НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

Звіт
із лабораторної роботи №1
з дисципліни «Нелінійний Аналіз»

Виконав:
студенти групи КМ-51
Мужилівський С.В.

Перевірив:
Сірик С.В.

Київ — 2018

ЗМІСТ

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Нехай є нелінійна автономна система диференціальних рівнянь:

$$\begin{cases} \dot{x} = P(x; y), \\ \dot{y} = Q(x; y), \end{cases} \qquad (1.1)$$

де x = x(t) , y = y(t) — траєкторії, t — змінна часу. Для побудови фазових кривих (траєкторій) даної системи можна скористатись наступними ітераційними наближеними формулами:

$$\begin{cases} x_k = x_{k-1} + \alpha_x \cdot \widetilde{V}_x^{(k)}, \\ y_k = y_{k-1} + \alpha_y \cdot \widetilde{V}_y^{(k)}, \end{cases} \qquad (1.2)$$

де $k \in 1, \dot{\ldots}, N$ — номер ітерації, N — кількість ітерацій, (x_0; y_0) — стартова точка (яка задається користувачем на початку алгоритму (1.2)), $\widetilde{V}^{(k)} = (\widetilde{V}_x^{(k)}; \widetilde{V}_y^{(k)})$— нормований вектор швидкості потоку в точці (x_{k-1}; y_{k-1}) (тобто, $\widetilde{V}_x^{(k)} = \frac{V_x^{(k)}}{\sqrt{V_x^{(k)} + V_y^{(k)}}}$, $\widetilde{V}_y^{(k)} = \frac{V_y^{(k)}}{\sqrt{V_x^{(k)} + V_y^{(k)}}}$, де $V_x^{(k)} = P(x_{k-1}; y_{k-1})$, $V_y^{(k)} = Q(x_{k-1}; y_{k-1})$), α_x та α_y — параметри кроків за напрямами осей x та y відповідно. Наведений алгоритм реалізує ідею плину в часі по траєкторії під дією поля швидкостей (точка з поточного "(k - 1)-го положення" пересувається в наступне "k-те положення" згідно з напрямом, який задається вектором швидкості у поточній точці). Потім, для отримання наближеної траєкторії системи, що виходить з точки (x_0; y_0), точки між сусідніми ітераціями (тобто, точки (x_{k-1}; y_{k-1}) та (x_k; y_k)) слід з'єднати відрізком прямої. При цьому, величина кроку задається (α_x; α_y) (чим більшими є ці величини, тим більшим буде крок, і тим меншою буде точність побудованої траєкторії).

2 АНАЛІТИЧНИЙ РОЗВ’ЯЗОК

Нехай дано систему(14 – варіант):

$$\begin{cases} \dot{x} = xy - 4 \\ \dot{y} = (y - x)\cdot(x - 4) \end{cases}$$

На рис. 2.1 показано пошук точок положення рівноваги та лінеаризація системи:

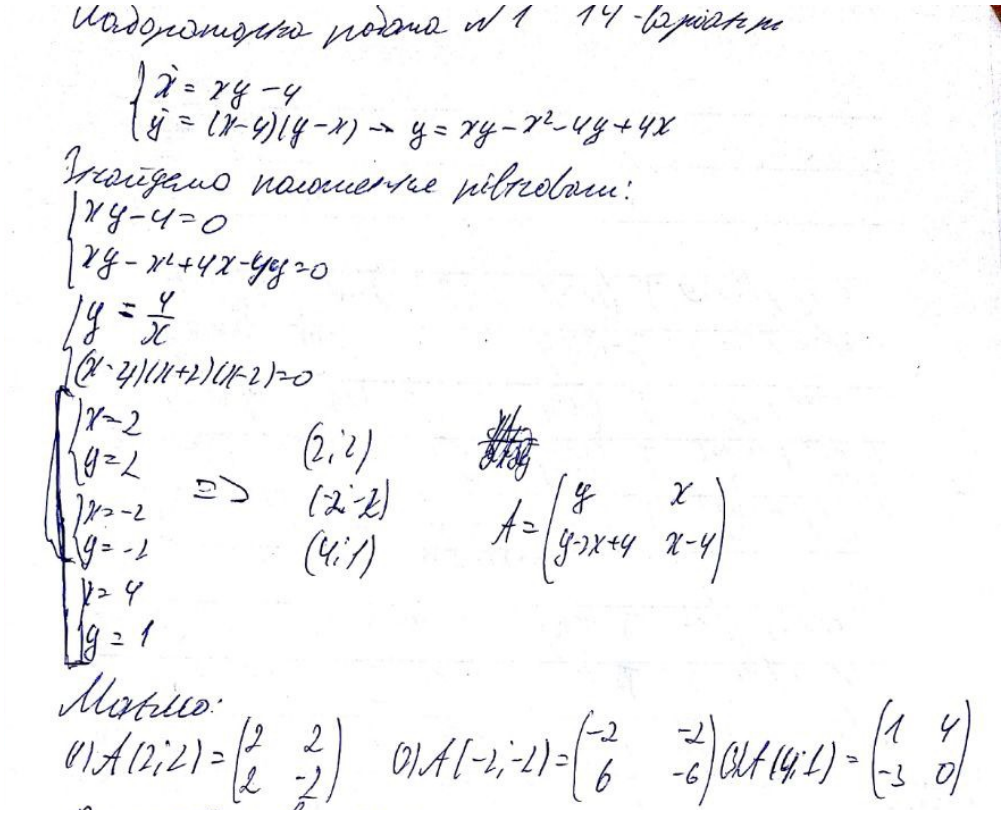

Рис. 2.1 – Точки рівноваги

Наступні рисунки демонструюють дослідження характеру фазового портрету у точках спокою:

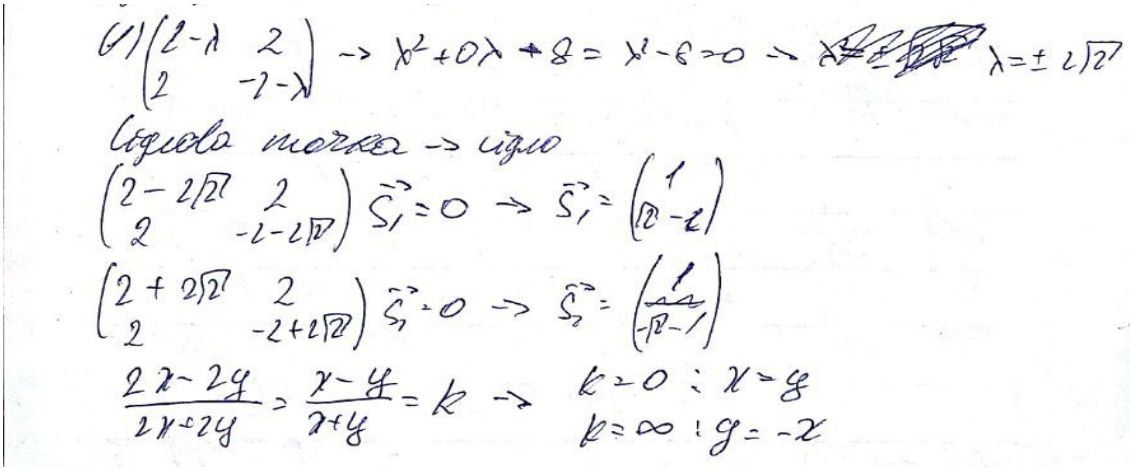

Рис. 2.2 – Характер фазового портрету в околі точки спокою (1)

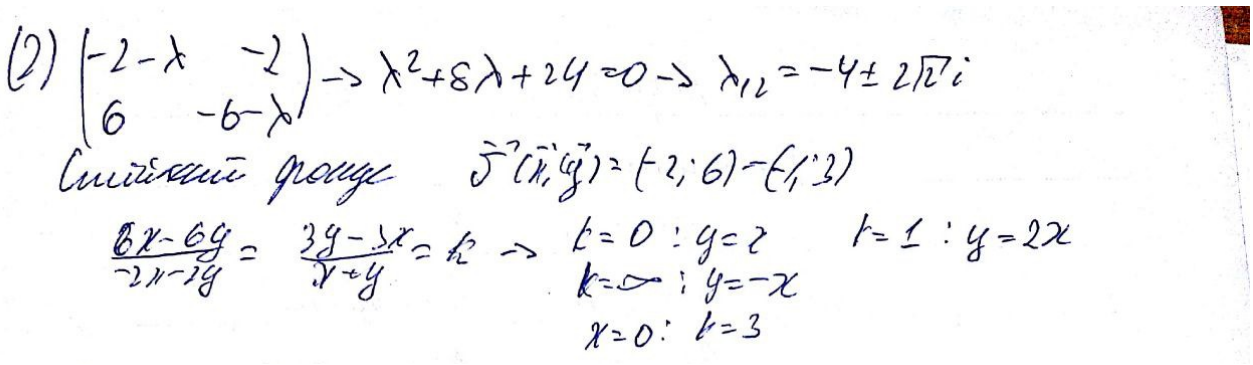

Рис. 2.3 – Характер фазового портрету в околі точки спокою (2)

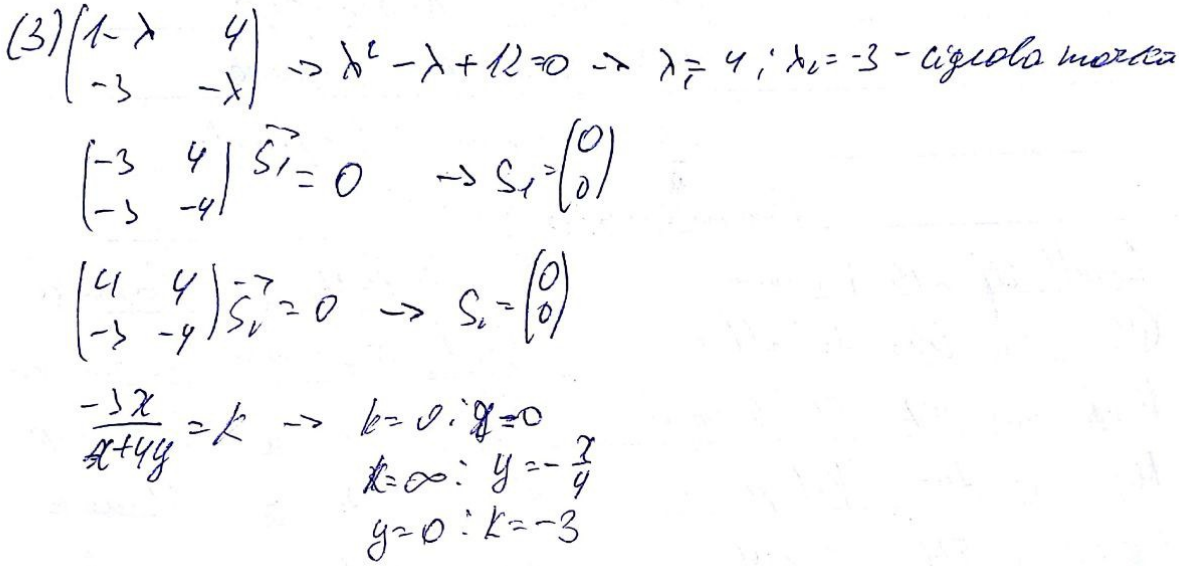

Рис. 2.4 - Характер фазового портрету в околі точки спокою (3)

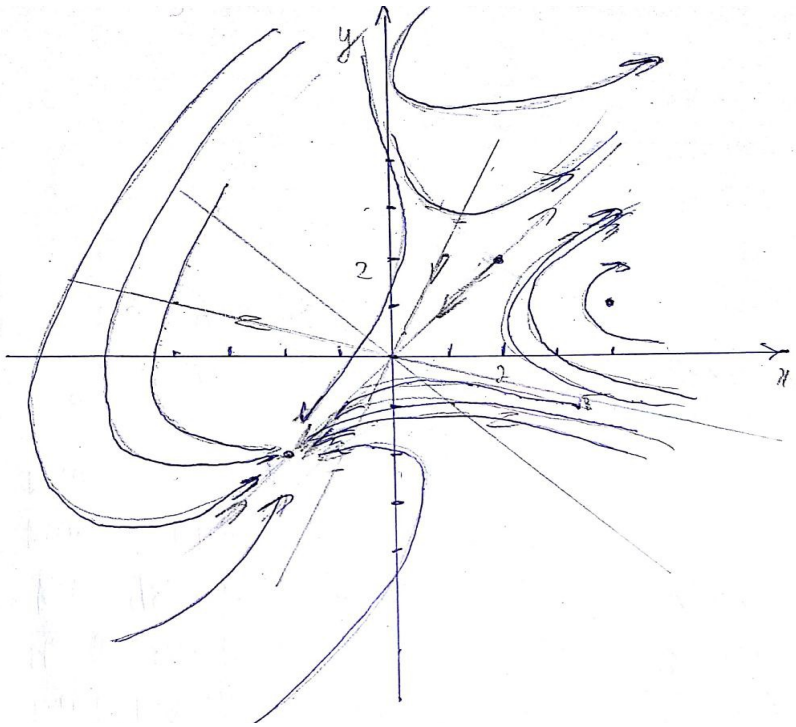

Рис. 2.5 — Фазовий портрет

3 КОМП’ЮТЕРНЕ МОДЕЛЮВАННЯ

У рамках даної лабораторної роботи, було розроблено програму для побудови фазовго портрету мовою python. На рис.3.1 зображено фазовий портрет у випадку, коли N = 100000, a_x=0.001, a_y= 0.001.

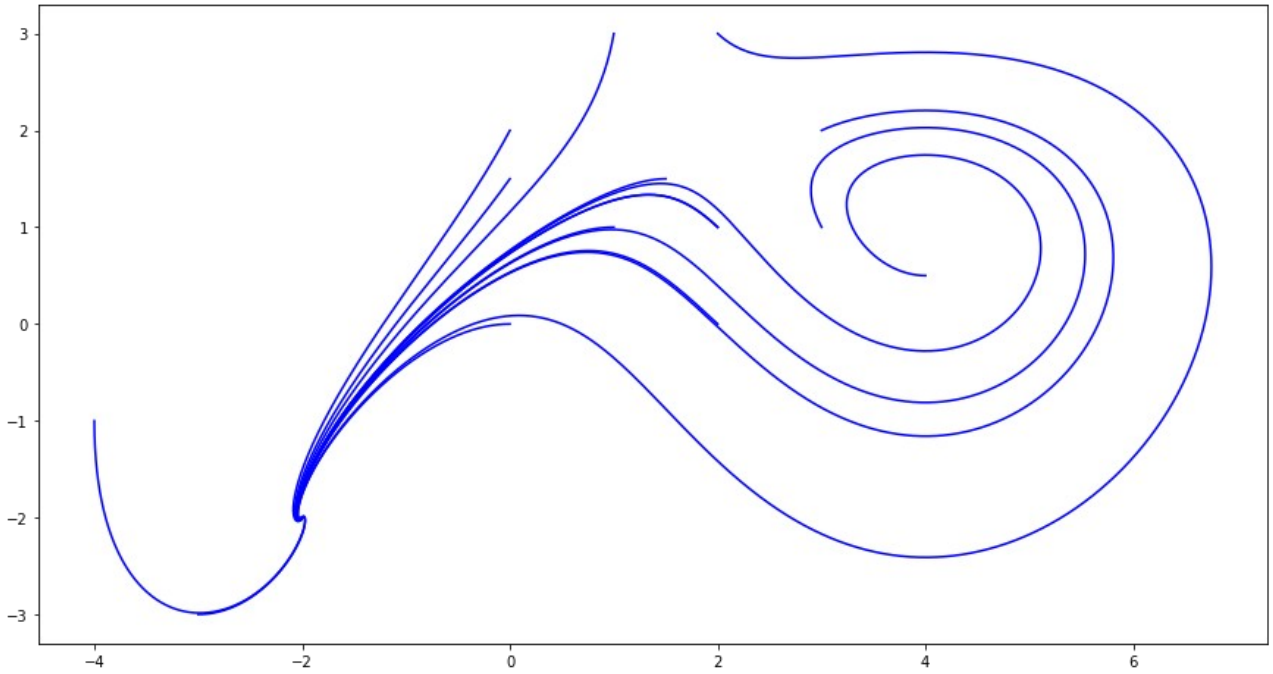

Рис. 3.1 – Фазовий портрет при сталих параметрах кроку

Зменшимо значення a_x та a_y. Очевидно, довжина кроку зменшиться, що і бачимо на рис.3.2:

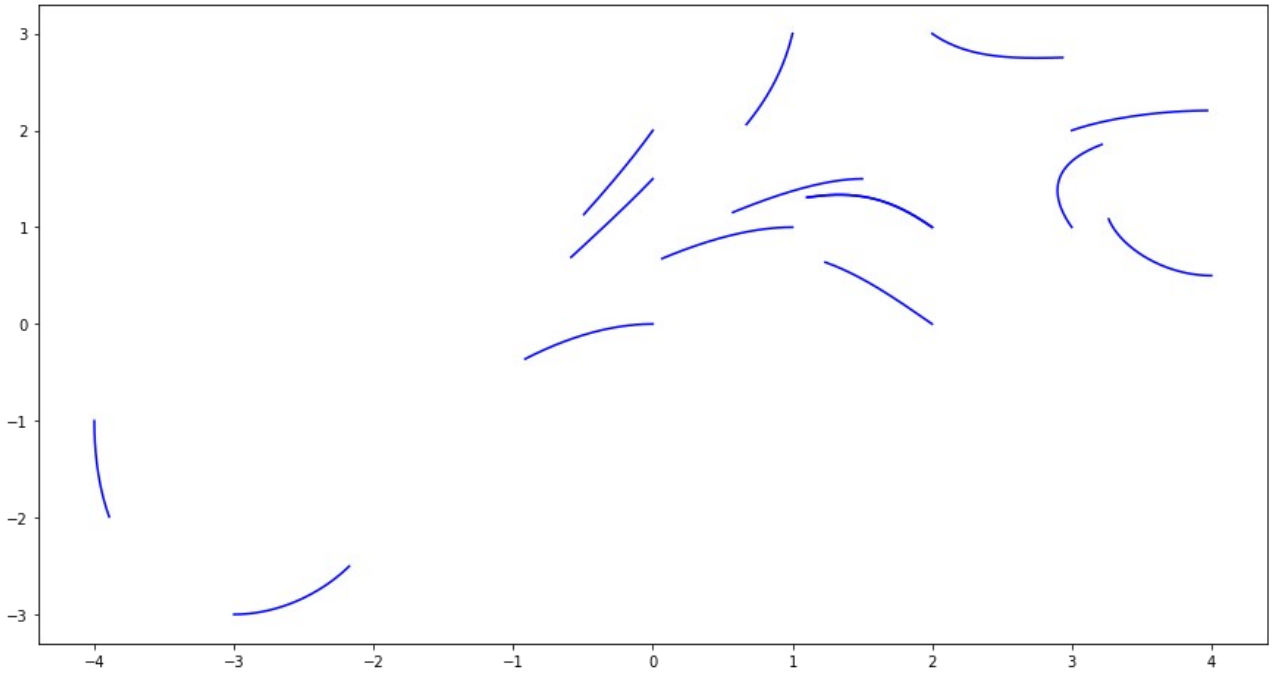

Рис. 3.2 – Фазовий портрет при зменшених параметрах кроку

Змінимо a_x та a_y, взявши їх за експоненційним законом.

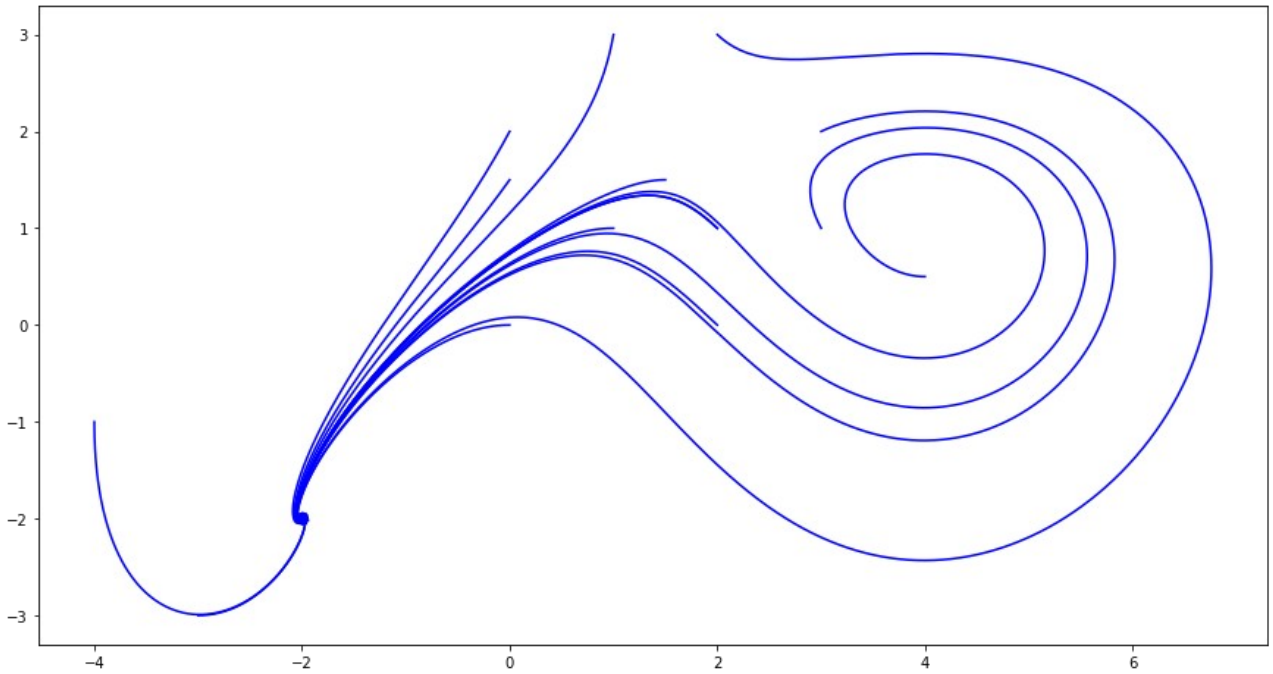

Рис. 3.3 – Фазовий портрет при “експоненційних” параметрах кроку

Змінимо a_x та a_y, взявши їх за нормальним законом розподілу.

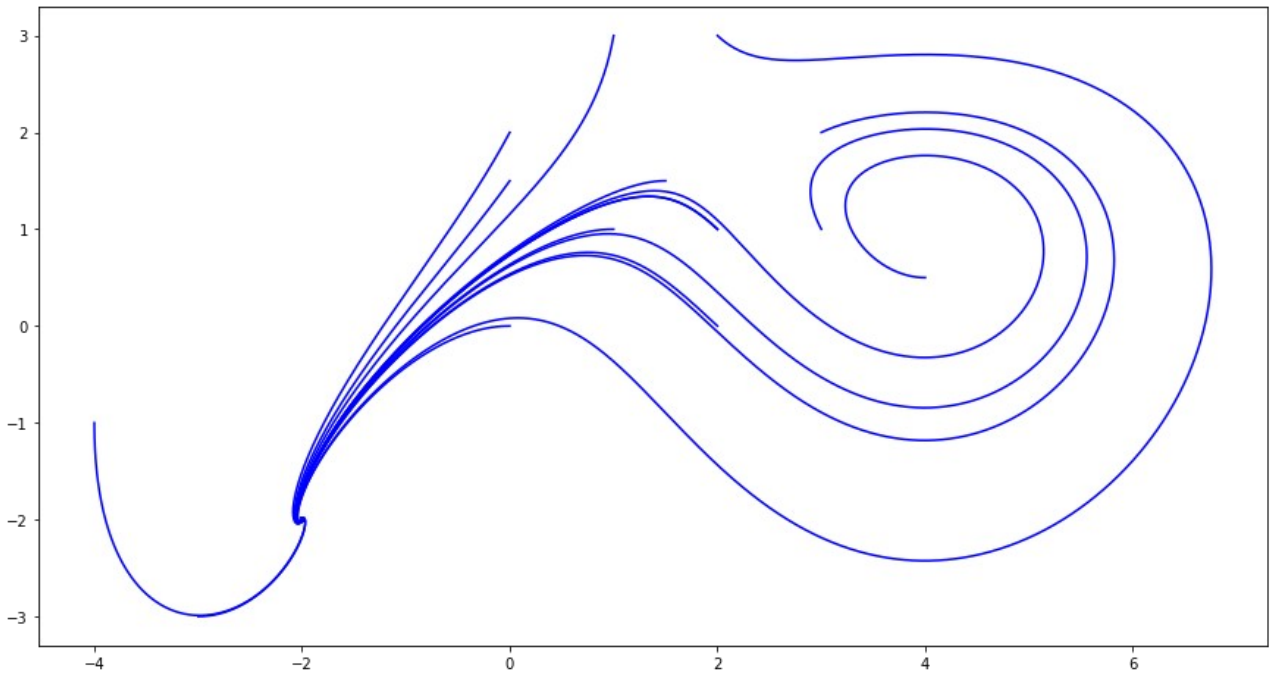

Рис. 3.4 – Фазовий портрет при “нормальних” параметрах кроку

ВИСНОВКИ

Під час виконання лабораторної роботи було проведене дослідження нелінійної системи, знайдено точки спокою, а саме (-2;-2), (2;2) та (4;1). В околах даних точок дізналися про особливості фазового портрету та на основі цієї інформації було побудовано портрет.

Було розроблено програму для побудови траєкторій системи, за допомогою якої побудовано чотири фазові портрети для випадків сталих значень параметрів кроку порядку 10^{-3}, сталих значень параметрів кроку порядку 10^{-5}, значень параметра кроку, які змінюються за «експоненціальним» законом та значень параметра кроку, які змінюються за «нормальним» законом. Фазові портрети у першому випадку та у випадках змінного параметра кроку майже не відрізняються, оскільки приріст параметру незначний.

Додаток А. Лістинг програми

```
import numpy as np
import random
def P(x):
    return (x[0] * x[1]) - 4
def P(x):
    return (x[0] * x[1]) - 4
def Q(x):
    return (x[0] - 4) * (x[1] - x[0])
def wVxk(vxk, vyk):
    return vxk / np.sqrt(vxk ** 2 + vyk ** 2)
def wVyk(vxk, vyk):
    return vyk / np.sqrt(vxk ** 2 + vyk ** 2)
def Vxk(x_1):
    return P(x_1)
def Vyk(x_1):
    return Q(x_1)
def wVk(wvxk, wvyk):
    return np.array([wvxk, wvyk])
def build(init_point: tuple, alpha_x: float, alpha_y: float, N: int):
    alpha = np.array([alpha_x, alpha_y])
    sequence = [np.array(init_point)]
    for i in range(N):
        p = sequence[-1]
        vxk, vyk = Vxk(p), Vyk(p)
        wvxk, wvyk = wVxk(vxk, vyk), wVyk(vxk, vyk)
        sequence.append(p + alpha * wVk(wvxk, wvyk))
    sequence = np.array(sequence).T
    return sequence[0], sequence[1]
def build_exp(init_point: tuple, N: int):
    sequence = [np.array(init_point)]
    for i in range(N):
        p = sequence[-1]
        alpha = random.expovariate(150)
        vxk, vyk = Vxk(p), Vyk(p)
        wvxk, wvyk = wVxk(vxk, vyk), wVyk(vxk, vyk)
        sequence.append(p + alpha * wVk(wvxk, wvyk))
    sequence = np.array(sequence).T
    return sequence[0], sequence[1]
def build_norm(init_point: tuple, N: int):
    sequence = [np.array(init_point)]
    for i in range(N):
        p = sequence[-1]
        alpha = random.normalvariate(0.01, 0.001)
        vxk, vyk = Vxk(p), Vyk(p)
        wvxk, wvyk = wVxk(vxk, vyk), wVyk(vxk, vyk)
        sequence.append(p + alpha * wVk(wvxk, wvyk))
    sequence = np.array(sequence).T
    return sequence[0], sequence[1]
fig, ax = plt.subplots(1, 1, figsize=(15, 8))
dots = [(2, 2), (-2, -2), (4, 1), (0, 0), (1, 1), (3, 1), (-4, -1), (-3, -3), (2, 1), (0, 2), (0, 1.5), (4, 0.5),
      (2, 0), (2, 1), (3, 2), (1.5, 1.5), (2, 3), (1, 3)]
for dot in dots:
    x, y = build(init_point=dot, alpha_x=1e-3, alpha_y=1e-3, N=100000)
    ax.plot(x, y, c="b")
fig, ax = plt.subplots(1, 1, figsize=(15, 8))
dots = [(2, 2), (-2, -2), (4, 1), (0, 0), (1, 1), (3, 1), (-4, -1), (-3, -3), (2, 1), (0, 2), (0, 1.5), (4, 0.5),
      (2, 0), (2, 1), (3, 2), (1.5, 1.5), (2, 3), (1, 3)]
for dot in dots:
```

```
    x, y = build(init_point=dot, alpha_x=1e-5, alpha_y=1e-5, N=100000)
    ax.plot(x, y, c="b")
fig, ax = plt.subplots(1, 1, figsize=(15, 8))
dots = [(2, 2), (-2, -2), (4, 1), (0, 0), (1, 1), (3, 1), (-4, -1), (-3, -3), (2, 1), (0, 2), (0, 1.5), (4, 0.5),
        (2, 0), (2, 1), (3, 2), (1.5, 1.5), (2, 3), (1, 3)]
for dot in dots:
    x, y = build_exp(init_point=dot, N=10000)
    ax.plot(x, y, c="b")
fig, ax = plt.subplots(1, 1, figsize=(15, 8))
dots = [(2, 2), (-2, -2), (4, 1), (0, 0), (1, 1), (3, 1), (-4, -1), (-3, -3), (2, 1), (0, 2), (0, 1.5), (4, 0.5),
        (2, 0), (2, 1), (3, 2), (1.5, 1.5), (2, 3), (1, 3)]
for dot in dots:
    x, y = build_norm(init_point=dot, N=10000)
    ax.plot(x, y, c="b")
```