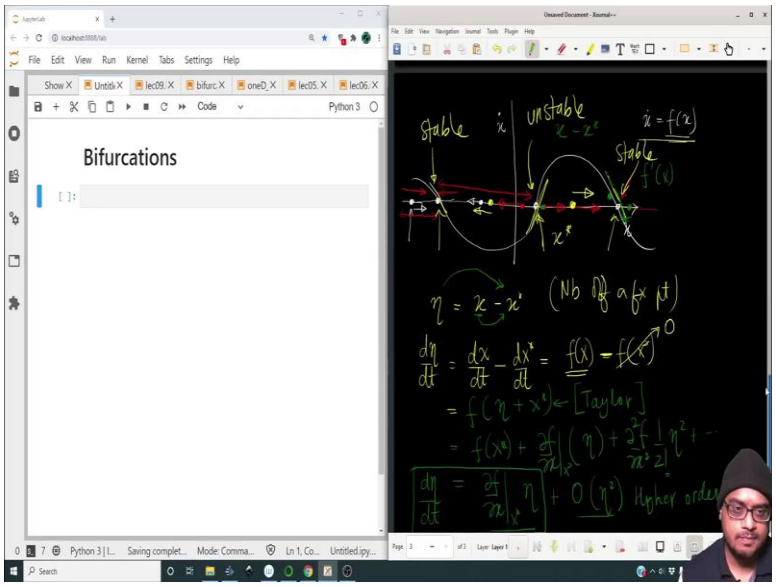
Tools in Scientific Computing

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

Lecture – 10
Bifurcations – Saddle-node Bifurcation

(Refer Slide Time: 00:27)


Bifurcations
stable
$\dot{x}$
unstable
$x - x^*$
$\dot{x} = f(x)$
stable
$f'(x)$
x^*
$\eta = x - x^*$
(Nb of a fix pt)
$\frac{d\eta}{dt} = \frac{dx}{dt} - \frac{dx^*}{dt} = f(x) - f(x^*)$
$= f(\eta + x^*) \leftarrow$ [Taylor]
$= f(x^*) + \frac{\partial f}{\partial x}\Big|_{x^*}(\eta) + \frac{\partial^2 f}{\partial x^2}\frac{1}{2!}\eta^2 + \cdots$
$\frac{d\eta}{dt} = \frac{\partial f}{\partial x}\Big|_{x^*}\eta + O(\eta^2)$ Higher order

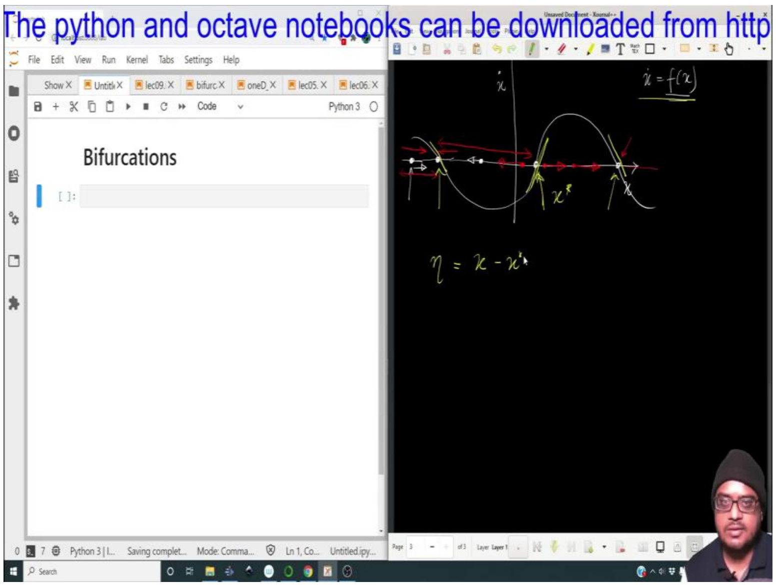
We are going look at Bifurcations. Before studying bifurcations, let us quickly take a look at something which we had studied in the previous lecture that is the stability of fixed points. So, we had seen fixed points in the last lecture. So, we have an equation $\dot{x} = f(x)$ and that curve looks something like this for example ok. So, wherever the curve $f(x)$ crosses 0, those are our fixed points.

And now, we had seen that if we have an initial condition over here, the $\dot{x}$ at this point is negative, as a consequence this point will try to move towards the left. If we have an initial condition over here, the $\dot{x}$ at this point is greater than 0 and so the point tries to move towards the right. Essentially, all the points in this place; all the points in this space and all the points in this space, they are attracted towards this point, this fixed point.

Whereas, if we have an a point over here, the $\dot{x}$ is positive so, it tries to go over here. So, this point is also an attractor, this point is a repeller because, point over here tries to go towards the right, a point over here tries to go towards the left. So, this helps us in defining stability of fixed points. So, there are some fixed points which attract like these and there are some fixed points which repel.

What dictates whether a fixed point attracts or not? It is quite obvious that the slope over here is positive so, it is a repelling point, the slope over here and over here is negative and so, its attracting, but let us try to prove that and while this is not a course for the theoretical aspects of all this, but I think this is something which would help us in the future.

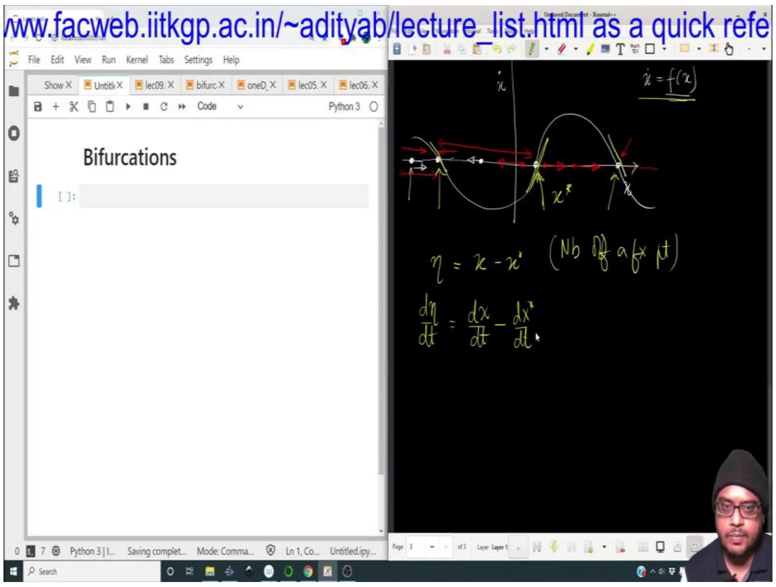
(Refer Slide Time: 02:57)


The python and octave notebooks can be downloaded from http

Bifurcations

So, let us consider the behavior of this equation near a fixed point. So, let us denote the fixed point by x^*.

(Refer Slide Time: 03:17)


ww.facweb.iitkgp.ac.in/~adityab/lecture_list.html as a quick refe

Bifurcations

$\dot{x} = f(x)$

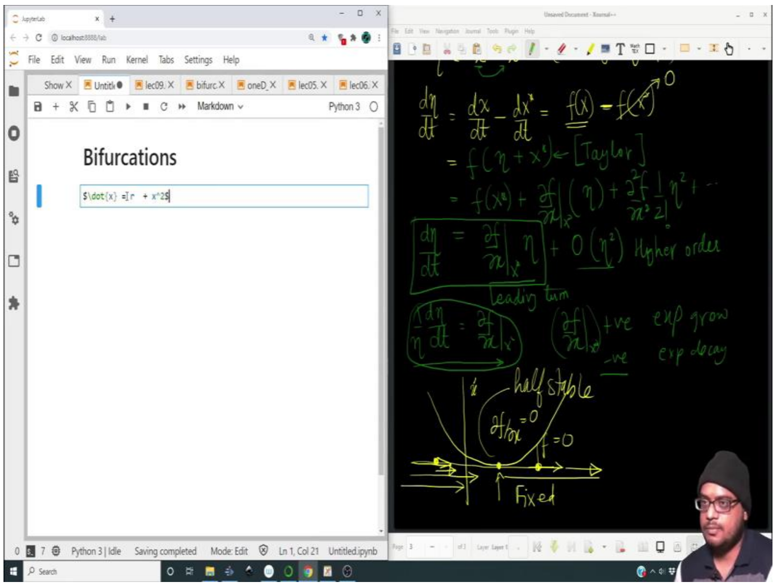
x^*

$\eta = x - x^*$ (Nb of a fx pt)

$\frac{d\eta}{dt} = \frac{dx}{dt} - \frac{dx^*}{dt}$

So, let us consider the behavior of the difference of x from x^* that is the behavior of x in the neighborhood of a fixed point. So, it is in the neighborhood of a fixed point. So, $\frac{d\eta}{dt} = \frac{dx}{dt} - \frac{dx^*}{dt}$ and $\frac{dx}{dt} = f(x)$ while $\frac{dx^*}{dt} = f(x^*) = 0$ because it is a fixed point by definition ok.

(Refer Slide Time: 05:32)


Bifurcations

```
$\dot{x} =[r + x^2$
```

$\frac{d\eta}{dt} = \frac{dx}{dt} - \frac{dx^*}{dt} = f(x) - \cancel{f(x^*)} \to 0$

$= f(\eta + x^*) \leftarrow$ [Taylor]

$= f(x^*) + \frac{\partial f}{\partial x}\Big|_{x^*}(\eta) + \frac{\partial^2 f}{\partial x^2}\frac{1}{2!}\eta^2 + \cdots$

$\boxed{\frac{d\eta}{dt} = \underbrace{\frac{\partial f}{\partial x}\Big|_{x^*}\eta}_{\text{leading term}}} + O(\eta^2)$ Higher order

$\frac{1}{\eta}\frac{d\eta}{dt} = \frac{\partial f}{\partial x}\Big|_{x^*}$

$\left(\frac{\partial f}{\partial x}\Big|_{x^*}\right)$ +ve exp grow

-ve exp decay

So, now let us write $f(x)$ in terms of a Taylor series expansion; so, x so, this can be written as $f(\eta + x^*)$. So, $f(x^*) + \left(\frac{\partial f}{\partial x}\right)_{x^*}(\eta) + \left(\frac{\partial^2 f}{\partial x^2}\right)\frac{1}{2!}(\eta^2) + \cdots\cdots$ and so on. So, this is just a Taylor series expansion. So, we have performed a Taylor series expansion of this particular expression and at the fixed point, $f(x^*) = 0$. So, this boils down to $\left(\frac{\partial f}{\partial x}\right)_{x^*}(\eta)$.

So, $\frac{d\eta}{dt} = \left(\frac{\partial f}{\partial x}\right)_{x^*}(\eta) + O(\eta^2)$. So, these are all higher order terms and if η is small that is if we are really looking into the neighborhood of x^*, then η will be quite small because x is very close to x^* so, the higher order term contributions will be small compared to the leading order term. So, this particular term is the leading order term.

So, if we just focus on this particular equation, it says that $\frac{1}{\eta}\frac{d\eta}{dt} = \left(\frac{\partial f}{\partial x}\right)_{x^*}$ and if $\left(\frac{\partial f}{\partial x}\right)$ is positive, then we have exponential growth ok. So, if $\left(\frac{\partial f}{\partial x}\right)_{x^*}$ is positive, then you have exponential growth locally and if $\left(\frac{\partial f}{\partial x}\right)$ is negative, then you have exponential decay locally and this is what we have physically seen over here.

Here, $\left(\frac{\partial f}{\partial x}\right)$ so, $f'(x)$ at this fixed point is negative and so, this is an attracting point because, points nearby will decay that is eta will reduce from this particular equation as time increases eta will reduce when $\left(\frac{\partial f}{\partial x}\right)$ is negative.

For this point as time increases, the η will increase that is the difference between x and x^* will increase so, this is a repelling point ok. So, it is an unstable point, it is also rather it is called as an unstable point while this is a stable point, this is a stable point.

And really, you do not need to go into all this algebra to really understand whether a point is stable or unstable. We can simply go by the same logic if you start at this point

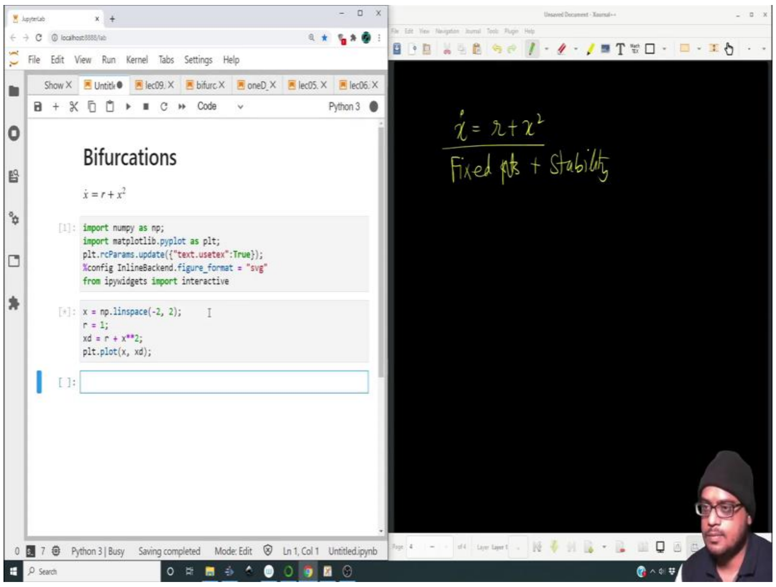
$\frac{dx}{dt}$ is positive and so, it will move towards the positive x. You start at this point $\frac{dx}{dt}$ is negative and so, you go towards lower values of x. So, what this small proof tells us that if $\left(\frac{\partial f}{\partial x}\right)$ is 0, then you have to look at the higher order terms, but geometrically it is much more easier to see.

So, let us draw the condition where $\left(\frac{\partial f}{\partial x}\right)$ is 0 something like this. At this point, $\left(\frac{\partial f}{\partial x}\right)$ is 0 and f(x) is also 0 so, obviously, this is a fixed point, but what happens over here? If we have this as an initial condition $\dot{x}$ at this point is positive so, at this point tends to go towards the right if we have an initial condition over here, once again the $\dot{x}$ is positive, and it tends to go over here.

So, all the points on this side they tend to go towards this, all the points on this side they tend to go away from this. So, it is called as a half stable point ok. It attracts half the points towards itself and it repels half the points away from it. I mean not half the points in the sense 50-50, but one set of points it attracts, one set of points it repels. So, in a 1D problem it is called as a half stable point.

So, with this thing out of the way, let us move on to bifurcations. So, often we will have conditions where the physics of the problem will be dictated by some controlling parameter. So, let us have a look at the ordinary differential equation $\dot{x} = r + x^2$ because this is a very famous prototypical problem.

(Refer Slide Time: 09:31)

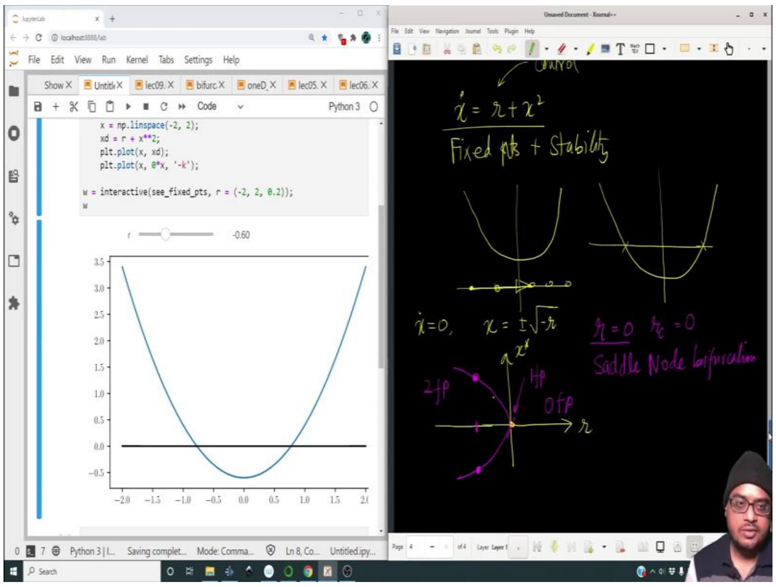


Bifurcations

$\dot{x} = r + x^2$

```
[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

[*]: x = np.linspace(-2, 2);
     r = 1;
     xd = r + x**2;
     plt.plot(x, xd);

[ ]:
```

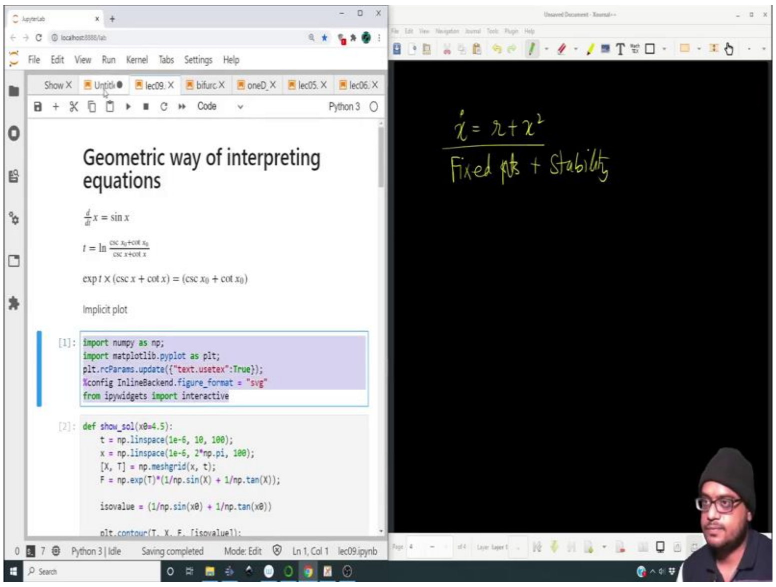
(Refer Slide Time: 09:38)



```
    x = np.linspace(-2, 2);
    xd = r + x**2;
    plt.plot(x, xd);
    plt.plot(x, 0*x, '-k');

w = interactive(see_fixed_pts, r = (-2, 2, 0.2));
w
```

So, the problem at hand is $\dot{x} = r + x^2$. So, let us try to look at this problem from a geometric viewpoint. Let us try to find out the fixed points and their stability.

(Refer Slide Time: 10:04)



Geometric way of interpreting equations

$\frac{d}{dt}x = \sin x$

$t = \ln \frac{\csc x_0 + \cot x_0}{\csc x + \cot x}$

$\exp t \times (\csc x + \cot x) = (\csc x_0 + \cot x_0)$

Implicit plot

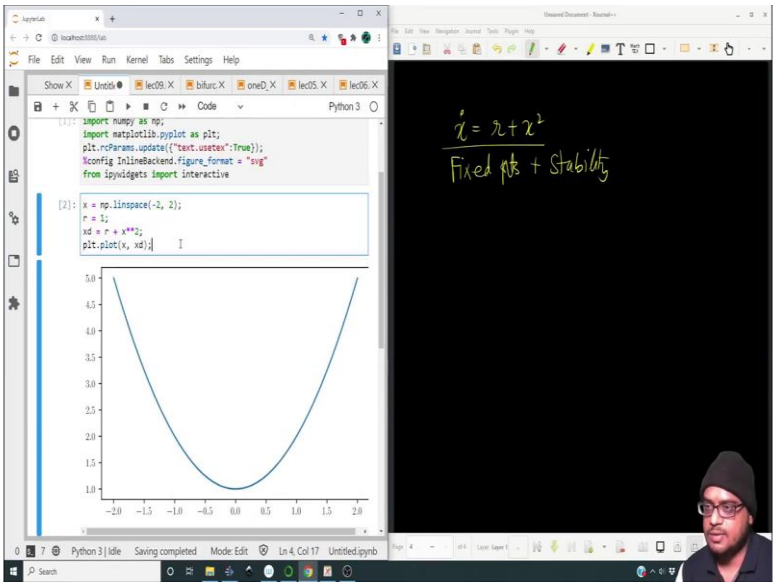
```
[1]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive

[2]: def show_sol(x0=4.5):
         t = np.linspace(1e-6, 10, 100);
         x = np.linspace(1e-6, 2*np.pi, 100);
         [X, T] = np.meshgrid(x, t);
         F = np.exp(T)*(1/np.sin(X) + 1/np.tan(X));

         isovalue = (1/np.sin(x0) + 1/np.tan(x0))

         plt.contour(T, X, F, [isovalue]);
```

$\dot{x} = r + x^2$

Fixed pts + Stability

So, let us go to our Python console. Let me copy paste this because we will be needing all of this alright.

(Refer Slide Time: 10:49)



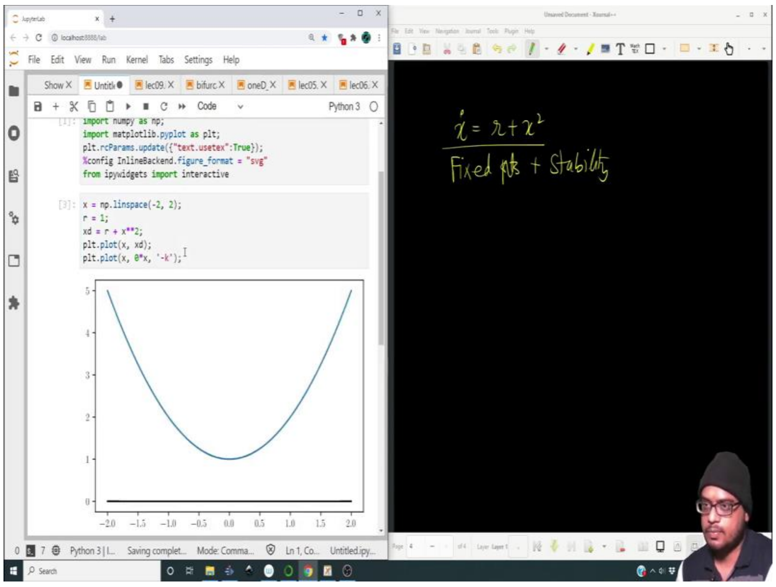
```
[1]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive

[2]: x = np.linspace(-2, 2);
     r = 1;
     xd = r + x**2;
     plt.plot(x, xd);
```

$\dot{x} = r + x^2$

Fixed pts + Stability

So, now, let us plot that diagram first. So, x = np.linspace(-2, 2), r=1 and xd = r + x**2, then we will do plt.plot(x, xd). So, it looks something like this.

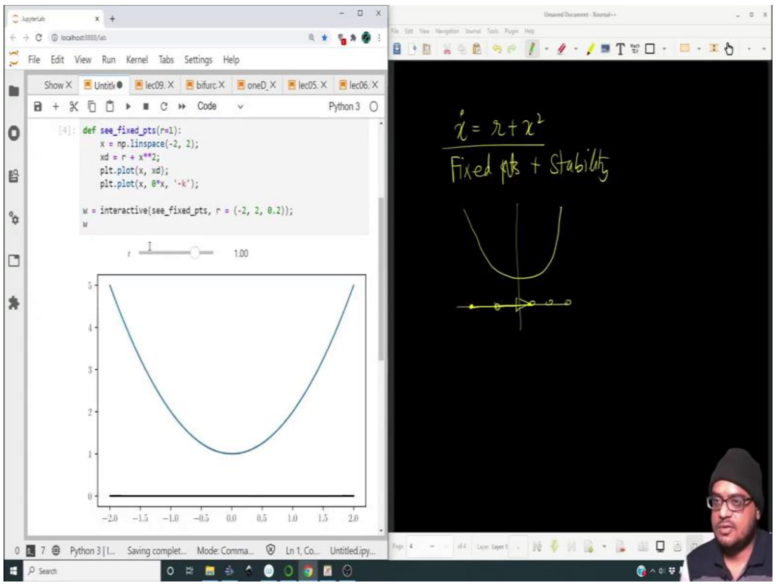
(Refer Slide Time: 10:55)



```
[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

[3]: x = np.linspace(-2, 2);
     r = 1;
     xd = r + x**2;
     plt.plot(x, xd);
     plt.plot(x, 0*x, '-k');
```

For completeness, let us also plot the x axis; so, plt.plot(x, 0*x, '-k'). Let us make it a black line alright. So, we see that when r is positive, this curve does not cut the x axis.

(Refer Slide Time: 11:47)

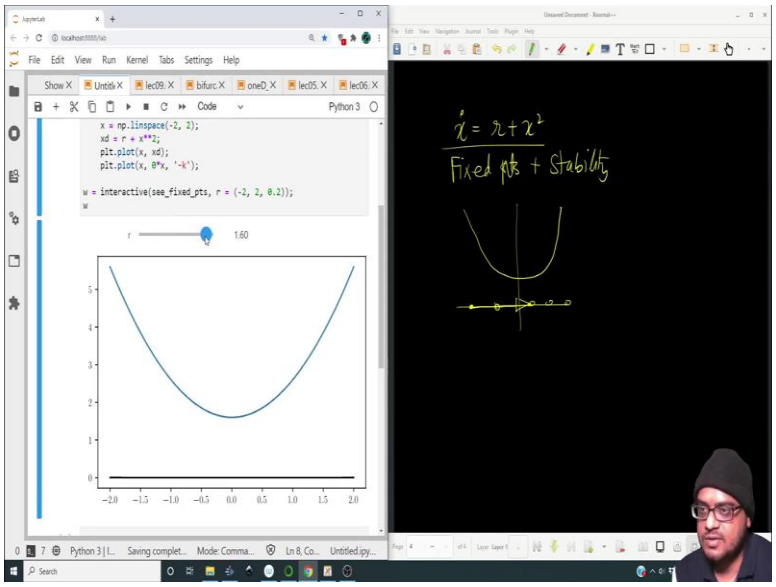


```
[4]: def see_fixed_pts(r=1):
         x = np.linspace(-2, 2);
         xd = r + x**2;
         plt.plot(x, xd);
         plt.plot(x, 0*x, '-k');

     w = interactive(see_fixed_pts, r = (-2, 2, 0.2));
     w
```

So, what is the meaning of this when the curve does not cut the x axis? It means there is no fixed point ok. So, all the initial conditions over here, they will be accelerated towards the right because for all points, everything is headed towards the right, all the x dots at all these points are positive and hence, all the flows are towards the right.

Let us make this interactive. Let me remove r, let me pass r as a input to the function. So, def see fixed points and the input will be r let us set a default value of 1 and over here, we will put everything inside the function definition and we will say w = interactive(see_fixed_pts, r = (-2, 2, 0.2)), we will display the widget. So, this is the widget.

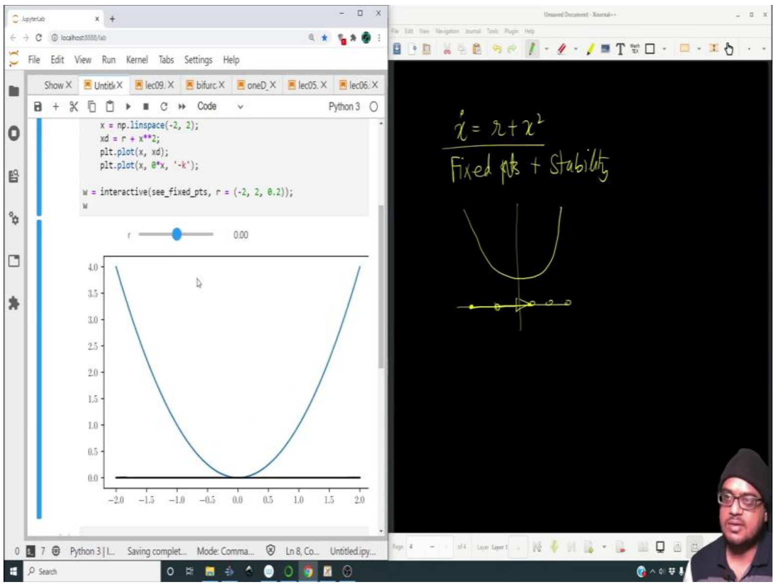
(Refer Slide Time: 12:35)



```
    x = np.linspace(-2, 2);
    xd = r + x**2;
    plt.plot(x, xd);
    plt.plot(x, 0*x, '-k');

w = interactive(see_fixed_pts, r = (-2, 2, 0.2));
w
```

So, let me change r, let me increase it so, obviously, increasing it makes it go further towards the upper direction. So, there is, there are no roots.

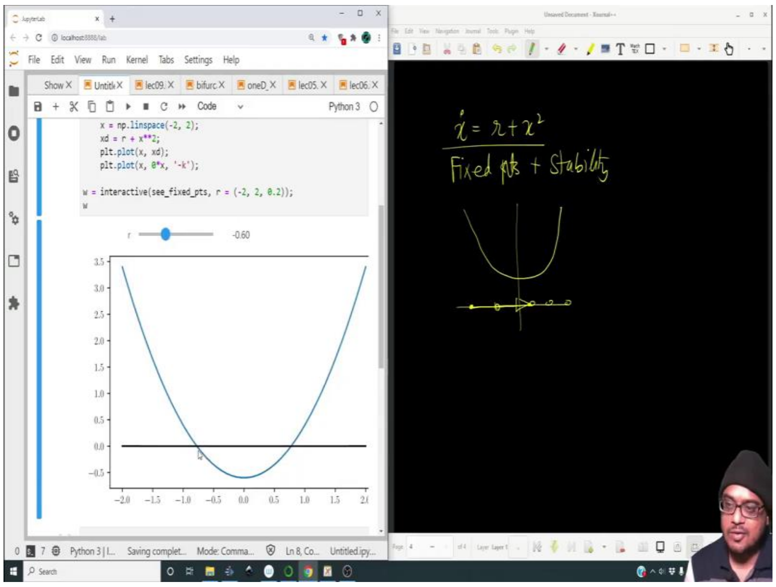
(Refer Slide Time: 12:48)



```
    x = np.linspace(-2, 2);
    xd = r + x**2;
    plt.plot(x, xd);
    plt.plot(x, 0*x, '-k');

w = interactive(see_fixed_pts, r = (-2, 2, 0.2));
w
```

As I reduce r, we see that as we reduce r, there is a point when r = 0, there is exactly one point of intersection that point of intersection is obviously, x = 0.

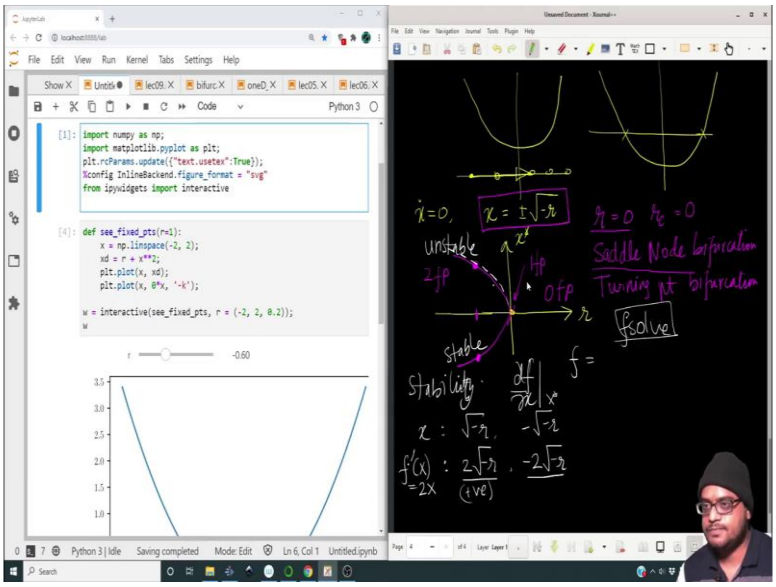
(Refer Slide Time: 12:58)



```
    x = np.linspace(-2, 2);
    xd = r + x**2;
    plt.plot(x, xd);
    plt.plot(x, 0*x, '-k');

w = interactive(see_fixed_pts, r = (-2, 2, 0.2));
w
```

When r is negative, we will have two points of intersection and so, we have two fixed points.

(Refer Slide Time: 15:43)



```
[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

[4]: def see_fixed_pts(r=1):
         x = np.linspace(-2, 2);
         xd = r + x**2;
         plt.plot(x, xd);
         plt.plot(x, 0*x, '-k');

     w = interactive(see_fixed_pts, r = (-2, 2, 0.2));
     w
```

r -0.60
$\dot{x}=0$, $x = \pm\sqrt{-r}$
$r = 0$ $x^* = 0$
Saddle Node bifurcation
Turning pt bifurcation
fsolve
unstable
2fp
x^*
0fp
r
stable
Stability
$\frac{\partial f}{\partial x}\Big|_{x^*}$
f =
x : $\sqrt{-r}$, $-\sqrt{-r}$
$f'(x) = 2x$: $2\sqrt{-r}$, $-2\sqrt{-r}$
(+ve)

So, when r < 0, we have two fixed points. So, what are the values of fixed points? Let us see. At the fixed point, $\dot{x}$ = 0 so, $x = \pm\sqrt{(-r)}$. So, when r is positive obviously, there are no roots, no real roots. When r is negative, there will be two roots, when r is 0 there will be one root which is x = 0.

So, what is happening? As we are changing the control parameter r so, r can be interpreted as a control parameter. As we are changing the value of r from negative to positive, we are having change in the fundamental behavior of the system ok. So, if I draw on this axis r and on this axis the fixed points.

For r positive, there are no fixed points. For r = 0, 0 is a fixed point. For r negative, there are two fixed points ok. So, for r = this value, the fixed points are this and this. So, there is a fundamental change in the behavior. As we cross r = 0 ok; r = 0 is a point, is a changeover point from having two roots to having no root ok.

So, here there are two fixed points, here there is one fixed point and here there are no fixed points. Therefore, we say that the system has suffered bifurcation at r = 0 and the r critical is = 0. This kind of behavior is called as bifurcation because, by changing r, we are changing the fundamental characteristic of the equation. So, this occurrence is called as saddle node bifurcation.

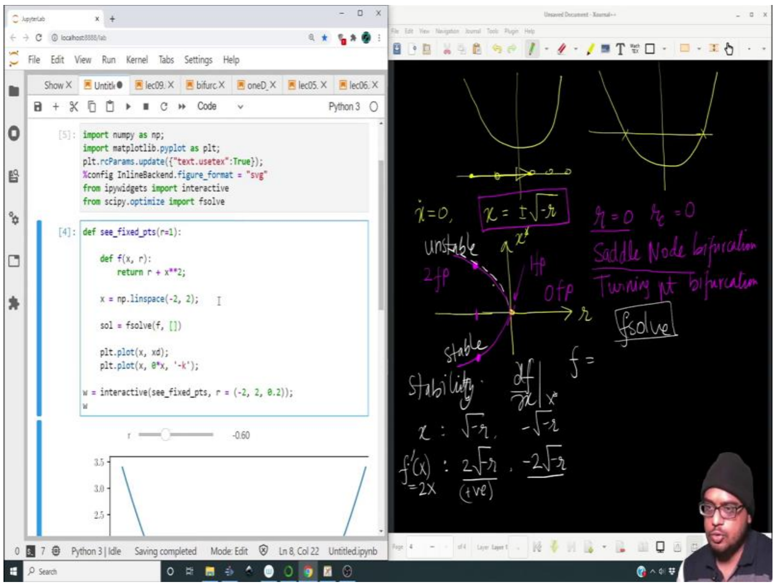
And the saddle node bifurcation in this particular context makes no sense because saddle node bifurcation is more generally valid for phase portraits in three-dimensions or bifurcation diagrams in three-dimensions. So, over here, it is also called as a turning point bifurcation. Its called a turning point because we will see why it is called a turning point.

So, now, $x = \pm\sqrt{(r)}$ is a root right. So, we know that they are fixed points, but what is the stability of the fixed points? So, we have just seen that the stability depends on $\left(\frac{\partial f}{\partial x}\right)$ at the fixed point. So, the fixed point so, the first fixed point is $\sqrt{(-r)}$, second fixed point is $-\sqrt{(-r)}$ and the function f that we are plotting is this $r + x^2$.

So, $\left(\frac{\partial f}{\partial x}\right)$ is to going to be 2x. So, $f'(x)$ so, remember primes are special derivatives and dots are temporal derivatives. So, $f'(x) =$ 2x, it will be $2\sqrt{(-r)}$ and $-2\sqrt{(-r)}$. So, when r is negative, this is obviously, negative. When r is negative, this is obviously, positive.

So, the positive root gives the positive value of the slope and hence, it is unstable. So, this particular branch is the unstable branch whereas, this particular branch is the stable branch and in this particular case, we could find out everything analytically, but let us see how to implement this on Python.

(Refer Slide Time: 17:57)



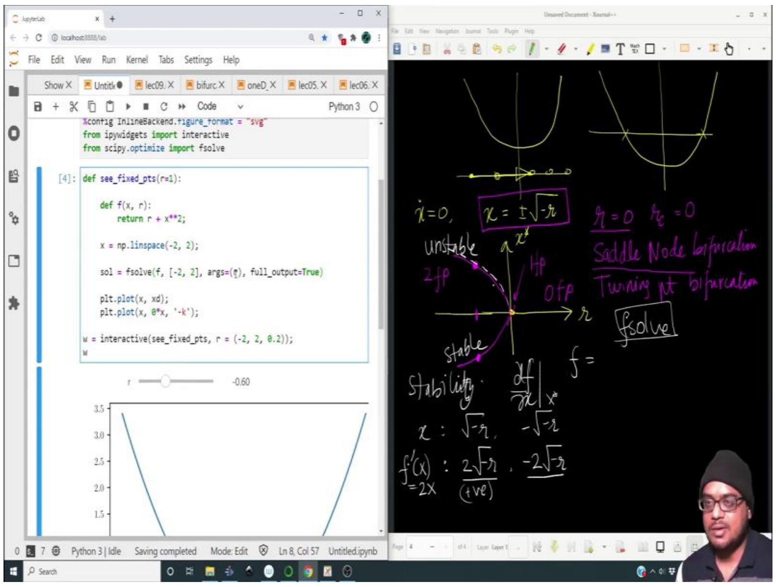
```
[5]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive
     from scipy.optimize import fsolve

[4]: def see_fixed_pts(r=1):

         def f(x, r):
             return r + x**2;

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

         sol = fsolve(f, [])

         plt.plot(x, xd);
         plt.plot(x, 0*x, '-k');

     w = interactive(see_fixed_pts, r = (-2, 2, 0.2));
     w
```

r -0.60
$\dot{x}=0,$
$\boxed{x = \pm\sqrt{-r}}$
$r = 0$ $r_c = 0$
Saddle Node bifurcation
Turning pt bifurcation
unstable
2fp
1fp
0fp
x^*
r
stable
fsolve
$f =$
Stability: $\left.\frac{\partial f}{\partial x}\right|_{x^*}$
$x : \sqrt{-r}, \; -\sqrt{-r}$
$f'(x) = 2x : 2\sqrt{-r} \; (+ve), \; -2\sqrt{-r}$

So, we do have this function where we have wrapped it and now, we are more interested to find out the roots. So, I mean here, analytically its possible, but still let us make use of Python’s libraries to find out the roots. So, for that let us import fsolve. So, if you recall from the previous lectures, fsolve is a solver which finds out roots of non-linear algebraic equations.

So, let us import that. From scipy dot optimize import fsolve alright. So, for using fsolve, we have to pass a function handle. So, this particular thing has to be made into a function handle. So, def f and it will take as an input x and it will return $r + x^2$ and let this function take as an optional parameter not an optional parameter as an argument r ok. So, it will take the value of x and r and it will return me the f(x) that we have in the analytical expression alright.

(Refer Slide Time: 19:20)



```
%config InlineBackend.figure_format = "svg"
from ipywidgets import interactive
from scipy.optimize import fsolve
```

```
[4]: def see_fixed_pts(r=1):

         def f(x, r):
             return r + x**2;

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

         sol = fsolve(f, [-2, 2], args=(r), full_output=True)

         plt.plot(x, xd);
         plt.plot(x, 0*x, '-k');

     w = interactive(see_fixed_pts, r = (-2, 2, 0.2));
     w
```

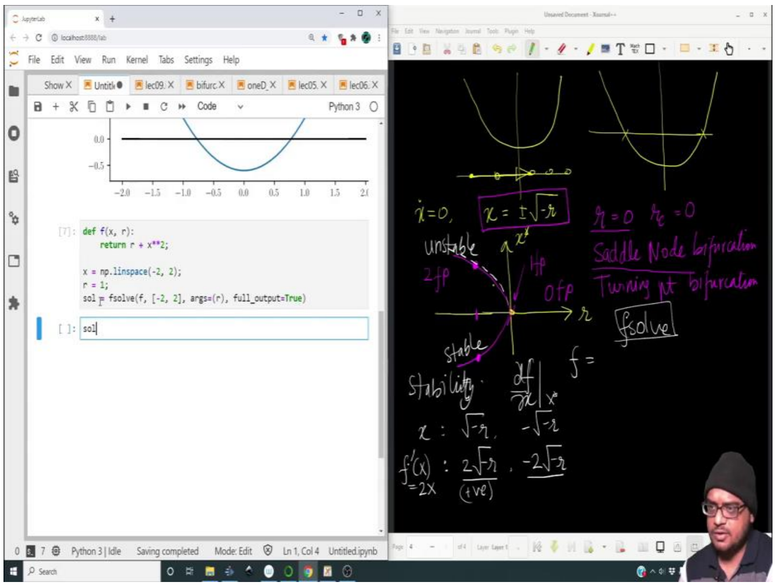
r -0.60
$\dot{x}=0$, $x=\pm\sqrt{-r}$
unstable
2fp
x^*
Hp
0fp
stable
$r=0$ $r_c=0$
Saddle Node bifurcation
Turning pt bifurcation
fsolve
$f=$
Stability: $\left.\frac{\partial f}{\partial x}\right|_{x^*}$
x : $\sqrt{-r}$, $-\sqrt{-r}$
$f'(x) = 2x$: $2\sqrt{-r}$ (+ve), $-2\sqrt{-r}$

So, once we have this, let us say sol is = fsolve the function handle f, the guess value so, let me put two guess values one on this side so, let one guess value be -2 and let one guess value be 2, the arguments to the function so, args = r and let us pass the full output to the solution.

So, we need the full output because we need to know whether there has there is a root found or not because obviously, for r positive, there will be no root. So, fsolve will say I did not find anything.

So, in order to assess that information whether it has been able to converge or not, it will pass all the information. In fact, let us take this in a different cell ok, let me copy this, let me go back to the previous cell, I do not want to disturb this cell, let me go to a different cell ok.

(Refer Slide Time: 20:14)

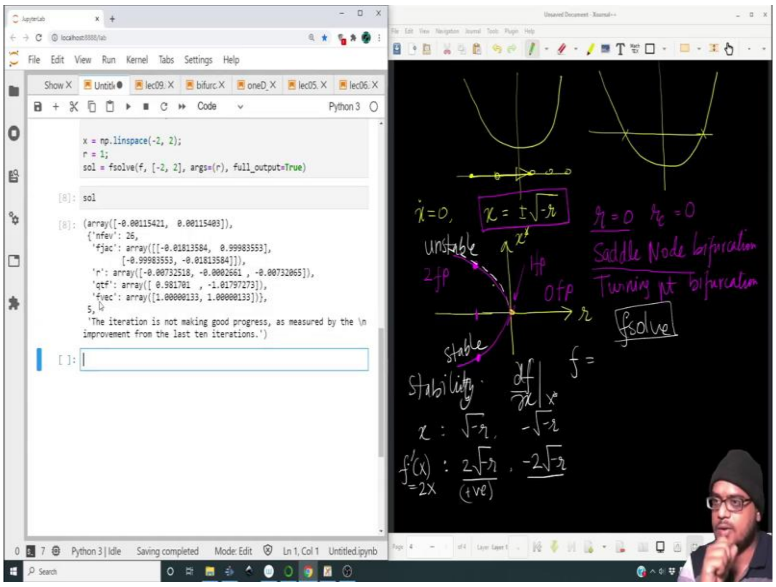


```
[7]: def f(x, r):
         return r + x**2;

     x = np.linspace(-2, 2);
     r = 1;
     sol = fsolve(f, [-2, 2], args=(r), full_output=True)

[ ]: sol
```

$\dot{x}=0, \quad \boxed{x = \pm\sqrt{-r}}$
unstable
2fp
0fp
stable
Stability
$\left.\frac{\partial f}{\partial x}\right|_{x^*}$
$f =$
$x : \sqrt{-r}, \quad -\sqrt{-r}$
$f'(x) : 2\sqrt{-r}, \quad -2\sqrt{-r}$
$=2x$
(+ve)
$r = 0 \quad r_c = 0$
Saddle Node bifurcation
Turning pt bifurcation
$\boxed{fsolve}$

So, let me make the correct ok. So, let me run this and see if this is an error. So, it ran properly. I have not yet passed r from this cell; it is using the r from the previous cell. So, let r = 1. So, now r is positive, and we expect the solution to not have converged. So, let us see what sol is excellent.

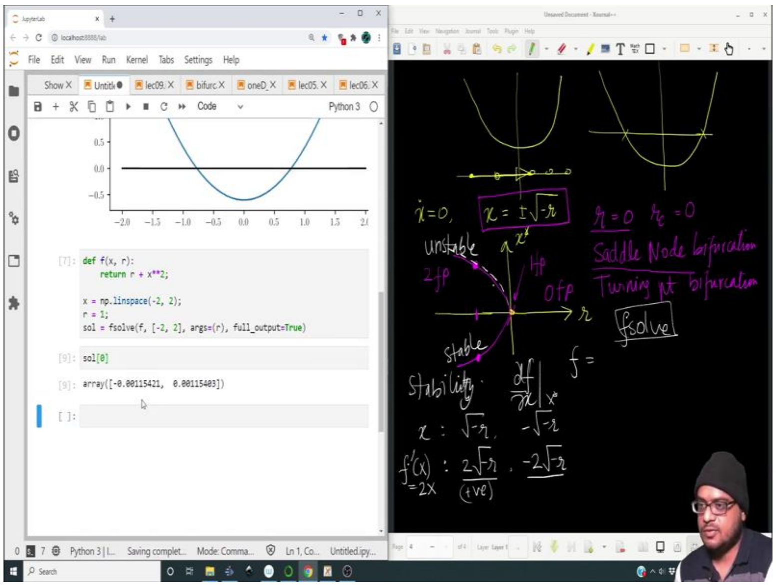
(Refer Slide Time: 20:40)



```
     x = np.linspace(-2, 2);
     r = 1;
     sol = fsolve(f, [-2, 2], args=(r), full_output=True)

[8]: sol

[8]: (array([-0.00115421,  0.00115403]),
      {'nfev': 26,
       'fjac': array([[-0.01813584,  0.99983553],
              [-0.99983553, -0.01813584]]),
       'r': array([-0.00732518, -0.0002661 , -0.00732065]),
       'qtf': array([ 0.981701  , -1.01797273]),
       'fvec': array([1.00000133, 1.00000133])},
      5,
      'The iteration is not making good progress, as measured by the \n improvement from the last ten iterations.')

[ ]:
```

So, it contains two values and then, it says that the iteration is not making good progress ok. So, the meaning of this is its not able to converge properly ok. It is trying to do iterations, but its finding nothing.

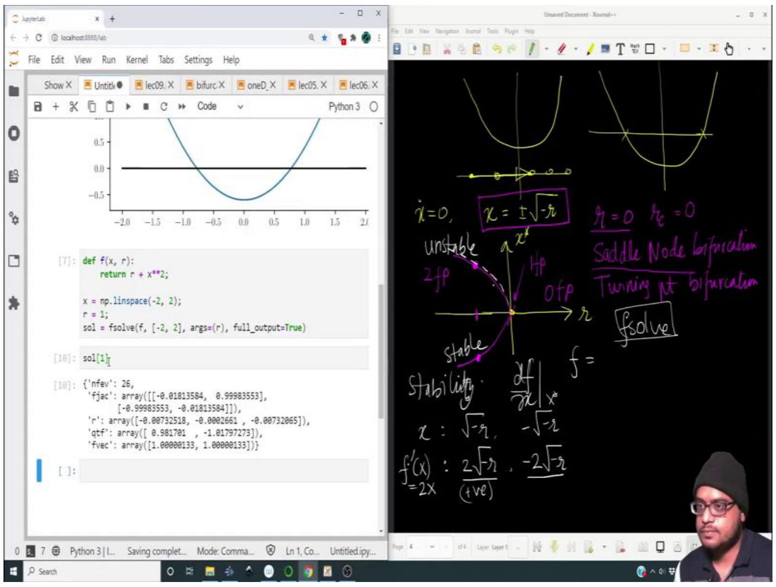
(Refer Slide Time: 21:16)



```
[7]: def f(x, r):
         return r + x**2;

     x = np.linspace(-2, 2);
     r = 1;
     sol = fsolve(f, [-2, 2], args=(r), full_output=True)

[9]: sol[0]

[9]: array([-0.00115421,  0.00115403])
```

So, in that case, let us see what sol[0] is? It gives this.

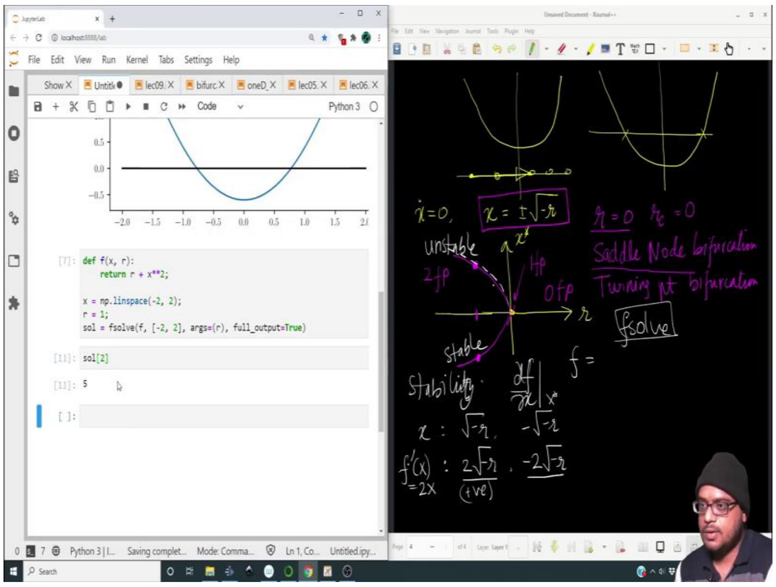
(Refer Slide Time: 21:23)



```
[7]: def f(x, r):
         return r + x**2;

     x = np.linspace(-2, 2);
     r = 1;
     sol = fsolve(f, [-2, 2], args=(r), full_output=True)

[10]: sol[1]

[10]: {'nfev': 26,
       'fjac': array([[-0.01813584,  0.99983553],
              [-0.99983553, -0.01813584]]),
       'r': array([-0.00732518, -0.0002661 , -0.00732065]),
       'qtf': array([ 0.981701  , -1.01797273]),
       'fvec': array([1.00000133, 1.00000133])}
```

What sol[1] is? Sorry.

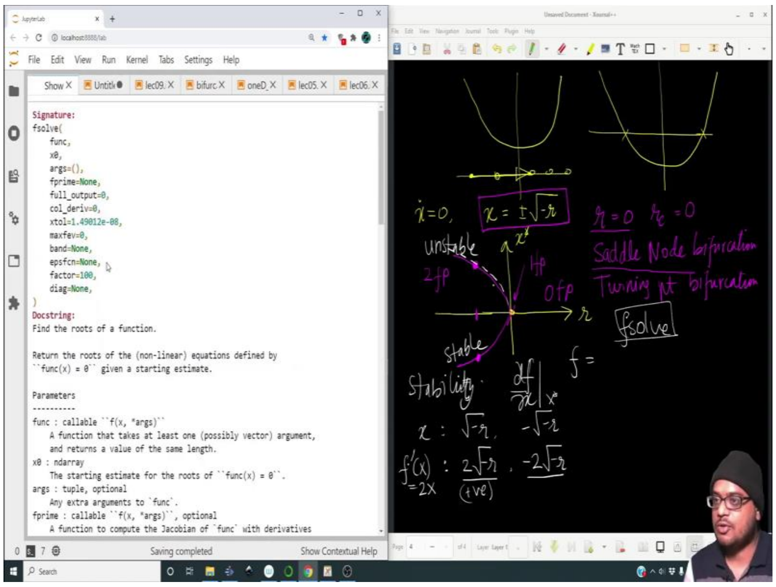
(Refer Slide Time: 21:26)



```
[7]: def f(x, r):
         return r + x**2;

     x = np.linspace(-2, 2);
     r = 1;
     sol = fsolve(f, [-2, 2], args=(r), full_output=True)

[11]: sol[2]

[11]: 5

[ ]:
```

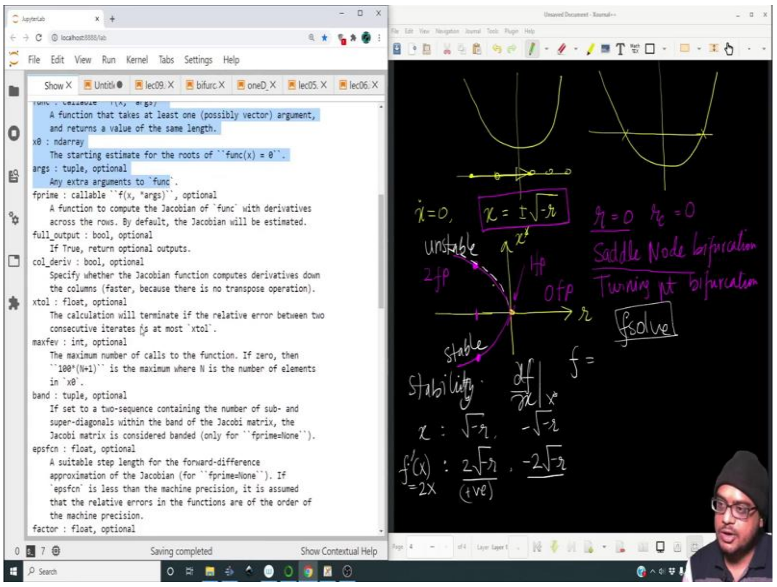
What sol[2] is? It says 5. So, the thing about fsolve is if it gives an output, this sol[2] object will be 1 ok, this particular value that it is printing if it is not 1 that it means it has not converged and how do I know that? Let me double click on this and go to contextual help ok.

(Refer Slide Time: 21:48)



```
Signature:
fsolve(
    func,
    x0,
    args=(),
    fprime=None,
    full_output=0,
    col_deriv=0,
    xtol=1.49012e-08,
    maxfev=0,
    band=None,
    epsfcn=None,
    factor=100,
    diag=None,
)
Docstring:
Find the roots of a function.

Return the roots of the (non-linear) equations defined by
``func(x) = 0`` given a starting estimate.

Parameters
----------
func : callable ``f(x, *args)``
    A function that takes at least one (possibly vector) argument,
    and returns a value of the same length.
x0 : ndarray
    The starting estimate for the roots of ``func(x) = 0``.
args : tuple, optional
    Any extra arguments to `func`.
fprime : callable ``f(x, *args)``, optional
    A function to compute the Jacobian of `func` with derivatives
```

(Refer Slide Time: 21:51)

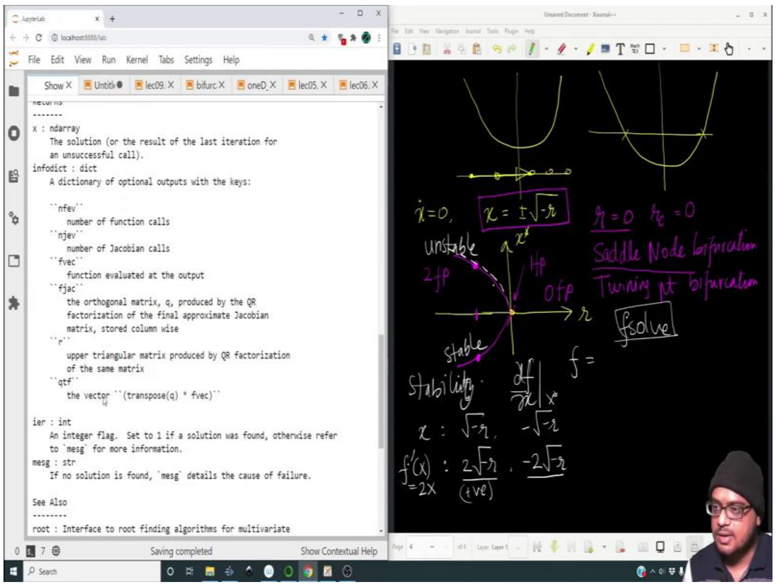


```
func : callable ``f(x, *args)``
    A function that takes at least one (possibly vector) argument,
    and returns a value of the same length.
x0 : ndarray
    The starting estimate for the roots of ``func(x) = 0``.
args : tuple, optional
    Any extra arguments to `func`.
fprime : callable ``f(x, *args)``, optional
    A function to compute the Jacobian of `func` with derivatives
    across the rows. By default, the Jacobian will be estimated.
full_output : bool, optional
    If True, return optional outputs.
col_deriv : bool, optional
    Specify whether the Jacobian function computes derivatives down
    the columns (faster, because there is no transpose operation).
xtol : float, optional
    The calculation will terminate if the relative error between two
    consecutive iterates is at most `xtol`.
maxfev : int, optional
    The maximum number of calls to the function. If zero, then
    ``100*(N+1)`` is the maximum where N is the number of elements
    in `x0`.
band : tuple, optional
    If set to a two-sequence containing the number of sub- and
    super-diagonals within the band of the Jacobi matrix, the
    Jacobi matrix is considered banded (only for ``fprime=None``).
epsfcn : float, optional
    A suitable step length for the forward-difference
    approximation of the Jacobian (for ``fprime=None``). If
    `epsfcn` is less than the machine precision, it is assumed
    that the relative errors in the functions are of the order of
    the machine precision.
factor : float, optional
```

$\dot{x}=0$, $x = \pm\sqrt{-r}$
$r=0$ $r_c=0$
Saddle Node bifurcation
Turning pt bifurcation
fsolve
unstable
2fp
x^*
fp
0fp
r
stable
Stability
$f =$
$\frac{\partial f}{\partial x}\Big|_{x^*}$
$x : \sqrt{-r}, \; -\sqrt{-r}$
$f'(x) : 2\sqrt{-r}, \; -2\sqrt{-r}$
$=2x$
(+ve)

So, function, x_0, args so, this is how you pass everything, we can pass the Jacobian as well, we do not need to do that.

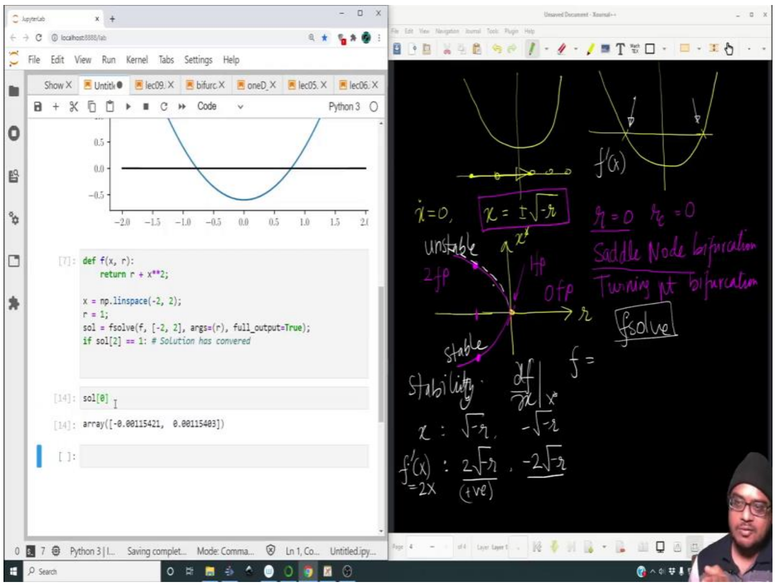
(Refer Slide Time: 22:03)



```
Returns
-------
x : ndarray
    The solution (or the result of the last iteration for
    an unsuccessful call).
infodict : dict
    A dictionary of optional outputs with the keys:

    ``nfev``
        number of function calls
    ``njev``
        number of Jacobian calls
    ``fvec``
        function evaluated at the output
    ``fjac``
        the orthogonal matrix, q, produced by the QR
        factorization of the final approximate Jacobian
        matrix, stored column wise
    ``r``
        upper triangular matrix produced by QR factorization
        of the same matrix
    ``qtf``
        the vector ``(transpose(q) * fvec)``

ier : int
    An integer flag.  Set to 1 if a solution was found, otherwise refer
    to `mesg` for more information.
mesg : str
    If no solution is found, `mesg` details the cause of failure.

See Also
--------
root : Interface to root finding algorithms for multivariate
```

It returns what? It returns x and the info directory. So, this is the info directory that we just read, number of function calls, number of Jacobian calls we do not need all that. It outputs an error integer. It is an integer flag, it is set to 1 if solution is found, otherwise it will not return 1, it will return some other value. So, it has returned to us a value of 5. It means the solution has not converged.

(Refer Slide Time: 22:39)



```
[7]: def f(x, r):
         return r + x**2;

     x = np.linspace(-2, 2);
     r = 1;
     sol = fsolve(f, [-2, 2], args=(r), full_output=True);
     if sol[2] == 1: # Solution has convered
```

```
[14]: sol[0]
```

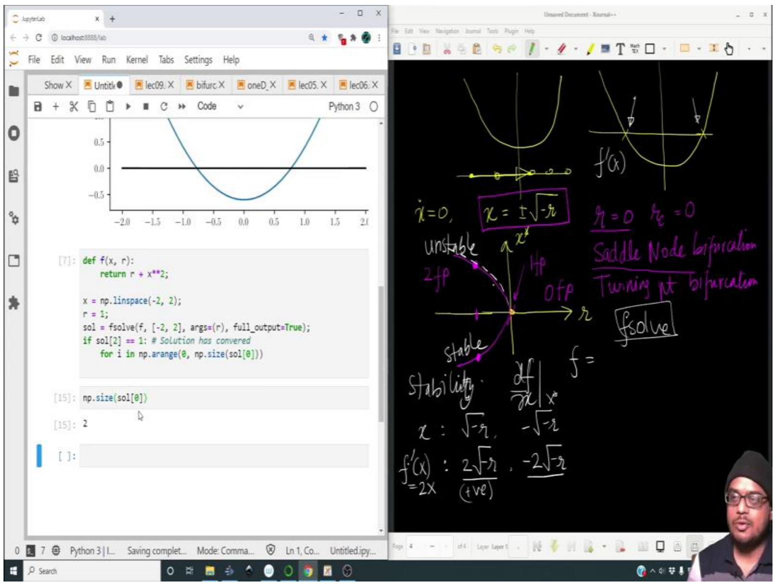
```
[14]: array([-0.00115421,  0.00115403])
```

$\dot{x}=0$, $\boxed{x=\pm\sqrt{-r}}$
unstable
2fp
Hp
0fp
stable
r
x^*
$f(x)$
$r=0$ $r_c=0$
Saddle Node bifurcation
Turning pt bifurcation
$\boxed{\text{fsolve}}$
$f=$
Stability $\left.\frac{\partial f}{\partial x}\right|_{x^*}$
x : $\sqrt{-r}$, $-\sqrt{-r}$
$f'(x)=2x$: $2\sqrt{-r}$ (+ve), $-2\sqrt{-r}$

So, let us make use of that information. So, after this line, we have obtained the value of sol. So, if sol[2] $\neq$ 1 or rather if sol[1] sol[2] = 1, then we should do something with it. So, if sol[2] = 1, let us classify the roots. Meaning, when I know that roots are there like this, I know that there are roots, let me check the value of $f'(x)$ at those roots and classify those roots as being a stable fixed point or an unstable fixed point.

So, if sol[2] = 1, this obviously, means solution has converged. So, now, I do not know how many roots this has ok. I know in this particular question that I have two roots. In general, I will not know how many roots I have. So, I must loop over sol[0] because look sol[0] will that number of elements of sol[0] will be the number of roots that fsolve has found.

(Refer Slide Time: 24:02)

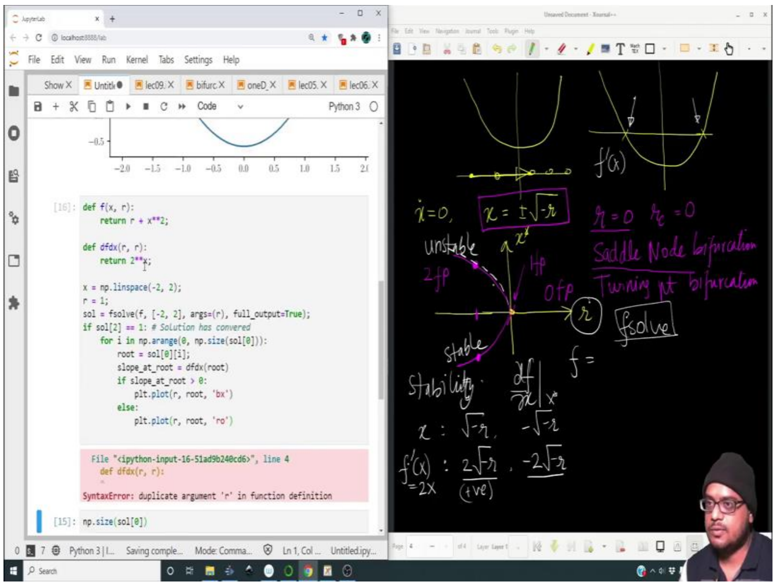


```
[7]: def f(x, r):
         return r + x**2;

     x = np.linspace(-2, 2);
     r = 1;
     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]))

[15]: np.size(sol[0])

[15]: 2
```

$\dot{x}=0$, $x=\pm\sqrt{-r}$
unstable
2 fp
1 fp
0 fp
stable
$f(x)$
$r=0$ $r_c=0$
Saddle Node bifurcation
Turning pt bifurcation
fsolve
Stability : $\left.\frac{\partial f}{\partial x}\right|_{x^*}$
$f=$
x : $\sqrt{-r}$, $-\sqrt{-r}$
$f'(x)=2x$: $2\sqrt{-r}$ (+ve), $-2\sqrt{-r}$

So, then I will say **for** i **in** np.arange(0, np.size(sol[0])). So, just to give you a context, np.size (sol[0]) = 2. If I had 5 roots, np dot size of sol would have been 5.

(Refer Slide Time: 24:37)



```
[16]: def f(x, r):
          return r + x**2;

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

      x = np.linspace(-2, 2);
      r = 1;
      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)
              if slope_at_root > 0:
                  plt.plot(r, root, 'bx')
              else:
                  plt.plot(r, root, 'ro')

  File "<ipython-input-16-51ad9b240cd6>", line 4
    def dfdx(r, r):
    ^
SyntaxError: duplicate argument 'r' in function definition

[15]: np.size(sol[0])
```

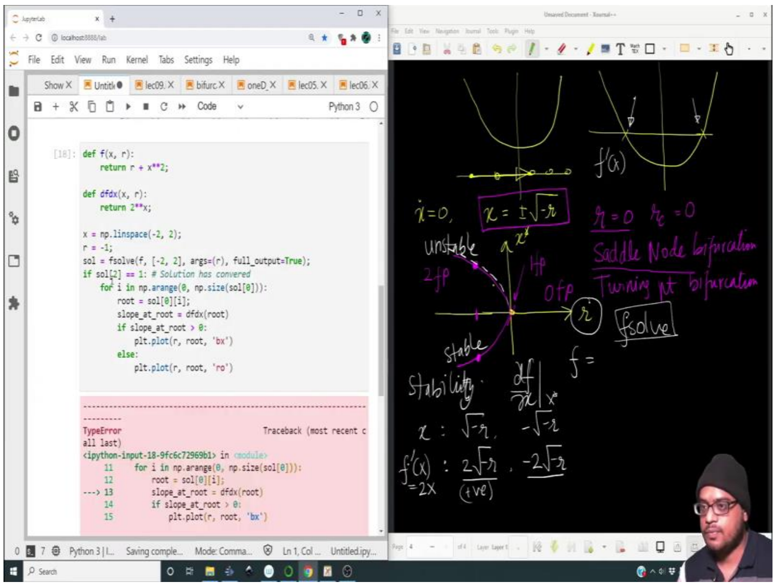
$\dot{x}=0$, $x=\pm\sqrt{-r}$
unstable
2 fp
1 fp
0 fp
stable
$f(x)$
$r=0$ $r_c=0$
Saddle Node bifurcation
Turning pt bifurcation
fsolve
Stability : $\left.\frac{\partial f}{\partial x}\right|_{x^*}$
$f=$
x : $\sqrt{-r}$, $-\sqrt{-r}$
$f'(x)=2x$: $2\sqrt{-r}$ (+ve), $-2\sqrt{-r}$

So, I do a loop over all the roots for i in this. What should I do? I should check the root. So, root = sol[0], i. So, this is the root that we extract. So, I will loop over all the roots and sol root will contain iteratively the first root, the second root, the third root, the fourth root, the fifth root how many number of roots you have.

So, after this, I will check slope at root is = $\left(\frac{\partial f}{\partial x}\right)$, I have to make a function $\left(\frac{\partial f}{\partial x}\right)$ which will evaluate the slope of root. So, let me make that function, **def** dfdx(x, r), **return** 2*x ok. So, it will return the slope at the root. If slope at root is greater than 0, then I should plot it. So, how should I plot it? I should plot as a cross because it is an unstable point.

So, plt.plot so, I have to plot on the x axis the r so, over on the x axis, I have to plot the value of r and on the y axis, I must plot the value of the root and I must plot it as an x. Let me make it as a blue x ok, else plot r comma root as a red circle. So, if it is a stable root, it will plot a red circle. If it is an unstable root, it will make a blue cross ok.

(Refer Slide Time: 27:05)



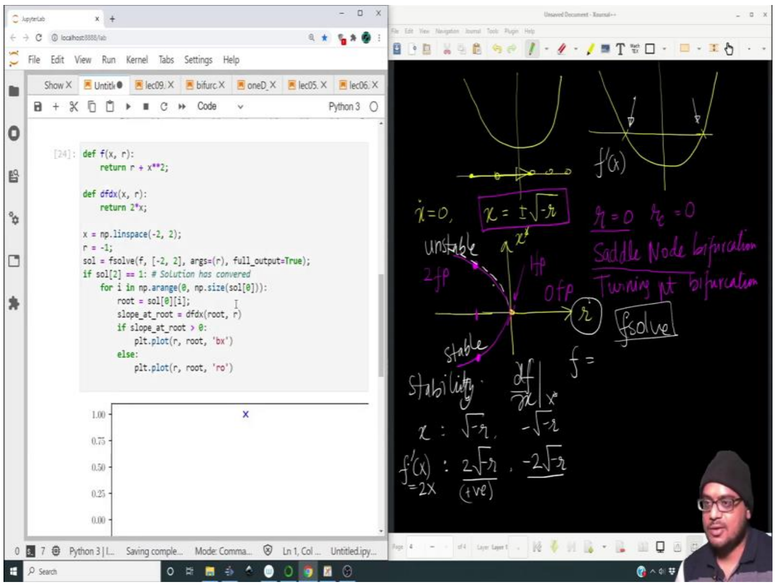
```
[18]: def f(x, r):
          return r + x**2;

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

      x = np.linspace(-2, 2);
      r = -1;
      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)
              if slope_at_root > 0:
                  plt.plot(r, root, 'bx')
              else:
                  plt.plot(r, root, 'ro')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-9fc6c72969b1> in <module>
     11     for i in np.arange(0, np.size(sol[0])):
     12         root = sol[0][i];
---> 13         slope_at_root = dfdx(root)
     14         if slope_at_root > 0:
     15             plt.plot(r, root, 'bx')
```

f(x)
$\dot{x}=0$, $x=\pm\sqrt{-r}$
$r=0$ $r_c=0$
Saddle Node bifurcation
Turning pt bifurcation
unstable
2 fp
1 fp
0 fp
stable
r
fsolve
f =
Stability
$\left.\frac{\partial f}{\partial x}\right|_{x^*}$
x : $\sqrt{-r}$, $-\sqrt{-r}$
$f'(x)=2x$: $2\sqrt{-r}$ (+ve), $-2\sqrt{-r}$

So, let us see what happens ok, there is an error duplicate argument in function. So, this has to be x ok.

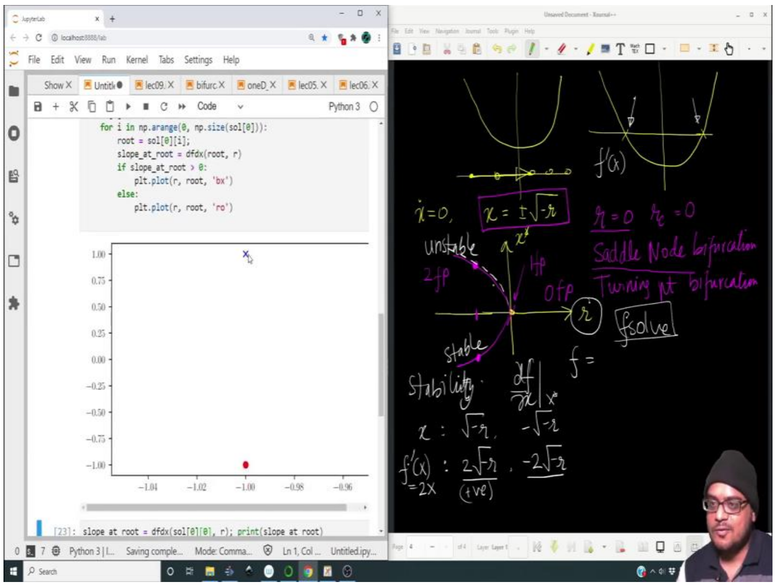
(Refer Slide Time: 27:13)



```
[24]: def f(x, r):
          return r + x**2;

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

      x = np.linspace(-2, 2);
      r = -1;
      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')
```

So, obviously, there was no root so, there is no plot. Let me make r = -1. I have to pass two functions. Well, obviously, we have made a small mistake, this should not be 2 ** x, this has to be only 2 * x ok.

(Refer Slide Time: 27:26)

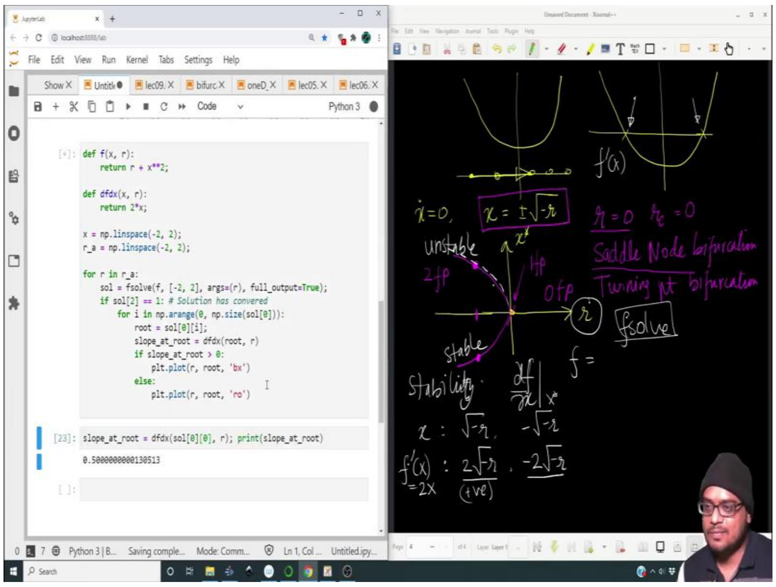


```
    for i in np.arange(0, np.size(sol[0])):
        root = sol[0][i];
        slope_at_root = dfdx(root, r)
        if slope_at_root > 0:
            plt.plot(r, root, 'bx')
        else:
            plt.plot(r, root, 'ro')
```

```
[23]: slope at root = dfdx(sol[0][0], r); print(slope at root)
```

So, let us run this. So, obviously, this is a unstable root as we have seen over here and this is a stable root as pointed by the red circle.

(Refer Slide Time: 27:46)

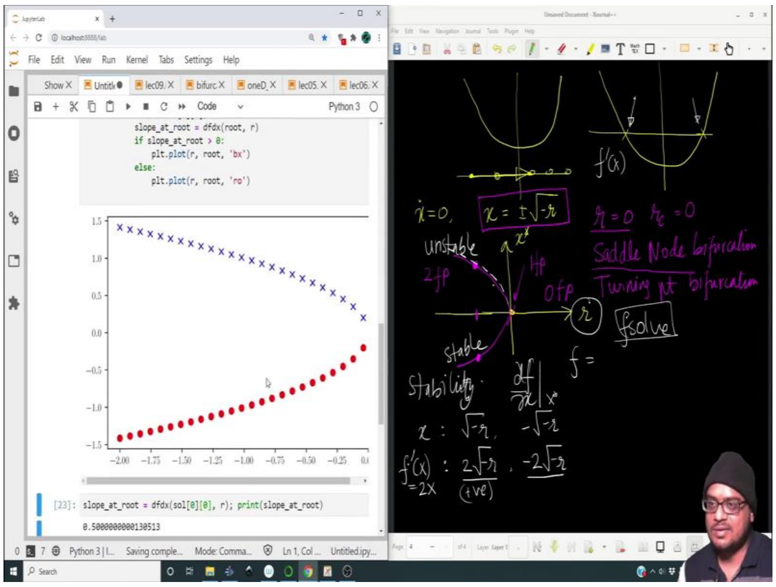

File Edit View Run Kernel Tabs Settings Help
Code
Python 3
$\dot{x}=0$, $x=\pm\sqrt{-r}$
$r=0$ $r_c=0$
Saddle Node bifurcation
Turning pt bifurcation
unstable
fsolve
stable
Stability
$f'(x)$: $2\sqrt{-r}$, $-2\sqrt{-r}$

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

     def dfdx(x, r):
         return 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')

[23]: slope_at_root = dfdx(sol[0][0], r); print(slope_at_root)

      0.5000000000130513
```

So, now, what do we need to do is to loop over various values of r and draw this entire diagram. So, what should I do? I have to make r_a = np.linspace(-2, 2) and I have to wrap everything inside a loop which will iterate over various values of r. So, I must put everything inside here over here inside a loop, inside a for loop.

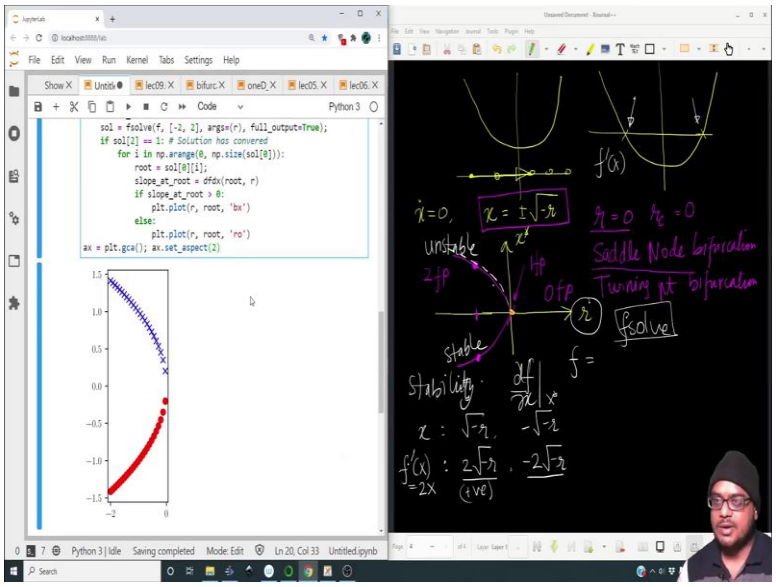
So, **for** r **in** r_a, then I have to indent everything because I want to execute this entire chunk of code inside the loop and that is it. So, let us execute this cell and see what happens excellent.

(Refer Slide Time: 28:25)



```
                 slope_at_root = dfdx(root, r)
                 if slope_at_root > 0:
                     plt.plot(r, root, 'bx')
                 else:
                     plt.plot(r, root, 'ro')

[23]: slope_at_root = dfdx(sol[0][0], r); print(slope_at_root)

      0.5000000000130513
```

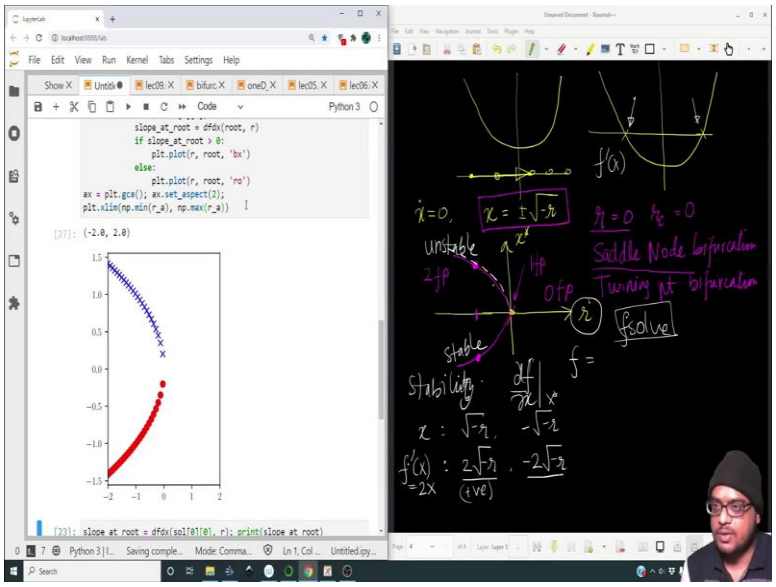
(Refer Slide Time: 28:35)



```
        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(2)
```

So, we do see, let me change the aspect ratio of the plot. So, ax = plt.gca(); ax.set_aspect(2) and let us make the x limit all the way to the r range that we have.

(Refer Slide Time: 28:56)

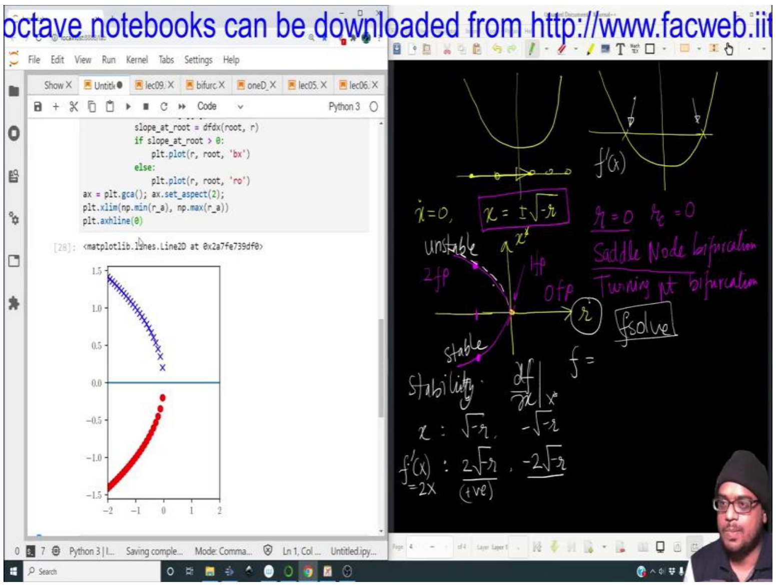


```
                slope_at_root = dfdx(root, r)
                if slope_at_root > 0:
                    plt.plot(r, root, 'bx')
                else:
                    plt.plot(r, root, 'ro')
ax = plt.gca(); ax.set_aspect(2);
plt.xlim(np.min(r_a), np.max(r_a))
```

```
[27]: (-2.0, 2.0)
```

So, plt.xlim(np.min(r_a), np.max(r_a));. So, this is how the bifurcation diagram looks like.

(Refer Slide Time: 29:21)

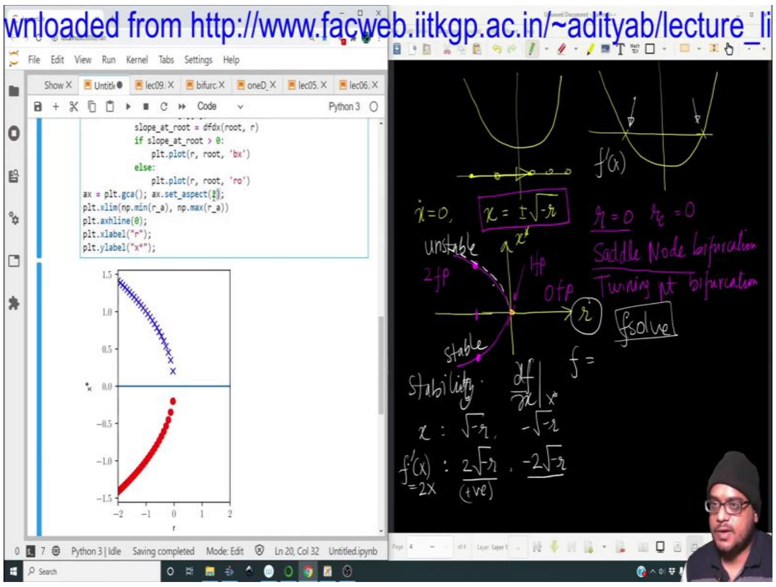

octave notebooks can be downloaded from http://www.facweb.iit

```
            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(2);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0)
```

```
[28]: <matplotlib.lines.Line2D at 0x2a7fe739df0>
```

Let us plot the x axis for completeness. So, plt plt.axhline(0). So, axhline is just a quick function for drawing a horizontal line passing through 0 ok. So, this shows us how the bifurcation occurs.

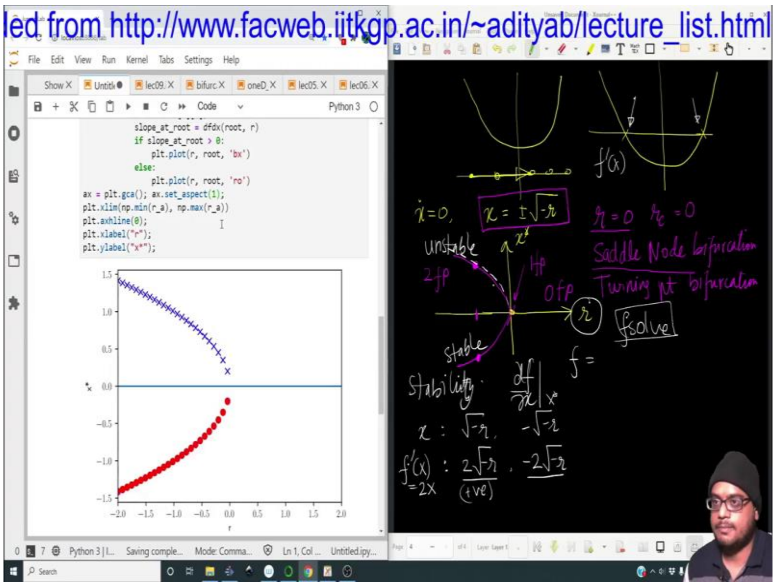
(Refer Slide Time: 29:48)


wnloaded from http://www.facweb.iitkgp.ac.in/~adityab/lecture_li

```
            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(2);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0);
plt.xlabel("r");
plt.ylabel("x*");
```

As we are changing the value of r right, let me make it 1.

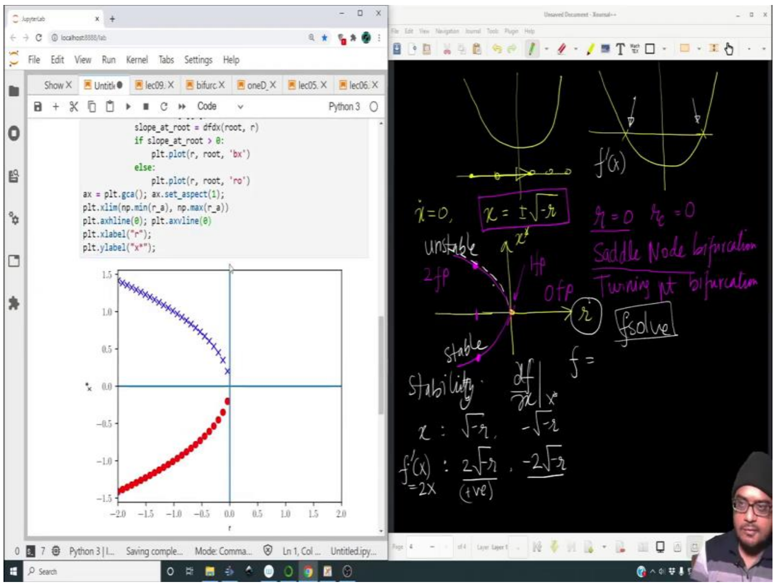
(Refer Slide Time: 29:51)



```
            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.xlabel("r");
plt.ylabel("x*");
```

So, as we are changing the value of r from left to right, we are going from a pair of roots one unstable, one stable, we are reaching the critical value of r at 0 after which we have no more stable, unstable points ok, everything vanishes, there is no fixed points in fact.

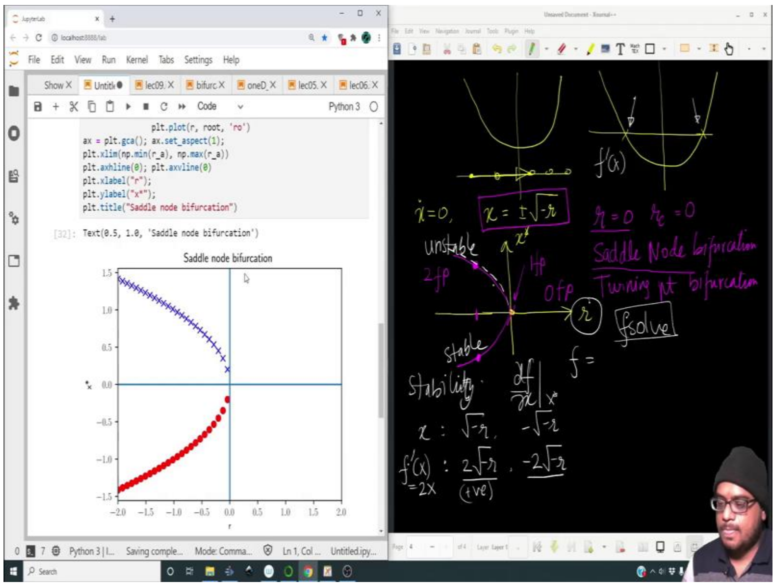
(Refer Slide Time: 30:13)



```
            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*");
```

So, r_c which is 0 it indicates that once the parameter crosses this value of r, we will have no more fixed points.

(Refer Slide Time: 30:32)



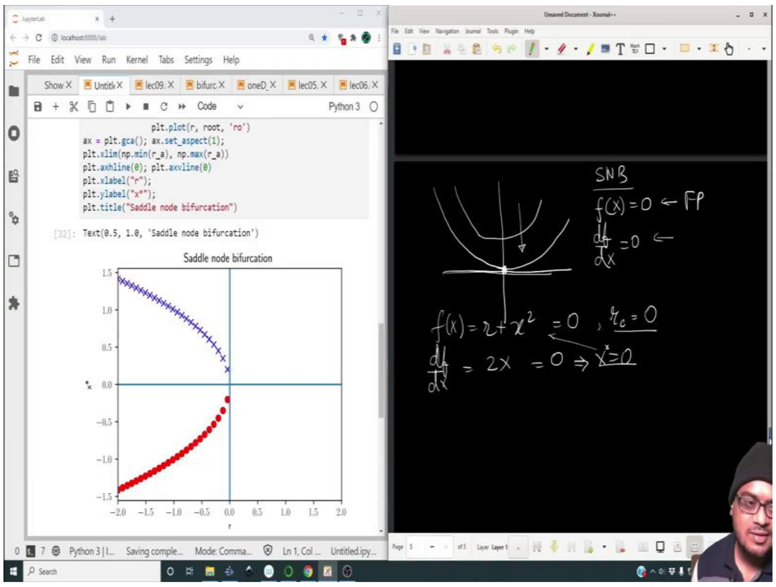
```
                    plt.plot(r, root, 'ro')
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Saddle node bifurcation")
```

```
[32]: Text(0.5, 1.0, 'Saddle node bifurcation')
```

Saddle node bifurcation
$f'(x)$
$\dot{x}=0,\ \boxed{x=\pm\sqrt{-r}}$
unstable
2fp
Hp
0fp
stable
r
$r=0\quad r_c=0$
Saddle Node bifurcation
Turning pt bifurcation
$\boxed{\text{fsolve}}$
$f=$
Stability: $\left.\frac{\partial f}{\partial x}\right|_{x^*}$
$x:\ \sqrt{-r},\ -\sqrt{-r}$
$f'(x)=2x:\ \underset{(+ve)}{2\sqrt{-r}},\ \underline{-2\sqrt{-r}}$

So, this kind of bifurcation is called as a saddle node bifurcation ok. So, this is how we can do it. Now, all this is fine I mean it may appear that this is a very very specific condition I mean why would I expect this kind of a ordinary differential equation to appear. Well, let us delve further into what is going on.

(Refer Slide Time: 31:06)



```
                    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("Saddle node bifurcation")
```

```
[32]: Text(0.5, 1.0, 'Saddle node bifurcation')
```

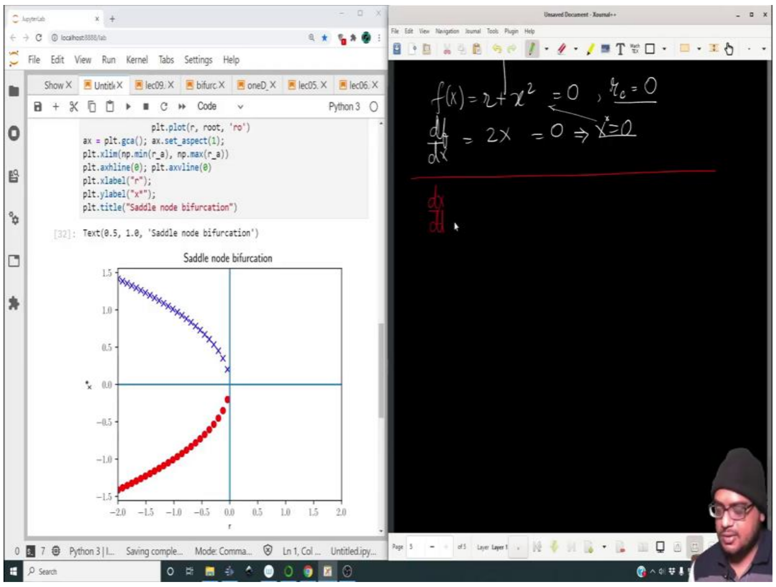
Saddle node bifurcation
SNB
$f(x)=0 \leftarrow$ FP
$\frac{df}{dx}=0 \leftarrow$
$f(x)=r+x^2=0,\ \underline{r_c=0}$
$\frac{df}{dx}=2x=0 \Rightarrow \underline{x^*=0}$

So, first of all, what is happening at the point of bifurcation? You have a function of x and which looks something like this and as you are changing the value of r, it reaches this critical condition where it is just becoming tangential to the x axis. So, for the saddle node bifurcation, we have what? We have f(x) vanishing we also have so, f(x) vanishing is trend amount to that point being a fixed point that is for sure. So, we just have a fixed point which appears and the other condition is it is tangential to the x axis. So, $\left(\frac{\partial f}{\partial x}\right)$ must also be 0 at that crossover condition.

So, what was xf(x)? So, f(x) was = $r+x^2$. So, for what value of r does it happen? So, $\left(\frac{\partial f}{\partial x}\right)$ = 2x. So, this has to be 0 and this has to be 0 at the point we achieve a crossover from two stable roots from two fixed points to no fixed point. So, when this is 0, it implies x = 0 and when we substitute x = 0 over here so, this is the that fixed point which takes part in the bifurcation and we substitute this over here, we obtain r_c that is the critical control parameter to be 0 as well.

So, r = 0 and x = 0 are like the critical pair. So, r_c = 0 is the control parameter critical and it corresponds to x = 0 and this is the place where bifurcation occurs.

(Refer Slide Time: 33:07)



```
                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("Saddle node bifurcation")
```

```
[32]: Text(0.5, 1.0, 'Saddle node bifurcation')
```

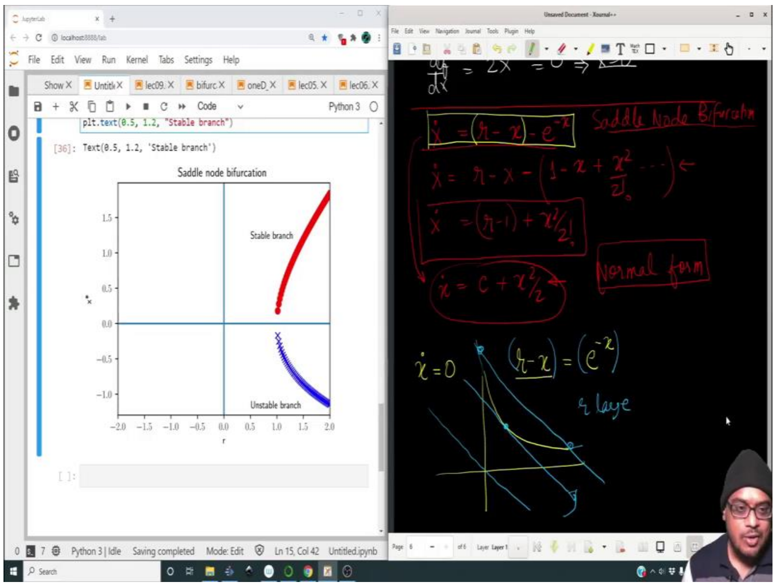
$$f(x) = r + x^2 = 0\ , \ \underline{r_c = 0}$$

$$\frac{df}{dx} = 2x = 0 \Rightarrow \underline{x^* = 0}$$

$$\frac{dx}{dt}$$

So, now, it is not just this equation that exhibits such a behavior. Let us look at an example. Let us look at an example such as $\frac{dx}{dt}$.

(Refer Slide Time: 33:11)

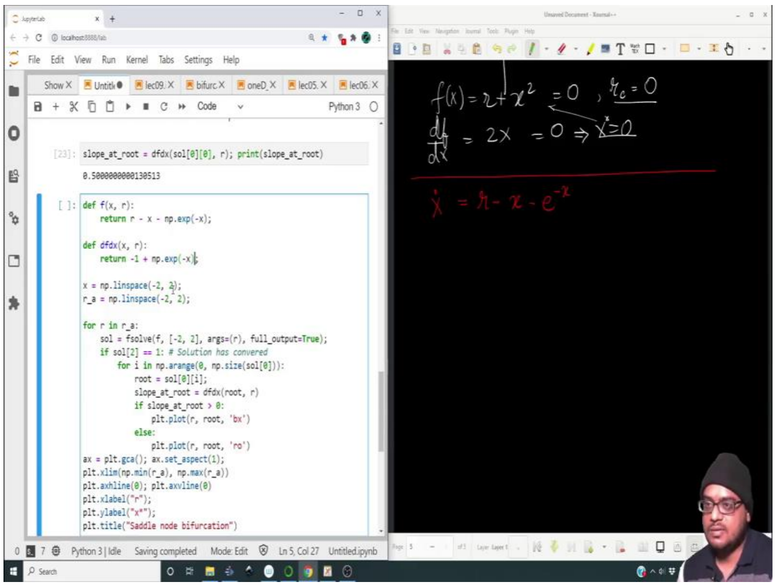


```
plt.text(0.5, 1.2, "Stable branch")
```

```
[36]: Text(0.5, 1.2, 'Stable branch')
```

Or in fact, let me write it as $\dot{x}$ is =r -x - e $^{-x}$ ok. Let us try to see what the bifurcation diagram for this looks like. So, because we have already written the code, we can simply modify our function and the derivative for having this kind of an output.

(Refer Slide Time: 33:48)



```
[23]: slope_at_root = dfdx(sol[0][0], r); print(slope_at_root)

      0.5000000000130513
```

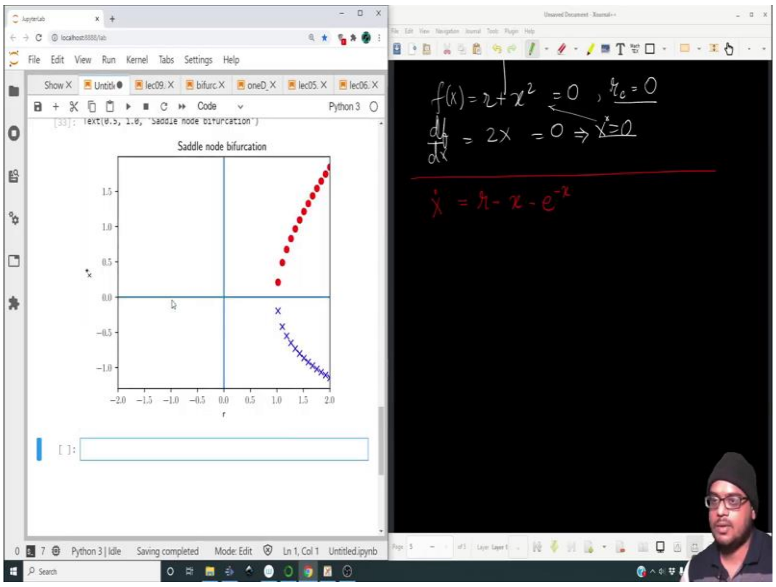
```
[ ]: def f(x, r):
         return r - x - np.exp(-x);

     def dfdx(x, r):
         return -1 + np.exp(-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("Saddle node bifurcation")
```

$f(x) = r + x^2 = 0$, $r_c = 0$
$\frac{df}{dx} = 2x = 0 \Rightarrow x^* = 0$
$\dot{x} = r - x - e^{-x}$

So, let me copy this entire cell, let me make a new cell this will be r - x - np.exp(-x) and the derivative will be -1 + np.exp(-x)

.

(Refer Slide Time: 34:13)



```
[33]: Text(0.5, 1.0, 'Saddle node bifurcation')
```

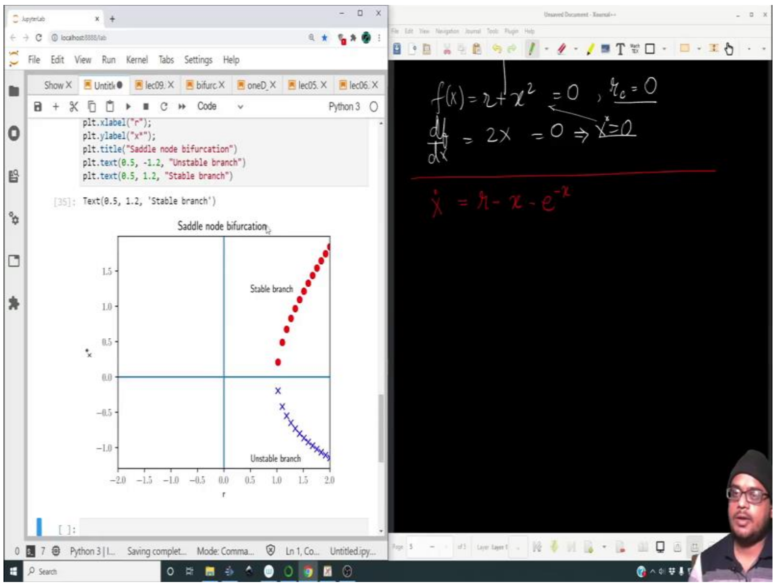
$f(x) = r + x^2 = 0$, $r_c = 0$

$\frac{df}{dx} = 2x = 0 \Rightarrow x^* = 0$

$\dot{x} = r - x - e^{-x}$

So, let me run this and see what happens. Well, we do have a bifurcation what happens over here, what do we see. So, as r varies from -2 to 2 at a certain value; at a certain value so, there are no roots so far, but after a certain point, there are two roots one is a stable root, the upper branch is a stable branch, the lower branch is an unstable branch ok.

(Refer Slide Time: 34:48)



```
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Saddle node bifurcation")
plt.text(0.5, -1.2, "Unstable branch")
plt.text(0.5, 1.2, "Stable branch")

[35]: Text(0.5, 1.2, 'Stable branch')
```

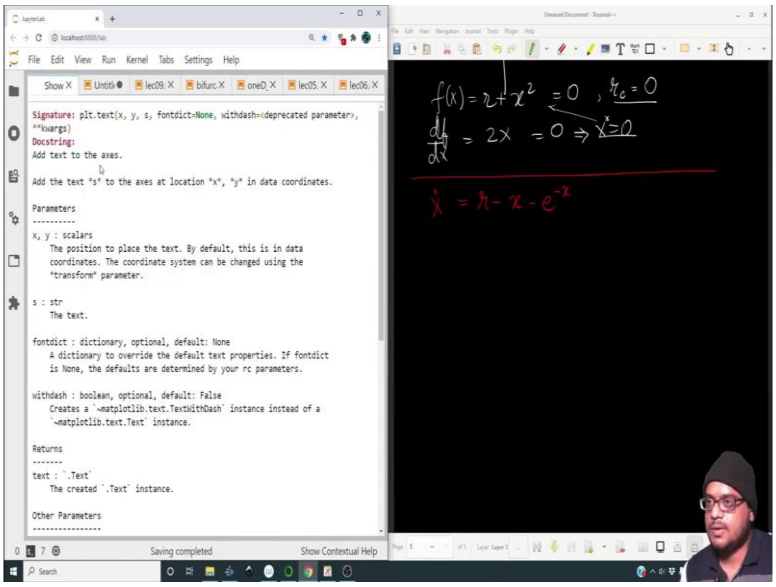
$f(x) = r + x^2 = 0$, $r_c = 0$

$\frac{df}{dx} = 2x = 0 \Rightarrow x^* = 0$

$\dot{x} = r - x - e^{-x}$

Let us put those annotations as well.

(Refer Slide Time: 34:52)



```
Signature: plt.text(x, y, s, fontdict=None, withdash=<deprecated parameter>,
**kwargs)
Docstring:
Add text to the axes.

Add the text *s* to the axes at location *x*, *y* in data coordinates.

Parameters
----------
x, y : scalars
    The position to place the text. By default, this is in data
    coordinates. The coordinate system can be changed using the
    *transform* parameter.

s : str
    The text.

fontdict : dictionary, optional, default: None
    A dictionary to override the default text properties. If fontdict
    is None, the defaults are determined by your rc parameters.

withdash : boolean, optional, default: False
    Creates a `~matplotlib.text.TextWithDash` instance instead of a
    `~matplotlib.text.Text` instance.

Returns
-------
text : `.Text`
    The created `.Text` instance.

Other Parameters
----------------
```

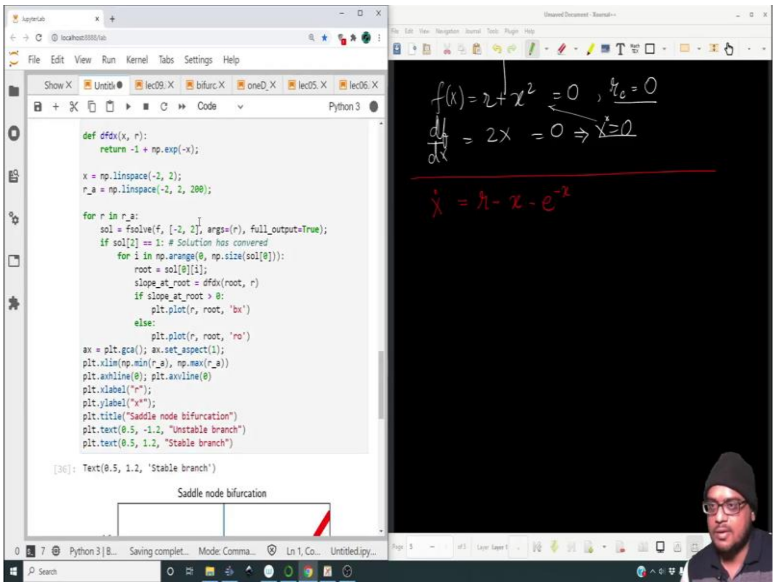
$$f(x) = r + x^2 = 0 \quad , \quad \underline{r_c = 0}$$

$$\frac{df}{dx} = 2x = 0 \Rightarrow \underline{x^* = 0}$$

$$\dot{x} = r - x - e^{-x}$$

Let us have a look at the function. So, x, y, string ok.

(Refer Slide Time: 35:38)



```
def dfdx(x, r):
    return -1 + np.exp(-x);

x = np.linspace(-2, 2);
r_a = np.linspace(-2, 2, 200);

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("Saddle node bifurcation")
plt.text(0.5, -1.2, "Unstable branch")
plt.text(0.5, 1.2, "Stable branch")
```

```
[36]: Text(0.5, 1.2, 'Stable branch')
```

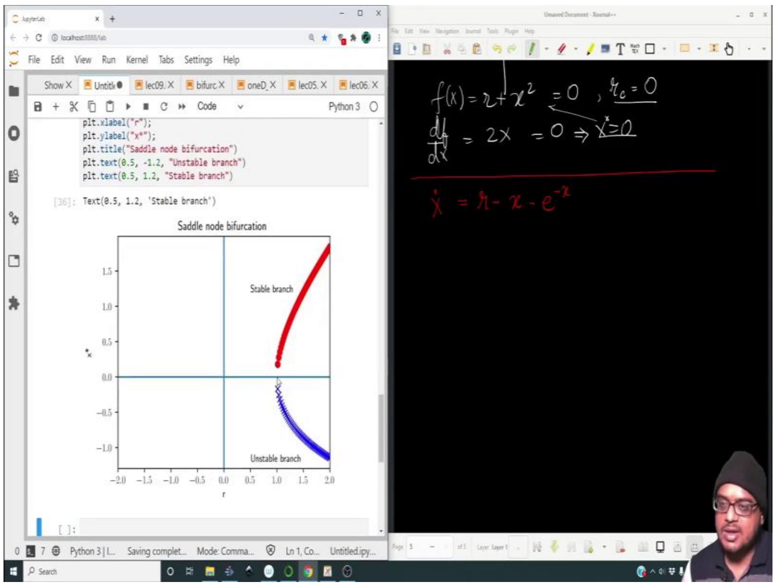
$$f(x) = r + x^2 = 0 \quad , \quad \underline{r_c = 0}$$

$$\frac{df}{dx} = 2x = 0 \Rightarrow \underline{x^* = 0}$$

$$\dot{x} = r - x - e^{-x}$$

So, I need to pass the x y over here. So, let us put the string at 0.5 and -1.2 ok. So, it is the unstable branch and let me put another string on top which will say it is a stable branch alright. So, the stable branch and unstable branch are these and there is a critical point somewhere over here. I mean we could obviously, increase the resolution of the r. So, let us increase the resolution of r_a, let me take 200 points to pinpoint where that happens ok.

(Refer Slide Time: 35:41)



```
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Saddle node bifurcation")
plt.text(0.5, -1.2, "Unstable branch")
plt.text(0.5, 1.2, "Stable branch")
```

```
[36]: Text(0.5, 1.2, 'Stable branch')
```

$f(x) = r + x^2 = 0$, $\underline{r_c = 0}$

$\frac{df}{dx} = 2x = 0 \Rightarrow \underline{x^* = 0}$

$\dot{x} = r - x - e^{-x}$

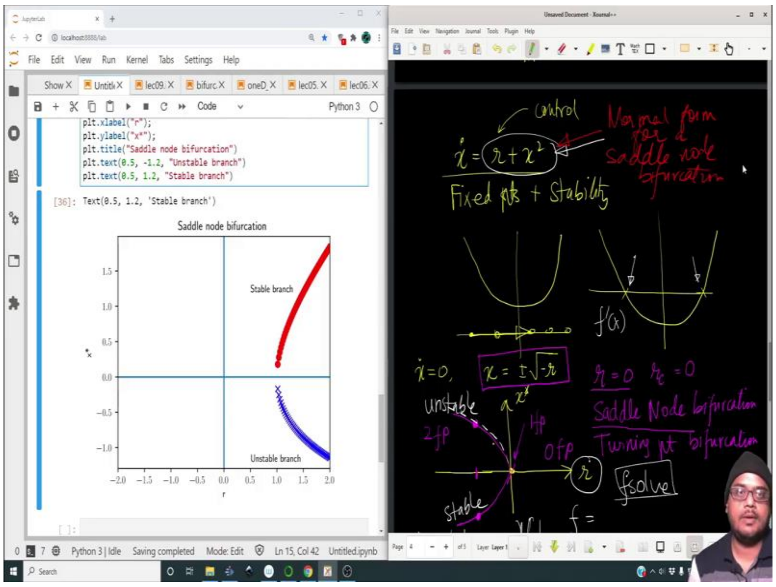
Actually, it is very difficult to pinpoint by just looking. So, you what you should do is find out the first occurrence of r where this condition fails ok; where this condition fails. When this does not return 1, it means that it is not having two solutions, but the moment it starts returning 1, you have two solution or you have roots not just two solutions when you have roots. I am not going to do it right now, but you can try it when you have time.

So, it is clear that even this equation does have a saddle node bifurcation behavior and why is that? Why does this equation also have a saddle node bifurcation behavior? Let us look deeply into how this equation reacts in the vicinity of the fixed point. So, let us expand let us expand this around x = 0. So, $\dot{x} = r - x - \exp{-x}$ so, this becomes what? $1 - x + x^2/2!$ and so on. So, this becomes r - 1 this x cancels out + $x^2/$ 2 factorial.

So, obviously, the first root that will occur over here is x = 0. So, this expansion of the function that we are doing is near the bifurcation point. So, near the bifurcation point, the equation behaves like this and it has the same form. So, $\dot{x} = c + x^2/2$. So, it does have the same form so obviously, near the bifurcation boundary, it has the same prototypical behavior as this particular equation alright.

So, that is why studying such prototypical problems is quite important and such a reduction of a problem in the vicinity of the bifurcation point is called as a normal mode or a normal form not a normal mode. The normal form for a saddle node bifurcation is this is this.

(Refer Slide Time: 38:24)



```
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Saddle node bifurcation")
plt.text(0.5, -1.2, "Unstable branch")
plt.text(0.5, 1.2, "Stable branch")
```

```
[36]: Text(0.5, 1.2, 'Stable branch')
```

$\dot{x} = \boxed{r + x^2}$

$\dot{x} = 0, \quad \boxed{x = \pm\sqrt{-r}}$

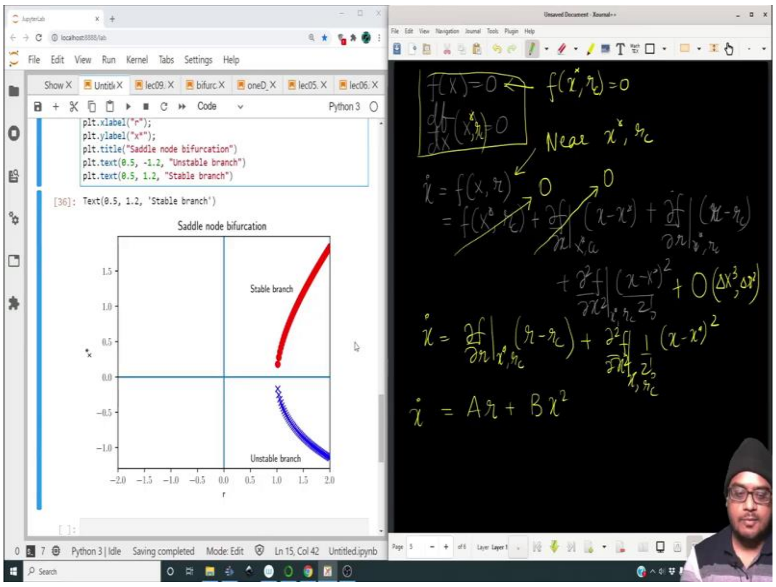
$r = 0 \quad r_c = 0$

Saddle Node bifurcation

Turning pt bifurcation

So, this is a normal form for a saddle node bifurcation. So, obviously, there are different normal forms for different kinds of bifurcation, but for now, we have only studied the saddle node bifurcation.

(Refer Slide Time: 39:10)



```
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Saddle node bifurcation")
plt.text(0.5, -1.2, "Unstable branch")
plt.text(0.5, 1.2, "Stable branch")
```

```
[36]: Text(0.5, 1.2, 'Stable branch')
```

$$\boxed{\begin{array}{l} f(x) = 0 \\ \frac{df}{dx}(x^*, r_c) = 0 \end{array}} \leftarrow f(x^*, r_c) = 0$$

Near x^*, r_c

$$\dot{x} = f(x, r) = \cancel{f(x^*, r_c)} + \cancel{\frac{\partial f}{\partial x}\Big|_{x^*, r_c}}(x - x^*) + \frac{\partial f}{\partial r}\Big|_{x^*, r_c}(r - r_c) + \frac{\partial^2 f}{\partial x^2}\Big|_{x^*, r_c}\frac{(x - x^*)^2}{2!} + O(\Delta x^3, \Delta r^2)$$

$$\dot{x} = \frac{\partial f}{\partial r}\Big|_{x^*, r_c}(r - r_c) + \frac{\partial^2 f}{\partial x^2}\Big|_{x^*, r_c}\frac{1}{2!}(x - x^*)^2$$

$$\dot{x} = Ar + Bx^2$$

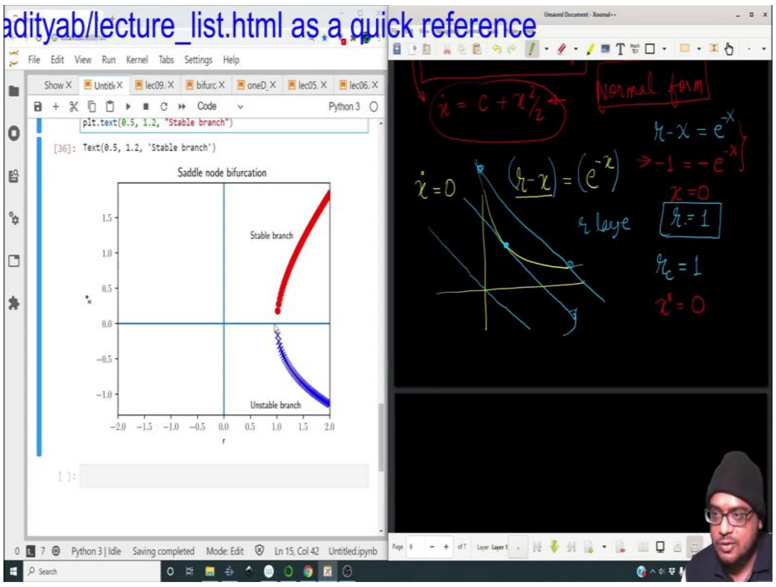
Can we prove this rigorously? Can we prove that near the bifurcation point? It will always reduce to a normal form and the answer is yes. Let us take a quick look into that. So, what are the conditions when the bifurcation occurs? So, the conditions are that f(x) has to be 0 and $\left(\frac{\partial f}{\partial x}\right)_x$ also has to be 0. So, let us write down $\dot{x} = f(x)$ and it is a parameter of r as well. So, it is a function of x and r. So, this can be written as $\dot{x} = f(x^*, r_c) + \left(\frac{\partial f}{\partial x}\right)_{x^*, r_c} (x - x^*) + \left(\frac{\partial f}{\partial r}\right)_{x^*, r_c} (r - r_c) + \left(\frac{\partial^2 f}{\partial x^2}\right)_{x^*, r_c} \frac{1}{2!}(x - x^*)^2 + \cdots\cdots$.

Now, by very definition of being a bifurcation point, f (x) will be 0 essentially, f(x, r) will be 0. So, $f(x^*, r_c)$ will be 0 because that is the bifurcation point and hence, this term will vanish.

Similarly, $\left(\frac{\partial f}{\partial x}\right)_{x^*, r_c} (x - x^*)$ will also be 0 and hence, this particular term also vanishes obviously, there are higher order terms, but because we are studying the behavior of the system near x star and r c, this is what we can write and essentially, x dot eventually boils down to $\dot{x} = \left(\frac{\partial f}{\partial r}\right)_{x^*, r_c} (r - r_c) + \left(\frac{\partial^2 f}{\partial x^2}\right)_{x^*, r_c} \frac{1}{2!}(x - x^*)^2$.

And quite obviously, you can obviously, cast this as $Ar + Bx^2$ where A and B are various constants which contain all those various terms. So, this has to be at (x^*, r_c) and hence, these such equations under the conditions of a saddle node bifurcation do reduce to that kind of a normal form ok. So, this is what I wanted to discuss about saddle node bifurcations.

(Refer Slide Time: 43:52).


```
plt.text(0.5, 1.2, "Stable branch")
```
```
[36]: Text(0.5, 1.2, 'Stable branch')
```

Before ending this lecture, let me also show you how you can geometrically think about this particular equation sorry this where is it, where you can geometrically think about this particular equation. So, how do you geometrically find fixed points for this? Obviously, I mean you can plot it and see, but it is comprised of two functions, (r-x) and e^{-x}.

So, at the fixed-point $\dot{x}$ will be 0 what that means, is (r-x) = e^{-x}. The curve for e^{-x} looks something like this and (r-x) is a straight line and depending on what the value of r will be we will have a family of straight lines which will look something like this. So, now, we all we have to do is probe whether or not (r-x) cuts e^{-x} at one point, at no points, at two points and so on.

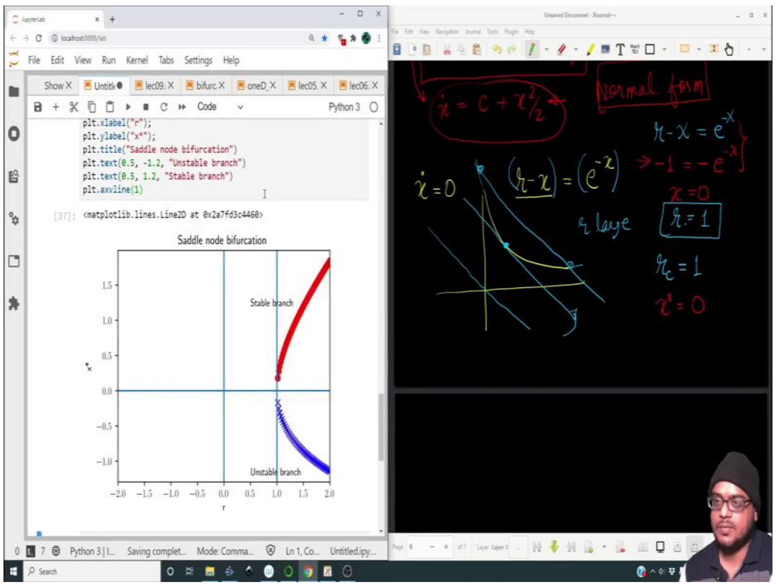
So, when r has a large value, it cuts the curve at two points. So, when r is large, obviously, there are two roots, and this is something which we can corroborate with this diagram over here.

When r is 0, this is what it will look like and obviously, there is no root. In fact, for quite some time, there will be no root until it reaches this tangentiality condition. So, when (r-x) becomes tangential to e^{-x}, we will have only one root and that is the condition for bifurcation, it is going from no roots to two roots through that particular tangency condition.

So, at the point of bifurcation; at the point of bifurcation, we will have two things; one is the blue line intersecting with the yellow curve that is (r-x) = e^{-x} that is one condition and the other condition is the slope of (r-x) = the slope of (r-x) ok. So, the slope of (r-x) is -1, the slope of e^{-x} is - e^{-x}.

So, what are the conditions under which this is satisfied? So, this is satisfied when x = 0, this equation is satisfied when x = 0 and when we substitute x = 0 over here, we have r = 1. So, r = 1 is the critical control parameter where bifurcation occurs and the fixed point which corresponds to this particular bifurcation is = 0. So, over here x = 0 so, the y axis is x and r = 1 is the point where bifurcation occurs. So, we can go ahead and plot that line as well.

(Refer Slide Time: 45:01)


```
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Saddle node bifurcation")
plt.text(0.5, -1.2, "Unstable branch")
plt.text(0.5, 1.2, "Stable branch")
plt.axvline(1)
```

```
[37]: <matplotlib.lines.Line2D at 0x2a7fd3c4460>
```

So, plt dot avline axvline and we have to plot it through 1. So, that is the zone of bifurcation and it is obviously, separating two roots; one is a stable branch, and one is a unstable branch. You can obviously, write a nice slider function to check the presence of roots, I am not going to do that you can try it out on your own. You know how to do it by now ok.

So, this is all I wanted to discuss about saddle node bifurcations. In the next lecture, we are going to look into the other kinds of bifurcation namely transcritical bifurcation and pitchfork bifurcation.

Until then, its goodbye from me. Try to experiment as much as you can. Experience is your best friend, bye.