

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

**Lecture – 16**  
**Phase portraits – nonlinear systems**

Hi, everyone. In this lecture, we are going to look at nonlinear systems and their phase plots.

(Refer Slide Time: 00:32)

The image shows a split-screen view. The left side is a JupyterLab interface with a code editor and a plot. The code defines a vector field and uses `streamplot` to visualize it. The plot shows a vector field with trajectories spiraling outwards from the origin. The right side is a whiteboard with handwritten text and diagrams. The text includes 'Non linear systems - Phase plots portraits', a list of stability types: 'Real - stable - -ve', '- unstable - +ve', and '- saddle node'. Below this is a vector field diagram for the system  $\dot{x} = x + e^{-y}$  and  $\dot{y} = -y$ , with arrows indicating the direction of flow.

So, let me create a new file and let us see what the even natural evolution. So, in the previous two lectures we have looked at behavior of linear systems and the behavior of linear systems near a fixed point was decided by the nature of the eigenvalues and eigenvectors.

So, if we have real eigenvalues, then if the eigenvalues are real, so, we can either have a stable node or unstable node or a saddle node. And, this depends whether both of the eigenvalues are negative, both of them are positive or either of them is positive and negative.

Then we have seen that the behavior for the case of imaginary eigenvalues either has a spiral inwards depending on the real part or spiral outwards, depending on whether the real part is positive or a closed orbit this is the case where the real part is equal to 0, it is purely imaginary.

So, now these ideas can be used to analyze non-linear systems as well. In particular, if we have fixed points then we can perform linearization near the fixed points and see the behavior. Let us start with an example. So,  $\dot{x} = x + e^{-y}$  and  $\dot{y} = -y$ .

So, this particular differential equation it is an example from Strogatz and, let us see how the phase plot for this equation looks like. And, the thing about the phase plots is we can reuse much of the code that we have made for the linear part we just have to change these particular so called velocities. So, this specifies the vector flow and our task is to specify this as the new set of velocities. And, finding out the trajectories is no different.

We can have in the phase portrait if this is  $x$  and this is  $y$ , we can have different initial conditions and we can have the evolution of trajectories which are in this case governed by this non-linear set of equations. There is no analytical solution available even if there is some analytical solution available it is often quite difficult to make sense or rather to interpret that equation because it is not clear as to how the variation will occur explicitly ok.

(Refer Slide Time: 03:41)

The image shows a JupyterLab environment with a Python code cell on the left and a handwritten slide on the right. The code cell contains the following Python code:

```
[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.integrate import solve_ivp
from mayavi import mlab

[2]: x = np.linspace(-3, 3, 20); y = np.linspace(-3, 3, 20);
X, Y = np.meshgrid(x, y);
U = X + Y;
V = 4*X - 2*Y;
plt.quiver(X, Y, U, V);
ax = plt.gca(); ax.set_aspect(1);

x1 = np.linspace(-3, 3, 100); y1 = np.linspace(-3, 3, 100);
X1, Y1 = np.meshgrid(x1, y1);
U1 = X1 + Y1;
V1 = 4*X1 - 2*Y1;
#seedpoints = np.array([[0, 1.5], [2.5, 0]]);
plt.streamplot(X1, Y1, U1, V1, integration_direction='forward', density=1);
plt.plot(x1, y1, '-k');
plt.plot(x1, -4*x1, '-k');
plt.ylim(-3, 3);

[3]:
```

The handwritten slide on the right is titled "Non linear systems - Phase plots portraits". It contains the following text and diagrams:

- Real - stable -  $-ve$
- unstable -  $+ve$
- saddle node

Below the text, there is a diagram of a phase portrait showing a spiral trajectory around a fixed point. To the right of this diagram, the text "Vector flow" is written. Below the text, there is a diagram of a vector field for the system of equations:

$$\begin{cases} \dot{x} = x + e^{-y} \\ \dot{y} = -y \end{cases}$$

The vector field diagram shows a coordinate system with  $x$  and  $y$  axes. Arrows indicate the direction of the vector flow, showing a saddle point at the origin and trajectories that curve around it.

So, let us go to the lecture and let us copy this because we need this. In fact, for the 3D plots we will require maya as well mayavi as well alright. We need to import this and let us import.

(Refer Slide Time: 04:05)

Non linear systems - Phase plots portraits

- Real - stable - -ve
- unstable - +ve
- saddle node

$$\begin{cases} \dot{x} = x + e^{-y} \\ \dot{y} = -y \end{cases} \text{ Vector flow}$$

Let us import this particular plot or this particular cell. In fact, let me run this to see everything is fine ok, everything is fine.

(Refer Slide Time: 04:16)

Non linear systems - Phase plots portraits

- Real - stable - -ve
- unstable - +ve
- saddle node

$$\begin{cases} \dot{x} = x + e^{-y} \\ \dot{y} = -y \end{cases} \text{ Vector flow}$$

So, let me now change the flows. So, this is  $x + e^{-y}$ .

(Refer Slide Time: 04:26)

```

x = np.linspace(-3, 3, 20); y = np.linspace(-3, 3, 20);
X, Y = np.meshgrid(x, y);
U = X + np.exp(-Y);
V = -Y;
plt.quiver(X, Y, U, V);
ax = plt.gca(); ax.set_aspect(1);

x1 = np.linspace(-3, 3, 100); y1 = np.linspace(-3, 3, 100);
X1, Y1 = np.meshgrid(x1, y1);
U1 = X1 + np.exp(-Y1);
V1 = -Y1;
#seedpoints = np.array([[0, 1.5], [2.5, 0]]);
plt.streamplot(X1, Y1, U1, V1, integration_direction='forward', density=1);
plt.plot(x1, x1, '-k');
plt.plot(x1, -4*x1, '-k');
plt.ylim(-3, 3);

```

Nonlinear systems - Phase plots portraits

- Real - stable -  $-ve$
- unstable -  $+ve$
- saddle node:

$$\begin{cases} \dot{x} = x + e^{-y} \\ \dot{y} = -y \end{cases}$$

Vector flow

And, this is simply going to be - Y alright. Let me change this as well. Let us first see how the flow looks like ok.

(Refer Slide Time: 04:50)

```

x1 = np.linspace(-3, 3, 100); y1 = np.linspace(-3, 3, 100);
X1, Y1 = np.meshgrid(x1, y1);
U1 = X1 + np.exp(-Y1);
V1 = -Y1;
#seedpoints = np.array([[0, 1.5], [2.5, 0]]);
plt.streamplot(X1, Y1, U1, V1, integration_direction='forward', density=1);
plt.plot(x1, x1, '-k');
plt.plot(x1, -4*x1, '-k');
plt.ylim(-3, 3);

```

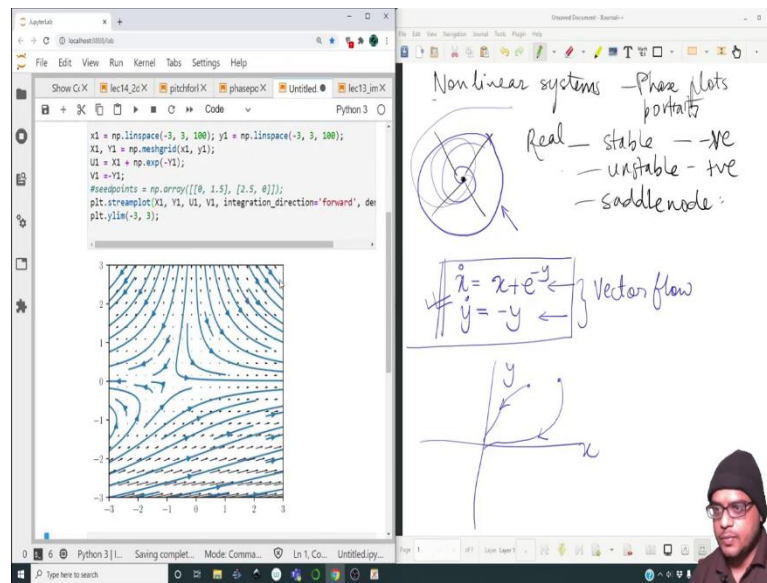
Nonlinear systems - Phase plots portraits

- Real - stable -  $-ve$
- unstable -  $+ve$
- saddle node:

$$\begin{cases} \dot{x} = x + e^{-y} \\ \dot{y} = -y \end{cases}$$

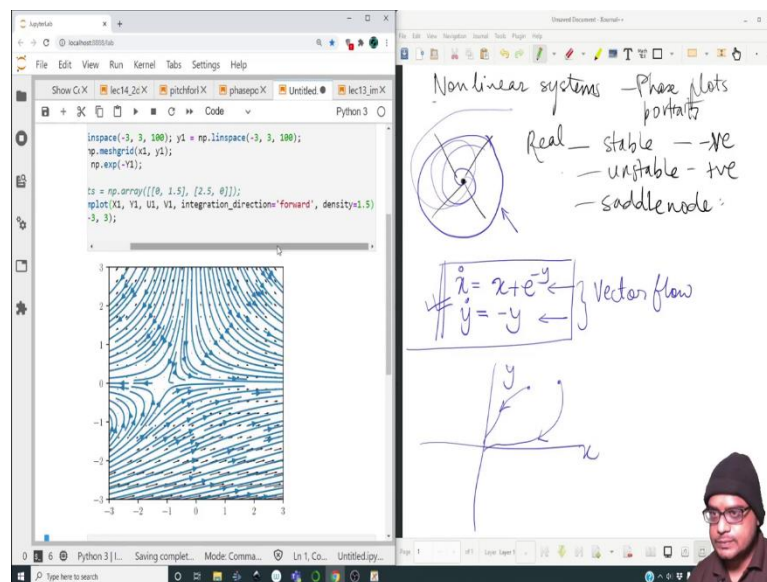
Vector flow

(Refer Slide Time: 04:54)



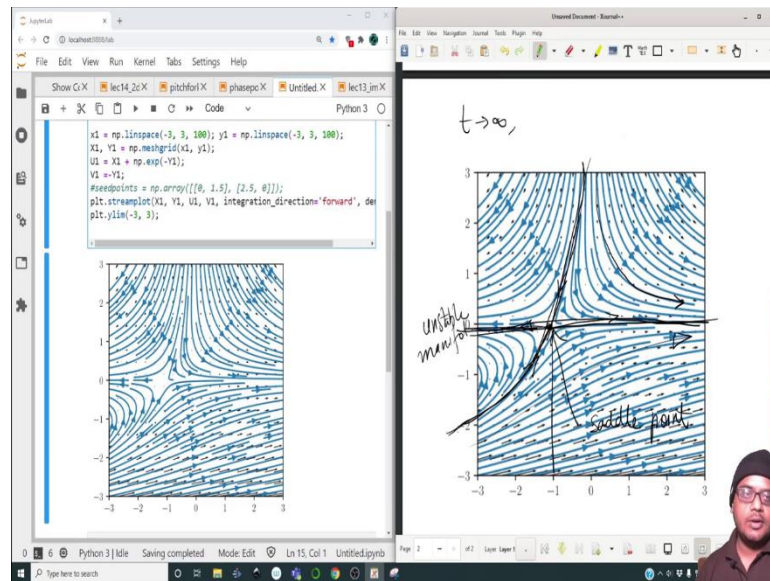
So, let me get rid of these eigenvectors that we have plotted, alright. So, this is how the flow looks like ok.

(Refer Slide Time: 05:04)



So, let me increase the density to 1.5 ok. So, write over here let me take this plot to our notebook over here ok.

(Refer Slide Time: 05:18)

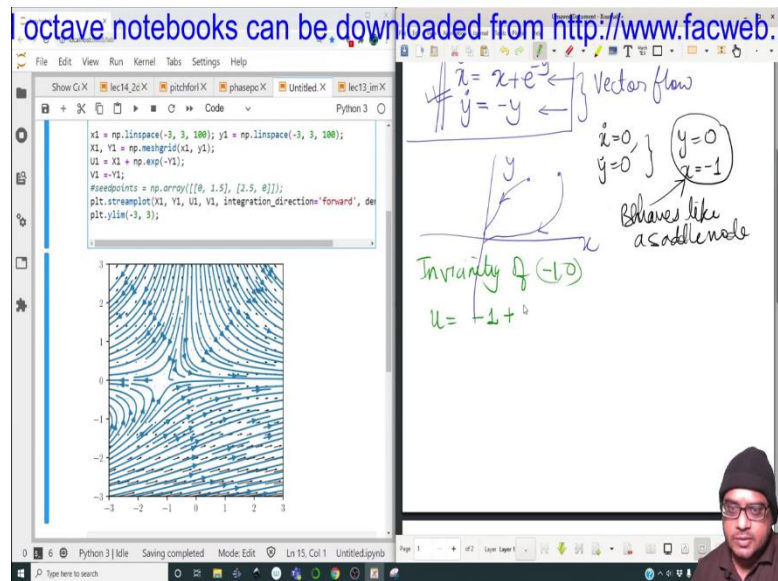


So, let us try to make sense of this. So, if you have a close look there appears to be a stagnation point over here; meaning, streamlines go like this and like this and like this and like this. So, things appear to be converging from in this particular line and it appears to be diverging along this particular line, alright and these two sets of curves appear to demarcate the entire phase plane into this kind of behavior.

So, as time goes to infinity as  $t$  goes to infinity the trajectories are attracted towards the unstable manifold. So, this appears to be the unstable manifold, while this is the stable manifold ok. So, as time goes to infinity we see that they are indeed going towards the unstable manifold.

So, this point appears to be a non-linear version of a saddle point ok. It appears to be the non-linear version of a saddle point. So, now let us try to make sense of this particular equation why does it look like this. Obviously, what are the fixed points of the system?

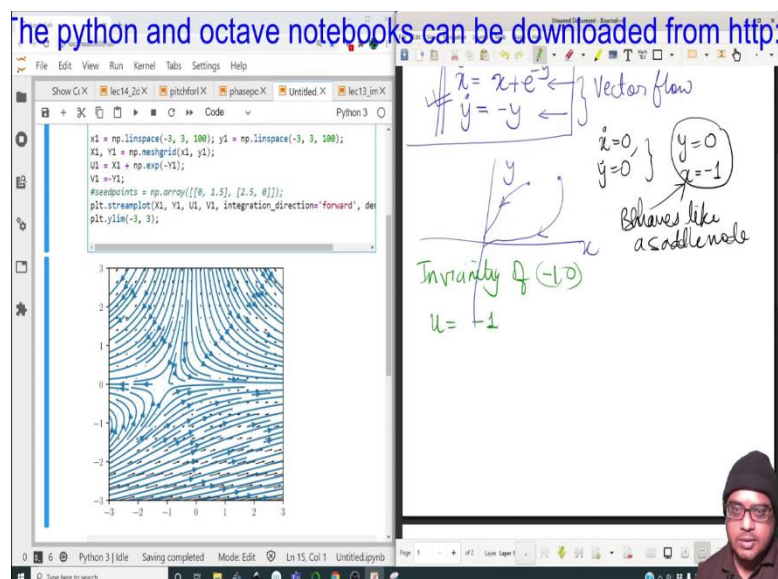
(Refer Slide Time: 06:56)



So, the fixed points correspond to  $\dot{x} = 0$  and  $\dot{y} = 0$ ; that means,  $y = 0$  and when  $y = 0$ ,  $x = -1$ . So, at  $x = -1$  and  $y = 0$  we do see that there is a saddle point this corresponds to  $(-1, 0)$ , alright. So, that explains the.

So, what about the behavior, what about the behavior of that point? Ok. So, why does it appear that this particular point behaves like a saddle node? The saddle node it implies that one eigen direction towards it is stable and the other is unstable.

(Refer Slide Time: 07:50)



So, let us look deeply into the structure of the vector field in the vicinity of the fixed point actually it is  $(-1, 0)$  ok. So, let  $u = -1 + 0$ .

(Refer Slide Time: 08:09)

[www.facweb.iitkgp.ac.in/~adityab/lecture\\_list.html](http://www.facweb.iitkgp.ac.in/~adityab/lecture_list.html) as a quick refer

```
x1 = np.linspace(-3, 3, 100); y1 = np.linspace(-3, 3, 100);  
X1, Y1 = np.meshgrid(x1, y1);  
U1 = X1 + np.exp(-Y1);  
V1 = -Y1;  
#seedpoints = np.array([[0, 1.5], [2.5, 0]]);  
plt.streamplot(X1, Y1, U1, V1, integration_direction='forward', density=1, linewidth=0.5, arrowsize=10, arrowcolor='b');  
plt.ylim(-3, 3);
```

Vector flow

$$\begin{cases} \dot{x} = x + e^{-y} \\ \dot{y} = -y \end{cases}$$

$\dot{x} = 0$   
 $\dot{y} = 0$

$y = 0$   
 $y = -1$

Behaves like a saddle node

Invariance of  $(-1)$

(Refer Slide Time: 08:19)

[d from http://www.facweb.iitkgp.ac.in/~adityab/lecture\\_list.html](http://www.facweb.iitkgp.ac.in/~adityab/lecture_list.html)

```
x1 = np.linspace(-3, 3, 100); y1 = np.linspace(-3, 3, 100);  
X1, Y1 = np.meshgrid(x1, y1);  
U1 = X1 + np.exp(-Y1);  
V1 = -Y1;  
#seedpoints = np.array([[0, 1.5], [2.5, 0]]);  
plt.streamplot(X1, Y1, U1, V1, integration_direction='forward', density=1, linewidth=0.5, arrowsize=10, arrowcolor='b');  
plt.ylim(-3, 3);
```

Vector flow

$$\begin{cases} \dot{x} = x + e^{-y} \\ \dot{y} = -y \end{cases}$$

$\dot{x} = 0$   
 $\dot{y} = 0$

$y = 0$   
 $y = -1$

Behaves like a saddle node

Invariance of  $(-1)$

Or rather  $x + 1$  and  $v$  be equal to  $y$ .

(Refer Slide Time: 08:30)



The Jupyter Notebook code is as follows:

```

x1 = np.linspace(-3, 3, 100); y1 = np.linspace(-3, 3, 100);
X1, Y1 = np.meshgrid(x1, y1);
U1 = X1 + np.exp(-Y1);
V1 = -Y1;
#seedpoints = np.array([[0, 1.5], [2.5, 0]]);
plt.streamplot(X1, Y1, U1, V1, integration_direction='forward', density=1);
plt.ylim(-3, 3);

```

The handwritten document contains the following text:

Vector flow  
 $\dot{x} = x + e^{-y}$   
 $\dot{y} = -y$

Fixed points:  
 $\dot{x} = 0$   
 $\dot{y} = 0$   
 $y = 0$   
 $x = -1$

Behaves like a saddle node

Invariance of  $(-1, 0)$   $(x^*, y^*)$   
Near  $(x^*, y^*)$   
 $u = x - x^*$ ,  $v = y - y^*$   
 $\dot{u} = \dot{x} - 0$ ,  $\dot{v} = \dot{y} - 0$   
 $\dot{u} = \dot{x}$ ,  $\dot{v} = \dot{y}$   
 $\dot{u} = f(x, y)$ ,  $\dot{v} = g(x, y)$   
 $\dot{u} = f(x^* + u, y^* + v)$   
 $\dot{x} = f(x, y)$   
 $\dot{y} = g(x, y)$

So, in fact, let us look at the let us denoted by  $x^* y^*$ , so that we can keep things symbolic until we make the final substitutions. So, let us choose a variable  $u = x - x^*$  and  $v = y - y^*$  in that case  $\dot{u} = \dot{x} - 0$   $\dot{v} = \dot{y} - 0$ . So,  $\dot{u} = \dot{x}$  and  $\dot{v} = \dot{y}$ .

So,  $u$  and  $v$  are sort of the; sort of the behavior. So, if this is the fixed point it is the behavior in the vicinity of the fixed point ok, near  $x^* y^*$ . So, then this is equal to. So, if I write down  $x^* \dot{x} = f(x, y)$  and  $\dot{y} = g(x, y)$  then I can write this down as  $f(x, y)$  and I can write this term as  $g(x, y)$ . Now,  $f(x, y)$  I can write it in terms of this as  $f(x^* + u, y^* + v)$  ok.

(Refer Slide Time: 09:57)

The Jupyter Notebook code is as follows:

```

x1 = np.linspace(-3, 3, 100); y1 = np.linspace(-3, 3, 100);
X1, Y1 = np.meshgrid(x1, y1);
U1 = X1 + np.exp(-Y1);
V1 = -Y1;
#seedpoints = np.array([[0, 1.5], [2.5, 0]]);
plt.streamplot(X1, Y1, U1, V1, integration_direction='forward', density=1);
plt.ylim(-3, 3);

```

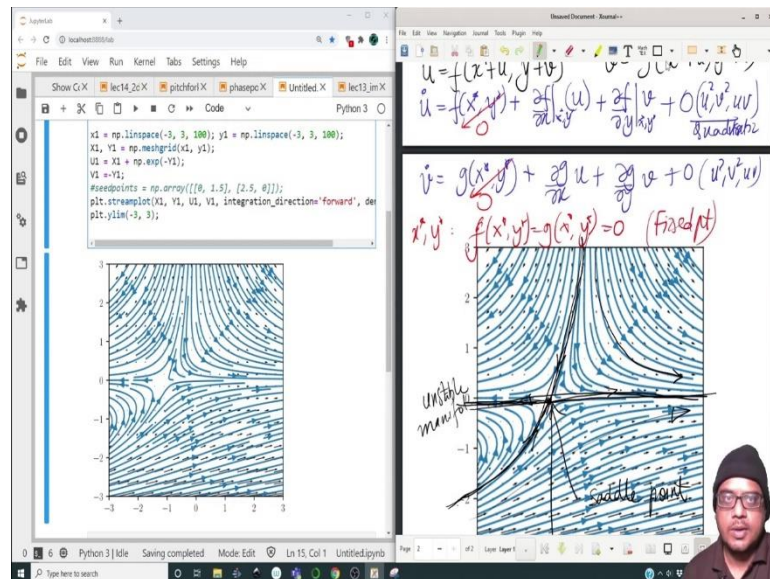
The handwritten document contains the following text:

$u = x - 0$ ,  $v = y - 0$   
 $\dot{u} = \dot{x}$ ,  $\dot{v} = \dot{y}$   
 $\dot{u} = f(x, y)$ ,  $\dot{v} = g(x, y)$   
 $\dot{u} = f(x^* + u, y^* + v)$ ,  $\dot{v} = g(x^* + u, y^* + v)$   
 $\dot{u} = f(x, y) + \frac{\partial f}{\partial x}(u) + \frac{\partial f}{\partial y}v + O(u^2, uv, v^2)$   
 $\dot{v} = g(x^*, y^*) + \frac{\partial g}{\partial x}u + \frac{\partial g}{\partial y}v + O(u^2, v^2, uv)$

unstable manifold

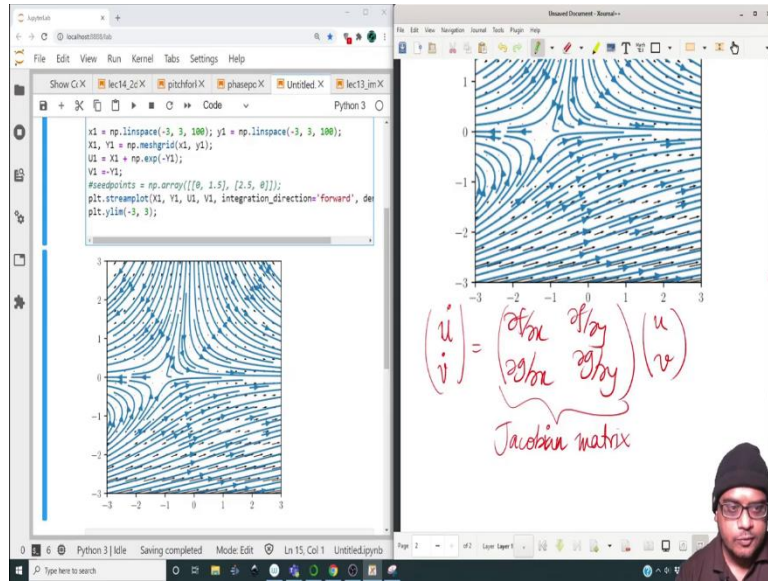
Similarly, I can write down  $\dot{v} = g(x^* + u, y^* + v)$ , alright. So, now we can use Taylor series expansion to write this as  $f(x^*, y^*) + \frac{\partial f}{\partial x}|_{x^*, y^*}u + \frac{\partial f}{\partial y}|_{x^*, y^*}v + \mathcal{O}(u^2, v^2, uv)$ . So, this is going to be  $u$  dot plus order  $u$  square  $v$  square and  $u v$  term; so, basically quadratic terms. Similarly, we can write down  $\dot{v} = g(x^*, y^*) + \frac{\partial g}{\partial x}|_{x^*, y^*}u + \frac{\partial g}{\partial y}|_{x^*, y^*}v + \mathcal{O}(u^2, v^2, uv)$ .

(Refer Slide Time: 11:01)



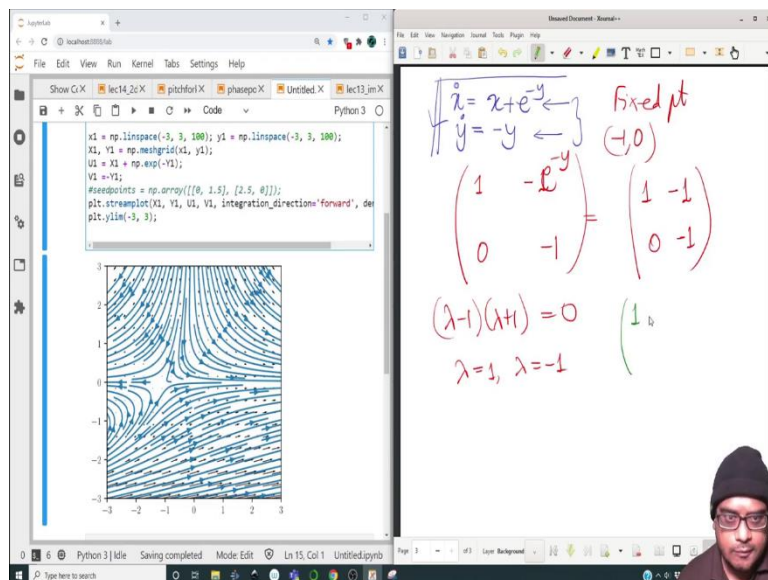
So, essentially we can remove these this term and this term because  $x^*$  and  $y^*$  are fixed points, it naturally means  $f(x^*, y^*)$  and  $g(x^*, y^*)$  will be equal to 0 because it is a fixed point.

(Refer Slide Time: 11:25)



And, hence well finally, we can write down  $\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \begin{bmatrix} \partial f / \partial x & \partial f / \partial y \\ \partial g / \partial x & \partial g / \partial y \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$ . So, we have linearized the system near the fixed point and it depends on the eigenvalues and eigenvectors of this particular matrix which is also called as the Jacobian matrix. It depends on the local slopes of the functions, alright. So, let us quickly look at the Jacobian of this.

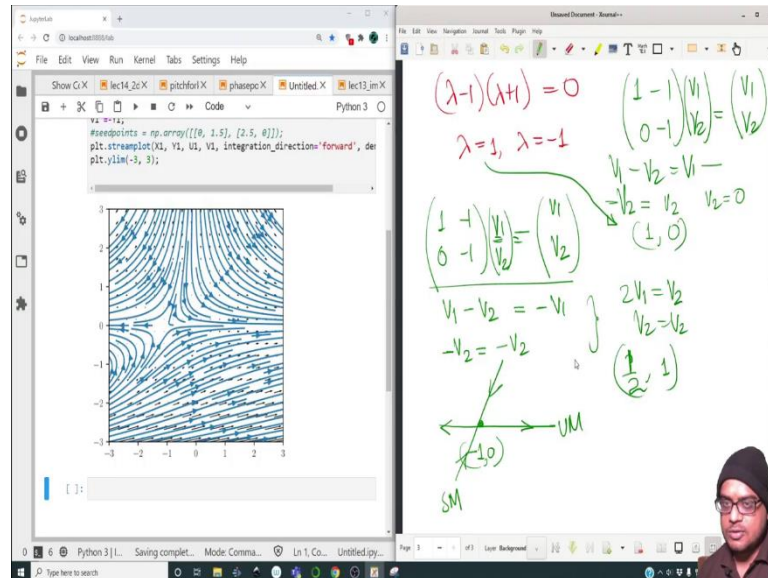
(Refer Slide Time: 12:10)



So, the Jacobian of this will be del f del x which is 1 del f del y which is equal to - y sorry - 1 well, - e to the power - y this will be 0, this will be - 1 and we have to evaluate the Jacobian and the fixed point. So, the fixed point is equal to 0 rather - 1, 0. So, this is equal to 1 and at 0 this is equal to - 1, 0, - 1, alright. So, what are the eigenvalues?

$(\lambda - 1)(\lambda + 1) = 0$ . So,  $\lambda = 1$  and  $\lambda = -1$ , these are the two eigenvalues.

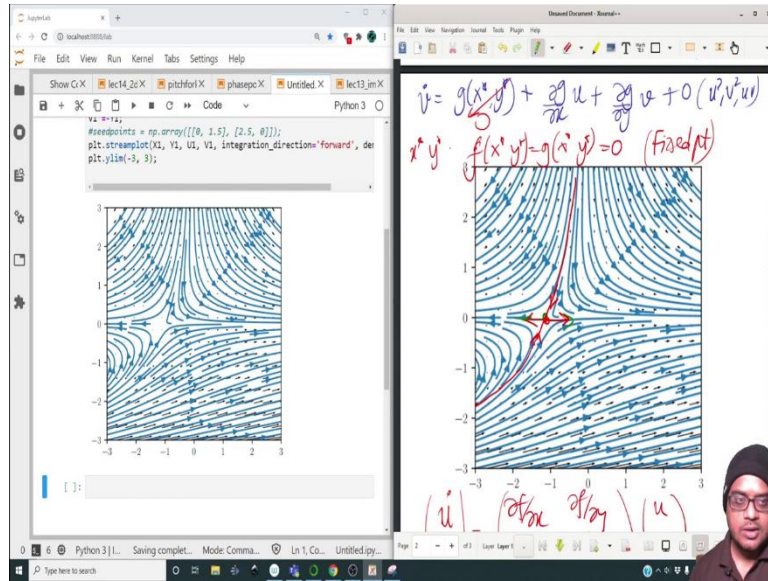
(Refer Slide Time: 13:16)



And, the corresponding eigenvector is going to be this corresponds to  $V_2 = 0$  and  $V_1 = V_1$  is an identity; the first eigenvector will be simply  $(1, 0)$ , second eigenvector will be  $\begin{bmatrix} 1 & -1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = - \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ . So, this eigenvector corresponds to  $\lambda = 1$  and the eigenvector correspondent to this will be. So, the other eigenvector will be  $(0.5, 1)$ , ok.

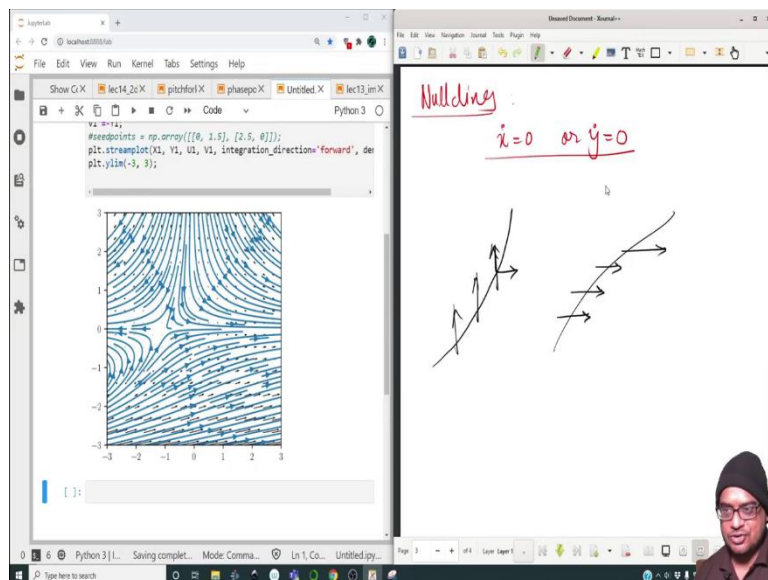
So, at equilibrium point that is  $(-1, 0)$ , one eigenvector is this and the corresponding eigenvalue is  $\lambda = 1$  because unstable at the same equilibrium point the other eigenvector is  $(0.5, 1)$  which is a slope like this and it is having an eigenvalue of  $-1$  that is it is an attracting manifold. So, this is the stable manifold, this is the unstable manifold.

(Refer Slide Time: 14:33)



So, over on this particular plot over at this point this is the linearization let me use red color. So, this is the linearization that will that we are; that we are looking at ok. So, the arrows will be like this, but because of the non-linear effects away from the fixed point it will tend to become curved and the curve really depends on what the nature of the equations are.

(Refer Slide Time: 15:07)



Another important aspect of these points rather of such curves they are called as nullclines. So, nullclines are curves in the phase portrait which correspond to either  $x$  equal to 0 or  $y$  equal to  $\dot{x} = 0$  or  $\dot{y} = 0$ , that is, they are demarcating curves where we know that there will be either no  $x$  velocity or no  $y$  velocity, ok.

So, if  $\dot{x} = 0$  we know that the flow at that point will be solely in the y direction when  $\dot{y} = 0$  we know that across that curve the velocity will be solely in the x direction. So, let us try to plot the nullclines as well, ok. It is quite easy to actually plot because we have all the x and y's.

(Refer Slide Time: 15:58)

The screenshot shows a JupyterLab notebook on the left and a presentation slide on the right. The notebook code is as follows:

```
[ ]: x = np.linspace(-3, 3, 20); y = np.linspace(-3, 3, 20);
X, Y = np.meshgrid(x, y);
U = X + np.exp(-Y);
V = -Y;
plt.quiver(X, Y, U, V);
ax = plt.gca(); ax.set_aspect(1);

x1 = np.linspace(-3, 3, 100); y1 = np.linspace(-3, 3, 100);
X1, Y1 = np.meshgrid(x1, y1);
U1 = X1 + np.exp(-Y1);
V1 = -Y1;
#seedpoints = np.array([[0, 1.5], [2.5, 0]]);
plt.streamplot(X1, Y1, U1, V1, integration_direction='forward', density=1);
plt.ylim(-3, 3);
plt.contour(X1, Y1, U1, levels=[0.0]);
plt.contour(X1, Y1, V1, levels=[0.0]);
```

The presentation slide on the right has the following handwritten text: "Nullclines:  $\dot{x}=0$  or  $\dot{y}=0$ ". Below the text are two diagrams of vector fields. The first diagram shows a vector field with a vertical nullcline at  $x=0$  and a horizontal nullcline at  $y=0$ . The second diagram shows a vector field with a vertical nullcline at  $x=0$  and a horizontal nullcline at  $y=0$ .

So, we simply need to plot `plt.contour X1, Y1, U1, V1` sorry `X1, Y1, U1` and levels equal to simply 0,0. So, this is the nullcline of  $\dot{x} = 0$ ; similarly we want the nullcline of  $\dot{y} = 0$  ok.

(Refer Slide Time: 16:28)

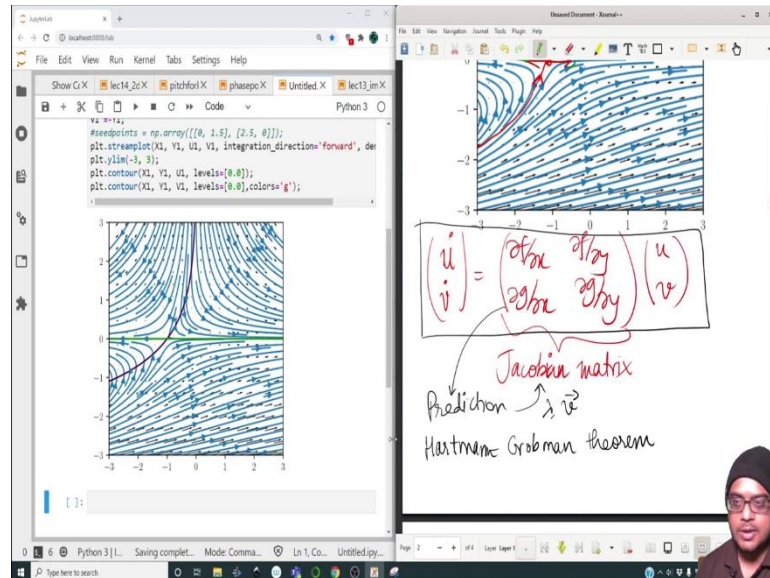
The screenshot shows a JupyterLab notebook on the left and a presentation slide on the right. The notebook code is as follows:

```
V1 = -Y1;
#seedpoints = np.array([[0, 1.5], [2.5, 0]]);
plt.streamplot(X1, Y1, U1, V1, integration_direction='forward', density=1);
plt.ylim(-3, 3);
plt.contour(X1, Y1, U1, levels=[0.0]);
plt.contour(X1, Y1, V1, levels=[0.0]);
```

The presentation slide on the right has the following handwritten text: "Nullclines:  $\dot{x}=0$  or  $\dot{y}=0$ ". Below the text are two diagrams of vector fields. The first diagram shows a vector field with a vertical nullcline at  $x=0$  and a horizontal nullcline at  $y=0$ . The second diagram shows a vector field with a vertical nullcline at  $x=0$  and a horizontal nullcline at  $y=0$ .

So, let us plot this ok. So, this is the nullcline of  $\dot{y} = 0$  and this is the nullcline sorry, let me put a different color ok.

(Refer Slide Time: 16:41)



So, this is the nullcline where  $\dot{y} = 0$ . So, the velocities are only in the x direction. So, along the green line obviously, the velocities are in the x direction while the purple line is the nullcline of  $u_1$  equal to 0 which corresponds to line in which there is only a y velocity and these are not the manifolds.

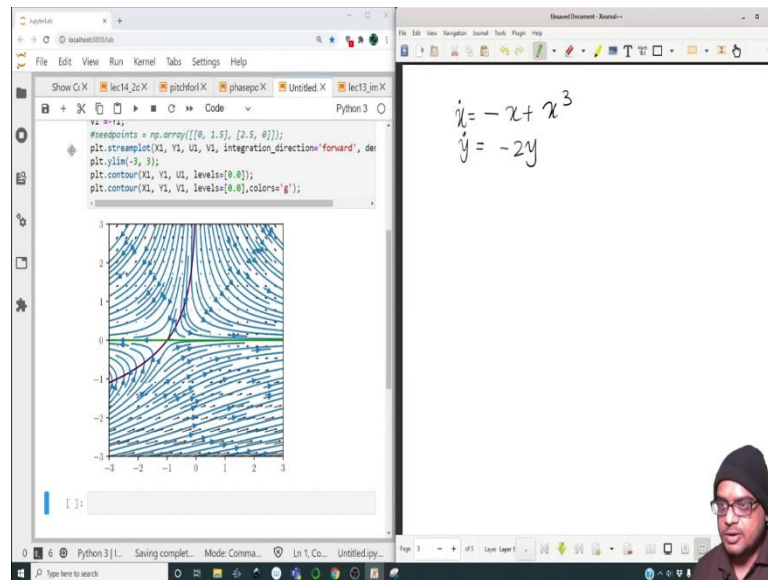
It is not to be confused with manifolds because those are something else altogether, but the nullclines give an idea where the trajectories will straighten or flatten out ok. So, in this particular example we have already seen what we expect. So, we have performed a simple linearization in order to ascertain and the linearization depends on the Jacobian.

So, then the question that people who study non-linear dynamics they ask is whether the structure of the Jacobian matrix dictates the entire picture? Whether the prediction using the Jacobian matrix that is the eigenvalues and eigenvectors are they strong enough to tell that yes, even the non-linear system behaves like that?

And, the answer is it depends, it depends whether you are having eigenvalues real and or even for a complex eigenvalue I mean whether or not you have a 0 real part or not. It all depends on the real part of the eigenvalue. We are not going to go deep into this, but for those of you are interested you can look up the Hartman-Grobman theorem.

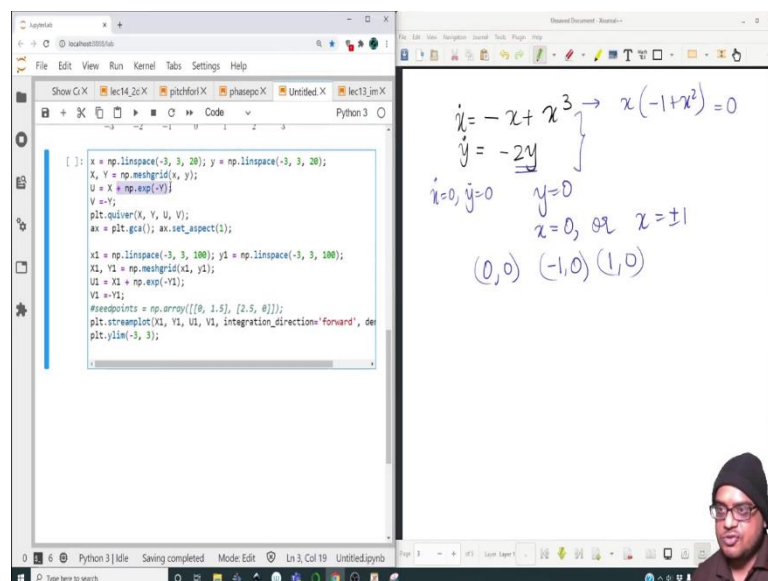
And, it tells you whether or not the linear picture near an equilibrium point is equivalent to the entire picture for the entire non-linear equation whether linearization is good enough to make predictions ok. So, for now we will continue onward and we will try to have a look at a different set of equations.

(Refer Slide Time: 19:15)



So, let us consider  $\dot{x} = -x + x^3$  and  $\dot{y} = -2y$  ok. So, before anything let us quickly see how these equations look like.

(Refer Slide Time: 19:29)

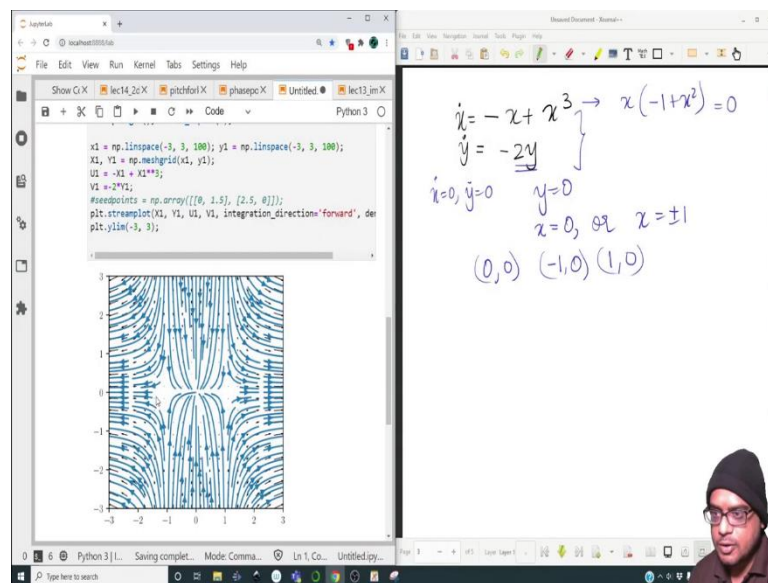




We do not really care about the nullclines for this particular equation. But, even before starting we should be in a position to say how many fixed points does this equation have. So, the fixed points are where  $\dot{x} = 0$  and  $\dot{y} = 0$ , obviously, it corresponds to  $y=0$  and  $x=0$  comma.

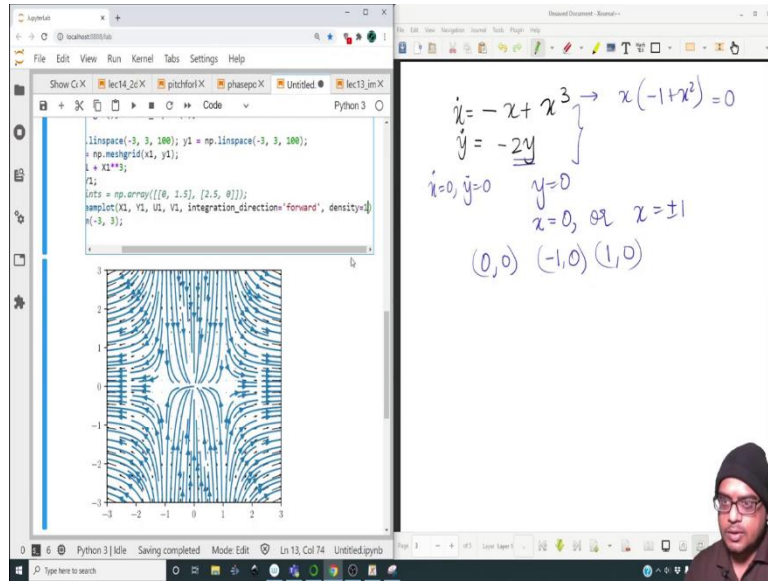
So, this equation is  $x(-1 + x^2)$  and if it is equal to 0  $x = 0$  or  $x = \pm 1$ . So, we have three fixed points –  $(0, 0)$ ,  $(-1, 0)$  and  $(1, 0)$ , alright. Let us see whether this checks out or not ok.

(Refer Slide Time: 20:19)



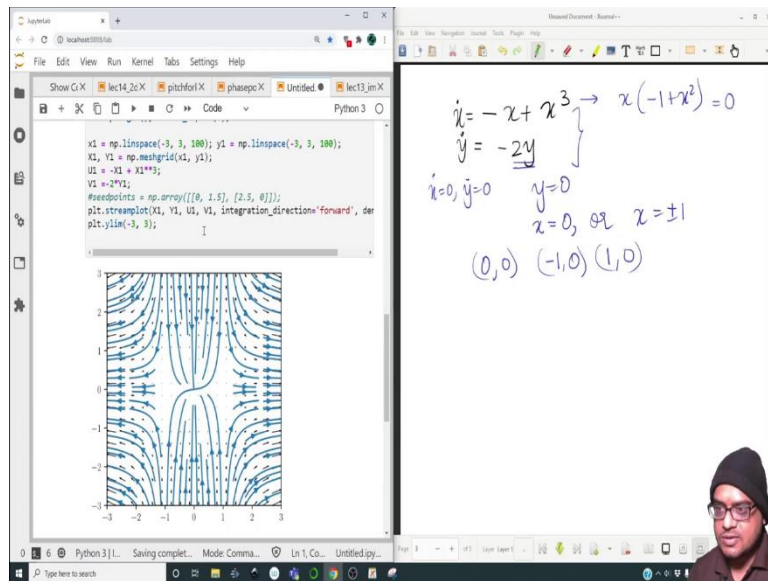
So, when we look at this particular plot we see that the trajectories or the stream lines are being attracted towards the origin while we do have some kind of a hyperbolic flow happening around  $1, 0$  and  $-1, 0$ , ok.

(Refer Slide Time: 20:40)



So, in fact, let me keep these particular streamlines as well along with this let me plot the trajectories.

(Refer Slide Time: 20:43)



So, let me copy a snippet that we had done in one of the previous example ok.

(Refer Slide Time: 20:53)

The screenshot shows a Jupyter Notebook on the left and a whiteboard on the right. The notebook code is as follows:

```

from scipy.integrate import solve_ivp

def showtraj(x0 = -2, y0 = 2):
    x = np.linspace(-3, 3, 20); y = np.linspace(-3, 3, 20);
    X, Y = np.meshgrid(x, y);
    U = X + Y;
    V = 4*X - 2*Y;
    plt.quiver(X, Y, U, V);
    ax = plt.gca(); ax.set_aspect(1);

def mysys(t, x): # returns the RHS
    return [-x[0] + x[0]**3, -2*x[1]];

tspan = [0, 1]
x0 = -2.5; y0 = 2.5
ics = [x0, y0]
sol = solve_ivp(mysys, tspan, ics, dense_output=True);

tout = np.linspace(0, np.max(tspan), 100);
xout = sol.sol(tout)[0];
yout = sol.sol(tout)[1];
plt.plot(xout, yout);
plt.xlim(-3, 3); plt.ylim(-3, 3);
plt.plot(x1, y1, '-k');
plt.plot(x2, -4*y2, '-k');
plt.plot(x0, y0, 'ok');
w = interactive(showtraj, x0 = (-2.5, 2.5, 0.05), y0 = (-2.5, 2.5,

```

The whiteboard contains the following handwritten text:

$$\dot{x} = -x + x^3 \rightarrow x(-1+x^2) = 0$$

$$\dot{y} = -2y$$

Equilibrium points are listed as  $(0,0)$ ,  $(-1,0)$ , and  $(1,0)$ . Additional notes include  $x=0, \text{ or } x=\pm 1$  and  $y=0$ .

So, we need this. We need this, alright.

(Refer Slide Time: 21:07)

The screenshot shows a Jupyter Notebook on the left and a whiteboard on the right. The notebook code is as follows:

```

V1 = 2*Y1;
#seedpoints = np.array([[0, 1.5], [2.5, 0]]);
plt.streamplot(X1, Y1, U1, V1, integration_direction='forward', density=0.5);
plt.ylim(-3, 3);

def mysys(t, x): # returns the RHS
    return [-x[0] + x[0]**3, -2*x[1]];

tspan = [0, 5]
x0 = -2.5; y0 = 2.5
ics = [x0, y0]
sol = solve_ivp(mysys, tspan, ics, dense_output=True);

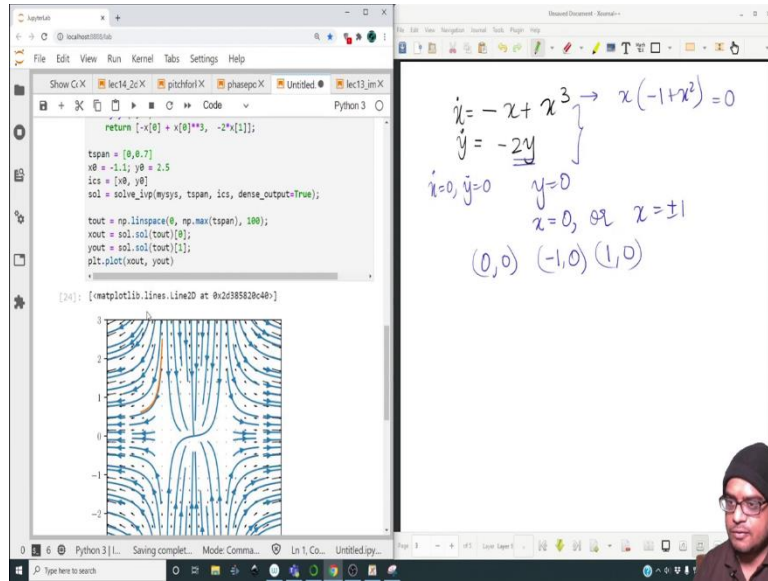
tout = np.linspace(0, np.max(tspan), 100);
xout = sol.sol(tout)[0];
yout = sol.sol(tout)[1];
plt.plot(xout, yout);

```

The whiteboard contains the same handwritten text as the previous slide.

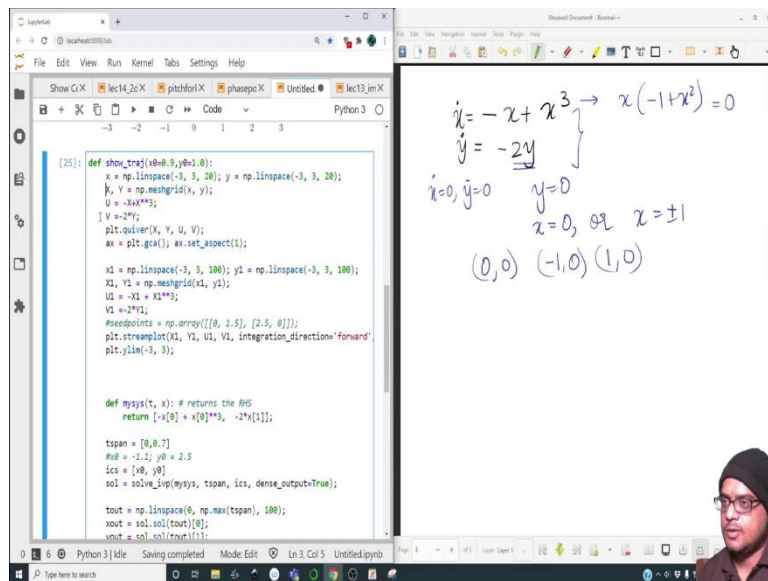
So, let us take that and so, tspan is going from say 0 to 5 the initial points let us keep this. We have to change the function, so, it will be - x plus x cube. So, it is - this plus x0 cubed and this will be - 2y and this is already set to - 2y; so, plt.plot(xout, yout), ok.

(Refer Slide Time: 21:55)



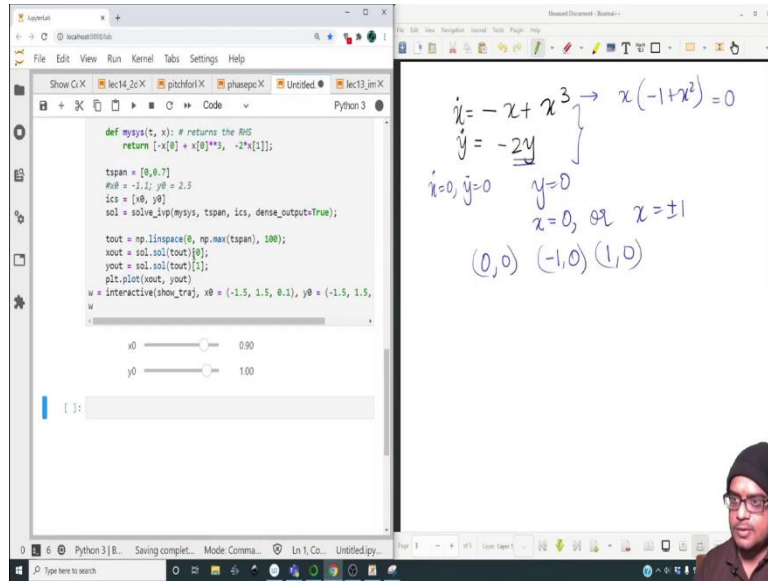
So, we have written on the code I have changed the tspan, so that the trajectories are not flying off to infinity or - infinity and let us wrap this entire thing inside a function ok. Let us try to put all of this inside a function, so that we can probe the different initial conditions.

(Refer Slide Time: 22:24)



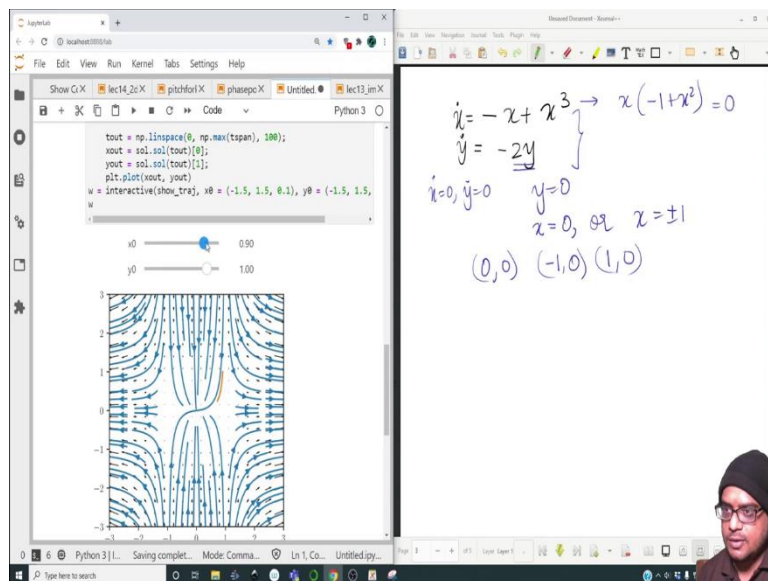
So, show traj  $x_0, y_0$ , we will wrap all of this  $x_0$  let us give some default values ok that is fine. Let us comment out this particular line, alright.

(Refer Slide Time: 22:54)



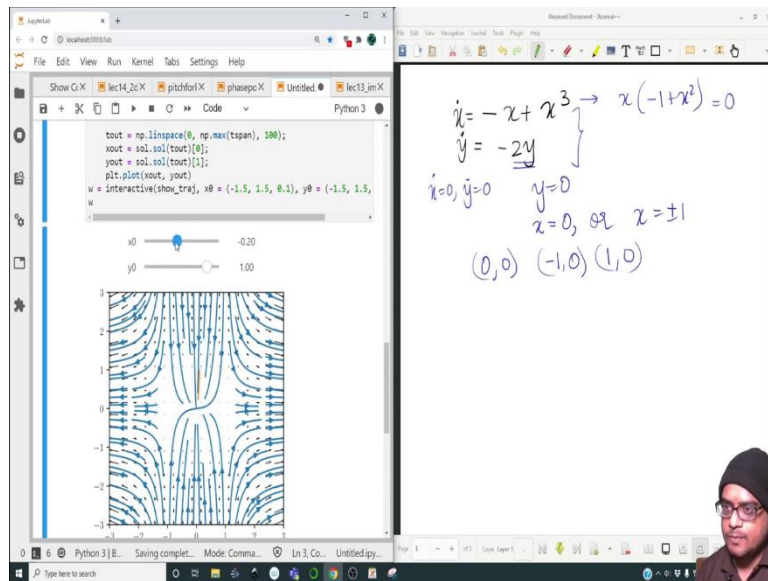
Let us create an interactive digit - 1.5 to 1.5 in steps of 0.1 y naught will be the same and we will display the result; so here, alright.

(Refer Slide Time: 23:29)



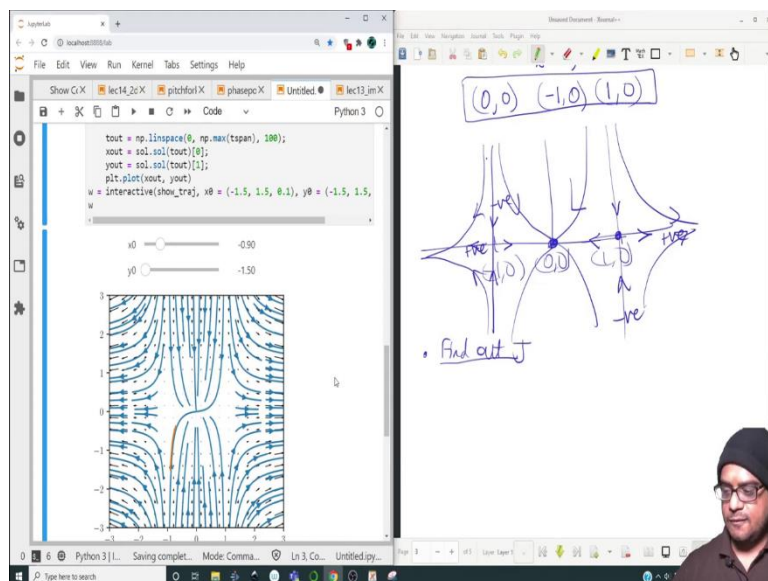
Let us go ok. So, the trajectory starts over here.

(Refer Slide Time: 23:34)



As we change this, we see that the trajectory can also be shown.

(Refer Slide Time: 23:41)



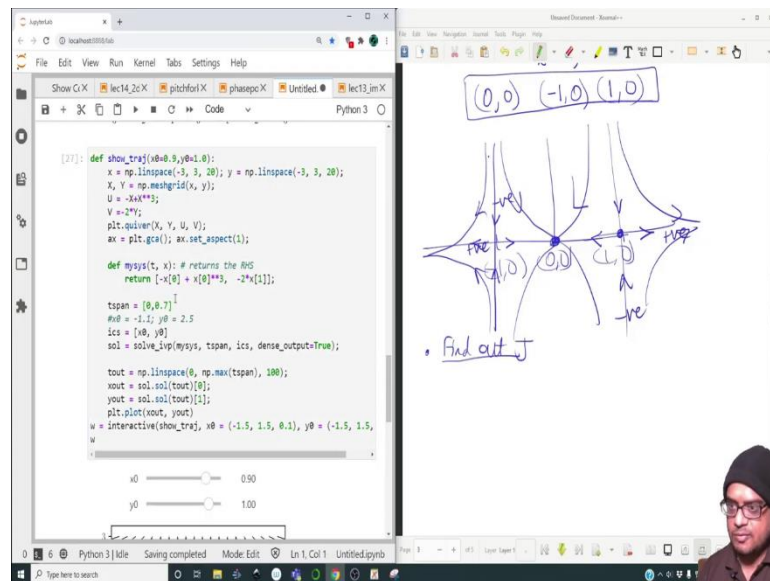
And, it crosses off to infinity. It is not as entirely feasible to have long trajectories ok. So, it is quite obvious that trajectories are flowing in this particular manner ok. So, they are being attracted towards this and they are repelled away from this ok. So, how can we find out whether I mean from theory what the nature of these points are ok.

So, we can do the following I am not going to do it for you find out the Jacobian for all these points and then based on that find out the eigenvalues and eigenvectors. So, it should

be clear that the eigenvalues one will be negative this incoming one and one will be positive this one ok.

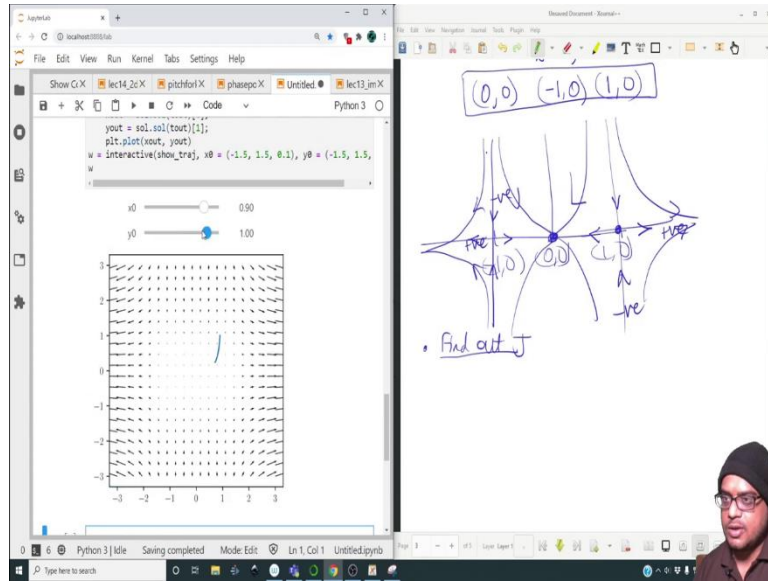
So, this is a stable node. All the trajectories are converging towards this in the vicinity at least and this is again an unstable load ok. So, this eigenvector should have a negative eigenvalue which should have a positive eigenvalue, positive eigenvalue, negative eigenvalue. So, you should be able to figure this out with the help of the Jacobian ok.

(Refer Slide Time: 25:30)



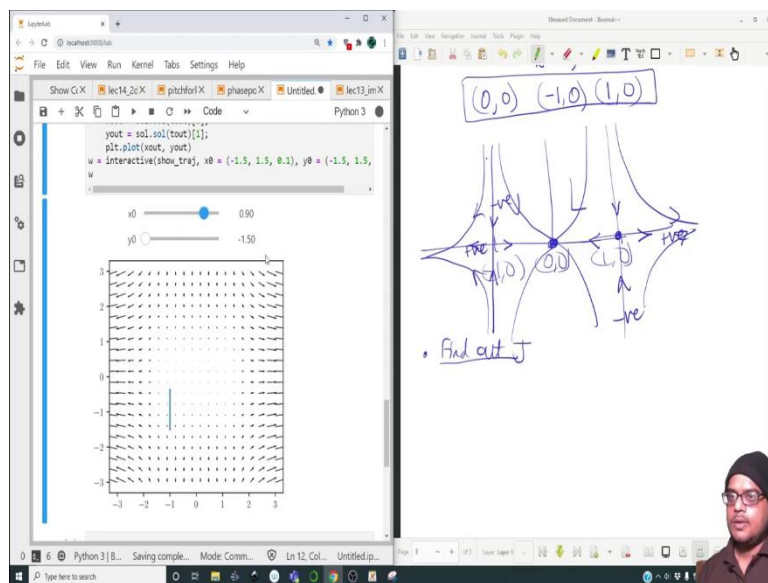
So, let us now move on to the third example where we will see certain case where such kinds of things may or may not always work the linearization where it will fail, so to say. So, actually we do not need all these stream plots we can make to with only the quiver plot ok. So, let me remove all these. We do not need sorry, we need the quiver plot we do not need the stream plot. So, we get rid of this. We have the trajectory solver let me keep time equal to 1 and let me just see whether I have broken something or not ok.

(Refer Slide Time: 26:06)



I have not broken something, it works well, ok.

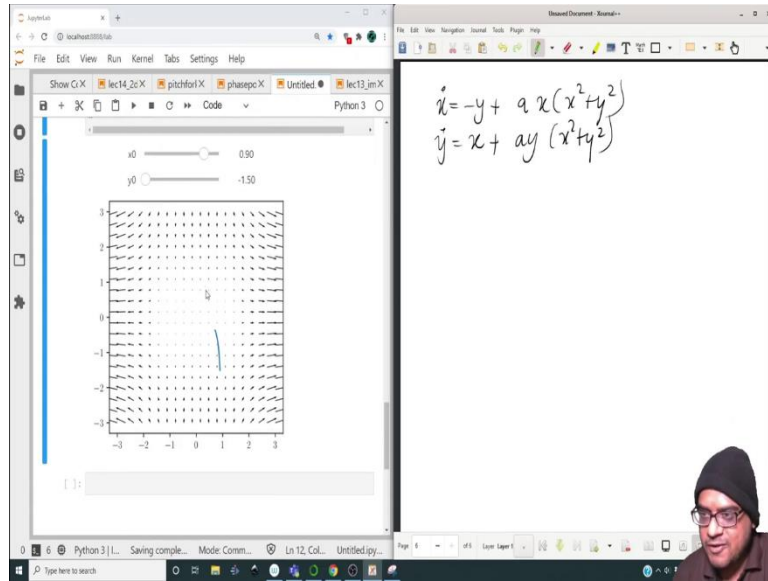
(Refer Slide Time: 26:08)



These are the trajectories great.

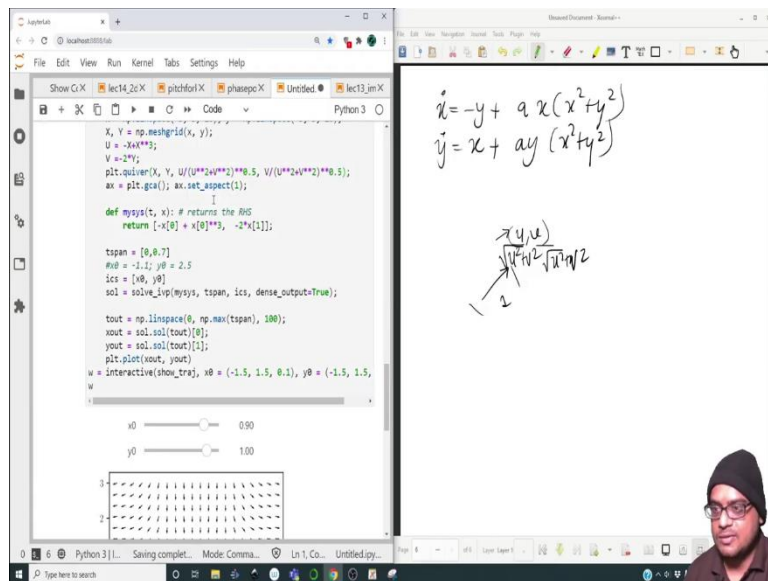
(Refer Slide Time: 26:15)





So, let us consider this particular problem. So, just for the record the previous problem was rather easy to figure out and you should be able to find out the Jacobian and the interpretation of the Jacobian with respect to the previous problem.

(Refer Slide Time: 26:34)

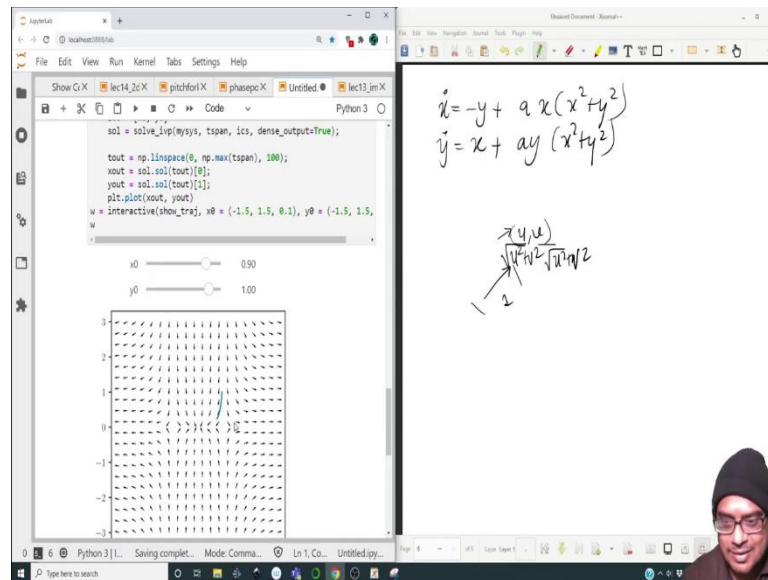


Let us consider this problem. So,  $\dot{x} = -y + ax(x^2 + y^2)$ ;  $\dot{y} = x + ay(x^2 + y^2)$ . So, this is an example from Strogatz again, but it serves as a very important example where linearization does not always provide a good insight into the problem.

But, before that the vectors that we see over here are quite small. In fact, let us fix that, we need to normalize the vectors. So, if you have a vector like this  $u, v$  in order to make the

magnitude of this vector 1. Ok the magnitude has to be 1 is simply divided by square root of u squared plus V square and square root of u square plus V square' basically, divided by the magnitude in order to render it as a unit vector. So, divided by square root of u square plus V square 0.5, let me copy this and let me paste this.

(Refer Slide Time: 27:47)



So, let us see ok. So, the arrows look much better you can sort of visualize things in a better fashion. So, let us change the function, so that we have this function over here ok. So, this will be - Y plus a times X times X square plus Y square and this will be X plus a times Y times X square plus Y square, ok.

(Refer Slide Time: 28:01)

The screenshot shows a Jupyter Notebook interface with the following Python code:

```
[28]: def show_traj(x0=0.5, y0=1.0, a = 1.0):
x = np.linspace(-3, 3, 20); y = np.linspace(-3, 3, 20);
X, Y = np.meshgrid(x, y); I
U = -y*a*(x**2+y**2);
V = X + a*(x**2 + y**2);
plt.quiver(X, Y, U/(U**2+V**2)**0.5, V/(U**2+V**2)**0.5);
ax = plt.gca(); ax.set_aspect(1);

def mysys(t, x): # returns the RHS
return [-x[0]**3, -2*x[1]];

tspan = [0, 0.7]
x0 = -1.5; y0 = 2.5
ics = [x0, y0]
sol = solve_ivp(mysys, tspan, ics, dense_output=True);

tout = np.linspace(0, np.max(tspan), 100);
xout = sol.sol(tout)[0];
yout = sol.sol(tout)[1];
plt.plot(xout, yout);
u = interactive(show_traj, x0 = (-1.5, 1.5, 0.1), y0 = (-1.5, 1.5, 0.1))
```

The whiteboard contains the following equations and diagram:

$$\dot{x} = -y + a x (x^2 + y^2)$$

$$\dot{y} = x + a y (x^2 + y^2)$$

The diagram shows a vector field in the  $xy$ -plane. The horizontal axis is labeled  $x$  and the vertical axis is labeled  $y$ . A vector is drawn from the origin, pointing into the first quadrant. The vector is labeled with the expression  $\frac{x y u}{\sqrt{u^2 + v^2}}$ , where  $u$  and  $v$  are the components of the vector field.

So, let us pass  $a$  as a parameter to the problem let it be 1.0 at the outset.

(Refer Slide Time: 28:39)

The screenshot shows a Jupyter Notebook interface with the following Python code:

```
[29]: a = 1.0;
20); y = np.linspace(-3, 3, 20);
);
);
2);
24**2)**0.5, V/(U**2+V**2)**0.5);
aspect(1);

ans the RHS
3]**3, -2*x[1]);

tspan, ics, dense_output=True);

p_max(tspan, 100);
);
);

x0 = (-1.5, 1.5, 0.1), y0 = (-1.5, 1.5, 0.1), a = (0.8, 1.2, 0.05)
```

The whiteboard contains the following equations and diagram:

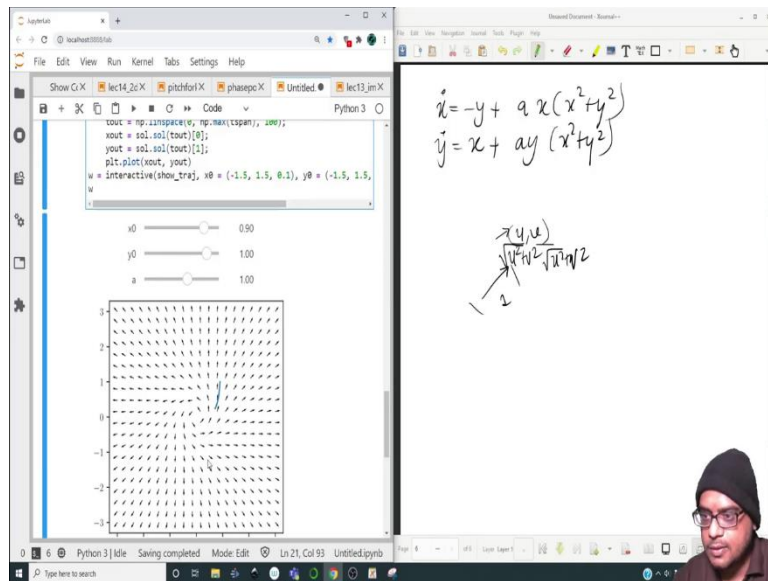
$$\dot{x} = -y + a x (x^2 + y^2)$$

$$\dot{y} = x + a y (x^2 + y^2)$$

The diagram shows a vector field in the  $xy$ -plane. The horizontal axis is labeled  $x$  and the vertical axis is labeled  $y$ . A vector is drawn from the origin, pointing into the first quadrant. The vector is labeled with the expression  $\frac{x y u}{\sqrt{u^2 + v^2}}$ , where  $u$  and  $v$  are the components of the vector field.

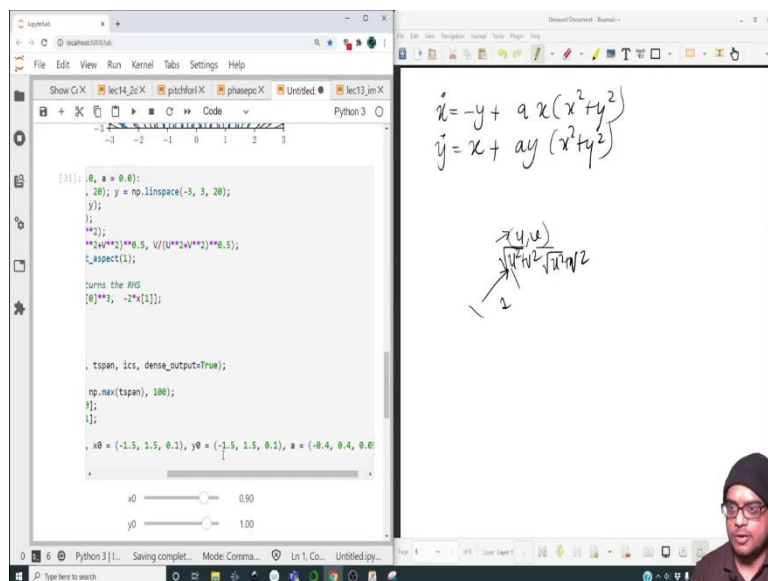
So,  $a$  will go from 0.8 to 1.2 in steps of 0.05. Let us run this, ok.

(Refer Slide Time: 28:50)



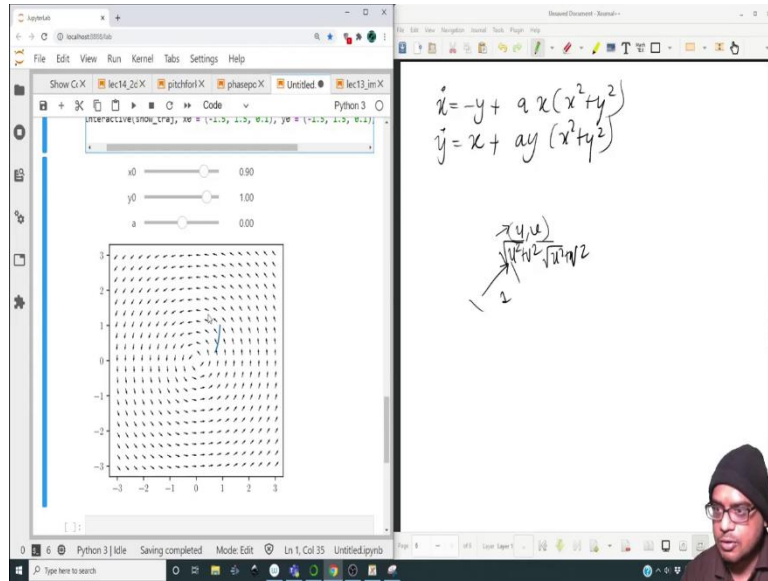
So, when  $a$  is equal to 1.

(Refer Slide Time: 29:01)



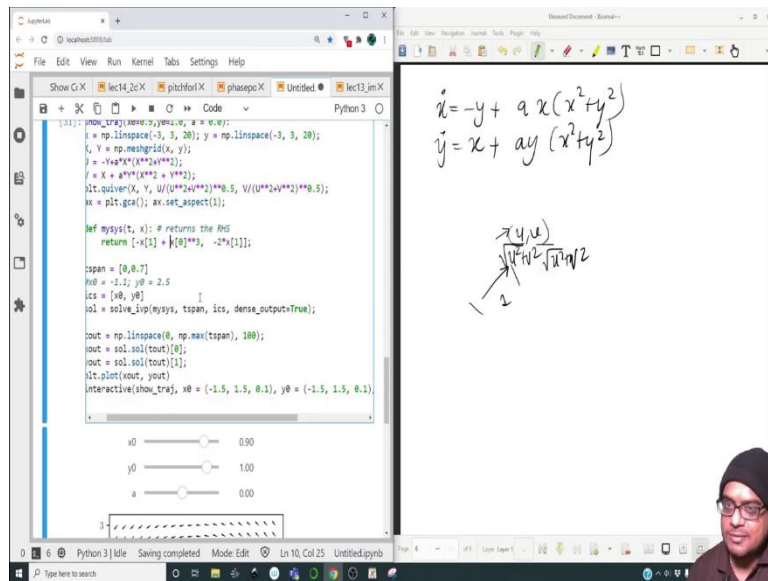
Actually, it has to go from let us say it goes from - 0.4 to 0.4, that is when the fundamental change in behavior will happen ok. So, this let it be 0 as the default value ok.

(Refer Slide Time: 29:18)

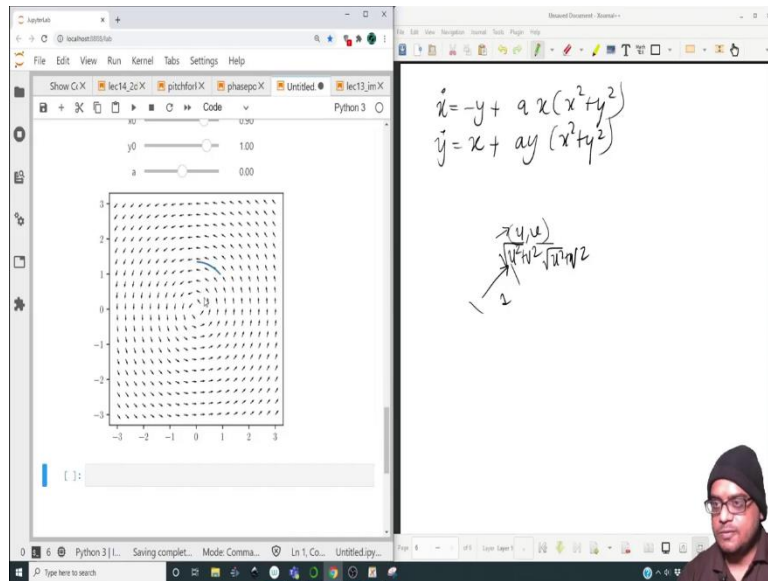


So, when it is 0, you see that the trajectory is tend to look like a concentric line and I have forgotten to change the equations which the initial value problem solver will use.

(Refer Slide Time: 29:34)

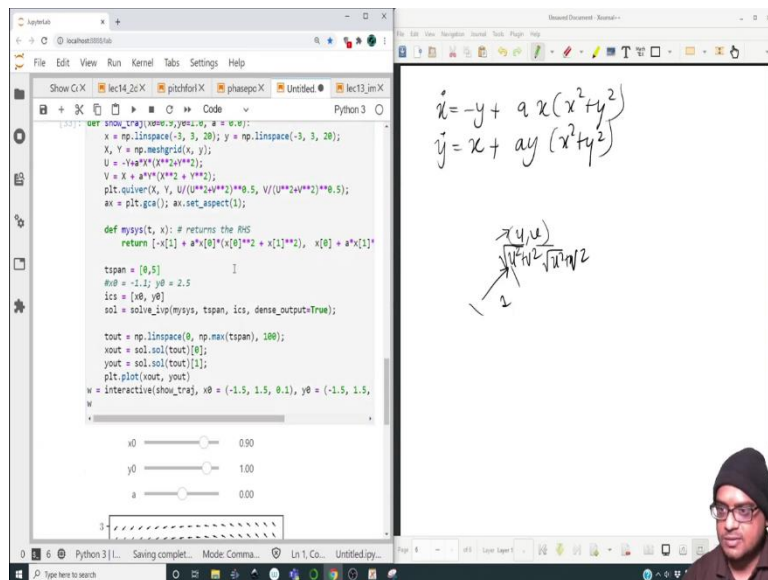


(Refer Slide Time: 29:36)

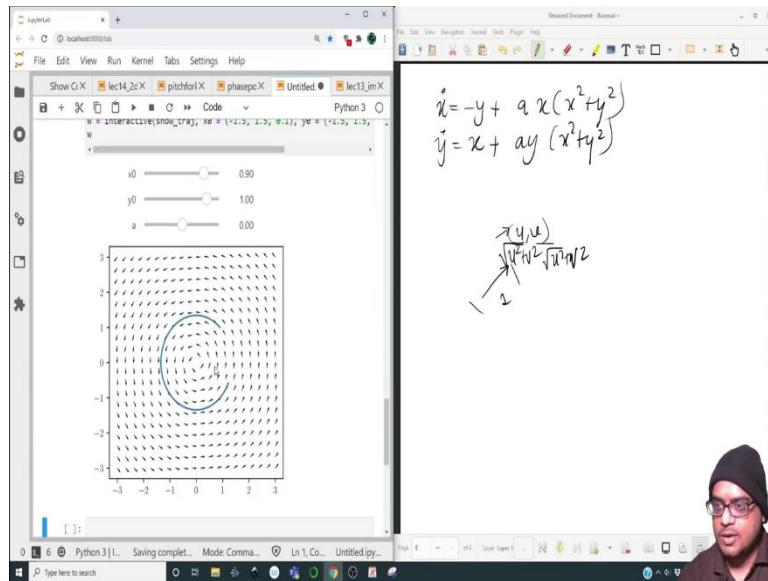


So, let us quickly fix that ok. So, I fixed it. The time span can be increased to say 5 seconds ok.

(Refer Slide Time: 29:42)

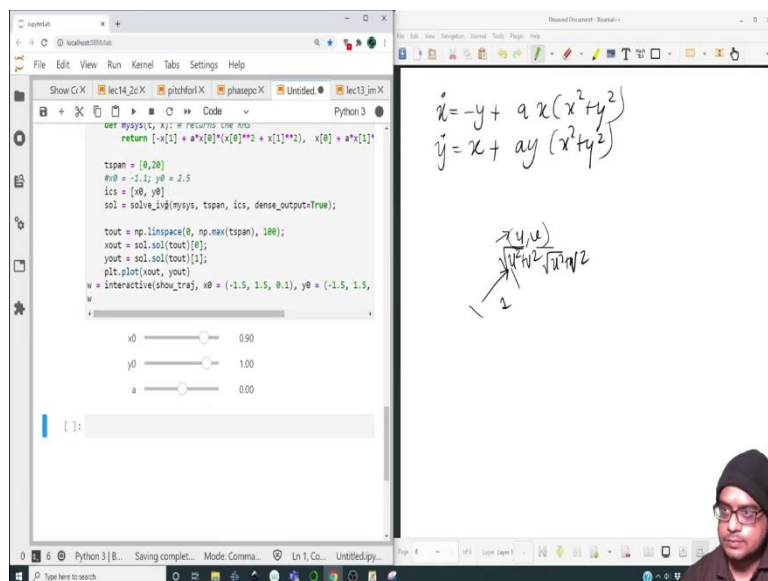


(Refer Slide Time: 29:43)

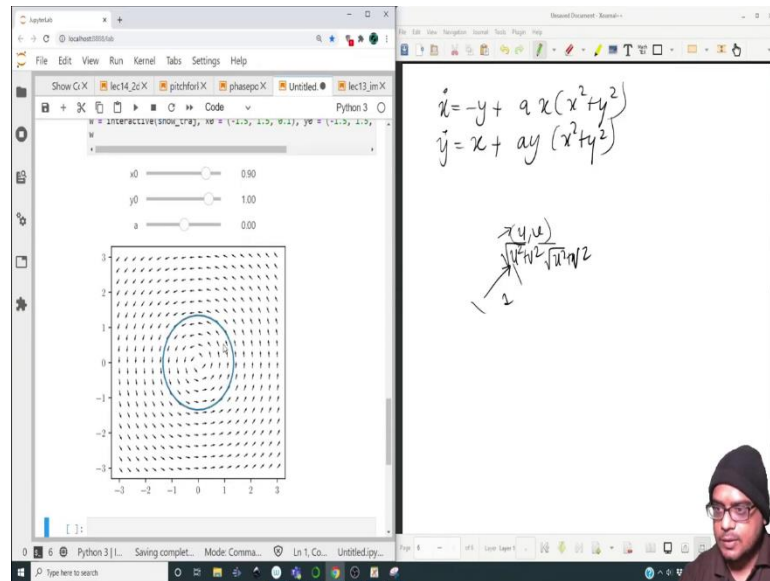


So, it does look like it is going to end up being a closed contour, alright.

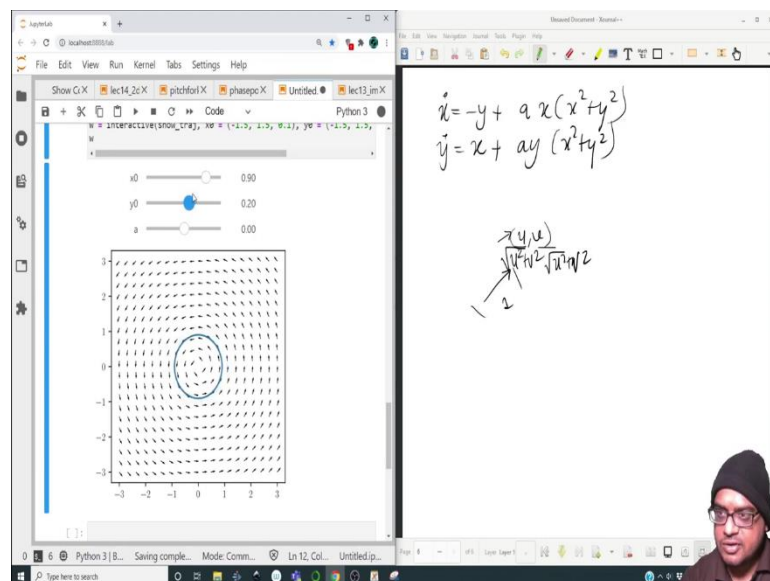
(Refer Slide Time: 29:52)



(Refer Slide Time: 29:53)



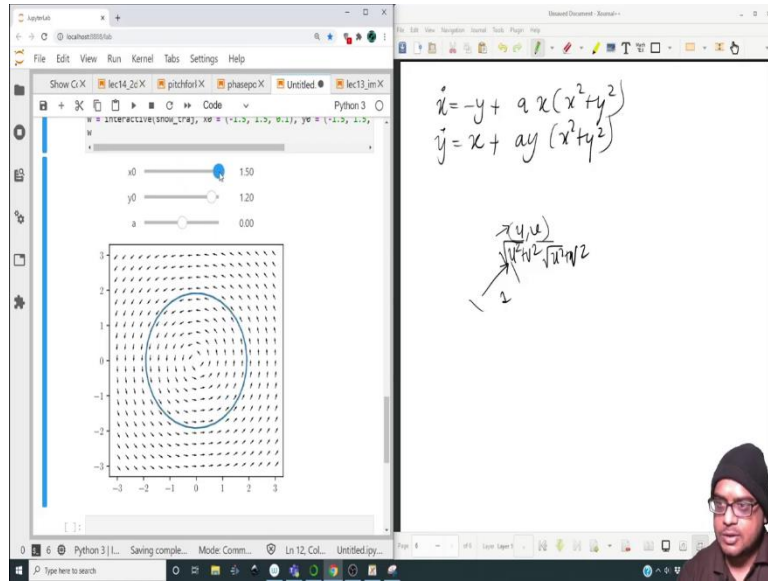
(Refer Slide Time: 29:57)



So, it does appear to be a closed contour let me change it and for all different initial conditions it does appear to be a closed contour ok.

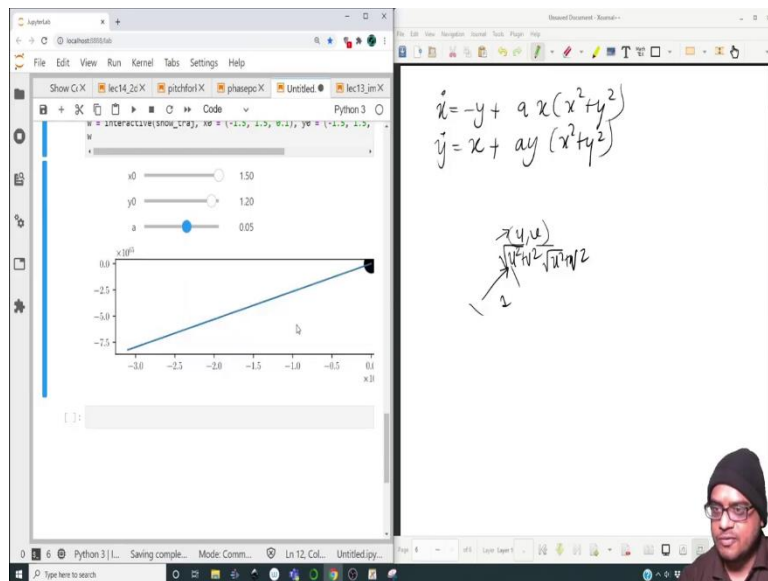
(Refer Slide Time: 30:01)





So, now, what happens when I change a ever so slightly? I am changing it to 0.5.

(Refer Slide Time: 30:08)



And, it has blown away to infinity.

(Refer Slide Time: 30:15)

The Jupyter Notebook code is as follows:

```

def mysys(t, x):
    return [-x[1] + a*x[0]**2 + x[1]**2, x[0] + a*x[1]**2]

tspan = [0, 2]
#x0 = -1.5; y0 = 2.5
ics = [x0, y0]
sol = solve_ivp(mysys, tspan, ics, dense_output=True);

tout = np.linspace(0, np.max(tspan), 100);
xout = sol.sol(tout)[0];
yout = sol.sol(tout)[1];
plt.plot(xout, yout)
w = Interactive(show_traj, x0 = (-1.5, 1.5, 0.1), y0 = (-1.5, 1.5,

```

The whiteboard contains the following handwritten content:

$$\dot{x} = -y + a x(x^2 + y^2)$$

$$\dot{y} = x + a y(x^2 + y^2)$$

Below the equations is a diagram showing a vector field with arrows pointing outwards from the origin, labeled with  $\sqrt{x^2 + y^2}$  and  $\sqrt{1 + a^2}$ .

So, let me change the time span. Let me make it only 2 seconds.

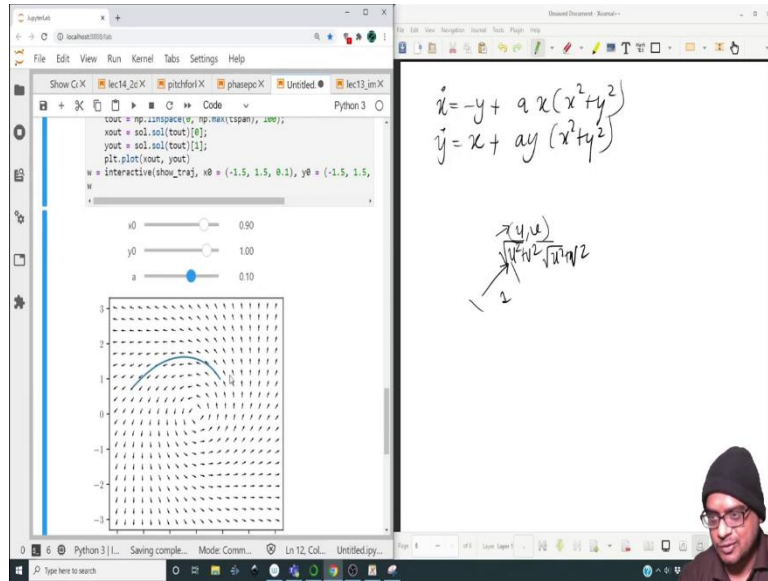
(Refer Slide Time: 30:17)

The Jupyter Notebook shows a phase plot of the system of differential equations. The plot displays a vector field with arrows and a trajectory starting from the origin and moving outwards. The axes range from -3 to 3.

The whiteboard content is identical to the previous slide, showing the differential equations and the vector field diagram.

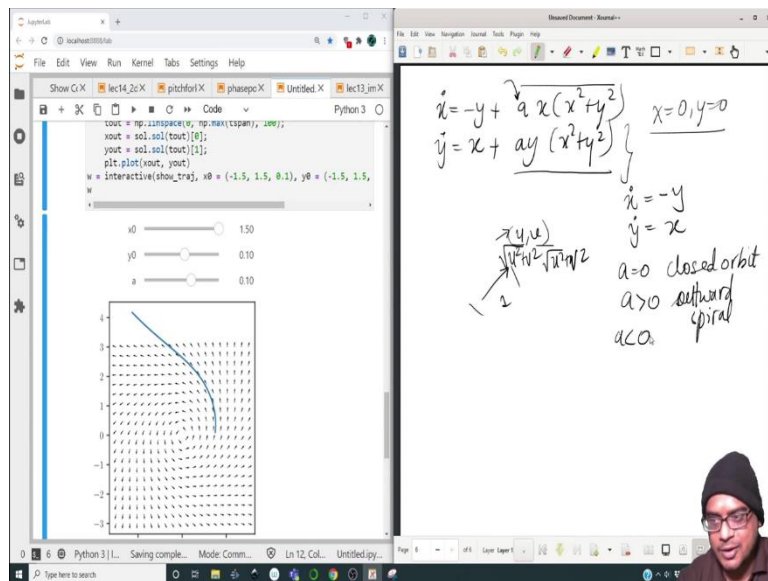
In fact, can you see what happens when  $a$  becomes positive?

(Refer Slide Time: 30:23)



The initial point is trying to spiral out.

(Refer Slide Time: 30:29)

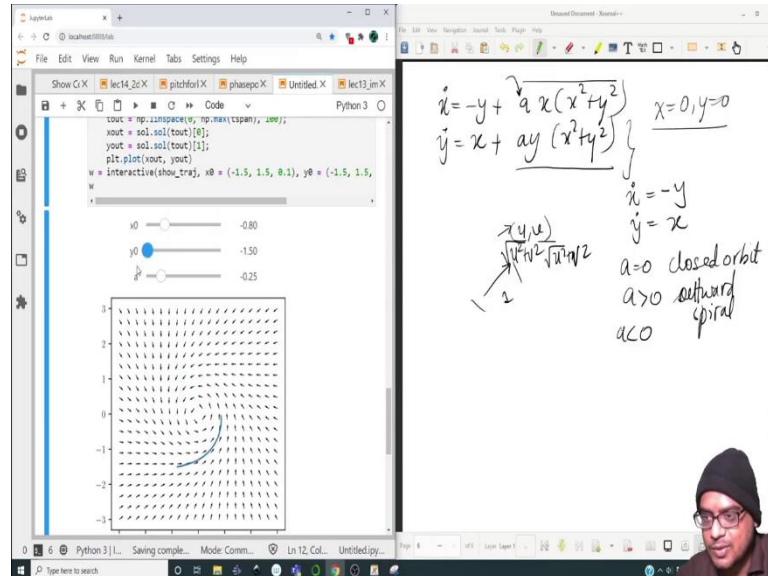


It is not trying to close into a loop. You can clearly see it when I choose the initial point quite far from this from the equilibrium point ok; so, first things first. Equilibrium point is  $x = 0, y = 0$ , this is a fixed point of the system. Now, when we linearize this ok when you linearize this you are obviously, going to ignore the cubic terms.

So, this term and this term they are the cubic terms in the absence of cubic terms it becomes an equation which resembles this. So, obviously, when you linearize the equation you are losing out on the information which is contained by this particular control parameter ok.

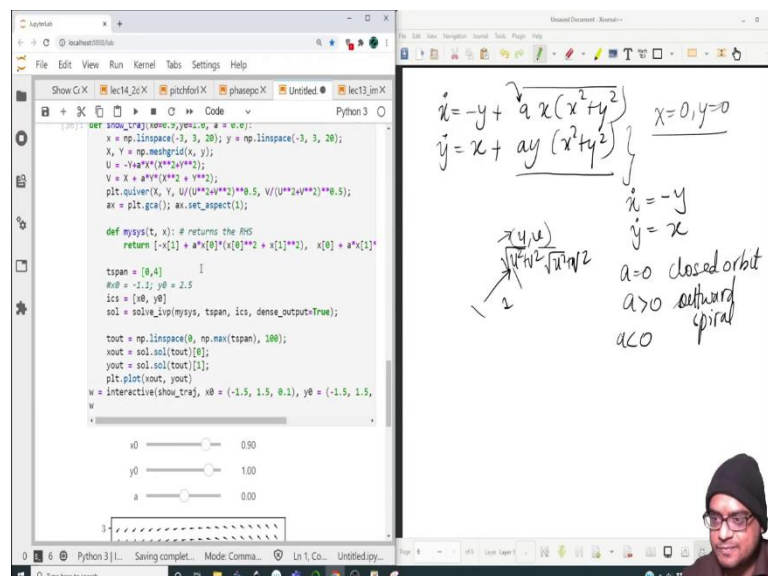
So, when  $a = 0$  we have seen that it is a closed orbit when  $a > 0$ , we see that it is a outward spiral and when  $a$  equal to 0. Can I guess what will happen?

(Refer Slide Time: 31:31)



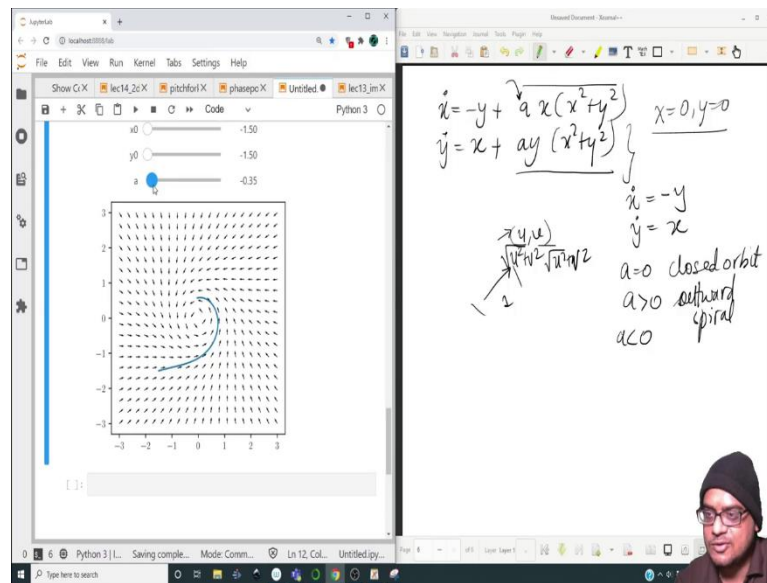
If you have guessed it becomes an attracting spiral you are correct, ok. So, it spirals in into the solution rather into the problem.

(Refer Slide Time: 31:49)



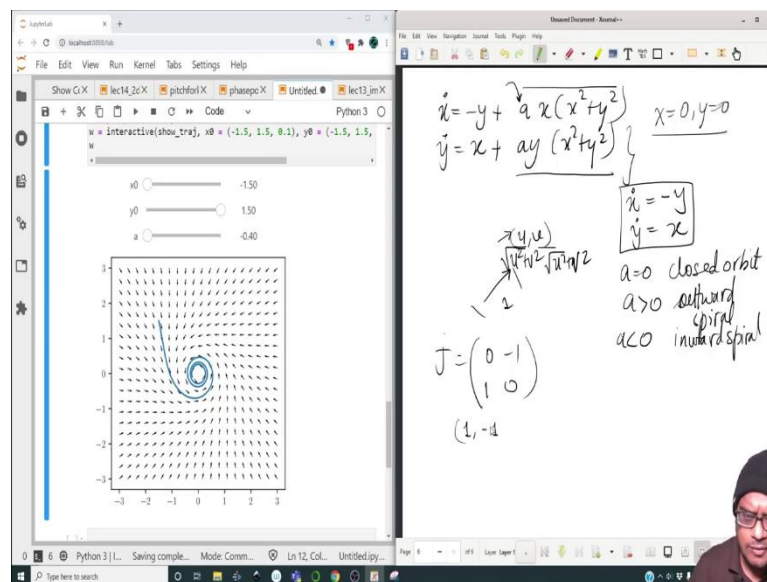
Let me just increase the time span a bit so that spiraling in is rather obvious ok.

(Refer Slide Time: 31:51)



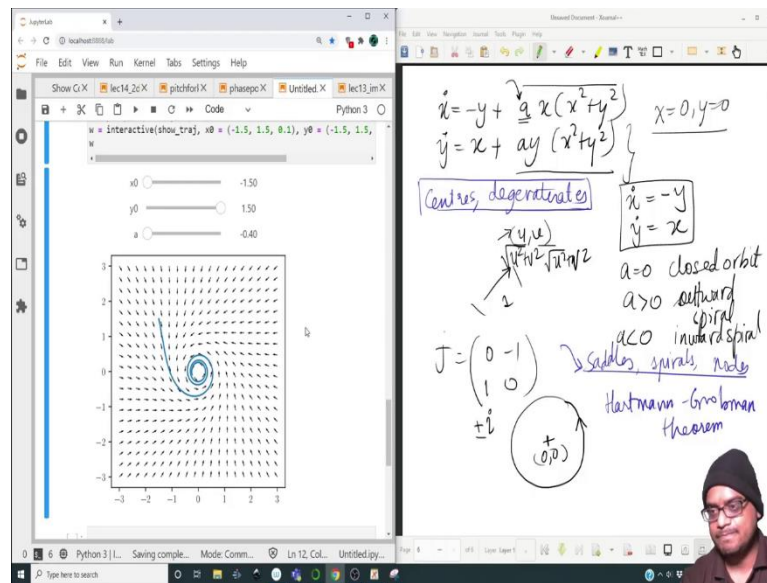
So, it is spiraling into the origin. Let me increase the time span even further. So, we know that it is spiraling in. So, nothing is going to change a lot ok.

(Refer Slide Time: 32:15)



So, look, it is spiraling in and looks very nice alright. So, it becomes an inward spiral, but when you linearize the system you only get a system for Jacobian is  $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ . Now, obviously, this has eigenvalues of 1, -1 rather it will not have one common answer, it will be purely imaginary ok.

(Refer Slide Time: 32:47)



So, the eigenvalues will be  $\pm i$  and the eigenvalues indicate it will be a closed orbit because it is purely imaginary. So, this is the fixed point  $0, 0$  and if you linearize the equation it will always predict it is going to be a closed orbit ok. It is going to be a closed loop, but the dependency on  $a$  has been washed away by the fact that we have linearized the system. If we do not pay attention to the entire nature of the equation, it will not work.

But, again for these particular cases for centers so, this is called the center because it is like a center for a closed orbit. So, for degenerate points and such points linearizing does not give you the entire picture, but for saddles or spirals or what do we have, nodes. So, for these particular cases the linearization does give you an accurate picture of what the equation is going to be happening ok.

So, go ahead and have a look through some other NPTEL courses where they discuss deeply about the non-linear dynamics part. All whatever I have discussed is a part of the Hartman-Grobman theorem and that theorem tells you that you can distort the space for these particular geometries or vector flows.

And, you can obtain a nice solution, but you cannot obtain the behavior using linearization for these kind of particular cases, but regardless of that Python does give you or Pythagorean octave does give you an ideal tool set to really change the value of the control parameter and see what difference it brings into the picture, alright.

So, this is how we can sort of look into the problem without really solving the problem, but of course, once you do have the tools it must be clear that your analysis is of utmost importance. What you can infer from these diagrams anyone? So, with the help of whatever we are discussing you should be now in a position to tell that this is happening or that is happening, but then what physical significance does it lie that is all up to you. Let us now proceed to another set of problems in which we have tractor or an oscillator.

(Refer Slide Time: 35:46)

The image shows a Jupyter Notebook with two panes. The left pane contains Python code for solving a differential equation and plotting the trajectory. The right pane contains handwritten mathematical notes for a conservative system.

**Handwritten notes (Right Pane):**

Conservative system

$$m\ddot{x} = F$$

$$F = -\frac{\partial V}{\partial x}$$

$$V = -\frac{1}{2}x^2 + \frac{1}{4}x^4$$

$$F = -(-x + x^3) = x - x^3$$

$$\ddot{x} = -x + x^3$$

$$\dot{x} = y$$

$$\dot{y} = -x + x^3$$

**Python Code (Left Pane):**

```
def show_traj(x0=0.5, y0=1.0, a = 0.0):
    x = np.linspace(-3, 3, 20); y = np.linspace(-3, 3, 20);
    X, Y = np.meshgrid(x, y);
    U = Y;
    V = -0.5*X**2;
    plt.quiver(X, Y, U((U**2+V**2)**0.5), V((U**2+V**2)**0.5));
    ax = plt.gca(); ax.set_aspect(1);

def mysys(t, x): # returns the RHS
    return ([x[1], -(x[0]+x[0]**3)]);

tspan = [0, 20]
#x0 = -1.1; y0 = 2.5
ics = [x0, y0]
sol = solve_ivp(mysys, tspan, ics, dense_output=True);

tout = np.linspace(0, np.max(tspan), 100);
xout = sol.sol[tout][0];
yout = sol.sol[tout][1];
plt.plot(xout, yout)

w = interactive(show_traj, x0 = (-1.5, 1.5, 0.1), y0 = (-1.5, 1.5,
```

So, in physics we have what is called as a conservative system and typically it is defined as  $m\ddot{x} = F$  and that will be equal to some force, but this force has if it is conservative then it can be given as the negative of some negative gradient of some potential ok. So, let us consider the potential  $V = -1/2x^2 + 1/4x^4$ . Let us consider this particular potential ok. So, what is the force? The force is going to be  $-x + x^3$ .

So, once we substitute this over here we have. So, let us forget about the m for now. So,  $\ddot{x} = -x + x^3$ . So, let  $v = \dot{x}$  alright y be equal to this and so,  $\dot{y} = -x + x^3$  ok. So, let me change the particular flows that we have over here. So, this is going to be  $-x + x^3$  ok. So, we have encoded the entire problem. So, let us now run this and see what happens ok.

(Refer Slide Time: 37:29)

When we run this everything is blown over to infinity. So, why is that? So, the reason is I have forgotten to take the - sign over here. So, f is - dV dx. So, this actually becomes  $x - x^3$ .

(Refer Slide Time: 37:45)

So, this will change and this will change.

(Refer Slide Time: 37:52)



Conservative system

$$m\ddot{x} = F = -\frac{\partial V}{\partial x}$$

$$V = -\frac{1}{2}x^2 + \frac{1}{4}x^4$$

$$F = -(x - x^3) = x - x^3$$

$$\ddot{x} = +x - x^3$$

$$\dot{x} = y$$

$$\dot{y} = +x - x^3$$

Consequently, our code has to change as well. Well, just do not make this kind of mistakes that all I can tell, ok.

(Refer Slide Time: 38:00)

Conservative system

$$m\ddot{x} = F = -\frac{\partial V}{\partial x}$$

$$V = -\frac{1}{2}x^2 + \frac{1}{4}x^4$$

$$F = -(x - x^3) = x - x^3$$

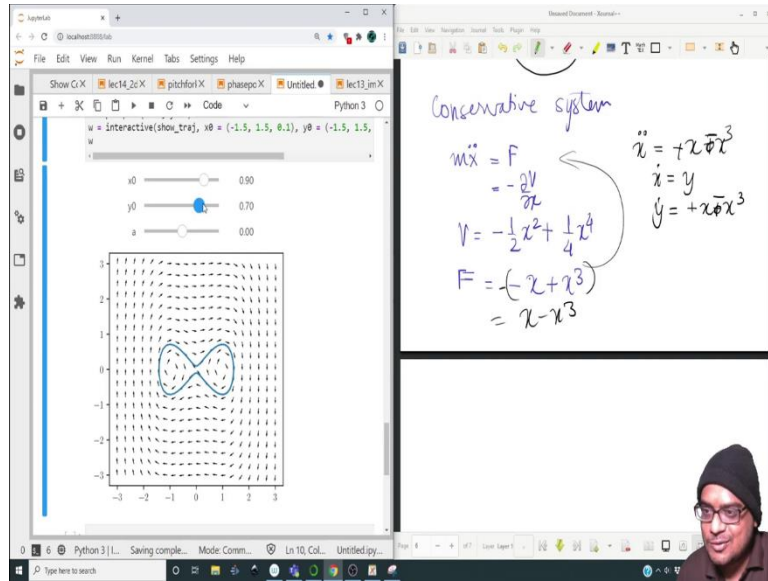
$$\ddot{x} = +x - x^3$$

$$\dot{x} = y$$

$$\dot{y} = +x - x^3$$

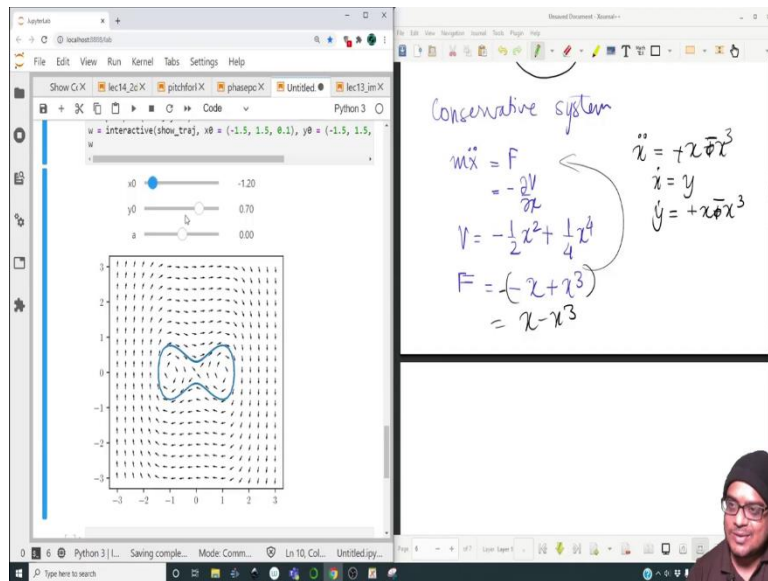
So, now we have the trajectory which looks something like this, it looks quite nice.

(Refer Slide Time: 38:07)

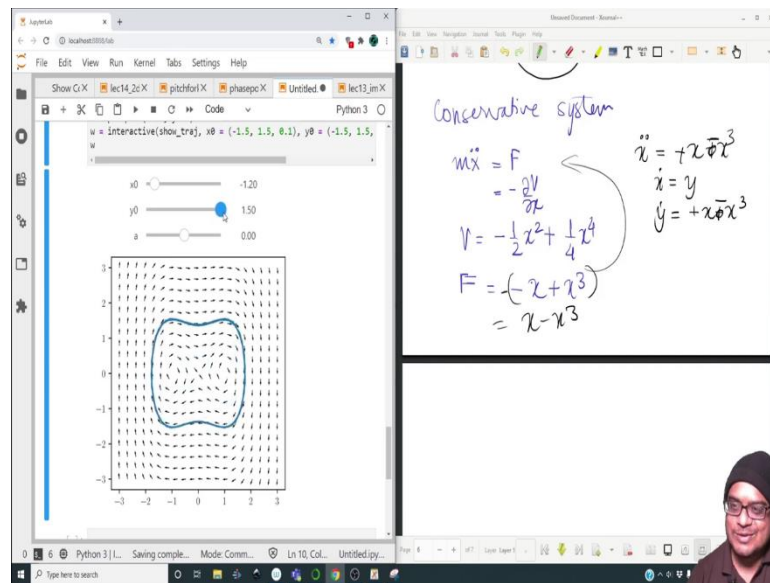


And, when you reduce the I mean change the initial condition you suddenly have this kind of change ok.

(Refer Slide Time: 38:16)

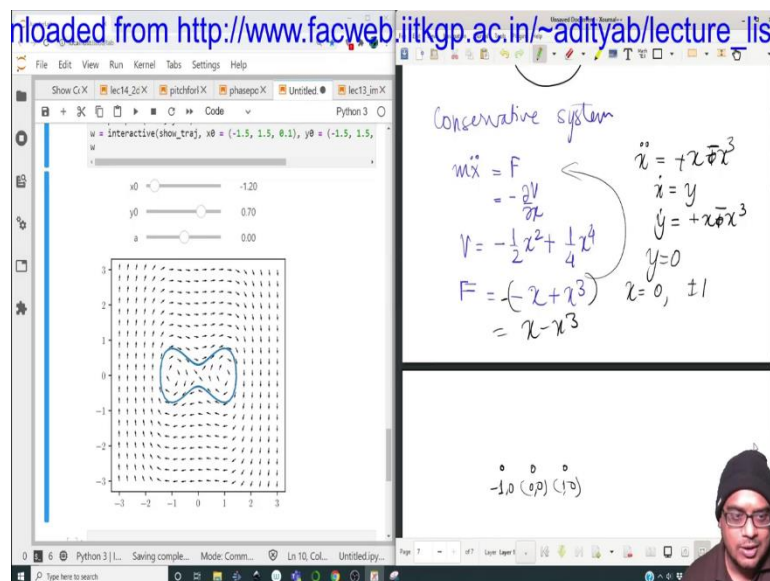


(Refer Slide Time: 38:25)



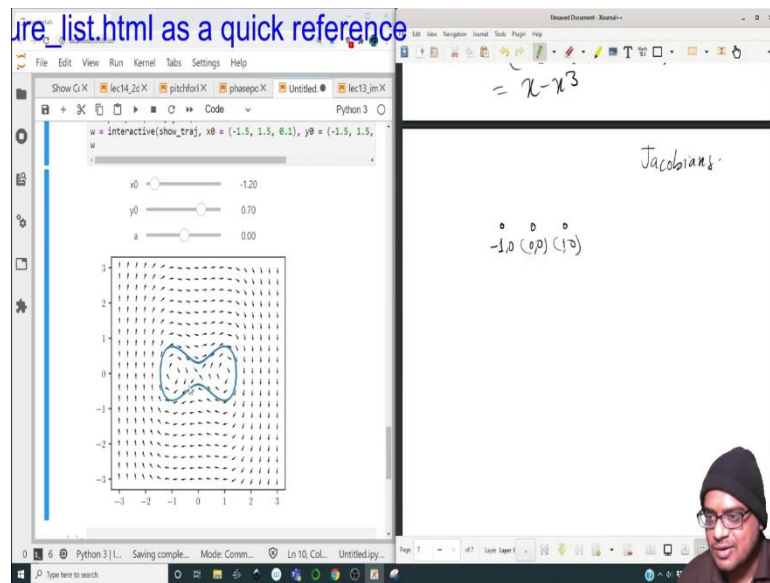
Now, let me it becomes a lobe and then two lobes and then it grows into a single orbit ok.

(Refer Slide Time: 38:34)



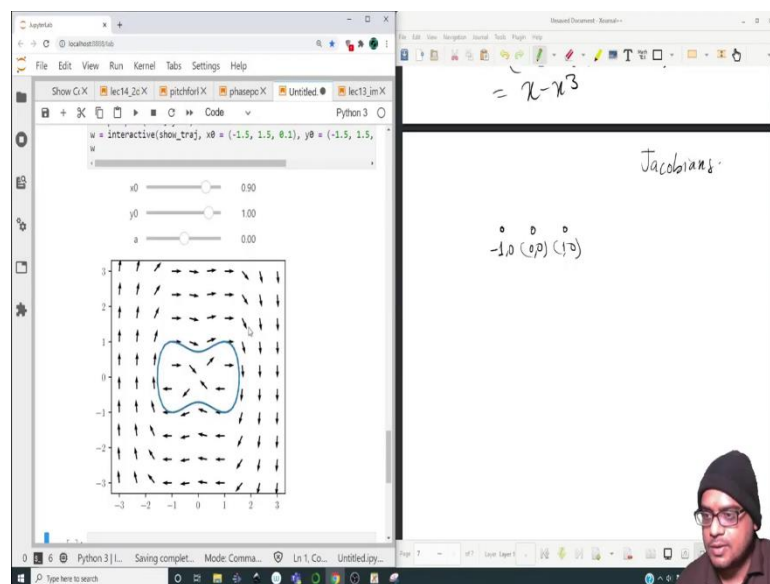
So, it is quite obvious looking at this diagram that there are actually two fixed points. Well, I mean by now you should be more than able to tell me what the fixed points are. The fixed points are points where  $x \dot{x} = 0$  and  $\dot{y} = 0$ . In this case  $y = 0$  and  $x$  will be either  $0$  or  $\pm 1$ . So, these are the fixed points; so,  $(-1, 0)$ ;  $(0, 0)$ ;  $(1, 0)$ .

(Refer Slide Time: 39:07)



And, what is the behavior of the equation near the three points? You should be able to tell me with the help of the Jacobians ok, but regardless of that it appears if you look closely over here that this point is attracting, this point is repeating.

(Refer Slide Time: 39:29)



In fact, let me reduce the number of arrows, so that things become slightly clearer ok yeah, well this is perhaps too few arrows.

(Refer Slide Time: 39:34)

The Jupyter Notebook code is as follows:

```
[41]: def show_traj(x0=0.5, y0=1.0, a = 0.0):
x = np.linspace(-3, 3, 15); y = np.linspace(-3, 3, 15);
X, Y = np.meshgrid(x, y);
U = Y;
V = X**3;
plt.quiver(X, Y, U/(U**2+V**2)**0.5, V/(U**2+V**2)**0.5);
ax = plt.gca(); ax.set_aspect(1);

def mysys(t, x): # returns the RHS
return ([1], x[0]-x[0]**3);

tspan = [0,20]
mi0 = -1.2; y0 = 2.5
ics = [x0, y0]
sol = solve_ivp(mysys, tspan, ics, dense_output=True);

tout = np.linspace(0, np.max(tspan), 100);
xout = sol.sol(tout)[0];
yout = sol.sol(tout)[1];
plt.plot(xout, yout);
w = interactive(show_traj, x0 = (-1.5, 1.5, 0.1), y0 = (-1.5, 1.5, 0.1))
```

The presentation slide shows the equation  $x - x^3$  and the Jacobian matrix at the origin  $(0,0)$ :

$$J = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Let me make it 15.

(Refer Slide Time: 39:41)

The Jupyter Notebook shows a plot of a vector field with a trajectory. A red error message is displayed:

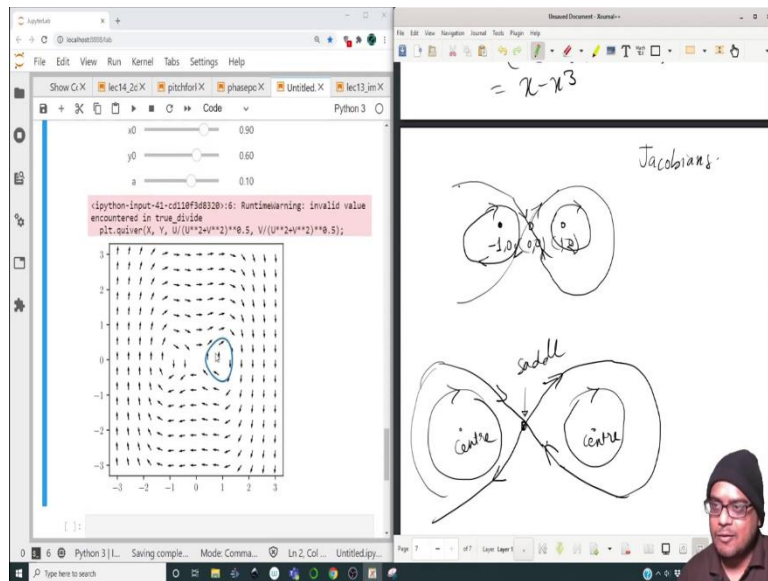
```
<ipython-input-41-c0110f3d832b>:6: RuntimeWarning: invalid value encountered in true_divide
plt.quiver(X, Y, U/(U**2+V**2)**0.5, V/(U**2+V**2)**0.5);
```

The presentation slide shows the equation  $x - x^3$  and the Jacobian matrix at the origin  $(0,0)$ :

$$J = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

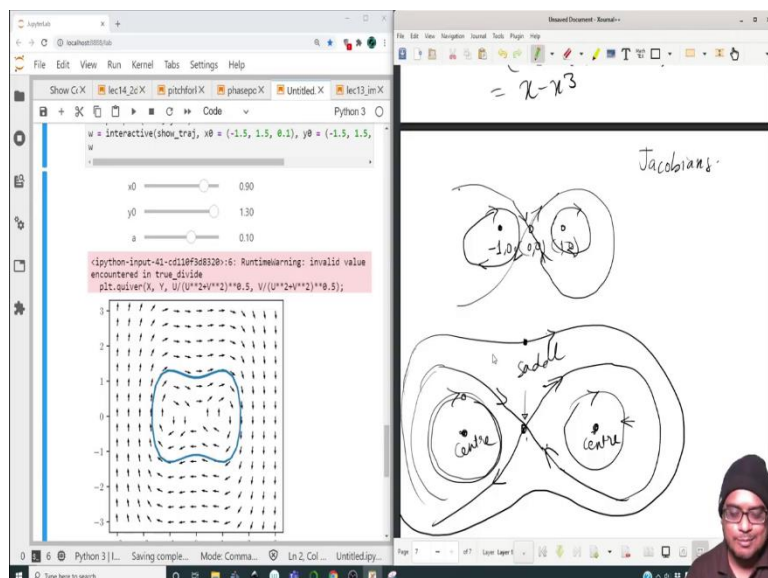
Often time it all depends on how you can interpret the equations ok. Obviously, something like this at the origin so, you have something like this and this and something like this and this ok.

(Refer Slide Time: 40:11)



So, you have this at this point you can change this and see. So, this trajectory is going around in circles ok. So, there is a bunch of closed orbits over here similarly there is a bunch of closed orbits over here as well, ok. So, essentially it is like you have a closed orbit like this, closed orbit like this and it going in circle like this ok. A picture is more or less like this. So, this is a saddle load, this is the center, this is also a center ok. So, this is the entire phase portrait for this particular plot.

(Refer Slide Time: 41:17)

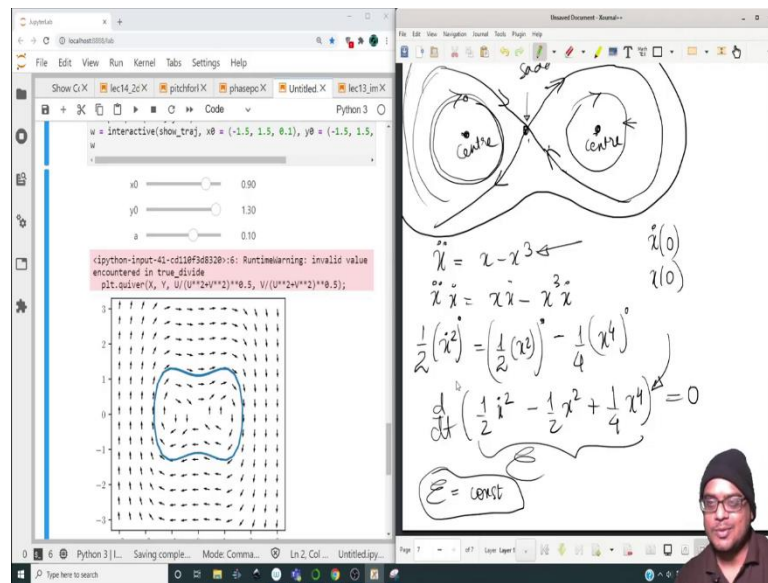


So, what is what happens when we change the different initial conditions? You see that if you choose an initial condition over here it will not go into these centers, but it will have a separate orbit which sort of envelopes this entire set of inside orbits. But, if you choose

an initial condition over here it goes around encircles around this center or it goes around encircles around this end.

But, once you have once you are having an initial condition over here it goes around in circles about this center, but it encompasses all the fixed points. So, what really is going on. So, such kind of systems are called as conservative systems for a good reason; so, this particular equation that we have ok.

(Refer Slide Time: 42:09)



So,  $\ddot{x} = x - x^3$ . Let us multiply this entire thing by  $\dot{x}$ ,  $\dot{x}\ddot{x} = x\dot{x} - x^3\dot{x}$ . Now, this is  $\frac{1}{2}(\dot{x}^2)$  this is going to be  $\frac{1}{2}((x^2))$  and this is going to be  $-1/4(x^4)$ . ok.

So, this implies  $d/dt(1/2\dot{x}^2 - 1/2x^2 + 1/4x^4) = 0$ ; it means that this entire term is equal to constant and let us call this as the energy. So, energy equal to constant. So, given this differential equation this solution in the phase space must also satisfy this additional constraint if you will that the energy along any contour is going to be conserved.

If you choose an initial condition that is  $x_0$  at 0 and  $\dot{x}$  at 0, once you choose this initial condition you are setting or you are dialing in what the value of the energy has to be. Once you dial in that particular value of energy, you simply follow the contour in the phase plane which preserves that energy that is how you can easily find the contour ok. So, with the help of this let me in fact, now get rid of the arrows as well we do not need the arrows. Let me in fact, get rid of let me keep this.

(Refer Slide Time: 44:04)

The screenshot shows a Jupyter Notebook interface on the left and a hand-drawn diagram on the right. The Jupyter Notebook displays a vector plot of a system with two centers and a separatrix. Below the plot is the following Python code:

```
[42]: def show_traj(x0=0.9, y0=1.0, a=0.0):  
def mysys(t, x): # returns the RHS  
return ([x[1], x[0]*x[1]**3])  
  
tspan = [0, 20]  
#x0 = -1.1; y0 = 2.5  
ics = [x0, y0]  
sol = solve_ivp(mysys, tspan, ics, dense_output=True);  
  
tout = np.linspace(0, np.max(tspan), 100);  
xout = sol.sol(tout)[0];  
yout = sol.sol(tout)[1];  
  
plt.plot(xout, yout)  
w = interactive(show_traj, x0 = (-1.5, 1.5, 0.1), y0 = (-1.5, 1.5,  
a = 0.00)
```

The hand-drawn diagram on the right shows a phase portrait with two centers and a separatrix. The diagram is annotated with the following mathematical expressions:

$$\ddot{x} = x - x^3$$
$$\dot{x} \ddot{x} = x \dot{x} - x^3 \dot{x}$$
$$\frac{1}{2}(\dot{x}^2) = \left(\frac{1}{2}(x^2)\right) - \frac{1}{4}(x^4)$$
$$\frac{d}{dt} \left( \frac{1}{2} \dot{x}^2 - \frac{1}{2} x^2 + \frac{1}{4} x^4 \right) = 0$$

The diagram also includes the text "E = const" and "Saddle" with arrows pointing to the saddle point in the phase portrait.

Let me copy this and create a new cell and let us get rid of the contour the vector plotting and let me simply have this, alright.

(Refer Slide Time: 44:16)

The screenshot shows a Jupyter Notebook interface on the left and a hand-drawn diagram on the right. The Jupyter Notebook displays a plot of the solution trajectory and the following Python code:

```
tout = np.linspace(0, np.max(tspan), 100);  
xout = sol.sol(tout)[0];  
yout = sol.sol(tout)[1];  
  
plt.plot(xout, yout)  
w = interactive(show_traj, x0 = (-1.5, 1.5, 0.1), y0 = (-1.5, 1.5,  
a = 0.00)
```

The hand-drawn diagram on the right is identical to the one in the previous slide, showing a phase portrait with two centers and a separatrix, and the same mathematical derivations for energy conservation.

We do not need the interactivity in this and whatever I am going to do now.



(Refer Slide Time: 44:18)

The screenshot shows a Jupyter Notebook on the left and a whiteboard on the right. The notebook code is as follows:

```
[43]:  
def mysys(t, x): # returns the RHS  
    return [x[1], x[0]*x[0]**3];  
  
tspan = [0, 20]  
x0 = [-1.1; 0]; y0 = 2.5  
ics = [x0, y0]  
sol = solve_ivp(mysys, tspan, ics, dense_output=True);  
  
tout = np.linspace(0, np.max(tspan), 100);  
xout = sol.sol(tout)[0];  
yout = sol.sol(tout)[1];  
  
plt.plot(xout, yout)
```

The whiteboard contains a phase portrait with two centers and a separatrix. The equations shown are:

$$\ddot{x} = x - x^3$$
$$\dot{x}\ddot{x} = x\dot{x} - x^3\dot{x}$$
$$\frac{1}{2}(\dot{x}^2) = \frac{1}{2}(x^2) - \frac{1}{4}(x^4)$$
$$\frac{d}{dt} \left( \frac{1}{2}\dot{x}^2 - \frac{1}{2}x^2 + \frac{1}{4}x^4 \right) = 0$$

The energy function is given as  $E = \text{const}$ .

Let me indent it back and let me remove this ok.

(Refer Slide Time: 44:28)

The screenshot shows a Jupyter Notebook on the left and a whiteboard on the right. The notebook code is as follows:

```
tout = np.linspace(0, np.max(tspan), 100);  
xout = sol.sol(tout)[0];  
yout = sol.sol(tout)[1];  
  
plt.plot(xout, yout)
```

The whiteboard contains a phase portrait with two centers and a separatrix. The equations shown are:

$$\ddot{x} = x - x^3$$
$$\dot{x}\ddot{x} = x\dot{x} - x^3\dot{x}$$
$$\frac{1}{2}(\dot{x}^2) = \frac{1}{2}(x^2) - \frac{1}{4}(x^4)$$
$$\frac{d}{dt} \left( \frac{1}{2}\dot{x}^2 - \frac{1}{2}x^2 + \frac{1}{4}x^4 \right) = 0$$

The energy function is given as  $E = \text{const}$ .

So, it shows a bunch of it shows the trajectory, but if you look closely the trajectories are really I mean wiggling around way too much to increase the number of points to make it smoother.

(Refer Slide Time: 44:40)

The screenshot shows a Jupyter Notebook on the left and a presentation slide on the right. The Jupyter Notebook code defines a function `mysys(t, x)` that returns the right-hand side of a differential equation system. It then solves the system using `solve_ivp` and plots the solution. The plot shows a closed, rounded rectangular curve in the  $x_1$ - $x_2$  plane.

The presentation slide on the right features a phase portrait with two centers labeled "Centre". Handwritten equations include:
 
$$\ddot{x} = x - x^3$$

$$\dot{x} \ddot{x} = x \dot{x} - x^3 \dot{x}$$

$$\frac{1}{2}(\dot{x}^2) = \left(\frac{1}{2}x^2\right) - \frac{1}{4}(x^4)$$

$$\frac{d}{dt} \left( \frac{1}{2} \dot{x}^2 - \frac{1}{2} x^2 + \frac{1}{4} x^4 \right) = 0$$
 A note at the bottom of the slide states  $E = \text{const}$ . A small video inset of a person is visible in the bottom right corner of the slide.

(Refer Slide Time: 44:42)

The screenshot shows a Jupyter Notebook on the left and a presentation slide on the right. The Jupyter Notebook code defines a function `mysys(t, x)` that returns the right-hand side of a differential equation system. It then solves the system using `solve_ivp` and plots the solution. The plot shows a closed, rounded rectangular curve in the  $x_1$ - $x_2$  plane.

The presentation slide on the right features a phase portrait with two centers labeled "Centre". Handwritten equations include:
 
$$\ddot{x} = x - x^3$$

$$\dot{x} \ddot{x} = x \dot{x} - x^3 \dot{x}$$

$$\frac{1}{2}(\dot{x}^2) = \left(\frac{1}{2}x^2\right) - \frac{1}{4}(x^4)$$

$$\frac{d}{dt} \left( \frac{1}{2} \dot{x}^2 - \frac{1}{2} x^2 + \frac{1}{4} x^4 \right) = 0$$
 A note at the bottom of the slide states  $E = \text{const}$ . A small video inset of a person is visible in the bottom right corner of the slide.

Not only that, let us increase the accuracy of the Runge-Kutta solver.

(Refer Slide Time: 44:45)

The Jupyter Notebook code is as follows:

```
def mysys(t, x): # returns the RHS
    return ([x[1], x[0]*x[1]**3]);

tspan = [0, 20]
x0 = -1.1, y0 = 2.5
ics = [x0, y0]
sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol=1e-6);

tout = np.linspace(0, np.max(tspan), 200);
xout = sol.sol(tout)[0];
yout = sol.sol(tout)[1];

plt.plot(xout, yout)
```

The presentation slide contains the following handwritten notes:

Phase portrait showing two centers. The equations are:

$$\ddot{x} = x - x^3$$

$$\dot{x} \ddot{x} = x \dot{x} - x^3 \dot{x}$$

$$\frac{1}{2}(\dot{x}^2) = \left(\frac{1}{2}(x^2)\right) - \frac{1}{4}(x^4)$$

$$\frac{d}{dt} \left( \frac{1}{2} \dot{x}^2 - \frac{1}{2} x^2 + \frac{1}{4} x^4 \right) = 0$$

Energy is conserved:  $E = \text{const}$

So, let me make rtol is equal to 1e - 6. Great.

(Refer Slide Time: 44:56)

The Jupyter Notebook code is as follows:

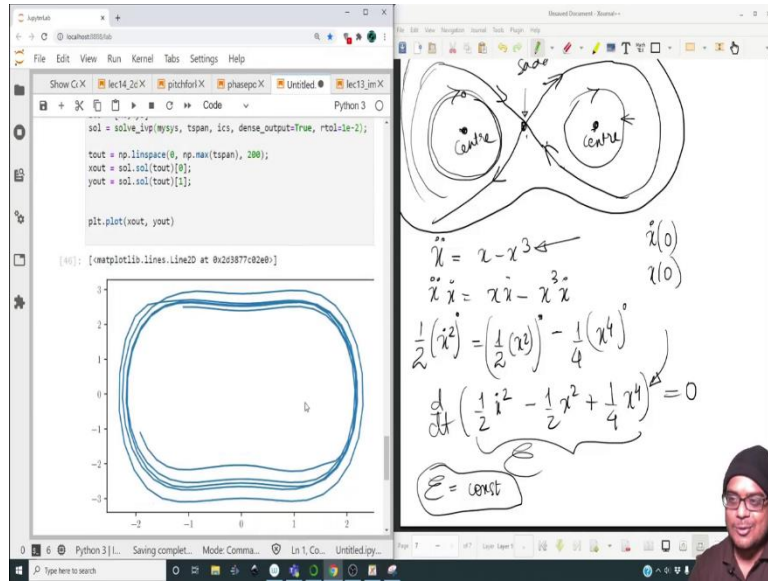
```
tout = np.linspace(0, np.max(tspan), 200);
xout = sol.sol(tout)[0];
yout = sol.sol(tout)[1];

plt.plot(xout, yout)
```

The plot shows a smooth closed curve. The presentation slide contains the same handwritten notes as above.

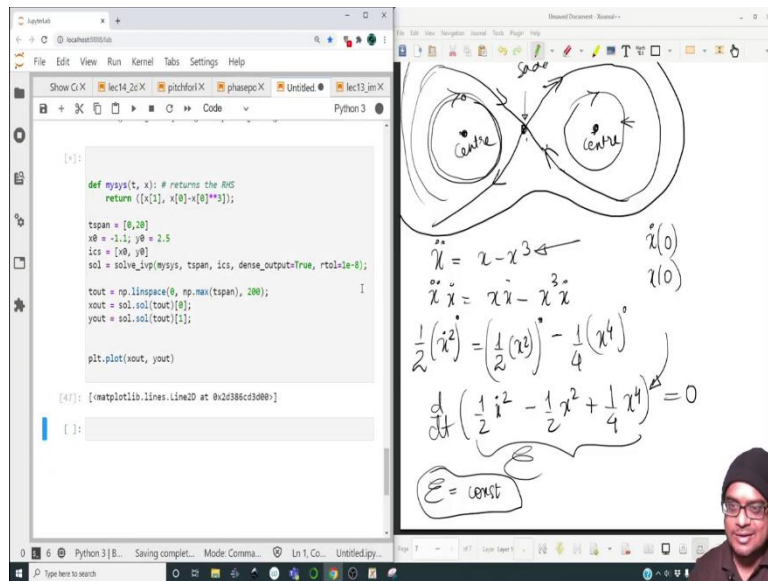
Now, we really have a smooth curve. It all depends on how well constrained you want to make the Runge-Kutta solver smaller the relative tolerance, better will be the smoothness of a solution.

(Refer Slide Time: 45:13)



If I reduce this to 1e-2 look at how errors are being accumulated ok. The orbit should not degrade into so many orbits.

(Refer Slide Time: 45:26)



It should be a single curve. That is why increasing this will really make things look much better ok.

(Refer Slide Time: 45:26)

The screenshot shows a Jupyter Notebook with a Python code cell and a plot. The code cell contains the following code:

```

tout = np.linspace(0, np.max(tspan), 200);
xout = sol.sol(tout)[0];
yout = sol.sol(tout)[1];

plt.plot(xout, yout)

```

The plot shows a trajectory in the phase space (x vs y) that forms a closed loop around a central point. The trajectory is blue and the axes range from -2 to 2.

Handwritten notes on the right side of the notebook include:

- A diagram of a figure-eight trajectory with two centers labeled "Centre".
- The equation  $\ddot{x} = x - x^3$ .
- The equation  $\dot{x} \ddot{x} = x \dot{x} - x^3 \dot{x}$ .
- The equation  $\frac{1}{2}(\dot{x}^2) = \left(\frac{1}{2}x^2\right) - \frac{1}{4}(x^4)$ .
- The equation  $\frac{d}{dt} \left( \frac{1}{2} \dot{x}^2 - \frac{1}{2} x^2 + \frac{1}{4} x^4 \right) = 0$ .
- The equation  $E = \text{const}$ .

(Refer Slide Time: 45:34)

The screenshot shows a Jupyter Notebook with a Python code cell and a plot. The code cell contains the following code:

```

def mysys(t, x): # returns the RHS
    return ([x[1], x[0]-x[0]**3]);

tspan = [0,20]
x0 = -1.1; y0 = 2.5
ics = [x0, y0]
sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol=1e-8);

tout = np.linspace(0, np.max(tspan), 200);
xout = sol.sol(tout)[0];
yout = sol.sol(tout)[1];
E = 1/2*yout**2 - 1/2*xout**2 + 1/4*xout**4;

plt.plot(xout, yout)
plt.plot(tout, E)

```

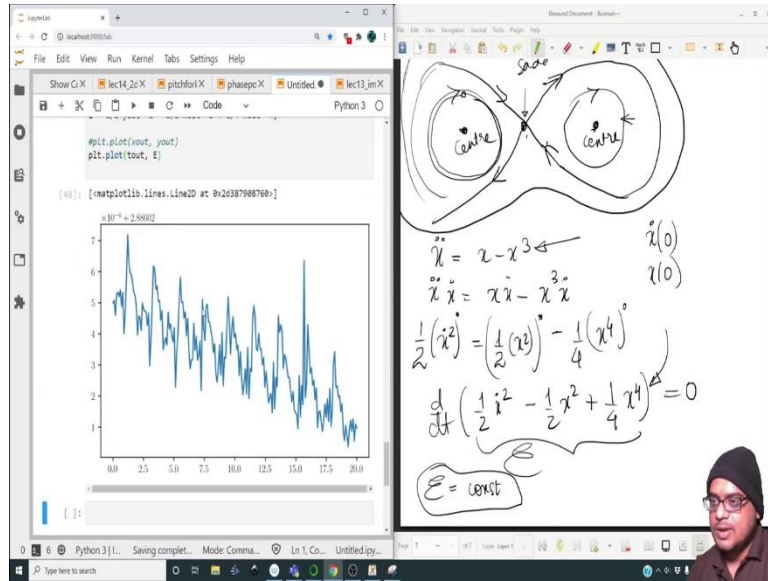
The plot shows a trajectory in the phase space (x vs y) that forms a closed loop around a central point. The trajectory is blue and the axes range from -2 to 2. A second plot shows the energy E vs time t, which is a constant horizontal line.

Handwritten notes on the right side of the notebook include:

- A diagram of a figure-eight trajectory with two centers labeled "Centre".
- The equation  $\ddot{x} = x - x^3$ .
- The equation  $\dot{x} \ddot{x} = x \dot{x} - x^3 \dot{x}$ .
- The equation  $\frac{1}{2}(\dot{x}^2) = \left(\frac{1}{2}x^2\right) - \frac{1}{4}(x^4)$ .
- The equation  $\frac{d}{dt} \left( \frac{1}{2} \dot{x}^2 - \frac{1}{2} x^2 + \frac{1}{4} x^4 \right) = 0$ .
- The equation  $E = \text{const}$ .

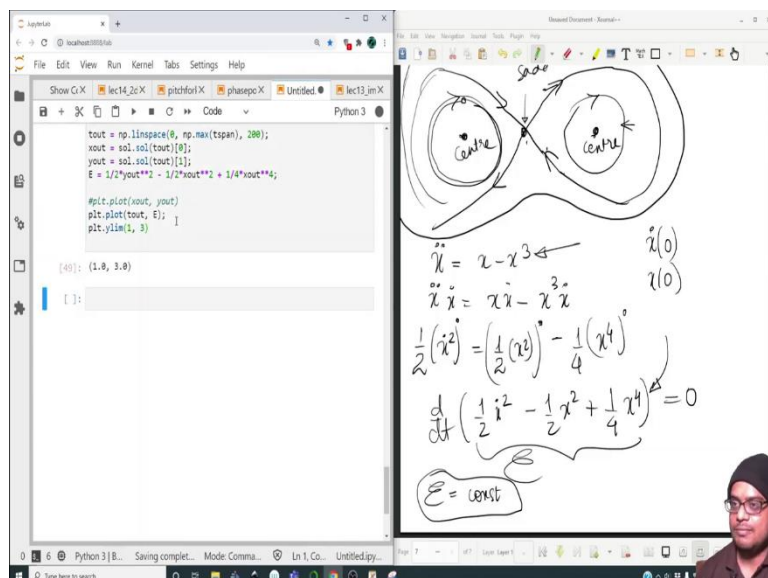
So, now let me create a new variable called as e and that will be this energy equation that we have not the equation, but the expression for the energy it is  $\frac{1}{2} * yout ** 2 - \frac{1}{2} * xout ** 2 + \frac{1}{4} * xout ** 4$ , alright. In fact, apart from this let me actually we have seen the trajectory, let me show you how energy looks like ok.

(Refer Slide Time: 46:08)



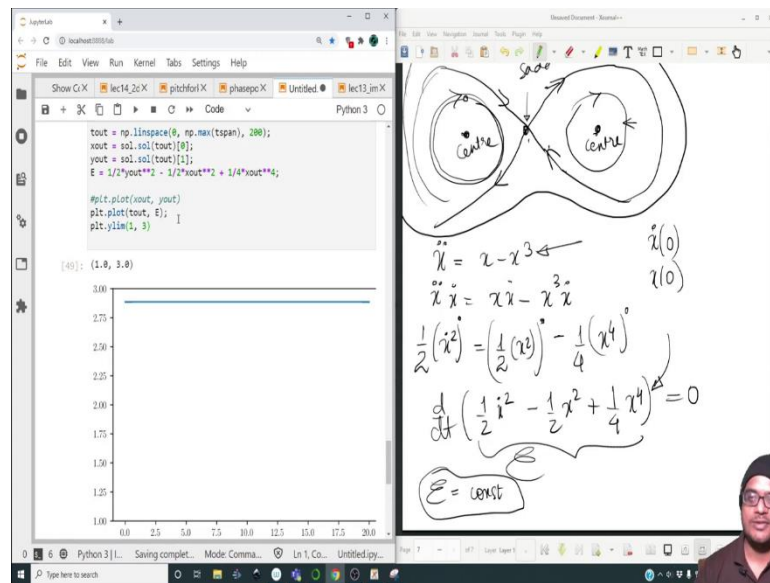
So, energy is varying over a extremely small bandwidth and this is due to the errors accumulating in the Runge-Kutta solver. So, the energy is decreasing, but it is not decreasing very fast over the entire time span it has hardly changed from 6 sorry, 2.886 to 2.88 something plus 5 into 10 to the power - 6 to something like 2.88602 plus 1 10 to the power - something. So, for a mean value of around 2.88 it is changing by 10 to the power - 6 is really quite negligible.

(Refer Slide Time: 46:55)



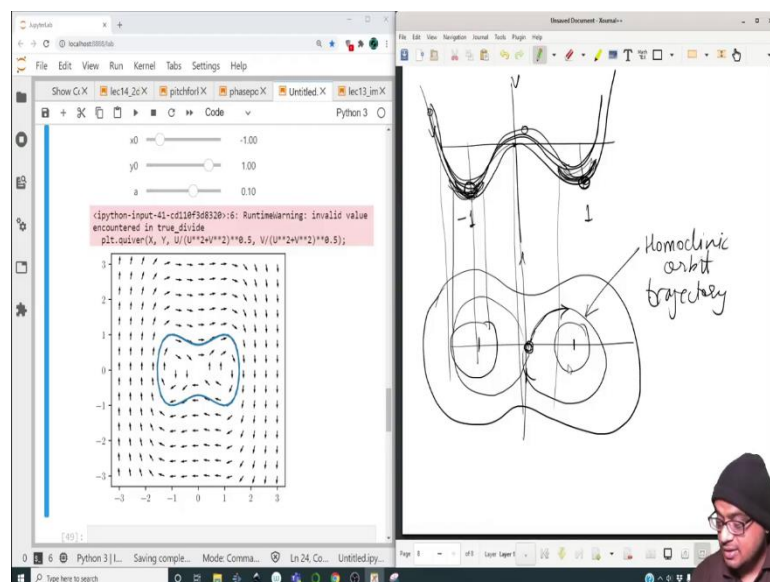
And, how can you show that it is negligible you simply has the y limits of this plot. So, you do plt.ylim and say you go from 0 to 2 or 1 to 3.

(Refer Slide Time: 47:05)



It is almost like a straight line. So, the energy is hardly changing. So, we now know that along a certain trajectory, the energy is not going to change ok. Can we somehow connect this particular phase plot? Or let me make a new page.

(Refer Slide Time: 47:30)



So, the double well potential actually looks something like this ok. So, this point is -1, this point is 1, this point is 0. So, as you can imagine if you have a ball over here and if you try to roll it like this, it will try to oscillate like this and this particular oscillation is nothing, but this particular trajectory.

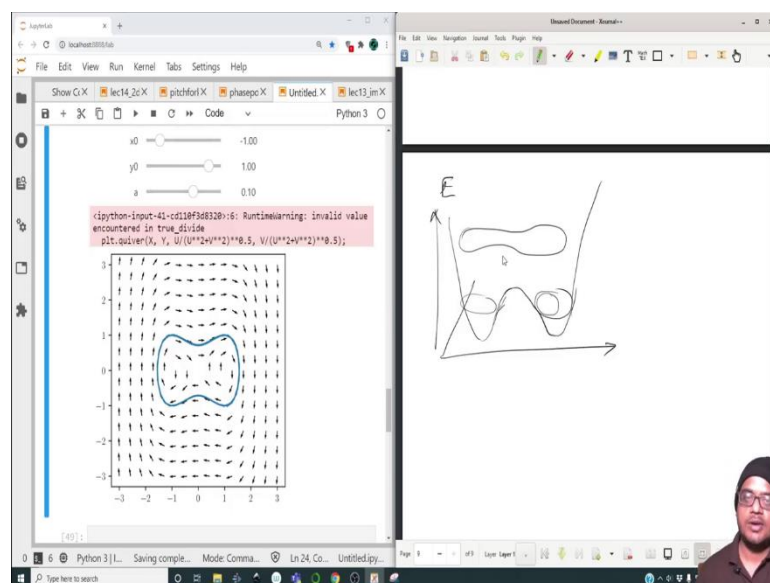
So, as it is oscillating in this potential it gives this trajectory in the xy plane if we initial if the initial point is somewhere over here let me just track this over here. So, then we have this trajectory and it corresponds to this trajectory. But, now if I increase the velocity if I increase the velocity even if I start over here it will sort of cross the barrier, alright.

So, it is having enough energy to oscillate all the way from this to this, it is doing this oscillation that corresponds to this entire sort of closed orbit. But, if you have small energies you are only oscillating inside the neighborhood of one and - one and hence you see that.

So, can we actually plot this? I mean obviously, you can sort of make the connection that this phase plot and the potential they do have a certain thing going on over here something like this and trajectories like this ok. So, what is this center point? That is called as a homoclinic not that point rather.

This particular orbit is called as a homoclinic orbit or a homoclinic trajectory because it starts the trajectory starts from this point and it ends at this equilibrium point ok. So, whenever a trajectory originates and ends at an equilibrium point that particular trajectory is called as a homogenic orbit, ok. So, this corresponds to this; this small oscillation will correspond to some internal oscillation similar to this over in this lobe as well, but can we plot it in the energy space?

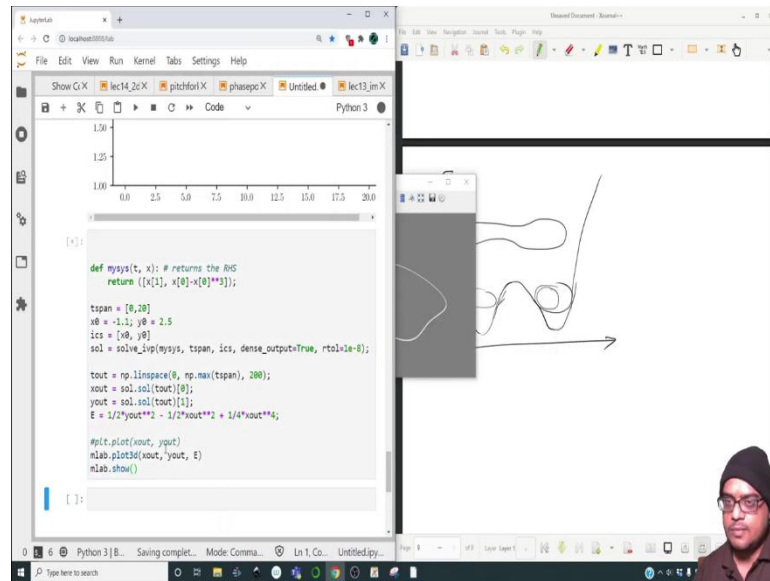
(Refer Slide Time: 50:35)





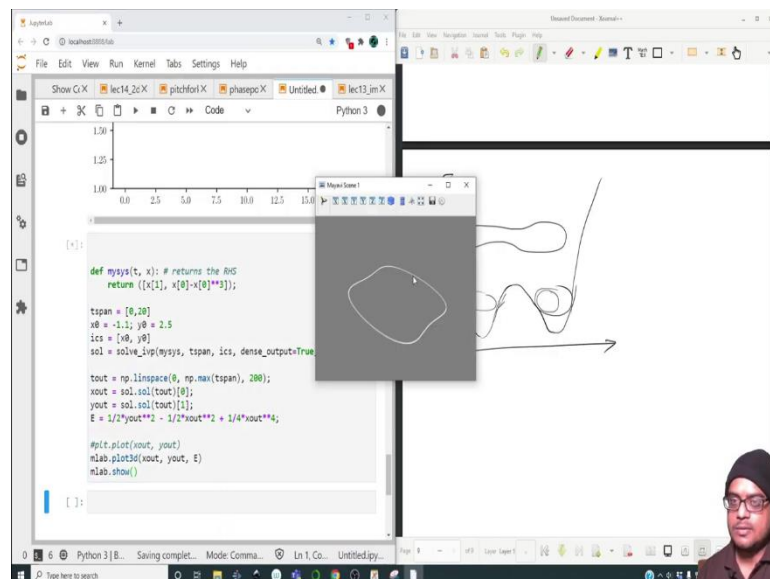
So, imagine if I were to plot energy on this axis. So, when we have small oscillations when we have small oscillations these are oscillations corresponding to low energy ok. So, there will be small lobes, but when you have large oscillations then the trajectory is whatever is shown over here. So, can we show that? Can we arrive at some kind of figure like this let us see; let us see how to do that. So, this is the plot that we have, let me copy this.

(Refer Slide Time: 51:21)



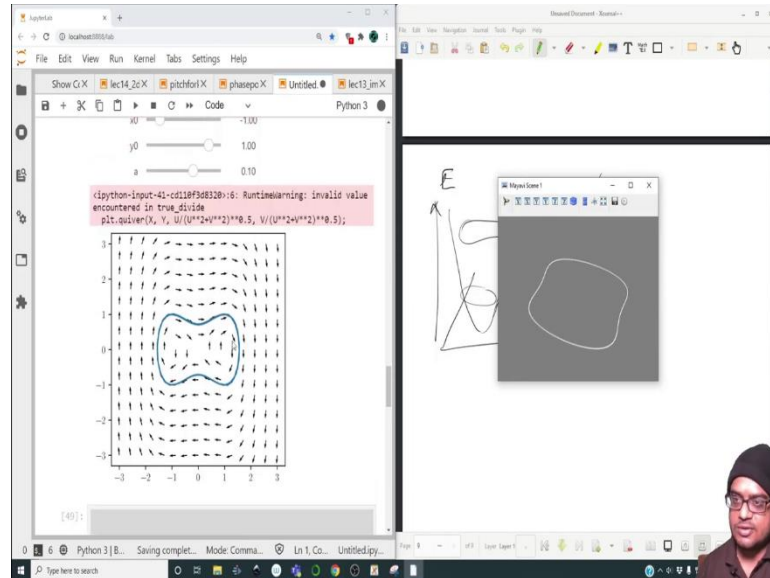
Let me copy this over here. So, now we have x, y and energy. So, we will do `mlab.plot3d(xout, yout, E)`; let me see what this looks like. So, we have to do `mlab.show` as well ok.

(Refer Slide Time: 51:49)



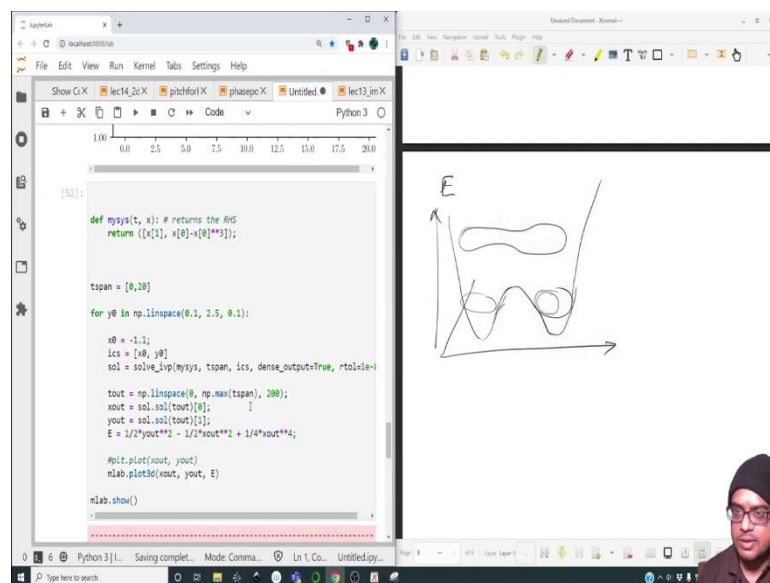
So, we have this trajectory and this trajectory is actually corresponding to this trajectory alright.

(Refer Slide Time: 51:56)



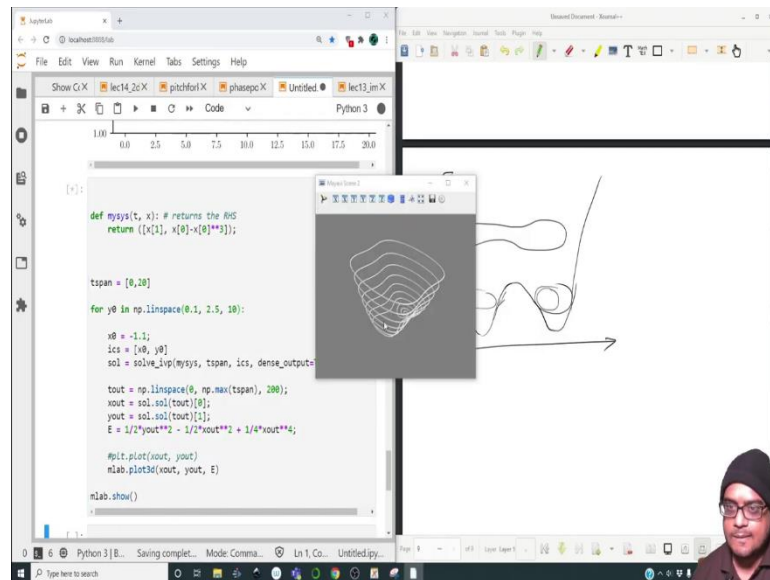
So, now let me choose series of initial points and let us reconstruct this entire 3-dimensional phase space. Well, it is not specifically speaking a phase space because  $E$  is basically a derived quantity from  $x$  and  $\dot{x}$  basically  $x$  and  $y$ . But, still it gives you a nice picture of how the double well potential connects to these trajectories ok.

(Refer Slide Time: 52:27)



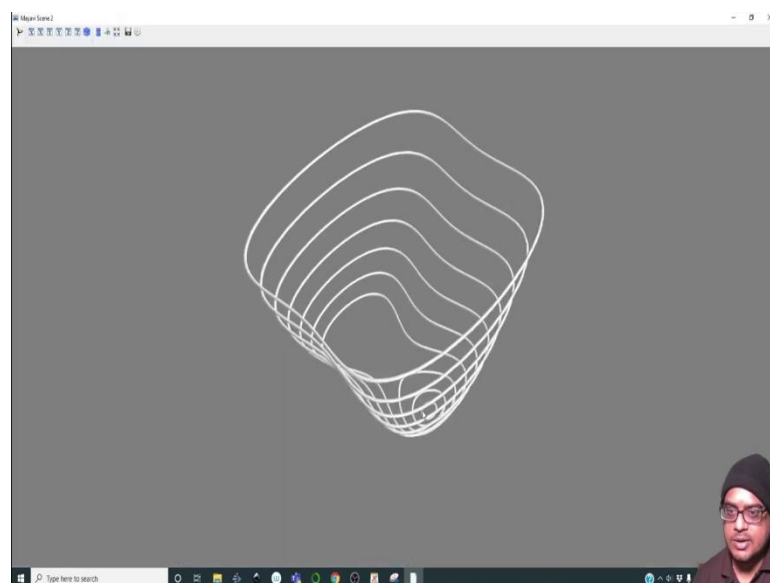
So, I am just going to pick this up I am going to make a loop. So, for  $y_0$  in  $\text{np.linspace}(0.1$  to 2.5 in steps of say 0.1. I am going to indent whatever is over here inside and lastly, I am going to show that plot let us run this let me close this let me run this ok. There is an error what is the error.

(Refer Slide Time: 53:09)



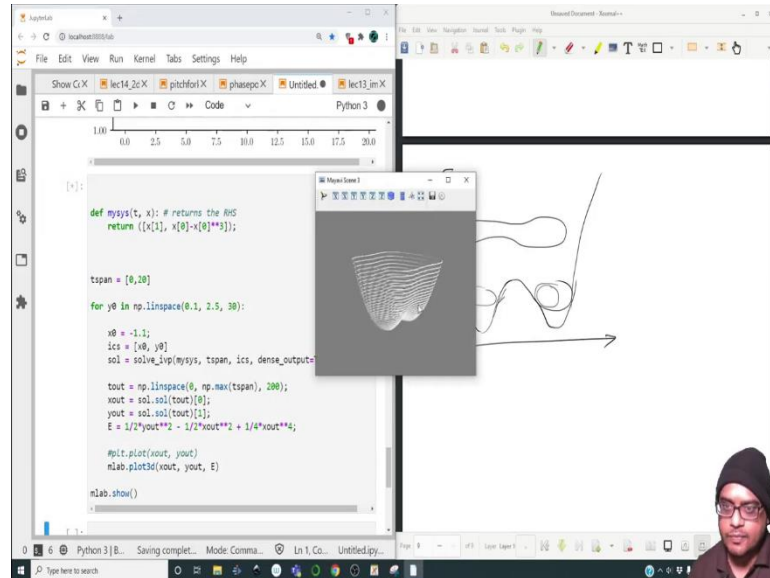
So, obviously, this has to be a number of intervals let me keep 10 ok. So, look, let me enlarge this.

(Refer Slide Time: 53:18)



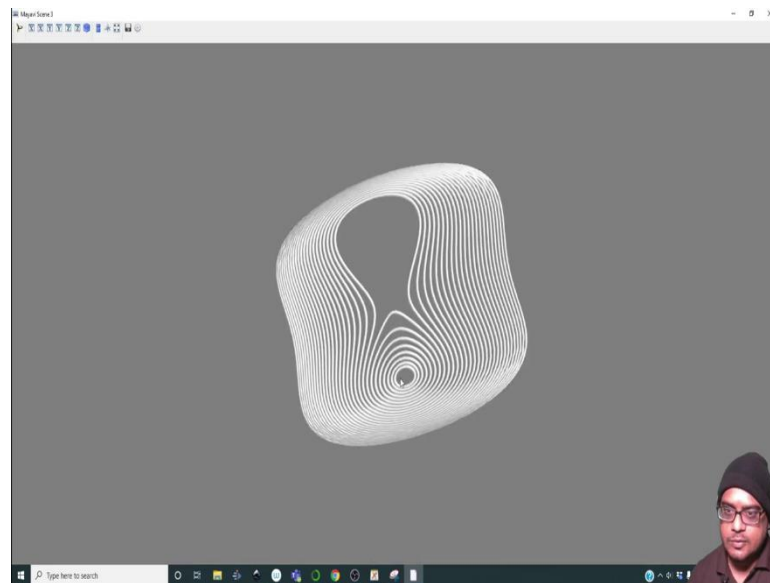
So, this is that limb; this is that limb where it is confined to the low energy oscillations as we go towards higher levels we have these outer orbits.

(Refer Slide Time: 53:41)



In fact, let me increase the number of contours so that it will be even clearer ok.

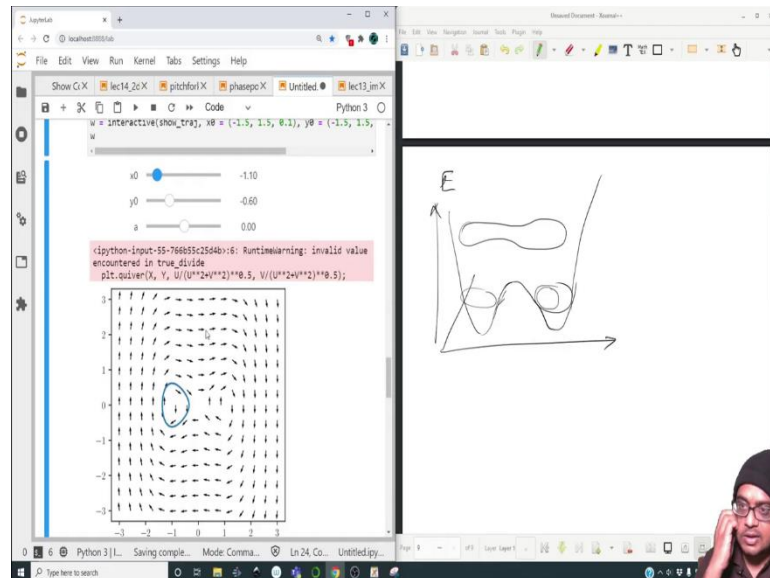
(Refer Slide Time: 53:48)



So, this is that initial condition and as  $y$  increases it transitions from having these small orbits. So, this is like the heteroclinic orbit it is just about to touch and you might be wondering why this orbit is not there because I am not starting initial points over in that

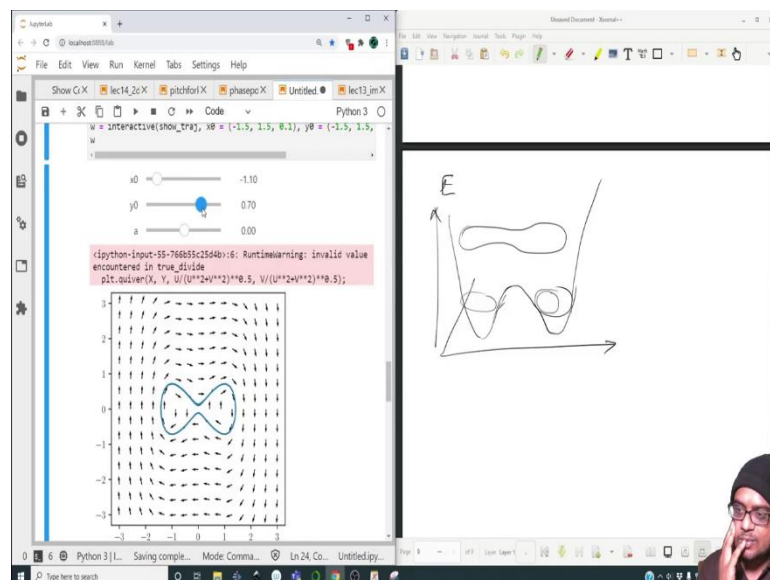
part of the loop ok. So, I am starting my initial conditions over at ok. So, I am starting the initial conditions somewhere over here ok.

(Refer Slide Time: 54:41)

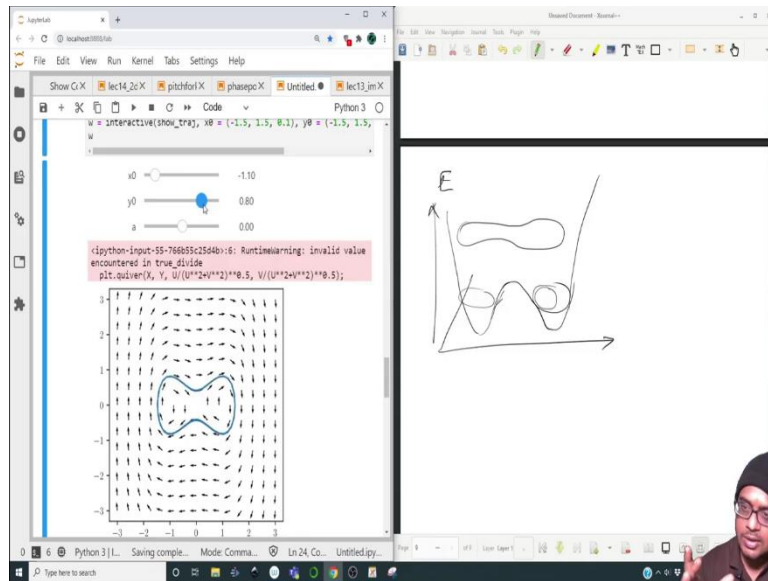


And, as I am slowly increasing  $y$  as I am slowly increasing  $y$ , I am transitioning into this double lobe.

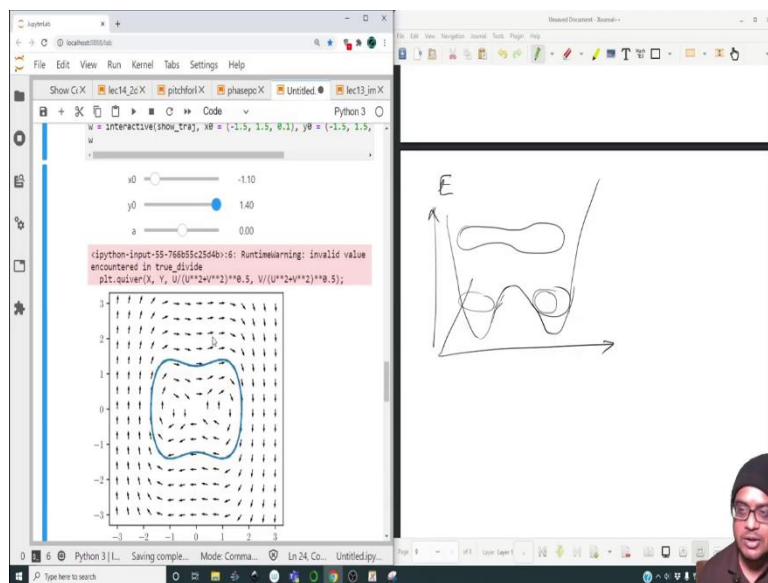
(Refer Slide Time: 54:52)



(Refer Slide Time: 54:55)

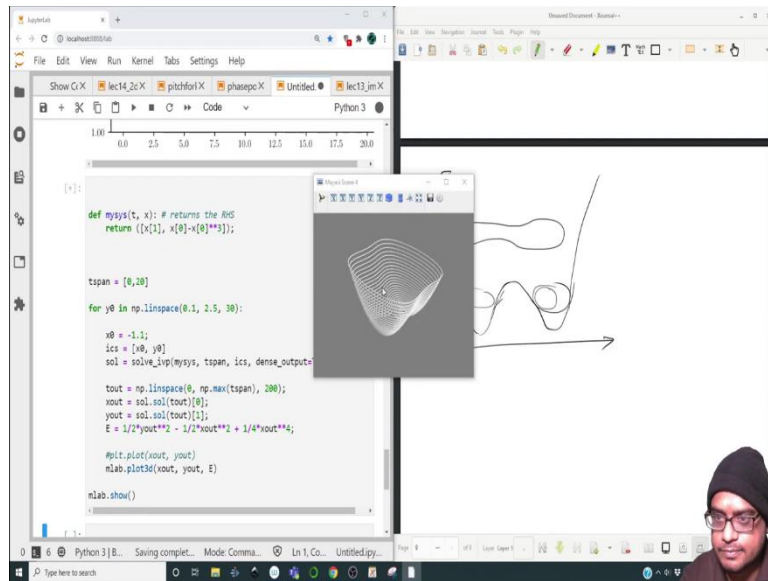


(Refer Slide Time: 54:56)

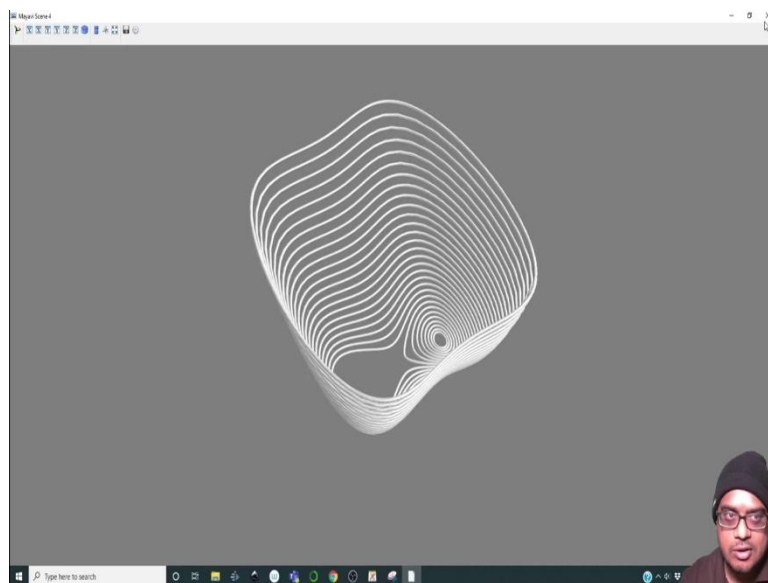


And, finally, into this continuous band and this is exactly what we see through this particular diagram ok.

(Refer Slide Time: 55:01)

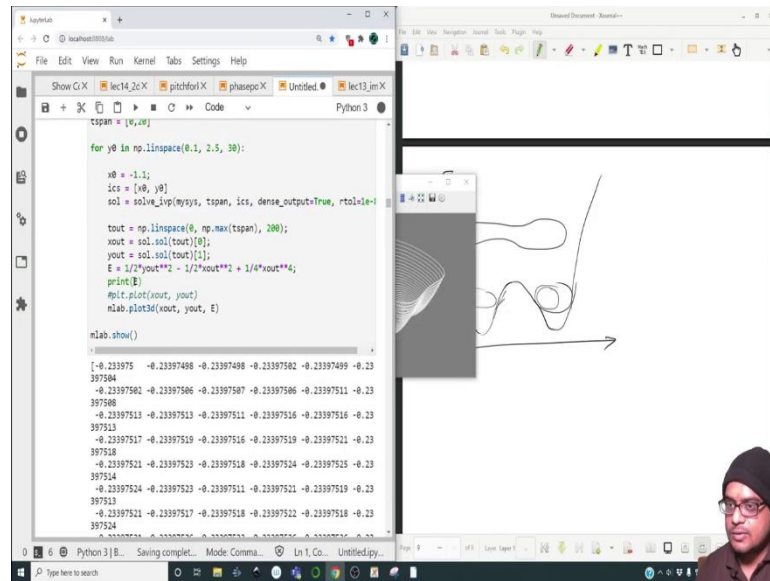


(Refer Slide Time: 55:06)



So, in fact, it will be nice if we can color these 3-dimensional objects. So, let us do that.

(Refer Slide Time: 55:22)



```
tspan = [y0,x0]

for y0 in np.linspace(0.1, 2.5, 30):
    x0 = -1.1;
    ics = [y0, y0]
    sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol=1e-10)

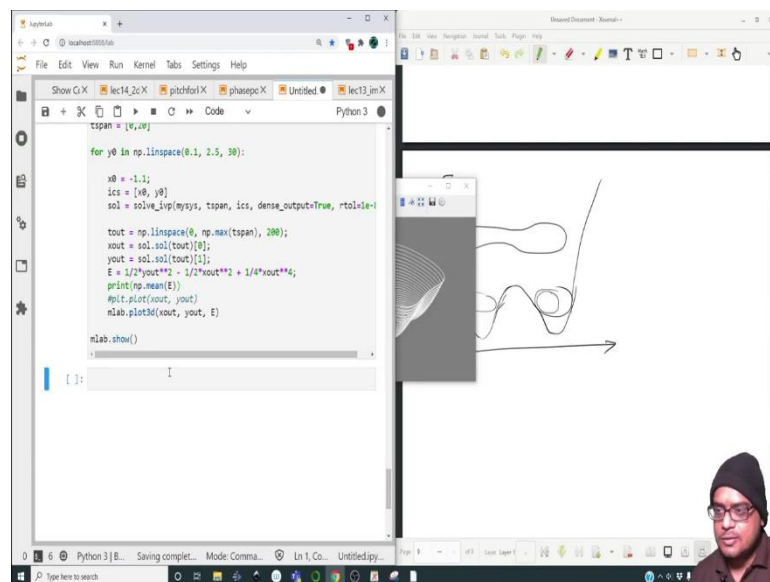
    tout = np.linspace(0, np.max(tspan), 200);
    xout = sol.sol(tout)[0];
    yout = sol.sol(tout)[1];
    E = 1/2*yout**2 - 1/2*xout**2 + 1/4*xout**4;
    print(E)
    #plt.plot(xout, yout)
    #lab.plot3d(xout, yout, E)

mlab.show()
```

```
[ -0.233975  -0.23397488 -0.23397498 -0.23397502 -0.23397499 -0.23397504
 -0.23397502 -0.23397506 -0.23397507 -0.23397506 -0.23397511 -0.23397509
 397509
 -0.23397513 -0.23397513 -0.23397511 -0.23397516 -0.23397516 -0.23397513
 397513
 -0.23397517 -0.23397519 -0.23397516 -0.23397519 -0.23397521 -0.23397518
 397518
 -0.23397521 -0.23397523 -0.23397523 -0.23397524 -0.23397525 -0.23397524
 397524
 -0.23397524 -0.23397523 -0.23397511 -0.23397521 -0.23397519 -0.23397513
 397513
 -0.23397521 -0.23397517 -0.23397518 -0.23397522 -0.23397518 -0.23397524
 397524
```

In fact, in order to color them nicely let me first output what the value of the energy is and I will tell you why I am doing that.

(Refer Slide Time: 55:28)



```
tspan = [y0,x0]

for y0 in np.linspace(0.1, 2.5, 30):
    x0 = -1.1;
    ics = [y0, y0]
    sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol=1e-10)

    tout = np.linspace(0, np.max(tspan), 200);
    xout = sol.sol(tout)[0];
    yout = sol.sol(tout)[1];
    E = 1/2*yout**2 - 1/2*xout**2 + 1/4*xout**4;
    print(np.mean(E))
    #plt.plot(xout, yout)
    #lab.plot3d(xout, yout, E)

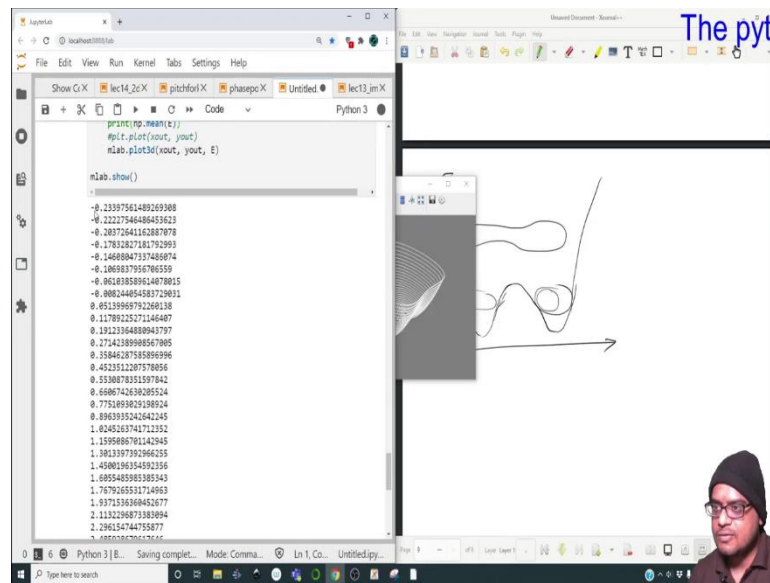
mlab.show()
```

```
[ 1.]:
```

So, let me just print mean of E because energy is obviously going to be preserved along it is going to be preserved in time for a given initial condition. So, I do not find any utility in printing out the entire string of energies for a given initial condition. So, I will just print the average energy for a given initial condition because we are looping over all the different initial conditions ok.

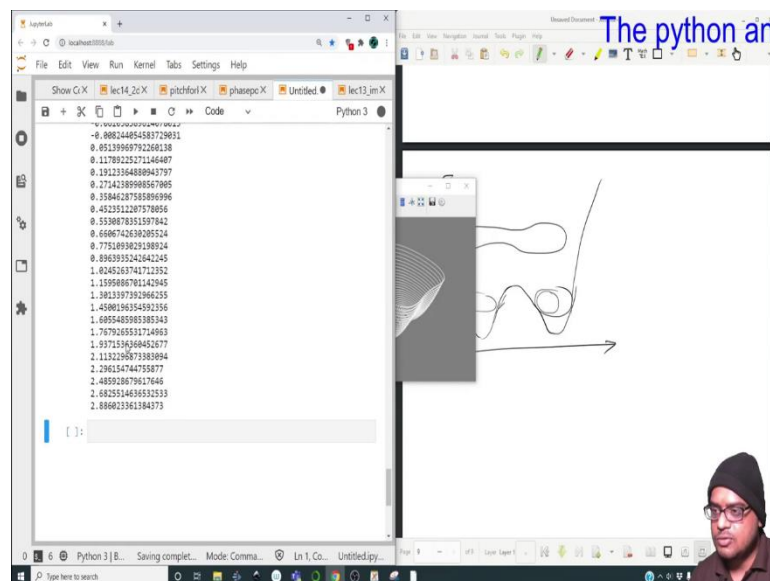


(Refer Slide Time: 55:56)



So, let me close this ok. So, it is starting from - 2.23 - 2 point.

(Refer Slide Time: 56:00)



So, let us say - 0.3 to 3 ok. So, the energies are going from this ok. So, color equal to and we have to interpolate between - 0.3 to 3.

(Refer Slide Time: 56:28)

```
span = [0, 200]

y0 in np.linspace(0.1, 2.5, 30):
    y0 = -1.1;
    ics = [y0, y0]
    sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol=1e-8)

    tout = np.linspace(0, np.max(tspan), 200);
    yout = sol.sol(tout)[0];
    yout = sol.sol(tout)[1];
    E = 1/2*yout**2 - 1/2*xout**2 + 1/4*xout**4;
    print(np.mean(E))
    plt.plot(xout, yout)
    mlab.plot3d(xout, yout, E, color=(1/3.3*(np.mean(E)+0.3),1,1))

lab.show()
```

Hand-drawn diagram showing a coordinate system with axes labeled  $E$  and  $G$ . A curve is plotted in the  $E$ - $G$  plane. A point  $(1, 1, 1)$  is marked. The equation  $y=0=1$  is written below the  $E$ -axis. The equation  $C-D = \frac{1}{3.3} (E+0.3)$  is written below the  $G$ -axis.

So, what we want to pass to color is a tuple rgb right something like this and we want to map the energy to how the color changes. So, we want to map. So, the minimum is suppose this is the energy axis and this is the color axis. So, it has to go from 0 to 1 over here and - 0.3 to 3 over here. We want a linear variation in color like this. Obviously,  $C - 0$  will be 1 upon 3.3 times  $E -$  of this; so,  $E$  plus 0.3.

So, let us encode this color over here. So, this will be color equals 1 upon 3.3 times np dot mean of  $E$  plus 0.3 and the other colors we can keep up 1 and 1. Let us run this and see what happens ok.

(Refer Slide Time: 57:40)

The screenshot shows a JupyterLab environment with a Python 3 kernel. The code defines a function `mysys(t, x)` that returns the right-hand side of a system of ordinary differential equations. The simulation parameters are `tspan = [0, 30]`, `x0 = [-1.1, 0.0]`, and `ics = [x0, y0]`. The solution is computed using `sol = solve_ivp(mysys, tspan, ics, dense_output=True)`. The trajectory is plotted using `mlab.plot3d(xout, yout, E, color=(1/3.3*(np.mean(E)+0.3), 1, 1))`. The 3D plot shows a complex, multi-lobed trajectory in a 3D space. A small window titled "MyScene1" displays the 3D plot. In the background, there is a hand-drawn diagram of a 3D coordinate system with axes labeled  $x$ ,  $y$ , and  $E$ . The origin is marked with  $(1, 1, 1)$ . The  $E$  axis has tick marks at  $-0.3$  and  $3$ . The equation  $C-D = \frac{1}{3.3}(E+0.3)$  is written below the diagram. A small video feed of the presenter is visible in the bottom right corner.

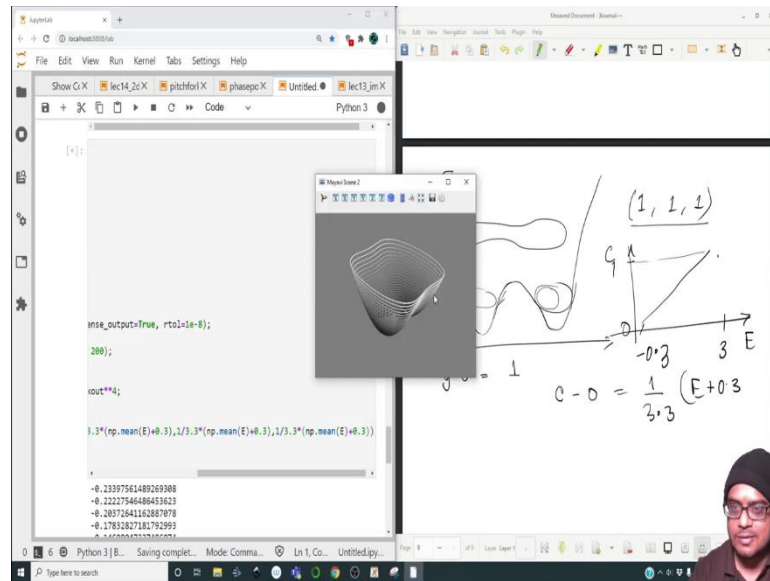
So, the diagram looks something like this.

(Refer Slide Time: 57:44)

The screenshot shows a 3D plot of a trajectory in a 3D space. The trajectory is a complex, multi-lobed shape. The color of the trajectory varies from blue to white, representing energy levels. The plot is displayed in a window titled "MyScene1". A small video feed of the presenter is visible in the bottom right corner.

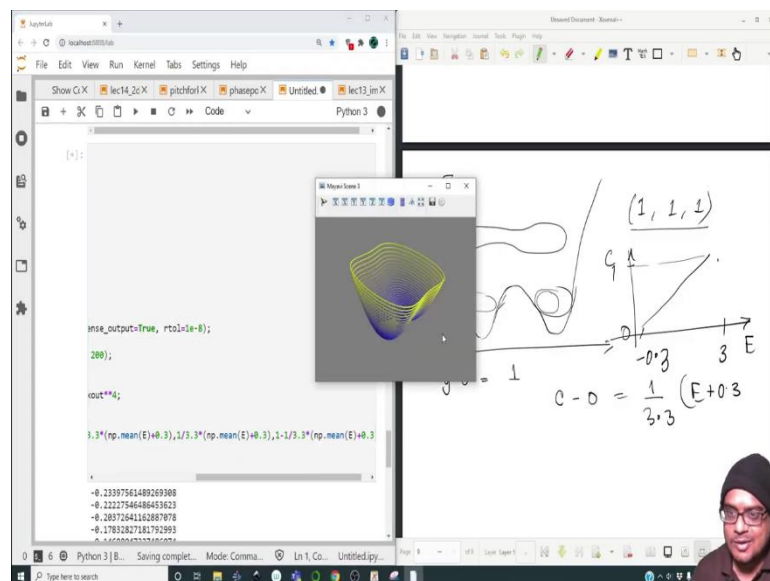
It is more colorful and yeah the whiter the diagram becomes the more energy that particular trajectory has; the bluer it is the lesser energy it has in fact. We can change this further by pasting this for the other color channels as well, ok.

(Refer Slide Time: 58:08)



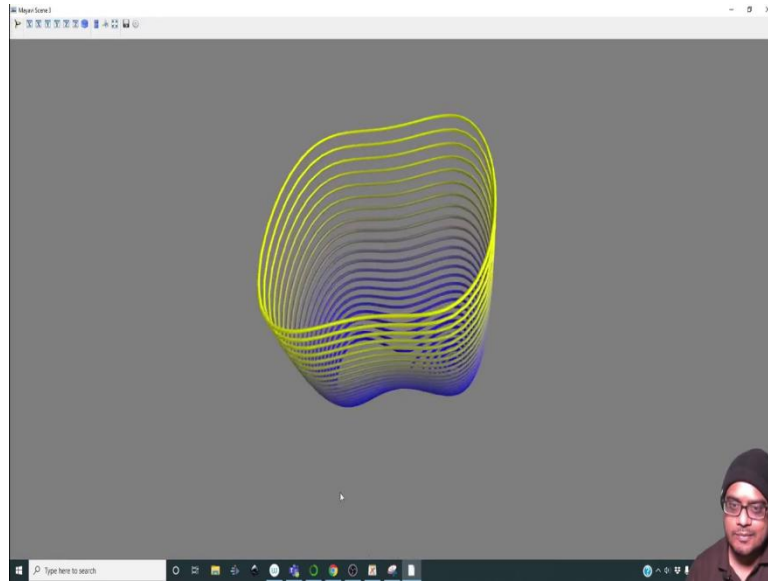
So, let us run this ok. It is all blackish.

(Refer Slide Time: 58:18)



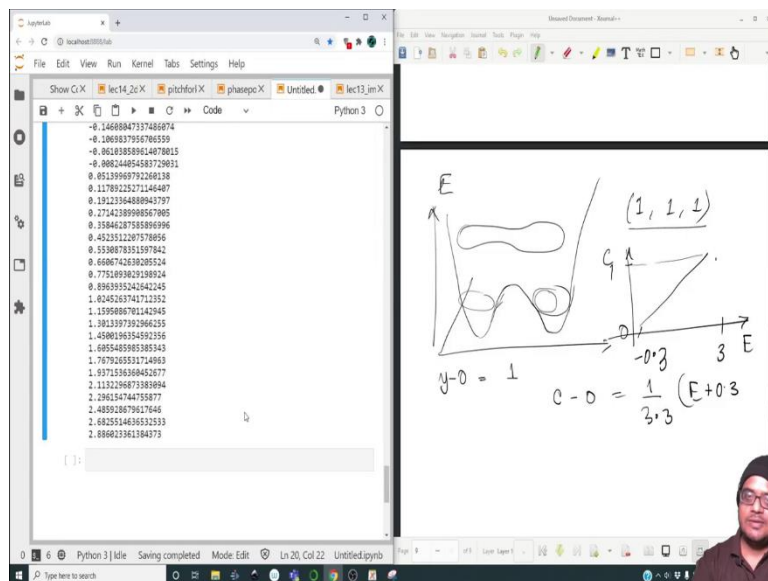
We can in fact, invert this to get some other color scheme then this looks this looks good.

(Refer Slide Time: 58:24)



This is how you can make this kind of curves and you can show the entire physics of how changing the energy is changing how the trajectories look like in the x, y energy space ok.

(Refer Slide Time: 58:42)



So, with this we end this particular lecture. There is been quite a lot of things we have discussed. In the assignments there will be questions which will ask you to make use of these tools to delve deeper into certain problems which are of interest to biologists or it deals with some other kinds of problems which you might have done in class 11 for example. And, it will hopefully whatever you have done it will enable you to delve deeper into the science and the physics of it.

So, with this I end this particular session. I will see you again next time. Bye.