Tools in Scientific Computing

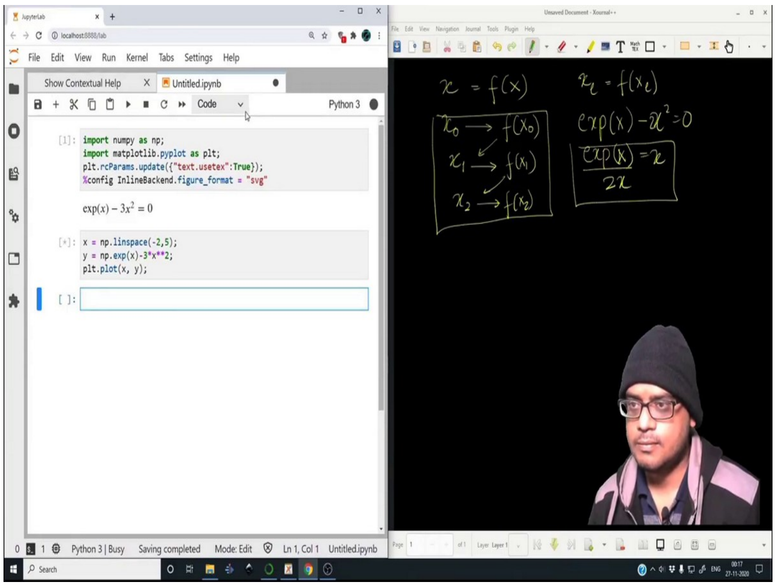
Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture – 05
Nonlinear algebraic equations - Visualizing convergence

Hello welcome to lecture 5 of this course. In this particular lecture we are going to look at Nonlinear algebraic equations. First we are going to look at fixed point iterations and then we are going to look at Newton iterations and finally, we are going to look at basins of attraction for the Newton's iteration. And while many of these things you may be aware there is a few lectures by in NPTEL itself the links will be there in the description.

And the point of this lecture will be to visualize how fixed point iterations converge or how a Newton's iterations converge and hopefully we will pick up a thing how to and how to encode everything in Python let us begin.

(Refer Slide Time: 01:20)



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

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

```
[*]: x = np.linspace(-2,5);
     y = np.exp(x)-3*x**2;
     plt.plot(x, y);
```

So, first things first let me import numpy as np. Let me import matplotlib . pyplot as plt. Let me change the Inline configurator. So, plt . rcParams . update. Inside this we will use text . usetex True and the Inline configInlineBackend . figure format equal to svg.

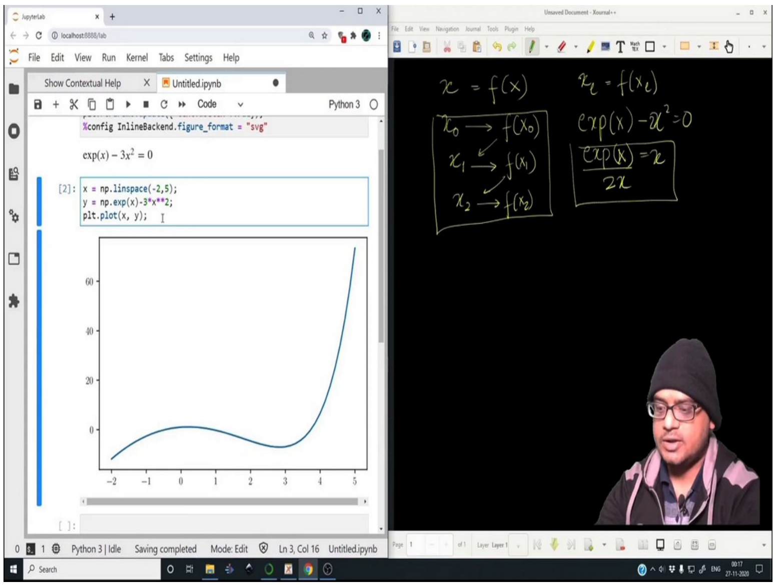
So, once we are done let us begin by looking at the fixed point iteration solution for the equation e^(-x) rather e^(x) – 3*x^2= 0. So, we are interested in solving this nonlinear equation and let us see how we can do that.

So, in fixed point iterations the target is to cast the equation in the following form and the way to numerically do this is to make some guess value say x0. So, using this guess value you find out f(x0), you assign it to x1 then you find f(x1) assign it to x2 then you assign it to f (x2).

So, ideally if we have the root or rather we have the solution of this nonlinear equation. Then x on the left hand side would be equal to the function of x on the right hand side. For example, this particular equation we have e^(x) – 2*x^2=0. So, we can write it in the following form. e^x =2x^2 and we can rearrange this to write it as this; so, x =e^x/2x.

So, if x is the solution then left hand side and right hand side would converge, but if x is not the solution there would be some error and these set of iterations are what are driving us towards the root. And at the end of this particular section it would be clear how the iterations do converge. So, before that let us quickly have a look at how the function actually looks like. So, let me define x =np . linspace (- 2 ,5) y =e^(x)-3 *x^2. So, let me do a plot plt.plot(x ,y).

(Refer Slide Time: 04:47)



```
%config InlineBackend.figure_format = "svg"
```

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

```
[2]: x = np.linspace(-2,5);
     y = np.exp(x)-3*x**2;
     plt.plot(x, y);
```

$x = f(x) \qquad x_l = f(x_l)$

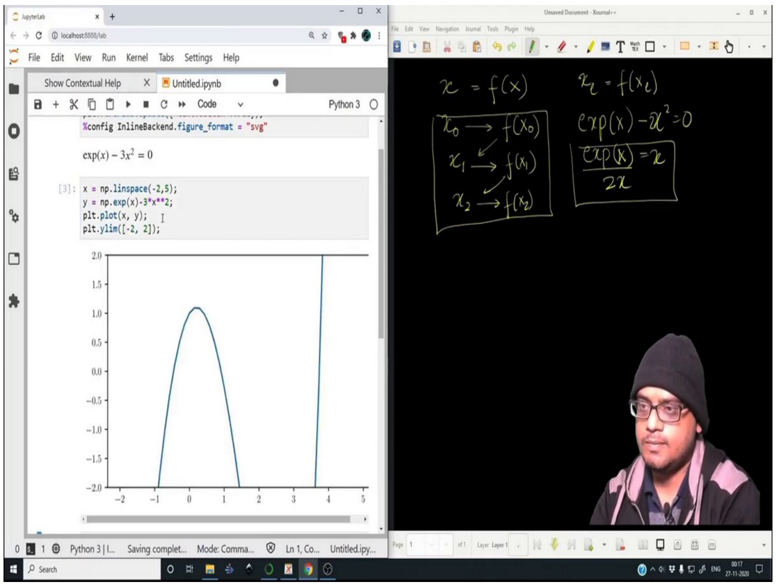
$$\boxed{\begin{tikzcd} x_0 \arrow[r] & f(x_0) \arrow[dl] \\ x_1 \arrow[r] & f(x_1) \arrow[dl] \\ x_2 \arrow[r] & f(x_2) \end{tikzcd}}$$

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

$$\boxed{\frac{\exp(x)}{2x} = x}$$

So, let us adjust the limits in the y direction. So, plt . ylim will make it (-2, 2), let us see.

(Refer Slide Time: 04:56)



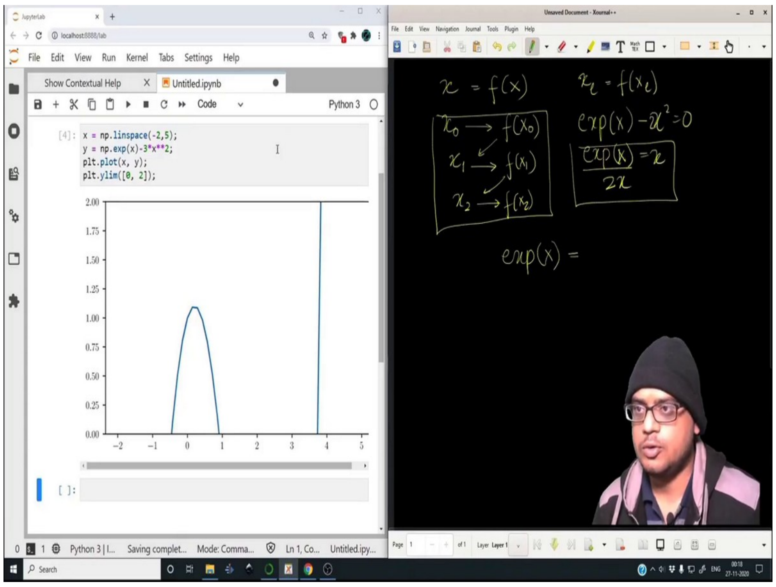
```
%config InlineBackend.figure_format = "svg"
```

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

```
[3]: x = np.linspace(-2,5);
     y = np.exp(x)-3*x**2;
     plt.plot(x, y);
     plt.ylim([-2, 2]);
```

So, let us draw a vertical line through 0.

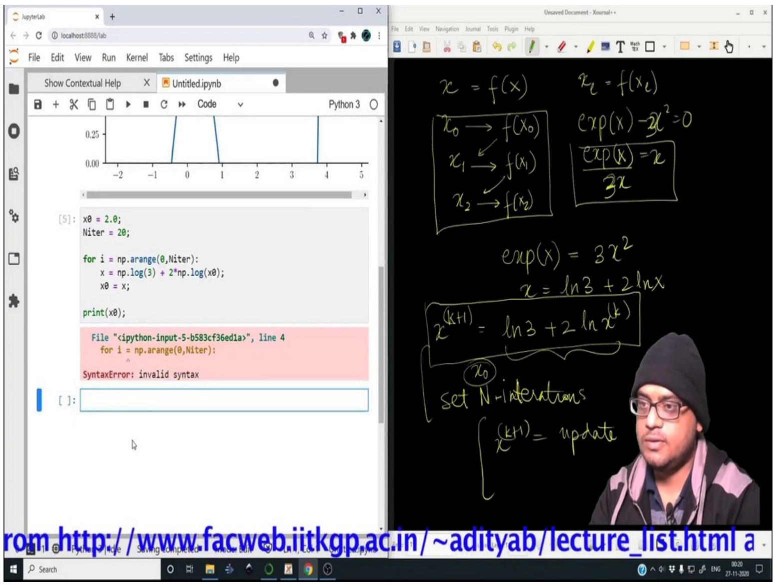
(Refer Slide Time: 05:09)



```
[4]: x = np.linspace(-2,5);
     y = np.exp(x)-3*x**2;
     plt.plot(x, y);
     plt.ylim([0, 2]);

[ ]:
```

In fact, let me just make the limit from 0 to 2. So, the intersection of the curve with the x axis happens at these points and obviously, these points are roots of this particular equation because at these points the whole equation is equal to 0, alright. So, let us see how we can; let us see one strategy of doing this. So, we can do the following. We have e^ x = 3x^2.

(Refer Slide Time: 05:52)



```
[5]: x0 = 2.0;
     Niter = 20;

     for i = np.arange(0,Niter):
         x = np.log(3) + 2*np.log(x0);
         x0 = x;

     print(x0);
```

```
  File "<ipython-input-5-b583cf36ed1a>", line 4
    for i = np.arange(0,Niter):
          ^
SyntaxError: invalid syntax
```

$x = f(x)$ $x_l = f(x_l)$

$$\boxed{\begin{tikzcd} x_0 \arrow[r] & f(x_0) \arrow[dl] \\ x_1 \arrow[r] & f(x_1) \arrow[dl] \\ x_2 \arrow[r] & f(x_2) \end{tikzcd}}$$

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

$\boxed{\dfrac{\exp(x)}{3x} = x}$

$\exp(x) = 3x^2$

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

$\boxed{x^{(k+1)} = \underbrace{\ln 3 + 2\ln x^{(k)}}}$

$\boxed{x_0}$

set N-interations

$x^{(k+1)} =$ update

rom http://www.facweb.iitkgp.ac.in/~adityab/lecture_list.html a

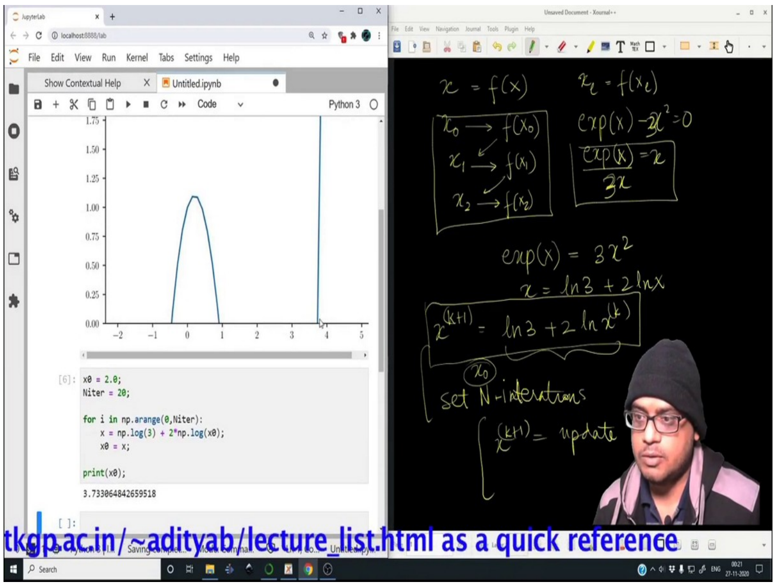
So, e^x=3x^2, let us take a log of this. So, we have log of exponential of x. So, essentially it will be simply x is equal to; so, log of this will be log 3 + 2 log x, alright. So, the iterations would look something like this.

The (k + 1)th value of x will be derived from the kth value of x. So, this is how these iterations are going on. So, it is quite easy to encode you can imagine that you just need to run a loop you just need to set a number of iterations and run a loop. Set iterations and inside this loop you will do an update for the value of x.

So, this is the particular update and; obviously, you will enter the loop with some value of guess alright. So, let me do that. So, let us first define the guess value. So, x0 is equal to; so, let us take a guess value between this, let us take a guess value over here. So, let x0 be equal to 2 alright. So, for; so, let us define the number of iterations.

So, Niter = say 50, 20, does not matter. So, for i equal to np .arange(0,Niter), we would do x is equal to np.log 3 + 2*np.log(x). So, this will be x0 because it is entering the loop with x0 and once we have the value of x will assign x0 the value of x. So, in this way by the time we converge x0 would have the final value. So, let us finally, print x0 alright. So, let me execute this.

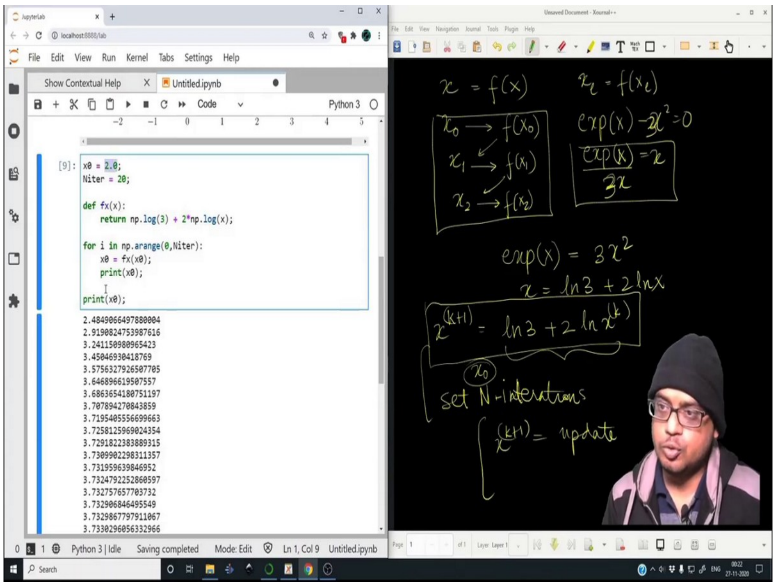
(Refer Slide Time: 08:19)



```
[6]: x0 = 2.0;
     Niter = 20;

     for i in np.arange(0,Niter):
         x = np.log(3) + 2*np.log(x0);
         x0 = x;

     print(x0);

     3.733064842659518
```

So, this is syntax error. So, it gives us a value of 3.7330. So, something close to this alright. So, let me do one thing, let me make this into a function ok.

(Refer Slide Time: 08:39)

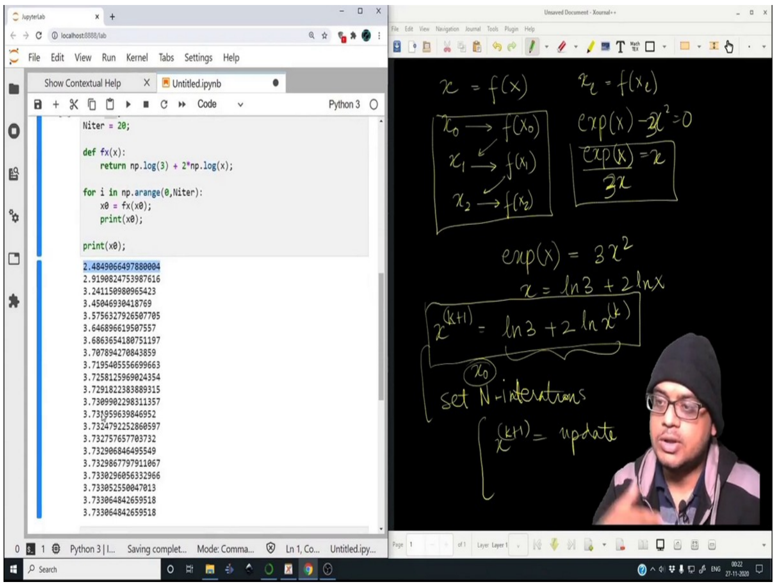


```
[9]: x0 = 2.0;
     Niter = 20;

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

     for i in np.arange(0,Niter):
         x0 = fx(x0);
         print(x0);

     print(x0);

     2.4849066497880004
     2.9190824753987616
     3.241150980965423
     3.45046930418769
     3.5756327926507705
     3.646896619507557
     3.6863654180751197
     3.707894270843859
     3.7195405556699663
     3.7258125969024354
     3.7291822383889315
     3.7309902298311357
     3.731959639846952
     3.7324792252860597
     3.732757657703732
     3.732906846495549
     3.7329867797911067
     3.7330296056332966
```

So, I will define fx which will take as an input x and it will return this ok. So, here I will simply call x0 and it should have the same result. So, it reduces it increases the readability of the program ok. In fact, I could have simply done this I do not need to do all this and I would have the same answer.

Now, suppose I want to visualize how the values of x have evolved of course, I can print all the values of x0 in the loop itself and this is how the iterations look like. So, we started with a guess value of 2, after the first iteration this is how the value of x0 has been updated using this particular function.

(Refer Slide Time: 09:33)

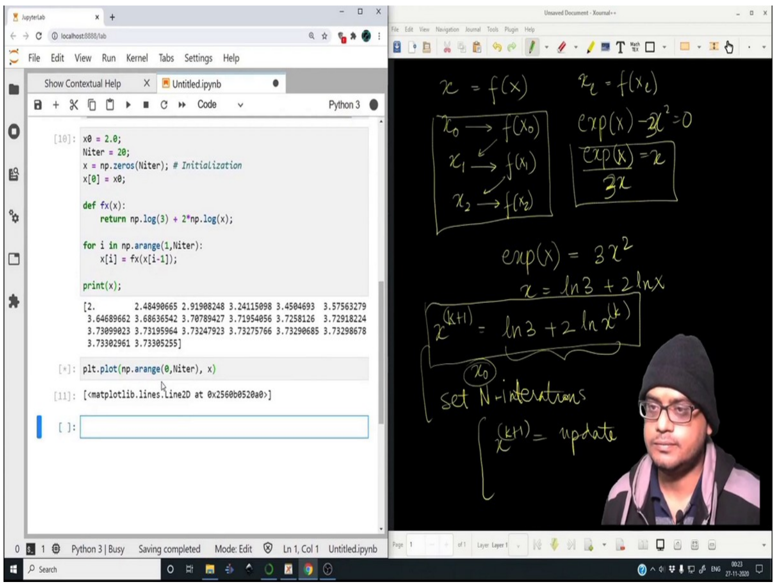


```
Niter = 20;

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

for i in np.arange(0,Niter):
    x0 = fx(x0);
    print(x0);

print(x0);
```

```
2.4849066497880004
2.9190824753987616
3.241150980965423
3.45046930418769
3.5756327926507705
3.646896619507557
3.6863654180751197
3.707894270843859
3.7195405556699663
3.7258125969024354
3.7291822383889315
3.7309902298311357
3.731959639846952
3.7324792252860597
3.732757657703732
3.732906846495549
3.7329867797911067
3.7330296056332966
3.733052550047013
3.733064842659518
3.733064842659518
```

So, each successive iteration leads to a correction in the root and eventually we do get a convergence. So, now, we can actually store all these values in case you want to plot them.

(Refer Slide Time: 09:48)



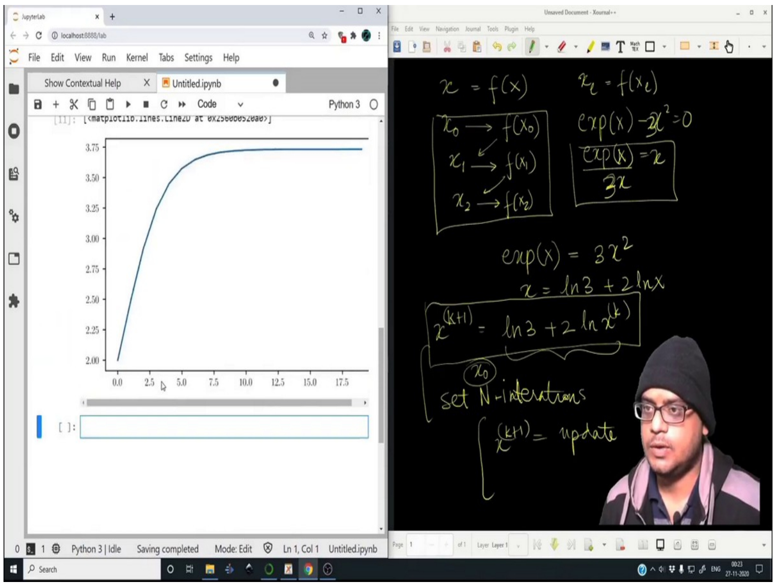
```
[10]: x0 = 2.0;
      Niter = 20;
      x = np.zeros(Niter); # Initialization
      x[0] = x0;

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

      for i in np.arange(1,Niter):
          x[i] = fx(x[i-1]);

      print(x);

      [2.         2.48490665 2.91908248 3.24115098 3.4504693  3.57563279
       3.64689662 3.68636542 3.70789427 3.71954056 3.7258126  3.72918224
       3.73099023 3.73195964 3.73247923 3.73275766 3.73290685 3.73298678
       3.73302961 3.73305255]

[*]: plt.plot(np.arange(0,Niter), x)

[11]: [<matplotlib.lines.Line2D at 0x2560b0520a0>]

[ ]:
```

So, let us define x equal to np.zeros(Niter). So, it will initialize an array all of the elements of that array will be 0. So, this is Initialization. Then we will assign the first value as x0 alright. So, here we will make the loop go from 1 to Niter because we already have the 0th value and here instead of passing x0 we will pass x[i – 1].

So, this modification to the code has the same effect. There is nothing new that we have done to the code and finally, we will print out the value of x. There you have it x is now an array and x contains all the values.

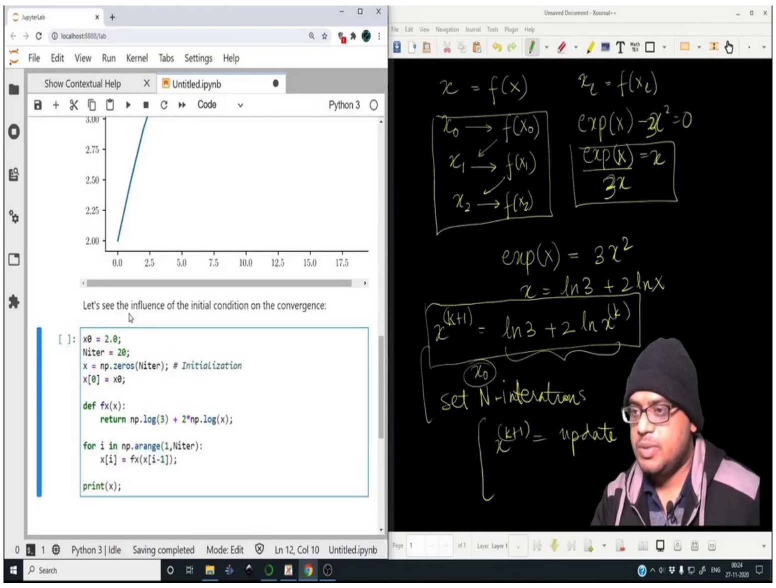
Now, let us see how the convergence has occurred. So, plt.plot; so, we need the iteration number. So, the iteration number will be simply np.arange. This will go from 0 to Niter and on the y axis we will plot x. So, we have started with a guess value of 2 and eventually we have converged towards this root 3.73 whatever ok.

(Refer Slide Time: 11:13)



Now, suppose someone asks well, does it depend really on what the guess value is and we can repurpose this code to find that out. So, let me copy this bit of code. Let me make this a markdown cell and let us write down what we are trying to do.

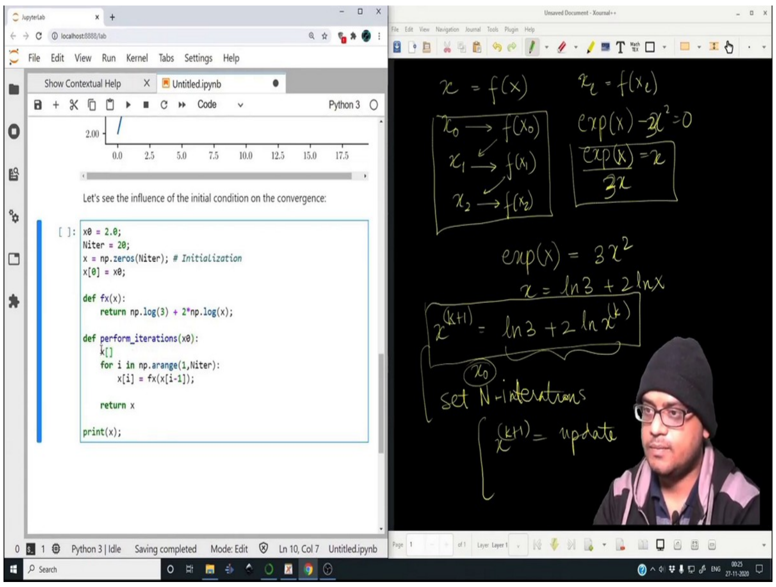
(Refer Slide Time: 11:48)


Let's see the influence of the initial condition on the convergence:

```
x0 = 2.0;
Niter = 20;
x = np.zeros(Niter); # Initialization
x[0] = x0;

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

for i in np.arange(1,Niter):
    x[i] = fx(x[i-1]);

print(x);
```

So, let us see the influence of the initial condition on the convergence, alright. Let me paste this. So, this particular guess value has to be also run in a particular loop. So, how can we do that? Ok.

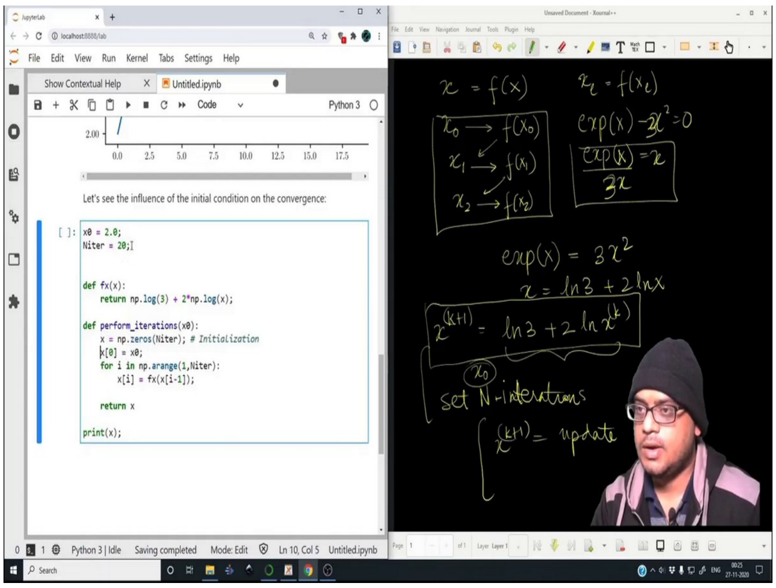
(Refer Slide Time: 12:19)


Let's see the influence of the initial condition on the convergence:

```
x0 = 2.0;
Niter = 20;
x = np.zeros(Niter); # Initialization
x[0] = x0;

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

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

    return x

print(x);
```

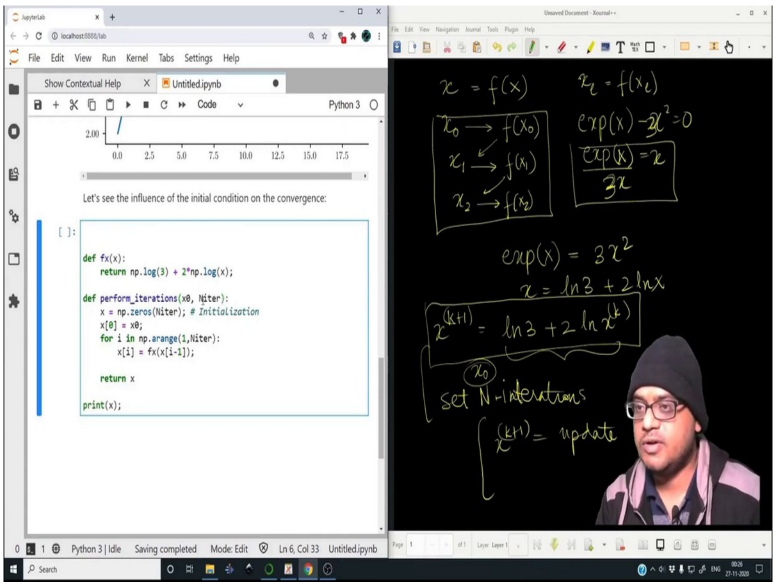
So, let me define f; let me define everything inside a function. Let us say we call it perform iterations ok. And inside perform iterations we will have a for loop and it will return x and the function should take as an input the guess value. Inside the function we will set we will initialize all this inside the function.

(Refer Slide Time: 13:03)


Let's see the influence of the initial condition on the convergence:

```
[ ]: x0 = 2.0;
     Niter = 20;


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

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

         return x

     print(x);
```

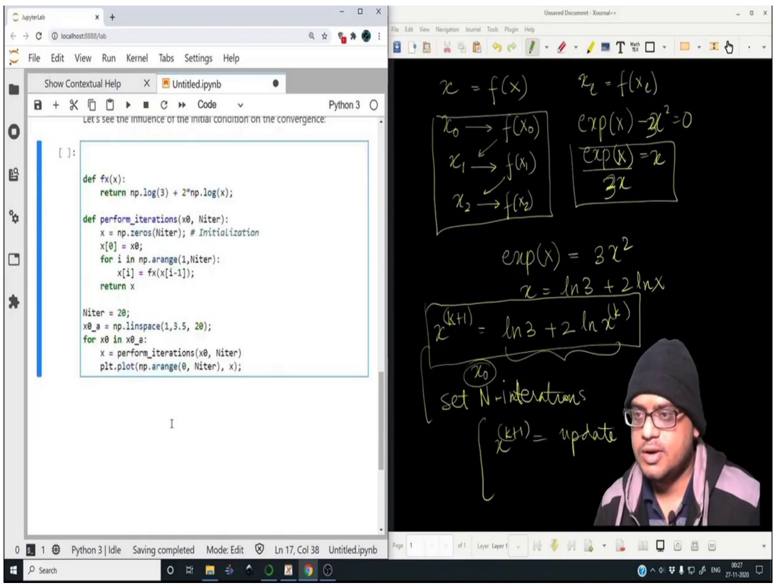
(Refer Slide Time: 13:16)


Let's see the influence of the initial condition on the convergence:

```
[ ]:

     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

     print(x);
```

And we will we do not need Niter anymore because we will I mean we can pass Niter to the function as well. Suppose, you want to test out how many iterations I mean what does the iteration number look like. So, let us pass it to the function perform iterations as well, ok. So, if this particular function will take the guess value and the number of iterations that we want and it will return us the sequence of how x converges ok.

(Refer Slide Time: 13:47)



Let's see the influence of the initial condition on the convergence:

```
[ ]:

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;
x0_a = np.linspace(1,3.5, 20);
for x0 in x0_a:
    x = perform_iterations(x0, Niter)
    plt.plot(np.arange(0, Niter), x);
```

$x = f(x)$ $\quad x_e = f(x_e)$

$$\boxed{\begin{tikzcd} x_0 \arrow[r] & f(x_0) \arrow[dl] \\ x_1 \arrow[r] & f(x_1) \arrow[dl] \\ x_2 \arrow[r] & f(x_2) \end{tikzcd}}$$

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

$$\boxed{\frac{\exp(x)}{3x} = x}$$

$\exp(x) = 3x^2$

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

$$\boxed{x^{(k+1)} = \underbrace{\ln 3 + 2\ln x^{(k)}}}$$

$\boxed{x_0}$

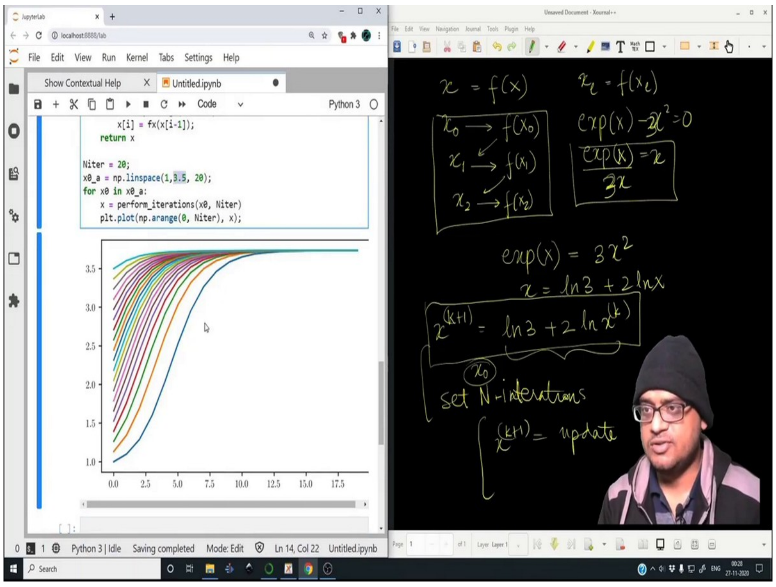
set N-interations

$x^{(k+1)} =$ update

So, for; so, let us define a value of the guess values. So, x0 a is the array which contains the guess values is np.linspace and let us say the guess values are let us have a look. Suppose they go from 1 to 3.5; suppose the guess values go from 1 to 3.5 and let us say we take 20 guess values ok. So, for x0 in x0_a, we will do we will call the function perform.

So, we will assign it to x and we will say perform iterations(x0 , Niter) and we have to define Niter, let it be 20 at the moment no problem. At the end of it we will plot we will plt .plot(np. arrange(0,Niter) , x), it is the same plotting function.

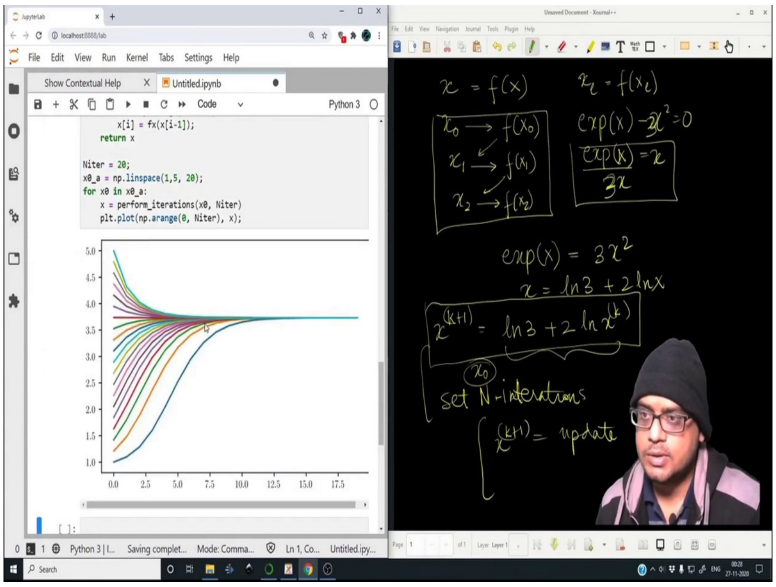
(Refer Slide Time: 14:55)



```
        x[i] = fx(x[i-1]);
    return x

Niter = 20;
x0_a = np.linspace(1,3.5, 20);
for x0 in x0_a:
    x = perform_iterations(x0, Niter)
    plt.plot(np.arange(0, Niter), x);
```

So, let us run this and see what happens alright. We have a bunch of curves for each starting point alright and it is seen that after 13 iterations everything converges ok. For each starting point we have made the plot and it is quite apparent that we converges the initial values do converge to the value of 3.73 whatever it is.

(Refer Slide Time: 15:26)



```
        x[i] = fx(x[i-1]);
    return x

Niter = 20;
x0_a = np.linspace(1,5, 20);
for x0 in x0_a:
    x = perform_iterations(x0, Niter)
    plt.plot(np.arange(0, Niter), x);
```

$x = f(x)$ $x_l = f(x_l)$

$x_0 \rightarrow f(x_0)$, $x_1 \rightarrow f(x_1)$, $x_2 \rightarrow f(x_2)$

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

$\boxed{\frac{\exp(x)}{3x} = x}$

$\exp(x) = 3x^2$

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

$\boxed{x^{(k+1)} = \underbrace{\ln 3 + 2 \ln x^{(k)}}}$

$\textcircled{x_0}$

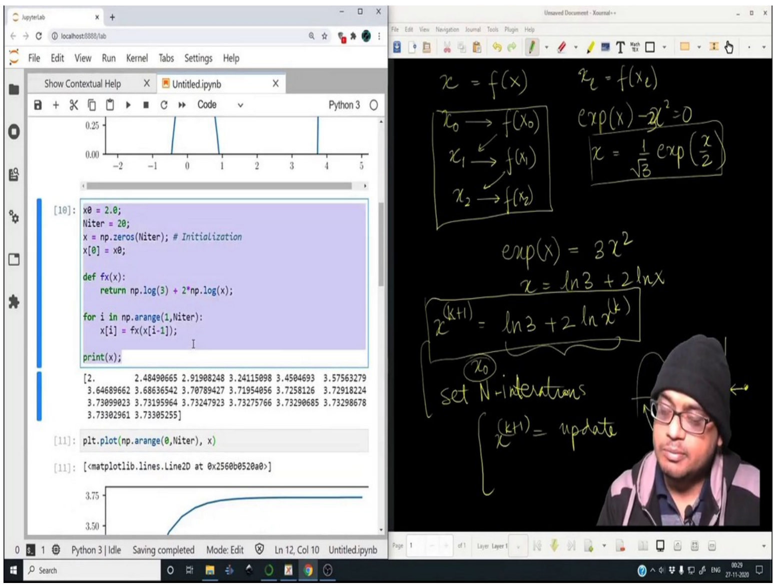
set N-interations

$x^{(k+1)} =$ update

In fact, let me increase this particular range. Let me make it till 5. Let us go above the converged value ok. So, even if you guess it beyond this particular point you seem to converge alright. So, if we have this is the figure right. So, we have if we guess over here we still converge over here, if we guess over here we still converge over here which is great. But, what about these 2 roots? Ok.

We do not seem to have we I mean even though we are starting at a point which is quite close to 1. We do not; even if we start over here we are not converging to this particular root. We are in fact, converging to this particular root. So, let us see what the effect of rearrangement would be ok. So, by rearrangement I mean instead of this let us use this.

(Refer Slide Time: 16:39)

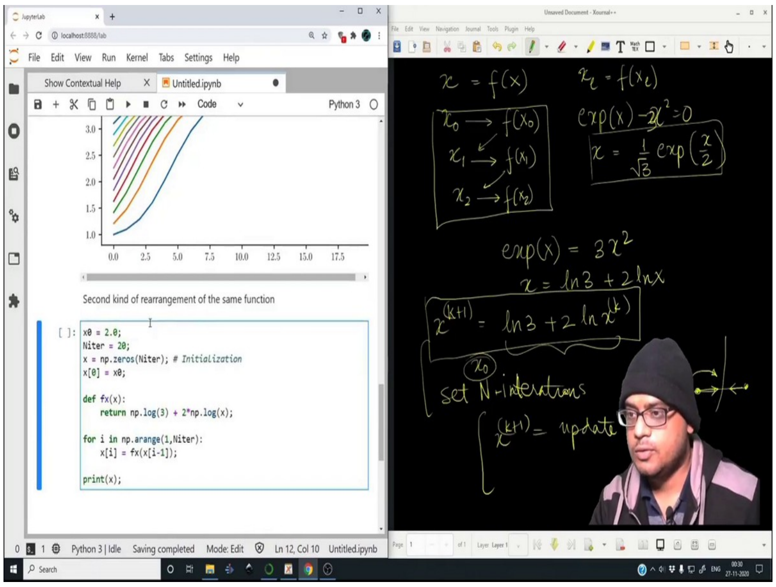


```
[10]: x0 = 2.0;
      Niter = 20;
      x = np.zeros(Niter); # Initialization
      x[0] = x0;

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

      for i in np.arange(1,Niter):
          x[i] = fx(x[i-1]);

      print(x);

[2.         2.48490665 2.91908248 3.24115098 3.4504693  3.57563279
 3.64689662 3.68636542 3.70789427 3.71954056 3.7258126  3.72918224
 3.73099023 3.73195964 3.73247923 3.73275766 3.73290685 3.73298678
 3.73302961 3.73305255]

[11]: plt.plot(np.arange(0,Niter), x)

[11]: [<matplotlib.lines.Line2D at 0x2560b0520a0>]
```

So, over here we had let me; in fact, erase this let us do a simpler rearrangement. So, let me rearrange this to x = 1/sqrt(3)* e^(x/2). So, this is one more rearrangement that is possible alright. So, let us go back to our program.

Let me copy this bit of code. I mean we can of course, reuse it because we have wrapped the RHS into a function. We do not need to re do many things. We just need to change the functional form, everything else remains the same alright.

(Refer Slide Time: 17:13)

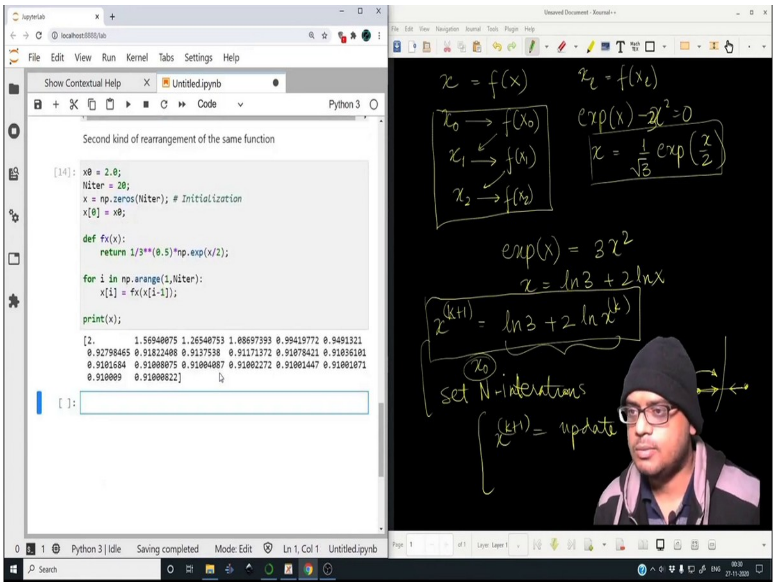


Second kind of rearrangement of the same function

```
[ ]: x0 = 2.0;
     Niter = 20;
     x = np.zeros(Niter); # Initialization
     x[0] = x0;

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

     for i in np.arange(1,Niter):
         x[i] = fx(x[i-1]);

     print(x);
```

So, this will be second kind of rearrangement alright. So, let us go ahead and change the return value.

(Refer Slide Time: 17:27)



Second kind of rearrangement of the same function

```
[14]: x0 = 2.0;
      Niter = 20;
      x = np.zeros(Niter); # Initialization
      x[0] = x0;

      def fx(x):
          return 1/3**(0.5)*np.exp(x/2);

      for i in np.arange(1,Niter):
          x[i] = fx(x[i-1]);

      print(x);
```

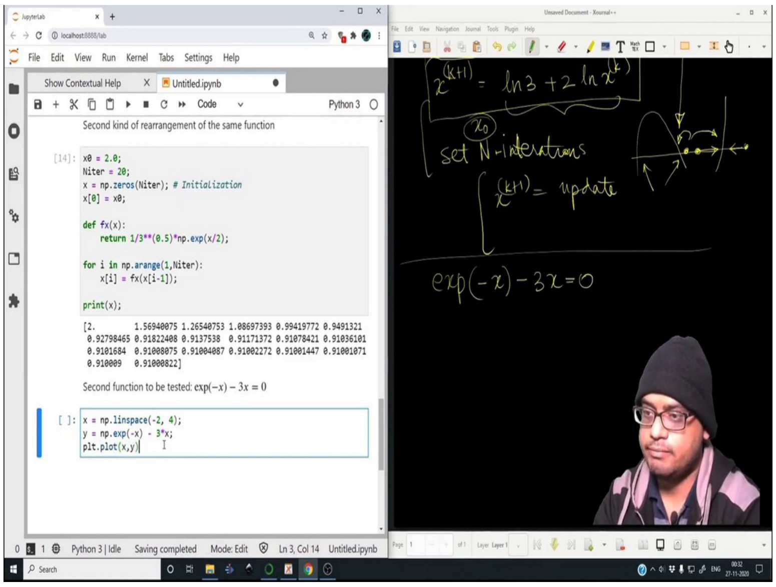
```
[2.         1.56940075 1.26540753 1.08697393 0.99419772 0.9491321
 0.92798465 0.91822408 0.9137538  0.91171372 0.91078421 0.91036101
 0.9101684  0.91008075 0.91004087 0.91002272 0.91001447 0.91001071
 0.910009   0.91000822]
```

$x = f(x)$
$x_1 = f(x_0)$
$x_0 \longrightarrow f(x_0)$
$x_1 \longrightarrow f(x_1)$
$x_2 \longrightarrow f(x_2)$
$\exp(x) - 3x^2 = 0$
$x = \frac{1}{\sqrt{3}} \exp\left(\frac{x}{2}\right)$
$\exp(x) = 3x^2$
$x = \ln 3 + 2 \ln x$
$x^{(k+1)} = \ln 3 + 2 \ln x^{(k)}$
x_0
set N-iterations
$x^{(k+1)} =$ update

So, this will be 1/saqrt(3)*e^(x/2) alright ok. So, let me let us take the guess value as 2 no problem. So, let us run this. Let us see what happens ok. So, we started at a guess value of 2 and we have converged to a value of 0.91 whatever it is. Let us see whether that is true. So, 0.9 was 0.91 is actually the second root.

So, by changing the type of rearrangement that we have we seem to converge to the second root. So, far we have seen two cases. I am not going to elaborate on this any further. These are fixed point iterations for this particular function. Let us have a look at a different kind of function and it will demonstrate as to how this procedure does not always work out. There may be cases where you will never converge. Let us see.

(Refer Slide Time: 18:48)


Second kind of rearrangement of the same function

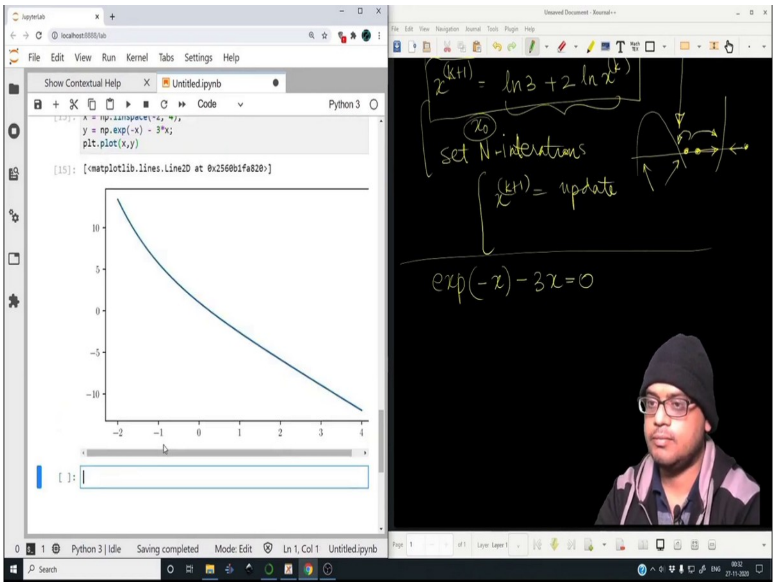
```
[14]: x0 = 2.0;
      Niter = 20;
      x = np.zeros(Niter); # Initialization
      x[0] = x0;

      def fx(x):
          return 1/3**(0.5)*np.exp(x/2);

      for i in np.arange(1,Niter):
          x[i] = fx(x[i-1]);

      print(x);

[2.         1.56940075 1.26540753 1.08697393 0.99419772 0.9491321
 0.92798465 0.91822408 0.9137538  0.91171372 0.91078421 0.91036101
 0.9101684  0.91008075 0.91004087 0.91002272 0.91001447 0.91001071
 0.910009   0.91000822]
```

Second function to be tested: $\exp(-x) - 3x = 0$

```
[ ]: x = np.linspace(-2, 4);
     y = np.exp(-x) - 3*x;
     plt.plot(x,y)
```

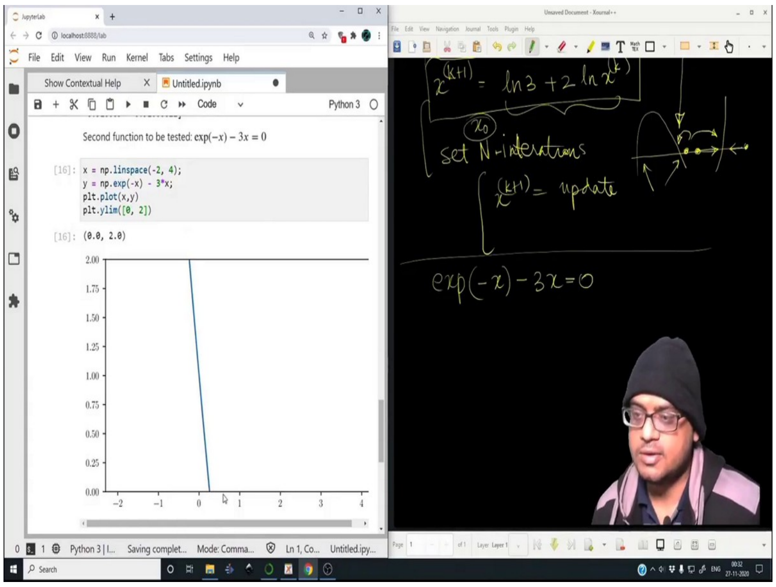
So, let the function let me write down the function. So, it is exponential of - x - 3x equal to 0, alright. So, let us plot this function alright. So, x = np . linspace(- 2,5) and I should do 4; y=e^(- x) – 3*x; plt.plot(x,y) alright.

(Refer Slide Time: 19:38)



```
      x = np.linspace(-2, 4);
      y = np.exp(-x) - 3*x;
      plt.plot(x,y)

[15]: [<matplotlib.lines.Line2D at 0x2560b1fa820>]
```

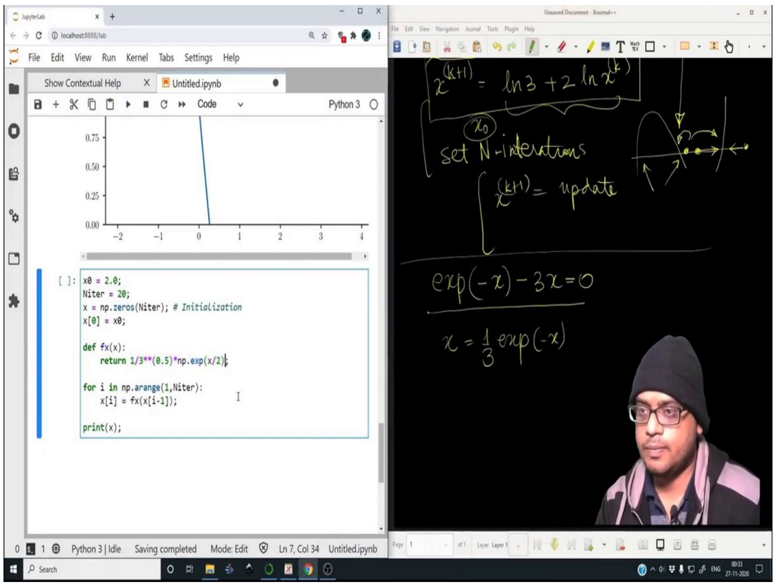
(Refer Slide Time: 19:48)


Second function to be tested: $\exp(-x) - 3x = 0$

```
[16]: x = np.linspace(-2, 4);
      y = np.exp(-x) - 3*x;
      plt.plot(x,y)
      plt.ylim([0, 2])

[16]: (0.0, 2.0)
```

This is how the function looks like and let me set the y limit; plt . ylim, this will be 0 , 2 for example. So, we do have a solution over here. It looks like something point 2 something ok. So, let me repackage this code.

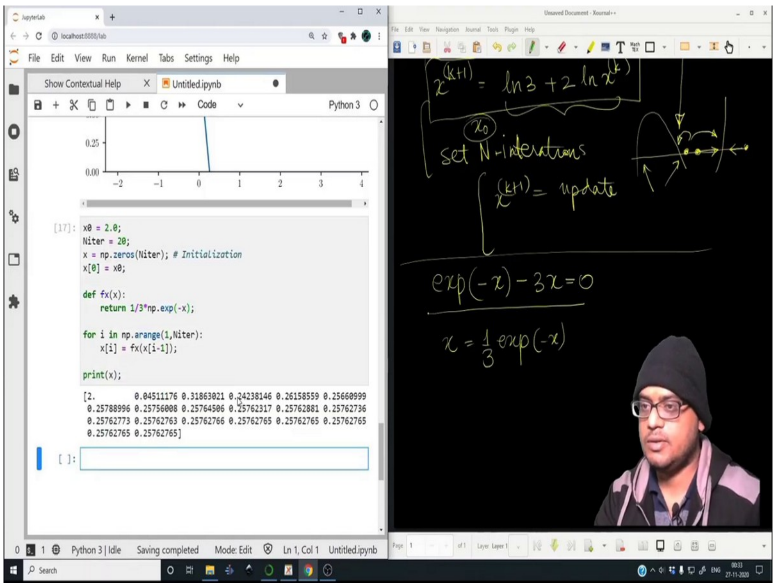
(Refer Slide Time: 20:13)


```
[ ]: x0 = 2.0;
     Niter = 20;
     x = np.zeros(Niter); # Initialization
     x[0] = x0;

     def fx(x):
         return 1/3**(0.5)*np.exp(x/2);

     for i in np.arange(1,Niter):
         x[i] = fx(x[i-1]);

     print(x);
```

$x^{(k+1)} = \ln 3 + 2 \ln x^{(k)}$
x_0
set N-iterations
$x^{(k+1)} = $ update
$\exp(-x) - 3x = 0$
$x = \frac{1}{3} \exp(-x)$

And let us see let us change this functional form. So, let us see two functional form. So, the first functional form it can be x = 1/3e^(- x) and that is a very straightforward looking functional form. Well, let us have a look.

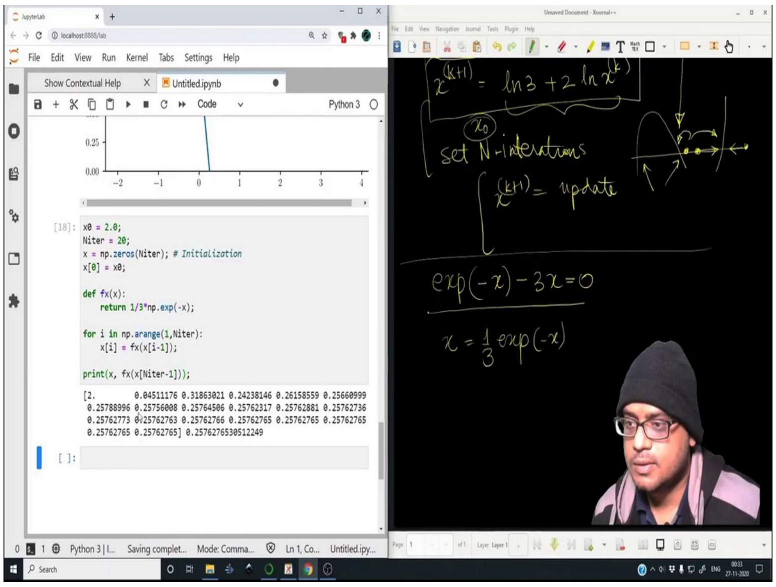
(Refer Slide Time: 20:38)



```
[17]: x0 = 2.0;
      Niter = 20;
      x = np.zeros(Niter); # Initialization
      x[0] = x0;

      def fx(x):
          return 1/3*np.exp(-x);

      for i in np.arange(1,Niter):
          x[i] = fx(x[i-1]);

      print(x);

[2.         0.04511176 0.31863021 0.24238146 0.26158559 0.25660999
 0.25788996 0.25756008 0.25764506 0.25762317 0.25762881 0.25762736
 0.25762773 0.25762763 0.25762766 0.25762765 0.25762765 0.25762765
 0.25762765 0.25762765]
```

So, 1/3e^(- x), so, let me run this. Well that is great. We do converge to a value of 0.25, it does visually look like its a root and there is an easier way of checking whether its a root.

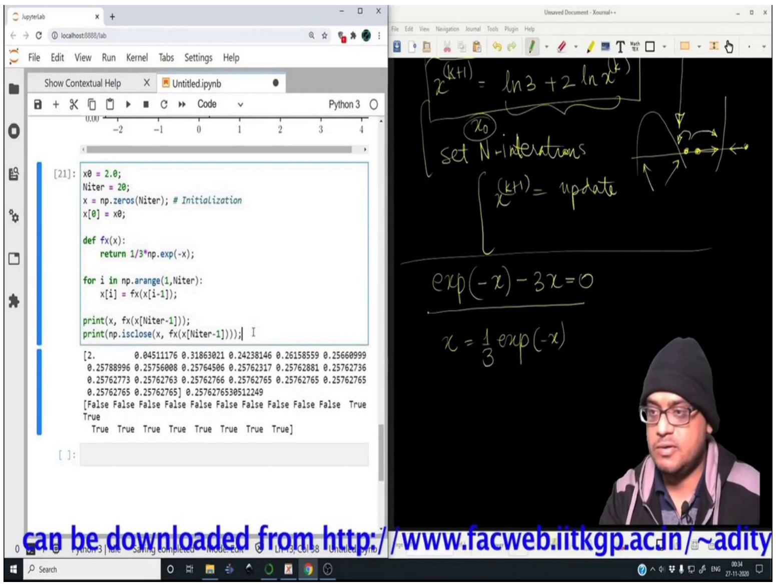
(Refer Slide Time: 21:00)



```
[18]: x0 = 2.0;
      Niter = 20;
      x = np.zeros(Niter); # Initialization
      x[0] = x0;

      def fx(x):
          return 1/3*np.exp(-x);

      for i in np.arange(1,Niter):
          x[i] = fx(x[i-1]);

      print(x, fx(x[Niter-1]));

[2.         0.04511176 0.31863021 0.24238146 0.26158559 0.25660999
 0.25788996 0.25756008 0.25764506 0.25762317 0.25762881 0.25762736
 0.25762773 0.25762763 0.25762766 0.25762765 0.25762765 0.25762765
 0.25762765 0.25762765] 0.2576276530512249
```

Let us print fx (x[Niter – 1]), essentially I am trying to check the last value ok; f x. So, the value of x and the value of this we do match. In fact, let us print this also, sorry ok.

(Refer Slide Time: 21:26)



```
[21]: x0 = 2.0;
      Niter = 20;
      x = np.zeros(Niter); # Initialization
      x[0] = x0;

      def fx(x):
          return 1/3*np.exp(-x);

      for i in np.arange(1,Niter):
          x[i] = fx(x[i-1]);

      print(x, fx(x[Niter-1]));
      print(np.isclose(x, fx(x[Niter-1])));
```

```
[2.         0.04511176 0.31863021 0.24238146 0.26158559 0.25660999
 0.25788996 0.25756008 0.25764506 0.25762317 0.25762881 0.25762736
 0.25762773 0.25762763 0.25762766 0.25762765 0.25762765 0.25762765
 0.25762765 0.25762765] 0.2576276530512249
[False False False False False False False False False False  True
True
  True  True  True  True  True  True  True  True]
```

$$\boxed{x^{(k+1)} = \ln 3 + 2 \ln x^{(k)}}$$

$$\text{set } N\text{-interations} \quad \boxed{x_0}$$

$$x^{(k+1)} = \text{update}$$

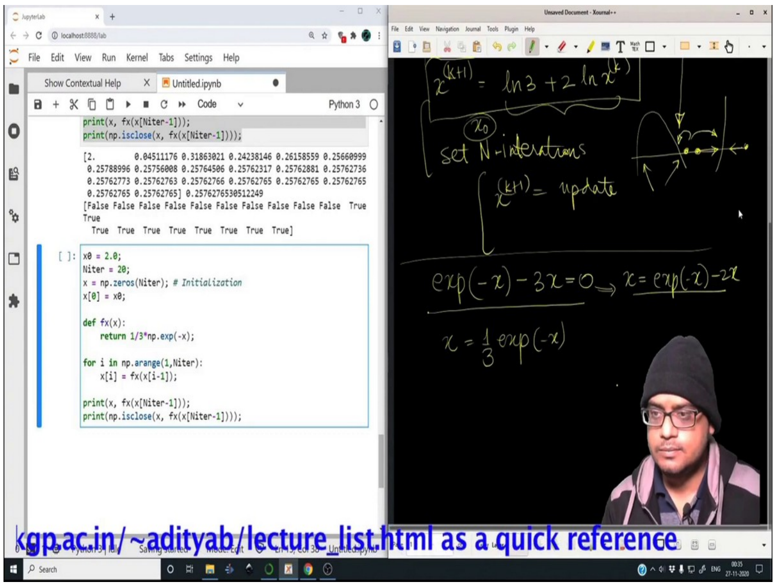
$$\exp(-x) - 3x = 0$$

$$x = \frac{1}{3}\exp(-x)$$

can be downloaded from http://www.facweb.iitkgp.ac.in/~adity

So, it does say that initially they are not close ok, initially they are not close, but as we go towards; so, this array shows it is checking this particular value with all these values with all the errors of x and after a certain point in the iteration sequence they become close and that is why we start getting True ok, this is how you can do a quick check. Anyway, so, let us try to do another rearrangement of this ok. Let me copy this code.

(Refer Slide Time: 22:19)



```
      print(x, fx(x[Niter-1]));
      print(np.isclose(x, fx(x[Niter-1])));

      [2.         0.04511176 0.31863021 0.24238146 0.26158559 0.25660999
       0.25788996 0.25756008 0.25764506 0.25762317 0.25762881 0.25762736
       0.25762773 0.25762763 0.25762766 0.25762765 0.25762765 0.25762765
       0.25762765 0.25762765] 0.2576276530512249
      [False False False False False False False False False False  True
      True
        True  True  True  True  True  True  True  True]

[ ]:  x0 = 2.0;
      Niter = 20;
      x = np.zeros(Niter); # Initialization
      x[0] = x0;

      def fx(x):
          return 1/3*np.exp(-x);

      for i in np.arange(1,Niter):
          x[i] = fx(x[i-1]);

      print(x, fx(x[Niter-1]));
      print(np.isclose(x, fx(x[Niter-1])));
```

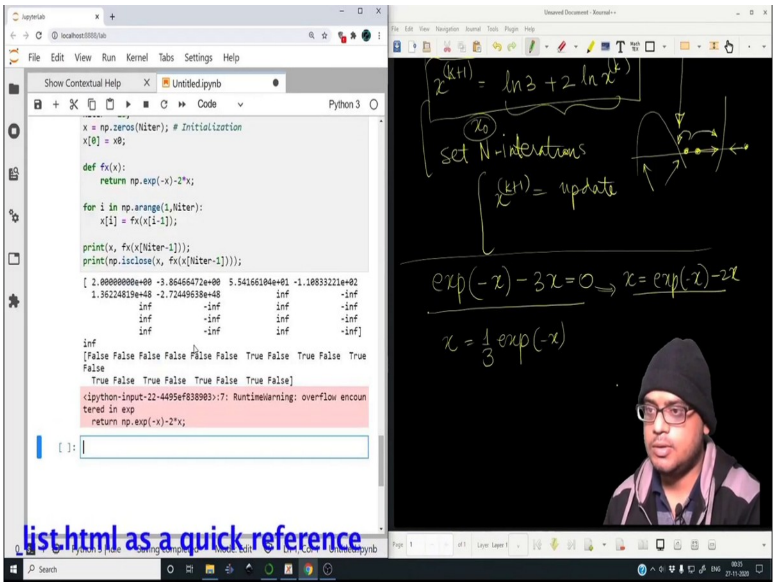
$$\exp(-x) - 3x = 0 \rightarrow x = \exp(-x) - 2x$$

$$x = \frac{1}{3}\exp(-x)$$

kgp.ac.in/~adityab/lecture_list.html as a quick reference

Let us do a rearrangement which is also quite trivial to be honest. Let us take this 3x and let us split it into 2x and x. Let us take one of the x on the other side. So, consequently we will have this. So, this is also an expression for fixed point iterations.

(Refer Slide Time: 22:45)



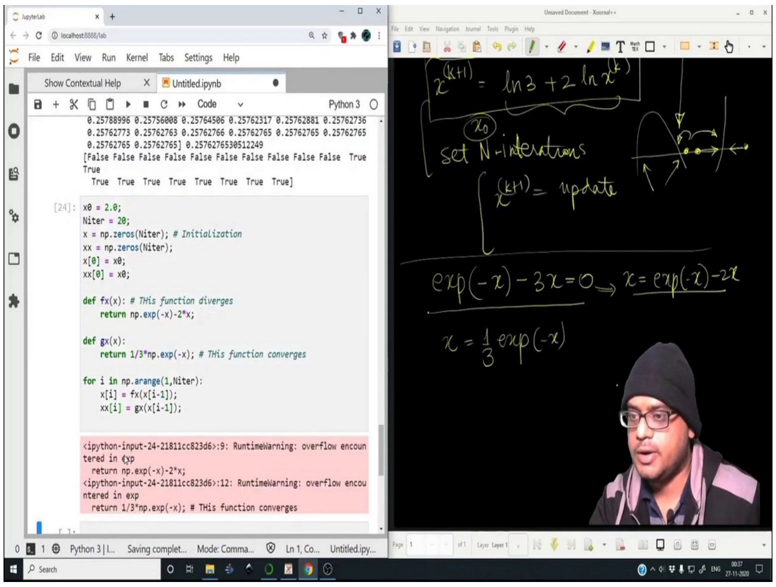
```
x = np.zeros(Niter); # Initialization
x[0] = x0;

def fx(x):
    return np.exp(-x)-2*x;

for i in np.arange(1,Niter):
    x[i] = fx(x[i-1]);

print(x, fx(x[Niter-1]));
print(np.isclose(x, fx(x[Niter-1])));
```

```
[ 2.00000000e+00 -3.86466472e+00  5.54166104e+01 -1.10833221e+02
  1.36224819e+48 -2.72449638e+48             inf            -inf
             inf            -inf             inf            -inf
             inf            -inf             inf            -inf
             inf            -inf             inf            -inf]
inf
[False False False False False False  True False  True False  True
False
  True False  True False  True False  True False]
<ipython-input-22-4495ef838903>:7: RuntimeWarning: overflow encoun
tered in exp
  return np.exp(-x)-2*x;
```

$x^{(k+1)} = \ln 3 + 2\ln x^{(k)}$
x_0
set N-iterations
$x^{(k+1)}=$ update
$\exp(-x)-3x=0 \rightarrow x=\exp(-x)-2x$
$x=\frac{1}{3}\exp(-x)$
list.html as a quick reference

So, let us reformulate the function, alright ok. So, when I execute the cell it seems and initially the value is 2 then something happens and then it flies off into infinity ok. It flies off it does not converge. In fact, let us plot these two sequences ok, let us try to plot these two sequences.

(Refer Slide Time: 23:25)



```
 0.25788996 0.25756008 0.25764506 0.25762317 0.25762881 0.25762736
 0.25762773 0.25762763 0.25762766 0.25762765 0.25762765 0.25762765
 0.25762765 0.25762765] 0.2576276530512249
[False False False False False False False False False False  True
True
  True  True  True  True  True  True  True  True]
```

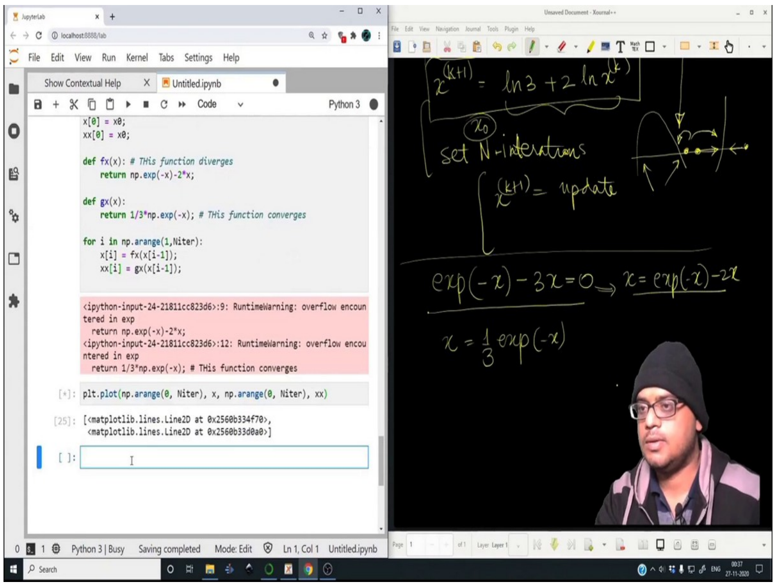
```
[24]: x0 = 2.0;
Niter = 20;
x = np.zeros(Niter); # Initialization
xx = np.zeros(Niter);
x[0] = x0;
xx[0] = x0;

def fx(x): # THis function diverges
    return np.exp(-x)-2*x;

def gx(x):
    return 1/3*np.exp(-x); # THis function converges

for i in np.arange(1,Niter):
    x[i] = fx(x[i-1]);
    xx[i] = gx(x[i-1]);
```

```
<ipython-input-24-21811cc823d6>:9: RuntimeWarning: overflow encoun
tered in exp
  return np.exp(-x)-2*x;
<ipython-input-24-21811cc823d6>:12: RuntimeWarning: overflow encou
ntered in exp
  return 1/3*np.exp(-x); # THis function converges
```

$x^{(k+1)} = \ln 3 + 2\ln x^{(k)}$
x_0
set N-iterations
$x^{(k+1)}=$ update
$\exp(-x)-3x=0 \rightarrow x=\exp(-x)-2x$
$x=\frac{1}{3}\exp(-x)$

Let me copy this. Let me call this gx. So, gx converges and this function appears to diverge. So, let me create another array xx. So, np . zeros(Niter) and xx[0]= x0. So, I will run a single loop, but I will perform the two iterations side by side and I am performing the two iterations for the two different functions side by side. Sorry this should be i, alright. So, it showing a overflow error because it is diverging to infinity, but that is.

(Refer Slide Time: 24:26)



```
x[0] = x0;
xx[0] = x0;

def fx(x): # THis function diverges
    return np.exp(-x)-2*x;

def gx(x):
    return 1/3*np.exp(-x); # THis function converges

for i in np.arange(1,Niter):
    x[i] = fx(x[i-1]);
    xx[i] = gx(x[i-1]);
```

```
<ipython-input-24-21811cc823d6>:9: RuntimeWarning: overflow encoun
tered in exp
  return np.exp(-x)-2*x;
<ipython-input-24-21811cc823d6>:12: RuntimeWarning: overflow encou
ntered in exp
  return 1/3*np.exp(-x); # THis function converges
```

```
[*]: plt.plot(np.arange(0, Niter), x, np.arange(0, Niter), xx)

[25]: [<matplotlib.lines.Line2D at 0x2560b334f70>,
       <matplotlib.lines.Line2D at 0x2560b33d0a0>]
```

$x^{(k+1)} = \ln 3 + 2\ln x^{(k)}$

x_0

set N-interations

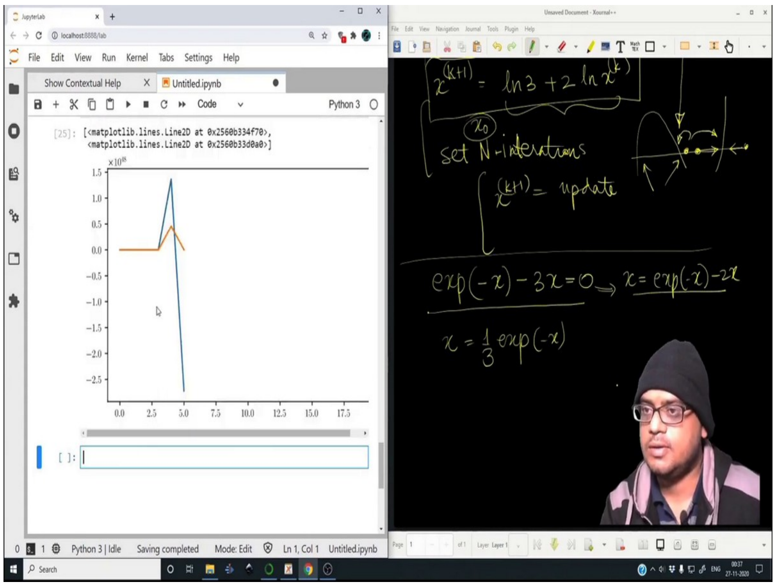
$x^{(k+1)} =$ update

$\exp(-x) - 3x = 0 \Rightarrow x = \exp(-x) - 2x$

$x = \frac{1}{3}\exp(-x)$

Let us do the plot. So, plt . plot (np.arange(0,Niter) , x, np.arange(0,Niter) , xx) alright.

(Refer Slide Time: 24:46)



```
[25]: [<matplotlib.lines.Line2D at 0x2560b334f70>,
       <matplotlib.lines.Line2D at 0x2560b33d0a0>]
```

$x^{(k+1)} = \ln 3 + 2\ln x^{(k)}$

x_0

set N-interations

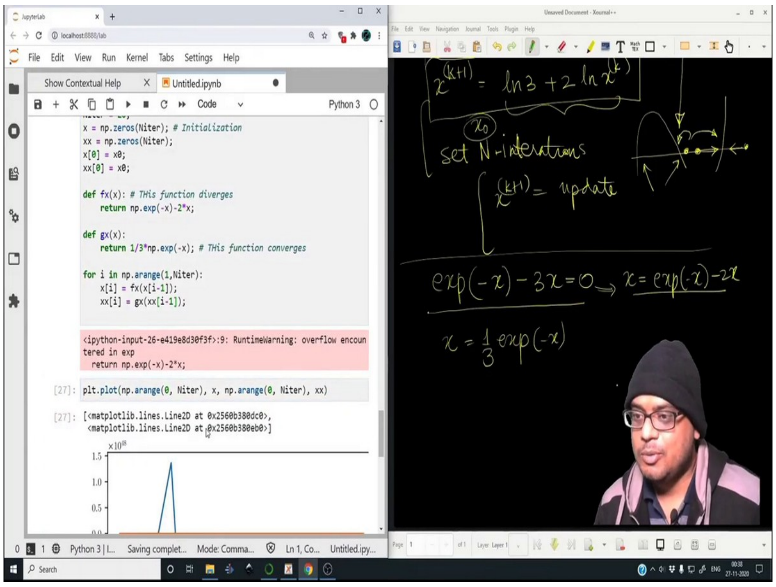
$x^{(k+1)} =$ update

$\exp(-x) - 3x = 0 \Rightarrow x = \exp(-x) - 2x$

$x = \frac{1}{3}\exp(-x)$

Let us see ok. So, quite obviously, the blue curve it diverges, it diverges quite widely, this should be xx.

(Refer Slide Time: 24:59)



```
x = np.zeros(Niter); # Initialization
xx = np.zeros(Niter);
x[0] = x0;
xx[0] = x0;

def fx(x): # THis function diverges
    return np.exp(-x)-2*x;

def gx(x):
    return 1/3*np.exp(-x); # THis function converges

for i in np.arange(1,Niter):
    x[i] = fx(x[i-1]);
    xx[i] = gx(xx[i-1]);
```

```
<ipython-input-26-e419e8d30f3f>:9: RuntimeWarning: overflow encoun
tered in exp
  return np.exp(-x)-2*x;
```

```
[27]: plt.plot(np.arange(0, Niter), x, np.arange(0, Niter), xx)
```

```
[27]: [<matplotlib.lines.Line2D at 0x2560b380dc0>,
       <matplotlib.lines.Line2D at 0x2560b380eb0>]
```

$x^{(k+1)} = \ln 3 + 2\ln x^{(k)}$

x_0

set N-iterations

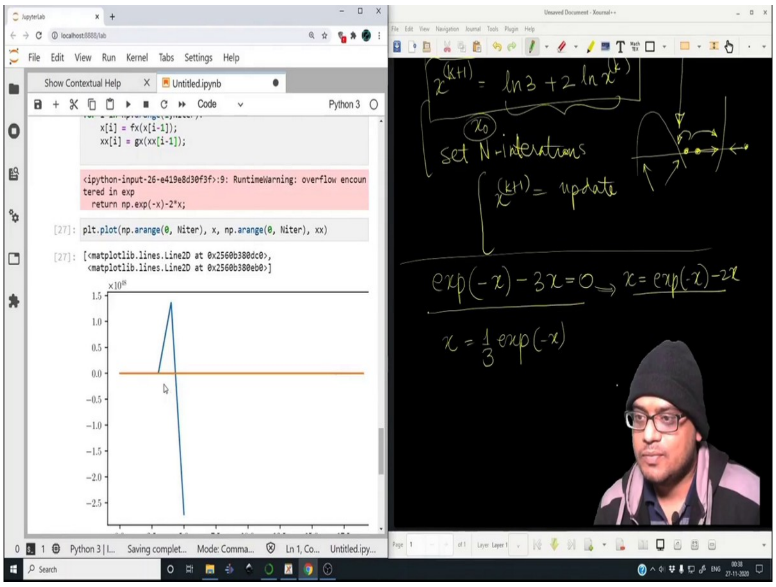
$x^{(k+1)} =$ update

$\exp(-x) - 3x = 0 \Rightarrow x = \exp(-x) - 2x$

$x = \frac{1}{3}\exp(-x)$

This was the small error because obviously, it has to take the its own array there is a small error alright.

(Refer Slide Time: 25:09)



```
    x[i] = fx(x[i-1]);
    xx[i] = gx(xx[i-1]);
```

```
<ipython-input-26-e419e8d30f3f>:9: RuntimeWarning: overflow encoun
tered in exp
  return np.exp(-x)-2*x;
```

```
[27]: plt.plot(np.arange(0, Niter), x, np.arange(0, Niter), xx)
```

```
[27]: [<matplotlib.lines.Line2D at 0x2560b380dc0>,
       <matplotlib.lines.Line2D at 0x2560b380eb0>]
```

$x^{(k+1)} = \ln 3 + 2\ln x^{(k)}$

x_0

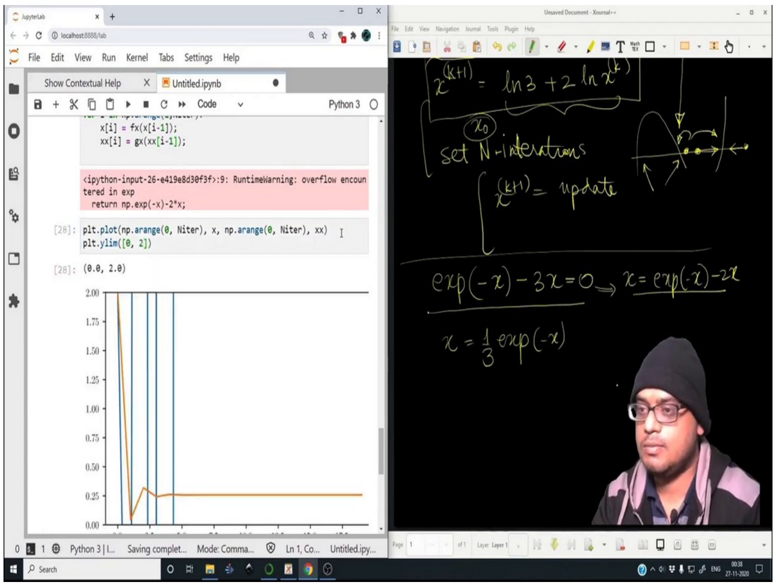
set N-iterations

$x^{(k+1)} =$ update

$\exp(-x) - 3x = 0 \Rightarrow x = \exp(-x) - 2x$

$x = \frac{1}{3}\exp(-x)$

(Refer Slide Time: 25:13)



```
    x[i] = fx(x[i-1]);
    xx[i] = gx(xx[i-1]);
```

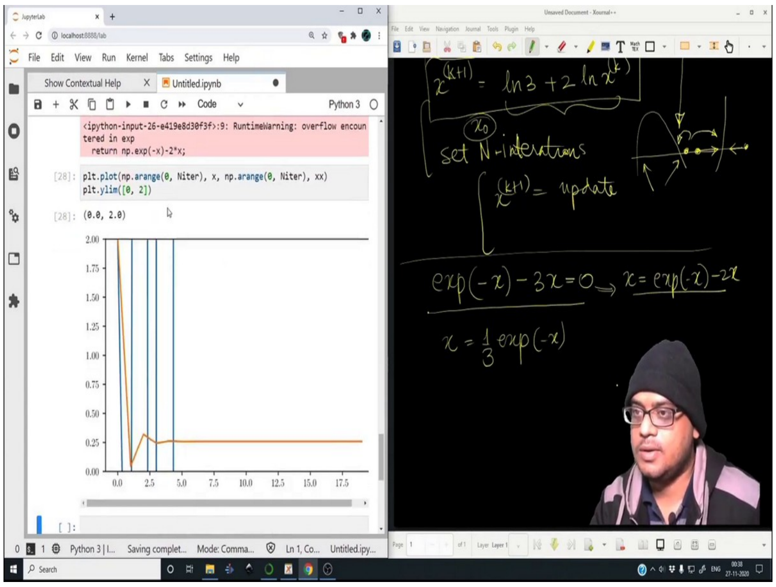
```
<ipython-input-26-e419e8d30f3f>:9: RuntimeWarning: overflow encoun
tered in exp
  return np.exp(-x)-2*x;
```

```
[28]: plt.plot(np.arange(0, Niter), x, np.arange(0, Niter), xx)
      plt.ylim([0, 2])

[28]: (0.0, 2.0)
```

$x^{(k+1)} = \ln 3 + 2 \ln x^{(k)}$
x_0
set N-iterations
$x^{(k+1)} =$ update
$\exp(-x) - 3x = 0 \rightarrow x = \exp(-x) - 2x$
$x = \frac{1}{3}\exp(-x)$

So, let me replot everything ok. So, let us change the y limit. So, plt . ylim ok.

(Refer Slide Time: 25:29)

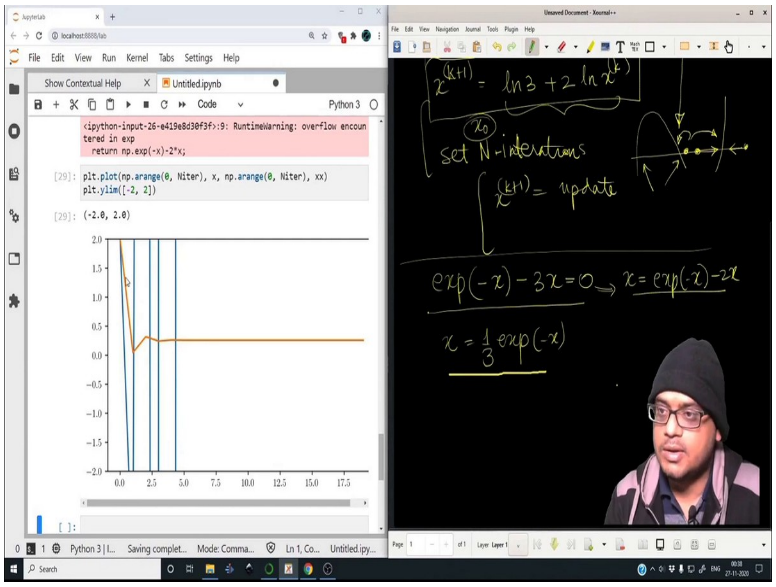


```
<ipython-input-26-e419e8d30f3f>:9: RuntimeWarning: overflow encoun
tered in exp
  return np.exp(-x)-2*x;
```

```
[28]: plt.plot(np.arange(0, Niter), x, np.arange(0, Niter), xx)
      plt.ylim([0, 2])

[28]: (0.0, 2.0)
```

$x^{(k+1)} = \ln 3 + 2 \ln x^{(k)}$
x_0
set N-iterations
$x^{(k+1)} =$ update
$\exp(-x) - 3x = 0 \rightarrow x = \exp(-x) - 2x$
$x = \frac{1}{3}\exp(-x)$

(Refer Slide Time: 25:23)



```
<ipython-input-26-e419e8d30f3f>:9: RuntimeWarning: overflow encoun
tered in exp
  return np.exp(-x)-2*x;
```

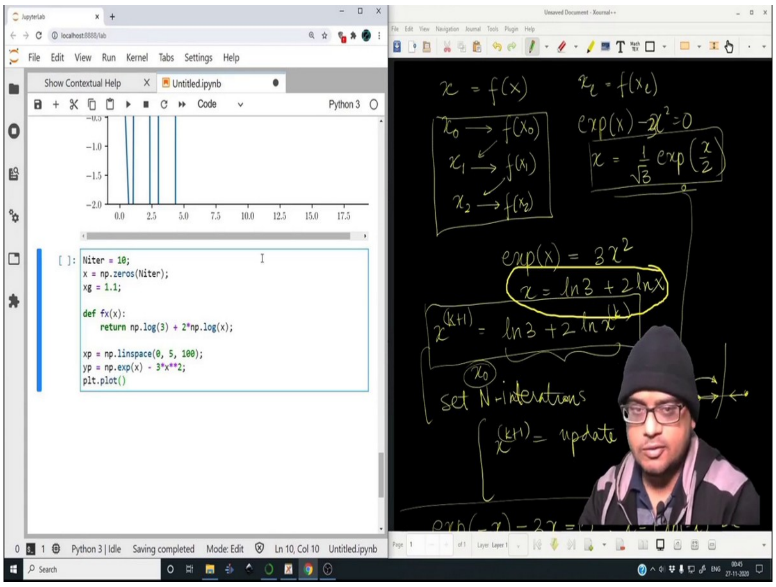
```
[29]: plt.plot(np.arange(0, Niter), x, np.arange(0, Niter), xx)
      plt.ylim([-2, 2])
```

```
[29]: (-2.0, 2.0)
```

$x^{(k+1)} = \ln 3 + 2\ln x^{(k)}$
x_0
set N-iterations
$x^{(k+1)} =$ update
$\exp(-x) - 3x = 0 \rightarrow x = \exp(-x) - 2x$
$x = \frac{1}{3}\exp(-x)$

So, what we see is the iterations for the function for this functional form it oscillates and then it converges, but for this functional form it oscillates and blows up to infinity. So, I mean it is quite obvious while programming a bit fast you are prone to making mistakes right that is ok. Try to pay attention; do not do the silly mistakes that I do. Let us now try to visualize how these iterations actually look like. So, let us go back to our initial example, this particular example ok.

(Refer Slide Time: 26:19)

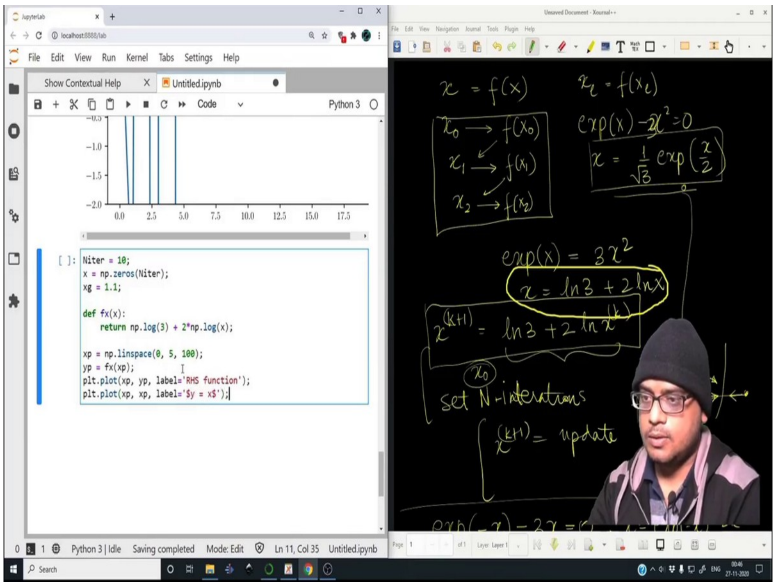


```
[ ]: Niter = 10;
     x = np.zeros(Niter);
     xg = 1.1;

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

     xp = np.linspace(0, 5, 100);
     yp = np.exp(x) - 3*x**2;
     plt.plot()
```

So, let us make the number of iterations 10 and let me again define the same thing. X= np . zeros(Niter); then x_guess = 1.1. I mean this can be change during the course of the program. Let me define fx as return np.log(3) + 2*np.log (x)ok. So, we have defined the return value.

So, let us define. So, let us define let us try to plot those 2 functions ok. Let us try to plot something. So, xp = np . linspace(0 ,5)say 100 points yp is that actual function. So, that function is e(x) – 3*x^2 alright.

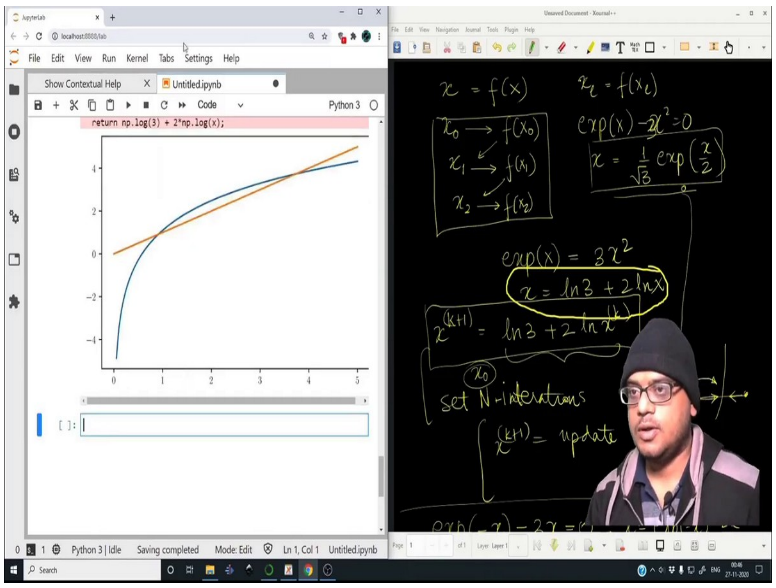
(Refer Slide Time: 27:58)



```
Niter = 10;
x = np.zeros(Niter);
xg = 1.1;

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

xp = np.linspace(0, 5, 100);
yp = fx(xp);
plt.plot(xp, yp, label='RHS function');
plt.plot(xp, xp, label='$y = x$');
```

So, let us plot this. So, plt . plot; actually this we do not want to plot this. We actually want to plot fx of xp. So, let us plot xp , yp and we will label it as the right hand side function and we will plot xp , xp and that is like the; I will explain what it is.

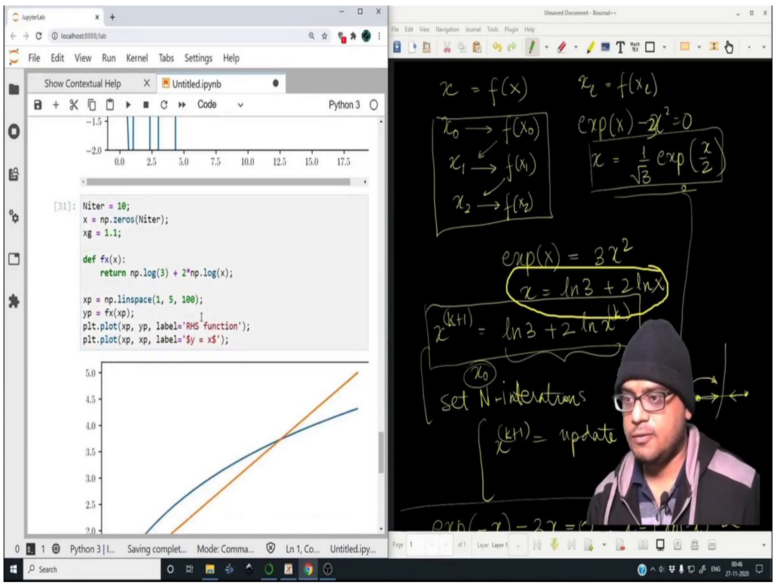
(Refer Slide Time: 28:34)



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

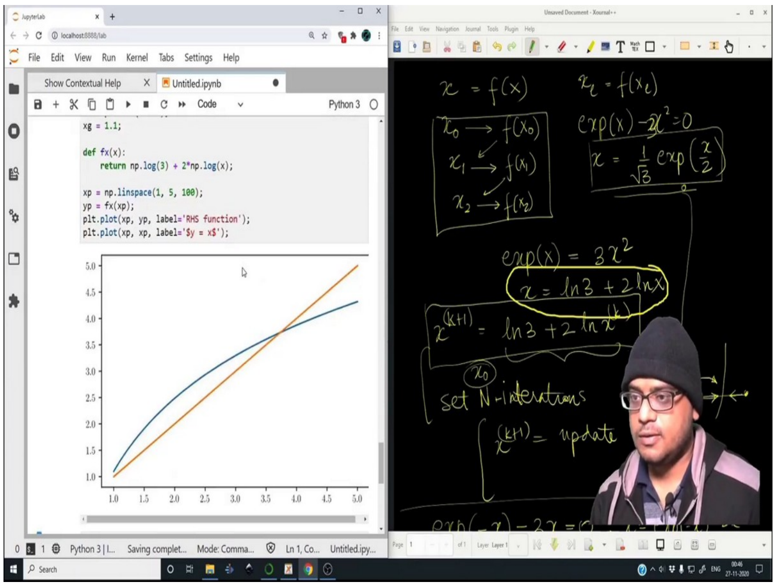
So, let us do this.

(Refer Slide Time: 28:42)



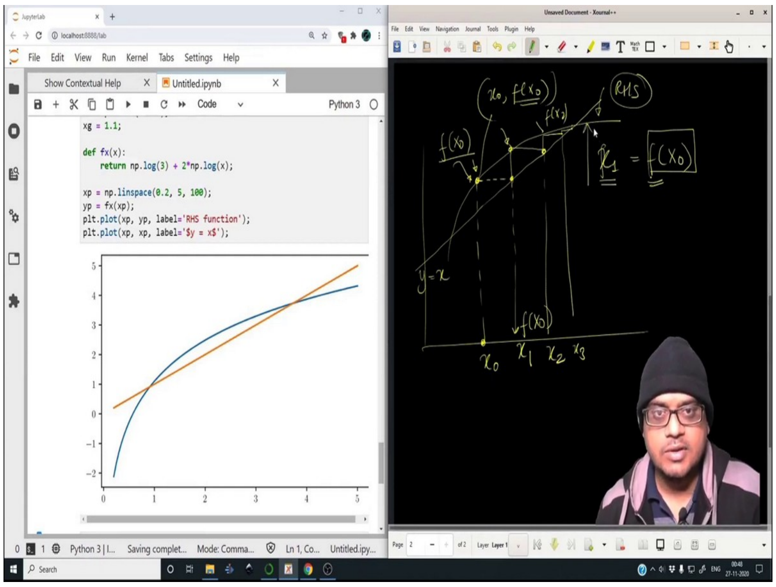
```
[31]: Niter = 10;
      x = np.zeros(Niter);
      xg = 1.1;

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

      xp = np.linspace(1, 5, 100);
      yp = fx(xp);
      plt.plot(xp, yp, label='RHS function');
      plt.plot(xp, xp, label='$y = x$');
```

(Refer Slide Time: 28:44)



```
xg = 1.1;

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

xp = np.linspace(1, 5, 100);
yp = fx(xp);
plt.plot(xp, yp, label='RHS function');
plt.plot(xp, xp, label='$y = x$');
```

(Refer Slide Time: 28:47)



```
xg = 1.1;

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='RHS function');
plt.plot(xp, xp, label='$y = x$');
```

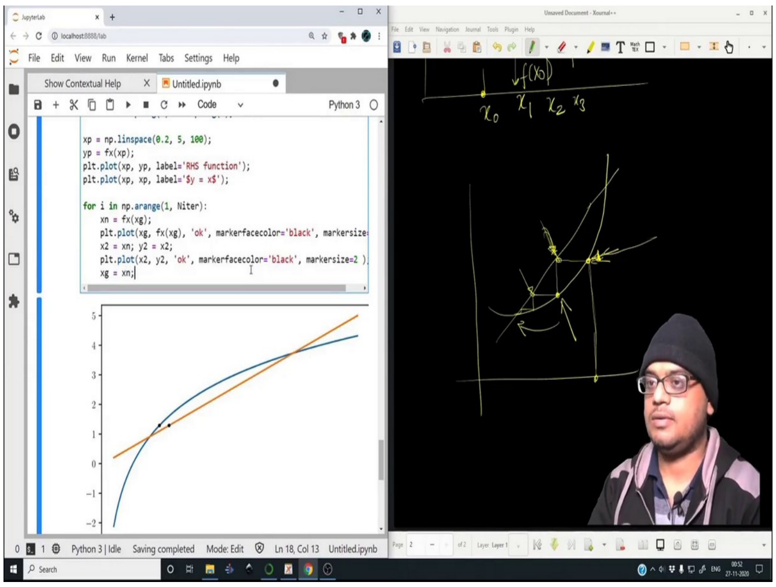
So, once we have plotted this ok; so, let me make it from 1 to 5, 0.2 to 5. So, what is really happening? Let me try to show it over here. So, basically we have one function which goes like this. This is the RHS function. This function is the function y equal to x alright.

So, suppose you make a guess value over here, so, then what is the procedure that we are following? So, x1 gets the value of f(x0), but let me split this ok. So, first thing you evaluate f(x0). So, this thing is the; so, f over here is the RHS function. So, this is f(x0).

So, this particular coordinate is x0 , f (x0). Then you assign it to a new value. So, this f (x0) gets a new assignment to a value of x. So, this particular y ok, this particular y value this y value is now mapped to a value of x. So, the way to map a value of y to x is to drop a projection onto the line y equal to x. By virtue of y being equal to x on this line, this particular x value has now received the y coordinate of this. So, this becomes f of x0 which is by definition x 1.

So, then we repeat. So, then this is f (x1) this becomes x2. So, this is f(x2) and this becomes x3. So, you see this is how the iterations are converging towards one particular root. Now, you may say that fine what if the curve does not look like this, the curve can actually look something like this as well right? No problem.

(Refer Slide Time: 31:02)



```
xp = np.linspace(0.2, 5, 100);
yp = fx(xp);
plt.plot(xp, yp, label='RHS function');
plt.plot(xp, xp, label='$y = x$');

for i in np.arange(1, Niter):
    xn = fx(xg);
    plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize
    x2 = xn; y2 = x2;
    plt.plot(x2, y2, 'ok', markerfacecolor='black', markersize=2 )
    xg = xn;
```

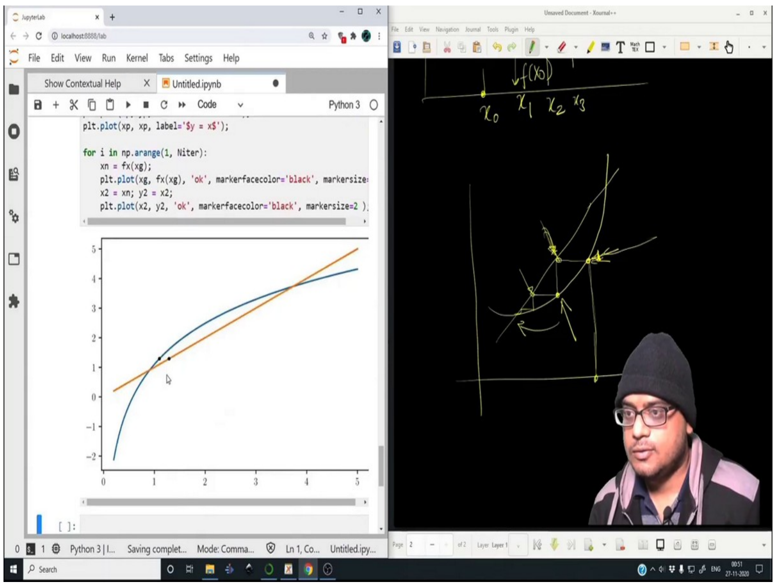
So, this is the initial guess. This is going to be f of x0 then you project it to the y equal to x line. Then you project it back to this, do this, do this, do this and you converge. So, if the curvature is reversed we see that it converges to the other root alright. So, let us try to encode this particular sequence of steps alright. So, for i in np . arrange(1, Niter), so, we will do xn is equal to fx (xg) guess alright.

Then we will do the plot. We will we this particular point. We will draw this particular point. So, we will do plt.plot and this particular point the x coordinate will be xg right, the y coordinate will be f of fx(xg), no problem. We will make it as a marker and let us set the markerfacecolor as black and let us put the markersize = 2, ok.

So, now with this new x we have to now map it to the other value. So, x2 let it take the value of xn alright because this is the new x alright and y2 will be equal to simply x2; plt .plot(x2 , y2) and the remaining string remains the same ok.

So, essentially this particular plot is this point, I am trying to plot this point ok. So, x2 is having the y coordinate of this that is why this particular thing should give us this point. So, the first plot is for this point second plot is for this point. Then when it runs into the loop again, this will become the first plot, this point will become the second plot ok. So, I mean let us run this and see ok.

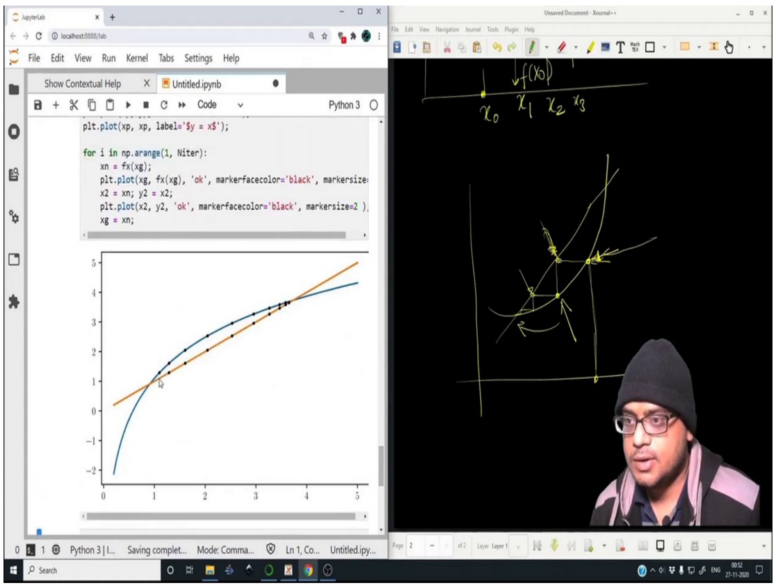
(Refer Slide Time: 33:55)



```
plt.plot(xp, xp, label='$y = x$');

for i in np.arange(1, Niter):
    xn = fx(xg);
    plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize
    x2 = xn; y2 = x2;
    plt.plot(x2, y2, 'ok', markerfacecolor='black', markersize=2 )
```

So, why is it giving me only two points? Because I have not updated anything. So, I have to update the value of xg also. So, xg should get the value of xn alright.

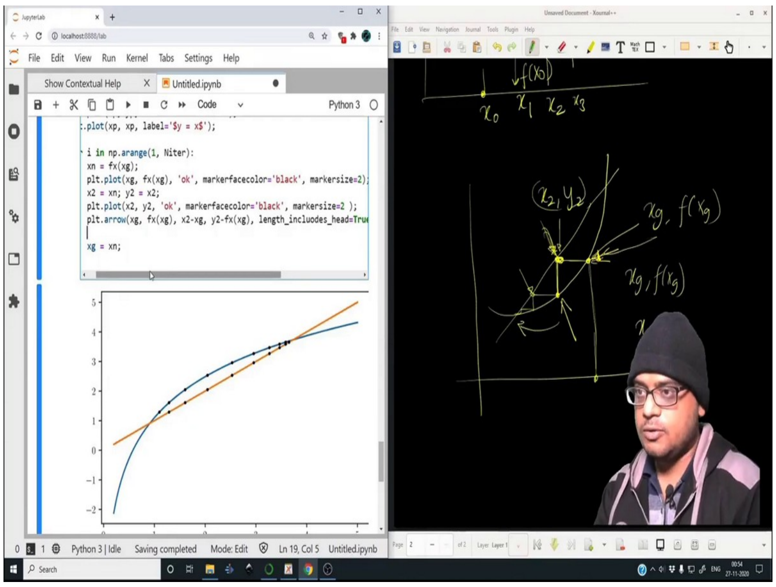
(Refer Slide Time: 34:12)



```
plt.plot(xp, xp, label='$y = x$');

for i in np.arange(1, Niter):
    xn = fx(xg);
    plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize
    x2 = xn; y2 = x2;
    plt.plot(x2, y2, 'ok', markerfacecolor='black', markersize=2 ),
    xg = xn;
```

There you go. So, the first guess is over here, it projects onto this. Then the second guess is over here, projects to this over here, over here. I mean it would be nice if we could join it by some arrows alright.

(Refer Slide Time: 34:32)

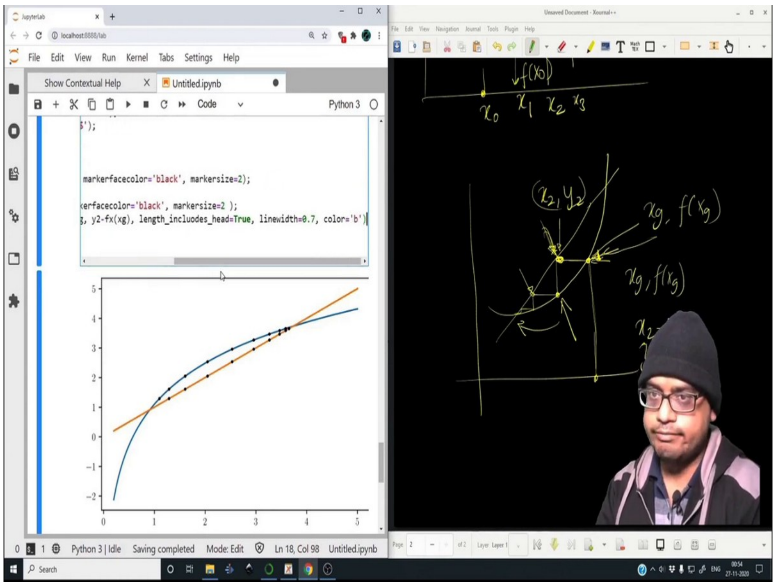


```
:.plot(xp, xp, label='$y = x$');

 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_incluodes_head=True

 xg = xn;
```

So, it is quite simple. So, we do plt.arrow. This has to be done prior to the assignment of xg equal to xn. So, plt.arrow. So, the first arrow has to be from xg fx(xg). So, the syntax for arrow is x,y,delta x,delta y. So, this coordinate over here is xg; f (xg) and this coordinate over here is x2 , y2. So, the first arrow has to be from xg, f(xg) and the delta x will be x2 - xg and delta y will be y2 – f(xg) alright.

So, let me do that. So, this will be x2 - xg , y2 – fx(xg) alright and there is a certain attribute of an arrow that is length includes head and we have to set this to True because the delta x has to also include the length of the arrow.

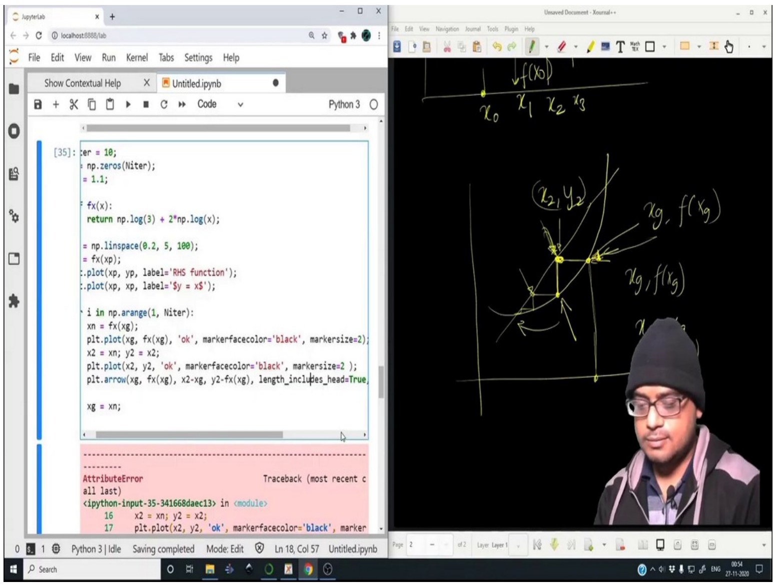
(Refer Slide Time: 35:56)



```
$');



 markerfacecolor='black', markersize=2);

kerfacecolor='black', markersize=2 );
g, y2-fx(xg), length_incluodes_head=True, linewidth=0.7, color='b')
```

So, let me set the line width of the arrow to 0.7 and let the color be equal to blue. Once this is done we can plot the second thing. So, the second arrow will be projection onto the curve again.

(Refer Slide Time: 36:32)

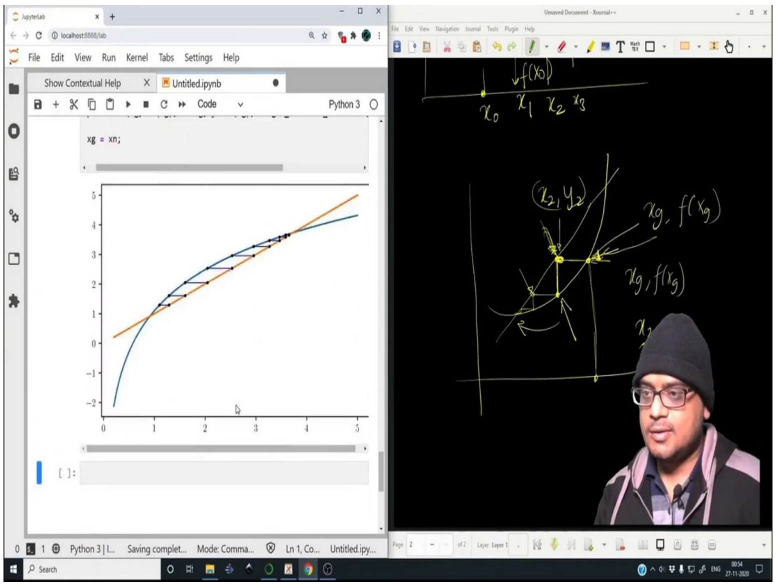


```
[35]: ter = 10;
      np.zeros(Niter);
      = 1.1;

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

      = np.linspace(0.2, 5, 100);
      = fx(xp);
      .plot(xp, yp, label='RHS function');
      .plot(xp, xp, label='$y = x$');

      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,

       xg = xn;
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent c
all last)
<ipython-input-35-341668daec13> in <module>
     16     x2 = xn; y2 = x2;
     17     plt.plot(x2, y2, 'ok', markerfacecolor='black', marker
```

So, this will be I mean let us have a look at how this looks ok. There is an error, this is typo this is small typo ok.

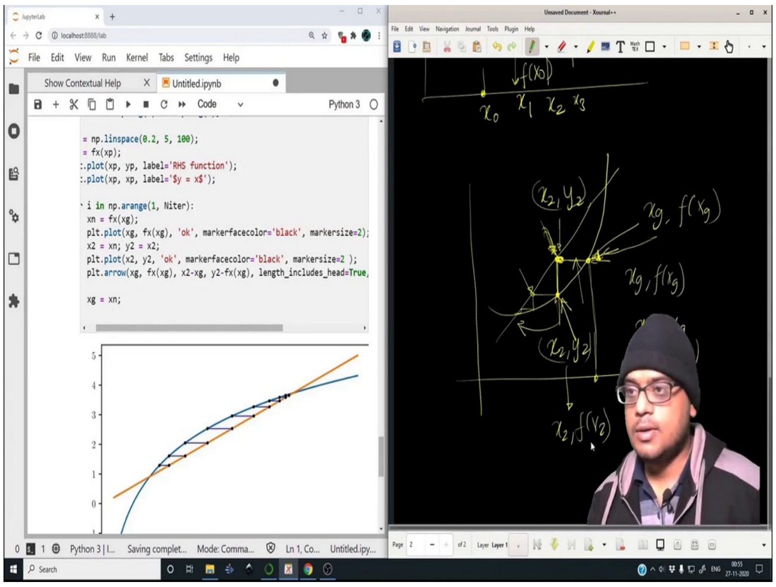
(Refer Slide Time: 36:44)



```
xg = xn;
```

So, now we have to we have drawn all the horizontal lines and we just need to focus on the vertical lines and that is quite easy.

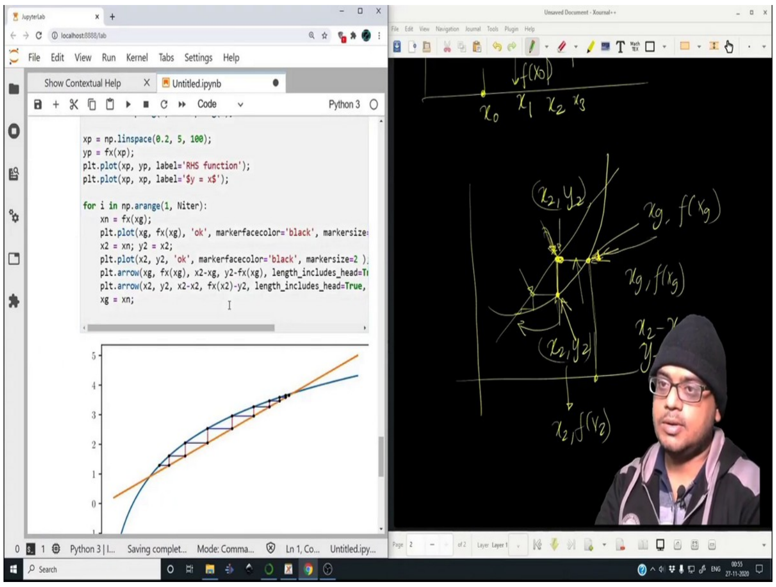
(Refer Slide Time: 36:59)



```
= np.linspace(0.2, 5, 100);
= fx(xp);
:.plot(xp, yp, label='RHS function');
:.plot(xp, xp, label='$y = x$');

 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,

 xg = xn;
```

Because this particular point this particular point is equivalent to drawing an arrow from x 2 y 2 to x 2 and f of x 2 that is it ok.

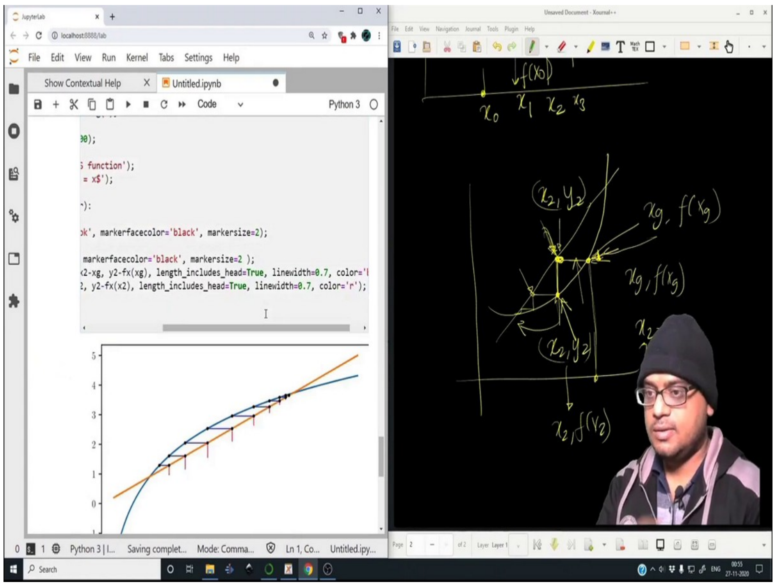
(Refer Slide Time: 37:18)



```
xp = np.linspace(0.2, 5, 100);
yp = fx(xp);
plt.plot(xp, yp, label='RHS function');
plt.plot(xp, xp, label='$y = x$');

for i in np.arange(1, Niter):
    xn = fx(xg);
    plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize
    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=T
    plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
    xg = xn;
```

So, we just need to copy this bit of code, paste it over here and change these values. So, x2 this will be y2 this will be x2 - x2 because it is a straight line and this will be y2 - f (x2) and let me make this line as a red.

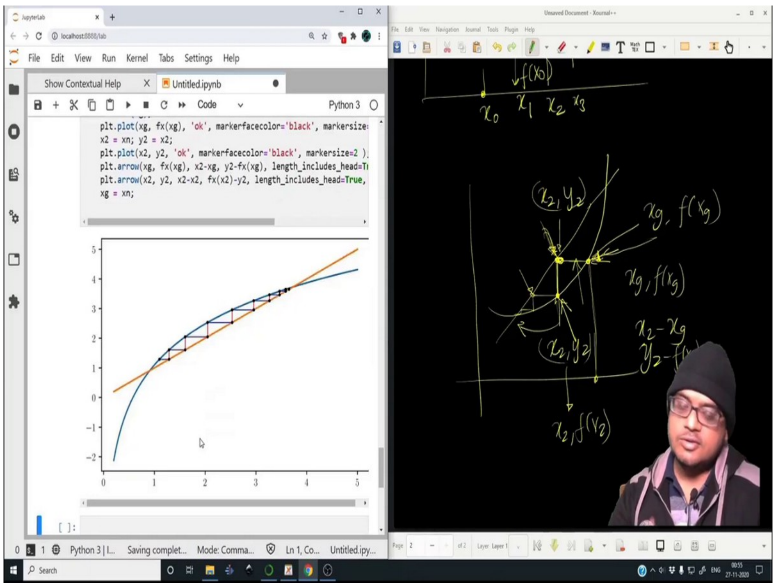
(Refer Slide Time: 37:40)



```
90);

 function');
 = x$');

):

ok', markerfacecolor='black', markersize=2);

 markerfacecolor='black', markersize=2 );
x2-xg, y2-fx(xg), length_includes_head=True, linewidth=0.7, color='
2, y2-fx(x2), length_includes_head=True, linewidth=0.7, color='r');
```

This is small error because ok, this has to be f (x - y2) ok. There you have it.

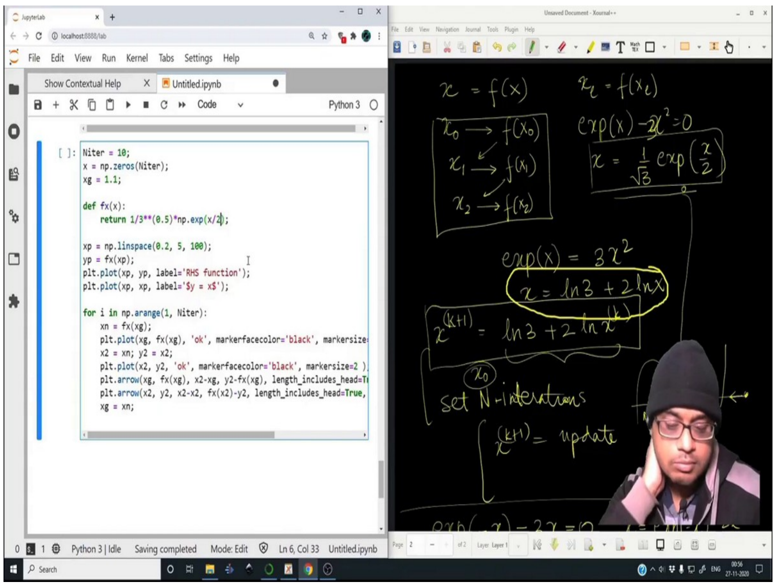
(Refer Slide Time: 37:52)



```
    plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize
    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=T
    plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
    xg = xn;
```

So, this shows how the iterations converge ok. So, now, because we have already done how the iterations converge let us change the function and see how the different choice of the iteration would look like.

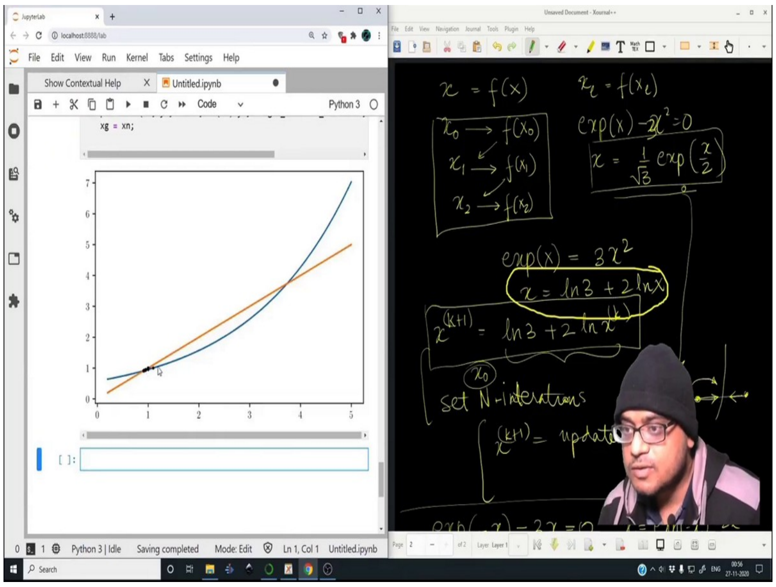
(Refer Slide Time: 38:20)



```
[ ]: Niter = 10;
     x = np.zeros(Niter);
     xg = 1.1;

     def fx(x):
         return 1/3**(0.5)*np.exp(x/2);

     xp = np.linspace(0.2, 5, 100);
     yp = fx(xp);
     plt.plot(xp, yp, label='RHS function');
     plt.plot(xp, xp, label='$y = x$');

     for i in np.arange(1, Niter):
         xn = fx(xg);
         plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize
         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=T
         plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
         xg = xn;
```

Let us go over here. So, the different choice was this alright. So, this was what? Sqrt(1 /3)*np .exp(x / 2), fine.

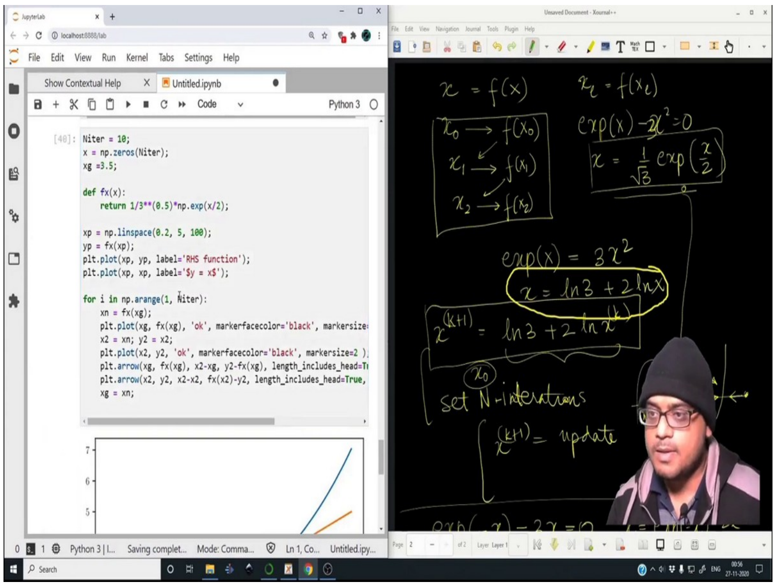
(Refer Slide Time: 38:30)



```
    xg = xn;
```

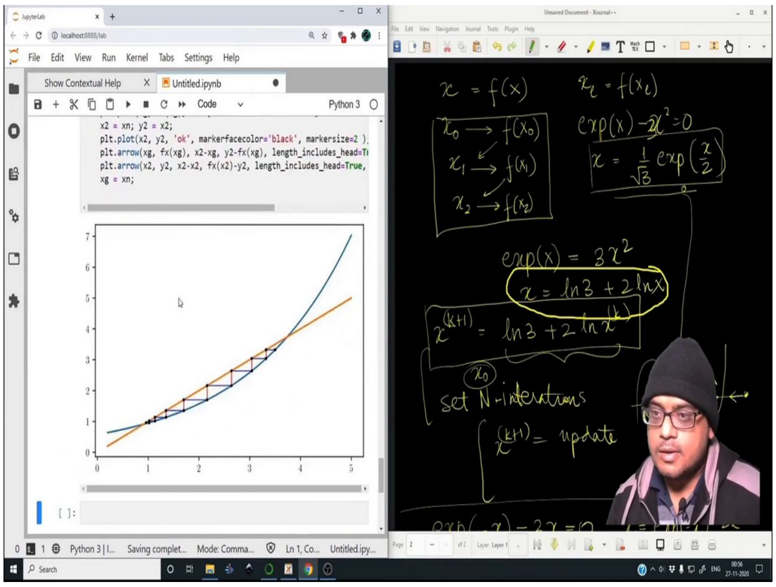
When we run this we see that everything converges over here. Let us take a different guess point. Let us take a guess point of something close to 3.5 for example, ok.

(Refer Slide Time: 38:40)



```
[40]: Niter = 10;
      x = np.zeros(Niter);
      xg =3.5;

      def fx(x):
          return 1/3**(0.5)*np.exp(x/2);

      xp = np.linspace(0.2, 5, 100);
      yp = fx(xp);
      plt.plot(xp, yp, label='RHS function');
      plt.plot(xp, xp, label='$y = x$');

      for i in np.arange(1, Niter):
          xn = fx(xg);
          plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize
          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=Tr
          plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
          xg = xn;
```

(Refer Slide Time: 38:42)



```
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=Tı
plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
xg = xn;
```

$x = f(x)$ $\quad x_i = f(x_i)$

$x_0 \longrightarrow f(x_0)$, $x_1 \longrightarrow f(x_1)$, $x_2 \longrightarrow f(x_2)$

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

$x = \frac{1}{\sqrt{3}} \exp\left(\frac{x}{2}\right)$

$\exp(x) = 3x^2$

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

$x^{(k+1)} = \ln 3 + 2 \ln x^{(k)}$

set N-iterations

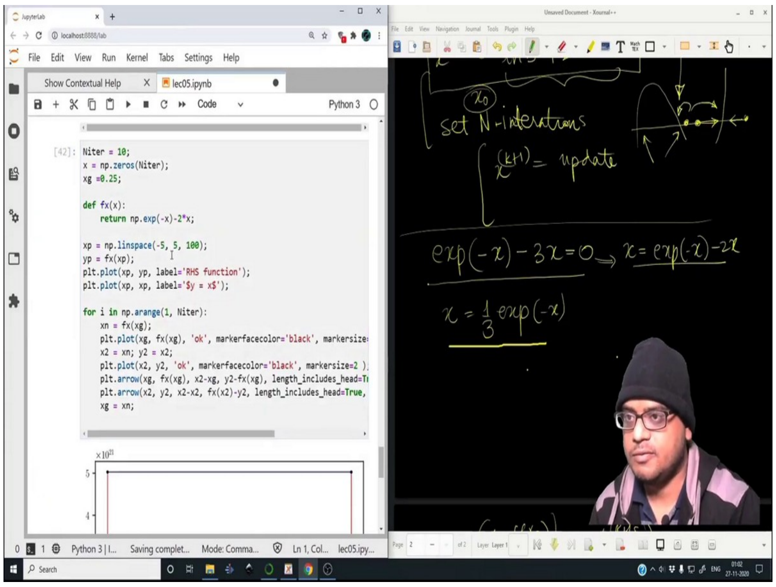
$x^{(k+1)} =$ update

So, when we start at 3.5 we see that it converges towards the root at the bottom and in the accompanying notes which you will find on my website, I will explain mathematically why this convergence happens in this way.

Well, this is not a course of an exposition on the mathematics of it and I am hoping that you would have seen the other NPTEL courses which deal with this kind of mathematics. I am trying to show you how you can make the best out of the tools available to you to really understand the process that goes on behind the machinery.

Well, we have seen the two cases where they seem to converge to a certain root. We did see a certain case where convergence was not there. In fact, everything diverged in particular this was the case in point. Let us have a look at how that particular set of iterations actually looks like. I mean what causes it to diverge and that should be the question that comes to your mind.

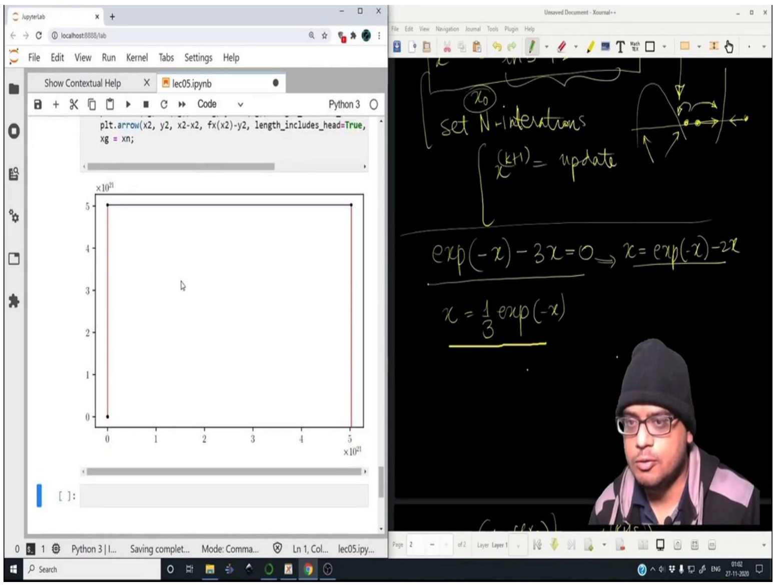
(Refer Slide Time: 39:56)



```
[42]: Niter = 10;
      x = np.zeros(Niter);
      xg =0.25;

      def fx(x):
          return np.exp(-x)-2*x;

      xp = np.linspace(-5, 5, 100);
      yp = fx(xp);
      plt.plot(xp, yp, label='RHS function');
      plt.plot(xp, xp, label='$y = x$');

      for i in np.arange(1, Niter):
          xn = fx(xg);
          plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize
          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=Tr
          plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
          xg = xn;
```

So, let us see let us replace this by np.exp(- x) - 2x. Let us take the guess value of 0.25.

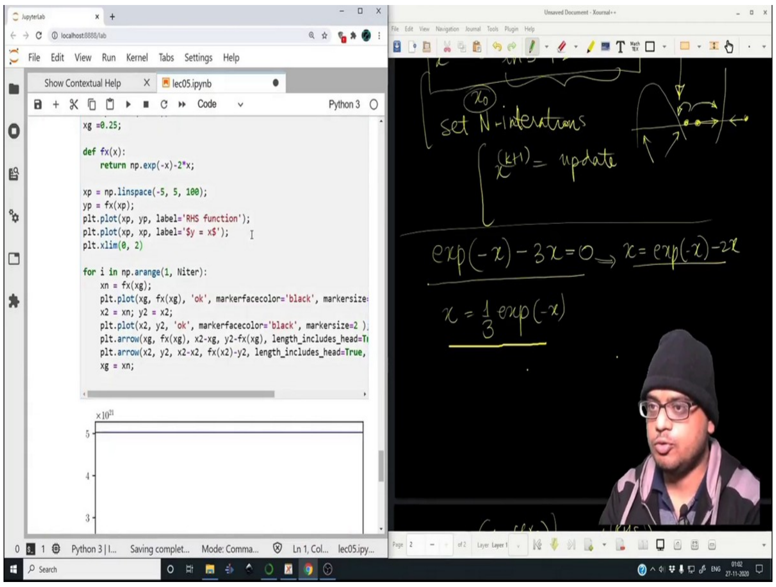
(Refer Slide Time: 40:08)



```
          plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
          xg = xn;
```

Let us run this ok. Something has happened. This seemed to be a very large; let me take it from - 5 to 5. Let us try to locate where the things have started somewhere over here.

(Refer Slide Time: 40:31)

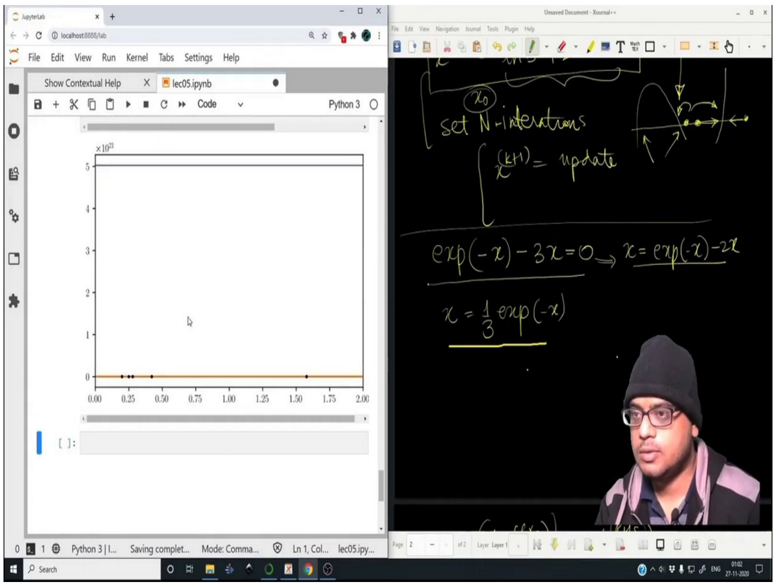


```
xg =0.25;

def fx(x):
    return np.exp(-x)-2*x;

xp = np.linspace(-5, 5, 100);
yp = fx(xp);
plt.plot(xp, yp, label='RHS function');
plt.plot(xp, xp, label='$y = x$');
plt.xlim(0, 2)

for i in np.arange(1, Niter):
    xn = fx(xg);
    plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize
    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=T
    plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
    xg = xn;
```

set N-iterations
x_0
$x^{(k+1)} = $ update
$\exp(-x) - 3x = 0 \Rightarrow x = \exp(-x) - 2x$
$x = \frac{1}{3}\exp(-x)$

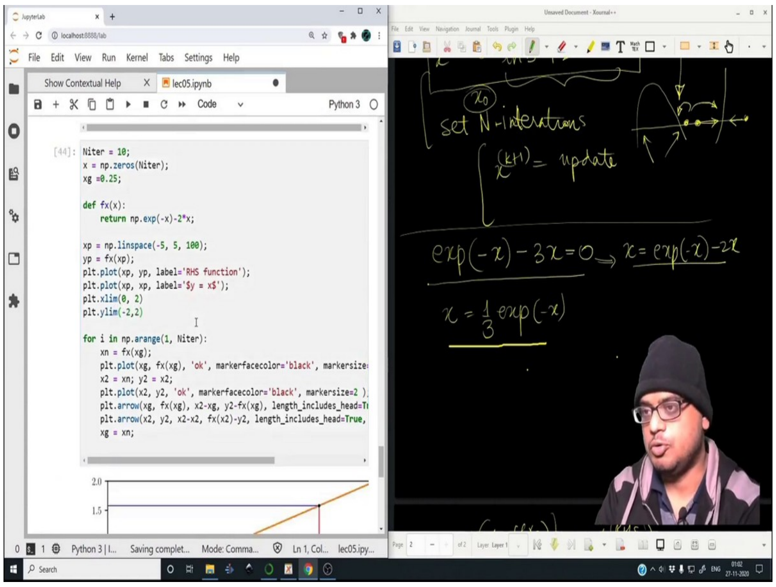
So, let us change the limit. Let us keep it confined to 0 to 2 and let us keep the y limit confined also between - 2 to 2 ok.

(Refer Slide Time: 40:37)



set N-iterations
x_0
$x^{(k+1)} = $ update
$\exp(-x) - 3x = 0 \Rightarrow x = \exp(-x) - 2x$
$x = \frac{1}{3}\exp(-x)$

(Refer Slide Time: 40:41)



```
[44]: Niter = 10;
      x = np.zeros(Niter);
      xg =0.25;

      def fx(x):
          return np.exp(-x)-2*x;

      xp = np.linspace(-5, 5, 100);
      yp = fx(xp);
      plt.plot(xp, yp, label='RHS function');
      plt.plot(xp, xp, label='$y = x$');
      plt.xlim(0, 2)
      plt.ylim(-2,2)

      for i in np.arange(1, Niter):
          xn = fx(xg);
          plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize
          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=T
          plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
          xg = xn;
```

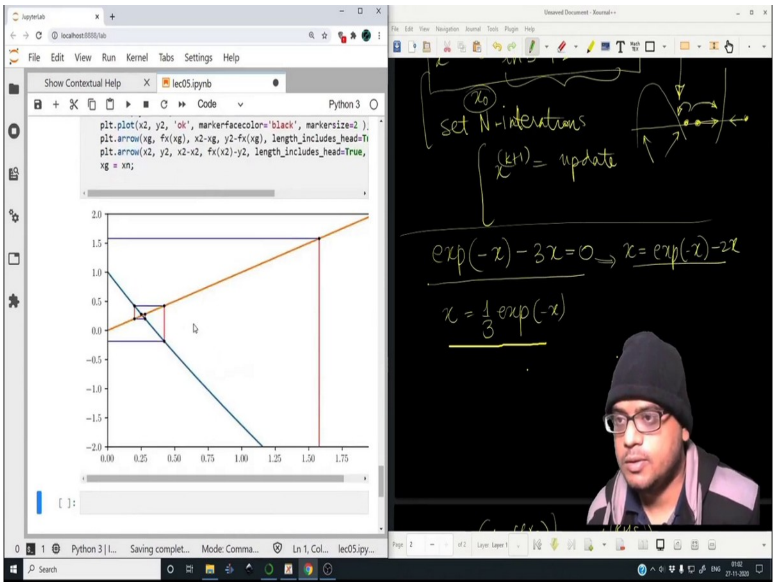
set N-interations x_0

$x^{(k+1)} =$ update

$\exp(-x) - 3x = 0 \Rightarrow x = \exp(-x) - 2x$

$x = \frac{1}{3}\exp(-x)$

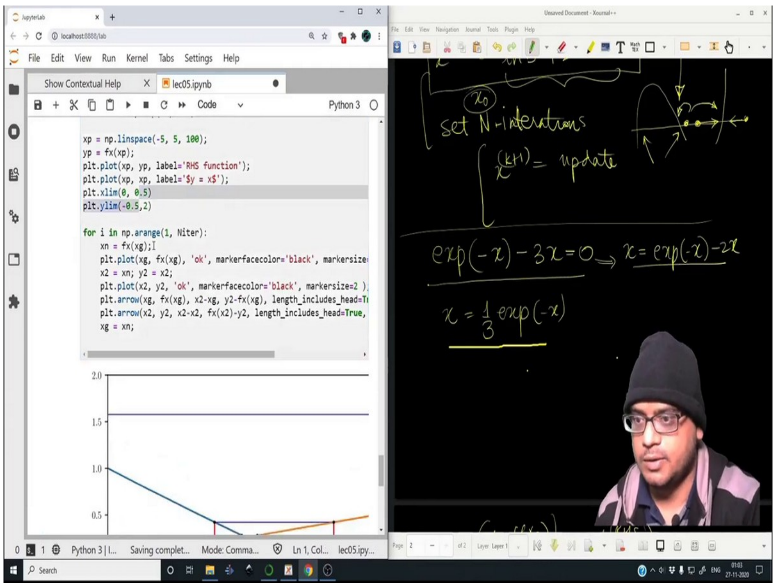
(Refer Slide Time: 40:47)



```
          plt.plot(x2, y2, 'ok', markerfacecolor='black', markersize=2 )
          plt.arrow(xg, fx(xg), x2-xg, y2-fx(xg), length_includes_head=T
          plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
          xg = xn;
```

There we see the culprit. In fact, let us zoom into that further and because I am using JupyterLab like I do not have the interactive zoom at this moment, but you can look it up on the internet how to do it right. It will not be required for explaining this particular phenomenon; so, alright ok.

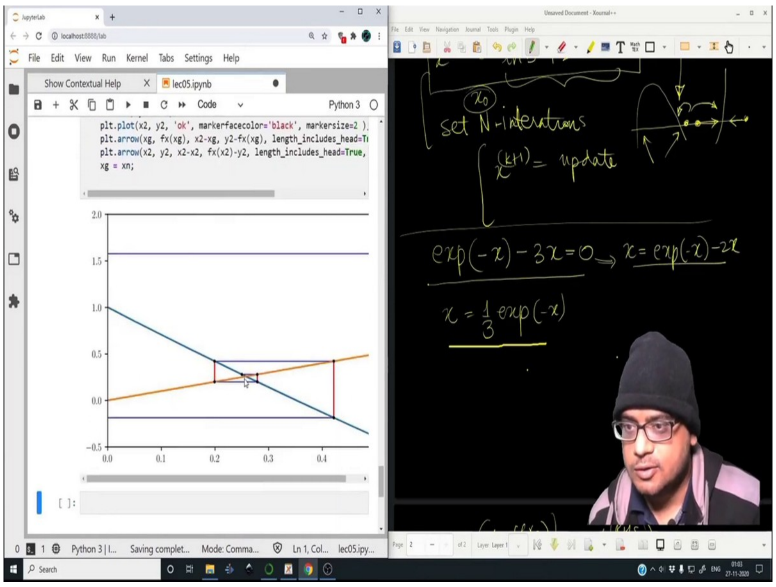
(Refer Slide Time: 41:09)



```
xp = np.linspace(-5, 5, 100);
yp = fx(xp);
plt.plot(xp, yp, label='RHS function');
plt.plot(xp, xp, label='$y = x$');
plt.xlim(0, 0.5)
plt.ylim(-0.5,2)

for i in np.arange(1, Niter):
    xn = fx(xg);
    plt.plot(xg, fx(xg), 'ok', markerfacecolor='black', markersize
    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=T
    plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
    xg = xn;
```

set N-iterations
x_0
$x^{(k+1)} =$ update
$\exp(-x) - 3x = 0 \rightarrow x = \exp(-x) - 2x$
$x = \frac{1}{3}\exp(-x)$

(Refer Slide Time: 41:12)

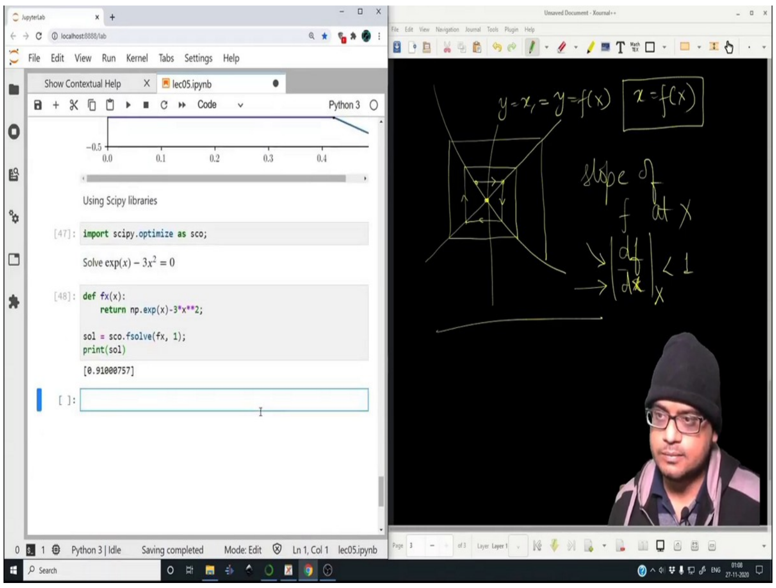


```
    plt.plot(x2, y2, 'ok', markerfacecolor='black', markersize=2 )
    plt.arrow(xg, fx(xg), x2-xg, y2-fx(xg), length_includes_head=T
    plt.arrow(x2, y2, x2-x2, fx(x2)-y2, length_includes_head=True,
    xg = xn;
```

set N-iterations
x_0
$x^{(k+1)} =$ update
$\exp(-x) - 3x = 0 \rightarrow x = \exp(-x) - 2x$
$x = \frac{1}{3}\exp(-x)$

So, this is the guess value. So, the guess value is 0.25. So, it starts over here then it maps on to the x equal to y line then it maps on to the curve maps onto the x equal to y line and it seems that these sequence of iterations are going out of hand, they do not seem to converge. Let me try to pictorially motivate this ok.

(Refer Slide Time: 41:43)


Using Scipy libraries

```
[47]: import scipy.optimize as sco;
```

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

```
[48]: def fx(x):
          return np.exp(x)-3*x**2;

      sol = sco.fsolve(fx, 1);
      print(sol)

[0.91000757]
```

So, we have x = y line like this and this is the curve. So, this is the initial guess, it goes over here then like this and like this, like this so on and so forth. Well, I have drawn a much more compact version of what is actually going on and this is the root. This is the point where y equal to x and y equal to f (x) they converge. Basically, when they become equal, it means, x = f (x).

So, this is actually the root that we are aiming at, but even though we are guessing very close to the root we see that the successive iterations are going out of hand that diverging very quickly to infinity and the reason why this actually happens. And I will put it upon the on the on the website as a side note is the slope of f at the fixed point. So, the fixed point I mean the root.

So, df/dx at the root must be less than 1, the magnitude of the slope must be less than 1. In this particular case the magnitude of the root is obviously, quite large and for the other two cases, for the other two cases it is converging to the root where the slope of the blue curve is less than the slope of the orange curve. Here it is less while in this case here it is less.

So, it is converging towards that root and in case you do not have the condition in the curve where the type of discretization that you do that is this particular form of discretization that you do. If it is not able to satisfy this condition that the slope at the root has to be less than 1; the magnitude of the slope has to be less than 1.

If we are not able to achieve this; well, achieving this a priori is not easy because you do not know where the root is, but still graphically you should be able to figure out when this will converge or whether it will converge or not ok.

So, the accompanying mathematics is not extremely difficult, it makes use of a Taylor series expansion around the root. So, now, let us proceed. Let us look at how we can make use or make Python do all our work. Obviously, these things are quite important for engineering, for mathematics, in general for biology. If you are trying to study non-linear equations like the Michaelis-Menten reaction often you need to find out what will be the equilibrium product. So, things like this you need to find out.

So, there are libraries to do that. So, let us go over here. So, most of the things are sort of in the sub module called as optimize in Scipy. And the reason why it is in the sub model Scipy . optimize is because these root finding problems can be sort of recast into a minimization problem.

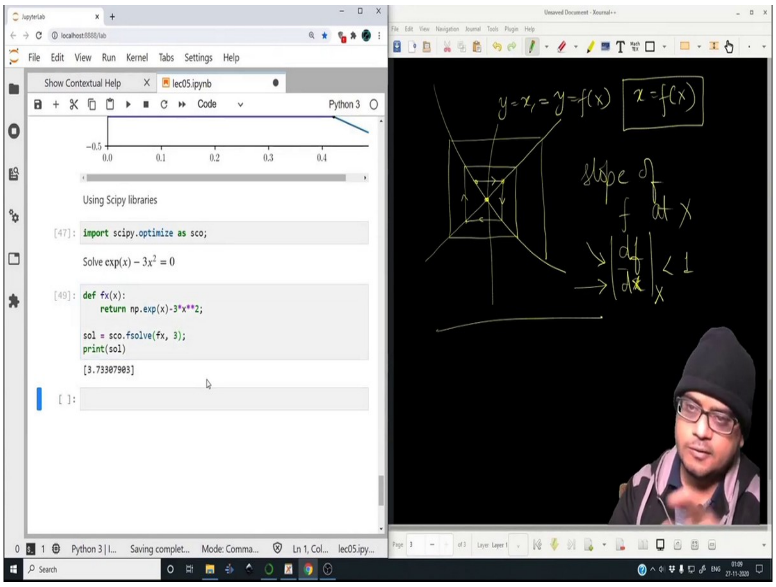
And, once you recast things into minimization problem you can sort of find out the minima if it is a convex problem you can find out the minima with the help of very sophisticated algorithms that is why it all these things lie in the optimize library.

So, we need to import the optimize library. So, it will be scipy . optimize. Let us import it as sco. Let us define ok, let me keep this in a separate cell. Let me define fx (x) as the following; return np .exp(x) – 3*x ^2, let me define it like this. So, this is the function that we have solved so far. Essentially this is e^(x) – 3x^2= 0.

So, this is the return value. So, now, we will say sol; the solution is sco . fsolve we will get the function handle. So, this is called as giving the function handle. You are telling fsolve that this is the function handle to from which you have to do all the iterations.

So, it does all the iterations internally and then you have to provide the guess value as well. So, let us give the guess value of 1 and in the end let us print sol, let us see what happens ok. So, it is giving us the converged value of 0.91 and if you recall that was indeed one of the roots.

(Refer Slide Time: 47:11)



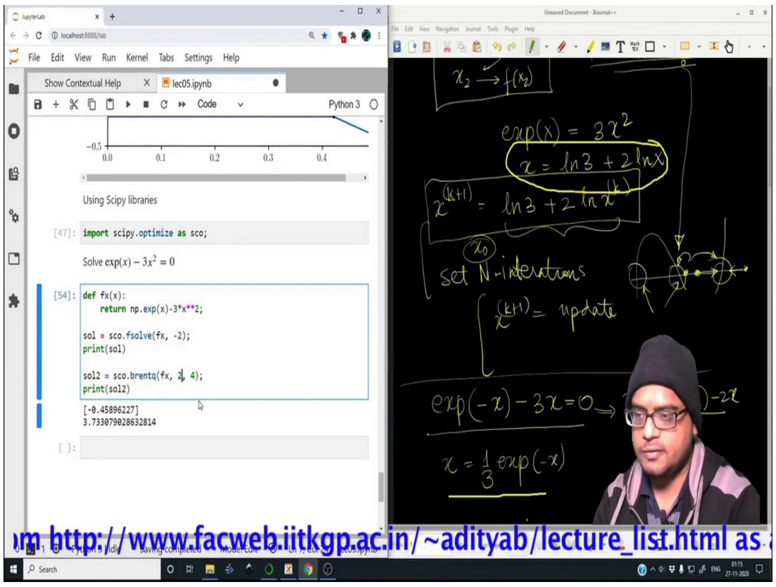
Using Scipy libraries

```
[47]: import scipy.optimize as sco;
```

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

```
[49]: def fx(x):
          return np.exp(x)-3*x**2;

      sol = sco.fsolve(fx, 3);
      print(sol)

      [3.73307903]
```

(Refer Slide Time: 47:17)



Using Scipy libraries

```
[47]: import scipy.optimize as sco;
```

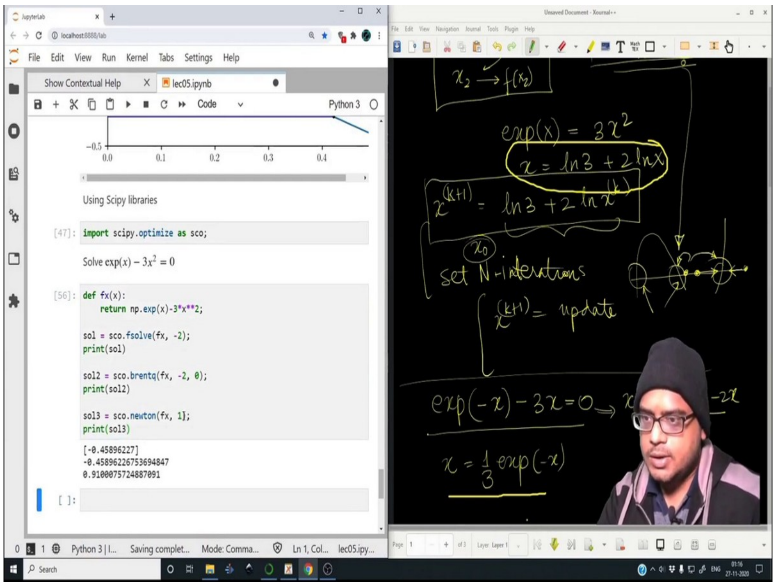
Solve $\exp(x) - 3x^2 = 0$

```
[54]: def fx(x):
          return np.exp(x)-3*x**2;

      sol = sco.fsolve(fx, -2);
      print(sol)

      sol2 = sco.brentq(fx, 2, 4);
      print(sol2)

      [-0.45896227]
      3.733079028632814
```

In fact, when we choose a different initial value we converge to the other root. In fact, if we choose another initial value we converge to the third root. If you recall there were three roots of this entire system. There were three roots. One was over here, one was over here and one was. So, depending on where you give the initial value it will converge.

So, it tries to find out whether you have to take a positive slope or a negative slope it has to keep the slope bounded. There is a bunch of algorithms that go on behind the scenes but, this is how you can minimally solve a non-linear algebraic equation. There are many other ways of doing it. There is also a method called as Brent Q. Essentially, it is a bracketing method in which you need to specify the domain inside which you expect the root to exist.

(Refer Slide Time: 48:20)


Using Scipy libraries

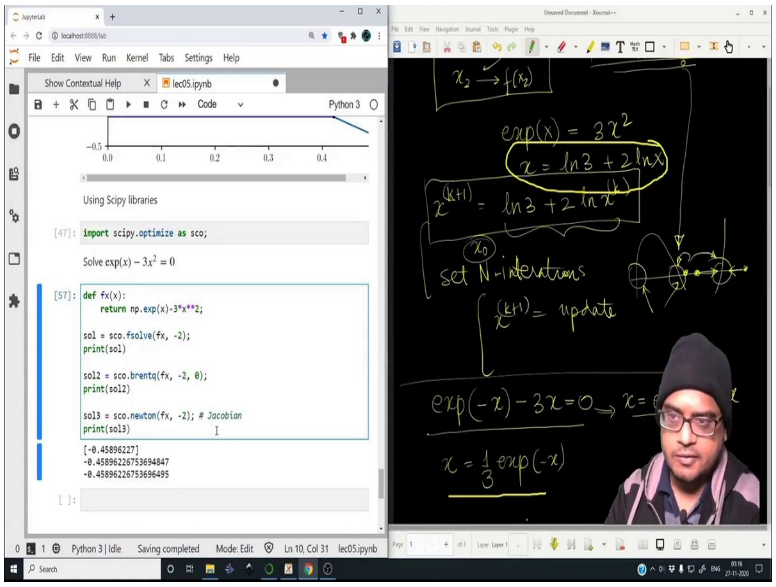
```
[47]: import scipy.optimize as sco;
```

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

```
[56]: def fx(x):
          return np.exp(x)-3*x**2;

      sol = sco.fsolve(fx, -2);
      print(sol)

      sol2 = sco.brentq(fx, -2, 0);
      print(sol2)

      sol3 = sco.newton(fx, 1);
      print(sol3)

[-0.45896227]
-0.45896226753694847
0.9100075724887091
```

So, if we give the initial; so, depending on the bracket it will give you the root that it encounters. It is a very useful method of finding out the roots. Lastly, I want to show you the famous method I mean it is called as the Newton iterations. So, the way to do it is sco . newton we have to give the function and 1. Let us print sol3 ok.

(Refer Slide Time: 49:03)


Using Scipy libraries

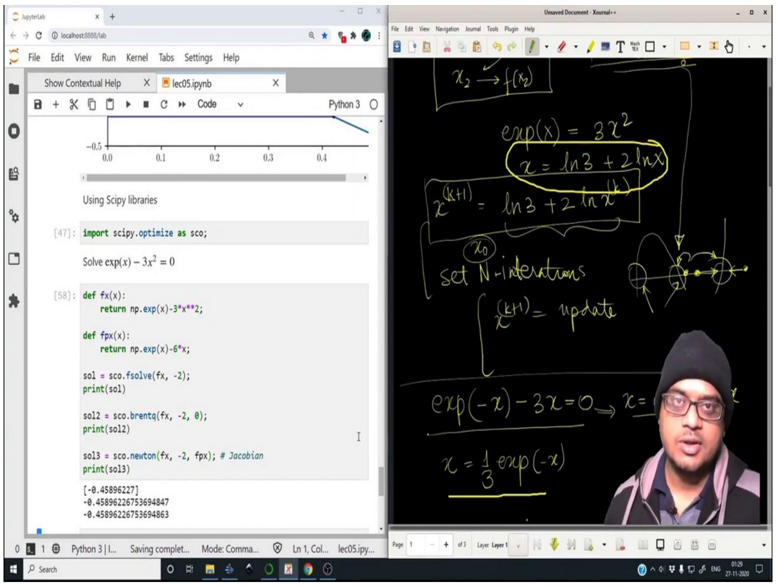
```
[47]: import scipy.optimize as sco;
```

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

```
[57]: def fx(x):
          return np.exp(x)-3*x**2;

      sol = sco.fsolve(fx, -2);
      print(sol)

      sol2 = sco.brentq(fx, -2, 0);
      print(sol2)

      sol3 = sco.newton(fx, -2); # Jacobian
      print(sol3)

[-0.45896227]
-0.45896226753694847
-0.45896226753696495
```

And depending on the initial guess, we have a different solution and typically for the Newton-Raphson method we have to also provide the Jacobian, alright. So, essentially it is the derivative analytical derivative of this particular function.

(Refer Slide Time: 49:30)



Using Scipy libraries

```
[47]: import scipy.optimize as sco;
```

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

```
[58]: def fx(x):
          return np.exp(x)-3*x**2;

      def fpx(x):
          return np.exp(x)-6*x;

      sol = sco.fsolve(fx, -2);
      print(sol)

      sol2 = sco.brentq(fx, -2, 0);
      print(sol2)

      sol3 = sco.newton(fx, -2, fpx); # Jacobian
      print(sol3)
```

```
[-0.45896227]
-0.45896226753694847
-0.45896226753694863
```

$x_2 \to f(x_2)$
$\exp(x) = 3x^2$
$x = \ln 3 + 2\ln x$
$x^{(k+1)} = \ln 3 + 2\ln x^{(k)}$
set N-iterations
x_0
$x^{(k+1)} =$ update
$\exp(-x) - 3x = 0 \Rightarrow x =$
$x = \frac{1}{3}\exp(-x)$

So, fpx is the derivative of x, the analytic expression for the derivative of f (x) and we will pass it to the function as well. So, it converges much faster. In the second part of this particular topic, we will be studying systems of non-linear equations and basins of attraction for the Newton-Raphson method that is points on the complex plane which converge to a specified roots. And we will see certain non intuitive observations in the basins of attraction. So, do join me in the next lecture.

I will see you again. Bye.