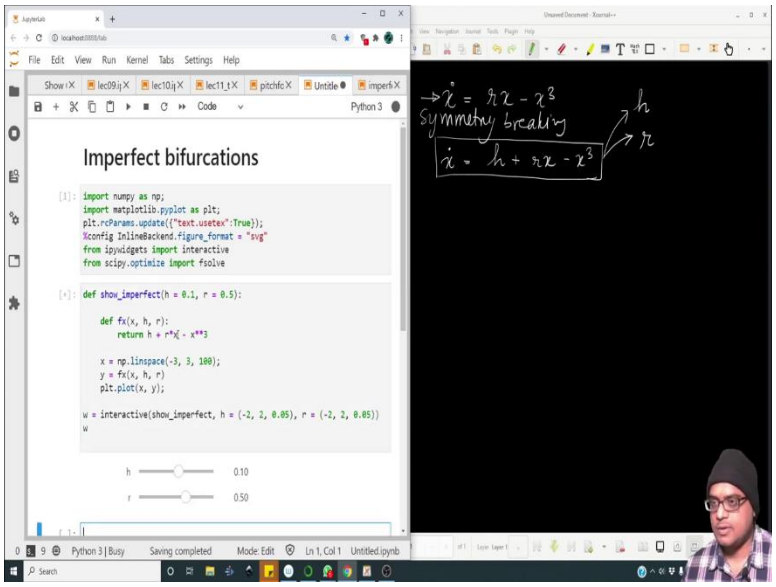
Tools in Scientific Computing

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

Lecture – 13
Imperfect bifurcations and catastrophies

(Refer Slide Time: 00:27)



Imperfect bifurcations

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

```
def show_imperfect(h = 0.1, r = 0.5):

    def fx(x, h, r):
        return h + r*x - x**3

    x = np.linspace(-3, 3, 100);
    y = fx(x, h, r)
    plt.plot(x, y);

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h 0.10

r 0.50

$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

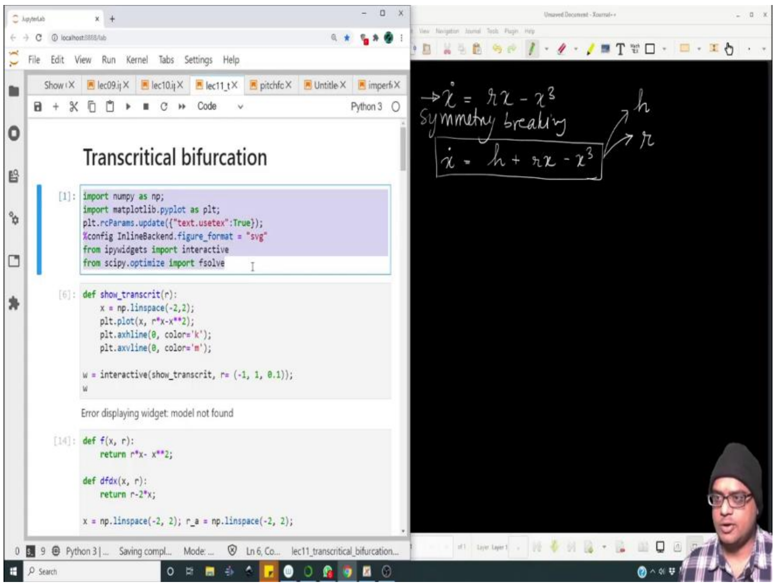
$$\boxed{\dot{x} = h + rx - x^3} \rightarrow h, r$$

Hello everyone. In this lecture we are going to study Imperfect bifurcations. So, far we have studied the pitchfork bifurcation in which we had a vector flow which looked something like this. You can it had a very distinct symmetry where x and -x they are evolving as per the same function. Now often in many questions there is something called a symmetry breaking.

And a parameter which can break the symmetry of this equation is a simple constant term something as simple as this ok. So, given this prototype equation there are now two control parameters.

First control parameter is h and the second control parameter is r. So, upon changing these two we can have various characteristics of the equation. So, when h = 0 we do reduce to a normal mode of a pitchfork bifurcation. So, let us first look at how this function actually looks like.

(Refer Slide Time: 01:51)



Transcritical bifurcation

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

```
[6]: def show_transcrit(r):
         x = np.linspace(-2,2);
         plt.plot(x, r*x-x**2);
         plt.axhline(0, color='k');
         plt.axvline(0, color='m');

     w = interactive(show_transcrit, r= (-1, 1, 0.1));
     w
```

```
Error displaying widget: model not found
```

```
[14]: def f(x, r):
          return r*x- x**2;

      def dfdx(x, r):
          return r-2*x;

      x = np.linspace(-2, 2); r_a = np.linspace(-2, 2);
```

$\rightarrow \dot{x} = rx - x^3$

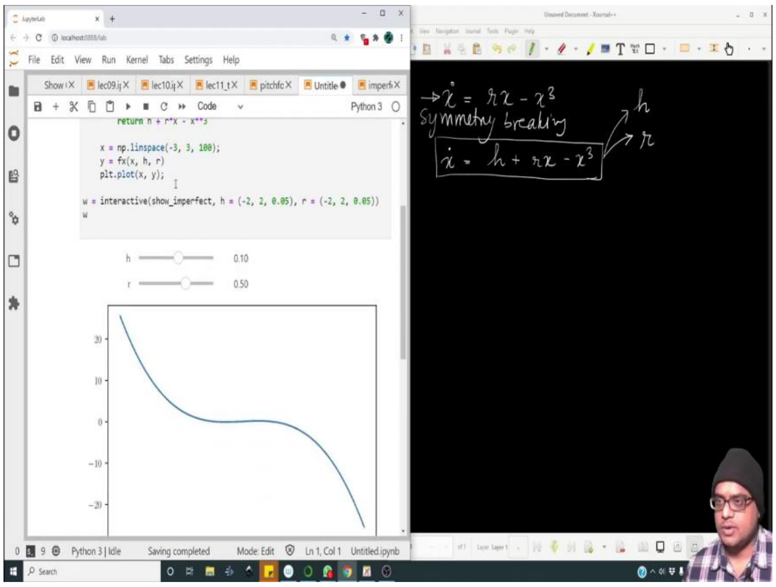
Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ $\nearrow h$ $\nearrow r$

So, let me copy this bit of code. We will need all of this, ok. So, now, let us define. So, let us define fx as a function of (x, h, r) and it will return $h + rx - x^3$. Let us wrap everything inside another function which will take inputs as h let me take the default value of 0.1 and r which will take a default value of 0.5 for example.

Let me put all this inside the function block. So, now, x = np.linspace(-3,3, 100) 100 points; y = fx(x,h,r); plt.plot(x,y). Let me define an interactive widget. So, w = interactive(show_imperfect, h = (-2 ,2, 0.05), r = (-2,2, 0.05)). Let us display the widget ok.

(Refer Slide Time: 03:41)



```
        return h + r*x - x**3

    x = np.linspace(-3, 3, 100);
    y = fx(x, h, r)
    plt.plot(x, y);

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h 0.10

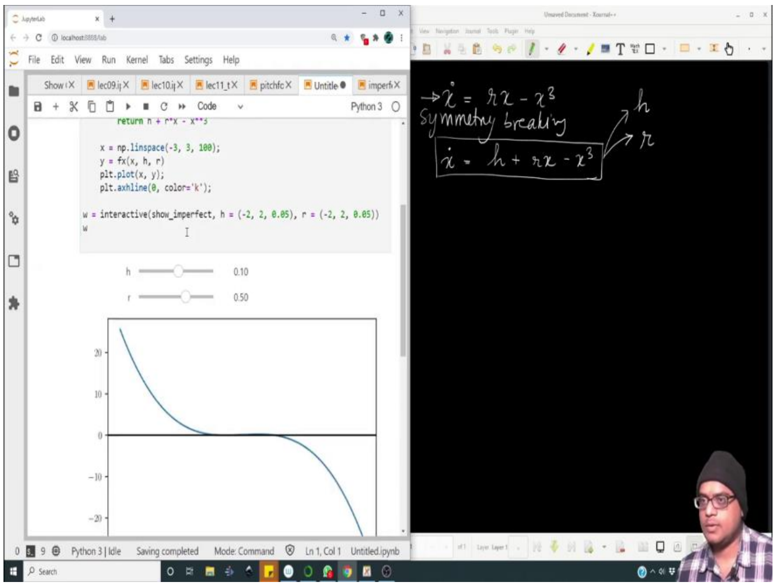
r 0.50

$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ → h, → r

(Refer Slide Time: 03:44)



```
        return h + r*x - x**3

    x = np.linspace(-3, 3, 100);
    y = fx(x, h, r)
    plt.plot(x, y);
    plt.axhline(0, color='k');

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h 0.10

r 0.50

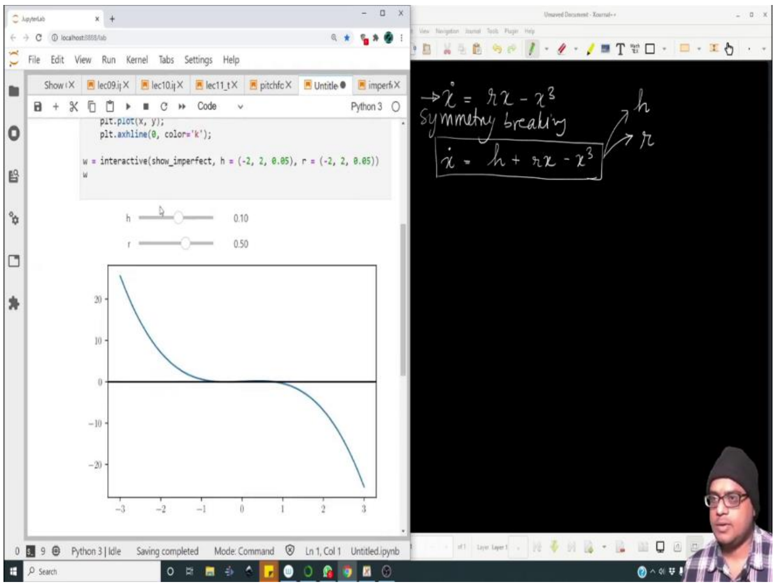
$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ → h, → r

So, after running this let me also draw the x axis of plt.axvline sorry plt.axhline(0, ‘k’), let me make it something like this sorry ok.

(Refer Slide Time: 04:08)



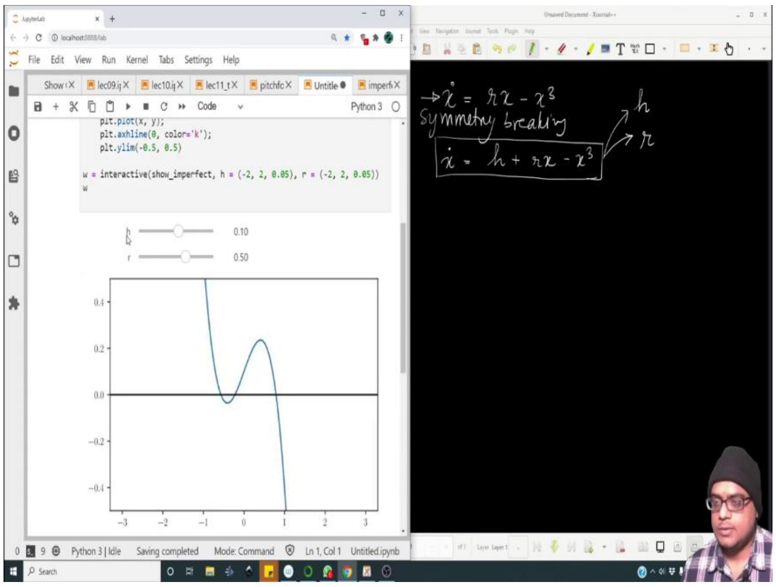
```
    plt.plot(x, y);
    plt.axhline(0, color='k');

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ $\nearrow h$ $\nearrow r$

(Refer Slide Time: 04:11)



```
    plt.plot(x, y);
    plt.axhline(0, color='k');
    plt.ylim(-0.5, 0.5)

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

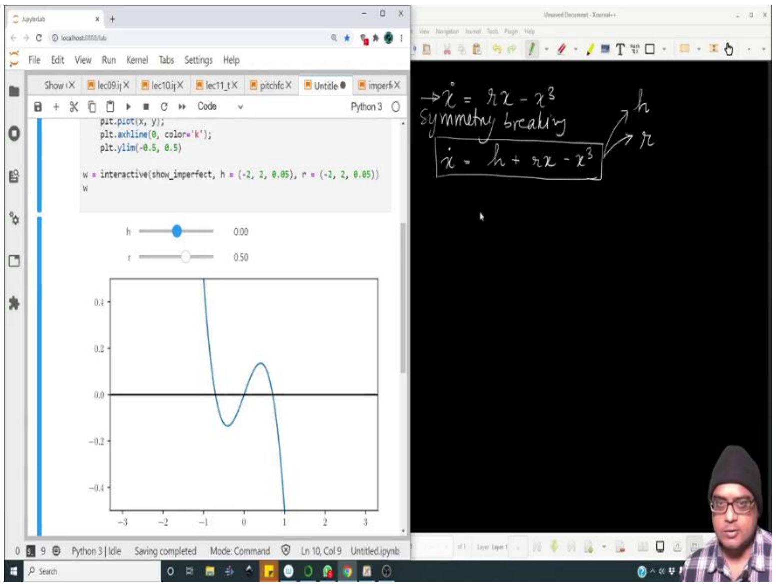
$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ $\nearrow h$ $\nearrow r$

Let me change the y limit also; - 0.5 to 0.5 yeah ok.

(Refer Slide Time: 04:27)



```
    plt.plot(x, y);
    plt.axhline(0, color='k');
    plt.ylim(-0.5, 0.5)

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

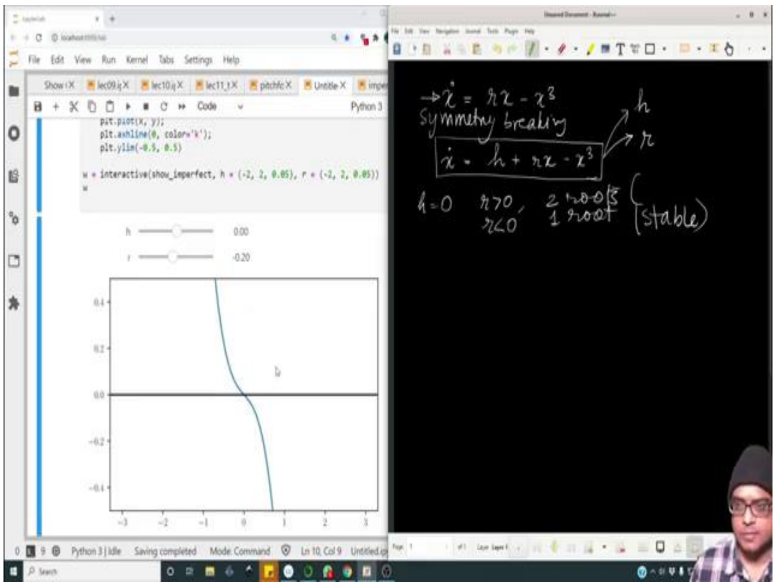
$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3} \rightarrow h, r$

So, now, let us vary h. So, when h is 0 we have 2 roots for r positive ok.

(Refer Slide Time: 04:33)



```
    plt.plot(x, y);
    plt.axhline(0, color='k');
    plt.ylim(-0.5, 0.5)

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

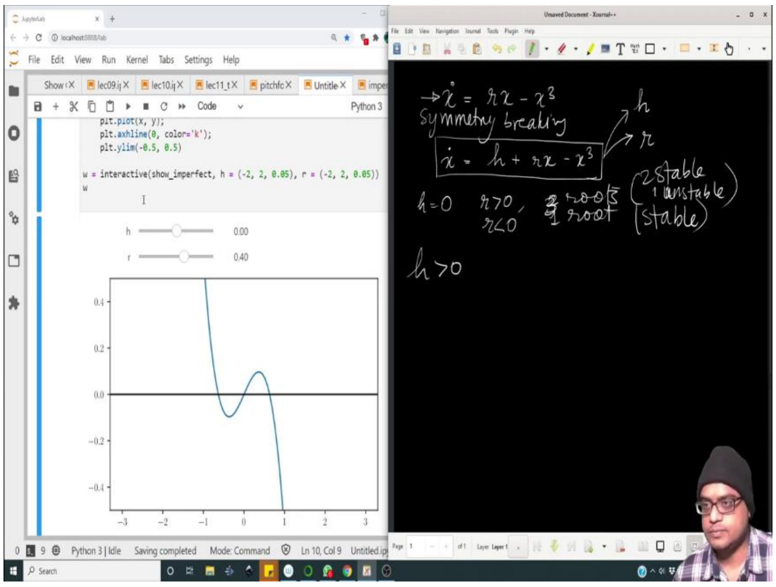
$\boxed{\dot{x} = h + rx - x^3} \rightarrow h, r$

$h = 0$ $r > 0$, 2 roots

$r < 0$ 1 root (stable)

So, when r is positive and h = 0 or rather let me classify it as this. So, h = 0, r is positive 2 roots when r is negative ok. So, there are no more 2 roots. So, when r falls below 0, there is only 1 root ok. And this is a stable root ok. So, this root is stable meaning it is a stable fixed point; when r > 0 what do we have?

(Refer Slide Time: 05:24)



```
    plt.plot(x, y);
    plt.axhline(0, color='k');
    plt.ylim(-0.5, 0.5)

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h 0.00

r 0.40

$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3} \rightarrow h, r$

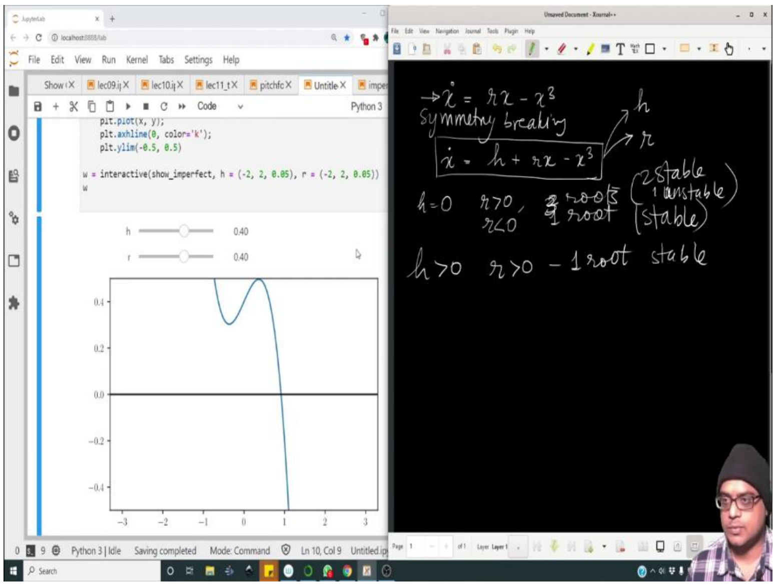
$h = 0 \quad r > 0, \quad$ ~~2~~ 3 roots $\begin{pmatrix} 2\text{ stable} \\ 1\text{ unstable} \end{pmatrix}$

$\qquad\quad r < 0 \quad$ 1 root (stable)

$h > 0$

This is stable, this is unstable, this is stable. So, there are 3 roots not 2 roots there are 3 roots 2 stable and 1 unstable. And this is obvious this simple pitchfork bifurcation ok.

(Refer Slide Time: 05:42)



```
    plt.plot(x, y);
    plt.axhline(0, color='k');
    plt.ylim(-0.5, 0.5)

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h 0.40

r 0.40

$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3} \rightarrow h, r$

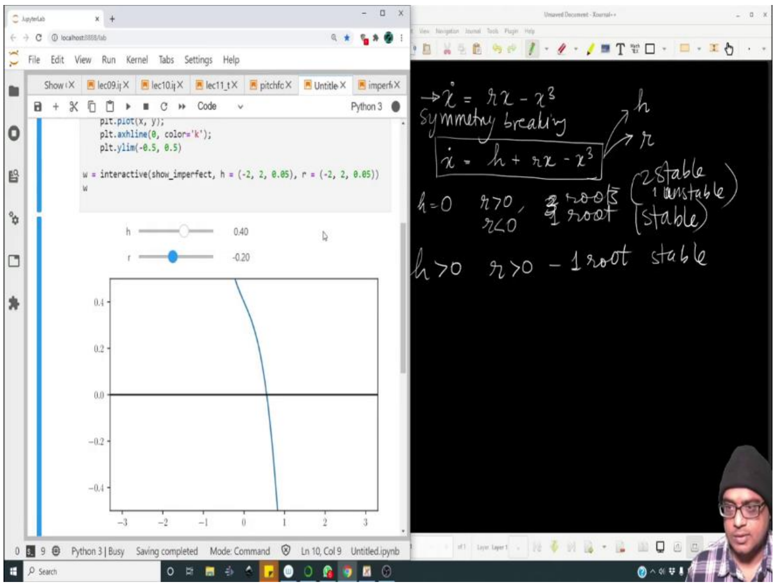
$h = 0 \quad r > 0, \quad$ ~~2~~ 3 roots $\begin{pmatrix} 2\text{ stable} \\ 1\text{ unstable} \end{pmatrix}$

$\qquad\quad r < 0 \quad$ 1 root (stable)

$h > 0 \quad r > 0$ – 1 root stable

When h > 0 there is only 1 root for r > 0 only 1 root and that is a stable root.

(Refer Slide Time: 05:58)



```
plt.plot(x, y);
plt.axhline(0, color='k');
plt.ylim(-0.5, 0.5)

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h 0.40

r -0.20

$\rightarrow \dot{x} = rx - x^3$

symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ h r

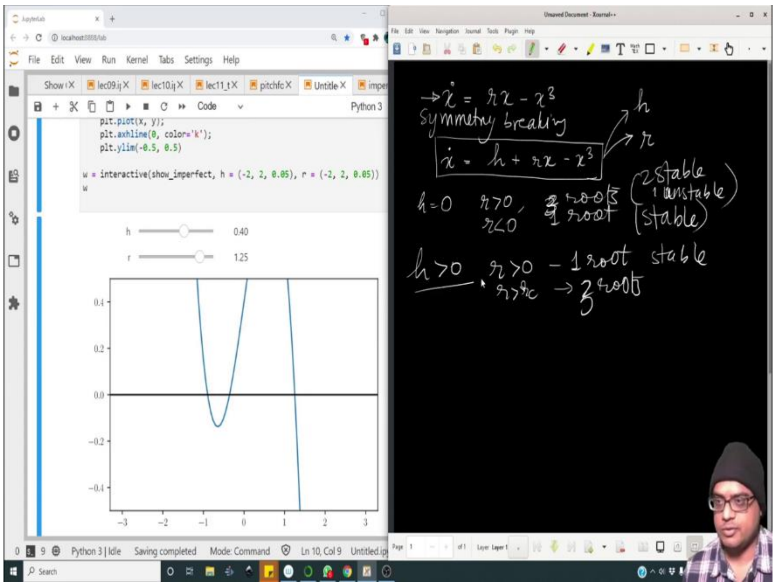
$h = 0$ $r > 0,$ 3 roots (2 stable, 1 unstable)

$r < 0$ 1 root (stable)

$h > 0$ $r > 0$ – 1 root stable

When r< 0, there is still only 1 root.

(Refer Slide Time: 06:06)



```
plt.plot(x, y);
plt.axhline(0, color='k');
plt.ylim(-0.5, 0.5)

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h 0.40

r 1.25

$\rightarrow \dot{x} = rx - x^3$

symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ h r

$h = 0$ $r > 0,$ 3 roots (2 stable, 1 unstable)

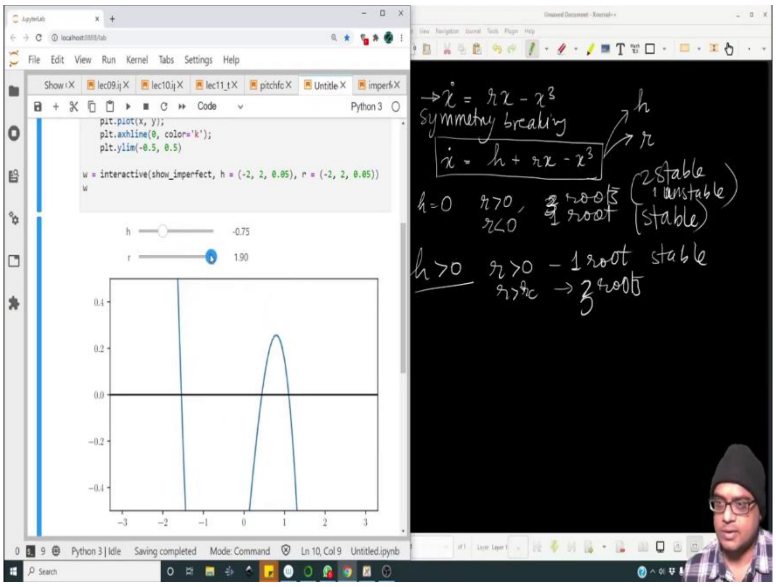
$r < 0$ 1 root (stable)

$h > 0$ $r > 0$ – 1 root stable

$r > r_c \rightarrow 3$ roots

When r actually is pretty large this presence of 2 roots and r greater than some critical value you have 2 roots and sorry not 2 roots you have 3 roots. So, depending on what value of r you have and the combination of h.

(Refer Slide Time: 06:29)



```
plt.plot(x, y);
plt.axhline(0, color='k');
plt.ylim(-0.5, 0.5)

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h -0.75

r 1.90

$\rightarrow \dot{x} = rx - x^3$

symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ h r

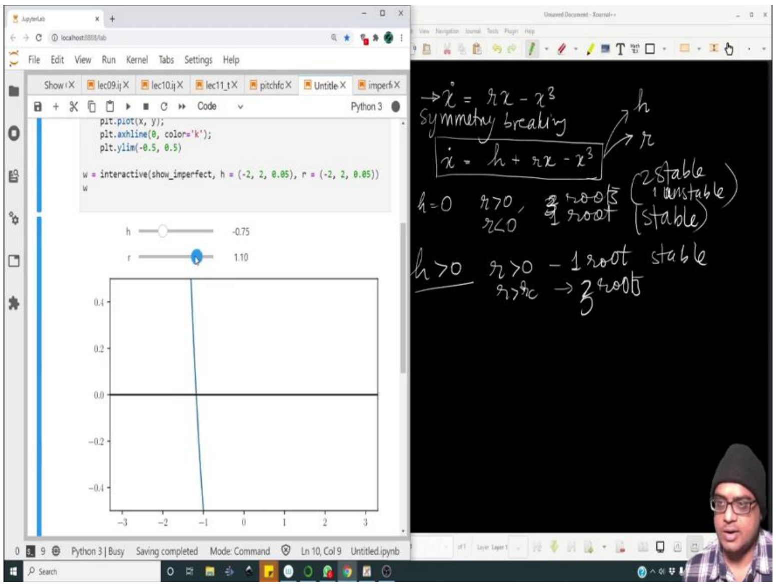
$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)

$r<0$ 1 root (stable)

$h>0$ $r>0$ - 1 root stable

$r>r_c \rightarrow$ 3 roots

(Refer Slide Time: 06:33)



```
plt.plot(x, y);
plt.axhline(0, color='k');
plt.ylim(-0.5, 0.5)

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h -0.75

r 1.10

$\rightarrow \dot{x} = rx - x^3$

symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ h r

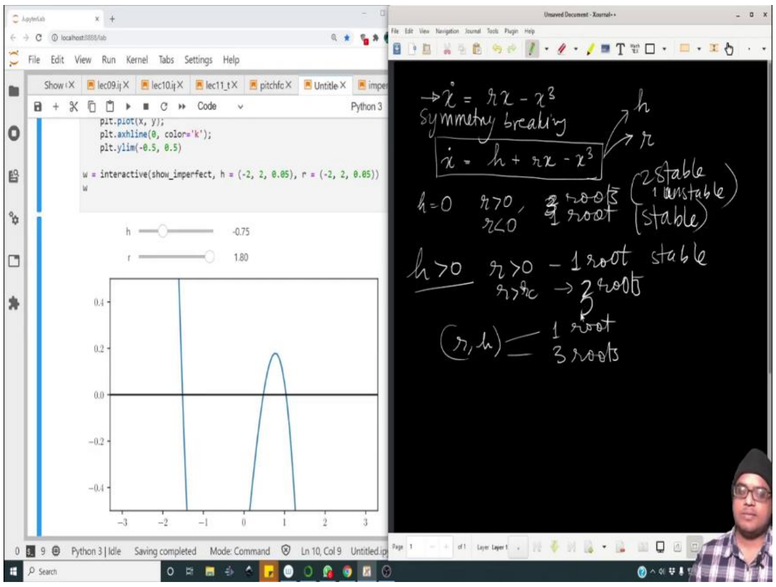
$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)

$r<0$ 1 root (stable)

$h>0$ $r>0$ - 1 root stable

$r>r_c \rightarrow$ 3 roots

(Refer Slide Time: 06:37)



```
    plt.plot(x, y);
    plt.axhline(0, color='k');
    plt.ylim(-0.5, 0.5)

w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

```
h  -0.75
r  1.80
```

$\to \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3} \to h, r$

$h=0$ $r>0$, 3 roots (2 stable 1 unstable)

$r<0$ 1 root (stable)

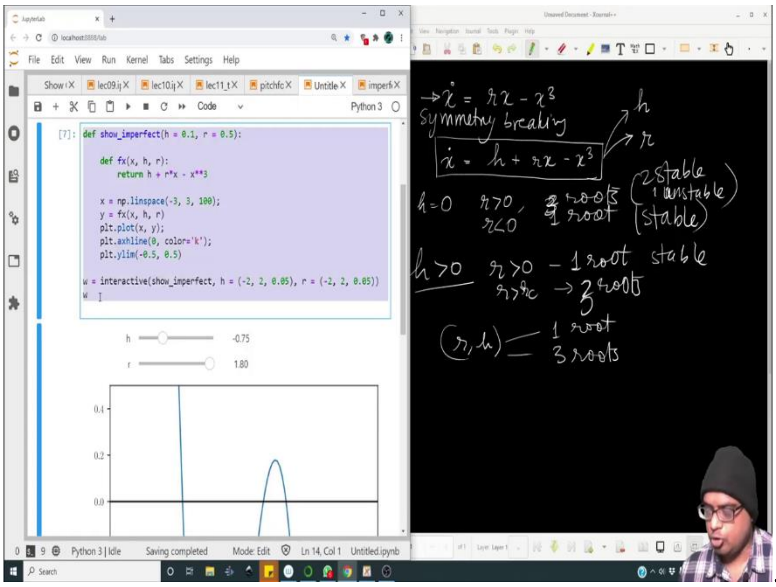
$\underline{h>0}$ $r>0$ – 1 root stable

$r>r_c \to$ 3 roots

(r,h) — 1 root — 3 roots

But when h is negative ok, again you can have a certain r at which again you have 3 roots. So, r and h they determine whether the equation has 1 root or the equation has 3 roots and well it may not be entirely clear what is going on in this cubic equation; we can actually split this equation into 2 parts.

(Refer Slide Time: 07:00)



```
[7]: def show_imperfect(h = 0.1, r = 0.5):

         def fx(x, h, r):
             return h + r*x - x**3

         x = np.linspace(-3, 3, 100);
         y = fx(x, h, r)
         plt.plot(x, y);
         plt.axhline(0, color='k');
         plt.ylim(-0.5, 0.5)

     w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
     w
```

```
h  -0.75
r  1.80
```

$\to \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3} \to h, r$

$h=0$ $r>0$, 3 roots (2 stable 1 unstable)

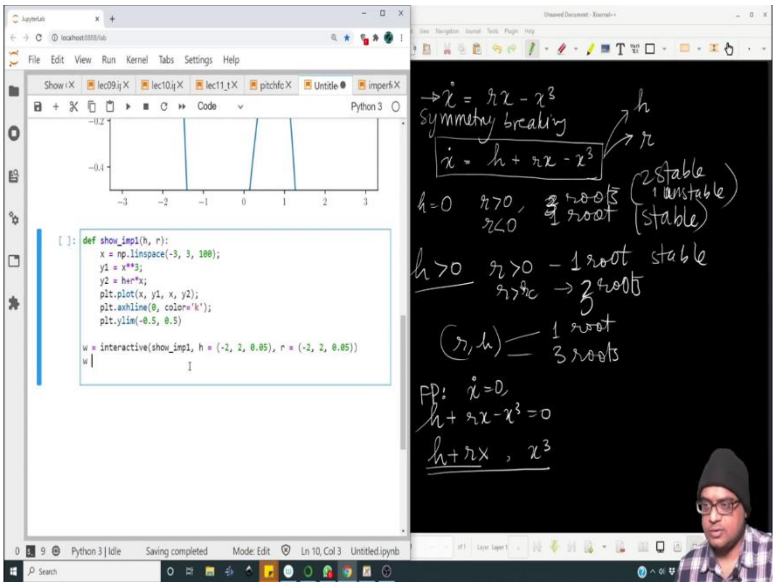
$r<0$ 1 root (stable)

$\underline{h>0}$ $r>0$ – 1 root stable

$r>r_c \to$ 3 roots

(r,h) — 1 root — 3 roots

(Refer Slide Time: 07:02)



```
def show_imp1(h, r):
    x = np.linspace(-3, 3, 100);
    y1 = x**3;
    y2 = h+r*x;
    plt.plot(x, y1, x, y2);
    plt.axhline(0, color='k');
    plt.ylim(-0.5, 0.5)

w = interactive(show_imp1, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3} \to h, r$

$h = 0 \quad r > 0, \quad 3$ roots $\left(\begin{array}{l}2 \text{ stable} \\ 1 \text{ unstable}\end{array}\right)$

$r < 0 \quad 1$ root (stable)

$\underline{h > 0} \quad r > 0 - 1$ root stable

$r > r_c \to 3$ roots

(r, h) — 1 root, 3 roots

FP: $\dot{x} = 0$

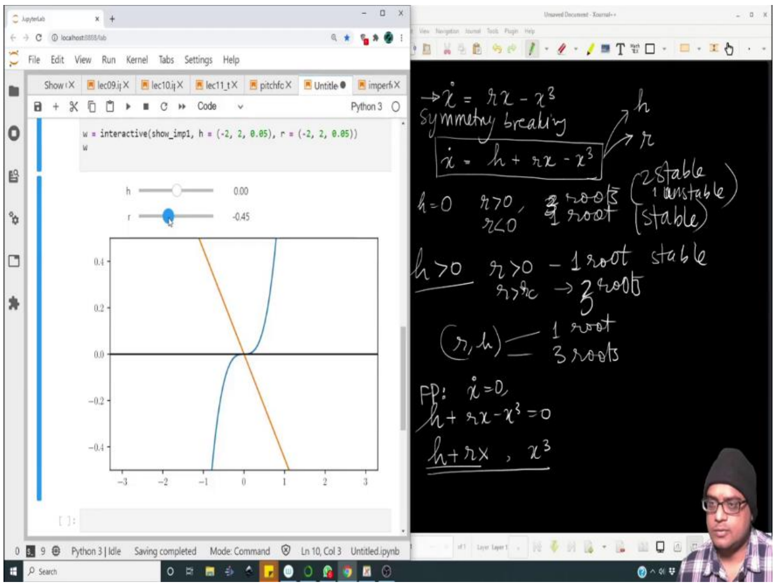
$h + rx - x^3 = 0$

$\underline{h + rx \;,\; x^3}$

Let me copy all of this, let me paste it over here we can actually break this equation into 2 functions. So, when we have we are searching for a fixed point; obviously, $\dot{x} = 0$. So, $h + rx - x^3 = 0$. So, we can plot 2 curves. So, the first curve can be $h + rx$ and the second curve can be x^3 and then we can look at the intersections of these 2 curves ok.

Let us actually try to plot this. So, we do not need this. So, actually let us create it from scratch, but let us not copy paste ok. So, let us remove this x = linspace. So, y = x**3 alright. So, let this be y1 = x**3 and y2=h + r*x. So, let us plot simultaneously plt.plot(x ,y1, x, y2) and the x axis. Let us now wrap everything inside a function show_imp1(h,r) let us put all of this inside the function let us remove this ok. So, let us pass h and r. So, show_imp1 alright.

(Refer Slide Time: 08:43)



```
w = interactive(show_imp1, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h 0.00

r -0.45

→ $\dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ → h, r

$h = 0$ $r > 0$, 3 roots (2 stable, 1 unstable)

$r < 0$ 1 root (stable)

$h > 0$ $r > 0$ - 1 root stable

$r > r_c$ → 3 roots

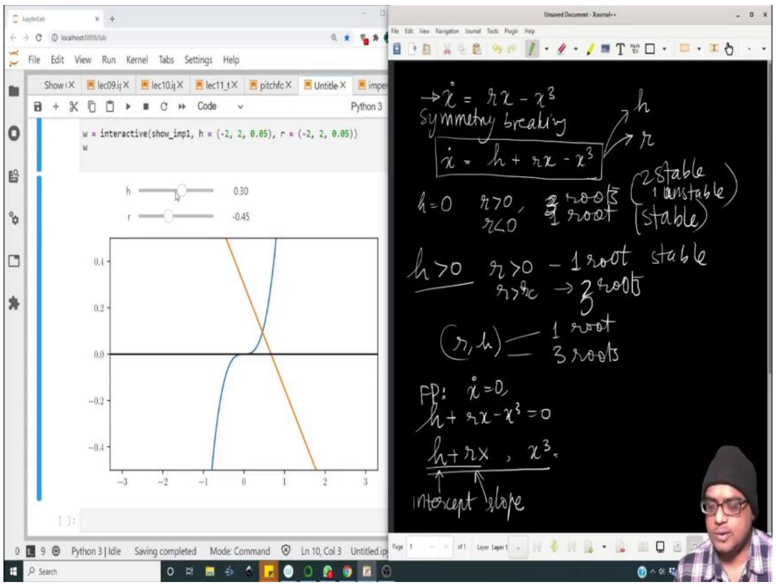
(r, h) — 1 root, 3 roots

FP: $\dot{x} = 0$

$h + rx - x^3 = 0$

$\underline{h + rx}$, $\underline{x^3}$

(Refer Slide Time: 08:55)



```
w = interactive(show_imp1, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h 0.30

r -0.45

→ $\dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ → h, r

$h = 0$ $r > 0$, 3 roots (2 stable, 1 unstable)

$r < 0$ 1 root (stable)

$h > 0$ $r > 0$ - 1 root stable

$r > r_c$ → 3 roots

(r, h) — 1 root, 3 roots

FP: $\dot{x} = 0$

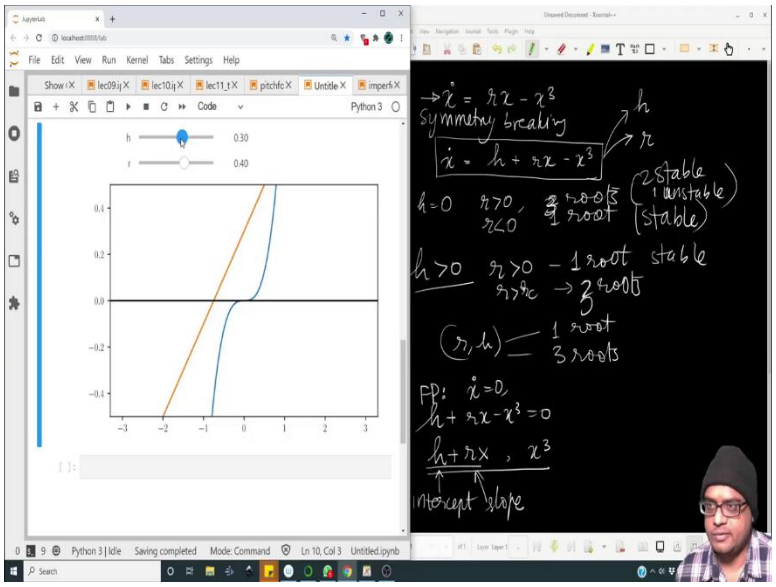
$h + rx - x^3 = 0$

$\underline{h + rx}$, x^3.

intercept slope

So, let us execute this alright. So, look when r is negative. So, right over here it is quite obvious that h is the intercept and r is the slope and x^3 is x^3 ok. So, $x^3$is not going to change hm, but what is going to change is this; this particular line.

(Refer Slide Time: 09:20)



h 0.30

r 0.40

$\to \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ → h, r

$h = 0$ $\quad r > 0$, 3 roots (2 stable, 1 unstable)

$\quad r < 0$, 1 root (stable)

$h > 0$ $\quad r > 0$ – 1 root stable

$\quad r > r_c \to 3$ roots

(r, h) — 1 root, 3 roots

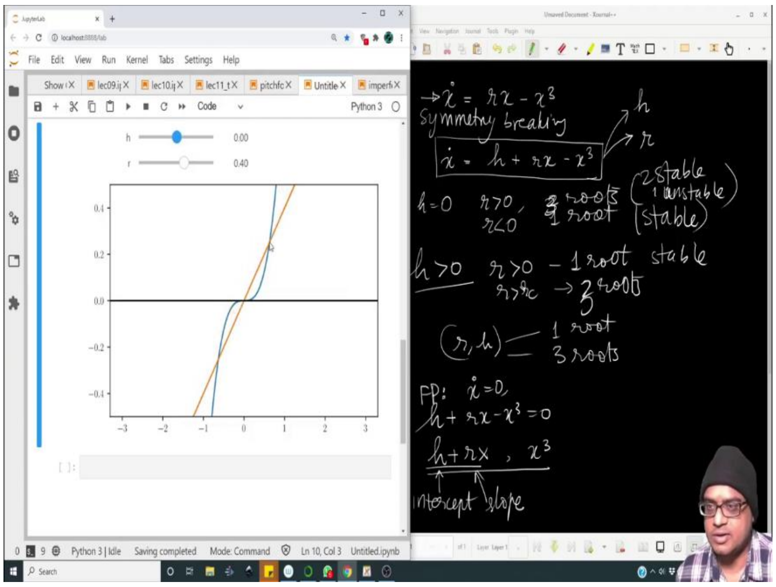
FP: $\dot{x} = 0$

$h + rx - x^3 = 0$

$h + rx \;,\; x^3$

intercept slope

(Refer Slide Time: 09:23)

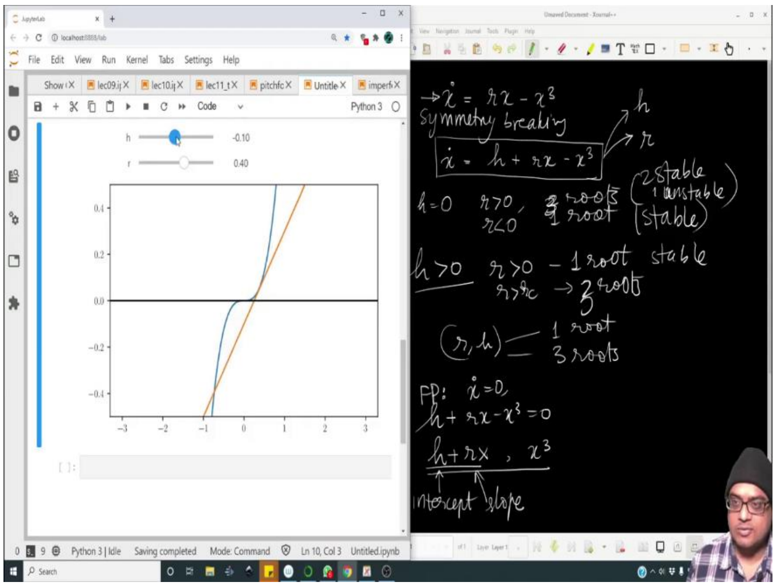


h 0.00

r 0.40

So, when r becomes positive and as h changes we can have 3 points where the intersect 1, 2, 3 ok. So, it is all about how we can rotate the line slide it around.

(Refer Slide Time: 09:36)



$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ $\nearrow h$ $\nearrow r$

$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)

$r<0$ 1 root (stable)

$h>0$ $r>0$ - 1 root stable

$r>r_c \rightarrow 3$ roots

(r,h) — 1 root, 3 roots

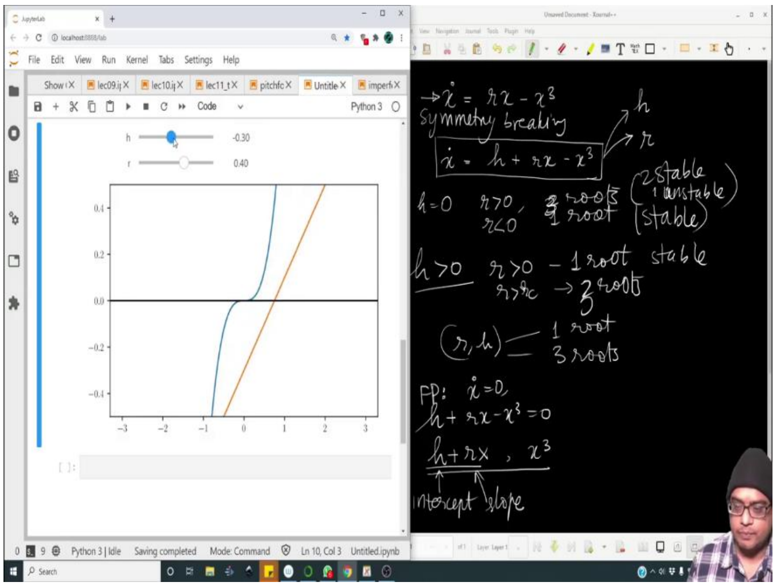
FP: $\dot{x}=0$

$h + rx - x^3 = 0$

$\underline{h+rx \;,\; x^3}$

intercept slope

(Refer Slide Time: 09:39)



$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ $\nearrow h$ $\nearrow r$

$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)

$r<0$ 1 root (stable)

$h>0$ $r>0$ - 1 root stable

$r>r_c \rightarrow 3$ roots

(r,h) — 1 root, 3 roots

FP: $\dot{x}=0$

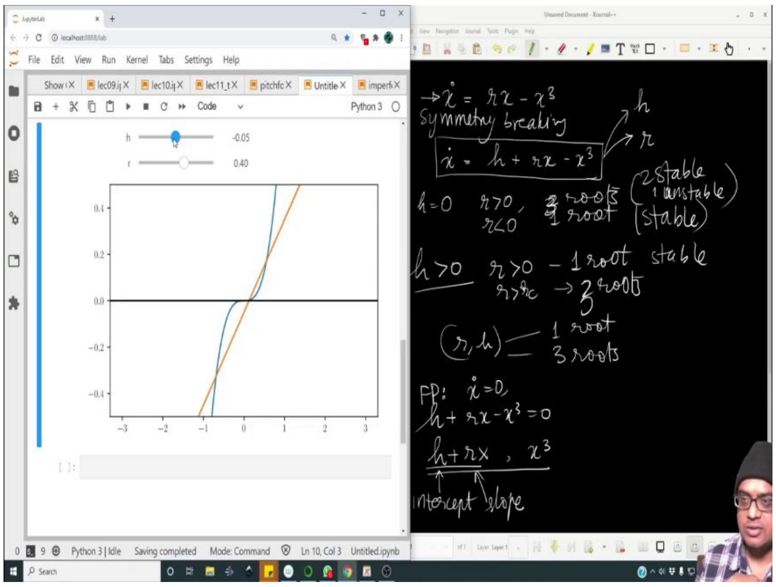
$h + rx - x^3 = 0$

$\underline{h+rx \;,\; x^3}$

intercept slope

So, we can have 3 intersections like this by change around h we are having just see over here we have one point of intersection near minus 1 and another point of intersection just about to originate around something between 0 and 0; 0 and 0.2. So, it is tangential, so, that straight line that is h + rx is now tangential to x^3. If I now increase h now, we have 3 points until this point.

(Refer Slide Time: 10:04)



$\rightarrow \dot{x} = rx - x^3$

Symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ → h, → r

$h = 0 \quad r > 0, \quad$ 3 roots (2 stable, 1 unstable)

$\qquad\quad r < 0 \quad$ 1 root (stable)

$h > 0 \quad r > 0$ – 1 root stable

$\qquad\quad r > r_c \rightarrow$ 3 roots

(r, h) — 1 root, — 3 roots

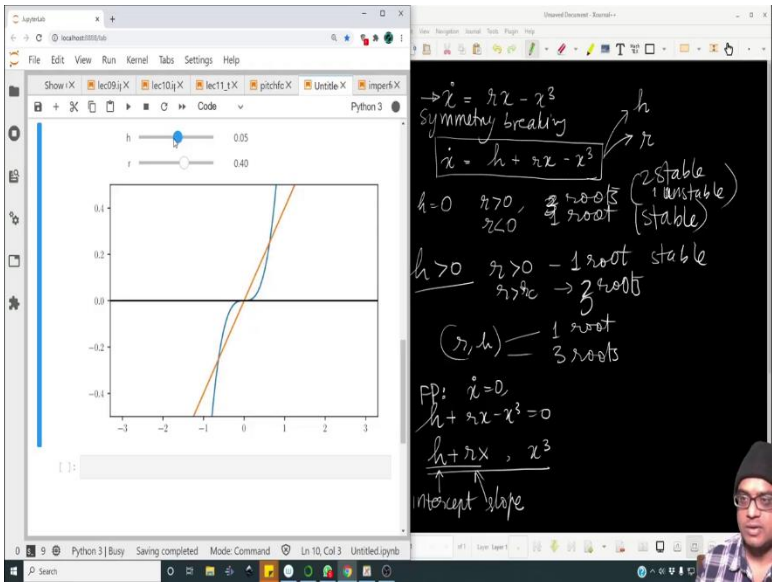
FP: $\dot{x} = 0$

$h + rx - x^3 = 0$

$\underline{h + rx}, \; x^3$

intercept slope

(Refer Slide Time: 10:06)



h 0.05

r 0.40

$\rightarrow \dot{x} = rx - x^3$

symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ $\rightarrow h$, $\rightarrow r$

$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)

$r<0$, 1 root (stable)

$h>0$ $r>0$ − 1 root stable

$r>r_c \rightarrow$ 3 roots

(r,h) — 1 root, 3 roots

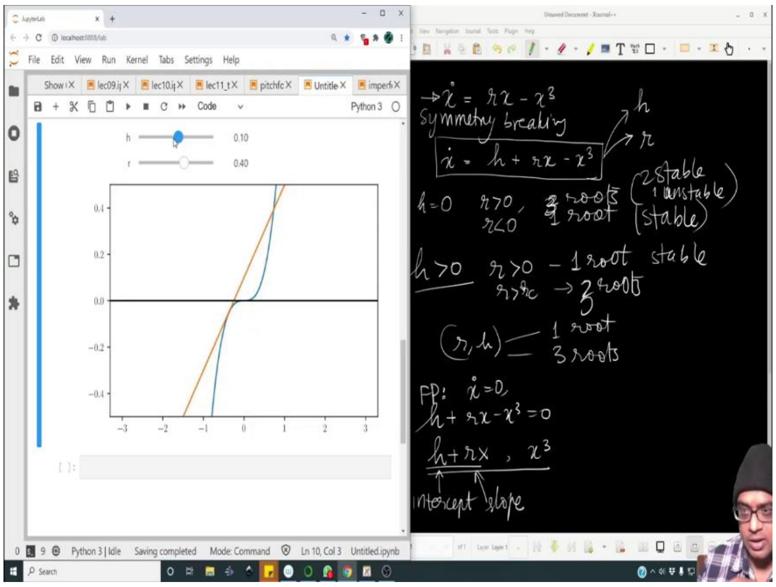
FP: $\dot{x}=0$

$h + rx - x^3 = 0$

$\underline{h + rx}$, x^3

intercept slope

(Refer Slide Time: 10:07)



h 0.10

r 0.40

$\rightarrow \dot{x} = rx - x^3$

symmetry breaking

$\boxed{\dot{x} = h + rx - x^3}$ $\rightarrow h$, $\rightarrow r$

$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)

$r<0$, 1 root (stable)

$h>0$ $r>0$ − 1 root stable

$r>r_c \rightarrow$ 3 roots

(r,h) — 1 root, 3 roots

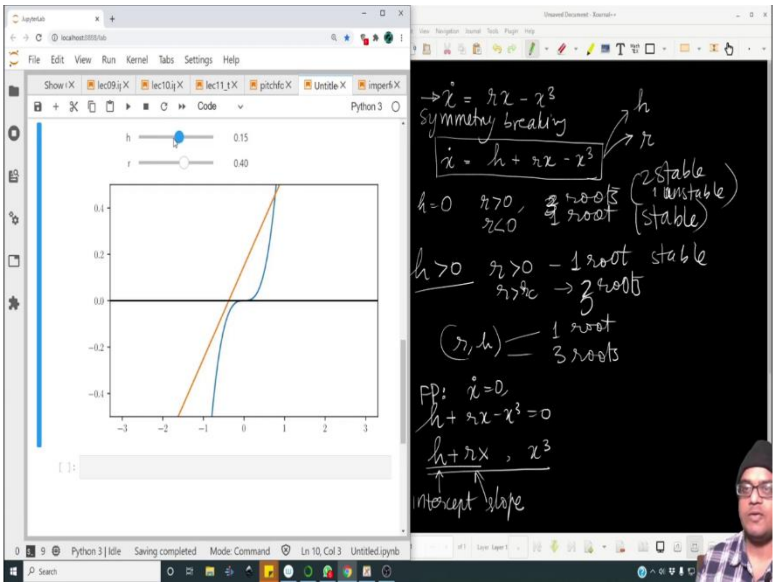
FP: $\dot{x}=0$

$h + rx - x^3 = 0$

$\underline{h + rx}$, x^3

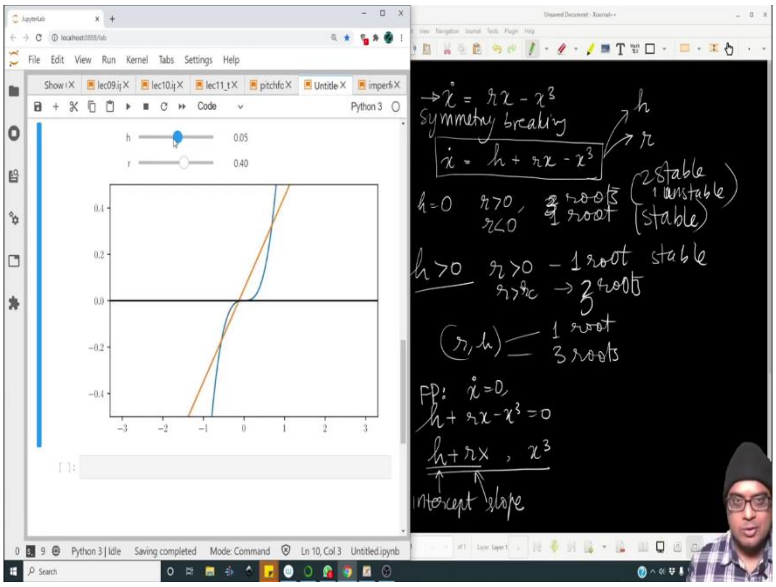
intercept slope

(Refer Slide Time: 10:15)


h 0.15
r 0.40
$\rightarrow \dot{x} = rx - x^3$
Symmetry breaking
$\boxed{\dot{x} = h + rx - x^3}$
h
r
$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)
$r<0$ 1 root (stable)
$h>0$ $r>0$ − 1 root stable
$r>r_c$ → 3 roots
(r,h) — 1 root, 3 roots
FP: $\dot{x}=0$
$h + rx - x^3 = 0$
$h+rx$, x^3
intercept slope

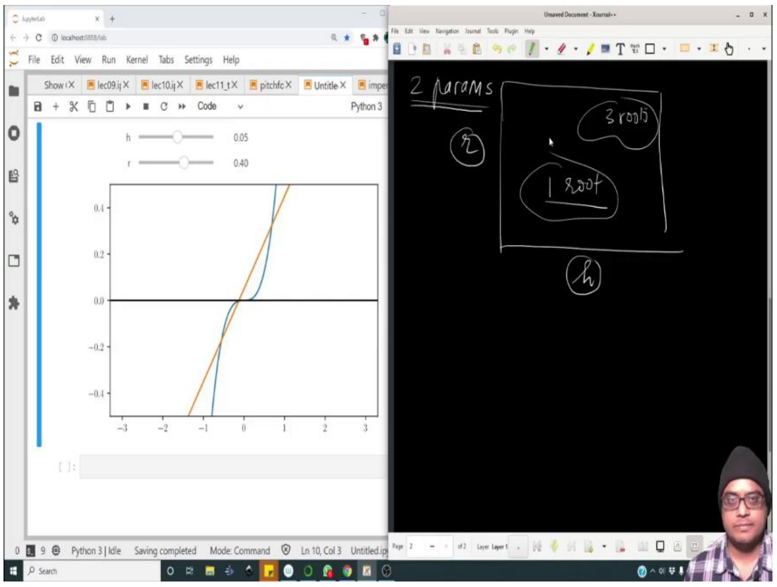
Now we have only 2 points. So, the lower branch of x^3 is not tangential and now we have only 1 root ok.

(Refer Slide Time: 10:17)


h 0.05
r 0.40
$\rightarrow \dot{x} = rx - x^3$
Symmetry breaking
$\boxed{\dot{x} = h + rx - x^3}$
h
r
$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)
$r<0$ 1 root (stable)
$h>0$ $r>0$ − 1 root stable
$r>r_c$ → 3 roots
(r,h) — 1 root, 3 roots
FP: $\dot{x}=0$
$h + rx - x^3 = 0$
$h+rx$, x^3
intercept slope

So, this is how we can sweep across the parameter space and find out the intersections. So, in the case of imperfect bifurcations the whole idea is one parameter is not solely guiding the nature of the equation.

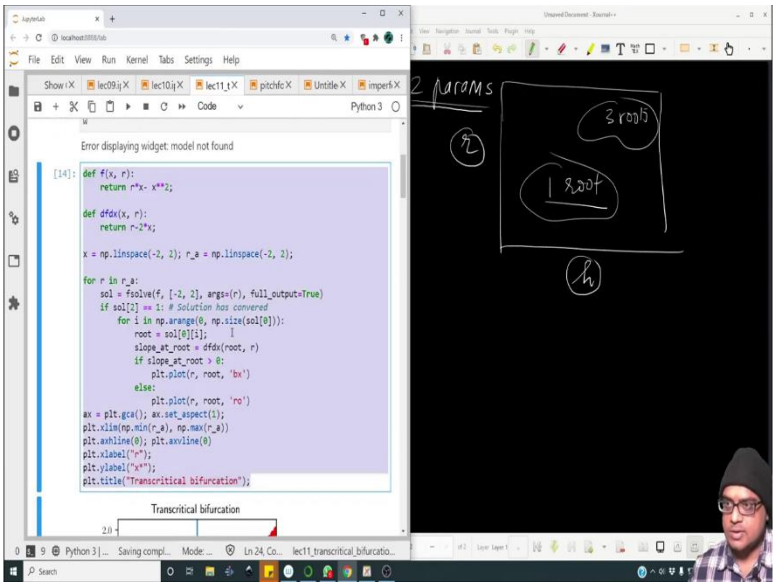
(Refer Slide Time: 10:38)



Rather there are 2 params and if we were to extend our discussion to something in the r h plane ok. So, there are various combinations of r and h where you have 2 roots rather 1 root. There are certain combinations of r h where we have 3 roots ok. So, now, let us try to identify exactly what is the parameter space and how many roots do we have.

Is there some demarcation and can we find out how r and h dictate the nature of the equation in at least the number of fixed points. We will discuss about the stability of the fixed points later on, but at least let us look at how we can summarize the number of fixed points of this particular equation. So, let us now write a snippet to find out the parameter space ok.

(Refer Slide Time: 11:54)



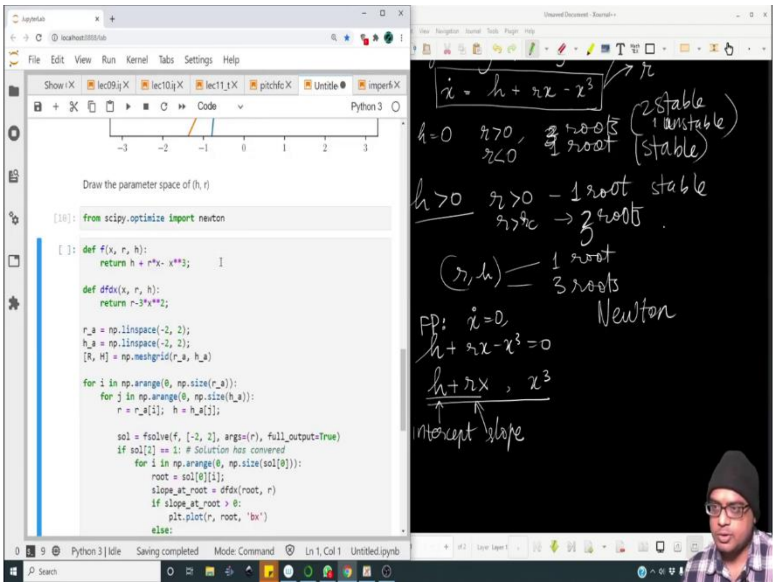
Error displaying widget: model not found

```
[14]: def f(x, r):
          return r*x- x**2;

      def dfdx(x, r):
          return r-2*x;

      x = np.linspace(-2, 2); r_a = np.linspace(-2, 2);

      for r in r_a:
          sol = fsolve(f, [-2, 2], args=(r), full_output=True)
          if sol[2] == 1: # Solution has convered
              for i in np.arange(0, np.size(sol[0])):
                  root = sol[0][i];
                  slope_at_root = dfdx(root, r)
                  if slope_at_root > 0:
                      plt.plot(r, root, 'bx')
                  else:
                      plt.plot(r, root, 'ro')
      ax = plt.gca(); ax.set_aspect(1);
      plt.xlim(np.min(r_a), np.max(r_a))
      plt.axhline(0); plt.axvline(0)
      plt.xlabel("r");
      plt.ylabel("x*");
      plt.title("Transcritical bifurcation");
```

Transcritical bifurcation

So, let us go to the previous lecture we will reuse this particular code.

(Refer Slide Time: 12:02)



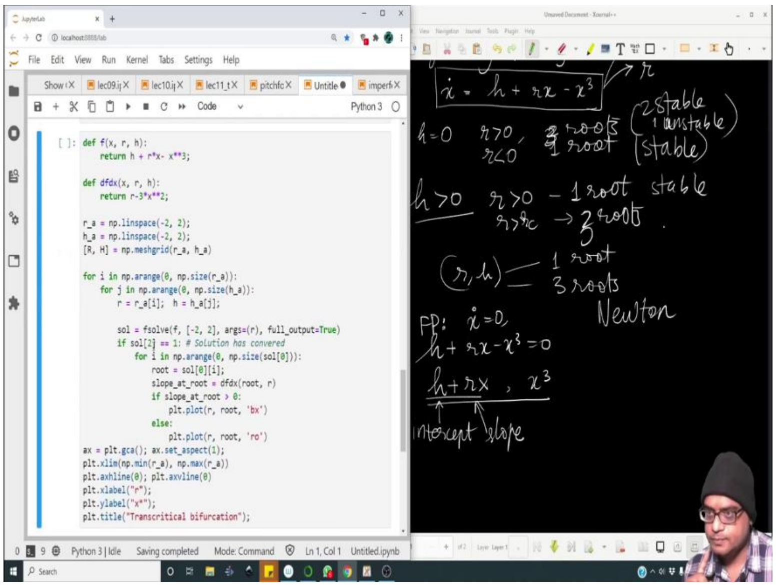
Draw the parameter space of (h, r)

```
[10]: from scipy.optimize import newton

[ ]: def f(x, r, h):
         return h + r*x- x**3;

     def dfdx(x, r, h):
         return r-3*x**2;

     r_a = np.linspace(-2, 2);
     h_a = np.linspace(-2, 2);
     [R, H] = np.meshgrid(r_a, h_a)

     for i in np.arange(0, np.size(r_a)):
         for j in np.arange(0, np.size(h_a)):
             r = r_a[i];  h = h_a[j];

             sol = fsolve(f, [-2, 2], args=(r), full_output=True)
             if sol[2] == 1: # Solution has convered
                 for i in np.arange(0, np.size(sol[0])):
                     root = sol[0][i];
                     slope_at_root = dfdx(root, r)
                     if slope_at_root > 0:
                         plt.plot(r, root, 'bx')
                     else:
```

So, parameter space of (h ,r) alright. So, we pasted this and so, the function that we are worried about is this. So, (x, r,h) (x , r , h) and this has to be $h + rx + x^3$. This will be $r - 3x^2$. So, we do not need x anymore. So, we need r_a and h_a and we will need will be bothered with r on the x axis and h this will be np.meshgrid(r_a,h_a) alright.

So, for r in r_a for h in h_a or in fact, this has to be in terms of i and j; for i in np.arange 1 sorry 0 to np.size(r_a) for h in np.arange(0,np.size(h_a)). This has to be j. So, we missed a comma we missed a bracket over here ok. So, then r will be r_a[i] and h will be h_a[j] alright. Now we must solve the equation ok.

(Refer Slide Time: 14:27)



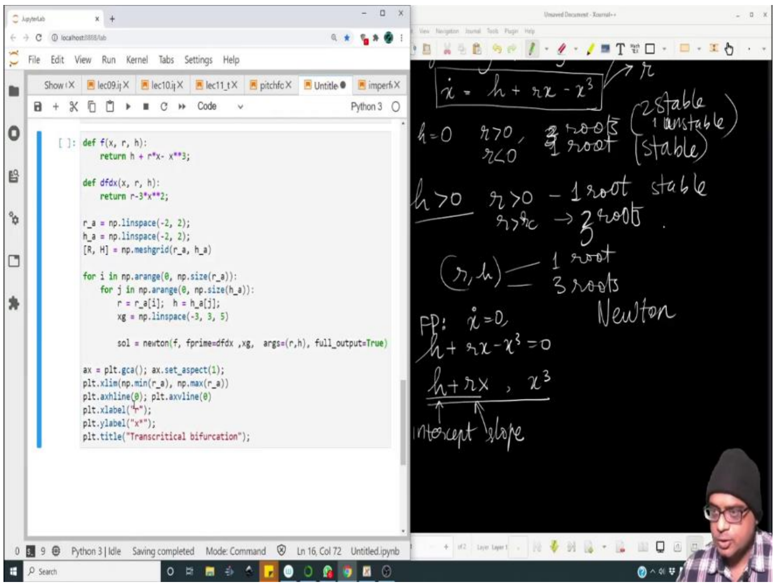
```
[ ]: def f(x, r, h):
         return h + r*x- x**3;

     def dfdx(x, r, h):
         return r-3*x**2;

     r_a = np.linspace(-2, 2);
     h_a = np.linspace(-2, 2);
     [R, H] = np.meshgrid(r_a, h_a)

     for i in np.arange(0, np.size(r_a)):
         for j in np.arange(0, np.size(h_a)):
             r = r_a[i];  h = h_a[j];

             sol = fsolve(f, [-2, 2], args=(r), full_output=True)
             if sol[2] == 1: # Solution has convered
                 for i in np.arange(0, np.size(sol[0])):
                     root = sol[0][i];
                     slope_at_root = dfdx(root, r)
                     if slope_at_root > 0:
                         plt.plot(r, root, 'bx')
                     else:
                         plt.plot(r, root, 'ro')
     ax = plt.gca(); ax.set_aspect(1);
     plt.xlim(np.min(r_a), np.max(r_a))
     plt.axhline(0); plt.axvline(0)
     plt.xlabel("r");
     plt.ylabel("x*");
     plt.title("Transcritical bifurcation");
```

File Edit View Run Kernel Tabs Settings Help
Python 3 | Idle Saving completed Mode: Command Ln 1, Col 1 Untitled.ipynb
$\boxed{\dot{x} = h + rx - x^3}$ → r
h=0 r>0, 3 roots (2 stable 1 unstable)
r<0 1 root (stable)
h>0 r>0 – 1 root stable
r>r_c → 3 roots
(r, h) – 1 root – 3 roots
Newton
FP: $\dot{x}=0$
$h + rx - x^3 = 0$
h+rx , x^3
intercept slope

Let me just indent everything. So, now, instead of using the fsolve solver we will now go to the Newton solver ok. So, for using that because fsolve solver is not really efficient over finding us all the roots in. I know that all in all the previous lectures we have been using the fsolve, but using the Newton will give you far better results try to rewrite all the other codes in terms of the Newton solver.

You do not need to change everything, but just follow along it should be rather simple to do. So, here let us import from scipy.integrate import Newton ok. Let us sorry some scipy .optimize import Newton and over here.

(Refer Slide Time: 15:24)



```
[ ]: def f(x, r, h):
         return h + r*x- x**3;

     def dfdx(x, r, h):
         return r-3*x**2;

     r_a = np.linspace(-2, 2);
     h_a = np.linspace(-2, 2);
     [R, H] = np.meshgrid(r_a, h_a)

     for i in np.arange(0, np.size(r_a)):
         for j in np.arange(0, np.size(h_a)):
             r = r_a[i];  h = h_a[j];
             xg = np.linspace(-3, 3, 5)

             sol = newton(f, fprime=dfdx ,xg,  args=(r,h), full_output=True)

     ax = plt.gca(); ax.set_aspect(1);
     plt.xlim(np.min(r_a), np.max(r_a))
     plt.axhline(0); plt.axvline(0)
     plt.xlabel("r");
     plt.ylabel("x*");
     plt.title("Transcritical bifurcation");
```

$\boxed{\dot{x} = h + rx - x^3} \rightarrow r$

$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)
$r<0$ 1 root (stable)

$h>0$ $r>0$ − 1 root stable
$r>r_c \rightarrow$ 3 roots

(r,h) — 1 root
— 3 roots

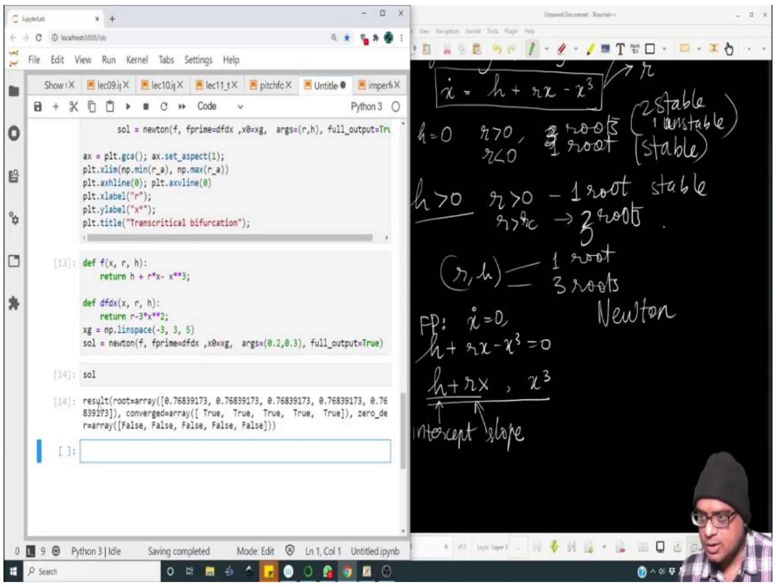
Newton

FP: $\dot{x}=0$
$h + rx - x^3 = 0$
$h+rx$, x^3
intercept slope

So, this has to be Newton and the Newton syntax if I remember correctly. So, f prime we have to supply. So, this will be df dx. The initial guess let me define it as xg and we will define the guess range = np.linspace(-3,3) let us take 5 points as the guess points in order to cover all the 3 roots args have we have to pass r , h and full output will be = true ok. So, we have worried about the number of roots.

So, we do not really bother about plotting it for now alright. So, we do not care about this. Let us get rid of this ok. So, in fact, let me create a new cell to show you what the output structure of Newton looks like ok.

(Refer Slide Time: 16:34)



```
        sol = newton(f, fprime=dfdx ,x0=xg,  args=(r,h), full_output=Tru

ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");

[13]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(0.2,0.3), full_output=True)

[14]: sol

[14]: result(root=array([0.76839173, 0.76839173, 0.76839173, 0.76839173, 0.76
      839173]), converged=array([ True,  True,  True,  True,  True]), zero_de
      r=array([False, False, False, False, False]))
```

$$\boxed{\dot{x} = h + rx - x^3} \nearrow r$$

$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)
$r<0$ 1 root (stable)

$\underline{h>0}$ $r>0$ – 1 root stable
$r>r_c \rightarrow 3$ roots .

(r, h) — 1 root
— 3 roots

Newton

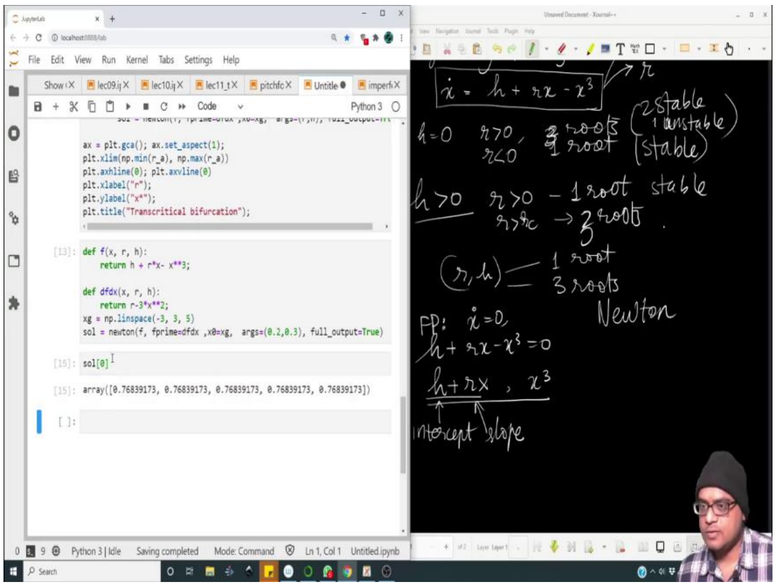
FP: $\dot{x}=0$,
$h + rx - x^3 = 0$

$\underline{\underline{h+rx}, x^3}$

intercept slope

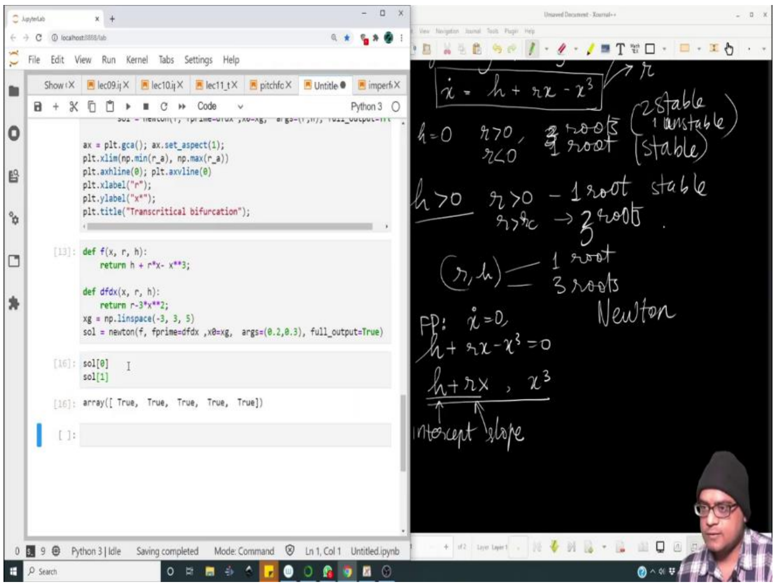
Let me paste this. Let me pass r = 0.2 and h = 0.3 positional. So, this has to be x0 =xg. We have not defined xg ok. So, let me fix this as well alright. So, this has run. Let us look at let us prove what sol is. So, sol contains these things; it contains root, it contains converged which is the array. So, all the roots have converged which is fine and ok.

(Refer Slide Time: 17:29)



```
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");

[13]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(0.2,0.3), full_output=True)

[15]: sol[0]

[15]: array([0.76839173, 0.76839173, 0.76839173, 0.76839173, 0.76839173])
```

(Refer Slide Time: 17:35)



```
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

```
[13]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(0.2,0.3), full_output=True)
```

```
[16]: sol[0]
      sol[1]

[16]: array([ True,  True,  True,  True,  True])
```

$\dot{x} = h + rx - x^3 \rightarrow r$

$h=0$ $r>0$, 3 roots (2 stable, 1 unstable); $r<0$, 1 root (stable)

$h>0$ $r>0$ − 1 root stable; $r>r_c$ → 3 roots.

(r,h) — 1 root / 3 roots

Newton

FP: $\dot{x}=0$

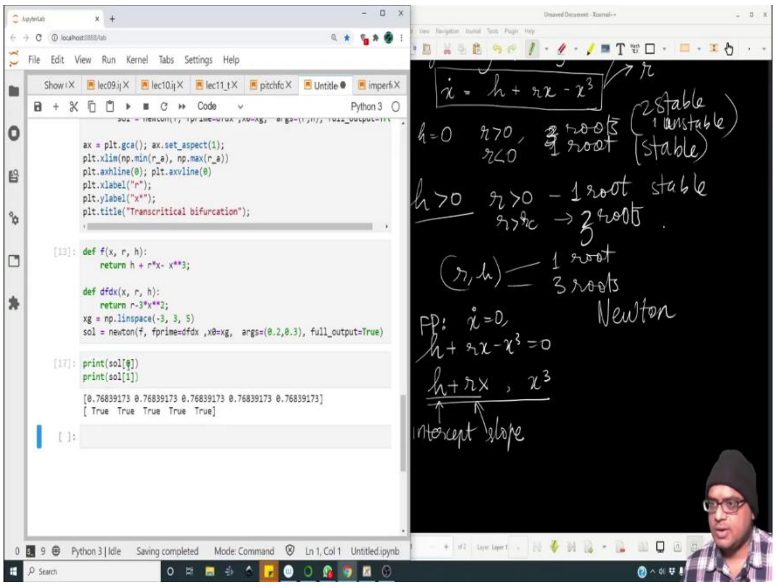
$h+rx-x^3=0$

$\underline{h+rx}$, x^3

intercept slope

So, we are more bothered with sol[0] which is the root array and sol[1] which says whether all the things have converged or not ok.

(Refer Slide Time: 17:39)

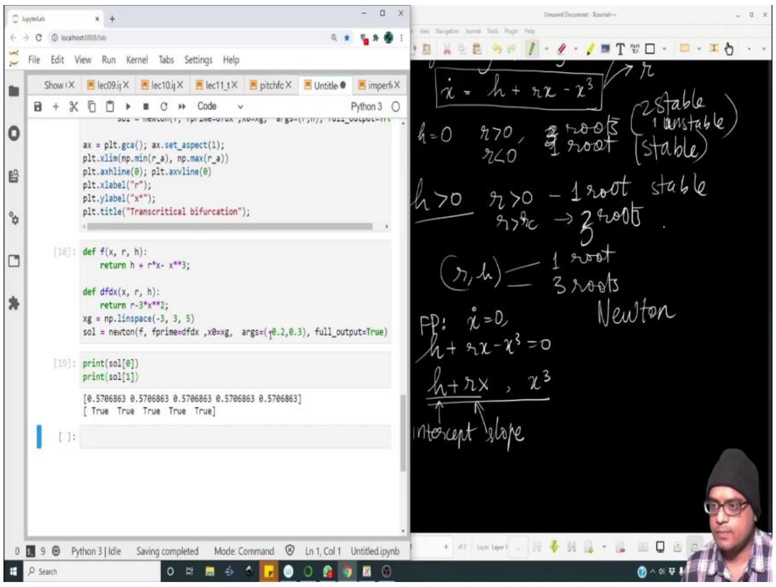


```
[13]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(0.2,0.3), full_output=True)
```

```
[17]: print(sol[0])
      print(sol[1])

      [0.76839173 0.76839173 0.76839173 0.76839173 0.76839173]
      [ True  True  True  True  True]
```

I am trying to show you this; because from this will really get to know something.

(Refer Slide Time: 17:52)



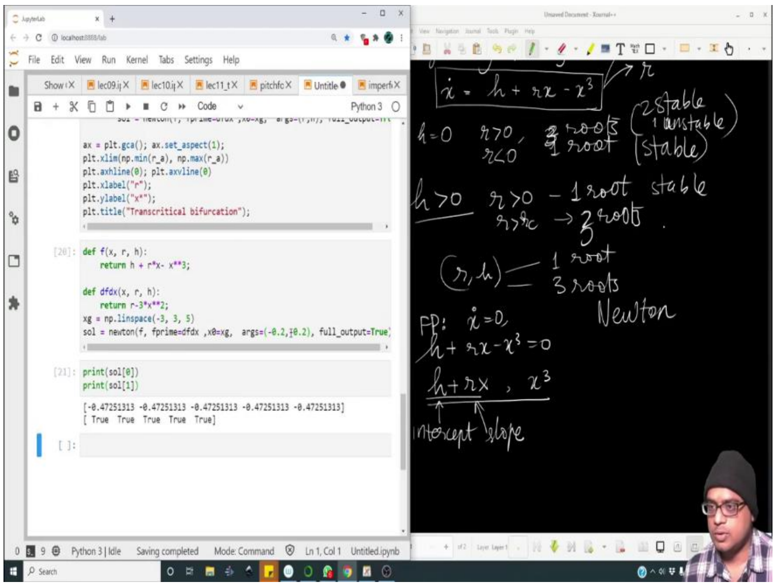
```
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

```
[18]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(0.2,0.3), full_output=True)
```

```
[19]: print(sol[0])
      print(sol[1])

[0.5706863 0.5706863 0.5706863 0.5706863 0.5706863]
[ True  True  True  True  True]
```

So, let me change the value of r to something maybe 0 point - 0.2 ok. Still all the roots have converged - 0.2, still all the roots have converged.

(Refer Slide Time: 18:02)

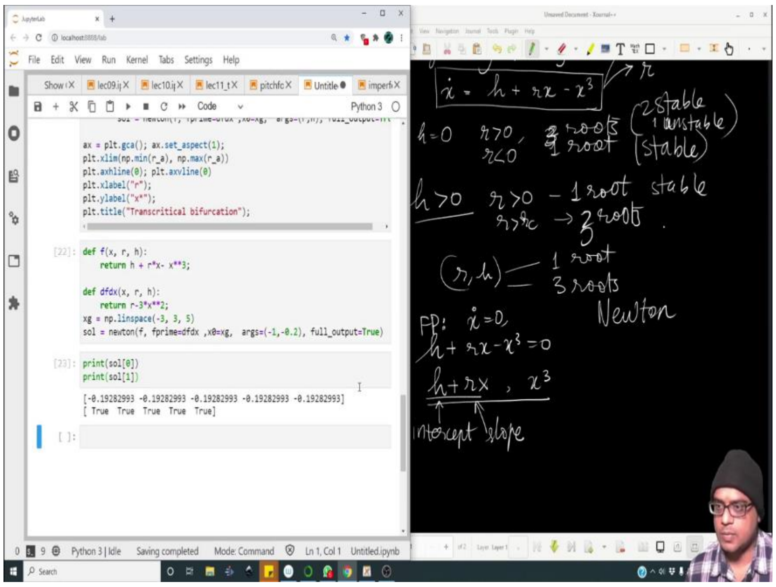


```
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

```
[20]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(-0.2,0.2), full_output=True)
```

```
[21]: print(sol[0])
      print(sol[1])

[-0.47251313 -0.47251313 -0.47251313 -0.47251313 -0.47251313]
[ True  True  True  True  True]
```

(Refer Slide Time: 18:11)

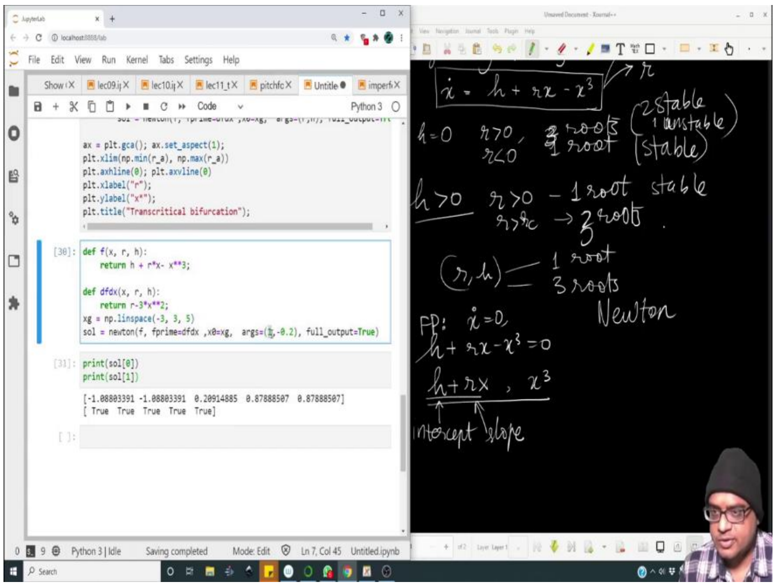


```
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

```
[22]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(-1,-0.2), full_output=True)
```

```
[23]: print(sol[0])
      print(sol[1])

[-0.19282993 -0.19282993 -0.19282993 -0.19282993 -0.19282993]
[ True  True  True  True  True]
```

(Refer Slide Time: 18:16)



```
[30]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(1,-0.2), full_output=True)
```

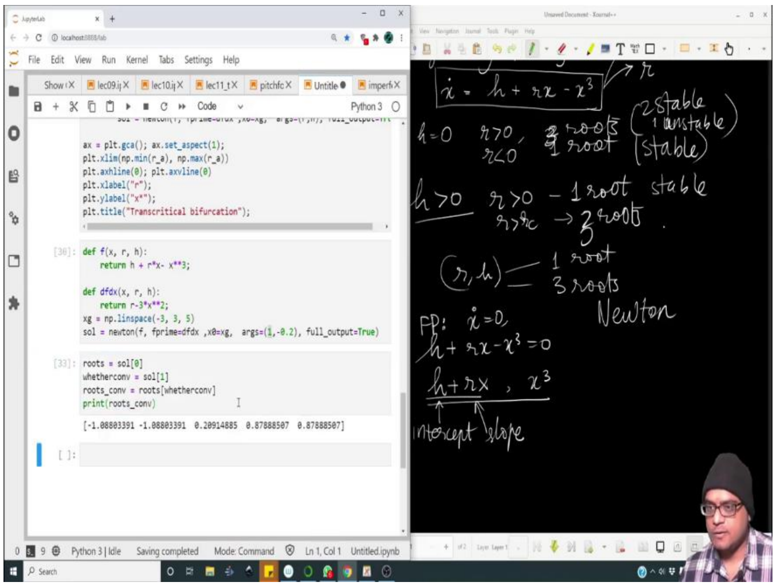
```
[31]: print(sol[0])
      print(sol[1])

[-1.08803391 -1.08803391  0.20914885  0.87888507  0.87888507]
[ True  True  True  True  True]
```

Still everything is converged ok. So, for this parametric set we have 2 roots not 2 roots we have 3 roots; one is this root, one is this root and one is this root and we necessarily give a large number of initial guest points.

So, that even if there are multiple roots we converge to those roots eventually. This has something to do with the Newtons basins of attraction ok. Regardless of where we start we are converging towards these roots. So, now, suppose one of them were to be false ok.

So, in case one of them were to be false it means that root whatever it has returned it is not actually a root which is truly returned ok. We have to discard that root.

(Refer Slide Time: 19:03)



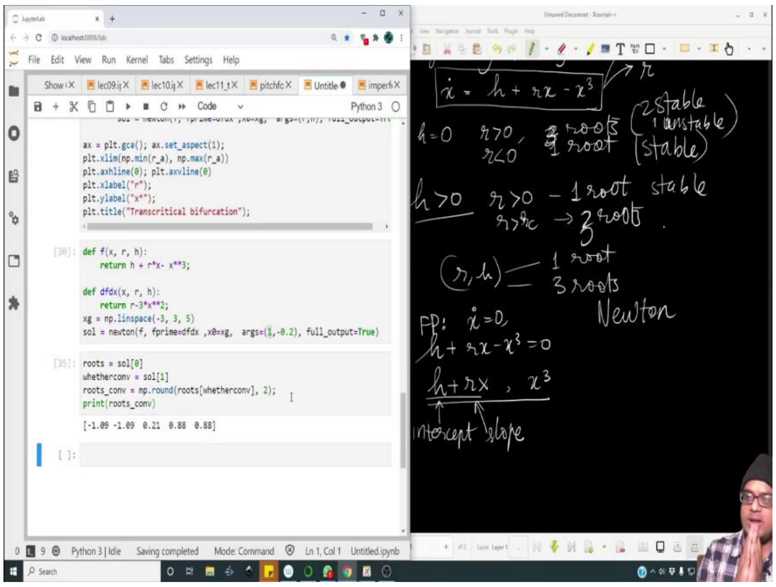
```
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

```
[30]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(1,-0.2), full_output=True)
```

```
[33]: roots = sol[0]
      whetherconv = sol[1]
      roots_conv = roots[whetherconv]
      print(roots_conv)

      [-1.08803391 -1.08803391  0.20914885  0.87888507  0.87888507]
```

So, then for extracting the true roots. So, roots = sol[0] whetherconv = sol[1] ok. So, now, these are the two things. So, now, roots which are converged will be roots of those indices which are true ok. Whether converged ok. So, now, roots converged will contain those root which have converged, but we want to find out the unique roots.

(Refer Slide Time: 19:50)



```
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

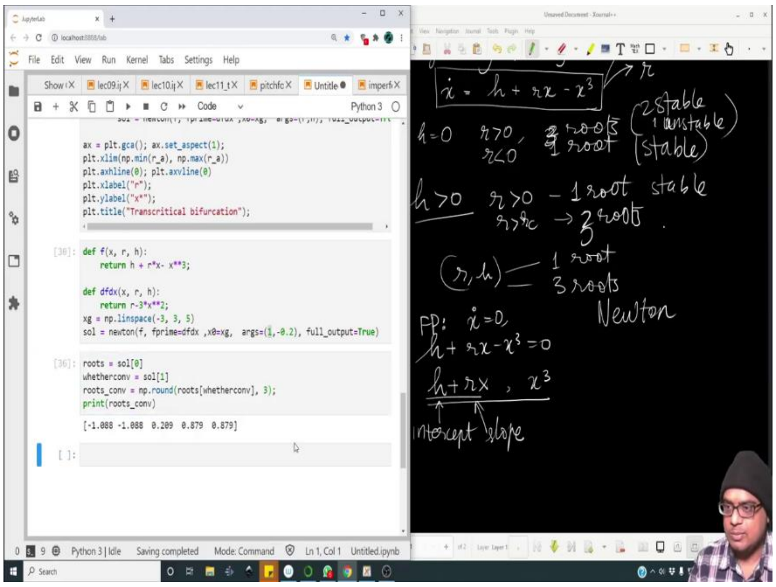
```
[30]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(1,-0.2), full_output=True)
```

```
[35]: roots = sol[0]
      whetherconv = sol[1]
      roots_conv = np.round(roots[whetherconv], 2);
      print(roots_conv)

      [-1.09 -1.09  0.21  0.88  0.88]
```

So, now, we will say np.round(roots[whetherconv], 2) let us say 2 decimal places. And the reason I do this is if the numerical algorithm has some issue in converging or if it converges to 10^{-8}. So, beyond that I mean although it is a is the same root in true mathematical sense, but the computer representation does not make it look like a same root.

If you do a = = b it will says false because it is maybe different at 10 to the power minus 12th place ok.

(Refer Slide Time: 20:36)



```
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

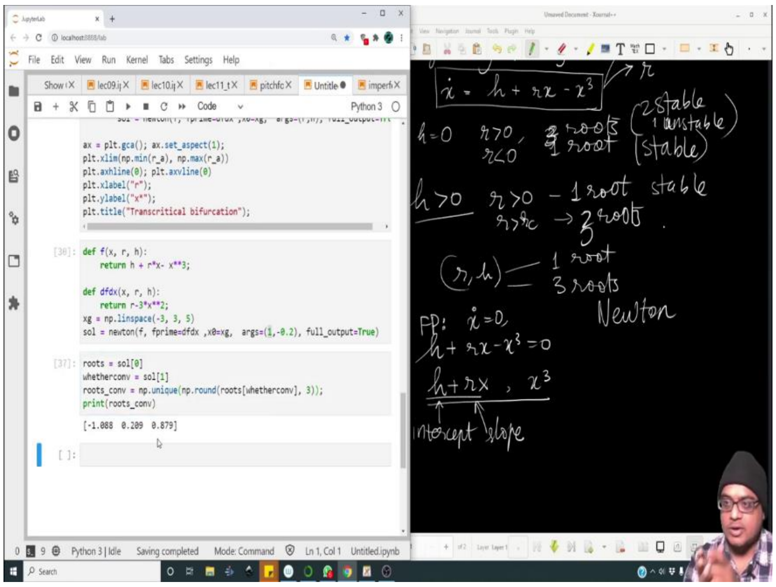
```
[30]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(1,-0.2), full_output=True)
```

```
[36]: roots = sol[0]
      whetherconv = sol[1]
      roots_conv = np.round(roots[whetherconv], 3);
      print(roots_conv)

      [-1.088 -1.088  0.209  0.879  0.879]
```

$\boxed{\dot{x} = h + rx - x^3} \to r$
$h=0$ $r>0$, $r<0$ 3 roots (2 stable 1 unstable) 1 root (stable)
$h>0$ $r>0$ - 1 root stable
$r>r_c \to$ 3 roots
(r,h) 1 root 3 roots
Newton
FP: $\dot{x}=0$,
$h+rx-x^3=0$
$h+rx$, x^3
intercept slope

So, in order to avoid that you can simply round it off to 3 decimal places it will still appear correct ok.

(Refer Slide Time: 20:42)



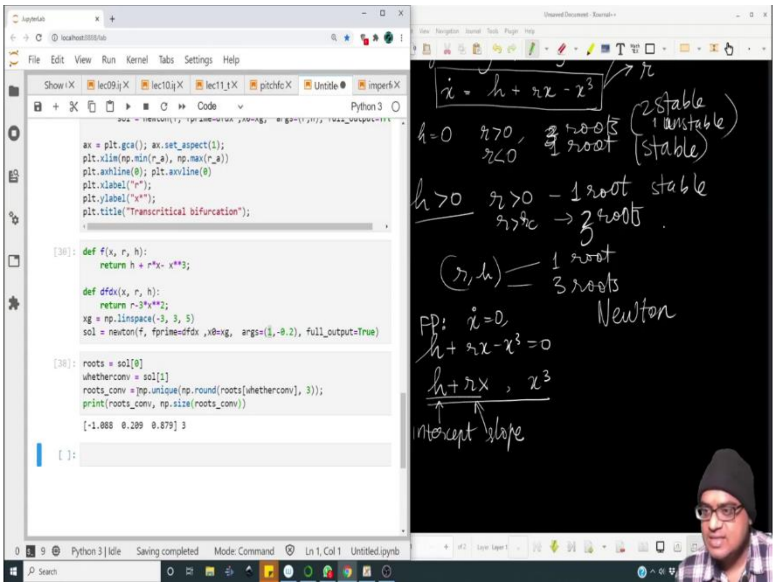
```
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

```
[30]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(1,-0.2), full_output=True)
```

```
[37]: roots = sol[0]
      whetherconv = sol[1]
      roots_conv = np.unique(np.round(roots[whetherconv], 3));
      print(roots_conv)

      [-1.088  0.209  0.879]
```

So, now, out of this we will extract the unique elements. So, np.unique. So, it has now given us only this set of unique roots.

(Refer Slide Time: 20:53)



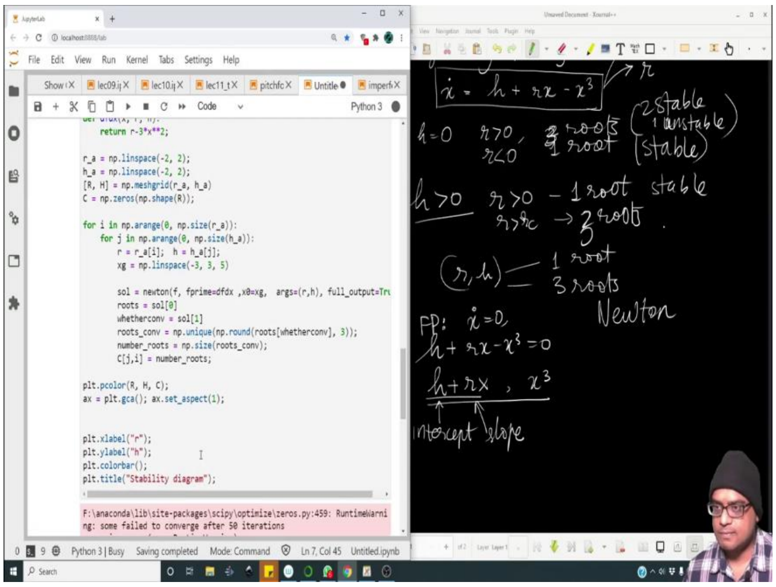
```
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

```
[30]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(1,-0.2), full_output=True)
```

```
[38]: roots = sol[0]
      whetherconv = sol[1]
      roots_conv = np.unique(np.round(roots[whetherconv], 3));
      print(roots_conv, np.size(roots_conv))

      [-1.088  0.209  0.879] 3
```

And so, now, let us print not only the roots which have converged, but also the number of roots; so, then np.size(roots_conv), excellent. So, now, we have all the things. So, this is the code that we are more interested in. So, we will copy this we will go over here. So, then for once we have the solution we will copy paste this thing.

(Refer Slide Time: 21:28)



```python
    return r-3*x**2;

r_a = np.linspace(-2, 2);
h_a = np.linspace(-2, 2);
[R, H] = np.meshgrid(r_a, h_a)
C = np.zeros(np.shape(R));

for i in np.arange(0, np.size(r_a)):
    for j in np.arange(0, np.size(h_a)):
        r = r_a[i];  h = h_a[j];
        xg = np.linspace(-3, 3, 5)

        sol = newton(f, fprime=dfdx ,x0=xg,  args=(r,h), full_output=Tru
        roots = sol[0]
        whetherconv = sol[1]
        roots_conv = np.unique(np.round(roots[whetherconv], 3));
        number_roots = np.size(roots_conv);
        C[j,i] = number_roots;

plt.pcolor(R, H, C);
ax = plt.gca(); ax.set_aspect(1);


plt.xlabel("r");
plt.ylabel("h");
plt.colorbar();
plt.title("Stability diagram");
```

```
F:\anaconda\lib\site-packages\scipy\optimize\zeros.py:459: RuntimeWarni
ng: some failed to converge after 50 iterations
```

$\boxed{\dot{x} = h + rx - x^3} \rightarrow r$

$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)

$r<0$ 1 root (stable)

$h>0$ $r>0$ - 1 root stable

$r>r_c \rightarrow$ 3 roots

(r, h) — 1 root, 3 roots

Newton

FP: $\dot{x}=0$,

$h + rx - x^3 = 0$

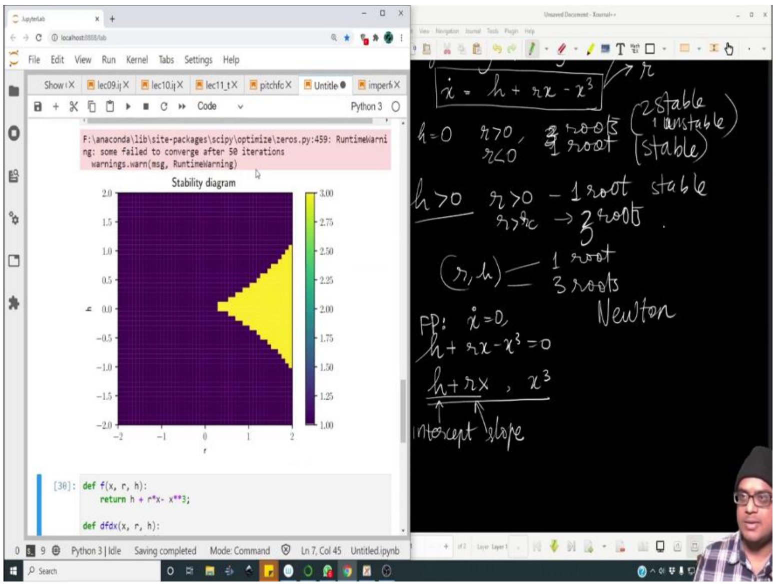
$h + rx$, x^3

intercept slope

So, roots will be sol[0] whether converged will be sol[1] roots converged will be this and number of roots will be np.size(roots_conv) and lastly we will assign everything to an array C. C[j,i] = number_roots and here we will define C= np.zeros(np.shape(R)).

So, this is just an initialization of the C array with zeros. So, now, np is the number of roots yeah that is pretty much it plt.pcolor(R, H, C) aspect ratio 1 we do not need the limit we do not need the x line y line. x label is x and y label is h; plt.colorbar and the title of the plot will be it will be bifurcation or the stability diagram. It is often called as stability diagram because it shows in the parameter space how many fixed points a given state has.

(Refer Slide Time: 22:56)



```
F:\anaconda\lib\site-packages\scipy\optimize\zeros.py:459: RuntimeWarning: some failed to converge after 50 iterations
  warnings.warn(msg, RuntimeWarning)
```

```
[30]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
```

$\dot{x} = h + rx - x^3 \rightarrow r$

$h=0$ $r>0$, 3 roots (2 stable, 1 unstable)

$r<0$, 1 root (stable)

$h>0$ $r>0$ − 1 root stable

$r>r_c$ → 3 roots .

(r,h) = 1 root / 3 roots

Newton

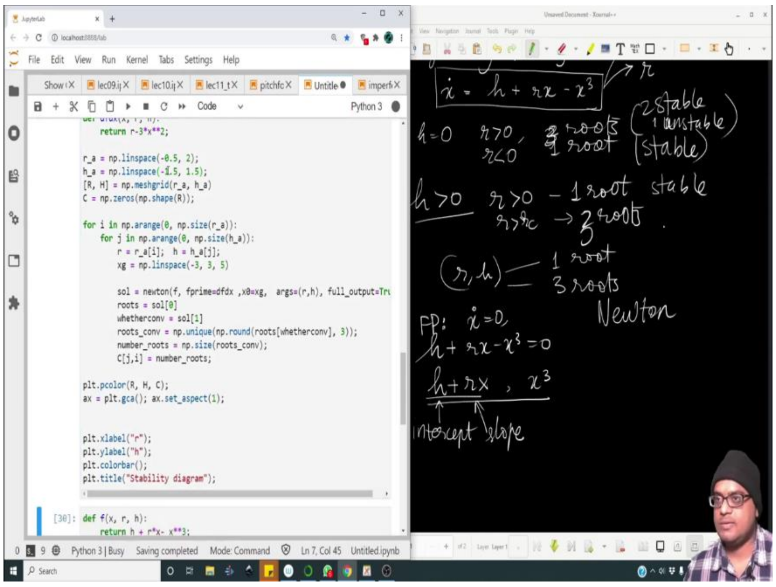
FP: $\dot{x}=0$

$h + rx - x^3 = 0$

$\underbrace{h+rx}_{\text{intercept}\ \ \text{slope}}$, x^3

So, let us run this so; obviously, there was some warning, but we ignore that warning for now. Maybe it started with this point where it is not able to converge. In fact, let us refine the grid. So, we see that something is happening over here.

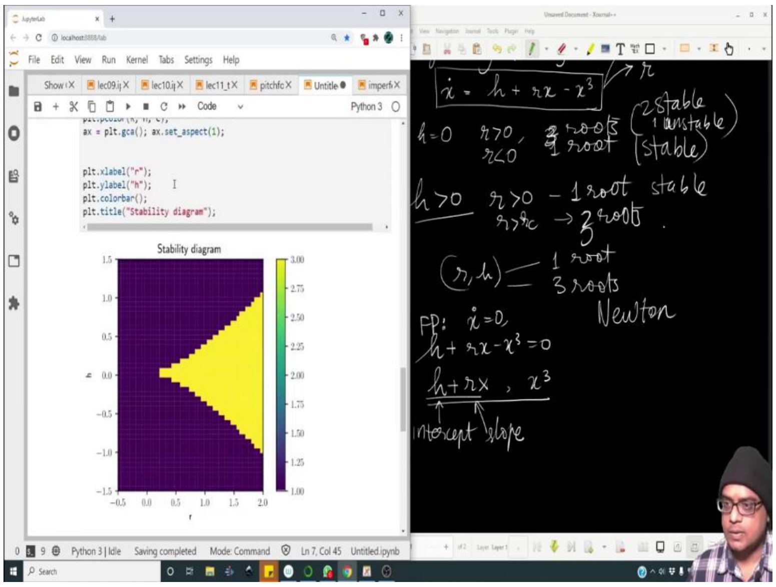
(Refer Slide Time: 23:26)



```
          return r-3*x**2;

      r_a = np.linspace(-0.5, 2);
      h_a = np.linspace(-1.5, 1.5);
      [R, H] = np.meshgrid(r_a, h_a)
      C = np.zeros(np.shape(R));

      for i in np.arange(0, np.size(r_a)):
          for j in np.arange(0, np.size(h_a)):
              r = r_a[i];  h = h_a[j];
              xg = np.linspace(-3, 3, 5)

              sol = newton(f, fprime=dfdx ,x0=xg,  args=(r,h), full_output=Tru
              roots = sol[0]
              whetherconv = sol[1]
              roots_conv = np.unique(np.round(roots[whetherconv], 3));
              number_roots = np.size(roots_conv);
              C[j,i] = number_roots;

      plt.pcolor(R, H, C);
      ax = plt.gca(); ax.set_aspect(1);


      plt.xlabel("r");
      plt.ylabel("h");
      plt.colorbar();
      plt.title("Stability diagram");

[30]: def f(x, r, h):
          return h + r*x- x**3;
```

So, let us define the grid from - 0.5 to 2 and - 1.5 to 1.5; - 0.5 to 2 and - 1.5 to 1.5. Let me run this.

(Refer Slide Time: 23:36)



```
ax = plt.gca(); ax.set_aspect(1);


plt.xlabel("r");
plt.ylabel("h");
plt.colorbar();
plt.title("Stability diagram");
```

$\boxed{\dot{x} = h + rx - x^3} \nearrow r$

$h=0 \quad r>0, \quad 3 \text{ roots} \begin{pmatrix} 2\text{ stable} \\ 1 \text{ unstable}\end{pmatrix}$

$\quad\quad\; r<0 \quad 1 \text{ root } (\text{stable})$

$\underline{h>0} \quad r>0 - 1 \text{ root stable}$

$\quad\quad\; r>r_c \rightarrow 3 \text{ roots}$.

(r,h) — 1 root, — 3 roots

Newton

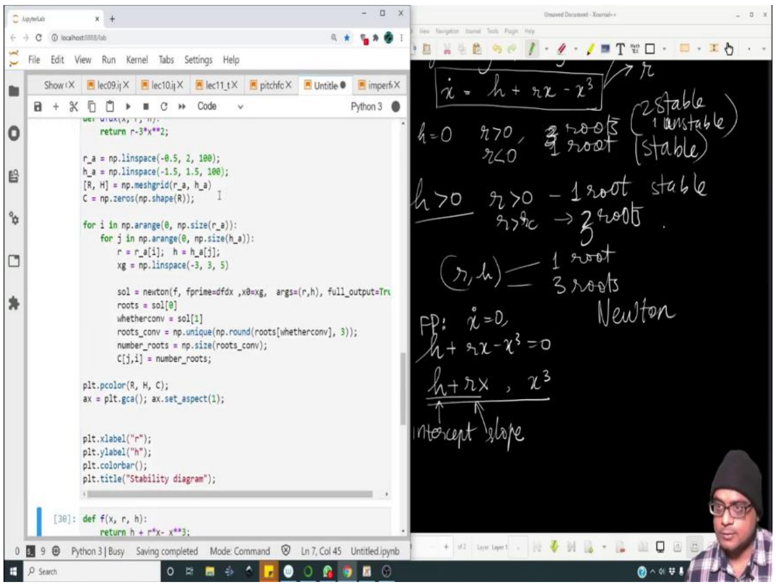
FP: $\dot{x}=0$,

$h + rx - x^3 = 0$

$\underline{\underbrace{h+rx}, \quad x^3}$

intercept slope

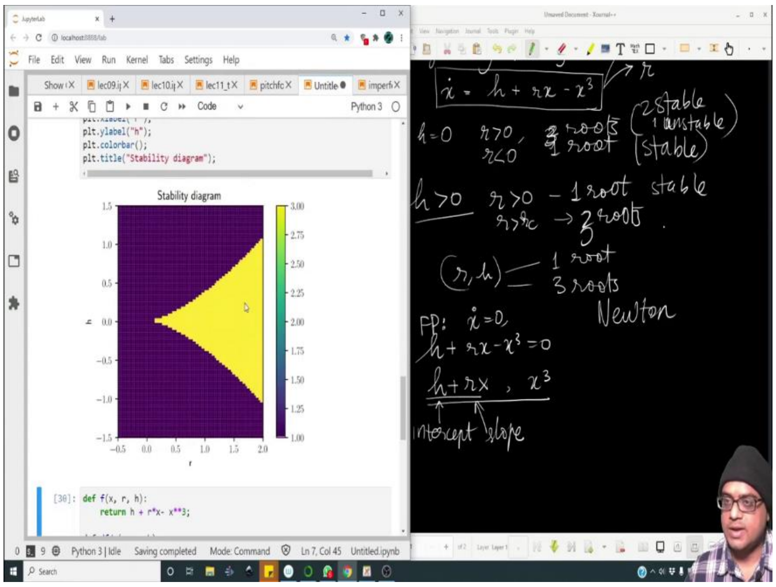
(Refer Slide Time: 23:42)



```
    return r-3*x**2;

r_a = np.linspace(-0.5, 2, 100);
h_a = np.linspace(-1.5, 1.5, 100);
[R, H] = np.meshgrid(r_a, h_a)
C = np.zeros(np.shape(R));

for i in np.arange(0, np.size(r_a)):
    for j in np.arange(0, np.size(h_a)):
        r = r_a[i];  h = h_a[j];
        xg = np.linspace(-3, 3, 5)

        sol = newton(f, fprime=dfdx ,x0=xg,  args=(r,h), full_output=Tru
        roots = sol[0]
        whetherconv = sol[1]
        roots_conv = np.unique(np.round(roots[whetherconv], 3));
        number_roots = np.size(roots_conv);
        C[j,i] = number_roots;

plt.pcolor(R, H, C);
ax = plt.gca(); ax.set_aspect(1);


plt.xlabel("r");
plt.ylabel("h");
plt.colorbar();
plt.title("Stability diagram");

[30]: def f(x, r, h):
    return h + r*x- x**3;
```

So, let me refine the grid, let me take 100 points, let me run this.

(Refer Slide Time: 23:54)



```
plt.ylabel("h");
plt.colorbar();
plt.title("Stability diagram");
```

```
[30]: def f(x, r, h):
          return h + r*x- x**3;
```

$\dot{x} = h + rx - x^3$

$h = 0$ $r > 0$, 3 roots (2 stable, 1 unstable); $r < 0$, 1 root (stable)

$h > 0$ $r > 0$ – 1 root stable; $r > r_c \to$ 3 roots.

(r, h) – 1 root, 3 roots   Newton

FP: $\dot{x} = 0$

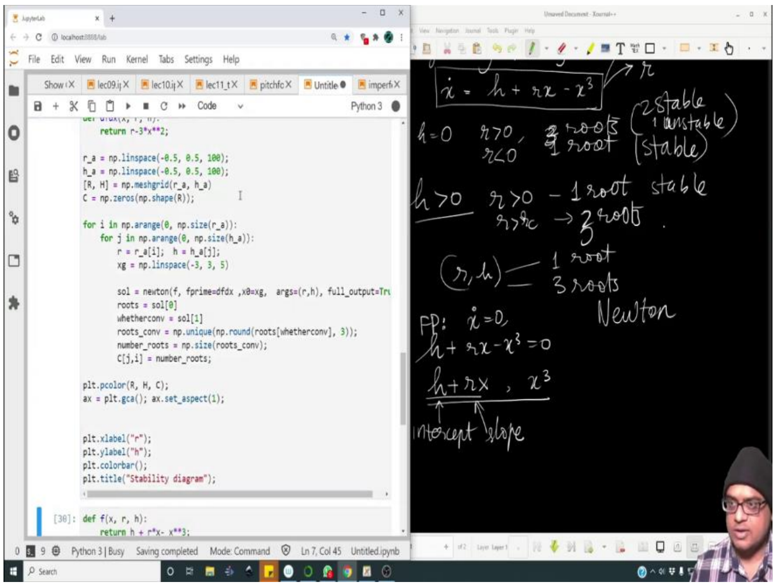
$h + rx - x^3 = 0$

$\underline{h + rx}$, x^3

intercept slope

And after a while we get this diagram. So, inside this there are 3 roots over here there are 1 root and this is the boundary which separates the zone from the set of parameters which have only 1 root from the set of parameters which have 3 roots. So, this shape is a cusp like shape of course, you can keep on refining this and get a pretty much refined shape. In fact, I can do this to show you what I mean.

(Refer Slide Time: 24:25)



```
    return r-3*x**2;

r_a = np.linspace(-0.5, 0.5, 100);
h_a = np.linspace(-0.5, 0.5, 100);
[R, H] = np.meshgrid(r_a, h_a)
C = np.zeros(np.shape(R));

for i in np.arange(0, np.size(r_a)):
    for j in np.arange(0, np.size(h_a)):
        r = r_a[i];  h = h_a[j];
        xg = np.linspace(-3, 3, 5)

        sol = newton(f, fprime=dfdx ,x0=xg,  args=(r,h), full_output=Tru
        roots = sol[0]
        whetherconv = sol[1]
        roots_conv = np.unique(np.round(roots[whetherconv], 3));
        number_roots = np.size(roots_conv);
        C[j,i] = number_roots;

plt.pcolor(R, H, C);
ax = plt.gca(); ax.set_aspect(1);


plt.xlabel("r");
plt.ylabel("h");
plt.colorbar();
plt.title("Stability diagram");
```

```
[30]: def f(x, r, h):
          return h + r*x- x**3;
```

$\dot{x} = h + rx - x^3$

$h = 0$ $r > 0$, 3 roots (2 stable, 1 unstable); $r < 0$, 1 root (stable)

$h > 0$ $r > 0$ – 1 root stable; $r > r_c \to$ 3 roots.

(r, h) – 1 root, 3 roots   Newton

FP: $\dot{x} = 0$

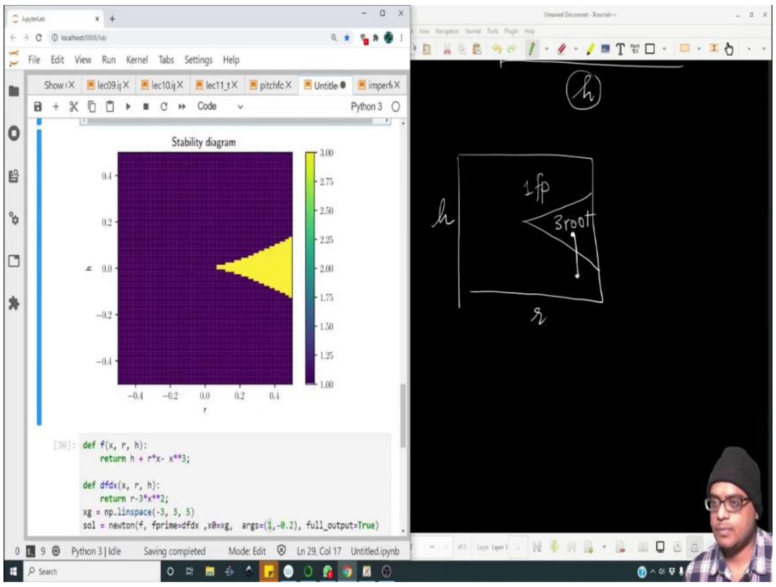
$h + rx - x^3 = 0$

$\underline{h + rx}$, x^3

intercept slope

But essentially what I am trying to say is the cusp shape is something which is a highlight of imperfect bifurcations.

(Refer Slide Time: 24:37)

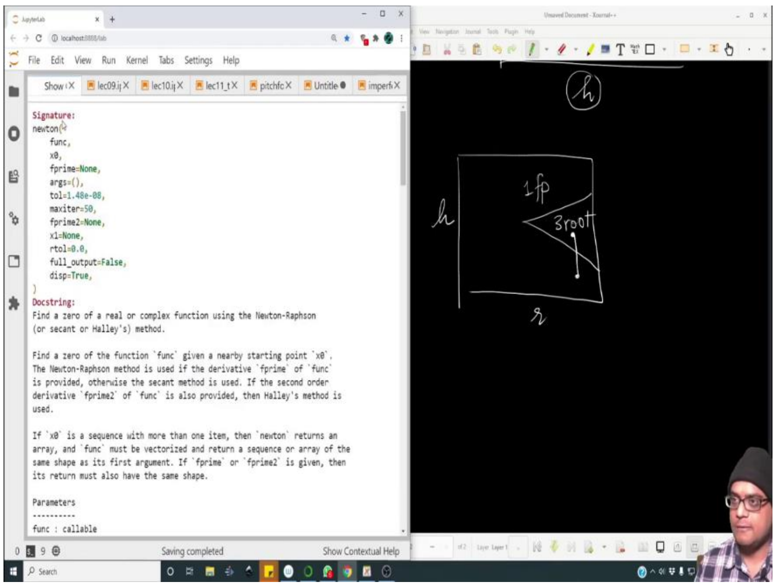


```
[30]: def f(x, r, h):
          return h + r*x- x**3;

      def dfdx(x, r, h):
          return r-3*x**2;
      xg = np.linspace(-3, 3, 5)
      sol = newton(f, fprime=dfdx ,x0=xg,  args=(1,-0.2), full_output=True)
```

There are 3 roots over here or the 3 fixed points over here and there is 1 fixed point ok. Something happens when we transit from a parameter space over here which has only 1 root over here which are certainly 3 roots. And now we are more interested in the stability of the roots.

So, now, let us proceed to sort of probe the stability of the roots. So, let us go back to our previous ok.

(Refer Slide Time: 25:31)

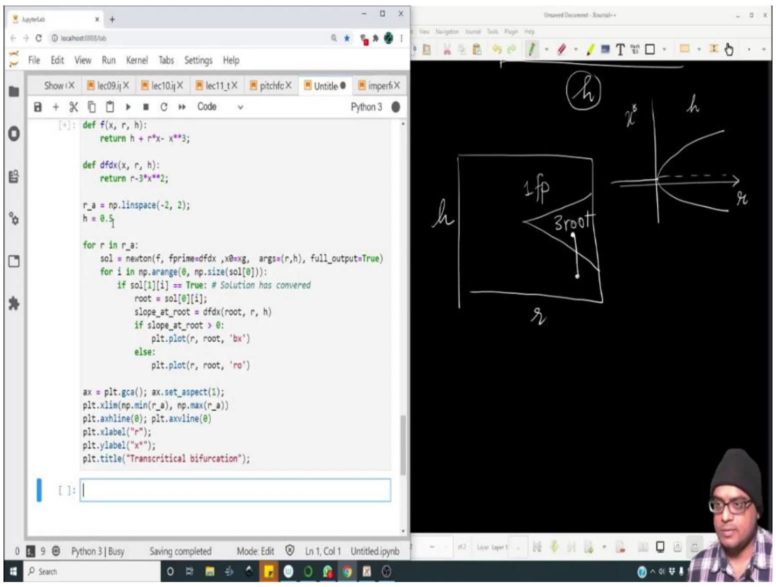


```
Signature:
newton(
    func,
    x0,
    fprime=None,
    args=(),
    tol=1.48e-08,
    maxiter=50,
    fprime2=None,
    x1=None,
    rtol=0.0,
    full_output=False,
    disp=True,
)
Docstring:
Find a zero of a real or complex function using the Newton-Raphson
(or secant or Halley's) method.

Find a zero of the function `func` given a nearby starting point `x0`.
The Newton-Raphson method is used if the derivative `fprime` of `func`
is provided, otherwise the secant method is used. If the second order
derivative `fprime2` of `func` is also provided, then Halley's method is
used.

If `x0` is a sequence with more than one item, then `newton` returns an
array, and `func` must be vectorized and return a sequence or array of the
same shape as its first argument. If `fprime` or `fprime2` is given, then
its return must also have the same shape.

Parameters
----------
func : callable
```

So, I do not we do not need this really, but with the help of this I could show you how the output of the Newton solver looks like. The Newton solver is slightly different from the fsolve. In fact, if you double click on Newton you can have a look at the contextual help of Jupyter Lab it says func x0 f prime and all these things and we have simply specified those in the function call ok. x naught, f prime, args and so on alright.

So, now, let us proceed to find out the bifurcation diagram for different values of h and r. So, let me delete the cells let me paste this alright.

(Refer Slide Time: 25:58)



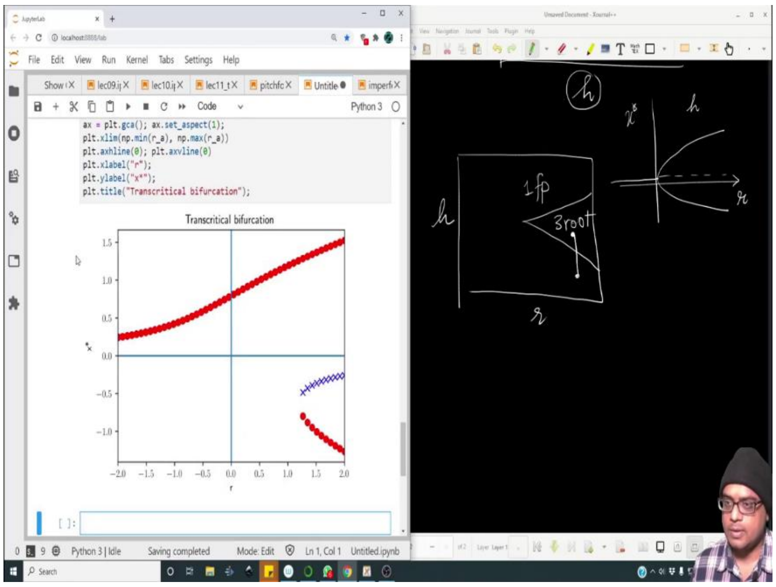
```
def f(x, r, h):
    return h + r*x- x**3;

def dfdx(x, r, h):
    return r-3*x**2;

r_a = np.linspace(-2, 2);
h = 0.5

for r in r_a:
    sol = newton(f, fprime=dfdx ,x0=xg,  args=(r,h), full_output=True)
    for i in np.arange(0, np.size(sol[0])):
        if sol[1][i] == True: # Solution has convered
            root = sol[0][i];
            slope_at_root = dfdx(root, r, h)
            if slope_at_root > 0:
                plt.plot(r, root, 'bx')
            else:
                plt.plot(r, root, 'ro')

ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

So, now we have to replace this by the Newton solver. Let me copy this line let me replace the fsolve by the Newton solver alright. Let me replace the function definitions from over here alright. So, we have changed the function definitions and let me change the looping as well alright h.

So, let us do it for fixed value of h over r. So, meaning we are trying to do this; x star versus r for various values of h. So, we know that when h is 0 we will have the pitchfork bifurcation something like this ok. So, now we have to make some more changes. So, the strategy is for i in np.arange.

So, this line so, let me indent everything for i in np dot this if sol[1][i] = true and this is from the previous cells that I deleted. So, root = sol[0][i] slope at root. So, we have to pass h as well. Now if it is greater than 0 yeah ok.

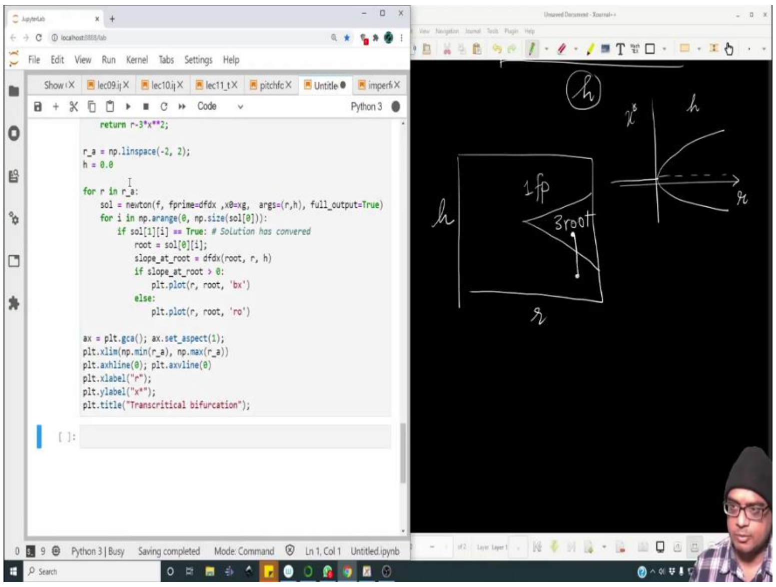
(Refer Slide Time: 27:49)



```
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

So, let me run this. So, let me define h = 0.5, let me start from h = 0.

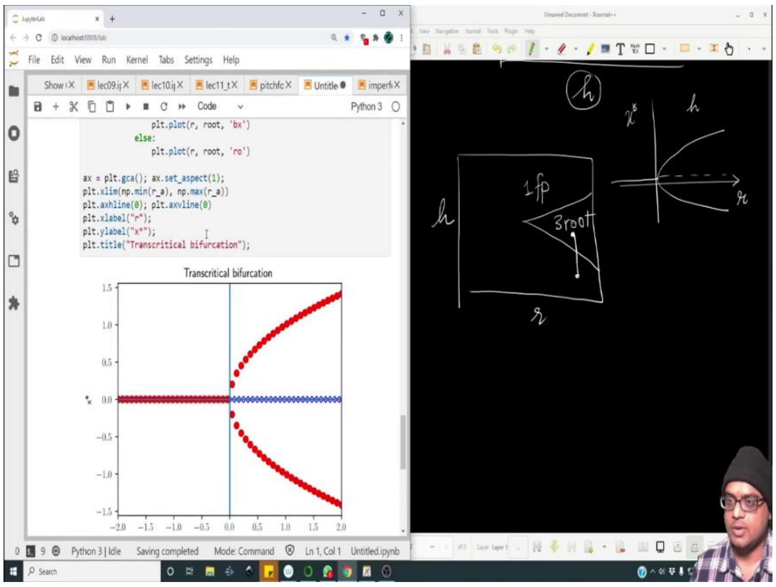
(Refer Slide Time: 27:58)



```
    return r-3*x**2;

r_a = np.linspace(-2, 2);
h = 0.0

for r in r_a:
    sol = newton(f, fprime=dfdx ,x0=xg,  args=(r,h), full_output=True)
    for i in np.arange(0, np.size(sol[0])):
        if sol[1][i] == True: # Solution has convered
            root = sol[0][i];
            slope_at_root = dfdx(root, r, h)
            if slope_at_root > 0:
                plt.plot(r, root, 'bx')
            else:
                plt.plot(r, root, 'ro')

ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

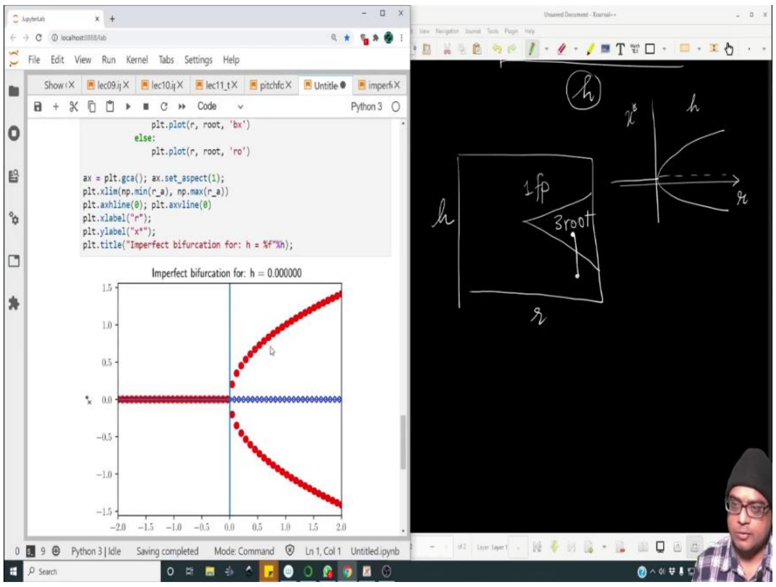
So, let me not jump the gun. So, let me start from h = 0 because it is a simple pitch for bifurcation excellent.

(Refer Slide Time: 28:00)



```
            plt.plot(r, root, 'bx')
        else:
            plt.plot(r, root, 'ro')

ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation");
```

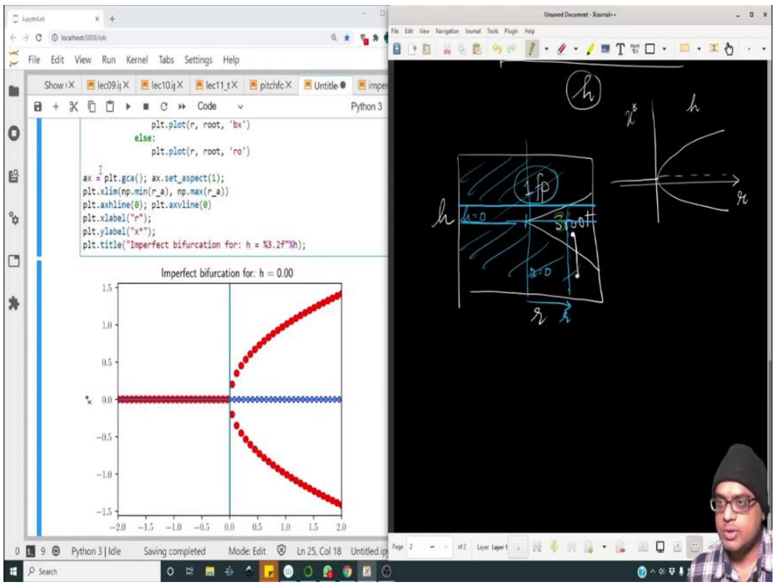
(Refer Slide Time: 28:11)



```
            plt.plot(r, root, 'bx')
        else:
            plt.plot(r, root, 'ro')

ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Imperfect bifurcation for: h = %f"%h);
```

So, this is the diagram of you have to change this as well. This is %f and over here we will print the value of h. Imperfect bifurcation for h = this alright. Let me run this again. So, where h = 0 we have something like this.

(Refer Slide Time: 28:34)



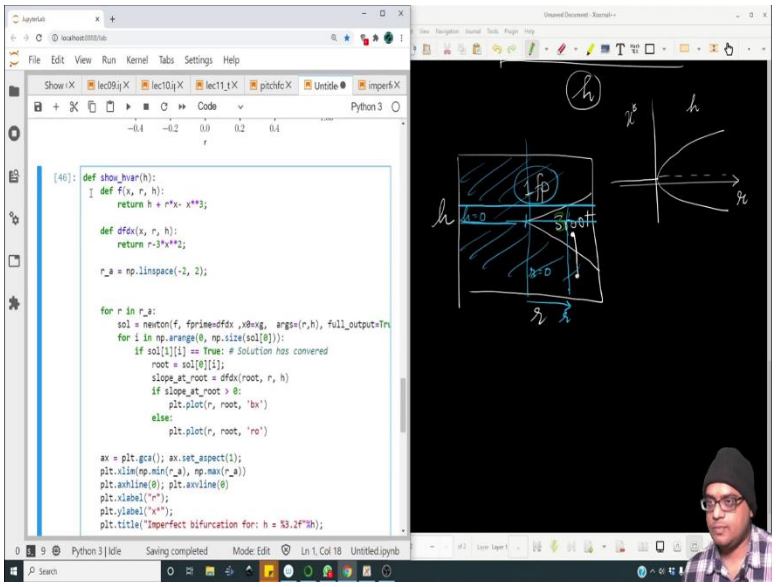
```
                plt.plot(r, root, 'bx')
            else:
                plt.plot(r, root, 'ro')

ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Imperfect bifurcation for: h = %3.2f"%h);
```

In fact, let me print only 2 decimal places ok. So, now let me choose a positive value of h. So, we are now proving this entire line and watch carefully what happens I mean. If you look very if you observe very keenly, we are actually transecting this line and we are seeing that up until this. So, this was r = 0 and this is obviously, h = 0. So, up until this point we have only 1 root and this entire zone is 1 root and over here we have 3 roots ok.

So, up until this point we have 1 root and then beyond this we have 3 roots. So, this checks out. Now when we increase the value of h over here. We will see that for a long for a larger value of r we will have only 1 root and then we will have 3 roots appearing.

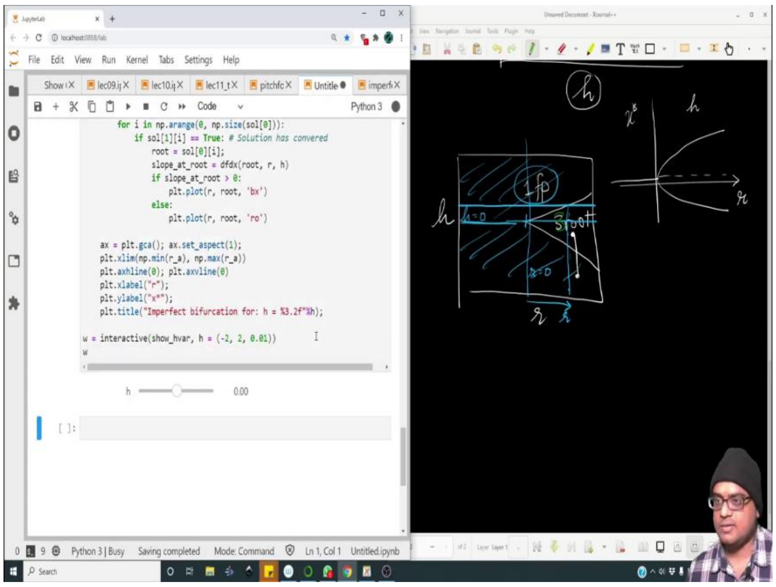
So, let me redo the plot for a higher value of h; in fact, why do not we put all of this inside an interactive function. Let me remove the value of h from this, let me wrap everything inside an interactive function.

(Refer Slide Time: 30:06)



```
[46]: def show_hvar(h):
          def f(x, r, h):
              return h + r*x- x**3;

          def dfdx(x, r, h):
              return r-3*x**2;

          r_a = np.linspace(-2, 2);


          for r in r_a:
              sol = newton(f, fprime=dfdx ,x0=xg,  args=(r,h), full_output=Tru
              for i in np.arange(0, np.size(sol[0])):
                  if sol[1][i] == True: # Solution has convered
                      root = sol[0][i];
                      slope_at_root = dfdx(root, r, h)
                      if slope_at_root > 0:
                          plt.plot(r, root, 'bx')
                      else:
                          plt.plot(r, root, 'ro')

          ax = plt.gca(); ax.set_aspect(1);
          plt.xlim(np.min(r_a), np.max(r_a))
          plt.axhline(0); plt.axvline(0)
          plt.xlabel("r");
          plt.ylabel("x*");
          plt.title("Imperfect bifurcation for: h = %3.2f"%h);
```

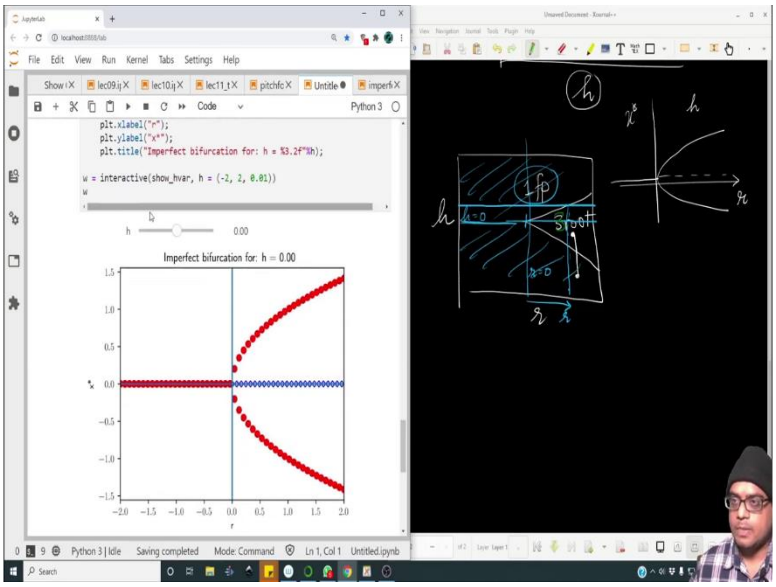
(Refer Slide Time: 30:15)



```
            for i in np.arange(0, np.size(sol[0])):
                if sol[1][i] == True: # Solution has convered
                    root = sol[0][i];
                    slope_at_root = dfdx(root, r, h)
                    if slope_at_root > 0:
                        plt.plot(r, root, 'bx')
                    else:
                        plt.plot(r, root, 'ro')

        ax = plt.gca(); ax.set_aspect(1);
        plt.xlim(np.min(r_a), np.max(r_a))
        plt.axhline(0); plt.axvline(0)
        plt.xlabel("r");
        plt.ylabel("x*");
        plt.title("Imperfect bifurcation for: h = %3.2f"%h);

w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

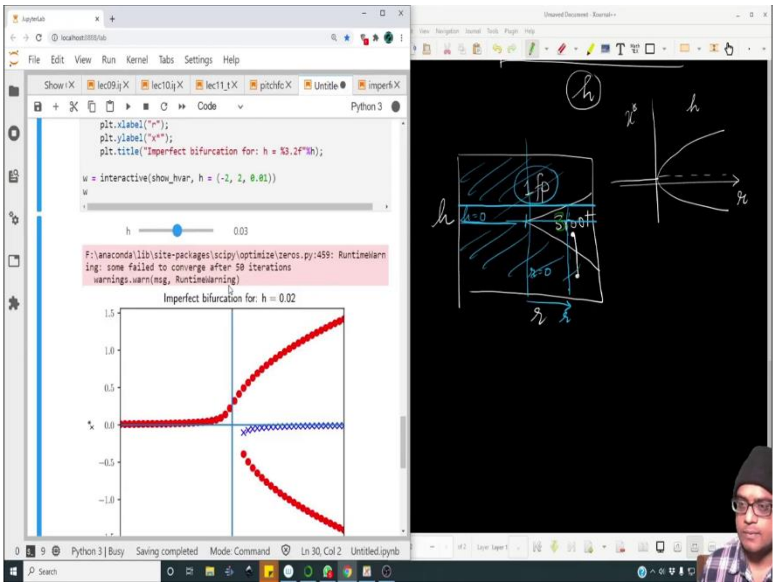
So, def show h variation with the input will be h. Let us make an interactive widget alright. Let me we have to show the widget as well excellent. So, when h increases, let us see what happens ok.

(Refer Slide Time: 30:35)



```
    plt.xlabel("r");
    plt.ylabel("x*");
    plt.title("Imperfect bifurcation for: h = %3.2f"%h);

w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

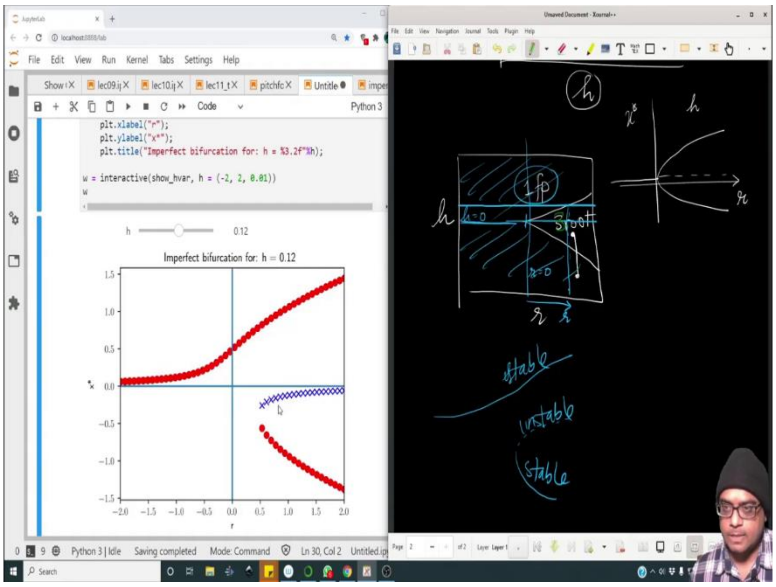
(Refer Slide Time: 30:40)



```
    plt.xlabel("r");
    plt.ylabel("x*");
    plt.title("Imperfect bifurcation for: h = %3.2f"%h);

w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

```
F:\anaconda\lib\site-packages\scipy\optimize\zeros.py:459: RuntimeWarning: some failed to converge after 50 iterations
  warnings.warn(msg, RuntimeWarning)
```

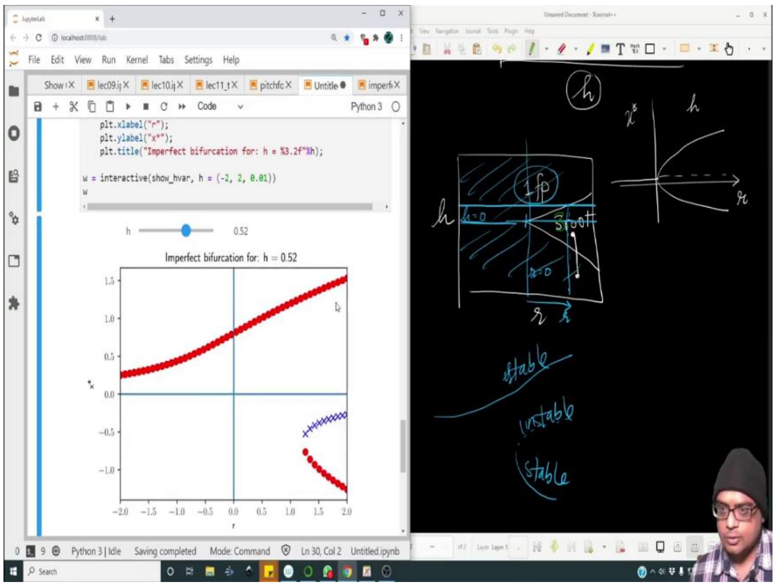
(Refer Slide Time: 30:50)



```
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Imperfect bifurcation for: h = %3.2f"%h);

w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

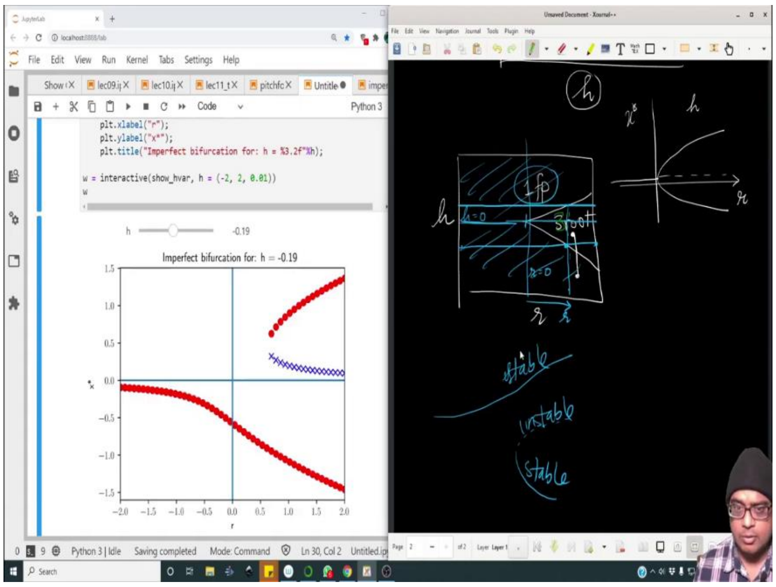
So, we see that for a larger value of r we have only 1 root and then we have 3 roots. Out of which the upper branch is still stable there is a unstable branch and a stable branch. So, this is unstable and this is stable ok. So, this is how the 3 roots look like alright.

(Refer Slide Time: 31:23)



```
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Imperfect bifurcation for: h = %3.2f"%h);

w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

(Refer Slide Time: 31:32)

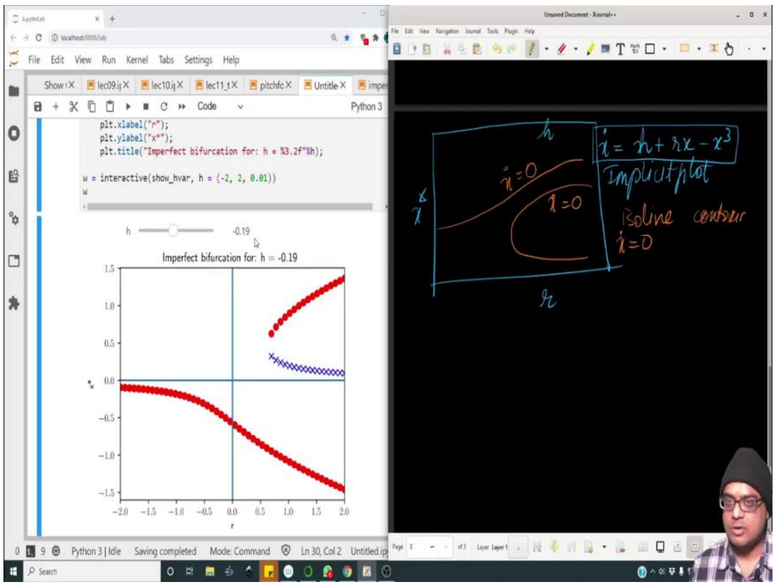


```
    plt.xlabel("r");
    plt.ylabel("x*");
    plt.title("Imperfect bifurcation for: h = %3.2f"%h);

w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

So, now, when we keep increasing this; this is how it looks ok. Same diagram or be it with a mirror symmetry it happens for negative values of h ok. Obviously, for this value also we will have a similar behavior, but the root behavior will be reflected about the x axis ok.

So, this is how the roots look like. In fact, we have plotted the stability of the points and all, but there was I mean there is obviously, a much easier way of plotting at least the fixed points ok.

(Refer Slide Time: 32:06)



```
    plt.xlabel("r");
    plt.ylabel("x*");
    plt.title("Imperfect bifurcation for: h = %3.2f"%h);

w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

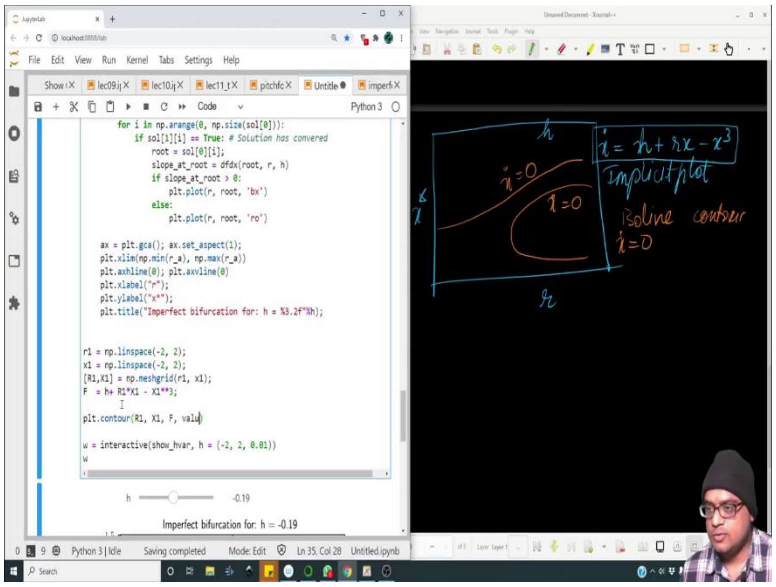
So, what are we drawing actually? We are drawing the fixed points as a function of r for various values of h. The equation that we are looking at is $h + rx - x^3$ and we have looked in one of the earliest lectures in which we were looking at geometric interpretations of this and we had made something called as an implicit plot.

So, we can find out the fixed points using a very simple implicit plot as well. What is then implicit plot? We will draw over this entire domain the isoline or the contour of $\dot{x} = 0$. We will try to find the locus of curves along which $\dot{x}$ will be = 0.

And those locus should be exactly these points, but just having the locus will not tell us whether that particular isoline corresponds to a stable fixed point or an unstable fixed point, but what we can do is we can later on pass those x and y values of the isoline check the slope and then change the color or something. I am not going to do that now. I am just going to show you the isolines ok.

I am just going to show you how we can make use of the implicit plot. I have not done this for the other plots, but it is really easy to understand. So, let me do this.

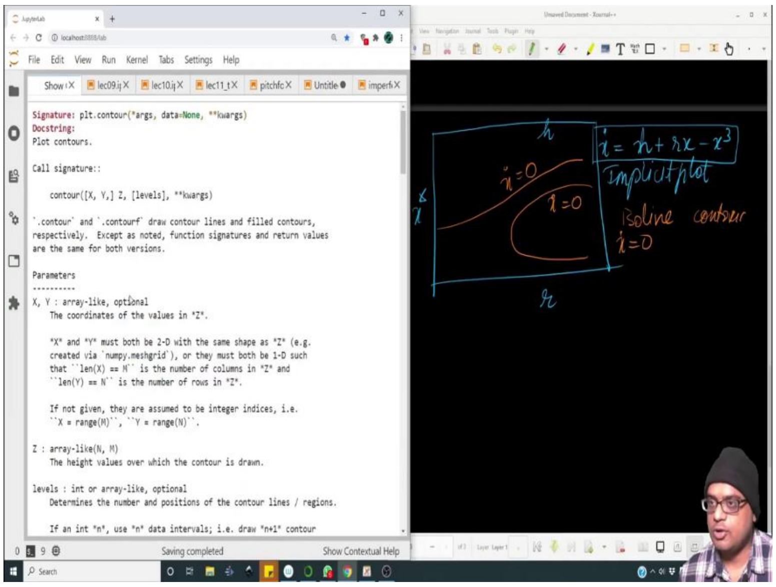
(Refer Slide Time: 33:47)



```
            for i in np.arange(0, np.size(sol[0])):
                if sol[1][i] == True: # Solution has convered
                    root = sol[0][i];
                    slope_at_root = dfdx(root, r, h)
                    if slope_at_root > 0:
                        plt.plot(r, root, 'bx')
                    else:
                        plt.plot(r, root, 'ro')

        ax = plt.gca(); ax.set_aspect(1);
        plt.xlim(np.min(r_a), np.max(r_a))
        plt.axhline(0); plt.axvline(0)
        plt.xlabel("r");
        plt.ylabel("x*");
        plt.title("Imperfect bifurcation for: h = %3.2f"%h);


r1 = np.linspace(-2, 2);
x1 = np.linspace(-2, 2);
[R1,X1] = np.meshgrid(r1, x1);
F  = h+ R1*X1 - X1**3;

plt.contour(R1, X1, F, valu)

w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

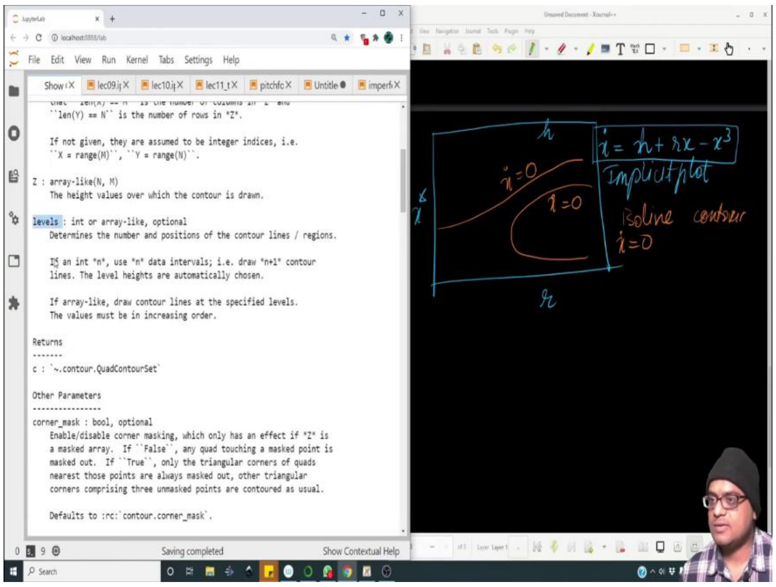
So, r1 = np.linspace(-2,2) x1 = np.linspace(-2,2) again then we will have [R1 , X1] = np .meshgrid(r1,x1). Now we will define the function $F = h + R1 * X1 – X1^3$. In fact, this has to be R1 then we will do plt.contour(R1 , X1 , F) value. So, just to be clear let us go to the contextual help of contour(x,y,z) and what is the extra.

(Refer Slide Time: 34:44)



```
Signature: plt.contour(*args, data=None, **kwargs)
Docstring:
Plot contours.

Call signature::

    contour([X, Y,] Z, [levels], **kwargs)

`.contour` and `.contourf` draw contour lines and filled contours,
respectively.  Except as noted, function signatures and return values
are the same for both versions.

Parameters
----------
X, Y : array-like, optional
    The coordinates of the values in *Z*.

    *X* and *Y* must both be 2-D with the same shape as *Z* (e.g.
    created via `numpy.meshgrid`), or they must both be 1-D such
    that ``len(X) == M`` is the number of columns in *Z* and
    ``len(Y) == N`` is the number of rows in *Z*.

    If not given, they are assumed to be integer indices, i.e.
    ``X = range(M)``, ``Y = range(N)``.

Z : array-like(N, M)
    The height values over which the contour is drawn.

levels : int or array-like, optional
    Determines the number and positions of the contour lines / regions.

    If an int *n*, use *n* data intervals; i.e. draw *n+1* contour
```

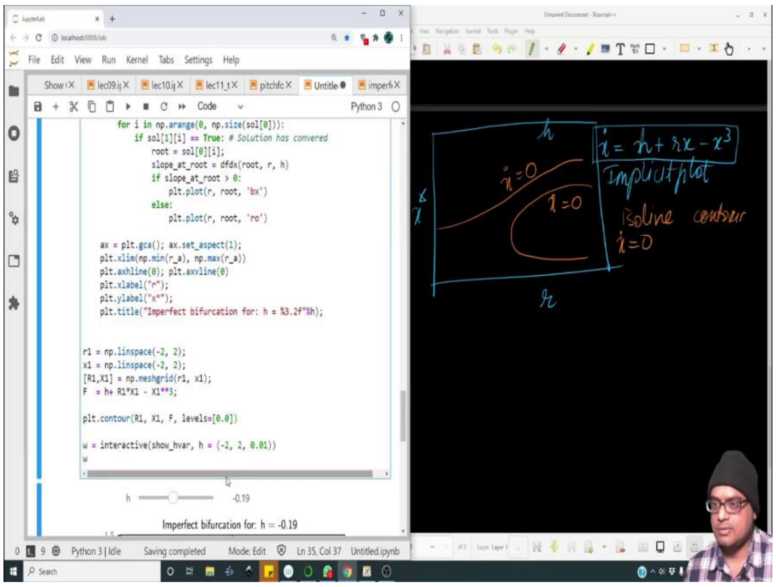
(Refer Slide Time: 34:58)



```
    ``len(Y) == N`` is the number of rows in *Z*.

    If not given, they are assumed to be integer indices, i.e.
    ``X = range(M)``, ``Y = range(N)``.

Z : array-like(N, M)
    The height values over which the contour is drawn.

levels : int or array-like, optional
    Determines the number and positions of the contour lines / regions.

    If an int *n*, use *n* data intervals; i.e. draw *n+1* contour
    lines. The level heights are automatically chosen.

    If array-like, draw contour lines at the specified levels.
    The values must be in increasing order.

Returns
-------
c : `~.contour.QuadContourSet`

Other Parameters
----------------
corner_mask : bool, optional
    Enable/disable corner masking, which only has an effect if *Z* is
    a masked array.  If ``False``, any quad touching a masked point is
    masked out.  If ``True``, only the triangular corners of quads
    nearest those points are always masked out, other triangular
    corners comprising three unmasked points are contoured as usual.

    Defaults to :rc:`contour.corner_mask`.
```

So, in order to draw certain levels, we can ask the computer to plot for us only a certain level.

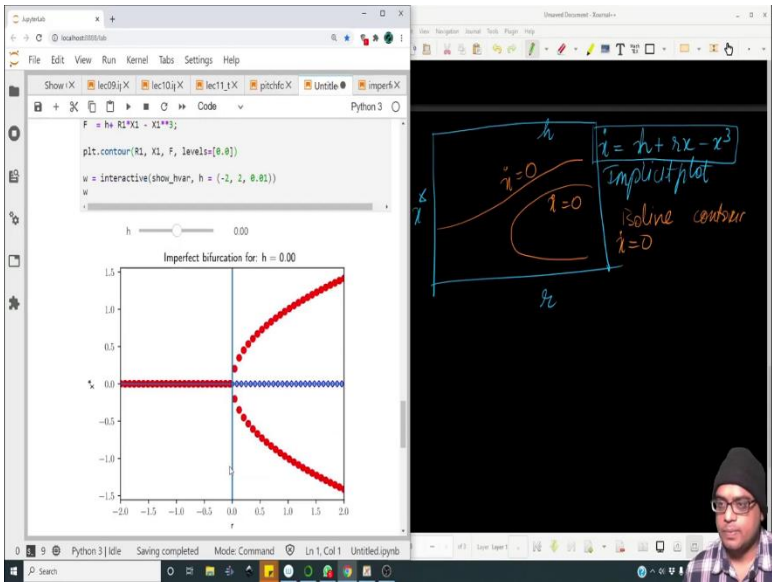
(Refer Slide Time: 35:04)



```
            for i in np.arange(0, np.size(sol[0])):
                if sol[1][i] == True: # Solution has convered
                    root = sol[0][i];
                    slope_at_root = dfdx(root, r, h)
                    if slope_at_root > 0:
                        plt.plot(r, root, 'bx')
                    else:
                        plt.plot(r, root, 'ro')

        ax = plt.gca(); ax.set_aspect(1);
        plt.xlim(np.min(r_a), np.max(r_a))
        plt.axhline(0); plt.axvline(0)
        plt.xlabel("r");
        plt.ylabel("x*");
        plt.title("Imperfect bifurcation for: h = %3.2f"%h);


r1 = np.linspace(-2, 2);
x1 = np.linspace(-2, 2);
[R1,X1] = np.meshgrid(r1, x1);
F  = h+ R1*X1 - X1**3;

plt.contour(R1, X1, F, levels=[0.0])

w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

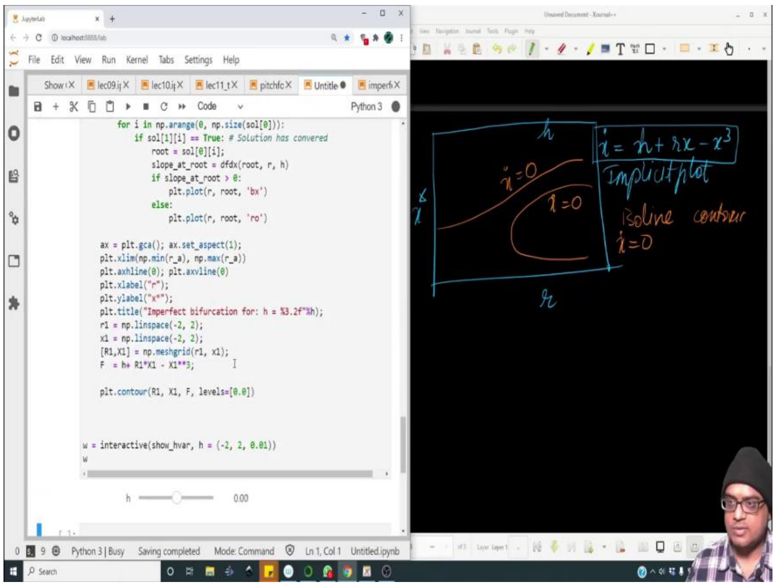
So, over here we will pass levels = 0.0 meaning it will plot for us the 0 line.

(Refer Slide Time: 35:12)



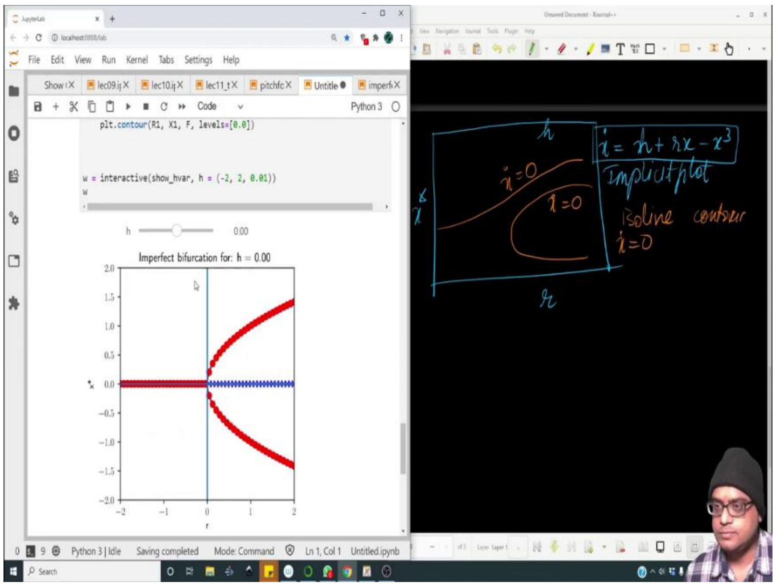
```
F  = h+ R1*X1 - X1**3;

plt.contour(R1, X1, F, levels=[0.0])

w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

(Refer Slide Time: 35:28)



```
        for i in np.arange(0, np.size(sol[0])):
            if sol[1][i] == True: # Solution has convered
                root = sol[0][i];
                slope_at_root = dfdx(root, r, h)
                if slope_at_root > 0:
                    plt.plot(r, root, 'bx')
                else:
                    plt.plot(r, root, 'ro')

    ax = plt.gca(); ax.set_aspect(1);
    plt.xlim(np.min(r_a), np.max(r_a))
    plt.axhline(0); plt.axvline(0)
    plt.xlabel("r");
    plt.ylabel("x*");
    plt.title("Imperfect bifurcation for: h = %3.2f"%h);
    r1 = np.linspace(-2, 2);
    x1 = np.linspace(-2, 2);
    [R1,X1] = np.meshgrid(r1, x1);
    F  = h+ R1*X1 - X1**3;

    plt.contour(R1, X1, F, levels=[0.0])



w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

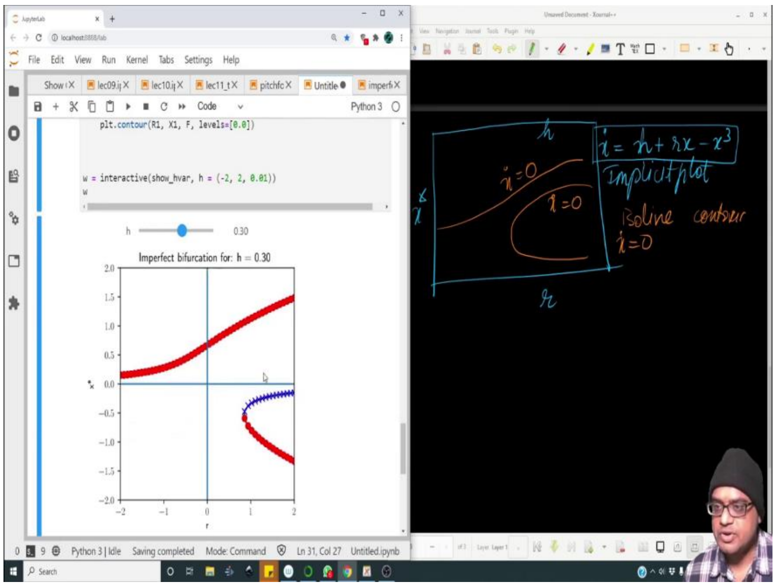
(Refer Slide Time: 35:33)



```
    plt.contour(R1, X1, F, levels=[0.0])



w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

So, let me run this ok. So, all this is outside the function. So, we must include them inside the function let me run this again ok.

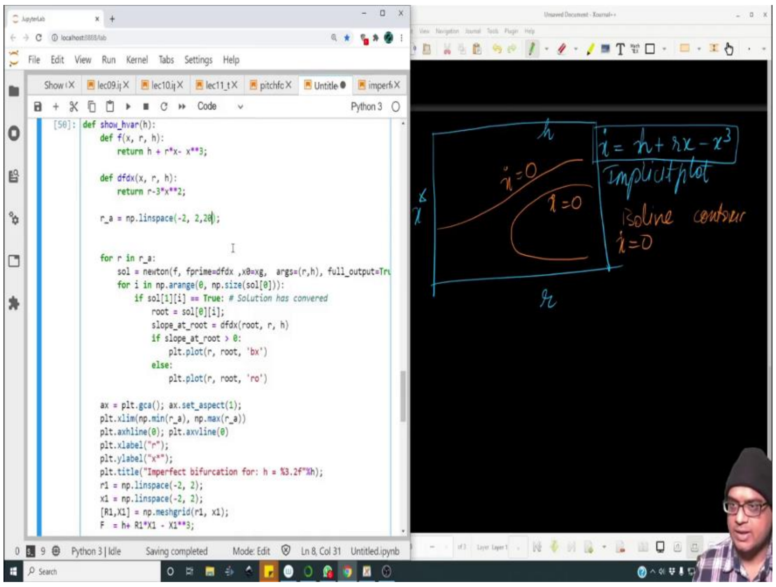
(Refer Slide Time: 35:38)



```
    plt.contour(R1, X1, F, levels=[0.0])


w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

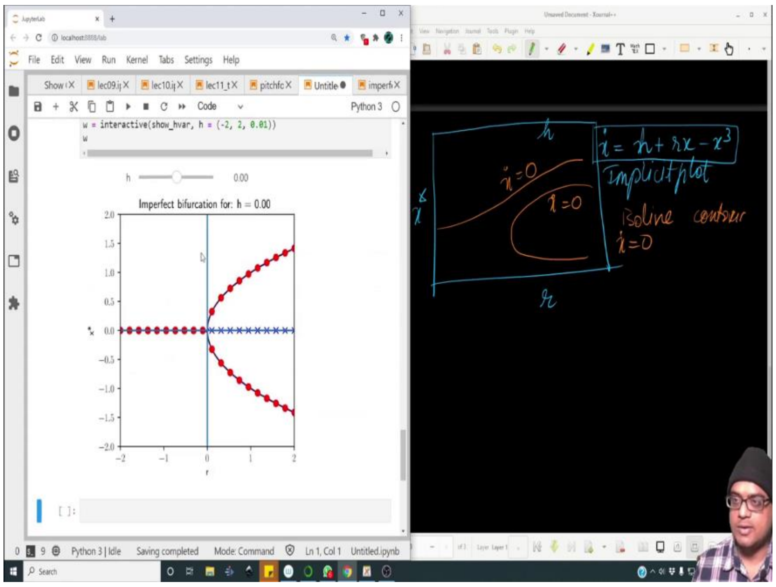
So, let me reduce the number of points we have in the r_a so that we can clearly see the overlapping of the 2 things.

(Refer Slide Time: 35:44)



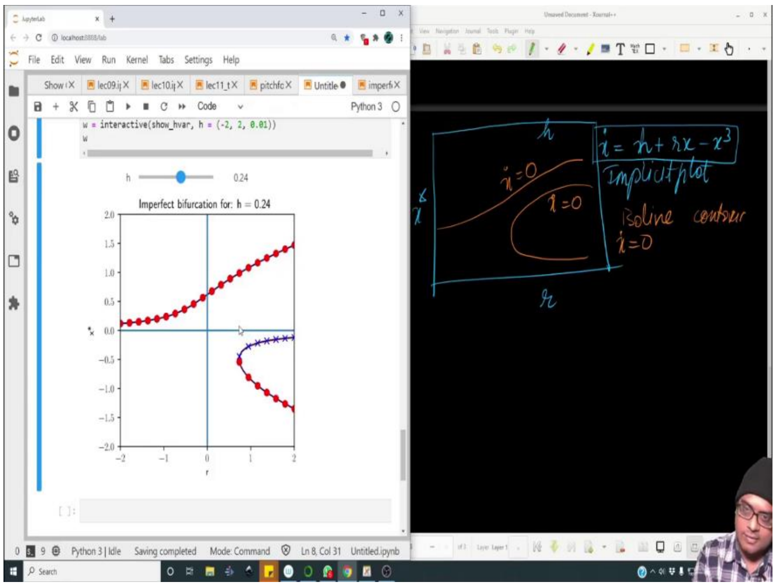
```
[50]: def show_hvar(h):
          def f(x, r, h):
              return h + r*x- x**3;

          def dfdx(x, r, h):
              return r-3*x**2;

          r_a = np.linspace(-2, 2,20);


          for r in r_a:
              sol = newton(f, fprime=dfdx ,x0=xg,  args=(r,h), full_output=Tru
              for i in np.arange(0, np.size(sol[0])):
                  if sol[1][i] == True: # Solution has convered
                      root = sol[0][i];
                      slope_at_root = dfdx(root, r, h)
                      if slope_at_root > 0:
                          plt.plot(r, root, 'bx')
                      else:
                          plt.plot(r, root, 'ro')

          ax = plt.gca(); ax.set_aspect(1);
          plt.xlim(np.min(r_a), np.max(r_a))
          plt.axhline(0); plt.axvline(0)
          plt.xlabel("r");
          plt.ylabel("x*");
          plt.title("Imperfect bifurcation for: h = %3.2f"%h);
          r1 = np.linspace(-2, 2);
          x1 = np.linspace(-2, 2);
          [R1,X1] = np.meshgrid(r1, x1);
          F  = h+ R1*X1 - X1**3;
```

(Refer Slide Time: 35:50)



```
w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

(Refer Slide Time: 35:54)



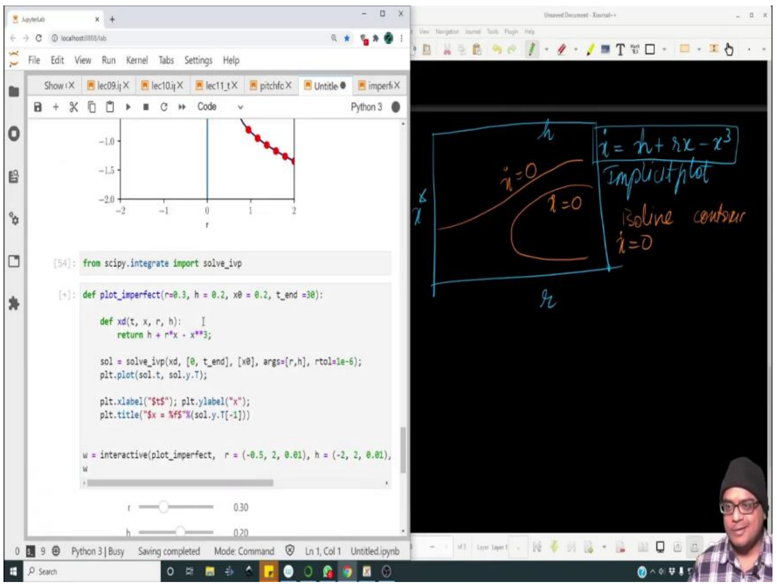
```
w = interactive(show_hvar, h = (-2, 2, 0.01))
w
```

That is let me just make 20 ok. So, let me run this great. So, now, we can see how the contour lines and the stability and the instability of the points are reflected in a single plot ok. So, I hope through this little exercise you will be in a position to figure out for any given equation or a vector flow what the fixed points or the contour of the fixed points should look like ok.

And with the help of the other snippet you will be able to determine whether those points are fixed stable points or fixed unstable points. In fact, by making use of the Newton solver you wish you should be able to find out the shape of the curve without actually using the implicit plots.

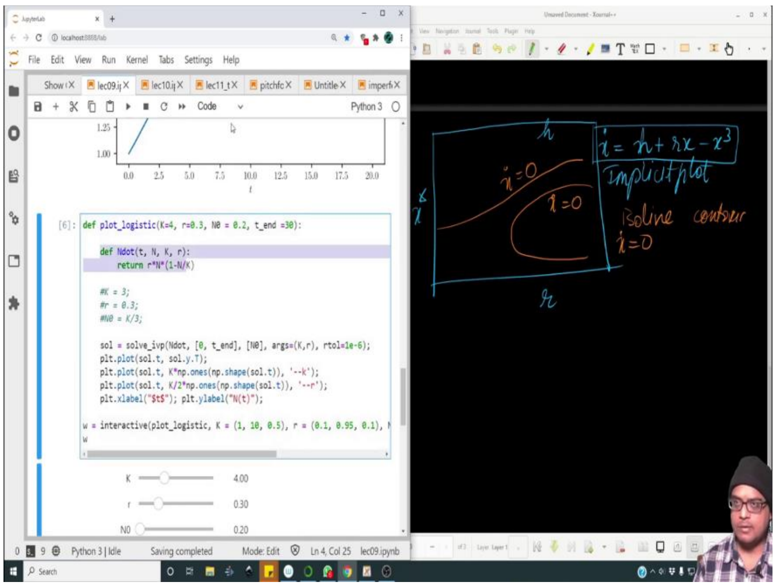
By using the Newton solver you will have the exact values by using those exact values and finding out the slope of the vector flow with respect to x, you should be in a position to tell whether that point will be stable or unstable.

(Refer Slide Time: 36:55)



```
[54]: from scipy.integrate import solve_ivp

[*]: def plot_imperfect(r=0.3, h = 0.2, x0 = 0.2, t_end =30):

         def xd(t, x, r, h):
             return h + r*x - x**3;

         sol = solve_ivp(xd, [0, t_end], [x0], args=[r,h], rtol=1e-6);
         plt.plot(sol.t, sol.y.T);

         plt.xlabel("$t$"); plt.ylabel("x");
         plt.title("$x = %f$"%(sol.y.T[-1]))


     w = interactive(plot_imperfect,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
     w
```

(Refer Slide Time: 37:02)

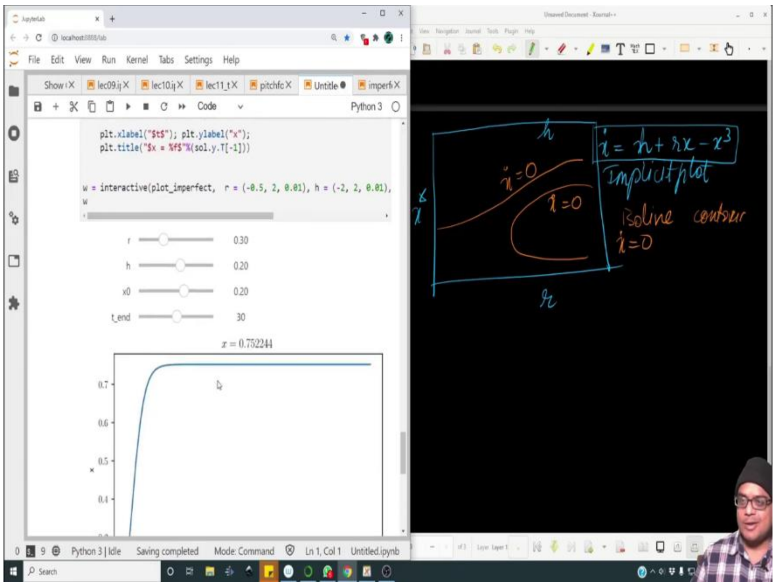


```
[6]: def plot_logistic(K=4, r=0.3, N0 = 0.2, t_end =30):

         def Ndot(t, N, K, r):
             return r*N*(1-N/K)

         #K = 3;
         #r = 0.3;
         #N0 = K/3;

         sol = solve_ivp(Ndot, [0, t_end], [N0], args=(K,r), rtol=1e-6);
         plt.plot(sol.t, sol.y.T);
         plt.plot(sol.t, K*np.ones(np.shape(sol.t)), '--k');
         plt.plot(sol.t, K/2*np.ones(np.shape(sol.t)), '--r');
         plt.xlabel("$t$"); plt.ylabel("N(t)");

     w = interactive(plot_logistic, K = (1, 10, 0.5), r = (0.1, 0.95, 0.1), N
     w
```

I have written up a small snippet actually I have basically copied whatever we had done in lecture 9 for the logistic plot and I have changed all the functional forms it appears over here. So, basically it is a it is a function which passes r h initial condition and the end time and we are solving the initial value problem of this particular form and we are plotting it for various values of r and h.

So, let me run this let us try to ok. So, solve_ivp we have to import from scipy.integrate. So, from scipy.integrate import solve_ivp oops. So, that is the question. I have written that line anyway.

(Refer Slide Time: 38:05)

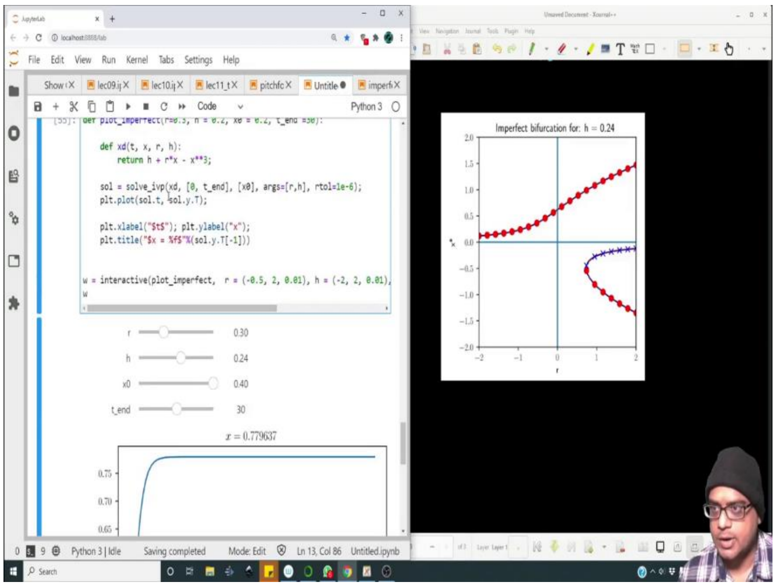


```
    plt.xlabel("$t$"); plt.ylabel("x");
    plt.title("$x = %f$"%(sol.y.T[-1]))


w = interactive(plot_imperfect,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
w
```

```
r      0.30
h      0.20
x0     0.20
t_end  30
```

So, yeah after running this let us see for h = 0.24. Let me copy this over here so that we can make sense of whatever is going on.

(Refer Slide Time: 38:23)

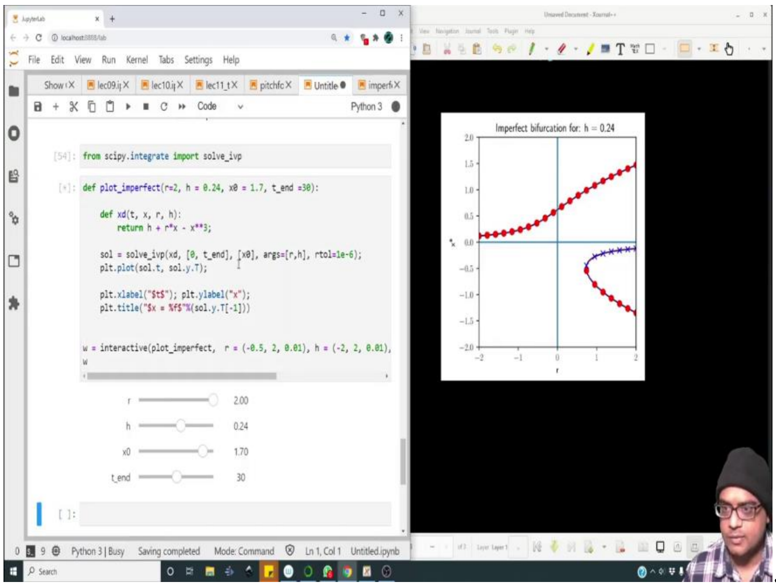


```
[55]: def plot_imperfect(r=0.3, h = 0.2, x0 = 0.2, t_end =30):

          def xd(t, x, r, h):
              return h + r*x - x**3;

          sol = solve_ivp(xd, [0, t_end], [x0], args=[r,h], rtol=1e-6);
          plt.plot(sol.t, sol.y.T);

          plt.xlabel("$t$"); plt.ylabel("x");
          plt.title("$x = %f$"%(sol.y.T[-1]))


      w = interactive(plot_imperfect,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
      w
```

```
      r      0.30
      h      0.24
     x0      0.40
  t_end      30

$x = 0.779637$
```

So, that we can you can make sense of whatever is going on. So, let h be 0.24. Now let us change r. So, let me go from this positive value ok. So, the fixed point is 1.5. So, possibly we need to change the fixed point to the initial condition further. So, let me increase this to 2 alright.

(Refer Slide Time: 39:19)

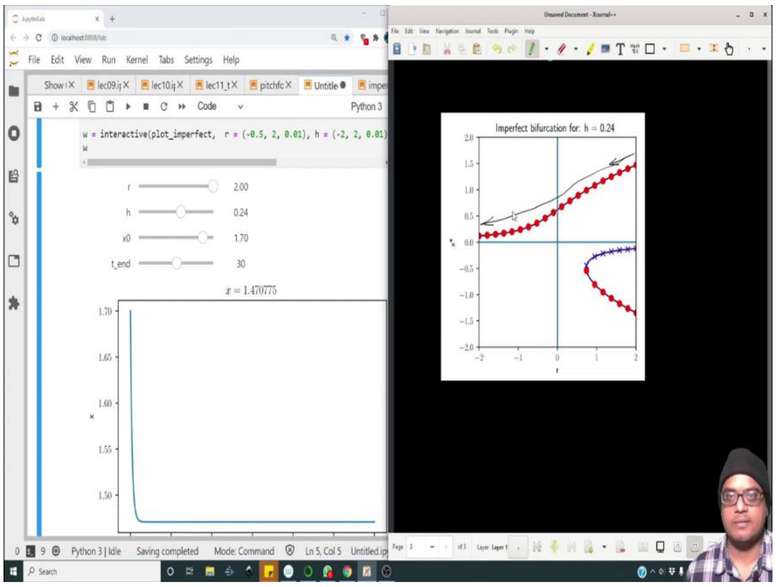


```
[54]: from scipy.integrate import solve_ivp

[*]:  def plot_imperfect(r=2, h = 0.24, x0 = 1.7, t_end =30):

          def xd(t, x, r, h):
              return h + r*x - x**3;

          sol = solve_ivp(xd, [0, t_end], [x0], args=[r,h], rtol=1e-6);
          plt.plot(sol.t, sol.y.T);

          plt.xlabel("$t$"); plt.ylabel("x");
          plt.title("$x = %f$"%(sol.y.T[-1]))


      w = interactive(plot_imperfect,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
      w
```

```
      r      2.00
      h      0.24
     x0      1.70
  t_end      30
```

Let me pass h = 0.24 as the default value and let the default value of r be = 2. So, let me let this be also 1.7. So, that with this initial condition default initial condition h and r we have an asymptotic value of something close to 1.5.

(Refer Slide Time: 39:50)



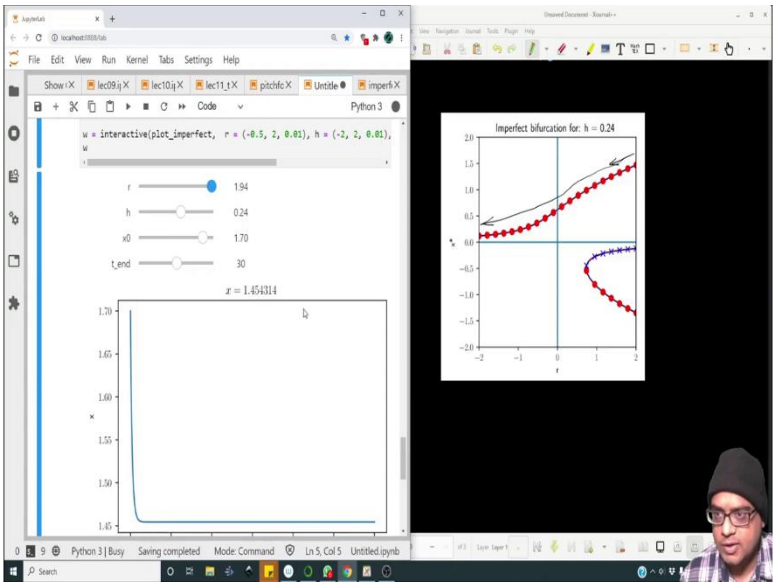
```
w = interactive(plot_imperfect,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01)
w
```

r 2.00
h 0.24
x0 1.70
t_end 30

$x = 1.470775$

So, x is 1.47 something, good. So, now, let us reduce the value of r. So, as we are reducing the value of r, we should be able to obtain these particular sets of fixed points.

(Refer Slide Time: 40:14)

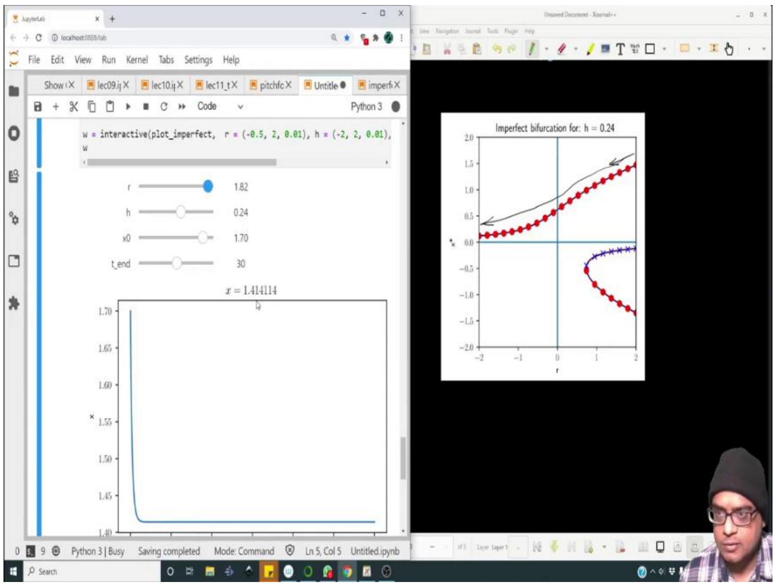


```
w = interactive(plot_imperfect,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
w
```

r 1.94
h 0.24
x0 1.70
t_end 30

$x = 1.454314$

(Refer Slide Time: 40:18)

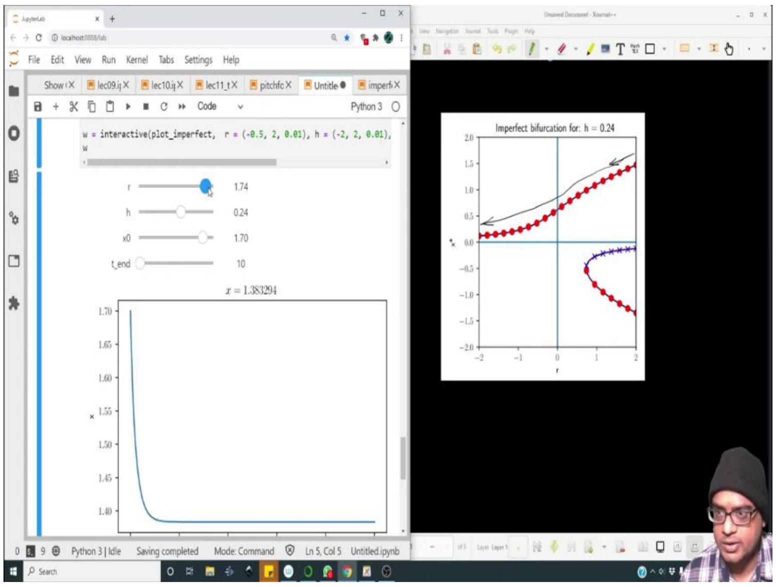


```
w = interactive(plot_imperfect, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
w
```

```
r      1.82
h      0.24
x0     1.70
t_end  30
```

$x = 1.414114$

(Refer Slide Time: 40:21)



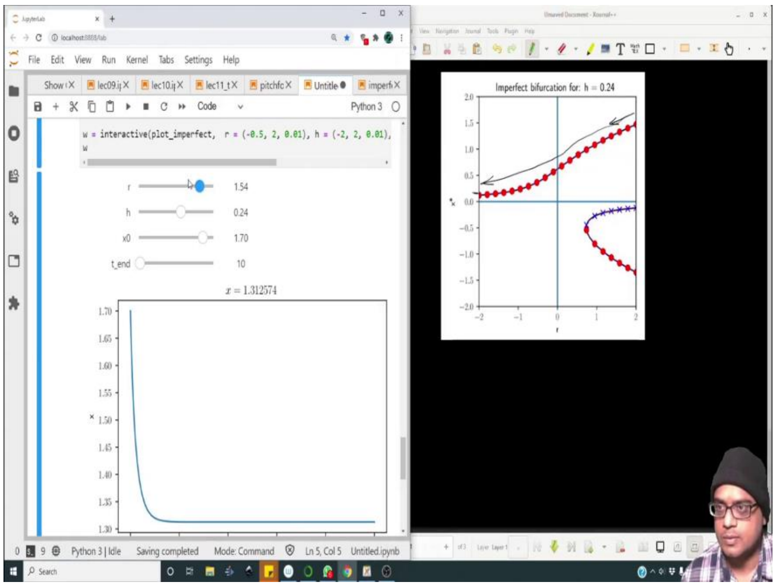
```
w = interactive(plot_imperfect, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
w
```

```
r      1.74
h      0.24
x0     1.70
t_end  10
```

$x = 1.383294$

Let us see. So, x is 1.45, x is 1.41 let me reduce t end. So, that we do not have to do so many integrations ok.

(Refer Slide Time: 40:29)


```
w = interactive(plot_imperfect, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
w
```

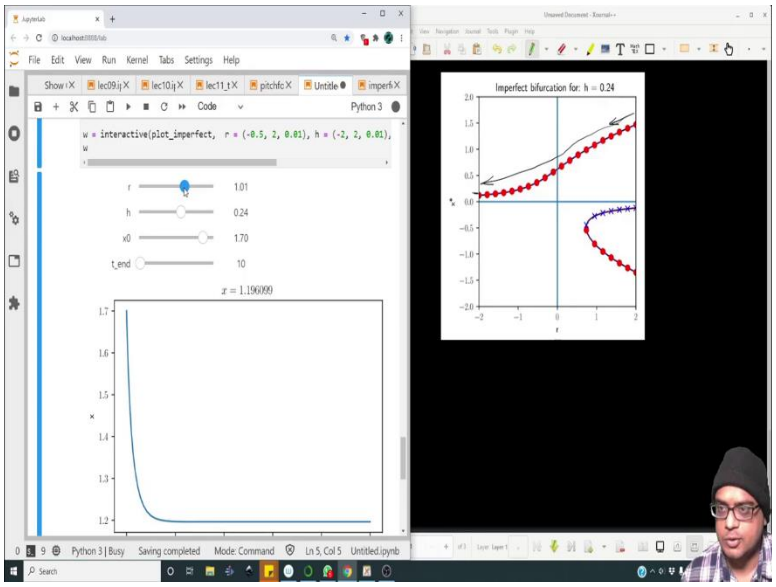
r 1.54
h 0.24
x0 1.70
t_end 10

$x = 1.312574$

Imperfect bifurcation for: h = 0.24

It is slowly decreasing and at r = - 2 we should come very close to 0.15 or something.

(Refer Slide Time: 40:41)

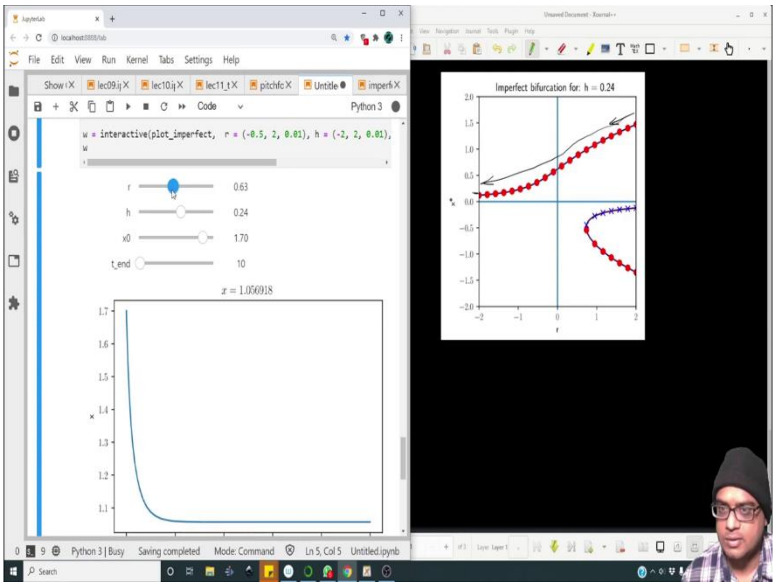

```
w = interactive(plot_imperfect, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
w
```

r 1.01
h 0.24
x0 1.70
t_end 10

$x = 1.196099$

Imperfect bifurcation for: h = 0.24

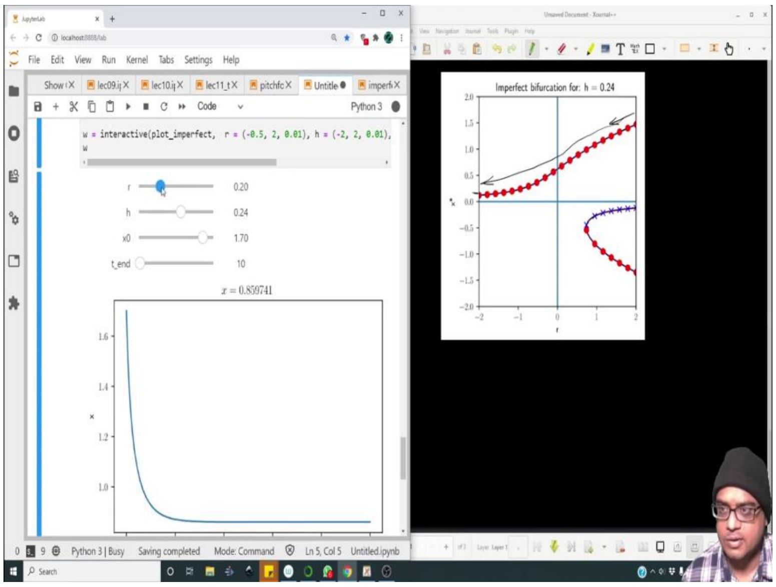
(Refer Slide Time: 40:42)



```
w = interactive(plot_imperfect, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
w
```

r 0.63
h 0.24
x0 1.70
t_end 10

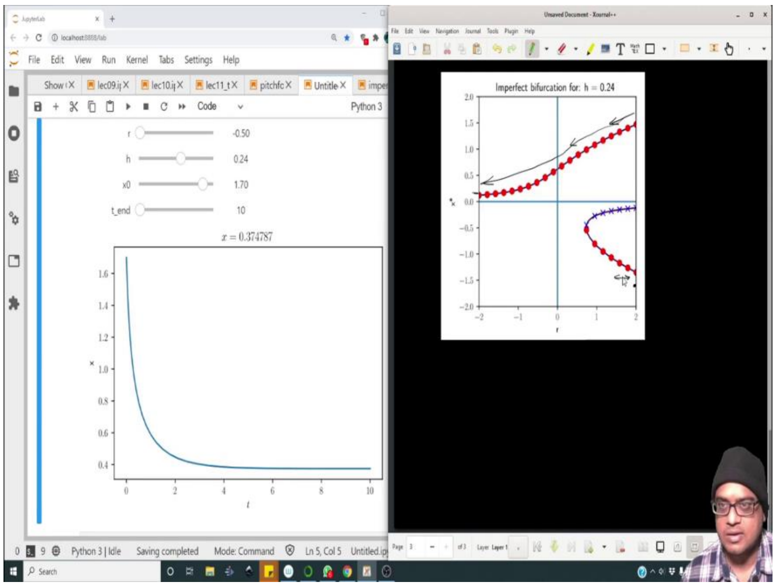
(Refer Slide Time: 40:44)



```
w = interactive(plot_imperfect, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
w
```

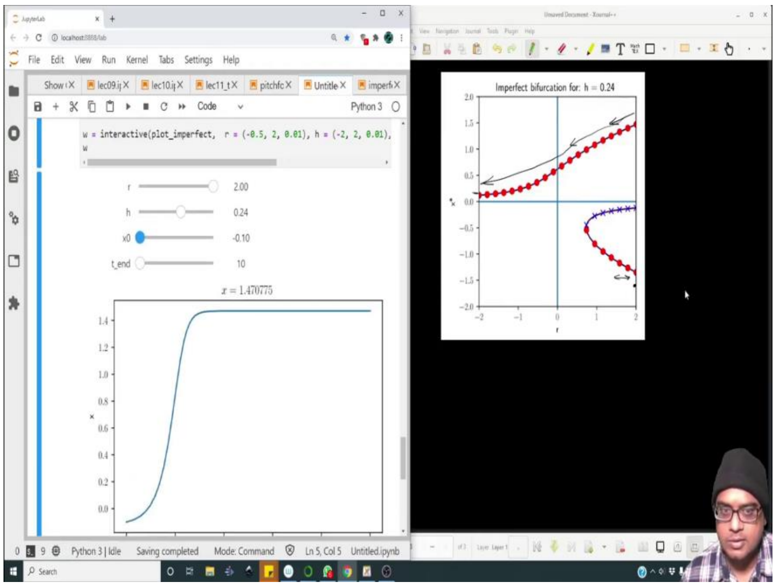
r 0.20
h 0.24
x0 1.70
t_end 10

(Refer Slide Time: 40:46)

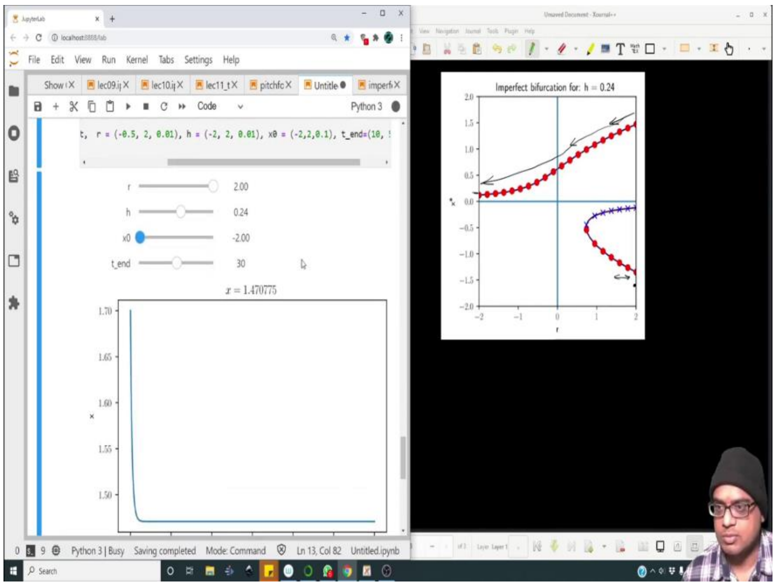

```
r      -0.50
h      0.24
x0     1.70
t_end  10
```

It is reducing, it is reducing is still positive, still positive and fine. So, we do have this monotonous reduction but now, when we start over a value somewhere near to this. So, let me change x0 so that for r = 2 we end up with this initial condition ok.

(Refer Slide Time: 41:02)


```
w = interactive(plot_imperfect,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01),
w
```
```
r      2.00
h      0.24
x0     -0.10
t_end  10
```

(Refer Slide Time: 41:12)

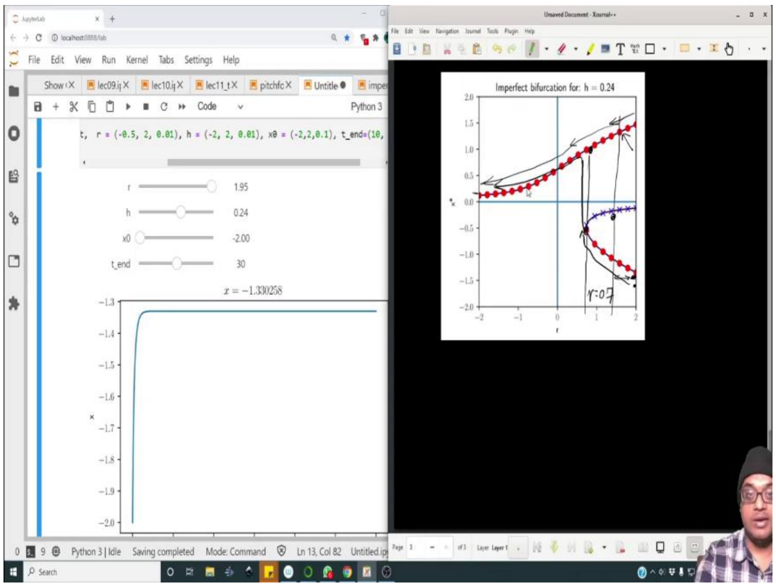


```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10,
```

r 2.00
h 0.24
x0 -2.00
t_end 30

$x = 1.470775$

(Refer Slide Time: 41:18)



```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10,
```

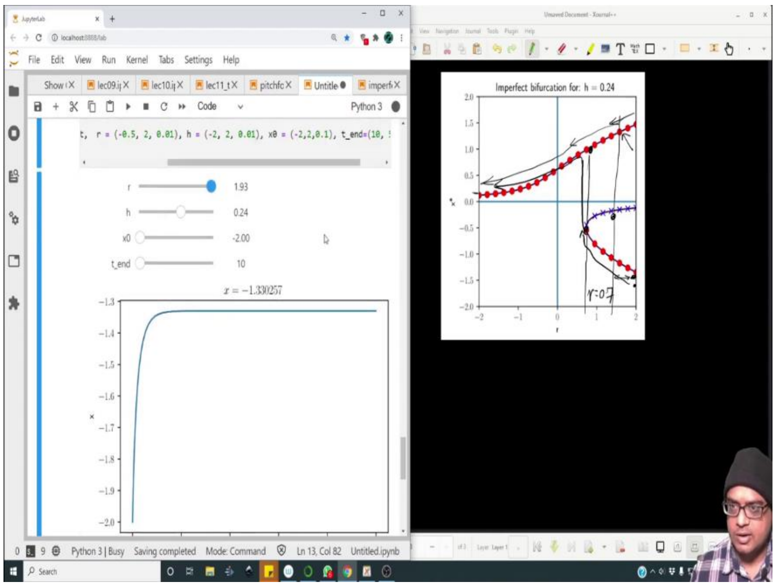
r 1.95
h 0.24
x0 -2.00
t_end 30

$x = -1.330258$

So, x0 has to be something close to -2. So, let me re execute this x0 has to be something close to -2, r has to be something like this so, yeah ok. So, x is -1.34 something. So, we are over here. Now as the value of r reduces, we will follow this stable branch up until this point this is; obviously, the unstable root, but for this parameter value this is also a stable point.

At this parameter value that is r = something between something close to 0.7. So, this is r = 0.7 this and this are like 2 stable points. After this once we reduce r further, it has to jump from this branch onto this branch and then continue along this branch.

(Refer Slide Time: 42:13)



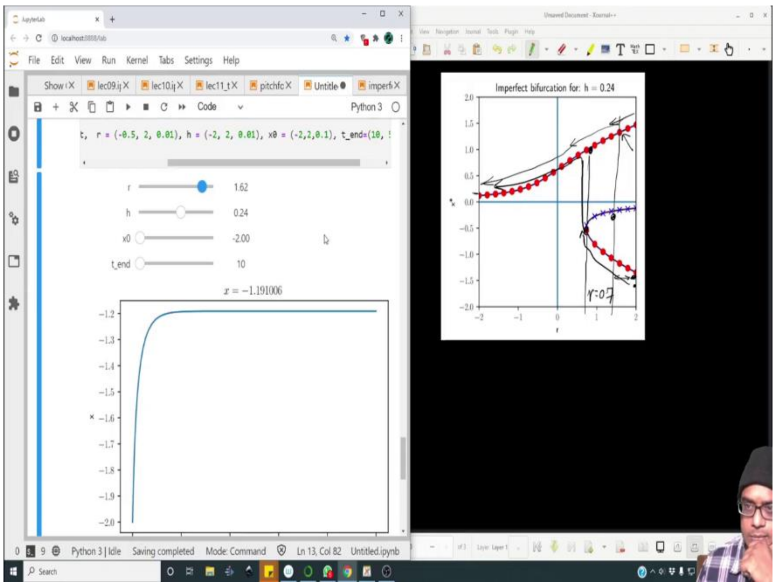
```
t,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10,
```

r 1.93
h 0.24
x0 -2.00
t_end 10

$x = -1.330257$

Let us see whether that happens or not. So, let me reduce the end ok, let me reduce r ok. So, x is reducing, x is still reducing, just a word of caution in order for this to happen quickly.

(Refer Slide Time: 42:23)

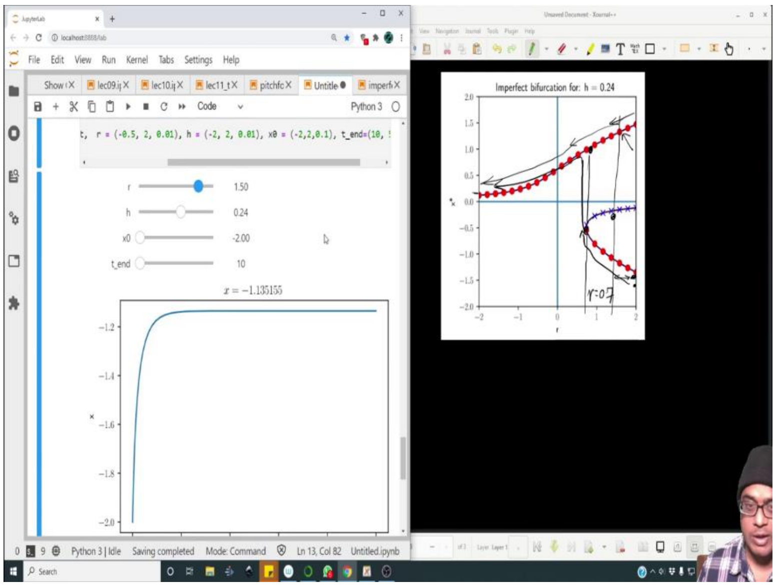


```
t,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10,
```

r 1.62
h 0.24
x0 -2.00
t_end 10

$x = -1.191006$

(Refer Slide Time: 42:28)



```
t,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10,
```

r 1.50

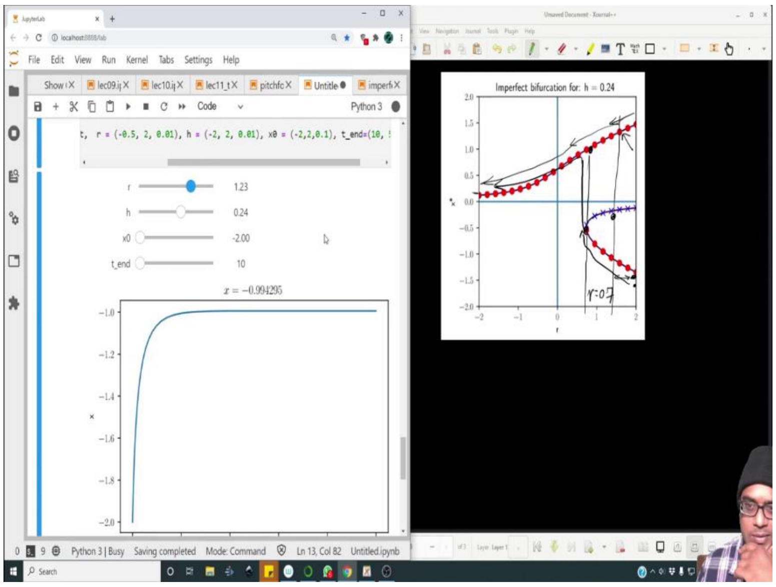
h 0.24

x0 -2.00

t_end 10

Your computer has to be quite fast because you are actually solving an ivp every time you change the value of the slider my computer is a decent computer and it runs reasonably well cannot complain ok.

(Refer Slide Time: 42:44)



```
t,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10,
```

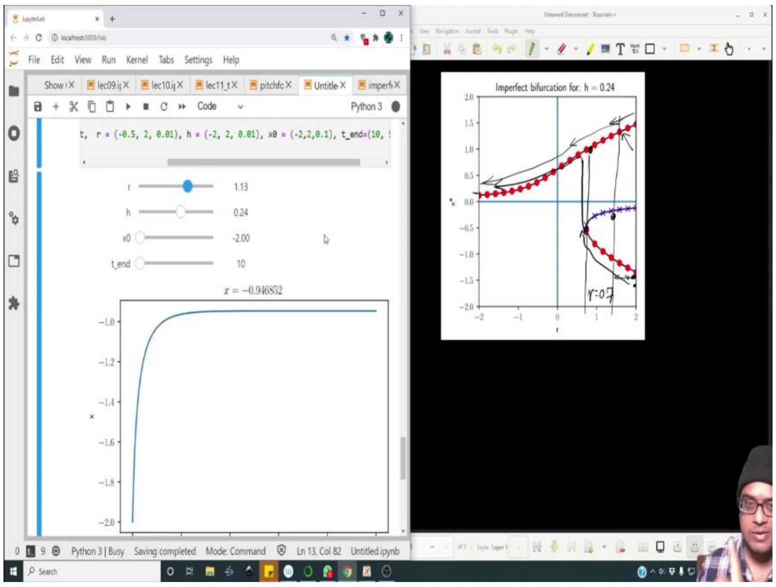
r 1.23

h 0.24

x0 -2.00

t_end 10

(Refer Slide Time: 42:49)



```
t,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, !
```

r 1.13

h 0.24

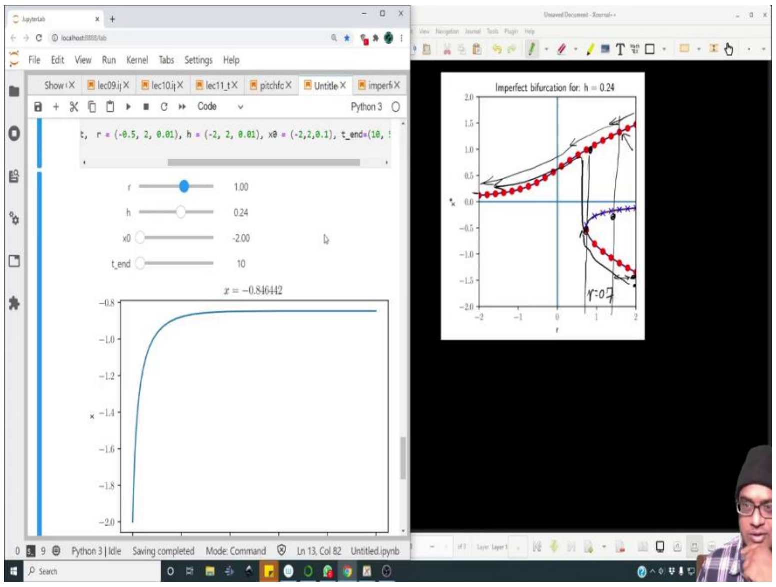
x0 -2.00

t_end 10

$x = -0.946852$

1.23 still negative there will be a certain point.

(Refer Slide Time: 42:51)



```
t,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, !
```

r 1.00

h 0.24

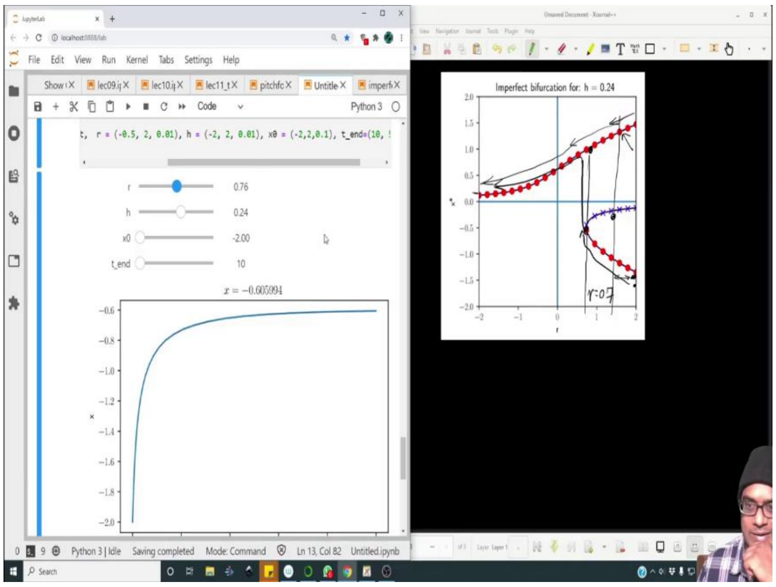
x0 -2.00

t_end 10

$x = -0.846442$

1.23 still negative. There will be a certain point when r is something close to 0.7 where there will be a sudden jump from this stable branch to that stable branch ok.

(Refer Slide Time: 43:05)

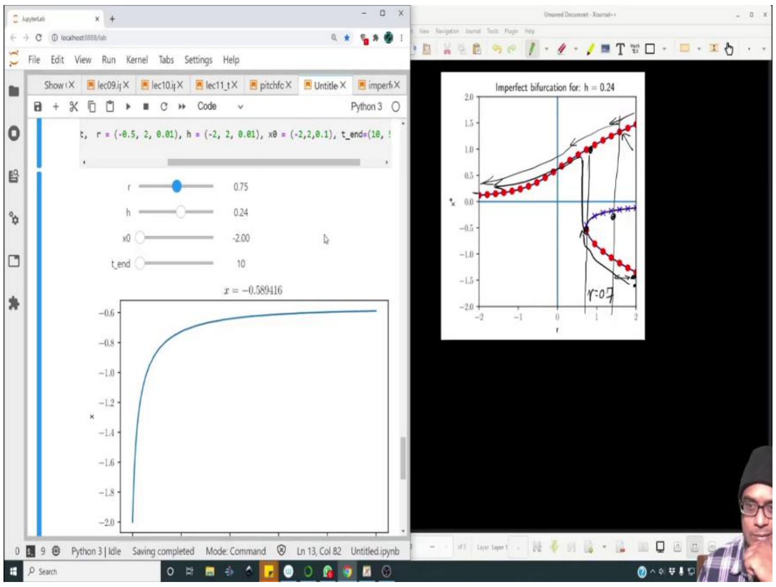


```
t,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10,
```

```
r      0.76
h      0.24
x0    -2.00
t_end  10
```

$x = -0.605994$

(Refer Slide Time: 43:08)



```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, 5
```

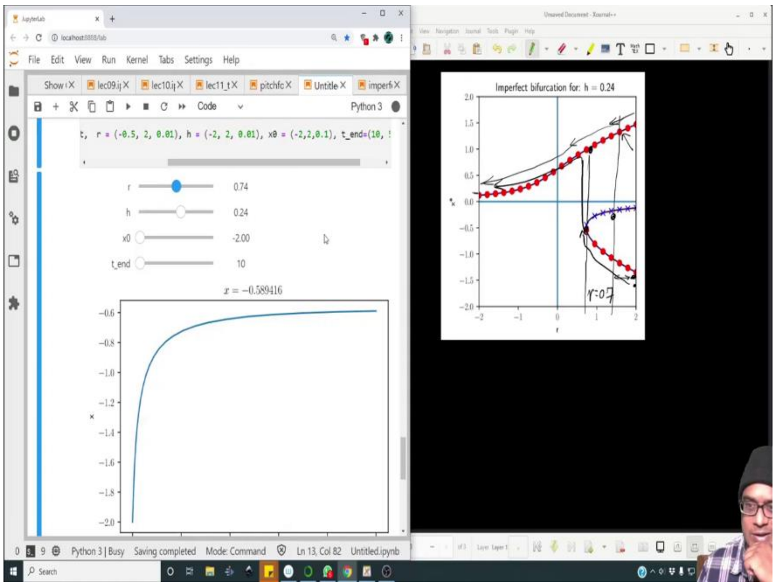
```
r      0.75
h      0.24
x0     -2.00
t_end  10
```

$x = -0.589416$

```
Python 3 | Idle   Saving completed   Mode: Command   Ln 13, Col 82   Untitled.ipynb
```

File Edit View Run Kernel Tabs Settings Help
Imperfect bifurcation for: h = 0.24
r=0.7

(Refer Slide Time: 43:10)



```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, 5
```

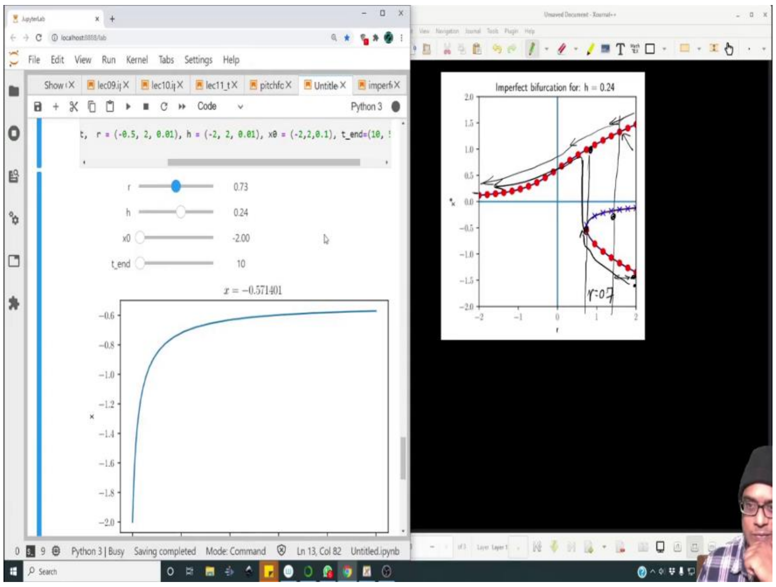
```
r      0.74
h      0.24
x0     -2.00
t_end  10
```

$x = -0.589416$

```
Python 3 | Busy   Saving completed   Mode: Command   Ln 13, Col 82   Untitled.ipynb
```

File Edit View Run Kernel Tabs Settings Help
Imperfect bifurcation for: h = 0.24
r=0.7

(Refer Slide Time: 43:11)



```
t,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, 
```

r 0.73

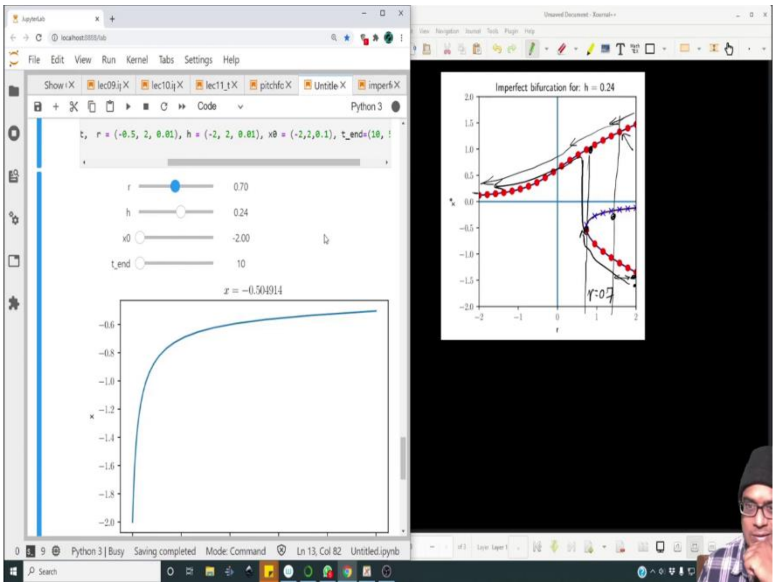
h 0.24

x0 -2.00

t_end 10

$x = -0.571401$

(Refer Slide Time: 43:15)



```
t,  r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, 
```

r 0.70

h 0.24

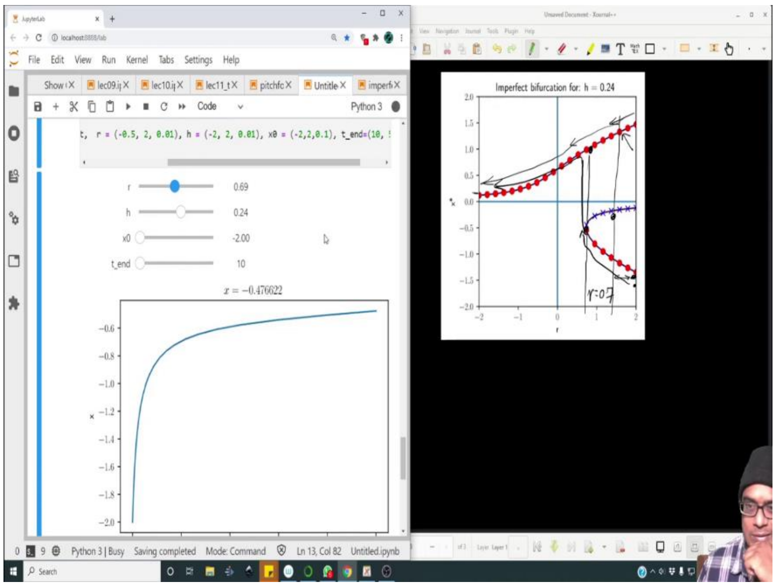
x0 -2.00

t_end 10

$x = -0.504914$

We are getting there r is 0.8, 0.76, 0.75, 0.73, 0.7, 0.69, 0.68, 0.67.

(Refer Slide Time: 43:17)

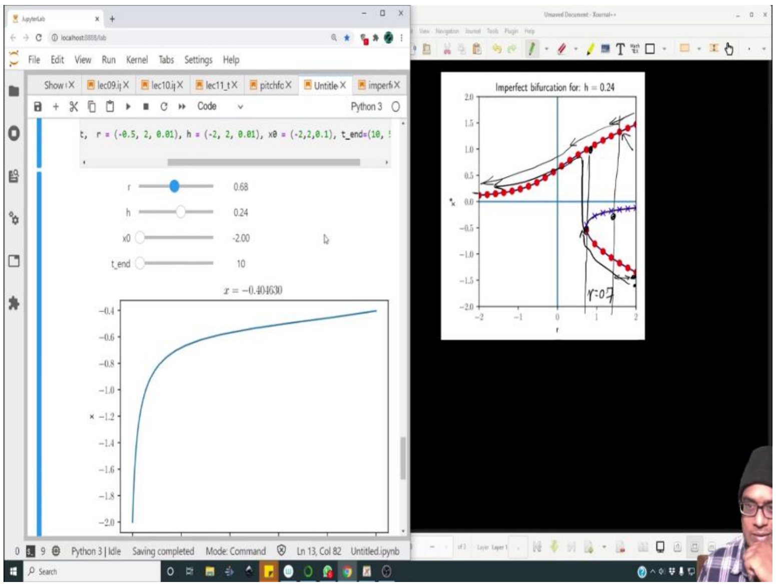


```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10,
```

r 0.69
h 0.24
x0 -2.00
t_end 10

$x = -0.476622$

(Refer Slide Time: 43:18)

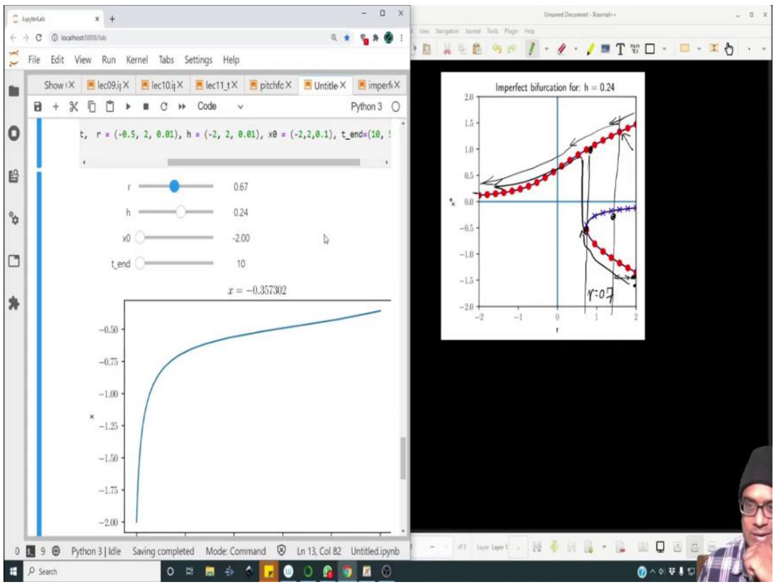


```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10,
```

r 0.68
h 0.24
x0 -2.00
t_end 10

$x = -0.404630$

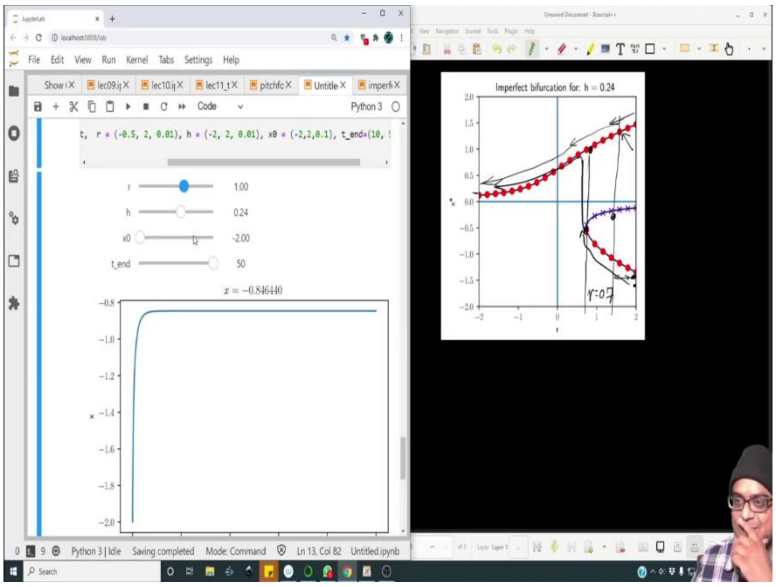
(Refer Slide Time: 43:20)



```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, !
```

```
r      0.67
h      0.24
x0     -2.00
t_end  10
```

(Refer Slide Time: 43:31)

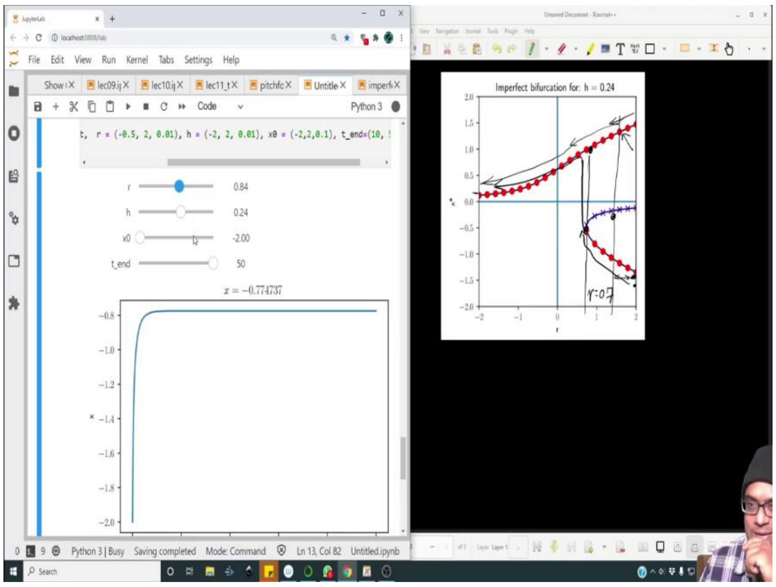


```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, !
```

```
r      1.00
h      0.24
x0     -2.00
t_end  50
```

Actually we have to increase the time there you go, my bad.

(Refer Slide Time: 43:40)

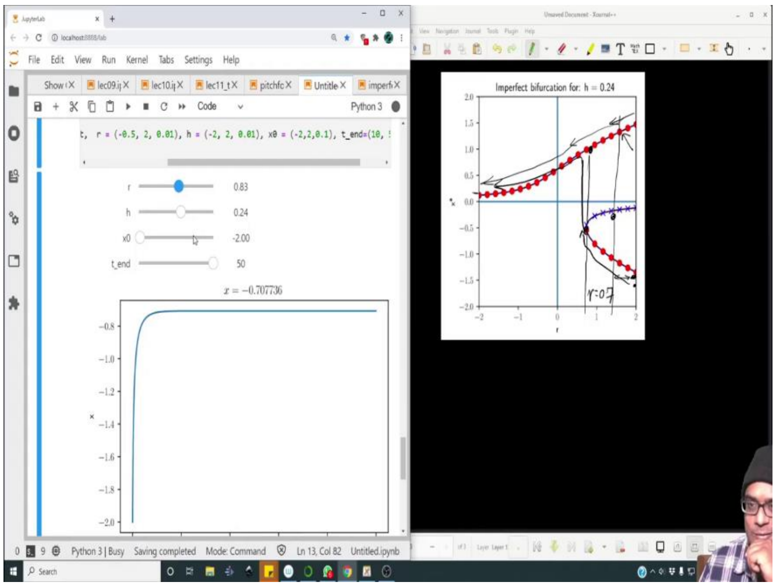


```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, 5
```

r 0.84
h 0.24
x0 -2.00
t_end 50

$x = -0.774737$

(Refer Slide Time: 43:41)

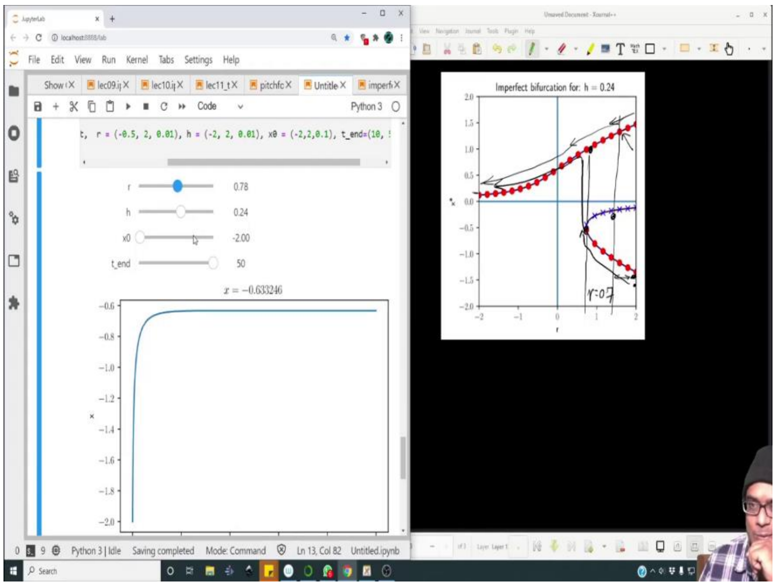


```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, 5
```

r 0.83
h 0.24
x0 -2.00
t_end 50

$x = -0.707736$

(Refer Slide Time: 43:44)

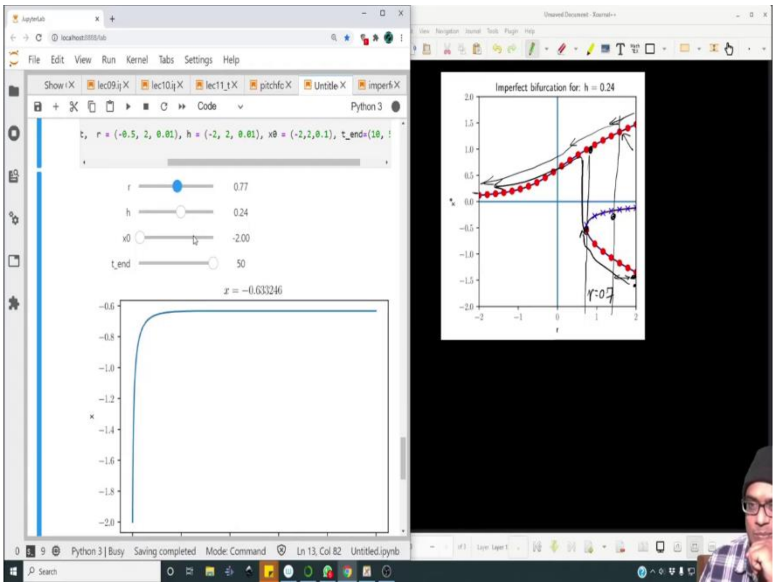


```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, 
```

r 0.78
h 0.24
x0 -2.00
t_end 50

$x = -0.633246$

(Refer Slide Time: 43:45)

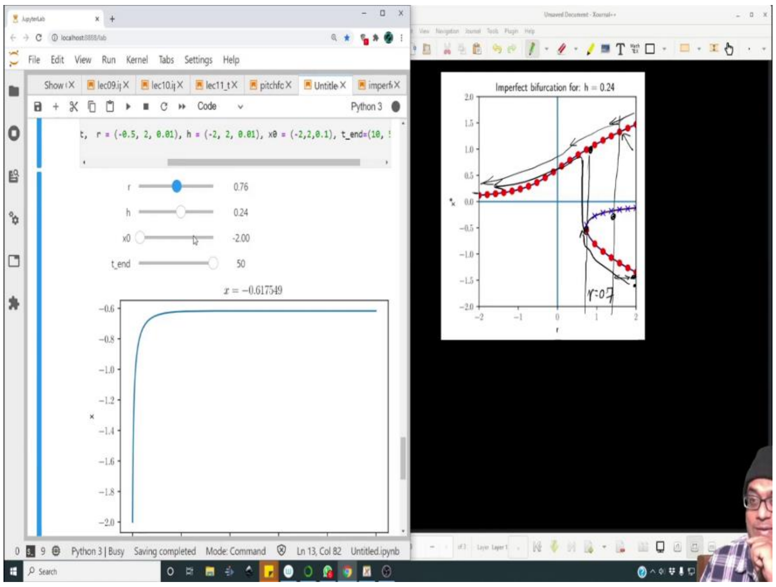


```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, 
```

r 0.77
h 0.24
x0 -2.00
t_end 50

$x = -0.633246$

(Refer Slide Time: 43:47)

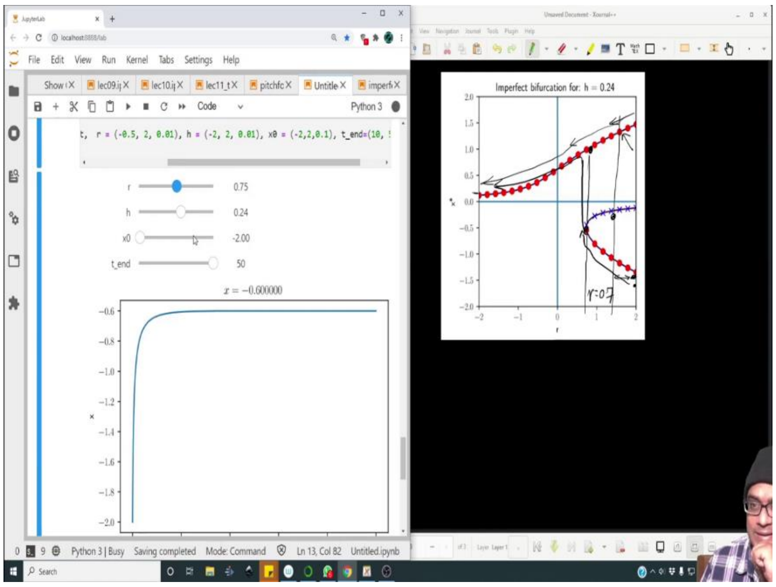


```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, 5
```

r 0.76
h 0.24
x0 -2.00
t_end 50

$x = -0.617549$

(Refer Slide Time: 43:48)

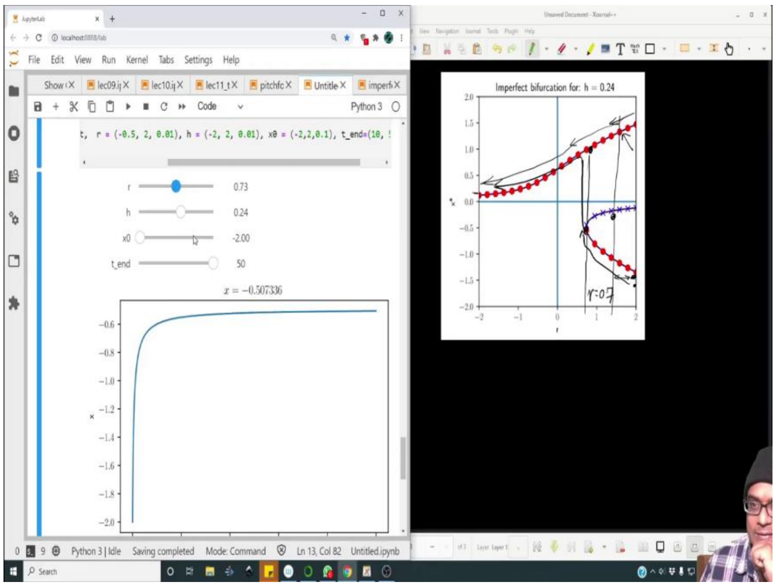


```
t, r = (-0.5, 2, 0.01), h = (-2, 2, 0.01), x0 = (-2,2,0.1), t_end=(10, 5
```

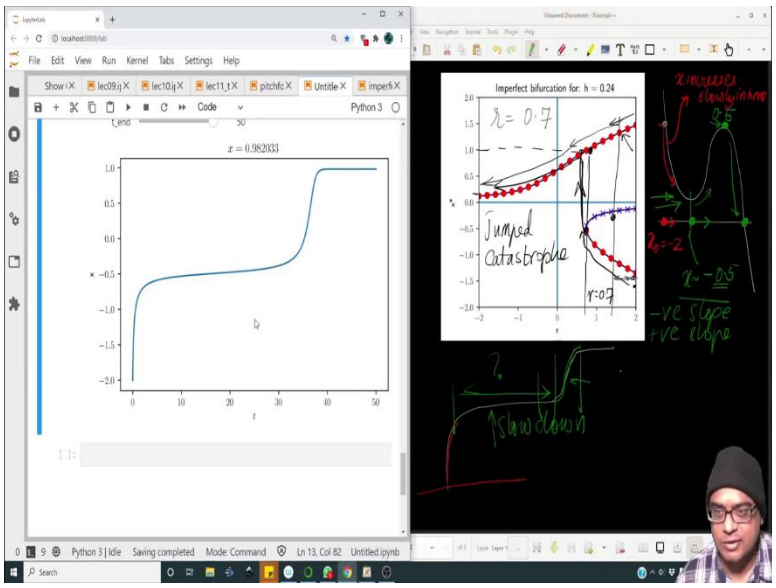
r 0.75
h 0.24
x0 -2.00
t_end 50

$x = -0.600000$

(Refer Slide Time: 43:51)



(Refer Slide Time: 43:54)

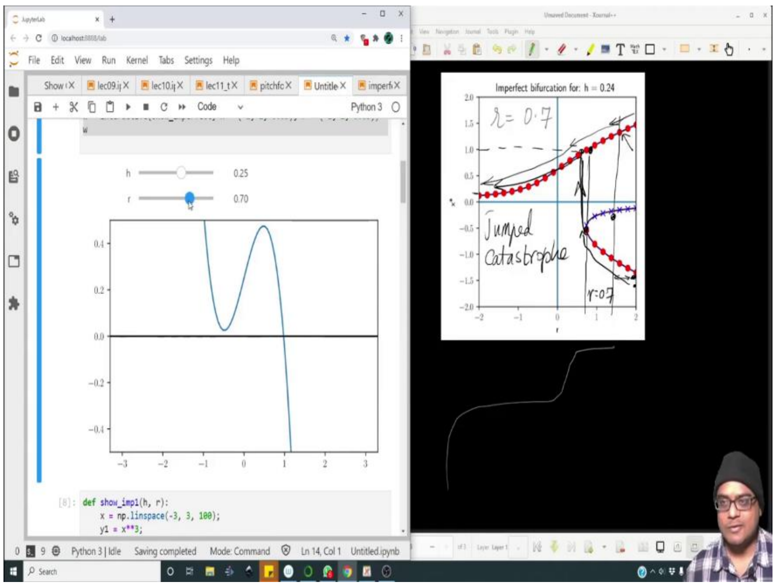


So, is - 0.84 r is reducing from 0.9, 0.84, 0.83, 0.78, 0.77, 0.76, 0.75, 0.73 is still in the negative branch boom from r = 0.72 we have the final asymptotic value of 0.98. So, we have made a jump from here to here and this value is actually pretty close to 1 and our mathematics and the intuition that we have obtained from the imperfect bifurcation diagram everything checks out.

So, this particular thing is called as. So, we have essentially what we have jumped between the 2 stable branches and we have a catastrophe ok, if catastrophically jumped from 1 stable branch to another. You can vary the value of h find out this bifurcation diagram then plot the you can solve the ivp and plot the asymptotic values and you can check for yourself whether everything works out.

Now a small task if you will just try to find out why exactly does the evolution of the ivp look something like this. Look we are at r = 0.72 and h = 0.24 ok.

(Refer Slide Time: 45:18)

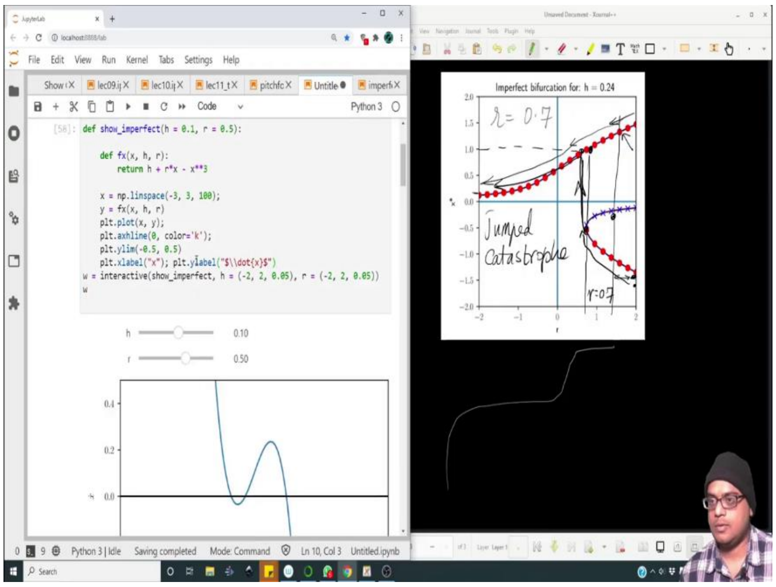


```
[8]: def show_imp1(h, r):
         x = np.linspace(-3, 3, 100);
         y1 = x**3;
```

```
0  9  Python 3 | Idle   Saving completed   Mode: Command   Ln 14, Col 1   Untitled.ipynb
```

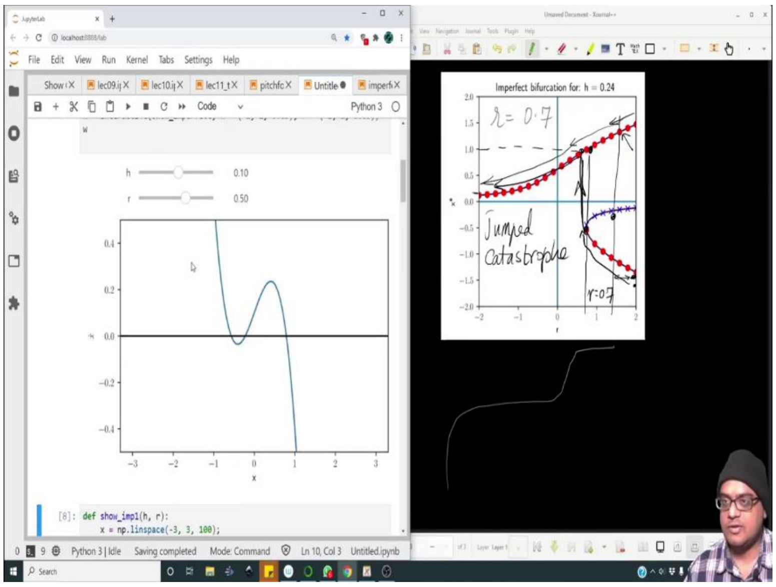
So, look r = 0.7 let us say we plot everything at 0.7. Can we explain why the plot looks the way it does? Let me make h = 0.24 and r = 0.6 ok. So, we have started so, this is the curve of $\dot{x}$ versus x ok.

(Refer Slide Time: 45:56)



```
[58]: def show_imperfect(h = 0.1, r = 0.5):

          def fx(x, h, r):
              return h + r*x - x**3

          x = np.linspace(-3, 3, 100);
          y = fx(x, h, r)
          plt.plot(x, y);
          plt.axhline(0, color='k');
          plt.ylim(-0.5, 0.5)
          plt.xlabel("x"); plt.ylabel("$\\dot{x}$")
      w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
      w
```

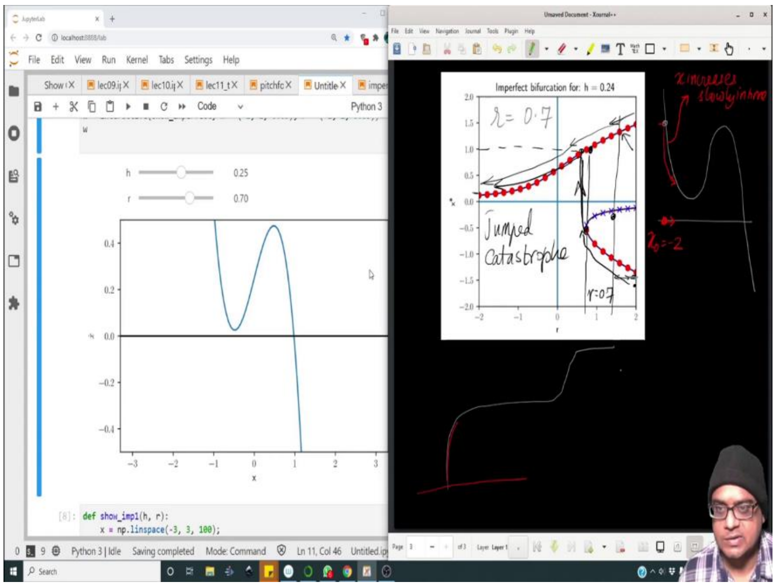
(Refer Slide Time: 46:11)



```
      w
```

```
[8]: def show_impl(h, r):
         x = np.linspace(-3, 3, 100);
```

(Refer Slide Time: 46:15)


h 0.25
r 0.70
Imperfect bifurcation for: h = 0.24
r = 0.7
Jumped Catastrophe
r=0.7
x increases slowly in time
$x_0 = -2$

```
def show_imp1(h, r):
    x = np.linspace(-3, 3, 100);
```

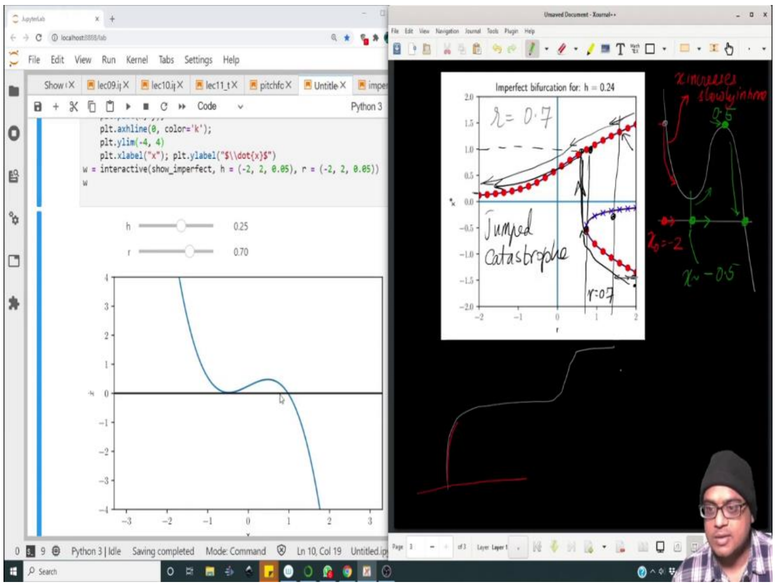
So, this is $\dot{x}$versus x h is 0.24 and r is something like 0.7 well give or take something like this. Now, look we have started the initial condition is somewhere on this curve; suppose the initial condition is somewhere over here. In fact, let us see what the actual initial condition was.

The actual initial condition was - 2 ok. So, the initial condition is - 2, let us do that. So, it starts from - 2. The initial condition is over here. Let me draw this curve parallely. So, the initial condition somewhere over here; I mean not over there. The initial condition is somewhere over here this is x0= -2.

So, the value of $\dot{x}$ is positive. So, the point starts to move towards this, but it moves very slowly initially. So, x so, from this to this x increases slowly in time and that explains that well is it really something like that let us check. It is slowly increasing in time.

So, there is a very fast behavior like this, but that fast behavior is quickly superseded by a very slow behavior. Actually that fast behavior also we should be able to explain.

(Refer Slide Time: 48:05)



```
    plt.axhline(0, color='k');
    plt.ylim(-4, 4)
    plt.xlabel("x"); plt.ylabel("$\\dot{x}$")
w = interactive(show_imperfect, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
w
```

h 0.25

r 0.70

Imperfect bifurcation for: h = 0.24
r = 0.7
Jumped Catastrophe
r:0.7
x increases slowly in time
0.5
x0 = -2
x ~ -0.5

Let me go back to the plot and in forget about the y limit and let us rather take a y limit from - 4 to 4 perhaps which is 0.24 and r is 0.7 ok. So, we are starting somewhere over here. So, there is a very fast motion and then slowing down. The slowing down continues up until this point.

So, this point corresponds to x = minus. So, this is slowing down up until this point and this corresponds to something between somewhere near - 0.5. So, at 0.5 there is a minimum velocity after that minimum velocity the velocity again increases. So, once the particle reaches this, there will be a fast motion towards the right that x will increase and eventually again there will be a slowing down and then a fast motion all the way to the fixed point ok.

So, this point and then the maxima is somewhere between the maxima is somewhere near 0.5 ok. So, let us see; let us see the max and the equilibrium is near 1 let us see that ok. So, x = 0 point - 0.2 we started at - 0.5 we have a change in behavior.

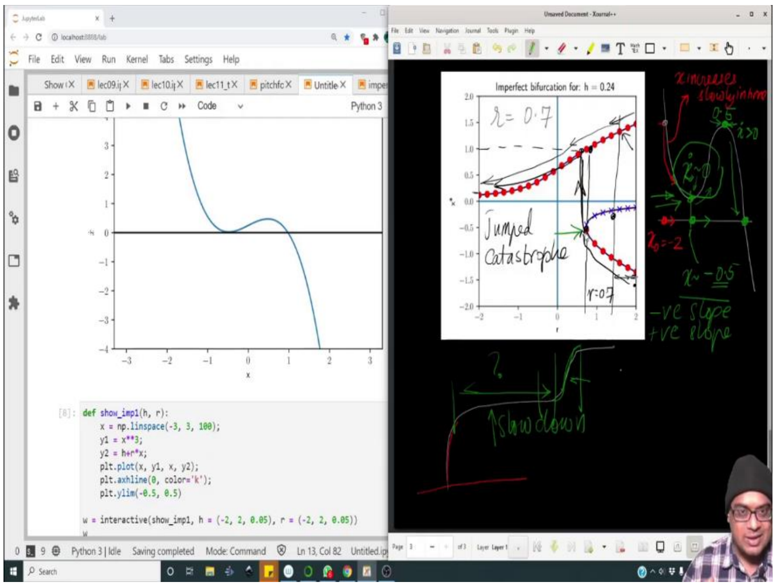
So, at - 0.5 we have very slow increase to a fast increase so, a transition from slow to fast. So, this is that inflection point in the x versus t curve. After that we see another change in curve from this. This fast increase to a slow increase again at x = 0.5.

So, in the x-t curve wherever you see inflection points be aware that in the $\dot{x}$ versus x curve you will encounter this kind of local minima or maximum. And it means that you are changing behavior from a positive slope to a negative slope ok.

So, for example, at - 0.5 we are changing from negative slope to a positive slope and this is that area of negative slope. So, you are increasing time, but x is changing very slowly and over here increasing time and now x begins to increase rapidly. So, that is the change over in the behavior of x versus t and over here also this is increasing in time this is decreasing in time. So, there is a change over here. And the change over corresponds to x = 0.5 something around, but it takes a long time for it to reach because of the slowing down of everything.

In fact, this particular slowdown is quite large. Can you tell why this slowing down is very large compared to this crossover? It is slowing down and there is a crossover, but over here there is also a crossover, but it is quite rapid. This is quite rapid than this. So, the question is why. And the answer is quite straightforward if you look at this curve the $\dot{x}$ over here is really close to 0.

(Refer Slide Time: 51:37)



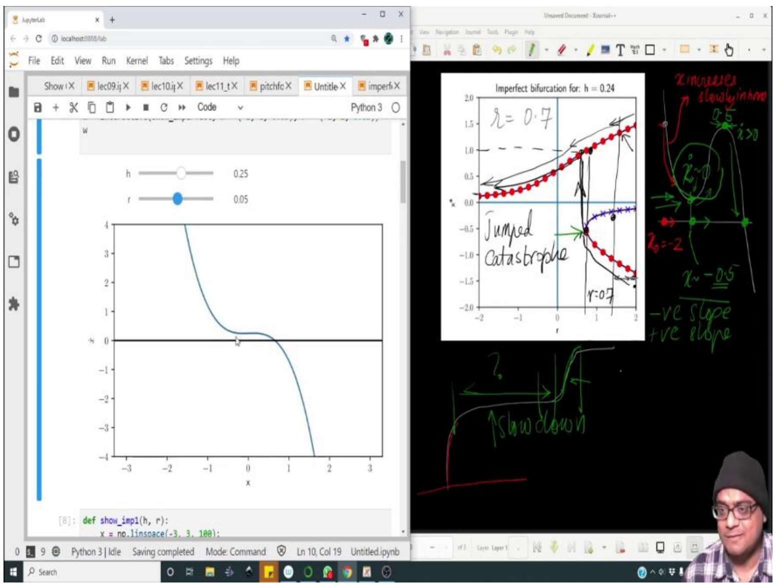
```
[8]: def show_imp1(h, r):
         x = np.linspace(-3, 3, 100);
         y1 = x**3;
         y2 = h+r*x;
         plt.plot(x, y1, x, y2);
         plt.axhline(0, color='k');
         plt.ylim(-0.5, 0.5)

     w = interactive(show_imp1, h = (-2, 2, 0.05), r = (-2, 2, 0.05))
     w
```

So, the $\dot{x}$ over at this point is going very close to 0 while the $\dot{x}$ over here is something which has a positive value ok. So, because it comes very close to 0 the velocity of the point on the number line it becomes very 0; because we are studying the motion of this red particle on the line. And when the velocity itself comes very close to 0; there is a very slow motion towards the positive side.

So, despite the velocity being positive it is a very slow it is a very it is magnitude is quite low and therefore, it encounters a very large slowing down and then the changeover happens ok. So, this slow dynamics is being captured in the vicinity where you would get 2 points.

So, this catastrophe happens and that catastrophe is sort of preceded by that is dramatic slowing down. Because, once we increase the parameter further we will get rid of that slowing down then it will not happen anymore ok.

(Refer Slide Time: 52:53)



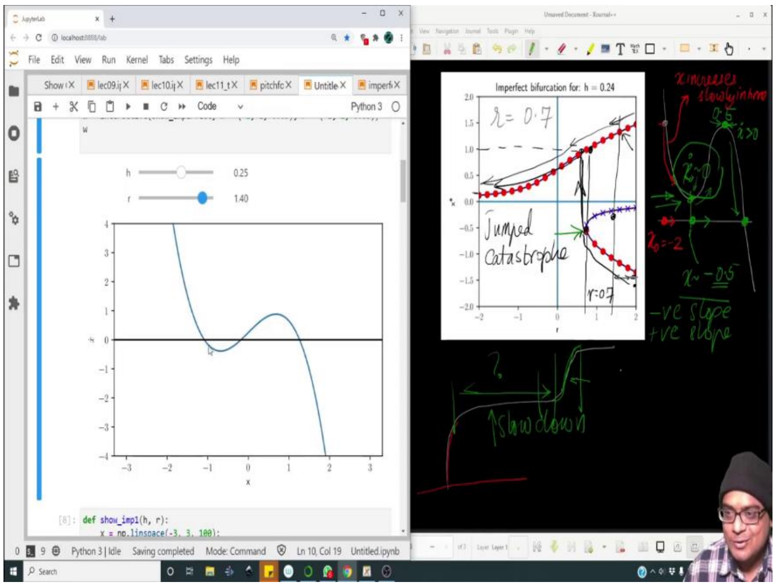
File Edit View Run Kernel Tabs Settings Help
h 0.25
r 0.05
x
Imperfect bifurcation for: h = 0.24
r = 0.7
Jumped Catastrophe
r = 0.7
r
x increases slowly in time
x > 0
x ~ 0
x₀ = -2
x ~ -0.5
-ve slope
+ve slope
slow down

```
[8]: def show_imp1(h, r):
         x = np.linspace(-3, 3, 100);
```

```
Python 3 | Idle   Saving completed   Mode: Command   Ln 10, Col 19   Untitled.ipynb
```

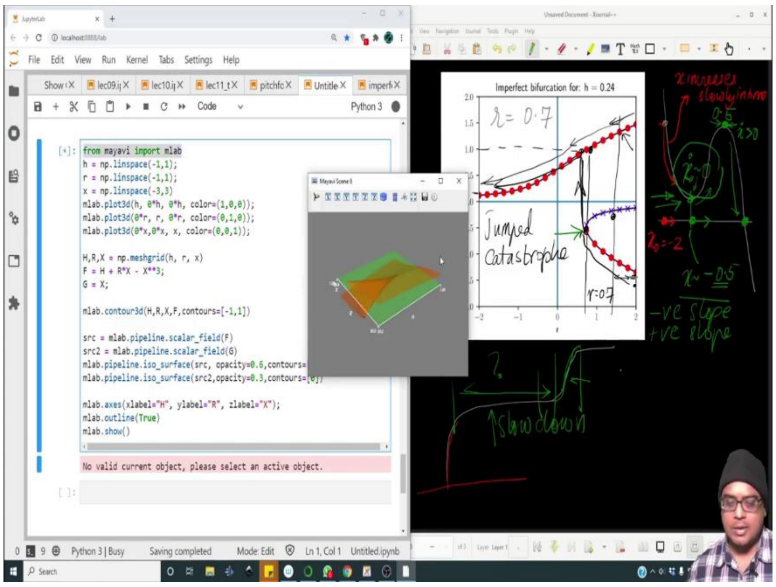
So, when r reduces when r reduces that slowing down is slowly lifting up ok. So, you have no longer have that dramatic slowing down. When r is very close, so that this branch comes very close to 0; you will have a very dramatics length now.

(Refer Slide Time: 53:12)



Now when r is increasing that branch is actually never reached because this root this particular point is splitting into 2 roots; this is 1 roots and this is 1 root. And it will continuously reach only this root it will not reach the other root ok. So, whole lot of thing happen in the bifurcation space, I am not going to discuss about the three dimensional view of this equation. I am however, going to put a small snippet of how to visualize this in 3D ok.

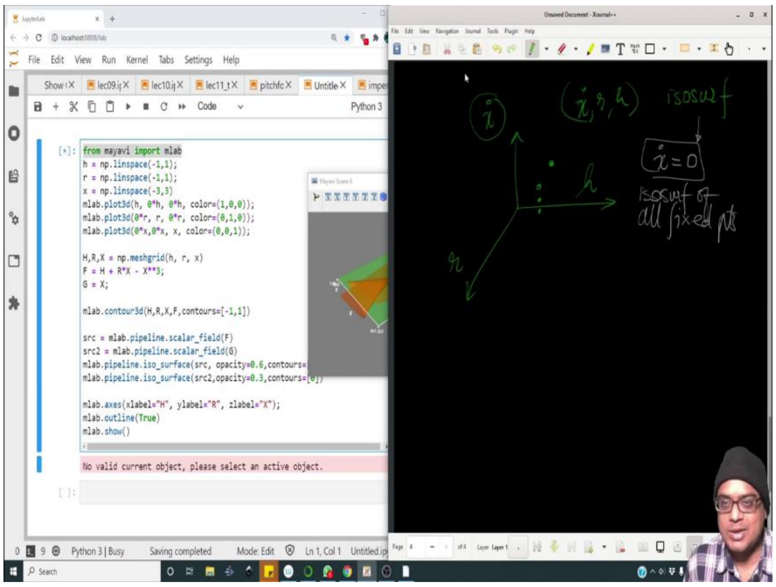
(Refer Slide Time: 53:50)



```
[*]: from mayavi import mlab
     h = np.linspace(-1,1);
     r = np.linspace(-1,1);
     x = np.linspace(-3,3)
     mlab.plot3d(h, 0*h, 0*h, color=(1,0,0));
     mlab.plot3d(0*r, r, 0*r, color=(0,1,0));
     mlab.plot3d(0*x,0*x, x, color=(0,0,1));

     H,R,X = np.meshgrid(h, r, x)
     F = H + R*X - X**3;
     G = X;

     mlab.contour3d(H,R,X,F,contours=[-1,1])

     src = mlab.pipeline.scalar_field(F)
     src2 = mlab.pipeline.scalar_field(G)
     mlab.pipeline.iso_surface(src, opacity=0.6,contours=
     mlab.pipeline.iso_surface(src2,opacity=0.3,contours=[0])

     mlab.axes(xlabel="H", ylabel="R", zlabel="X");
     mlab.outline(True)
     mlab.show()

No valid current object, please select an active object.
```

So, just for completeness I have put in the small snippet of visualizing the 3 dimensional plot in mayavi. So, basically let me just tell you what I am trying to plot over here.

(Refer Slide Time: 54:05)



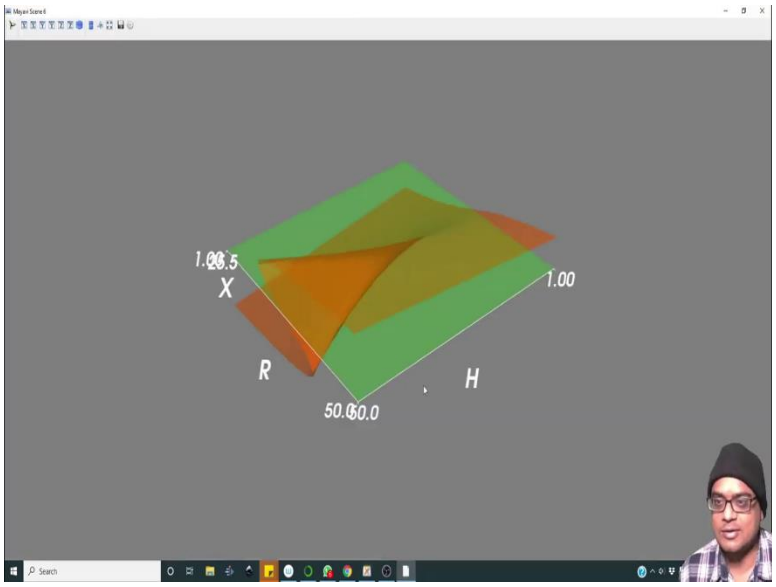
```python
from mayavi import mlab
h = np.linspace(-1,1);
r = np.linspace(-1,1);
x = np.linspace(-3,3)
mlab.plot3d(h, 0*h, 0*h, color=(1,0,0));
mlab.plot3d(0*r, r, 0*r, color=(0,1,0));
mlab.plot3d(0*x,0*x, x, color=(0,0,1));

H,R,X = np.meshgrid(h, r, x)
F = H + R*X - X**3;
G = X;

mlab.contour3d(H,R,X,F,contours=[-1,1])

src = mlab.pipeline.scalar_field(F)
src2 = mlab.pipeline.scalar_field(G)
mlab.pipeline.iso_surface(src, opacity=0.6,contours=
mlab.pipeline.iso_surface(src2,opacity=0.3,contours=[0])

mlab.axes(xlabel="H", ylabel="R", zlabel="X");
mlab.outline(True)
mlab.show()
```

```
No valid current object, please select an active object.
```

So, if we have r over here h over here and the fixed points. So, the fixed points should give us some curve in this plane and there will be zones in which we have only 1 fixed point and then there will be three fixed points in a certain zone and there will be a surface in this $\dot{x}$ r h volume which will correspond to an iso surface which will correspond to a surface in which \dot{x} is 0 all over. And that surface will be the surface of all the fixed points ok.

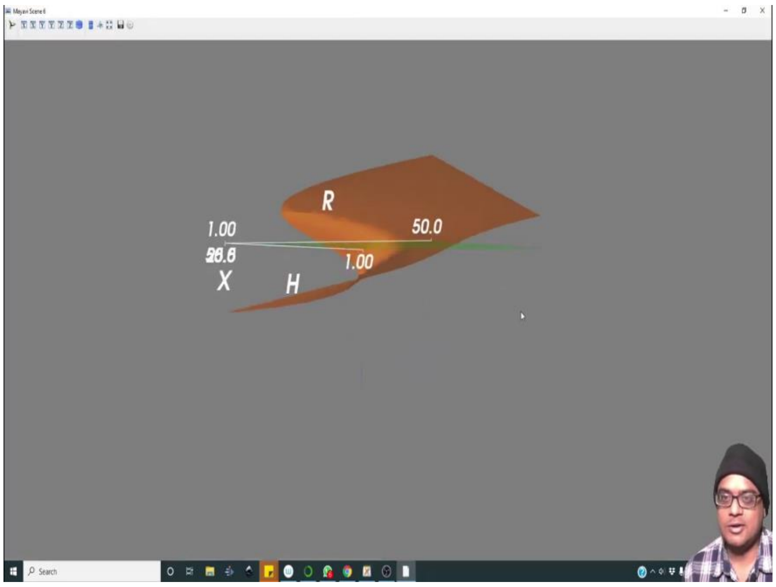
That is the meaning of that. So, this will be the iso surface of all fixed points and those three fixed points should also be visible.

(Refer Slide Time: 55:04)

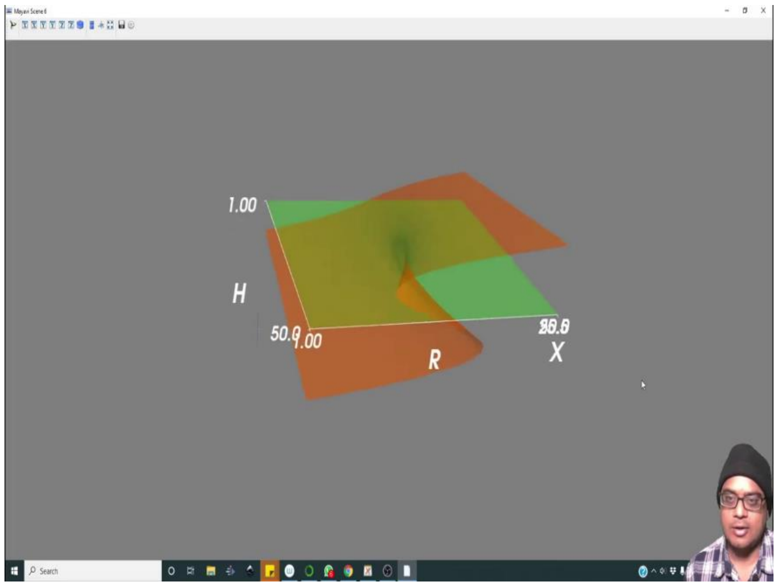


So, let me enlarge this further ok.

(Refer Slide Time: 55:06)

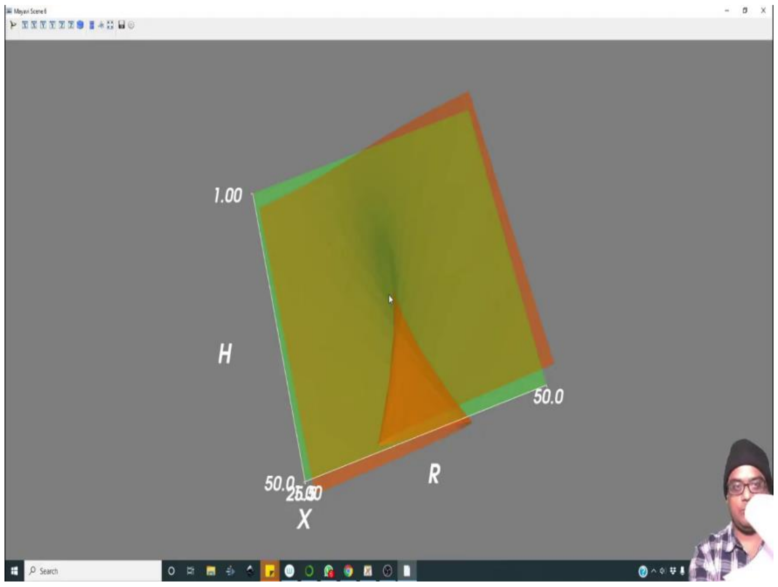


(Refer Slide Time: 55:13)



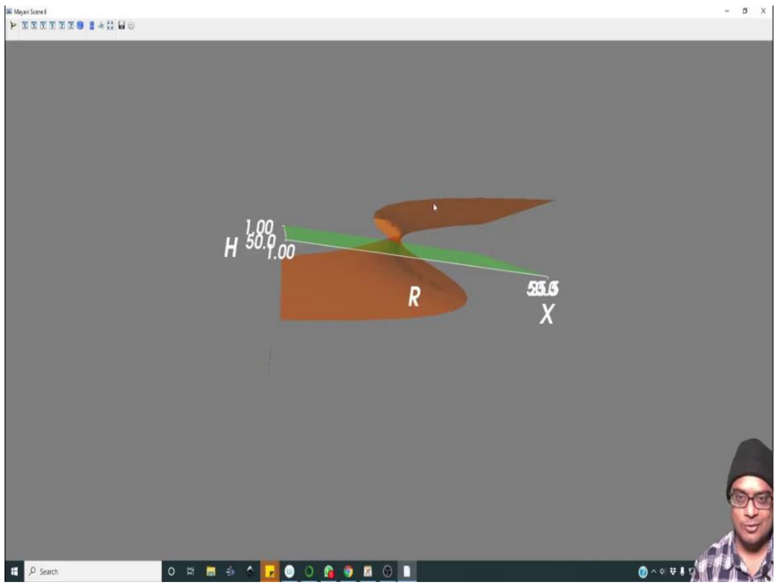
So, I have drawn the R H plane in green and the plane the iso surface of the fixed point is shown in orange ok.

(Refer Slide Time: 55:22)



So, look at how what things are, here is the cusp here is the cusp. If you look closely at this triangle not triangle, but this thing is the cusp it tells you the folding of a sheet of paper ok. So, well I cannot really show it with this A4 sheet, but you get the point ok. It is a S shaped curve this S shaped curve and where it starts becoming flat again. So, it goes from this S shaped curve to being flat again and that entails this cusp over here ok. So, that cusp is visible in the HR plane. So, this is the HR plane and here there are 3 routes here there is only 1 root.

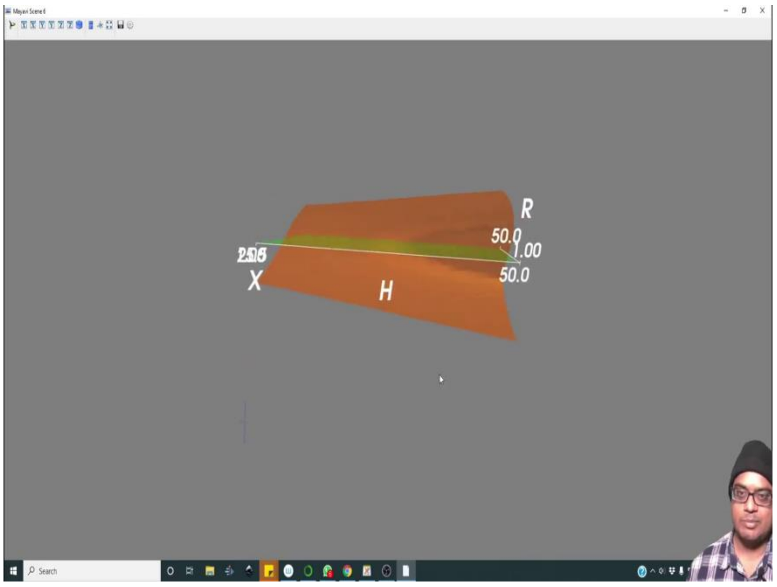
(Refer Slide Time: 56:22)



There are 3 roots; 1 root, 1 root, 1 root ok; there are 3 surfaces. The top surface, the middle surface and the bottom surface. So, these things these overlaps .I have made it transparent so that if you stack multiple transparent layers you have something which is more opaque.

So, it helps us in effectively visualizing that these there are more layers over here and more layers corresponding correspond to larger number of roots. There is only 1 root over here there is only 1 root over here. So, that cusp is something which we had seen in the HR plane ok. So, in the 3D zone, the HR plane has that cusp and we do have that cusp grid.

(Refer Slide Time: 57:16)

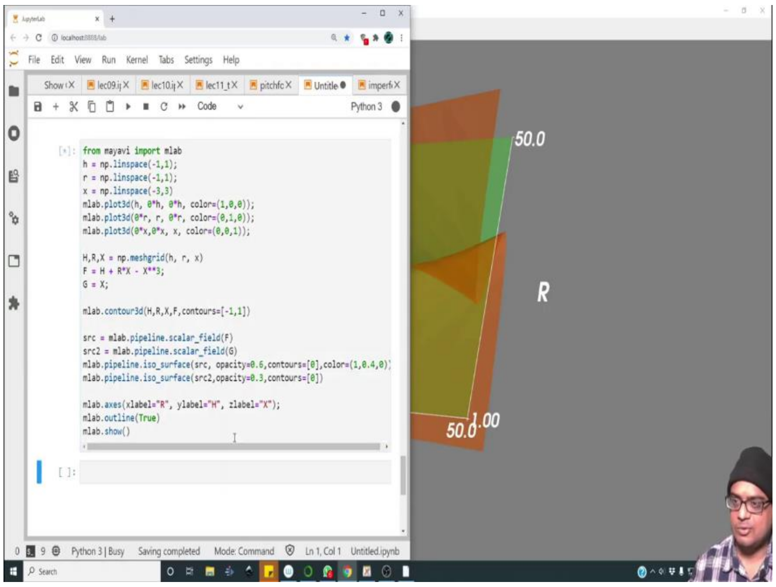


In the XR plane. So, this is the H axis this is the H axis. So, if we look at this the fixed points as a function of R for different values of H ok. So, different transects will give you different values of H and then you can vary R and see what happen. And the thing is this inside fold that is an unstable branch ok.

These things this is stable this is stable, but this inside fold is unstable. And we are now looking at the HR curve let me show the HR curve. We are essentially looking at this across different transects. So, H = 0.24 it will be somewhere over here that transact will be somewhere over here ok.

It is just a matter of how you can visualize whatever is going on. In fact, the H and R are swapped, let me quickly check this H and R are swapped if I am correct X level is H it is correct anyway. So, yeah I think the H and R axis are swapped and that swapping happens because of this mesh grid structure.

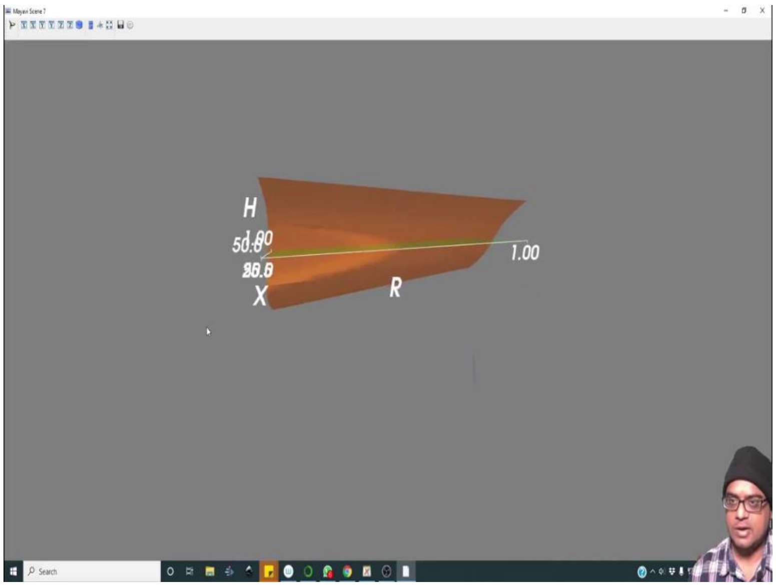
(Refer Slide Time: 58:55)



```
from mayavi import mlab
h = np.linspace(-1,1);
r = np.linspace(-1,1);
x = np.linspace(-3,3)
mlab.plot3d(h, 0*h, 0*h, color=(1,0,0));
mlab.plot3d(0*r, r, 0*r, color=(0,1,0));
mlab.plot3d(0*x,0*x, x, color=(0,0,1));

H,R,X = np.meshgrid(h, r, x)
F = H + R*X - X**3;
G = X;

mlab.contour3d(H,R,X,F,contours=[-1,1])

src = mlab.pipeline.scalar_field(F)
src2 = mlab.pipeline.scalar_field(G)
mlab.pipeline.iso_surface(src, opacity=0.6,contours=[0],color=(1,0.4,0))
mlab.pipeline.iso_surface(src2,opacity=0.3,contours=[0])

mlab.axes(xlabel="R", ylabel="H", zlabel="X");
mlab.outline(True)
mlab.show()
```

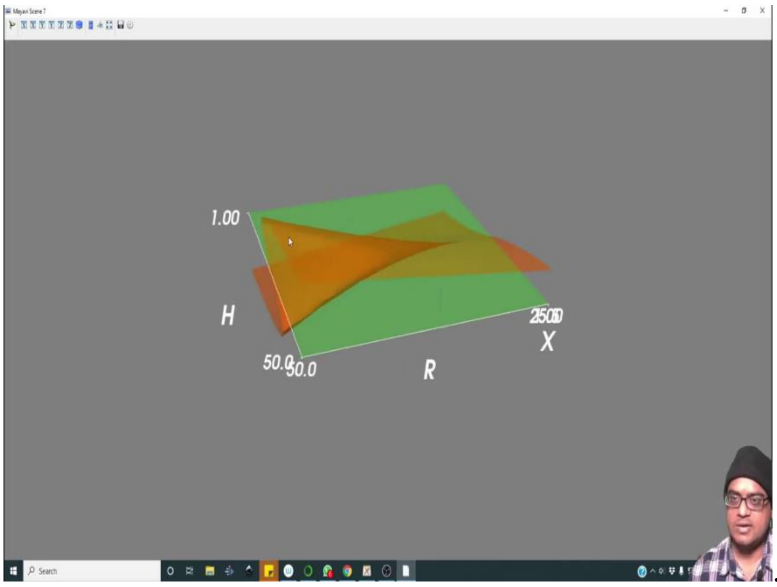
It does not give you something which is like ji stuff that ji transpose business happens. So, let me close this, or let me redo it yeah ok.

(Refer Slide Time: 59:07)



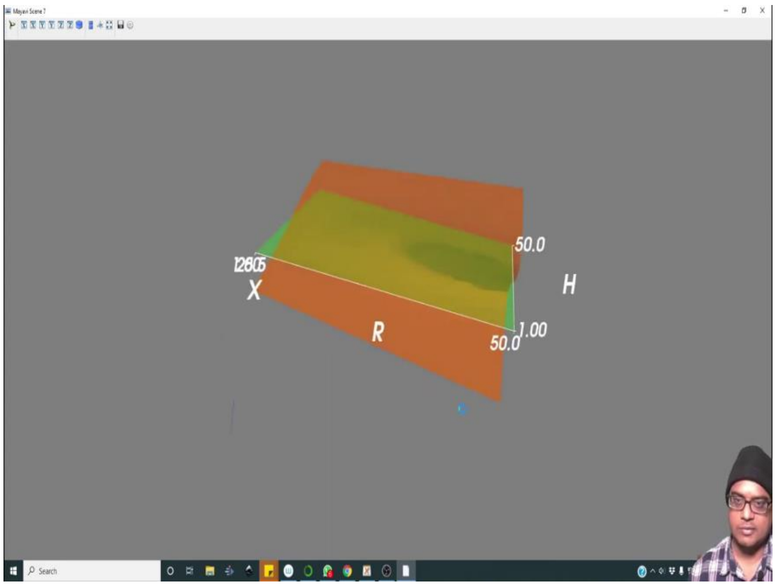
So, if we look at the X versus R stuff and we have to look at it at different transects then you will get that kind of a behavior where jumping across different kinds of curves ok.

(Refer Slide Time: 59:18)

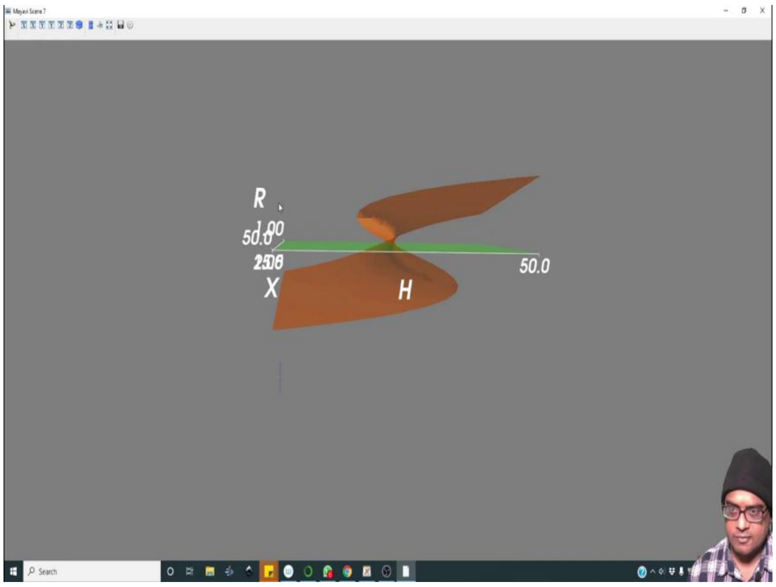


So, you have to take different transects in this direction to find the iso H lines and similarly if you have the time you can do the plot fixed point versus H for different values of R for different transacts of R and you will see this kind of S shaped curve.

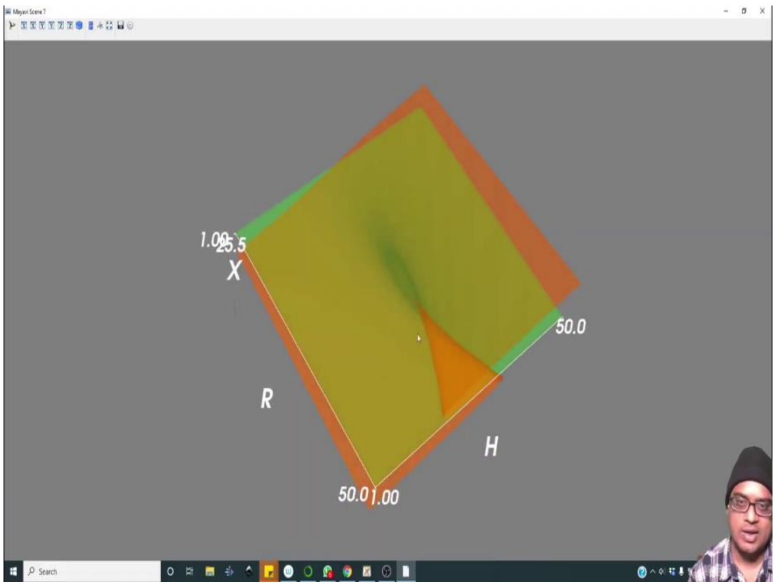
(Refer Slide Time: 59:26)



(Refer Slide Time: 59:38)



(Refer Slide Time: 59:45)



And that s shaped curve as you keep changing R you will end up with only 1 root ok. So, this 3D stuff is visualized not with the help of matplotlib, you can do certain things with matplotlib, but typically in scientific computing you use data structure called as VTK. And VTK is a data structure which is quite robust and you can really visualize large data sets and Mayavi is the name of a module which is available in Python.

It has been made by Enthought and it is a very robust library to deal with 3 dimensional data sets. Another very popular software that deals with VTK files is called as ParaView, you can have a look at that as well. You can export various random NumPy arrays of

Python to VTK files which you can then later on visualize using ParaView and ParaView is also a free open source software ok.

So, with this we end this particular series of lectures on bifurcations and next week we will start with flows on a circle and 2 dimensional flows. So, with this it is goodbye from me, happy problem solving and it is goodbye until then bye.