ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА
МЕХАНІКО-МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА МАТЕМАТИЧНОЇ ЕКОНОМІКИ ТА ЕКОНОМЕТРИКИ

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Оптимальне керування системою реакції дифузії та його чисельна реалізація

студента VI курсу
групи МТЕМ-21
Войтовича Ярослава

Науковий керівник:
доцент
Флюд В. М.

Львів — 2022

Зміст

Розділ 1

Вступ

Моделювання динамічних систем лежить в основі багатьох галузей людської життєдіяльності, які мають справу зі змінними в часі процесами. Традиційно можна виділити три задачі такого моделювання, а саме описову, предиктивну та прескриптивну. Рішення описових задач зазвичай полягає у відповіді на запитання про поточний або минулий стан розглянутої моделі, тоді як предиктивні задачі потребують пошуку можливих станів системи у майбутньому. Прескриптивні задачі включають більш глибокий аналіз системи в тому сенсі, що не лише вимагають відповіді на запитання про майбутній стан системи, а також встановлення множини рішень, що допоможуть досягнути деяких бажаних станів відповідно до контексту задачі. Задачі оптимального керування є підмножиною таких прескриптивних задач математичного моделювання. Метою задач оптимального керування є пошук контролю динамічної системи, при якому деяка задана цільова функція набуває оптимального значення. Розвитку набули підходи оптимального керування як для детермінованих так і для стохастичних динамічних систем, з неперервними та дискретними значеннями часу. Значна частина підходів базується на методах варіаційного числення, методах динамічного програмування, описаних Річардом Беллманом, та принципі максимуму Понтрягіна.

У цій роботі розглядається оптимальне керування виключно детермінованими процесами та чисельні методи з принципом Понтрягіна [] в їх основі.

1.1 Постановка задачі

Розглянемо задачу оптимального керування, що визначається цільовою функцією, та деякою динамічною системою, яка змінюється з часом і залежить від керування. Динамічна система визначається заданою ситсемою диференційних рівнянь. Якщо задача є коректною для використання принципу максимуму Понтрягіна, можна встановити систему з необхідних умов для максимізації функціоналу та визначити множину в просторі допустимих керувань, що дозволяє знайти оптимальний розв'язок. Відносно прості системи дозволяють знайти розв'язок аналітично, проте в загальному випадку системи з багатьох рівнянь з розмірністю змінної простору понад дві координати часто необхідно використати чисельні апроксимації для знаходження розв'язку. Реалізації алгоритмів для пошуку оптимального керування є актуальними на сьогодні. Існує велика кількість програмних пакетів та бібліотек, що на різних рівнях абстракції пропонують рішення задач оптимального керування. Основною проблемою створення універсального інструмента є індивідуальні підходи для вирішення кожного класу задач оптимального керування. Часто для розвязку подібних задач достатньо поєднати методи з пакетів лінійного та нелінійного програмування, а також чисельної апроксамиції диференційних систем методи скінченних різниць, скінченних елементів або скінченних об'ємів. Так компанія-розробник популярного середовища MathWorks не пропонує окремого пакета для написання рішень оптимального керування, але надає ряд статей та рекомендацій по розробці подібних програм з використанням вже існуючих пакетів [?], [?].

Розділ 2

Основна частина

2.1 Визначення принципу Понтрягіна

Введемо формальне означення задачі оптимального керування для динамічної системи в загальному вигляді.

Означення 2.1.0.1 *Максимізувати функціонал*

$$\mathcal{L}(u, x^u) = \int_0^T G(t, u(t), x^u(t))\, dt + \varphi(x^u(T)) \tag{2.1}$$

по $u \in K \subset L^2(0, T; \mathbb{R}^m)\,(T > 0)$*, де* x^u *є розв'язком наступної системи, що називається рівнянням стану системи*

$$\begin{cases} x'(t) = f(t, u(t), x(t)), \quad t \in (0, T) \\ x(0) = x_0 \end{cases} \tag{2.2}$$

Для задачі на мінімум можна переформулювати задачу наступним чином:

$$\textit{Мінімізувати}\, \{-\mathcal{L}(u, x^u)\} \;\; \textit{відносно}\; u \in K \subset L^2(0, T; \mathbb{R}^m)$$

Введемо означення принципу максимуму Понтрягіна. Для цього задамо необхідні умови для функцій задачі оптимального керування. Нехай функції задачі задовольняють наступні умови:

$$\begin{aligned} &G : [0, T] \times \mathbb{R}^m \times \mathbb{R}^N \to \mathbb{R} \\ &\varphi : \mathbb{R}^N \to \mathbb{R} \\ &f : [0, T] \times \mathbb{R}^m \times \mathbb{R}^N \to \mathbb{R}^N \end{aligned}$$

$$x_0 \in \mathbb{R}^N, m, N \in \mathbb{N}^*, \text{ and } K \subset L^2(0, T; \mathbb{R}^m)$$

Назвемо $u^* \in K$ назвемо оптимальним розв'язком задачі оптимального контролю, якщо $\mathcal{L}\left(u^*, x^{u^*}\right) \geq \mathcal{L}(u, x^u)$, для усіх $u \in K$. Пара $\left(u^*, x^{u^*}\right)$ називається оптимальною парою і $\mathcal{L}\left(u^*, x^{u^*}\right)$ називається оптимальним значенням функції вартості. Ми також називаємо (u^*, x^*) оптимальною парою якщо u^* є оптимальним керуванням і $x^* = x^{u^*}$. Нехай $u^* \in K$ є оптимальним задачі оптимального керування, тоді:

$$\int_0^T G\left(t, u^*(t), x^{u^*}(t)\right) dt + \varphi\left(x^{u^*}(T)\right) \geq \int_0^T G\left(t, u(t), x^u(t)\right) dt + \varphi\left(x^u(T)\right) \tag{2.3}$$

для усіх $u \in K$. Далі вважатимемо, що для функцій задачі виконуються наступні умови:

$$\begin{cases} f_u = \frac{\partial f}{\partial u}, \quad f_x = \frac{\partial f}{\partial x} \\ G_u = \frac{\partial G}{\partial u}, \quad G_x = \frac{\partial G}{\partial x} \\ \varphi_x = \frac{\partial \varphi}{\partial x} \end{cases}$$

Розглянемо множину V.

$$V = \left\{v \in L^2\left(0, T; \mathbb{R}^m\right); u^* + \varepsilon v \in K \text{ для усіх достатньо малих } \varepsilon > 0\right\}$$

Введемо функцію z як диференціал від x по v: $z = dx^{u^*}(v)$. Оскільки за визначенням повного диференціалу:

$$dz = z'_x dx + z'_y dy$$

функція z є розв'язком наступної системи:

$$\begin{cases} z'(t) = f_u\left(t, u^*(t), x^{u^*}(t)\right) v(t) + f_x\left(t, u^*(t), x^{u^*}(t)\right) z(t), \quad t \in (0, T) \\ z(0) = 0 \end{cases} \tag{2.4}$$

З умови оптимальності **??** випливає нерівність:

$$\begin{aligned} \int_0^T G\left(t, u^*(t), x^{u^*}(t)\right) dt + \varphi\left(x^{u^*}(T)\right) \geq \int_0^T G\left(t, u^*(t) + \varepsilon v(t), x^{u^*+\varepsilon v}(t)\right) dt \\ +\varphi\left(x^{u^*+\varepsilon v}(T)\right), \end{aligned} \tag{2.5}$$

Перегрупуємо нерівність **??** та поділом на ε:

$$\begin{aligned} &\int_0^T \frac{1}{\varepsilon}\left[G\left(t, u^*(t) + \varepsilon v(t), x^{u^*+\varepsilon v}(t)\right) - G\left(t, u^*(t), x^{u^*}(t)\right)\right] dt \\ &+ \frac{1}{\varepsilon}\left[\varphi\left(x^{u^*+\varepsilon v}(T)\right) - \varphi\left(x^{u^*}(T)\right)\right] \leq 0 \end{aligned}$$

Оскільки можемо обирати ε як завгодно малим, то перейдемо до диференціалів. Формула повного диференціала функції G:

$$dG = G'_u du + G'_x dx$$

Враховуючи значення диференціалів описані вище, перейдемо до наступного вигляду:

$$\int_0^T \left[v(t) \cdot G_u\left(t, u^*(t), x^{u^*}(t)\right) + z(t) \cdot G_x\left(t, u^*(t), x^{u^*}(t)\right)\right] dt$$
$$+ z(T) \cdot \varphi_x\left(x^{u^*}(T)\right) \leq 0$$

Далі розглянемо так звану спряжену проблему. p - це спряжений стан. Це необхідна умова першого порядку у визначенні принципу Понтрягіна.

$$\begin{cases} p'(t) = -f_x^*\left(t, u^*(t), x^{u^*}(t)\right) p(t) - G_x\left(t, u^*(t), x^{u^*}(t)\right), \quad t \in (0, T) \\ p(T) = \varphi_x\left(x^{u^*}(T)\right) \end{cases} \tag{2.6}$$

Якщо помножити рівняння **??** на p та проінтегрувати частинами, то отримаємо рівність:

$$z(T) \cdot p(T) - \int_0^T z(t) \cdot p'(t) dt$$
$$= \int_0^T \left[f_u\left(t, u^*(t), x^{u^*}(t)\right) v(t) + f_x\left(t, u^*(t), x^{u^*}(t)\right) z(t)\right] \cdot p(t) dt$$

Далі підставляємо p, за визначенням зі спряженої задачі:

$$\begin{aligned} & z(T) \cdot \varphi_x\left(x^{u^*}(T)\right) \\ & + \int_0^T z(t) \cdot \left[f_x^*\left(t, u^*(t), x^{u^*}(t)\right) p(t) + G_x\left(t, u^*(t), x^{u^*}(t)\right)\right] dt \\ & = \int_0^T \left[v(t) \cdot f_u^*\left(t, u^*(t), x^{u^*}(t)\right) p(t) + z(t) \cdot f_x^*\left(t, u^*(t), x^{u^*}(t)\right) p(t)\right] dt \end{aligned} \tag{2.7}$$

Можна звести **??** до вигляду:

$$\int_0^T z(t) \cdot G_x\left(t, u^*(t), x^{u^*}(t)\right) dt + z(T) \cdot \varphi_x\left(x^{u^*}(T)\right)$$
$$= \int_0^T v(t) \cdot f_u^*\left(t, u^*(t), x^{u^*}(t)\right) p(t) dt$$

$$\int_0^T v(t) \cdot \left[G_u\left(t, u^*(t), x^{u^*}(t)\right) + f_u^*\left(t, u^*(t), x^{u^*}(t)\right) p(t)\right] dt \leq 0$$

$$G_u\left(\cdot, u^*, x^{u^*}\right) + f_u^*\left(\cdot, u^*, x^{u^*}\right) p \in N_K\left(u^*\right)$$

Це також необхідна умова першого порядку у визначенні принципу Понтрягіна. Ця множина є нормальним конусом. Далі розглянемо схему доведення існування оптимального за Понтрягіним контролю. Також визначимо принцип Понтрягіна через гамільтоніан Умови трансверсальності для принципу максимуму Понтрягіна

2.1.1 Схема довдення існування оптимального керування

Нехай

$$d = \sup_{u \in K} \mathcal{L}\left(u, x^u\right) \in \mathbb{R}.$$

Для усіх $n \in \mathbb{N}^*$, існує $u_n \in K$ такий, що:

$$d - \frac{1}{n} < \mathcal{L}\left(u_n, x^{u_n}\right) \leq d$$

Крок 1. Довести, що існує підпослідовність $\{u_{n_k}\}$, така що:

$$u_{n_k} \longrightarrow u^* \text{ слабко збігається в } L^2\left(0, T; \mathbb{R}^m\right)$$

Крок 2. Довести, що існує підпослідовність $\{x^{u_{n_r}}\}$ послідовності $\{x^{u_{n_k}}\}$, що збігається до x^{u^*} на $C\left([0, T]; IR^N\right)$. Крок 3. З нерівності:

$$d - \frac{1}{n_r} < \mathcal{L}\left(u_{n_r}, x^{u_{n_r}}\right) \leq d$$

Переходимо границі і отрмаємо:

$$\mathcal{L}\left(u^*, x^{u^*}\right) = d$$

u^* - оптимальне керування задачі.

2.2 Градієнтний метод для оптмального керування

Розглянемо задачу оптимального керування:

$$\text{Максимізувати } \Phi(u) = \mathcal{L}\left(u, x^u\right) = \int_0^T G\left(t, u(t), x^u(t)\right) dt \tag{2.8}$$

відносно $u \in K \subset U = L^2\left(0, T; \mathbb{R}^m\right)(T > 0)$, де x^u є розв'язком рівняння стану:

$$\begin{cases} x'(t) = f(t, u(t), x(t)), \quad t \in (0, T) \\ x(0) = x_0 \end{cases} \tag{2.9}$$

Переформулюємо як задачу пошуку мінімуму:

$$\text{Мінімізувати } \Psi(u) = -\mathcal{L}\left(u, x^u\right)$$

$$\Psi(u) = -\Phi(u)$$

Використаємо метод проекції градієнта для пошуку мінімуму.
Крок 1. Обираємо довільний вектор керування $u^{(0)} \in K$, присвоюємо $k := 0$.
Крок 2. Шукаємо розвязок $x^{(k)}$ з рівняння стану:

$$\begin{cases} x'(t) = f\left(t, u^{(k)}(t), x(t)\right), \quad t \in (0, T) \\ x(0) = x_0 \end{cases}$$

Крок 3. Шукаємо спряжений стан $p^{(k)}$ зі спряженої системи:

$$\begin{cases} p'(t) = -f_x^*\left(t, u^{(k)}(t), x^{(k)}(t)\right) p(t) - G_x\left(t, u^{(k)}(t), x^{(k)}(t)\right), \quad t \in (0, T) \\ p(T) = 0. \end{cases}$$

Крок 4. Обчислюємо градієнт функції втрат:

$$w^{(k)} := \Phi_u\left(u^{(k)}\right) = G_u\left(\cdot, u^{(k)}, x^{(k)}\right) + f_u^*\left(\cdot, u^{(k)}, x^{(k)}\right) p^{(k)}.$$

Крок 5. Визначаємо розмір кроку градієнтного спуску $\rho_k \geq 0$:

$$\Phi\left(P_K\left(u^{(k)} + \rho_k w^{(k)}\right)\right) = \max_{\rho \geq 0} \Phi\left(P_K\left(u^{(k)} + \rho w^{(k)}\right)\right)$$

Крок 6.

$$u^{(k+1)} := P_K\left(u^{(k)} + \rho_k w^{(k)}\right)$$

Критерії зупинки.

$$\text{Якщо } \left\|u^{(k+1)} - u^{(k)}\right\| < \varepsilon$$

$u^{(k+1)}$ є шуканаю апроксимацією оптимального керування . В іншому разі виконуємо присвоєння $k := k + 1$ і повертаємось до кроку 2.

P_K - оператор проекції градієнта на опуклу множину K. Метод проекції градієнта враховує можливість додаткових обмежень на множині допустимого оптимального керування U.

$$P_K : U \to K$$

Якщо оптимальне керування не є обмеженим на U, то крок 5 набуває вигляду:

$$\Phi\left(u^{(k)} + \rho_k w^{(k)}\right) = \max_{\rho \geq 0}\left\{\Phi\left(u^{(k)} + \rho w^{(k)}\right)\right\}$$

Алгоритм у форматі псевдокоду приведений нижче:

Метод Арміджіо

while Евклідова норма вектора градієнта строго перевищує ε або крок градієнта строго перевищує ρ_j **do**
    Розв'язати рівняння стану відповідно до поточних значень функції керування $currentU$;
    Розв'язати рівняння стану відповідно до поточних значень функції керування $currentU$ та стану x;
    Обчислити градієнт функції втрат з урахуванням обчислених функцій $currentU$, x та p;
    if Евклідова норма градієнта менша ε **then**
        break
    end if
    Встановити початкове значення кроку градієнта $currentGradientStep = initGradientStep$;
    for $gradientIteration = 1, 2, \ldots, MaxGradientIteration$ **do**
        Оновити функцію керування $newU = prevU - currentGradientStep * currentGradient$
        Розв'язати рівняння стану відповідно до поточних значень функції керування $newU$;
        Розв'язати рівняння стану відповідно до поточних значень функції керування $newU$ та стану x;
        Обчислити значення функції втрат $newCost$
        if $newCost >= prevCost$ **then**
            Поправка кроку $currentGradientStap = currentGradientStap * gradientAdjustment$
        else
            break
        end if
    end for
end while

2.3 Програмна реалізація методу Арміджіо мовою Python

У даній роботі методі Арміджіо було реалізовно на Python3 з використанням пакетів NumPy та SciPy. Пакет NumPy надає функціонал для роботи з багатовимірними масивами даних, функції лінійної алгебри. Усі операції ефективно реалізовано мовою C, з використанням векторних обчислень. Завдяки цьому, NumPy часто використовується як основа для розробки інших математичних пакетів. SciPy містить велику кількість методів оптимізації та математичної статистики, а також функції для пошуку розв'язків рівнянь. У даній роботі пакет SciPy використовується для перевірки реалізованих методів Рунге-Кутта, усі матриці задано як масиви NumPy. Метод Арміджіо реалізовано з використанням принципів ООП (обєктно-орієнтоване програмування). Класи організовано ієрархічно: найстарший клас $PMPProjectedGradientSolver$ визначає функцію $gradientDescentLoop$, в якій міститься сам алгоритм Арміджіо. Класи $PMPODESolver$ та $PMPPDESolver$ наслідують поведінку базового, та реалізовують інші спеціальні методи.

2.4 Задача оптимального споживання. Приклад аналітичного пошуку оптимального керування

Розглянемо задачу оптимального керування споживанням з [?] Розглянемо наступну задачу:

$$x(t) = I(t) + C(t), \quad t \geq 0 \tag{2.10}$$

Де $x(t)$ - це кількість виробництва, $I(t)$ - кількість інвестицій, $C(t)$ - кількість споживання. Введемо керування $u(t) \in [0, 1]$, що позначає частку виробництва, що переходить у інвестиції.

$$I(t) = u(t)x(t) \tag{2.11}$$

Тоді

$$C(t) = (1 - u(t))x(t), \quad t \geq 0 \tag{2.12}$$

Введемо рівняння стану, як простий випадок коли, приріст виробництва пропорційний інветиціям:

$$x'(t) = \gamma u(t)x(t) \tag{2.13}$$

де $\gamma \in (0, +\infty)$. Введемо функцію корисності $F(C)$ і інтеграл "добробуту":

$$\int_0^T e^{-\delta t} F(C(t))dt \tag{2.14}$$

Розглянемо простий випадок моделі, коли $F(C) = C$ і $\delta = 0$. Тоді інтеграл добробуту

$$\int_0^T C(t)dt = \int_0^T (1 - u(t))x(t)dt. \tag{2.15}$$

Сформулюємо задачу оптимального керування

$$\text{Максимізувати} \int_0^T (1 - u(t))x^u(t)dt \tag{2.16}$$

Де $u \in L^2(0, T), 0 \leq u(t) \leq 1$, $t \in (0, T)$, $x^u(t)$ є розв'язком рівняння стану:

$$\begin{cases} x'(t) = \gamma u(t)x(t), & t \in (0, T) \\ x(0) = x_0 > 0 \end{cases} \tag{2.17}$$

Аналітичний розв'язок рівняння стану має вигляд:

$$x^u(t) = x_0 \exp\left(\int_0^t \gamma u(s)ds\right), \quad t \in [0, T]. \tag{2.18}$$

У позначеннях з означення принципу Понтрягіна:

$$G(t,u,x) = (1-u)x$$
$$\varphi(x) = 0$$
$$f(t,u,x) = \gamma u x$$

$$K = \{w \in L^2(0,T); 0 \le w(t) \le 1 \text{ a.e. } t \in (0,T)\}$$

Існування оптимальної пари Доведемо існування оптимальної пари: Позначимо

$$\Phi(u) = \int_0^T (1-u(t))x^u(t)dt, \quad u \in K \tag{2.19}$$

Введемо супремум

$$d = \sup_{u \in K} \Phi(u)$$

Оскільки $u \in K$, то

$$0 < x^u(t) \le x_0 e^{\gamma t}, \quad t \in [0,T]$$

Далі отримуємо, що

$$0 \le \Phi(u) = \int_0^T (1-u(t))x^u(t)dt \le x_0 T e^{\gamma T}$$

Отже $d \in [0,+\infty)$. Для усіх $n \in \mathbb{N}^*$ існує $u_n \in K$ таке, що

$$d - \frac{1}{n} < \Phi(u_n) \le d$$

K є обмеженою підмножиною на $L^2(0,T)$. Отже існує підпослідовність $\{u_{n_k}\}_{k \in \mathbb{N}^*}$, така що

$$u_{n_k} \longrightarrow u^* \quad \text{слабо збігається на } L^2(0,T)$$

$$d - \frac{1}{n_k} < \int_0^T \left(1 - u_{n_k}(t)\right) x^{u_{n_k}}(t)dt \le d \quad \text{for any } k \in \mathbb{N}^*$$

Перейдемо до границі по d

$$d = \int_0^T \left(1 - u^*(t)\right) x^{u^*}(t)dt$$

Отже, існує оптимальна пара для задачі. Принцип максимум для задачі максимізації споживання Для будь якого фіксованого $v \in V = \{w \in L^2(0,T); u^* + \varepsilon w \in K$ для будь-якого достатньо малого $\varepsilon > 0\}$ позначимо як z розв'язок задачі Коші:

$$\begin{cases} z'(t) = \gamma u^*(t)z(t) + \gamma v(t)x^*(t), \quad t \in (0,T) \\ z(0) = 0 \end{cases}$$

Можемо отримати аналітичний розв'язок задачі:

$$z(t) = \int_0^t \exp\left\{\int_s^t \gamma u^*(\tau)d\tau\right\}\gamma v(s)x^*(s)ds, \quad t \in [0,T].$$

За визначенням оптимальної пари:

$$\int_0^T (1-u^*(t))\,x^*(t)dt \geq \int_0^T (1-u^*(t)-\varepsilon v(t))\,x^{u^*+\varepsilon v}(t)dt$$

для будь-якого достатньо малого $\varepsilon > 0$, маємо що:

$$\int_0^T \left[(1-u^*(t))\,\frac{x^{u^*+\varepsilon v}(t)-x^*(t)}{\varepsilon} - v(t)x^{u^*+\varepsilon v}(t)\right]dt \leq 0$$

Доведемо, що

$$x^{u^*+\varepsilon v} \longrightarrow x^* \quad \text{на } C([0,T])$$

та

$$\frac{x^{u^*+\varepsilon v}-x^*}{\varepsilon} \longrightarrow z \quad \text{in}\, C([0,T])$$

Для будь-якого достатньо малого $\varepsilon > 0$, маємо що:

$$x^{u^*+\varepsilon v}(t) = x_0 \exp\left\{\gamma\int_0^t (u^*(s)+\varepsilon v(s))\,ds\right\} = x^{u^*}(t)\exp\left\{\varepsilon\gamma\int_0^t v(s)ds\right\}, \quad t \in [0,T],$$

Отже,

$$\left|x^{u^*+\varepsilon v}(t) - x^{u^*}(t)\right| = \left|x^{u^*}(t)\right| \cdot \left|\exp\left\{\varepsilon\gamma\int_0^t v(s)ds\right\} - 1\right|, \quad t \in [0,T].$$

Оскільки,

$$\left|\exp\left\{\varepsilon\gamma\int_0^t v(s)ds\right\} - 1\right| \longrightarrow 0$$

можна заключити, що

$$x^{u^*+\varepsilon v} \longrightarrow x^* \quad \text{на } C([0,T])$$

Для будь-якого достатньо малого $\varepsilon > 0$, маємо що:

$$w_\varepsilon(t) = \frac{x^{u^*+\varepsilon v}-x^*}{\varepsilon} - z(t), \quad t \in [0,T].$$

є розв'язком для задачі

$$\begin{cases} w'(t) = \gamma u^*(t)w(t) + \gamma v(t)\left[x^{u^*+\varepsilon v}(t) - x^{u^*}(t)\right], & t \in (0,T) \\ w(0) = 0 \end{cases}$$

Аналітичний розв'язок цієї задачі має вигляд:

$$w_\varepsilon(t) = \gamma\int_0^t \exp\left\{\gamma\int_s^t u^*(\tau)d\tau\right\}v(s)\left[x^{u^*+\varepsilon v}(s) - x^{u^*}(s)\right]ds, \quad t \in [0,T]$$

Можна побачити, що

$$w_\varepsilon \longrightarrow 0 \quad \text{на } C([0,T])$$

а отже

$$\frac{x^{u^*+\varepsilon v} - x^*}{\varepsilon} \longrightarrow z \quad \text{на } C([0,T])$$

Тому отримуємо, що

$$\int_0^T \left[(1-u^*(t))\, z(t) - v(t)x^*(t)\right] dt \leq 0$$

Розглянемо спряжену задачу

$$\begin{cases} p'(t) = -\gamma u^*(t)p(t) + u^*(t) - 1, \quad t \in (0,T) \\ p(T) = 0. \end{cases}$$

Аналітичний розв'язок:

$$p(t) = -\int_t^T \exp\left\{\int_t^s \gamma u^*(\tau)d\tau\right\}(u^*(s)-1)\, ds, \quad t \in [0,T].$$

Використовуючи рівняння вище, можемо прийти до:

$$\int_0^T (1-u^*(t))\, z(t)dt = \int_0^T \gamma v(t)x^*(t)p(t)dt$$

Отже

$$\int_0^T x^*(t)(\gamma p(t)-1)v(t)dt \leq 0$$

для усіх $v \in V$. Це еквівалентно

$$(\gamma p - 1)x^* \in N_K(u^*)$$

Отже

$$u^*(t) = \begin{cases} 0 \text{ if } \gamma p(t) - 1 < 0 \\ 1 \text{ if } \gamma p(t) - 1 > 0 \end{cases}$$

2.5 Керування запасами

Розглянемо задачу керування запасами з книги [?]. Постановка задачі. Розглянемо наступну задачу. Нехай $x(t)$ - запас деякого ресурсу компанії. $u(t)$ - виробництво ресурсу, в моменту часу, $g(t)$ - частка ресурсу, що має бути продана за контрактом.

$$\begin{cases} x'(t) = u(t) - g(t), \quad t \in (0,T) \\ x(0) = x_0 \in \mathbb{R}. \end{cases}$$

Тоді запас ресурсу визначається наступним інтегралом:

$$x(t) = x_0 + \int_0^t [u(s) - g(s)]ds.$$

Можливі наступні випадки: Якщо $x(t) \geq 0$ то немає затримки виходу на ринок товару в момент t. Якщо $x(t) > 0$ то компанія має надлишкові запаси ресурсу. Якщо $x(t) < 0$ є затримка постачання ресурсу компанією в момент t. Команія повинна випустити $|x(t)|$ продукту. Введемо витрати пов'язані з цими випадками:

$$\psi(x) = \begin{cases} c_1 x^2, & x \geq 0 \\ c_2 x^2, & x < 0 \end{cases}$$

Введемо витрати компанії на виробництво товару:

$$\Phi(u) = \int_0^T \left[au(t) + \psi\left(x^u(t)\right) \right] dt$$

$a > 0$ - вартість виробництва одиниці товару. Задача очевидно наступна:

$$\text{Мінімізувати } \Phi(u)$$

відносно $u \in K = \{w \in L^2(0,T); u_1 \leq u(t) \leq u_2$ на усій множині $t \in (0,T)\}$ З очевидних міркувань:

$$u_1 \leq g(t) \leq u_2, \quad t \in [0,T]$$

Принцип Понтрягіна Запишемо рівняння спряженого стану системи:

$$\begin{cases} p'(t) = -\psi'\left(x^u(t)\right), & t \in (0,T) \\ p(T) = 0 \end{cases}$$

Для довільних $u, V \in U$ і $\varepsilon \in \mathbb{R}^*$ маємо

$$\frac{1}{\varepsilon}[\Phi(u+\varepsilon v) - \Phi(u)] = \int_0^T \frac{1}{\varepsilon}\left[\psi\left(x^{u+\varepsilon v}(t)\right) - \psi\left(x^u(t)\right)\right] dt + a \int_0^T v(t) dt.$$

Розглянемо рівняння:

$$\begin{cases} z'(t) = v(t), & t \in (0,T) \\ z(0) = 0 \end{cases}$$

Можна показати, що

$$x^{u+\varepsilon v} \longrightarrow x^u \quad \text{in}\, C([0,T])$$

і

$$\frac{1}{\varepsilon}\left(x^{u+\varepsilon v} - x^u\right) \longrightarrow z \quad \text{in } C([0,T])$$

$$\left(v, \Phi_u(u)\right)_{L^2(0,T)} = a \int_0^T v(t) dt + \int_0^T \psi'\left(x^u(t)\right) z(t) dt$$

$$\int_0^T \left(p^u\right)'(t) z(t) dt = -\int_0^T \psi'\left(x^u(t)\right) z(t) dt$$

$$\int_0^T \psi'\left(x^u(t)\right) z(t) dt = \int_0^T p^u(t) v(t) dt$$

В результаті отримуємо рівність:

$$(v, \Phi_u(u))_{L^2(0,T)} = \int_0^T v(t) \left[p^u(t) + a\right] dt$$

Можемо зробити висновок, що

$$\Phi_u(u) = p^u + a$$

Алгоритм проекції градієнта За методом проекції градієнта описаного в, запишемо алгоритм для задачі вище: Крок 1. Оберемо $u^{(0)} \in K$ як початкове припущення про оптимальне керування. Виконаємо присвоєння j := 0. Крок 2. Обчислимо $x(t)$

$$\begin{cases} x'(t) = u^{(j)}(t) - g(t), & t \in (0,T) \\ x(0) = x_0 \end{cases}$$

Крок 3. Обчислимо $p(t)$

$$\begin{cases} p'(t) = -\psi'\left(x^{(j)}(t)\right), & t \in (0,T) \\ p(T) = 0 \end{cases}$$

Крок 4. Обчислимо градієнт:

$$w^{(j)} := \Phi_u\left(u^{(j)}\right) = p^{(j)} + a$$

Якщо $\left\|\Phi_u\left(u^{(j)}\right)\right\| < \varepsilon$, то $u^{(j)}$ є шуканою апроксимацією оптимального керування. В іншому випадку перейти до кроку 5. Крок 5. Обчислити значення кроку градієнтного спуску з рівності:

$$\Phi\left(P_K\left(u^{(j)} - \rho_j w^{(j)}\right)\right) = \min_{\rho \geq 0} \left\{\Phi\left(P_K\left(u^{(j)} - \rho w^{(j)}\right)\right)\right\}$$

Крок 6. Обчислити нове значення оптимального керування.

$$u^{(j+1)} := P_K\left(u^{(j)} - \rho_j w^{(j)}\right)$$

Виконати присвоєння $j := j + 1$ і перейти до кроку 2. Оператор проекції $P_K : L^2(0,T) \to K$ визначаємо як функцію:

$$P_K(u)(t) = \text{Proj}(u(t)) \text{ для усіх } t \in (0,T)$$

$$\text{Proj}(w) = \begin{cases} w & \text{якщо } u_1 \leq w \leq u_2 \\ u_1 & \text{якщо } w < u_1 \\ u_2 & \text{якщо } w > u_2 \end{cases}$$

Розв'язок задачі

$$g(t) = \begin{cases} g_1 t + g_2, & t \in [0, T/2] \\ g_3 - g_4 t, & t \in (T/2, T] \end{cases}$$

$$g_1 = g_4 = \frac{2}{T}(u_2 - u_1), \quad g_2 = u_1, \quad g_3 = 2u_2 - u_1$$

Результати чисельної апроксимації методом Арміджіо для набору параметрів $x(0) = 5, c_1 = 4e-3, c_2 = 1e-3, g_1 = g_4 = 1, g_2 = 10,$ і $g_3 = 22, u_1 = 10, u_2 = 16, u(0) = 14$ можна знайти у розділі **??**

2.6 Логістична модель Фішера

Логістичне рівняння Фішера ([**?**]) Модель має характер рівняння дифузії-реакції.

$$\begin{cases} \frac{\partial y}{\partial t} - \gamma \Delta y = ry\left(1 - \frac{y}{k}\right) - m(x)u(x,t)y(x,t), & (x,t) \in Q_T \\ \frac{\partial y}{\partial \nu}(x,t) = 0, & (x,t) \in \Sigma_T \\ y(x,0) = y_0(x), & x \in \Omega \end{cases}$$

У загальному випадку задача розглядається в багатовимірному просторі. У конкретному випадку вважаємо, що потік на краях рівний нулю, функція розподілу популяції по простору задана як $y_0(x)$. Візьмемо $k = 1$ - "пропускна здатність"середовища в сенсі можливості проживання певної кількості виду на одиниці площі. Природній приріст $r > 0$ будемо розглядати в діапазоні $[0;1]$. Коефіцієнт γ будемо вважати рівним 1.

$$m(x)u(x,t) = \begin{cases} u(x,t), x \in \omega, & t \in (0,T) \\ 0, \quad x \in \Omega \backslash \bar{\omega}, & t \in (0,T) \end{cases}$$

2.7 Оптимальний контроль для рівняння дифузії-реакції на прикладі логістичної моделі Фішера

Задача оптимального керування для моделі Фішера Формулювання задачі Розглянемо задачу на ареалі Ω, обмеженому квадратом:

$$\Omega = \{(x_1, x_2) \in \Omega : 0 \leq x_1 \leq L; 0 \leq x_2 \leq L\}$$

Нехай ареал застосування керування визначається підпростором ω. У прикладі чисельної апроксимації на двовимірному просторі будемо розглядати цей підпростір як коло із заданими параметрами.

$$\omega = \left\{(x_1, x_2) \in \Omega : (x_1 - a)^2 + (x_2 - b)^2 = R^2\right\}$$

2.8 Оптимальне вирощування

Оптимальне збір урожаю і модель дифузії-реакції Формулювання задачі Наступна модель Фішера описує динаміку популяції, що може переміщатись у деякому ареалі проживання Ω.

$$\begin{cases} \frac{\partial y}{\partial t} - \gamma \Delta y = ry\left(1 - \frac{y}{k}\right) - m(x)u(x,t)y(x,t), & (x,t) \in Q_T \\ \frac{\partial y}{\partial \nu}(x,t) = 0, & (x,t) \in \Sigma_T \\ y(x,0) = y_0(x), & x \in \Omega \end{cases}$$

Де $\Omega \in \mathbb{R}^N$ $(N \in \mathbb{N}^*)$; T, γ, r, k - додатні константи; $Q_T = \Omega \times (0,T), \Sigma_T = \partial\Omega \times (0,T)$ і $y_0 \in L^\infty(\Omega), y_0(x) > 0$ для усіх $x \in \Omega$ - початковий розподіл популяції в просторі. u - зусилля по збору урожаю, що застосовуються до деякої не порожньої множини популяції $\omega \subset \Omega$. m - характеристична функція для ω, така що:

$$m(x)u(x,t) = \begin{cases} u(x,t), x \in \omega, & t \in (0,T) \\ 0, \quad x \in \Omega \backslash \bar{\omega}, & t \in (0,T) \end{cases}$$

Загальна корисність від збору урожаю на $[0,T]$ визначається інтегралом:

$$\int_0^T \int_\omega u(x,t) y^u(x,t) dx dt$$

Отримуємо задачу оптимального керування:

$$\text{Максимізувати } \int_0^T \int_\omega u(x,t) y^u(x,t) dx dt$$

відносно $u \in K = \{w \in L^2(\omega \times (0,T)); 0 \leq w(x,t) \leq L \text{ a.e. } \}$ $(L > 0)$ Існування оптимального керування. Введемо позначення

$$\Phi(u) = \int_0^T \int_\omega u(x,t) y^u(x,t) dx dt, \quad u \in K$$

Нехай

$$d = \sup_{u \in K} \Phi(u)$$

$$0 < y^u(x,t) \leq y^0(x,t) \text{ a.e. } (x,t) \in \Omega \times (0,T)$$

а отже

$$0 \leq \int_0^T \int_\omega u(x,t) y^u(x,t) dx dt \leq L \int_0^T \int_\omega y^0(x,t) dx dt$$

Тоді отримуємо, що

$$d \in \mathbb{R}^+$$

Нехай $\{u_n\}_{n\in\mathbb{N}^*} \subset K$ - посліжовність контроллерів , що задовільняють умову:

$$d - \frac{1}{n} < \Phi(u_n) \leq d$$

(1) Оскільки $\{u_n\}_{n\in\mathbb{N}}$ є обмеженою на $L^2(\omega \times (0,T))$, тоді існує підпослідовність така , що

$$u_n \longrightarrow u^* \text{ слабо на } L^2(\omega \times (0,T))$$

і

$$mu_n \longrightarrow mu^* \text{ слабо на} L^2(\omega \times (0,T))$$

Позначимо

$$a_n(x,t) = ry^{u_n}(x,t)\left(1 - \frac{y^{u_n}(x,t)}{k}\right) - m(x)u_n(x,t)y^{u_n}(x,t), \quad (x,t) \in Q_T$$

Тоді початкова задача має вигляд:

$$\begin{cases} \frac{\partial y}{\partial t} - \gamma\Delta y = a_n(x,t), & (x,t) \in Q_T \\ \frac{\partial y}{\partial \nu}(x,t) = 0, & (x,t) \in \Sigma_T \\ y(x,0) = y_0(x), & x \in \Omega \end{cases}$$

$$\begin{cases} a_{n_k} \longrightarrow a^* & \text{weakly in } L^2(Q_T) \\ y^{u_{n_k}} \longrightarrow y^* & \text{a.e. in } Q_T \end{cases}$$

y^* - розв'язок наступної задачі:

$$\begin{cases} \frac{\partial y}{\partial t} - \gamma\Delta y = a^*(x,t), & (x,t) \in Q_T \\ \frac{\partial y}{\partial \nu}(x,t) = 0, & (x,t) \in \Sigma_T \\ y(x,0) = y_0(x), & x \in \Omega \end{cases}$$

$$a_{n_k} \longrightarrow ry^*\left(1 - \frac{y^*}{k}\right) - mu^*y^* \text{ in } L^2(Q_T)$$

оскільки $\{y^{u_n}\}_{n\in\mathbb{N}^*}$ обмежена на $L^\infty(Q_T)$ Отже

$$a^* = ry^*\left(1 - \frac{y^*}{k}\right) - mu^*y^* \text{ in } L^2(Q_T)$$

Якщо перейти до границі в (1) то отримуємо:

$$d = \Phi(u^*)$$

Принцип максимуму для задачі дифузіїї реакції Задачу можна переписати у формі початкової задачі:

$$\begin{cases} y'(t) = f(t, u(t), y(t)), \quad t \in (0, T) \\ y(0) = y_0, \end{cases}$$

де

$$f(t, u, y) = Ay + ry\left(1 - \frac{y}{k}\right) - muy$$

A - необмежений лінійний оператор, що визначається наступним чином:

$$D(A) = \left\{ w \in H^2(\Omega); \frac{\partial w}{\partial \nu} = 0 \text{ on } \partial\Omega \right\}$$

$$Ay = \gamma \Delta y, \quad y \in D(A).$$

Рівняння спряженого стану для задачі буде мати вигляд:

$$p'(t) = -A^* p - rp + \frac{2r}{k} y^u p + mu(1 + p), \quad t \in (0, T)$$

За теоремою доведеною в книжці маємо, що якщо x, x^u- оптимальна пара і p - спряжений стан, то:

$$\begin{cases} \frac{\partial p}{\partial t} + \gamma \Delta p = -rp + \frac{2r}{k} y^{u^*} p + mu^*(1 + p), & (x, t) \in Q_T \\ \frac{\partial p}{\partial \nu}(x, t) = 0, & (x, t) \in \Sigma_T \\ p(x, T) = 0, & x \in \Omega \end{cases}$$

$$u^*(x, t) = \begin{cases} 0, & \text{if } 1 + p(x, t) < 0 \\ L, & \text{if } 1 + p(x, t) > 0 \end{cases}$$

для усіх $(x, t) \in \omega \times (0, T)$ Рівняння спряженого стану можна звести до:

$$\begin{cases} \frac{\partial p}{\partial t} + \gamma \Delta p = -rp + mL(1 + p)^+, & (x, t) \in Q_T \\ \frac{\partial p}{\partial \nu}(x, t) = 0, & (x, t) \in \Sigma_T \\ p(x, T) = 0, & x \in \Omega. \end{cases}$$

Отже p не залежить від u, y і його можна апроксимувати окремо, а далі обчислити оптимальний контроль. Результати чисельної апроксимації для одновимірної задачі методом Арміджйо для набору параметрів $y(0) = 10, \gamma = 0.006, r = 0.01, k = 1, u_1 = 0, u_2 = 10, u(0) = 10$ можна знайти у розділі **??** Результати чисельної апроксимації для двовимірної задачі методом Арміджйо для набору параметрів $y(0) = 10, \gamma = 0.005, r = 0.01, k = 1, u_1 = 0, u_2 = 20, u(0) = 0$ можна знайти у розділі **??**

2.9 Чисельні результати

2.9.1 Задача керування запасами

Рис. 2.1: Оптимальне керування в задачі оптимального управління запасами

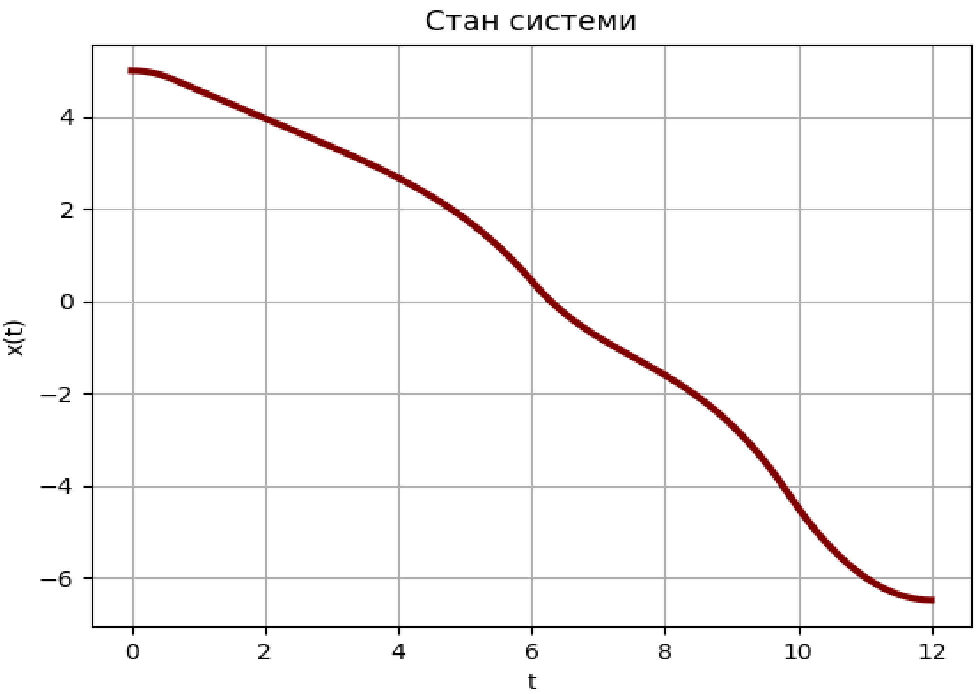

Рис. 2.2: Оптимальна траєкторія в задачі оптимального управління запасами

2.9.2 Приклад задачі з аналітичним рішенням

Розглянемо наступну задачу оптимального керування:

$$\text{Мінімізувати } \frac{1}{2}\int_0^1 \big(u^2(t) + x^2(t)\big)\, dt$$

, де $u \in L^2(0,1)$, $x = x^u$ - оптимальна траєкторія Рівняння стану має вигляд:

$$\begin{cases} x'(t) = u(t), \quad t \in (0,1) \\ x(0) = 1 \end{cases}$$

Можна встановити, що рівняння спряженого стану має вигляд:

$$\begin{cases} p'(t) = -x^u(t), \quad t \in (0,1) \\ p(1) = 0 \end{cases}$$

Градієнт функції втрат має вигляд:

$$\Phi_u(u) = u + p^u$$

Оптимальне керування задачі:

$$u^*(t) = -\frac{\sinh(1-t)}{\cosh(1)}$$

Оптимальна траєкторія:

$$x^*(t) = \frac{\cosh(1-t)}{\cosh(1)}$$

На цьому прикладі можна порівняти аналітичний розвязок та його чисельну апроксимацію методом Арміджіо

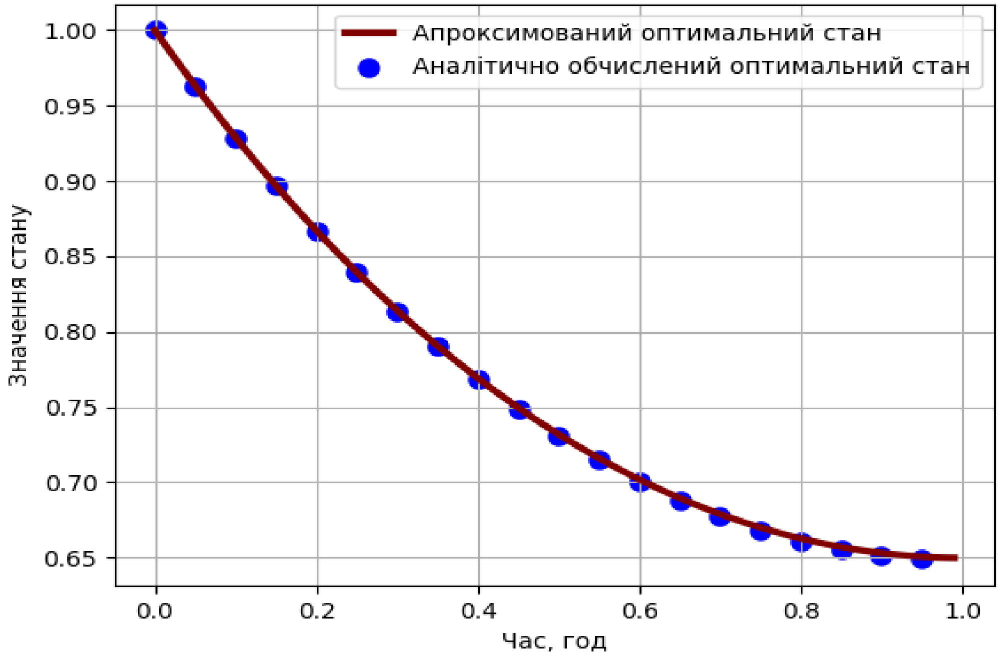

Рис. 2.3: Оптимальна траєкторія в задачі з розділу ???

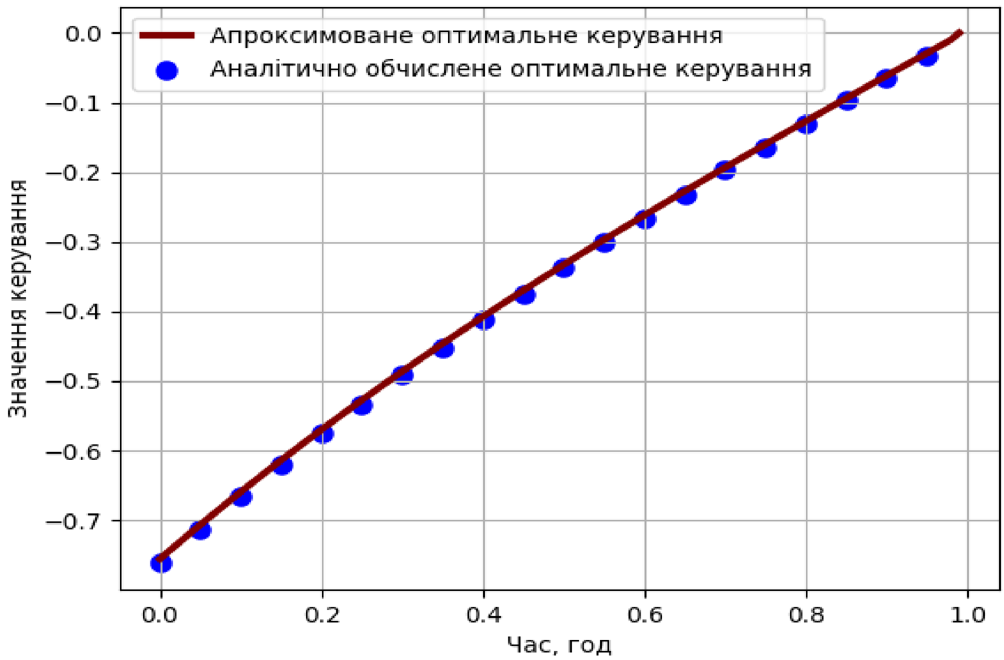

Рис. 2.4: Оптимальне керування в задачі в задачі з аналітичним рішенням

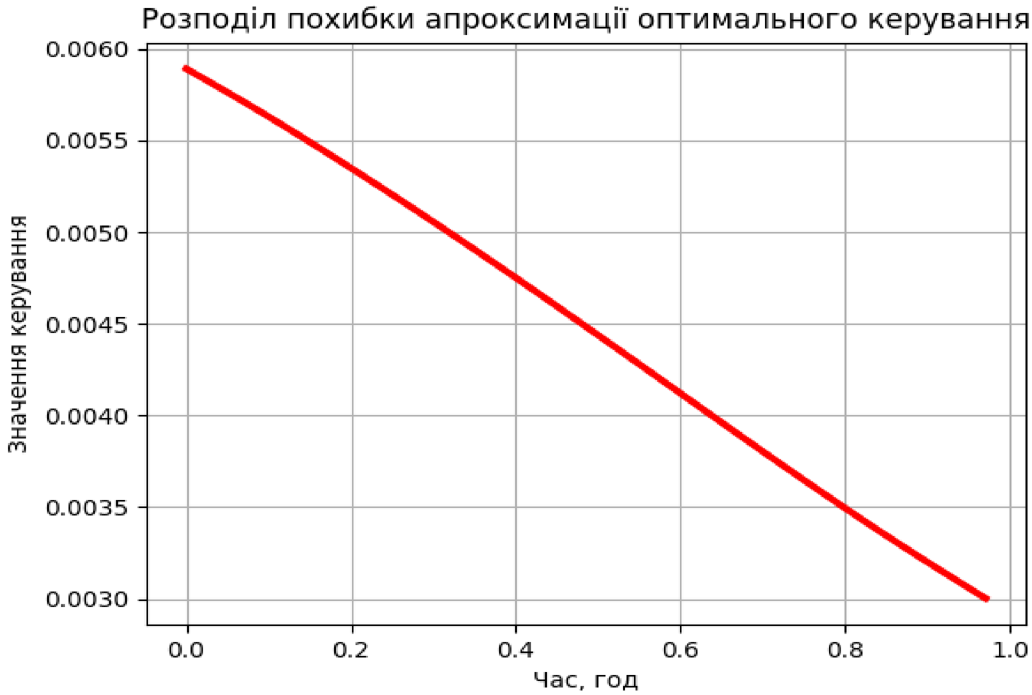

Рис. 2.5: Похибка в апроксимації оптимального керування в задачі з аналітичним рішенням

2.9.3 Одновимірна задача оптимального збору урожаю

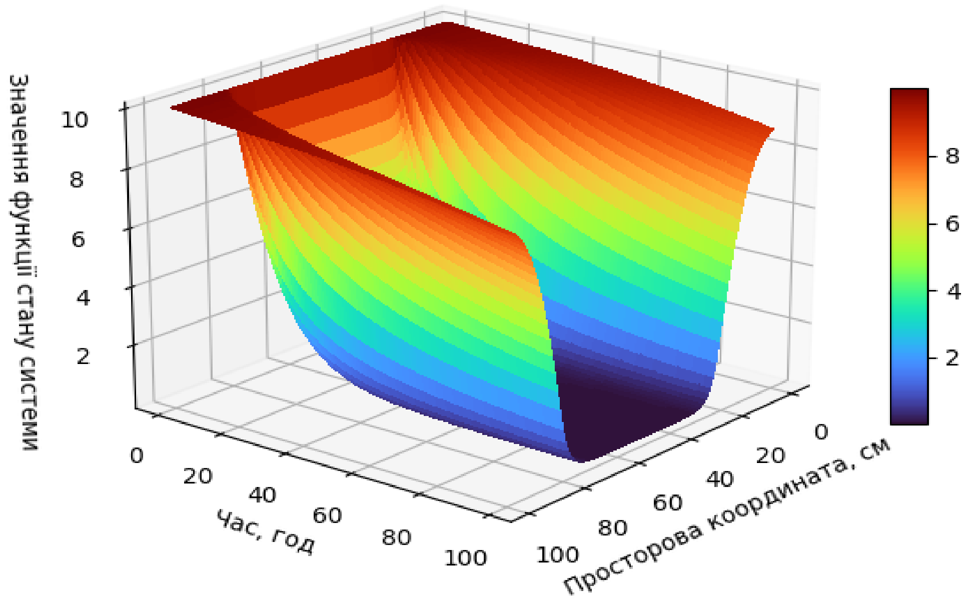

Рис. 2.6: Оптимальна траєкторія в задачі оптимального збору урожаю

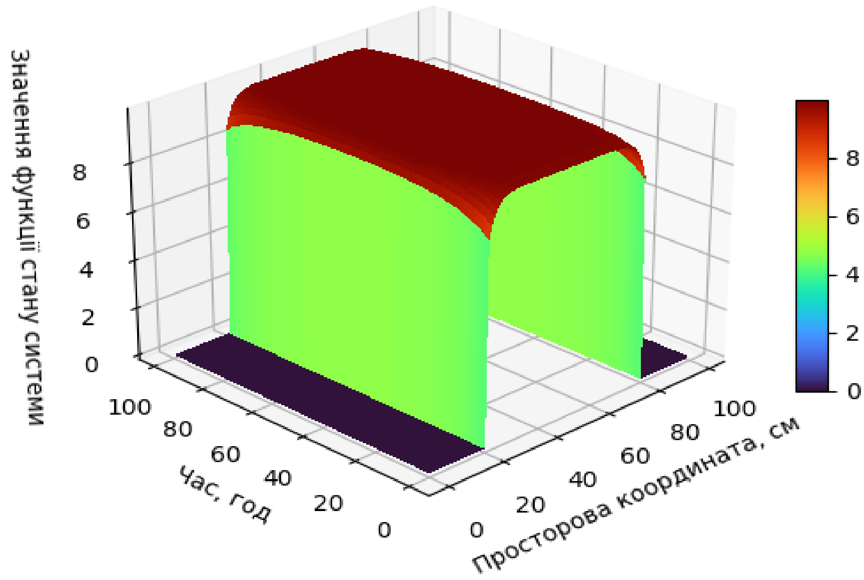

Рис. 2.7: Оптимальне керування в задачі оптимального збору урожаю

2.9.4 Двовимірна задача оптимального вирощування

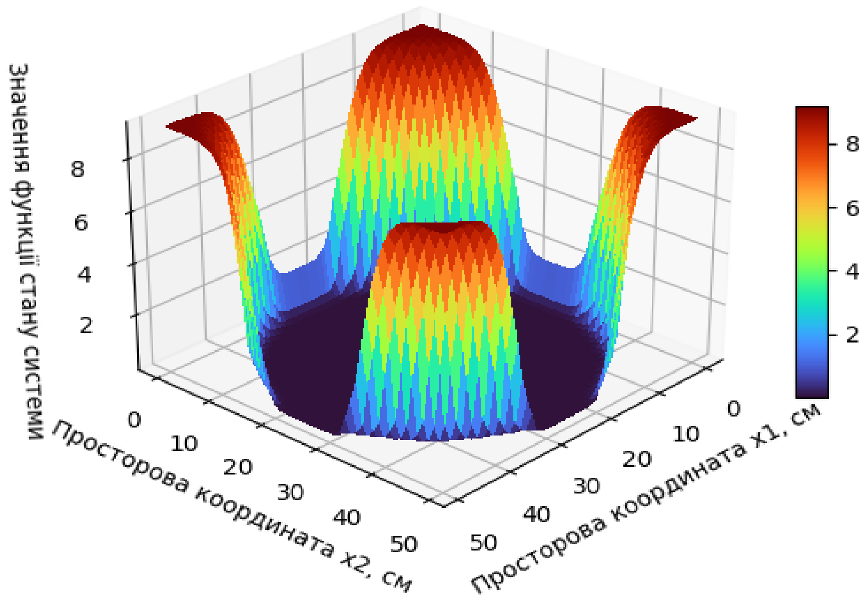

Рис. 2.8: Значення оптимальної траєкторії в кінцевий момент часу для двовимірної задачі оптимального збору урожаю

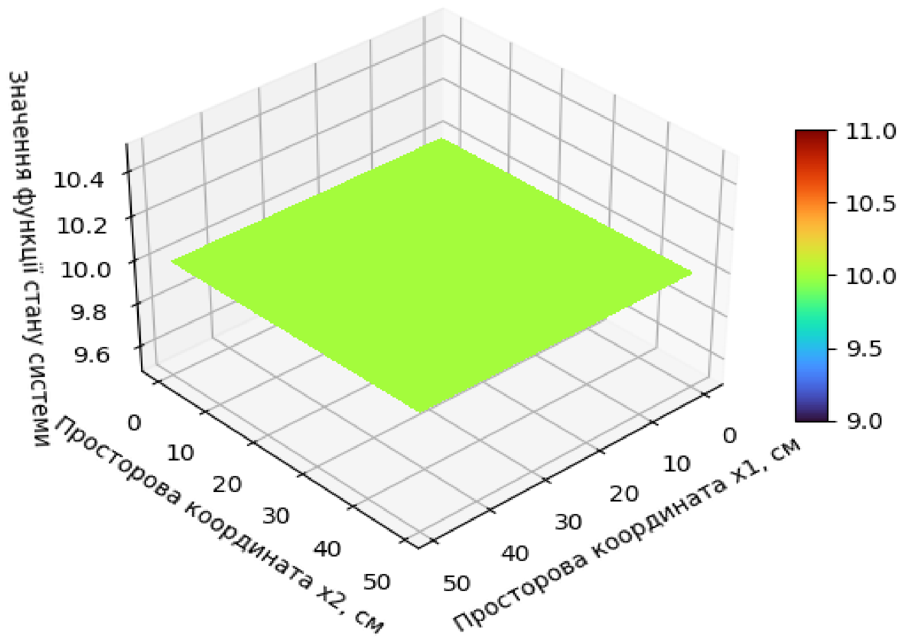

Рис. 2.9: Значення оптимальної траєкторії в початковий момент часу для двовимірної задачі оптимального збору урожаю

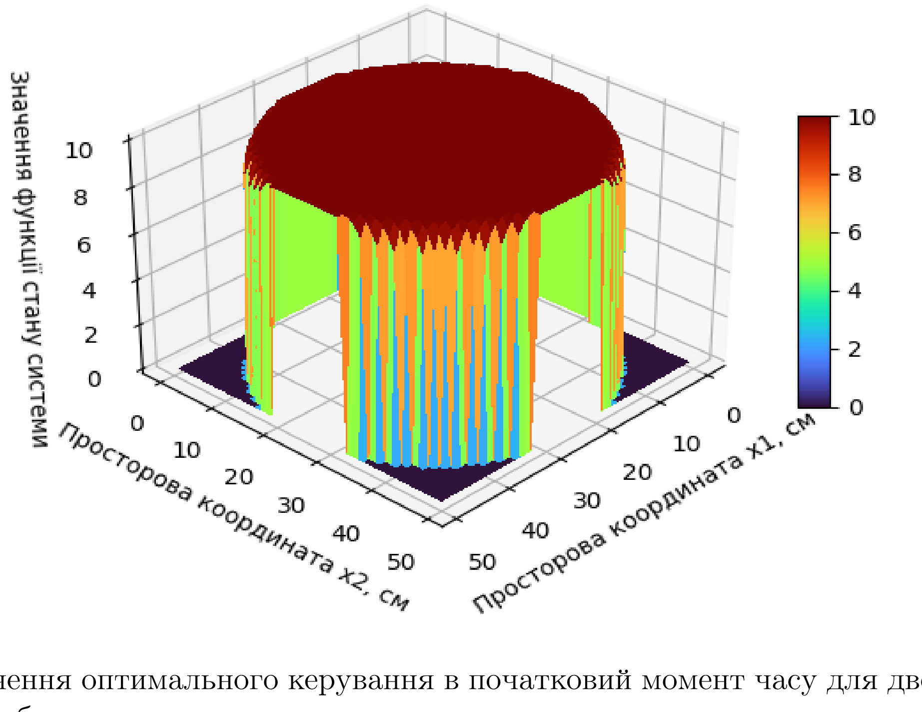

Рис. 2.10: Значення оптимального керування в початковий момент часу для двовимірної задачі оптимального збору урожаю

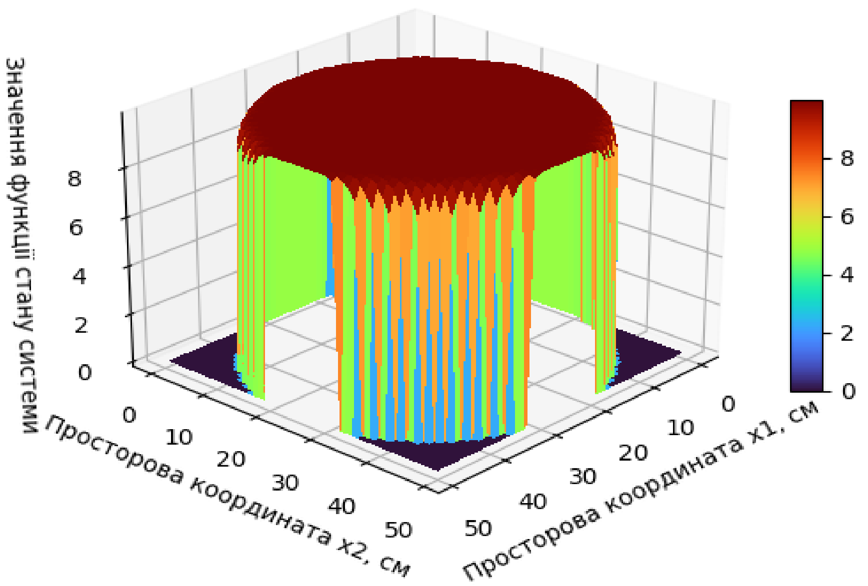

Рис. 2.11: Значення оптимального керування в кінцевий момент часу для двовимірної задачі оптимального збору урожаю

Розділ 3

Висновок

У даній роботі розглянуто ряд класичних задач оптимального керування динамічними система з чисельною реалізацією на мові Python з використанням пакетів з відкритим кодом NumPy та SciPy. Наведені приклади включають моделі зі звичайними рівняннями стану, а також рівняннями у частинних похідних. Результати наведені з використанням дво- та трьохвимірних графіків з пакету Matplotlib. Програманий код реалізовано з використанням парадигми об'єктно-орієнтованого програмування, що дозволяє перевикористовувати спільні частини алгоритму для різних типів задач. Ієрархія класів побудована таким чином, що метод Арміджіо визначено у найстаршому класі-усі класи нащадки визначають поведінку конкретної задачі, що дозволяє витратити незначні зусилля для розв'язку нових типів задач. Розробка програмних пакетів для задач оптимального керування залишається актуальною, оскільки кожна проблема є певною мірою індивідуальна і складно піддається узагальненням.

Бібліоґрафія

[1] https://www.mathworks.com/matlabcentral/fileexchange/25889-an-optimal-control-tutorial-for-beginners

[2] https://www.mathworks.com/matlabcentral/fileexchange/71566-openocl-open-optimal-control-library

[3] An Introduction to Optimal Control Problems in Life Sciences and Economics, Sebastian Anita,Viorel Arnautu, Vincenzo Capasso

[4] https://digital.library.adelaide.edu.au/dspace/bitstream/2440/15125/1/152.pdf

Додатки

Метод Рунге-Кутта

Розглянемо задачу Коші:

$$\begin{cases} y'(x) = f(x, y(x)) \\ y(x_0) = y_0 \end{cases}$$

де $y_0 \in \mathbb{R}$, $f : D \to \mathbb{R}$, де

$$D = \left\{(x, y) \in \mathbb{R}^2; |x - x_0| \leq a, |y - y_0| \leq b\right\} \quad (a, b > 0)$$

Тоді розв'язок шукається за наступною рекурентною формулою:

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) h \\ t_{n+1} &= t_n + h \end{aligned}$$

де h - крок дискретизації,

$$\begin{aligned} k_1 &= f\left(t_n, y_n\right) \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right) \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right) \\ k_4 &= f\left(t_n + h, y_n + hk_3\right). \end{aligned}$$

для усіх $n = 0, 1, 2, 3, \ldots, N$, де $N = [T/h], T$ - верхня межа інтегрування по часу.

Приклад коду мовою Python:

```
import numpy as np


def runge_kutta_4order(t_prev, func_value_prev, func, h):
    k1 = func(t_prev, func_value_prev)
    k2 = func(t_prev + h/2, func_value_prev + k1*h/2)
    k3 = func(t_prev + h/2, func_value_prev + k2*h/2)
    k4 = func(t_prev + h, func_value_prev + k3*h)
    return func_value_prev + (k1 + 2*k2 + 2*k3 + k4)*h/6


def solve_ivp(right_side_function, initial_state, terminate_argument,
                                         discrete_param, backward=False):
    if backward:
        adjusted_right_side_function = lambda argument, state: -
                                                 right_side_function(argument,
                                                 state)
    else:
        adjusted_right_side_function = right_side_function
    states = [initial_state, ]
    state_prev = states[0]
    arg_prev = 0
    arg_space = np.arange(discrete_param, terminate_argument, discrete_param) if
                                             not backward else \
        np.arange(terminate_argument-2*discrete_param, -discrete_param, -
                                                 discrete_param)

    for arg in arg_space:
        state_new = runge_kutta_4order(arg_prev, state_prev,
                        adjusted_right_side_function,
                        discrete_param)
        states.append(state_new)
        arg_prev = arg
        state_prev = state_new
    if backward:
        states = states[::-1]
    return np.array(states)
```

Код оформлено з урахуванням можливості розв'язку зворотного ходу в задачі Коші.

Апроксимація оператора Лапласа для одно- та двовимірної задачі

Приклад коду мовою Python:

```
def laplacian_operator_approximation_2d(state_t, h):
    transformed_state = np.zeros_like(state_t)
    for x_1 in range(transformed_state.shape[0]):
        for x_2 in range(transformed_state.shape[1]):
            left = (state_t[x_1 - 1, x_2] if x_1 != 0 else state_t[x_1, x_2])
            right = (state_t[x_1 + 1, x_2] if x_1 != state_t.shape[0]-1 else
                                                        state_t[x_1, x_2])
            bottom = (state_t[x_1, x_2 - 1] if x_2 != 0 else state_t[x_1, x_2])
            top = (state_t[x_1, x_2 + 1] if x_2 != state_t.shape[1]-1 else
                                                        state_t[x_1, x_2])
            current = state_t[x_1, x_2]
            transformed_state[x_1, x_2] = (left + right + bottom + top - 4 *
                                                        current)/(h ** 2)
    return transformed_state


def laplacian_operator_approximation_1d(state_t, h):
    transformed_state = np.zeros_like(state_t)
    for x_1 in range(transformed_state.shape[0]):
        if x_1 == 0:
            transformed_state[x_1] = (state_t[x_1] + state_t[x_1 + 1] - 2 *
                                                        state_t[x_1]) / (h ** 2)
        elif x_1 == transformed_state.shape[0] - 1:
            transformed_state[x_1] = (state_t[x_1 - 1] + state_t[x_1] - 2 *
                                                        state_t[x_1]) / (h ** 2)
        else:
             transformed_state[x_1] = (state_t[x_1 - 1] + state_t[x_1 + 1] - 2 *
                                                         state_t[x_1]) / (h ** 2)
    return transformed_state
```

Алгоритм Armijo мовою Python

Приклад коду мовою Python:

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


from typing import Callable
from dataclasses import dataclass
from abc import ABC, abstractmethod

from src.utils import l2_norm, viz_1d_control, viz_2d_heatmap, viz_2d_time_gif,
                                              viz_3d_plot
from src.ode_utils import solve_ivp


class PMPProjectedGradientSolver(ABC):
        def __init__(self, state_equation_function: Callable,
                                                 adjoint_state_equation_function:
                                                  Callable,
                     integrand_cost_function: Callable,
                                                             cost_derivative_u_function
                                                             : Callable,
                     projection_gradient_operator: Callable, problem_name: str,
                                                             init_u: np.array,
                     terminate_time: int, boundary_space: np.array = None,
                                                             initial_state: np.
                                                             array=None,
                     eps_cost_derivative: np.float16 = 1e-2, eps_gradient_step:
                                                             np.float16 = 1e-2,
                     init_gradient_step: np.float16 = 1.0,
                     gradient_adjustment: np.float16 = 0.6,
                     time_grid_step: np.float16 = 1e-2, space_grid_step: np.
                                                             float16 = None,
                     gradient_step_max_iter: int = 20,
                     ):

            self.logger = logging.getLogger('PMPSolver_logger')
            self.logger.setLevel(logging.DEBUG)
            ch = logging.StreamHandler()
            ch.setLevel(logging.INFO)
            formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s
                                                       - %(message)s')
            ch.setFormatter(formatter)
            self.logger.addHandler(ch)
            self.problem_name = problem_name
            self.logger.info(f'                                                   {self.
```

```
                                        problem_name}')
self.eps_cost_derivative = eps_cost_derivative
self.eps_gradient_step = eps_gradient_step
self.init_gradient_step = init_gradient_step
self.gradient_adjustment = gradient_adjustment
self.time_grid_step = time_grid_step
self.terminate_time = terminate_time
self.init_u = init_u
self.boundary_space = boundary_space
self.init_state = initial_state

self.gradient_step_max_iter = gradient_step_max_iter
self.logger.info(f'

                                                          : {
                                        self.eps_cost_derivative}')
self.logger.info(f'
                                                     : {self.
                                        eps_gradient_step}')
self.logger.info(f'                                          :
                                        {self.init_gradient_step}')
self.logger.info(f'                                           :
                                         {self.time_grid_step}')
self.logger.info(f'                                   : {self.
                                        terminate_time}')
if space_grid_step is not None:
    self.space_grid_step = space_grid_step
    self.logger.info(f'

                                                     : {self.
                                            space_grid_step}')
self.logger.info(f'
                                                     : {self.
                                        init_u}')
self.logger.info(f'







                                                 : {self.
                                        gradient_step_max_iter}')

#                                  (                  )



self.state_equation_function = state_equation_function
```

```
        self.adjoint_state_equation_function =
                                            adjoint_state_equation_function

        self.cost_derivative_u_function = cost_derivative_u_function
        self.integrand_cost_function = integrand_cost_function
        self.gradient_projection_function = projection_gradient_operator

        #
        self.time_range = np.arange(0, self.terminate_time, self.
                                            time_grid_step)

        #
        self.current_gradient_iteration = 0
        self.current_cost_derivative_u = np.array([np.inf])
        self.current_cost = np.inf
        self.new_cost = self.current_cost
        self.current_gradient_step = self.init_gradient_step
        self.current_gradient_step_iteration = 0
        self.current_u = self.init_u
        self.new_u = self.current_u
        if self.init_state is np.ndarray:
            self.space_dimension = self.init_state.shape
        else:
            self.space_dimension = 1
        print(f'                                                   : {
                                            self.space_dimension}')

    def norm_gradient_stop_condition(self) -> bool:
        grad_norm = l2_norm(self.current_cost_derivative_u)
        return grad_norm < self.eps_cost_derivative

    def gradient_descent_loop_stop_condition(self) -> bool:
        stop_condition = (self.norm_gradient_stop_condition() &
                          (self.current_gradient_step < self.
                                                        eps_gradient_step
                                                        ))
        return stop_condition

    @abstractmethod
    def visualize_control(self, *args, **kwargs) -> None:
        pass

    @abstractmethod
    def solve_state_problem(self,  *args, **kwargs) -> np.array:
        pass

    @abstractmethod
    def solve_adjoint_state_problem(self, *args, **kwargs) -> np.array:
```

```
        pass

    @abstractmethod
    def integrate_cost(self, integrand_cost_function) -> np.float32:
        pass

    def adjust_gradient_step(self, *args, **kwargs) -> None:
        self.current_gradient_step *= self.gradient_adjustment

    def gradient_descent_loop(self) -> None:

        #


        while not self.gradient_descent_loop_stop_condition():

            self.current_cost = self.new_cost

            #          '
            self.current_state = self.solve_state_problem(self.current_u)
            self.logger.info('State')
           # self.logger.info(self.current_state)
            #          '

            self.current_adjoint_state = self.solve_adjoint_state_problem(
                                                    self.current_state, self
                                                    .current_u)
            self.logger.info('adjoint State')
            #self.logger.info(self.current_adjoint_state)
            #






            self.current_cost_derivative_u = self.cost_derivative_u_function(
                                                    self.current_u, self.
                                                    current_state,
                                                                        self







            #

            if self.norm_gradient_stop_condition():
```

```
                self.logger.info('                                        ')
                break

            self.logger.info(f'''                                                              :
                                                                {self.current_cost};
                                                            l2-

                                                                                       : {
                                                                                       l2_norm
                                                                                       (
                                                                                       self
                                                                                       .
                                                                                       current_cos
                                                                                       )};
                                                                {self.
                                                                                       current_gra
                                                                                       }.
                                                                                       ''')

            #
            #

            self.current_gradient_step = self.init_gradient_step
            for i in range(0, self.gradient_step_max_iter):
                if self.current_gradient_step < self.eps_gradient_step:
                    break

                print('Current u',self.current_u)
                print('Current gradient step', self.current_gradient_step)
                print('Current cost derivative', self.
                                                        current_cost_derivative_u
                                                        )
                #
                self.new_u = self.gradient_projection_function(self.current_u
                                                         - self.
                                                        current_gradient_step
                                                         *
                                                             self.

                print('cost_derivative')
                print('New u',self.new_u)
```

```
        #          ,
        self.new_state = self.solve_state_problem(self.new_u)

        #          ,

        self.new_adjoint_state = self.solve_adjoint_state_problem(
                                        self.new_state, self
                                        .new_u)

        #


        cost_func = np.vectorize(self.integrand_cost_function(self.
                                        new_state, self.
                                        new_adjoint_state,
                                        self.new_u))
        self.new_cost = self.integrate_cost(cost_func)

        #print(self.new_cost)
        if self.new_cost >= self.current_cost:
            self.adjust_gradient_step()
        else:
            self.current_u = self.new_u
            break
    else:
        self.current_u = self.new_u

    self.current_gradient_iteration += 1

    #                                             ,



    if np.abs(self.new_cost - self.current_cost) <= self.
                                    eps_cost_derivative:
        self.logger.info(f'                                      .


                                        : {-self.
                                        current_cost}')
        if self.space_dimension == 2:
            viz_2d_time_gif([self.current_state[slice_idx, :, :] for
                                            slice_idx in
                                            range(self.
                                            current_state.
                                            shape[0])],
                          ,
```

```
                                                                                  ,
                                                                                  2
                                                                                  ,
                                                                                  )
                        viz_2d_time_gif(
                            [self.current_state[slice_idx, :, :] for slice_idx in
                                                                    range(self.
                                                                   current_state
                                                                   .shape[0])],
                            ,
                                                                   , 2
                                                                   ')
                    break
            else:
                self.logger.info(f'                                    .
                                                                              :
                                                        {-self.current_cost}')
                if self.space_dimension == 2:
                    viz_2d_time_gif([self.current_state[slice_idx, :, :] for
                                                              slice_idx in range(
                                                              self.current_state.
                                                              shape[0])],
                                ,
                                                                          , 2
                                                                          ')
                    viz_2d_time_gif(
                        [self.current_state[slice_idx, :, :] for slice_idx in
                                                                range(self.
                                                                current_state.
                                                                shape[0])],
                        ,
                                                                , 2
                                                                ')


class PMPODESolver(PMPProjectedGradientSolver):
```

```
    def __init__(self, state_equation_function: Callable,
                                            adjoint_state_equation_function:
                                             Callable,
                 integrand_cost_function: Callable,
                                                         cost_derivative_u_function
                                                         : Callable,
                 projection_gradient_operator: Callable, problem_name: str,
                                                         init_u: np.array,
                 terminate_time: int, boundary_space: np.array = None,
                                                         initial_state: np.
                                                         array=None,
                 eps_cost_derivative: np.float16 = 1e-3, eps_gradient_step:
                                                         np.float16 = 1e-3,
                 init_gradient_step: np.float16 = 1.0, gradient_adjustment:
                                                         np.float16 = 0.6,
                 time_grid_step: np.float16 = 1e-2, space_grid_step: np.
                                                         float16 = None,
                 gradient_step_max_iter: int = 20):

        super().__init__(state_equation_function,
                                         adjoint_state_equation_function
                                         ,
                 integrand_cost_function, cost_derivative_u_function,
                 projection_gradient_operator, problem_name, init_u,
                 terminate_time, boundary_space, initial_state,
                 eps_cost_derivative, eps_gradient_step,
                 init_gradient_step,gradient_adjustment,
                 time_grid_step, space_grid_step,
                 gradient_step_max_iter)

    def visualize_control(self):
        viz_1d_control(self.time_range, self.current_u, "
                                                                u(t)", "
                                         t", "u(t)")
        viz_1d_control(self.time_range, self.current_state, "
                                                           ", "t", "x(t)"
                                         )

    def solve_state_problem(self, u) -> np.array:
        state = solve_ivp(self.state_equation_function(u), self.init_state,
                                         self.terminate_time, self.
                                         time_grid_step)
        return state

    def solve_adjoint_state_problem(self, state, u) -> np.array:
        adjoint_state = solve_ivp(self.adjoint_state_equation_function(state,
                                          u), 0.0,
                              self.terminate_time, self.time_grid_step,
```

```
                                                                        backward
                                                                        =
                                                                        True
                                                                        )
            return adjoint_state

        def integrate_cost(self, integrand_cost_function: Callable):
            return np.trapz(y=integrand_cost_function(self.time_range), x=self.
                                                     time_range, dx=self.
                                                     time_grid_step)


class PMPPDESolver(PMPProjectedGradientSolver):
        def __init__(self, *args, **kwargs):
            super().__init__(*args, **kwargs)

        def visualize_control(self, dimensions: int = 2) -> None:

            if dimensions == 2:
                print(self.current_u)
                viz_2d_heatmap(self.current_u, '
                                                                    ', save=
                                                     True)
                viz_2d_heatmap(self.current_state, '                    ',
                                                     save=True)
                viz_2d_heatmap(self.current_state, '                    ',
                                                     save=True)

                viz_3d_plot(self.current_u, '
                                                                    ', save=
                                                     True)
                viz_3d_plot(self.current_state, '
                                                                    ', save=
                                                     True)
            elif dimensions == 3:
                print(self.current_u)
                print(self.current_u.shape)
                viz_2d_heatmap(self.current_u[0, :, :], '

                                                                  ', save=
                                                     True)
                viz_2d_heatmap(self.current_state[0, :, :], '
                                                                  ', save=
                                                     True)
                viz_2d_heatmap(self.current_u[-1, :, :], '

                                                              ', save=True)
                viz_2d_heatmap(self.current_state[-1, :, :], '
```

```
                                                            ', save=True)
            viz_3d_plot(self.current_state[-1, :, :], '
                                                                     ',
                                                    save=True)
            viz_3d_plot(self.current_state[0, :, :], '
                                                                     ',
                                                    save=True)
            viz_3d_plot(self.current_u[-1, :, :], '
                                                                     ',
                                                    save=True)
            viz_3d_plot(self.current_u[0, :, :], '
                                                                     ',
                                                    save=True)

        else:
            viz_1d_control(self.time_range, self.current_u)

    def solve_state_problem(self, u) -> np.array:
        state = solve_ivp(self.state_equation_function(u), self.init_state,
                                              self.terminate_time, self.
                                              time_grid_step)
        return state

    def solve_adjoint_state_problem(self, state, u) -> np.array:
        adjoint_state = solve_ivp(self.adjoint_state_equation_function(state,
                                               u), np.zeros_like(self.
                                              init_state),
                                  self.terminate_time, self.time_grid_step,
                                                                     backward
                                                                     =
                                                                     True
                                                                     )
        return adjoint_state

    def integrate_cost(self, integrand_cost_function: Callable):
        return np.trapz(y=integrand_cost_function(self.time_range), x=self.
                                              time_range, dx=self.
                                              time_grid_step)
```