Харківський національний університет імені В.Н. Каразіна
Факультет математики і інформатики
Кафедра прикладної математики

**Кваліфікаційна робота**

**бакалавра**

на тему «Лінеаризація керованих систем і задачі керованості»

Виконав: студент групи МП-41 IV курсу,
спеціальності 113 Прикладна математика
освітньо-професійна програма
Прикладна математика
Гальченко П.С.
Науковий керівник:
д.ф.-м.н., професор кафедри прикладної математики Ігнатович С.Ю
Рецензент:
Кандидат фіз.-мат. наук, доцент, в.о. зав.каф. фундаментальної математики
Гефтер С.Л.

Харків - 2024 рік

**АНОТАЦІЯ**

**Гальченко П.С., Лінеаризація керованих систем і задачі керованості.**

Метою даної роботи є вивчення методів лінеаризації нелінійних керованих систем та розв'язання задач керованості. Спочатку, розглядаються матриці Якобі як інструмент для аналізу поведінки системи у фазовому просторі.

Далі розглядаються трикутні та майже трикутні керовані системи і метод знаходження керувань u(t), які переводять систему з однієї точки в іншу за скінчений вибраний час. Особлива увага приділяється вибору різних видів функцій керування.

**ANNOTATION**

**Halchenko P.S., Linearization of control systems and controllability problems.**

The aim of this work is to study methods for linearizing nonlinear control systems and solving controllability problems. Initially, Jacobian matrices are considered as a tool for analyzing system behavior in state space.

Then, triangular and almost triangular control systems are examined, along with the method of finding controls u(t) that transfer the system from one point to another in a finite chosen time. Special attention is given to the choice of different types of control functions.

# ЗМІСТ

# 1.ВСТУП

На сучасному етапі розвитку науки та технологій важливість досліджень у галузі лінеаризації нелінійних систем є надзвичайно актуальною та необхідною. Це пов'язано зі зростанням складності систем у різних галузях, таких як авіація, космос, автомобільна промисловість, робототехніка, фінансові ринки та інші. Нелінійність у таких системах створює значні труднощі при їх аналізі та керуванні.

Для лінійних систем диференційних рівнянь і лінійних керованих систем часто можна отримати розв’язок у явному аналітичному вигляді та дослідити якісні властивості систем. Для нелінійних систем такий аналіз дуже рідко можна провести явно, і зазвичай потрібно використовувати чисельні методи. Тому, якщо можна будь-яким чином звести нелінійну систему до лінійної, то це суттєво спрощує розв’язання задачі.

У цій роботі буде розглянуто два різні підходи до ідеї лінеаризації нелінійних систем. По-перше, можна замінити праву частину системи її лінійним наближенням за допомогою розкладання у ряд Тейлора, та дослідити отриману лінійну систему в околі точки спокою. По-друге, можна спробувати знайти такі координати, у яких задана система стане лінійною. Зрозуміло, що це не завжди можливо, але якщо це вдається, то це суттєво спрощує розв’язання задачі. Ця ідея особливо корисна ще й для керованих систем, тому що окрім переходу до нових змінних, можна ще й замінити керування.

Зауважимо, що після отримання лінійної системи постає задача пошуку функції керування. Пошук функцій керування включає вибір найбільш вдалого виду функції u(t). Лінійні функції, квадратичні, кубічні, експоненціальні та тригонометричні функції мають свої переваги та обмеження, що залежать від конкретних вимог до системи та її характеристик. Наприклад, лінійні функції є простими для реалізації, але можуть бути недостатніми для складних систем. У таких випадках використовуються функції вищих порядків, які здатні точніше моделювати динаміку системи.

## 2.ЛІНЕАРИЗАЦІЯ СИСТЕМ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ ЗА ДОПОМОГОЮ МАТРИЦЬ ЯКОБІ

Розглянемо нелінійну автономну систему:

$$\frac{dx_i}{dt} = f_i(x_1, x_2, \dots, x_n), \qquad i = 1, \dots, n.$$

Нехай $x^*$- нерухома точка нелінійної системи $\dot{x} = f(x)$. Ми дослідимо поведінку розв'язку в околі $x^*$. Введемо вектор відхилень розв'язку від $x^*$: $\Delta x = x - x^*$. Елементи цього вектору називаються локальними координатами в точці $x^*$.

Початкова нелінійна система рівнянь в околі точки $x^*$ апроксимується за допомогою розкладу функції правої частини системи рівнянь у ряд Тейлора [4]. Зберігаємо члени ряду до першого порядку включно:

$$\frac{d(x_i + \Delta x_i)}{dt} = f_i(x + \Delta x), \qquad i = 1,2, \dots, n$$

$$\frac{dx_i}{dt} + \frac{d(\Delta x)}{dt} \approx f_i(x) + \sum_{j=1}^{n} \frac{\partial f_i(x)}{\partial x_i} \Delta x_j$$

Для стаціонарного розв'язку маємо:

$$\left.\frac{dx_i}{dt}\right|_{x=x^*} = 0\,, \qquad f_i(x^*) = 0$$

З розкладу отримуємо:

$$\frac{d(\Delta x_i)}{dt} = \sum_{j=1}^{n} \frac{\partial f_i(x^*)}{\partial x_j} \Delta x_j, \qquad i = 1,2, \dots, n$$

або у матричному записі:

$$\frac{d\Delta x}{dt} = J\Delta x,$$

де $J = \left[\frac{\partial f_i}{\partial x_j}\right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}, i = 1, \dots, n\,, j = 1, \dots, n.$

Тут J – матриця Якобі, при цьому значення всіх похідних обчислені в точці $x^*$. Отримана система рівнянь є лінійною системою рівнянь зі сталими

коефіцієнтами і називається лінеаризацією нелінійної системи $\dot{x} = f(x)$ в точці $x^*$.

Розглянемо нелінійну систему диференціальних рівнянь з керуванням:

$$\dot{x} = f(x, u)$$

Для аналізу системи навколо точки спокої обираємо деяку точку $(x_0, u_0)$, в якій $f(x_0, u_0) = 0$. Це дозволяє нам сконцентруватися на околі цієї точки та використовувати лінійну апроксимацію.

У даному випадку матриці Якобі визначаються, як частинні похідні функції $f(x, u)$ за змінними стану $x$ та керування $u$ в точці $(x_0, u_0)$:

- Матриця Якобі за змінними стану $x$:

$$A = \frac{\partial f}{\partial x}\Big|_{x_0, u_0}$$

- Матриця Якобі за змінними керування $u$:

$$B = \frac{\partial f}{\partial u}\Big|_{x_o, u_0}$$

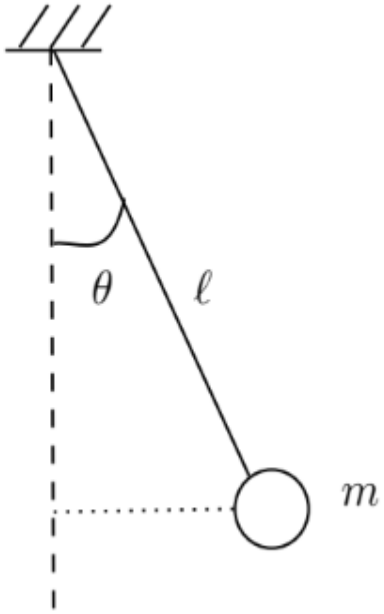
За допомогою матриць Якобі можна отримати лінеаризовані рівняння:

$$\delta\dot{x} = A\delta x + B\delta u$$

де:

- $\delta x = x - x_0$ представляє відхилення стану від точки спокою,
- $\delta u = u - u_0$ представляє відхилення керування від його фіксованого значення.

### 2.1.Приклад використання




Розглянемо модель коливань маятника з урахуванням тертя $k > 0$. На стрижні фіксованої довжини $l$ підвішено матеріальну точку масою $m$, на яку діє сила тяжіння.

Отримаємо рівняння руху маятника з тертям, використовуючи другий закон Ньютона для обертального руху [3], [5].

Обертовий момент інерції $I$ для маятника довжиною $l$ і масою $m$ відносно точки підвісу можна записати як $I = ml^2$.

Сума моментів сил навколо точки підвісу дорівнює добутку моменту інерції $I$ на кутове прискорення $\ddot{\theta}$

$$I * \ddot{\theta} = \sum \tau$$

Розглянемо сили, що діють на маятник:

1. Гравітаційна сила ($F_g$): Гравітаційна сила, що діє на масу $m$, створює момент сили $F_g * l * \sin\theta$ відносно точки підвісу, оскільки вона створює компоненту, перпендикулярну до напрямку маятника

$$F_g = mg.$$

2. Сила тертя ($F_f$): Сила тертя, пропорційна кутовій швидкості $\dot{\theta}$, створює момент сили $-F_f * l$, протилежний напрямку руху маятника

$$F_f = k * \dot{\theta}.$$

Тепер ми можемо записати рівняння руху:

$$ml^2 * \ddot{\theta} = -mg * l * \sin\theta - k * \dot{\theta} * l.$$

Для перетворення рівняння на систему диференціальних рівнянь (СДР) зі змінними $x = \theta,\ y = \dot{\theta}$ виконаємо заміну змінних та виразимо похідні

$$ml^2 * \ddot{x} = -mg * l * \sin x - k * l * y.$$

Розділимо обидві частини рівняння на $ml^2$:

$$\ddot{x} = -\frac{g}{l} * \sin x - \frac{k}{ml} * y$$

Таким чином ми отримали рівняння другого порядку. Його можна замінити на систему двох рівнянь першого порядку.

$$\dot{x} = y$$

$$\dot{y} = -\frac{g}{l} * \sin x - \frac{k}{ml} * y$$

Ми можемо додати якесь керування $u$ до нашої системи.

Для врахування керування в системі маятника ми додали змінну керування "$u$" в рівняння руху. Змінна "$u$" представляє собою зовнішню силу або момент, який може бути застосований до маятника. Це дозволяє нам контролювати та змінювати поведінку маятника. Рівняння тепер включає додатковий доданок $u$, який представляє вплив керування на кутове прискорення маятника. Ми можемо використовувати змінну "$u$" для реалізації різних стратегій керування, таких як стабілізація маятника в вертикальному положенні або створення контрольованих коливань

$$\dot{x} = y$$

$$\dot{y} = -\frac{g}{l} * \sin x - \frac{k}{ml} * y + u$$

Тепер все готово для лінеаризації.

Візьмемо фіксовану точку спокою $(x_0, y_0. u_0)$.

Далі обчислимо частинні похідні правої частини відносно змінних стану $x, y$ а також відносно керування $u$ в точці спокою:

Позначимо праві частини системи як $F_1, F_2$ відповідно.

$$\frac{\partial F_1}{\partial x} = 0, \qquad \frac{\partial F_1}{\partial y} = 1, \qquad \frac{\partial F_1}{\partial u} = 0$$

$$\frac{\partial F_2}{\partial x} = -\frac{g}{l}\cos x_0, \qquad \frac{\partial F_2}{\partial y} = -\frac{k}{ml}, \qquad \frac{\partial F_2}{\partial u} = 1$$

На основі частинних похідних формується матриця Якобі $J$ та матриця b:

$$J = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l}cosx_0 & -\frac{k}{ml} \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

І, нарешті, використовуючи матрицю Якобі J, можна записати лініаризовану систему у наступному вигляді:

$$\Delta\dot{z} = J * \Delta z + b * \Delta u$$

де:

- $\Delta z$ - це вектор відхилення від точки спокою, який складається з відхилень змінних стану.

$$\Delta z = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

Ця лініаризована система дозволяє нам аналізувати лінійну поведінку маятника в околі обраної точки спокою та розробляти стратегії керування на основі лінійних моделей.

### 2.2.Використання матриці Якобі для аналізу поведінки системи

Матриця Якобі дає нам інформацію про поведінку системи навколо точок спокою. Для одновимірних систем матриця Якобі є числом (похідною функції), яке вказує, чи відбувається зближення або віддалення точок від точки спокою.

Для багатовимірних систем матриця Якобі стає матрицею, і ми використовуємо власні значення цієї матриці, щоб визначити тип особливої точки (вузол, сідло, фокус і т. д.) і її стійкість [5]. Фазовий портрет, у свою чергу, надає візуальне представлення цієї динаміки, дозволяючи нам бачити траєкторії системи в просторі стану.

На прикладі нашої системи ми можемо також дослідити її поведінку у фазовому просторі біля точок спокою.

Для того щоб знайти точки спокою, необхідно розв'язати систему:

$$\begin{cases} y = 0 \\ -w^2 \sin x - \dfrac{k}{ml} * y = 0 \end{cases}$$

$$w = \sqrt{\frac{g}{l}}$$

Точки, які задовольняють цій системі: $(\pi n, 0)$, де $n$ – ціле. Насправді, через періодичність синусу, нам потрібні лише дві точки: $(0,0), (\pi, 0)$. Зауважимо, що ці точки відповідають нижньому та верхньому положенню рівноваги відповідно.

Тепер треба знайти власні значення матриць Якобі для наших точок спокою відповідно:

$$J = \begin{bmatrix} 0 & 1 \\ -w^2 \cos 0 & -\dfrac{k}{ml} \end{bmatrix}, \qquad J = \begin{bmatrix} 0 & 1 \\ -w^2 \cos \pi & -\dfrac{k}{ml} \end{bmatrix}$$

Знайшовши власні значення цих матриць, робимо висновок, що у точці (0,0) ми маємо стійкий фокус (при відносно малому $k$), а у точці (π,0) – сідло. За допомогою програми на Python, ми можемо подивитися як виглядає фазовий портрет системи при k=1/2. (див. рисунки 11, 12)

Отже, матриця Якобі і фазовий портрет спільно допомагають нам аналізувати та розуміти поведінку динамічних систем.

# 3.ТРИКУТНІ ТА МАЙЖЕ ТРИКУТНІ КЕРОВАНІ СИСТЕМИ

## 3.1.Означення та теорема

До цього лінеаризацію було розглянуто лише у сенсі апроксимації, за допомогою лінійного наближення правої частини системи. У цьому розділі буде розглянуто інше поняття лінеаризації, яке є особливо важливим для керованих систем.

У 1973 році В.І. Коробов у своїй роботі [1] вперше запропонував клас нелінійних керованих систем, які можуть бути відображені на лінійні за допомогою заміни змінних та керування. Такі системи автор запропонував називати **трикутними.** Вони мають наступний вигляд:

$$\begin{cases} \dot{x}_1 = f_1(x_1, x_2) \\ \dot{x}_2 = f_2(x_1, x_2, x_3) \\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots \\ \dot{x}_{n-1} = f_{n-1}(x_1, \dots, x_n) \\ \dot{x}_n = f_n(x_1, \dots, x_n, u) \end{cases}$$

*Теорема. Нехай у трикутній системі функції* $f_i(x_1, \dots, x_{i+1}), i = 1, \dots, n$ *мають неперервні частинні похідні до (n-i+1)-го порядку включно, і нехай*

$\left|\frac{\partial f_i}{\partial x_{i+1}}\right| \geq a > 0,\ i = 1, \dots, n,$ *при всіх* $x_1, \dots, x_{n+1},$

*де a-стала, яка не залежить від* $x_1, \dots, x_{n+1}$. *Тоді трикутна система повністю керована.*

Доведення цієї теореми спирається на запропонований автором метод заміни змінних і керування, за допомогою якого можна відобразити систему на лінійну.

Цей підхід було узагальнено у багатьох напрямках. Зокрема, нещодавно В.І. Коробов у роботах [2], [6] узагальнив поняття трикутної системи і запропонував розглядати задачу керованості для класу майже трикутних систем

$$\begin{cases} \dot{x}_1 = f_1(x_1, x_2) \\ \dot{x}_2 = f_2(x_1, x_2, x_3) \\ \dots\dots\dots\dots\dots\dots\dots \\ \dot{x}_{n-2} = f_{n-2}(x_1, \dots, x_{n-1}) \\ \dot{x}_{n-1} = f_{n-1}(x_1, \dots, x_n, u) \\ \dot{x}_n = f_n(x_1, \dots, x_n, u) \end{cases}$$

Далі буде наведено кілька прикладів трикутних та майже трикутних систем і продемонстровано процес побудування функцій керування.

### 3.2.Приклади пошуку керування для трикутних та майже трикутних систем

3.2.1. Квадратична функція керування.

Розглянемо нелінійну керовану систему:

$$\begin{cases} \dot{x}_1 = x_1^2 + x_2 \\ \dot{x}_2 = x_1^3 - x_2 + x_3 \\ \dot{x}_3 = x_1^3 - 2x_2^2 + u \end{cases} \quad (1)$$

Знайдемо керування u(t), яке переводить систему з початкової точки $x(0) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, до точки $x(T) = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$ за час T=1.

У даному випадку нам знадобиться означення трикутної системи та алгоритм пошуку керування u(t) за допомогою заміни змінних, який було запропоновано у роботі [1].

Зробимо наступну заміну:

$$y_1 = x_1$$

$$\dot{y}_1 = x_1^2 + x_2 = y_2$$

$$\dot{y}_2 = 2x_1\dot{x}_1 + \dot{x}_2 = 2x_1(x_1^2 + x_2) + x_1^3 - x_2 + x_3 = 3x_1^3 + 2x_1x_2 - x_2 + x_3 = y_3$$

$$\dot{y}_3 = 9x_1^2\dot{x}_1 + 2x_1\dot{x}_2 + 2x_2\dot{x}_1 - \dot{x}_2 + \dot{x}_3 = 9x_1^2(x_1^2 + x_2) + 2x_1(x_1^3 - x_2 + x_3) + 2x_2(x_1^2 + x_2) - x_1^3 + x_2 - x_3 + x_1^3 - 2x_2^2 + u = 11x_1^4 + 11x_1^2x_2 - 2x_1x_2 + 2x_1x_3 + x_2 - x_3 + u = v$$

Таким чином ми отримуємо канонічну лінійну систему з новим керуванням *v(t),* яке має переводити нову систему:

$$\begin{cases} \dot{y}_1 = x_1^2 + x_2 = y_2 \\ \dot{y}_2 = 3x_1^3 + 2x_1x_2 - x_2 + x_3 = y_3 \\ \dot{y}_3 = 11x_1^4 + 11x_1^2 - 2x_1x_2 + 2x_1x_3 + x_2 - x_3 + u = v \end{cases} \quad (2)$$

з точки $y(0) = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}$, у точку $y(T) = \begin{pmatrix} 0 \\ 1 \\ -2 \end{pmatrix}$ за той же час T=1

У цьому прикладі розглянемо керування *v(t)* у вигляді поліному другого степеня:

$$v(t) = a + bt + ct^2$$

Тепер можна проінтегрувати рівняння канонічної системи для пошуку нового керування *v(t)*.

$$\int_0^T \dot{y}_3 dt = \int_0^T v(t)dt$$

$$y_3(T) - y_3(0) = at + \frac{bt^2}{2} + \frac{ct^3}{3}\Big|_0^T$$

$$y_3(t) = at + \frac{bt^2}{2} + \frac{ct^3}{3} + 3$$

$$\int_0^T \dot{y}_2 dt = \int_0^T y_3 dt$$

$$y_2(T) - y_2(0) = \frac{at^2}{2} + \frac{bt^3}{6} + \frac{ct^4}{12} + 3t\Big|_0^T$$

$$y_2(t) = \frac{at^2}{2} + \frac{bt^3}{6} + \frac{ct^4}{12} + 3t + 1$$

$$\int_0^T \dot{y}_1 dt = \int_0^T y_2 dt$$

$$y_1(T) - y_1(0) = \frac{at^3}{6} + \frac{bt^4}{24} + \frac{ct^5}{60} + \frac{3t^2}{2} + t\Big|_0^T$$

$$y_1(t) = \frac{at^3}{6} + \frac{bt^4}{24} + \frac{ct^5}{60} + \frac{3t^2}{2} + t + 1$$

Тепер ми маємо розв’язати систему лінійних рівнянь відносно *a,b,c*.

Підставимо T=1:

$$\begin{cases} -5 = a + \frac{b}{2} + \frac{c}{3} \\ 0 = \frac{a}{2} + \frac{b}{6} + \frac{c}{12} + 3 \\ -1 = \frac{a}{6} + \frac{b}{24} + \frac{c}{60} + \frac{3}{2} + 1 \end{cases} \Leftrightarrow \begin{cases} -30 = 6a + 3b + 2c \\ 6a + 2b + c + 36 = 0 \\ -240 = 20a + 5b + 2c + 180 \end{cases} \Leftrightarrow$$

$$\begin{cases} 6a + 3b + 2c = -30 \\ 6a + 2b + c = -36 \\ 20a + 5b + 2c = -420 \end{cases} \Leftrightarrow \begin{cases} c = 6 - b \\ b = -42 - 6a \\ 20a - 210 - 30a + 96 + 12a = -420 \end{cases} \Leftrightarrow$$

$$\begin{cases} c = 48 + 6a \\ b = -42 - 6a \\ 2a = -306 \end{cases} \Leftrightarrow \begin{cases} a = -153 \\ b = 876 \\ c = -870 \end{cases}$$

Таким чином, ми отримуємо керування $v(t) = -153 + 876t - 870t^2$, за допомогою якого ми знаходимо керування *u(t)* у неявному вигляді, шляхом підстановки *v(t)* до системи (2).

Для перевірки знайденого керування, скористуємося програмою на Python, та подивимося на поведінку системи на графіку (див. рисунки 3, 4).

3.2.2 Кубічна функція

Будемо розглядати ті ж системи (1), (2) тільки тепер візьмемо керування іншого виду, та подивимося, чи зміниться результат пошуку такого керування.

Шукатимемо керування *v(t)* у вигляді: $v(t) = a + bt + ct^3$, тобто поліном третього степеня, але з трьома невідомими в контексті нашої СЛР.

Алгоритм пошуку не відрізняється, інтегруємо рівняння канонічної системи (2) та переходимо у результаті до СЛР, для подальшого пошуку невідомих параметрів *a,b,c*.

$$\int_0^T \dot{y}_3 dt = \int_0^T v(t) dt$$

$$y_3(T) - y_3(0) = at + \frac{bt^2}{2} + \frac{ct^4}{4}\Big|_0^T$$

$$y_3(t) = at + \frac{bt^2}{2} + \frac{ct^4}{4} + 3$$

$$\int_0^T \dot{y}_2 dt = \int_0^T y_3 dt$$

$$y_2(T) - y_2(0) = \frac{at^2}{2} + \frac{bt^3}{6} + \frac{ct^5}{20} + 3t \Big|_0^T$$

$$y_2(t) = \frac{at^2}{2} + \frac{bt^3}{6} + \frac{ct^5}{20} + 3t + 1$$

$$\int_0^T \dot{y}_1 dt = \int_0^T y_2 dt$$

$$y_1(T) - y_1(0) = \frac{at^3}{6} + \frac{bt^4}{24} + \frac{ct^6}{120} + \frac{3t^2}{2} + t \Big|_0^T$$

$$y_1(t) = \frac{at^3}{6} + \frac{bt^4}{24} + \frac{ct^6}{120} + \frac{3t^2}{2} + t + 1$$

Переходимо до СЛР взявши T=1:

$$\begin{cases} \frac{a}{6} + \frac{b}{24} + \frac{c}{120} + \frac{3}{2} + 1 + 1 = 0 \\ \frac{a}{2} + \frac{b}{6} + \frac{c}{20} + 3 + 1 = 1 \\ a + \frac{b}{2} + \frac{c}{4} + 3 = -2 \end{cases} \Leftrightarrow \begin{cases} 20a + 5b + c = -420 \\ 30a + 10b + 3c = -180 \\ 4a + 2b + c = -20 \end{cases} \Leftrightarrow$$

$$\begin{cases} 60a + 15b + 3c = -1260 \\ 60a + 20b + 6c = -360 \\ 4a + 2b + c = -20 \end{cases} \Leftrightarrow \begin{cases} 5b = 900 - 3c \\ 20a = 2c - 1320 \\ 20a + 10b + 5c = -100 \end{cases} \Leftrightarrow \begin{cases} 5b = 900 - 3c \\ 20a = 2c - 1320 \\ c = -580 \end{cases}$$

$$\Leftrightarrow \begin{cases} a = -124 \\ b = 528 \\ c = -580 \end{cases}$$

Тепер ми отримуємо керування іншого вигляду $v(t) = -124 + 528t - 580t^3$

Для перевірки отриманого результату, подивимося на роботу програми з таким керуванням та подивимося на перехід системи до нашої точки $x(T) = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$ за час T=1 (див. рисунки 5, 6)

### 3.2.3. Функція вищого порядку

До цього ми розглядали лише випадки, коли кількість параметрів обраної функції керування дорівнюють порядку системи лінійних рівнянь. Тому кожного разу ми отримували однозначну відповідь у вигляді значень параметрів для нашої функції керування. Але що буде, якщо кількість невідомих параметрів перевищуватиме порядок системи?

У такому разі ми отримаємо ціле сімейство функцій керування, яке задовольняє нашому запиту щодо переведення системи в задану точку за кінцевий час.

Візьмемо $v(t) = a + bt + ct^2 + dt^3$

Тоді:

$$\int_0^T \dot{y}_3 dt = \int_0^T v(t) dt$$

$$y_3(T) - y_3(0) = at + \frac{bt^2}{2} + \frac{ct^3}{3} + \frac{dt^4}{4} \Big|_0^T$$

$$y_3(t) = at + \frac{bt^2}{2} + \frac{ct^3}{3} + \frac{dt^4}{4} + 3$$

$$\int_0^T \dot{y}_2 dt = \int_0^T y_3 dt$$

$$y_2(T) - y_2(0) = \frac{at^2}{2} + \frac{bt^3}{6} + \frac{ct^4}{12} + \frac{dt^5}{20} + 3t \Big|_0^T$$

$$y_2(t) = \frac{at^2}{2} + \frac{bt^3}{6} + \frac{ct^4}{12} + \frac{dt^5}{20} + 3t + 1$$

$$\int_0^T \dot{y}_1 dt = \int_0^T y_2 dt$$

$$y_1(T) - y_1(0) = \frac{at^3}{6} + \frac{bt^4}{24} + \frac{ct^5}{60} + \frac{dt^6}{120} + \frac{3t^2}{2} + t \Big|_0^T$$

$$y_1(t) = \frac{at^3}{6} + \frac{bt^4}{24} + \frac{ct^5}{60} + \frac{dt^6}{120} + \frac{3t^2}{2} + t + 1$$

Отримуємо СЛР, підставивши час T=1:

$$\begin{cases} 0 = \frac{a}{6} + \frac{b}{24} + \frac{c}{60} + \frac{d}{120} + \frac{3}{2} + 1 + 1 \\ 1 = \frac{a}{2} + \frac{b}{6} + \frac{c}{12} + \frac{d}{20} + 3 + 1 \\ -2 = a + \frac{b}{2} + \frac{c}{3} + \frac{d}{4} + 3 \end{cases} \Leftrightarrow \begin{cases} 40a + 10b + 4c + 2d = -840 \\ 30a + 10b + 5c + 3d = -180 \\ 12a + 6b + 4c + 3d = -60 \end{cases} \Leftrightarrow$$

$$\begin{cases} 10a = c + d - 660 \\ 8c + 6d + 10b = 1800 \\ 6c + 6d - 3960 + 30b + 20c + 15d = -300 \end{cases} \Leftrightarrow \begin{cases} 10a = c + d - 660 \\ 2c + 3d = -1740 \\ 8c + 6d + 10b = 1800 \end{cases} \Leftrightarrow$$

$$\begin{cases} 2c = -1740 - 3d \\ 20a = -3060 - d \\ 5b = 4380 + 3d \end{cases} \Leftrightarrow \begin{cases} c = -870 - \frac{3d}{2} \\ a = -153 - \frac{d}{20} \\ b = 876 + \frac{3d}{5} \end{cases}$$

Візьмемо d=20, тоді керування буде мати вигляд:

$$v(t) = -154 + 888t - 900t^2 + 20t^3$$

Знову перевіряємо отриманий результат у нашій програмі Python (див. рисунки 7, 8)

3.2.4. Експоненціальна функція

Розглянемо у якості функції керування експоненціальну функцію:

$v(t) = ae^t + be^{-t} + ce^{2t}$

$$\int_0^T \dot{y}_3 dt = \int_0^T v(t) dt$$

$$y_3(T) - y_3(0) = ae^t - be^{-t} + \frac{ce^{2t}}{2} - a + b - \frac{c}{2}$$

$$y_3(t) = ae^t - be^{-t} + \frac{ce^{2t}}{2} - a + b - \frac{c}{2} + 3$$

$$\int_0^T \dot{y}_2 dt = \int_0^T y_3 dt$$

$$y_2(T) - y_2(0) = ae^t + be^{-t} + \frac{ce^{2t}}{4} - at + bt - \frac{ct}{2} + 3t - a - b - \frac{c}{4}$$

$$y_2(t) = ae^t + be^{-t} + \frac{ce^{2t}}{4} - at + bt - \frac{ct}{2} + 3t - a - b - \frac{c}{4} + 1$$

$$\int_0^T \dot{y}_1 dt = \int_0^T y_2 dt$$

$$y_1(T) - y_1(0) = ae^t - be^{-t} + \frac{ce^{2t}}{8} - \frac{at^2}{2} + \frac{bt^2}{2} - \frac{ct^4}{4} + \\ + \frac{3t^2}{2} - at - bt - \frac{ct}{4} + t - a + b - \frac{c}{8}$$

$$y_1(t) = ae^t - be^{-t} + \frac{ce^{2t}}{8} - \frac{at^2}{2} + \frac{bt^2}{2} - \frac{ct^4}{4} + \frac{3t^2}{2} - at - bt - \frac{ct}{4} + t - a + b \\ - \frac{c}{8} + 1$$

Отримуємо СЛР, підставивши час T=1:

$$\begin{cases} a(8e^2 - 20e) + b(4e - 8) + c(e^3 - 5e) = -28e \\ a(4e^2 - 8e) + 4b + c(e^3 - 3e) = -12e \\ a(2e^2 - 2e) + b(2e - 2) + c(e^3 - e) = -10e \end{cases}$$

Для розв'язання даної системи будемо використовувати метод Гауса.

Через те, що обчислення займатимуть забагато місця, запишемо одразу значення параметрів, знайдені за допомогою програми:

$$\begin{cases} a = \dfrac{-5e^3 + 15e^2 - 3e + 23}{2e^4 - 12e^3 + 18e^2 + 4e - 12} \\ b = \dfrac{-3e^4 + 9e^3 + e^2 - 17e}{2e^4 - 12e^3 + 18e^2 + 4e - 12} \\ c = \dfrac{-2e + 22}{e^3 - 3e^2 + 2} \end{cases}$$

Перевіримо знайдене керування у програмі Python (див. рисунки 9, 10)

3.2.5.Лінійна функція керування

Розглянемо приклад майже трикутної системи:

$$\begin{cases} \dot{x}_1 = u \\ \dot{x}_2 = x_1^3 \end{cases}$$

Нехай керування $u(t)$ задано у вигляді: $u(t) = \alpha + \beta t$

Ми хочемо знайти такі значення $\alpha$ та $\beta$, щоб система переводила початкові умови $x_1(0) = x_2(0) = 0$, у довільну точку $\left(x_1(t_0); x_2(t_0)\right)$, де $t_0$ – кінцевий час.

Проінтегруємо рівняння системи:

$$\int_0^{t_0} \dot{x}_1(t)\,dt = \int_0^{t_0} u(t)\,dt$$

$$\int_0^{t_0} \dot{x}_1(t)\,dt = \int_0^{t_0} (\alpha + \beta t)\,dt$$

$$x_1(t_0) = \alpha t_0 + \frac{1}{2}\beta t_0^2$$

Підставимо це в друге рівняння системи:

$$\dot{x}_2(t) = \left(\alpha t\ + \frac{1}{2} * \beta t^2\right)^3$$

$$\dot{x}_2(t) = \alpha^3 t^3 + \frac{3}{2}\alpha^2\beta t^4 + \frac{3}{4}\alpha\beta^2 t^5 + \frac{1}{8}\beta^3 t^6$$

$$\int_0^{t_0} \dot{x}_2(t)\,dt = \int_0^{t_0} (\alpha^3 t^3 + \frac{3}{2}\alpha^2\beta t^4 + \frac{3}{4}\alpha\beta^2 t^5 + \frac{1}{8}\beta^3 t^6)\,dt$$

$$x_2(t_0) = \frac{1}{4}\alpha^3 t_0^4 + \frac{3}{10}\alpha^2\beta t_0^5 + \frac{1}{8}\alpha\beta^2 t_0^6 + \frac{1}{56}\beta^3 t_0^7$$

Отримуємо наступну систему:

$$\begin{cases} x_1(t_0) = \alpha t_0 + \frac{1}{2}\beta t_0^2 \\ x_2(t_0) = \frac{1}{4}\alpha^3 t_0^4 + \frac{3}{10}\alpha^2\beta t_0^5 + \frac{1}{8}\alpha\beta^2 t_0^6 + \frac{1}{56}\beta^3 t_0^7 \end{cases}$$

Візьмемо кінцевий час $t_0 = 1$

$$\begin{cases} x_1(1) = \alpha + \frac{1}{2}\beta \\ x_2(1) = \frac{1}{4}\alpha^3 + \frac{3}{10}\alpha^2\beta + \frac{1}{8}\alpha\beta^2 + \frac{1}{56}\beta^3 \end{cases}$$

$\Leftrightarrow$

$$\begin{cases} \beta = 2x_1(1) - 2\alpha \\ x_2(1) = \frac{1}{4}\alpha^3 + \frac{3}{10}\alpha^2(2x_1(1) - 2\alpha) + \frac{1}{8}\alpha(2x_1(1) - 2\alpha)^2 + \frac{1}{56}(2x_1(1) - 2\alpha)^3 \end{cases}$$

$\Leftrightarrow$

$$\begin{cases} \beta = 2x_1(1) - 2\alpha \\ 140x_2(1) = \alpha^3 + 4\alpha^2 x_1(1) + 10\alpha x_1^2(1) + 20x_1^3(1) \end{cases}$$

$\Leftrightarrow$

$$\begin{cases} \beta = 2x_1(1) - 2\alpha \\ \alpha^3 + 4\alpha^2 x_1(1) + 10\alpha x_1^2(1) + 20x_1^3(1) - 140x_2(1) = 0 \end{cases}$$

Кожен дійсний многочлен непарного степеня має принаймні один дійсний корінь. Друге рівняння системи є кубічним рівнянням відносно $\alpha$, тому ми зможемо знайти дійсні значення $\alpha$ і $\beta$ для будь-яких $x_1(1), x_2(1)$.

Візьмемо $\big(x_1(1), x_2(1)\big) = (2,2)$, тоді:

$$\begin{cases} \beta = 4 - 2\alpha \\ \alpha^3 + 8\alpha^2 + 40\alpha + 160 - 280 = 0 \end{cases}$$

$\Leftrightarrow$

$$\begin{cases} \beta = 4 - 2\alpha \\ (\alpha - 2)(\alpha^2 + 10\alpha + 60) = 0 \end{cases}$$

$$(D < 0)$$

$\Rightarrow$

$$\begin{cases} \alpha = 2 \\ \beta = 0 \end{cases}$$

У цьому випадку ми отримуємо керування $u(t) = 2$ для усіх значень $t$ у межах від 0 до 1 для переведення нашої системи з точки (0;0) у точку (2;2) Візьмемо $\big(x_1(1), x_2(1)\big) = (4,2)$, тоді:

$$\begin{cases} \beta = 8 - 2\alpha \\ \alpha^3 + 16\alpha^2 + 160\alpha + 1000 = 0 \end{cases}$$

$\Leftrightarrow$

$$\begin{cases} \beta = 8 - 2\alpha \\ (\alpha + 10)(\alpha^2 + 6\alpha + 100) = 0 \end{cases}$$

$$(D < 0)$$

$\Rightarrow$

$$\begin{cases} \beta = 28 \\ \alpha = -10 \end{cases}$$

Таким чином ми отримуємо керування $u(t) = -10 + 28t$, яке переводить нашу систему з точки (0;0) у точку (4;2) за час $t$

За допомогою Python знайдемо вигляд графіків траєкторій для $x_1$, $x_2$ та траєкторії системи. (див. рисунки 1, 2)

Для того, щоб знайти керування у цьому прикладі нам знадобилося лише проінтегрувати рівняння системи. Але, якщо обрати більш складну систему, нам не вистачить лише інтегрування для пошуку функції керування, тому, як для майже трикутної системи, доведеться також застосовувати заміну змінних.

# 4.ВИСНОВКИ

У цій роботі було розглянуто два різні підходи до поняття лінеаризації систем. Спочатку була розглянута лінеаризація за допомогою апроксимації та лінійного наближення, що є важливим методом в теорії диференціальних рівнянь. Окрім цього, ми наочно дослідили, як поводиться нелінійна система, що описує цілком реальну фізичну модель математичного маятника з урахуванням тертя, у фазовому просторі в околицях точок спокою. Для цього були використані матриці частинних похідних, що звуться матрицями Якобі.

Далі була розглянута лінеаризація в сенсі відображення нелінійної системи на лінійну шляхом заміни змінних і керування, що є одним з важливих підходів в теорії керування. Зокрема для трикутних та майже трикутні систем цей метод дозволяє будувати керування, що переводить систему з однієї заданої точки до іншої. На прикладах ми розглянули різні види функцій керування для трикутних систем та проаналізували графіки траєкторій системи за допомогою програми на Python.

**ВИКОРИСТАНІ ДЖЕРЕЛА**


1. Korobov V. I., Controllability, stability of certain nonlinear systems. Differential Equations, 9:466–469, 1973.

2. Korobov V.I., Almost linearizable control systems. Mathematics of Control, Signals, and Systems, 33(3):473–497, 2021.

3. Анісімов І.О., Коливання та хвилі: Навчальний посібник, КНУ ім. Т.Шевченка, 2001.

4. Хусаінов Д.Я., Харченко І.І., Шатирко А.В., Введення в моделювання динамічних систем: Навчальний посібник, КНУ ім. Т.Шевченка, 2010.

5. Kaplan D., Glass L., Understanding nonlinear dynamics, Springer, 1995.

6. Korobov V.I., Sklyar K.V., Ignatovich S.Y., Almost feedback linearizable systems of the class $C^1$ and solving the constructive controllability problem, IMA Journal of Mathematical Control and Information, 2024.

## ДОДАТКИ

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

def x1(t):
    return -10 * t + 14 * t**2

def x2(t):
    return -250 * t**4 + 840 * t**5 - 980 * t**6 + 392 * t**7

t_values = np.linspace(0, 1, 100)
x1_values = x1(t_values)
x2_values = x2(t_values)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(t_values, x1_values, label='X1(t)')
plt.scatter([1], [4], color='red', label='Кінцевий стан (t=1)')
plt.xlabel('t')
plt.ylabel('x1')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(t_values, x2_values, label='X2(t)')
plt.scatter([1], [4], color='red', label='Кінцевий стан (t=1)')
plt.xlabel('t')
plt.ylabel('x2')
plt.legend()

plt.suptitle('Траєкторії системи')
plt.show()
```

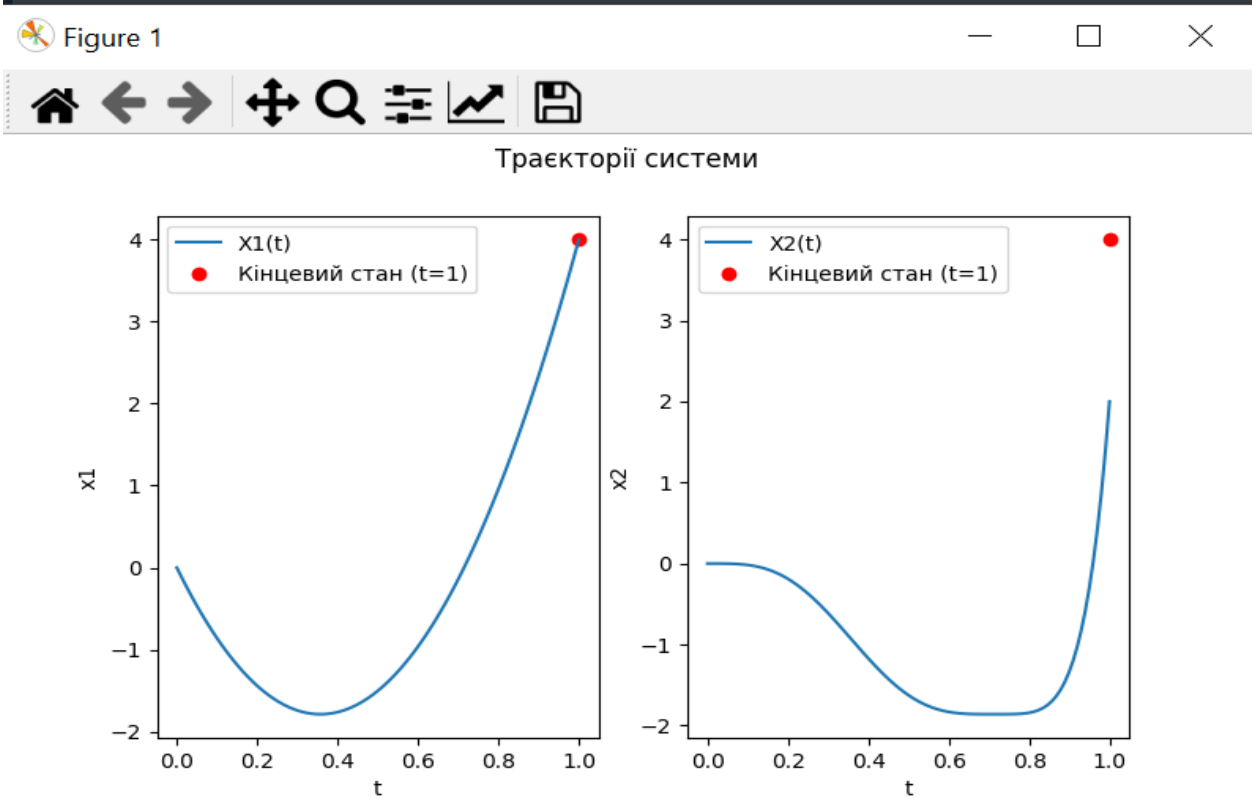



Рис. 1

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

def solution(t):
    x1 = -10 * t + 14 * t**2
    x2 = -250 * t**4 + 840 * t**5 - 980 * t**6 + 392 * t**7
    return x1, x2

initial_conditions = [0, 0]
t = np.arange(0, 1.01, 0.01)
X1_1, X2_1 = solution(t)

plt.plot(X1_1, X2_1, label='Траєкторія X(t)')
plt.scatter([0], [0], color='green', label='Початкова точка (0, 0)')
plt.scatter([4], [2], color='red', label='Кінцева точка (4, 2)')
plt.title('Траєкторія системи')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.grid(True)
plt.show()
```

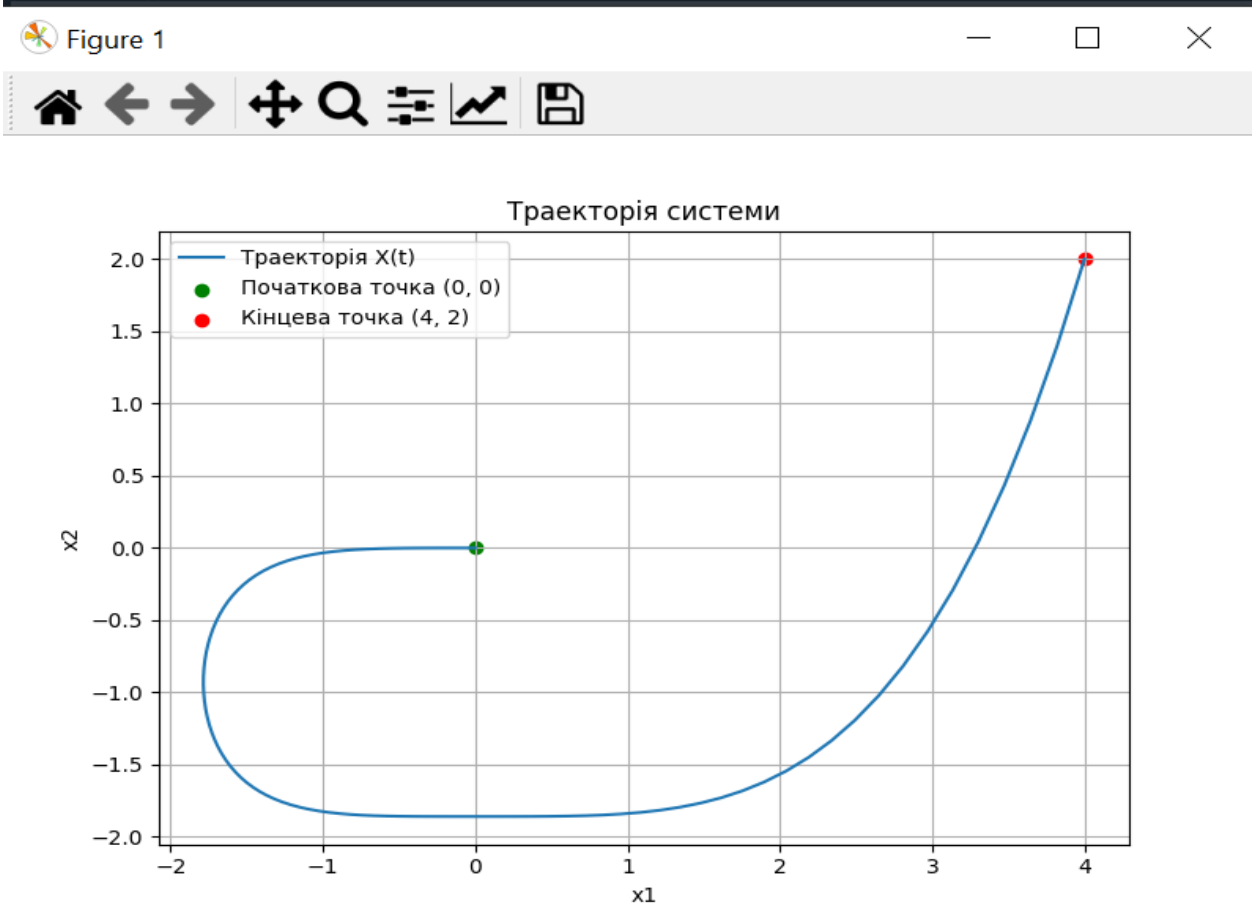



Рис. 2

```python
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import math
```

```
    # Система ДР
    def system(t, state):
        x1, x2, x3 = state
        u = -153 + 876*t -870*(t**2) - 11*(x1**4) - 11*(x1**2)*x2 + 2*x1*x2 - 2*x1*x3
- x2 + x3

        dx1_dt = (x1)**2 + x2
        dx2_dt = (x1)**3 - x2 + x3
        dx3_dt = (x1)**3 - 2*((x2)**2) + u

        return [dx1_dt, dx2_dt, dx3_dt]

    # Початковий стан (1, 0, 0)
    initial_state = [1, 0, 0]

    # Час від 0 до 1
    t_span = (0, 1)
    t_eval = np.linspace(t_span[0], t_span[1], 1000)

    # Розв'язуємо систему ДР
    solution = solve_ivp(system, t_span, initial_state, t_eval=t_eval, method='RK45')

    # Отримуємо розв'язки x1, x2, x3
    x1 = solution.y[0]
    x2 = solution.y[1]
    x3 = solution.y[2]

    # Будуємо графік
    plt.figure(figsize=(10, 6))
    plt.plot(t_eval, x1, label='x1(t)')
    plt.plot(t_eval, x2, label='x2(t)')
    plt.plot(t_eval, x3, label='x3(t)')
    plt.xlabel('Час t')
    plt.ylabel('Стан системи')
    plt.title('Рух системи')
    plt.legend()
    plt.grid(True)
    plt.show()
```

Рис. 3

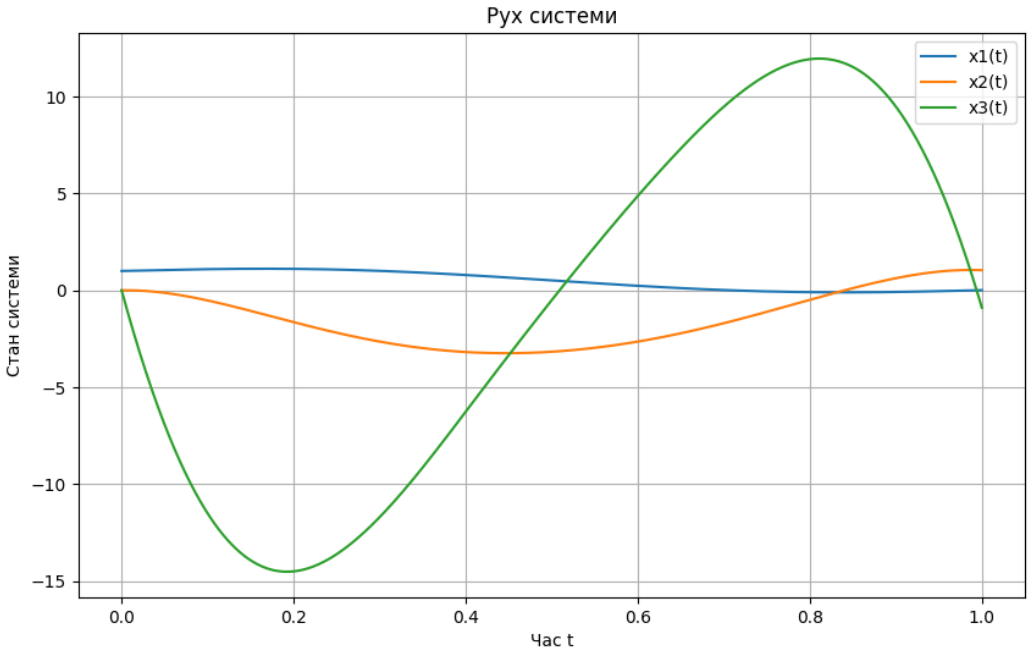



Рис. 4

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import math

# Система ДР
def system(t, state):
    x1, x2, x3 = state
    u = -124 + 528*t -580*(t**3) - 11*(x1**4) - 11*(x1**2)*x2 + 2*x1*x2 - 2*x1*x3
- x2 + x3

    dx1_dt = (x1)**2 + x2
    dx2_dt = (x1)**3 - x2 + x3
    dx3_dt = (x1)**3 - 2*((x2)**2) + u

    return [dx1_dt, dx2_dt, dx3_dt]

# Початковий стан (1, 0, 0)
initial_state = [1, 0, 0]

# Час від 0 до 1
t_span = (0, 1)
t_eval = np.linspace(t_span[0], t_span[1], 1000)

# Розв'язуємо систему ДР
solution = solve_ivp(system, t_span, initial_state, t_eval=t_eval, method='RK45')
```

```
# Отримуємо розв'язки x1, x2, x3
x1 = solution.y[0]
x2 = solution.y[1]
x3 = solution.y[2]

# Будуємо графік
plt.figure(figsize=(10, 6))
plt.plot(t_eval, x1, label='x1(t)')
plt.plot(t_eval, x2, label='x2(t)')
plt.plot(t_eval, x3, label='x3(t)')
plt.xlabel('Час t')
plt.ylabel('Стан системи')
plt.title('Рух системи')
plt.legend()
plt.grid(True)
plt.show()
```

Рис. 5

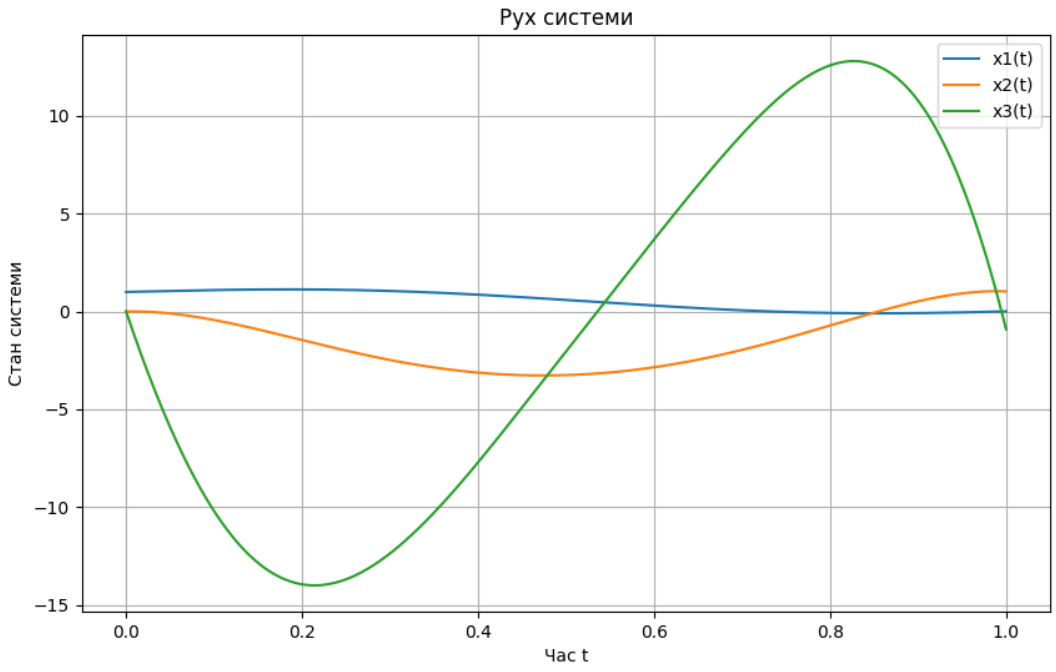



Рис. 6

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import math

# Система ДР
def system(t, state):
```

```
        x1, x2, x3 = state
        u = -154 + 888*t - 900*(t**2) + 20*(t**3) - 11*(x1**4) - 11*(x1**2)*x2 +
2*x1*x2 - 2*x1*x3 - x2 + x3

        dx1_dt = (x1)**2 + x2
        dx2_dt = (x1)**3 - x2 + x3
        dx3_dt = (x1)**3 - 2*((x2)**2) + u

        return [dx1_dt, dx2_dt, dx3_dt]

    # Початковий стан (1, 0, 0)
    initial_state = [1, 0, 0]

    # Час від 0 до 1
    t_span = (0, 1)
    t_eval = np.linspace(t_span[0], t_span[1], 1000)

    # Розв'язуємо систему ДР
    solution = solve_ivp(system, t_span, initial_state, t_eval=t_eval, method='RK45')

    # Отримуємо розв'язки x1, x2, x3
    x1 = solution.y[0]
    x2 = solution.y[1]
    x3 = solution.y[2]

    # Будуємо графік
    plt.figure(figsize=(10, 6))
    plt.plot(t_eval, x1, label='x1(t)')
    plt.plot(t_eval, x2, label='x2(t)')
    plt.plot(t_eval, x3, label='x3(t)')
    plt.xlabel('Час t')
    plt.ylabel('Стан системи')
    plt.title('Рух системи')
    plt.legend()
    plt.grid(True)
    plt.show()
```

Рис. 7

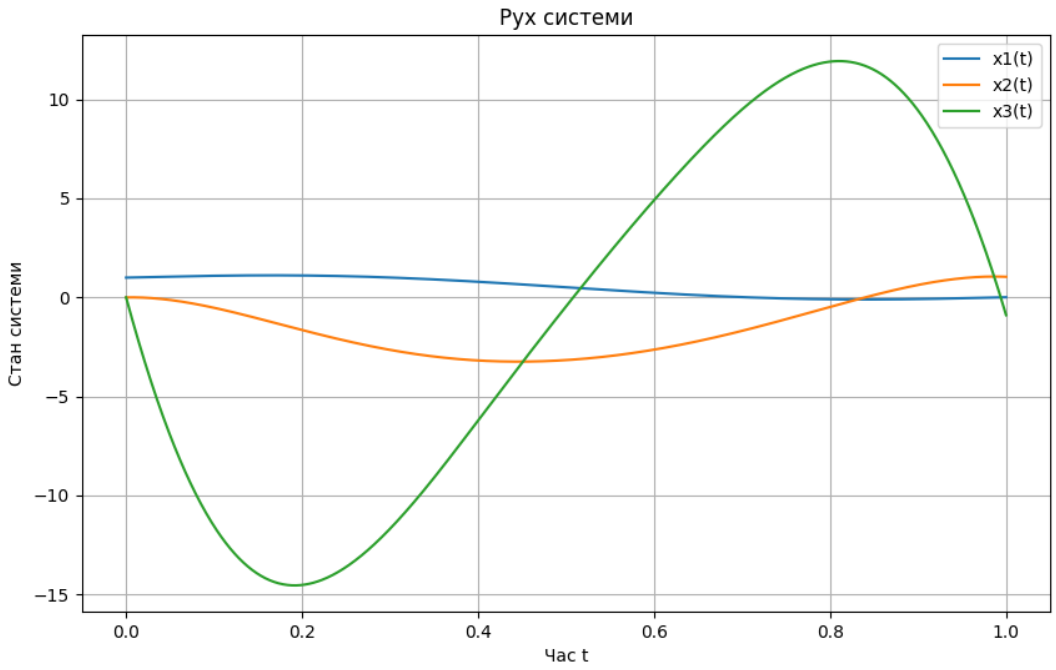



Рис. 8

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import math

# Система ДР
def system(t, state):
    x1, x2, x3 = state
    u = (-5*math.exp(t+3)+15*math.exp(t+2)-3*math.exp(t+1)+23*math.exp(t))/(2*math.exp(4)-12*math.exp(3)+18*math.exp(2)+4*math.exp(1)-12) + (-3*math.exp(4-t)+9*math.exp(3-t)+math.exp(2-t)-17*math.exp(1-t))/(2*math.exp(4)-12*math.exp(3)+18*math.exp(2)+4*math.exp(1)-12) + (-2*math.exp(1+2*t)+22*math.exp(2*t))/(math.exp(3)-3*math.exp(2)+2) - 11*(x1**4) - 11*(x1**2)*x2 + 2*x1*x2 - 2*x1*x3 - x2 + x3

    dx1_dt = (x1)**2 + x2
    dx2_dt = (x1)**3 - x2 + x3
    dx3_dt = (x1)**3 - 2*((x2)**2) + u

    return [dx1_dt, dx2_dt, dx3_dt]

# Початковий стан (1, 0, 0)
initial_state = [1, 0, 0]

# Час від 0 до 1
```

```
t_span = (0, 1)
t_eval = np.linspace(t_span[0], t_span[1], 1000)

# Розв'язуємо систему ДР
solution = solve_ivp(system, t_span, initial_state, t_eval=t_eval, method='RK45')

# Отримуємо розв'язки x1, x2, x3
x1 = solution.y[0]
x2 = solution.y[1]
x3 = solution.y[2]

# Будуємо графік
plt.figure(figsize=(10, 6))
plt.plot(t_eval, x1, label='x1(t)')
plt.plot(t_eval, x2, label='x2(t)')
plt.plot(t_eval, x3, label='x3(t)')
plt.xlabel('Час t')
plt.ylabel('Стан системи')
plt.title('Рух системи')
plt.legend()
plt.grid(True)
plt.show()
```

Рис. 9

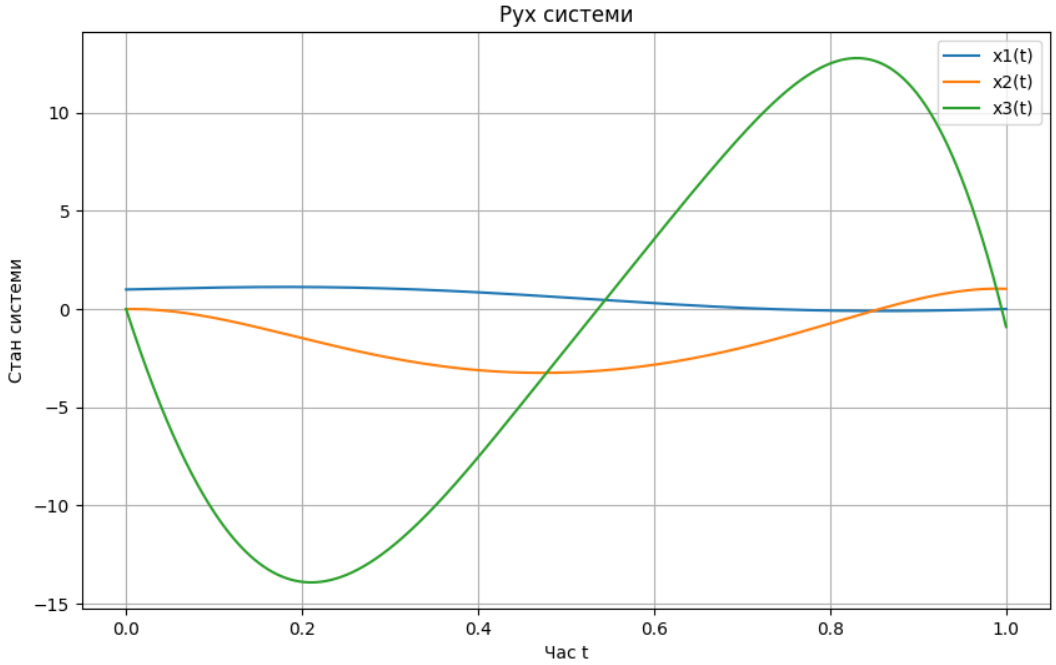



Рис. 10

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

def f(x, y, k):
    return y, -np.sin(x) - k*y

fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot()
ax.grid()
ax.set_aspect('equal')

x = np.linspace(-2 * np.pi, 2 * np.pi, 20)
y = np.linspace(-3, 3, 20)
xx, yy = np.meshgrid(x, y)

k = 0.5  # Коефіцієнт тертя
f1, f2 = f(xx, yy, k)
ax.streamplot(xx, yy, f1, f2, density=1.8, color='blue', linewidth=1)

plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

Рис. 11

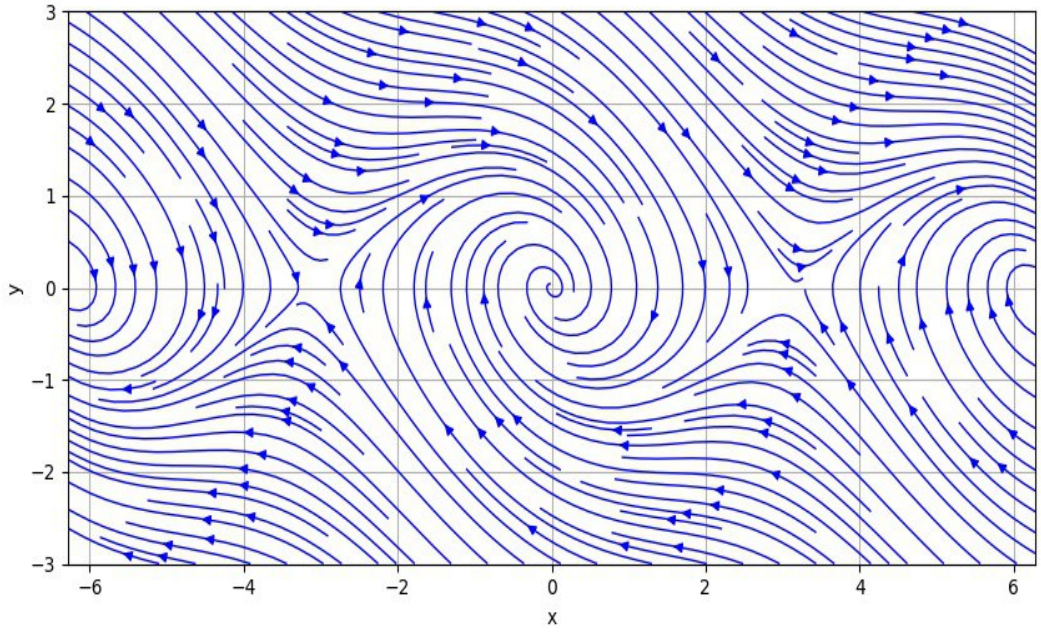



Рис.12