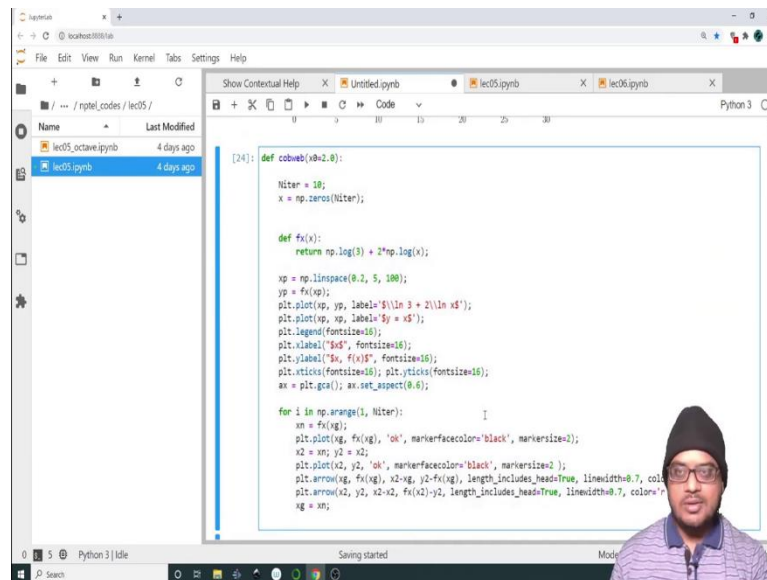
for i in np.arange(1, Niter):
    xm = fx(xg);
    plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize=2);
    x2 = xm; y2 = x2;
    plt.plot(x2, y2, 'ok', markerfacecolor='black', markersize=2);
    plt.arrow(xg, fx(xg), x2-xg, y2-fx(xg), length_includes_head=True, linewidth=0.7, color='r');
    plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True, linewidth=0.7, color='r');
    xg = xm;
```

(Refer Slide Time: 14:38)



So, for example, I can make it 2, let me run this. So, when you start at 2 this is how it converges, but I want to play around with the initial value ok. I want to play around with the initial value. So, let us do it. So, let us remove this particular line and we will wrap everything inside a function.

(Refer Slide Time: 14:54)

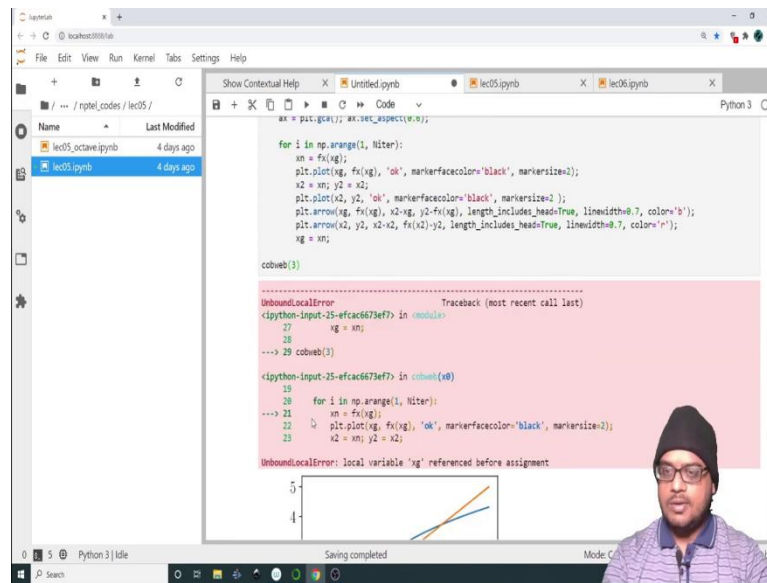


```
[24]: def cobweb(x0=0.2):  
    Niter = 10;  
    x = np.zeros(Niter);  
  
    def f(x):  
        return np.log(3) + 2*np.log(x);  
  
    xp = np.linspace(0.2, 5, 100);  
    yp = f(xp);  
    plt.plot(xp, yp, label='ln 3 + 2*ln x^2');  
    plt.plot(xp, xp, label='y = x^2');  
    plt.legend(fontsize=16);  
    plt.xlabel('x^2', fontsize=16);  
    plt.ylabel('f(x)', fontsize=16);  
    plt.xticks(fontsize=16); plt.yticks(fontsize=16);  
    ax = plt.gca(); ax.set_aspect(0.6);  
  
    for i in np.arange(1, Niter):  
        xn = f(xg);  
        plt.plot(xg, f(xg), 'ok', markerfacecolor='black', markersize=2);  
        x2 = xn; y2 = x2;  
        plt.plot(x2, y2, 'ok', markerfacecolor='black', markersize=2);  
        plt.arrow(xg, f(xg), x2-xg, y2-f(xg), length_includes_head=True, linewidth=0.7, color='r');  
        xg = xn;
```

So, we will call it define. So, let us call it cobweb and the input will be x naught and let us say the input default is 0.2. So, because we are defining a function whatever goes inside the function has to be indented; it is very important in Python.

In octave and all you just need to I mean first of all iterative interactive plots are not possible in octave as far as I know, but commercial software such as Mathematica, Maple they do allow you to do such kinds of interactive things and that is the whole charm of it. You can do you can play around with various values ok. So, we have done this is the code ok. Let us quickly see whether the code runs or not whether we have broken something.

(Refer Slide Time: 15:49)



```
ax = plt.gca(); ax.set_aspect('square');

for i in np.arange(1, Niter):
    xn = fx(xg);
    plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize=2);
    x2 = xn; y2 = x2;
    plt.plot(x2, y2, 'ok', markerfacecolor='black', markersize=2);
    plt.arrow(xg, fx(xg), x2-xg, y2-fx(xg), length_includes_head=True, linewidth=0.7, color='b');
    plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True, linewidth=0.7, color='r');
    xg = xn;

cobweb(3)

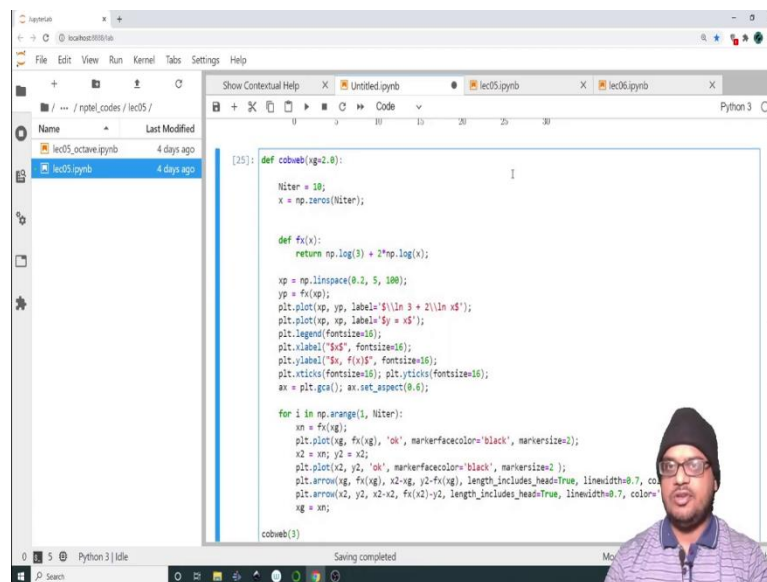
UnboundLocalError                                Traceback (most recent call last)
<ipython-input-25-efcac673ef7> in <module>
     27     xg = xn;
     28
--> 29 cobweb(3)

<ipython-input-25-efcac673ef7> in cobweb(x0)
     29
--> 21     for i in np.arange(1, Niter):
     22         xn = fx(xg);
     23         plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize=2);
         x2 = xn; y2 = x2;

UnboundLocalError: local variable 'xg' referenced before assignment
```

So, cobweb let me call cobweb with 3 as the initial guess, oh xj unreferenced ok. Let us see what we have broken ok.

(Refer Slide Time: 16:06)



```
[25]: def cobweb(x0):
      Niter = 10;
      x = np.zeros(Niter);

      def fx(x):
          return np.log(3) + 2*np.log(x);

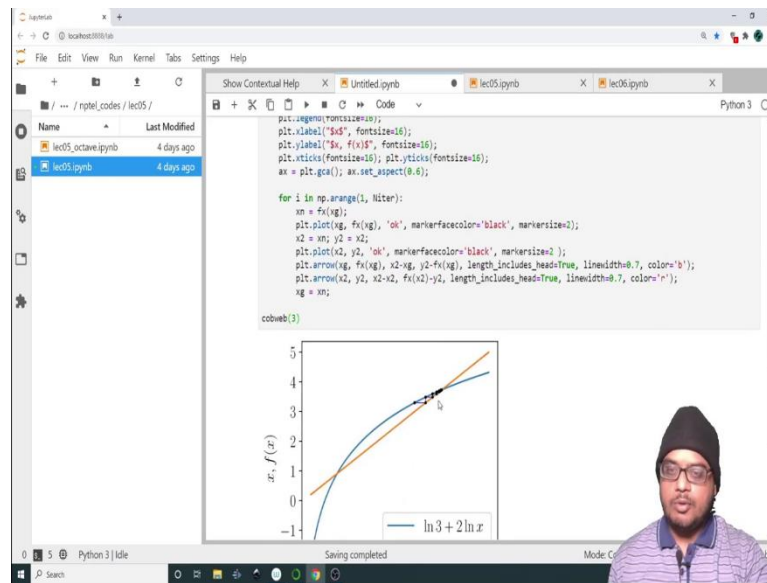
      xp = np.linspace(0.2, 5, 100);
      yp = fx(xp);
      plt.plot(xp, yp, label='$\ln 3 + 2\ln x$');
      plt.plot(xp, xp, label='$y = x$');
      plt.legend(fontsize=16);
      plt.xlabel('$x$', fontsize=16);
      plt.ylabel('$f(x)$', fontsize=16);
      plt.xticks(fontsize=16); plt.yticks(fontsize=16);
      ax = plt.gca(); ax.set_aspect(0.6);

      for i in np.arange(1, Niter):
          xn = fx(xg);
          plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize=2);
          x2 = xn; y2 = x2;
          plt.plot(x2, y2, 'ok', markerfacecolor='black', markersize=2);
          plt.arrow(xg, fx(xg), x2-xg, y2-fx(xg), length_includes_head=True, linewidth=0.7, color='b');
          plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True, linewidth=0.7, color='r');
          xg = xn;

      cobweb(3)
```

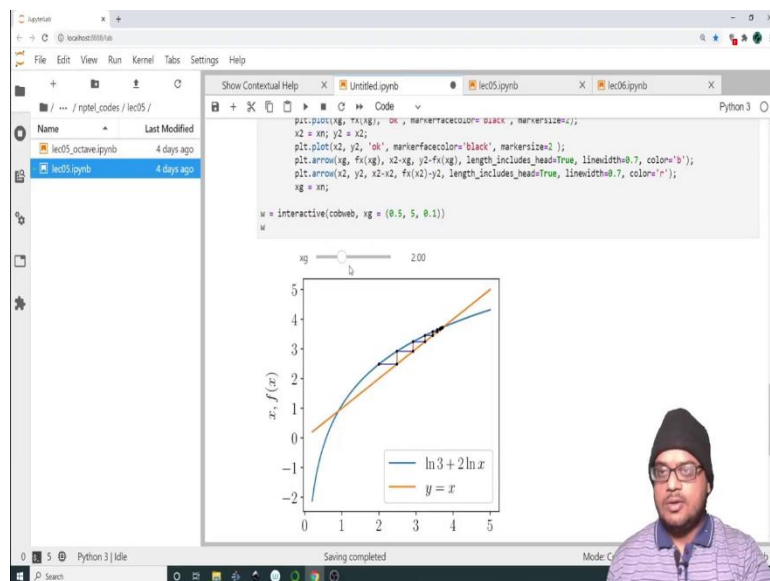
So, x naught would be I think it was xg, let us see. Yeah it was xg. So, we need to remember whatever you pass is sort of the xg that you are using over here ok. It is that same variable name that you are using you cannot define x naught over here and xg over here. So, xj is this. So, fine.

(Refer Slide Time: 16:27)



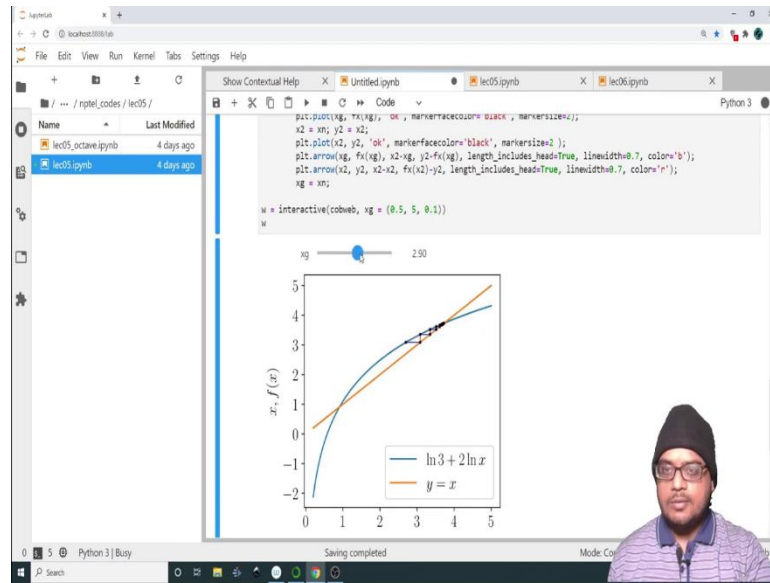
Let me now run this and we do have a plot which shows this convergence fine.

(Refer Slide Time: 16:37)

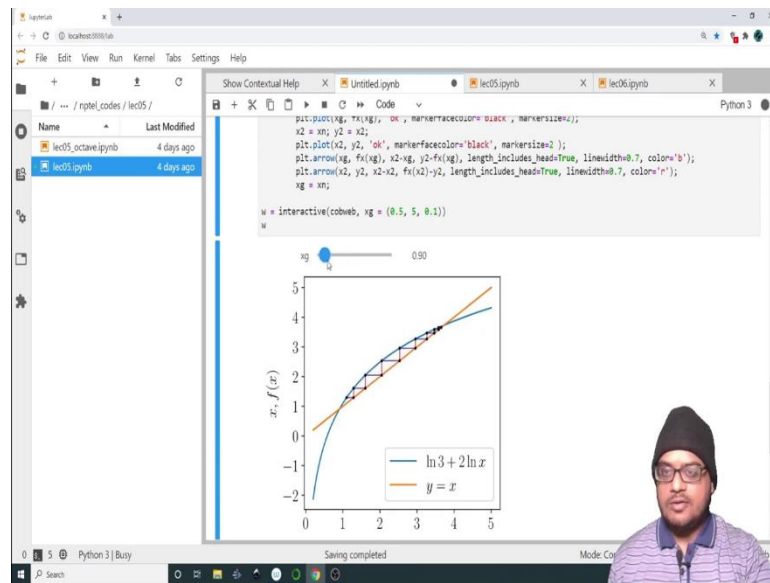


So, now let me pass this to an interactive function. So, we equal to `interactive` then `cobweb` `xg` equal to let us choose a range from say 0.5 all the way to 5 and say the step is 0.1. Let me run this. We need to display `w` as well just ok.

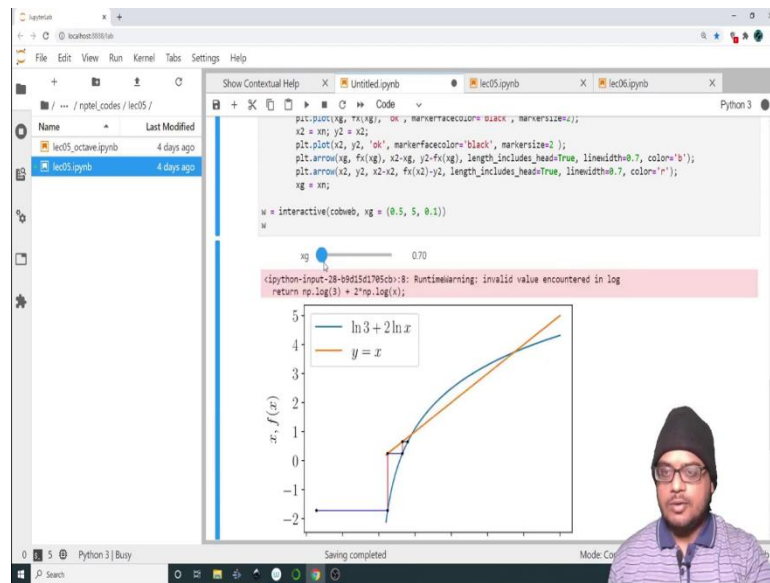
(Refer Slide Time: 17:09)



(Refer Slide Time: 17:11)

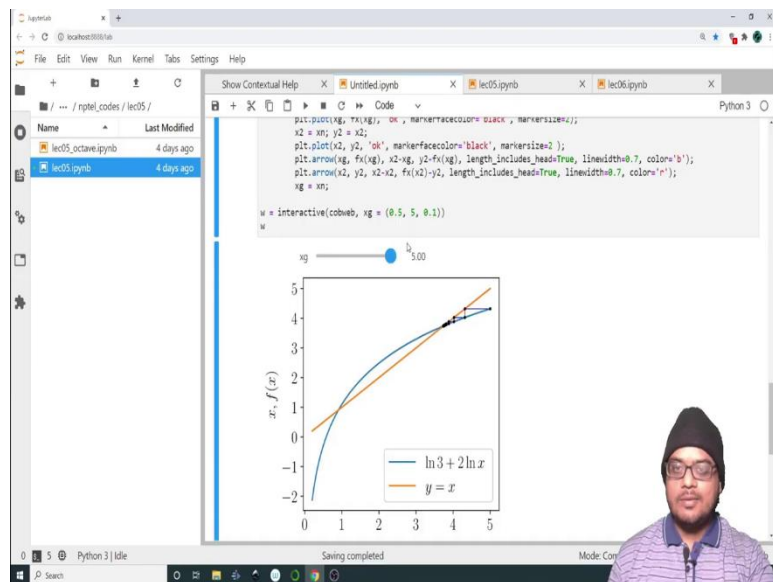


(Refer Slide Time: 17:12)



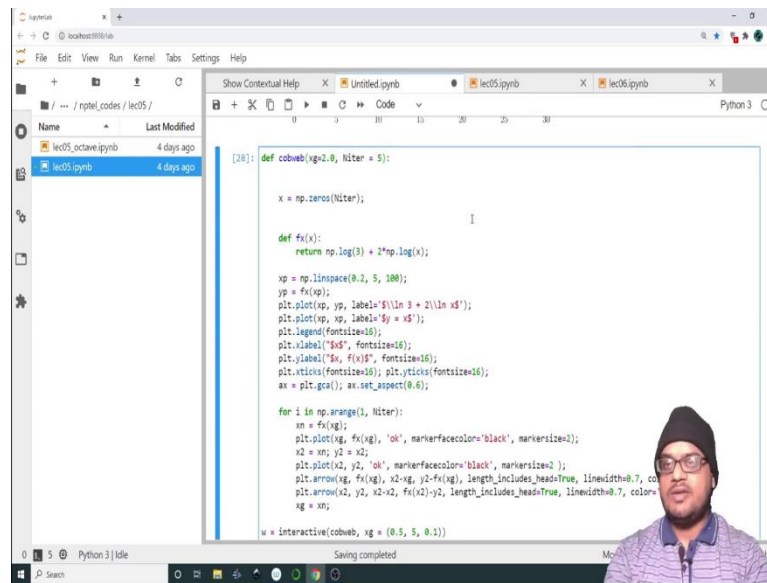
So, see how we can see the convergence of the root ok. So, for guess values on this side there is obviously, something bad that happens ok, it flies off to the other side. So, that shows you that you need to guess between those two points you cannot guess beyond that point ok, excellent.

(Refer Slide Time: 17:30)



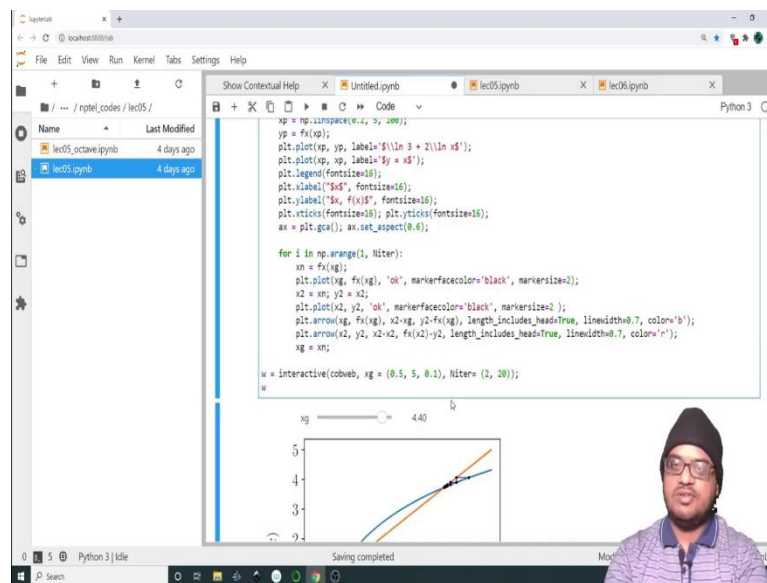
Let me also change the number of iterations. So, Niter let me remove it and call it inside the function itself.

(Refer Slide Time: 17:39)



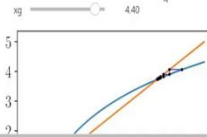
```
[28]: def cobweb(xg=2.0, Niter = 5):  
  
    x = np.zeros(Niter);  
  
    def fx(x):  
        return np.log(3) + 2*np.log(x);  
  
    xp = np.linspace(0.2, 5, 100);  
    yp = fx(xp);  
    plt.plot(xp, yp, label='f(ln 3 + 2*ln x^2)');  
    plt.plot(xp, xp, label='y = x^2');  
    plt.legend(fontsize=16);  
    plt.xlabel("$x$", fontsize=16);  
    plt.ylabel("$y = f(x)$", fontsize=16);  
    plt.xticks(fontsize=16); plt.yticks(fontsize=16);  
    ax = plt.gca(); ax.set_aspect(0.6);  
  
    for i in np.arange(1, Niter):  
        xm = fx(xg);  
        plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize=2);  
        x2 = xm; y2 = x2;  
        plt.plot(x2, y2, 'ok', markerfacecolor='black', markersize=2);  
        plt.arrow(xg, fx(xg), x2-xg, y2-fx(xg), length_includes_head=True, linewidth=0.7, color='b');  
        plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True, linewidth=0.7, color='r');  
        xg = xm;  
  
    w = interactive(cobweb, xg = (0.5, 5, 0.1))  
    w
```

(Refer Slide Time: 17:45)



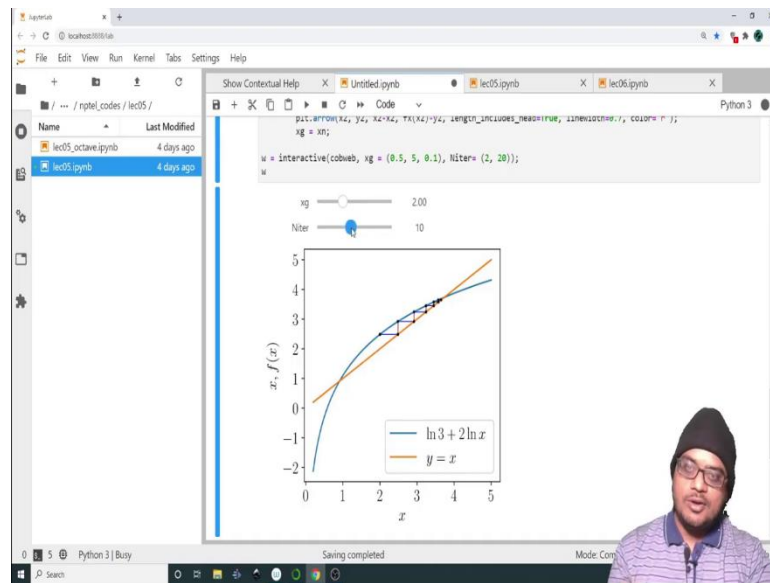
```
xp = np.linspace(xg, 5, 100);  
yp = fx(xp);  
plt.plot(xp, yp, label='f(ln 3 + 2*ln x^2)');  
plt.plot(xp, xp, label='y = x^2');  
plt.legend(fontsize=16);  
plt.xlabel("$x$", fontsize=16);  
plt.ylabel("$y = f(x)$", fontsize=16);  
plt.xticks(fontsize=16); plt.yticks(fontsize=16);  
ax = plt.gca(); ax.set_aspect(0.6);  
  
for i in np.arange(1, Niter):  
    xm = fx(xg);  
    plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize=2);  
    x2 = xm; y2 = x2;  
    plt.plot(x2, y2, 'ok', markerfacecolor='black', markersize=2);  
    plt.arrow(xg, fx(xg), x2-xg, y2-fx(xg), length_includes_head=True, linewidth=0.7, color='b');  
    plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True, linewidth=0.7, color='r');  
    xg = xm;  
  
w = interactive(cobweb, xg = (0.5, 5, 0.1), Niter = (2, 20));  
w
```

xg = 4.40

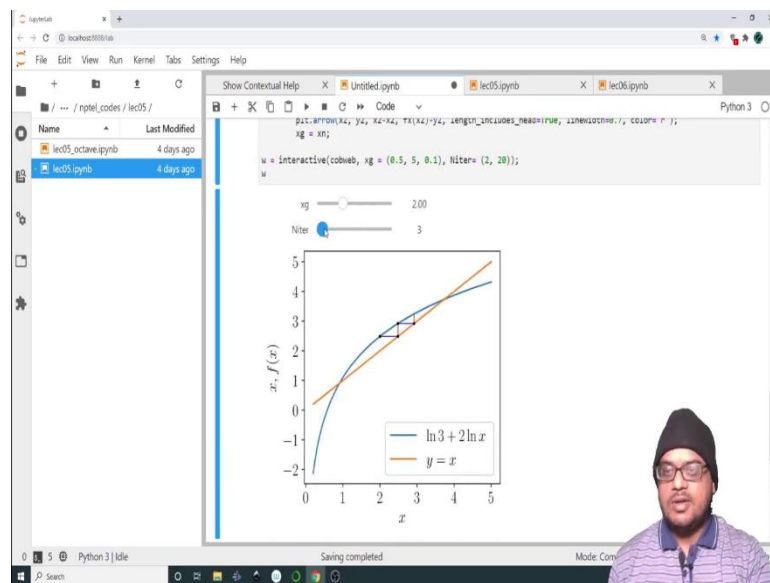


Let me have by default 5 iterations. This will say go from 2 to 20 ok in steps of 1, alright.

(Refer Slide Time: 17:54)

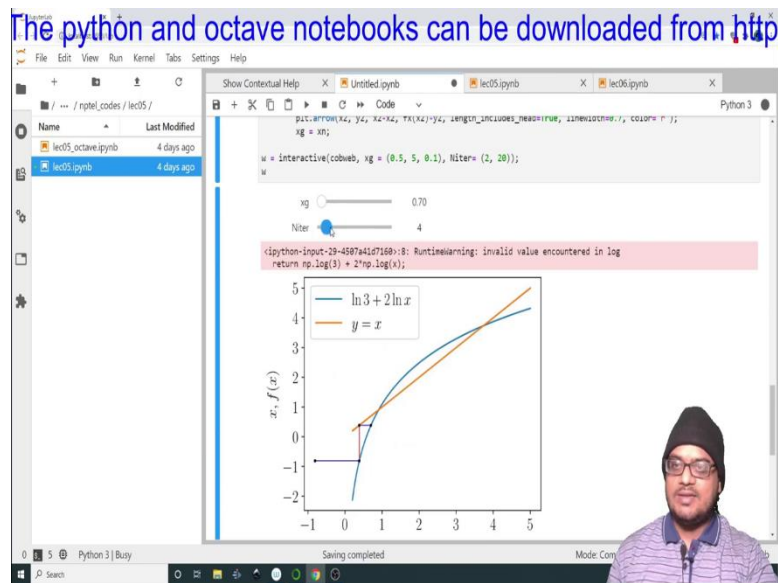


(Refer Slide Time: 17:58)



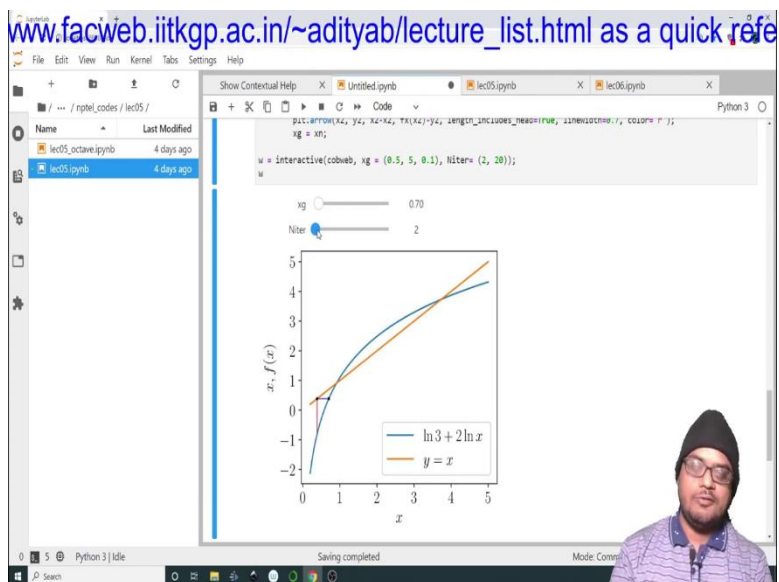
So, if I reduce the number of iterations you see that you are slowly converging toward the root ok. So, this is your guess value and you are slowly converging toward the root.

(Refer Slide Time: 18:08)



Let me change the guess value. Let me slide the guess value over here and let me see how this is how the convergence looks like ok.

(Refer Slide Time: 18:26)



So, it is quite an interesting way of quickly assessing what is going on and in our later studies of dynamical systems and all for understanding how parameter changes can bring about bifurcations for example, so, it will be very useful for us to study that. We can simply change the value of a parameter with the help of a slider and then understand what is actually going on with the various other parameters that we will see.

I mean in the next class were going to start with dynamical systems, but I thought it is very instructive to cover how to make an interactive widget in Jupyter Lab. And this will be very useful for research scholars who are studying multi parameter systems when your advisor comes in and he says can you change this value and show me you can simply move a slider and you can show on the fly how everything looks like, provided of course, your function runs quite fast ok.

So, with this positive note I end this particular lecture and see you again next time when we will start dynamical systems. Bye.