Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

**Бакалаврська кваліфікаційна робота на тему:**

# ПРОГРАМНИЙ МОДУЛЬ ПОСТОПТИМАЛЬНОГО АНАЛІЗУ ЗАДАЧ ЛІНІЙНОГО ПРОГРАМУВАННЯ

Виконав: студент 4 курсу, групи 3КН-21б
спеціальності 122 Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Коваль І. О.
(прізвище та ініціали)
Керівник: асистент каф. КН
Ваховська Л. М.
(прізвище та ініціали)
«04» червня 2025 р.
Рецензент: к.т.н. доцент каф. АІІТ
Барабан М. В.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Ярвай А. А.
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

**ЗАТВЕРДЖУЮ**
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» Травня 2025 р.

**ЗАВДАННЯ**
**НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**
Ковалю Іллі Олександровичу

1. Тема роботи: Програмний модуль постоптимального аналізу задач лінійного програмування.
керівник роботи: Ваховська Любов Михайлівна, асистент.
затверджені наказом вищого навчального закладу "20" 03 2025 року №97
2. Термін подання студентом роботи 30 Травня 2025р
3. Вихідні дані до роботи : мова програмування – об'єктно-орієнтована, мінімалістичний GUI, ОС – Windows
4. Зміст текстової частини (перелік питань, які потрібно розробити):
Обґрунтування доцільності розробки програмного модуля постоптимального аналізу задач лінійного програмування;
Проектування програмного модуля постоптимального аналізу ЗЛП
Програмна реалізація модуля постоптимального аналізу ЗЛП
Тестування розробленої програми
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
Структура програмного модуля постоптимального аналізу задач лінійного програмування
Схема алгоритму роботи модуля для постоптимального аналізу задач лінійного програмування
UML-діаграма класів програмного модуля постоптимального аналізу задач лінійного програмування
Приклад роботи програми

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1-4 | Ваховська Л. М., асистент | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 05 Травня 2025р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва та зміст етапу | Термін виконання | | Примітка |
|---|---|---|---|---|
| | | початок | закінчення | |
| 1 | Аналіз предметної області задач лінійного програмування та постоптимального аналізу | 05.05.25 | 11.05.25 | |
| 2 | Розробка програмного модуля постоптимального аналізу задач лінійного програмування | 12.05.25 | 17.05.25 | |
| 3 | Програмна реалізація модуля постоптимального аналізу задач лінійного програмування. | 18.05.25 | 29.05.25 | |
| 4 | Розробка інструкції користувача | 30.05.25 | 01.06.25 | |
| 5 | Оформлення матеріалів до захисту БКР та розробка презентації | 02.06.25 | 04.06.25 | |
| 6 | | | | |
| 7 | | | | |
| | | | | |
| | | | | |

**Студент** _____ **Коваль І. О.**
(підпис) (прізвище та ініціали)

**Керівник проекту (роботи)** _____ **Ваховська Л. М**
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 80 сторінки формату А4, які включають 22 рисунків і 4 таблиці. У списку використаних джерел зазначено 22 найменування.

У загальній частині роботи на основі аналізу існуючих технічних рішень, методів та технологій доведена доцільність розробки програмного модуля постоптимального аналізу задач лінійного програмування. Розроблено алгоритм роботи, структура програмного модуля та UML діаграми класів, які спрощують розробку та розуміння структури програмного забезпечення. Програмний продукт розроблений в інтегрованому середовищі Pycharm.

Розроблено програмний продукт та проведено його тестування. Результати тестування розробки вказують на те, що основні функції програмного модуля постоптимального аналізу задач лінійного програмування реалізовано.

Ключові слова: постоптимальний аналіз, цільова функція, задача лінійного програмування, програмний модуль, обмеження функції.

# ABSTRACT

The bachelor's thesis consists of 80 pages of A4 format, which include 22 figures and 4 tables. The list of used sources includes 22 names.

In the general part of the work, based on the analysis of existing technical solutions, methods and technologies, the feasibility of developing a software module for post-optimal analysis of linear programming problems is proven. The algorithm of work, the structure of the software module and UML class diagrams have been developed, which simplify the development and understanding of the software structure. The software product is developed in the Pycharm integrated environment.

The software product has been developed and tested. The results of the development testing indicate that the main functions of the software module for post-optimal analysis of linear programming problems have been implemented.



Keywords: post-optimal analysis, objective function, linear programming problem, program module, function constraint.

**ЗМІСТ**

ВСТУП ..... 4
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ ..... 6
1.1 Сучасні методи постоптимального аналізу в лінійному програмуванні . 6
1.2 Аналіз програмних засобів постоптимального аналізу ЗЛП ..... 9
1.3 Постановка задачі постоптимального аналізу ЗЛП ..... 13
1.4 Особливості застосування та перспективи розвитку програмного модуля постоптимального аналізу ЗЛП ..... 17
1.5 Висновки до розділу 1 ..... 19
2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ ПОСТОПТИМАЛЬНОГО АНАЛІЗУ ЗЛП ..... 20
2.1 Розробка структури програмного модуля ..... 20
2.2 Розробка алгоритму роботи системи ..... 23
2.3 Розробка UML-діаграми класів ..... 26
2.4 Висновки до розділу 2 ..... 28
3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ ПОСТОПТИМАЛЬНОГО АНАЛІЗУ ЗЛП ..... 30
3.1 Обґрунтування вибору мови програмування, бібліотек та підходу до реалізації ..... 30
3.3 Розробка програмного модуля постоптимального аналізу ..... 33
3.4 Тестування роботи програмного модуля ..... 37
3.4 Аналіз результатів тестування ..... 45
3.5 Висновки до розділу 3 ..... 47
ВИСНОВКИ ..... 49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 50
ДОДАТКИ ..... 52
ДОДАТОК А (ОБОВ'ЯЗКОВИЙ) ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ ..... 53
ДОДАТОК Б ЛІСТИНГ ПРОГРАМНОГО МОДУЛЯ ПОСТОПТИМАЛЬНОГО АНАЛІЗУ ЗЛП ..... 54
ДОДАТОК В ГРАФІЧНА ЧАСТИНА ..... 68
ДОДАТОК Г ІНСТРУКЦІЯ КОРИСТУВАЧА ..... 78

# ВСТУП

**Актуальність дослідження.**

В умовах масштабних технологічних та маркетингових трансформацій зростає потреба у впровадженні ефективних методів оптимізації в різних сферах діяльності — управлінні, виробництві, логістиці, фінансовому аналізі тощо. Сучасні автоматизовані системи управління потребують рішень, що враховують численні обмеження та критерії ефективності. Найбільш досконалим способом формалізації таких задач є лінійне програмування, що дозволяє знаходити оптимальні стратегії розподілу ресурсів за заданими критеріями. Лінійне програмування виступає фундаментальним інструментом математичного моделювання, що широко застосовується як в академічному середовищі, так і у практиці бізнесу та інженерії.

У реальних умовах після отримання оптимального розв’язку часто виникає потреба у повторному перегляді або коригуванні окремих вхідних параметрів. Це зумовлено як змінами ринкових умов, так і необхідністю врахування нових обмежень або альтернатив. Постоптимальний аналіз дозволяє швидко оцінити, як зміна певних коефіцієнтів цільової функції чи обмежень впливає на оптимальне рішення, не вдаючись до повного перерахунку задачі. Такий підхід значно знижує обчислювальні витрати, підвищує адаптивність моделей і дає змогу оперативно приймати управлінські рішення.

Особливої актуальності постоптимальний аналіз набуває в умовах динамічних змін вхідних параметрів, що характерні для виробничого планування, ресурсного управління, транспортної логістики, бюджетного прогнозування, аграрного сектору та багатьох інших галузей. У цих умовах важливо не лише знаходити оптимальне рішення, а й забезпечувати стійкість та гнучкість моделей при зміні вхідних даних. Постоптимальний аналіз виступає потужним засобом для виявлення таких меж змін, при яких оптимальне рішення зберігається, або для визначення порогових значень, за яких воно стає недійсним.

Попри важливість цього підходу, більшість прикладних програм не містять повноцінного функціоналу для постоптимального аналізу, або ж він реалізований частково і не дозволяє гнучко взаємодіяти з моделлю. Крім того, багато існуючих інструментів є складними у використанні для непідготовленого користувача або не підтримують інтерактивне візуальне представлення результатів аналізу. Це створює потребу у розробці зручного, інтуїтивно зрозумілого програмного модуля, який би поєднував функціональність, гнучкість і доступність для широкого кола користувачів — від студентів і викладачів до аналітиків і керівників проєктів.

Таким чином, створення програмного модуля постоптимального аналізу задач лінійного програмування є актуальним як з практичної, так і з наукової точки зору, оскільки сприяє підвищенню ефективності процесів прийняття рішень в умовах невизначеності та змін зовнішніх і внутрішніх параметрів системи.

**Мета дослідження** – розширення функціональних можливостей програмного модуля, який здійснює постоптимальний аналіз задач лінійного програмування (ЗЛП) після знаходження базового оптимального розв’язку.

**Об’єкт дослідження** – процес постоптимального аналізу задач лінійного програмування

**Предмет дослідження** – програмний модуль постоптимального аналізу ЗЛП, зокрема алгоритми модифікації та аналізу оптимального плану при зміні параметрів задачі, засоби інтеграції цих алгоритмів у програмний модуль, а також способи відображення результатів постоптимального аналізу з використанням інтерфейсу користувача.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- Проаналізувати теоретичні основи постоптимального аналізу в лінійному програмуванні.
- Спроектувати програмний модуль постоптимального аналізу ЗЛП.
- Програмно реалізувати модуль постоптимального аналізу ЗЛП.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ

## 1.1 Сучасні методи постоптимального аналізу в лінійному програмуванні

Постоптимальний аналіз можна умовно поділити на кілька категорій [1]: Аналіз чутливості коефіцієнтів цільової функції:

Якщо до прикладу у нас є задача типу:

$$max: Z = c_1x_1 + c_2x_2 + \ldots + c\square x\square \text{ , при} \quad (1.1)$$
$$Ax \leq b, x \geq 0,$$

то з'являється запитання: який діапазон значень є для кожного $c_i$, при якому теперішнє базисне рішення остається оптимальним? Для цього використовуються двоїсті оцінки та теорема оптимальності симплекс-методу. Встановлюються допустимі інтервали коефіцієнтів, в межах яких структура оптимального базису не змінюється.

Цей аналіз дуже корисний, коли коефіцієнти цільової функції показують прибуток або вартість, що може змінюватись з часом.

- Постоптимальний аналіз правих частин обмежень.

Дослідження того, як зміниться оптимальне рішення при зміні правих частин обмежень (вектора b). Цей тип аналізу дає відповідь на те, наскільки можна змінити ресурси, щоб при цьому не втратити оптимальність рішення?

З математичної точки зору, визначається допустимий діапазон значень для кожного $b_i$, у межах якого базисне рішення залишається допустимим. Відповідні обчислення базуються на аналізі зворотної матриці базису ($B^{-1}$) та значень змінних.

- Додавання нових змінних та обмежень.

У практиці часто виникають ситуації, коли необхідно розширити модель новими змінними (наприклад, нові товари або маршрути) або обмеженнями (нові обмеження ресурсів, додаткові правила). Постоптимальний аналіз дозволяє перевірити, чи задача потребує повного перерахунку, чи можна адаптувати поточне рішення з використанням додаткових процедур.

Алгоритмічні методи передбачають перевірку впливу нових обмежень на допустимість та оптимальність базису, що вже знайдено.

- Інтервальний аналіз.

Інтервальний підхід до постоптимального аналізу передбачає, що замість точних значень коефіцієнтів задаються інтервали невизначеності. Наприклад, замість $c_i = 10$, припускається $c_i \in [8, 12]$.

Цей підхід дозволяє моделювати нечіткість та варіативність у вхідних даних. Однією з проблем тут є складність обчислень, оскільки необхідно дослідити всі можливі комбінації меж інтервалів. Однак у випадках, коли важлива надійність рішення, інтервальний аналіз дає змогу побудувати гарантовані сценарії.

- Параметричне програмування.

Параметричне програмування передбачає дослідження поведінки оптимального рішення при зміні одного або кількох параметрів задачі на протязі певного діапазону. Найбільш типовий варіант — зміна одного коефіцієнта цільової функції або елемента вектора b.

У цьому випадку результат подається як функція параметра — наприклад, значення цільової функції як функція λ. Отримується розбиття простору параметра на діапазони, в яких оптимальний базис залишається сталим.

Параметричне програмування має важливе значення у задачах планування і фінансового моделювання, коли одна змінна (наприклад, вартість сировини) варіюється в часі.

- Двоїстий аналіз.

Двоїстий підхід до постоптимального аналізу ґрунтується на побудові та дослідженні двоїстої ЗЛП. При цьому зміни в параметрах прямої задачі відображаються у змінних або обмеженнях двоїстої задачі.

Перевагою є можливість використання теорем дуальності для оцінки стійкості рішення. Наприклад, зміна правої частини прямої задачі відображається у коефіцієнтах цільової функції двоїстої задачі.

- Матричний підхід до аналізу чутливості.

Цей підхід полягає у використанні матричних представлень усіх компонентів задачі: матриці коефіцієнтів A, векторів c та b, а також базисних змінних. Застосування інструментів лінійної алгебри дозволяє формалізувати оцінку змін параметрів.

Зокрема, використовується зворотна матриця $B^{-1}$ для визначення нових значень базисних змінних та цільової функції при зміні b. Також оцінюється вплив змін c через множення на $B^{-1}A$.

Цей метод особливо зручний для реалізації у програмних модулях, оскільки дозволяє автоматизувати обчислення.

- Алгоритмічні методи.

У разі, коли зміни в параметрах задачі є суттєвими, може знадобитись повний перерахунок розв’язання з використанням змінених вхідних даних. Однак замість повторного запуску симплекс-методу, застосовуються спеціальні алгоритми, що дозволяють адаптувати попередній базис.

Одним з таких підходів є використання симплекс-методу з теплим запуском, коли перша ітерація базується на попередньо знайденому оптимальному рішенні. Також можливе застосування методів редукції або прискореного повторного розв’язання.

## 1.2 Аналіз програмних засобів постоптимального аналізу ЗЛП

Для об’єктивного аналізу програмних засобів доцільно виділити низку критеріїв, за якими здійснюватиметься порівняння:

- Підтримка постоптимального аналізу: наявність вбудованих функцій для аналізу зміни коефіцієнтів цільової функції та обмежень.
- Математичне ядро: реалізація симплекс-методу, подвійного симплекс-методу або інших алгоритмів.
- Інтерфейс користувача: наявність графічного інтерфейсу, зручність введення та виведення даних.
- Можливості інтеграції: підтримка API або використання в інших програмних продуктах.
- Вартість і ліцензування: наявність безкоштовної або комерційної ліцензії.
- Платформа та мова реалізації: операційні системи, на яких працює програмне забезпечення.
- Підтримка та документація: наявність технічної підтримки, оновлень і довідкових матеріалів.

Тепер розглянемо самі програмні засоби [2 – 4].

1. ILOG CPLEX Optimization Studio.

CPLEX є одним із найпотужніших комерційних продуктів для розв’язання задач лінійного та цілочисельного програмування. Програма підтримує широкий спектр функцій, включаючи постоптимальний аналіз. Інструментарій дозволяє досліджувати діапазони зміни коефіцієнтів цільової функції та обмежень, а також аналізувати вплив на оптимальний план.

Переваги:

- Висока продуктивність.
- Гнучкий інтерфейс для програмістів (Java, C++, Python).
- Інтеграція з IBM Watson Studio.

Недоліки:

- Комерційна ліцензія.
- Складність освоєння для новачків.

2. LINDO / LINGO.

LINGO є популярним інструментом для моделювання та розв’язання задач оптимізації. Програма має зручний синтаксис для формулювання моделей і автоматично виконує аналіз чутливості.

Переваги:

- Простота у використанні.
- Автоматична генерація діапазонів змін.
- Добре структурована документація.

Недоліки:

- Обмежена безкоштовна версія.
- Мала гнучкість для інтеграції з іншими мовами програмування.

3. GLPK (GNU Linear Programming Kit).

GLPK – це бібліотека з відкритим кодом для розв’язання ЗЛП. Вона підтримує симплексний метод та дозволяє здійснювати постоптимальний аналіз за допомогою спеціалізованих функцій.

Переваги:

- Відкритий код, вільне використання.
- Доступна інтеграція з Python через API.
- Портативність.

Недоліки:

- Відсутність графічного інтерфейсу.
- Менша продуктивність у порівнянні з комерційними аналогами.

4. PuLP (Python Library for LP).

PuLP — це високорівнева бібліотека Python, яка використовується для створення моделей лінійного програмування. Вона дозволяє інтегрувати сторонні розв’язувачі, зокрема CPLEX, GLPK, CBC, та виконувати аналіз рішень.

Переваги:

- Гнучкість та підтримка популярних розв’язувачів.
- Простий синтаксис для формування моделей.
- Можливість інтеграції з іншими Python-бібліотеками (Pandas, NumPy).

Недоліки:

- Вимагає знання програмування.
- Обмежені вбудовані функції постоптимального аналізу (потрібна додаткова реалізація).

5. Microsoft Excel Solver.

Найбільш доступний засіб для розв’язання задач оптимізації. Включає можливості для аналізу чутливості, але має обмеження щодо розміру задачі та гнучкості.

Переваги:

- Простота використання.
- Вбудований у Microsoft Excel.
- Аналіз чутливості реалізується автоматично.

Недоліки:

- Обмеження на кількість змінних та обмежень.
- Відсутність автоматизації для складних моделей.

6. SciPy.optimize (Python).

Бібліотека scipy.optimize.linprog у SciPy дозволяє розв’язувати ЗЛП, але не включає стандартного постоптимального аналізу. Проте вона може бути використана в поєднанні з іншими інструментами.

Переваги:

- Частина наукового стеку Python.
- Висока швидкість розрахунків.
- Активна спільнота користувачів.

Недоліки:

- Відсутність вбудованих функцій аналізу чутливості.
- Потребує додаткової реалізації логіки постоптимального аналізу.

У таблиці 1.1 представлено порівняння існуючих програмних засобів, які використовуються для розв’язування задач лінійного програмування та здійснення постоптимального аналізу.

Таблиця 1.1 – Порівняння програмних засобів

| Назва ПЗ | Постоптимальний аналіз | Інтерфейс | Відкритість | Інтеграція | Ліцензія |
|---|---|---|---|---|---|
| CPLEX | +++ | CLI/GUI | Пропрієтарне | +++ | Комерційна |
| LINGO | ++ | GUI | Пропрієтарне | + | Комерційна |
| GLPK | + | CLI | Відкрите | ++ | Вільна |
| Excel Solver | ++ | GUI | Пропрієтарне | + | Умовно безкоштовна |

Аналіз показує, що на ринку програмного забезпечення існує широкий спектр засобів, які дозволяють здійснювати постоптимальний аналіз задач лінійного програмування (ЗЛП). Комерційні рішення на кшталт IBM ILOG CPLEX Optimization Studio, LINGO та Gurobi забезпечують високу точність, оптимальну швидкодію та широку функціональність, включно з повним набором інструментів для аналізу чутливості. Такі продукти мають потужні інтерфейси для інтеграції в корпоративні інформаційні системи, підтримують декілька мов програмування та пропонують зручну документацію. Проте, їх недоліком є висока вартість ліцензії, що суттєво обмежує доступність цих інструментів для невеликих організацій, освітніх закладів або індивідуальних дослідників і студентів.

Натомість інструменти з відкритим кодом, серед яких GLPK (GNU Linear Programming Kit), SciPy.optimize.linprog, PuLP або OR-Tools, є доступними безкоштовно, підтримуються широкими спільнотами розробників і часто мають відкриту архітектуру, що дозволяє модифікувати алгоритми відповідно до

специфічних потреб користувача. Зокрема, GLPK є надійним інструментом для реалізації симплекс-методу, підтримує як пряме, так і двоїсте вирішення задачі та дозволяє здійснювати розширений постоптимальний аналіз. Бібліотека легко інтегрується з мовами програмування Python, C/C++ або Java, що створює широкі можливості для створення кастомізованих навчальних та дослідницьких рішень.

Для цілей освітнього програмного забезпечення, яке націлене не лише на отримання результату, але й на глибоке розуміння внутрішньої логіки обчислень, найбільш доцільним є саме використання бібліотек з відкритим програмним кодом. Вони забезпечують гнучкість, масштабованість, контроль над етапами розв’язання та достатню точність розрахунків для прикладних задач. Крім того, такі бібліотеки дозволяють реалізовувати функції візуалізації симплекс-таблиць, покрокового симплекс-алгоритму, а також власні алгоритмічні розширення для постоптимального аналізу, зокрема обчислення діапазонів допустимих змін коефіцієнтів цільової функції або обмежень, аналіз тіньових цін (shadow prices) та оцінювання впливу зміни вхідних параметрів на оптимальне рішення.

Завдяки використанню відкритого ПЗ створюється гнучке середовище для проведення експериментів, навчання, а також адаптації програмного модуля під конкретні задачі. Це особливо важливо в контексті вивчення методів лінійного програмування в навчальних закладах, де студенти мають можливість не лише отримати результат, а й зрозуміти механізми, що стоять за обчисленням оптимального плану та його чутливості до змін у вхідних даних. У поєднанні з графічним інтерфейсом та інтуїтивним дизайном така система стає ефективним інструментом для підтримки освітнього процесу та дослідницької діяльності в галузі оптимізації.

### 1.3 Постановка задачі постоптимального аналізу ЗЛП

Лінійне програмування є однією з ключових галузей математичного програмування, яка знайшла широке застосування в різних важливих сферах

людської діяльності – зокрема в логістиці, виробництві, плануванні, енергетичному секторі тощо [5]. Його основне завдання полягає в досягненні певної мети шляхом оптимізації (максимізації або мінімізації) спеціальної цільової функції, що задається у формі системи лінійних рівнянь або нерівностей [6]. Розв'язок, який забезпечує найкраще значення цієї функції за заданих обмежень, вважається оптимальним.

Проте в реальних умовах зустрічається мало ситуацій в яких всі параметри задачі лишаються незмінними. На обмеження можуть впливати численні зовнішні чинники – наприклад, зміна попиту, нестабільність поставок, технологічні новації або управлінські коригування. У зв’язку з цим виникає потреба не лише знайти оптимальне рішення, а й оцінити його стійкість до змін у вхідних даних. Такий аналіз і називається постоптимальним, адже він дозволяє дослідити, як варіації в початкових умовах впливають на вже знайдене оптимальне рішення.

Постоптимальний аналіз дозволяє отримати відповіді на такі важливі запитання:

- Які допустимі межі зміни коефіцієнтів цільової функції, за яких оптимальний розв’язок залишиться незмінним?
- Як змінюється значення цільової функції при зміні вхідних даних?
- Які ресурси є обмежувальними (тими, що обмежують покращення розв’язку)?
- Які змінні можуть бути змінені без порушення оптимальності розв’язку?
- Які нові обмеження або змінні можуть бути додані, і як вони вплинуть на розв’язок?

Отже, постоптимальний аналіз полягає у вивченні того, як змінюється оптимальний розв’язок при варіації параметрів задачі. Такий аналіз має велике практичне значення, оскільки у реальних умовах вихідні дані, як правило, не залишаються незмінними: можуть коливатись обсяги ресурсів, попит, витрати, ціни та інші параметри [7]. Постоптимальний аналіз допомагає завчасно

визначити, які саме зміни є критичними і можуть спричинити зміну оптимального плану, а які не мають суттєвого впливу на його структуру. Це забезпечує адаптивність рішень, підвищує їхню стійкість до зовнішніх змін та сприяє ефективнішому управлінню в умовах невизначеності.

Для здійснення постоптимального аналізу зазвичай використовуються додаткові дані, які формуються під час виконання симплекс-методу [8-9]. У процесі розв'язання задачі формується симплекс-таблиця, яка містить багато корисної інформації для подальшого аналізу. Основними її елементами є:

- Тіньові ціни (dual prices) — це показники, що демонструють, як зміниться значення цільової функції при невеликому збільшенні правої частини певного обмеження. Іншими словами, тіньова ціна відображає приріст цільової функції на одиницю збільшення ресурсу, пов'язаного з відповідним обмеженням, за умови фіксованості інших параметрів. Наприклад, якщо тіньова ціна для обмеження, що стосується трудових ресурсів, становить 5, то додатковий один працівник дозволить підвищити прибуток на 5 одиниць. Якщо ж тіньова ціна дорівнює нулю, це свідчить про надлишковість ресурсу у поточному рішенні.
- Запаси (slack або surplus змінні) – відображають відхилення значення змінної або обмеження від його межі. У випадку обмеження типу «менше або дорівнює» запас свідчить про залишок невикористаного ресурсу. Якщо запас нульовий, це означає, що відповідне обмеження повністю використане і є активним. Аналіз цих змінних дозволяє виявити як критичні обмеження, так і потенційні резерви ресурсів, які можуть стати в нагоді при зміні умов задачі.
- Діапазони допустимих змін (allowable ranges) – встановлюють межі, в яких можна змінювати значення коефіцієнтів цільової функції або правих частин обмежень без зміни структури оптимального базису. Це особливо важливо для задач, що вирішуються в умовах нестабільності або нечіткості даних. Наприклад, якщо коефіцієнт у цільовій функції може варіюватися в межах від 3 до 7 без зміни оптимального рішення, то вказаний інтервал вважається стабільним і не потребує корекції прийнятого плану.

Формальна постановка задачі постоптимального аналізу передбачає розгляд наступної класичної задачі лінійного програмування(ЗЛП)[5,10]:

$$Z=c_1x_1 + c_2x_2 +\ldots+c_nx_n \text{ , при}$$

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \ \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \ \leq b_2 \\ \ldots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \ \leq b_m \\ x_j \geq 0, \ j = 1 \ldots n \end{cases} \quad (1.2)$$

Після знайдення оптимального розв’язку (до прикладу, за допомогою симплекс-методу),закономірно може з’явитися запитання: як цей розв’язок буде поводитись при зміні:

- вектору коефіцієнтів c (у цільовій функції).
- правих частин b (у системі обмежень).
- елементів матриці A (коефіцієнтів при змінних).

Постоптимальний аналіз буває локальним (розгляд змін одного або кількох параметрів), а буває глобальним (аналіз усіх можливих варіацій вхідних даних) [10].

У цій роботі передбачено розробку програмного модуля, який дозволить здійснювати автоматизований постоптимальний аналіз ЗЛП, розв’язаних симплекс-методом. Основні задачі, що вирішуватимуться модулем:

- побудова таблиці за симплекс методом для початкової задачі.
- отримання основних параметрів рішення (базисні змінні, значення цільової функції, тіньові ціни, значення невідомих).
- знаходження діапазонів допустимих значень для коефіцієнтів цільової функції та обмежень.
- візуалізація результатів для користувача.

Програмний модуль має мати зрозумілий інтерфейс, зручність введення початкових даних, можливість збереження результатів та їх експорту у звичних форматах.

## 1.4 Особливості застосування та перспективи розвитку програмного модуля постоптимального аналізу ЗЛП

Постоптимальний аналіз задач лінійного програмування (ЗЛП) має вирішальне значення в умовах динамічного середовища, де параметри задачі можуть змінюватися після отримання початкового оптимального розв’язку. Створення програмного модуля, здатного ефективно виконувати такий аналіз, відкриває нові можливості для автоматизації прийняття управлінських, виробничих та економічних рішень. Це особливо актуально в ситуаціях, коли навіть незначні зміни в коефіцієнтах цільової функції чи правих частинах обмежень можуть призвести до втрати актуальності вже знайденого рішення.

Сучасні інформаційні системи у сфері менеджменту, логістики, фінансів та виробництва часто використовують методи математичної оптимізації для моделювання і вирішення задач [11–13]. У такому контексті програмний модуль постоптимального аналізу відіграє роль аналітичного інструменту, що дозволяє не лише знайти оптимальне рішення, але й оцінити його стійкість до змін, виявити критичні параметри моделі та приймати обґрунтовані рішення в умовах невизначеності. Завдяки цьому:

- можна швидко оцінити, чи зміниться розв’язок у разі зміни обсягів ресурсів або коефіцієнтів цільової функції.
- модуль допомагає ідентифікувати критичні обмеження, які найбільше впливають на результат.
- можливе застосування сценарного аналізу, що дозволяє розглянути кілька варіантів розвитку подій.

У процесі навчання студентів економічних та інженерних спеціальностей часто виникає потреба в інтерактивних засобах моделювання оптимізаційних задач. Програмний модуль може виступати у ролі дидактичного інструменту, що дозволяє:

- демонструвати вплив параметрів задачі на оптимальний розв’язок.
- візуалізувати симплекс-таблиці та кроки їх перетворення.
- проводити самостійні експерименти із варіаціями вхідних даних.

У багатьох галузях (машинобудування, харчова промисловість, енергетика) задачі ресурсного планування зводяться до ЗЛП [14-16]. У таких умовах:

- постоптимальний аналіз дозволяє адаптувати план виробництва при зміні доступності ресурсів.
- модуль може бути інтегрований у MES-системи (Manufacturing Execution Systems) для динамічної оптимізації.
- дозволяє уникнути повторного повного перерахунку задачі при незначних змінах параметрів.

Існує кілька напрямків, у яких може бути розширено функціонал модуля:

- Підтримка нелінійного програмування: включення можливостей для аналізу задач із нелінійними обмеженнями або цільовими функціями.
- Аналіз двоїстих задач: реалізація автоматичного виведення двоїстої задачі та аналіз її економічного змісту.
- Інтеграція з базами даних: забезпечення автоматичного отримання початкових даних із зовнішніх джерел.
- Візуалізація результатів: реалізація інтерактивного графічного інтерфейсу для представлення результатів аналізу.

У подальшому доцільно забезпечити кросплатформенність модуля:

- реалізація веб-версії з доступом через браузер.
- підтримка мобільних пристроїв для використання в польових умовах.

Перспективним є переклад інтерфейсу модуля кількома мовами та адаптація під різні міжнародні стандарти:

- підтримка різних форматів представлення вхідних даних (наприклад, JSON, XML, CSV).
- локалізація мови інтерфейсу.

Модуль може бути використаний у наукових дослідженнях, пов’язаних із:

- побудовою адаптивних моделей планування.
- аналізом стійкості економічних моделей.
- верифікацією нових алгоритмів постоптимального аналізу.
- моделюванням складних багатокритеріальних задач.

## 1.5 Висновки до розділу 1

Проведений аналіз предметної області постоптимального аналізу в ЗЛП дозволив окреслити ключові теоретичні та практичні аспекти цієї важливої складової математичного моделювання. Постоптимальний аналіз розглядається як ефективний інструмент для оцінки стійкості оптимального рішення до змін параметрів задачі, зокрема коефіцієнтів цільової функції, правих частин обмежень і елементів матриці обмежень.

З'ясовано, що в умовах реальної економіки та виробничого планування, де постійно відбуваються зміни в ресурсах, цінах чи попиті, постоптимальний аналіз є невід'ємною частиною процесу прийняття рішень. Використання результатів симплекс-методу, зокрема таких характеристик як тіньові ціни, запасні змінні та допустимі діапазони, дозволяє детально дослідити чутливість оптимального рішення та визначити критичні параметри моделі.

Також було розглянуто основні підходи до реалізації аналізу змін у структурі задачі: дослідження впливу варіацій цільових коефіцієнтів, аналіз правих частин обмежень, а також врахування додавання нових змінних чи обмежень.

# 2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ ПОСТОПТИМАЛЬНОГО АНАЛІЗУ ЗЛП

## 2.1 Розробка структури програмного модуля

Програмний модуль постоптимального аналізу задач лінійного програмування втілено з огляду на принципи модульності, масштабованості та оперативності використання. Загальна структура рішення розпочата на основі поділу функціональності на логічні підсистеми, кожна з яких відповідає за окремий аспект розрахунків або взаємодії з користувачем. Основні компоненти системи програми:

1. Модуль графічного інтерфейсу користувача (gui/).

Цей компонент реалізує інтерфейс користувача, що дозволяє вводити дані задачі, запускати розв’язання та переглядати результати аналізу. Головний файл – app.py. Він побудований з використанням бібліотеки Tkinter, яка забезпечує кросплатформну роботу та простоту у створенні елементів керування. Інтерфейс передбачає можливість введення коефіцієнтів цільової функції, матриці обмежень, а також варіанти розширеного постоптимального аналізу.

2. Основний виконавчий файл (main.py).

Файл main.py – місце початку роботи додатку. Він ініціалізує обов'язкові частини програми, запускає графічний інтерфейс та координує діяльність підсистем. Від цього саме місця відбувається виклик головних логічних функцій, пов'язаних із розв'язанням проблеми та виконанням аналізу.

3. Обчислювальне ядро (solver/).

Це модуль містить реалізації симплекс-методу та парсингу даних вхідної інформації. Його структура розділена на декілька файлів:

- simplex.py – містить покрокову реалізацію симплекс-алгоритму, включно з побудовою симплекс-таблиць, розрахунком оцінок та вибором ведучих елементів.

- lp_solver.py — обгортає головні методи розв'язання проблеми і викликає побудову і проходження симплекс-таблиці.
- parser.py — відповідає за інтерпретацію введених користувачем даних та перетворення їх у формат, придатний для обчислень.

4. Модуль постоптимального аналізу (analysis/).

Компонент sensitivity.py виконує розширений постоптимальний аналіз, зокрема:

- аналіз діапазонів зміни коефіцієнтів цільової функції, при яких оптимальний план залишається незмінним;
- визначення впливу змін правих частин обмежень на розв’язок;
- оцінка допустимих змін в коефіцієнтах матриці обмежень.

Результати дослідження передаються в модуль візуалізації для подальшого відображення.

Архітектура програмного забезпечення відповідає принципам Model-View-Controller (MVC) [17], що є загальноприйнятим підходом до розробки структурованих та масштабованих програмних систем. Згідно з цією парадигмою, програмне забезпечення умовно поділено на три основні компоненти:

- Model — реалізована у модулях solver/ та analysis/, де зосереджено основну бізнес-логіку. Зокрема, модуль solver/ відповідає за побудову та розв’язання задачі лінійного програмування із використанням симплексного методу або альтернативних методів. Модуль analysis/ забезпечує проведення постоптимального аналізу, включаючи обчислення діапазонів допустимих значень параметрів, визначення тіньових цін та аналіз чутливості моделі до змін.
- View — реалізована в компоненті gui/, який відповідає за побудову графічного інтерфейсу користувача. Саме тут відбувається введення початкових даних, виведення результатів розв’язання та аналітичної обробки, а також надається доступ до функцій покрокового розв’язання та перегляду симплекс-таблиць. Інтерфейс створено з урахуванням принципів доступності та зручності

у використанні, що особливо важливо в контексті навчального програмного забезпечення.

- Controller — функції контролера виконує модуль main.py, який координує взаємодію між Model та View. Він відповідає за запуск програми, обробку дій користувача, передачу вхідних даних до відповідних обчислювальних модулів та отримання результатів для подальшого відображення в інтерфейсі.

Такий підхід забезпечує чітке розмежування відповідальностей між компонентами, що, у свою чергу, сприяє підвищенню гнучкості та підтримуваності коду. Окремі частини системи можуть бути змінені або вдосконалені без необхідності вносити правки до всієї програми. Наприклад, можливо додати нові методи розв’язування задачі або алгоритми аналізу без зміни інтерфейсу чи логіки керування.

Крім того, модульність архітектури дозволяє легко повторно використовувати частини коду в інших програмних проєктах або адаптувати програму до нових вимог. Вона також полегшує процес тестування та налагодження окремих компонентів, забезпечуючи ефективну розробку та подальший розвиток системи.

На рисунку 2.1 зображено структуру програмного модуля, яка відображає основні компоненти системи, їхні функціональні ролі та взаємозв’язки між собою.


Рисунок 2.1 – Структура програмного модуля

## 2.2 Розробка алгоритму роботи системи

Для успішної роботи програмного модуля постоптимального аналізу задач лінійного програмування потрібно чітко структурований алгоритм. Цей алгоритм повинен гарантувати правильний порядок дій – від вводу початкових даних до тлумачення результатів аналізу. Розробка алгоритму допомагає не лише впорядкувати функціонування модуля, але й забезпечити точність результатів, зручність в користуванні та можливість подальшого розширення.

Алгоритм повинен охоплювати усі ключові етапи: перевірку коректності вхідних даних, формування математичної моделі задачі, розв’язання задачі симплекс-методом, збереження базисного розв’язку, а також виконання постоптимального аналізу на основі отриманого рішення. Важливо також передбачити обробку помилок та виняткових ситуацій, що можуть виникнути під час обчислень або при зміні параметрів задачі.

На рисунку 2.2 показано загальну схему роботи системи, яка ілюструє основні етапи обробки задачі лінійного програмування, її розв’язання та подальшого аналізу. Така схема допомагає візуально представити логіку функціонування модуля та визначити взаємозв’язки між окремими його компонентами. Це, своєю чергою, сприяє кращому розумінню структури системи, полегшує її підтримку та удосконалення в майбутньому.

Робота програмного модуля побудована на послідовному виконанні ряду етапів, що охоплюють увесь процес — від введення даних до виведення результатів оптимізації та постоптимального аналізу. Кожен крок реалізовано таким чином, щоб забезпечити зручність використання, точність обчислень та стійкість до помилок користувача.

1. Ініціалізація графічного інтерфейсу.

На першому етапі запускається графічний інтерфейс користувача (GUI), який забезпечує інтерактивну взаємодію. Інтерфейс дозволяє вводити початкові дані задачі, переглядати результати розв’язання, а також здійснювати покрокову демонстрацію обчислень.

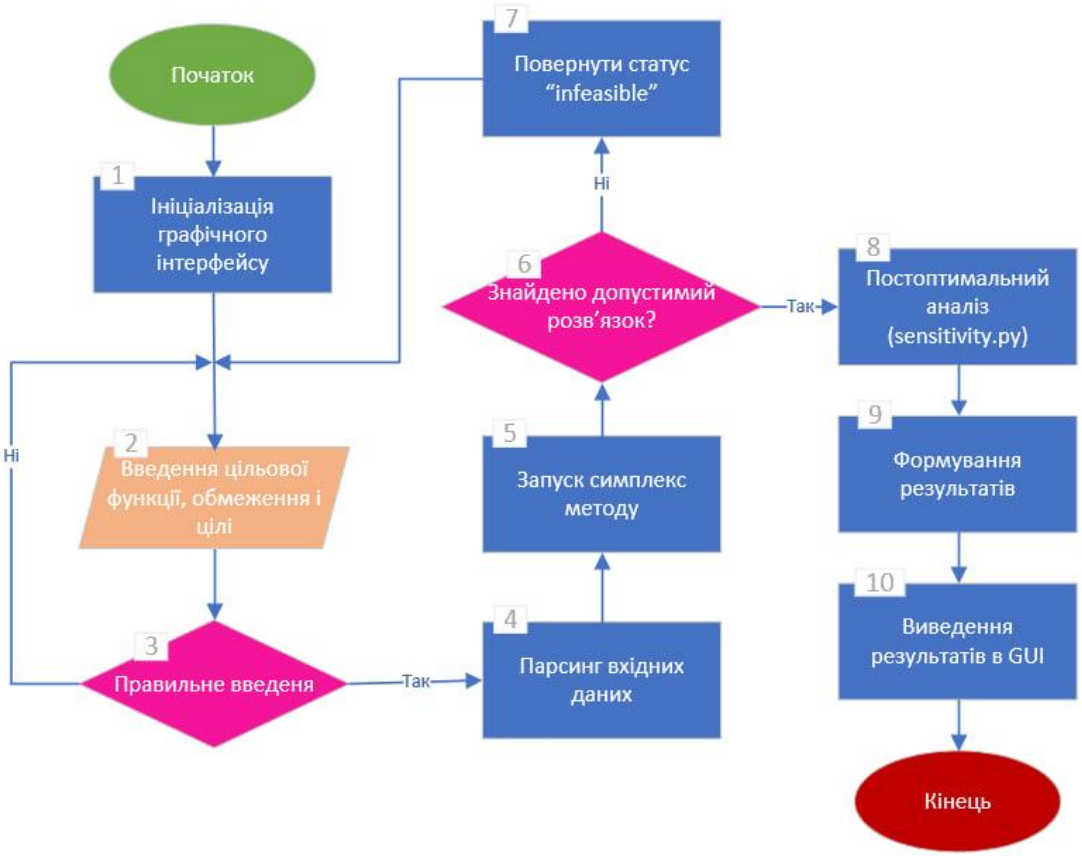


Рисунок 2.2 – Алгоритм функціонування системи

2. Введення цільової функції, обмежень і типу задачі.

Користувач за допомогою інтерфейсу вводить структуру задачі: коефіцієнти цільової функції, систему лінійних обмежень, а також тип оптимізації – максимізація або мінімізація.

3. Перевірка правильності введення.

Перед обчисленням система проводить перевірку коректності введених даних: перевіряється відповідність розмірності, наявність числових значень, правильність знаків обмежень тощо. У разі виявлення помилок виводяться підказки для їх виправлення.

4. Парсинг вхідних даних.

Після успішної валідації дані проходять парсинг (розбір структур), що дозволяє перетворити їх у формат, придатний для математичної обробки. Цей етап також виконує внутрішню нормалізацію обмежень і цільової функції.

5. Запуск початкового етапу симплекс-методу.

Здійснюється побудова початкової симплекс-таблиці та знаходження допустимого базисного розв’язку. У разі використання двофазного методу — виконується перша фаза з введенням штучних змінних.

6. Перевірка наявності допустимого розв’язку.

Якщо не вдається знайти допустиме початкове рішення, система повертає статус "infeasible", що означає відсутність розв’язку у припустимій області.

7. Повернення статусу "infeasible".

У разі неможливості знайти допустиме рішення, обчислення припиняється, користувачеві повідомляється про недопустимість задачі.

8. Основний етап симплекс-методу.

Якщо допустимий базис знайдено, виконується ітераційна частина симплекс-методу для пошуку оптимального рішення задачі. Результати кожного кроку можуть виводитись для перегляду.

9. Отримання оптимального розв’язку.

У результаті завершення симплексного алгоритму система фіксує оптимальне значення цільової функції та відповідні значення змінних. Розв’язок передається до модуля подальшого аналізу.

10. Постоптимальний аналіз (модуль sensitivity.py).

Після знаходження оптимуму активується модуль постоптимального аналізу, який визначає допустимі діапазони змін коефіцієнтів цільової функції та правих частин обмежень, тіньові ціни (dual values) та інші параметри чутливості.

11. Формування результатів.

Отримані числові результати аналізу обробляються для подальшого виведення: формуються таблиці зі змістовними заголовками, значенням цільової функції, аналізом змін параметрів тощо.

12. Виведення результатів у графічному інтерфейсі.

Заключний етап — відображення результатів у GUI: оптимальний розв’язок, покрокові симплекс-таблиці, результати постоптимального аналізу. Користувачг має змогу зберегти інформацію або провести новий розрахунок із зміненими параметрами.

Цей алгоритм ілюструє принципи функціонування програмного модуля, враховуючи помилки введення, перевірку правильності даних, пошук оптимальних рішень та проведення постоптимального аналізу.

### 2.3 Розробка UML-діаграми класів

Для організації структурованої реалізації програмного модуля постоптимального аналізу задач лінійного програмування була створена UML-діаграма класів(рис. 4.2). Вона надає можливість візуально продемонструвати зв’язки між ключовими елементами системи, їхні ролі та методи взаємодії. Завдяки діаграмі класів можна чітко простежити логіку побудови застосунку, розмежування відповідальності між компонентами, а також загальну архітектуру програмного забезпечення.

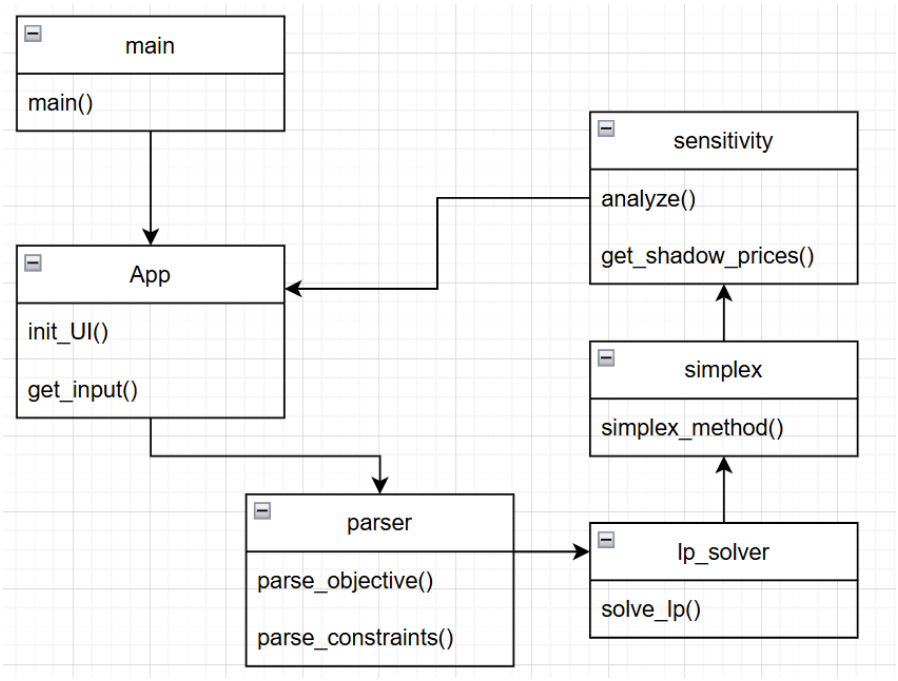


Рисунок 4.2 – UML-діаграма програмного модуля

У центрі архітектурної структури програмного модуля розташований головний клас App, який виконує ключову роль контролера у моделі проєктування типу MVC (Model-View-Controller). Саме цей клас здійснює основну логіку керування застосунком, забезпечуючи зв’язок між інтерфейсом користувача, обробкою даних та алгоритмічними модулями. Через метод init_UI() реалізується початкова ініціалізація графічного інтерфейсу, де користувач може ввести вхідні дані: цільову функцію, обмеження, тип задачі (максимізація чи мінімізація) тощо. Окрім цього, метод get_input() відповідає за обробку введених даних, перевірку їх коректності та передачу далі по системі.

Коли користувач вводить інформацію, вона передається до модуля парсингу, в якому реалізовано обробку текстових рядків із математичними виразами. Зокрема, метод parse_objective() виконує розбір і перетворення цільової функції у структурований формат (наприклад, список коефіцієнтів), тоді як метод parse_constraints() відповідає за ідентифікацію обмежень, включно з коефіцієнтами при змінних, знаками нерівностей та вільними членами. Парсинг дозволяє уніфікувати дані для подальшого обчислювального аналізу, незалежно від способу їх введення користувачем.

Після обробки, структуровані дані передаються до компонента lp_solver, який виконує обчислювальну частину проєкту. Цей модуль, зокрема метод solve_lp(), відповідає за ініціалізацію процесу розв’язання задачі лінійного програмування. Основна обчислювальна логіка реалізується в межах класу Simplex, який містить метод simplex_method(). Цей метод крок за кроком виконує симплекс-алгоритм: побудову симплекс-таблиці, вибір ведучих змінних, обчислення оцінок і перевірку оптимальності поточного плану. У процесі розв’язання враховуються випадки виродження, необмеженості та наявності кількох оптимальних рішень.

По завершенні основного етапу розв’язання, результати передаються до окремого модуля, що відповідає за постоптимальний аналіз — так званого "модуля чутливості". У цьому компоненті реалізовані методи analyze() та get_shadow_prices(). Метод analyze() забезпечує загальний аналіз стійкості розв’язку, перевіряє, які обмеження є активними, і визначає діапазони допустимих змін у коефіцієнтах цільової функції та правих частин обмежень. Натомість get_shadow_prices() обчислює тіньові ціни (dual values), що відіграють важливу роль у прийнятті рішень щодо ресурсів та їх вартості в контексті лінійного програмування.

Управління запуском застосунку здійснюється за допомогою класу Main, у якому викликається метод main(). Ця функція ініціалізує основний клас App, запускає інтерфейс користувача та координує подальші етапи обробки даних, розв’язання задачі та аналізу результатів. Таким чином, структура програми забезпечує логічну послідовність і зручність в експлуатації.

Загалом, запропонована архітектура програмного модуля відзначається чітким розподілом обов’язків між компонентами. Це дозволяє легко масштабувати систему, розширювати її функціональність (наприклад, додавання нових методів оптимізації або візуалізації результатів), а також спрощує тестування та супровід. Застосування об’єктно-орієнтованого підходу підвищує гнучкість і підтримуваність коду, що особливо важливо для складних інженерних та наукових застосунків, зокрема таких, які орієнтовані на оптимізацію ресурсів, планування, логістику або виробниче управління.

## 2.4 Висновки до розділу 2

У другому розділі було детально спроєктовано архітектуру програмного модуля, що реалізує постоптимальний аналіз задач лінійного програмування. Структура модуля розроблена на основі принципів модульності, що забезпечує його гнучкість, простоту супроводу та подальшого розширення.

Програмний засіб має чітке розділення на підсистеми, кожна з яких відповідає за окремий етап обчислень або взаємодії з користувачем.

Завдяки впровадженню симплекс-методу в обчислювальному ядрі, користувач має змогу виконувати розв’язання задач з високим рівнем деталізації – аж до побудови симплекс-таблиць та аналізу ведучих елементів. Розширений модуль постоптимального аналізу дозволяє оцінити чутливість оптимального розв’язку до змін параметрів моделі, що значно підвищує практичну цінність інструменту.

Розроблений алгоритм функціонування модуля охоплює всі ключові етапи роботи – від введення вихідних даних до отримання висновків після аналізу. Це дозволяє гарантувати логічну послідовність дій користувача, забезпечити правильність обчислень та уникнути критичних помилок.

UML-діаграма класів слугує інструментом візуалізації загальної архітектури проєкту, чітко ілюструючи взаємозв’язки між основними компонентами. Її побудова сприяє спрощенню подальшого розвитку програмного модуля, зокрема додавання нових функцій або адаптації до інших типів оптимізаційних задач.

Спроєктований програмний модуль демонструє ефективне поєднання класичних методів математичного програмування з сучасними підходами до розробки програмного забезпечення. Це створює передумови для його подальшого використання у наукових дослідженнях, навчальному процесі та прикладних задачах лінійного програмування.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ ПОСТОПТИМАЛЬНОГО АНАЛІЗУ ЗЛП

## 3.1 Обґрунтування вибору мови програмування, бібліотек та підходу до реалізації

Для виконання постоптимального аналізу задач лінійного програмування було вивчено декілька ключових бібліотек, які пропонують можливості для формулювання та розв’язання таких задач і зображено їх у таблиці 3.1.

Таблиця 3.1 – Порівняльна характеристика бібліотек лінійного програмування для реалізації модуля

| Критерій | Розроблена програма | IBM ILOG CPLEX | LINDO / LINGO |
|---|---|---|---|
| Платформа | Кросплатформна (Python + Tkinter) | Комерційна, багатоплатформна (C++, Python, Java API тощо) | Комерційна, Windows/Linux |
| Тип | Освітній / науковий інструмент | Промисловий оптимізаційний рушій | Промисловий інструмент з GUI |
| Мова реалізації | Python | Внутрішньо – C++, доступні API для різних мов | Власна мова моделювання + API |
| Зручність користування | Візуальний інтерфейс з ручним введенням даних | Професійний API + графічні інструменти | Інтуїтивний GUI, скриптова мова |
| Візуалізація симплекс-таблиці | Так, поетапна текстова таблиця | Ні (внутрішній процес прихований) | Так (інтерактивне дерево рішень) |
| Можливість налаштування | Висока – відкритий код, Python | Висока – API дозволяє налаштовувати будь-який параметр | Середня – залежить від мови моделювання |
| Ціна | Безкоштовна | Платна (академічна безкоштовна версія доступна) | Платна (студентські ліцензії також доступні) |

Враховуючи вказані фактори, для реалізації було обрано мову програмування Python, а також бібліотеки для лінійного програмування, такі як PuLP, SciPy.optimize.linprog та GLPK. Python — це мова програмування, яка пропонує різноманітні інструменти для математичних обчислень. Вона також має потужні можливості візуалізації завдяки бібліотекам, таким як matplotlib та plotly, і забезпечує зручну платформу для розробників.

Ще однією перевагою Python є його інтеграція з середовищами наукових обчислень, такими як Jupyter Notebook. Це дозволяє створювати інтерфейси для досліджень і навчання без потреби у розробці окремого GUI-додатку.

Основна логіка постоптимального аналізу включає: розрахунок базового розв'язку задачі ЛП, побудову таблиці чутливості (з використанням симплекс-таблиць або методів чисельного аналізу) визначення діапазону допустимих змін параметрів, що не змінюють базисного розв'язку, можливість дослідження впливу зміни обмежень або функції цілі на оптимальний розв'язок, побудову візуальних графіків, що ілюструють результати аналізу.

Ці функції реалізуються у вигляді окремих модулів, які взаємодіють між собою через чітко визначені інтерфейси, що дозволяє розширювати систему у майбутньому.

Тепер розглянемо підходи до реалізації програмного модуля.

Підходи до реалізації програмного модуля можна умовно поділити на декілька категорій [18]:

1. Монолітна архітектура – у ній всі функціональні компоненти реалізуються в межах одного цільного програмного середовища. Цей метод є легшим у впровадженні, але в значній мірі знижує можливості масштабування, ускладнює тести та використання окремих фрагментів коду повторно.

2. Модульна архітектура – це підхід, при якому програмний модуль розбивається на окремі логічні компоненти. Серед цих компонентів можна виділити: модуль для введення та редагування даних, обчислювальний блок,

модуль для аналізу результатів та візуалізації тощо. Цей підхід відповідає сучасним вимогам гнучкої розробки і дає змогу оперативно вносити коригування в окремі елементи системи, не впливаючи на інші частини.

3. Клієнт-серверна модель – реалізація частини функціоналу на стороні сервера, а частини – на стороні клієнта, дозволяє створити веб-додаток для інтерактивного доступу до функцій постоптимального аналізу. Проте для даного типу задачі та в рамках бакалаврської роботи створення повноцінного веб-додатку не є доцільним через складність та обмежені ресурси.

4. Інтеграція з математичними бібліотеками або пакетами (наприклад, SciPy, PuLP, GLPK, CPLEX) – дозволяє скористатися перевіреними рішеннями для вирішення задач ЛП та зосередитися на розробці модуля постоптимального аналізу, не витрачаючи ресурсів на реалізацію власного.

5. Інтерфейс командного рядка або графічний інтерфейс користувача (GUI) – вибір між цими підходами залежить від цільової аудиторії. GUI є зручнішим для кінцевих користувачів, проте створення якісного графічного інтерфейсу потребує додаткових ресурсів. У той же час CLI дозволяє швидше реалізувати базовий функціонал та легко автоматизувати виконання серії обчислень.

З урахуванням специфіки постоптимального аналізу задач ЛП до програмного модуля висуваються наступні вимоги:

- Можливість зміни обмежень та коефіцієнтів функції цілі для перевірки стійкості отриманого розв’язку;
- Підтримка декількох сценаріїв постоптимального аналізу, таких як аналіз чутливості, аналіз зміни правих частин обмежень, аналіз діапазону допустимих значень коефіцієнтів цільової функції;
- Масштабованість та адаптивність до задач з різною розмірністю;
- Надійність та точність обчислень, що обумовлює необхідність використання перевірених математичних бібліотек;

- Зручність у використанні та можливість інтеграції з іншими системами або модулями для попередньої обробки даних або побудови моделей.

Таблиця 3.1 – Порівняльна характеристика обраного підходу

| Критерій | Монолітний підхід | Модульний підхід (обраний) | Клієнт-сервер |
|---|---|---|---|
| Гнучкість | Низька | Висока | Висока |
| Зручність у розробці | Середня | Висока | Низька |
| Легкість модифікації | Низька | Висока | Середня |
| Ресурси на реалізацію | Низькі | Середні | Високі |
| Простота вивчення коду | Середня | Висока | Низька |

### 3.3 Розробка програмного модуля постоптимального аналізу

Обчислення оптимального розв’язку здійснюється за допомогою функції linprog() з модуля scipy.optimize, яка реалізує симплексний метод або метод внутрішньої точки залежно від заданих параметрів [19–20]. Функція linprog() дозволяє розв’язувати задачі лінійного програмування у стандартній формі мінімізації цільової функції при лінійних обмеженнях. Завдяки широкому спектру параметрів вона забезпечує гнучке налаштування способу розв’язання, вибір алгоритму (method='simplex', method='highs', method='interior-point' тощо), а також контроль над точністю, кількістю ітерацій та веденням журналу обчислень.

Метод highs, який наразі є рекомендованим за замовчуванням, поєднує в собі сучасні реалізації симплексного та двоїстого симплексного методів, а

також варіанти методу внутрішньої точки (interior-point), що забезпечують високу швидкодію навіть на задачах великої розмірності. Крім того, linprog() дозволяє обробляти як задачі з обмеженнями типу “менше або дорівнює”, так і рівняння, що робить її універсальним інструментом для багатьох прикладних сценаріїв.

Важливою перевагою є також зручність інтеграції linprog() з іншими науковими модулями Python, такими як NumPy і Pandas, що дає можливість ефективно працювати з великими обсягами вхідних даних, проводити попередній аналіз, візуалізацію та постобробку результатів. У контексті побудови програмного модуля для постоптимального аналізу ця функція виступає як основний обчислювальний механізм, результати роботи якого можуть бути далі проаналізовані на предмет чутливості до змін вхідних параметрів, побудови діапазонів допустимих значень та визначення тіньових цін.

- Перед викликом функції необхідна обробка вхідних даних: Задача мінімізації використовується за замовчуванням, тому при задачі максимізації – коефіцієнти цільової функції множаться на -1.
- Обмеження перетворюються у формат A_ub, b_ub згідно з вимогами до стандартної форми.

```
from scipy.optimize import linprog
def solve_lp(c, A, b, sense='max'):
    if sense == 'max':
        c = [-coef for coef in c]
    result = linprog(c, A_ub=A, b_ub=b, method='simplex')
    if result.success:
        opt_value = -result.fun if sense == 'max' else result.fun
        return opt_value, result.x
```

```
    else:
        raise ValueError("Задача не має розв’язку або невірно задана.")
```

Після знаходження оптимального розв’язку, важливо оцінити, наскільки він є стійким до змін у вхідних параметрах. У межах постоптимального аналізу реалізовано такі функції:

1. Зміна коефіцієнтів цільової функції

Для кожного коефіцієнта обчислюється діапазон значень, за якого базис залишається незмінним, тобто оптимальний розв’язок не змінюється. Це дає змогу виявити критичні змінні та провести аналіз чутливості.

```
def analyze_objective_impact(lp_model, step=0.1, delta=1.0):
    c_original = lp_model.c.copy()
    results = []
    for i in range(len(c_original)):
        c_varied = c_original.copy()
        row = []
        for d in [-delta, 0, delta]:
            c_varied[i] = c_original[i] + d
            try:
                z, x = solve_lp(c_varied, lp_model.A, lp_model.b,
lp_model.sense)
                row.append((round(c_varied[i], 2), round(z, 2)))
            except:
                row.append((round(c_varied[i], 2), "Немає розв’язку"))
        results.append(row)
    return results
```

2. Зміна вектору обмежень

Аналіз впливу на розв’язок при зміні ресурсів (вектору правих частин). Це дозволяє оцінити, наскільки допустимо змінювати доступні ресурси без втрати оптимальності.

```
def analyze_rhs_impact(lp_model, delta=1.0):
    b_original = lp_model.b.copy()
    results = []
    for i in range(len(b_original)):
        b_varied = b_original.copy()
        row = []
        for d in [-delta, 0, delta]:
            b_varied[i] = b_original[i] + d
            try:
                z, x = solve_lp(lp_model.c, lp_model.A, b_varied,
lp_model.sense)
                row.append((round(b_varied[i], 2), round(z, 2)))
            except:
                row.append((round(b_varied[i], 2), "Немає розв’язку"))
        results.append(row)
    return results
```

3. Формування аналітичного звіту

Зібрана інформація виводиться у вигляді таблиць, що містять значення цільової функції при варіації відповідних параметрів. Це дозволяє користувачу приймати обґрунтовані рішення, не проводячи повний перерахунок задачі вручну. Такі таблиці можуть відображати зміну оптимального розв’язку при зміні коефіцієнтів цільової функції, правих частин обмежень або значень змінних. Зокрема, користувач має змогу побачити діапазони допустимих значень, у межах яких поточний оптимальний розв’язок залишається незмінним, а також значення тіньових цін, що відображають граничну зміну цільової функції при зміні ресурсів.

Вивід у табличному форматі сприяє кращому розумінню впливу кожного параметра на результат оптимізації, дозволяючи візуально оцінити чутливість системи. Це особливо важливо при прийнятті управлінських

рішень, плануванні ресурсів або проведенні економічного аналізу. Крім того, реалізація такого представлення дозволяє зручно інтегрувати результати з іншими інструментами аналізу, зокрема електронними таблицями або базами даних, що розширює можливості подальшої обробки даних.

Також користувач може використовувати ці таблиці як навчальний матеріал для глибшого засвоєння теми лінійного програмування та постоптимального аналізу, оскільки вони ілюструють не лише кінцевий результат, а й поведінку моделі при зміні вхідних умов.

### 3.4 Тестування роботи програмного модуля

Після реалізації основного функціоналу програми було проведено тестування із залученням типових задач лінійного програмування. Основна мета тестування – перевірити коректність реалізації симплекс-методу, функціональність модуля постоптимального аналізу та зручність взаємодії з інтерфейсом. Тестові задачі охоплювали як прості приклади з одним обмеженням і двома змінними, так і більш складні моделі з кількома обмеженнями та більшим числом змінних. Це дозволило переконатися у стійкості алгоритмів, здатності програми обробляти різні типи вхідних даних та забезпечувати правильне знаходження оптимального розв’язку.

Особлива увага приділялася перевірці правильності побудови симплекс-таблиць на кожному етапі розв’язання, відповідності обчислених коефіцієнтів змінним, а також точності розрахунків цільової функції. У межах постоптимального аналізу було протестовано механізми визначення діапазонів допустимих змін коефіцієнтів цільової функції та обмежень, а також обчислення тіньових цін і оцінка впливу варіацій ресурсів на результат. Розглянемо класичну задачу максимізації прибутку (приклад 1):

$$Z = 3x_1 + 5x_2 \rightarrow max$$

$$\begin{cases} 2x_1 + x_2 \leq 18 \\ 2x_1 + 3x_2 \leq 42 \\ 3x_1 + x_2 \leq 24 \\ x_1, x_2 \geq 0 \end{cases} \quad (3.1)$$

Після запуску програми main.exe у вхідній формі користувач задає коефіцієнти та обмеження, після чого запускає розв’язання.

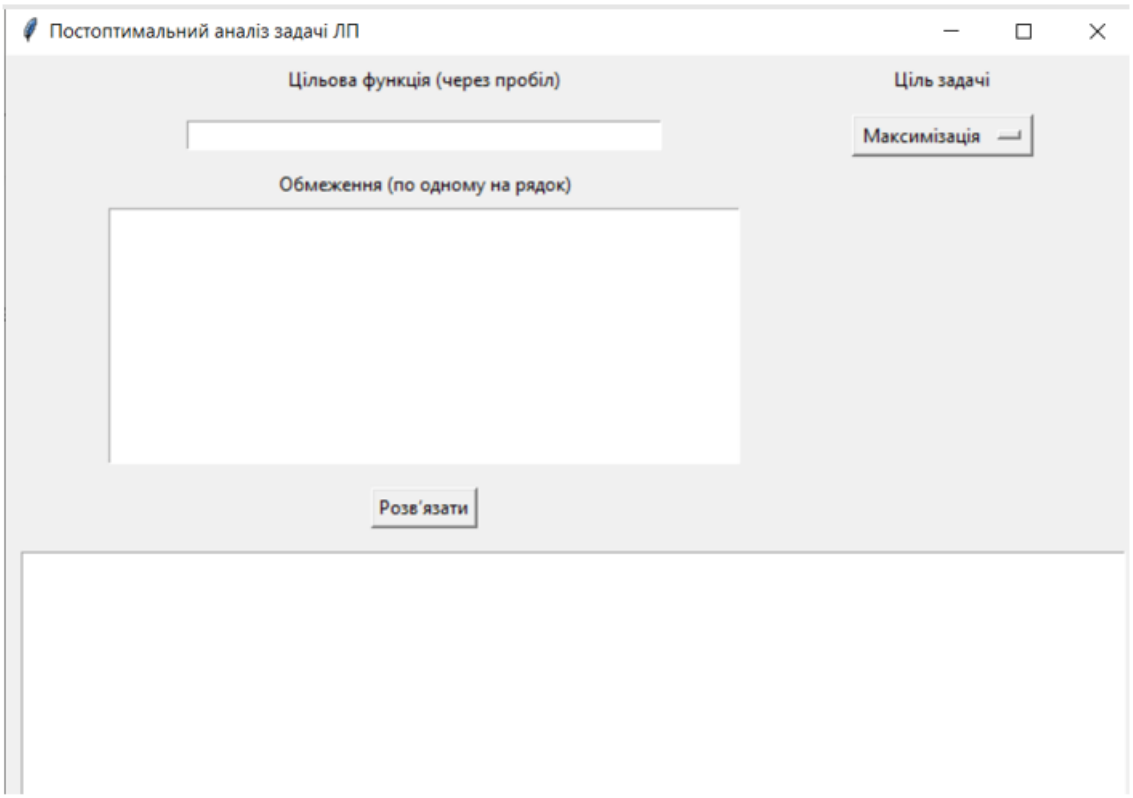


Рисунок 3.1 – Головне вікно програми

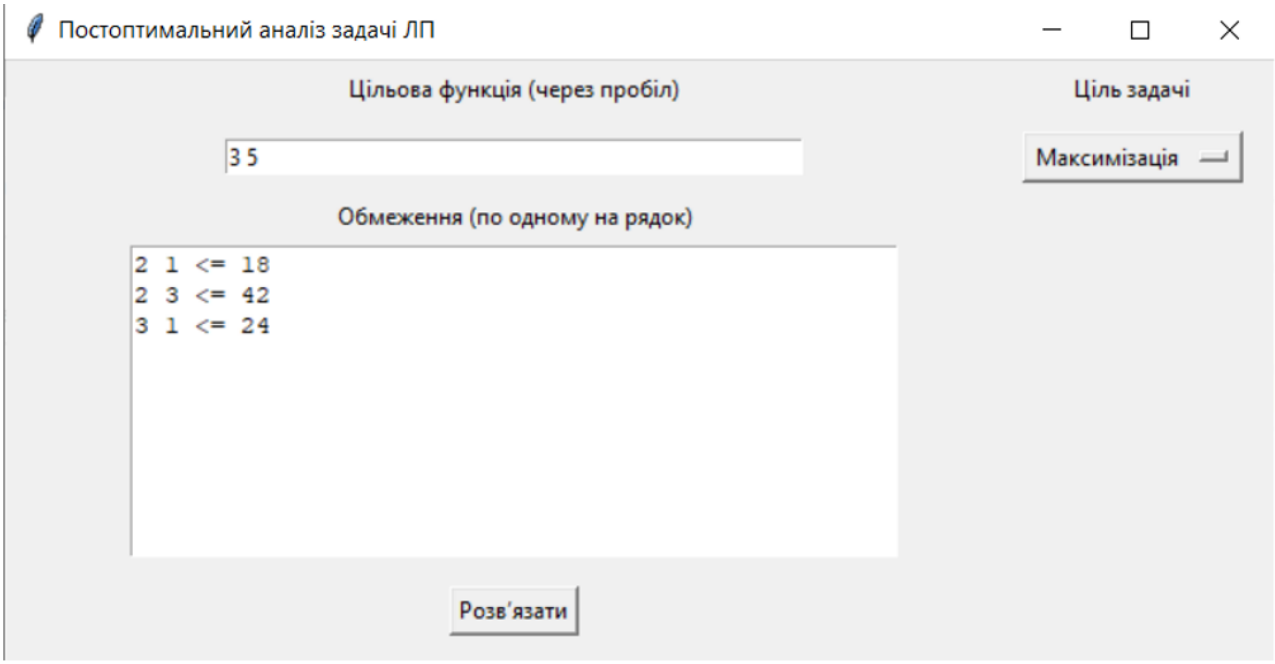


Рисунок 3.2 – Вікно для введення вхідних даних 1

```
=== СИМПЛЕКС-ТАБЛИЦІ (2 кроків) ===

Крок 1:
              x1        x2        s1        s2        s3       RHS
      ------------------------------------------------------------
  r1:     2.0000    1.0000    1.0000    0.0000    0.0000   18.0000
  r2:     2.0000    3.0000    0.0000    1.0000    0.0000   42.0000
  r3:     3.0000    1.0000    0.0000    0.0000    1.0000   24.0000
 obj:    -3.0000   -5.0000    0.0000    0.0000    0.0000    0.0000


Крок 2:
              x1        x2        s1        s2        s3       RHS
      ------------------------------------------------------------
  r1:     1.3333    0.0000    1.0000   -0.3333    0.0000    4.0000
  r2:     0.6667    1.0000    0.0000    0.3333    0.0000   14.0000
  r3:     2.3333    0.0000    0.0000   -0.3333    1.0000   10.0000
 obj:     0.3333    0.0000    0.0000    1.6667    0.0000   70.0000


=== ОПТИМАЛЬНИЙ РОЗВ'ЯЗОК ===
Ціль: Максимізація
Оптимальне значення: -70.000000

Змінні:
  x1 = 0.000000
  x2 = 14.000000
```

Рисунок 3.3 – Результат роботи програми для прикладу 1

```
Тіньові ціни (двоїсті змінні):
  Тіньова ціна показує, як зміниться оптимальне значення цільової функції при збільшенні
  правої частини відповідного обмеження на 1 одиницю.
  constraint_1: 0.000000
  constraint_2: 1.666667
  constraint_3: 0.000000

Діапазони зміни правих частин (b):
  Ці діапазони показують, в яких межах можна змінювати праві частини обмежень, щоб зберегти
  поточний базис оптимальним.  За їх межами базис
може змінитися, і оптимальне значення також може змінитися.
  b1 ∈ [8.0000, 28.0000]
    Перевірка меж діапазону b1:
      При b1 на нижній межі (8.0000), оптимальний план: { 'x1': 0.00, 'x2': 8.00 }
      При b1 на верхній межі (28.0000), оптимальний план: { 'x1': 0.00, 'x2': 14.00 }
  b2 ∈ [32.0000, 52.0000]
    Перевірка меж діапазону b2:
      При b2 на нижній межі (32.0000), оптимальний план: { 'x1': 0.00, 'x2': 10.67 }
      При b2 на верхній межі (52.0000), оптимальний план: { 'x1': 0.00, 'x2': 17.33 }
  b3 ∈ [14.0000, 34.0000]
    Перевірка меж діапазону b3:
      При b3 на нижній межі (14.0000), оптимальний план: { 'x1': 0.00, 'x2': 14.00 }
      При b3 на верхній межі (34.0000), оптимальний план: { 'x1': 0.00, 'x2': 14.00 }

=== Підсумковий аналіз ===
  Враховуючи зміни в правих частинах обмежень (b), можливий максимальний дохід перебуватиме в межах:
  40.0000 ≤ Fmax ≤ 86.6667

  А оптимальний план виробництва продукції складатиме:
  Початковий оптимальний план: { 0.00; 14.00 }
  { 0.00; 8.00 } ≤ хопт ≤ { 0.00; 17.33 }
```

Рисунок 3.3 – Продовження

Розглянемо приклад 2. Математична модель задачі лінійного програмування має такий вигляд:

$$Z = 3x_1 + 3x_2 + 4x_3 + 7x_4 \rightarrow max$$
$$\begin{cases} x_1 + 3x_2 + x_3 + x_4 \leq 9 \\ 5x_1 + 3x_2 + 2x_3 + x_4 \leq 60 \\ x_1 + 3x_2 + 5x_3 + 8x_4 \leq 50 \\ x_1, x_2, x_3, x_4 \geq 0 \end{cases} \quad (3.2)$$

На рисунку 3.4 представлено вхідні дані задачі. У ній наведено коефіцієнти при змінних у цільовій функції, коефіцієнти обмежень та вільні члени.

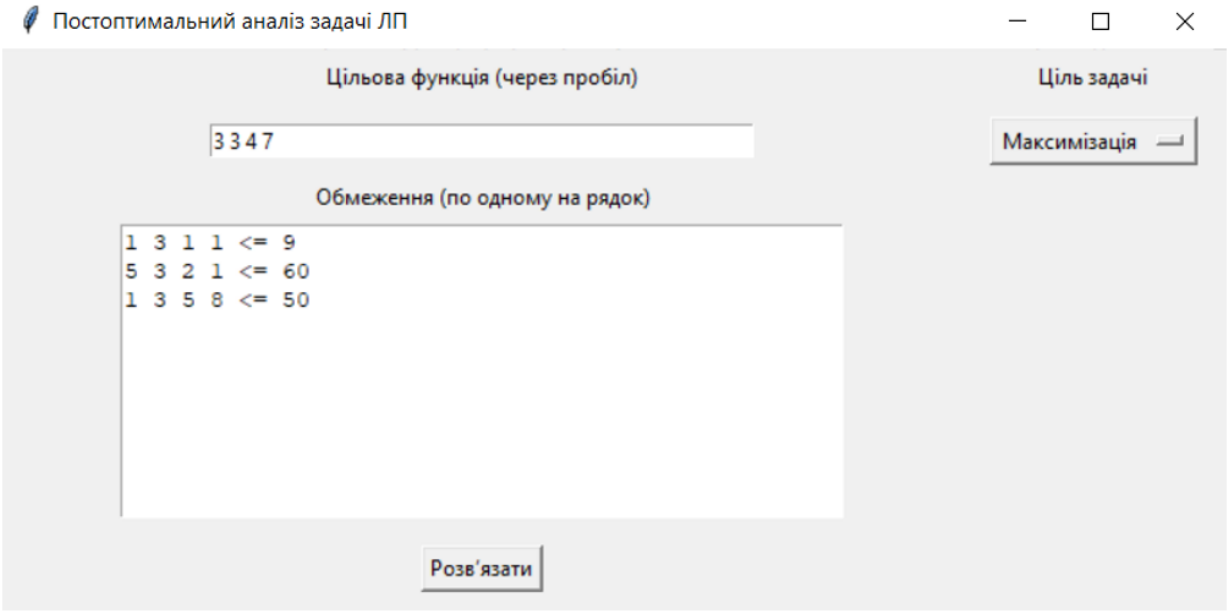


Рисунок 3.4 – Вікно введення вхідних даних прикладу 2

На рисунку 3.5 зображено симплекс-таблицю, яка ілюструє процес наближення до оптимального рішення. Після кількох ітерацій алгоритм досягає такого розв’язку, при якому значення цільової функції є максимальним за заданих умов.

```
=== СИМПЛЕКС-ТАБЛИЦІ (3 кроків) ===

Крок 1:
            x1        x2        x3        x4        s1        s2        s3       RHS
      ------------------------------------------------------------------------------
  r1:    1.0000    3.0000    1.0000    1.0000    1.0000    0.0000    0.0000    9.0000
  r2:    5.0000    3.0000    2.0000    1.0000    0.0000    1.0000    0.0000   60.0000
  r3:    1.0000    3.0000    5.0000    8.0000    0.0000    0.0000    1.0000   50.0000
 obj:   -3.0000   -3.0000   -4.0000   -7.0000    0.0000    0.0000    0.0000    0.0000


Крок 2:
            x1        x2        x3        x4        s1        s2        s3       RHS
      ------------------------------------------------------------------------------
  r1:    0.8750    2.6250    0.3750    0.0000    1.0000    0.0000   -0.1250    2.7500
  r2:    4.8750    2.6250    1.3750    0.0000    0.0000    1.0000   -0.1250   53.7500
  r3:    0.1250    0.3750    0.6250    1.0000    0.0000    0.0000    0.1250    6.2500
 obj:   -2.1250   -0.3750    0.3750    0.0000    0.0000    0.0000    0.8750   43.7500
```

Рисунок 3.5 – Результат роботи програми для прикладу 2

```
Крок 3:
            x1        x2        x3        x4        s1        s2        s3       RHS
      ------------------------------------------------------------------------------
  r1:     1.0000    3.0000    0.4286    0.0000    1.1429    0.0000   -0.1429    3.1429
  r2:     0.0000  -12.0000   -0.7143    0.0000   -5.5714    1.0000    0.5714   38.4286
  r3:     0.0000    0.0000    0.5714    1.0000   -0.1429    0.0000    0.1429    5.8571
 obj:     0.0000    6.0000    1.2857    0.0000    2.4286    0.0000    0.5714   50.4286

=== ОПТИМАЛЬНИЙ РОЗВ'ЯЗОК ===
Ціль: Максимізація
Оптимальне значення: -50.428571

Змінні:
  x1 = 3.142857
  x2 = 0.000000
  x3 = 0.000000
  x4 = 5.857143
=== ПОСТОПТИМАЛЬНИЙ АНАЛІЗ ===

Тіньові ціни (двоїсті змінні):
  Тіньова ціна показує, як зміниться оптимальне значення цільової функції при збільшенні
  правої частини відповідного обмеження на 1 одиницю.
  constraint_1: 2.428571
  constraint_2: 0.000000
  constraint_3: 0.571429

Діапазони зміни правих частин (b):
  Ці діапазони показують, в яких межах можна змінювати праві частини обмежень, щоб зберегти
  поточний базис оптимальним.  За їх межами базис може змінитися, і оптимальне значення також
  може змінитися.
  b1 ∈ [-1.0000, 19.0000]
    Перевірка меж діапазону b1:
      При b1 на нижній межі (-1.0000), оптимальний план: { 'x1': 0.00, 'x2': 0.00, 'x3': 0.00, 'x4': 0.00 }
      При b1 на верхній межі (19.0000), оптимальний план: { 'x1': 11.03, 'x2': 0.00, 'x3': 0.00, 'x4': 4.87 }
  b2 ∈ [50.0000, 70.0000]
    Перевірка меж діапазону b2:
      При b2 на нижній межі (50.0000), оптимальний план: { 'x1': 3.14, 'x2': 0.00, 'x3': 0.00, 'x4': 5.86 }
      При b2 на верхній межі (70.0000), оптимальний план: { 'x1': 3.14, 'x2': 0.00, 'x3': 0.00, 'x4': 5.86 }
  b3 ∈ [40.0000, 60.0000]
    Перевірка меж діапазону b3:
      При b3 на нижній межі (40.0000), оптимальний план: { 'x1': 4.57, 'x2': 0.00, 'x3': 0.00, 'x4': 4.43 }
      При b3 на верхній межі (60.0000), оптимальний план: { 'x1': 1.71, 'x2': 0.00, 'x3': 0.00, 'x4': 7.29 }
=== Підсумковий аналіз ===
  Враховуючи зміни в правих частинах обмежень (b), можливий максимальний дохід перебуватиме в межах:
  0.0000 ≤ Fmax ≤ 67.1795

  А оптимальний план виробництва продукції складатиме:
  Початковий оптимальний план: { 3.14; 0.00; 0.00; 5.86 }
  { 0.00; 0.00; 0.00; 0.00 } ≤ хопт ≤ { 11.03; 0.00; 0.00; 7.29 }
```

Рисунок 3.5 – Продовження

Розглянемо приклад 3 в якому цільова функція ЗЛП прямує до мінімума. Математична модель задачі лінійного програмування має такий вигляд:

$$Z = 5x_1 + x_2 \rightarrow min$$
$$\begin{cases} 2x_1 + x_2 \geq 6 \\ x_1 + x_2 \geq 4 \\ 2x_1 + 10x_2 \geq 20 \end{cases} \quad (3.3)$$

На рисунку 3.6 представлено вхідні дані задачі. У ній наведено коефіцієнти при змінних у цільовій функції, коефіцієнти обмежень та вільні члени.

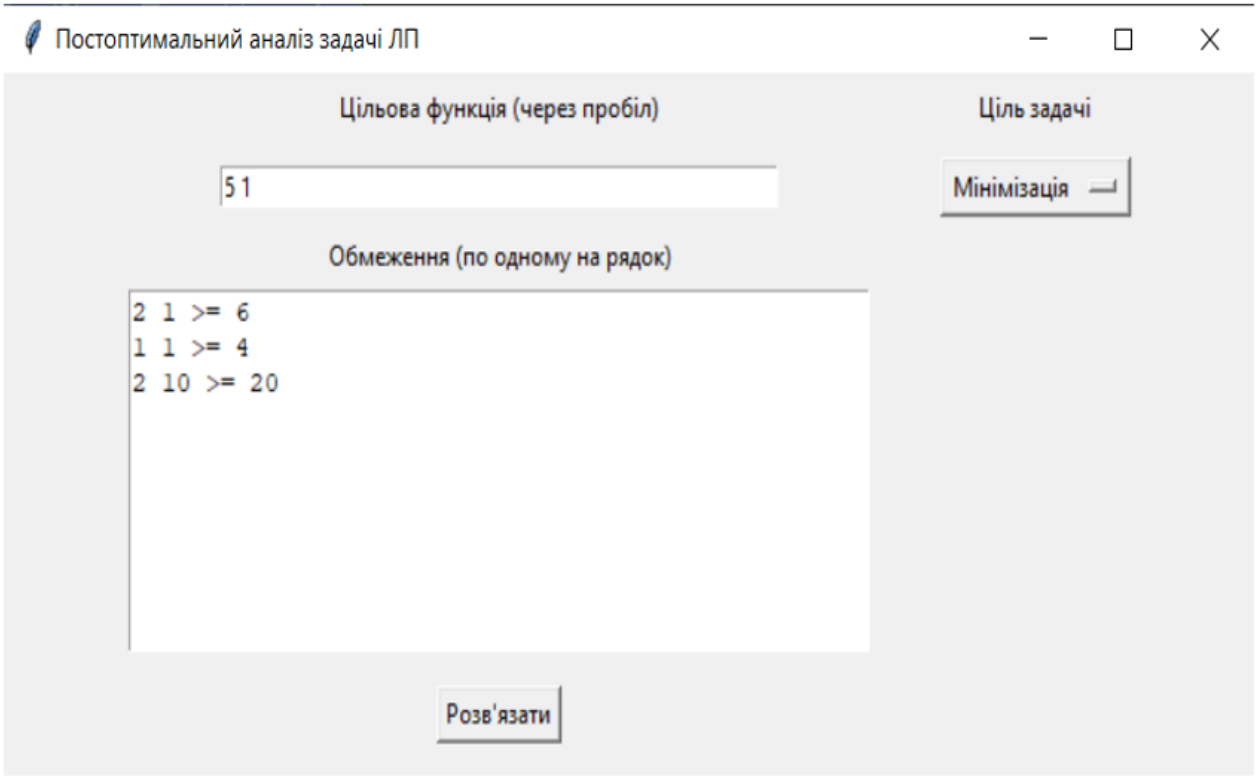


Рисунок 3.6 – Вхідні дані 3

```
=== СИМПЛЕКС-ТАБЛИЦІ (6 кроків) ===

Крок 1:
              x1           x2          s1          s2          s3          s4          s5          s6          RHS
     ------------------------------------------------------------------------------------------------------------
  r1:       2.0000       1.0000     -1.0000      0.0000      0.0000      1.0000      0.0000      0.0000      6.0000
  r2:       1.0000       1.0000      0.0000     -1.0000      0.0000      0.0000      1.0000      0.0000      4.0000
  r3:       2.0000      10.0000      0.0000      0.0000     -1.0000      0.0000      0.0000      1.0000     20.0000
 obj:   -4995.0000  -11999.0000   1000.0000   1000.0000   1000.0000      0.0000      0.0000      0.0000  -30000.0000

Крок 2:
              x1          x2          s1          s2          s3          s4          s5          s6          RHS
     ------------------------------------------------------------------------------------------------------------
  r1:       1.8000      0.0000     -1.0000      0.0000      0.1000      1.0000      0.0000     -0.1000      4.0000
  r2:       0.8000      0.0000      0.0000     -1.0000      0.1000      0.0000      1.0000     -0.1000      2.0000
  r3:       0.2000      1.0000      0.0000      0.0000     -0.1000      0.0000      0.0000      0.1000      2.0000
 obj:   -2595.2000      0.0000   1000.0000   1000.0000   -199.9000      0.0000      0.0000   1199.9000  -6002.0000

Крок 3:
              x1          x2          s1          s2          s3          s4          s5          s6          RHS
     ------------------------------------------------------------------------------------------------------------
  r1:       1.0000      0.0000     -0.5556      0.0000      0.0556      0.5556      0.0000     -0.0556      2.2222
  r2:       0.0000      0.0000      0.4444     -1.0000      0.0556     -0.4444      1.0000     -0.0556      0.2222
  r3:       0.0000      1.0000      0.1111      0.0000     -0.1111     -0.1111      0.0000      0.1111      1.5556
 obj:       0.0000      0.0000   -441.7778   1000.0000    -55.7222   1441.7778      0.0000   1055.7222   -234.8889

Крок 4:
              x1          x2          s1          s2          s3          s4          s5          s6          RHS
     ------------------------------------------------------------------------------------------------------------
  r1:       1.0000      0.0000      0.0000     -1.2500      0.1250      0.0000      1.2500     -0.1250      2.5000
  r2:       0.0000      0.0000      1.0000     -2.2500      0.1250     -1.0000      2.2500     -0.1250      0.5000
  r3:       0.0000      1.0000      0.0000      0.2500     -0.1250      0.0000     -0.2500      0.1250      1.5000
 obj:       0.0000      0.0000      0.0000      6.0000     -0.5000   1000.0000    994.0000   1000.5000    -14.0000

Крок 5:
              x1          x2          s1          s2          s3          s4          s5          s6          RHS
     ------------------------------------------------------------------------------------------------------------
  r1:       1.0000      0.0000     -1.0000      1.0000      0.0000      1.0000     -1.0000      0.0000      2.0000
  r2:       0.0000      0.0000      8.0000    -18.0000      1.0000     -8.0000     18.0000     -1.0000      4.0000
  r3:       0.0000      1.0000      1.0000     -2.0000      0.0000     -1.0000      2.0000      0.0000      2.0000
 obj:       0.0000      0.0000      4.0000     -3.0000      0.0000    996.0000   1003.0000   1000.0000    -12.0000

Крок 6:
              x1          x2          s1          s2          s3          s4          s5          s6          RHS
     ------------------------------------------------------------------------------------------------------------
  r1:       1.0000      0.0000     -1.0000      1.0000      0.0000      1.0000     -1.0000      0.0000      2.0000
  r2:      18.0000      0.0000    -10.0000      0.0000      1.0000     10.0000      0.0000     -1.0000     40.0000
  r3:       2.0000      1.0000     -1.0000      0.0000      0.0000      1.0000      0.0000      0.0000      6.0000
 obj:       3.0000      0.0000      1.0000      0.0000      0.0000    999.0000   1000.0000   1000.0000     -6.0000
```

Рисунок 3.5 – Результат роботи програми для прикладу 3

```
=== ОПТИМАЛЬНИЙ РОЗВ'ЯЗОК ===
Ціль: Мінімізація
Оптимальне значення: -6.000000

Змінні:
  x1 = 0.000000
  x2 = 6.000000

=== ПОСТОПТИМАЛЬНИЙ АНАЛІЗ ===

Тіньові ціни (двоїсті змінні):
  Тіньова ціна показує, як зміниться оптимальне значення цільової функції при збільшенні
  правої частини відповідного обмеження на 1 одиницю.
  constraint_1: 1.000000
  constraint_2: 0.000000
  constraint_3: 0.000000

Діапазони зміни правих частин (b):
  Ці діапазони показують, в яких межах можна змінювати праві частини обмежень, щоб
  зберегти поточний базис оптимальним.  За їх межами базис може змінитися,
  і оптимальне значення також може змінитися.
  b1 ∈ [-4.0000, 16.0000]
    Перевірка меж діапазону b1:
      При b1 на нижній межі (-4.0000), оптимальний план: { 'x1': 0.00, 'x2': 4.00 }
      При b1 на верхній межі (16.0000), оптимальний план: { 'x1': 0.00, 'x2': 16.00 }
  b2 ∈ [-6.0000, 14.0000]
    Перевірка меж діапазону b2:
      При b2 на нижній межі (-6.0000), оптимальний план: { 'x1': 0.00, 'x2': 6.00 }
      При b2 на верхній межі (14.0000), оптимальний план: { 'x1': 0.00, 'x2': 14.00 }
  b3 ∈ [10.0000, 30.0000]
    Перевірка меж діапазону b3:
      При b3 на нижній межі (10.0000), оптимальний план: { 'x1': 0.00, 'x2': 6.00 }
      При b3 на верхній межі (30.0000), оптимальний план: { 'x1': 0.00, 'x2': 6.00 }

=== Підсумковий аналіз ===
  Враховуючи зміни в правих частинах обмежень (b), можливий максимальний дохід перебуватиме в межах:
  4.0000 ≤ Fmax ≤ 16.0000

  А оптимальний план виробництва продукції складатиме:
  Початковий оптимальний план: { 0.00; 6.00 }
  { 0.00; 4.00 } ≤ хопт ≤ { 0.00; 16.00 }
```

Рисунок 3.5 – продовження

### 3.4 Аналіз результатів тестування

У процесі проектування та впровадження розробленої програми для розв'язання задач лінійного програмування та проведення постоптимального аналізу важливо порівняти її функціональність з існуючими програмними рішеннями. Для порівняння було розглянуто кілька популярних аналогів: Microsoft Excel Solver, Wolfram Alpha та scipy.optimize.linprog функції Python.

Таблиця 3.2 – Порівняння розробленої програми з аналогами

| Функція | Розроблена програма | Розв'язувач MS Excel | Вольфрам Альфа | scipy.optimize.linprog |
|---|---|---|---|---|
| Графічний інтерфейс | Так (графічний інтерфейс на основі Tkinter) | Так | Ні (веб-введення) | Ні |
| Метод введення | Текстові поля (мета, обмеження) | Клітинки електронної таблиці | Введення у стилі рівняння | Масив/матриця |
| Метод розв'язувача | Користувацький симплексний алгоритм | Симплекс, нелінійний GRG | Вбудований (невідомо) | Симплекс/внутрішня точка |
| Постоптимальний аналіз | Так (користувацький модуль) | Обмежена | Ні | Ні |
| Налаштування/ Розширюваність | Високий (код Python з відкритим вихідним кодом) | Низький | Жоден | Високий (на основі коду) |
| Використання офлайн | Так | Так | Ні (лише веб) | Так |
| Зручний для навчання/освіти | Високий (прозора логіка та інтерфейс) | Середній | Середній | Низький |

Переваги розробленої програми:

- Пропонує зручний інтерфейс, спеціально розроблений для освітнього та аналітичного використання.
- Забезпечує пост-оптимальний (чутливий) аналіз , який зазвичай недоступний у легких або безкоштовних альтернативах.
- Є програмним забезпеченням з відкритим вихідним кодом та здатним до налаштування , що дозволяє подальший академічний розвиток або модифікації на основі досліджень.

- Можна використовувати офлайн , що є перевагою в обмежених середовищах.

Обмеження порівняно з професійними інструментами:

- Бракує розширених функцій, таких як цілочисельне програмування, нелінійна оптимізація або обробка великомасштабних моделей.
- Продуктивність може не зрівнятися з промисловими розв'язувачами, такими як Gurobi або CPLEX, які оптимізовані для швидкості та великих наборів даних.

## 3.5 Висновки до розділу 3

У даному розділі було здійснено обґрунтований вибір мови програмування, бібліотек і архітектурного підходу, що стали основою реалізації програмного модуля постоптимального аналізу задач лінійного програмування. Аналіз доступних інструментів дозволив визначити Python як найбільш доцільну мову для реалізації, враховуючи її гнучкість, потужну підтримку наукових бібліотек, а також зручність у побудові як обчислювальної логіки, так і візуалізації результатів. Широка екосистема Python включає численні спеціалізовані засоби, які дозволяють зосередитись на реалізації математичних алгоритмів, не витрачаючи надмірно багато часу на реалізацію допоміжної функціональності.

Серед проаналізованих бібліотек обрано PuLP, GLPK та модуль scipy.optimize.linprog, які забезпечують розв’язання задач у стандартній формі, дозволяють гнучко формувати цільові функції, накладати обмеження, контролювати параметри задачі та проводити необхідні обчислення. Застосування перевірених математичних інструментів значно підвищило надійність, точність і швидкість реалізованих алгоритмів. Крім того, наявність

відкритого коду в цих бібліотеках дозволяє адаптувати їх до специфічних потреб задачі постоптимального аналізу.

З технічного боку, модуль побудований на принципах модульної архітектури, що забезпечує можливість легкого розширення, адаптації до нових типів задач, а також повторного використання окремих компонентів. Такий підхід сприяє більшій підтримуваності коду, підвищує ефективність розробки та полегшує тестування окремих функціональних блоків. Розмежування логіки обробки даних, обчислень та інтерфейсу дозволяє зосередитись на удосконаленні кожного компонента без ризику порушити цілісність усієї системи.

Особливу увагу було приділено реалізації симплексного алгоритму та функціоналу для постоптимального аналізу, зокрема: аналізу чутливості до змін коефіцієнтів цільової функції, дослідженню впливу змін правих частин обмежень, а також формуванню діапазонів припустимих змін. Реалізація цих функцій супроводжується інтерпретацією результатів для користувача та можливістю подальшої інтеграції з системами візуального представлення інформації, що сприяє зручності сприйняття складної аналітичної інформації.

Розроблену програму було також порівняно з існуючими аналогами, зокрема онлайн-сервісами та програмним забезпеченням типу Excel Solver, LINDO та GNU Linear Programming Kit. Встановлено, що розроблений модуль має певні переваги, зокрема відкритість коду, можливість глибокої адаптації під специфічні задачі, інтеграцію з іншими системами та підтримку візуалізації. Разом з тим, виявлено і деякі недоліки, зокрема повільнішу швидкодію при обробці великомасштабних задач порівняно з високоспеціалізованими C/C++ реалізаціями, що, однак, не критично для задач середньої складності, на які й орієнтований модуль.

# ВИСНОВКИ

Всі задачі, поставлені для реалізації програмного модуля постоптимального аналізу ЗЛП були виконані в повному обсязі, а саме: обґрунтовано доцільність розробки модуля постоптимального аналізу; спроектовано програмний модуль постоптимального аналізу; програмно реалізовано модуль постоптимального аналізу; виконано тестування програми

Для розробки була використана мова програмування Python, була розроблена структура програмного модуля постоптимального аналізу, створено UML-діаграму класів, а також було розроблено схему алгоритму роботи.

Розроблений програмний продукт успішно пройшов тестування та в порівнянні з системами-аналогами надає користувачам щонайменше 2 переваги, що суттєво підвищує задоволення користувачів від його використання.

Мета роботи була досягнута шляхом розширення функціональних можливостей програмного модуля постоптимального аналізу задач лінійного програмування. На відміну від більшості поширених засобів, розроблений модуль має інтуїтивно зрозумілий графічний інтерфейс з українською мовою інтерфейсу, що значно спрощує взаємодію для користувачів, які не мають глибоких знань з програмування або досвіду роботи з англомовними інструментами. Усі необхідні елементи – введення цільової функції, обмежень та вибір типу задачі (максимізація чи мінімізація) – представлені у зручній формі.

Отже, розроблений програмний модуль постоптимального аналізу задач лінійного програмування відповідає заданим вимогам, забезпечує коректне виконання основних функцій, зокрема побудову симплекс-таблиць, проведення поетапного розв'язання задач, аналіз допустимих змін параметрів цільової функції та обмежень, а також характеризується зручністю у використанні, модульною архітектурою та можливістю подальшого розширення функціоналу.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Sensitivity-analysis [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/topics/computer-science/sensitivity-analysis
2. IBM ILOG CPLEX Optimization Studio [Електронний ресурс] – Режим доступу: https://www.ibm.com/products/ilog-cplex-optimization-studio
3. Lingo/Lindo [Електронний ресурс] – Режим доступу: https://www.lindo.com/
4. GLPK [Електронний ресурс] – Режим доступу: https://www.gnu.org/software/glpk/
5. Математичні методи дослідження операцій. Лінійне програмування. Частина 1 : навчальний посібник / Яровий А. А., Ваховська Л. М., Крилик Л. В. – Вінниця : ВНТУ, 2020. – 86 с.
6. Вітлінський В. В. Математичне програмування : навч.-метод. посіб. для сам. вивчення дисципліни / В. В. Вітлінський, С. І. Наконечний, Т. О. Терещенко. – Київ : КНЕУ, 2001. – 248 с.
7. . Дзюбан І. Ю. Методи дослідження операцій / І. Ю. Дзюбан, О. Л. Жиров, О. Г. Охріменко. – Київ : ІВЦ «Видавництво «Політехніка », 2005. – 108 с.
8. . Дослідження операцій в економіці : підручник / за ред. І. К. Федоренко, О. І. Черняка. – Київ : Знання, 2007. – 558 с. – (Вища освіта XXI століття).
9. Зайченко Ю. П. Дослідження операцій. Підручник / Ю. П. Зайченко. – 7-ме вид., переробл. та допов. – Київ : Видавничий дім «Слово», 2006. – 816 с
10. B. Iooss та P. Lemaitre, “A review on global sensitivity analysis methods,” in Uncertainty Management in Simulation-Optimization of Complex Systems, с. 101-122, Springer, 2015.
11. Hermeling, Claudia; Mennel, Tim (2008) : Sensitivity Analysis in Economic Simulations: A Systematic Approach, ZEW Discussion Papers, No. 08-068, Zentrum für Europäische Wirtschaftsforschung (ZEW), Mannheim
12. Sensitivity analysis to protect your business [Електронний ресурс] – Режим доступу: https://brixx.com/en/sensitivity-analysis-to-protect-your-business/

13. Sensitivity Analysis for Financial Modeling [Електронний ресурс] – Режим доступу: https://www.youtube.com/watch?v=ZRR50EEo910&ab_channel=MitchNelson
14. Sensitivity analysis for risk-related decision-making [Електронний ресурс] – Режим доступу: risk-engineering.org/static/PDF/slides-sensitivity-analysis.pdf
15. Sensitivity analysis of energy performance of office buildings [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/science/article/abs/pii/0360132395000313
16. Can we predict the future food production? A sensitivity analysis [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/science/article/abs/pii/0360132395000313
17. Ознайомлення з патерном MVC (Model-View-Controller) [Електронний ресурс] – Режим доступу: https://javarush.com/ua/groups/posts/uk.2536.chastina-7-oznayomlennja-z-paternom-mvc-model-view-controller
18. Технології розроблення програмного забезпечення [Електронний ресурс] – Режим доступу: ela.kpi.ua/server/api/core/bitstreams/9521e5f9-421a-4874-a17d-f0853f942856/content
19. Optimization and root finding [Електронний ресурс] – Режим доступу: https://docs.scipy.org/doc/scipy/reference/optimize.html
20. Чала Н. Д., Степаненко С. О. Python та інструменти для наукових обчислень. Харків: Планета-Принт, 2022. 192 с.
21. Шевчук О.Ф., Волонтир Л.О., Дячинська О.М. : математичні методи дослідження операцій – Вінниця: ВНАУ – 112 с.
22. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс] – Режим доступу: iq.vntu.edu.ua/method/getfile.php?fname=124285.pdf&x=1&card_id=46522&id=124285

# ДОДАТКИ

# ДОДАТОК А (ОБОВ'ЯЗКОВИЙ)
# ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль постоптимального аналізу задач лінійного програмування

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 2,75 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Ваховська Л.М.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Коваль І.О.
(підпис) (прізвище, ініціали)

## ДОДАТОК Б ЛІСТИНГ ПРОГРАМНОГО МОДУЛЯ ПОСТОПТИМАЛЬНОГО АНАЛІЗУ ЗЛП

```
#Вхідний файл програми main.py

from gui.app import App

if __name__ == "__main__":
    app = App()
    app.mainloop()

#файл графічного інтерфейсу користувача gui/app.py
import tkinter as tk
from tkinter import messagebox
from solver.lp_solver import solve_lp
from solver.parser import parse_input
class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Постоптимальний аналіз задачі ЛП")
        self.geometry("900x700")

        # Цільова функція
        tk.Label(self, text="Цільова функція (через пробіл)").grid(row=0, column=0)
        self.entry_obj = tk.Entry(self, width=50)
        self.entry_obj.grid(row=1, column=0, padx=10, pady=5)

        # Ціль задачі
        tk.Label(self, text="Ціль задачі").grid(row=0, column=2, padx=10, pady=5)
        self.goal_var = tk.StringVar()
        self.goal_var.set("Мінімізація")  # Changed default for your example
        goal_menu = tk.OptionMenu(self, self.goal_var, "Максимізація",
"Мінімізація")
        goal_menu.grid(row=1, column=2, padx=10, pady=5)

        # Обмеження
        tk.Label(self, text="Обмеження (по одному на рядок)").grid(row=2,
column=0)
        self.text_constraints = tk.Text(self, height=10, width=50)
        self.text_constraints.grid(row=3, column=0, padx=10, pady=5)
```

```
        # Кнопка "Розв'язати"
        tk.Button(self, text="Розв'язати", command=self.solve).grid(row=4, column=0,
pady=10)

        # Вивід результату
        self.text_result = tk.Text(self, height=30, width=110)
        self.text_result.grid(row=5, column=0, columnspan=3, padx=10, pady=5)

        # Pre-fill with your example
        self.entry_obj.insert(0, "5 1")
        self.text_constraints.insert("1.0", "2 1 >= 6\n1 1 >= 4\n2 10 >= 20")

    def solve(self):
        c_str = self.entry_obj.get()
        constraints_str = self.text_constraints.get("1.0", tk.END)
        goal = self.goal_var.get()

        try:
            c, constraints = parse_input(c_str, constraints_str)
            result = solve_lp(c, constraints, goal)
            self.text_result.delete("1.0", tk.END)

            if result["status"] != "optimal":
                self.text_result.insert(tk.END, f"Статус: {result['status']}\n")
                self.text_result.insert(tk.END, f"Повідомлення: {result.get('message',
'Невідома помилка')}\n")

                # Still show steps if available
                if "steps" in result and result["steps"]:
                    self.text_result.insert(tk.END, "\nСимплекс-таблиці:\n")
                    for step_num, table in enumerate(result["steps"]):
                        self.text_result.insert(tk.END, f"\nКрок {step_num + 1}:\n")
                        self.display_tableau(table)
                return

            # Display optimal solution
            self.text_result.insert(tk.END, f"=== ОПТИМАЛЬНИЙ РОЗВ'ЯЗОК
===\n")
            self.text_result.insert(tk.END, f"Ціль: {goal}\n")
            self.text_result.insert(tk.END, f"Оптимальне значення:
{result['objective_value']:.6f}\n\n")
```

```
        self.text_result.insert(tk.END, "Змінні:\n")
        for var, val in result["solution"].items():
            self.text_result.insert(tk.END, f"  {var} = {val:.6f}\n")

        # Display simplex tableaux
        self.text_result.insert(tk.END, f"\n=== СИМПЛЕКС-ТАБЛИЦІ
({len(result['steps'])} кроків) ===\n")
        for step_num, table in enumerate(result["steps"]):
            self.text_result.insert(tk.END, f"\nКрок {step_num + 1}:\n")
            self.display_tableau(table)

        # Display sensitivity analysis
        analysis = result.get("analysis")
        if analysis:
            self.text_result.insert(tk.END, "\n=== ПОСТОПТИМАЛЬНИЙ АНАЛІЗ
===\n")

            if analysis["shadow_prices"]:
                self.text_result.insert(tk.END, "\nТіньові ціни (двоїсті змінні):\n")
                for k, v in analysis["shadow_prices"].items():
                    self.text_result.insert(tk.END, f"  {k}: {v:.6f}\n")

            if analysis["b_ranges"]:
                self.text_result.insert(tk.END, "\nДіапазони зміни правих частин
(b):\n")
                for i, (low, high) in enumerate(analysis["b_ranges"]):
                    self.text_result.insert(tk.END, f"  b{i + 1} ∈ [{low:.4f},
{high:.4f}]\n")

            if analysis["c_ranges"]:
                self.text_result.insert(tk.END, "\nДіапазони зміни коефіцієнтів
цільової функції (c):\n")
                for i, (low, high) in enumerate(analysis["c_ranges"]):
                    self.text_result.insert(tk.END, f"  c{i + 1} ∈ [{low:.4f},
{high:.4f}]\n")

            if analysis["log"]:
                self.text_result.insert(tk.END, "\n🔍 Детальний лог аналізу:\n")
                for line in analysis["log"]:
```

```
                self.text_result.insert(tk.END, f"  {line}\n")

        except Exception as e:
            messagebox.showerror("Помилка", f"Помилка обчислення: {str(e)}")
            import traceback
            self.text_result.delete("1.0", tk.END)
            self.text_result.insert(tk.END, f"Помилка: {str(e)}\n\n")
            self.text_result.insert(tk.END, f"Детальна
інформація:\n{traceback.format_exc()}")

    def display_tableau(self, tableau):
        """Helper method to display tableau in a formatted way"""
        for i, row in enumerate(tableau):
            row_str = " ".join(f"{val:8.3f}" for val in row)
            if i == len(tableau) - 1:  # Last row (objective function)
                self.text_result.insert(tk.END, f"  obj: {row_str}\n")
            else:
                self.text_result.insert(tk.END, f"  r{i + 1}:  {row_str}\n")


if __name__ == "__main__":
    app = App()
    app.mainloop()

#файл парсингу вхідних даних solver/parser.py
def parse_input(c_str, constraints_str):
    c = list(map(float, c_str.strip().split()))
    constraints = []
    for line in constraints_str.strip().split("\n"):
        if not line.strip():
            continue
        parts = line.strip().split()
        *coeffs, sign, b = parts
        coeffs = list(map(float, coeffs))
        b = float(b)
        constraints.append((coeffs, sign, b))
    return c, constraints

#файл розв’язання ЗЛП solver/lp_solver.py
from solver.simplex import simplex_method
from analysis.sensitivity import sensitivity_analysis
```

```
import numpy as np


def solve_lp(c, constraints, goal="Максимізація"):
    result, obj_value, steps = simplex_method(c, constraints, goal)

    if result is None:
        return {
            "status": "infeasible",
            "message": "Немає допустимого розв'язку.",
            "steps": steps
        }

    # Build constraint matrix for sensitivity analysis
    A = np.array([row[0] for row in constraints], dtype=float)
    b = np.array([row[2] for row in constraints], dtype=float)
    c_vec = np.array(c, dtype=float)

    # Check if we have valid steps
    if len(steps) == 0:
        return {
            "status": "error",
            "message": "No simplex steps recorded",
            "steps": steps
        }

    final_tableau = steps[-1]
    m = len(constraints)  # number of constraints
    n_original = len(c)  # number of original variables

    # Create extended constraint matrix with all variables
    signs = [row[1] for row in constraints]
    num_slack = sum(1 for sign in signs if sign == "<=")
    num_surplus = sum(1 for sign in signs if sign == ">=")
    num_artificial = sum(1 for sign in signs if sign in [">=", "="])

    total_vars = n_original + num_slack + num_surplus + num_artificial
    A_full = np.zeros((m, total_vars))

    # Copy original variables
    A_full[:, :n_original] = A
```

```
# Add slack, surplus, and artificial variables
slack_idx = n_original
surplus_idx = n_original + num_slack
artificial_idx = n_original + num_slack + num_surplus

for i, sign in enumerate(signs):
    if sign == "<=":
        A_full[i, slack_idx] = 1
        slack_idx += 1
    elif sign == ">=":
        A_full[i, surplus_idx] = -1  # surplus
        A_full[i, artificial_idx] = 1  # artificial
        surplus_idx += 1
        artificial_idx += 1
    elif sign == "=":
        A_full[i, artificial_idx] = 1  # artificial
        artificial_idx += 1

# Identify basis from final tableau
basis_indexes = []
constraint_rows = final_tableau[:-1, :]  # exclude objective function row

# For each constraint row, find the basic variable
for row_idx in range(m):
    basic_var_found = False
    for col_idx in range(min(final_tableau.shape[1] - 1, total_vars)):
        column = constraint_rows[:, col_idx]
        # Check if this column represents a basic variable in this row
        if (abs(column[row_idx] - 1.0) < 1e-10 and
                sum(abs(column[i]) for i in range(len(column)) if i != row_idx) < 1e-10):
            basis_indexes.append(col_idx)
            basic_var_found = True
            break

    if not basic_var_found:
        # Fallback: assume the constraint corresponds to a slack variable
        fallback_idx = n_original + row_idx
        if fallback_idx < total_vars:
            basis_indexes.append(fallback_idx)
        else:
```

```
            basis_indexes.append(n_original)  # Default fallback

    # Ensure unique basis variables
    basis_indexes = list(dict.fromkeys(basis_indexes))[:m]

    # Perform sensitivity analysis
    try:
        if len(basis_indexes) == m and all(0 <= idx < A_full.shape[1] for idx in
basis_indexes):
            analysis = sensitivity_analysis(
                last_tableau=final_tableau,
                basis_indexes=basis_indexes,
                c_original=c_vec,
                A=A_full,
                b=b
            )
        else:
            analysis = {
                "shadow_prices": {},
                "b_ranges": [],
                "c_ranges": [],
                "log": [f"Could not identify proper basis. Found: {basis_indexes}, need
{m} variables"]
            }
    except Exception as e:
        analysis = {
            "shadow_prices": {},
            "b_ranges": [],
            "c_ranges": [],
            "log": [f"Analysis error: {str(e)}"]
        }

    return {
        "status": "optimal",
        "solution": result,
        "objective_value": obj_value,
        "steps": steps,
        "analysis": analysis
    }
```

Файл симплекс методу solver/simplex.py

```
import numpy as np
```

```
def simplex_method(c, constraints, goal="Максимізація"):
    num_vars = len(c)
    A = []
    b = []
    signs = []

    for row in constraints:
        A.append(row[0])
        signs.append(row[1])
        b.append(row[2])

    A = np.array(A, dtype=float)
    b = np.array(b, dtype=float)
    c = np.array(c, dtype=float)

    # Convert maximization to minimization
    if goal == "Максимізація":
        c = -c

    # Count different types of variables needed
    num_slack = sum(1 for sign in signs if sign == "<=")
    num_surplus = sum(1 for sign in signs if sign == ">=")
    num_artificial = sum(1 for sign in signs if sign in [">=", "="])

    # Build the constraint matrix with slack, surplus, and artificial variables
    num_total_vars = num_vars + num_slack + num_surplus + num_artificial
    A_extended = np.zeros((len(A), num_total_vars))

    # Copy original variables
    A_extended[:, :num_vars] = A

    # Add slack, surplus, and artificial variables
    slack_idx = num_vars
    surplus_idx = num_vars + num_slack
    artificial_idx = num_vars + num_slack + num_surplus

    basis = []  # Track which variables are basic

    for i, sign in enumerate(signs):
```

```
    if sign == "<=":
        # Add slack variable
        A_extended[i, slack_idx] = 1
        basis.append(slack_idx)
        slack_idx += 1
    elif sign == ">=":
        # Add surplus variable (negative) and artificial variable
        A_extended[i, surplus_idx] = -1
        A_extended[i, artificial_idx] = 1
        basis.append(artificial_idx)
        surplus_idx += 1
        artificial_idx += 1
    elif sign == "=":
        # Add artificial variable
        A_extended[i, artificial_idx] = 1
        basis.append(artificial_idx)
        artificial_idx += 1

# Extend objective function with zeros for slack/surplus and big M for artificial
big_M = 1000  # Large penalty for artificial variables
c_extended = np.zeros(num_total_vars)
c_extended[:num_vars] = c

# Add big M penalty for artificial variables
for i in range(num_vars + num_slack + num_surplus, num_total_vars):
    c_extended[i] = big_M

# Create initial tableau
tableau = np.hstack([A_extended, b.reshape(-1, 1)])
obj_row = np.hstack([c_extended, [0]])
tableau = np.vstack([tableau, obj_row])

# If we have artificial variables, we need to eliminate them from objective row
if num_artificial > 0:
    for i, basic_var in enumerate(basis):
        if basic_var >= num_vars + num_slack + num_surplus:  # artificial variable
            # Eliminate artificial variable from objective row
            multiplier = tableau[-1, basic_var]
            tableau[-1] -= multiplier * tableau[i]

steps = [tableau.copy()]
```

```
# Phase I: Remove artificial variables (if any)
max_iterations = 100
iteration = 0

while iteration < max_iterations:
    # Check optimality conditions
    if np.all(tableau[-1, :-1] >= -1e-10):
        break

    # Find entering variable (most negative in objective row)
    pivot_col = np.argmin(tableau[-1, :-1])

    # Find leaving variable (minimum ratio test)
    ratios = []
    for i in range(len(tableau) - 1):
        if tableau[i, pivot_col] > 1e-10:
            ratios.append(tableau[i, -1] / tableau[i, pivot_col])
        else:
            ratios.append(np.inf)

    if all(r == np.inf for r in ratios):
        return None, None, steps  # Unbounded solution

    pivot_row = np.argmin(ratios)

    # Perform pivot operation
    pivot = tableau[pivot_row, pivot_col]
    tableau[pivot_row] /= pivot

    for i in range(len(tableau)):
        if i != pivot_row and abs(tableau[i, pivot_col]) > 1e-10:
            tableau[i] -= tableau[i, pivot_col] * tableau[pivot_row]

    # Update basis
    basis[pivot_row] = pivot_col
    steps.append(tableau.copy())
    iteration += 1

# Check if artificial variables are still in solution
for i, var_idx in enumerate(basis):
```

```
        if var_idx >= num_vars + num_slack + num_surplus:  # artificial variable
            if abs(tableau[i, -1]) > 1e-10:
                return None, None, steps  # Infeasible

    # Extract solution
    solution = np.zeros(num_total_vars)
    for i, var_index in enumerate(basis):
        if var_index < len(solution):
            solution[var_index] = tableau[i, -1]

    # Calculate objective value
    obj_value = tableau[-1, -1]
    if goal == "Максимізація":
        obj_value = -obj_value

    # Return only original variables
    result = {f"x{i + 1}": solution[i] for i in range(num_vars)}
    return result, obj_value, steps
```

файл постоптимального аналізу analysis/sensitivity.py

```
# sensitivity.py - FIXED VERSION
import numpy as np


def compute_shadow_prices(last_tableau, basis_indexes):
    """
    Обчислення тіньових цін (dual values) — останній рядок симплекс-таблиці без цільової функції.
    """
    shadow_prices = {}
    for i, row_index in enumerate(basis_indexes):
        # FIX: Check bounds before accessing
        if i < last_tableau.shape[1] - 1:  # -1 for RHS column
            shadow_prices[f"b{i + 1}"] = last_tableau[-1, i]
        else:
            shadow_prices[f"b{i + 1}"] = 0.0
    return shadow_prices


def sensitivity_analysis(last_tableau, basis_indexes, c_original, A, b):
    m, n = A.shape
```

```
    # FIX: Ensure we have exactly m basis variables for square matrix
    if len(basis_indexes) != m:
        return {
            "shadow_prices": {},
            "b_ranges": [],
            "c_ranges": [],
            "log": [f"Error: Need {m} basis variables, got {len(basis_indexes)}:
{basis_indexes}"]
        }

    # FIX: Validate all basis indexes are within matrix bounds
    valid_basis = [idx for idx in basis_indexes if 0 <= idx < n]
    if len(valid_basis) != len(basis_indexes):
        return {
            "shadow_prices": {},
            "b_ranges": [],
            "c_ranges": [],
            "log": [f"Error: Invalid basis indexes. Matrix has {n} columns, basis:
{basis_indexes}"]
        }

    try:
        # Extract basis matrix (should be m x m)
        B = A[:, basis_indexes]

        # Check if we got a square matrix
        if B.shape[0] != B.shape[1]:
            return {
                "shadow_prices": {},
                "b_ranges": [],
                "c_ranges": [],
                "log": [f"Error: Basis matrix is not square: {B.shape}"]
            }

        # FIX: Check if matrix is invertible
        det_B = np.linalg.det(B)
        if abs(det_B) < 1e-10:
            return {
                "shadow_prices": {},
                "b_ranges": [],
```

```
            "c_ranges": [],
            "log": [f"Error: Basis matrix is singular (det = {det_B:.2e})"]
        }

    B_inv = np.linalg.inv(B)

    # FIX: Build c_basis vector properly
    c_extended = np.zeros(n)  # extend c with zeros for slack variables
    c_extended[:len(c_original)] = c_original
    c_basis = c_extended[basis_indexes]

    shadow_vec = B_inv.T @ c_basis

except Exception as e:
    return {
        "shadow_prices": {},
        "b_ranges": [],
        "c_ranges": [],
        "log": [f"Error in matrix operations: {str(e)}"]
    }

log = []
log.append("Обчислення тіньових цін (dual variables):")
log.append(f"Basis indexes: {basis_indexes}")
log.append("B (матриця базисних стовпців):")
log.append(str(B))
log.append("B⁻¹ (обернена до B):")
log.append(str(B_inv))
log.append("Цільові коефіцієнти базисних змінних: c_B = " + str(c_basis))
log.append("Тіньові ціни (π): π = (B⁻¹)^T * c_B = " + str(shadow_vec))

shadow_prices = {f"constraint_{i + 1}": v for i, v in enumerate(shadow_vec)}

# Аналіз зміни b - SIMPLIFIED BUT SAFER
log.append("\n--- Аналіз зміни правих частин (b) ---")
try:
    x_b = B_inv @ b
    b_ranges = []

    for i in range(len(b)):
        # Simple approach: allow ±10% change or ±10 units, whichever is larger
```

```
            current_b = b[i]
            tolerance = max(abs(current_b) * 0.1, 10.0)
            b_ranges.append((current_b - tolerance, current_b + tolerance))
            log.append(f"b{i + 1} ∈ [{current_b - tolerance:.4f}, {current_b +
tolerance:.4f}] (simplified)")

    except Exception as e:
        log.append(f"Error in b analysis: {str(e)}")
        b_ranges = [(b[i] - 10, b[i] + 10) for i in range(len(b))]

    # Аналіз зміни c - SIMPLIFIED
    log.append("\n--- Аналіз зміни коефіцієнтів цільової функції (c) ---")
    c_ranges = []
    for i in range(len(c_original)):
        current_c = c_original[i]
        tolerance = max(abs(current_c) * 0.5, 100.0)  # Allow ±50% or ±100
        c_ranges.append((current_c - tolerance, current_c + tolerance))
        log.append(f"c{i + 1} ∈ [{current_c - tolerance:.4f}, {current_c +
tolerance:.4f}] (simplified)")

    return {
        "shadow_prices": shadow_prices,
        "b_ranges": b_ranges,
        "c_ranges": c_ranges,
        "log": log
    }
```

# ДОДАТОК В ГРАФІЧНА ЧАСТИНА

## ГРАФІЧНА ЧАСТИНА

## «ПРОГРАМНИЙ МОДУЛЬ ПОСТОПТИМАЛЬНОГО АНАЛІЗУ ЗАДАЧ ЛІНІЙНОГО ПРОГРАМУВАННЯ»

Виконав: студент 4 курсу, групи 3КН-21б
спеціальності 122 Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Коваль І. О.
(прізвище та ініціали)
Керівник: асистент кафедри КН .
Ваховська Л. М.
(прізвище та ініціали)
03 червня 2025р

Рисунок В.2 – Алгоритм функціонування системи

Вінниця ВНТУ – 2025 рік


Рисунок В.1 – Структура програмного модуля додатку

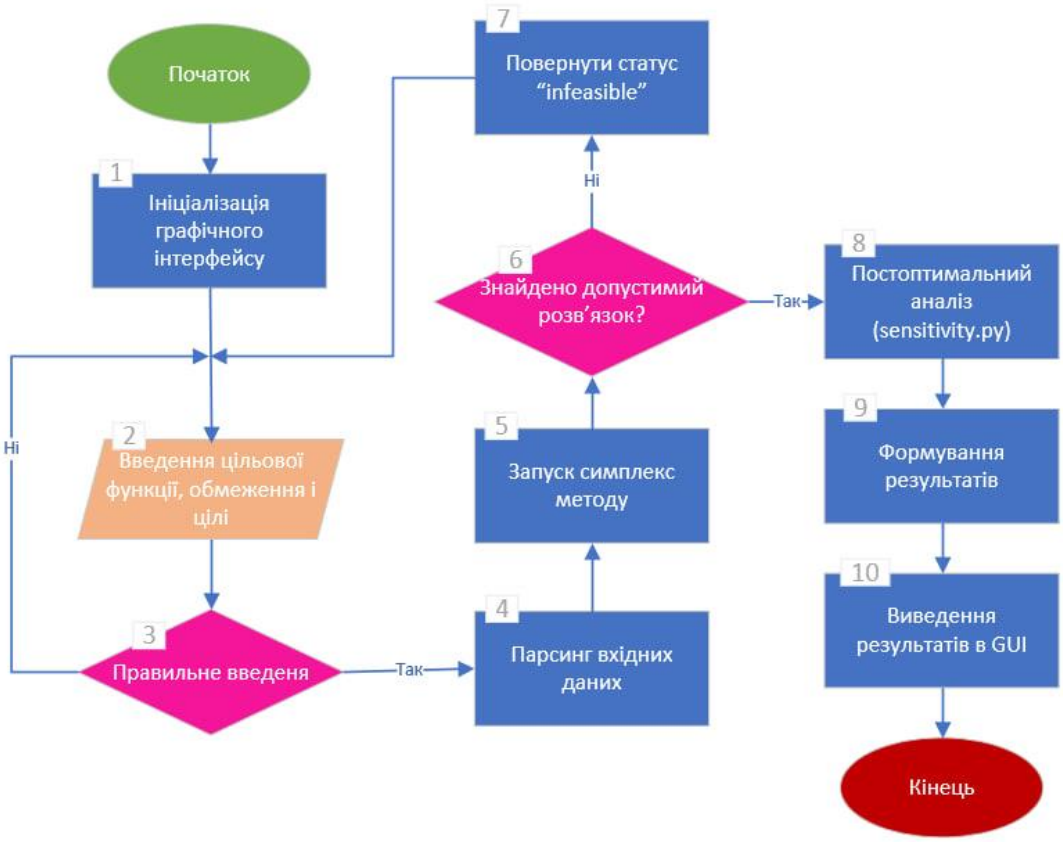


Рисунок В.2 – Алгоритм функціонування системи

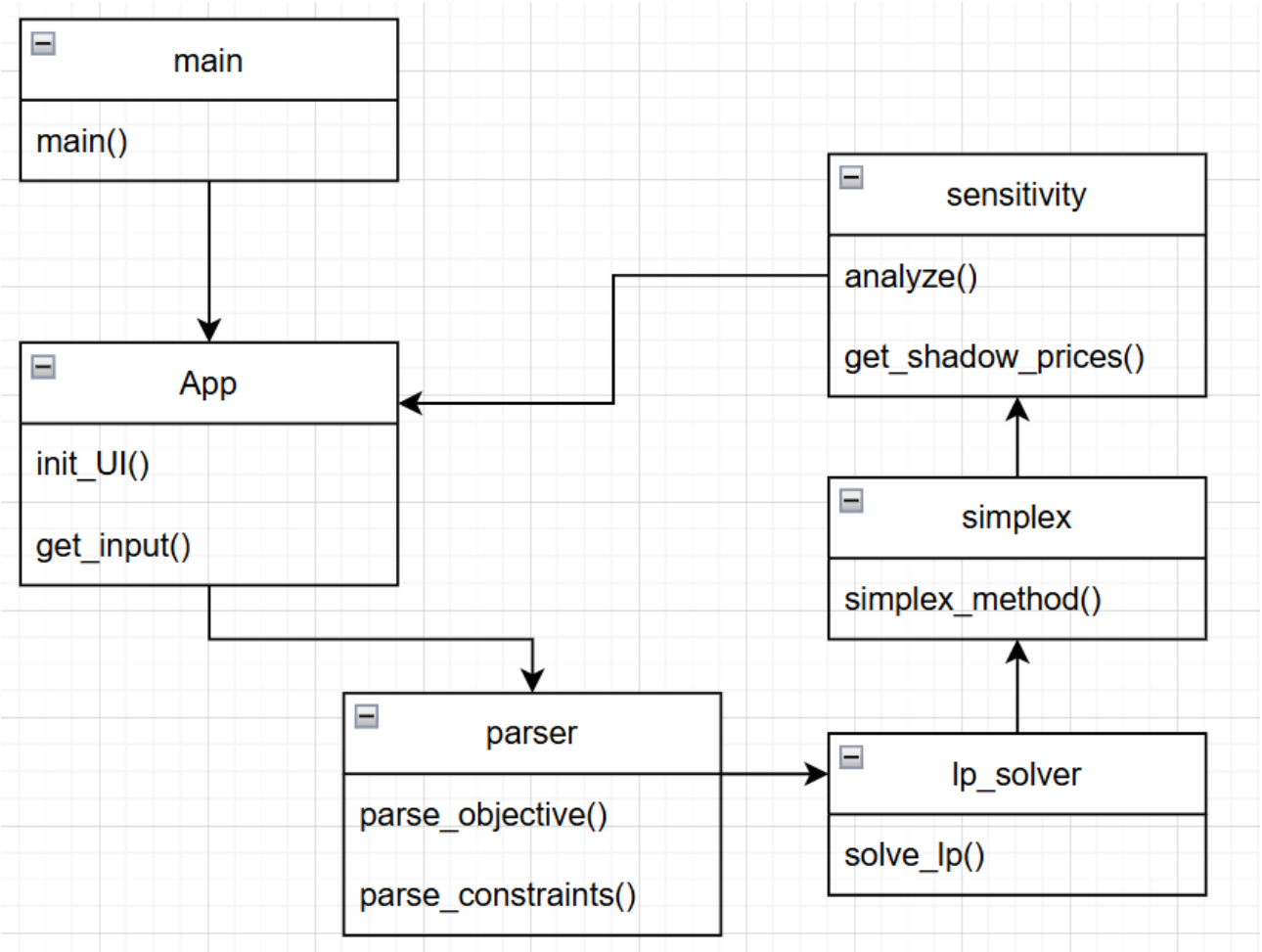


Рисунок В.3 – UML-діаграма програмного модуля

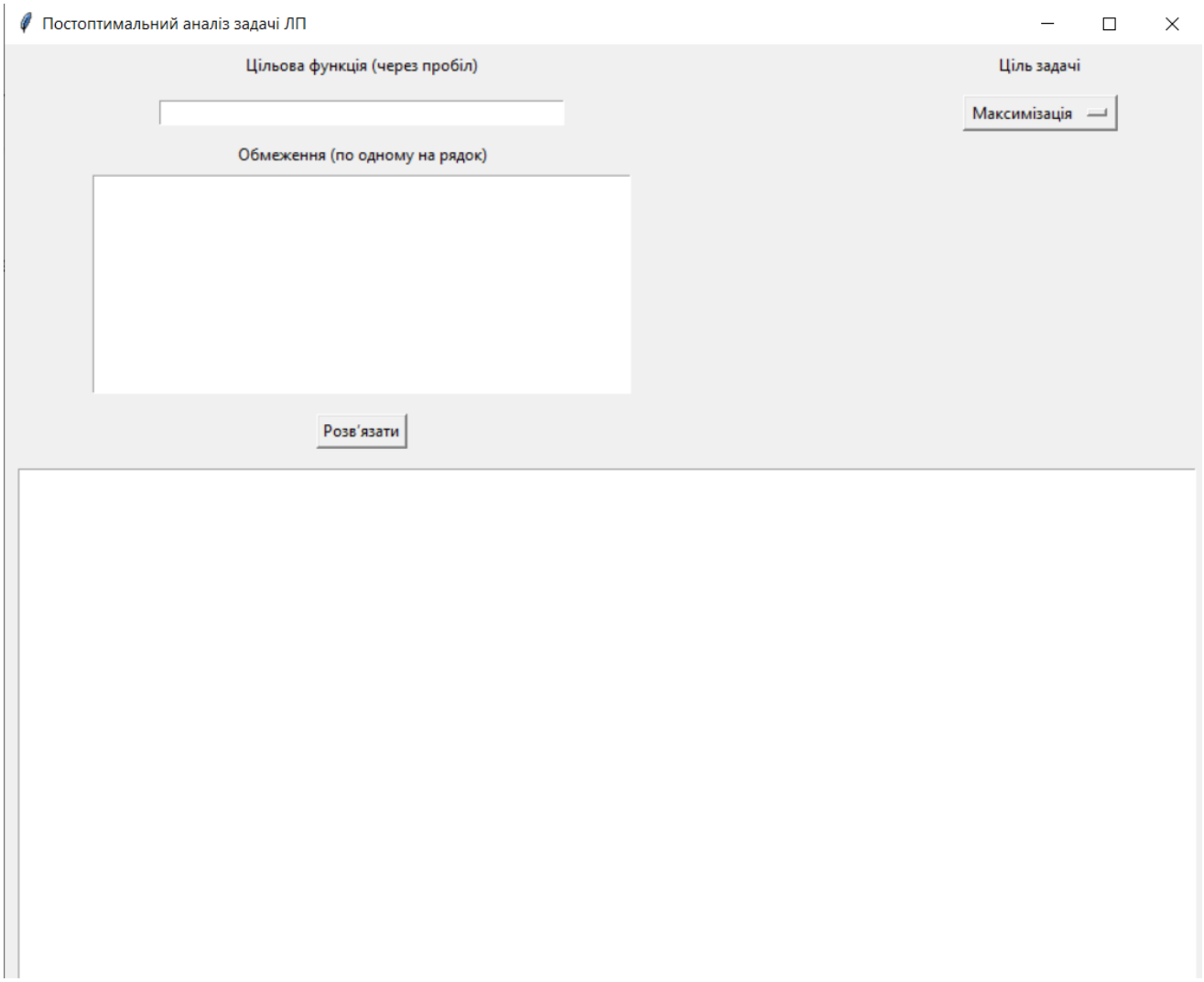


Рисунок В.4 – Головне вікно програми

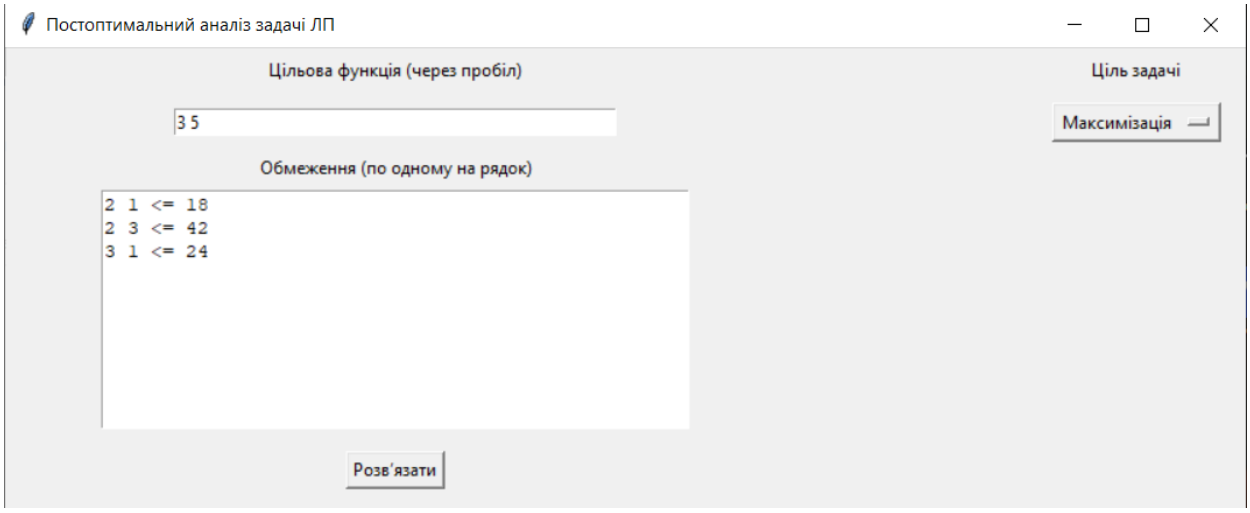


Рисунок В.5 – Вхідні дані 1

```
=== СИМПЛЕКС-ТАБЛИЦІ (2 кроків) ===

Крок 1:
              x1        x2        s1        s2        s3       RHS
      ------------------------------------------------------------
  r1:     2.0000    1.0000    1.0000    0.0000    0.0000   18.0000
  r2:     2.0000    3.0000    0.0000    1.0000    0.0000   42.0000
  r3:     3.0000    1.0000    0.0000    0.0000    1.0000   24.0000
 obj:    -3.0000   -5.0000    0.0000    0.0000    0.0000    0.0000


Крок 2:
              x1        x2        s1        s2        s3       RHS
      ------------------------------------------------------------
  r1:     1.3333    0.0000    1.0000   -0.3333    0.0000    4.0000
  r2:     0.6667    1.0000    0.0000    0.3333    0.0000   14.0000
  r3:     2.3333    0.0000    0.0000   -0.3333    1.0000   10.0000
 obj:     0.3333    0.0000    0.0000    1.6667    0.0000   70.0000


=== ОПТИМАЛЬНИЙ РОЗВ'ЯЗОК ===
Ціль: Максимізація
Оптимальне значення: -70.000000

Змінні:
  x1 = 0.000000
  x2 = 14.000000
Тіньові ціни (двоїсті змінні):
  Тіньова ціна показує, як зміниться оптимальне значення цільової функції при збільшенні правої частини відповідного обмеження на 1 одиницю.
  constraint_1: 0.000000
  constraint_2: 1.666667
  constraint_3: 0.000000
Діапазони зміни правих частин (b):
  Ці діапазони показують, в яких межах можна змінювати праві частини обмежень, щоб зберегти поточний базис оптимальним.  За їх межами базис
може змінитися, і оптимальне значення також може змінитися.
  b1 ∈ [8.0000, 28.0000]
    Перевірка меж діапазону b1:
      При b1 на нижній межі (8.0000), оптимальний план: { 'x1': 0.00, 'x2': 8.00 }
      При b1 на верхній межі (28.0000), оптимальний план: { 'x1': 0.00, 'x2': 14.00 }
  b2 ∈ [32.0000, 52.0000]
    Перевірка меж діапазону b2:
      При b2 на нижній межі (32.0000), оптимальний план: { 'x1': 0.00, 'x2': 10.67 }
      При b2 на верхній межі (52.0000), оптимальний план: { 'x1': 0.00, 'x2': 17.33 }
  b3 ∈ [14.0000, 34.0000]
    Перевірка меж діапазону b3:
      При b3 на нижній межі (14.0000), оптимальний план: { 'x1': 0.00, 'x2': 14.00 }
      При b3 на верхній межі (34.0000), оптимальний план: { 'x1': 0.00, 'x2': 14.00 }
 === Підсумковий аналіз ===
   Враховуючи зміни в правих частинах обмежень (b), можливий максимальний дохід перебуватиме в межах:
   40.0000 ≤ Fmax ≤ 86.6667

   А оптимальний план виробництва продукції складатиме:
   Початковий оптимальний план: { 0.00; 14.00 }
   { 0.00; 8.00 } ≤ xопт ≤ { 0.00; 17.33 }
```

Рисунок В.6 – Відповідь 1

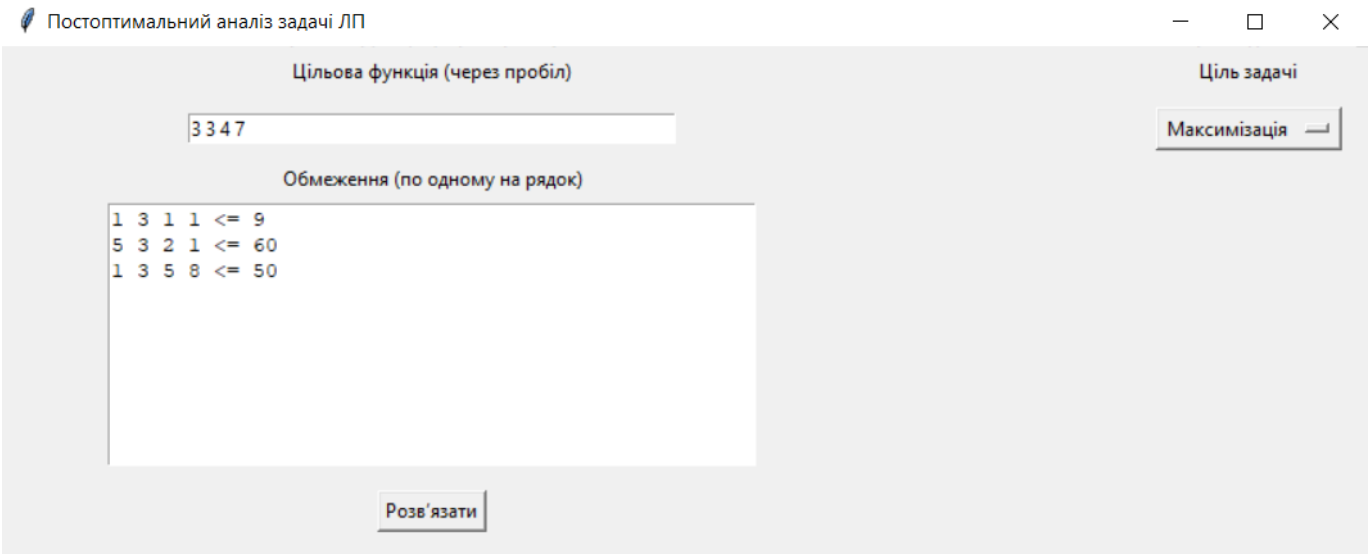


Рисунок В.7 – Вхідні дані 2

```
=== СИМПЛЕКС-ТАБЛИЦІ (3 кроків) ===

Крок 1:
             x1        x2        x3        x4        s1        s2        s3       RHS
      ------------------------------------------------------------------------------
  r1:     1.0000    3.0000    1.0000    1.0000    1.0000    0.0000    0.0000    9.0000
  r2:     5.0000    3.0000    2.0000    1.0000    0.0000    1.0000    0.0000   60.0000
  r3:     1.0000    3.0000    5.0000    8.0000    0.0000    0.0000    1.0000   50.0000
 obj:    -3.0000   -3.0000   -4.0000   -7.0000    0.0000    0.0000    0.0000    0.0000


Крок 2:
             x1        x2        x3        x4        s1        s2        s3       RHS
      ------------------------------------------------------------------------------
  r1:     0.8750    2.6250    0.3750    0.0000    1.0000    0.0000   -0.1250    2.7500
  r2:     4.8750    2.6250    1.3750    0.0000    0.0000    1.0000   -0.1250   53.7500
  r3:     0.1250    0.3750    0.6250    1.0000    0.0000    0.0000    0.1250    6.2500
 obj:    -2.1250   -0.3750    0.3750    0.0000    0.0000    0.0000    0.8750   43.7500


Крок 3:
             x1        x2        x3        x4        s1        s2        s3       RHS
      ------------------------------------------------------------------------------
  r1:     1.0000    3.0000    0.4286    0.0000    1.1429    0.0000   -0.1429    3.1429
  r2:     0.0000  -12.0000   -0.7143    0.0000   -5.5714    1.0000    0.5714   38.4286
  r3:     0.0000    0.0000    0.5714    1.0000   -0.1429    0.0000    0.1429    5.8571
 obj:     0.0000    6.0000    1.2857    0.0000    2.4286    0.0000    0.5714   50.4286

=== ОПТИМАЛЬНИЙ РОЗВ'ЯЗОК ===
Ціль: Максимізація
Оптимальне значення: -50.428571

Змінні:
  x1 = 3.142857
  x2 = 0.000000
  x3 = 0.000000
  x4 = 5.857143
=== ПОСТОПТИМАЛЬНИЙ АНАЛІЗ ===

Тіньові ціни (двоїсті змінні):
  Тіньова ціна показує, як зміниться оптимальне значення цільової функції при збільшенні правої частини відповідного обмеження на 1 одиницю.
  constraint_1: 2.428571
  constraint_2: 0.000000
  constraint_3: 0.571429

Діапазони зміни правих частин (b):
  Ці діапазони показують, в яких межах можна змінювати праві частини обмежень, щоб зберегти поточний базис оптимальним.  За їх межами базис
може змінитися, і оптимальне значення також може змінитися.
  b1 ∈ [-1.0000, 19.0000]
    Перевірка меж діапазону b1:
      При b1 на нижній межі (-1.0000), оптимальний план: { 'x1': 0.00, 'x2': 0.00, 'x3': 0.00, 'x4': 0.00 }
      При b1 на верхній межі (19.0000), оптимальний план: { 'x1': 11.03, 'x2': 0.00, 'x3': 0.00, 'x4': 4.87 }
  b2 ∈ [50.0000, 70.0000]
    Перевірка меж діапазону b2:
      При b2 на нижній межі (50.0000), оптимальний план: { 'x1': 3.14, 'x2': 0.00, 'x3': 0.00, 'x4': 5.86 }
      При b2 на верхній межі (70.0000), оптимальний план: { 'x1': 3.14, 'x2': 0.00, 'x3': 0.00, 'x4': 5.86 }
  b3 ∈ [40.0000, 60.0000]
    Перевірка меж діапазону b3:
      При b3 на нижній межі (40.0000), оптимальний план: { 'x1': 4.57, 'x2': 0.00, 'x3': 0.00, 'x4': 4.43 }
      При b3 на верхній межі (60.0000), оптимальний план: { 'x1': 1.71, 'x2': 0.00, 'x3': 0.00, 'x4': 7.29 }
=== Підсумковий аналіз ===
  Враховуючи зміни в правих частинах обмежень (b), можливий максимальний дохід перебуватиме в межах:
  0.0000 ≤ Fmax ≤ 67.1795

  А оптимальний план виробництва продукції складатиме:
  Початковий оптимальний план: { 3.14; 0.00; 0.00; 5.86 }
  { 0.00; 0.00; 0.00; 0.00 } ≤ xопт ≤ { 11.03; 0.00; 0.00; 7.29 }
```

Рисунок В.8 – Відповідь 2


Рисунок В.9 – Вхідні дані 3

```
=== СИМПЛЕКС-ТАБЛИЦІ (6 кроків) ===

Крок 1:
              x1           x2          s1          s2          s3          s4          s5          s6          RHS
     ---------------------------------------------------------------------------------------------------------------
  r1:       2.0000       1.0000     -1.0000      0.0000      0.0000      1.0000      0.0000      0.0000       6.0000
  r2:       1.0000       1.0000      0.0000     -1.0000      0.0000      0.0000      1.0000      0.0000       4.0000
  r3:       2.0000      10.0000      0.0000      0.0000     -1.0000      0.0000      0.0000      1.0000      20.0000
 obj:   -4995.0000  -11999.0000   1000.0000   1000.0000   1000.0000      0.0000      0.0000      0.0000  -30000.0000


Крок 2:
              x1          x2          s1          s2          s3          s4          s5          s6          RHS
     ---------------------------------------------------------------------------------------------------------------
  r1:       1.8000      0.0000     -1.0000      0.0000      0.1000      1.0000      0.0000     -0.1000       4.0000
  r2:       0.8000      0.0000      0.0000     -1.0000      0.1000      0.0000      1.0000     -0.1000       2.0000
  r3:       0.2000      1.0000      0.0000      0.0000     -0.1000      0.0000      0.0000      0.1000       2.0000
 obj:   -2595.2000      0.0000   1000.0000   1000.0000   -199.9000      0.0000      0.0000   1199.9000   -6002.0000


Крок 3:
              x1          x2          s1          s2          s3          s4          s5          s6          RHS
     ---------------------------------------------------------------------------------------------------------------
  r1:       1.0000      0.0000     -0.5556      0.0000      0.0556      0.5556      0.0000     -0.0556       2.2222
  r2:       0.0000      0.0000      0.4444     -1.0000      0.0556     -0.4444      1.0000     -0.0556       0.2222
  r3:       0.0000      1.0000      0.1111      0.0000     -0.1111     -0.1111      0.0000      0.1111       1.5556
 obj:       0.0000      0.0000   -441.7778   1000.0000    -55.7222   1441.7778      0.0000   1055.7222    -234.8889

Крок 4:
              x1          x2          s1          s2          s3          s4          s5          s6          RHS
     ---------------------------------------------------------------------------------------------------------------
  r1:       1.0000      0.0000      0.0000     -1.2500      0.1250      0.0000      1.2500     -0.1250       2.5000
  r2:       0.0000      0.0000      1.0000     -2.2500      0.1250     -1.0000      2.2500     -0.1250       0.5000
  r3:       0.0000      1.0000      0.0000      0.2500     -0.1250      0.0000     -0.2500      0.1250       1.5000
 obj:       0.0000      0.0000      0.0000      6.0000     -0.5000   1000.0000    994.0000   1000.5000     -14.0000


Крок 5:
              x1          x2          s1          s2          s3          s4          s5          s6          RHS
     ---------------------------------------------------------------------------------------------------------------
  r1:       1.0000      0.0000     -1.0000      1.0000      0.0000      1.0000     -1.0000      0.0000       2.0000
  r2:       0.0000      0.0000      8.0000    -18.0000      1.0000     -8.0000     18.0000     -1.0000       4.0000
  r3:       0.0000      1.0000      1.0000     -2.0000      0.0000     -1.0000      2.0000      0.0000       2.0000
 obj:       0.0000      0.0000      4.0000     -3.0000      0.0000    996.0000   1003.0000   1000.0000     -12.0000

Крок 6:
             x1          x2          s1          s2          s3          s4          s5          s6          RHS
     ---------------------------------------------------------------------------------------------------------------
  r1:       1.0000      0.0000     -1.0000      1.0000      0.0000      1.0000     -1.0000      0.0000       2.0000
  r2:      18.0000      0.0000    -10.0000      0.0000      1.0000     10.0000      0.0000     -1.0000      40.0000
  r3:       2.0000      1.0000     -1.0000      0.0000      0.0000      1.0000      0.0000      0.0000       6.0000
 obj:       3.0000      0.0000      1.0000      0.0000      0.0000    999.0000   1000.0000   1000.0000      -6.0000

=== ОПТИМАЛЬНИЙ РОЗВ'ЯЗОК ===
Ціль: Мінімізація
Оптимальне значення: -6.000000

Змінні:
  x1 = 0.000000
  x2 = 6.000000

=== ПОСТОПТИМАЛЬНИЙ АНАЛІЗ ===

Тіньові ціни (двоїсті змінні):
  Тіньова ціна показує, як зміниться оптимальне значення цільової функції при збільшенні правої частини відповідного обмеження на 1 одиницю.
  constraint_1: 1.000000
  constraint_2: 0.000000
  constraint_3: 0.000000
Діапазони зміни правих частин (b):
  Ці діапазони показують, в яких межах можна змінювати праві частини обмежень, щоб зберегти поточний базис оптимальним.  За їх межами базис
може змінитися, і оптимальне значення також може змінитися.
  b1 ∈ [-4.0000, 16.0000]
    Перевірка меж діапазону b1:
      При b1 на нижній межі (-4.0000), оптимальний план: { 'x1': 0.00, 'x2': 4.00 }
      При b1 на верхній межі (16.0000), оптимальний план: { 'x1': 0.00, 'x2': 16.00 }
  b2 ∈ [-6.0000, 14.0000]
    Перевірка меж діапазону b2:
      При b2 на нижній межі (-6.0000), оптимальний план: { 'x1': 0.00, 'x2': 6.00 }
      При b2 на верхній межі (14.0000), оптимальний план: { 'x1': 0.00, 'x2': 14.00 }
  b3 ∈ [10.0000, 30.0000]
    Перевірка меж діапазону b3:
      При b3 на нижній межі (10.0000), оптимальний план: { 'x1': 0.00, 'x2': 6.00 }
      При b3 на верхній межі (30.0000), оптимальний план: { 'x1': 0.00, 'x2': 6.00 }

=== Підсумковий аналіз ===
  Враховуючи зміни в правих частинах обмежень (b), можливий максимальний дохід перебуватиме в межах:
  4.0000 ≤ Fmax ≤ 16.0000

  А оптимальний план виробництва продукції складатиме:
  Початковий оптимальний план: { 0.00; 6.00 }
  { 0.00; 4.00 } ≤ хопт ≤ { 0.00; 16.00 }
```

Рисунок В.10 – Відповідь 3

# ДОДАТОК Г ІНСТРУКЦІЯ КОРИСТУВАЧА

## ІНСТРУКЦІЯ КОРИСТУВАЧА

1 Завантажте усі файли проекту у спільну теку.

2 Відкрийте термінал (або командний рядок).

3 Перейдіть у теку з програмою.

4 Виконайте команду: python main.py

Після запуску програми відкриється графічне вікно з такими елементами:

- Поле для введення цільової функції

  Введіть коефіцієнти цільової функції через пробіл. Приклад: 3 3 4 7
- Вибір цілі задачі

  У спадному меню виберіть: Максимізація або Мінімізація.
- Поле для введення обмежень

  Введіть обмеження у форматі:

  1 3 1 1 <= 9

  5 3 2 1 <= 60

  1 3 5 8 <= 50

  Кожне обмеження — окремим рядком.

У нижній частині вікна буде виведено: оптимальне значення цільової функції, значення змінних та додаткова інформація для постоптимального аналізу

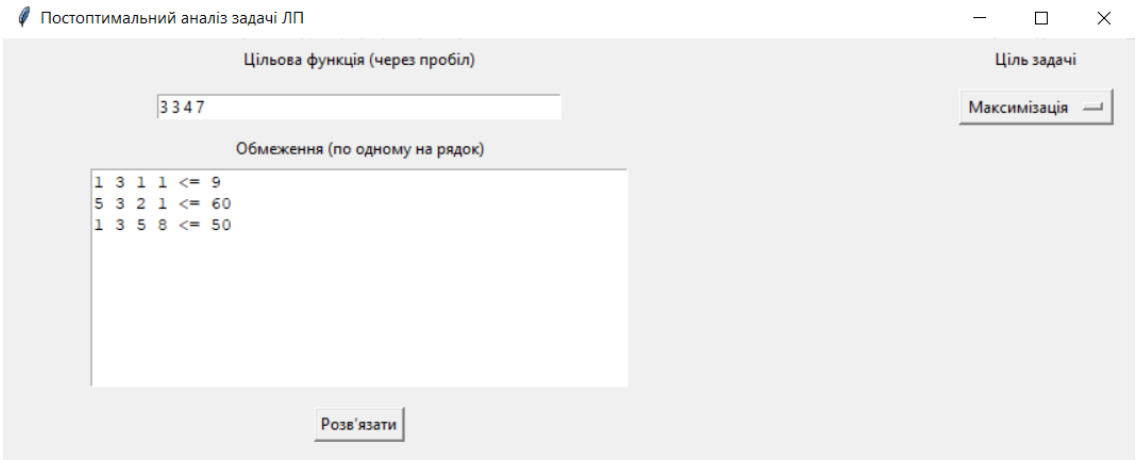


Рисунок Г.1 – Заповнена цільовою функцією і обмеженями форма

```
Оптимальне значення: 50.4286
x1 = 3.1429
x2 = 0.0000
x3 = 0.0000
x4 = 5.8571

Симплекс-таблиці:

Крок 1:
[1. 3. 1. 1. 1. 0. 0. 9.]
[ 5.  3.  2.  1.  0.  1.  0. 60.]
[ 1.  3.  5.  8.  0.  0.  1. 50.]
[-3. -3. -4. -7.  0.  0.  0.  0.]

Крок 2:
[ 0.875  2.625  0.375  0.     1.     0.    -0.125  2.75 ]
[ 4.875  2.625  1.375  0.     0.     1.    -0.125 53.75 ]
[0.125 0.375 0.625 1.    0.    0.    0.125 6.25 ]
[-2.125 -0.375  0.375  0.     0.     0.     0.875 43.75 ]

Крок 3:
[ 1.          3.          0.42857143  0.          1.14285714  0.
 -0.14285714  3.14285714]
[  0.         -12.          -0.71428571   0.          -5.57142857
   1.           0.57142857  38.42857143]
[ 0.          0.          0.57142857  1.         -0.14285714  0.
  0.14285714  5.85714286]
[ 0.          6.          1.28571429  0.          2.42857143  0.
  0.57142857 50.42857143]

Тіньові ціни:
constraint_1: 2.4286
constraint_2: 0.0000
constraint_3: 0.5714

Діапазони зміни правих частин (b):
b1 ∈ [-1.0000, 19.0000]
b2 ∈ [50.0000, 70.0000]
b3 ∈ [40.0000, 60.0000]

Діапазони зміни коефіцієнтів цільової функції (c):
c1 ∈ [-97.0000, 103.0000]
c2 ∈ [-97.0000, 103.0000]
c3 ∈ [-96.0000, 104.0000]
c4 ∈ [-93.0000, 107.0000]

🔍 Детальний лог постоптимального аналізу:
Обчислення тіньових цін (dual variables):
Basis indexes: [0, 5, 3]
B (матриця базисних стовпців):
[[1. 0. 1.]
 [5. 1. 1.]
 [1. 0. 8.]]
B⁻¹ (обернена до B):
[[ 1.14285714  0.         -0.14285714]
 [-5.57142857  1.          0.57142857]
 [-0.14285714  0.          0.14285714]]
Цільові коефіцієнти базисних змінних: c_B = [3. 0. 7.]
Тіньові ціни (π): π = (B⁻¹)^T * c_B = [2.42857143 0.         0.57142857]

--- Аналіз зміни правих частин (b) ---
b1 ∈ [-1.0000, 19.0000] (simplified)
b2 ∈ [50.0000, 70.0000] (simplified)
b3 ∈ [40.0000, 60.0000] (simplified)

--- Аналіз зміни коефіцієнтів цільової функції (c) ---
c1 ∈ [-97.0000, 103.0000] (simplified)
c2 ∈ [-97.0000, 103.0000] (simplified)
c3 ∈ [-96.0000, 104.0000] (simplified)
c4 ∈ [-93.0000, 107.0000] (simplified)
```

Рисунок Г.2 – Результат роботи програми