```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint # import integrator routine


def simulate(f,
             init_state,
             t0=0,
             tf=1,
             N=500,
             size =(6, 4),
             show_plot=False):

    t = np.linspace(t0, tf, N)  # Create time span

    x_sol = []
    for x_init in init_state:
        # integrate system "sys_ode" from initial state $x0$
        x_sol.append(odeint(f, x_init, t))

    plt.figure(figsize=size)
    if show_plot:
        for sol in x_sol:
            plt.plot(t, sol, linewidth=2.0)
        plt.grid(color='black', linestyle='--', linewidth=1.0, alpha=0.7)
        plt.grid(True)
        plt.xlim([t0, tf])
        plt.ylabel(r'State $x$')
        plt.xlabel(r'Time $t$ (s)')
        plt.tight_layout()
    # show()
        plt.show()


    return x_sol


def phase_portrait(f,
                   x_range=[1, 1],
                   cmap='gray',
                   contour=False,
                   #    show_plot=False,
                   size=(7, 5),
                   density=0.95,
                   draw_grid=False,
                   ):

    x1_max, x2_max = x_range
    x1_span = np.arange(-1.1*x1_max, 1.1*x1_max, 0.1)
    x2_span = np.arange(-1.1*x2_max, 1.1*x2_max, 0.1)
```

```
    x1_grid, x2_grid = np.meshgrid(x1_span, x2_span)
    dx1, dx2 = f([x1_grid, x2_grid], 0)

    dist = (x1_grid**2 + x2_grid**2)**0.5
    lw = 0.8*(2*dist + dist.max()) / dist.max()

    # figure(figsize=size)
    plt.title('Phase Portrait')

    if contour:
        plt.contourf(x1_span, x2_span, dist, cmap=cmap, alpha=0.15)

    plt.streamplot(x1_span, x2_span, dx1, dx2, arrowsize=1.2,   density=density, color=c
               cmap=cmap, linewidth=lw, arrowstyle='->')  # ,color=L, cmap='autumn', lir

    plt.xlabel(r'State  $x_1$')
    plt.ylabel(r'State  $x_2$')

    plt.xlim([-x1_max, x1_max])
    plt.ylim([-x2_max, x2_max])
    if draw_grid:
        plt.grid(color='black', linestyle='--', linewidth=1.0, alpha=0.3)
        plt.grid(True)
    plt.tight_layout()
    # show()

    return None
```

Modern Control Paradigms:

Lecture 3: Stability of Nonlinear systems, Lyapunov analysis, Region of Attraction

Lyapunov's Direct Method

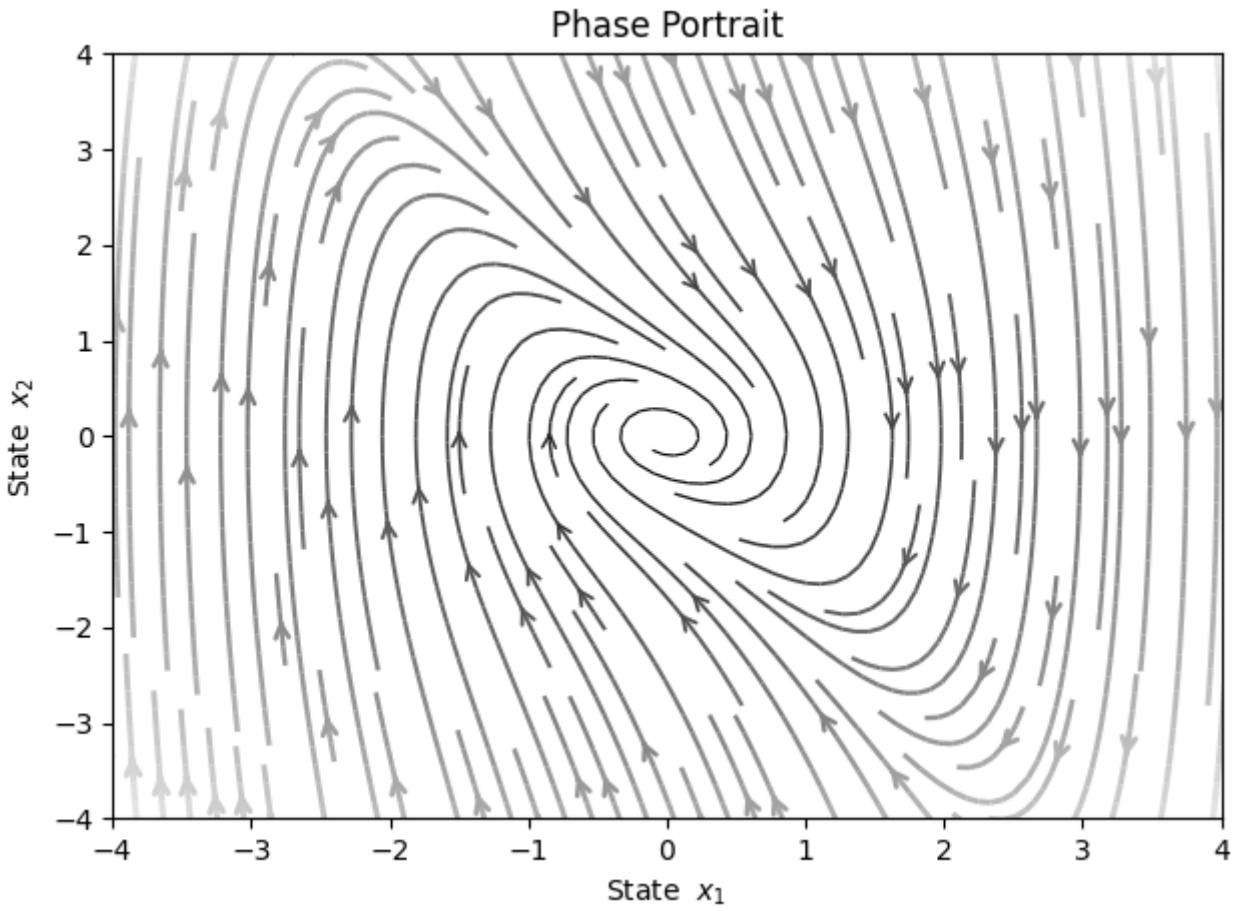
The basic philosophy of Lyapunov's direct method is the mathematical extension of a fundamental physical observation: if the total energy of a mechanical (or electrical) system is continuously dissipated, then the system, whether linear or nonlinear, must eventually settle down to an equilibrium point. Thus, we may conclude the stability of a system by examining the variation of a single scalar function $V(\mathbf{x})$.

Specifically, let us consider the nonlinear mass-damper-spring system:

$$m\ddot{y} + b|\dot{y}|\dot{y} + k_0 y + k_1 y^3 = 0$$

Assume that the mass is pulled away from the
natural length of the spring by a large distance, and then released.
Will the resulting motion be stable? A physical intuition said that it should be.

```
m, b, k0, k1 = 1, 1, 1, 1
def f(x,t):
    y = x[0]
    dy = x[1]
    ddy = -(b*abs(dy)*dy + k0*y + k1*y**3)/m
    return dy, ddy
phase_portrait(f, x_range=[4, 4], density=1.5)
```



However if one will find the Jacobian:

```
x = sp.symbols(r'y \dot{y}', real = True)
f_sym = sp.Matrix([f(x, 0)]).T

equlibriums = sp.solve(f_sym, x)
print(f'Equilibria are:\n{equlibriums}\n')
jacobian = f_sym.jacobian(x)
print(f'Jacobian is:')
jacobian
```

```
Equilibria are:
[(0, 0)]

Jacobian is:
```

$$\begin{bmatrix} 0 & 1 \\ -3y^2 - 1 & -\dot{y}\,\mathrm{sign}\,(\dot{y}) - |\dot{y}| \end{bmatrix}$$

```
from numpy import array, real
from numpy.linalg import eig

jacobian_num = sp.lambdify([x], jacobian)
for equlibrium in equlibriums:
    x_e = array(equlibrium, dtype='double')
    A = array(jacobian_num(x_e), dtype='double')
    print(f'The real part of poles for equlibrium {x_e} are: \n {real(eig(A)[0])} ')
```

```
The real part of poles for equlibrium [0. 0.] are:
 [ 0. -0.]
```

Evaluation of the Jacobian around trivial equilibrium yields marginal stability of linearized system, thus we can't say anything on nonlinear system.

However, lets take a bit different approach. Consider the following function (energy):

$$V(y, \dot{y}) = \frac{1}{2} m \dot{y}^2 + \frac{1}{2} k_0 y^2 + \frac{1}{4} k_1 y^4$$

The rate of this function (power) during the system's motion is obtained easily by differentiating:

$$\dot{V}(y, \dot{y}) = m \dot{y} \ddot{y} + (k_0 y + k_1 y^3) \dot{y} = -b |\dot{y}|^3$$

this implies the energy of the system, starting from some initial value,

is continuously dissipated by the damper until the mass settles down, i.e. $y = 0$.

Physically, it is easy to see that the mass must finally settle down at the natural length of the spring, because it is subjected to a non-zero spring force at any position other than the natural length.

Comparing the definitions of stability and mechanical energy, one can easily see some relations between the mechanical energy and the stability concepts described earlier:

- zero energy corresponds to the equilibrium point ($y = 0, \dot{y} = 0$)
- asymptotic stability implies the convergence of mechanical energy to zero
- instability is related to the growth of mechanical energy

These relations indicate that the **value of a scalar quantity**, the mechanical energy, **indirectly reflects the magnitude of the state vector**.

The direct method of Lyapunov is based on a generalization of the concepts in the above mass-spring-damper system to more complex systems. Faced with a set of nonlinear differential equations, the basic procedure of **Lyapunov's direct method** is to generate a **scalar "energy-like" function** for the dynamical system, and examine the time variation of that scalar function. In this way, conclusions may be drawn on the **stability of the set of differential equations without using the difficult stability definitions or requiring explicit knowledge of solutions**

Such **candidate Lyapunov function** should satisfy the energy-like properties, namely be:

- Strictly positive unless both state variables $\mathbf{x}$ are zero.
- Monotonically decreasing when the variables $\mathbf{x}$ vary along system trajectories

In Lyapunov's direct method, the first property is formalized by the notion of **positive definite functions** (PD)

Positive Definite Functions

A scalar continuous function $V(\mathbf{x})$ is said to be ***locally positive definite*** (LPD) in ball $\mathcal{B}_R = \{\mathbf{x} : \|\mathbf{x}\| \leq R\}$ if:

- $V(0) = 0$
- $V(\mathbf{x}) > 0,\ \forall \mathbf{x} \neq \mathbf{0}$

If above property holds $\forall \mathbf{x} \in \mathbb{R}^n$ then $V(\mathbf{x})$ is said to be ***globally positive definite*** (GPD)

Note that the above definition implies that the function V has a unique minimum at the origin.

For instance:

- The $V(\mathbf{x}) = \frac{1}{2}x_2^2 + 1 - \cos x_1$ is locally positive definite
- While $V(\mathbf{x}) = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2$ is globally positive definite

A few related concepts can be defined similarly, in a local or global sense, i.e., a function $V(\mathbf{x})$ is negative definite if — $V(\mathbf{x})$ is positive definite; $V(\mathbf{x})$ is positive semi-definite if $V(\mathbf{0}) = 0$ and $V(\mathbf{x}) \geq 0$ for $\mathbf{x} \neq 0$ $V(\mathbf{x})$ is negative semi-definite if - $V(\mathbf{x})$ is positive semi-definite.

The prefix "semi" is used to reflect the possibility of V being equal to zero eventhough $\mathbf{x}$ is not.

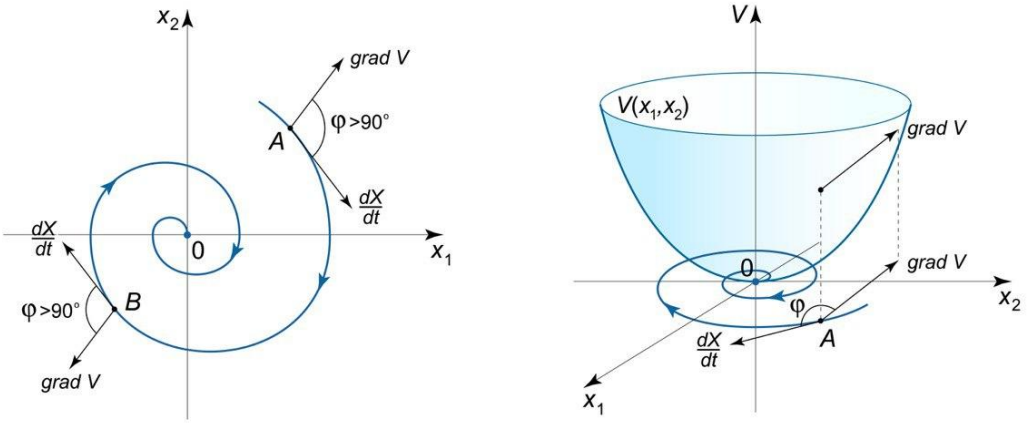
Global Stability via Direct Method

Assume that there exists a scalar function $V(\mathbf{x})$ with continuous first order derivatives such that:

- $V(\mathbf{x})$ is positive definite
- $\dot{V}(\mathbf{x})$ is negative definite
- $V(\mathbf{x}) \to \infty$ as $\|\mathbf{x}\| \to \infty$

then the equilibrium at the origin is **globally asymptotically stable**

Geometrical Interpretation



Example:

Consider the following system:

$$\begin{cases} \dot{x}_1 = -x_1 + x_2 \\ \dot{x}_2 = -x_1 - x_2^3 \end{cases}$$

with following Lyapunov candidate:

$$V(\mathbf{x}) = x_1^2 + x_2^2$$

```
x = sp.symbols('x_1, x_2')
V_symb = x[0]**2 + x[1]**2
print(f'Lyapunov candidate:')
V_symb
```

```
Lyapunov candidate:
```

$x_1^2 + x_2^2$

```
grad_V = sp.Matrix([V_symb]).jacobian(x)
print(f'Gradient of Lyapunov candidate:')
grad_V
```

```
Gradient of Lyapunov candidate:
```

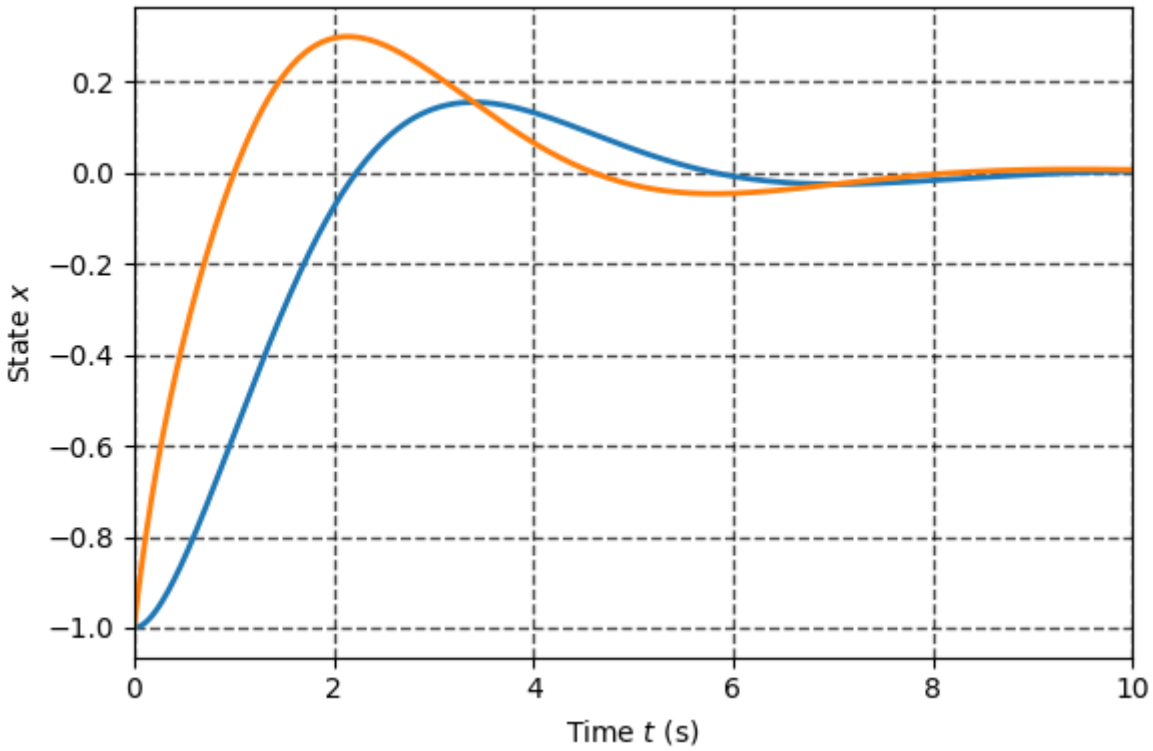
$\begin{bmatrix} 2x_1 & 2x_2 \end{bmatrix}$

```
f_symb = sp.Matrix([-x[0] + x[1],
                    -x[0] - x[1]**3])

dV = sp.simplify(grad_V*f_symb)
print(f'Time derivative of Lyapunov candidate:')
dV[0]
```

```
Time derivative of Lyapunov candidate:
```

$-2x_1^2 - 2x_2^4$

```
# Create a numerical function from symbolic one
f_num = sp.lambdify([x], f_symb)

def f(x, t):
    dx = f_num(x)[:,0]
    return dx

x_sol = simulate(f, [[-1,-1]], tf = 10, show_plot=True)
```



Example: Lyapunov Functions for LTI systems

The direct Lyapunov method can be in fact applied to linear systems as well:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$$

Consider the quadratic Lyapunov candidate:

$$V = \mathbf{x}^T\mathbf{P}\mathbf{x}$$

with time derivative:

$$\dot{V} = \mathbf{x}^T(\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A})\mathbf{x} = -\mathbf{x}^T\mathbf{Q}\mathbf{x}$$

where $\mathbf{Q}$ is P.D.

One can conclude that the LTI system is stable if there is P.D solution of matrix equation:

$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} = -\mathbf{Q}$$

Example: Common Lyapunov Functions

This property alone may not be very usefull, still we now how to do linear analysis just with eigen values, however it may be used in more general circumstances, for instance, suppose that the matrix $\mathbf{A}$ is unknown, but its uncertain elements can be bounded as the convex combination of a number of known matrices:

$$\mathbf{A} = \sum_i \beta_i \mathbf{A}_i, \quad \sum_i \beta_i = 1, \quad \beta_i > 0, \forall i$$

Then one can use the direct method with the very same Lyapunov function as above to arrive to stability criteria:

$$\mathbf{A}_i^T\mathbf{P} + \mathbf{P}\mathbf{A}_i \preccurlyeq 0$$

Thus if one can find the common $\mathbf{P}$ that satisfy constraints above the system may be proven to be stable for whatever $\mathbf{A}$ in the given set

Example: Stability of The Linear Descriptor Systems

The Lyapunov theory can be applied to so called descriptor systems in form:

$$\mathbf{E}\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$$

Indeed by defining the Lyapunov function as:

$$V = \mathbf{x}^T\mathbf{E}^T\mathbf{P}\mathbf{E}\mathbf{x}$$

One can arrive to following condition stability condition:

$$\mathbf{A}^T\mathbf{P}\mathbf{E} + \mathbf{E}^T\mathbf{P}\mathbf{A} \preccurlyeq 0$$

To match this condition one should solve for: $\mathbf{A}^T\mathbf{P}\mathbf{E} + \mathbf{E}^T\mathbf{P}\mathbf{A} = -\mathbf{Q}$, which is known as generalized Lyapunov equation.

Example: Convergence Rates for Nonlinear Systems

One can use tha Lyapunov like arguments to deduce not only stability but the **convergence rates** for trajectories, let us consider the real function $V(t)$ that satisfy inequality:

$$\dot{V} + \alpha V \leq 0$$

where α is real number, then:

$$V(t) \leq V(0)e^{-\alpha t}$$

Thus one can estimate **speed of convergence** for system trajectories just by finding apropriate V

Local Stability and Direct Method

If, in a ball $\mathcal{B}_R$, there exists a scalar function $V(\mathbf{x})$
with continuous first partial derivatives such that:

- $V(\mathbf{x})$ is positive definite (locally in $\mathcal{B}_R$)
- $\dot{V}(\mathbf{x})$ is negative semi-definite (locally in $\mathcal{B}_R$)

then the equilibrium point $\mathbf{0}$ is **stable**. If, actually, the derivative $\dot{V}(\mathbf{x})$ is locally negative definite in $\mathcal{B}_R$, then the ***stability is asymptotic***.

To dig more into the theory let us study the stability of the nonlinear system:

$$\begin{cases} \dot{x}_1 = x_1(x_1^2 + x_2^2 - 2) - 4x_1x_2^2 \\ \dot{x}_2 = 4x_1^2x_2 + x_2(x_1^2 + x_2^2 - 2) \end{cases}$$

with following Lyapunov candidate:

$$V(\mathbf{x}) = x_1^2 + x_2^2$$

```
x = sp.symbols('x_1, x_2')
V_symb = x[0]**2 + x[1]**2
print(f'Lyapunov candidate:')
V_symb
```

```
Lyapunov candidate:
```

$x_1^2 + x_2^2$

One may use a chain rule in order to find $\dot{V}$ as follows:

$$\dot{V} = \sum_{i=1}^{n} \frac{\partial V}{\partial \mathbf{x}_i} \dot{\mathbf{x}}_i = \sum_{i=1}^{n} \frac{\partial V}{\partial \mathbf{x}_i} \boldsymbol{f}_i = \nabla V \cdot \boldsymbol{f}$$

```
grad_V = sp.Matrix([V_symb]).jacobian(x)
print(f'Gradient of Lyapunov candidate:')
grad_V
```

```
Gradient of Lyapunov candidate:
```

$$\begin{bmatrix}2x_1 & 2x_2\end{bmatrix}$$

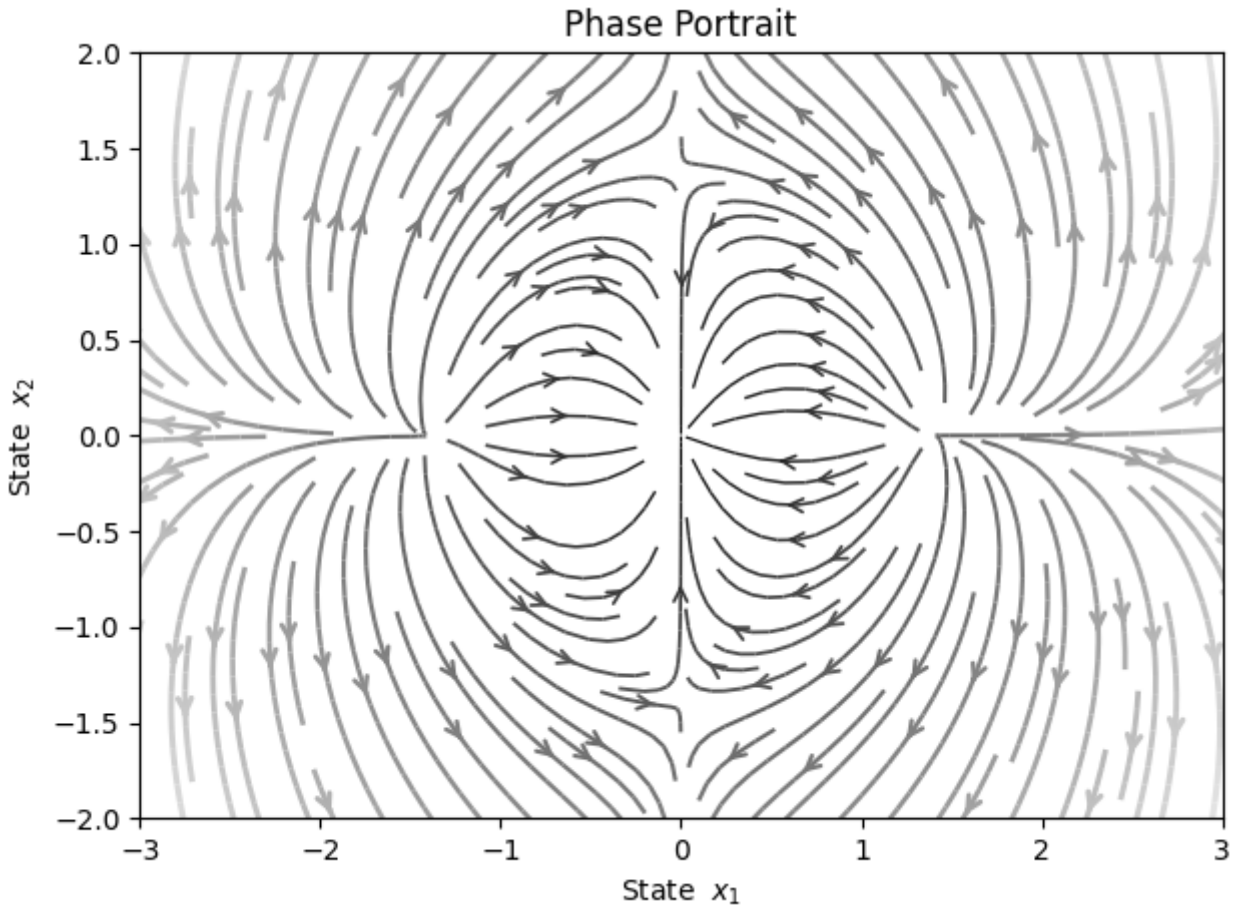
```
f_symb = sp.Matrix([x[0]*(x[0]**2 + x[1]**2 - 2) - 4*x[0]*x[1]**2,
                    4*x[0]**2 *x[1] +x[1]*(x[0]**2 + x[1]**2 - 2),])

dV = grad_V*f_symb
print(f'Time derivative of Lyapunov candidate:')
dV[0]
```

```
Time derivative of Lyapunov candidate:
```

$$2x_1\left(-4x_1x_2^2+x_1\left(x_1^2+x_2^2-2\right)\right)+2x_2\cdot\left(4x_1^2x_2+x_2\left(x_1^2+x_2^2-2\right)\right)$$

The derivative above is locally N.D in the ball $\mathbf{x}_1^2+\mathbf{x}_2^2<2$

```
# Create a numerical function from symbolic one
f_num = sp.lambdify([x], f_symb)

def f(x, t):
    dx = f_num(x)[:,0]
    return dx

phase_portrait(f, x_range=[3, 2], density=1.5)
```



Invariant Set Theorems and Lyapunov Functions

In practice we are always interested in the regions in which system is tend to stay forever, it may be stable equalibrium, the error funel etc. However the vannila Lyapunov theory does not directly allow us to find such regions. Here we can use the help of the powerful **invariant set theorems**, attributed to **La Salle** as **invariance principle**.

The central concept in these theorems is that of **invariant set**

A set $\mathcal{G}$ is an invariant set for a dynamical system if every system trajectory which starts from a point in $\mathcal{G}$ remains in $\mathcal{G}$ for all future time

$$\mathbf{x}(0) \in \mathcal{G} \Rightarrow \mathbf{x}(t) \in \mathcal{G}, \forall t \in \mathbb{R}$$

A particular examples of invariant sets are, equilibria, limit cycles, energy level sets of conservative system and any particular solution of ODE.

In most cases, it is impractical to directly verify that a set is forward invariant or to construct forward invariant sets by examining all of the trajectories of a system. Instead, if a system has continuous solutions, we can evaluate if a set is forward invariant by checking that trajectories never leave the set through its boundary. We do this by comparing the angle between the dynamics $\mathbf{f}$ and the normal to the set's boundary. Yet again the Lyapunov functions provide a convenient tool to estimate the boundaries of invariant sets. One can easily find the the boundaries of invariance set Ω defined as $\partial\Omega$ by looking on such region $\{\mathbf{x} : \mathbf{V}(\mathbf{x}) = c\}$ where $\dot{V}(\mathbf{x}) < 0$

Region of Attraction

There is another very important connection between Lyapunov functions and the concept of an invariant set: **any sublevel set of a Lyapunov function is also an invariant set**. This gives us the ability to use sublevel sets of a Lyapunov function as approximations of the **region of attraction** for nonlinear systems.

For locally stable fixed point $\mathbf{x}_e$ the region of attraction (RoA) is the largest set $\mathcal{R}$:

$$\mathbf{x}(0) \in \mathcal{R} \Rightarrow \lim_{t\to\infty} \mathbf{x}(t) = \mathbf{x}_e$$

LaSalle's Invariance

Consider an autonomous system, with $\mathbf{f}$ continuous, and let $V(\mathbf{x})$ be a scalar function with continuous
first partial derivatives. Assume that:

- for some $r > 0$ the region Ω_r defined by $V(\mathbf{x}) < r$ is bounded
- $\dot{V}(\mathbf{x}) \leq 0$ for all $\mathbf{x} \in \Omega_r$

Let $\mathcal{R}$ be the set of all points within Ω_r where $\dot{V}(\mathbf{x}) = 0$, and $\mathcal{M}$ be the largest invariant set in $\mathcal{R}$. Then, every solution $\mathbf{x}$ originating in Ω_r tends to $\mathcal{M}$ as $t \to \infty$

Furthermore, if
in , then the origin is locally asymptotically stable and the set is inside the region of attraction of this fixed point. Alternatively, if
in and is the only invariant subset of where
, then the origin is asymptotically stable and the set is inside the region of attraction of this fixed point.

Estimation of Region of Attraction via Sampling

The idea is to test conditions of theorem on some domain of state space, more specifically on random points $\mathbf{x}_i$ within this domain. If conditions of theorem fails at least for one sample, namely $\dot{V}(x_i) \geq 0$ then the level set $V(\mathbf{x}) = V(\mathbf{x}) = c_i$ is not fully contained inside the region of attraction. Testing for the *large* amount of samples intuitevelly will yield the upper estimate of level set $V(\mathbf{x}) < c < \hat{c}_u$. While increasing the amount of samples $\mathbf{x}_i$ one would expect for roa estimates to be tighter.

```
def sampling_roa(f, V, nablaV, x_bounds, N=1000):
    x_min, x_max = x_bounds
    x_range = np.array(x_max) - np.array(x_min)
    n = np.shape(x_min)[0]

    c = np.inf
    ct = []

    for i in range(N):
        x_i = x_min + x_range*np.random.rand(n)
        V_i = V(x_i)
        if V_i <= c:
            dV_i = np.array(nablaV(x_i))@ np.array(f(x_i))
            if dV_i >= 0:
                c = V_i
                ct.append(c)
    return c, ct
```

Example: The RoA for cubic system

Let us test RoA estimates on the cubic dynamical sytem:

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

```
def f(x):
    return -x +x**3

def V(x):
    return x**2

def nablaV(x):
    return 2*x

x_bounds = [-2], [2]


r, rt = sampling_roa(f, V, nablaV, x_bounds, N=500)
print(f'The set $V(x)<{r[0]}$ is estimate of the region of attraction')
# print(rt)
```

```
The set $V(x)<1.0111317838631 38$ is estimate of the region of attraction
```

We can do very same with various systems and Lyapunov functions. Lets for intance use the

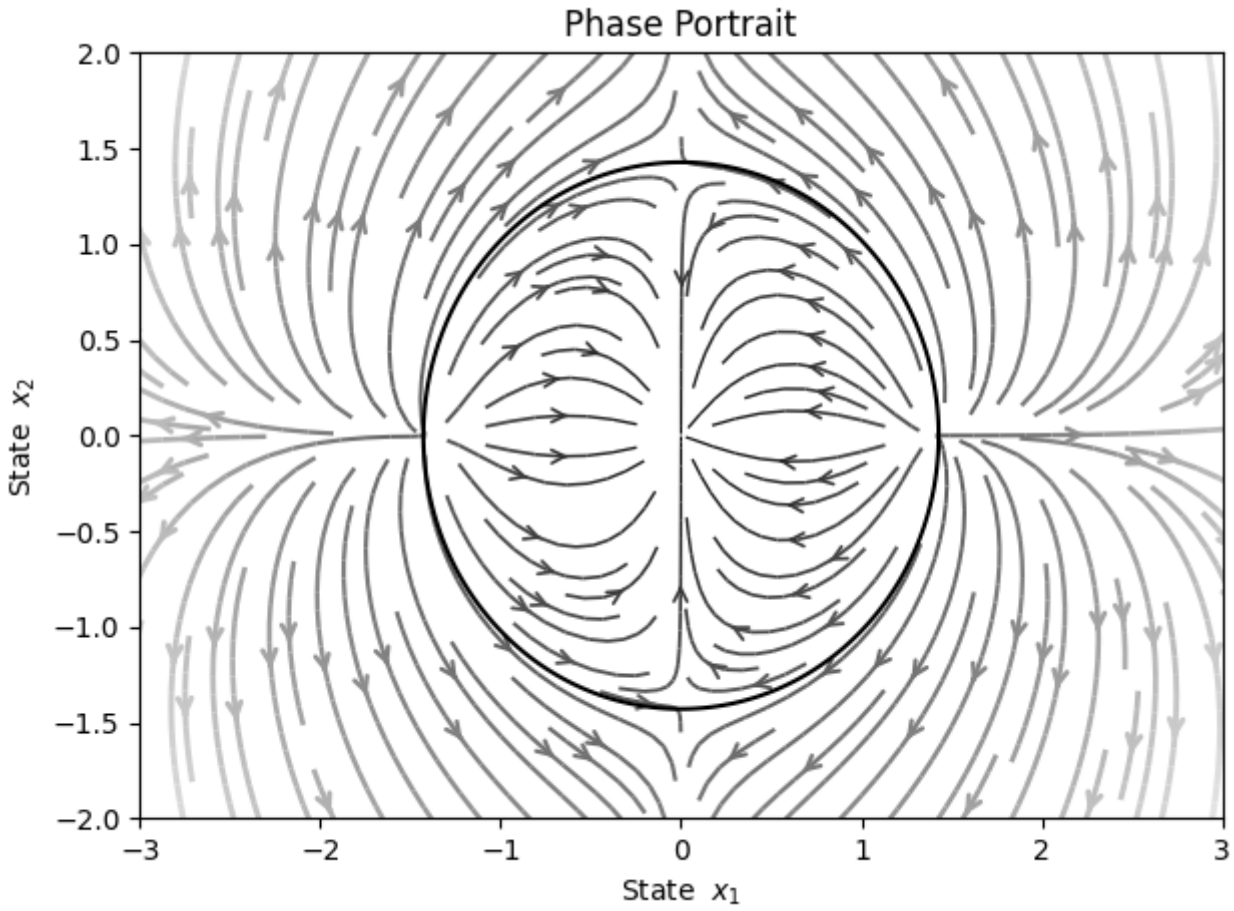
```
def f(x,t = 0):
    dx1 = x[0]*(x[0]**2 + x[1]**2 - 2) - 4*x[0]*x[1]**2
    dx2 = 4*x[0]**2 *x[1] +x[1]*(x[0]**2 + x[1]**2 - 2)
    return dx1, dx2

def V(x):
    return x[0]**2 + x[1]**2


x_sym = sp.symbols('x_1, x_2')
V_sym = V(x_sym)

grad_V = sp.Matrix([V_sym]).jacobian(x)
grad_V
```

$$\begin{bmatrix} 2x_1 & 2x_2 \end{bmatrix}$$

```
nablaV = sp.lambdify([x_sym], grad_V)


x_bounds = [-4, -4], [4, 4]

r, rt = sampling_roa(f, V, nablaV, x_bounds, N=1000)
r
```

```
2.0400317572945017
```

```
phase_portrait(f, x_range=[3, 2], density=1.5)

# P, r = roa_params
delta = 0.025
x1range = np.arange(-2, 2, delta)
x2range = np.arange(-3, 3, delta)
X1, X2 = np.meshgrid(x1range,x2range)
X = [X1, X2]
ROA = V(X) - r
# ROA = P[0,0]*X1**2 + P[0,1]*2*X1*X2 + P[1,1]*X2**2 - r
plt.contour( X1, X2, ROA, [0], colors='k')
plt.show()
```



As we can see we are able to get satisfactory results in estimating RoA when **Lyapunov function is given** and match the shape of the desired region, however, it is not clear how to obtain the good candidate for general system.

Exercise: Region of attraction for Cart Pole balancing

Let us now test the sampling method on roa estimation for linear controllers.
As example consider the cart pole system presented in the colab notebook.

Suppose that somebedy already give you the values for feedback gains for full state controller $\mathbf{u} = -\mathbf{K}\mathbf{x}$, where $\mathbf{x} = [\theta, \dot{\theta}, p, \dot{p}]^T$:

```
K = np.array([[139.78445986, 25.00055637, -31.6227766, -29.03589873]])
```

The Lyapunov function is given as well as $\mathbf{V} = \mathbf{x}^T\mathbf{P}\mathbf{x}$, with p.d. $\mathbf{P}$:

```
P = np.array([[19.43822824,  2.62513909, -7.90587009, -5.95477445],
              [ 2.62513909,  0.43677025, -1.30436178, -0.9558897 ],
              [-7.90587009, -1.30436178,  9.18195739,  3.71541707],
              [-5.95477445, -0.9558897 ,  3.71541707,  2.60558103]])
```

For the given feedback and Lyapunov candidate the estimate for the region of attraction

The machinery so far has used optimization to find the largest region of attraction that can be certified **given a candidate Lyapunov function**.

Meanwhile for the linear systems the candidate is given by the solution of Lyapunov equation. So we can use this as a good starting point. The other approach is to use so called cost-to-go which is readily provided by optimal methods like LQR that we will study in further lectures.

Linearization based Lyapunov Candidate for RoA Estimation

For some stable fixed points, we can certify the local stability with a linear analysis, and this linear analysis gives us a candidate quadratic Lyapunov function:

$$V = \mathbf{x}^T\mathbf{P}\mathbf{x}$$

where $\mathbf{A}$ is the linearization of original nonlinear dynamics nearby $\mathbf{x}_e$ and $\mathbf{Q}$ is p.d matrix.

Given $\mathbf{Q}$ one can then solve Lyapunov equation :

$$\mathbf{A}^T\mathbf{P} + \mathbf{A}\mathbf{P} = -\mathbf{Q}$$

And find region of attraction as level set $\{\mathbf{x} : \mathbf{x}^T\mathbf{P}\mathbf{x} \leq r\}$

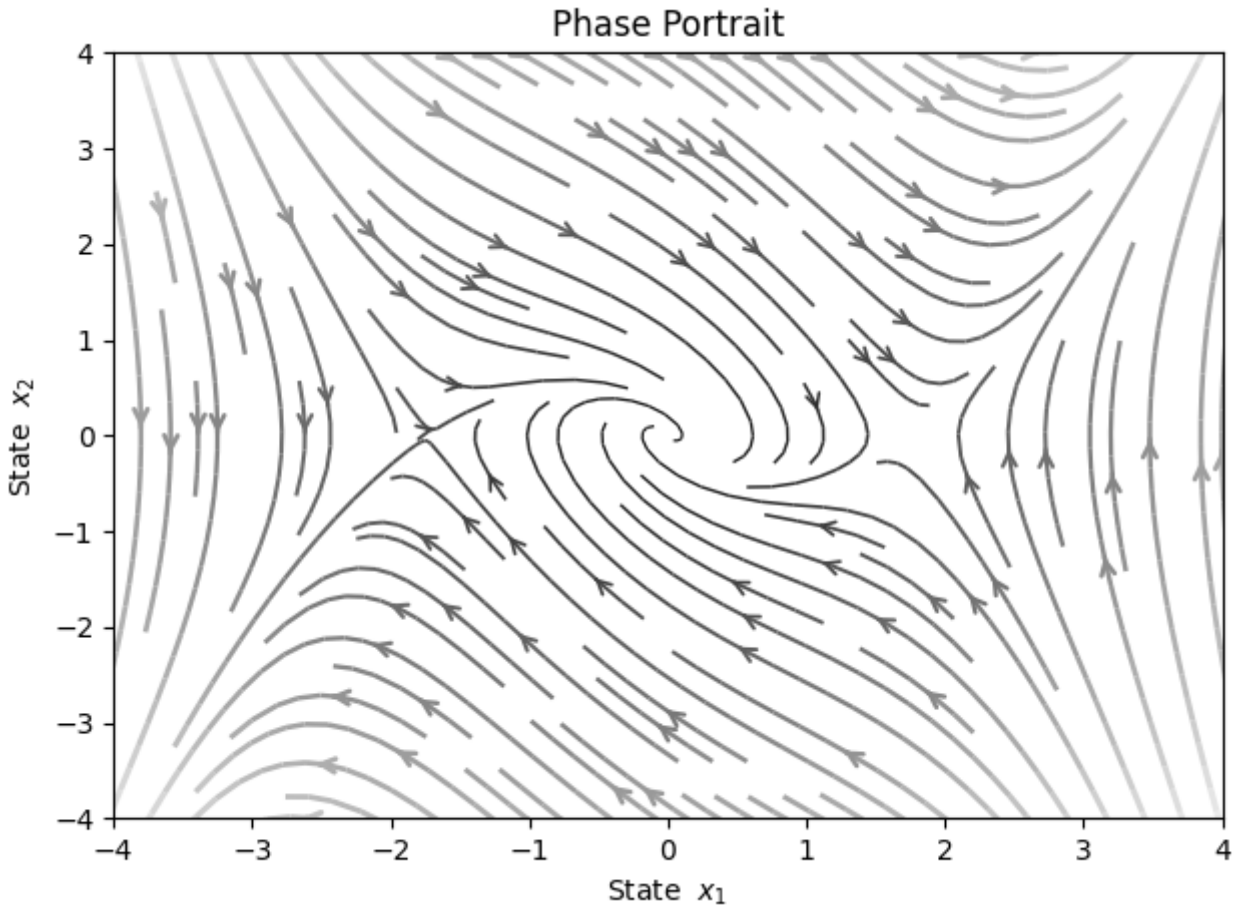
Example: Two dimensional cubic system

Let us test this approach on following system:

```
def f(x, t =0):
    x1, x2 = x
    dx1 = x2
    dx2 = -x1 + x1**3/3 - x2
    return dx1, dx2

x = sp.symbols(r'x_1, x_2')
f_sym = sp.Matrix([f(x, 0)]).T
f_sym
```

$$\begin{bmatrix} x_2 \\ \frac{x_1^3}{3} - x_1 - x_2 \end{bmatrix}$$

```
phase_portrait(f, x_range=[4, 4], density=1.2)
```



Firstly we find the Jacobian and evaluate at equilibrium:

```
jacobian = f_sym.jacobian(x)
jacobian
```

$$\begin{bmatrix} 0 & 1 \\ x_1^2 - 1 & -1 \end{bmatrix}$$

```
jacobian_num = sp.lambdify([x], jacobian)
A = np.array(jacobian_num([0, 0]), dtype='double')
print(f'The linearization nearby equilibrium is given by:\n{A}')
```

```
The linearization nearby equilibrium is given by:
[[ 0.  1.]
 [-1. -1.]]
```

```
from scipy.linalg import solve_continuous_lyapunov as lyap
Q = array([[1, 0],
           [0, 1]])
P = lyap(A.T, -Q)
print(f'The solution of Lyapunov equation is:\n{P}')
print(f'\nThe eigen values are:\n{eig(P)[0]}')
```

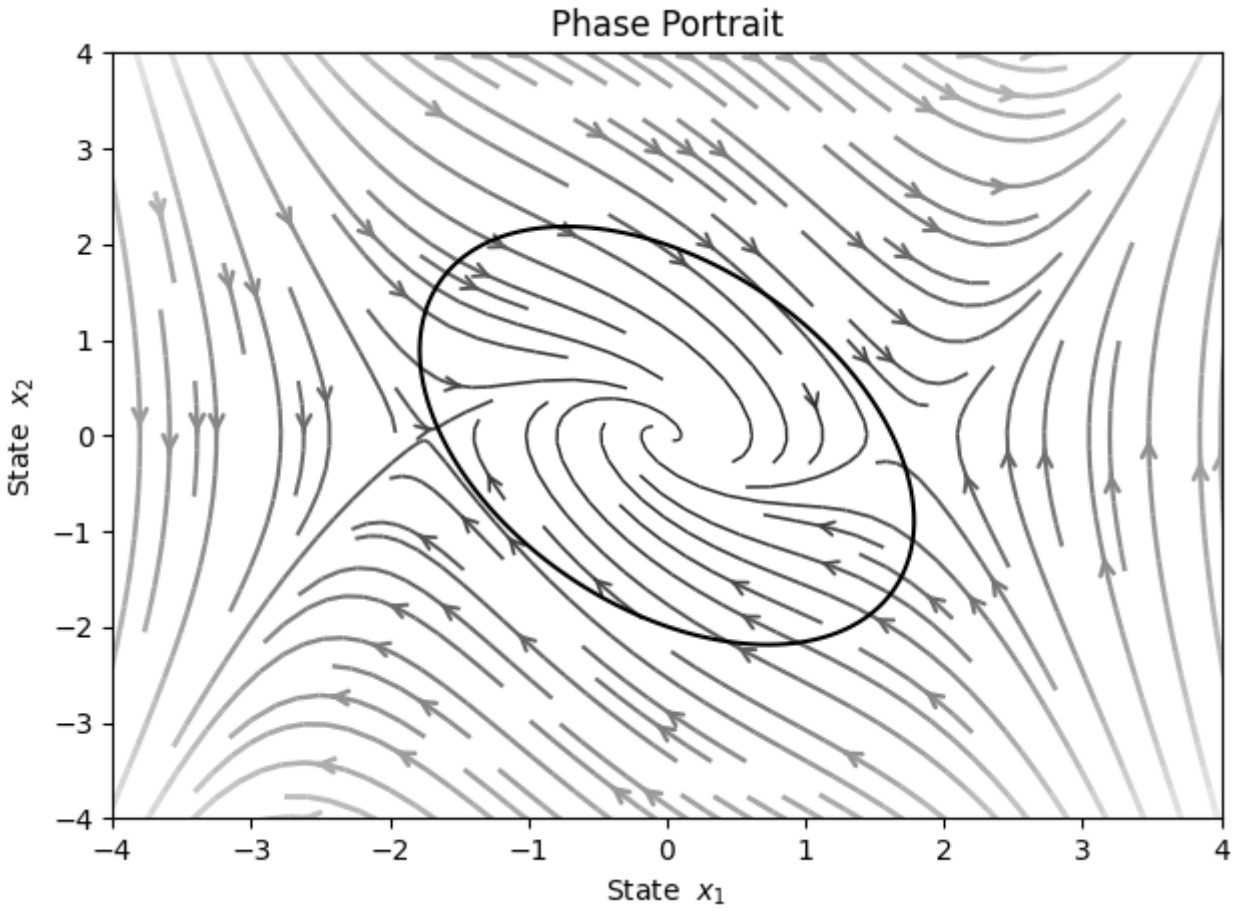
```
The solution of Lyapunov equation is:
[[1.5 0.5]
 [0.5 1. ]]

The eigen values are:
[1.80901699 0.69098301]
```

Once $\mathbf{P}$ is given we define Lyapunov candidate as $\mathbf{x}^T\mathbf{P}\mathbf{x}$ and estimate the region of attraction:

```
def V(x):
    return x[0]**2 *P[0,0] + 2*x[0]*x[1]*P[0,1] + x[1]**2 *P[1,1]

x_sym = sp.symbols('x_1, x_2')
V_sym = V(x_sym)

grad_V = sp.Matrix([V_sym]).jacobian(x)
nablaV = sp.lambdify([x_sym], grad_V)

x_bounds = [-4, -4], [4, 4]

r, rt = sampling_roa(f, V, nablaV, x_bounds, N=10000)
r
```

```
3.9759506557685684
```

```
phase_portrait(f, x_range=[4, 4], density=1.2)

# P, r = roa_params
delta = 0.025
x1range = np.arange(-4, 4, delta)
x2range = np.arange(-4, 4, delta)
X1, X2 = np.meshgrid(x1range,x2range)
X = [X1, X2]
ROA = V(X) - r
plt.contour( X1, X2, ROA, [0], colors='k')
plt.show()
```


Phase Portrait
State x_2
State x_1
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3
4