

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

Lecture – 17
2D phase portraits - limit cycles

(Refer Slide Time: 00:35)

```
[ ]: import numpy as np;
import matplotlib.pyplot as plt;
plt.rcParams.update({'text.usetex': True});
%config InlineBackend.figure_format = 'svg'
from IPython.display import Interactive
from scipy.integrate import solve_ivp
from mayavi import mlab

[ ]: def show_traj(x0=0.9,y0=1.0, a = 0.8):
x = np.linspace(-3, 3, 15); y = np.linspace(-3, 3, 15);
X, Y = np.meshgrid(x, y);
U = Y;
V = X-Y**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 ([x[1], x[0]-x[0]**3]);

tspan = [0,20]
#t0 = -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)
```

Hi everyone. In this lecture, we are going to continue on our path and study some more non-linear systems. Let us begin by considering what is called as the Manta-Ray system and the reason it is called as the Manta-Ray system because eventually, the phase portrait does resemble the sea animal which is called as a Manta-Ray and the equation which describes this is $\dot{x} = y - y^3$; $\dot{y} = -x - y^2$ ok. So, first things first, what are the fixed points?

So, obviously, the origin is a fixed point; so, $(0, 0)$ and we can see that at the fixed-point \dot{x} and \dot{y} will be 0. So, $y(1 - y^2)$ will be equal to 0, it implies $y = \pm 1$ and if $y = \pm 1$, x will be equal to $-y^2$ and so, the other two fixed points are $(-1, 1)$ and $(-1, -1)$.

So, we can already start making the phase space. This is the x axis; this is the y axis. So, this is one fixed point of this floor and the other fixed point is this and this. Now, let us apply linearization and see what behavior we expect at the fixed points ok.

(Refer Slide Time: 02:15)

The image shows a Jupyter Notebook window with the following Python code:

```
[1]: import numpy as np;
import matplotlib.pyplot as plt;
plt.rcParams.update({'text.usemathtex': True});
%config InlineBackend.figure_format = 'svg'
from IPython.display import Interactive
from scipy.integrate import solve_ivp
from mayavi import mlab

[2]: x = 0; y = 0;
A = np.array([[0, 1-3*y**2], [-1, -2*y]])
l, v = np.linalg.eig(A)
print(l)
print(v)

[[0.+1.j 0.-1.j]
 [0.70710678+0.j 0.70710678-0.j]
 [0.
  0.70710678j 0.
 -0.70710678j]]

[3]: def show_traj(x0=0, y0=1, a = 0.8):
x = np.linspace(-3, 3, 15); y = np.linspace(-3, 3, 15);
X, Y = np.meshgrid(x, y);
U = Y;
V = X-Y**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 [x[1], x[0]-x[0]**3];

tspan = [0, 20]
```

Handwritten notes on the right side of the notebook:

Mantle-Ray

$$\begin{cases} \dot{x} = y - y^3 \\ \dot{y} = -x - y^2 \end{cases} \quad y(1-y^2) = 0 \Rightarrow y = \pm 1$$

origin (0,0) (-1,1) (-1,-1)

Centre

$$J_{(0,0)} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$
$$J = \begin{pmatrix} 0 & 1-3y^2 \\ -1 & -2y \end{pmatrix}$$

So, the Jacobian of this particular set of equations will be $0, 1-3y^2$. This will be -1 and this will be $-2y$ ok. So, at the origin, the Jacobian will be $0, 1, -1$ and -2 , this will be 0 . We have already encountered such kinds of Jacobian and it is quite obvious that this Jacobian will be that of a center; meaning, the orbits near the origin will resemble a center.

What about this point and this point? So, that instead of doing this by hand let us go to the computer and do it on the PC. So, let me run the initial script that we have in every program and it takes a while to load initially, that is ok. So, let me define A as np.dot array. Let me create the matrix and this will be $0, 1-3y^2$ and this will be $-1, -2y$.

Let me define $x = 0, y$ and $y = 0$ and when A is this, let me then get $l, v = \text{np.linalg.eig}$ of A . Let me then print l and let me also print v . So, let me run this ok. So, for the case $x = 0, y = 0$, the eigenvalues are $+i$ and $-i$ and that is quite obvious. Eigenvectors do not really matter in this case because it is the center.

(Refer Slide Time: 04:34)

Python Code:

```
[1]: import numpy as np;
import matplotlib.pyplot as plt;
plt.rcParams.update({'text.usetex': True});
%config InlineBackend.figure_format = 'svg'
from ipynbutils import interactive
from scipy.integrate import solve_ivp
from mayavi import mlab

[2]: x = -1; y = 1;
A = np.array([[0, 1-3*y**2],[-1, -2*y]])
l, v = np.linalg.eig(A)
print(l)
print(v)

[[ 0.73205081 -2.73205081]
 [ -0.59097068  0.59097068]
 [ -0.34372377  0.86089822]]

[3]: def show_traj(x0=0.9,y0=1.0, a = 0.8):
x = np.linspace(-3, 3, 15); y = np.linspace(-3, 3, 15);
X, Y = np.meshgrid(x, y);
U = Y;
V = X-Y**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 [[x[1], x[0]-x[0]**3]]

tspan = [0, 20]
```

Handwritten Notes:

Mantle-Ray

$$\begin{cases} \dot{x} = y - y^3 \\ \dot{y} = -x - y^2 \end{cases} \quad y(1-y^2) = 0 \Rightarrow y = \pm 1$$

origin (0,0), (-1,1), (-1,-1)

Jacobian matrix: $J = \begin{pmatrix} 0 & 1-2y \\ -1 & -2y \end{pmatrix}$

Eigenvalues: 0.73, -2.73

Centres: (-1,1), (-1,-1)

Now, let me change the x and y coordinates to (-1, 1). So, at this particular point, the eigenvectors are 0.73 and -2.73. So, the stable manifold is strongly attracting than the unstable one and the direction of the unstable manifold is along this. So, +1 and - this or something like this ok.

So, this is the unstable manifold and the stable manifold is along this direction. So, it is more like this. Sorry, it has to be something like this ok. So, this is the stable manifold and this is the unstable manifold.

(Refer Slide Time: 05:35)

Python Code:

```
[1]: import numpy as np;
import matplotlib.pyplot as plt;
plt.rcParams.update({'text.usetex': True});
%config InlineBackend.figure_format = 'svg'
from ipynbutils import interactive
from scipy.integrate import solve_ivp
from mayavi import mlab

[4]: x = -1; y = -1;
A = np.array([[0, 1-3*y**2],[-1, -2*y]])
l, v = np.linalg.eig(A)
print(l)
print(v)

[[ -0.73205081  2.73205081]
 [ -0.59097068  0.59097068]
 [ -0.34372377  0.86089822]]

[3]: def show_traj(x0=0.9,y0=1.0, a = 0.8):
x = np.linspace(-3, 3, 15); y = np.linspace(-3, 3, 15);
X, Y = np.meshgrid(x, y);
U = Y;
V = X-Y**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 [[x[1], x[0]-x[0]**3]]

tspan = [0, 20]
```

Handwritten Notes:

Mantle-Ray

$$\begin{cases} \dot{x} = y - y^3 \\ \dot{y} = -x - y^2 \end{cases} \quad y(1-y^2) = 0 \Rightarrow y = \pm 1$$

origin (0,0), (-1,1), (-1,-1)

Jacobian matrix: $J = \begin{pmatrix} 0 & 1-2y \\ -1 & -2y \end{pmatrix}$

Eigenvalues: 0.73, -2.73

Centres: (-1,1), (-1,-1)

Direction of flow: $\frac{dx}{dt} = y - y^3$, $\frac{dy}{dt} = -x - y^2$

What about $x = -1$ and $y = -1$? So, the eigenvalues are the reverse. So, it is -0.73 and 2.73 and obviously, the eigen direction will also be a mirror image of this ok. So, it will be something like this or something like this ok.

So, the key point to note over here is the eigenvalues are simply flipping their signs and the direction as well ok. And so, this is what we get from linearization and we recall from the one of the examples earlier that centers are not really something which are quite robust geometrically.

Meaning, linearization of this particular equation is sort of dropping the influence of this cubic term, it is dropping the influence of this quadratic term. But in the case of time reversible systems that is in this particular set of equations, if we make the following change, if we let t go as minus t right and in this particular case, y go as $-y$. What do we have?

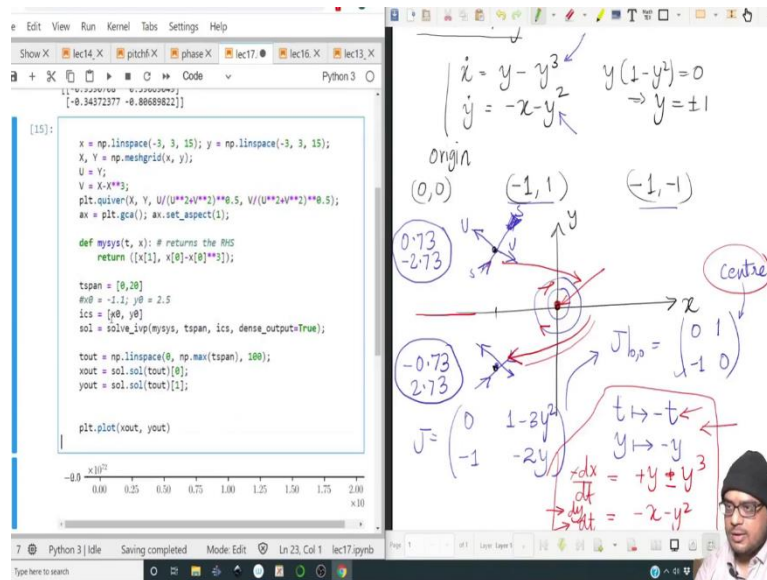
So, in this particular case, let us see what the equation transforms to. The first equation is $-\frac{dx}{dt}$ because we have made this particular transformation. This will be $-y + y^3$ and so, this minus sign cancels out and we get the same equation ok. I have flipped over the signs of y as well. Similarly, for this equation $\frac{dy}{dt}$, the sign will remain unchanged because there is a change in sign of this and the change in sign of this simultaneously. So, this sign remains unaltered. This is $-x - y$.

So, the equations are unaltered under this kind of time reversal symmetry and this symmetry implies the problem, if a trajectory goes like this in the lower half that is if time is increasing or the trajectory goes like this; in the upper half as time reduces as time reverses, the trajectory must go something like this. And the presence of this time reversal in implies that this particular center that we do not know whether it is robust or not, will actually turn out to be a robust center ok.

So, center is robust in the presence of time reversal symmetries and let me just tell this much that all this is much beyond the scope of this particular course. But I want to throw out some key ideas of non-linear dynamics. Obviously, I will link some useful videos in which the theory and the mathematics behind this will be discussed in greater details.

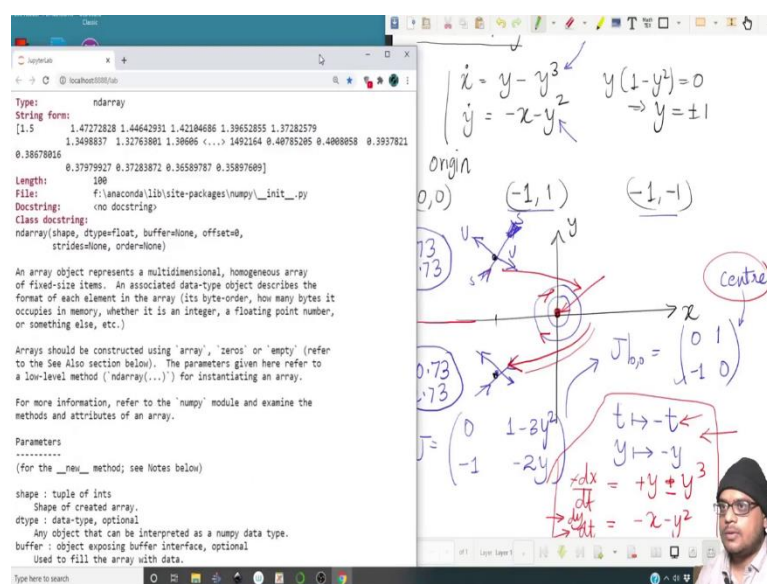
But I hope through this, you get an appreciation of how you can actually make an in-depth study with the help of Python org and octane ok. So, this is what we expect ok, a trajectory like this or trajectory like this, this implies time symmetry.

(Refer Slide Time: 09:38)



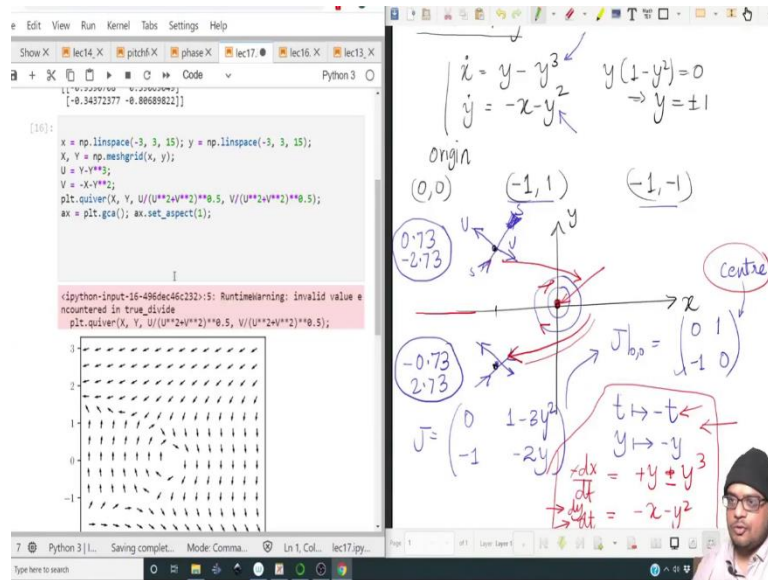
So, now let us go ahead and plot this particular phase diagram. So, let us go over here and change this particular function. In fact, I do not want to plot trajectories on this. I will let you do that.

(Refer Slide Time: 09:47)



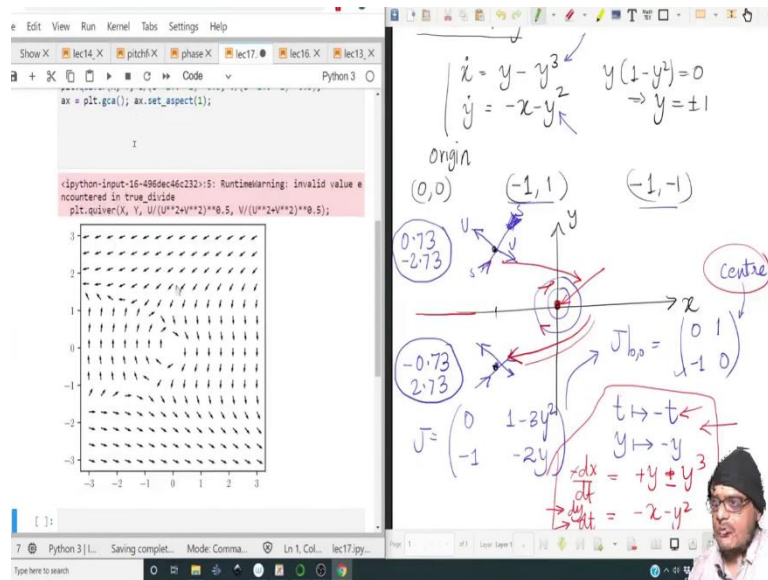
Let me. So, let me quickly wrap everything as a normal code. We will show the quiver plot and will get rid of the trajectories and will directly plot the stream lines ok.

(Refer Slide Time: 10:09)



So, let me just fix this. This has to be $y - y^3$. This has to be $-x - y^2$ ok.

(Refer Slide Time: 10:26)



And let me just run this and show what happens ok. So, over here, we do expect an orbit somewhere over here and there does appear to be an attracting point, then points which fly over to minus infinity, points which fly over to infinity. So, looking at the vector plot, it

does appear I mean one can sort of gauge what is happening in this flow field. Let us plot the stream lines to really drive home the message.

(Refer Slide Time: 11:03)

The screenshot shows a Python IDE with the following code:

```

print(1)
print(v)
[-0.7305981  2.7305981]
[-0.9398768  0.5906594]
[-0.34372377  0.88089822]]

[16]:
x = np.linspace(-3, 3, 100); y = np.linspace(-3, 3, 100);
X, Y = np.meshgrid(x, y);
U = X**2*Y**3;
V = -X-Y**2;
#plt.quiver(X, Y, U/(U**2+V**2)**0.5, V/(U**2+V**2)**0.5);
plt.stream
ax = plt.gca(); ax.set_aspect(1);

```

Handwritten notes on the right side of the slide include:

- Equations: $\dot{x} = y - y^3$, $\dot{y} = -x - y^2$, and $y(1-y^2) = 0 \Rightarrow y = \pm 1$.
- Origin: $(0,0)$, $(-1,1)$, $(-1,-1)$.
- Jacobian matrix: $J = \begin{pmatrix} 0 & 1-2y^2 \\ -1 & -2y \end{pmatrix}$.
- Phase portrait showing trajectories and a center point.
- Time reversal: $t \mapsto -t$, $x \mapsto -x$, $y \mapsto -y$.
- Derivatives: $\frac{dx}{dt} = +y - y^3$, $\frac{dy}{dt} = -x - y^2$.

So, let me take 100 and yeah, 100 points. I am using the code. I will not plot the vectors anymore because in the streamline plot, the vectors will be apparent as it is.

(Refer Slide Time: 11:25)

The screenshot shows a Python IDE with the following code:

```

[1]: import numpy as np;
import matplotlib.pyplot as plt;
plt.rcParams.update({'text.usetex': True});
%config InlineBackend.figure_format = "svg"
from IPython.display 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**2*Y**3;
V = -X-Y**2;
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**2*Y1**3;
V1 = -X1-Y1**2;
#seedpoints = np.array([[0, 1.5], [2.5, 0]]);
plt.streamplot(X1, Y1, U1, V1, integration_direction='forward', density=3);
plt.contour(X1, Y1, U1, levels=[0.0]);
plt.contour(X1, Y1, V1, levels=[0.0], colors='g');

```

Handwritten notes on the right side of the slide are identical to the previous slide, showing the equations, Jacobian matrix, and phase portrait.

(Refer Slide Time: 11:29)

The screenshot shows a Jupyter Notebook interface with a Python script and a handwritten slide. The Python script defines a vector field $\vec{v} = (x - y^3, -x - y^2)$ and plots it using `plt.streamplot`. The handwritten slide contains the following content:

- Equations: $\dot{x} = x - y^3$, $\dot{y} = -x - y^2$, and $y(1 - y^2) = 0 \Rightarrow y = \pm 1$.
- Origin: $(0, 0)$, $(-1, 1)$, $(-1, -1)$.
- Phase portrait: A plot of the vector field with arrows indicating the direction of flow. The origin is labeled "Origin" and the center is labeled "Centre".
- Jacobian matrix: $J_{top} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$.
- Jacobian matrix: $J = \begin{pmatrix} 0 & 1 - 3y^2 \\ -1 & -2y \end{pmatrix}$.
- Directional derivatives: $\frac{dx}{dt} = +y - y^3$ and $\frac{dy}{dt} = -x - y^2$.

So, let me remove this quiver plot and let me do `plt.streamplot`. In fact let me copy the snippet from one of the earlier codes. I do not have to rewrite everything.

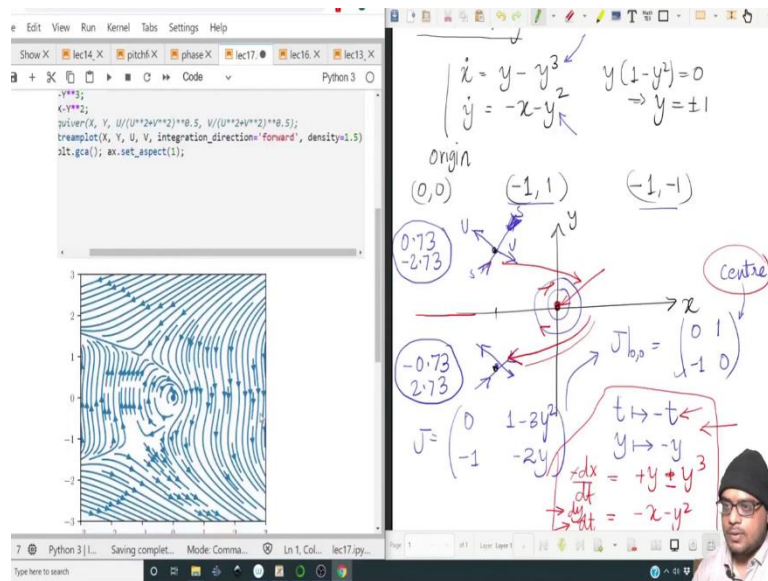
(Refer Slide Time: 11:37)

The screenshot shows a Jupyter Notebook interface with a Python script that has a `NameError` and a handwritten slide. The Python script defines a vector field $\vec{v} = (x - y^3, -x - y^2)$ and plots it using `plt.streamplot`. The handwritten slide contains the following content:

- Equations: $\dot{x} = x - y^3$, $\dot{y} = -x - y^2$, and $y(1 - y^2) = 0 \Rightarrow y = \pm 1$.
- Origin: $(0, 0)$, $(-1, 1)$, $(-1, -1)$.
- Phase portrait: A plot of the vector field with arrows indicating the direction of flow. The origin is labeled "Origin" and the center is labeled "Centre".
- Jacobian matrix: $J_{top} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$.
- Jacobian matrix: $J = \begin{pmatrix} 0 & 1 - 3y^2 \\ -1 & -2y \end{pmatrix}$.
- Directional derivatives: $\frac{dx}{dt} = +y - y^3$ and $\frac{dy}{dt} = -x - y^2$.

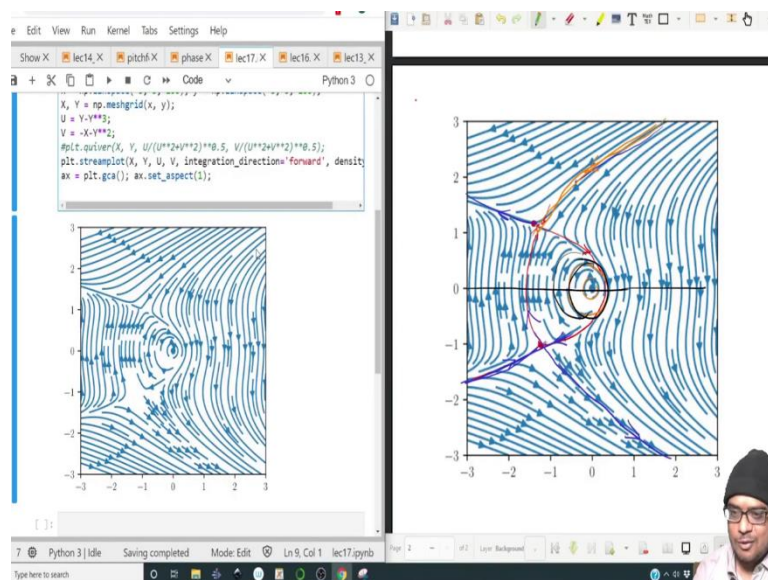
So, let me run this ok.

(Refer Slide Time: 11:42)



There is an error. So, X_1 is not defined ok. So, this has to be X, Y, U and V alright. So, what do we see? This is the direction of the stable manifold and let me take this diagram to our notebook. So, we can discuss it ok.

(Refer Slide Time: 12:09)



So, we had predicted that (-1, -1) is one fixed point; (-1, 1) is the other fixed point. There appears to be a trajectory which would join it like this and there appears to be a trajectory

which would go around like this ok. So, what you can do is later on use this neighborhood as an initial condition, see how the trajectory goes.

So, this is the direction of the stable manifold. This is the direction of the stable manifold. So, this entire thing stable. This is the direction of the unstable manifold. This is the direction of the unstable manifold and this is the direction of the stable manifold ok.

So, we have already predicted these and there are fixed orbits over here. There are fixed orbits over here and because of the time symmetry, whatever we have predicted through linearization still holds true. There is no issue going on ok. So, this whole plot sort of looks like a mandatory, if you squint yes squinter eyes hard enough and yeah.

So, this particular problem highlights an important aspect that of time reversal symmetry and I hope you can learn more on that in one of the later and one of the links that I provided below.

(Refer Slide Time: 14:12)

The python and octave notebooks can be downloaded fr

```

[28]:
x = np.linspace(-6, 6, 100); y = np.linspace(-3, 3, 100);
X, Y = np.meshgrid(x, y);
U = Y;
V = -np.sin(X);
#plt.quiver(X, Y, U/(U**2+V**2)**0.5, V/(U**2+V**2)**0.5);
plt.streamplot(X, Y, U, V, integration_direction='forward', density
ax = plt.gca(); ax.set_aspect(1);

```

Simple pendulum

$$\ddot{\theta} + \sin \theta = 0$$

$$\ddot{x} + \sin x = 0$$

$$x = y$$

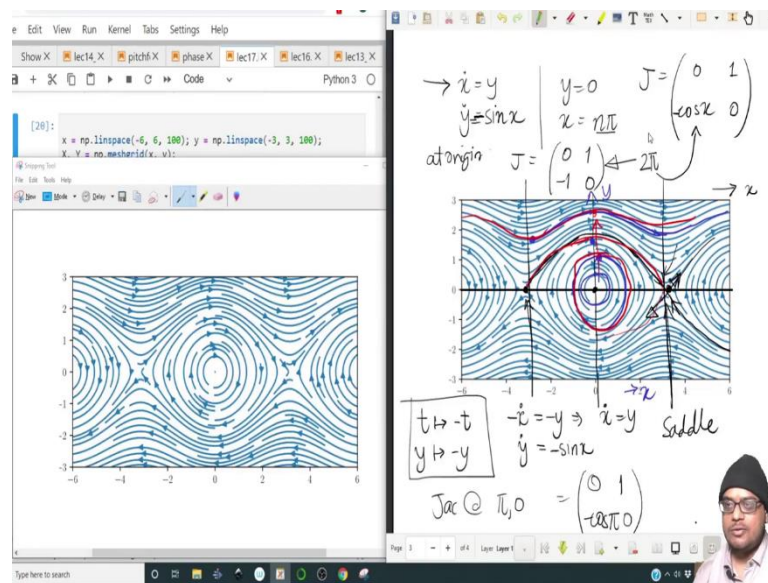
$$\dot{y} = -\sin x$$

Let us revisit a very popular example taught in grade school that of a simple pendulum. So, simple pendulum is described in terms of $\ddot{\theta} + \sin \theta = 0$. Well, obviously, this is a very I mean simplified form of it, it is a dimensionless form of it. So, let us write the following.

Let me first of all represent θ as x ; so, $\ddot{x} + \sin x = 0$. Let $\dot{x} = y$ and \dot{y} will be equal to $-\sin x$ ok. So, this is the system that we are dealing with. And yeah, let us have a look at how this phase plane looks like and this is quite simple.

This is simply going to be Y and this is simply going to be $-\sin X$. So, let us plot this ok. Let me increase the span in the x axis; let me make it from -6 to 6 . So, look at this. What do you see and what do you interpret from this particular diagram?

(Refer Slide Time: 15:49)



Let me take this to the notebook. So, the equation was $\dot{x} = y$ and $\dot{y} = -\sin x$. So, quite naturally, this is the x axis and the fixed points correspond to $y = 0$ and $x = n\pi$ ok. So, this is $(0, 0)$ or obviously, for $n = 0$ this is the point. This is equal to $\pi - \pi$, this is equal to $+\pi$.

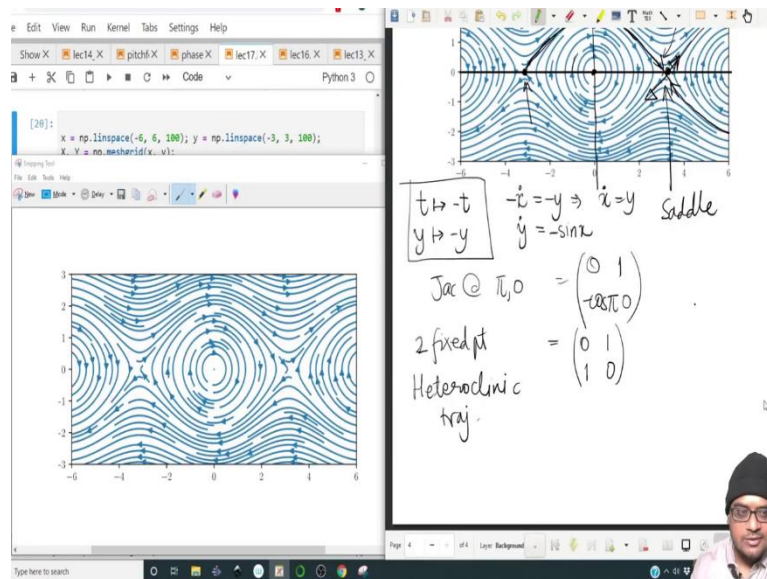
So, all these are sort of the fixed points of the system and it is obviously, infinitely periodic in the x direction. Now, what about the Jacobian? So, the Jacobian for the system is given by $0, 1, \cos x$ and this will be 0 . So, at the origin, so the this is minus $\sin x$ here. So, this is minus $\sin x$, I made a sign mistake. This has to be minus $\cos x$. So, at origin, Jacobian is equal to $0, 1, \text{minus } 1, 0$. This obviously represents a center.

Now, do we have time reversal symmetry? Let us see. Let us do this transform. So, obviously, this will be $-\dot{x}$, this will be $-y$. So, this reduces to the same equation. This is the first equation over here and the second equation \dot{y} will be unchanged because y and t both have a sign reversal and $-\sin x$ remains the same.

So, the equations are unaltered because of this time reversal symmetry. As a result of which the center that we predict at the origin is a stable center or I mean not a stable center, but it is a robust center ok. So, it is a robust center and this causes the streamlines to go like this.

And obviously, this will repeat after every 2π . The repeating is because of this particular cos function, it is periodic after a time period 2π alright; that is fine. What about these points? They do not appear to be centers. So, what are they? So, let us substitute π over here. So, let us substitute only π over here.

(Refer Slide Time: 18:37)

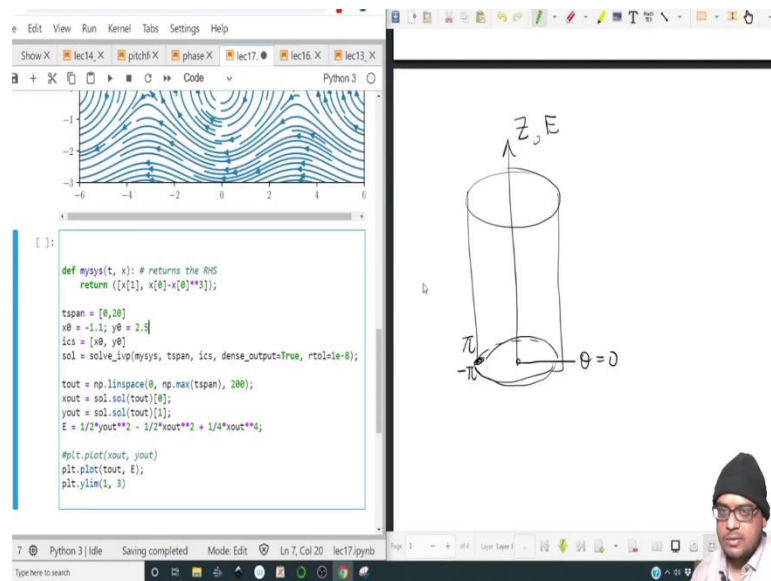


So, the Jacobian over at $(\pi, 0)$ that will be $\begin{bmatrix} 0 & 1 \\ -\cos x & 0 \end{bmatrix}$ and this is equal to this. So, obviously, it has 2 eigenvectors, eigenvalues, both of them are real; it is ± 1 . So, this represents the unstable manifold. This represents the stable manifold ok. Trajectories are attracted towards this along this particular trajectory; they are attracted towards this fixed point along this trajectory.

Now, this trajectory is also an unstable. So, for this particular point, this is an unstable manifold; but that same manifold is a stable manifold for this particular point. So, this particular trajectory which connects 2 fixed points ok. So, in this example 2 fixed points are being connected by a trajectory and hence, it is called as a Heteroclinic trajectory. And obviously, these points are saddle points and it is a saddle point because one of the eigenvalues is positive and one of them is negative; both are real

Similarly, between these two points, there was a trajectory which connected it like this. So, this is also heteroclinic trajectory and this is also heteroclinic trajectory; it connects these two fixed points ok. So, now in reality, whatever is going on, it is supposed to be infinitely periodic. So, the practical aspect is the phase space must be something on a cylinder ok.

(Refer Slide Time: 20:36)

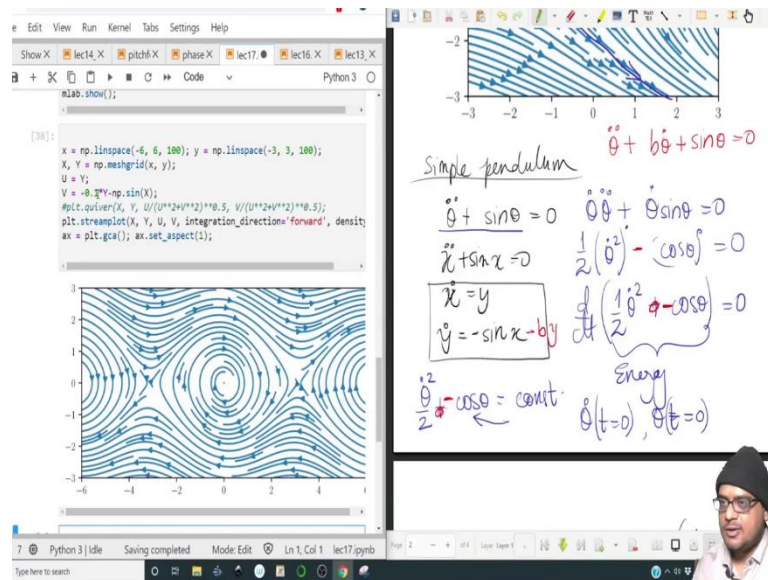


So, the phase space must be something on a cylinder. So, how can we plot it? So, on us you can imagine that you wrap this phase portrait on a cylinder that is you take a periodic part, you take one period out of it and you wrap it around the cylinder. So, how do you wrap it around the cylinder? So, let this be theta equal to 0.

This is the z axis ok. So, or I mean you can yeah, let this be theta equal to 0. So, this point over here, you are going forward in theta and this point, you are going backward in theta. So, this point could be pi and this point could be minus pi as well. So, the pi and minus pi points are sort of connected through a cylinder.

On the z axis, we can plot the energy. Why do we say that we can plot the energy? And the reason is this particular system is a conservative system. So, by conservative system, we mean that the energy along a trajectory is going to be conserved.

(Refer Slide Time: 22:00)



So, what is this? Let us multiply everything by $\dot{\theta}$. So, this is $\dot{\theta}\ddot{\theta} + \dot{\theta}\sin\theta = 0$. So, this is $\frac{1}{2}(\dot{\theta}^2) - \cos\theta = 0$. Essentially, it means $\frac{d}{dt} \left(\frac{1}{2}\dot{\theta}^2 + \cos\theta \right) = 0$.

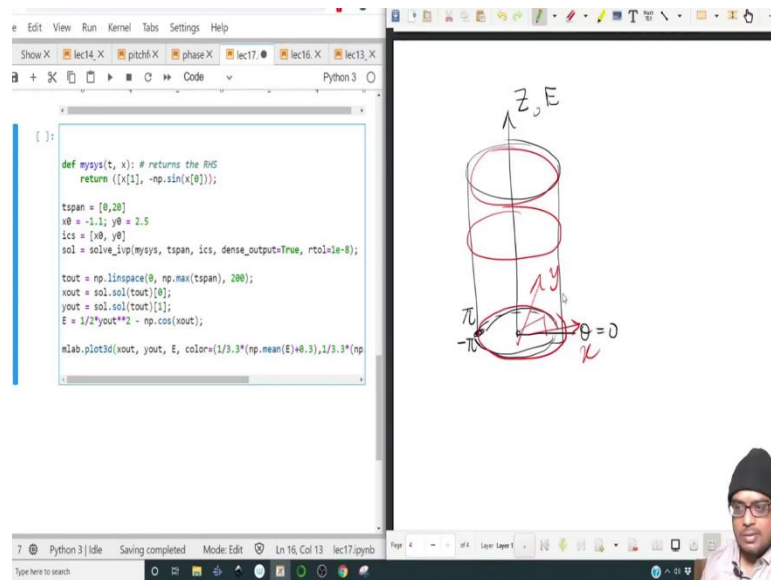
This is what it means, the energy. So, this is this represents the energy of the system and if this is 0, it implies that $\frac{1}{2}\dot{\theta}^2 + \cos\theta$ is constant. So, if I choose an initial condition that is $\dot{\theta}$ at $t = 0$ and θ equal at $t = 0$, if they are fixed, then the evolution of the trajectory will be constrained along iso e lines. That is what a conservative system means.

That is once I choose an initial condition over here, it will move along a trajectory so that as the x and as the y are changing, the energy along a trajectory, it remains conserved ok. The energy along this, it will remain conserved. So, if you add some energy to the system, you get jump you get boosted to a different trajectory and then, you have a different behavior and the behavior changes fundamentally as we cross the heteroclinic trajectory. So, this is the heteroclinic trajectory, where it is trying to make this closed loop.

But once you have enough energy to cross it, you simply loop over everything. You do not sort of have a have a situation, where you have 0 velocity because you are always traveling along non-zero velocities. Here there are points where you are crossing 0 velocity.

So, in that case, let us try to represent whatever this is in a 3 D diagram. So, let us copy the appropriate snippet. We had written out the snippet in the previous lecture. So, let me copy this ok. So, we have this we have a tspan. Let me just; let it be 20, no problem.

(Refer Slide Time: 24:47)



So, now let me change the system. The system is \dot{x} is equal to this and this is $-\sin x$ and the energy will be half the expression for energy is $\frac{1}{2} y^2$ plus I think ok. We have made a sign mistake because there will be a negative sign over here, this has a negative sign over here; it is a small oversight on my part. But regardless, the discussion does not change; you can simply correct it over in the code. So, this will be minus np dot cos of xout ok.

So, what do we need to plot? Let me grab hold of the maya command. Have we imported maya? Yes, we have imported mayavi. So, we will plot. So, we want to plot this cylinder. So, the cylinder is actually what? It is going to be it is a unit cylinder. But what are the coordinates of the cylinder ok?

So, why it does not play any role in this access, instead of y we are simply plotting this and we are plotting it for different energy levels? As energy levels change, we should see a different in trajectory. So, this will be simply. So, if the radius of the cylinder is 1. So, the x-y coordinate for this particular cylinder. So, the x y coordinate will be simply cos theta and sin theta.

(Refer Slide Time: 26:43)

```
[ ]:  
  
def mysys(t, x): # returns the RHS  
    return ([x[1], -np.sin(x[0])]);  
  
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 - np.cos(xout);  
  
mlab.plot3d(np.cos(xout), np.sin(xout), E, color=(1/3.*np.mean(E
```

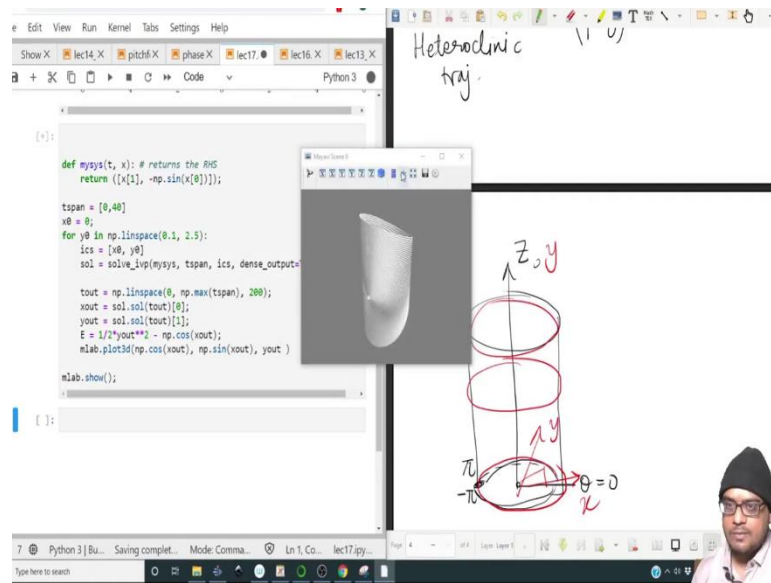
So, let me do this.

(Refer Slide Time: 26:55)

```
[ ]:  
  
dense_output=True, rtol=1e-8);  
, 200);  
  
out), E, color=(1/3.*np.mean(E)+0.3),1/3.*np.mean(E)+0.3),1-1/3
```

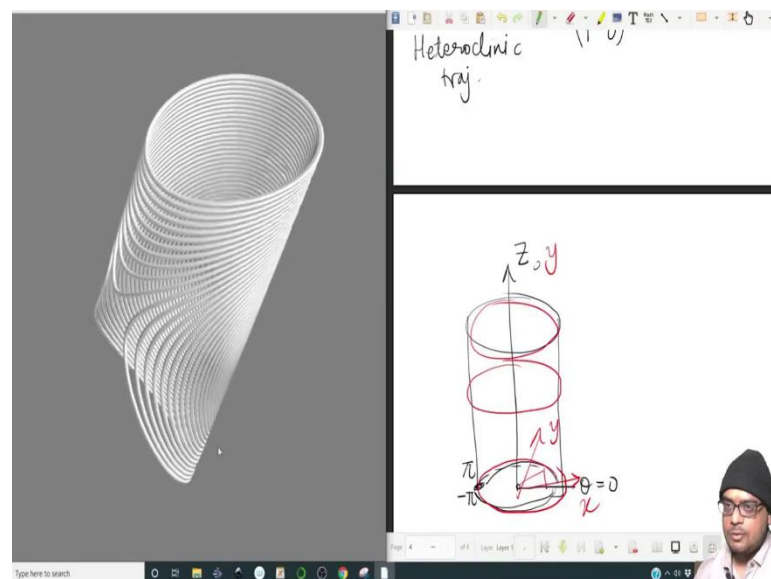
So, this will be $\text{np.cos}(x_{\text{out}})$, this will be $\text{np.sin}(x_{\text{out}})$ and on this z axis, we will have the energy and let me remove the color for now because we do not know the bounds for energy at this particular point in time.

(Refer Slide Time: 27:02)



So, let me remove the bound. In fact, in order to wrap this particular phase portrait, we should wrap it in such a way that the y axis now becomes the z axis. So, when we roll this, if you imagine rolling this on the surface of a cylinder, this should not be the energy; but yeah, this should be simply the velocity y . So, we can easily fix this, not an issue. This will be y out. So, let me run this. Let us see what happens ok.

(Refer Slide Time: 27:47)



So, ok, we have something which looks like this. Now, what is this? In fact, in order to make it completely symmetric, let me change this from -2.5 to 2.5 .

(Refer Slide Time: 28:03)

Python code in the notebook:

```
[36]:
def mysys(t, x): # returns the RHS
    return ([x[1], -np.sin(x[0])]);

tspan = [0, 40]
x0 = 0;
for y0 in np.linspace(-2.5, 2.5, 100):
    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];
    E = 1/2*yout**2 - np.cos(xout);
    mlab.plot3d(np.cos(xout), np.sin(xout), yout)
mlab.show();
```

Handwritten notes on the phase portrait:

$y = \sin x$, $x = \pi/2$
 at origin $J = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \leftarrow 2\pi$

$t \mapsto -t$
 $y \mapsto -y$

$-\dot{x} = -y \Rightarrow \dot{x} = y$ Saddle
 $\dot{y} = -\sin x$

Jac @ $\pi, 0 = \begin{pmatrix} 0 & 1 \\ -\cos \pi & 0 \end{pmatrix}$
 2 fixed pt = $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
 Heteroclinic

We should have something interesting ok.

(Refer Slide Time: 28:11)

3D plot of a periodic curve.

Handwritten notes on the phase portrait:

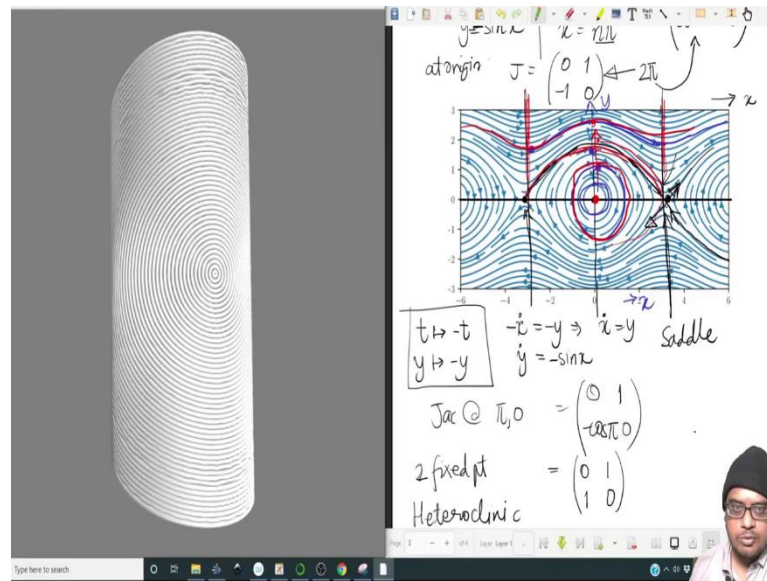
$-\dot{x} = -y \Rightarrow \dot{x} = y$ Saddle
 $\dot{y} = -\sin x$

Jac @ $\pi, 0 = \begin{pmatrix} 0 & 1 \\ -\cos \pi & 0 \end{pmatrix}$
 2 fixed pt = $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
 Heteroclinic traj.

So, look at this, look at the structure. We have essentially taken a periodic segment of this particular curve; this particular phase portrait and we have wrapped it around the cylinder that is you are joining this edge with this edge. So, this I, if you want to call it corresponds to this point, this is the I ok. Here is the I and as we go towards higher energy orbits, so each of the ring corresponds to a separate energy level ok.

So, as we go towards higher energy levels, the velocity is not going to 0. In fact, it is just making a round on top of the cylinder, this is these are the same energy levels; but with an initial condition which is below the x axis. Now, look at this trajectory in particular. Let me increase the number of points, so that we can have a clear interpretation. It will take a while to run ok.

(Refer Slide Time: 29:35)



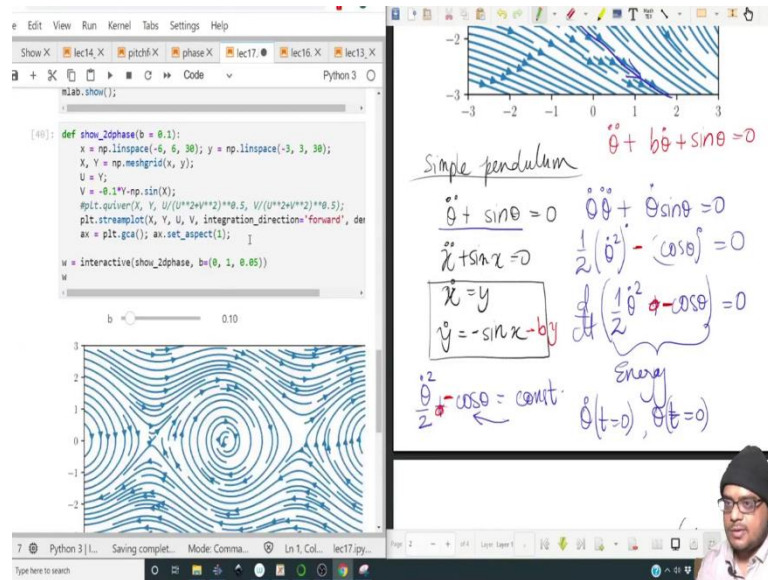
So, these closed orbits correspond to the orbits which are bounded by the heteroclinic trajectories. So, whatever so this heteroclinic trajectory will correspond to that trajectory which sort of just touches the back of this ok. If you take even more points, you will have a trajectory. So, this point over here is sort of corresponding to this point over here and this point over here is corresponding to this point over here ok. So, though this is the center, this is that particular center over here ok.

So, I hope you can appreciate how you can wrap things around in 3 D and get a deeper understanding of how things look. In fact, you can change the color scheme, we have discussed on how to change the color of this particular trajectory in my way and hopefully, you can make that change and plot this in a more colorful manner.

So, the color will correspond primarily to the energy of the system. We can convert this conservative system into a non-conservative system by means of adding a certain damping to this entire system ok.

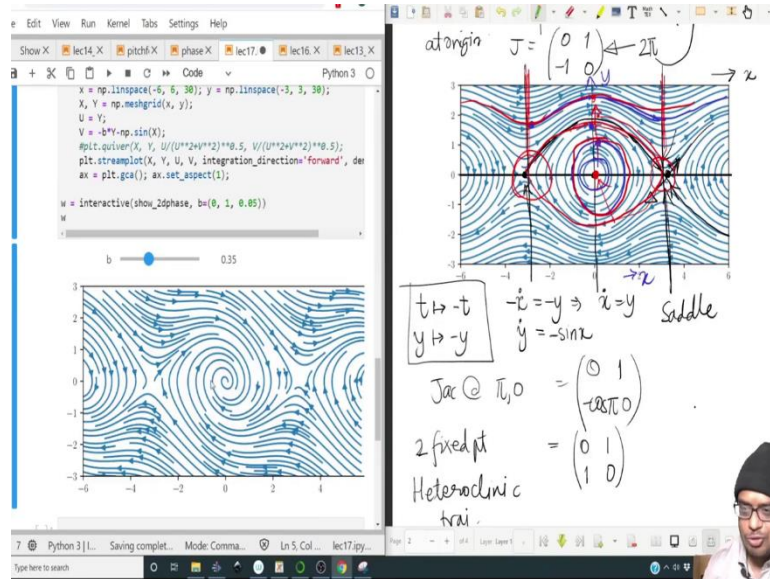
So, the equation for a damped oscillator will be $\ddot{\theta} + b\dot{\theta} + \sin \theta = 0$ and so, over here, we will simply have another term -by ok. So, it is not that difficult to implement really; it is simply a matter of say b is 0.1; 0.1y. Let us see how the trajectories look like; 0.1 is not a very strong ok.

(Refer Slide Time: 31:43)



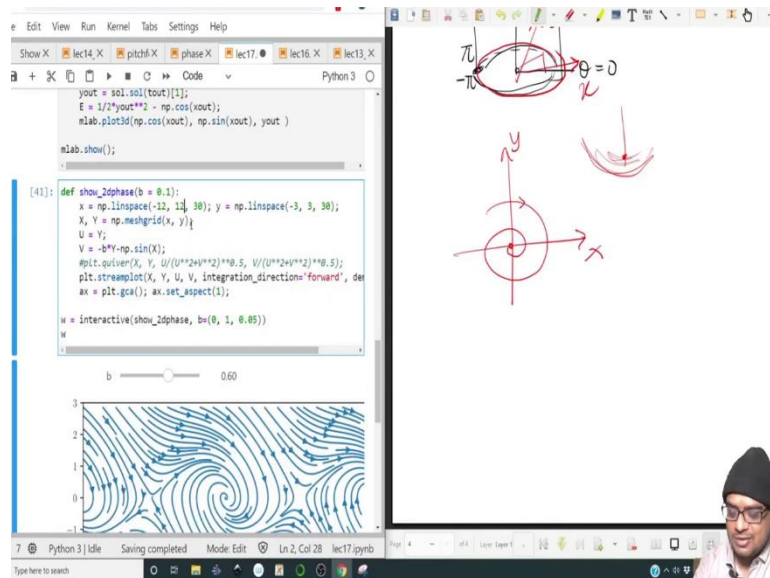
Let me actually do this. Let me take 30 points. Let me plot this and let me make, let me wrap this inside an interactive function. So, def show 2dphase b equal to say 0.1, let us wrap all of this inside the function. So, w equal to interactive show 2dphase, b, it will go from say 0 to 1 in steps of 0.05 and then, we will display the widget and let us see when we increase the damping, ok.

(Refer Slide Time: 32:40)



So, it has to be minus b ok. So, what do you observe? So, by changing or by introducing damping, the first thing that we observe is that the center is no longer a closed orbit. I mean there are no closed orbits near the center. So, the origin is no longer the center; the origin becomes a spiral and it spirals inwards.

(Refer Slide Time: 33:15)

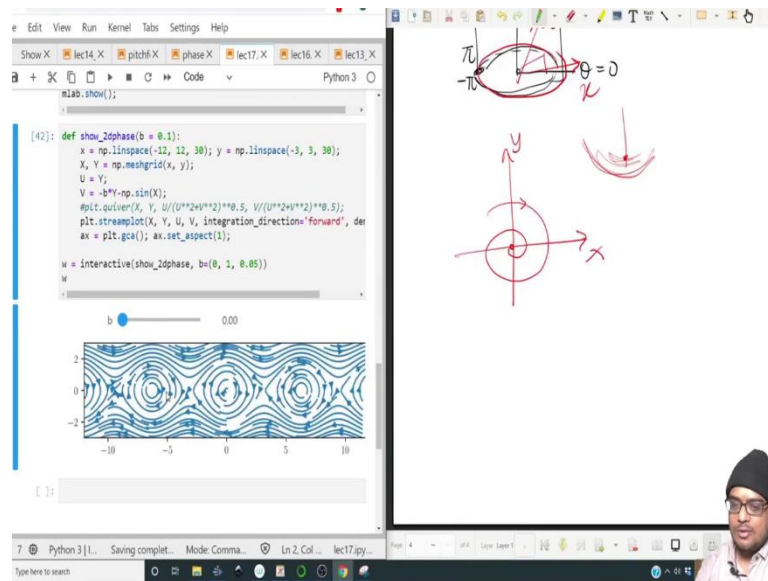


And the physical meaning of a trajectory spiraling inwards into the origin is that the velocity is reducing continually while rotating. So, the pendulum is still oscillating, but the velocity is going 0 and it is stopping at $\theta = 0$. So, this is the meaning of that incoming

spiral ok. There are trajectories if they have enough energies, they are sort of avoiding this particular spiral; but eventually, they will be trapped by some other spiral ok.

Let me increase the damping further. You will see that even trajectories which had larger energy, they are attracted towards this and really a trajectory being attracted by another spiral. So, the spiral on this part, let me show you that spiral; in fact, let me increase the range of this ok.

(Refer Slide Time: 34:13)

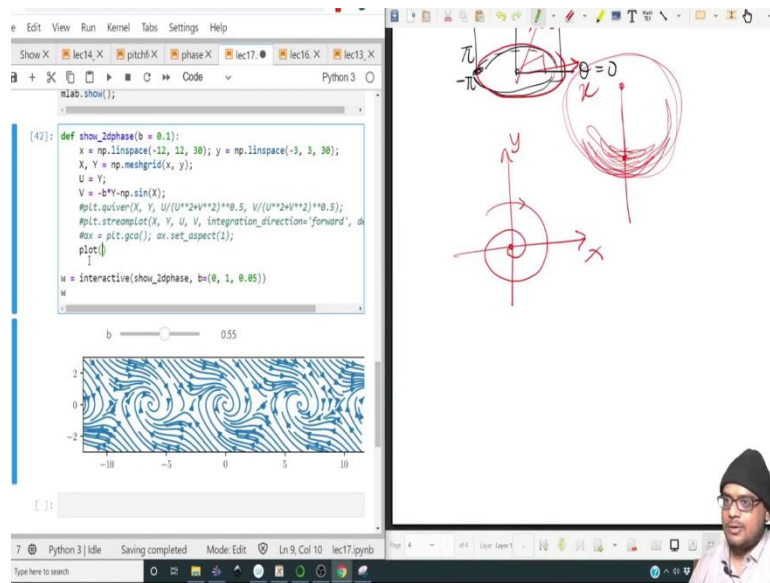


So, when b is 0, these are all individual centers ok. So, each of this point is a center. But as damping increases, so the trajectory which is starting over here, it goes like this and eventually, it spirals on into this. Trajectory which has started somewhere over here maybe it will meander above the centers and go to this.

But really speaking, this entire system is sort of periodic and when you wrap around, you simply realize that it is actually coming down to the same center and its quite obvious because the pendulum only has this much to play with. There is not a whole lot of centers ok.

So, even if you have larger energy, it will whirl around and then eventually, come back to orbit. So, this is what those things represent. So, try to interpret these equations on your own. You have all the tools. In fact, in order to interpret all of this, what you can also do is you can plot the time series as well right.

(Refer Slide Time: 35:34)

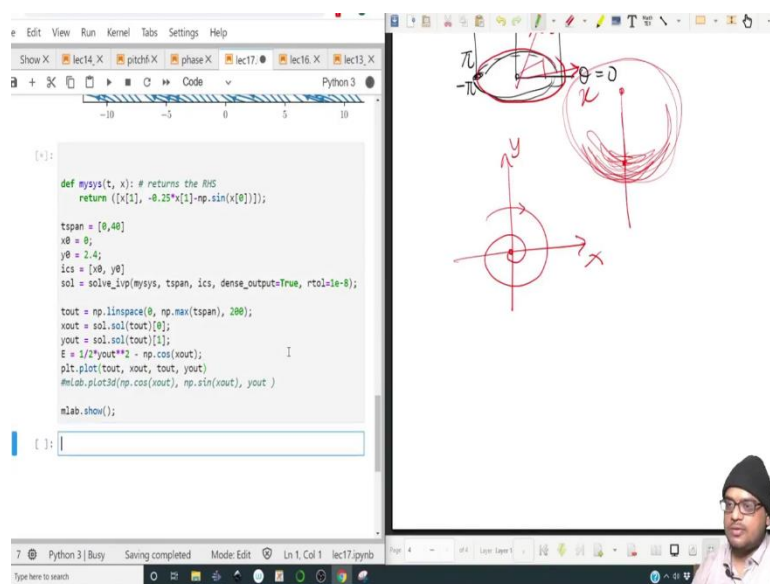


The screenshot shows a Jupyter Notebook interface. The code cell contains the following Python code:

```
[42]: def show_2dphase(b = 0.1):  
    x = np.linspace(-12, 12, 30); y = np.linspace(-3, 3, 30);  
    X, Y = np.meshgrid(x, y);  
    U = V;  
    V = -b*Y*np.sin(X);  
    #plt.quiver(X, Y, U/(U**2+V**2)**0.5, V/(U**2+V**2)**0.5);  
    #plt.streamplot(X, Y, U, V, integration_directions='forward',  
    #ax = plt.gca()); ax.set_aspect(1);  
    plot()  
w = interactive(show_2dphase, b=(0, 1, 0.05))  
w
```

Below the code is a slider for parameter b set to 0.55, and a stream plot showing blue streamlines in a 2D phase space. To the right of the notebook, there are handwritten red diagrams on a white background. One diagram shows a coordinate system with x and y axes and a circular path. Another diagram shows a similar coordinate system with a path that spirals inward towards the origin, labeled with $\theta = 0$ and π .

(Refer Slide Time: 35:46)



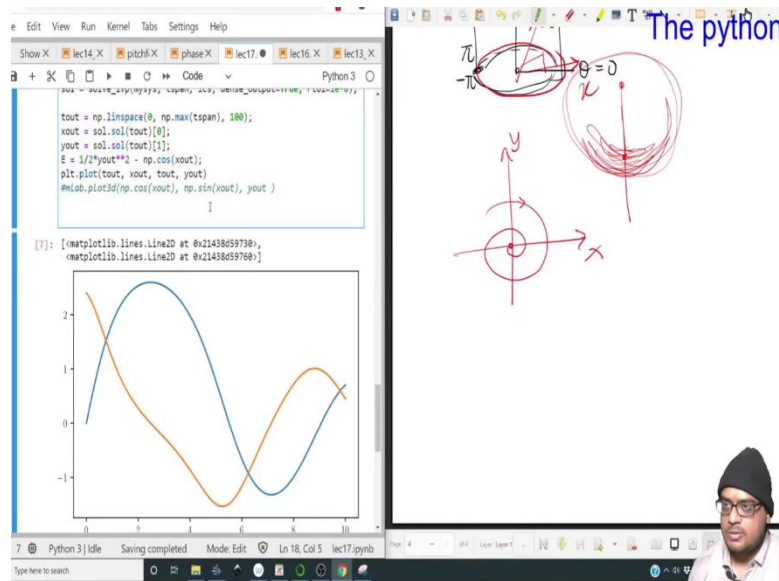
The screenshot shows a Jupyter Notebook interface. The code cell contains the following Python code:

```
[1]: def mysys(t, x): # returns the RHS  
    return [[x[1], -0.25*x[1]-np.sin(x[0])]];  
  
tspan = [0, 40]  
x0 = 0;  
y0 = 2.4;  
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 - np.cos(xout);  
plt.plot(tout, xout, tout, yout)  
#mlab.plot3d(np.cos(xout), np.sin(xout), yout)  
mlab.show();
```

Below the code is a time series plot showing x and y over time. To the right of the notebook, there are handwritten red diagrams on a white background, identical to the ones in the previous slide.

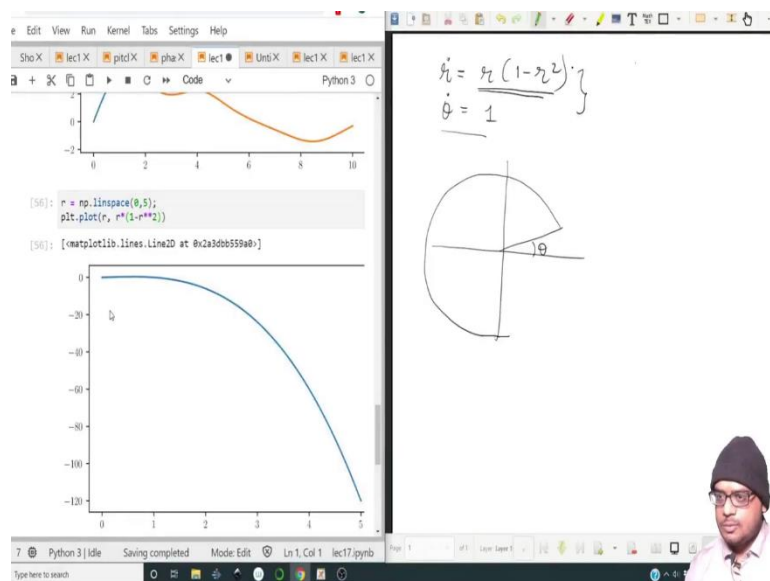
So, let me get rid of this stream plot for now, let me only show plot ah. So, this is let me copy this in order to show the time series, let me get this back to how it was ok. So, over here let me add a small damping. This has added damping; instead of making the 3d plot and instead of choosing different y_0 , say y_0 is 2.4, let me wrap let me indent this back and let us plot tout, xout, tout, yout and let us see what it looks like ok.

(Refer Slide Time: 36:29)



So, this is how the time series looks like. So, this is how you can plot the time series as well. Let us move on to the next example that of limit cycles. Moving on to fixed limit cycles.

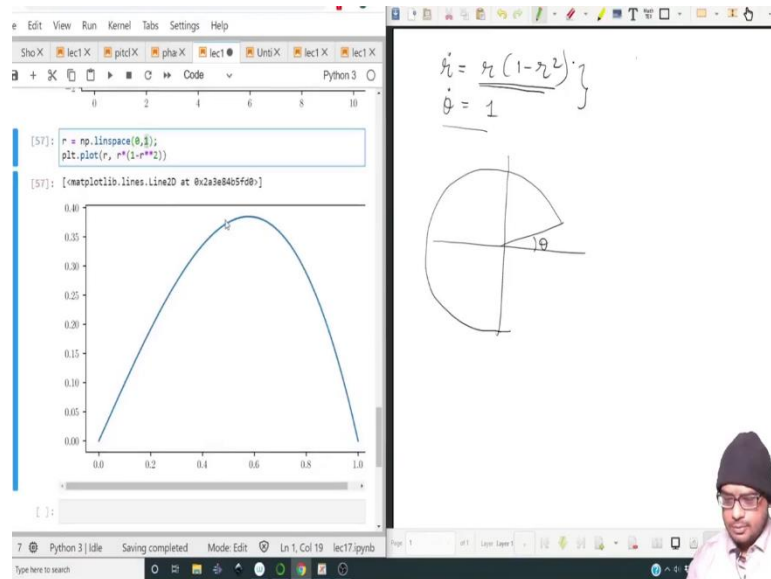
(Refer Slide Time: 36:50)



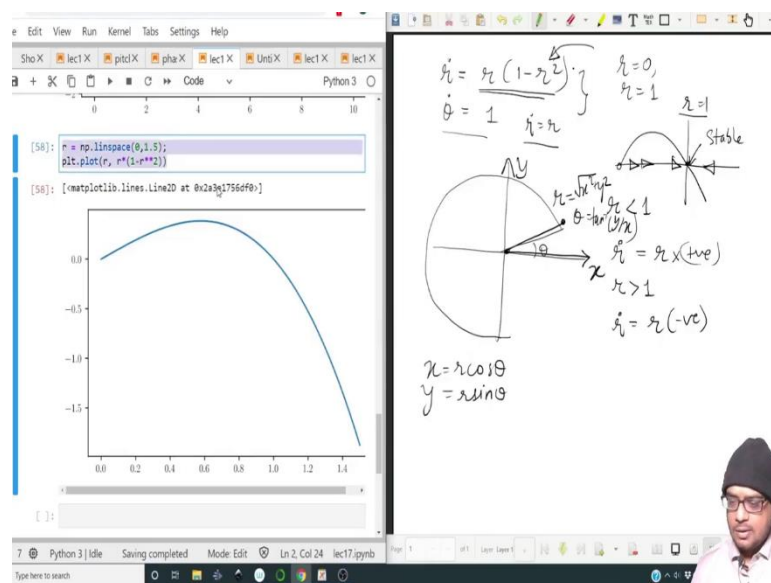
So, let me write down a vector flow. It is given by $\dot{r} = r(1 - r^2)$ and $\dot{\theta} = 1$. So, the meaning of $\dot{\theta} = 1$ is that in the polar coordinate, something is constantly changing; not something, but the quantity is the quantity theta is linearly increasing in time and what about this?

So, these two equations are linearly decoupled; meaning, \dot{r} is only a function of r albeit non-linearly while $\dot{\theta}$ is constant which is to say it is a function of θ only. Because of this, we can sort of use some of the tricks that we had learned in one-dimensional flows. So, when \dot{r} is something like this; let me just quickly plot it for you ok. So, it looks something like this and we are interested in small region.

(Refer Slide Time: 37:57)



(Refer Slide Time: 38:04)



So, let me just take 0 to 1 maybe slightly more yeah. So, it obviously, has a fixed point at $r = 0$ and a fixed point at r equal to plus minus 1; but in this case because it is a polar

coordinate system, the fixed point will be at $r = 1$ ok. So, it goes something like this, this fixed point is unstable because \dot{r} is positive.

So, the flow on a vector on a line, it will head towards the right while the flow over here is something like this. So, this is a stable fixed point; but now, if it were to be only \dot{r} equal to r ok, then what would happen is \dot{r} would exponentially grow and reach a certain value or it would blow over to infinity. But this particular term gives you a sort of feedback.

So, if as long as $r < 1$, this \dot{r} is some r times a positive number; that means, that r is going to grow. It is going to head along this line. But when r is greater than 1, this particular term becomes negative. So, you have \dot{r} equal to r times a negative number which means that the trajectories are going to head towards the left ok. So, this is r equal to 1.

So, this means that its always trying to non-linearly hunt for the value of r equal to 1 and as to how that manifests itself in the $x y$ plane remains to be seen. So, it is obvious that in an $x y$ plane, $r = \sqrt{x^2 + y^2}$ and $\theta = \tan^{-1} \frac{y}{x}$ ok. So, alternately, $x = r \cos \theta$ and $y = r \sin \theta$. So, this is just a simple transformation between the cylindrical coordinate system and or a Polar coordinate system and a Cartesian coordinate system.

(Refer Slide Time: 40:31)

The image shows a Jupyter Notebook interface with a Python script and a hand-drawn diagram. The Python script defines a function `mysys` that returns a vector field for a system of equations. The diagram shows a polar coordinate system with a circle at $r=1$, a vector field, and handwritten notes explaining the stability of the fixed point at $r=1$ and the transformation between polar and Cartesian coordinates.

```

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

tspan = [0,10]
x0 = 0;
y0 = 1;
ics = [x0, y0]
sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol=1e-8);

tout = np.linspace(0, np.max(tspan), 100);
rout = sol.sol(tout)[0];
tout = sol.sol(tout)[1];

xout = rout*np.cos(tout); yout = rout*np.sin(tout);

plt.plot(xout, yout)
#mlab.plot3d(np.cos(xout), np.sin(xout), yout)

```

Handwritten notes in the diagram include:

- $\dot{r} = r(1-r^2)$
- $\dot{\theta} = 1$
- $r = 0, r = 1$
- $r = 1$ is labeled as "Stable".
- A polar coordinate system with $r = \sqrt{x^2 + y^2}$ and $\theta = \tan^{-1}(y/x)$.
- For $r < 1$, $\dot{r} = r(\text{+ve})$.
- For $r > 1$, $\dot{r} = r(\text{-ve})$.
- Transformation equations: $x = r \cos \theta$ and $y = r \sin \theta$.

So, let us try to visualize how this flow looks like. So, let me just grab a snippet from here and let me. So, we have a differential equation in r and θ . What we will do is we will

solve it in r and theta and then, cast it in a Cartesian form ok. So, instead of x, let me call this r; let me write down the appropriate equations ok.

So, once I have changed it to this, this will no longer be xout; but this will be rout and this will be theta out and we can then we can remove this energy, we do not need it now. The rout can be converted to xout as rout times np dot cos of theta out and yout can be written as rout times np dot sin of theta out and then, finally, we will plot the trajectory and that will be xout another trajectory, but yeah.

So, trajectory in the face space. So, it will be xout comma yout. So, well before even that, the entire r equal to 1 circle is like a fixed contour. So, let me plot that and you can do the Jacobian analysis, but I am going to skip over that you can try it on your own.

(Refer Slide Time: 42:01)

So, let me plot this particular contour. Let me get rid of this weird line ok. So, x0 and y0 ok. So, these have to be recast in the r and theta terms.

(Refer Slide Time: 42:13)

The screenshot shows a Jupyter Notebook interface with a Python code cell and a handwritten slide. The code defines a function `mysys(t, r)` that returns the right-hand side of a differential equation. It then uses `np.linspace` to define a time span, `np.atan2` to calculate the initial angle θ_0 , and `solve_ivp` to solve the system. The solution is plotted using `plt.plot` and `mplot3d`.

The handwritten slide contains the following content:

- Equations: $\dot{r} = r(1-r^2)$, $\dot{\theta} = 1$, $\dot{r} = r$
- Stability analysis: $r=0$, $r=1$, $r=1$ is labeled as "Stable".
- Polar coordinate diagram: A circle in the $x-y$ plane with radius r and angle θ . The radial coordinate is $r = \sqrt{x^2 + y^2}$ and the angular coordinate is $\theta = \arctan(y/x)$.
- Conversion formulas: $x = r \cos \theta$, $y = r \sin \theta$.
- Sign conventions: $\dot{r} = r$ for $r < 1$, $\dot{r} = r(x + ve)$ for $r > 1$, and $\dot{r} = r(-ve)$ for $r > 1$.

So, if x_0 and y_0 are specified, we can say $r_0 = n$ or simply $x_0^2 + y_0^2$, this whole thing raised to 0.5 and θ_0 it will be a time 2 rather `np.atan2` and this will be y_0, x_0 ok. So, we have declared what the corresponding r and θ will be. Because we are specifying the initial condition in terms of the Cartesian point; x_0, y_0 ok.

(Refer Slide Time: 42:59)

The screenshot shows a Jupyter Notebook interface with a Python traceback error and a handwritten slide. The error message is:

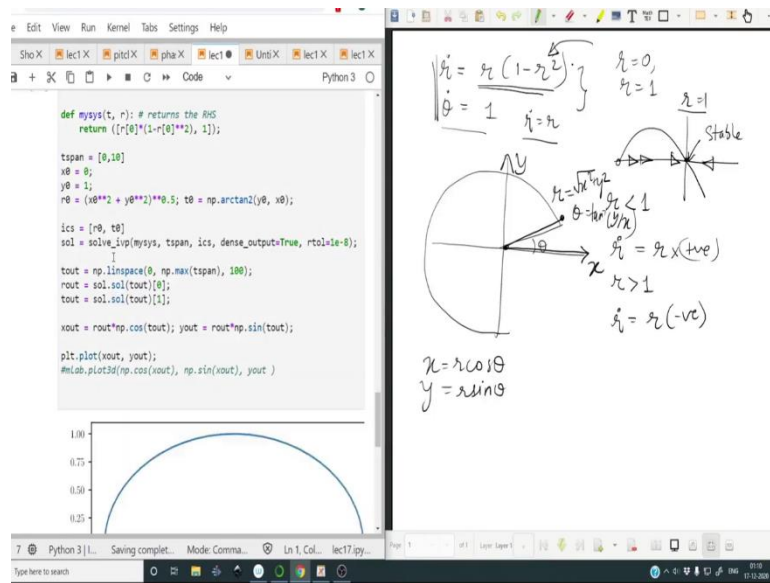
```

AttributeError                                Traceback (most recent c
all last)
<ipython-input-60-d4cd98e16d92> in <module>
      5 r0 = 0;
      6 y0 = 1;
----> 7 r0 = (x0**2 + y0**2)**0.5; t0 = np.atan2(y0, x0);
      8 ics = (r0, y0)
      9 sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol
=1e-8);
File ~/anaconda/lib/site-packages/numpy/_init_.py in _get_attr__(att
r)
    217         return Tester
    218     else:
--> 219         raise AttributeError("module {r} has no a
ttribute "
    220                               "{r}".format(_name_
_, attr))
    221
AttributeError: module 'numpy' has no attribute 'atan2'

```

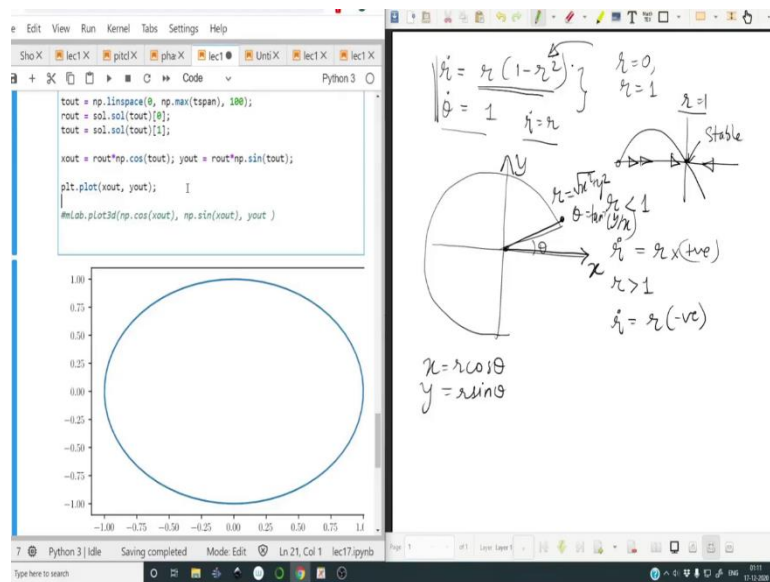
The handwritten slide is identical to the one in the first image, showing the differential equations, stability analysis, and polar coordinate conversion formulas.

(Refer Slide Time: 43:05)



So, let me run this and there appears to be an error. Sorry, so this is not atan2, but this is arctan2 ok. Once again, this has to be r0 and this has to be t0 ok. We have to pass the initial conditions appropriately ok.

(Refer Slide Time: 43:19)



(Refer Slide Time: 43:31)

The screenshot shows a Jupyter Notebook interface. On the left, the code cell contains the following Python code:

```

tout = np.linspace(0, np.max(tspan), 100);
rout = sol.sol(tout)[0];
tout = sol.sol(tout)[1];

xout = rout*np.cos(tout); yout = rout*np.sin(tout);

plt.plot(xout, yout);
ax = plt.gca(); ax.set_aspect(1);
#mlab.plot3d(np.cos(xout), np.sin(xout), yout)
    
```

Below the code, a 2D plot shows a closed circular orbit centered at the origin with a radius of 1.0. The axes range from -1.00 to 1.00.

On the right, a handwritten diagram illustrates polar coordinates. It shows a circle of radius r in the x - y plane. The angle θ is measured from the positive x -axis. The diagram includes the following equations and labels:

- $\dot{r} = r(1-r^2)$
- $\dot{\theta} = 1$
- $r = 0, r = 1$
- $r = 1$ is labeled as "Stable".
- $r = \sqrt{1-r^2}$
- $\theta = \arctan(y/x)$
- $r < 1$
- $r > 1$
- $\dot{r} = r(+ve)$
- $\dot{r} = r(-ve)$
- $x = r \cos \theta$
- $y = r \sin \theta$

So, when we do this, we do get a closed contour that is a fixed I mean closed orbit. So, let me just set the aspect ratio of this alright. So, this is that particular fixed orbit. Now, let us see what happens when I choose a contour which is slightly beyond $y = 1$ that is we choose this, then what happens? Let us see.

(Refer Slide Time: 43:50)

The screenshot shows a Jupyter Notebook interface. On the left, the code cell contains the following Python code:

```

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

tspan = [0,10]
x0 = 0;
y0 = 1.2;
r0 = (x0**2 + y0**2)**0.5; t0 = np.arctan2(y0, x0);

ics = [r0, t0]
sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol=1e-8);

tout = np.linspace(0, np.max(tspan), 100);
rout = sol.sol(tout)[0];
tout = sol.sol(tout)[1];

xout = rout*np.cos(tout); yout = rout*np.sin(tout);

plt.plot(xout, yout);
ax = plt.gca(); ax.set_aspect(1);
#mlab.plot3d(np.cos(xout), np.sin(xout), yout)
    
```

Below the code, a 2D plot shows an open orbit starting at $(0, 1.2)$ and moving upwards and to the right. The axes range from -1.0 to 1.0.

On the right, a handwritten diagram illustrates polar coordinates, identical to the one in the previous slide. It shows a circle of radius r in the x - y plane. The angle θ is measured from the positive x -axis. The diagram includes the following equations and labels:

- $\dot{r} = r(1-r^2)$
- $\dot{\theta} = 1$
- $r = 0, r = 1$
- $r = 1$ is labeled as "Stable".
- $r = \sqrt{1-r^2}$
- $\theta = \arctan(y/x)$
- $r < 1$
- $r > 1$
- $\dot{r} = r(+ve)$
- $\dot{r} = r(-ve)$
- $x = r \cos \theta$
- $y = r \sin \theta$

(Refer Slide Time: 43:51)

The screenshot shows a Jupyter Notebook interface. On the left, the code cell contains the following Python code:

```

xout = rout*np.cos(tout); yout = rout*np.sin(tout);
plt.plot(xout, yout);
ax = plt.gca(); ax.set_aspect(1);
#mlab.plot3d(np.cos(xout), np.sin(xout), yout )

```

The plot on the left shows a unit circle in the xy-plane, with axes ranging from -1.0 to 1.0. On the right, there is a hand-drawn diagram illustrating polar coordinates. The diagram shows a circle of radius r centered at the origin. The angle θ is measured from the positive x-axis. The coordinates are given by $x = r \cos \theta$ and $y = r \sin \theta$. The diagram also shows a vector \vec{r} from the origin to a point on the circle. The angle θ is labeled as $\theta = \arctan(y/x)$. The diagram includes the following text:

- $\dot{\theta} = 1$
- $\dot{r} = r$
- $r = 0, \dot{r} = 1$
- $r = 1$ (Stable)
- $r = \sqrt{x^2 + y^2}$
- $\theta = \arctan(y/x)$
- $\dot{r} = r$ (positive)
- $\dot{r} > 1$
- $\dot{r} = r$ (negative)

Below the diagram, the equations $x = r \cos \theta$ and $y = r \sin \theta$ are written.

So, let me increase the value to 1.2 ok. So, we see that the contour starts over here and it is sort of spiraling in into that radius $r = 1$. Let me now choose something which is less than 1.

(Refer Slide Time: 44:11)

The screenshot shows a Jupyter Notebook interface. On the left, the code cell contains the following Python code:

```

tspan = [0,10]
x0 = 0;
y0 = 0.8;
r0 = (x0**2 + y0**2)**0.5; t0 = np.arctan2(y0, x0);

ics = [r0, t0]
sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol=1e-8);

tout = np.linspace(0, np.max(tspan), 100);
rout = sol.sol(tout)[0];
tout = sol.sol(tout)[1];

xout = rout*np.cos(tout); yout = rout*np.sin(tout);
plt.plot(xout, yout);
ax = plt.gca(); ax.set_aspect(1);
#mlab.plot3d(np.cos(xout), np.sin(xout), yout )

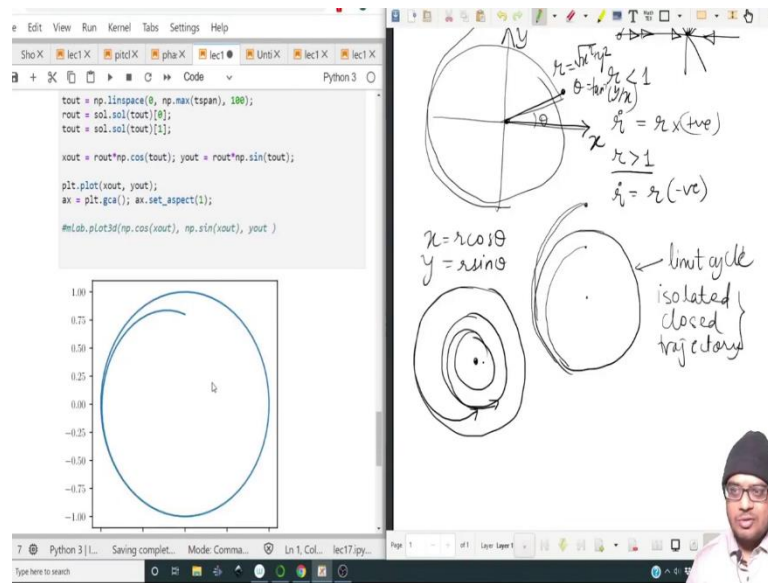
```

The plot on the left shows a curve starting at $(0, 0.8)$ and spiraling inward towards the unit circle. On the right, there is a hand-drawn diagram illustrating polar coordinates, identical to the one in the previous slide. The diagram includes the following text:

- $\dot{\theta} = 1$
- $\dot{r} = r$
- $r = 0, \dot{r} = 1$
- $r = 1$ (Stable)
- $r = \sqrt{x^2 + y^2}$
- $\theta = \arctan(y/x)$
- $\dot{r} = r$ (positive)
- $\dot{r} > 1$
- $\dot{r} = r$ (negative)

Below the diagram, the equations $x = r \cos \theta$ and $y = r \sin \theta$ are written.

(Refer Slide Time: 44:14)



So, let me make it 0.8. So, it starts over here and it spirals onto a circle again. So, what do we observe? We observe the following phenomenon. So, we have a closed contour like this. If we start at a point outside this, it sort of homes in onto this closed contour. If we start at an inner point, it also homes on to this.

So, this kind of an orbit is called as a limit cycle and a limit cycle is an isolated closed trajectory because it is called another isolated closed trajectory because it is to distinguish between a limit cycle and a circle or a center. So, a center will have a structure something like this ok.

You will have a series of closed orbits; but in the case of limit cycle, there will be exactly one closed orbit and the other orbits are sort of homing on to that particular orbit. It is not like you have a series of fixed orbits ok. You will have one cycle which is locked and all the other cycles will be sort of trying to reach that particular point. So, now let us draw a bunch of trajectories ok.

(Refer Slide Time: 45:49)

Python 3

```

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

tspan = [0,10]

for x0 in -1:
    x0 = 0;
    y0 = 0.8;
    r0 = (x0**2 + y0**2)**0.5; t0 = np.arctan2(y0, x0);

    ics = [r0, t0]
    sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol=1e-
        6, atol=1e-6);

    tout = np.linspace(0, np.max(tspan), 100);
    rout = sol.sol(tout)[0];
    tout = sol.sol(tout)[1];

    xout = rout*np.cos(tout); yout = rout*np.sin(tout);

    plt.plot(xout, yout);
    ax = plt.gca(); ax.set_aspect(1);

    #mlab.plot3d(np.cos(xout), np.sin(xout), yout)
  
```

Handwritten notes:

- $r = \sqrt{x^2 + y^2}$
- $\theta = \arctan(y/x)$
- $r < 1 \Rightarrow \dot{r} = r(x + ve)$
- $r > 1 \Rightarrow \dot{r} = r(-ve)$
- $x = r \cos \theta$
- $y = r \sin \theta$
- limit cycle
- isolated closed trajectory

So, how do we do a bunch of trajectories? We can simply wrap this entire thing inside a double for loop.

(Refer Slide Time: 46:02)

Python 3

```

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

tspan = [0,10]

for x0 in np.linspace(-1.5, 1.5, 15):
    for y0 in np.linspace(-1.5, 1.5, 15):
        x0 = 0;
        y0 = 0.8;
        r0 = (x0**2 + y0**2)**0.5; t0 = np.arctan2(y0, x0);

        ics = [r0, t0]
        sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol=
            1e-6, atol=1e-6);

        tout = np.linspace(0, np.max(tspan), 100);
        rout = sol.sol(tout)[0];
        tout = sol.sol(tout)[1];

        xout = rout*np.cos(tout); yout = rout*np.sin(tout);

        plt.plot(xout, yout);
        ax = plt.gca(); ax.set_aspect(1);

        #mlab.plot3d(np.cos(xout), np.sin(xout), yout)
  
```

Handwritten notes:

- $r = \sqrt{x^2 + y^2}$
- $\theta = \arctan(y/x)$
- $r < 1 \Rightarrow \dot{r} = r(x + ve)$
- $r > 1 \Rightarrow \dot{r} = r(-ve)$
- $x = r \cos \theta$
- $y = r \sin \theta$
- limit cycle
- isolated closed trajectory

So, for x naught in, what do we have? -1 $np.linspace(-1.5, 1.5)$. Let us take 15 points for y naught in $np.linspace(-1.5, 1.5)$. Let us take 15 points. Let me indent this further because we have a nested for loop ok.

(Refer Slide Time: 46:28)

```

e Edit View Run Kernel Tabs Settings Help
ShoX lectX ptdX phaX lect UntX lectX lectX
Python 3
rout = sol.sol[tout][0];
tout = sol.sol[tout][1];

xout = rout*np.cos(tout); yout = rout*np.sin(tout);

plt.plot(xout, yout);
ax = plt.gca(); ax.set_aspect(1);

#mlab_plot3d(np.cos(xout), np.sin(xout), yout)

```

$r = \sqrt{x^2 + y^2}$
 $\theta = \arctan\left(\frac{y}{x}\right)$
 $r < 1$
 $r_i = r_x(+ve)$
 $r > 1$
 $r_i = r_x(-ve)$

$x = r \cos \theta$
 $y = r \sin \theta$

limit cycle
isolated
closed
trajectory

(Refer Slide Time: 46:43)

```

e Edit View Run Kernel Tabs Settings Help
ShoX lectX ptdX phaX lect UntX lectX lectX
Python 3
for x0 in np.linspace(-1.5, 1.5, 15):
    for y0 in np.linspace(-1.5, 1.5, 15):
        #theta = theta;
        #r0 = r0;
        r0 = (x0**2 + y0**2)**0.5; t0 = np.arctan2(y0, x0);
        ics = [r0, t0]
        sol = solve_ivp(mysys, tspan, ics, dense_output=True, rtol=
        tout = np.linspace(0, np.max(tspan), 100);
        rout = sol.sol[tout][0];
        tout = sol.sol[tout][1];

        xout = rout*np.cos(tout); yout = rout*np.sin(tout);

        plt.plot(xout, yout);
        ax = plt.gca(); ax.set_aspect(1);

#mlab_plot3d(np.cos(xout), np.sin(xout), yout)

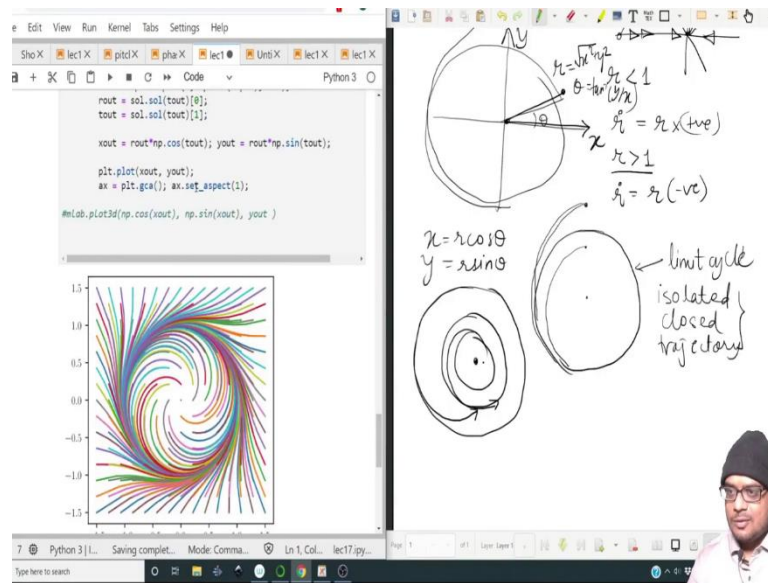
```

$r = \sqrt{x^2 + y^2}$
 $\theta = \arctan\left(\frac{y}{x}\right)$
 $r < 1$
 $r_i = r_x(+ve)$
 $r > 1$
 $r_i = r_x(-ve)$

$x = r \cos \theta$
 $y = r \sin \theta$

limit cycle
isolated
closed
trajectory

(Refer Slide Time: 46:44)

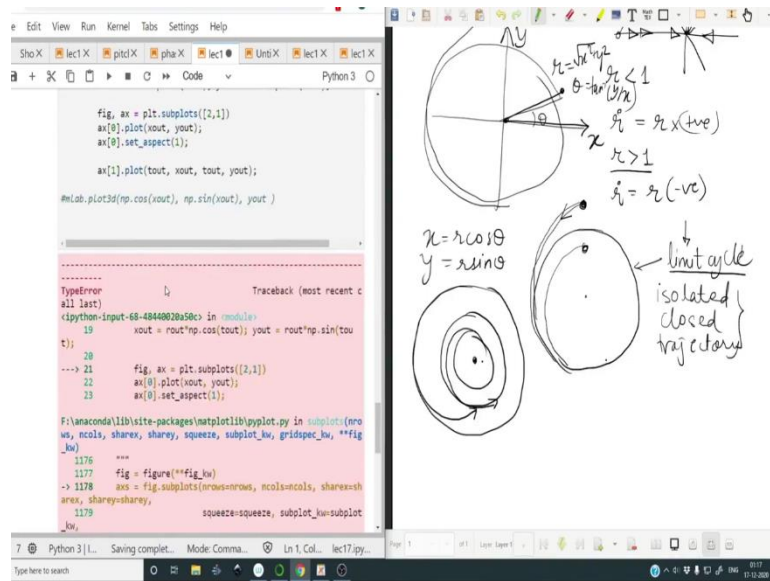


So, let me run this. Let us see all the different trajectories. Oops, we have to remove these two lines because even though, we are declaring them separately, I am resetting them to x_0 and y_0 . So, we have to remove this. Let me comment it for now, let me not ok. So, alright, great.

So, this phase portrait clearly highlights what the nature of the limit cycle is. The limit cycle is like a closed orbit which is attracting all the orbits towards itself and it is also called as an Isolated close trajectory and it has a certain property that it is sort of periodic in nature, but at the same time, it is self-regulating; meaning, if you push the system outside the limit cycle, it will again attract it back to the limit cycle. If you push it below the limit cycle, it will again rise and go towards the limit cycle ok.

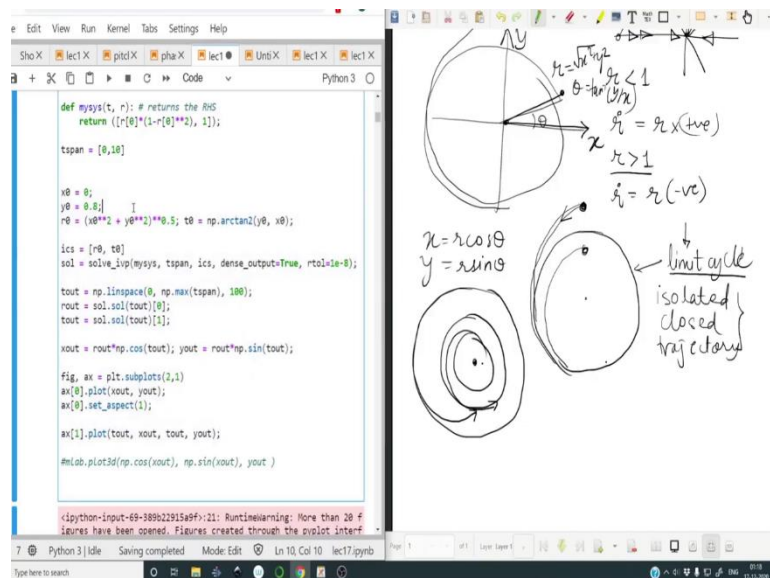
And that is the reason why it is called as a limit cycle. For long times, it tends to go towards that particular closed trajectory. So, think about it. I am not going to going to great details. But hopefully, this will give you enough food for thought. Before proceeding, I also want to give you an idea how the time series also looks like. So, in order to plot the time series, let us make two subplots. I think this is the first time, we are making subplots.

(Refer Slide Time: 48:11)



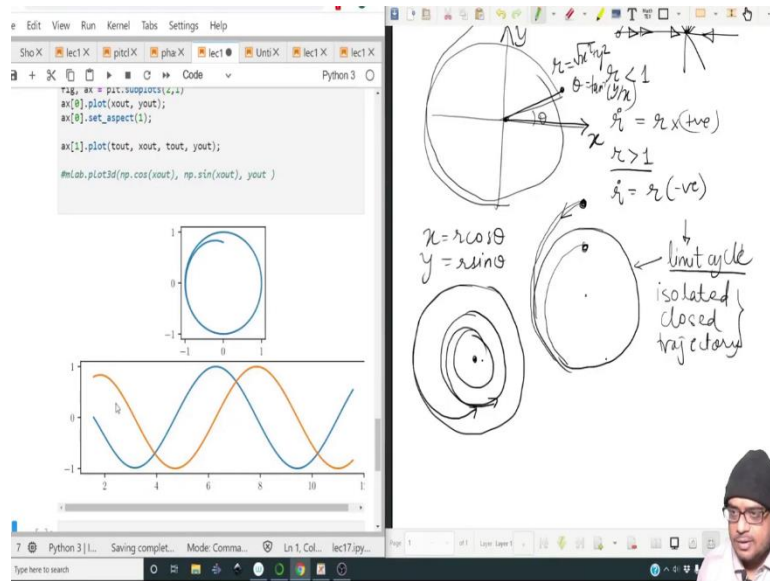
So, `fig, x = plt.subplots` and this will be 2 rows and 1 column because I want to plot this phase portrait as a single subplot and the time series as another subplot ok. So, after this, we will say `x0.plot` is this and we do not need this anymore. This will be `ax[0].set_aspect(1)`; then, we will have `ax[1].plot(tout, xout, tout, yout)` ok. So, let me run this and there is an error.

(Refer Slide Time: 49:07)



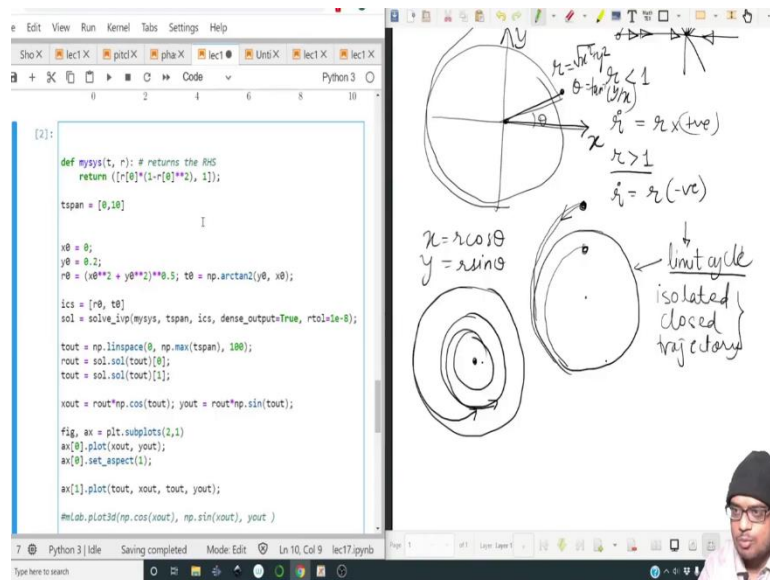
So, this has to be passed as 2.1 ok. In fact, because we are trying to plot the time series as well, I am going to get rid of this loop. I am going to introduce these two initial conditions back. Let me indent everything back.

(Refer Slide Time: 49:17)

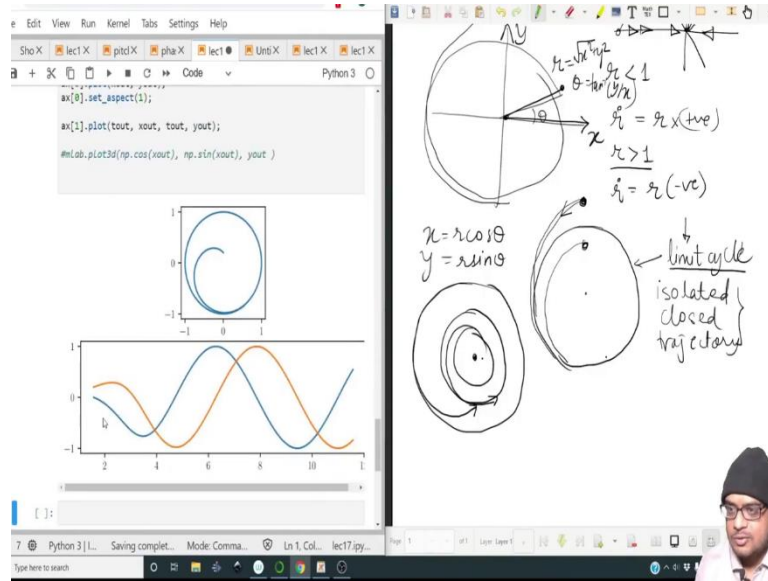


So, this is how the time series looks like ok. It is starting at a lower point than 1; but it settles down to 1, let me make it a bit more severe.

(Refer Slide Time: 49:28)

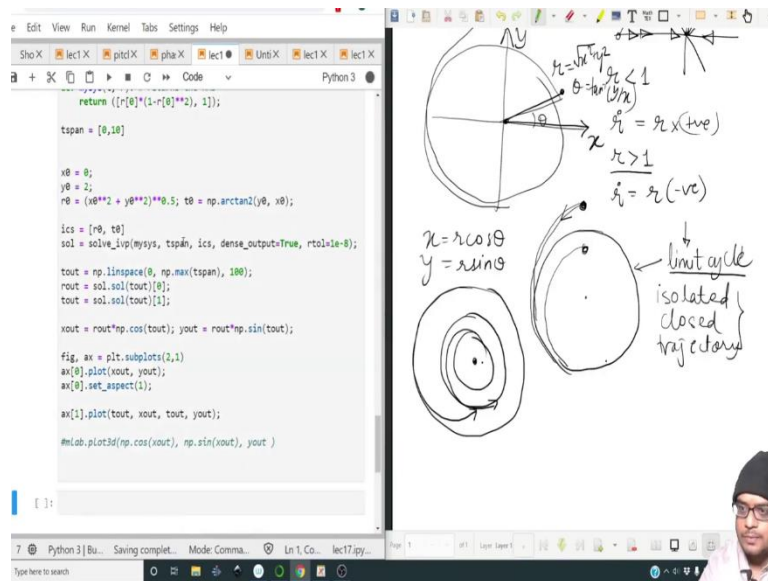


(Refer Slide Time: 49:31)

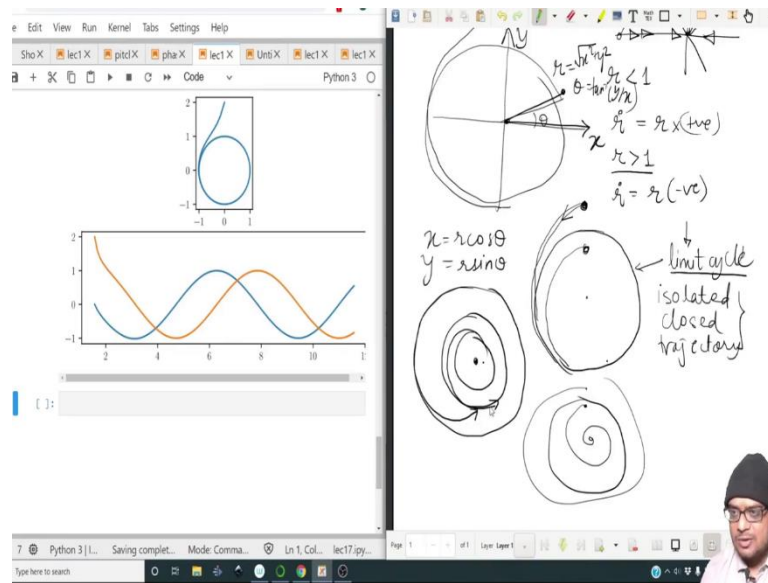


So, as to impress upon you how it approaches the limit cycle, let me make it at 0.2. It starts at 0.1 immediately regulates itself back to the limit cycle ok. So, it goes back to the case, where it is sort of shadowing the limit cycle which is simply going to be 2 waves; one is going to be a sin wave, one is going to be cos wave because a circle can be parameterized by cos theta and sin theta ok.

(Refer Slide Time: 49:54)



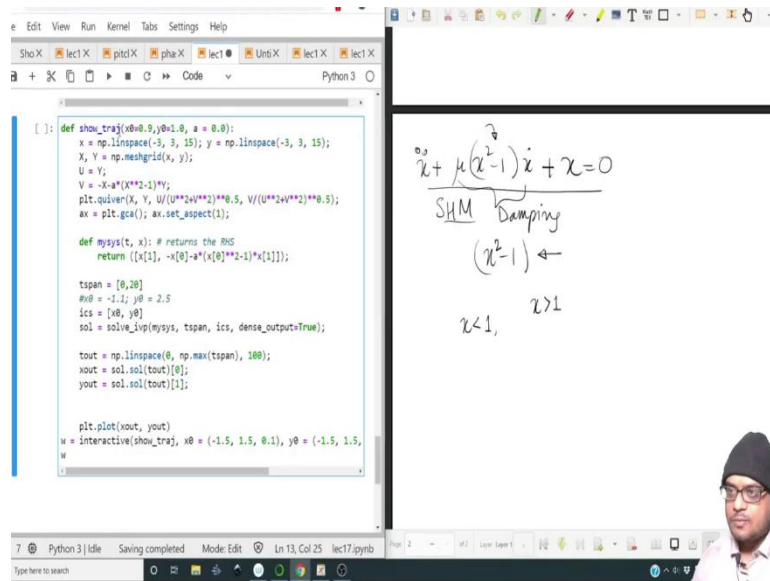
(Refer Slide Time: 49:57)



Similarly, when we have a condition which is say $y = 2$. So, it starts like this, then immediately it settles on to the limit cycle. So, it is called a stable limit cycle because it is a limit cycle which attracts. Similarly, there can be unstable limit cycles as well, these are cycles which repel. So, if you have a limit cycle like this ok, it will cause trajectories which are pushed away from the limit cycle to go away to a fixed point somewhere inside or if you have a trajectory outside, it will try to go away from this ok.

There can also be limit cycles which are I mean partially attracting that is the inner orbits, it will attract towards the limit cycle; whereas, the outer orbits, it will repel away from the limit cycle or even vice versa ok. This is how the time series looks like. You can have multiple plots like this. It is a very easy technique ok. So, now limit cycles need not always necessarily be something which resembles a circle ok. The most famous example of this is the Van-Der-Pol oscillator.

(Refer Slide Time: 51:22)



So, let me grab hold of um; in fact, let me find out a snippet which will be most useful for us. I think this snippet is going to be most useful for us. It is from one of the earlier lectures. So, yeah. So, it this particular snippet contains both a quiver plot that is a vector plot and it contain the trajectory as well and this will help us in clearly understanding what the nature of the flow is and the trajectory as that vector field is carrying it through.

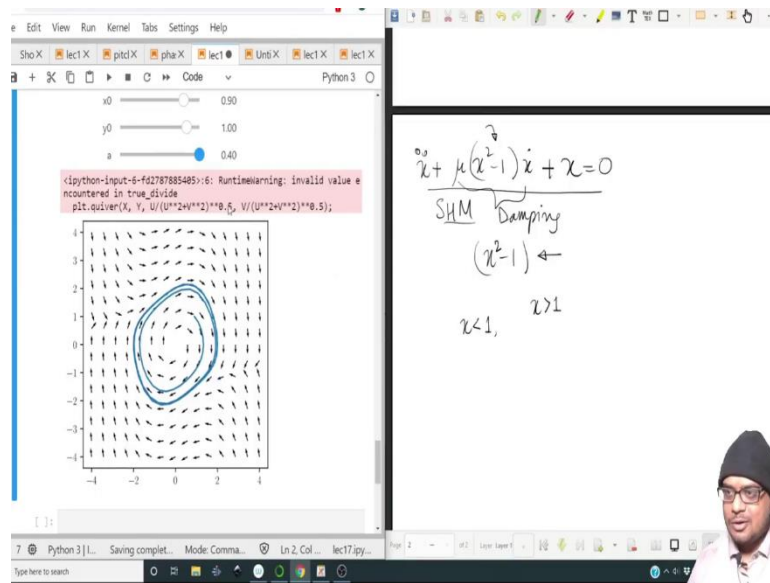
So, the Van-Der-Pol oscillator is described by an equation $\ddot{x} + \mu(x^2 - 1)\dot{x} + x = 0$. So, you can imagine this to be a simple harmonic oscillator; yes, but with a non-linear damping term. So, this particular term if this $(x^2 - 1)$ will to be not there, this would be a damping term; it would be a linear damping term.

But the $(x^2 - 1)$, this particular thing gives rise to that self regulatory behavior of this particular cycle; meaning, if $x < 1$, then this term is going to be positive, this term is going to be negative. That means, this particular damping term is not actually going to be a damping term, but it is going to push the cycle, it is going to give some energy to the cycle; whereas, if $x > 1$ ok, this term becomes positive and this will act like a damping term ok.

So, the presence of that particular nonlinearity is responsible for eventually what we will see as a limit cycle. But limit cycles are present only because of this kind of a feedback non-linearity. If you have a linear system, it will not occur. Linear systems cannot have such kinds of limit cycles.

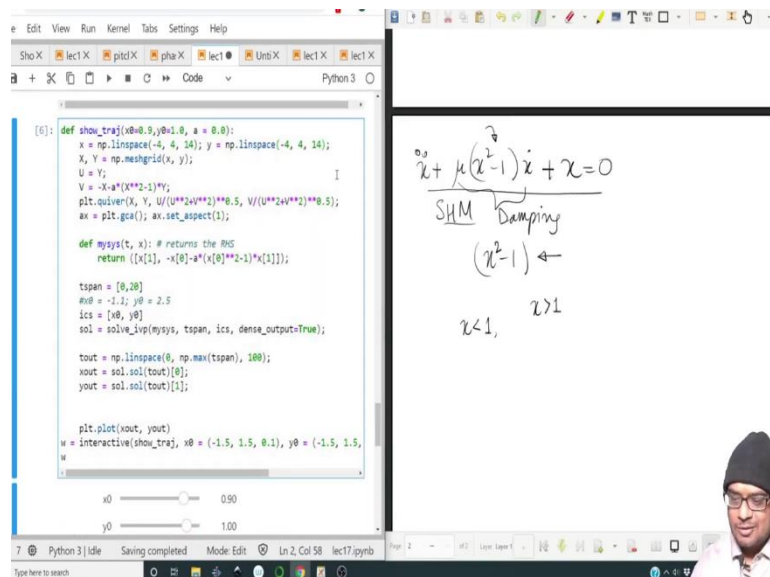
So, let us encode this particular flow. So, this should be \ddot{x} ok. So, the flow will be quite simple \dot{x} equal times x 1 ok. So, the time span is this and yeah, so the initial conditions ok.

(Refer Slide Time: 53:40)



So, let me run this and see what happens ok. So, for a equal to 0, obviously this behaves like a simple harmonic oscillator and it is a closed orbit, it is a non-isolated closed orbit. But when a increases ok, so we the flow was indeed correct, I just increased the axis of the problem. So, forget about this because at the origin, some 0 going on.

(Refer Slide Time: 54:21)



(Refer Slide Time: 54:22)

The screenshot shows a Jupyter Notebook interface on the left and a whiteboard on the right. The Jupyter Notebook displays the following code and plot:

```
yout = sol.sol(tout)[1];
```

```
plt.plot(yout, yout)
```

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

Below the code are three sliders for parameters: $x_0 = 0.90$, $y_0 = 1.00$, and $a = 0.00$. The plot shows a vector field with a central diamond-shaped orbit.

The whiteboard on the right contains the following handwritten text:

$$x'' + k(x^2 - 1)x + x = 0$$

SHM Damping

$$(x^2 - 1) \leftarrow$$

$x < 1$, $x > 1$

Let me change the number of points by once to avoid the origin ok. So, when a becomes something like this, you see that the initial point which is somewhere over here. It is going in a loop and it is settling onto this particular diamond shaped orbit ok.

(Refer Slide Time: 54:43)

The screenshot shows a Jupyter Notebook interface on the left and a whiteboard on the right. The Jupyter Notebook displays the following code and plot:

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

Below the code are three sliders for parameters: $x_0 = 0.90$, $y_0 = 1.00$, and $a = 0.40$. The plot shows a vector field with a diamond-shaped orbit.

The whiteboard on the right contains the following handwritten text:

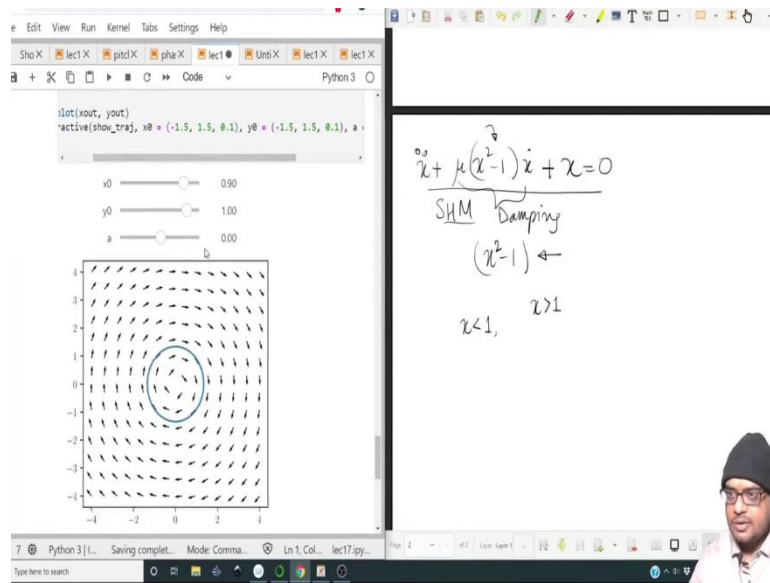
$$x'' + k(x^2 - 1)x + x = 0$$

SHM Damping

$$(x^2 - 1) \leftarrow$$

$x < 1$, $x > 1$

(Refer Slide Time: 54:52)



The screenshot shows a Jupyter Notebook interface. On the left, there is a code cell with the following code:

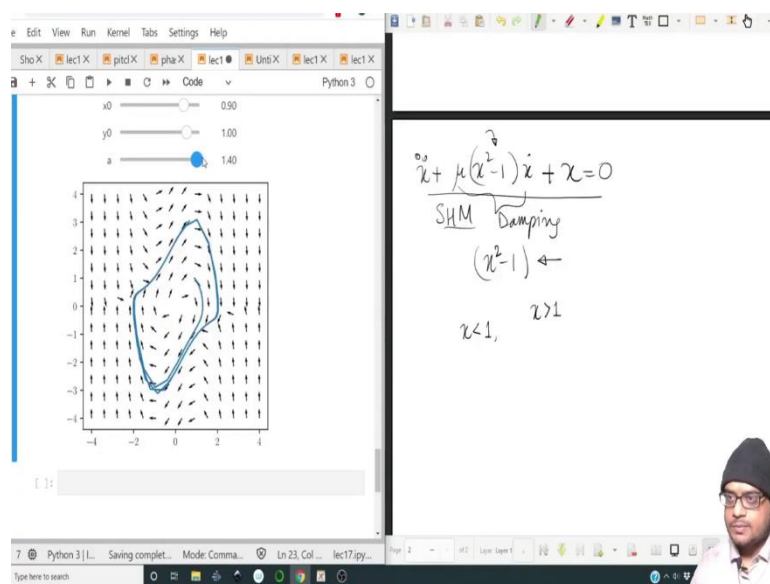
```
plot(xout, yout)
active(show_traj, x0 = (-1.5, 1.5, 0.1), y0 = (-1.5, 1.5, 0.1), a
```

Below the code are three sliders for parameters: x_0 (set to 0.90), y_0 (set to 1.00), and a (set to 0.00). The plot shows a vector field with a central closed orbit. On the right side, there are handwritten notes:

$$\ddot{x} + k(x^2-1)\dot{x} + \gamma x = 0$$

SHM Damping
 $(x^2-1) \leftarrow$
 $\gamma < 1, \quad \gamma > 1$

(Refer Slide Time: 54:56)



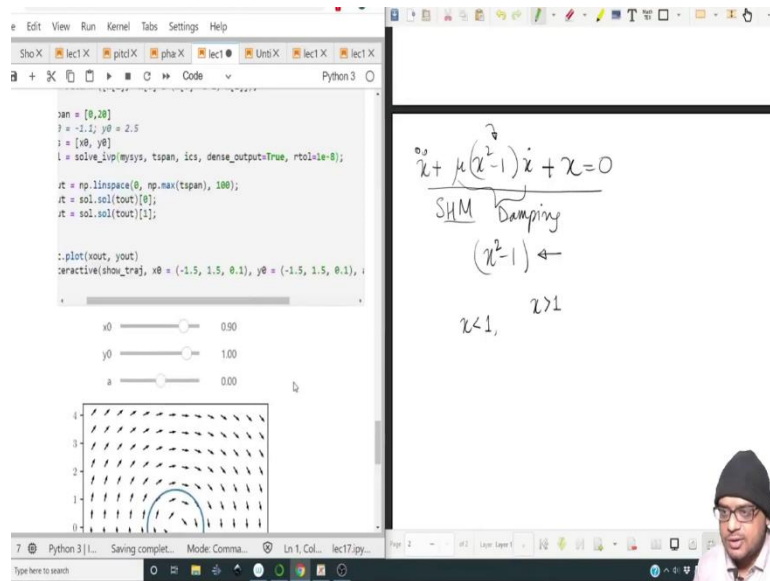
The screenshot shows the same Jupyter Notebook interface as the previous slide. The parameter a is now set to 1.40. The plot shows a vector field with a central closed orbit highlighted in blue. On the right side, the handwritten notes are identical to the previous slide:

$$\ddot{x} + k(x^2-1)\dot{x} + \gamma x = 0$$

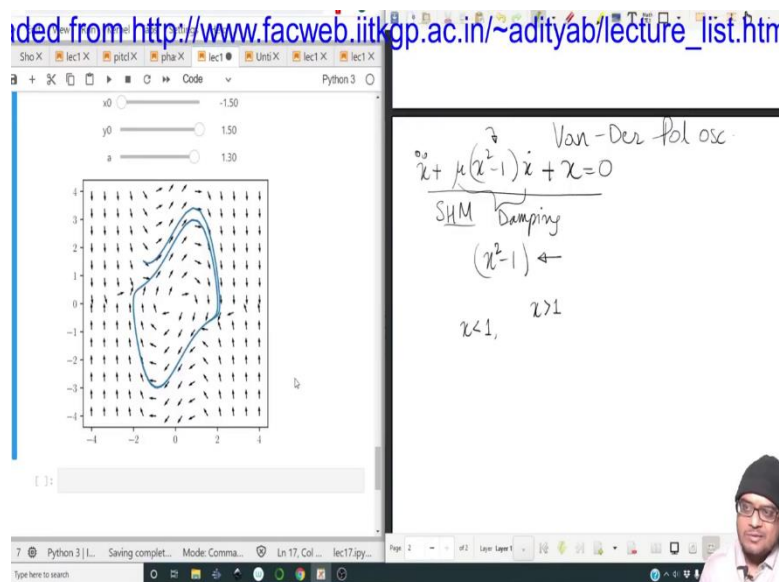
SHM Damping
 $(x^2-1) \leftarrow$
 $\gamma < 1, \quad \gamma > 1$

In fact, let me allow a to have even larger values ok. So, let me re-run this ok. So, this is how the flow looks like. So, even if I change the initial condition, it appears to always settle down onto that closed orbit ok. Let me increase the number of points over here ok.

(Refer Slide Time: 55:16)



(Refer Slide Time: 55:25)



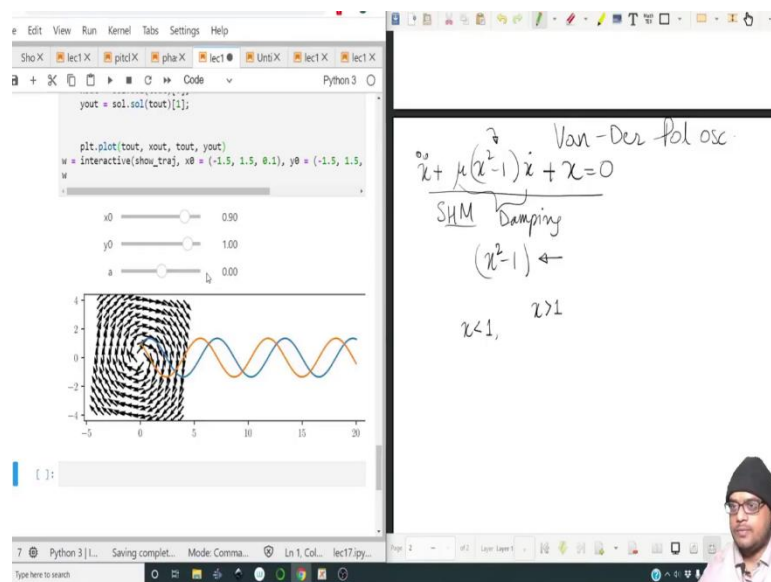
So, we have not yet specified the relative tolerance in this. So, it should be $1e-6$ or $1e-8$; just for good measure ok. So, this is how the fixed orbit or the limit cycle looks like. It starts at a point, it eventually holds on to this rhombus looking shape and even if I change the initial condition, it will always do that that fixed cycle is not going to change and it looks jagged because I have taken very few sampling points in the plot.

Let me take 200, it should immediately be smoother ok; there you go. So, now even if I change x , so that it is lying outside the limit cycle, it is homing on to the limit cycle ok. So, even this Van-Der-Pol oscillator because of this non-linear damping effect, it is always going to hold on to this limit cycle.

So, to learn more on the theory of limit cycles I suggest you look at some of the links below and some of those lectures are by Professor Strogatz and yeah, he explains them in a very easy fashion.

But with the help of these tools that you have learned, you can easily implement many of the things which is showing in class on your own and once you start doing these things on your own, you can really plot a lot of things, you can really visually understand because this topic of non-linear dynamics is all about having a visual interpretation, having a deep visual interpretation of what is going on ok.

(Refer Slide Time: 57:09)



So, now let us also plot the trajectories ok. I want to show you how the trajectories also look like. So, in fact, let me just go ahead and do tout, xout tout, yout. Yeah, you have to suppress the quiver plot. We have to suppress this ok.

(Refer Slide Time: 57:18)

```

def show_traj(x0=0.9,y0=1.0, a = 0.0):
    x = np.linspace(-4, 4, 14); y = np.linspace(-4, 4, 14);
    X, Y = np.meshgrid(x, y);
    U = Y;
    V = -X*(X**2-1)*Y;
    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]*a*(x[0]**2-1)*x[1]]);

    tspan = [0,20]
    #x0 = -1.1; y0 = 2.5
    ics = [x0, 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];

    plt.plot(tout, xout, tout, yout)
    w = interactive(show_traj, x0 = (-1.5, 1.5, 0.1), y0 = (-1.5, 1.5, 0.1))

```

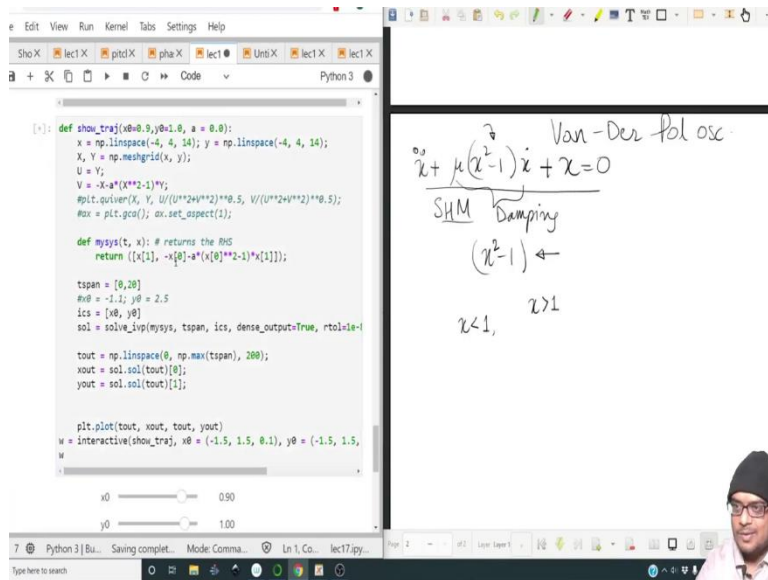
Van-Dez Pol osc.

$$\ddot{x} + \mu(x^2-1)\dot{x} + x = 0$$

SHM Damping

$$(x^2-1) \leftarrow$$

$x < 1$ $x > 1$

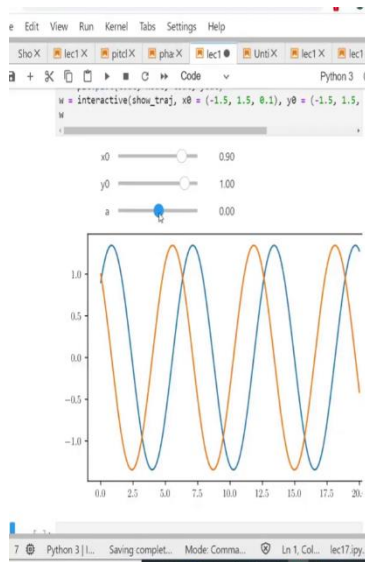


(Refer Slide Time: 57:24)

```

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

```



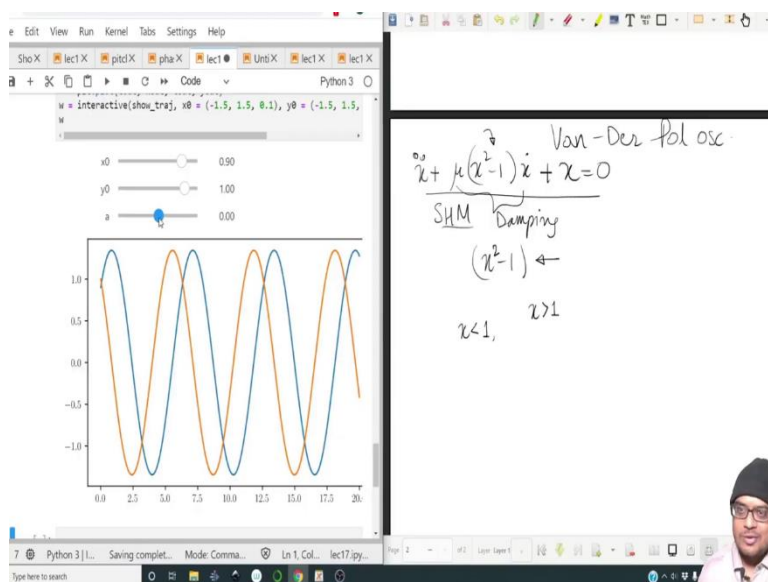
Van-Dez Pol osc.

$$\ddot{x} + \mu(x^2-1)\dot{x} + x = 0$$

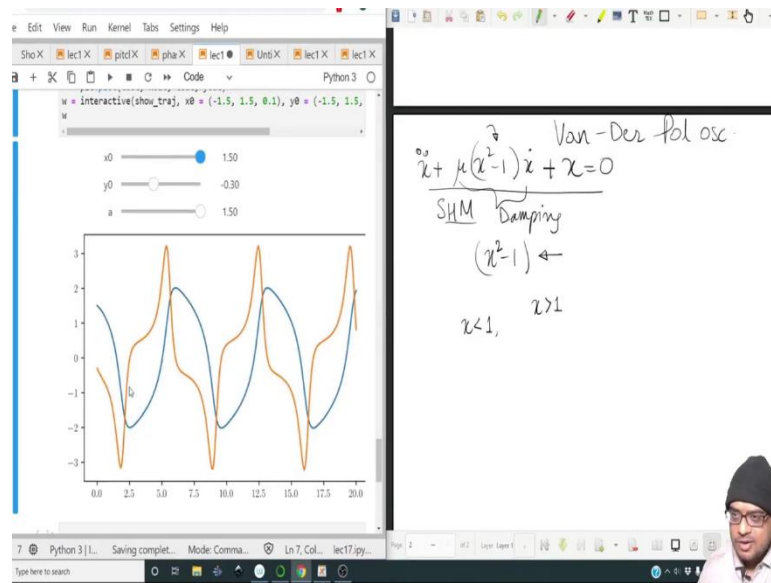
SHM Damping

$$(x^2-1) \leftarrow$$

$x < 1$ $x > 1$



(Refer Slide Time: 57:29)



So, for a equal to 0, it is a simple harmonic oscillator and a increases, look at how the waveforms change. So, it is quantified by a slow or it has two-timed skills what I am trying to get it ok. It has one fast time scale, a slower time scale and a fast time scale and a fast decay, then a slow decay, then a fast decay.

Similarly, fast rise, slow decay, fast fall, slow increase, fast increase, slow decrease, fast decrease and so on. So, it is a problem which is sort of quantified by the presence of multiple time scales and all this, when you start doing it analytically falls within the embed of multiple time scale perturbation theory.

It is something which I teach in a different course; but in view of a shortage of time, we are going to skip that in these particular lectures ok. So, this is how you can visualize the time series as well ok. So, now, lastly, we move on to a very famous reaction and it was shown that this particular set of reactions can be reduced to two non-linear equations. So, the name of the name of the reaction is glycolysis.

(Refer Slide Time: 58:49)

The image shows a Python Jupyter notebook interface on the left and handwritten mathematical notes on the right. The notebook code defines a function `show_traj` to plot the trajectory of a system of differential equations. The equations are:

$$\dot{x} = -x + ay + x^2y - \text{ATP}$$

$$\dot{y} = b - ay - x^2y - \text{ADP}$$

The code uses `scipy.integrate.solve_ivp` to solve these equations and `matplotlib` to plot the trajectory. The handwritten notes on the right show the equation $x^2 = 1$ with arrows pointing to $x < 1$ and $x > 1$. Below this, the word "Glycolysis" is written, followed by the differential equations for x and y . The fixed point equations are derived as $-x + b = 0$ and $x^2 = b$, leading to the solutions $x^* = b$ and $y^* = \frac{b}{a+b^2}$.

So, glycolysis is the reaction in which sugar molecule is converted into energy and it is one of the most fundamental energy producing cycle. It is an incredibly complex cycle consisting of many sub cycles and all that. But all said and done, it does boil down to something which can be sort of mathematically modeled as two coupled non-linear equations.

So, let me write down the form of the equations. So, \dot{x} is going to be $-x + ay + x^2y$ and \dot{y} is going to be $b - ay - x^2y$, where x represents the concentration or the evolution of ATP and y represents the evolution of ADP ok. So, the evolution of these two molecules is what sort of governs the entire chemical set of chemical reactions behind glycolysis and let us try to analyze these two governing equations.

So, first of all what is the fixed point? So, the fixed point can be found out as follows; \dot{x} will be 0 and \dot{y} will be 0. So, 0 is this; 0 is this. So, let us add these two equations. So, you obtain $-x + b = 0$ for the fixed point and consequently, you obtain $x^* = b$; whereas, for the y , so if this is true, then using this we can obtain $y^* = \frac{b}{(a+b)^2}$. That is just rearranging this I am getting.

So, y^* is this, x^* is this. So, you can go ahead and find out the nature of this particular fixed point by finding out the Jacobian. I am not going to do it. I am just going to show you how the flow looks like and how we can easily view by changing the values of a and b , what the nature of the fixed point is going to be like ok.

So, with the fixed point out of the way, let us grab hold of in fact, this is fine for our purpose. So, let me paste it over here and let me now change the various values. So, obviously, the flows have to change; the vector flow and the phase space is given by this plus $aY + X^2Y$ ok.

(Refer Slide Time: 61:46)

The screenshot shows a Jupyter Notebook interface. On the left, a plot displays a function with three distinct minima. Below the plot is a Python code block that defines a system of differential equations and solves them using the Runge-Kutta method. On the right, handwritten notes in blue ink discuss the 'Glycolyst's' model, showing the equations $\dot{x} = -x + ay + x^2y$ (ATP) and $\dot{y} = b - ay - x^2y$ (ADP). It also shows the steady-state conditions $-x + b = 0$ leading to $x^* = b$ and $y^* = \frac{b}{a+b^2}$.

(Refer Slide Time: 61:47)

This screenshot shows the same Jupyter Notebook interface but with a time series plot instead of the function plot. The plot shows two variables, x and y, over time. The x-axis ranges from 0.0 to 20.0, and the y-axis ranges from 0.0 to 2.0. Above the plot are four sliders for parameters: x0 (0.90), y0 (1.00), a (0.00), and b (0.10). The handwritten notes on the right are identical to the previous slide, discussing the 'Glycolyst's' model equations and steady-state values.

So, this is how the time series looks like. But we are not, I mean we have we should plot the time series as well; but why not we plot the phase space as well.

(Refer Slide Time: 62:01)

The image shows a Python IDE window with the following code:

```

def show_traj(x0=0.9, y0=1.0, a = 0.0, b = 0.1):
    x = np.linspace(-4, 4, 14); y = np.linspace(-4, 4, 14);
    X, Y = np.meshgrid(x, y);
    U = -X + a*Y + X**2*Y;
    V = b - a*Y - X**2*Y;
    #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] + a*x[1] + x[0]**2*x[1], b - a*x[1] - 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-6)

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

    fig, ax = plt.subplots(2, 1)
    ax[0].quiver(X, Y, U/(U**2+V**2)**0.5, V/(U**2+V**2)**0.5);
    ax[0].set_aspect(1)
    ax[0].plot(xout, yout)

    ax[1].plot(tout, xout, tout, yout)

w = interactive(show_traj, x0 = (0, 1.5, 0.1), y0 = (0, 1.5, 0.1),

```

Handwritten notes on the right side of the IDE:

Sampling
 $(x^2-1) \leftarrow$
 $x < 1, \quad x > 1$

Glycolysts \leftarrow

$$\begin{aligned} 0 &= \dot{x} = -x + ay + x^2y - \text{ATP} \\ 0 &= \dot{y} = b - ay - x^2y - \text{ADP} \end{aligned}$$

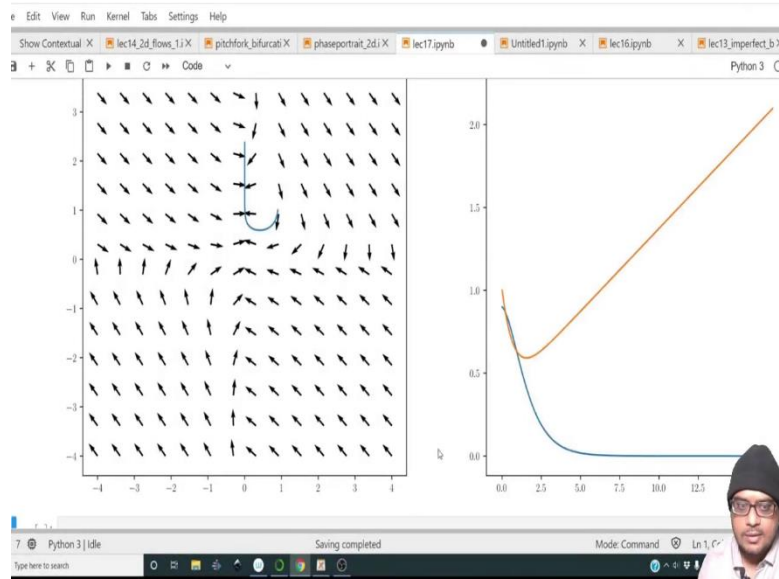
$$\dot{x} - x + b = 0 \quad , \quad x^3 = b$$

$$y^3 = \frac{b}{a+b^2} \quad , \quad y^3 = \frac{b}{a+b^2}$$

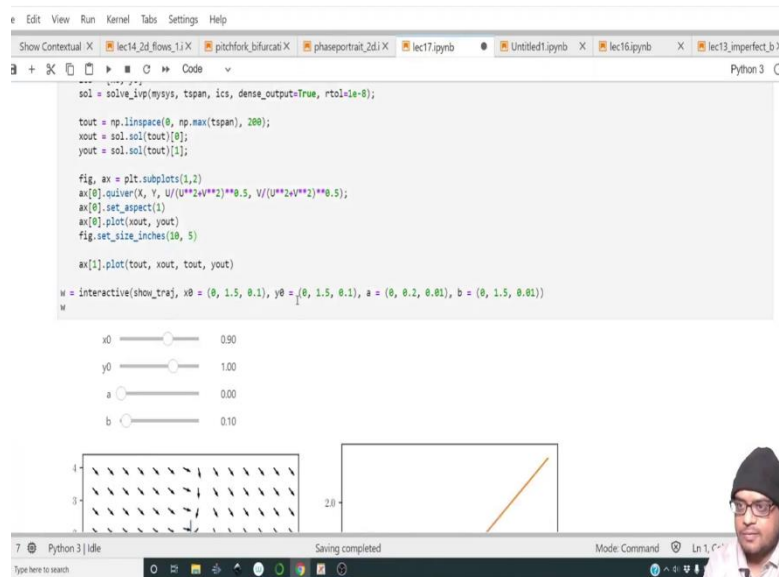
So, let me do that. So, fig, ax = plt.subplots(2, 1) and ax. ax[0] will let us make the flow. So, let us copy this. So, the first axis will contain the quiver plot and we will set it to the aspect_ratio to 1. Then, on the next plot, we will plot the time series; on the same plot, we will also plot the xout and yout that trajectory of the particle which starts at the specified initial condition ok.

So, just a quick overview what is going on, we have made a widget which calls this function with the initial values x naught and y naught; x naught and y naught will be passed on to the function and we will first plot the quiver plot. After we plot the quiver plot, we will solve the initial value problem with the specified initial condition and the parameters are passed on by the widget. So, once we solve this, we will plot on top of it, how the trajectory in that phase portrait also looks like.

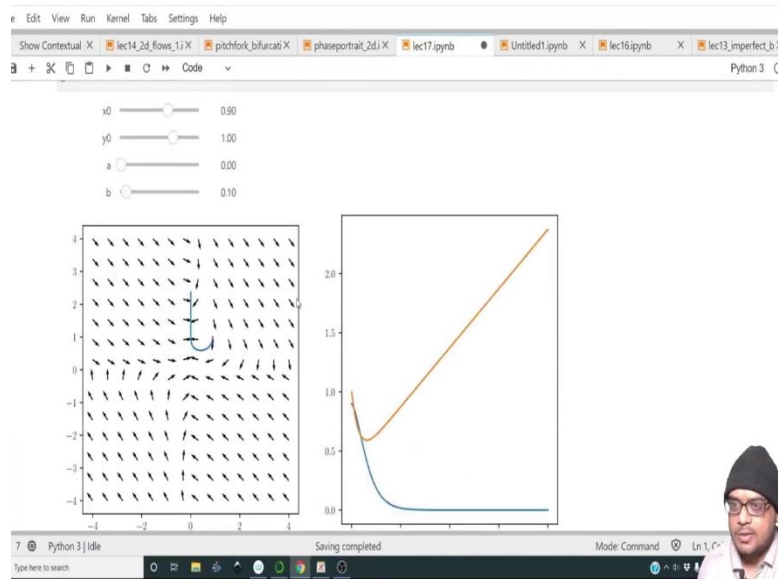
(Refer Slide Time: 63:46)



(Refer Slide Time: 63:55)

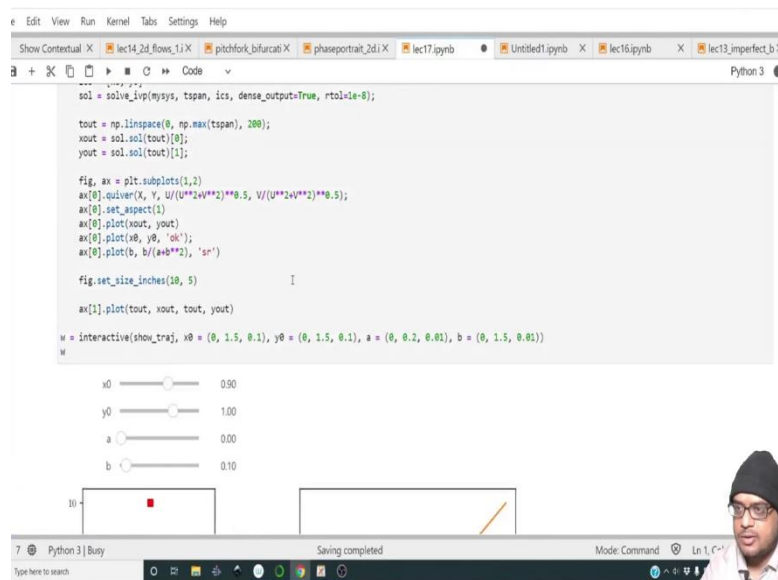


(Refer Slide Time: 63:59)

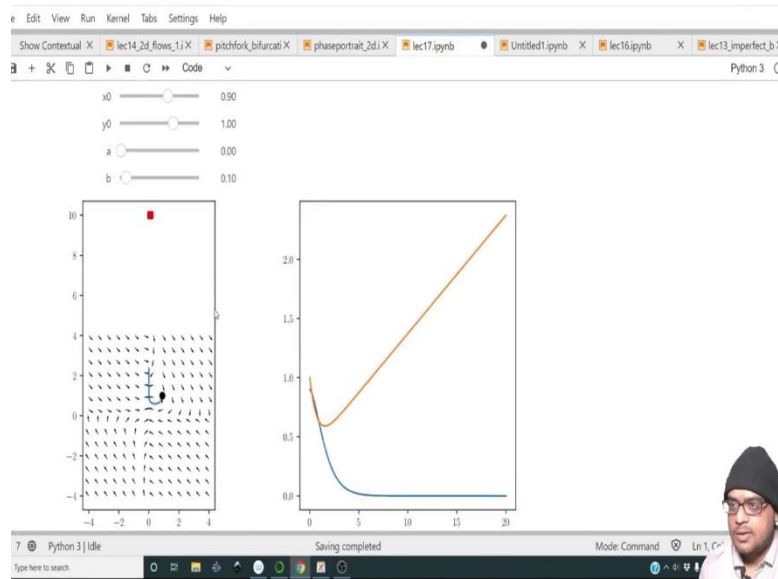


So, what we see over here is the quiver plot. In fact, this is perhaps a bit too big. Let me make it 10 by 5 ok. So, this is the quiver plot of the entire process. Let me go ahead and plot the initial point on top of this.

(Refer Slide Time: 64:11)

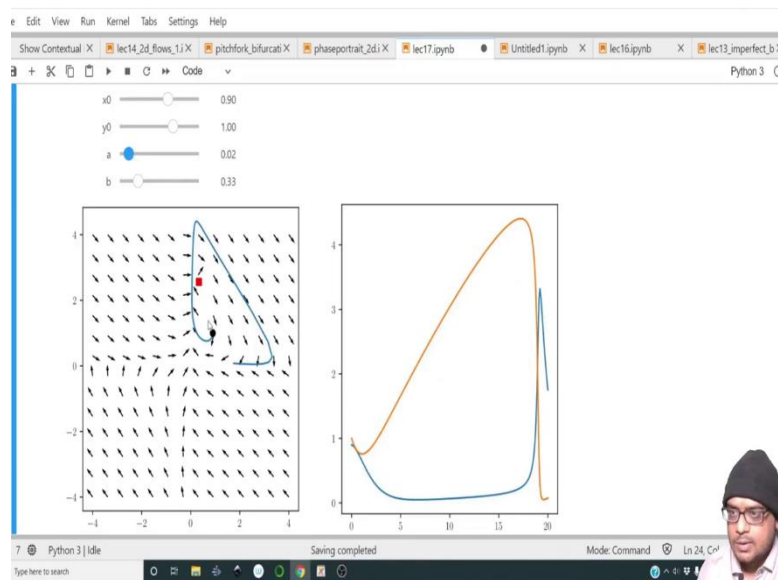


(Refer Slide Time: 64:54)



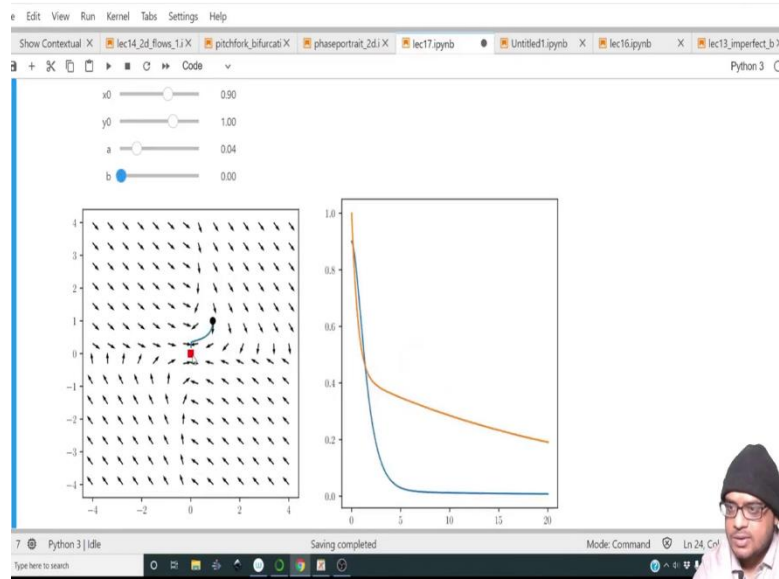
So, it will be `ax[0].plot(x0, y0)` and let me plot it as a field marker. In fact, let us also plot the fixed point in this particular figure. So, `ax[0].plot`. So, the fixed point is $(b, b/(a+b)^2)$. So, $(b, b/(a+b)^2)$ let me mark it by a square a red square ok.

(Refer Slide Time: 65:08)



Let me run this ok. So, see what happens. When I choose, suppose I choose a value of a to be 0.4, now let us see what happens when the value of b is changed ok.

(Refer Slide Time: 65:33)



So, now when b is 0, the red square that is the fixed point, it is attracting ok. The trajectory heads towards. So, the time series also show that it is decaying towards certain value.

(Refer Slide Time: 65:46)

```

tspan = [0, 20]
#y0 = -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];

fig, ax = plt.subplots(1,2)
ax[0].quiver(X, Y, U/(U**2+V**2)**0.5, V/(U**2+V**2)**0.5);
ax[0].set_aspect(1)
ax[0].plot(xout, yout)
ax[0].plot(x0, y0, 'ok');
ax[0].plot(b, b/(a+b**2), 'sr')

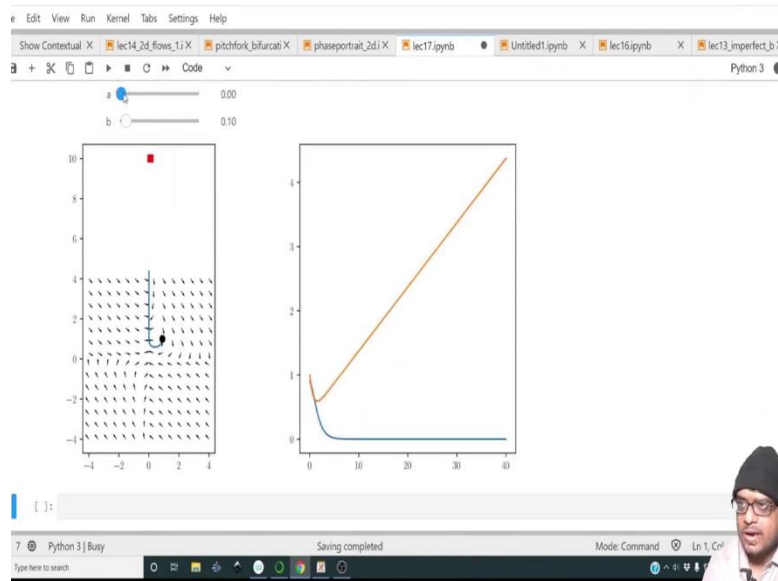
fig.set_size_inches(10, 5)

ax[1].plot(tout, xout, tout, yout)

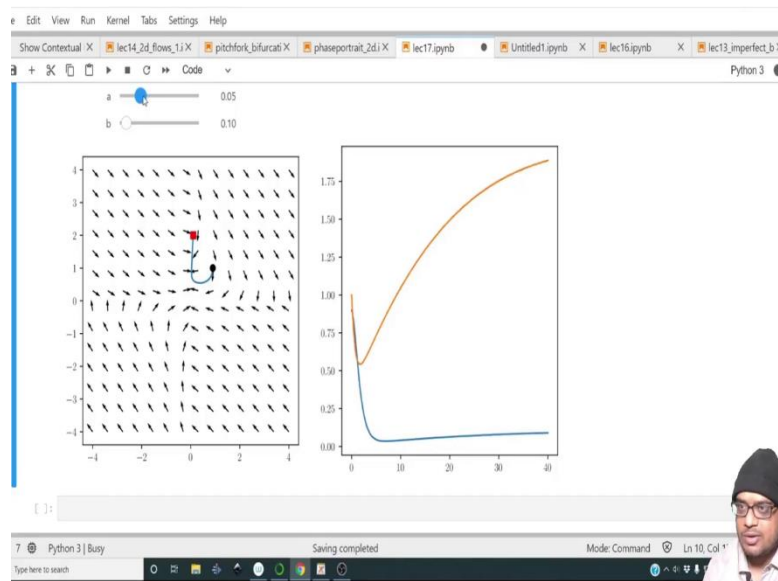
w = interactive(show_traj, x0 = (0, 1.5, 0.1), y0 = (0, 1.5, 0.1), a = (0, 0.2, 0.01), b = (0, 1.5, 0.01))
w

```

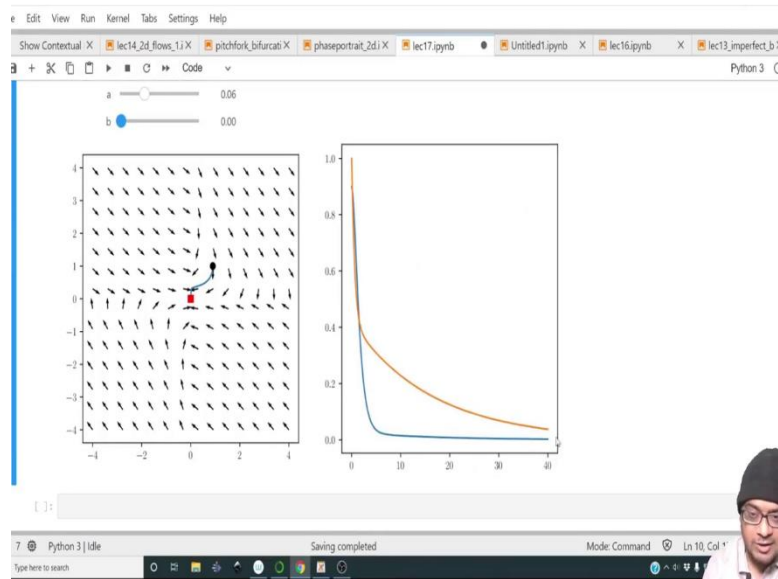
(Refer Slide Time: 65:50)



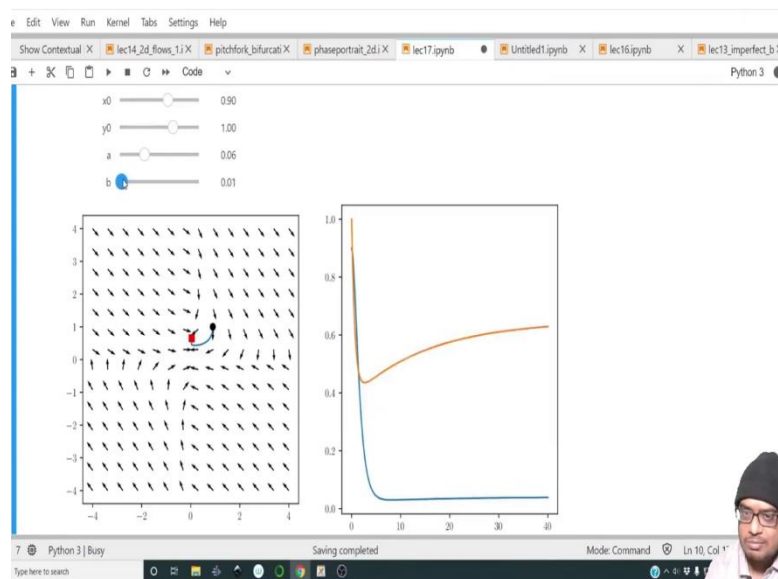
(Refer Slide Time: 65:53)



(Refer Slide Time: 65:59)

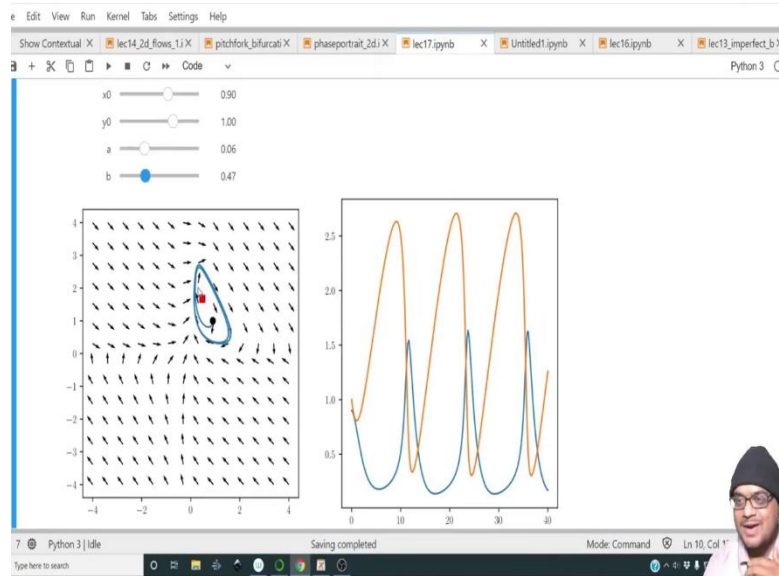


(Refer Slide Time: 66:04)



Let me increase the time span by 2 times so that it will be clear. So, let me make a as 0.6 and let me make b to be 0. So, they are approaching the fixed point. So, when b is 0, everything is a fixed point. Let me increase b slowly. When b is 0.1, also you have a fixed point. But when b is 0.02, it is also a fixed point ok, it is going towards that value.

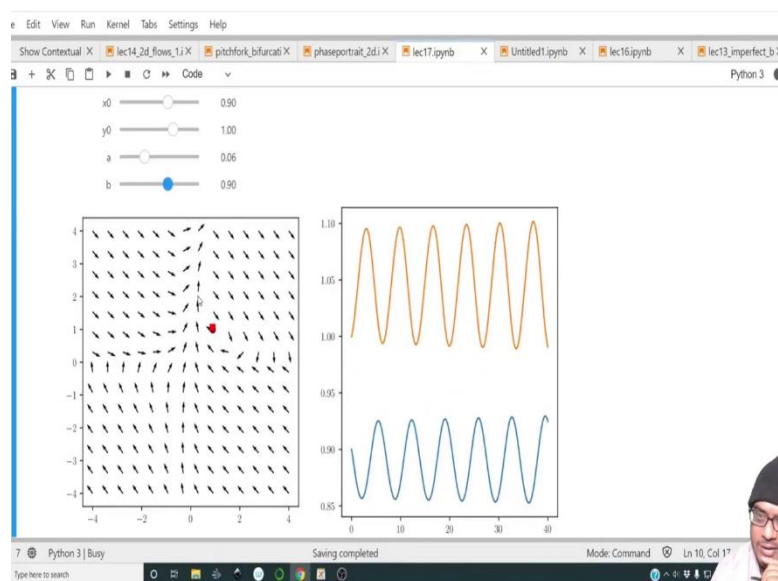
(Refer Slide Time: 66:19)



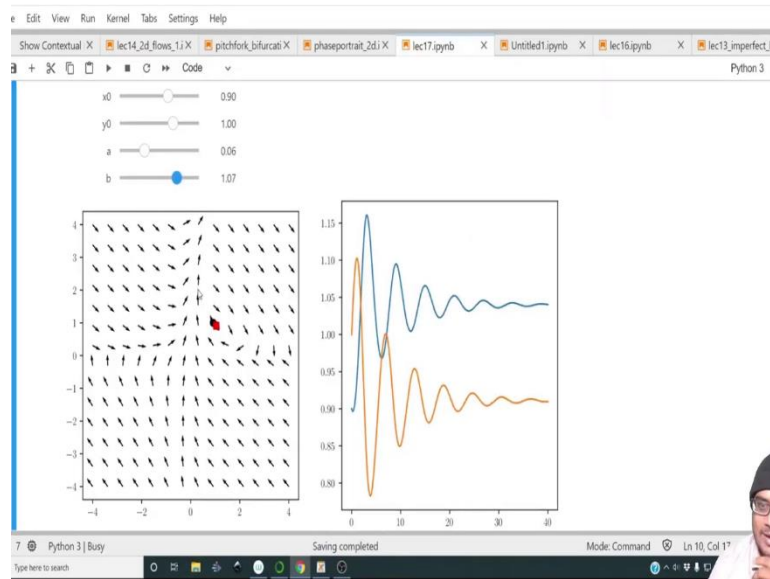
Now, when b increases further, you see that something weird begins to happen ok. So, now it is also it is still a fixed point, but now suddenly you have these self-sustaining oscillations ok. So, it has gone into a fixed, it has gone to a stable oscillation and it is a limit cycle. So, the orbit around the fixed point is not attracting the point all the way to the center; but it has it is now behaving like a limit cycle ok. This keeps on going on the limit cycle, in fact grows in size.

So, when a limit cycle grows in size, it means that the concentrations of ATP and ADP are oscillating across a wide range of values. But regardless of that, they do have a definitive oscillatory behavior as seen from both the phase portrait and from the time series.

(Refer Slide Time: 67:26)

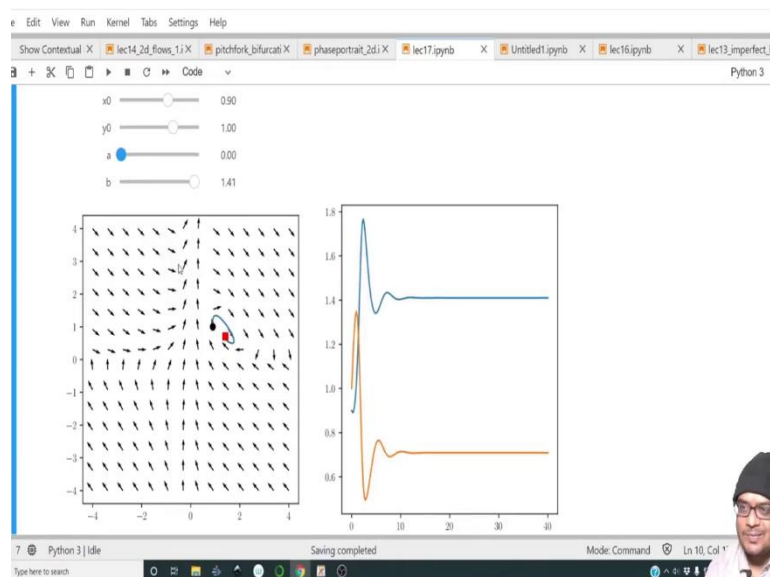


(Refer Slide Time: 67:36)

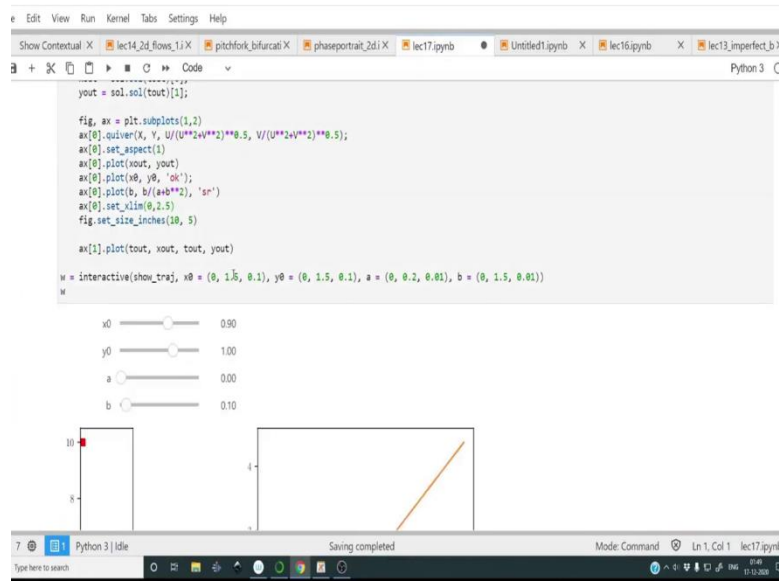


When it further increases, now look what happens. It is going, it is going, it is still fixed, it is still a fixed point, it is still a fixed point, it is obviously, a fixed point, it is still a limit cycle. But now, after a certain value, it no longer is able to sustain that and it becomes an attracting orbit again. So, now for higher values of b , the red square is no longer the center of a limit cycle, but it becomes a stable attracting point ok.

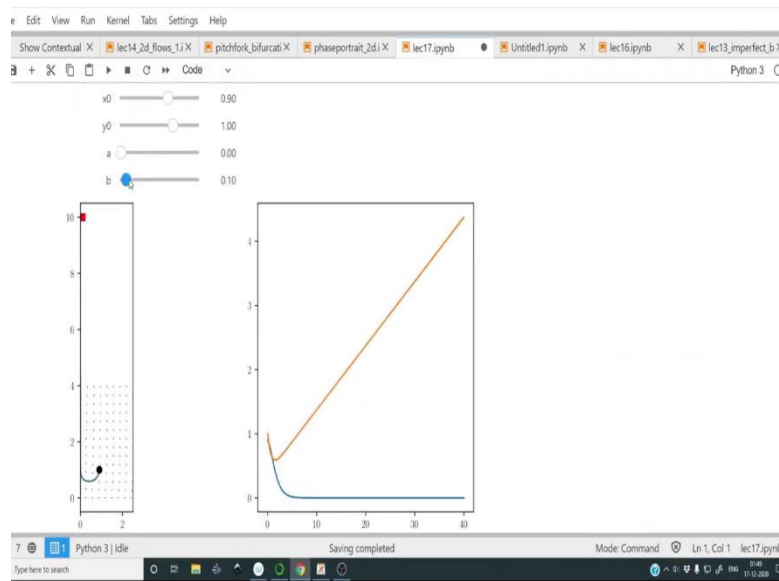
(Refer Slide Time: 68:00)



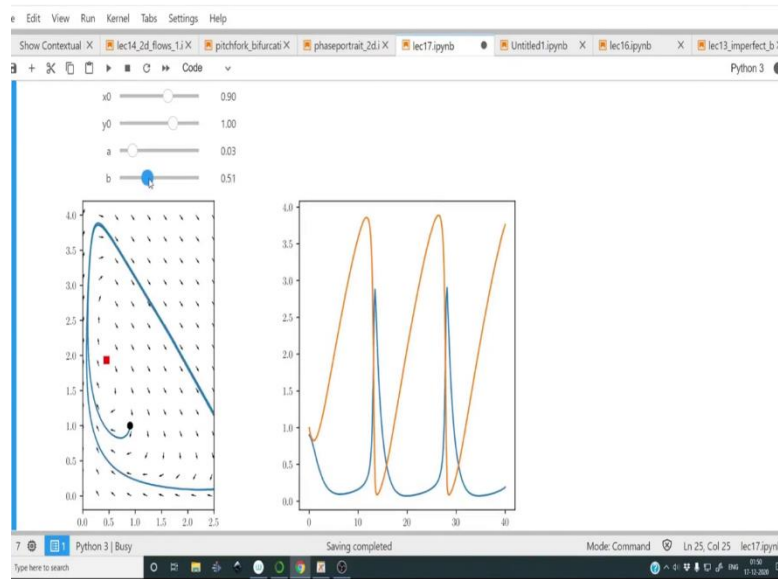
(Refer Slide Time: 68:10)



(Refer Slide Time: 68:16)

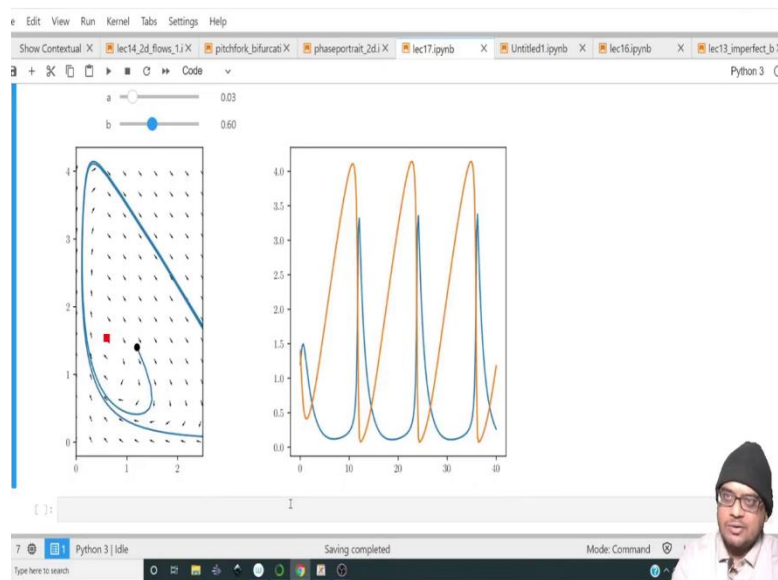


(Refer Slide Time: 68:20)



And a similar behavior can be done for different values of a as well. So, we can set the x axis limits as shown with the help of this. We can focus on one particular area; we can now change the value of a and b and visualize how the limit cycle actually looks like. The corresponding time series is also shown on the right sub figure.

(Refer Slide Time: 68:29)



It is quite interesting to see the transition from being a stable point to a stable limit cycle and you can do this for various initial conditions as well and they wrap around several times before coming into the limit cycle. So, with this, we wrap up this particular set of

lectures in which we have looked at various properties of non-linear phase diagrams including saddle nodes, attractors, repellers, centers, degenerate stars and lastly, limit cycles.

The next lecture, we are going to look at how we can analyze bifurcations in two-dimensions and it will draw heavily on the bifurcations in one-dimensional non-linear problems. So, with this I take leave; I will see you again next time. Bye.