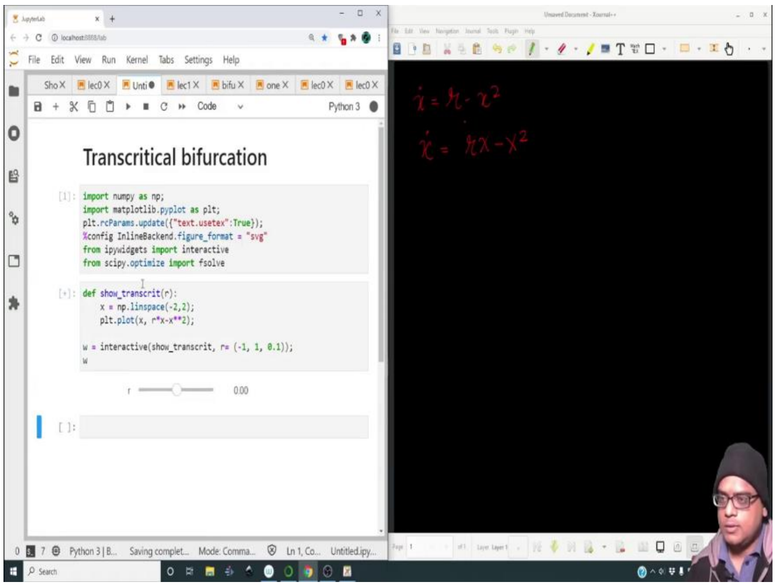
Tools in Scientific Computing

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

Lecture – 11
Bifurcations – Transcritical Bifurcation

(Refer Slide Time: 00:27)



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
```

```
[*]: def show_transcrit(r):
         x = np.linspace(-2,2);
         plt.plot(x, r*x-x**2);

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

r 0.00

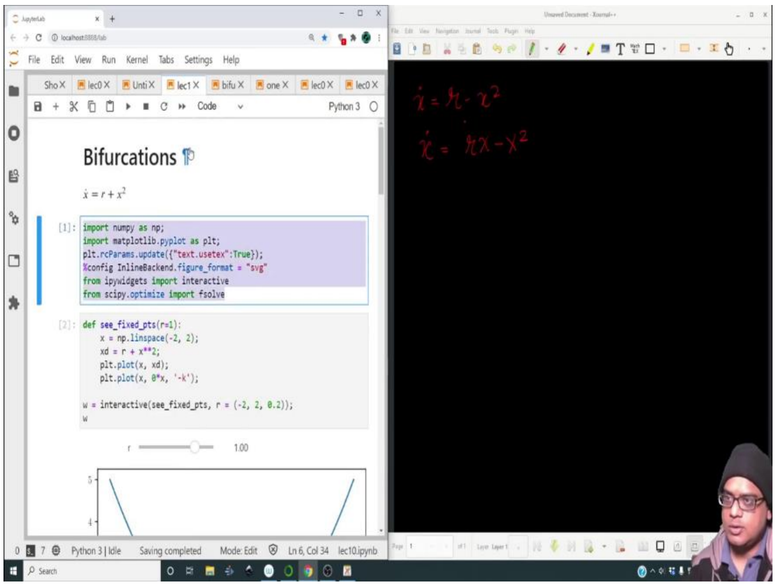
$\dot{x} = r - x^2$

$\dot{x} = rx - x^2$

Hey guys, it is me again. In this lecture, we are going to look at Transcritical Bifurcations. In the previous lecture, we had looked at saddle node bifurcations. So, we will in this lecture, we will start off with the transcritical bifurcation discussion and then we will move on to pitchfork bifurcation. So, in the previous lecture, we have already looked at the normal form of a saddle node bifurcation.

The normal form for a trans; so, saddle node bifurcation was of the form $r - x^2 = \dot{x}$. So, a transcritical bifurcation is quantified or rather is characterized by a form $\dot{x} = xr - x^2$. Let us have a look at how the plot looks like, but before that let me copy the usual snippet ok.

(Refer Slide Time: 01:25)



Bifurcations

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

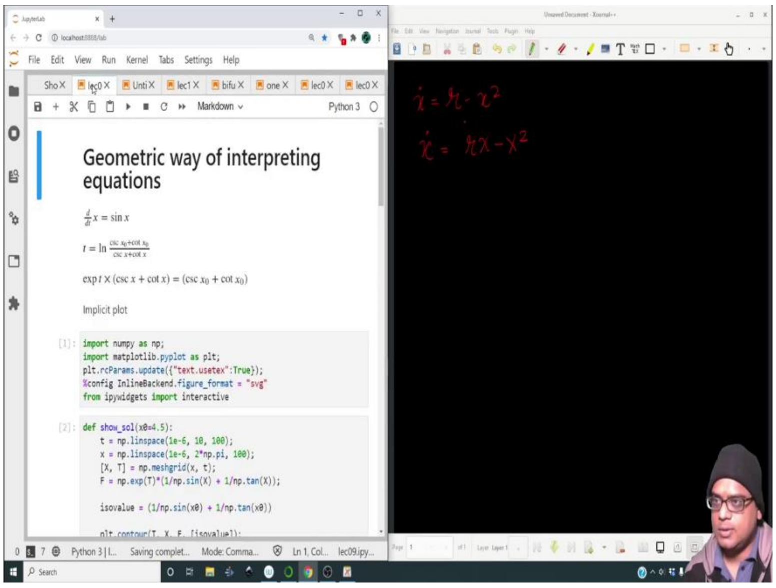
```
[1]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive
     from scipy.optimize import fsolve
```

```
[2]: 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
```

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

$\dot{x} = rx - x^2$

(Refer Slide Time: 01:30)



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
```

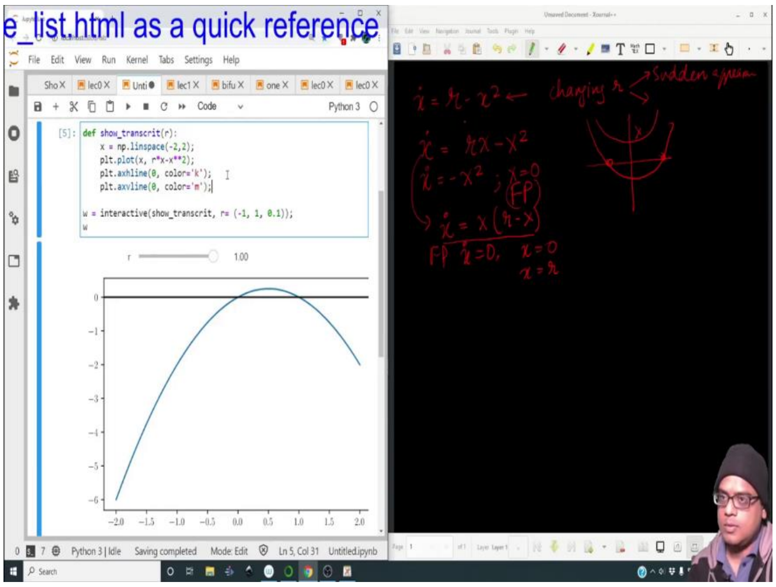
```
[2]: def show_sol(x0=4.5):
         t = np.linspace(1e-6, 10, 100);
         x = np.linspace(1e-6, 2*np.pi, 100);
         [X, T] = np.meshgrid(x, t);
         F = np.exp(T)*(1/np.sin(X) + 1/np.tan(X));

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

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

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

$\dot{x} = rx - x^2$

So, let x = np.linspace(-2,2). Let us define a function show transcrit. Let r be the input to the function and we will do a plot(x,$rx - x^2$) and let us call the widget. So, w =interactive (show_transcrit,r=(-1,1,0.1)) alright and lastly we have to show the widget as well.

(Refer Slide Time: 02:30)



```
[5]: 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
```

e_list.html as a quick reference
r 1.00
$\dot{x} = r - x^2 \leftarrow$ changing r → sudden appearance
$\dot{x} = rx - x^2$
$\dot{x} = -x^2$; $x=0$ FP
$\dot{x} = x(r-x)$
FP $\dot{x}=0$, $x=0$, $x=r$

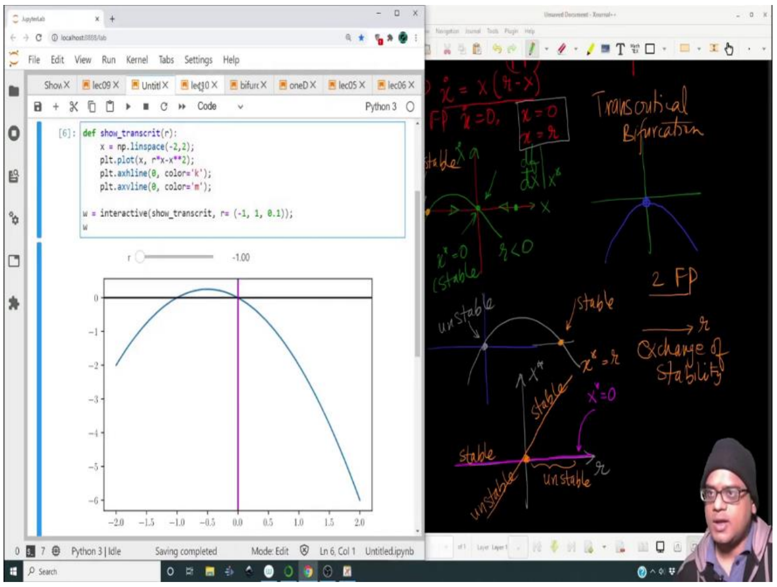
So, this is the plot, let us additionally plot the x axis. So, plt.axhline(0). So, this plots horizontal line; let us set the color of the line to black alright. So, when r= 0; obviously, this particular equation boils down to $\dot{x} = -x^2$ which is a parabola facing downwards and there is exactly one root that is this x = 0 is the fixed point.

Again that is a half stable point whereas, when r becomes negative we have the occurrence of two roots. And when r becomes positive, we again have occurrence of two roots.

So, unlike the saddle node bifurcation where changing r, it lead to two conditions; it lead to a sudden appearance of roots ok. So, essentially we had a parabola which did not intersect and when we changed the parameter, we had an intersection. So, no roots; no fixed points fixed points.

But in this case there seems to be a common root and; obviously, the common root is x = 0. So, we can write this as $\dot{x} = xr - x^2$ and the roots of this so, the fixed points are positions where $\dot{x} = 0$ it is 0 at x=0 and x = r. So; obviously, when r is negative, we have the other root appearing at negative values.

(Refer Slide Time: 04:44)



```
[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
```

Let me also plot the vertical line ok. So, that is the vertical line. So, you see one root is always 0 while the other root simply shifts around ok. So, what is actually happening? We have basically these conditions one is this. So, in this particular case, what is the stability of this point? So, we have seen in the previous lecture that the stability of a fixed point depends on the slope of the function at that fixed point; slope over here is negative. So, this point is stable.

So, the flow of points on this line will be towards this point and towards this point. And it is obvious if you do the whole reasoning with an initial condition some where and how the point travels on the line.

So, $\dot{x}$ so, this is $\dot{x}$ so, and this is x. So, if the initial condition are over here, the particle has a negative velocity and it moves towards this point. Over here the particle has a positive velocity and it move towards the right ok. So, 0 or rather the fixed point $x^* = 0$ is stable. So, this is the condition for r < 0.

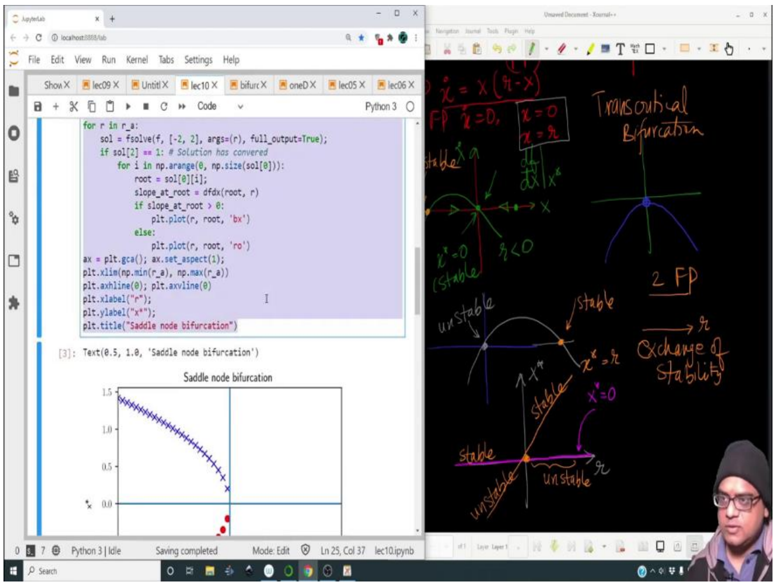
What about the condition for r = 0 ok? So, at r = 0, the slope is equal to 0 and this is a half stable point. What about r > 0? So, this is the fixed point, but the slope over here is positive and that implies at this particular fixed point that is $x^* = 0$, it becomes unstable. So, if we draw this particular curve; if we draw r on this axis and x^* on this axis so, we know already the roots are x= 0 and x =r ok.

So, let me mark the x = zeroroot with magenta line. So, this particular line are the fixed shows the fixed points $x^* = 0$ while this particular line is $x^* = r$. So, in the plot x^* versus r this will be the line, but when r < 0; we see that the origin rather the $x^* = 0$ line is stable. So, this is stable, but once it crosses r = 0; this branch becomes unstable.

Now, similar reasoning we can see that when x^* is stable $x^* = 0$ is stable this other root that is $x^* = r$ is unstable because the slope over here is positive. So, this is unstable whereas, over here the other root $x^* = r$ it becomes stable because the slope over here is negative. So, this is unstable and this become stable. So, essentially we have two roots or two fixed points and the changing parameter r leads to an exchange of stabilities of the two roots.

So, in a physical problem; if we have the condition where the changing of a parameter leads to that same fixed point, it still has the same fixed point ok. We are not changing the number of fixed points, it still has the same fixed point, but the nature of the fixed point changes as we cross such a kind of bifurcation is called as transcritical bifurcation alright. So, let us make use of the code that we had in the previous lecture.

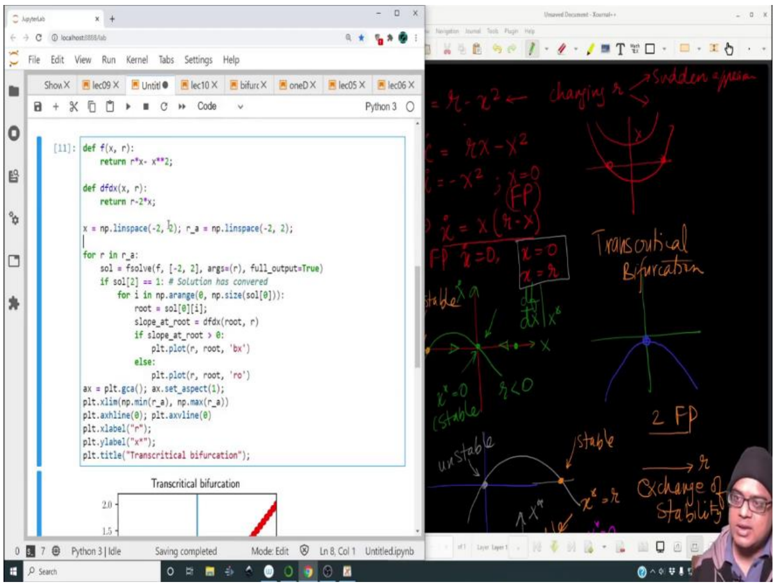
(Refer Slide Time: 09:41)



```
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")
```

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

(Refer Slide Time: 09:47)

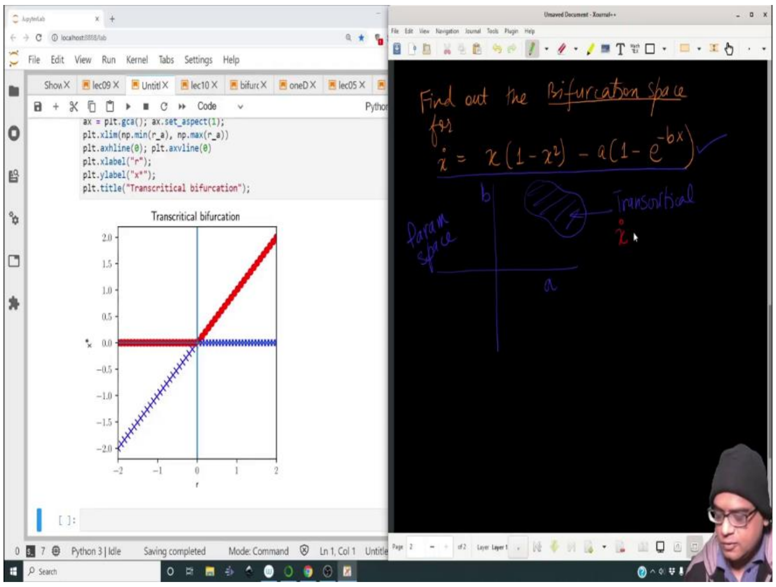


```
[11]: def f(x, r):
          return r*x- x**2;

      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");
```

$= r - x^2 \leftarrow$ changing r → sudden appearance
$\dot{x} = rx - x^2$
$\dot{x} = -x^2$; $x = 0$ (FP)
$\dot{x} = x(r - x)$
FP $\dot{x} = 0$, $x = 0$, $x = r$
$\frac{df}{dx}\Big|_{x^*}$
Transcritical Bifurcation
stable
$x^* = 0$ (stable
$r < 0$
2 FP
unstable
stable
$x^* = r$
Exchange of Stability
Transcritical bifurcation

We can make use of this; we can reuse this code that we had written in the previous lecture. And instead of the saddle node normal form, we will now use the transcritical normal form that is $rx - x^2$ and the slope will be $r - 2x$ alright; everything else remains the same and the title of the plot will be transcritical bifurcation. So, let us run this program and see what happens ok.

(Refer Slide Time: 10:24)



```
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
Find out the Bifurcation space for
$\dot{x} = x(1 - x^2) - a(1 - e^{-bx})$
Param space
b
a
Transcritical
$\dot{x}$

So, we do see; so, see the field markers were stable points while the cross markers were the unstable points let me suppress this. So, one little thing I want to just get out of the way; in python you do not really need to put semicolons at the end of an expression or a statement unlike octave where you want to suppress output you have to put a semicolon. But in doing stuff on Jupiter lab; if I do not suppress this, I will get that annoying text.

So, I want to suppress that. So, I put a semicolon and it is an old habit I have from using octave or MATLAB ok. It makes no difference if I put a semicolon over here or not ok. I can remove all the semicolons everything still is the same. But semicolons do help if I want to put multiple expressions on the same line ok. If I can put multiple expressions over here on the same line and I can separate them by semicolon ok. It is, but if I remove this, then there will be an error.

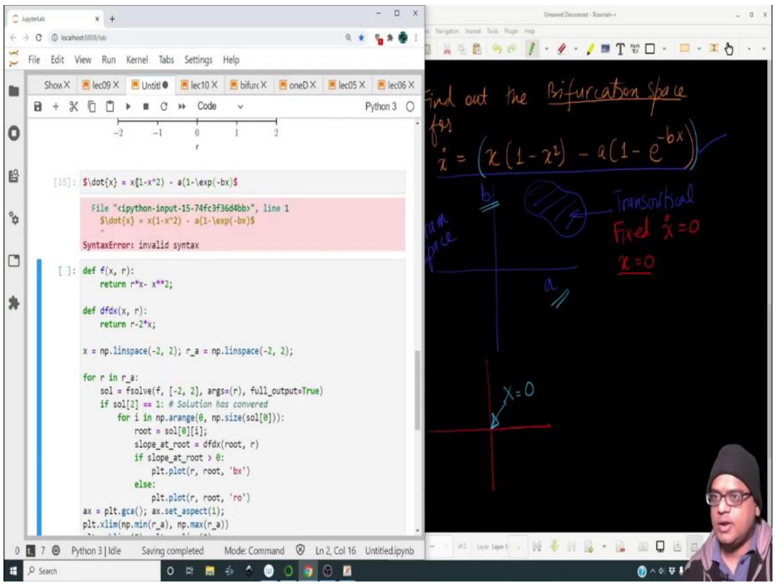
I tries to interpret the entire line as a single executable statement that something I just wanted to get out of the game. So, over here, we had a stable branch which became unstable; we had a unstable branch which became stable. So, this is an example of a transcritical bifurcation.

So, now let us go to a very simple problem and that problem is to find out the bifurcation diagram or the bifurcation space for $\dot{x} = \mathrm{x}(1 - x^2) - \mathrm{a}(1 - e^{-bx})$. So, this is the question.

So, by specifying the bifurcation space, it means there is a certain combination of a and b. If I plot b on this axis and a on this axis so, there is a certain combination of a and b for which this particular vector flow ok. This is also called as a vector flow, it will show transcritical bifurcation, but whether or not it is a region in the (a,b) space. So, we are not working in the (x,x`) space anymore, we are trying to analyze this equation in the abstract space (a,b). So, its a parameter space.

So, in the (a,b) space what is the zone or curve or whatever it is which causes this equation to exhibit transcritical bifurcation.

(Refer Slide Time: 13:37)



```
[15]: $\dot{x} = x(1-x^2) - a(1-\exp(-bx)$

  File "<ipython-input-15-74fc3f36d4bb>", line 1
    $\dot{x} = x(1-x^2) - a(1-\exp(-bx)$
    ^
SyntaxError: invalid syntax
```

```
[ ]: 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))
```

Find out the Bifurcation space for
$\dot{x} = (x(1-x^2) - a(1-e^{-bx}))$
b
Param space
Transcritical
Fixed $\dot{x}=0$
$x=0$
a
$x=0$

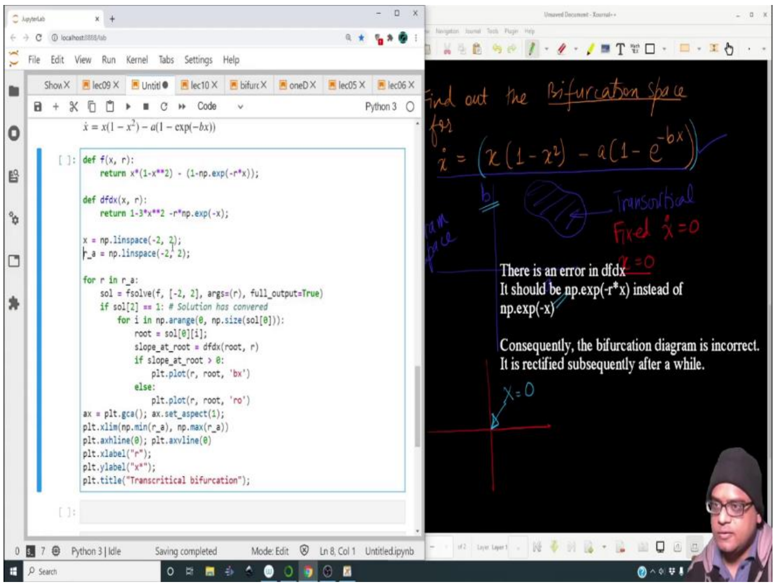
So, straight off the bat we can see that x`. So, the fixed point of this particular equation fixed point where $\dot{x} = 0$ corresponds to x=0. If you substitute x=0, you see that this entire thing becomes 0 and this becomes1. So, 1 -1 is 0.

So, x = 0 is a fixed point. Now, what do we want to know is whether it exhibits transcritical bifurcation. So, when does mathematically how can we say whether a certain point say this fixed point x =0 whether that point has some transcritical bifurcation going on?

So, over here we had x = 0 and we could say and there is a transcritical bifurcation because in changing the parameter there was a change in the slope of this entire curve at this point. So, if we can find out the parameter space a and b for which this particular fixed point changes slope, then we can say that yes that equation does show a transcritical bifurcation. But even before that we can sort of fix a and vary b to see what goes on ok.

Let me copy this, let me paste this, let me give this cell some context. So, this is $\dot{x} =$ x$(1 - x^2) -$ a$(1 - e^{-bx})$. So, we are trying to sorry this has to be interpreted as markdown.

(Refer Slide Time: 15:39)



$\dot{x} = x(1 - x^2) - a(1 - \exp(-bx))$

```
[ ]: def f(x, r):
         return x*(1-x**2) - (1-np.exp(-r*x));

     def dfdx(x, r):
         return 1-3*x**2 -r*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("Transcritical bifurcation");
```

There is an error in dfdx
It should be np.exp(-r*x) instead of np.exp(-x)

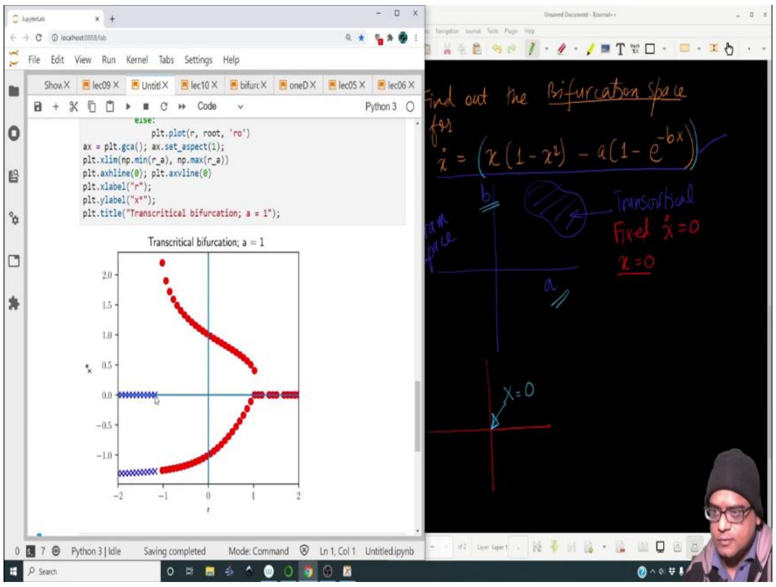
Consequently, the bifurcation diagram is incorrect.
It is rectified subsequently after a while.

Find out the Bifurcation space for
$\dot{x} = (x(1-x^2) - a(1-e^{-bx}))$
b
Transcritical
Fixed $\dot{x}=0$
$x=0$
$x=0$

So, we are trying to solve this equation rather trying to find out the properties of this equation if you like. So, let me change this. So, this has to be x$(1 - x^2) -$ a$(1 - e^{-bx})$. So, now let me fix a. So, let a = 1. So, let b be represented by r because we are already passing r as an input to the function. So, let b be representative of r rather r be representative of b.

So, the derivative over here will be $1 - 3x^2$ and there will be so, the derivative will be $-re^{-x}$ ok. So, r is varying from - 2 to 2 that is fine ok.

(Refer Slide Time: 17:01)

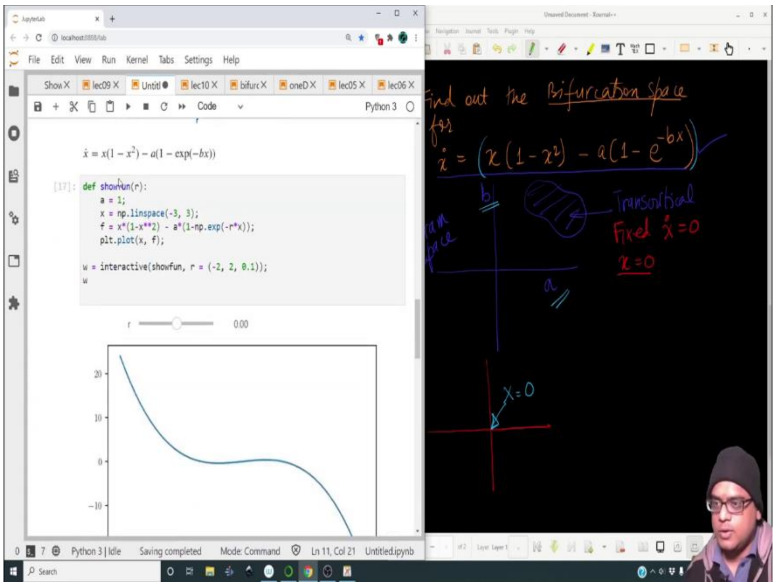


```
                 else:
                     plt.plot(r, root, 'ro')
     ax = plt.gca(); ax.set_aspect(1);
     plt.xlim(np.min(r_a), np.max(r_a))
     plt.axhline(0); plt.axvline(0)
     plt.xlabel("r");
     plt.ylabel("x*");
     plt.title("Transcritical bifurcation; a = 1");
```

Transcritical bifurcation; a = 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2
r
x*
Find out the Bifurcation space for
$\dot{x} = (x(1-x^2) - a(1-e^{-bx}))$
b
Transcritical
Fixed $\dot{x}=0$
$x=0$
a
$x=0$

So, essentially this will be the plot where a = 1 alright. So, let us run this cell and see what happens so; obviously, we are ok. So, we are missing a few roots over here and the reason is. So, the reason is clear; we are not passing enough guess points. So, why is it important to pass enough guess points? Let me show you.

(Refer Slide Time: 17:43)



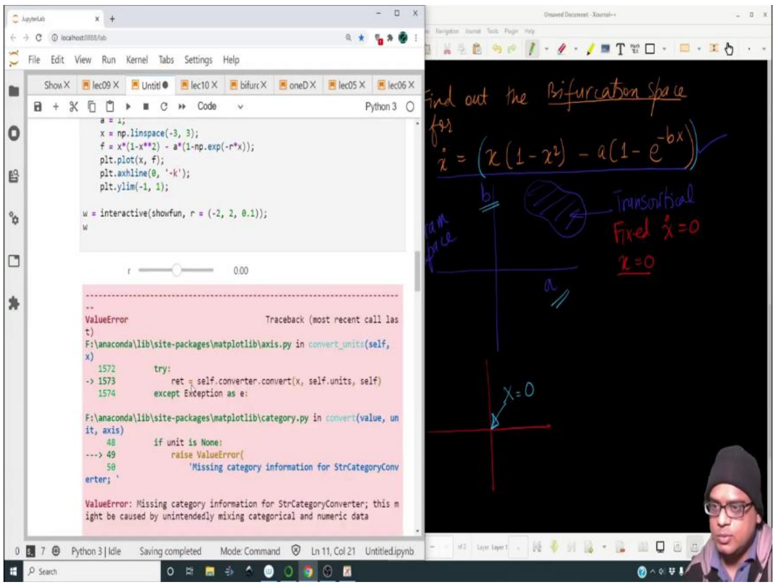
$\dot{x} = x(1 - x^2) - a(1 - \exp(-bx))$

```
[17]: def showfun(r):
          a = 1;
          x = np.linspace(-3, 3);
          f = x*(1-x**2) - a*(1-np.exp(-r*x));
          plt.plot(x, f);

      w = interactive(showfun, r = (-2, 2, 0.1));
      w
```

File Edit View Run Kernel Tabs Settings Help
r 0.00
20
10
0
−10
Find out the Bifurcation space for
$\dot{x} = (x(1-x^2) - a(1-e^{-bx}))$
b
a
ram space
Transcritical
Fixed $\dot{x}=0$
$x=0$
$x=0$
Python 3 | Idle Saving completed Mode: Command Ln 11, Col 21 Untitled.ipynb

Let me first plot this. So, let x = np.linspace(-3, 3), let a =1 and let us pass everything into a function. So, def show fun and let the input be r. Let us put all of this inside the function and let f =x (1 – x^2) -a(1 -np.exp(-rx)). Plt.plot(x ,f). Let us put it in a interactive widget alright. We have to show the widget also.

(Refer Slide Time: 18:48)

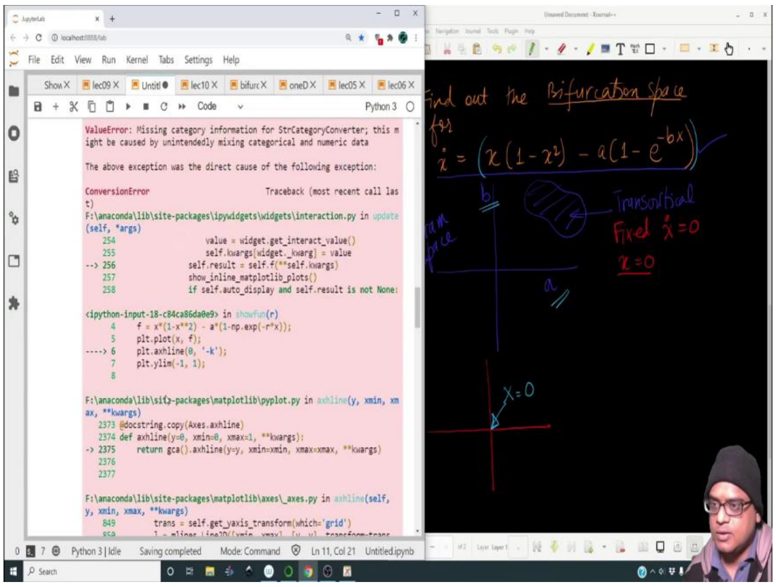

Find out the Bifurcation space for
$\dot{x} = (x(1-x^2) - a(1-e^{-bx}))$
Transcritical
Fixed $\dot{x}=0$
$x=0$
b
a
$x=0$

```
    a = 1;
    x = np.linspace(-3, 3);
    f = x*(1-x**2) - a*(1-np.exp(-r*x));
    plt.plot(x, f);
    plt.axhline(0, '-k');
    plt.ylim(-1, 1);

w = interactive(showfun, r = (-2, 2, 0.1));
w
```

```
r  0.00
```

```
---------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
F:\anaconda\lib\site-packages\matplotlib\axis.py in convert_units(self,
x)
   1572         try:
-> 1573             ret = self.converter.convert(x, self.units, self)
   1574         except Exception as e:

F:\anaconda\lib\site-packages\matplotlib\category.py in convert(value, un
it, axis)
     48         if unit is None:
---> 49             raise ValueError(
     50                 'Missing category information for StrCategoryConv
erter; '

ValueError: Missing category information for StrCategoryConverter; this m
ight be caused by unintendedly mixing categorical and numeric data
```

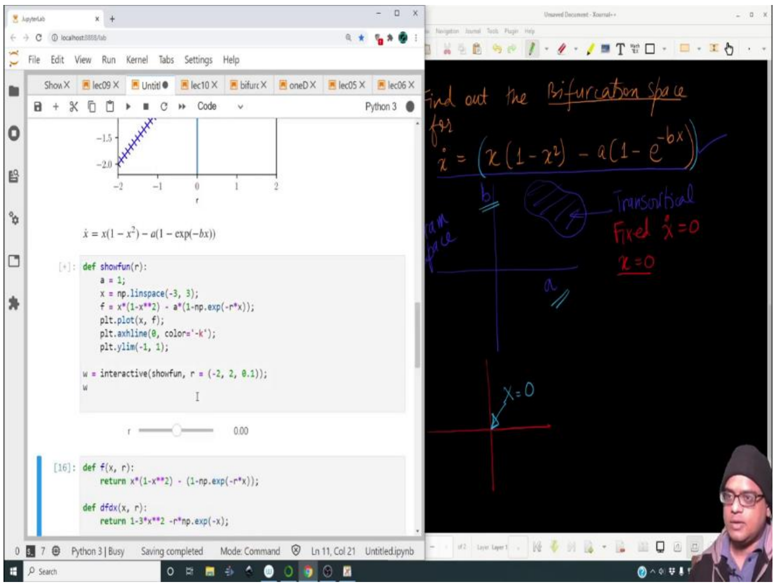
(Refer Slide Time: 19:07)


Find out the Bifurcation space for
$\dot{x} = (x(1-x^2) - a(1-e^{-bx}))$
Transcritical
Fixed $\dot{x}=0$
$x=0$
b
a
$x=0$

```
ValueError: Missing category information for StrCategoryConverter; this m
ight be caused by unintendedly mixing categorical and numeric data

The above exception was the direct cause of the following exception:

ConversionError                           Traceback (most recent call las
t)
F:\anaconda\lib\site-packages\ipywidgets\widgets\interaction.py in update
(self, *args)
    254                     value = widget.get_interact_value()
    255                     self.kwargs[widget._kwarg] = value
--> 256                 self.result = self.f(**self.kwargs)
    257                 show_inline_matplotlib_plots()
    258                 if self.auto_display and self.result is not None:

<ipython-input-18-c84ca86da0e9> in showfun(r)
      4     f = x*(1-x**2) - a*(1-np.exp(-r*x));
      5     plt.plot(x, f);
----> 6     plt.axhline(0, '-k');
      7     plt.ylim(-1, 1);
      8

F:\anaconda\lib\site-packages\matplotlib\pyplot.py in axhline(y, xmin, xm
ax, **kwargs)
   2373 @docstring.copy(Axes.axhline)
   2374 def axhline(y=0, xmin=0, xmax=1, **kwargs):
-> 2375     return gca().axhline(y=y, xmin=xmin, xmax=xmax, **kwargs)
   2376
   2377

F:\anaconda\lib\site-packages\matplotlib\axes\_axes.py in axhline(self,
y, xmin, xmax, **kwargs)
    849         trans = self.get_yaxis_transform(which='grid')
```

Let me show the x axis as well. Let me make it black and; let me set the y lim -1 to 1 that is fine. There seems to be an error; let us see what the error is so, color equal to -k ok.

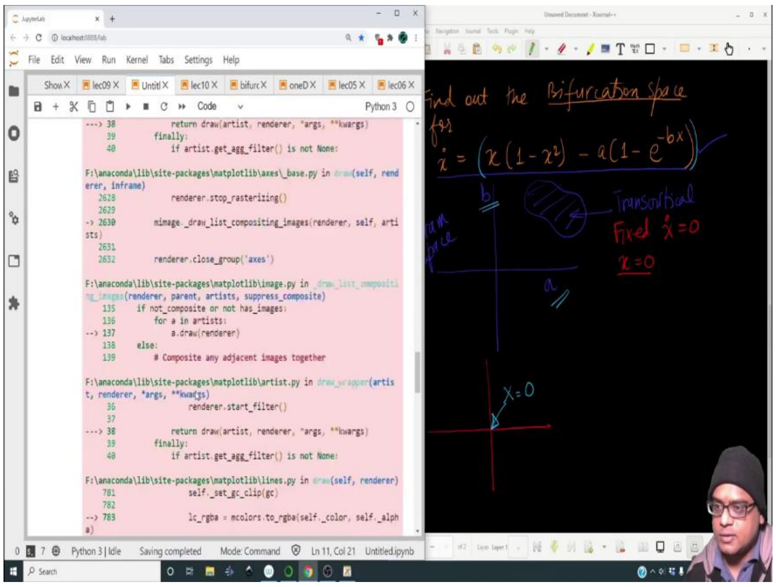
(Refer Slide Time: 19:12)



$\dot{x} = x(1-x^2) - a(1-\exp(-bx))$

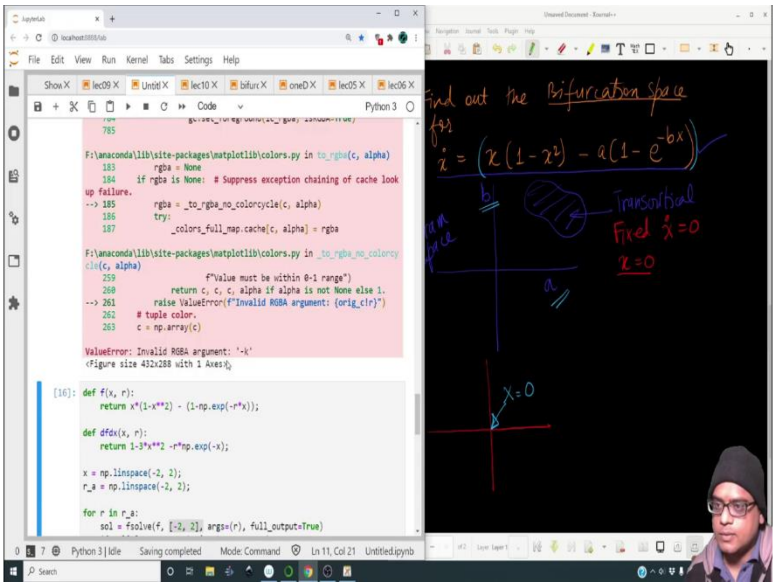
```
[*]: def showfun(r):
         a = 1;
         x = np.linspace(-3, 3);
         f = x*(1-x**2) - a*(1-np.exp(-r*x));
         plt.plot(x, f);
         plt.axhline(0, color='-k');
         plt.ylim(-1, 1);

     w = interactive(showfun, r = (-2, 2, 0.1));
     w

         r              0.00

[16]: def f(x, r):
          return x*(1-x**2) - (1-np.exp(-r*x));

      def dfdx(x, r):
          return 1-3*x**2 -r*np.exp(-x);
```

(Refer Slide Time: 19:19)



```
---> 38                 return draw(artist, renderer, *args, **kwargs)
     39             finally:
     40                 if artist.get_agg_filter() is not None:

F:\anaconda\lib\site-packages\matplotlib\axes\_base.py in draw(self, rend
erer, inframe)
   2628                 renderer.stop_rasterizing()
   2629
-> 2630             mimage._draw_list_compositing_images(renderer, self, arti
sts)
   2631
   2632             renderer.close_group('axes')

F:\anaconda\lib\site-packages\matplotlib\image.py in _draw_list_compositi
ng_images(renderer, parent, artists, suppress_composite)
    135     if not_composite or not has_images:
    136         for a in artists:
--> 137             a.draw(renderer)
    138     else:
    139         # Composite any adjacent images together

F:\anaconda\lib\site-packages\matplotlib\artist.py in draw_wrapper(artis
t, renderer, *args, **kwargs)
     36                     renderer.start_filter()
     37
---> 38                 return draw(artist, renderer, *args, **kwargs)
     39             finally:
     40                 if artist.get_agg_filter() is not None:

F:\anaconda\lib\site-packages\matplotlib\lines.py in draw(self, renderer)
    781                     self._set_gc_clip(gc)
    782
--> 783                     lc_rgba = mcolors.to_rgba(self._color, self._alph
a)
```

(Refer Slide Time: 19:20)



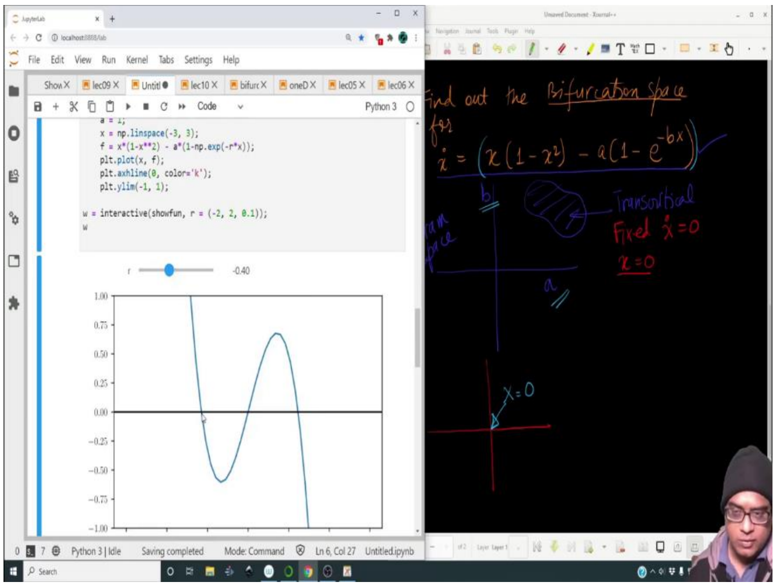
```
    785

F:\anaconda\lib\site-packages\matplotlib\colors.py in to_rgba(c, alpha)
    183         rgba = None
    184     if rgba is None:  # Suppress exception chaining of cache look
up failure.
--> 185         rgba = _to_rgba_no_colorcycle(c, alpha)
    186         try:
    187             _colors_full_map.cache[c, alpha] = rgba

F:\anaconda\lib\site-packages\matplotlib\colors.py in _to_rgba_no_colorcy
cle(c, alpha)
    259                     f"Value must be within 0-1 range")
    260             return c, c, c, alpha if alpha is not None else 1.
--> 261         raise ValueError(f"Invalid RGBA argument: {orig_c!r}")
    262     # tuple color.
    263     c = np.array(c)

ValueError: Invalid RGBA argument: '-k'
<Figure size 432x288 with 1 Axes>
```

```
[16]: def f(x, r):
          return x*(1-x**2) - (1-np.exp(-r*x));

      def dfdx(x, r):
          return 1-3*x**2 -r*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)
```

Find out the Bifurcation space
for
$\dot{x} = (x(1-x^2) - a(1-e^{-bx}))$
b
Transcritical
Fixed $\dot{x}=0$
$x=0$
a
$x=0$

There is still some error. What is the error? We cannot give line style ok.

(Refer Slide Time: 19:26)

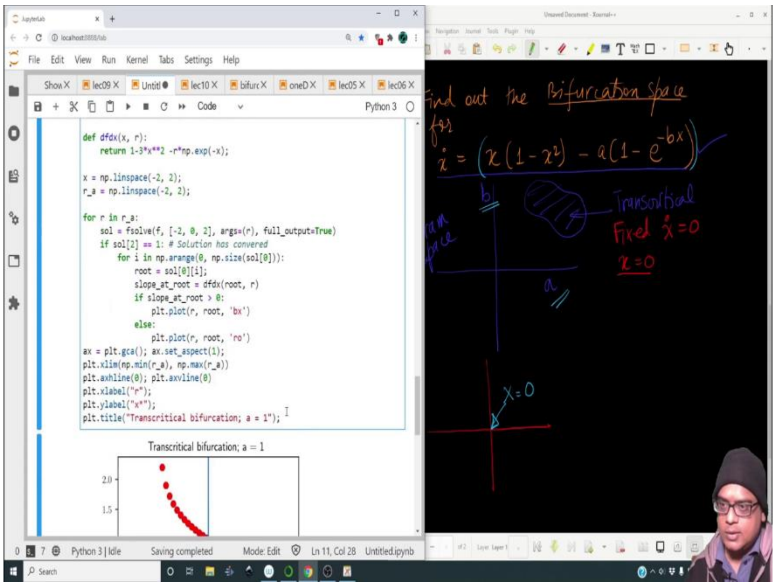


```
        a = 1;
        x = np.linspace(-3, 3);
        f = x*(1-x**2) - a*(1-np.exp(-r*x));
        plt.plot(x, f);
        plt.axhline(0, color='k');
        plt.ylim(-1, 1);

w = interactive(showfun, r = (-2, 2, 0.1));
w
```

r
-0.40
Find out the Bifurcation space
for
$\dot{x} = (x(1-x^2) - a(1-e^{-bx}))$
b
Transcritical
Fixed $\dot{x}=0$
$x=0$
a
$x=0$

So, it has to be only k fine. So, let us see what happens ok. So, we have the presence of 3 roots, but over here we were passing only 2 guess points.

(Refer Slide Time: 19:43)

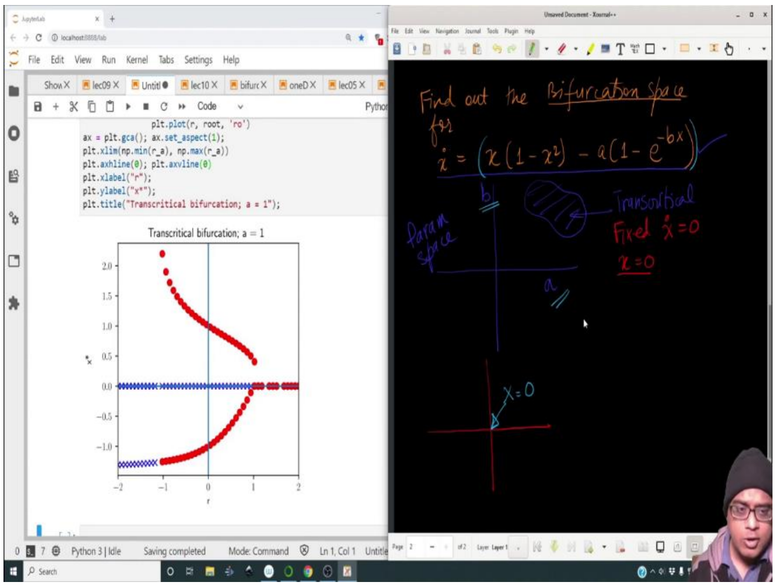


```
def dfdx(x, r):
    return 1-3*x**2 -r*np.exp(-x);

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

for r in r_a:
    sol = fsolve(f, [-2, 0, 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; a = 1");
```

Find out the Bifurcation space for
$\dot{x} = (x(1-x^2) - a(1-e^{-bx}))$
b
a
Param space
Transcritical
Fixed $\dot{x}=0$
$x=0$
$x=0$
Transcritical bifurcation; a = 1

So, when we are passing the guess point - 2 and + 2, it is converging to this root over here and this root over here and we are not able to capture this root. So, let me pass an additional guess point of 0 ok.

(Refer Slide Time: 20:03)



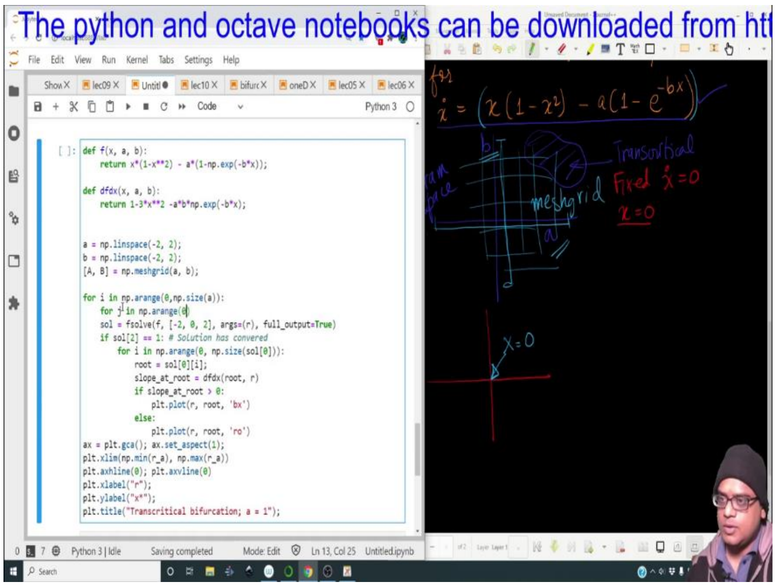
```
            plt.plot(r, root, 'ro')
ax = plt.gca(); ax.set_aspect(1);
plt.xlim(np.min(r_a), np.max(r_a))
plt.axhline(0); plt.axvline(0)
plt.xlabel("r");
plt.ylabel("x*");
plt.title("Transcritical bifurcation; a = 1");
```

Transcritical bifurcation; a = 1
x*
r
Find out the Bifurcation space for
$\dot{x} = (x(1-x^2) - a(1-e^{-bx}))$
Param space
Transcritical
Fixed $\dot{x}=0$
$x=0$
$x=0$

So, now let me rerun the cell (Refer Time: 20:04). We do have the zeroroot. So, what watch closely what happens, we have the 0 branch and suddenly it loses its stability. So, its so rather it gain stability. So, its unstable up until this point and then it becomes stable. So; obviously, there is a transcritical bifurcation occurring for the root x = 0 at b = 1 ok.

But we are more interested in probing this entire parameter space. For what parameter space does this happen? So, let us have a look. Let us take this entire code.

(Refer Slide Time: 20:42)


The python and octave notebooks can be downloaded from htt

```
def f(x, a, b):
    return x*(1-x**2) - a*(1-np.exp(-b*x));

def dfdx(x, a, b):
    return 1-3*x**2 -a*b*np.exp(-b*x);


a = np.linspace(-2, 2);
b = np.linspace(-2, 2);
[A, B] = np.meshgrid(a, b);

for i in np.arange(0,np.size(a)):
    for j in np.arange(0
    sol = fsolve(f, [-2, 0, 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; a = 1");
```

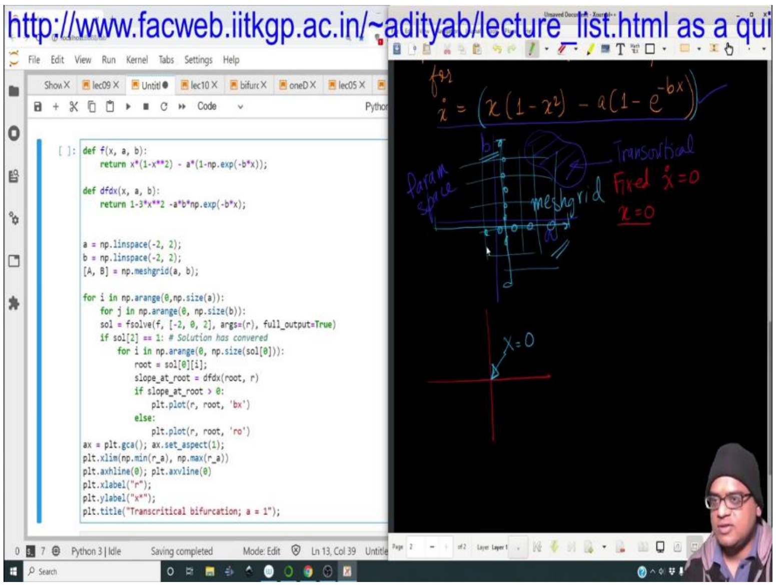
So, the beauty about doing this in Jupyter lab is to we can may reuse all the snippets that we have already done rather than having to write everything all over again ok. So, let us pass a as well. We need to eventually pass a and this will be a times this and this will be this. In fact, let me make it b.

So, that we maintain the sanctity of this problem and this will be b this should be b times x. So; obviously, there was an error over here. This should be r times x ok. Let me run this ok; no harm done great. So, avoid doing such errors; I am struck for time. So, I am trying to go through the code as quick as possible ok.

So, let me define a as np.linspace. We do not need x array over here, I do not know why we have kept this. Let b =np.linspace(-2,2). So, far we have defined the parameter space a and b, this has to be b. Let me make a 2 dimensional space A B. So, if you recall we can use mesh grid to declare 2 arrays like this and to form an entire grid from these two arrays ok. So, a and b will contain all the grid values of a and b alright.

So, for r in r a this should be in fact, let us do this for i in np.arange(0,np.size(a)); for j in np.arange(0,np.size(b)). So, whatever a and b you declare depending on that size, I will loop over all the different values of a and j will loop over all the different values of b ok.

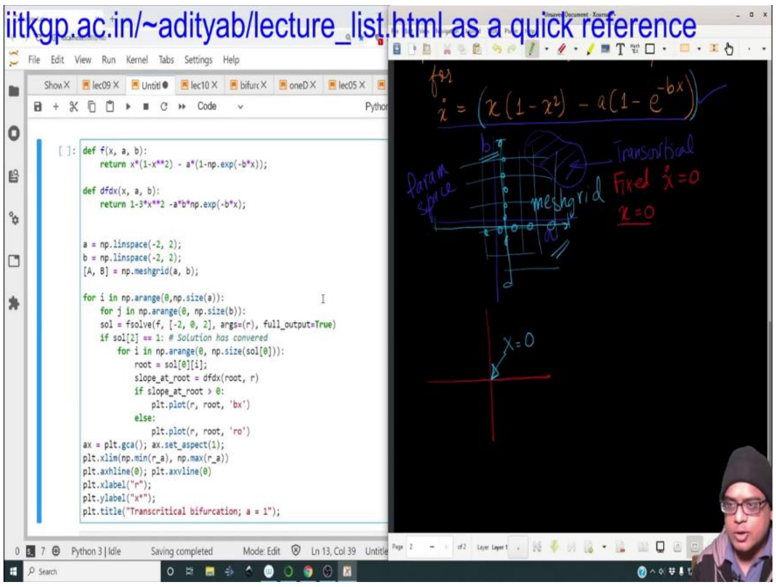
(Refer Slide Time: 23:06)



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

```
[ ]: def f(x, a, b):
         return x*(1-x**2) - a*(1-np.exp(-b*x));

     def dfdx(x, a, b):
         return 1-3*x**2 -a*b*np.exp(-b*x);


     a = np.linspace(-2, 2);
     b = np.linspace(-2, 2);
     [A, B] = np.meshgrid(a, b);

     for i in np.arange(0,np.size(a)):
         for j in np.arange(0, np.size(b)):
         sol = fsolve(f, [-2, 0, 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; a = 1");
```

(Refer Slide Time: 23:11)

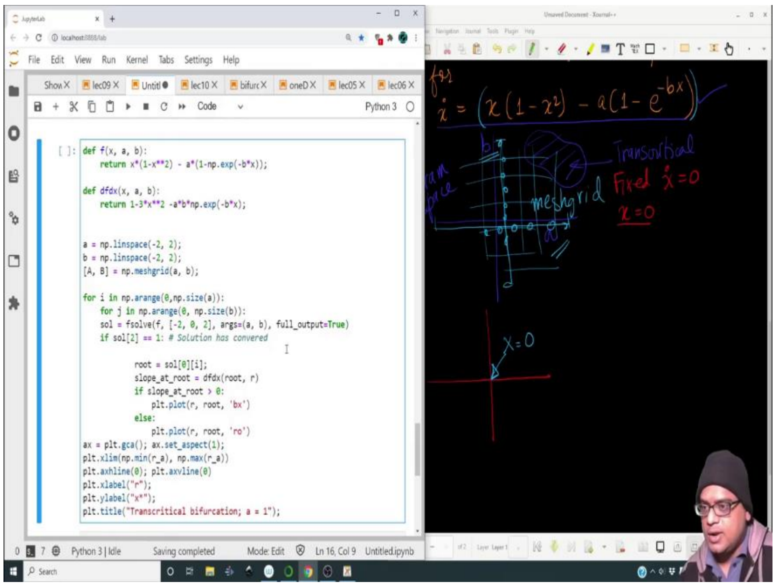


iitkgp.ac.in/~adityab/lecture_list.html as a quick reference

```
[ ]: def f(x, a, b):
         return x*(1-x**2) - a*(1-np.exp(-b*x));

     def dfdx(x, a, b):
         return 1-3*x**2 -a*b*np.exp(-b*x);


     a = np.linspace(-2, 2);
     b = np.linspace(-2, 2);
     [A, B] = np.meshgrid(a, b);

     for i in np.arange(0,np.size(a)):
         for j in np.arange(0, np.size(b)):
         sol = fsolve(f, [-2, 0, 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; a = 1");
```

So, essentially this is a double loop, I mean there are techniques where you can avoid using a double loop. But in order to keep the code readable at this stage; if you are not comfortable with those tips and tricks, I am keeping it as readable as possible.

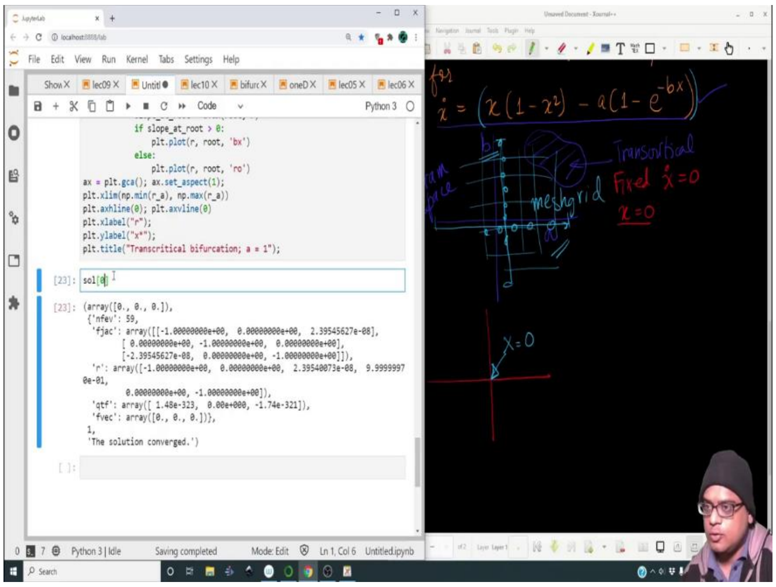
(Refer Slide Time: 23:26)



```
[ ]: def f(x, a, b):
         return x*(1-x**2) - a*(1-np.exp(-b*x));

     def dfdx(x, a, b):
         return 1-3*x**2 -a*b*np.exp(-b*x);


     a = np.linspace(-2, 2);
     b = np.linspace(-2, 2);
     [A, B] = np.meshgrid(a, b);

     for i in np.arange(0,np.size(a)):
         for j in np.arange(0, np.size(b)):
         sol = fsolve(f, [-2, 0, 2], args=(a, b), full_output=True)
         if sol[2] == 1: # Solution has convered

                 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; a = 1");
```

So, we have to pass to f solve the two parameters. We have to pass a and b as two parameters to the function ok. If the solution is converged, then what should we do? If the solution has converged, we must look into the roots ok. So, instead of looping over all the roots checking the stability, we are more interested in the x = zeroroot. So, how do we extract the x = zeroroot? That is the question.

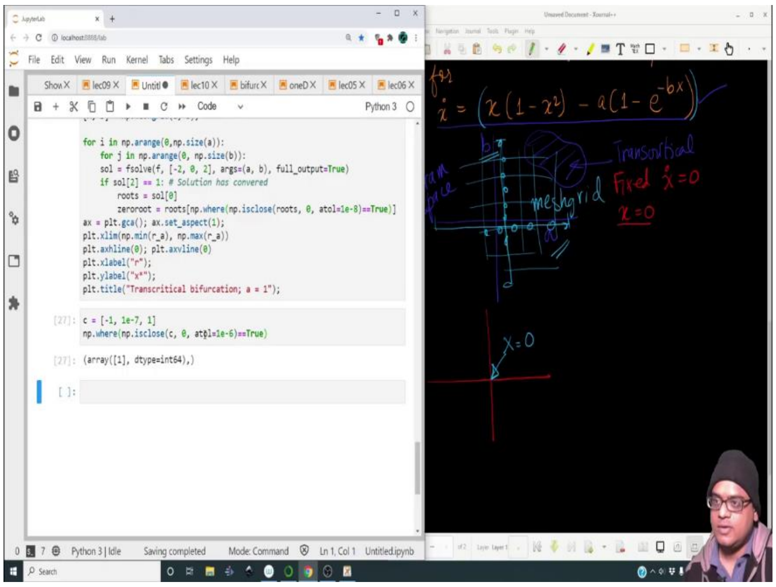
(Refer Slide Time: 24:05)



```
                 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; a = 1");

[23]: sol[0]

[23]: (array([0., 0., 0.]),
       {'nfev': 59,
        'fjac': array([[-1.00000000e+00,  0.00000000e+00,  2.39545627e-08],
              [ 0.00000000e+00, -1.00000000e+00,  0.00000000e+00],
              [-2.39545627e-08,  0.00000000e+00, -1.00000000e+00]]),
        'r': array([-1.00000000e+00,  0.00000000e+00,  2.39540073e-08,  9.9999997
      0e-01,
               0.00000000e+00, -1.00000000e+00]),
        'qtf': array([ 1.48e-323,  0.00e+000, -1.74e-321]),
        'fvec': array([0., 0., 0.])},
       1,
       'The solution converged.')

[ ]:
```

Let us see what sol contains. So, sol contains these things. So, sol 0 it contains all the roots. So, we must find which index contains the root. So, in general these will not be 0, there will be three different values, but one of those values will be close to 0. This is what we expect. So, because we are probing the change of stability of the x = 0 branch. So, now we have to extract the x = zeroroot. So, how do we do that?

(Refer Slide Time: 24:40)



```
for i in np.arange(0,np.size(a)):
    for j in np.arange(0, np.size(b)):
    sol = fsolve(f, [-2, 0, 2], args=(a, b), full_output=True)
    if sol[2] == 1: # Solution has convered
        roots = sol[0]
        zeroroot = roots[np.where(np.isclose(roots, 0, atol=1e-8)==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; a = 1");
```

```
[27]: c = [-1, 1e-7, 1]
      np.where(np.isclose(c, 0, atol=1e-6)==True)

[27]: (array([1], dtype=int64),)
```

So, roots equal to sol 0, we have extracted all the roots in. So, forget about all this we do not need all this for now. So, now, all the roots are dumped into the variable roots. Now let me say zeroroot.

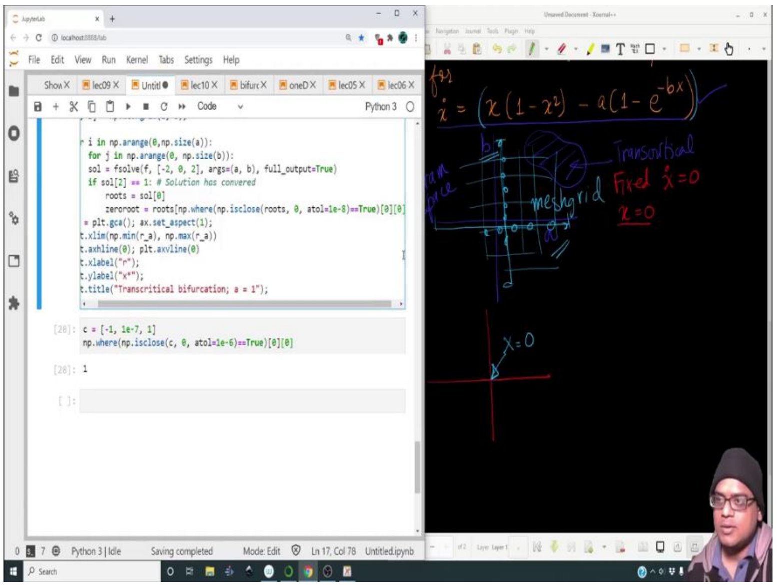
So, zeroroot has to be that root inside roots which is close to 0 and I say close to 0 because when the computer uses a non-linear algebraic solver, it will return roots which are not exactly 0. It will return roots which are 10^{-9} 10^{-6} or something like that. So, zeroroot, we have to check for something which is inside roots and it is something close to 0 ok.

So, how do we do that? We will say so, np.isclose(roots,0). So, this particular statement and let us give and atol of $1E-8$. So, this particular statement will say out of these roots which one is closest to 0.

So, now, we have to extract that root. So, this will be roots of this which will satisfy the condition, this is equal to true and we have to give the index. So, np.where this, this condition holds true. In fact, let me demonstrate this; let me define a variable c. So, let it c = [-1 1E-7 1] ok. So, this is c.

So, now, let me say np.isclose(c,0,atoll=1E-6). So, it says (False, True, False). So, I must extract this index of c. So, I said this is equal to True and I say np.where. So, it says the array index is 1. So, I must extract that.

(Refer Slide Time: 27:19)



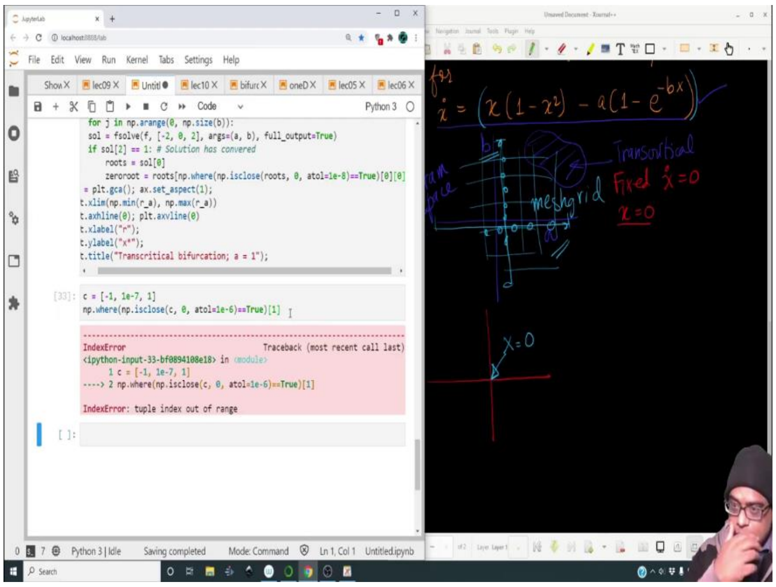
```
r i in np.arange(0,np.size(a)):
  for j in np.arange(0, np.size(b)):
  sol = fsolve(f, [-2, 0, 2], args=(a, b), full_output=True)
  if sol[2] == 1: # Solution has convered
      roots = sol[0]
      zeroroot = roots[np.where(np.isclose(roots, 0, atol=1e-8)==True)[0][0]
 = plt.gca(); ax.set_aspect(1);
t.xlim(np.min(r_a), np.max(r_a))
t.axhline(0); plt.axvline(0)
t.xlabel("r");
t.ylabel("x*");
t.title("Transcritical bifurcation; a = 1");
```

```
[28]: c = [-1, 1e-7, 1]
      np.where(np.isclose(c, 0, atol=1e-6)==True)[0][0]
```

```
[28]: 1
```

So, because the output of np.where() is inside a tuple, I must extract that value. So, it has to be [0][0] and this gives me the array index. So, this gives me an idea that I have to write it like this ok. So, where is the np.dot.where it is equal to 0 and I must extract the tuple. So, these are some idiosyncrasies of Python where you have to extract the value of the tuple; meaning if I remove these 0s and I print this so, it says its returns array one dtype data type.

(Refer Slide Time: 28:04)



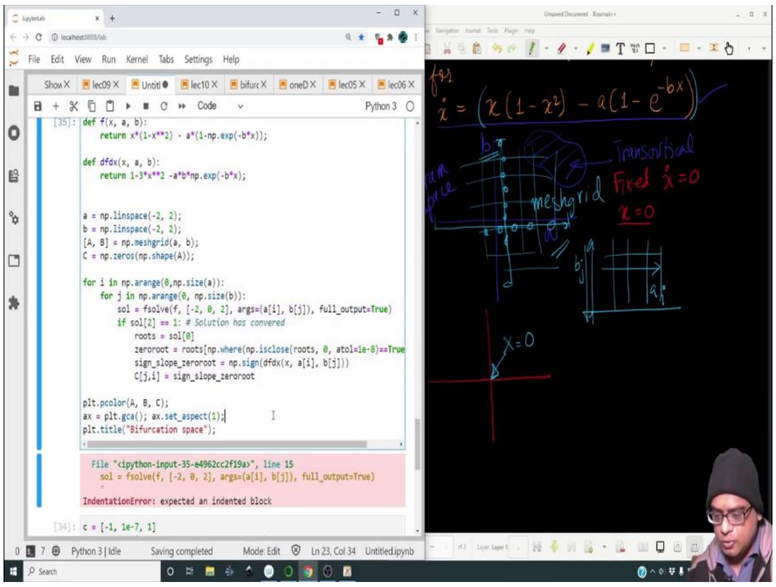
```
        for j in np.arange(0, np.size(b)):
        sol = fsolve(f, [-2, 0, 2], args=(a, b), full_output=True)
        if sol[2] == 1: # Solution has convered
            roots = sol[0]
            zeroroot = roots[np.where(np.isclose(roots, 0, atol=1e-8)==True)[0][0]
 = plt.gca(); ax.set_aspect(1);
t.xlim(np.min(r_a), np.max(r_a))
t.axhline(0); plt.axvline(0)
t.xlabel("r");
t.ylabel("x*");
t.title("Transcritical bifurcation; a = 1");
```

```
[33]: c = [-1, 1e-7, 1]
      np.where(np.isclose(c, 0, atol=1e-6)==True)[1]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-33-bf0894108e18> in <module>
      1 c = [-1, 1e-7, 1]
----> 2 np.where(np.isclose(c, 0, atol=1e-6)==True)[1]

IndexError: tuple index out of range
```

So, I if I say just 0, it will give array index data type and if I extract this, it will give me 1. If I extract this, it will give me the ok; its out of bounds. If I extract this or do I get nothing. So, the tuple is out of bound, but in order to extract the index, this particular index we have to do this fancy thing ok. So, this should give us the zeroroot. Now after finding the zeroroot what should we do? We should check for the slope at that point.

(Refer Slide Time: 28:39)



```
[35]: def f(x, a, b):
          return x*(1-x**2) - a*(1-np.exp(-b*x));

      def dfdx(x, a, b):
          return 1-3*x**2 -a*b*np.exp(-b*x);


      a = np.linspace(-2, 2);
      b = np.linspace(-2, 2);
      [A, B] = np.meshgrid(a, b);
      C = np.zeros(np.shape(A));

      for i in np.arange(0,np.size(a)):
          for j in np.arange(0, np.size(b)):
              sol = fsolve(f, [-2, 0, 2], args=(a[i], b[j]), full_output=True)
              if sol[2] == 1: # Solution has convered
                  roots = sol[0]
                  zeroroot = roots[np.where(np.isclose(roots, 0, atol=1e-8)==True
                  sign_slope_zeroroot = np.sign(dfdx(x, a[i], b[j]))
                  C[j,i] = sign_slope_zeroroot

      plt.pcolor(A, B, C);
      ax = plt.gca(); ax.set_aspect(1);
      plt.title("Bifurcation space");
```

```
  File "<ipython-input-35-e4962cc2f19a>", line 15
    sol = fsolve(f, [-2, 0, 2], args=(a[i], b[j]), full_output=True)
    ^
IndentationError: expected an indented block
```

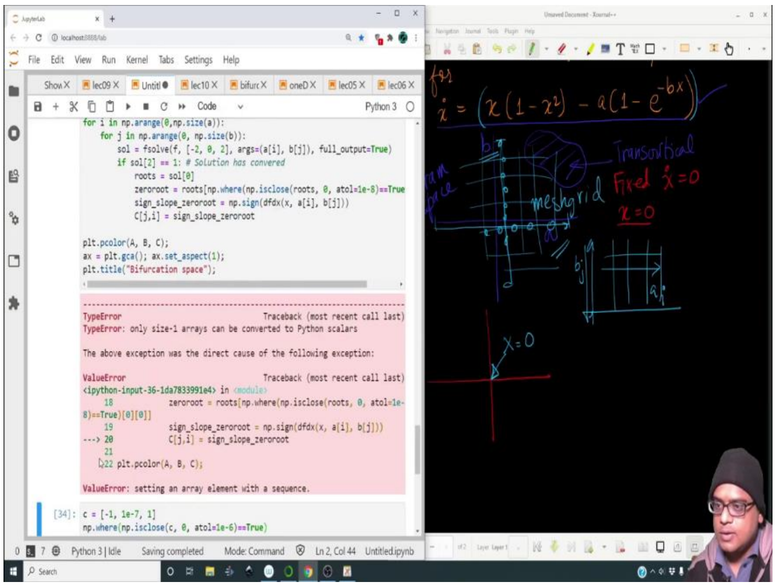
```
[34]: c = [-1, 1e-7, 1]
```

So, slope_zeroroot =dfdx(x, a[i],b[j]). So, now we have the slope at the root, but we are more interested in the sign of the slope of the root. So, let us say sign of the slope of the root and we will call the np.sign() function. So, if the sign of something is less than is negative np.sign will return -1 if the sign of the slope is 0, it will give you 0. If it is positive, it will give you 1 ok.

So, now, we have extracted the sign of the slope and we must now dump this value minus 1 0 or 1 into an empty matrix. So, let C = np.zeros(np.shape(A)) alright. So, we have initialized C as 0 and C[j,i] will take the value of sign_slope_zeroroot. So, its [j,i] and the reason is look over here a[i] is this and b[j] is this, but [i] is actually the column number [j] is the row number and [i] is the column number ok. So, that is something you need to keep in mind alright.

So, this gets the value. Once everything is done, we should do a pcolor plot a pseudo color plot. So, plt.pcolor(a,b,c); ax.set_aspect(1). We do not need these things and we will select the title later on. So, the title should be bifurcation space alright. So, let us run this cell and see what happen expected and in ok. So, we forgot to indent everything in the for loop.

(Refer Slide Time: 31:07)

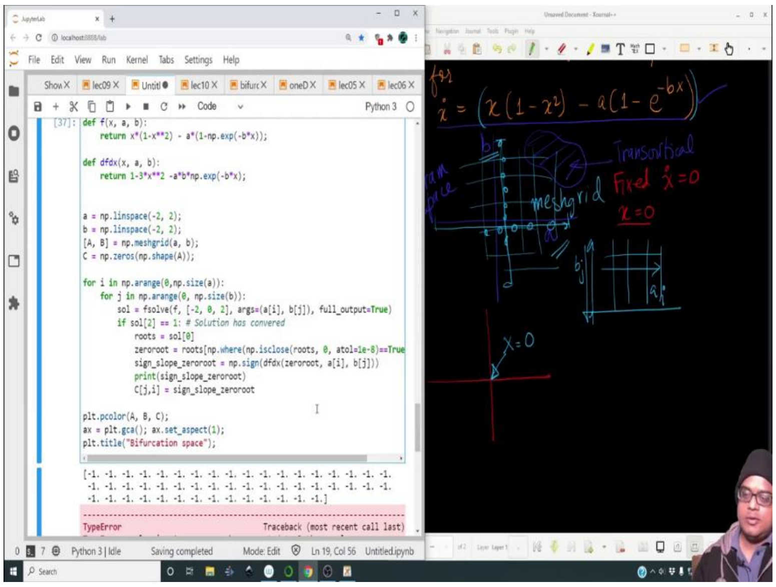


```
for i in np.arange(0,np.size(a)):
    for j in np.arange(0, np.size(b)):
        sol = fsolve(f, [-2, 0, 2], args=(a[i], b[j]), full_output=True)
        if sol[2] == 1: # Solution has convered
            roots = sol[0]
            zeroroot = roots[np.where(np.isclose(roots, 0, atol=1e-8)==True
            sign_slope_zeroroot = np.sign(dfdx(x, a[i], b[j]))
            C[j,i] = sign_slope_zeroroot

plt.pcolor(A, B, C);
ax = plt.gca(); ax.set_aspect(1);
plt.title("Bifurcation space");
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
TypeError: only size-1 arrays can be converted to Python scalars

The above exception was the direct cause of the following exception:

ValueError                                Traceback (most recent call last)
<ipython-input-36-1da7833991e4> in <module>
     18             zeroroot = roots[np.where(np.isclose(roots, 0, atol=1e-
8)==True)[0][0]]
     19             sign_slope_zeroroot = np.sign(dfdx(x, a[i], b[j]))
---> 20             C[j,i] = sign_slope_zeroroot
     21 
     22 plt.pcolor(A, B, C);

ValueError: setting an array element with a sequence.
```

```
[34]: c = [-1, 1e-7, 1]
      np.where(np.isclose(c, 0, atol=1e-6)==True)
```

(Refer Slide Time: 31:12)

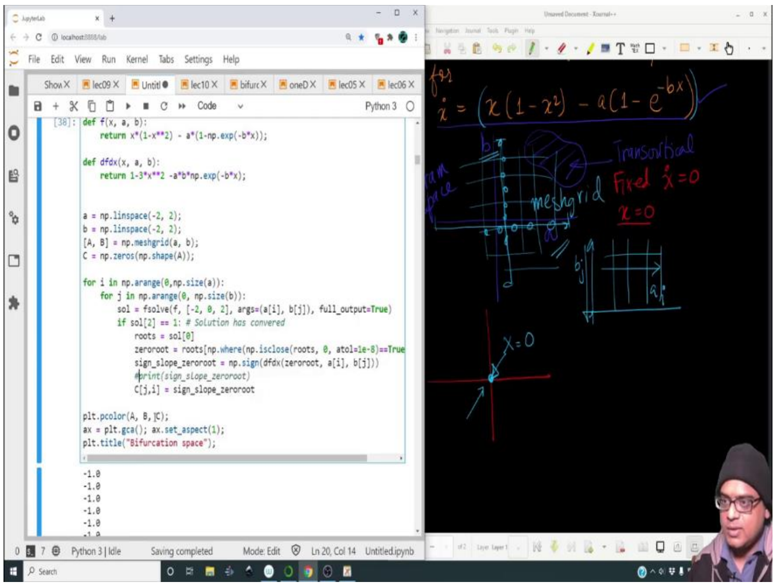


```
[37]: def f(x, a, b):
          return x*(1-x**2) - a*(1-np.exp(-b*x));

      def dfdx(x, a, b):
          return 1-3*x**2 -a*b*np.exp(-b*x);


      a = np.linspace(-2, 2);
      b = np.linspace(-2, 2);
      [A, B] = np.meshgrid(a, b);
      C = np.zeros(np.shape(A));

      for i in np.arange(0,np.size(a)):
          for j in np.arange(0, np.size(b)):
              sol = fsolve(f, [-2, 0, 2], args=(a[i], b[j]), full_output=True)
              if sol[2] == 1: # Solution has convered
                  roots = sol[0]
                  zeroroot = roots[np.where(np.isclose(roots, 0, atol=1e-8)==True
                  sign_slope_zeroroot = np.sign(dfdx(zeroroot, a[i], b[j]))
                  print(sign_slope_zeroroot)
                  C[j,i] = sign_slope_zeroroot

      plt.pcolor(A, B, C);
      ax = plt.gca(); ax.set_aspect(1);
      plt.title("Bifurcation space");
```

```
[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

Now, everything is inside the double for loop ok. There is an error. So, the error is because we are passing x. We should not pass x, we have to pass the value of the zeroroot. So, this has to be zeroroot. So, essentially what we are doing is; we are finding the zeroroot, we are taking that value and we pass the zeroroot to the function which finds us the slope at that point and this is that function. So, instead of passing x, we should pass zeroroot ok.

(Refer Slide Time: 31:46)

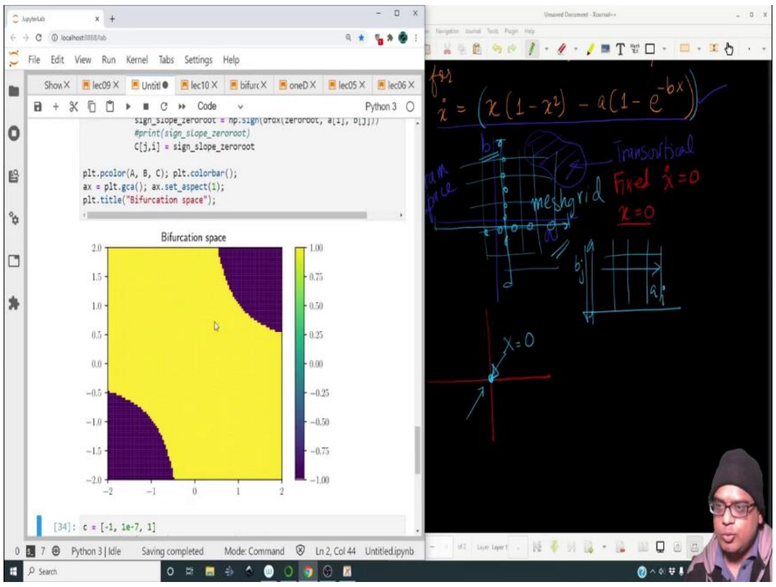


```
[38]: def f(x, a, b):
          return x*(1-x**2) - a*(1-np.exp(-b*x));

      def dfdx(x, a, b):
          return 1-3*x**2 -a*b*np.exp(-b*x);


      a = np.linspace(-2, 2);
      b = np.linspace(-2, 2);
      [A, B] = np.meshgrid(a, b);
      C = np.zeros(np.shape(A));

      for i in np.arange(0,np.size(a)):
          for j in np.arange(0, np.size(b)):
              sol = fsolve(f, [-2, 0, 2], args=(a[i], b[j]), full_output=True)
              if sol[2] == 1: # Solution has convered
                  roots = sol[0]
                  zeroroot = roots[np.where(np.isclose(roots, 0, atol=1e-8)==True
                  sign_slope_zeroroot = np.sign(dfdx(zeroroot, a[i], b[j]))
                  #print(sign_slope_zeroroot)
                  C[j,i] = sign_slope_zeroroot

      plt.pcolor(A, B, C);
      ax = plt.gca(); ax.set_aspect(1);
      plt.title("Bifurcation space");
```

```
-1.0
-1.0
-1.0
-1.0
-1.0
```

So, with that small error out of the way; let us run this and see what happens. Let me suppress the printing. One little thing before we continue in order to avoid some spurious things happening, let us give f solve an initial guess of 0 rather than something which has a range.

(Refer Slide Time: 32:06)



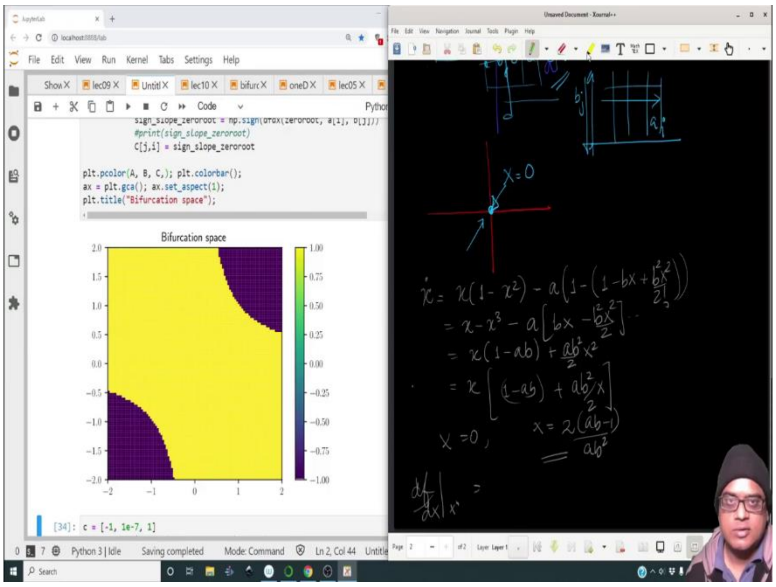
```
            sign_slope_zeroroot = np.sign(dfdx(zeroroot, a[i], b[j]))
            #print(sign_slope_zeroroot)
            C[j,i] = sign_slope_zeroroot

plt.pcolor(A, B, C); plt.colorbar();
ax = plt.gca(); ax.set_aspect(1);
plt.title("Bifurcation space");
```

```
[34]: c = [-1, 1e-7, 1]
```

So, this will help us to isolate things better. I do not; I do not care about the other roots; all I am worried about is the x = zeroroot ok. So, let me run this and see what happens ok. So, the bifurcation space, it shows that in the yellow region. So, let me put a color bar. So, the slope over here is positive, the slope in this zone is negative. So, when it crosses this curve, we have a transcritical bifurcation at the fixed point x = 0.

So, this is the zone where the transcritical bifurcation occurs. Let us quickly look at what that curve actually is and it looks something oddly specific and is there a theoretical way, we can figure out what that bifurcation curve should be. So, its; obviously, a curve it; it is a curve which looks like this and it is a curve which looks like this, but what is it really. Let us look at it.

(Refer Slide Time: 33:21)



```
            sign_slope_zeroroot = np.sign(dfdx(zeroroot, a[i], b[j]))
            #print(sign_slope_zeroroot)
            C[j,i] = sign_slope_zeroroot

plt.pcolor(A, B, C,); plt.colorbar();
ax = plt.gca(); ax.set_aspect(1);
plt.title("Bifurcation space");
```

```
[34]: c = [-1, 1e-7, 1]
```

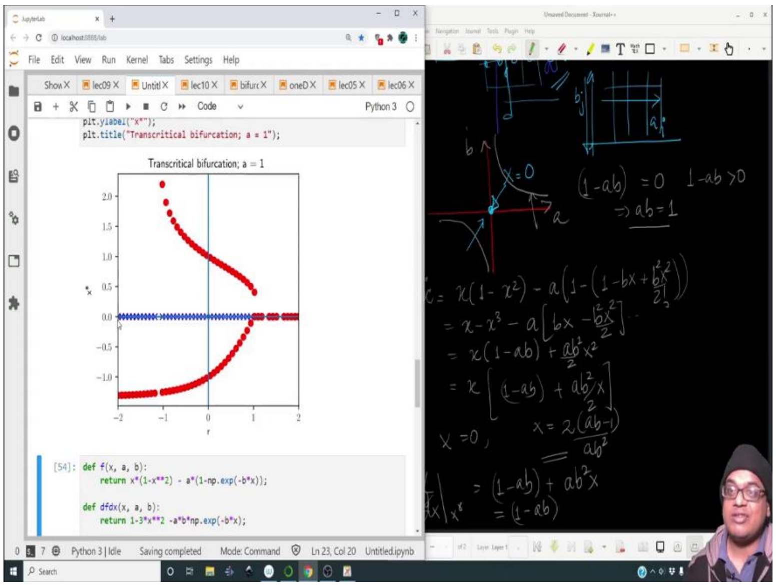
$$\dot{x} = x(1-x^2) - a\left(1-\left(1-bx+\frac{b^2x^2}{2!}\right)\right)$$
$$= x - x^3 - a\left[bx - \frac{b^2x^2}{2}\right] \cdots$$
$$= x(1-ab) + \frac{ab^2}{2}x^2$$
$$= x\left[(1-ab) + \frac{ab^2}{2}x\right]$$
$$x = 0, \quad x = \frac{2(ab-1)}{ab^2}$$
$$\left.\frac{df}{dx}\right|_{x^*} =$$

So, $\dot{x} = x(1 - x^2) - a(1 - e^{-bx})$ is this, it means that $\dot{x}$ in the vicinity of small x. It will be $\dot{x} = x(1 - x^2) - a\big(1 - (1 - bx + b^2x^2/2!)\big)$. So, when x is small rather b x is small when the argument bx is small, we can use a Taylor series expansion to write this. And we can simplify this as $\dot{x} = x - x^3 - a(bx - b^2x^2/2)$ higher order terms.

So, what does this become? This becomes $\dot{x} = x(1 - ab) + ab^2/2x^2$.So, what is the fixed point? So, the fixed point is $x((1 - ab) + ab^2/2\ x)$. So, one fixed point is; obviously, x = 0 and the other fixed point is; obviously, this root.

So, this root will be $x = 2(ab - 1)/(ab^2)$. So, when you set this inside thing to 0, this is the other root that you get. So, these are the two roots and what is df/dx for this entire thing? df dx in the vicinity of x in the vicinity of x; I mean this is not the correct way of writing it.

(Refer Slide Time: 35:01)



```
plt.ylabel("x*");
plt.title("Transcritical bifurcation; a = 1");
```

```
[54]: def f(x, a, b):
          return x*(1-x**2) - a*(1-np.exp(-b*x));

      def dfdx(x, a, b):
          return 1-3*x**2 -a*b*np.exp(-b*x);
```

$(1-ab) = 0 \quad 1-ab>0$

$\Rightarrow \underline{ab=1}$

$\dot{x} = x(1-x^2) - a\left(1-\left(1-bx+\frac{b^2x^2}{2!}\right)\right)$

$= x - x^3 - a\left[bx - \frac{b^2x^2}{2}\right]\cdots$

$= x(1-ab) + \frac{ab^2}{2}x^2$

$= x\left[(1-ab) + \frac{ab^2}{2}x\right]$

$x=0, \quad x = \frac{2(ab-1)}{ab^2}$

$\left.\frac{df}{dx}\right|_{x^*} = (1-ab) + ab^2x$

$= (1-ab)$

So, df dx in the vicinity of x is df/dx = $1 - ab + ab^2x$. So, when we find out the slope at the fixed point, we said this to be 0. So, this becomes $df/dx = 1 - ab$. So, the slope 1-ab when it is 0, it implies ab=1 and ab = 1is a rectangular hyperbola which looks something like this and it is exactly what we see over here. So, that is the demarcation that is the bifurcation curve in the abstract a b space.

So, when (1-ab) > 0, what happens? This zone this is that zone where we lie; when (1- ab) < 0, this is the zone that we lie. And hence this is the boundary which separates the two zones where the slope changes for the fixed point x = 0 and that is the description that is the mathematical description of the curve in the bifurcation space ok.

So, let us conclude this over here. In the next lecture, we will look at pitch fork bifurcations. So, far in this lecture, we have looked at how a fixed point can change its stability when a parameter is changed and it is different from a saddle node bifurcation in the sense that no roots are created or destroyed.

We are focusing on one fixed point and how it exchanges stability. We have looked at a problem in which we have studied the bifurcation space description of the same problem and how we can numerically plot the bifurcation curve ok.

So, with this we end this particular lecture. I will see you again next time with a lecture on pitchfork bifurcations. Until then, it is goodbye bye.