Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра математики факультету інформатики

АНАЛІЗ РЕАЛІЗАЦІЙ СИМПЛЕКС-МЕТОДУ ЛІНІЙНОГО ПРОГРАМУВАННЯ НА ПРИКЛАДІ ЗАДАЧІ ЛОГІСТИЧНОЇ ОПТИМІЗАЦІЇ

Кваліфікаційна робота

за спеціальністю “Комп’ютерні науки” 122

освітній ступінь - бакалавр

Виконала: студентка 4-го року навчання

Спеціальності

122 Комп’ютерні науки

Мудра Катерина Володимирівна

Керівник: Силенко І. В.

старший викладач

Рецензент: _____

(прізвище та ініціали)

Кваліфікаційна робота захищена

з оцінкою _____

Секретар ЕК_____

(підпис)

“____” _____ 2025 р.

Київ 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра математики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри математики

доцент, кандидат фіз.-мат. наук

_____ Р.К. Чорней

(підпис)

“____”_____ 2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студентці Мудрій Катерині Володимирівні 4-го року навчання факультету інформатики

ТЕМА: АНАЛІЗ РЕАЛІЗАЦІЙ СИМПЛЕКС-МЕТОДУ ЛІНІЙНОГО ПРОГРАМУВАННЯ НА ПРИКЛАДІ ЗАДАЧІ ЛОГІСТИЧНОЇ ОПТИМІЗАЦІЇ

Зміст кваліфікаційної роботи:

Дата видачі “____”_____ 2025 р. Керівник

(підпис)

Завдання отримав

(підпис)

Календарний план виконання кваліфікаційної роботи

Тема: Аналіз реалізацій симплекс-методу лінійного програмування на прикладі задачі логістичної оптимізації

№ з/п	Назва етапу	Термін виконання	Примітка
1	Отримання теми кваліфікаційної роботи	Жовтень 2024	
2	Ознайомлення з темою кваліфікаційної роботи	Жовтень 2024	
3	Розробка плану та структури роботи	Січень 2025	
4	Робота з науковою літературою, опис основних означень. Написання вступу та анотації	Лютий 2025	
5	Реалізація алгоритмів	Березень 2025	
6	Робота над текстовим оформленням теоретичної частини та одержаних результатів	Квітень 2025	
7	Попередній аналіз кваліфікаційної роботи. Виправлення помилок	Травень 2025	
8	Попередній захист кваліфікаційної роботи	28 травня 2025	
9	Захист кваліфікаційної роботи	5 червня 2025	

Студентка Мудра К.В.

Керівник Силенко І. В.

“__”_____2025 р.

ЗМІСТ

АНОТАЦІЯ

У дипломній роботі досліджено застосування методів лінійного програмування для оптимізації логістичних рішень у сфері гуманітарної допомоги, зокрема в умовах воєнного стану в Україні. Метою дослідження є побудова математичної моделі багатокритеріальної оптимізації, яка враховує вартість, ризик та час доставки вантажів до постраждалих регіонів.

У роботі проведено теоретичний аналіз лінійного програмування, розглянуто класичні задачі (транспортну, цілочислову, багатоцільову) та геометричну інтерпретацію допустимих рішень. Запропоновано багатоцільову модель логістичної оптимізації із застосуванням методу зваженої згортки, що дозволяє адаптувати модель до різних сценаріїв гуманітарних місій. Розроблена модель реалізована трьома методами: з використанням бібліотеки PuLP, функції scipy.optimize.linprog та власної реалізації симплекс-методу. Проведено порівняльний аналіз результатів для малих та великих масштабів задачі, а також мікроаналіз чутливості моделі до вагових коефіцієнтів α, β, γ.

Результати дослідження демонструють, що власноруч реалізований симплекс-метод забезпечує більшу стійкість і точність у складних умовах, ніж стандартні бібліотечні засоби. Робота має практичну цінність для організацій, що займаються плануванням і координацією гуманітарної логістики в умовах обмежених ресурсів та підвищеної невизначеності.

Ключові слова:

лінійне програмування, симплекс-метод, гуманітарна логістика, багатокритеріальна оптимізація, Python, метод зваженої згортки, транспортна задача.

ВСТУП

У сучасних умовах повномасштабної війни в Україні питання ефективного розподілу гуманітарних ресурсів набуло критичного значення. Через значні руйнування інфраструктури, обмежений доступ до постраждалих територій та постійні загрози безпеці, гуманітарна логістика стикається з надзвичайно складними викликами. У таких умовах необхідно приймати оптимізаційні рішення, що враховують не лише економічну доцільність, але й часові обмеження та ризики, пов’язані з переміщенням вантажів. Саме тому використання методів лінійного програмування, зокрема у багатокритеріальному форматі, є актуальним і затребуваним інструментом підтримки прийняття рішень у кризових ситуаціях.

Окрему важливість ця тема має для громадських і благодійних організацій, які здійснюють постачання допомоги до деокупованих або небезпечних регіонів України. Підвищення ефективності логістичних рішень за допомогою математичних моделей може суттєво покращити координацію, мінімізувати витрати і водночас забезпечити своєчасне реагування на потреби цивільного населення.

Метою дипломної роботи є розробка та практична реалізація моделі багатоцільової оптимізації задачі гуманітарної логістики з використанням методів лінійного програмування.

Для досягнення цієї мети були поставлені такі завдання:

- провести теоретичний огляд методів лінійного програмування, зокрема симплекс-методу та багатоцільової оптимізації;
- сформулювати прикладну задачу оптимізації гуманітарного постачання з урахуванням критичних факторів (вартість, ризик, час);
- побудувати математичну модель задачі у вигляді задачі лінійного програмування;
- реалізувати задачу трьома методами: з використанням бібліотек PuLP і scipy.optimize.linprog, а також за допомогою власноручного симплекс-методу;
- здійснити мікроаналіз чутливості та оцінити ефективність кожного підходу;
- сформулювати висновки щодо застосовності різних інструментів у гуманітарному контексті.

Об’єктом дослідження є процес оптимального розподілу гуманітарних ресурсів між складами та пунктами призначення в умовах обмеженості ресурсів, ризиків і часових обмежень.

Предметом дослідження є математичні моделі лінійного програмування та методи їх реалізації для багатокритеріальної логістичної задачі в гуманітарному середовищі.

У роботі використано методи лінійного програмування, включно з канонічною формалізацією задач, симплекс-методом, багатокритеріальною оптимізацією (метод зваженої згортки), аналізом чутливості та сценарним плануванням. Для реалізації моделей застосовано мовні та бібліотечні засоби Python – зокрема, PuLP, scipy.optimize.linprog та власну реалізацію симплекс-методу. Також використано методи верифікації, порівняння ефективності та оцінки стабільності рішень.

Дипломна робота складається з трьох основних розділів:

- У першому розділі подано теоретичні засади лінійного програмування, класифікацію задач та методів їх розв’язання.
- У другому розділі сформульовано прикладну задачу гуманітарної логістики, побудовано математичну модель і описано її компоненти.
- У третьому розділі здійснено реалізацію задачі різними методами, проведено експерименти, інтерпретовано результати та виконано аналіз чутливості моделі.

РОЗДІЛ 1. ОГЛЯД ТЕОРЕТИЧНИХ ЗАСАД ЛІНІЙНОГО ПРОГРАМУВАННЯ

1.1. Формальне визначення задачі лінійного програмування (ЗЛП)

Лінійне програмування – це метод математичного моделювання, який використовується для знаходження оптимального (максимального або мінімального) значення лінійної цільової функції за наявності системи обмежень у вигляді лінійних нерівностей або рівнянь. Цей підхід базується на припущенні обмеженості ресурсів і кількісній вимірюваності мети оптимізації. [1]

У прикладних задачах лінійне програмування дозволяє приймати обґрунтовані рішення в умовах обмежених ресурсів. Наприклад, підприємства можуть використовувати його для визначення оптимального обсягу виробництва з метою максимізації прибутку, враховуючи доступність матеріалів та робочої сили. [1] Цей метод активно застосовується в бізнес-плануванні, інженерії, ресурсному управлінні, а також у соціальних і природничих науках. [2]

У наукових і прикладних контекстах, лінійне програмування – це частина ширшої галузі методів оптимізації, яка вивчає ефективний розподіл обмежених ресурсів у великих системах з багатьма змінними. У навчальному курсі алгебри, однак, розглядаються здебільшого базові задачі з двома змінними, які легко зобразити графічно. [1]

Задача лінійного програмування (ЗЛП) – це задача математичної оптимізації, у якій необхідно максимізувати або мінімізувати лінійну цільову функцію при дотриманні системи лінійних обмежень. Усі функції (як цільова, так і обмеження) залежать лише від лінійних комбінацій змінних, а самі змінні мають бути невід’ємними (тобто додатними або нульовими)

Формалізація задачі лінійного програмування звучить так:

Нехай маємо $x_1, x_2, \dots, x_n$, *що представляють кількісні рішення (наприклад, скільки продукту потрібно виробити, які обсяги відправити або скільки працівників залучити).*

Для розв’язання задачі лінійного програмування потрібно визначити цільову функцію.

Цільова функція (англ. objective function) – це лінійна математична функція, яку потрібно максимізувати або мінімізувати в межах допустимих значень змінних, що визначаються системою обмежень. Вона відображає ключову мету задачі оптимізації – наприклад, зменшити витрати, збільшити прибуток, мінімізувати ризик, максимізувати ефективність тощо.

Цільова функція має вигляд:

$$Z = c_1x_1 + c_2x_2 + \ldots + c_nx_n = \sum_{j=1}^{n} c_jx_j$$

Z – значення цільової функції (оптимізаційний результат, який шукаємо),

$x_1, x_2, \ldots, x_n$ – змінні рішення, тобто ті величини, які підбираються,

$c_2, \ldots, c_n$ – коефіцієнти ваги, які показують внесок кожної змінної в загальне значення Z

Змінні задачі лінійного програмування мають задовільняти систему обмежень.

Система обмежень (англ. constraint system) визначає допустиму область значень змінних, в межах якої повинно шукатися оптимальне рішення. Обмеження в задачі лінійного програмування є лінійними нерівностями або рівняннями, які відображають обмеженість ресурсів, часових, матеріальних, логістичних чи інших факторів.

Формально система обмежень позначається так:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \le b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \le b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \le b_m \end{cases}$$

Або у матричному вигляді:

$$A \cdot x \le b$$

де:

$A = [a_{ij}] -$ матриця коефіцієнтів обмежень розміром $m \times n$,

що визначають вплив змінних на ресурси або обмеження,

$x = [x_1, x_2, \dots, x_n]^T -$ вектор змінних

$b = [b_1, b_2, \dots, b_m]^T -$ вектор правих частин,

що визначає межі доступних ресурсів.

Правила системи обмежень:

1. Кожне обмеження обмежує можливі значення змінних.
2. Всі обмеження разом формують область допустимих рішень – тобто множину точок x, які задовольняють всі обмеження та умови невід'ємності.
3. Оптимальний розв'язок може бути знайдений тільки всередині цієї області.

Окрім цього, усі змінні задачі мають зберігати **умову невід'ємності**:

$$x_j \geq 0, \quad \text{для всіх } j = 1,2, \dots, n$$

У реальних задачах змінні часто не можуть бути від'ємними – не можна, наприклад, виробити «-10 одиниць» продукту.

Щоб краще зрозуміти компоненти задачі лінійного програмування, пропонуємо розглянути практичний приклад:

Задача: *Підприємство виготовляє два види продукції – A та B. Прибуток з одиниці продукції A становить 40 грн, а з одиниці B – 30 грн.*

Тоді цільова функція:

$$Maximize\ Z = 40x_1 + 30x_2$$

де

$x_1 -$ кількість продукції A,

$x_2 -$ кількість продукції B,

$$Z - \text{загальний прибуток, який потрібно максимізувати}$$

Потрібно визначити, скільки одиниць кожного продукту виготовляти, щоб максимізувати загальний прибуток, враховуючи певні обмеження.

Нехай:

- наявно 100 годин робочого часу
- наявно 180 одиниць сировини
- продукт А вимагає 2 години і 3 одиниці сировини
- продукт В вимагає 1 годину і 2 одиниці сировини

Тоді система обмежень матиме вигляд:

$$\begin{cases} 2x_1 + x_2 \leq 100 \text{ - обмеження по часу} \\ 3x_1 + 2x_2 \leq 180 \text{ - обмеження по сировині} \end{cases}$$

Умови допустимості (невід'ємності:

$$x_1 \geq 0, x_2 \geq 0$$

Повна математична модель виглядатиме таким чином:

$Maximize$: $Z = 40x_1 + 30x_2$

$Subject\ to$: $2x_1 + 1x_2 \leq 100$ (обмеження по часу)

$3x_1 + 2x_2 \leq 180$ (обмеження по сировині)

$x_1 \geq 0, x_2 \geq 0$ (невід'ємність змінних)

1.2. Типи задач ЛП

1.2.1. Транспортна задача

Транспортна задача є однією з найвідоміших прикладних задач лінійного програмування, яка широко використовується в логістиці, управлінні постачанням, гуманітарних операціях та виробничому плануванні. Вона передбачає оптимізацію витрат на транспортування однорідного вантажу з декількох пунктів постачання (складів) до декількох пунктів споживання (отримувачів), при дотриманні обмежень на доступний обсяг ресурсів і рівень попиту. [3]

Ключова мета транспортної задачі полягає у мінімізації загальної вартості перевезень за умови, що:

- кожен склад має фіксований обсяг ресурсів;
- кожен пункт призначення має фіксований обсяг потреби;
- транспортування здійснюється між усіма допустимими парами "джерело – призначення". [3]

Під час формулювання транспортної задачі використовуються такі припущення:

1. Перевозиться один вид продукції.
2. Кожен пункт постачання має відому кількість одиниць ресурсу , яку повністю необхідно розподілити.
3. Кожен пункт призначення має чітко заданий попит , який повинен бути повністю задоволений.
4. Транспортування можливе між усіма дозволеними маршрутами.
5. Усі витрати на перевезення відомі та не змінюються.
6. Величини поставок і попиту є детермінованими та відомими наперед.
7. Сумарна пропозиція дорівнює сумарному попиту:

$$\sum_{i=1}^{m} S_i = \sum_{j=1}^{n} D_j$$

Якщо ця умова не виконується, задача вважається незбалансованою. Для її вирівнювання додається фіктивний склад або пункт призначення з нульовими витратами транспортування. [3]

Математична модель транспортної задачі будується як задача лінійного програмування. Позначимо:

$\boldsymbol{x_{ij}}$ – кількість вантажу, що транспортується зі складу $\boldsymbol{i}$ до пункту $\boldsymbol{j}$

$\boldsymbol{c_{ij}}$ – вартість транспортування одиниці вантажу між $\boldsymbol{i}$ та $\boldsymbol{j}$

$\boldsymbol{S_i}$ – обсяг ресурсу на складі $\boldsymbol{i}$

$\boldsymbol{D_j}$ – попит у пункті $\boldsymbol{j}$

Тоді задача формулюється як:

Цільова функція (мінімізація витрат)

$$\min Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \cdot x_{ij}$$

Обмеження:

- Для кожного складу

$$\sum_{j=1}^{n} x_{ij} = S_i, \quad i = 1,2, \dots, m$$

- Для кожного пункту призначення

$$\sum_{i=1}^{m} x_{ij} = D_j, \quad j = 1,2, \dots, n$$

- Умови невід'ємності

$$x_{ij} \geq 0, \quad \forall i, j$$

Для транспортних задач використовуються спеціальні методи, які дозволяють знайти допустимий початковий базисний розв'язок. До них належать:

- **Метод північно-західного кута.** Початкове заповнення таблиці здійснюється з верхнього лівого кута, рухаючись по черзі вправо та вниз. У кожну клітинку вноситься максимально можливий обсяг поставки або попиту. Метод простий, але не враховує транспортні витрати, тому часто дає неоптимальні рішення.
- **Метод найменшої вартості.** На кожному кроці обирається клітинка з найнижчими витратами транспортування, куди записується допустима кількість постачання. Процес повторюється з урахуванням залишків. Метод зазвичай ефективніший за попередній.
- **Метод Фогеля (VAM).** Для кожного рядка та стовпця визначається різниця між двома найменшими витратами. Обирається той рядок або стовпець, де ця різниця найбільша, і здійснюється перевезення у клітинці з мінімальною вартістю. Метод забезпечує якісніший стартовий розв'язок серед евристичних підходів. [3]

1.2.2. Задача про призначення

Задача про призначення є окремим випадком збалансованої транспортної задачі, в якій усі запаси та попити дорівнюють одиниці. Це означає, що кожному виконавцю потрібно призначити одне і лише одне завдання, і кожне завдання має бути виконане одним і лише одним виконавцем. Завдяки цій властивості задача має чітку структуру та допускає ефективні методи розв’язання. [4]

Існують різні варіанти задачі призначення, які можна класифікувати за такими ознаками:

1. Задача на мінімізацію

У цьому випадку завдання полягає в тому, щоб розподілити виконавців між задачами так, щоб загальні витрати були якнайменшими. До витрат можуть належати час, енергетичні ресурси, кошти або інші ресурси.

Приклад: оптимізація призначення співробітників на завдання для мінімізації загального часу виконання робіт.

2. Задача на максимізацію

Ціль полягає в тому, щоб здійснити призначення так, аби отримати якомога більше користі, наприклад, максимізувати прибуток, продуктивність або якість виконання. Матриця, яка описує доцільність призначень, може відображати оцінку ефективності або переваг.

Приклад: вибір найкращого поєднання викладачів і навчальних курсів з урахуванням їхньої спеціалізації.

3. Нерівноквадратні задачі

У практиці часто трапляються ситуації, коли кількість об'єктів у двох групах не збігається. Щоб зробити задачу обчислюваною стандартними методами, її перетворюють у симетричну, додаючи до таблиці фіктивного виконавця або фіктивне завдання з нейтральним (наприклад, нульовим) ефектом.

Приклад: маємо 5 технічних фахівців, але лише 4 доступні завдання.

4. Задачі з обмеженнями

У деяких ситуаціях певні призначення не дозволені або небажані. Наприклад, через відсутність навичок, технічну несумісність, безпекові вимоги чи етичні міркування.

Такі обмеження моделюються шляхом:

- надання дуже високої "штрафної" вартості в матриці призначення,
- або ж явного виключення можливості призначення (заданням =0).

Приклад: новачкові не можна доручати складне завдання; певна машина не підтримує конкретний тип операції. [4]

Математична модель задачі про призначення виглядає таким чином:

Нехай:

n – кількість виконавців та завдань

c_{ij} – вартість призначення виконавця i на завдання j

$x_{ij} \in \{0,1\}$ – бінарна змінна:

$x_{ij} = 1$, якщо i виконує завдання j, інакше -0

Тоді задача формулюється так:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_{ij}$$

за умов:

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i = 1, \dots, n \text{ (кожен виконавець має 1 завдання)}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \forall j = 1, \dots, n \text{ (кожне завдання має 1 виконавця)}$$

$$x_{ij} \in \{0,1\}, \quad \forall i, j$$

1.2.3. Задача з цілочисловими змінними

Цілочислове програмування (Integer Linear Programming, ILP) – це тип математичної задачі оптимізації, в якій деякі або всі змінні обмежені цілочисловими значеннями. Як і в класичному лінійному програмуванні, цільова функція та обмеження мають лінійний характер. Проте наявність цілочислових змінних значно ускладнює обчислення. [5]

У випадках, коли усі змінні обмежені як цілочислові, говоримо про повне цілочислове програмування (Pure ILP). Якщо лише частина змінних повинна бути цілою, задача називається змішаним цілочисловим програмуванням (MILP). Коли змінні можуть набувати лише значень 0 або 1, маємо справу з бінарним ILP (0–1 програмуванням).

У канонічній формі задача ILP формулюється так:

$$\max \ \mathrm{c}^T x$$

за умови:

$$A\,x \ \leq b,$$

$$x \ \geq 0,$$

$$x \in Z^n$$

Для переходу до стандартної форми вводяться змінні запасу s для перетворення нерівностей на рівності:

$$max\ c^T x$$

за умови:

$$Ax \ + \ s \ = \ b,$$

$$x \ \geq 0, \quad s \ \geq 0,$$

$$x \in Z^n$$

Задачі ILP належать до NP-важких (NP-hard), а у випадку 0–1 (булевого) програмування – до NP-повних (NP-complete). Це означає, що на відміну від класичних задач лінійного програмування, цілочислові задачі

складно розв’язувати для великих розмірів, особливо якщо потрібно знайти гарантовано оптимальний розв’язок.

Хоча сучасні алгоритми (гілки і межі, розсічення, комбіновані методи) дозволяють розв’язувати задачі середнього масштабу, на практиці розмір моделі ILP, яку можна вирішити за прийнятний час, значно менший, ніж у випадку класичних ЛП-задач. [6]

Задачі цілочисельного лінійного програмування поділяються на такі 4 типи:

- Pure ILP – усі змінні повинні бути цілими числами.
- Mixed ILP (MILP) – частина змінних цілочислова, частина – дійсна.
- Binary (0–1) ILP – змінні набувають значень лише 0 або 1; особливо корисні для моделювання логічних рішень (вибір/відмова).
- Кодування цілого через бінарні змінні – будь-яке обмежене ціле значення може бути подане як сума бітів:

$$x = x_1 + 2x_2 + 4x_3 + \cdots + 2^k x_{k+1}, \quad x_i \in \{0,1\}$$

$$k = \lfloor \log_2 U \rfloor, \quad U - \text{верхня межа змінної}$$

Задачі цілочисельного програмування є одними з найбільш популярних у лінійному програмуванні. Наприклад, їх використовують у таких сферах:

- **Виробниче планування.** Наприклад, планування сільськогосподарських культур, коли ресурси (земля, праця, капітал) обмежені, а результат повинен бути цілим (неможливо посадити 3,4 поля пшениці).
- **Розклад і логістика.** Призначення транспортних засобів на маршрути, формування графіків роботи, планування проектів – часто використовують бінарні змінні для опису рішень "призначено/не призначено".
- **Зонування територій.** Завдання поділу регіону на округи з урахуванням компактності, меж, рівноваги населення (наприклад, виборчі округи, райони обслуговування шкіл або лікарень).
- **Проектування телекомунікаційних мереж.** Оптимізація топології мережі, де змінні визначають, які лінії встановлювати та з якими потужностями (що є цілими величинами).

– **Частотне планування в мобільних мережах.** Розподіл частот між антенами з урахуванням мінімізації перешкод – модель із бінарними змінними, що вказують, чи призначена частота конкретній антені. [5]

1.2.4. Двоїста задача

Двоїстість (dual problem) - фундаментальне поняття теорії лінійного програмування, яке полягає в тому, що для кожної задачі (прямої, або примальної) існує інша, пов'язана з нею задача (двоїста), що має тісну математичну і логічну структуру. Ці задачі відображають одну і ту ж ситуацію, але з різних точок зору - наприклад, виробничу модель (пряма задача) та модель вартості ресурсів (двоїста задача). [7]

Формально двоїсту задачу можна записати так:

Нехай задана пряма задача лінійного програмування у канонічній формі максимізації:

$$\max z = \sum_{j=1}^{n} c_j x_j$$

при умові

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1,2,\dots,m$$

$$x_j \geq 0, \quad j = 1,2,\dots,n$$

Їй відповідає двоїста задача у формі мінімізації:

$$v = \sum_{i=1}^{m} b_i y_i$$

при умові

$$\sum_{i=1}^{m} a_{ij} y_i \geq c_j, \quad j = 1,2,\dots,n$$

$$y_i \geq 0, \quad i = 1,2,\dots,m$$

У кожному обмеженні прямої задачі виникає змінна у двоїстій, і навпаки. Коефіцієнти матриці A зберігаються, але їхні ролі "змінюються місцями": рядки перетворюються на стовпці. Тип оптимізації змінюється з максимізації на мінімізацію, а знак обмежень - з ≤ на ≥. [7]

Змінні двоїстої задачі мають природну економічну інтерпретацію. Вони часто представляють тіньові ціни або граничну вартість ресурсів у прямій задачі. Наприклад, якщо y_2=5, це означає, що збільшення доступного ресурсу b_2 на одну одиницю дозволить підвищити значення цільової функції на 5 одиниць. Крім того, умови двоїстої задачі можна інтерпретувати як критерії оптимальності: кожна змінна у прямій задачі не повинна покращувати розв’язок, якщо вона не належить до базису.

В основу задач двоїстості лягають відповідні теореми:

1. **Теорема слабкої двоїстості (Weak Duality Theorem)**

   Якщо пряма задача є задачею максимізації, а її двоїста задача - мінімізації, і обидві мають допустимі розв’язки, тоді значення цільової функції прямої задачі не перевищує значення цільової функції двоїстої задачі:

$$\text{opt(Пряма)} \leq \text{opt(Двоїста)}$$

   Це означає, що жодне допустиме рішення у двоїстій задачі не занижує справжнє значення максимального результату прямої задачі, і навпаки. [8]

2. **Теорема сильної двоїстості (Strong Duality Theorem)**

   Якщо одна з двох задач - пряма або двоїста - має допустимий розв’язок і обмежена, тоді й інша також має допустимий розв’язок, і обидві дають однакове значення цільової функції:

$$\text{opt(Пряма)} = \text{opt(Двоїста)}$$

Це твердження є ключовим для розв’язання задач ЛП, оскільки дає змогу перевірити оптимальність рішення прямої задачі через двоїсту. У практичних умовах саме через це часто обчислюють двоїсту модель паралельно з прямою. [8]

Залежно від типу допустимості й обмеженості задач, можливі наступні випадки:

Пряма задача	Двоїста задача	Випадок
Обмежена та допустима	Обмежена та допустима	Оптимуми збігаються
Необмежена	Несумісна	Немає граничного значення
Несумісна	Необмежена або несумісна	Немає допустимого рішення

Поняття двоїстості дозволяє не лише краще інтерпретувати та аналізувати розв’язки задач лінійного програмування, а й суттєво підвищити ефективність їхнього обчислення. Теореми слабкої та сильної двоїстості гарантують, що розв’язки двох пов’язаних задач - прямої та двоїстої - несуть узгоджену інформацію. Це дозволяє перевіряти оптимальність, проводити чутливий аналіз та розробляти ефективні алгоритми на практиці.

1.2.5. Багатоцільова задача

Багатоцільове лінійне програмування (MOLP, Multi-Objective Linear Programming) передбачає оптимізацію декількох цільових функцій одночасно. Такі задачі виникають у ситуаціях, коли потрібно досягнути балансу між кількома критеріями, які можуть бути суперечливими. Наприклад, зменшення витрат може збільшити час виконання, а підвищення ефективності може вимагати більших ресурсів.

Оскільки в загальному випадку не існує єдиного розв’язку, який одночасно оптимізує всі цілі, результатом багатоцільової оптимізації є множина Парето-оптимальних рішень - таких, у яких неможливо покращити одну ціль без погіршення принаймні однієї іншої. Для знаходження таких рішень застосовуються спеціальні методи, що дозволяють сформулювати компромісні варіанти, виходячи з переваг або обмежень конкретної ситуації.

Для розв’язання багатоцільових задач використовуються конкретні методи:

Метод зваженої суми (Weighted Sum Method).

Один із найпростіших способів перетворити багатоцільову задачу на одноцільову - об’єднати всі цільові функції в одну шляхом зваженого додавання. Кожна функція має відповідний коефіцієнт ваги, який відображає її відносну важливість. Загальна форма:

$$\max Z = \alpha_1 z_1(x) + \alpha_2 z_2(x) + \cdots + \alpha_k z_k(x),$$

$$\sum_{i=1}^{k} \alpha_i = 1, \quad \alpha_i \geq 0$$

Цей метод дозволяє отримати Парето-оптимальні рішення, змінюючи ваги. Його основною перевагою є простота реалізації - задача зводиться до класичної лінійної. Проте метод не дозволяє дослідити всі ділянки Парето-фронту, особливо у випадку невипуклих множин рішень.

Метод ε-обмежень (Epsilon-Constraint Method).

У цьому підході одна цільова функція оптимізується, тоді як інші перетворюються на обмеження з допустимими граничними значеннями ε:

$$\max z_1\,(x)$$

за умов:

$$z_i(x) \leq \varepsilon_i, \quad i = 2, \dots, k$$

Метод ε-обмежень дозволяє точково досліджувати Парето-фронт і будувати рішення, які не можна отримати методом зважених коефіцієнтів. Він є більш гнучким і підходить для ситуацій, коли користувач готовий визначити допустимі рівні для другорядних цілей. Кожна комбінація ε вимагає розв’язання окремої задачі, що збільшує обчислювальне навантаження.

Метод ідеальної точки (Ideal Point Method)

Цей підхід базується на концепції ідеального (але, як правило, недосяжного) розв’язку, у якому всі цілі досягають своїх найкращих значень. Метод полягає у пошуку такого рішення, яке найближче (за певною мірою відстані, наприклад, евклідовою) до цього ідеального вектора:

$$\min|z(x) - z^*|$$

де $z^* = (z_1^*, z_2^*, \dots, z_k^*)$ - вектор найкращих значень кожної цілі

Цей метод дозволяє формально моделювати «збалансоване» рішення без потреби встановлювати ваги чи граничні значення вручну. Він особливо корисний тоді, коли важко сформулювати чіткі пріоритети, але є уявлення про бажану поведінку системи.

1.2.6. Задача з параметричними або нечіткими змінними

Задача з параметричними коефіцієнтами - це задача лінійного програмування, в якій один або кілька коефіцієнтів цільової функції або обмежень залежать від змінного параметра (наприклад, $\lambda \in R$). Такі задачі використовуються для проведення чутливого аналізу та дослідження стабільності рішень при зміні зовнішніх умов (ціни, ресурси, попит тощо).

Задача з нечіткими коефіцієнтами (fuzzy linear programming) - це узагальнення класичної ЗЛП, у якому частина параметрів (коефіцієнти, права частина або цільова функція) задані як нечіткі (fuzzy) числа або множини. Такі задачі моделюють невизначеність, коли точне значення параметра не відоме, але задане наближено, інтервально або мовними оцінками ("бажано", "достатньо", "приблизно").

Узагальнені формальні постановки цих задач виглядають таким чином:

Параметрична задача

$$\max z\,(\lambda) = c(\lambda)^T x \quad \text{за умови} \quad A(\lambda)x \le b(\lambda), \quad x \ge 0$$

де $\lambda \in R$ - параметр, що змінюється в заданому діапазоні.

Задача з нечіткими змінними

$$\text{Знайти } x \in X \text{ таке, що } \tilde{A}x \le \tilde{b}, \quad \text{та максимізувати } \tilde{z} = \widetilde{c^T}x$$

де $\tilde{a}, \tilde{b}, \tilde{c}$ - нечіткі (fuzzy) коефіцієнти, що задаються через функції належності.

У свою чергу, нечітке лінійне програмування (fuzzy LP) використовується в умовах, коли точні значення параметрів моделі відомі лише приблизно або задані у вигляді мовних оцінок (наприклад, «низький

ризик», «високий пріоритет», «достатній ресурс»). У таких моделях використовуються нечіткі множини, що задаються через функції належності, а оптимізація проводиться з урахуванням максимізації міри задоволення обмежень. Застосовуються такі підходи, як α-рівневий аналіз, інтервальна оптимізація, а також розширення симплекс-методу для нечітких систем (за підходами Zadeh, Zimmermann та ін.).

Обидва типи задач - параметричні та нечіткі - відіграють важливу роль у моделюванні систем з невизначеністю або залежністю від зовнішніх умов. Їх застосування охоплює низку сфер:

- Параметричне ЛП використовується для аналізу чутливості до змін вартості, ресурсів, попиту, а також для оцінки стійкості рішень у складних логістичних або економічних моделях.
- Нечітке ЛП знаходить застосування у гуманітарній логістиці, соціальному управлінні, плануванні освітніх або проєктних рішень, коли важливо враховувати неповну, якісну або експертну інформацію.

Такі задачі дозволяють приймати більш гнучкі та адаптивні рішення, орієнтовані не лише на точні розрахунки, але й на реалістичні обмеження та ризики, притаманні складним середовищам.

1.3. Характеристики задачі ЛП

Задача лінійного програмування (ЛП) є математичною моделлю оптимізації, яка має чітко визначену структуру. Для того щоб задача могла бути розв’язана методами ЛП, вона повинна відповідати низці характеристик:

1. Наявність обмежень (constraints)

У задачі мають бути присутні обмеження, які формалізують обмеженість ресурсів або вимоги до рішень. [1] Ці обмеження подаються у вигляді лінійних нерівностей або рівнянь:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1$$

Обмеження визначають допустиму область (множину рішень), в межах якої шукається оптимальне значення цільової функції.

2. Чітко визначена цільова функція (objective function)

Ціль задачі - максимізація або мінімізація деякої функції, яка відображає мету моделювання (наприклад, прибуток, витрати, час, ефективність). Цільова функція повинна бути лінійною функцією від змінних рішення:

$$Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

3. Лінійність зв'язків між змінними

Всі вирази в задачі - як у цільовій функції, так і в обмеженнях - мають бути лінійними. [1] Це означає, що:

- змінні не підносяться до степенів вище 1;
- не містять добутків змінних між собою;
- не включають нелінійні функції (наприклад, log, exp, sqrt тощо).

4. Скінченність множини значень (finite input/output)

У задачі мають бути задані кінцеві (скінченні) числові значення для всіх коефіцієнтів, правих частин і меж змінних. Наявність нескінченностей або невизначених значень робить задачу некоректною для розв'язання стандартними методами ЛП. [1]

5. Умова невід'ємності (non-negativity)

Змінні рішення, як правило, повинні бути невід'ємними. [1] Це відповідає реальним обмеженням у задачах ресурсного розподілу, виробництва:

$$x_j \geq 0, \quad \text{для всіх } j = 1, \dots, n$$

6. Залежність від змінних рішення (decision variables)

Ключова роль у задачі належить змінним рішення (decision variables) - саме від них залежить результат оптимізації. [1] Завданням дослідника є визначити ці змінні та знайти їх оптимальні значення, які максимізують або мінімізують цільову функцію з урахуванням заданих обмежень.

Задача вважається задачею лінійного програмування лише тоді, коли всі перелічені характеристики дотримані одночасно. Лише за цих умов можна застосовувати такі методи, як симплекс-метод, двоїстий симплекс,

метод потенціалів тощо. Порушення хоча б однієї з вимог переводить задачу у клас нелінійного, цілочислового або іншого типу програмування.

1.4. Геометрична інтерпретація (для 2D, 3D)

1.4.1. Загальний концепт

Геометричний підхід до розв’язання задачі лінійного програмування полягає у візуалізації допустимої області рішень (feasible region) та цільової функції у вигляді геометричних об’єктів у двовимірному або тривимірному просторі. Цей підхід дозволяє наочно продемонструвати логіку знаходження оптимального розв’язку та механізм роботи таких методів, як симплекс-алгоритм.

Основні елементи геометричної інтерпретації включають:

1. Допустима область (feasible region)

Це область, яка утворена перетином всіх лінійних обмежень задачі. У двовимірному випадку - це опуклий багатокутник, у загальному - опукла багатогранна множина (polyhedron). Усі можливі рішення задачі належать цій області.

2. Цільова функція (objective function)

Вона представлена у вигляді сімейства паралельних прямих або площин. Оптимізація полягає в зсуві цієї прямої (або гіперплощини) в напрямку поліпшення значення цільової функції до останньої точки дотику з допустимою областю.

3. Оптимальне рішення

За геометричним підходом, оптимум досягається на одній із вершин (extreme points) допустимої області. Це випливає з властивості опуклих множин: якщо максимум (або мінімум) існує, то він досягається на вершині.

Геометричний підхід для двох змінних (2D) та трьох (3D) дещо відрізняється

1.4.2. 2D

У задачі з двома змінними допустима область визначається системою лінійних нерівностей, які утворюють опуклий багатокутник у першому квадранті (якщо $x_1, x_2 \geq 0$). Кожне обмеження задає напівплощину, і перетин усіх таких напівплощин формує допустиму область.

Цільова функція $z = c_1x1 + c_2x_2$ геометрично подається як пряма рівня z = const, яка паралельно зсувається у напрямку зростання (при максимізації) або спадання (при мінімізації), поки не «зіткнеться» з допустимою областю. Точка дотику (одна з вершин багатокутника) буде оптимальним розв'язком. Оптимум завжди досягається на вершині багатокутника, якщо така вершина існує в межах допустимої області.

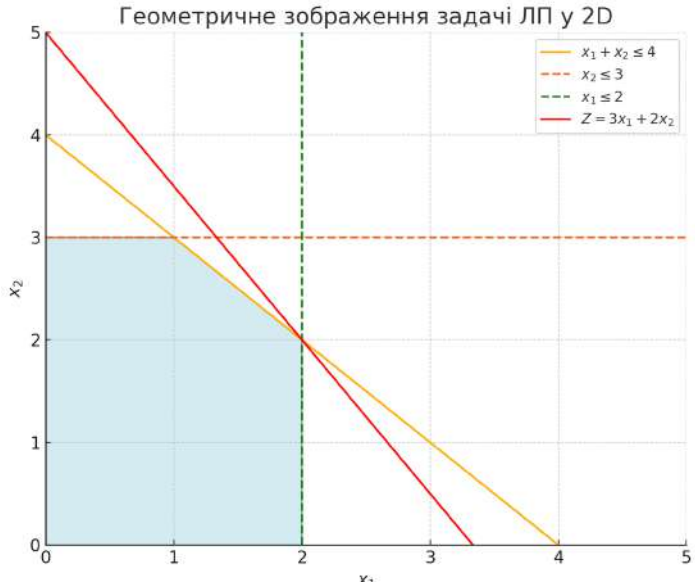


Рисунок 1. Геометричне зображення задачі ЛП у 2D

1.4.3. 3D

У задачах з трьома змінними допустима область є опуклим багатогранником (polyhedron) у тривимірному просторі. Обмеження задають площини, а перетин усіх таких площин формує тривимірну фігуру, всередині якої знаходиться допустима множина.

Цільова функція $z = c_1x1 + c_2x_2 + c_3x_3$ у цьому випадку подається як паралельна площина, яка поступово зсувається в напрямку оптимального значення. Оптимум знову досягається на вершині або ребрі багатогранника, залежно від напрямку зсуву цільової площини. Хоча візуалізація складніша,

ніж для 2D, принцип лишається той самий - оптимальне рішення розташоване на межі допустимої області, найчастіше на вершині багатогранника.

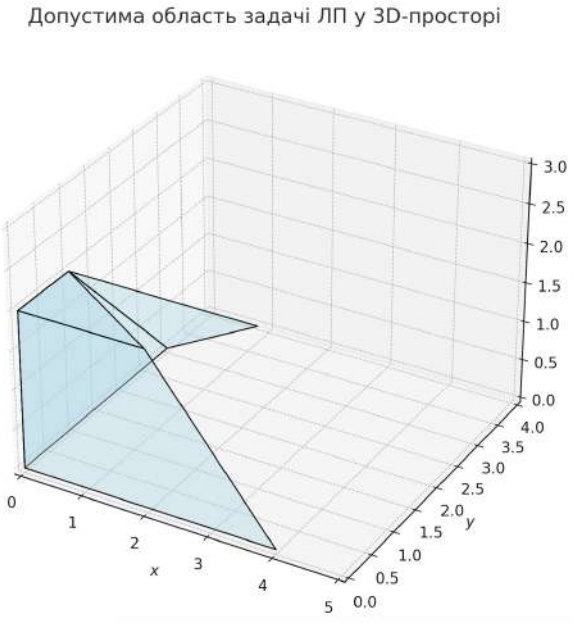


Рисунок 2. Геометричне зображення задачі ЛП у 3D

1.5. Симплекс-метод: історія, суть, алгоритм

1.5.1. Історія

Ідея симплекс-методу виникла в 1947 році, коли Джордж Данциг запропонував новий підхід до розв’язання задач оптимізації з лінійними обмеженнями. Спочатку він інтуїтивно відкинув ідею послідовного проходження по вершинах багатогранника (edge-walking), оскільки вона здавалася неефективною, й намагався знайти методи, які б працювали через внутрішні точки множини допустимих рішень. [9]

Однак саме цей геометричний підхід - переміщення від однієї вершини до іншої вздовж меж багатогранника - і став основою симплекс-методу, який згодом довів свою ефективність. [10]

До 1947 року існувало лише кілька ізольованих праць, присвячених задачам, подібним до лінійного програмування, зокрема роботи Фур’є (1824), де ла Валле Пуссена (1911), Канторовича (1939) та Хічкока (1941). Вони містили ідеї, схожі на симплекс, але не отримали широкого визнання і залишилися невідомими самому Данцигу на момент створення методу.

У 1947 році Данциг вперше запропонував використання моделі лінійних нерівностей для планування великомасштабних процесів - цей підхід швидко набув популярності. Протягом двох років після його публікації була організована перша наукова конференція з математичного програмування в Чикаго, яку очолив Тьяллінг Купманс.

Розвиток симплекс-методу і лінійного програмування співпав із появою цифрових обчислювальних машин. Це дозволило розв’язувати задачі з тисячами змінних і обмежень, які раніше були нереальними через обмеження ручних обчислень.

Поява симплекс-методу стала потужним каталізатором для розвитку таких напрямів як Operations Research та Management Science. Багато вчених, зокрема Джон фон Нейман, Тьяллінг Купманс, Альберт Такер, Герберт Саймон, Пол Самуельсон і Кеннет Ерроу, зробили значний внесок у теоретичну і практичну базу лінійного програмування. [9]

Данциг згадував, що коли він вперше запропонував симплекс-метод, для нього було важливіше зрозуміти геометрію допустимої області, ніж просто "спускатися" по цільовій функції, як це роблять методи внутрішніх точок. [10] Хоча йому здавалося, що метод "по ребрах" є непродуктивним, саме він і став фундаментом ефективного розв’язання ЗЛП. [9]

1.5.2. Ідея симплекс-методу

Симплекс-метод - це алгоритм розв’язання задачі лінійного програмування, що базується на геометричній властивості: якщо оптимум задачі існує, то він досягається в одній з вершин (тобто кутових точках) багатогранника допустимих розв’язків.

Ідея методу полягає в ітеративному переході від однієї вершини до іншої суміжної вершини вздовж ребра багатогранника, таким чином, що значення цільової функції при кожному кроці не погіршується. Алгоритм завершується, коли досягнуто вершину, в якій неможливо покращити значення цільової функції - тобто досягнуто оптимуму.

На кожному кроці виконується так зване “покращення базису” - тобто заміна однієї змінної, що входить до базису, на іншу, що не входить, з метою підвищення (для задачі максимізації) або зниження (для задачі мінімізації) значення цільової функції.

Для зручності реалізації симплекс-методу використовують симплекс-таблицю - це компактна таблична форма, яка містить усю необхідну інформацію для ітерацій алгоритму. Вона дозволяє уникнути повторного розв'язання системи рівнянь на кожному кроці.

Типова симплекс-таблиця включає:

- Коефіцієнти обмежень: кожен рядок відповідає одному обмеженню у стандартній формі.
- Базисні змінні: змінні, що наразі належать до базису, вказуються в першому стовпці.
- Вільні члени: значення правих частин обмежень.
- Рядок оцінок (z-рядок або Δ-рядок): містить величини, що дозволяють оцінити вплив введення нових змінних до базису.
- Цільова функція: її значення оновлюється на кожній ітерації.

Симплекс-таблиця дозволяє визначити:

1. Яку змінну вводити в базис (вхідна змінна);
2. Яку змінну виводити з базису (вихідна змінна);
3. Як оновити значення таблиці (проведення повороту або переобчислення).

Симплекс-таблиці є дуже помічними при розв'язуванні задач за домопогою симплекс-методу.

Базовий алгоритм симплекс-методу застосовують до задач максимізації:

$$\max Z = c^T x$$

У такій задачі симплекс-таблиці будується з оцінками $\Delta_j = c_j - z_j$, і вибираються найбільші позитивні значення, щоб покращити Z.

Якщо ж потрібно розв'язати задачу мінімізацію, застосовуються два основі підходи:

- **Зміна знаку цільової функції.**

У задачі вигляду

$$\min Z = c^T x$$

Цільову функцію множимо на –1 і розв'язуємо як задачу максимізації:

$$\max(-Z) = -c^T x$$

Після знаходження максимального значення $-Z^*$ повертаємось до оригінальної цільової функції:

$$Z^* = -(-Z^*) = \text{оптимум мінімізації}$$

- **Використання двоїстої задачі (Dual Problem)**

Мінімізацію можна розв'язати, сформулювавши двоїсту задачу. Якщо пряма задача має вигляд:

$$\min c^T x \quad \text{за умови } Ax \geq b, \text{ x} \geq 0$$

То її дуальна форма:

$$\max b^T y \quad \text{за умови } A^T y \leq c, \text{ x} \geq 0$$

1.5.3. Алгоритм

Після того, як була сформована цільова функція та була приведена до канонічної форми, застосовується сам симплекс-метод:

Крок 1. Формування початкової симплекс-таблиці

Створюється таблиця, де:

- у верхній частині розташовані коефіцієнти обмежень та праві частини,
- нижній рядок містить оцінки приросту цільової функції:

$$\Delta_j = c_j - z_j, \qquad \text{де } z_j = \sum_i c_{B_i} \cdot a_{ij}$$

Тут:

$c_j -$ коефіцієнт цільової функції для змінної x_j,

$c_{B_i} -$ коефіцієнт базисної змінної у цільовій функції,

$a_{ij} -$ коефіцієнт змінної x_j в обмеженні i

Приклад симплекс-таблиці:

Базис	x_1	x_2	s_1	s_2	RHS
s_1	1	2	1	0	6
s_2	3	2	0	1	12
Z	-3	-5	0	0	0

Крок 2. Перевірка оптимальності

- Для задачі максимізації: якщо всі $\Delta_j \leq 0$, поточне рішення - оптимальне.
- Для задачі мінімізації: якщо всі $\Delta_j \geq 0$, поточне рішення - оптимальне.

Якщо це не так - переходимо до наступного кроку.

Крок 3: Вибір ведучого стовпця (вхідної змінної)

Обирається змінна, яка дає найбільший приріст цільової функції:

- Для максимізації - обираємо j^*, де $\Delta_{j^*} = \max(\Delta_j), \quad \Delta_j > 0$
- Для мінімізації - обираємо j^*, де $\Delta_{j^*} = \min(\Delta_j), \quad \Delta_j < 0$

Це змінна, яка входить у базис.

Крок 4: Вибір ведучого рядка (вихідної змінної)

Розраховується співвідношення для кожного обмеження:

$$\theta_i = \frac{b_i}{a_{ij^*}}, \quad \text{де } a_{ij^*} > 0$$

Рядок з найменшим позитивним значенням θ_i вказує, яка змінна виходить із базису. Це гарантує збереження допустимості рішення.

Крок 5: Проведення симплекс-кроку (перетворення таблиці)

Проводяться елементарні перетворення для оновлення таблиці:

- Ведучий елемент (на перетині ведучого рядка та стовпця) нормалізується до 1.
- Усі інші елементи ведучого стовпця обнуляються, включно з рядком цільової функції.

Крок 6: Повторення алгоритму

Після оновлення таблиці повертаємося до кроку 2. Ітерації тривають, поки не будуть виконані умови оптимальності (залежно від типу задачі).

Крок 7: Отримання оптимального рішення

Після завершення ітерацій у таблиці:

- базисні змінні матимуть конкретні значення у стовпці
- небазисні змінні дорівнюватимуть нулю,
- всі оцінки приросту задовільняють критерій оптимальності
- останнє значення в рядку цільової функції – це оптимальне значення

1.5.4. Випадки виродження, нескінченності, відсутності розв'язку

У процесі реалізації симплекс-методу можуть виникати особливі ситуації, які впливають на хід обчислень або навіть унеможливлюють знаходження оптимального розв'язку. Розглянемо ключові з них: випадки виродження, нескінченності та відсутності розв'язку.

Виродження (degeneracy)

Виродженим називається базисне розв'язання, в якому принаймні одна базисна змінна має нульове значення, тобто:

$$x_{B_i} = 0 \quad \text{для деякого } i$$

Це може спричинити зациклення симплекс-методу, коли алгоритм повертається до вже відвіданого базису, не покращуючи значення цільової

функції. Така ситуація виникає, коли кілька обмежень одночасно активні (перетинаються в одній точці).

Методи усунення зациклення:

- Правило Бленда (Bland's Rule): завжди вибирати змінну з найменшим індексом для входу та виходу з базису;
- Використання антивироджувальних стратегій, які забезпечують просування до нових вершин.

Нескінченність (unboundedness)

Задача лінійного програмування є необмеженою, якщо цільова функція може зростати (при максимізації) або спадати (при мінімізації) без межі у припустимій області. Це спостерігається тоді, коли на якомусь кроці симплекс-методу:

$$\Delta_k > 0, \quad \text{але } a_{ik} \leq 0 \ \forall i$$

Тобто не існує жодного обмеження, яке б обмежувало зростання змінної, яка покращує цільову функцію. У такому разі алгоритм повідомляє про необмеженість задачі.

Відсутність розв'язку (infeasibility)

Задача називається несумісною або такою, що не має розв'язку, якщо система обмежень не має жодного спільного розв'язку. Тобто множина допустимих рішень порожня:

$$\{x \in R^n \mid Ax \leq b,\ x \geq 0\} = \emptyset$$

Найчастіше така ситуація виявляється у штучній фазі симплекс-методу або при побудові двохфазного симплекс-алгоритму (варіант симплекс-методу, для задах, у яких не вдається знайти початкове допустиме базисне рішення), коли оптимальне значення допоміжної задачі не дорівнює нулю.

1.6. Огляд сучасних інструментів реалізації ЛП у Python

1.6.1. PuLP

PuLP – це потужна Python-бібліотека для побудови та розв'язання задач лінійного і змішаного цілочисельного програмування (MILP). Вона дозволяє формулювати оптимізаційні моделі у декларативному стилі, близькому до математичного запису, і взаємодіє з широким набором сторонніх солверів, зокрема GLPK, CBC, HiGHS, CPLEX, GUROBI тощо.

Бібліотека PuLP є частиною проєкту COIN-OR (Computational Infrastructure for Operations Research), який об'єднує відкриті інструменти для дослідження операцій. [11]

Основні компоненти PuLP

1. Створення задачі: LpProblem

Для формулювання задачі використовується клас LpProblem. Він приймає два основні параметри:

name – ім'я задачі (опціональне, але корисне при збереженні у файл);

sense – напрямок оптимізації: LpMinimize (мінімізація, значення 1) або LpMaximize (максимізація, значення -1).

```
model = LpProblem("Transportation_Model", LpMinimize)
```

2. Змінні: LpVariable

Змінні створюються через клас LpVariable, який дозволяє вказати:

- name – назва змінної;
- lowBound / upBound – межі;
- cat – категорія змінної:
- LpContinuous – неперервна змінна,
- LpInteger – цілочисельна змінна,
- LpBinary – булева змінна (0 або 1).

```
x = LpVariable("x", lowBound=0)
y = LpVariable("y", upBound=5, cat=LpInteger)
```

Для масових задач змінні можна створювати у вигляді словників:

```
x = LpVariable.dicts("x", indices=[(i, j) for i in range(m) for j in range(n)], lowBound=0)
```

3. Цільова функція

Цільова функція задається як звичайне алгебраїчне вираження. Для зручності можна використовувати lpSum():

```
model += lpSum(costs[i][j] * x[i, j] for i in range(m) for j in range(n)), "Total_Cost"
```

4. Обмеження: LpConstraint

Обмеження створюються за допомогою операторів <=, >=, ==:

```
model += lpSum(x[i, j] for j in range(n)) <= supply[i], f"Supply_Constraint_{i}"
```

Кожне обмеження можна іменувати, що полегшує аналіз.

Можливості PuLP:

1. **Вибір солвера під час виклику методу solve():**

   ```
   model.solve(pulp.PULP_CBC_CMD())
   ```

2. **Вивід статусу рішення задачі за допомогою model.status:**

   ```
   from pulp import LpStatus
   print("Status:", LpStatus[model.status])
   ```

   Всього бібліотека має 5 статусів для розв’язку задач:

Назва статусу	**Статус**	**Опис**
LpStatusOptimal	1	Оптимальне рішення знайдено
LpStatusNotSolved	0	Рішення не було знайдено
LpStatusInfeasible	-1	Задача не має розв’язку
LpStatusUnbounded	-2	Розв’язок не обмежений
LpStatusUndefined	-3	Невизначений результат

3. **Запис та збереження моделей.**
   Для налагодження та інтеграції PuLP дозволяє зберігати моделі у стандартних форматах:

- .lp – текстовий формат для LP-солверів (читабельний людиною),
- .mps – машинний формат (більш компактний),
- .json – формат для серіалізації задачі.

```
model.writeLP("model.lp")
model.writeMPS("model.mps")
model.toJson("model.json")
```

Або

```
vars, loaded_model = LpProblem.fromJson("model.json")
```

4. **Отримання та виведення результатів за допомогою функції value().**

```
from pulp import value
print("x =", value(x), "y =", value(y))
```

Інші додаткові можливості включають:

– Elastic constraints – перетворення жорстких обмежень у еластичні (з введенням штрафу).
– Комбінаторику – функції allcombinations, allpermutations допомагають будувати комбінаторні задачі.
– Warm Start – можливість ініціалізувати змінні початковими значеннями.
– Категоризацію – cat='Integer', cat='Binary', cat='Continuous'.

Приклад повної реалізації базової задачі лінійного програмування за допомогою бібліотеки PuLP:

```python
from pulp import *

model = LpProblem("Simple_Model", LpMinimize)

x = LpVariable("x", lowBound=0)
y = LpVariable("y", lowBound=0, upBound=5, cat='Integer')

model += 3 * x + 4 * y, "Objective"
model += x + 2 * y >= 8, "Constraint_1"
model += x - y <= 3, "Constraint_2"

model.solve()
print("Status:", LpStatus[model.status])
print("x =", value(x))
print("y =", value(y))
```

```
Status: Optimal
x = 0.0
y = 4.0
```

1.6.2. SciPy: scipy.optimize.linprog

Функція scipy.optimize.linprog з бібліотеки SciPy є базовим інструментом для розв’язання задач лінійного програмування у Python. Вона надає можливість мінімізувати лінійну цільову функцію за наявності лінійних рівнянь, нерівностей та обмежень на змінні.

Цей інструмент орієнтований на низькорівневу роботу з числовими масивами та дозволяє гнучко налаштовувати методи розв’язання та параметри обчислень. [12]

Основні параметри функції включають:

- c – коефіцієнти цільової функції (1D-масив).
- A_ub, b_ub – система нерівностей.
- A_eq, b_eq – система рівнянь.
- bounds – обмеження для кожної змінної у вигляді пари (min, max).
- method – вибір алгоритму розв’язання: 'highs' (за замовчуванням), 'highs-ds', 'highs-ipm', 'interior-point', 'revised simplex' тощо.

- options – словник із налаштуваннями, наприклад {"disp": True} для виводу проміжних результатів.
- callback, x0, integrality – додаткові параметри для специфічних методів (наприклад, підтримка цілочисельних змінних в HiGHS).

Методи розв’язання.

З останніх версій SciPy основними рекомендованими методами є:

- highs – автоматичний вибір між highs-ds (dual simplex) та highs-ipm (interior point).
- highs-ds – реалізація симплекс-методу з подвійною симплексною стратегією.
- highs-ipm – інтер’єрний метод з покращеною точністю.

Методи 'simplex', 'revised simplex' та 'interior-point' вважаються застарілими та не рекомендуються для нових проєктів.

Додаткові можливості бібліотеки включають:

- Presolve – автоматичне спрощення задачі перед розв’язанням: видалення нульових рядків, фіксованих змінних, дубльованих обмежень.
- Автомасштабування (autoscale) – корисне при великій різниці між значеннями коефіцієнтів.
- Redundancy Removal (rr) – автоматичне видалення надлишкових обмежень.
- Інтегральні обмеження (integrality) – за підтримки HiGHS.

Приклад реалізації задачі за допомогою бібліотеки SciPy:

```python
from scipy.optimize import linprog
from scipy.optimize import OptimizeResult

# Цільова функція (мінімізуємо -прибуток для max)
c = [-30, -20]

# Обмеження у вигляді A_ub x ≤ b_ub
A_ub = [
    [1, 1],    # x + y ≤ 100
    [1, 0],    # x ≤ 40
    [0, -1]    # -y ≤ -30  → y ≥ 30
]
b_ub = [100, 40, -30]

# Межі для змінних
bounds = [(0, None), (0, None)]

# Розв'язання задачі
result: OptimizeResult = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method="highs")

# Виведення результатів
print("=== Результати оптимізації ===")
print("Статус:", result.message)
print("Код статусу:", result.status)

if result.success:
    print(f"Оптимальний розподіл часу:")
    print(f" - Проєкт A: {result.x[0]:.2f} годин")
    print(f" - Проєкт B: {result.x[1]:.2f} годин")
    print(f"Максимальний прибуток: {-result.fun:.2f}")
else:
    print("Рішення не знайдено.")
```

```
=== Результати оптимізації ===
Статус: Optimization terminated successfully. (HiGHS Status 7: Optimal)
Код статусу: 0
Оптимальний розподіл часу:
 - Проєкт A: 40.00 годин
 - Проєкт B: 60.00 годин
Максимальний прибуток: 2400.00
```

1.7. Висновок до розділу 1

У цьому розділі було проведено комплексний огляд ключових теоретичних основ лінійного програмування (ЛП) – однієї з центральних моделей у сфері математичної оптимізації. Розглянуто як загальні принципи побудови ЛП-моделей, так і специфічні приклади прикладного використання в логістиці, плануванні, розподілі ресурсів та прийнятті управлінських рішень.

Задача лінійного програмування була формалізована як процес максимізації або мінімізації лінійної цільової функції за умов лінійних обмежень і невід’ємності змінних. Наведена математична структура задачі дозволяє описати широке коло практичних ситуацій з використанням

чітких і вимірюваних параметрів. Було показано, що успішне застосування ЛП вимагає дотримання кількох важливих умов – лінійності, обмеженості, скінченності даних та наявності визначених змінних рішень.

Окрему увагу було приділено прикладам класичних задач, таких як транспортна задача, задача про призначення, цілочислове програмування та багатоцільове лінійне програмування (MOLP). Це дозволило окреслити широту застосування ЛП: від оптимізації перевезень і виробництва до управління проєктами, ресурсами та стратегічного планування. Багатоцільові та нечіткі моделі розширюють межі ЛП, даючи змогу враховувати суперечливі цілі, невизначеність та нечіткі критерії, що є надзвичайно актуальними для задач у гуманітарній логістиці, соціальному управлінні та освітньому плануванні.

Описано геометричну інтерпретацію задач ЛП, яка дає інтуїтивне розуміння структури допустимої області та принципу досягнення оптимуму на вершинах багатогранника. Це розуміння лягло в основу симплекс-методу – основного алгоритму розв’язання задач ЛП, який ітеративно переходить від вершини до вершини, покращуючи значення цільової функції. Детально розглянуто історію симплекс-методу, його базову логіку, побудову симплекс-таблиць, критерії оптимальності, а також випадки виродження, нескінченності й відсутності розв’язку.

Окремий підрозділ присвячено сучасним інструментам реалізації ЛП у Python. Зокрема, було порівняно бібліотеки PuLP і scipy.optimize.linprog, які дозволяють реалізовувати моделі різної складності – від базових до багатоцільових, з можливістю використання цілочислових, бінарних змінних і підтримкою зовнішніх солверів. Ці інструменти роблять лінійне програмування доступним для широкого кола дослідників, практиків та розробників.

Отже, лінійне програмування – це потужний інструмент прийняття рішень в умовах обмежених ресурсів і конфліктних цілей. Теоретична база, описана в цьому розділі, формує основу для практичної реалізації складних моделей оптимізації, які будуть розглянуті у подальших розділах цієї роботи, зокрема в контексті задач гуманітарної логістики під час війни.

РОЗДІЛ 2. ФОРМАЛІЗАЦІЯ ЗАДАЧІ ЛОГІСТИЧНОЇ ОПТИМІЗАЦІЇ

2.1. Постановка прикладної задачі.

У сучасних умовах повномасштабної війни на території України особливої ваги набуває ефективне планування постачання гуманітарної допомоги до постраждалих регіонів. Одним із ключових завдань стає організація логістики в умовах обмежених ресурсів – як матеріальних, так і часових – із урахуванням ризиків, пов’язаних із безпекою маршрутів.

У межах цієї роботи розглядається прикладна задача, яка імітує процес розподілу гуманітарних вантажів із кількох логістичних центрів (складів) до населених пунктів, що потребують допомоги. Кожен склад має обмежений обсяг наявних ресурсів, а кожен населений пункт – заздалегідь визначений рівень потреб.

Ключовими факторами, які впливають на прийняття рішень щодо вибору маршрутів постачання, є:

- вартість доставки – враховує фінансові витрати на переміщення ресурсів між пунктами,
- час доставки – критичний для забезпечення своєчасної підтримки, особливо у випадках із загрозою гуманітарної катастрофи,
- ризик маршруту – пов’язаний із безпековими викликами, такими як бойові дії, заміновані території, або відсутність стабільного транспортного сполучення.

У результаті планування необхідно знайти таку схему розподілу вантажів, яка б дозволила задовольнити потреби населення, не перевищуючи наявні запаси ресурсів і водночас мінімізуючи сумарний показник "вартості-ризику-часу". Важливою є також можливість змінювати пріоритети між цими трьома чинниками відповідно до конкретної гуманітарної ситуації: наприклад, у певних умовах ризик може бути критичнішим за вартість, або навпаки.

Розроблена модель дозволяє досліджувати різні сценарії логістики та порівнювати результати за допомогою трьох підходів до розв'язання задачі: за допомогою бібліотеки PuLP, функції linprog із SciPy, а також власноручної реалізації симплекс-методу, що дає змогу здійснити мікроаналіз чутливості до зміни параметрів задачі.

2.2. Побудова математичної моделі

На основі описаної логістичної ситуації будується математична модель лінійного програмування, яка дозволяє визначити оптимальну схему транспортування гуманітарної допомоги з урахуванням трьох основних критеріїв: вартості, ризику та часу доставки. Модель враховує обмежені ресурси складів, потреби населених пунктів, а також дозволяє варіювати вагу кожного з критеріїв залежно від гуманітарного контексту.

2.1.1. Змінні

Вводиться набір змінних:

x_{ij} - кількість одиниць гуманітарної допомоги, що доставляється зі складу i до пункту призначення j. Змінні x_{ij} є невід'ємними і визначають, як розподіляються ресурси між складами та пунктами.

2.1.2. Цільова функція

Метою є мінімізація зваженої суми трьох показників – вартості, ризику та часу транспортування:

- Кожен маршрут (i, j) має відповідні коефіцієнти: c_{ij} - вартість , r_{ij} - ризик, t_{ij} – час
- Враховується вагове значення кожного з критеріїв: α, β, γ, що відповідають пріоритетності економічного, безпекового та часовго факторів.

Таким чином, цільова функція має вигляд:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (\alpha \cdot c_{ij} + \beta \cdot r_{ij} + \gamma \cdot t_{ij}) \cdot x_{ij}$$

де:

α, β, γ – вагові коефіцієнти,
що визначають важливість кожного з критеріїв (вартість, ризик, час),

c_{ij} , r_{ij} , t_{ij} – відповідні значення для кожного маршруту доставки,

x_{ij} – змінна, що визначає обсяг доставки з точки i в точку j

2.1.3. Система обмежень

Модель включає стандартні логістичні обмеження:

1. **Обмеження на запаси кожного складу.**
   Кількість відправлених ресурсів не повинна перевищувати наявний запас:

$$\sum_{j=1}^{n} x_{ij} = S_i, \quad \forall i = 1, \dots, m$$

2. **Обмеження на задоволення потреб кожного пункту призначення.**
   Сумарна кількість отриманих ресурсів має відповідати потребі:

$$\sum_{i=1}^{m} x_{ij} = D_j, \quad \forall j = 1, \dots, n$$

3. **Обмеження на невід'ємність:**

$$x_{ij} \geq 0, \quad \forall i, j$$

Ця модель дозволяє задавати гнучкі сценарії: змінюючи значення параметрів α, β, γ, можна моделювати ситуації, коли, наприклад, пріоритетним є мінімізація часу, а не вартості, або коли ключовим є уникнення ризикових напрямків.

2.3. Багатокритеріальний підхід: зважування критеріїв

У задачах гуманітарної логістики ключове значення має не лише ефективність постачання, але й якість маршруту з точки зору ризику та часу. Проте ці критерії часто суперечать один одному: безпечні маршрути можуть бути дорожчими чи тривалішими, тоді як швидкі маршрути можуть проходити через зони підвищеної небезпеки.

Саме тому доцільним є використання багатокритеріального підходу, який дозволяє врахувати кілька факторів одночасно. У межах цієї роботи реалізовано підхід на основі зважування критеріїв, коли всі критерії об’єднуються в одну агреговану цільову функцію.

2.3.1. Суть зважування

Замість того, щоб оптимізувати лише один параметр (наприклад, мінімізувати вартість), ми комбінуємо три критерії в єдину функцію за допомогою вагових коефіцієнтів:

- α – ваговий коефіцієнт для вартості доставки,
- β – ваговий коефіцієнт для ризику маршруту,
- γ – ваговий коефіцієнт для часу транспортування.

Ці ваги не обов’язково мають сумуватись до 1, але їх співвідношення визначає "погляд" моделі на бажаний тип рішення. Значення можуть бути підібрані експертно або варіативно протестовані.

2.3.2. Типові сценарії гуманітарної логістики

У межах цієї роботи було реалізовано аналіз чутливості рішення до зміни вагових коефіцієнтів. Розглянемо типові сценарії, які можуть мати місце в реальних умовах:

- Сценарій "економії бюджету"
    - Високе значення α, низькі β та γ: модель зосереджується на мінімізації вартості транспортування.
    - Типовий для місій з обмеженим фінансуванням або при наявності великої кількості вантажів.
- Сценарій "оперативного реагування"

    - Високе γ, середнє α, низьке β: пріоритет – мінімізувати час доставки за прийнятного ризику і вартості.
    - Актуально у разі нагальних потреб, наприклад, доставки води в зону стихійного лиха.
- Сценарій "максимальної безпеки"
    - Високе β, низьке α і γ: модель уникає небезпечних маршрутів, навіть якщо це дорожче чи довше.
    - Використовується при загрозі обстрілів, блокпостів чи мінування доріг.
- Збалансований сценарій (компроміс)
    - Приблизно однакові ваги: α = β = γ = 1.0: Приблизно однакові ваги: дає зважене рішення, яке не надто жертвує жодним із критеріїв.
    - Найчастіше використовується для базового планування, коли ситуація стабільна.

Такий підхід до задачі дозволяє:

- моделювати та оцінювати різні стратегії логістики в мінливих умовах,
- відповідати на запит конкретних гуманітарних місій, підбираючи найдоцільнішу конфігурацію ваг,
- проводити мікроаналіз чутливості, який демонструє, як навіть незначна зміна ваг може вплинути на розв’язок.

2.4. Підготовка тестових даних та сценаріїв

Для перевірки працездатності математичної моделі та подальшого аналізу її результатів необхідно сформувати репрезентативні тестові дані, що відповідають реалістичним умовам гуманітарної логістики. У цьому дослідженні було підготовлено два набори даних різного масштабу, які імітують типові логістичні задачі з різною складністю.

Мета підготовки тестових даних полягає у тому, щоб:

- верифікувати правильність математичної постановки задачі;
- порівняти точність і швидкодію різних реалізацій (бібліотеки vs ручна реалізація);

– провести мікроаналіз чутливості до зміни ваг;
– візуалізувати вплив сценаріїв на розподіл поставок і цільову функцію.

Тестові дані включають:

- масив запасів S_i
- масив потреб D_j
- матриці вартості, ризику та часу доставки:
  - $C = [c_{ij}]$ – вартість доставки між кожною парою i та j
  - $\mathrm{R} = [\mathrm{r}_{ij}]$ – рівень ризику маршруту
  - $\mathrm{T} = [\mathrm{t}_{ij}]$ – час доставки

Усі матриці мають розмірність $\mathrm{m} \times \mathrm{n}$, де m – кількість складів, n – кількість пунктів призначення.

Для перевірки реалізацій симплекс-методу було використано два тестові набори:

Малий набір – для ілюстрації структури моделі. Містить:

- Склади: 3
- Пункти призначення: 4
- Запаси по складах: [100, 150, 120]
- Потреби пунктів призначення: [80, 100, 90, 100]
- Матриця вартості C (грн):
  [[4, 6, 9, 7],
  [5, 4, 7, 8],
  [6, 7, 4, 5]]
- Матриця ризику R (0-10):
  [[3, 5, 8, 6],
  [4, 3, 5, 7],
  [5, 6, 3, 4]]
- Матриця часу доставки T (год):
  [[12, 18, 25, 20],
  [15, 12, 20, 22],
  [18, 20, 10, 15]]

Великий набір – для симуляції більш масштабного логістичного кейсу. Містить:

- Склади: 6
- Пункти призначення: 8
- Запаси по складах: [120, 140, 100, 90, 130, 110]
- Потреби пунктів призначення: [80, 75, 95, 60, 85, 70, 100, 100]
- Матриця вартості C (грн):
  [[6, 8, 5, 9, 7, 6, 4, 8],
  [5, 7, 6, 8, 5, 5, 6, 9],
  [7, 6, 8, 7, 6, 7, 5, 6],
  [6, 5, 6, 9, 7, 6, 8, 7],
  [5, 8, 7, 6, 7, 5, 6, 5],
  [7, 6, 6, 7, 8, 6, 7, 6]]
- Матриця ризику R (0-10):
  [[4, 5, 4, 6, 3, 5, 3, 4],
  [5, 4, 3, 4, 4, 3, 5, 4],
  [4, 4, 5, 3, 5, 4, 3, 3],
  [3, 5, 4, 4, 3, 5, 4, 5],
  [4, 3, 5, 4, 4, 4, 3, 4],
  [5, 4, 3, 5, 5, 3, 4, 3]]
- Матриця часу доставки T (год):
  [[15, 20, 12, 18, 14, 17, 13, 16],
  [13, 17, 14, 15, 16, 12, 14, 18],
  [18, 14, 17, 16, 15, 18, 13, 17],
  [14, 16, 15, 17, 14, 16, 15, 15],
  [15, 17, 16, 14, 18, 14, 16, 14],
  [16, 14, 13, 16, 17, 15, 14, 15]]

Дані представлені у вигляді масивів Python (numpy.array) з можливістю збереження або імпорту у форматах CSV, Excel або JSON.

Для моделювання різних ситуацій застосовано набір сценаріїв, у яких ваги α, β, γ варіюються. Використання сценаріїв дозволяє:

– оцінити вплив зміни ваг на значення цільової функції,
– порівняти, як кожен із методів реагує на зміну пріоритетів,
– дослідити стабільність структури розв’язку за зміни умов.

Для тестування були застосовані такі конфігурації:

Сценарій	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\gamma}$	**Пріоритет**
Економічний	1.0	0.2	0.1	Мін. вартості
Швидка доставка	0.3	0.1	1.0	Мін. часу
Максимальна безпека	0.2	1.0	0.1	Мін. ризику
Збалансований (нейтральний)	1.0	1.0	1.0	Компроміс

2.5. Висновок до розділу 2

У другому розділі було здійснено повноцінну формалізацію задачі логістичної оптимізації в умовах гуманітарної кризи, з урахуванням складних реалій воєнного часу в Україні. Сформульовано прикладну задачу, яка враховує обмеженість ресурсів, наявність ризиків і часових пріоритетів, притаманних реальним логістичним операціям у постраждалих регіонах.

Побудовано математичну модель у вигляді задачі лінійного програмування, яка включає такі ключові компоненти:

- змінні рішення – кількість вантажів, що транспортуються між конкретними складами та пунктами призначення;
- багатокритеріальна цільова функція, що узагальнює три аспекти: вартість, ризик і час доставки;
- система обмежень, яка гарантує задоволення потреб при збереженні ресурсних і безпекових обмежень.

Одним із головних методологічних рішень стало застосування методу зваженої згортки (weighted sum method), що дозволив інтегрувати три окремі критерії в єдину цільову функцію. Такий підхід є гнучким і дає змогу адаптувати модель до різних логістичних сценаріїв, шляхом зміни вагових коефіцієнтів α, β, γ.

Розроблена модель була протестована на двох типах вхідних даних:

- малий набір, який дозволяє наочно проаналізувати внутрішню структуру задачі;
- великий набір, що імітує реальні умови багатопунктової логістики у кризовій зоні.

Було також сформовано набір сценаріїв планування, що відображають різні гуманітарні пріоритети – від економії бюджету до забезпечення максимальної безпеки постачання. Такий сценарний підхід відкриває можливість для проведення глибокого аналізу чутливості моделі до змін у пріоритетах, що є особливо актуальним у швидкозмінних польових умовах.

Завдяки гнучкості постановки та чіткості формалізації, розроблена модель стала надійною основою для подальшої реалізації та дослідження трьох підходів до її розв’язання в наступному розділі роботи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ АЛГОРИТМІВ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1. Реалізація задачі з використанням бібліотеки PuLP

Для одного з розв'язань сформульованої задачі лінійного програмування було використано бібліотеку PuLP, яка дозволяє будувати та вирішувати задачі оптимізації у декларативному стилі.

Нижче показано реалізацію ключових компонентів:

1. **Змінні рішень**

В моделі створюється $m \times n$ змінних, де кожна з них відповідає транспортуванню від складу i до пункту j.

```
x_vars = [
    pulp.LpVariable(f"x_{i}_{j}", lowBound=0)
    for i in range(num_suppliers)
    for j in range(num_destinations)
]
```

– f"x_{i}_{j}" – унікальне ім'я змінної.
– lowBound=0 – усі змінні є невід'ємними, що відповідає умові $x_{ij} \geq 0$

2. **Цільова функція**
   Використовується метод згортки трьох критеріїв (вартість, ризик, час) у єдиний коефіцієнт. Після згортки матриця коефіцієнтів "сплющується" у вектор c.

```
prob += pulp.lpDot(c, x_vars)
```

– lpDot(c, x_vars) – скалярний добуток зважених коефіцієнтів та змінних.

3. **Обмеження: запаси складів**

Для кожного складу і сума відправленого ресурсу не повинна перевищувати запасу. В моделі – це рівність, якщо доставка точно покриває наявність.

```
for i in range(num_suppliers):
    prob += pulp.lpSum(
        x_vars[i * num_destinations + j]
        for j in range(num_destinations)
    ) == supplies[i]
```

– x_vars[i * n + j] – звернення до змінної x_{ij}
– pulp.lpSum(...) == supplies[i] – ліву частину обмеження формуємо через суму відповідних змінних.

4. **Обмеження: задоволення потреб пунктів призначення.**
   Аналогічно, для кожного пункту j має бути забезпечена відповідна потреба:

```
for j in range(num_destinations):
    prob += pulp.lpSum(
        x_vars[i * num_destinations + j]
        for i in range(num_suppliers)
    ) == demands[j]
```

Вивід результатів

При виводі результатів користувач може бачити:

- Статус та код статусу задачі
- Значення цільової функції
- Значення змінних

```
print(f"[PuLP] Статус: {status_str}")
print(f"[PuLP] Z = {pulp.value(prob.objective):.2f}")
for var in x_vars:
    if var.varValue > 1e-6:
        print(f"    {var.name} = {var.varValue:.2f}")
```

Приклад виведених результатів:

```
[PuLP] Статус: Optimal
[PuLP] Z = 3192.00
    x_0_0 = 80.00
    x_0_3 = 20.00
    x_1_1 = 100.00
    x_1_3 = 50.00
    x_2_2 = 90.00
    x_2_3 = 30.00
```

3.2. Реалізація задачі з використанням SciPy.optimize.linprog

Другим способом реалізації симплекс-методу було обрано бібліотеку SciPy.optimize.linprog.

Метод, який ініціалізує задачу, виглядає таким чином:

```
def solve_with_scipy(c, A_eq, b_eq, bounds):
    return linprog(
        c=c,                # Вектор коефіцієнтів цільової функції
        A_eq=A_eq,          # Матриця рівнянь
        b_eq=b_eq,          # Вектор правих частин
        bounds=bounds,      # Обмеження змінних
        method='highs'      # Сучасний розв'язувач (HiGHS)
    )
```

Основні компоненти коду:

1. **Вектор цільової функції c.**
   Цей вектор формується як зважена згортка трьох критеріїв:

```
c = (alpha * costs + beta * risks + gamma * times).flatten()
```

– costs, risks, times – матриці розміром $m \times n$
– alpha, beta, gamma – вагові коефіцієнти
– .flatten() – перетворення матриці $m \times n$ на вектор довжини $m \cdot n$

2. **Матриця A_eq і вектор b_eq,**
   Ці об’єкти задають систему обмежень:
   – Для кожного складу: сумарна відправка дорівнює запасу.
   – Для кожного пункту призначення: сумарне отримання дорівнює потребі.

```
def build_constraints(supplies, demands):
    m, n = len(supplies), len(demands)
    A_eq, b_eq = [], []

    # Склади: ∑ x_{ij} = S_i
    for i in range(m):
        row = [0] * (m * n)
        for j in range(n):
            row[i * n + j] = 1
        A_eq.append(row)
        b_eq.append(supplies[i])

    # Пункти: ∑ x_{ij} = D_j
    for j in range(n):
        row = [0] * (m * n)
        for i in range(m):
            row[i * n + j] = 1
        A_eq.append(row)
        b_eq.append(demands[j])

    return np.array(A_eq), np.array(b_eq)
```

3. **Область визначення змінних bounds.**
   Кожна змінна $x_k \geq 0$ – кількість одиниць ресурсу, які можна транспортувати:

```
bounds = [(0, None)] * (num_suppliers * num_destinations)
```

Як і у випадку з реалізацією на PuLP, вивід результатів включає:

- Статус та код статусу задачі
- Значення цільової функції
- Значення змінних

```
res = solve_with_scipy(c, A_eq, b_eq, bounds)
print(f"[SciPy] Статус: {res.message}")
if res.success:
    print(f"[SciPy] Z = {res.fun:.2f}")
    for i, val in enumerate(res.x):
        if val > 1e-6:
            print(f"    x_{i // num_destinations}_{i % num_destinations} = {val:.2f}")
else:
    print("[SciPy] No solution")
```

Приклад виведених результатів:

```
[SciPy] Статус: Optimization terminated successfully. (HiGHS Status 7: Optimal)
[SciPy] Z = 3192.00
    x_0_0 = 80.00
    x_0_3 = 20.00
    x_1_1 = 100.00
    x_1_3 = 50.00
    x_2_2 = 90.00
    x_2_3 = 30.00
```

3.3. Власноруч реалізований симплекс-метод

У межах цього дослідження було реалізовано власну версію симплекс-методу – одного з найпоширеніших алгоритмів розв'язання задач лінійного програмування. Така реалізація дозволяє глибше зрозуміти суть роботи симплекс-алгоритму, перевірити точність бібліотечних рішень і провести мікроаналіз ходу ітерацій.

Алгоритм симплексу у цьому варіанті було реалізовано покроково:

Крок 0: Побудова початкової симплекс-таблиці

Початкова симплекс-таблиця включає:

- A — основна матриця обмежень (розмір $m \times n$),
- I — штучня одинична матриця для додавання штучних змінних,

- b — вектор правих частин,
- $-c$ — оцінки приросту в останньому рядку (для мінімізації)
- Останній стовпець – RHS $(right - hand\ side)$.
- Додаткові m стовпців — штучні змінні s_i,
  які спочатку входять до базису

```
tableau = np.zeros((m + 1, n + m + 1))
tableau[:m, :n] = A
tableau[:m, n:n + m] = np.eye(m)
tableau[:m, -1] = b
tableau[-1, :n] = -c
```

Крок 1: Перевірка оптимальності

```
pivot_col = np.argmin(tableau[-1, :-1])  # Індекс найменшої оцінки приросту
min_z_val = tableau[-1, pivot_col]

if min_z_val >= 0:
    print("Усі оцінки приросту ≥ 0. Оптимальне рішення знайдено.")
    break
```

- pivot_col – індекс вхідної змінної
- Якщо $\min \Delta_j \geq 0$, виходимо з циклу

Крок 2: Розрахунок коефіцієнтів θ

```
ratios = []
for i in range(m):
    a_ik = tableau[i, pivot_col]
    b_i = tableau[i, -1]
    if a_ik > 1e-8:
        theta = b_i / a_ik
        ratios.append(theta)
        print(f"  θ_{i+1} = {b_i:.2f} / {a_ik:.2f} = {theta:.2f}")
    else:
        ratios.append(np.inf)
        print(f"  θ_{i+1} = ∞ (a_ik ≤ 0)")
```

– Якщо $a_{ij} \leq 0$, змінна не може вийти з базису (∞).
– ratios – список θ для кожного рядка.

Крок 3: Вибір вихідної змінної

```
pivot_row = np.argmin(ratios)
min_ratio = ratios[pivot_row]

if min_ratio == np.inf:
    raise Exception("Розв'язок не обмежений – усі a_ij ≤ 0")
```

– pivot_row – номер рядка з мінімальним θ
– Якщо всі значення $\theta = \infty$, задача не обмежена (unbounded).

Крок 4: Півот-операція (перетворення таблиці)

```
pivot_elem = tableau[pivot_row, pivot_col]
tableau[pivot_row, :] /= pivot_elem  # Нормалізація

for i in range(m + 1):
    if i != pivot_row:
        tableau[i, :] -= tableau[i, pivot_col] * tableau[pivot_row, :]
```

Після цього змінна x_{j^*} входить до базису, а змінна з i^*-го рядка виходить.

Крок 5: Оновлення базису та історії Z

Оновлення індексу базисної змінної:

```
basis[pivot_row] = pivot_col
```

Збереження історії зміни Z:

```
z_history.append(-tableau[-1, -1] if show_true_Z else tableau[-1, -1])
```

Історія використовується для графічного аналізу.

Завершення циклу:

Алгоритм завершується, коли:

$$\text{всі } \Delta_j \geq 0$$

або

якщо розв’язок необмежений

Тоді побудовується фінальний вектор x і значення цільової функції Z.

Для зручності перевірки проміжних кроків та результатів реалізовано вивід сиплекс-таблиць на кожному кроці, пояснень між таблицями, фінальний результат цільової функції і змінних та графічне представлення «шляху» цільової функції після віднайдення оптимуму.

Симплекс-таблиця та пояснення

```
def print_simplex_tableau(tableau, basis, step):
    ...
    print("Z\t", tableau[-1])
```

Виводяться:

- Симплекс-таблиця
- значення θ
- півот-елемент
- зміна базису

Приклад симплекс-таблиці та пояснень:

```
--- Симплекс-таблиця. Крок 1 ---
Базис   x1     x2     x3     x4     x5     x6     x7     x8     x9     x10    x11    s1     s2     s3     s4     s5     s6     s7     s8     RHS
s2      1.00   1.00   0.00   1.00   0.00   0.00   -1.00  0.00   0.00   0.00   -1.00  0.00   1.00   0.00   0.00   0.00   0.00   -1.00  0.00   10.00
s3      0.00   0.00   0.00   0.00   1.00   1.00   1.00   1.00   0.00   0.00   0.00   0.00   0.00   1.00   0.00   0.00   0.00   0.00   0.00   150.00
s4      0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   1.00   1.00   1.00   1.00   0.00   0.00   1.00   0.00   0.00   0.00   0.00   120.00
s5      1.00   0.00   0.00   0.00   1.00   0.00   0.00   0.00   1.00   0.00   0.00   0.00   0.00   0.00   0.00   1.00   0.00   0.00   0.00   80.00
s6      0.00   1.00   0.00   0.00   0.00   1.00   0.00   0.00   0.00   1.00   0.00   0.00   0.00   0.00   0.00   0.00   1.00   0.00   0.00   100.00
x3      0.00   0.00   1.00   0.00   0.00   0.00   1.00   0.00   0.00   0.00   1.00   0.00   0.00   0.00   0.00   0.00   0.00   1.00   0.00   90.00
s8      0.00   0.00   0.00   1.00   0.00   0.00   0.00   1.00   0.00   0.00   0.00   1.00   0.00   0.00   0.00   0.00   0.00   0.00   1.00   100.00
Z       -7.30  -11.10 0.00   -12.80 -9.20  -7.30  3.90   -14.50 -11.10 -12.80 9.50   -9.20  0.00   0.00   0.00   0.00   0.00   16.40  0.00   1476.00

Розрахунок θ для кожного рядка:
  θ_1 = ∞ (a_ik = 0.00 ≤ 0)
  θ_2 = 150.00 / 1.00 = 150.00 для s2
  θ_3 = ∞ (a_ik = 0.00 ≤ 0)
  θ_4 = ∞ (a_ik = 0.00 ≤ 0)
  θ_5 = ∞ (a_ik = 0.00 ≤ 0)
  θ_6 = ∞ (a_ik = 0.00 ≤ 0)
  θ_7 = 100.00 / 1.00 = 100.00 для s7

=== Пояснення переходу до наступного кроку ===
Вибрано вхідну змінну: x_8 (мінімальна оцінка приросту: -14.50)
Мінімальне θ = 100.00, отже, вихідна змінна: s7
Півот-елемент: 1.00
Оновлюємо базис: замінюємо s7 на x_8
```

Фінальні результати

```python
if "simplex" in modes:
    try:
        label = f"{test_name}, α={alpha}, β={beta}, γ={gamma}"
        x_opt, z_opt = simplex(np.array(A_eq), np.array(b_eq), np.array(c), plot=True, label=label, show_true_Z=True)
        print(f"[Власна реалізація] Z = {z_opt:.2f}")
        for i, val in enumerate(x_opt):
            if val > 1e-6:
                print(f"    x_{i // num_destinations}_{i % num_destinations} = {val:.2f}")
    except Exception as e:
        print(f"[Власна реалізація] {str(e)}")
```

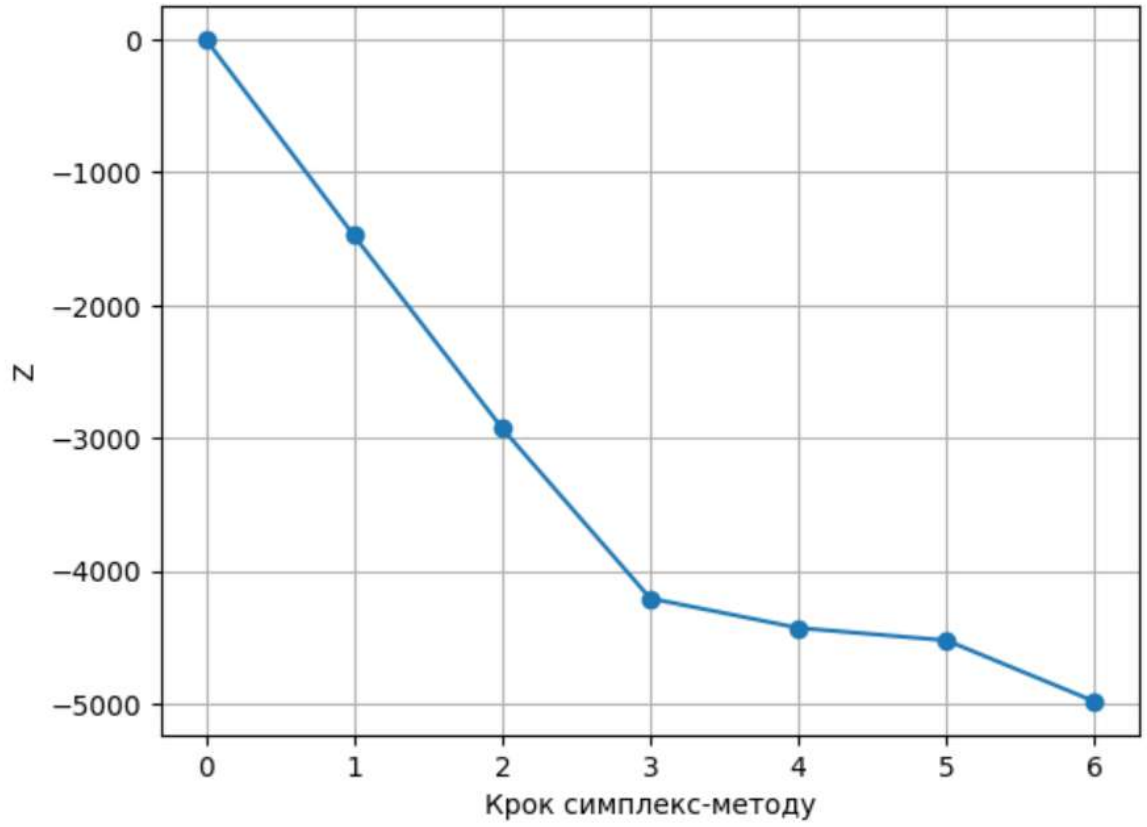
Приклад виводу фінальних результатів:

```
[Власна реалізація] Z = 4982.00
    x_0_1 = 10.00
    x_0_2 = 90.00
    x_1_0 = 50.00
    x_1_3 = 100.00
    x_2_0 = 30.00
    x_2_1 = 90.00
```

Графічне представлення

```
plt.plot(range(len(z_history)), z_history, marker='o')
plt.title(f"Зміна цільової функції Z ({label})")
```

Приклад графічного представлення:



3.4. Інтерпретація результатів на прикладі логістичного кейсу

Фінальні результати усіх трьох версій також демонструють оптимальний план розподілу – набір значень змінних x_{ij}, які вказують, скільки одиниць ресурсу слід доставити з кожного складу до кожного пункту призначення.

Приклад (для великого набору даних у сценарії "максимальна безпека"):

```
[Власна реалізація] Z = 5324.00

x_0_6 = 80.00      — склад 0 → пункт 6 (безпечний, не найдешевший)
x_1_1 = 75.00      — склад 1 → пункт 1
x_2_0 = 100.00     — склад 2 → пункт 0
x_3_3 = 60.00      — склад 3 → пункт 3
x_4_5 = 70.00      — склад 4 → пункт 5
x_5_2 = 95.00      — склад 5 → пункт 2
x_5_4 = 85.00      — склад 5 → пункт 4
x_5_7 = 100.00     — склад 5 → пункт 7
```

Інтерпретація:

- Маршрути з найменшим ризиком обираються навіть за умов вищої вартості або тривалості – відповідно до встановлених ваг.
- Рішення уникає небезпечних напрямків, навіть якщо вони коротші.
- Модель обирає логістичну комбінацію, яка є компромісом між ресурсною досяжністю та безпекою.

3.5. Верифікація результатів

Для підтвердження коректності реалізації власного симплекс-методу було проведено порівняння результатів, отриманих трьома різними способами:

- Власноручна реалізація симплекс-методу (з кроковою побудовою таблиці),
- Бібліотека PuLP (з використанням об’єктної моделі та зовнішнього розв’язувача),
- Функція scipy.optimize.linprog (з методом HiGHS).

Мета верифікації

- Перевірити, що всі методи дають однакове або дуже близьке значення цільової функції.
- Перевірити, що вони дають еквівалентний план доставки, навіть якщо змінні неідентичні – адже розв’язок задачі ЛП може бути не єдиним.

- Віднайти аномалії, що можуть виникати під час розв’язань задач ЛП комп’ютерними засобами

У процесі експериментального дослідження виявлено помітні розбіжності в поведінці бібліотечних реалізацій (PuLP, SciPy) та власного симплекс-методу, особливо при роботі з великим набором даних.

На малому наборі даних:

- Реалізації PuLP, SciPy дали ідентичне значення цільової функції.
- Власна реалізація в деяких випадках обирала інший базис, що призводило до трохи гіршого результату.

На великому наборі даних:

- **SciPy (linprog з методом HiGHS):**
  - Повертає статус infeasible – не існує оптимального розв’язку.
  - Не видає жодного плану доставки.
  - Не дозволяє проаналізувати ситуацію – повертає лише res.success = False.
- **PuLP (розв’язувач CBC):**
  - Повертає розв’язок незважаючи на статус "Infeasible".
  - План доставки порушує обмеження на потреби – перевищено обсяги доставки до деяких пунктів.
  - Навіть за додаткової перевірки на жорсткість обмежень розв’язувач не може знайти оптимального розв’язку.
- **Власна реалізація:**
  - Успішно знаходить допустиме рішення.
  - Не порушує жодного з обмежень.
  - Формує базис вручну, кожен крок можна відслідкувати та перевірити.

Власноручна реалізація симплекс-методу виявилась найбільш стійкою та надійною до особливостей задачі, особливо в умовах великої розмірності.

Бібліотеки можуть давати некоректні або суперечливі результати при:

- недостатній чисельній точності,

- складній структурі обмежень,
- виродженні задачі або нестачі допустимих базисів.

3.6. Мікроаналіз чутливості: вплив коефіцієнтів вартості, ризику та часу

Для виявлення впливу вагових коефіцієнтів α, β, γ на результати оптимізації було проведено мікроаналіз чутливості. Він полягав у покроковій зміні співвідношень критеріїв (вартість, ризик, час доставки) у зведеній цільовій функції, з наступним порівнянням отриманих результатів трьома методами: PuLP, SciPy.optimize.linprog, та власноручною реалізацією симплекс-методу.

Було протестовано чотири основні сценарії:

- Економічний – акцент на мінімізації вартості,
- Швидка доставка – пріоритет часу,
- Максимальна безпека – мінімізація ризику,
- Збалансований (нейтральний) – рівнозначний урахунок усіх трьох факторів.

Результати тестувань:

Сценарій – економічний

$$\alpha = 1.0\,, \beta = 0.2\,, \gamma = 0.1$$

Набір даних	Версія	Z	Коментар
Мала	PuLP	2551	
Мала	SciPy	2551	
Мала	Власна	3974	
Велика	PuLP	5112	Статус infeasible, результат перевищує попит
Велика	SciPy	-	No solution, статус infeasible
Велика	Власна	6935	

Сценарій – швидка доставка

$$\alpha = 0.3\,, \beta = 0.1\,, \gamma = 1.0$$

Набір даних	Версія	Z	Коментар
Мала	PuLP	5861	
Мала	SciPy	5861	
Мала	Власна	8775	
Велика	PuLP	10606	Статус infeasible, результат перевищує попит
Велика	SciPy	-	No solution, статус infeasible
Велика	Власна	13301.50	

Сценарій – максимальна безпека

$$\alpha = 0.2\,, \beta = 1.0\,, \gamma = 0.1$$

Набір даних	Версія	Z	Коментар
Мала	PuLP	2220	
Мала	SciPy	2220	
Мала	Власна	3678	
Велика	PuLP	3931	Статус infeasible, результат перевищує попит
Велика	SciPy	-	No solution, статус infeasible
Велика	Власна	5324	

Сценарій – збалансований (нейтральний)

$$\alpha = 1.0\,, \beta = 1.0\,, \gamma = 1.0$$

Набір даних	Версія	Z	Коментар
Мала	PuLP	8180	
Мала	SciPy	8180	
Мала	Власна	12810	
Велика	PuLP	15525	Статус infeasible, результат перевищує попит

Велика	SciPy	-	No solution, статус infeasible
Велика	Власна	19645	

Інтерпретація результатів та висновки:

- Зміна вагових коефіцієнтів суттєво впливає на структуру оптимального плану і значення цільової функції.
- Збалансовані сценарії дають найвищі значення Z, що підтверджує: одночасна оптимізація суперечливих критеріїв вимагає компромісів.
- Найефективніші (у плані витрат) рішення – у економічному сценарії, але вони можуть не задовольняти безпекові чи часові обмеження гуманітарних місій.
- Вибір значень α, β, γ має не лише математичне, але й етичне значення в контексті гуманітарних місій.
- Надмірна концентрація на одному критерії може спричинити небажані наслідки: дешеві, але повільні або небезпечні маршрути.

3.7. Висновок до розділу 3

У цьому розділі було здійснено комплексну реалізацію задачі багатокритеріальної оптимізації для логістичного постачання у гуманітарному контексті. Основна мета полягала в дослідженні можливостей різних підходів до розв’язання задачі лінійного програмування, а також у верифікації результатів, чутливості моделі до вагових коефіцієнтів та їх практичної інтерпретації.

Перш за все, було реалізовано три варіанти розв’язання задачі: за допомогою бібліотеки PuLP, функції SciPy.optimize.linprog та власноручної реалізації симплекс-методу. Усі методи дозволили сформувати коректну постановку задачі у канонічній формі, з урахуванням реальних обмежень – запасів на складах, потреб пунктів призначення та невід’ємності змінних.

Результати експериментів продемонстрували важливі спостереження. Для малої розмірності задачі (3 склади × 4 пункти) всі три методи показали задовільну точність і узгодженість результатів. Незначна різниця у значеннях цільової функції між бібліотечними методами та ручною реалізацією пояснюється альтернативними виборами базису при однаковій оптимальній вартості.

Однак для великої розмірності задачі (6 складів × 8 пунктів) було виявлено суттєві відмінності. Бібліотека linprog повертала статус infeasible – тобто, відсутність допустимого розв'язку. У свою чергу, PuLP (розв'язувач CBC) іноді надавав план доставки, однак із порушенням обмежень на попит, що ставить під сумнів його коректність. Водночас, власноручна реалізація симплекс-методу успішно знаходила розв'язки, які повністю задовольняли всі задані обмеження, що підтверджує її обчислювальну стійкість та придатність до вирішення задач гуманітарної логістики в умовах високої складності.

Окрему увагу було приділено реалізації багатокритеріального підходу до оптимізації. Шляхом згортки трьох критеріїв (вартість, ризик, час доставки) у зважену цільову функцію, вдалося гнучко керувати пріоритетами планування залежно від типу місії. Чотири сценарії – економічний, швидка доставка, максимальна безпека та збалансований – показали, наскільки істотно змінюється не лише значення цільової функції, а й структура оптимального плану постачання. Важливо, що збалансовані розв'язки виявилися найменш економними, але такими, що найбільш повно відображають складні реалії гуманітарних рішень, де компроміс між вартістю, безпекою та оперативністю є необхідним.

Проведений мікроаналіз чутливості підтвердив, що вибір вагових коефіцієнтів α, β, γ має не лише математичне, а й концептуальне значення. Орієнтація виключно на вартість може призвести до ризикованих або неприйнятно тривалих рішень, що є критичним у гуманітарних ситуаціях. Натомість зміщення акценту на безпеку чи час доставки формує більш збалансовані маршрути, хоча й з вищими витратами.

Таким чином, третій розділ продемонстрував як практичну реалізованість, так і аналітичну цінність побудованої моделі. Застосування кількох підходів до розв'язання задачі дозволило не лише перевірити коректність реалізації, а й виявити потенційні обмеження бібліотечних засобів, що критично важливо при побудові систем підтримки рішень у сфері гуманітарної логістики. Власноручна реалізація симплекс-методу довела свою надійність, гнучкість і здатність адаптуватися до складних умов, що робить її рекомендованим інструментом для подібних задач.

ВИСНОВКИ

У межах дипломної роботи було комплексно досліджено можливості застосування лінійного програмування для розв’язання задачі багатокритеріальної оптимізації в умовах гуманітарної логістики, зокрема в ситуаціях, обумовлених воєнним конфліктом на території України. Результати дослідження охоплюють повний цикл аналітичної та практичної реалізації задачі – від теоретичного обґрунтування до математичного моделювання, реалізації в середовищі програмування Python та аналізу отриманих рішень у різних сценаріях.

У першому розділі роботи було представлено ґрунтовний теоретичний огляд основ лінійного програмування. Детально розглянуто класичну постановку задачі лінійного програмування: побудову цільової функції, систему обмежень, умови допустимості та невід’ємності. Окрему увагу приділено прикладним класам задач – транспортній задачі, задачі призначення, цілочисловому та багатоцільовому лінійному програмуванню. Також було проаналізовано геометричну інтерпретацію задач ЛП у просторі двох та трьох змінних, що дозволило візуально зрозуміти, як формується область допустимих рішень і чому оптимум досягається на її вершині.

Особливо важливим у теоретичній частині стало вивчення симплекс-методу як основного алгоритму розв’язання задач ЛП. Було пояснено не лише базову логіку методу, а й особливі випадки його реалізації – виродження, нескінченність та відсутність розв’язку. Завдяки цьому створено міцну базу для реалізації симплекс-методу у власному програмному забезпеченні. Крім того, було представлено сучасні інструменти розв’язання задач ЛП у Python: бібліотеки PuLP і scipy.optimize.linprog, що забезпечують широкий спектр технічних можливостей та підтримку великомасштабних моделей.

У другому розділі було здійснено прикладну формалізацію задачі гуманітарної логістики. Вперше задача логістичного постачання в умовах воєнного часу була представлена як багатокритеріальна задача лінійного програмування, в якій одночасно враховуються три критично важливі аспекти: вартість доставки, ризик логістичного маршруту (враховуючи безпекові аспекти, зокрема небезпеку активних бойових дій), та час доставки гуманітарного вантажу.

Ключовим елементом моделювання стало застосування методу зваженої згортки, який дозволив перетворити багатокритеріальну задачу в однокритеріальну, з можливістю варіювання вагових коефіцієнтів α, β, γ залежно від поставлених пріоритетів (наприклад, швидкість чи безпека). Це дало змогу сформулювати гнучку модель, адаптовану до реальних сценаріїв польової роботи в гуманітарних місіях.

Задача була протестована на двох масштабах:

- малий тестовий набір (3 склади × 4 пункти), який дозволив проаналізувати логіку та правильність побудови моделі;
- великий набір даних (6 складів × 8 пунктів), що відображає складну мережу логістичних маршрутів у кризовому регіоні.

Було також сформовано сценарії планування, які демонструють різні стратегії реагування на гуманітарні виклики: мінімізація витрат, максимізація швидкості постачання, фокус на безпеці та збалансований підхід. Такий сценарний аналіз дозволив виявити гнучкість моделі і її придатність до використання в системах оперативного планування.

У третьому розділі відбулася реалізація моделі тривимірної багатокритеріальної задачі за допомогою трьох підходів:

- Бібліотека PuLP, яка забезпечує декларативний стиль побудови моделі;
- Функція scipy.optimize.linprog, орієнтована на чисельну оптимізацію;
- Власноручна реалізація симплекс-методу, що стала головною перевіркою достовірності та ефективності запропонованого алгоритму.

Проведені чисельні експерименти підтвердили, що при малій розмірності всі три методи дають подібні результати, хоча можуть мати різні шляхи досягнення оптимуму через відмінності у виборі базису. Проте при великій розмірності задачі саме власноруч реалізований симплекс-метод показав найвищу стійкість – він успішно знаходив рішення, в той час як linprog часто повертав статус "infeasible", а CBC (через PuLP) іноді порушував обмеження на попит.

Особливої уваги заслуговує проведений мікроаналіз чутливості до вагових коефіцієнтів α, β, γ. Аналіз показав, що навіть незначна зміна ваг може суттєво вплинути як на структуру оптимального плану, так і на рівень ризику або час доставки. Це підтверджує необхідність гнучкого інструментарію для прийняття рішень у гуманітарному секторі, де кожна місія має свій унікальний контекст і набір обмежень.

Загалом, результати дослідження демонструють, що методи лінійного програмування, зокрема симплекс-метод у поєднанні з багатокритеріальним підходом, є потужним інструментом для моделювання та оптимізації гуманітарних логістичних операцій. Розроблена модель дозволяє не лише ефективно розподіляти ресурси з урахуванням множини реальних критеріїв, а й оперативно адаптуватися до змін середовища.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Linear Programming Explained: Formulas and Examples - Spiceworks. Spiceworks Inc. URL: https://www.spiceworks.com/tech/it-strategy/articles/linear-programming/ (date of access: 02.06.2025).
2. The Editors of Encyclopaedia Britannica. Linear programming | Definition & Facts | Britannica. Encyclopedia Britannica. URL: https://www.britannica.com/science/linear-programming-mathematics (date of access: 02.06.2025).
3. Mohammed, Elfarazdag & Hussein, Mohammed & Mohammed, Amin & Haleeb, Ali. (2023). Application of Linear Programming (Transportation Problem). 12. 7. 10.21275/SR21222020051. URL:https://www.researchgate.net/publication/369260376_Application_of_Linear_Programming_Transportation_Problem (date of access: 02.06.2025).
4. Mohamed, Elsiddig & Mohammed, Elfarazdag. (2015). Application of Linear Programming (Assignment Model). International Journal of Science and Research (IJSR). 438. 1446-1449. URL: https://www.researchgate.net/profile/Elsiddig-Mohamed/publication/277622280_Application_of_Linear_Programming_Assignment_Model/links/556ed0f608aefcb861dbac39/Application-of-Linear-Programming-Assignment-Model.pdf (date of access: 02.06.2025).
5. Contributors to Wikimedia projects. Integer programming - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Integer_programming (date of access: 02.06.2025).
6. IBM ILOG CPLEX Optimization Studio. IBM - United States. URL: https://www.ibm.com/docs/en/icos/22.1.0?topic=problem-what-is-integer-programming (date of access: 02.06.2025).
7. Applied mathematical programming / ed. by Hax, Arnoldo C., joint author., Magnanti, Thomas L., joint author. Reading, Mass : Addison-Wesley Pub. Co., 1977. 716 p. URL: https://web.mit.edu/15.053/www/AMP-Chapter-04.pdf (date of access: 02.06.2025).
8. Luca Trevisan. Lecture 6. URL: https://theory.stanford.edu/~trevisan/cs261/lecture06.pdf (date of access: 02.06.2025).
9. Dantzig G. Origins of the Simplex Method. 1990. URL: https://doi.org/10.1145/87252.88081 (date of access: 02.06.2025).

10. John C. Nash. THE (DANTZIG) SIMPLEX METHOD FOR LINEAR PROGRAMMING. 2000. URL: https://www.cs.fsu.edu/~lacher/courses/COT4401/notes/cise_v2_i1/simplex.pdf (date of access: 02.06.2025).
11. pulp.apis Interface to Solvers – PuLP 3.0.2 documentation. COIN-OR Documentation Site | COIN-OR Documentation. URL: https://coin-or.github.io/pulp/technical/solvers.html (date of access: 02.06.2025).
12. What is optimize.linprog() in SciPy?. HowDev. URL: https://how.dev/answers/what-is-optimizelinprog-in-scipy (date of access: 02.06.2025).

ДОДАТОК А. РЕАЛІЗАЦІЯ

```
import numpy as np
import pulp
from scipy.optimize import linprog
import matplotlib.pyplot as plt

# === Тестові дані ===
def get_test_cases():
    return {
        "малий": {
            "supplies": [100, 150, 120],
            "demands": [80, 100, 90, 100],
            "costs": np.array([
                [4, 6, 9, 7],
                [5, 4, 7, 8],
                [6, 7, 4, 5]
            ]),
            "risks": np.array([
                [3, 5, 8, 6],
                [4, 3, 5, 7],
                [5, 6, 3, 4]
            ]),
            "times": np.array([
                [12, 18, 25, 20],
                [15, 12, 20, 22],
                [18, 20, 10, 15]
            ])
        },
        "великий": {
            "supplies": [120, 140, 100, 90, 130, 110],
            "demands": [80, 75, 95, 60, 85, 70, 100, 100],
            "costs": np.array([
                [6, 8, 5, 9, 7, 6, 4, 8],
                [5, 7, 6, 8, 5, 5, 6, 9],
                [7, 6, 8, 7, 6, 7, 5, 6],
                [6, 5, 6, 9, 7, 6, 8, 7],
                [5, 8, 7, 6, 7, 5, 6, 5],
                [7, 6, 6, 7, 8, 6, 7, 6]
            ]),
```

```
            "risks": np.array([
                [4, 5, 4, 6, 3, 5, 3, 4],
                [5, 4, 3, 4, 4, 3, 5, 4],
                [4, 4, 5, 3, 5, 4, 3, 3],
                [3, 5, 4, 4, 3, 5, 4, 5],
                [4, 3, 5, 4, 4, 4, 3, 4],
                [5, 4, 3, 5, 5, 3, 4, 3]
            ]),
            "times": np.array([
                [15, 20, 12, 18, 14, 17, 13, 16],
                [13, 17, 14, 15, 16, 12, 14, 18],
                [18, 14, 17, 16, 15, 18, 13, 17],
                [14, 16, 15, 17, 14, 16, 15, 15],
                [15, 17, 16, 14, 18, 14, 16, 14],
                [16, 14, 13, 16, 17, 15, 14, 15]
            ])
        }
    }

# === Побудова матриць A_eq, b_eq ===
def build_constraints(supplies, demands):
    num_suppliers = len(supplies)
    num_destinations = len(demands)
    A_eq = []
    b_eq = []
    for i in range(num_suppliers):
        row = [0] * (num_suppliers * num_destinations)
        for j in range(num_destinations):
            row[i * num_destinations + j] = 1
        A_eq.append(row)
        b_eq.append(supplies[i])
    for j in range(num_destinations):
        row = [0] * (num_suppliers * num_destinations)
        for i in range(num_suppliers):
            row[i * num_destinations + j] = 1
        A_eq.append(row)
        b_eq.append(demands[j])
    return np.array(A_eq, dtype=float), np.array(b_eq, dtype=float), num_suppliers, num_destinations
```

```
# === Власна реалізація симплекс-методу з поясненням та графіком ===
def simplex(A, b, c, plot=False, label=None, show_true_Z=False):
    m, n = A.shape
    tableau = np.zeros((m + 1, n + m + 1))
    tableau[:m, :n] = A
    tableau[:m, n:n + m] = np.eye(m)
    tableau[:m, -1] = b
    tableau[-1, :n] = -c
    basis = list(range(n, n + m))
    step = 0
    z_history = [-tableau[-1, -1] if show_true_Z else tableau[-1, -1]]

    def print_simplex_tableau(tableau, basis, step):
        m, n = tableau.shape
        print(f"\n--- Симплекс-таблиця. Крок {step} ---")
        header = [f"x{i+1}" for i in range(n - m - 1)] + [f"s{i+1}" for i in range(m)] + ["RHS"]
        print("Базис\t" + "\t".join(header))
        for i in range(m - 1):
            row_label = f"x{basis[i]+1}" if basis[i] < n - m - 1 else f"s{basis[i] - (n - m - 1) + 1}"
            row_data = "\t".join(f"{val:.2f}" for val in tableau[i])
            print(f"{row_label}\t{row_data}")
        print("Z\t" + "\t".join(f"{val:.2f}" for val in tableau[-1]))

    while True:
        print_simplex_tableau(tableau, basis, step)

        # Вибір вхідної змінної
        pivot_col = np.argmin(tableau[-1, :-1])
        min_z_val = tableau[-1, pivot_col]
        if min_z_val >= 0:
            print("Усі оцінки приросту ≤ 0. Поточне рішення оптимальне.")
            break
```

```
        # Обчислення θ
        ratios = []
        print("\nРозрахунок θ для кожного рядка:")
        for i in range(m):
            a_ik = tableau[i, pivot_col]
            b_i = tableau[i, -1]
            if a_ik > 1e-8:
                theta = b_i / a_ik
                ratios.append(theta)
                var_label = f"x{basis[i]+1}" if basis[i] < n else f"s{basis[i] - n + 1}"
                print(f"  θ_{i+1} = {b_i:.2f} / {a_ik:.2f} = {theta:.2f} для {var_label}")
            else:
                ratios.append(np.inf)
                print(f"  θ_{i+1} = ∞ (a_ik = {a_ik:.2f} ≤ 0)")

        pivot_row = np.argmin(ratios)
        min_ratio = ratios[pivot_row]

        if min_ratio == np.inf:
            raise Exception("Розв'язок не обмежений — усі коефіцієнти в обраному стовпці ≤ 0")

        pivot_elem = tableau[pivot_row, pivot_col]

        # === ПОЯСНЕННЯ ПЕРЕХОДУ ===
        print("\n=== Пояснення переходу до наступного кроку ===")
        print(f"Вибрано вхідну змінну: x_{pivot_col+1} (мінімальна оцінка приросту: {min_z_val:.2f})")
        leaving_var = f"x{basis[pivot_row]+1}" if basis[pivot_row] < n else f"s{basis[pivot_row] - n + 1}"
        print(f"Мінімальне θ = {min_ratio:.2f}, отже, вихідна змінна: {leaving_var}")
        print(f"Півот-елемент: {pivot_elem:.2f}")
        print(f"Оновлюємо базис: замінюємо {leaving_var} на x_{pivot_col+1}\n")

        # Нормалізація півот-рядка
        tableau[pivot_row, :] /= pivot_elem

        # Занулення інших елементів у півот-стовпці
        for i in range(m + 1):
            if i != pivot_row:
                tableau[i, :] -= tableau[i, pivot_col] * tableau[pivot_row, :]

        basis[pivot_row] = pivot_col
        step += 1
        z_history.append(-tableau[-1, -1] if show_true_Z else tableau[-1, -1])
```

```
    # Побудова графіка, якщо потрібно
    if plot:
        import matplotlib.pyplot as plt
        plt.plot(range(len(z_history)), z_history, marker='o')
        plt.title(f"Зміна цільової функції Z ({label})")
        plt.xlabel("Крок симплекс-методу")
        plt.ylabel("Z")
        plt.grid(True)
        plt.show()

    # Побудова розв'язку
    x = np.zeros(n)
    for i in range(m):
        if basis[i] < n:
            x[basis[i]] = tableau[i, -1]

    z = abs(tableau[-1, -1])
    return x, z


# === PuLP ===
def solve_with_pulp(c, supplies, demands, num_suppliers, num_destinations):
    prob = pulp.LpProblem("Sensitivity_Analysis_PuLP", pulp.LpMinimize)
    x_vars = [pulp.LpVariable(f"x_{i}_{j}", lowBound=0)
              for i in range(num_suppliers) for j in range(num_destinations)]
    prob += pulp.lpDot(c, x_vars)
    for i in range(num_suppliers):
        prob += pulp.lpSum(x_vars[i * num_destinations + j] for j in range(num_destinations)) == supplies[i]
    for j in range(num_destinations):
        prob += pulp.lpSum(x_vars[i * num_destinations + j] for i in range(num_suppliers)) == demands[j]
    prob.solve()
    return prob, x_vars

# === SciPy ===
def solve_with_scipy(c, A_eq, b_eq, bounds):
    return linprog(c=c, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method='highs')

# === Аналіз чутливості ===
def run_sensitivity_analysis(modes=("pulp", "scipy", "simplex"), test_case_names=None, alpha_vals=None, beta_vals=None, gamma_vals=None):
    alpha_vals = alpha_vals if alpha_vals is not None else [0.5, 1.0, 2.0]
    beta_vals = beta_vals if beta_vals is not None else [0.2, 0.5, 1.0]
    gamma_vals = gamma_vals if gamma_vals is not None else [0.1, 0.2, 0.5]
    test_cases = get_test_cases()
```

```
  for test_name, data in test_cases.items():
      if test_case_names and test_name not in test_case_names:
          continue
      print(f"\n===== ТЕСТОВИЙ НАБІР: {test_name.upper()} =====")
      supplies, demands = data["supplies"], data["demands"]
      costs, risks, times = data["costs"], data["risks"], data["times"]
      A_eq, b_eq, num_suppliers, num_destinations = build_constraints(supplies, demands)
      bounds = [(0, None)] * (num_suppliers * num_destinations)

      for alpha in alpha_vals:
          for beta in beta_vals:
              for gamma in gamma_vals:
                  print(f"\n--- alpha={alpha}, beta={beta}, gamma={gamma} ---")
                  c = (alpha * costs + beta * risks + gamma * times).flatten()

                  if "pulp" in modes:
                      prob, x_vars = solve_with_pulp(c, supplies, demands, num_suppliers, num_destinations)
                      status_str = pulp.LpStatus[prob.status]
                      print(f"[PuLP] Статус: {status_str}")
                      print(f"[PuLP] Z = {pulp.value(prob.objective):.2f}")
                      for var in x_vars:
                          if var.varValue > 1e-6:
                              print(f"    {var.name} = {var.varValue:.2f}")

                  if "scipy" in modes:
                      res = solve_with_scipy(c, A_eq, b_eq, bounds)
                      print(f"[SciPy] Статус: {res.message}")
                      if res.success:
                          print(f"[SciPy] Z = {res.fun:.2f}")
                          for i, val in enumerate(res.x):
                              if val > 1e-6:
                                  print(f"    x_{i // num_destinations}_{i % num_destinations} = {val:.2f}")
                      else:
                          print("[SciPy] No solution")
```

```
                  if "simplex" in modes:
                      try:
                          label = f"{test_name}, α={alpha}, β={beta}, γ={gamma}"
                          x_opt, z_opt = simplex(np.array(A_eq), np.array(b_eq), np.array(c), plot=True, label=label, show_true_Z=True)
                          print(f"[Власна реалізація] Z = {z_opt:.2f}")
                          for i, val in enumerate(x_opt):
                              if val > 1e-6:
                                  print(f"    x_{i // num_destinations}_{i % num_destinations} = {val:.2f}")
                      except Exception as e:
                          print(f"[Власна реалізація] {str(e)}")
```

Приклад виклику функції:

```
run_sensitivity_analysis(
        modes=("pulp", "scipy", "simplex"),
        test_case_names=["малий"],
        alpha_vals=[1.0],
        beta_vals=[0.2],
        gamma_vals=[0.1]
    )
```