Complex Systems

CS2024/problem_1.pdf

---

1. Nonlinear dynamics

The purpose of this project is to familiarize you with the basics of nonlienar dynamics: fixed points, stability, and linearization technique.

1.1 Find the fixed points $f(x^*) = 0$ for the equation:

$$\dot{x} = f(x) = x(x-1)(x-2)$$

Answer:

$$x_1 = 0, \quad x_2 = 1, \quad x_3 = 2$$

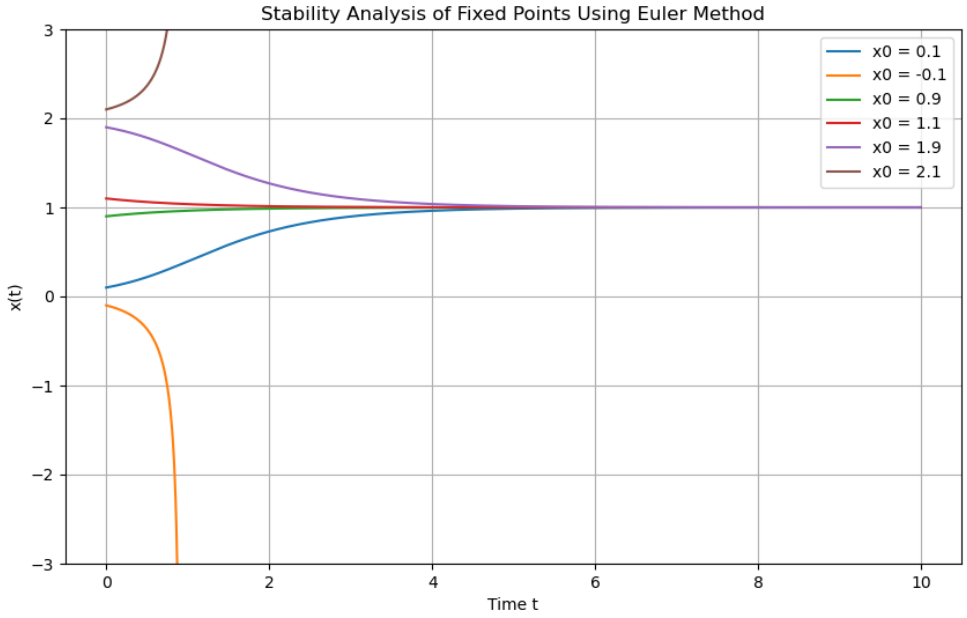
and analyze their stability by solving the equation numerically. For this purpose, choose initial conditions from the vicinity of x_* and observe the behavior of $x(t)$. Use the simplest Euler method (its only advantage is simplicity),

Euler's Method:

$$y_1 = y_0 + h * f(t_0, y_0)$$

```
import numpy as np
import matplotlib.pyplot as plt


def f(x):
    return x * (x - 1) * (x - 2)

# Euler method for numerical integration

def euler_method(f, x0, dt, t_end):
    t_values = np.arange(0, t_end, dt)
    x_values = np.zeros(len(t_values))
    x_values[0] = x0

    for i in range(1, len(t_values)):
        x_values[i] = x_values[i - 1] + dt * f(x_values[i - 1])

    return t_values, x_values


# Parameters
dt = 0.01  # time step
t_end = 10  # end time

# initial conditions close to fixed points
initial_conditions = [0.1, -0.1, 0.9, 1.1, 1.9 ,2.1]

# Simulate and plot for each initial condition
plt.figure(figsize=(10, 6))
```

```
    for x0 in initial_conditions:
        t_values, x_values = euler_method(f, x0, dt, t_end)
        plt.plot(t_values, x_values, label=f"x0 = {x0}")
        plt.ylim(-3, 3)

    plt.xlabel("Time t")
    plt.ylabel("x(t)")
    plt.title("Stability Analysis of Fixed Points Using Euler Method")
    plt.legend()
    plt.grid(True)
    plt.show()
```

```Python (version 3.11.7)




We can see on the plot that the stability point is around 1.

1.2. Implement an approximate method for drawing the phase portraits, and apply it to the following
equations:

$$\ddot{x} = -x,$$
$$\ddot{x} = -\sin(x),$$
$$\ddot{x} = -x + x^3,$$
$$\ddot{x} = x - x^3$$

In the first step, by substituting $\dot{x} = y$, reduce the second-order equation to a system of two first-order equations. In subsequent steps, for given initial conditions $(x_0, y_0)$, i.e., the starting point in the phase space, we numerically calculate the trajectory $(x(t), y(t))$. Use the so-called midpoint method (one of the second-order Runge-Kutta methods):
```

$$k_x = \Delta t f_x(x_n, y_n),$$
$$k_y = \Delta t f_y(x_n, y_n),$$
$$x_{n+1} = x_n + \Delta t f_x(x_n + \frac{k_x}{2}, y_n + \frac{k_y}{2}),$$
$$y_{n+1} = y_n + \Delta t f_y(x_n + \frac{k_x}{2}, y_n + \frac{k_y}{2}),$$

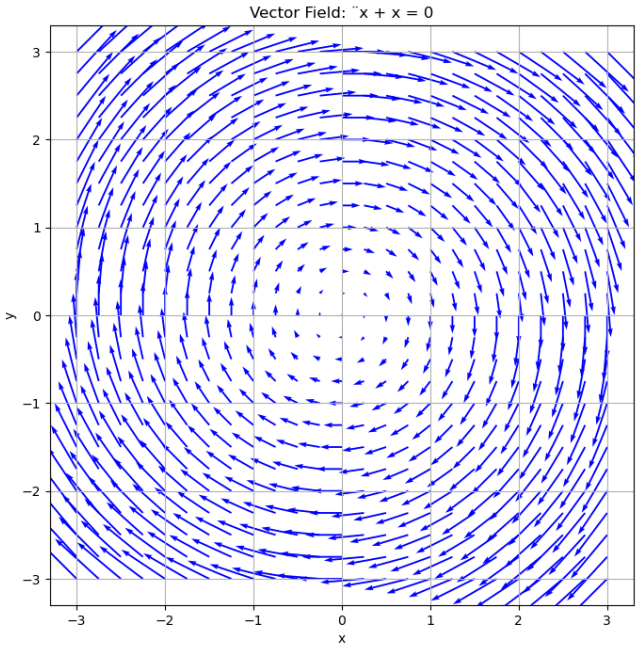
Repeat the entire procedure for different, evenly distributed starting points in the phase space. For the obtained phase portraits, determine the isoclines and describe how trajectories behave in their vicinity. What property do points at the intersection of isoclines have?

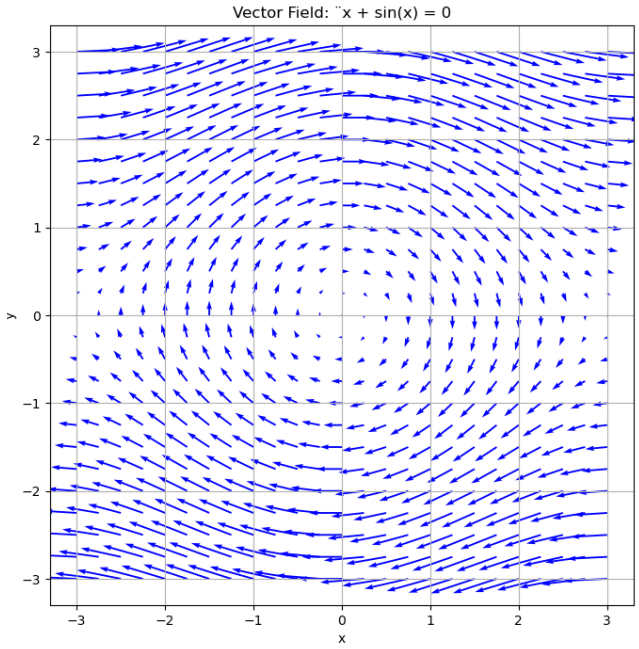
Answer: Intersection of isoclines is significant because it typically marks the locations of fixed points, and analyzing these intersections can help determine the stability and overall behavior of the system near these points.

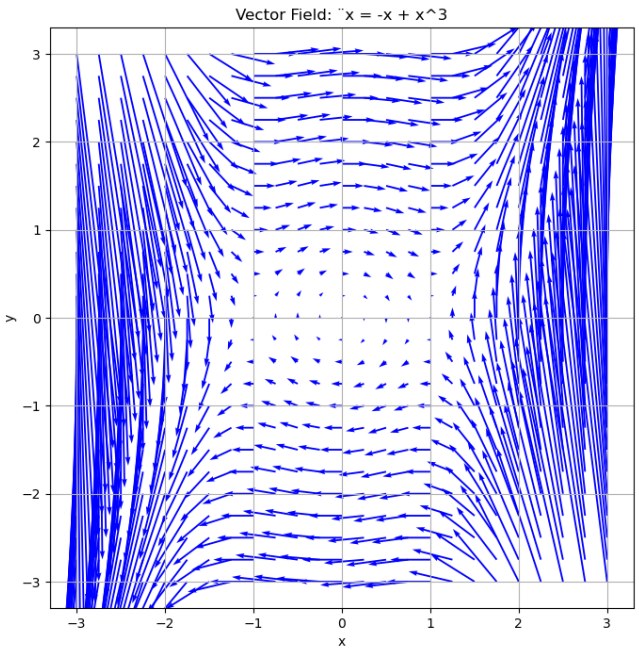
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-3, 3, 25)
y = np.linspace(-3, 3, 25)
X, Y = np.meshgrid(x, y)

def vector_field_1(X, Y):
    return Y, -X


def vector_field_2(X, Y):
    return Y, -np.sin(X)


def vector_field_3(X, Y):
    return Y, -X + X**3


def vector_field_4(X, Y):
    return Y, X - X**3



def plot_clear_vector_field(system, system_name):
    U, V = system(X, Y)

    plt.figure(figsize=(8, 8))
    plt.quiver(X, Y, U, V, color='blue', scale=40, width=0.003)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title(f"Vector Field: {system_name}")
    plt.grid(True)
    plt.show()
```

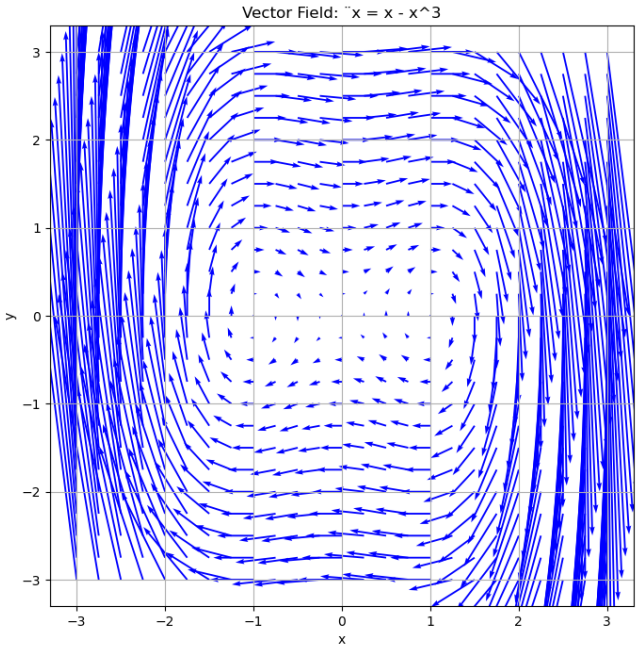
```
    vector_fields = [
        (vector_field_1, "¨x + x = 0"),
        (vector_field_2, "¨x + sin(x) = 0"),
        (vector_field_3, "¨x = -x + x^3"),
        (vector_field_4, "¨x = x - x^3")
    ]

    for system, name in vector_fields:
        plot_clear_vector_field(system, name)
```

```Python (version 3.11.7)
```

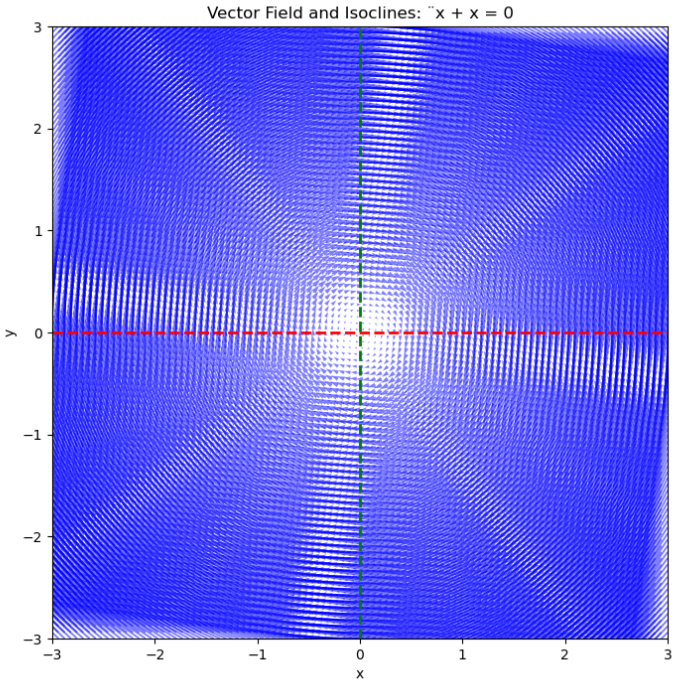


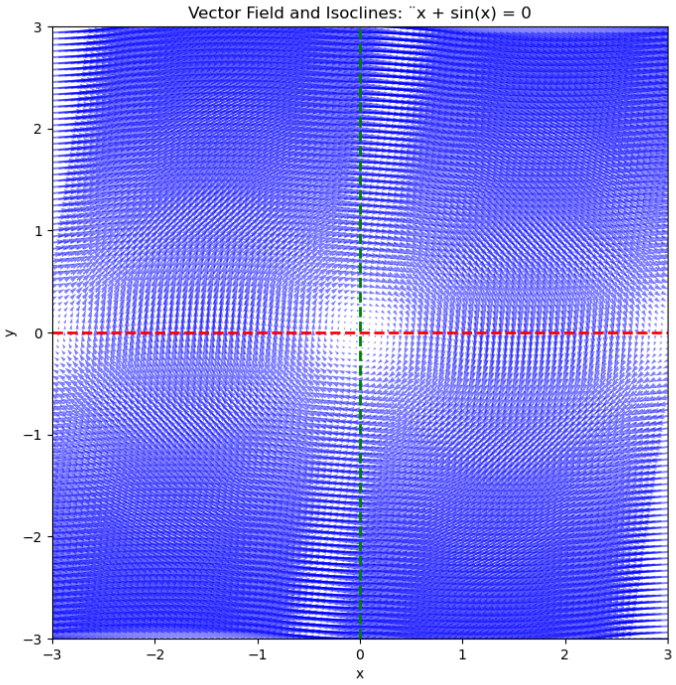


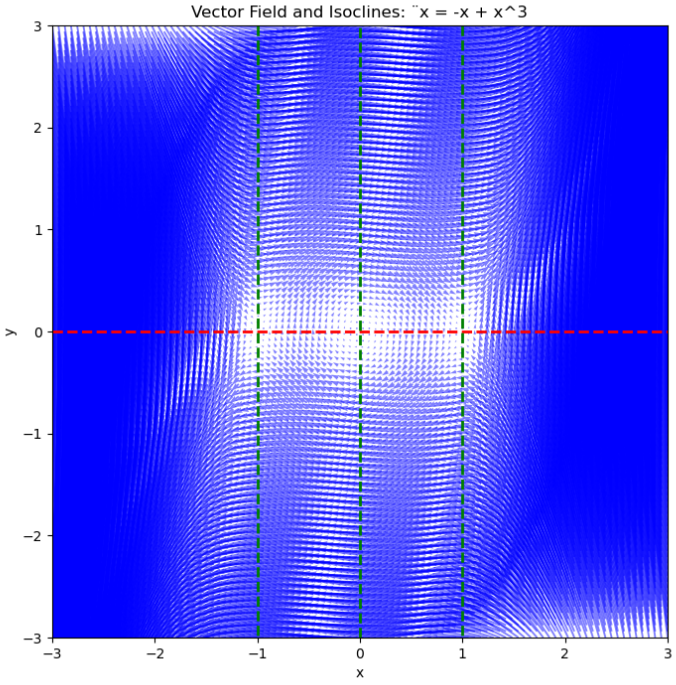


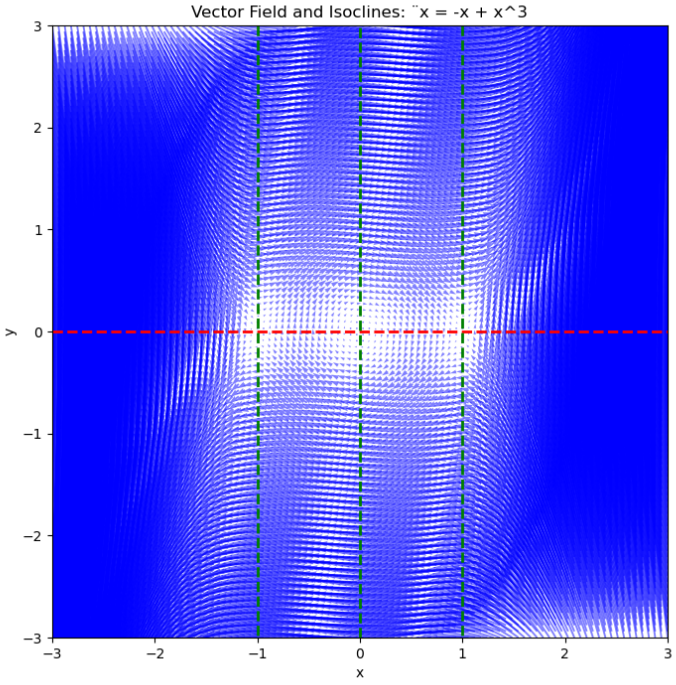


```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-3, 3, 100)
y = np.linspace(-3, 3, 100)
X, Y = np.meshgrid(x, y)

def vector_field_1(X, Y):
    return Y, -X


def vector_field_2(X, Y):
    return Y, -np.sin(X)


def vector_field_3(X, Y):
    return Y, -X + X**3


def vector_field_4(X, Y):
    return Y, X - X**3


def isocline_plot(system, system_name):
    U, V = system(X, Y)

    plt.figure(figsize=(8, 8))
    plt.quiver(X, Y, U, V, color='blue', scale=40, width=0.003, alpha=0.5)

    plt.contour(X, Y, U, levels=[0], colors='red',
                linewidths=2, linestyles='--')
    plt.contour(X, Y, V, levels=[0], colors='green',
                linewidths=2, linestyles='--')

    plt.xlabel("x")
    plt.ylabel("y")
    plt.title(f"Vector Field and Isoclines: {system_name}")
    # plt.grid(True)
    plt.show()

vector_fields = [
    (vector_field_1, "¨x + x = 0"),
    (vector_field_2, "¨x + sin(x) = 0"),
    (vector_field_3, "¨x = -x + x^3"),
    (vector_field_4, "¨x = x - x^3")
]

for system, name in vector_fields:
    isocline_plot(system, name)
```
```Python (version 3.11.7)

This code will generate plots for each of the four systems, showing both the vector field and the isoclines.We can see on the crossings of the red lines in isoclines plots the stable point.
```


Vector Field and Isoclines: ¨x + x = 0
x
y


Vector Field and Isoclines: ¨x + sin(x) = 0
x
y


Vector Field and Isoclines: ¨x = -x + x^3
x
y


Vector Field and Isoclines: ¨x = -x + x^3
x
y

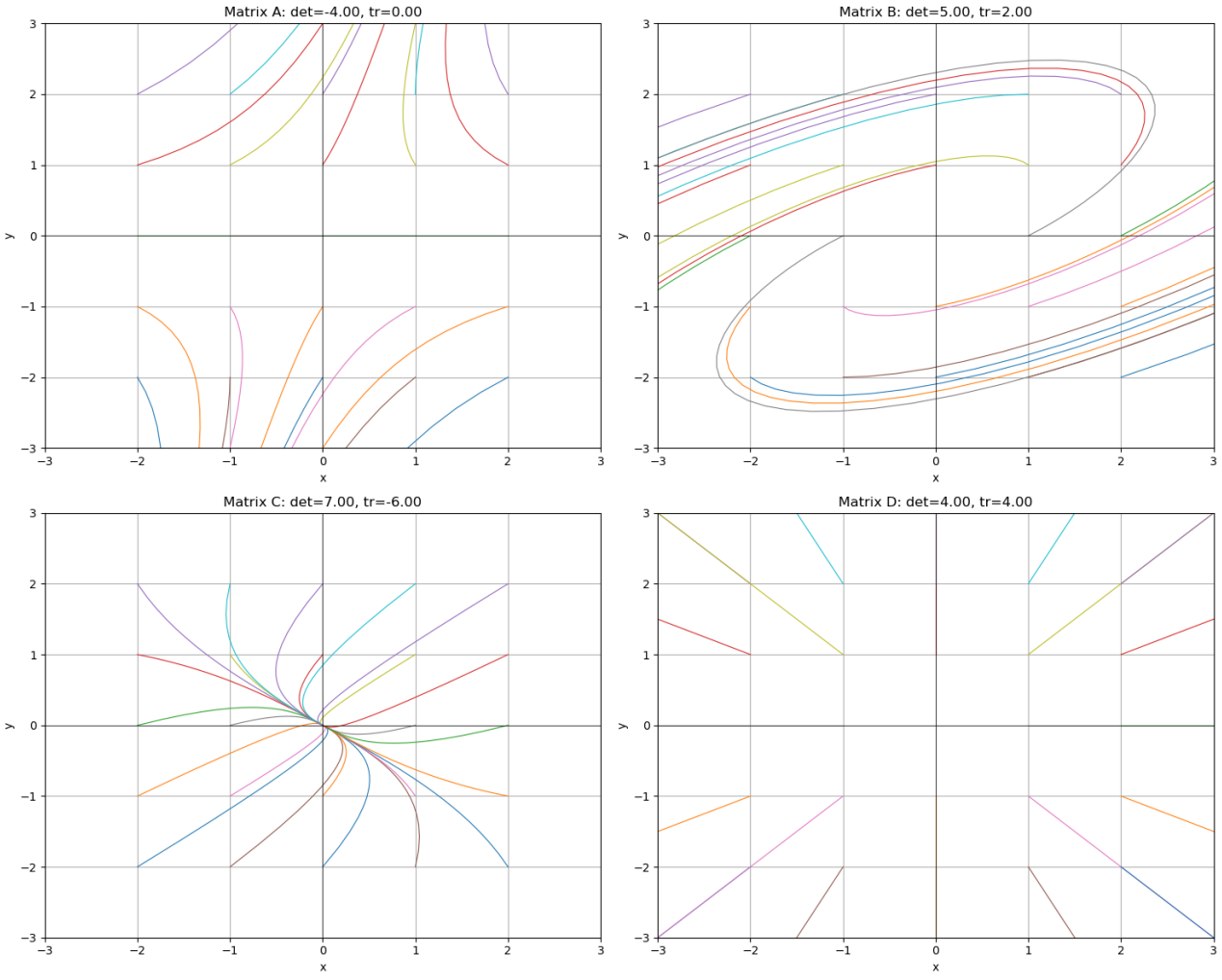
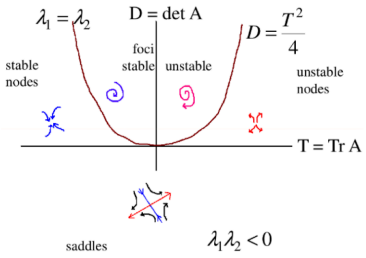
1.3. Use the implemented method for drawing phase portraits to analyze the behavior of trajectories in the vicinity of fixed points for the linear equation $\dot{x} = A\,x$, where $x \in \mathbb{R}^2$and $A$is given by the following matrices:

$$a)\begin{pmatrix} -2 & 1 \\ 0 & 2 \end{pmatrix}\ b)\begin{pmatrix} 3 & -4 \\ 2 & -1 \end{pmatrix}\ c)\begin{pmatrix} -3 & -2 \\ -1 & -3 \end{pmatrix}\ d)\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

Please verify the obtained behavior of the trajectories according to the diagram by calculating the determinant (det A) and the trace (tr A):

```
import numpy as np
import matplotlib.pyplot as plt

# Define matrices
A_matrices = {
    'a': np.array([[-2, 1], [0, 2]]),
    'b': np.array([[3, -4], [2, -1]]),
    'c': np.array([[-3, -2], [-1, -3]]),
    'd': np.array([[2, 0], [0, 2]])
}

def calculate_det_trace(A):
    det = np.linalg.det(A)
    tr = np.trace(A)
    return det, tr


def midpoint_method_linear(A, x0, dt, t_end):
    t_values = np.arange(0, t_end, dt)
    x_values = np.zeros((len(t_values), 2))
    x_values[0] = x0

    for i in range(1, len(t_values)):
        k = dt * A @ x_values[i - 1]
        x_values[i] = x_values[i - 1] + dt * A @ (x_values[i - 1] + 0.5 * k)

    return x_values[:, 0], x_values[:, 1]


dt = 0.05
t_end = 10
initial_conditions = [(x, y) for x in np.linspace(-2, 2, 5)
                      for y in np.linspace(-2, 2, 5)]

fig, axes = plt.subplots(2, 2, figsize=(15, 12))

for idx, (key, A) in enumerate(A_matrices.items()):
    ax = axes[idx // 2, idx % 2]

    det, tr = calculate_det_trace(A)

    for x0, y0 in initial_conditions:
        x_vals, y_vals = midpoint_method_linear(
            A, np.array([x0, y0]), dt, t_end)
        ax.plot(x_vals, y_vals, lw=0.8)
```

```
        ax.set_title(f"Matrix {key.upper()}: det={det:.2f}, tr={tr:.2f}")
        ax.set_xlabel("x")
        ax.set_ylabel("y")
        ax.grid(True)
        ax.axhline(0, color='black', lw=0.5)
        ax.axvline(0, color='black', lw=0.5)
        ax.set_xlim(-3, 3)
        ax.set_ylim(-3, 3)

    plt.tight_layout()
    plt.show()
```

```Python (version 3.11.7)




And by using provided graf we can determinate solution by performing:




$$D = \frac{T^2}{A}$$
```

4. Sketch the phase portrait for the nonlinear system:

$$\begin{cases} \dot{x} = x(3 - x - 2y), \\ x_{n+1} = y(2 - x - y)\,. \end{cases}$$

Find the fixed points and determine the behavior of the trajectories in their vicinity. To illustrate the orientation of the trajectories, you can change the color of the curve based on the value of the progressing time. Lotka-Volterra model:

Let $x, y \geqslant 0$. Modify the system (equations) to obtain a stable nonzero population for both species.

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, Eq, solve

# Define differential equations


def dx_dt(x, y):
    return x * (3 - x - 2 * y)


def dy_dt(x, y):
    return y * (2 - x - y)


# Create a grid for plotting
x = np.linspace(0, 3, 20)
y = np.linspace(0, 3, 20)
X, Y = np.meshgrid(x, y)
U = dx_dt(X, Y)
V = dy_dt(X, Y)

# Generate the phase portrait with more details
plt.figure(figsize=(8, 8))
plt.quiver(X, Y, U, V, color='blue', width=0.005, alpha=0.6)

# Add more details to the plot
plt.xlabel("x", fontsize=12, weight='bold')
plt.ylabel("y", fontsize=12, weight='bold')
plt.title("Phase Portrait for Nonlinear System", fontsize=14, weight='bold')
plt.grid(True, linestyle='--', alpha=0.6)

# Highlight the axes
plt.axhline(0, color='black', linewidth=0.5, linestyle='--')
plt.axvline(0, color='black', linewidth=0.5, linestyle='--')

# Use larger arrows to show direction more clearly
plt.quiver(X, Y, U, V, color='blue', width=0.005, alpha=0.6, scale=40)

# Add and label the fixed points
x_sym, y_sym = symbols('x y')
equations = [
    Eq(dx_dt(x_sym, y_sym), 0),
    Eq(dy_dt(x_sym, y_sym), 0)
```

```
]

fixed_points = solve(equations, (x_sym, y_sym))
for point in fixed_points:
    plt.plot(point[0], point[1], 'ro')  # Fixed points in red
    plt.text(point[0] + 0.1, point[1] + 0.1,
             f'({point[0]}, {point[1]})', color='red')

# Save and show the plot
plt.savefig("phase_portrait_detailed.png")
plt.show()

# Print fixed points for reference
print("Fixed Points:", fixed_points)
```

```Python (version 3.11.7)
```

We have obtain the phase portrait for Nonlinear System