BME Faculty of Mechanical Eng.	NONLINEAR VIBRATIONS	Name: Kelle Gergő
Department of Applied Mechanics	HOMEWORK 1	Neptun code: GBBNUL
Semester: 2023/24/2 (Spring)	Deadline: 2024 April 09	Signature: Kelle Gergő

Analysis of a nonlinear system

The differential equations of a nonlinear dynamical system are given as follows:

$$\begin{aligned} \dot{x} &= f(x,y) \equiv -xy^2 - 4xy + 2y^2 + 4y \\ \dot{y} &= g(x,y) \equiv -x^3 - xy^2 - 4x^2 + 5x \end{aligned}$$

Analyze the possible motions and sketch the phase space of the system.

Detailed tasks

1. Determine the *nullclines* of the phase plane. The nullclines are the curves in the xy plane which satisfy either $f(x,y) = 0$ (the vector field is "vertical") or $g(x,y) = 0$ (the vector field is "horizontal").
2. Find the equilibrium points. Sketch the nullcline curves and the equilibrium points in the phase plane.
3. Write down the linearized differential equations at each equilibrium point. Determine the type of the equilibrium points (saddle, node, focus, centre, non-generic) and the eigenvectors of the saddles and nodes.
4. Show the equilibrium points in the Trace–Det plane.
5. Make a *freehand drawing* of the phase space containing as many details as you can determine from the analysis (e.g. equilibrium points, eigenvectors, some typical trajectories, asymptotic behaviour).
6. Check the freehand phase portrait by using any kind of computer algebra or numerical software package (Recommended softwares: XPPAUT, Mathematica, Maple, Matlab, Octave, Maxima).

Results

Fill the table with the properties of the equilibrium points! Indicate the equilibrium points with a $\star$ which are at multiple roots of the nullclines. For each data set, there exist different number of equilibria, please strike through the unused cells. Please use the reference numbers (1,2,...) of the equilibrium points all along the documentation!

	1	2	3	4	5	6	7
x coord.	0	-5	1	0	-2		
y coord.	0	0	0	-2	-3		
type	Saddle	Center	Non-generic	Unstable node	Saddle		
eigenvalues	4.47 + 0i	0 + 26.83i	0	2 + 0i	3 + 0i		
	-4.47 + 0i	0 - 26.83i	0	2 + 0i	-12 + 0i		

MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Mechanical Engineering

Nonlinear Vibrations
I. Homework

Gergő Kelle

April 9, 2024

Contents

1 Task - Determining the nullclines of the phase plane

To determine the nullclines we have to solve the problem for the case when $f(x, y)$ and $g(x, y) = 0$. The given equations for generated for my NEPTUN code is showed in Eqation (1) and (2).

$$\dot{x} = f(x, y) = -xy^2 - 4xy + 2y^2 + 4y \tag{1}$$

$$\dot{y} = g(x, y) = -x^3 - xy^2 - 4x^2 + 5x \tag{2}$$

While the trivial solution ($x = 0,\ y = 0$) is evident, we are equally concerned with nontrivial solutions. Simplifying Equation (2), we can identify additional nullclines:

$$0 = x \underbrace{\left(-x^2 - y^2 - 4x + 5\right)}_{=0}. \tag{3}$$

This reorganized equation reveals $x = 0$ as a trivial solution. Further solutions can be found if we consider the part inside the brackets as zero.

$$0 = -x^2 - y^2 - 4x + 5 \tag{4}$$

$$y = \pm\sqrt{-x^2 - 4x + 5} \tag{5}$$

Similarly, analyzing Equation (1) yields:

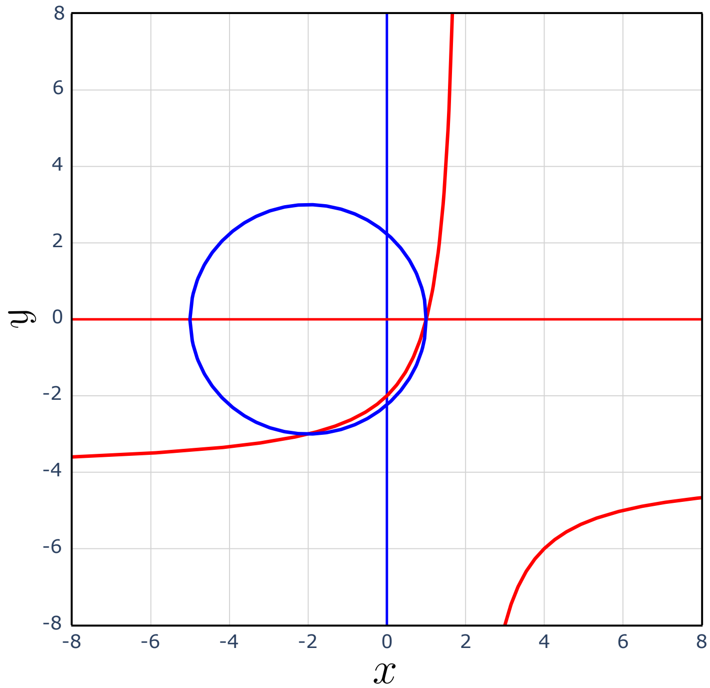
$$0 = y \underbrace{\left(-xy - 4x + 2y + 4\right)}_{=0} \tag{6}$$

Acknowledging $y = 0$ as a potential solution, considering the bracketed expression offers another nullcline curve:

$$0 = -xy - 4x + 2y + 4 \tag{7}$$

$$y = -\frac{4\,(x-1)}{x-2} \tag{8}$$

Where the constraint $x \neq 2$ arises due to division by zero.



$f(x,y) = 0$:

$$y = -\frac{4(x-1)}{x-2} \tag{8}$$

$g(x,y) = 0$:

$$y = \pm\sqrt{-x^2 - 4x + 5} \tag{5}$$

Figure 1: The nullclines of the nonlinear dynamical system.

2 Task - Find the equlibrium points

The equilibrium points arise at the intersections of the nullcline curves. Consequently, the equilibrium points, including the trivial solution, are as follows:

i	**Equilibrium Points**
1	$(0,0)$
2	$(-5,0)$
3	$(1,0)$
4	$(0,-2)$
5	$(-2,-3)$

Table 1: Equlibrium points

For the calculations, the *Python* programming language was employed, with the code provided in the appendix. Notably, two of the equilibrium points manifest as complex numbers, thereby disregarded, as a real dynamical system cannot attain equilibrium at non-real points. The visualization is presented in Figure 2.

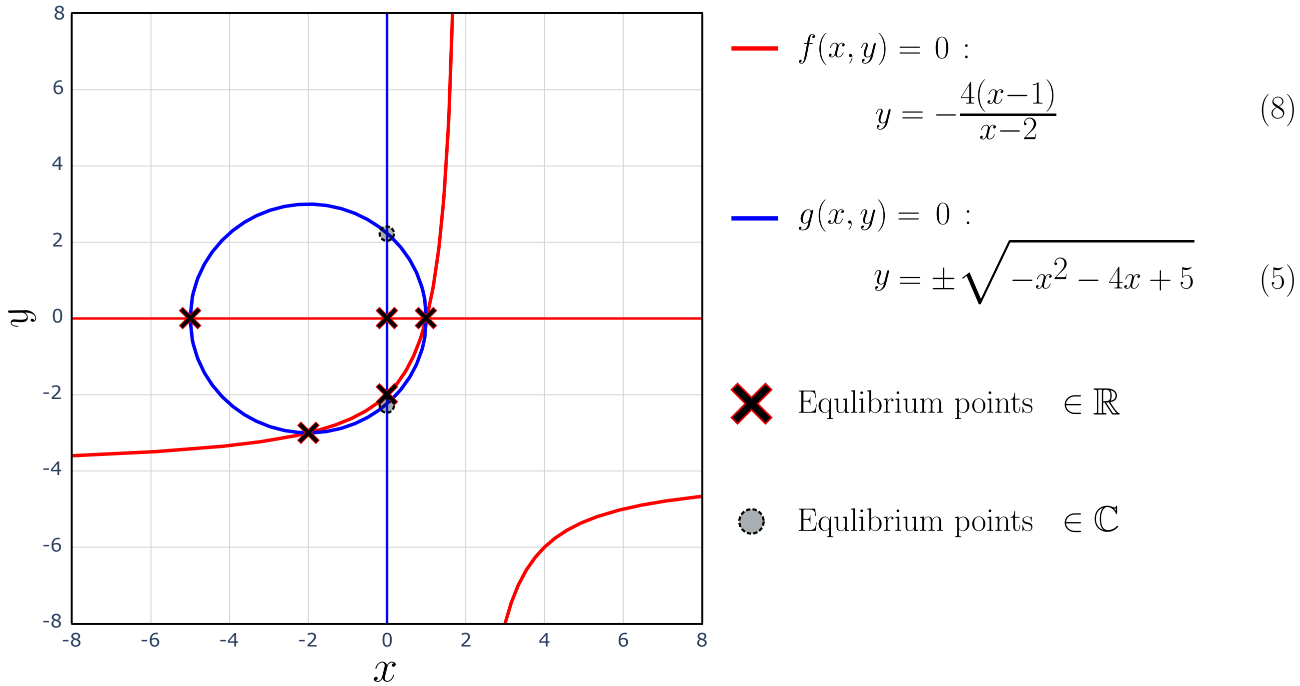


Figure 2: The nullclines and the equlibrium points of the nonlinear dynamical system.

3 Task - Linearization and type of equlibrium points

The linearization can be done with the help of Jacobian linearization using partial derivations in the following way:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{9}$$

Where the matrices of partial derivatives have to be evaluated at the determined equlibrium points.

$$\frac{\partial f}{\partial x} = -y^2 - 4y \tag{10}$$

$$\frac{\partial f}{\partial y} = -2xy - 4x + 4y + 4 \tag{11}$$

$$\frac{\partial g}{\partial x} = -3x^2 - 8x - y^2 + 5 \tag{12}$$

$$\frac{\partial g}{\partial y} = -2xy \tag{13}$$

Substituting the corresponding equlibrium point coordinate we get the numerical values that is showed in Table 2.

i	Equilibrium Points	Jacobian matrix
1	$(0,0)$	$\begin{bmatrix} 0 & 4 \\ 5 & 0 \end{bmatrix}$
2	$(-5,0)$	$\begin{bmatrix} 0 & 24 \\ -30 & 0 \end{bmatrix}$
3	$(1,0)$	$\begin{bmatrix} 0 & 0 \\ -6 & 0 \end{bmatrix}$
4	$(0,-2)$	$\begin{bmatrix} 4 & -4 \\ 1 & 0 \end{bmatrix}$
5	$(-2,-3)$	$\begin{bmatrix} 3 & -12 \\ 0 & -12 \end{bmatrix}$

Table 2: Equlibrium points with Jacobian linearization matrix

In order to be able to determine the type of the equlibrium points we need to calculate eigenvalues and eigenvectors for each Jacobian linearization matrix as

$$(\lambda \mathbf{I} - \mathbf{J}_i)\,\mathbf{A} = 0 \tag{14}$$

Where λ is the eigenvalue while $\mathbf{A}$ is the eigenvector. The solution is showed in Table 3, which was calculated using *Pthon*.

i	Equilibrium Points	Jacobian matrix	$\lambda_{1,i}$	$\lambda_{2,i}$	$\mathbf{A}_{1,i}$	$\mathbf{A}_{2,i}$
1	$(0,0)$	$\begin{bmatrix} 0 & 4 \\ 5 & 0 \end{bmatrix}$	$4.47+0i$	$-4.47+0i$	$\begin{bmatrix} 0.667+0i \\ -0.667+0i \end{bmatrix}$	$\begin{bmatrix} 0.745+0i \\ 0.745+0i \end{bmatrix}$
2	$(-5,0)$	$\begin{bmatrix} 0 & 24 \\ -30 & 0 \end{bmatrix}$	$0+26.83i$	$0-26.83i$	$\begin{bmatrix} 0-0.667i \\ 0+0.667i \end{bmatrix}$	$\begin{bmatrix} 0.745+0i \\ 0.745+0i \end{bmatrix}$
3	$(1,0)$	$\begin{bmatrix} 0 & 0 \\ -6 & 0 \end{bmatrix}$	0	0	$\begin{bmatrix} 0 \\ 1+0i \end{bmatrix}$	$\begin{bmatrix} 1+0i \\ 0 \end{bmatrix}$
4	$(0,-2)$	$\begin{bmatrix} 4 & -4 \\ 1 & 0 \end{bmatrix}$	$2+0i$	$2+0i$	$\begin{bmatrix} 0.894+0i \\ 0.447+0i \end{bmatrix}$	$\begin{bmatrix} 0.894+0i \\ 0.447+0i \end{bmatrix}$
5	$(-2,-3)$	$\begin{bmatrix} 3 & -12 \\ 0 & -12 \end{bmatrix}$	$3+0i$	$-12+0i$	$\begin{bmatrix} 1+0i \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.625+0i \\ 0.781+0i \end{bmatrix}$

Table 3: Equilibrium points with Jacobian linearization matrix, eigenvalues, and eigenvectors

The type of equlibrium point now can be determined based on the eigenvalues and visualized with the help of the eigenvectors. An alternative way to determine the type of each equlibrium point is to use the Poincaré diagram showed in Figure 3.

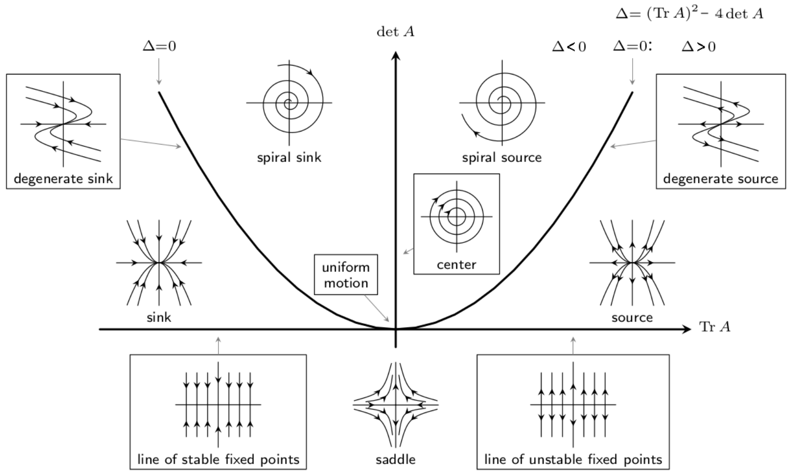


Figure 3: Poincaré diagram: classification of phase portraits [1]

The resulting types is summarized in Table 4 with the help of [2].

i	**Equilibrium Points**	**Type of stability**
1	$(0, 0)$	Saddle
2	$(-5, 0)$	Center
3	$(1, 0)$	Non-generic
4	$(0, -2)$	Unstable node
5	$(-2, -3)$	Saddle

Table 4: Equlibrium points

4 Task - Trace-Det plane

Utilizing the Poincaré diagram, which employs the trace-determinant plane to delineate equilibrium point types, our analysis yields consistent conclusions. Figure 4 presents a visual validation of this approach. The corresponding point of the second equlibrium point is not included on the plot, because of it's high determinant value (since it's trace is zero, it is a center).

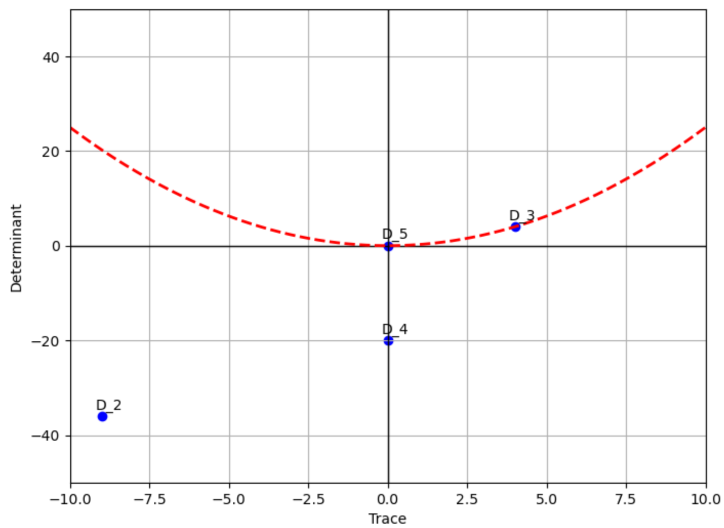


Figure 4: Trace-Det plane plot of the system

5 Task - Freehand drawing of the phase space

The freehand draw in Figure 5 were done before the actual visualization of the phase space based on Table 4 and 3.

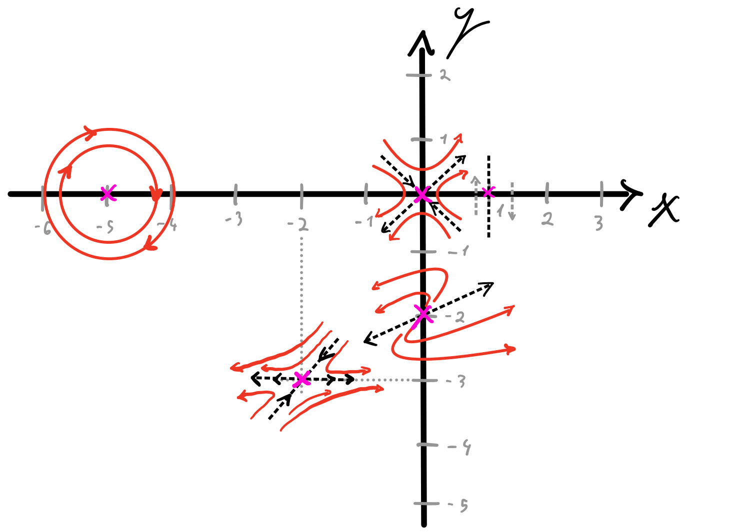


Figure 5: Freehand drawing of the phase space

6 Task - Computer aided drawing of the phase space

To visualize the phase space, I utilized the *Python* library *matplotlib.pyplot*. Initially, the phase space was depicted with actual magnitudes as shown in Figure 6. Subsequently, to enhance clarity regarding vector field directions, the vectors were normalized, resulting in Figure 7.

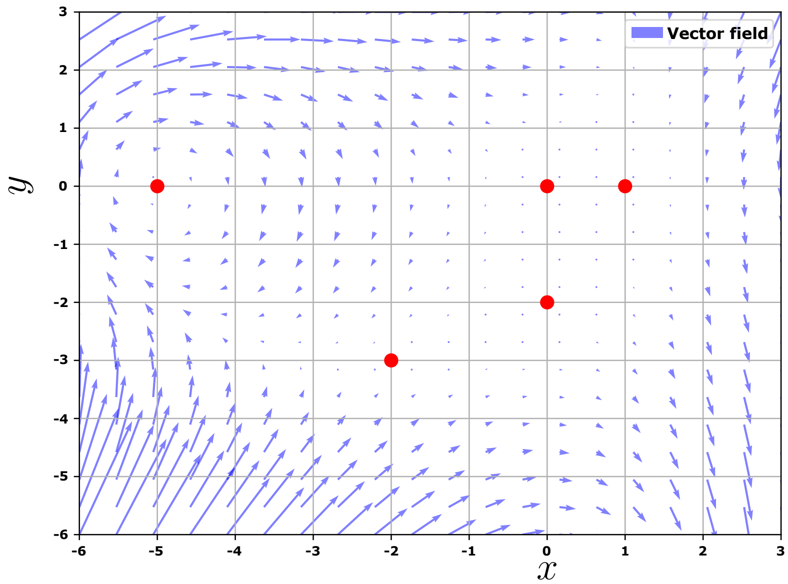


Figure 6: Computer aided phase space plot

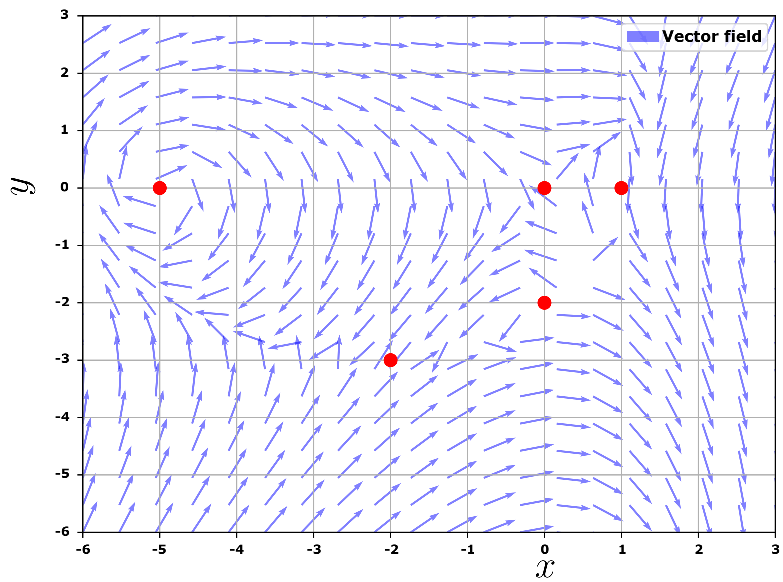


Figure 7: Phase space plot with normalized vectors for clarity

The computer-assisted visualization of the phase space serves as validation for the hand-drawn plot depicted in Figure 5.

References

[1] Egwald Mathematics - Linear Algebra: Systems of Linear Differential Equations: Linear Stability Analysis (link)

[2] MIT Mathlets - Linear phase portraits: Matrix Entry (link)

nonlin_hw1_done

April 9, 2024

[15]:
```
## Name:
#       Gergő Kelle
## NEPTUN code:
#       GBBNUL
```

[1]:
```
# Imported libraries
import sympy as sp
import numpy as np
from scipy.linalg import eig
import plotly.graph_objects as go
import matplotlib.pyplot as plt
```

[2]:
```
# Syms
x, y = sp.symbols('x y')

# Functions
f = -x*y**2 - 4*x*y + 2*y**2 + 4*y
g = -x**3 - x*y**2 - 4*x**2 + 5*x
equations = [f, g]
solutions = sp.solve(equations, (x, y))

print("Solutions to the system of equations:")
for solution in solutions:
    print(solution)
```

```
Solutions to the system of equations:
(-5, 0)
(-2, -3)
(0, -2)
(0, 0)
(1, 0)
(2*(-sqrt(7) - 3*I)/(sqrt(7) - 5*I), -5/2 - sqrt(7)*I/2)
(2*(-sqrt(7) + 3*I)/(sqrt(7) + 5*I), -5/2 + sqrt(7)*I/2)
```

[3]:
```
def calculate_values(solution, x_value, y_value):
    f_val = f.subs({x: x_value, y: y_value})
    g_val = g.subs({x: x_value, y: y_value})
    return f_val, g_val
```

```python
for i, solution in enumerate(solutions):
    f_val, g_val = calculate_values(solution, solutions[i][0], solutions[i][1])
    print("Solution", i + 1, ":")
    print("x =", solution[0], ", y =", solution[1])
    print("f(x, y) =", f_val)
    print("g(x, y) =", g_val)
    print()
```

```
Solution 1 :
x = -5 , y = 0
f(x, y) = 0
g(x, y) = 0

Solution 2 :
x = -2 , y = -3
f(x, y) = 0
g(x, y) = 0

Solution 3 :
x = 0 , y = -2
f(x, y) = 0
g(x, y) = 0

Solution 4 :
x = 0 , y = 0
f(x, y) = 0
g(x, y) = 0

Solution 5 :
x = 1 , y = 0
f(x, y) = 0
g(x, y) = 0

Solution 6 :
x = 2*(-sqrt(7) - 3*I)/(sqrt(7) - 5*I) , y = -5/2 - sqrt(7)*I/2
f(x, y) = -10 - 8*(-5/2 - sqrt(7)*I/2)*(-sqrt(7) - 3*I)/(sqrt(7) - 5*I) -
2*sqrt(7)*I - 2*(-5/2 - sqrt(7)*I/2)**2*(-sqrt(7) - 3*I)/(sqrt(7) - 5*I) +
2*(-5/2 - sqrt(7)*I/2)**2
g(x, y) = 10*(-sqrt(7) - 3*I)/(sqrt(7) - 5*I) - 8*(-sqrt(7) - 3*I)**3/(sqrt(7) -
5*I)**3 - 2*(-5/2 - sqrt(7)*I/2)**2*(-sqrt(7) - 3*I)/(sqrt(7) - 5*I) -
16*(-sqrt(7) - 3*I)**2/(sqrt(7) - 5*I)**2

Solution 7 :
x = 2*(-sqrt(7) + 3*I)/(sqrt(7) + 5*I) , y = -5/2 + sqrt(7)*I/2
f(x, y) = -10 + 2*(-5/2 + sqrt(7)*I/2)**2 - 2*(-5/2 + sqrt(7)*I/2)**2*(-sqrt(7)
+ 3*I)/(sqrt(7) + 5*I) + 2*sqrt(7)*I - 8*(-5/2 + sqrt(7)*I/2)*(-sqrt(7) +
3*I)/(sqrt(7) + 5*I)
```

```
g(x, y) = -16*(-sqrt(7) + 3*I)**2/(sqrt(7) + 5*I)**2 - 2*(-5/2 +
sqrt(7)*I/2)**2*(-sqrt(7) + 3*I)/(sqrt(7) + 5*I) - 8*(-sqrt(7) +
3*I)**3/(sqrt(7) + 5*I)**3 + 10*(-sqrt(7) + 3*I)/(sqrt(7) + 5*I)
```

[4]:

```python
# Remove the last two solutions from the list since they are complex numbers
solutions = solutions[:-2]

print("Remaining solutions:")
for i, solution in enumerate(solutions):
    print("Solution", i + 1, ":", solution)
```

```
Remaining solutions:
Solution 1 : (-5, 0)
Solution 2 : (-2, -3)
Solution 3 : (0, -2)
Solution 4 : (0, 0)
Solution 5 : (1, 0)
```

[5]:

```python
def calculate_values(solution, x_value, y_value):
    f_val = f.subs({x: x_value, y: y_value})
    g_val = g.subs({x: x_value, y: y_value})
    return f_val, g_val

for i, solution in enumerate(solutions):
    f_val, g_val = calculate_values(solution, solutions[i][0], solutions[i][1])
    print("Solution", i + 1, ":")
    print("x =", solution[0], ", y =", solution[1])
    print("f(x, y) =", f_val)
    print("g(x, y) =", g_val)
    print()
```

```
Solution 1 :
x = -5 , y = 0
f(x, y) = 0
g(x, y) = 0

Solution 2 :
x = -2 , y = -3
f(x, y) = 0
g(x, y) = 0

Solution 3 :
x = 0 , y = -2
f(x, y) = 0
g(x, y) = 0

Solution 4 :
x = 0 , y = 0
```

```
f(x, y) = 0
g(x, y) = 0

Solution 5 :
x = 1 , y = 0
f(x, y) = 0
g(x, y) = 0
```

[6]:

```python
# Linearization
f_x = sp.diff(f, x)
f_y = sp.diff(f, y)
g_x = sp.diff(g, x)
g_y = sp.diff(g, y)
D = [[f_x, f_y], [g_x, g_y]]

for row in D:
    print(row)
```

```
[-y**2 - 4*y, -2*x*y - 4*x + 4*y + 4]
[-3*x**2 - 8*x - y**2 + 5, -2*x*y]
```

[7]:

```python
Ds = []

# Substitute x and y from each solution into D
for solution in solutions:
    D_substituted = [[D_ij.subs({x: solution[0], y: solution[1]}) for D_ij in␣
 ↪D_i] for D_i in D]
    Ds.append(D_substituted)

for i, D_i in enumerate(Ds, start=1):
    print(f"D_{i}:")
    for row in D_i:
        print(row)
    print()

D_1 = Ds[0]
D_2 = Ds[1]
D_3 = Ds[2]
D_4 = Ds[3]
D_5 = Ds[4]
```

```
D_1:
[0, 24]
[-30, 0]

D_2:
[3, -12]
[0, -12]
```

```
D_3:
[4, -4]
[1, 0]

D_4:
[0, 4]
[5, 0]

D_5:
[0, 0]
[-6, 0]
```

[8]:
```python
# Identity matrix
I = np.eye(2)
eigenvalues_list = []
eigenvectors_list = []

# Calculate eigenvalues and eigenvectors for each D_i matrix
for D_i in Ds:
    # Convert D_i to a numpy array
    D_i_np = np.array(D_i, dtype=float)

    # Calculate:  *I - D_i
    diff_matrix = np.linalg.inv(I) @ D_i_np
    eigenvalues, eigenvectors = eig(diff_matrix)

    # Normalize eigenvectors
    normalized_eigenvectors = [eigenvector / np.linalg.norm(eigenvector) for␣
 ↪eigenvector in eigenvectors.T]

    # Creating the lists
    eigenvalues_list.append(eigenvalues)
    eigenvectors_list.append(normalized_eigenvectors)

print("Eigenvalues and Eigenvectors:")
print("   _1                _2                Eigenvector_1       Eigenvector_2")
for i, (eigenvalues, eigenvectors) in enumerate(zip(eigenvalues_list,␣
 ↪eigenvectors_list), start=1):
    print(f"D_{i}:")
    for j in range(len(eigenvalues)):
        print(f"{eigenvalues[j]: .3f}   {eigenvalues[j]: .3f}      ␣
 ↪{eigenvectors[j][0]: .3f},     {eigenvectors[j][1]: .3f}")
    print()
```

```
Eigenvalues and Eigenvectors:
   _1                _2                Eigenvector_1       Eigenvector_2
```

```
D_1:
 0.000+26.833j    0.000+26.833j        0.000-0.667j,      0.745+0.000j
 0.000-26.833j    0.000-26.833j        0.000+0.667j,      0.745-0.000j

D_2:
 3.000+0.000j    3.000+0.000j        1.000,      0.000
-12.000+0.000j   -12.000+0.000j        0.625,      0.781

D_3:
 2.000+0.000j    2.000+0.000j        0.894+0.000j,      0.447-0.000j
 2.000-0.000j    2.000-0.000j        0.894-0.000j,      0.447+0.000j

D_4:
 4.472+0.000j    4.472+0.000j        0.667,      0.745
-4.472+0.000j   -4.472+0.000j       -0.667,      0.745

D_5:
 0.000+0.000j    0.000+0.000j        0.000,      1.000
 0.000+0.000j    0.000+0.000j        0.000,      1.000
```

[9]:
```python
equilibrium_types = []

# Define a function to determine equilibrium point type
def determine_equilibrium_type(eigenvalues):
    if np.allclose(eigenvalues.real, 0) and np.allclose(eigenvalues.imag, 0):  # Both eigenvalues are zero
        return "-"
    elif np.all(eigenvalues.real < 0):
        return "Stable Node"
    elif np.all(eigenvalues.real > 0):
        return "Unstable Node"
    elif np.any(eigenvalues.real < 0) and np.any(eigenvalues.real > 0):
        return "Saddle"
    elif np.any(eigenvalues.imag < 0) and np.any(eigenvalues.imag > 0):
        return "Center"
    else:
        return "Focus"

# Determine the equilibrium point types for each D_i matrix
for i, (eigenvalues, _) in enumerate(zip(eigenvalues_list, eigenvectors_list), start=1):
    equilibrium_type = determine_equilibrium_type(eigenvalues)
    print(f"Equilibrium Point Type for D_{i}: {equilibrium_type}")
```

```
Equilibrium Point Type for D_1: Center
Equilibrium Point Type for D_2: Saddle
Equilibrium Point Type for D_3: Unstable Node
```

```
Equilibrium Point Type for D_4: Saddle
Equilibrium Point Type for D_5: -
```

[10]:
```
# Functions
def f(x):
    return -((4 * (x-1))/(x-2))

def g_positive(x):
    return np.sqrt(-(x**2) - 4*x + 5)

def g_negative(x):
    return -np.sqrt(-(x**2) - 4*x + 5)

x1 = np.linspace(-8, 1.9, 400)
x2 = np.linspace(2.1, 8, 400)
y1 = f(x1)
y2 = f(x2)
y2_positive = g_positive(x1)
y2_negative = g_negative(x1)

# Create traces for function 1
trace1 = go.Scatter(x=x1, y=y1, mode='lines', name='f(x)=-((4(x-1))/(x-2))',␣
 ↪line=dict(color='red'))
trace12 = go.Scatter(x=x2, y=y2, mode='lines', name='f(x)=-((4(x-1))/(x-2))',␣
 ↪line=dict(color='red'))

# Create traces for function 2
trace2_positive = go.Scatter(x=x1, y=y2_positive, mode='lines',␣
 ↪name='g(x)=(-(x^2)-4x+5)^(1/2)', line=dict(color='blue'))
trace2_negative = go.Scatter(x=x1, y=y2_negative, mode='lines',␣
 ↪name='g(x)=(-(x^2)-4x+5)^(-1/2)', line=dict(color='blue'))

# Combine functions
data = [trace1, trace12, trace2_positive, trace2_negative]

layout = go.Layout(
    #title='Visualization of Functions',
    xaxis=dict(title='x', linecolor='black', mirror=True),
    yaxis=dict(title='y', linecolor='black', mirror=True),
    xaxis_range=[-8, 8],
    yaxis_range=[-8, 8],
    showlegend=False,
    plot_bgcolor='white',
    paper_bgcolor='white',
    xaxis_gridcolor='lightgrey',
    yaxis_gridcolor='lightgrey',
    width=800,  # Set width to 1600 pixels for 16:9 aspect ratio
```

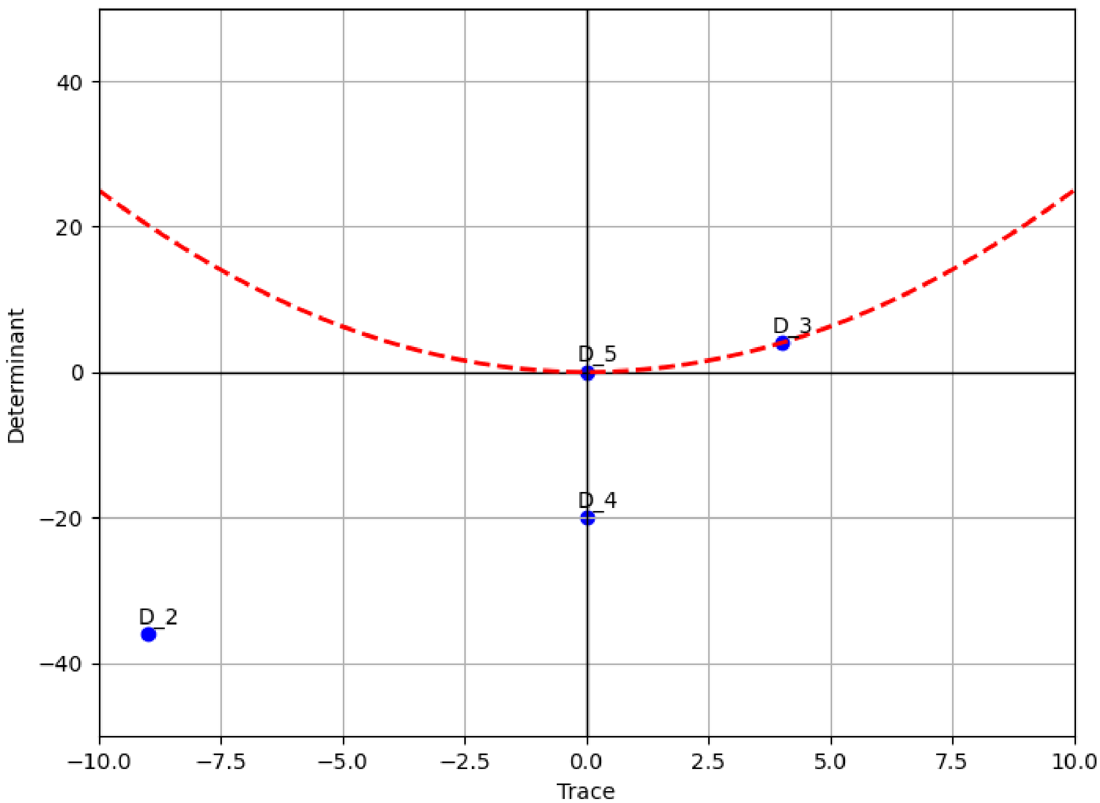
```
    height=800  # Set height to 900 pixels for 16:9 aspect ratio)
)

fig = go.Figure(data=data, layout=layout)
fig.write_image("functions_visualization_plotly.pdf")
fig.show()
```

```
C:\Users\Kelle Gergő\AppData\Local\Temp\ipykernel_28884\2602317077.py:6:
RuntimeWarning: invalid value encountered in sqrt
  return np.sqrt(-(x**2) - 4*x + 5)
C:\Users\Kelle Gergő\AppData\Local\Temp\ipykernel_28884\2602317077.py:9:
RuntimeWarning: invalid value encountered in sqrt
  return -np.sqrt(-(x**2) - 4*x + 5)
```

[11]:

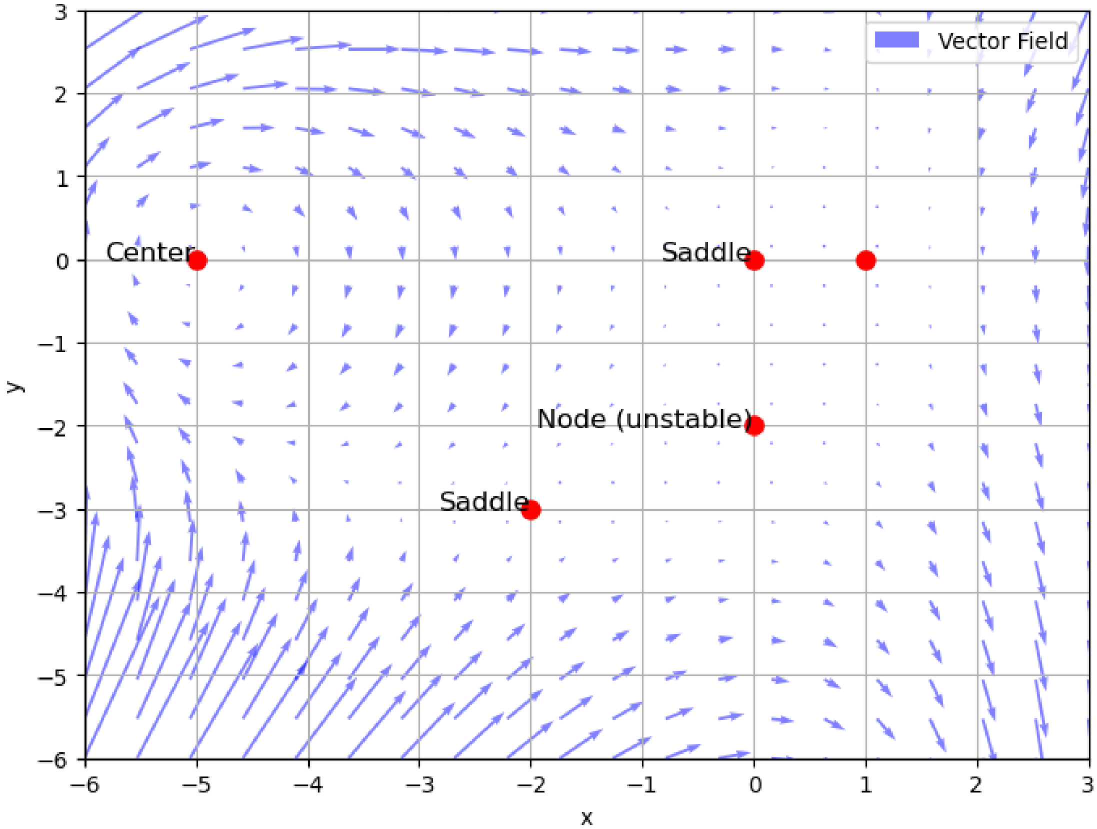
```
Ds2 = [[[0, 24], [-30, 0]],
      [[3, -12], [0, -12]],
      [[4, -4], [1, 0]],
      [[0, 4], [5, 0]],
      [[0, 0], [-6, 0]]]

trace_values = []
det_values = []

for D in Ds2:
    D = np.array(D)  # Convert list of lists to numpy array
    trace = np.trace(D)
    det = np.linalg.det(D)
    trace_values.append(trace)
    det_values.append(det)

# Plot the Trace-Det plane
plt.figure(figsize=(8, 6))

# Scatter plot for trace and determinant
plt.scatter(trace_values, det_values, color='blue')

# Annotate each point with its corresponding index
for i, (trace, det) in enumerate(zip(trace_values, det_values)):
    plt.annotate(f'D_{i+1}', (trace, det), textcoords="offset points",␣
 ↪xytext=(5,5), ha='center')


plt.xlabel('Trace')
plt.ylabel('Determinant')
plt.axhline(0, color='black', linewidth=1)  # x-axis at y=0
plt.axvline(0, color='black', linewidth=1)  # y-axis at x=0
```

```
# Define the function x^2 - 4*y
x = np.linspace(-10, 10, 400)
y = (x ** 2) / 4
plt.plot(x, y, color='red', linestyle='--', linewidth=2)

# Set x and y axis limits
plt.xlim(-10, 10)
plt.ylim(-50, 50)
plt.grid(True)
plt.show()
```



[12]:
```
def vector_field(x, y):
    dx = -x*y**2 - 4*x*y + 2*y**2 + 4*y
    dy = -x**3 - x*y**2 - 4*x**2 + 5*x
    return dx, dy

# Range
x_range = np.linspace(-6, 3, 20)
y_range = np.linspace(-6, 3, 20)
X, Y = np.meshgrid(x_range, y_range)
```

```python
DX, DY = vector_field(X, Y)

plt.figure(figsize=(8, 6))
plt.quiver(X, Y, DX, DY, color='b', alpha=0.5, label='Vector Field')

# Eq points
equilibrium_points = [(-5, 0), (-2, -3), (0, -2), (0, 0), (1, 0)]
for eq_point in equilibrium_points:
    plt.plot(eq_point[0], eq_point[1], 'ro', markersize=8)

# Type
stability = ['Center', 'Saddle', 'Node (unstable)', 'Saddle', '']
for i, eq_point in enumerate(equilibrium_points):
    plt.text(eq_point[0], eq_point[1], f'{stability[i]}', fontsize=12,
 ↪ha='right')

plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-6, 3)
plt.ylim(-6, 3)
plt.legend()
plt.grid()
plt.savefig('phase_space_plot.pdf')
plt.show()
```



[13]:
```python
def vector_field(x, y):
    dx = -x*y**2 - 4*x*y + 2*y**2 + 4*y
    dy = -x**3 - x*y**2 - 4*x**2 + 5*x
    magnitude = np.sqrt(dx**2 + dy**2)  # Magnitude
    dx /= magnitude  # Normalize
    dy /= magnitude
    return dx, dy

# Range
x_range = np.linspace(-6, 3, 20)
y_range = np.linspace(-6, 3, 20)
X, Y = np.meshgrid(x_range, y_range)

DX, DY = vector_field(X, Y)

plt.figure(figsize=(8, 6))
plt.quiver(X, Y, DX, DY, color='b', alpha=0.5, scale=20, label='Vector Field') ␣
 ↪# Adjust scale parameter as needed

# Eq points
```

```
equilibrium_points = [(-5, 0), (-2, -3), (0, -2), (0, 0), (1, 0)]
for eq_point in equilibrium_points:
    plt.plot(eq_point[0], eq_point[1], 'ro', markersize=8)

# Type
stability = ['Center', 'Saddle', 'Node (unstable)', 'Saddle', '']
for i, eq_point in enumerate(equilibrium_points):
    plt.text(eq_point[0], eq_point[1], f'{stability[i]}', fontsize=12,␣
 ↪ha='right')


plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-6, 3)
plt.ylim(-6, 3)
plt.legend()
plt.grid()
plt.savefig('phase_space_plot_normalized.pdf')
plt.show()
```