**

*Міністерство освіти і науки України*

*Національний технічний університет України*

*«Київський політехнічний інститут імені Ігоря Сікорського»*

*Кафедра АСОІУ*

*Спеціальність 126 Інформаційні системи та технології*

**КУРСОВА РОБОТА**

з дисципліни

***“Моделювання систем”***

на тему

«Моделювання процесів, що зводяться до моделі дробно-лінійного програмування в умовах невизначеності»

|  |  |  |
| --- | --- | --- |
| *Виконала*  студентка групи ІС-71  Вознюк О. В.  N зал. кн. ІС-7104  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оцінка) | *Прийняли*  Ст.вик.  НОВІКОВА П.А.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис)  Проф.  СТЕЦЕНКО І.В.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис)  Доц. каф. АСОІУ, к.т.н.  ЖДАНОВА О.Г.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис) |

Київ-2020

Національний технічний університет України «КПІ ім. І.Сікорського»

Кафедра автоматизованих систем обробки інформації та управління

Дисципліна «Моделювання систем»

Спеціальність Інформаційні управляючи системи та технології

Курс 4 Група ІС - 71 Семестр 7

ЗАВДАННЯ

на курсову роботу студента(ки)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Вознюк Олександри Віталіївни\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім'я, по батькові)

1.Тема роботи Моделювання процесів, що зводяться до моделі дробно-лінійного програмування в умовах невизначеності

2. Термін здачі студентом закінченої роботи "25" грудня 2020 р.

3. Вихідні дані до проекту

Варіант №

\_4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Зміст розрахунково-пояснювальної записки (перелік питань, що розробляються)

1. Аналіз існуючих методів вирішення завдання 2. Розробка концептуальної моделі 3. Опис програмного продукту 4. Оцінка адекватності моделі 5. Організація експериментів з моделлю. Висновки.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень)

Графічного матеріалу не має.

6. Дата видачі завдання "29" вересня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів курсового проекту (роботи) | Термін виконання етапів роботи | Примітка |
| 1 | Отримання завдання | 29.09.2020 |  |
|  | Формулювання теми курсової роботи | 08.10.2020 |  |
| 2 | Аналіз можливих методів вирішення поставленого завдання | 29.10.2020 |  |
| 3 | Розробка концептуальної моделі | 05.11.2020 |  |
| 4 | Перший контроль за процесом виконання курсового проекту (роботи), консультація у викладача | 11.11.2020 |  |
| 5 | Опис імітаційної моделі | 25.11.2020 |  |
| 6 | Опис програмної реалізації імітаційної моделі |  |  |
| 7 | Другий контроль за процесом виконання курсового проекту (роботи), консультація у викладача | 29.11.2020 |  |
| 8 | Аналіз та оцінка результатів | 05.12.2020 |  |
| 9 | Оформлення пояснювальної записки | 15.12.2020 |  |
| 10 | Здача пояснювальної записки | 22.12.2020 |  |
| 11 | Захист курсового проекту (роботи) | 25.12.2020 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище, ім'я, по батькові)

"29" вересня 2020 р.

РЕФЕРАТ

Курсова робота: 46 с., 15 рис., 3 табл., 1 додаток, 6 джерел літератури.

Об'єкт дослідження – модель задачі дробно-лінійного програмування в умовах невизначеності.

Мета роботи – дослідження ефективності критеріїв знаходження компромісних розв’язків задачі в недетермінованій постановці та вплив вхідних параметрів на кінцевий результат розв’язання.

Метод дослідження – Було розроблено програмне забезпечення, що дозволяє виконати дослідження критеріїв знаходження компромісних розв’язків задачі в недетермінованій постановці та провести експерименти на вплив вхідних параметрів на кінцевий результат розв’язання.

НЕВИЗНАЧЕНІСТЬ, ДРОБНО-ЛІНІЙНЕ ПРОГРАМУВАННЯ, ЗАДАЧА ЛІНІЙНОГО ПРОГРАМУВАННЯ, КОМПРОМІСНЕ РІШЕННЯ, КОМБІНАТОРНА ОПТИМІЗАЦІЯ, ПЛАНУВАННЯ ЕКСПЕРИМЕНТІВ, МОДЕЛЬ ЗАДАЧІ

ЗМІСТ

[РЕФЕРАТ 4](#_Toc59560778)

[ПОСТАНОВКА ЗАВДАННЯ 7](#_Toc59560779)

[ВСТУП 9](#_Toc59560780)

[1 АНАЛІЗ МОЖЛИВИХ МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ПОСТАВЛЕНОГО ЗАВДАННЯ 10](#_Toc59560781)

[1.1 Теоретичні відомості 10](#_Toc59560782)

[1.2 Опис методів розв’язання задачі 10](#_Toc59560783)

[1.2.1 Критерії оцінки рішень 10](#_Toc59560784)

[1.2.1.1 Критерій A 11](#_Toc59560785)

[1.2.1.2 Критерій B 11](#_Toc59560786)

[1.2.2 Побудова компромісного рішення 11](#_Toc59560787)

[1.2.3 Знаходження компромісного рішення за критеріями A та B 12](#_Toc59560788)

[1.3 Приклади розв’язання задачі 13](#_Toc59560789)

[1.3.1 Приклад розв’язання задачі у детермінованій постановці 13](#_Toc59560790)

[1.3.2 Приклад розв’язання задачі у недетермінованій постановці 14](#_Toc59560791)

[1.4 Огляд програмних засобів 15](#_Toc59560792)

[1.4.1 Google OR-Tools 15](#_Toc59560793)

[1.4.2 LINGO Software 16](#_Toc59560794)

[1.5 Наявні способи генерації випадкових чисел і змінних 17](#_Toc59560795)

[2 РОЗРОБКА КОНЦЕПТУАЛЬНОЇ МОДЕЛІ 18](#_Toc59560796)

[2.1 Вхідні та вихідні змінні 18](#_Toc59560797)

[2.2 Обмеження на можливі зміни величин 18](#_Toc59560798)

[2.3 Цільова функція 18](#_Toc59560799)

[3 ОПИС ПРОГРАМНОГО ПРОДУКТУ 20](#_Toc59560800)

[3.1 Рішення з програмного забезпечення 20](#_Toc59560801)

[3.2 Архітектура програмного забезпечення 21](#_Toc59560802)

[3.3 Опис класів 22](#_Toc59560803)

[3.4 Специфікація функцій 23](#_Toc59560804)

[4 ОЦІНКА АДЕКВАТНОСТІ МОДЕЛІ 24](#_Toc59560805)

[4.1 Приклади розв’язання задачі у ПП 24](#_Toc59560806)

[4.1.1 Задача з пункту «Приклад розв’язання задачі у недетермінованій постановці» 24](#_Toc59560807)

[4.1.2 Задача згенерована випадково 26](#_Toc59560808)

[5. ОРГАНІЗАЦІЯ ЕКСПЕРИМЕНТІВ З МОДЕЛЛЮ 28](#_Toc59560809)

[5.1 План експериментів 28](#_Toc59560810)

[5.1.1 Експеримент типу 1 28](#_Toc59560811)

[5.1.2 Експеримент типу 2 28](#_Toc59560812)

[5.1.3 Експеримент типу 3 28](#_Toc59560813)

[5.2. Аналіз і оцінка результатів 28](#_Toc59560814)

[ВИСНОВКИ 32](#_Toc59560815)

[ПЕРЕЛІК ПОСИЛАНЬ 33](#_Toc59560816)

[ДОДАТОК. ЛІСТИНГ ПРОГРАМИ 34](#_Toc59560817)

ПОСТАНОВКА ЗАВДАННЯ

Недостатність інформації про умови та обмеження, що властиві багатьом системам приводить до необхідності розробляти нові методи прийняття рішень. В роботах [1, 2] були викладені основи конструктивної теорії знаходження компромісного рішення для класу задач комбінаторної оптимізації в умовах невизначенності виду

(1)

де — числа, — *i*-та довільна числовая характеристика допустимого рішення   — множина допустимих рішень.

Під невизначенністю тут розуміється невизначенність значень коефіцієнтів

В [3] підтверджена ефективність цих теоретичних положень на прикладі однопродуктової та багатопродуктової транспортних задач. Результати отримані в [1–3] поширюються також і на задачу дробно- лінійного програмування, де величини и зв’язні нелінійно і формально не належать до класу задач комбінаторної оптимізації.

Задача дробно-лінійного програмування у детермінованій постановці має вигляд:

(2)

(3)

де, , , , — дійсні числа, , - змінні задачи.

Для того, щоб при поясненнях уникнути необхідності розгляду множини різних можливих варіантів, припустимо, що на накладаються такі обмеження, при яких знаменник в (2) строго додатній для всіх допустимих значень , а також, що максимум є кінцевим:

(4)

(5)

Отже, існує R наборів коефіцієнтів можливих значень коефіцієнтів , . Знайти за заданими компромісними критеріями рішення задачі (1)–(2) в умовах сформульованої вище невизначеності. Ціллю є знайти такий компромісний розв’язок, який би задовільняв усі альтернативи не менше ніж на якусь встановлену величину.

ВСТУП

Темою курсової роботи є “Моделювання процесів, що зводяться до моделі дробно-лінійного програмування в умовах невизначеності”.

Дробно-лінійне програмування використовується у випадку коли необхідно максимізувати (мінімізувати) значення відношення деяких функцій. Наприклад, відношення прибутку до витрат.

В реальному житті буває складно вирішити такого типу задачу, якщо існує декілька варіантів значення для одного і того ж самого параметра, тому на практиці визначення найкращого варіанту дій часто проводиться в умовах невизначенності. Наслідки прийнятих рішень залежать від майбутнього розвитку подій, яке може відбуватися за різними сценаріями. В цьому і полягає умова невизначеності – існування множини альтернативних варіантів. Задачі в умовах невизначеності виникають в ситуаціях, коли немає попередньої ймовірносної оцінки можливих майбутніх ситуацій або значень параметрів, які їх характеризують.

Отже, метою даної роботи є дослідження критеріїв знаходження компромісних розв’язків задачі в недетермінованій постановці та вплив вхідних параметрів на кінцевий результат розв’язання.

1 Аналіз можливих методів та засобів вирішення поставленого завдання

Так як в даній роботі важливу роль буде грати так звана задача лінійного програмування (ЗЛП), оскільки задача дробно-лінійного програмування зводиться до ЗЛП, то наведемо деякі основні її положення у наступному підпункті 1.1.

1.1 Теоретичні відомості

Загальною задачею лінійного програмування називається задача, що має на меті визначення макимального (мінімального) значення функції [6]:

При умовах

де – задані постійні величини і

Функція (1.1) називається цільовою функцією (або лінійною формою) задачі (1.1)-(1.4), а умови (1.2)-(1.4) – обмеженнями даної задачі.

1.2 Опис методів розв’язання задачі

1.2.1 Критерії оцінки рішень

Оскільки в умовах невизначеності достатньо складно визначити чіткі критерії ефективності рішень, у статті [4] пропонується декілька компромісних критеріїв оцінки рішення задачі, що розглядається. Для даної роботи були обрані критерії 3 та 4 описані в [4]. Будемо називати їх критерій A та B відповідно.

1.2.1.1 Критерій A

Знайти компромісне рішення , що задовольняє (3), у якого:

де для задачі на мінімум:

А для задачі на максимум:

1.2.1.2 Критерій B

Якщо компромісного рішення, що задовольняє критерію A не існує, то знайти , що задовольняє (3) на якому досягається

де , - відомі експертні вагові коефіцієнти.

1.2.2 Побудова компромісного рішення

Як відомо [5], задача (2)-(3) зводиться до задачі лінійного програмування (ЗЛП) наступним чином.

Введемо нові змінні

Тоді задача (2)-(3) прийме вигляд

де .

По рішенню ЗЛП (1.11)-(1.13) знаходиться оптимальне рішення задачі (2)-(3):  При цьому оптимальне значення функціоналів (2) і (1.11) приймають однакове значення.

1.2.3 Знаходження компромісного рішення за критеріями A та B

Компромісне рішення ( ) за критеріями A та B (якщо за критерієм A рішення не існує) знаходиться за рішенням наступної ЗЛП:

Якщо вихідна задача (2)-(3) є задачею на максимум, то в задачі (1.14)-(1.17) нерівності (1.17) мають вигляд:

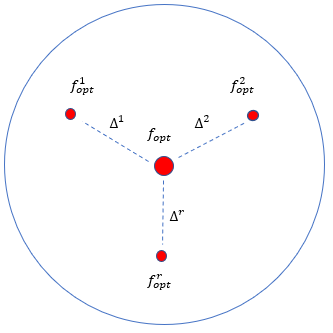


Рисунок 1.1 – Ілюстрація знаходження оптимального розв’язку при умові невизначеності

1.3 Приклади розв’язання задачі

1.3.1 Приклад розв’язання задачі у детермінованій постановці

Умова:

Розв’язання:

Введемо нові змінні

, ,

Тоді задача прийме вигляд

Таким чином ми отримали задачу лінійного програмування. Далі її можна вирішити симплекс-методом і тоді отримаємо , , .

Знайдемо кінцевий розв’язок за допомогою формули :

1.3.2 Приклад розв’язання задачі у недетермінованій постановці

Маємо два набори коефіцієнтів , тоді умова задачі прийме такий вигляд:

Таблиця 1.1 – Приклад розв’язання задачі

|  |  |  |
| --- | --- | --- |
| Умова |  |  |
| Відповідь |  |  |

Введемо нові змінні

, ,

Тоді задача прийме вигляд

Таким чином ми отримали задачу лінійного програмування. Далі її можна вирішити симплекс-методом і тоді отримаємо , , .

Знайдемо кінцевий розв’язок за допомогою формули :

*,*

*,*

*.*

1.4 Огляд програмних засобів

Так як не виявлено програмного забезпечення, що має інструменти для вирішення та дослідження задачі дробно-лінійного програмування в умовах невизначенності в межах даної теми доцільно аналізувати програмні засоби, що можуть бути корисні при вирішенні ЗЛП, тому що ця дія повторюється не один раз впродовж моделювання даної задачі.

1.4.1 Google OR-Tools

OR-Tools - це пакет програмного забезпечення з відкритим кодом для оптимізації, призначений для вирішення найскладніших проблем у світі з маршрутизацією транспортних засобів, потоками, цілочисельним та лінійним програмуванням та програмуванням обмежень.

Змоделювавши свою проблему вибраною мовою програмування, ви можете використовувати будь-який із півтора десятка вирішувачів: комерційні вирішувачі, такі як Gurobi або CPLEX, або вирішувачі з відкритим кодом, такі як SCIP, GLPK або GLOP від ​​Google та CP-SAT.

Лінійний оптимізатор Glop знаходить оптимальне значення лінійної цільової функції, враховуючи набір лінійних нерівностей як обмеження (наприклад, призначення людей на роботу або пошук найкращого розподілу набору ресурсів при мінімізації витрат).

OR-Tools написаний на C ++, але ви також можете використовувати його з Python, Java або C #.

1.4.2 LINGO Software

LINGO - це комплексний інструмент, призначений для швидшого, простішого та ефективнішого побудови та вирішення моделей лінійної, нелінійної (опуклої та неопуклої / глобальної), квадратичної, квадратично обмеженої, напіввизначеної, стохастичної та цілочисельної оптимізації. LINGO пропонує повністю інтегрований пакет, який включає потужну мову для вираження моделей оптимізації, повнофункціональне середовище для побудови та редагування проблем, а також набір швидких вбудованих вирішувачів.

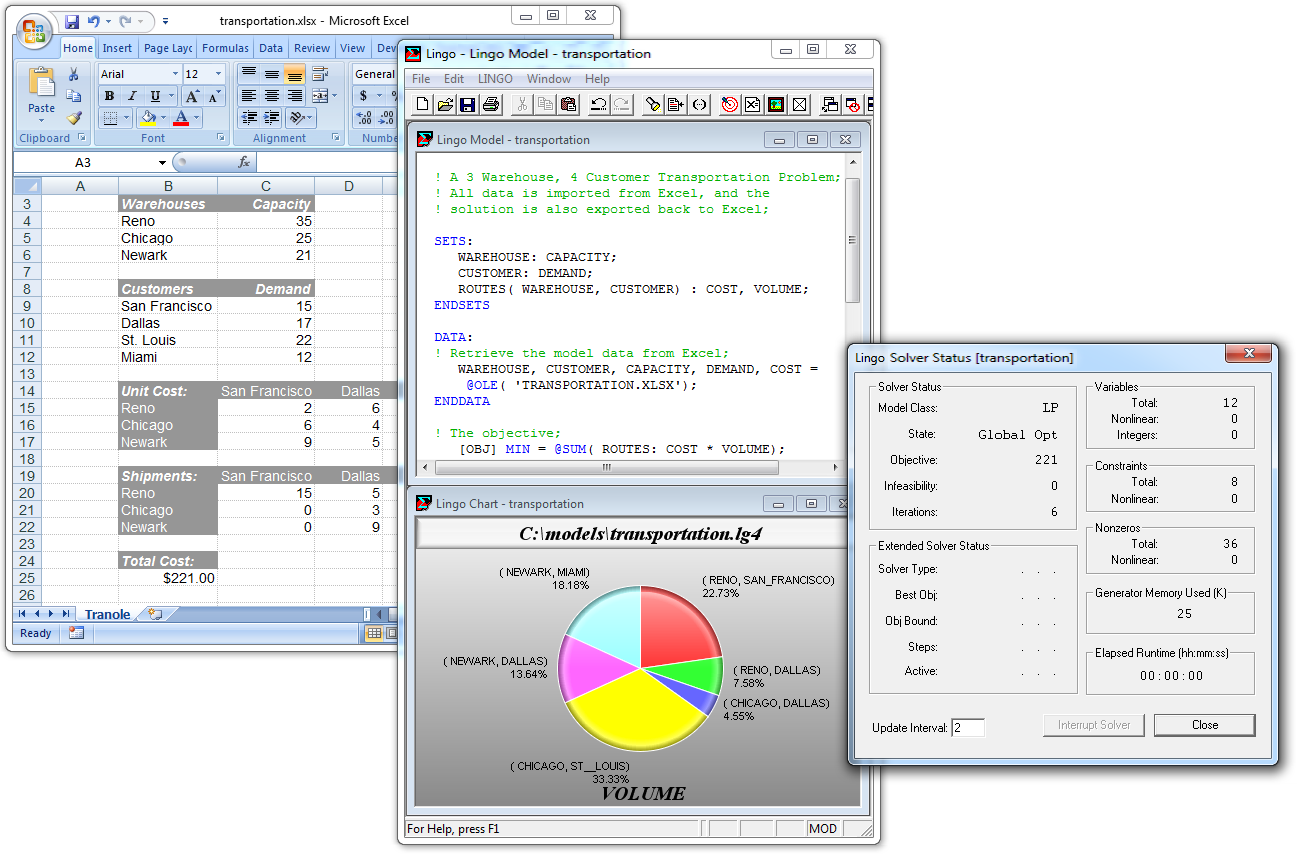


Рисунок 1.2 – Приклад інтерфейсу LINGO Software

1.5 Наявні способи генерації випадкових чисел і змінних

Імітаційні моделі складних систем містять випадкові величини, що мають різні закони розподілу. При побудові алгоритму імітації ці випадкові величини реалізуються генераторами випадкових чисел. Від якості генераторів випадкових чисел, що використовуються, залежить точність результатів імітаційного моделювання.

Відомі наступні способи генерування випадкових величин:

* зберігання у комп’ютері таблиці випадкових чисел і отримання потім з неї даних для імітаційного моделювання;
* використання деякого фізичного пристрою, наприклад електронної лампи, для генерації випадкового шуму;
* застосування рекурсивних формул коли на підставі і-того випадкового числа обчислюється *і+*1-ше випадкове число.

Недоліком першого способу є зберігання великого обсягу інформації та повільна швидкість. Недоліком другого способу - неможливість направленого експерименту з параметрами моделі. Третій спосіб не має недоліків попередніх способів і в теперішній час є найбільш прийнятним.

2 Розробка концептуальної моделі

2.1 Вхідні та вихідні змінні

Вхідними зміними є:

* R наборів коефіцієнтів .
* Набір коефіцієнтів .
* Набір експертних вагів .
* Набір максимально можливих значень дельта

Вихідними зміними є:

* оптимальні значення змінних задачі дробно-лінійного програмування в умовах невизначенності
* .
* птимальне значення допоміжної цільової функції Z.

2.2 Обмеження на можливі зміни величин

Згідно постановки задачі існують обмеження (2)-(5).

Також усі змінні ЗЛП повинні бути додатніми або дорівнювати нулю крім

, - відомі експертні вагові коефіцієнти.

максимальне значення

2.3 Цільова функція

Часткові цільові функції

де, , , — дійсні числа, , - змінні задачи.

Допоміжна цільова функція Z:

де,

3 ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1 Рішення з програмного забезпечення

Було розроблено незалежну бібліотеку “ProblemSolver” для вирішення задачі дробно-лінійного програмування в умовах невизначенності, а також для генерації задач даного типу для подальшого дослідження. У якості інтерфейса для користувача був розроблений веб застосунок MVC, що використовує бібліотеку згадану вище.

Для створення програмного забезпечення була обрана платформа .NET Core зокрема ASP.NET Core з використанням патерну MVC та мова програмування C#.

.NET Core - це еволюція технології Microsoft .NET у модульну, крос-платформну, відкриту та хмарну платформу, яка працює на Windows, Mac, Android, IoT та Linux. Легкий фреймворк .NET Core забезпечує набагато швидшу продуктивність у порівнянні з деякими конкуруючими технологіями, зокрема Node.js та GO.

Переваги, які були для мене головними при виборі ASP.NET Core:

* Razor Pages робить створення кодів сценаріїв для сторінок простіше і ефективніше
* Можливість розробки і запуску в ОС Windows, macOS і Linux
* Відкритий вихідний код і орієнтація на співтовариство
* Інструментарій, що спрощує процес сучасної веб-розробки

Головною перевагою патерну MVC є чітке розділення логіки інтерфейсу і логіки застосування.

Також було вирішено використовувати бібліотеку для вирішення проблем лінійної оптимізації, що є частиною пакету Google OR-Tools, тому що її можна використовувати на мові C# та інтегрувати у застосунок будь-якого типу.

3.2 Архітектура програмного забезпечення

Нижче на рисунку 3.1 наведена схеха архітектури веб застосунку, що використовує розроблену бібліотеку “ProblemSolver” для вирішення задачі дробно-лінійного програмування в умовах невизначеності.

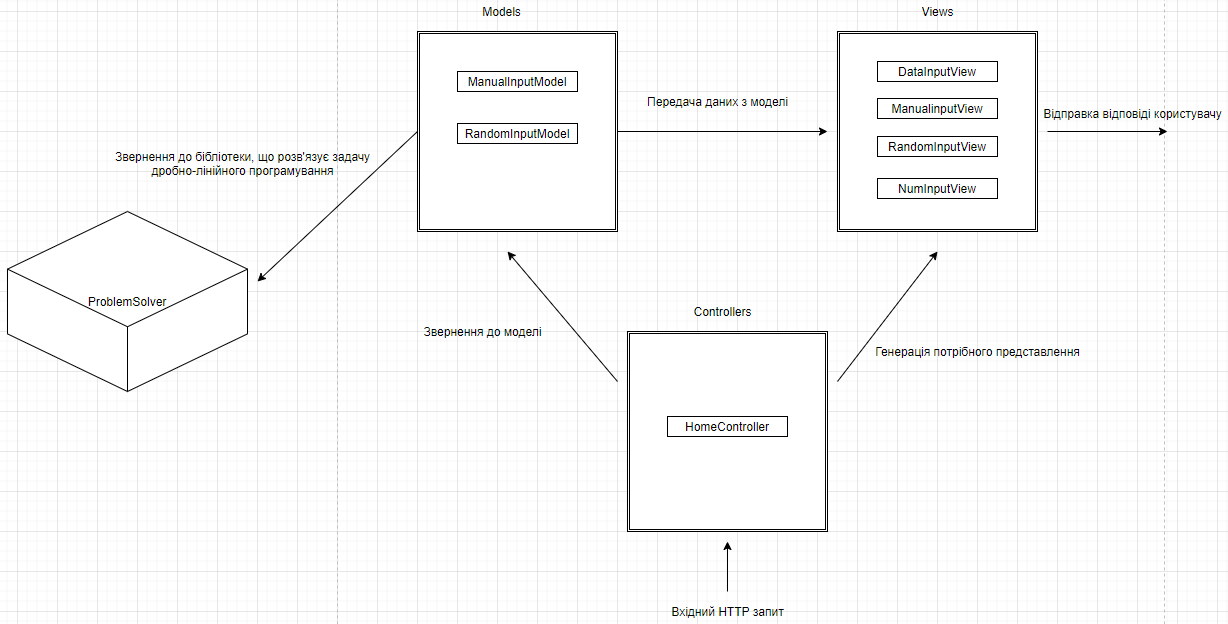


Рисунок 3.1 – Схема архітектури ПП

– Модель (model): описує використовувані в застосунку дані, а також логіку, яка пов'язана безпосередньо з даними

– Представлення (view): відповідають за візуальну частину або призначений для користувача інтерфейс, нерідко html-сторінка, через який користувач взаємодіє з застосуванням

– Контролер (controller): представляє центральний компонент MVC, який забезпечує зв'язок між користувачем та програмою, представленням і сховищем даних. Він містить логіку обробки запиту користувача. Контролер отримує дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді представлення, наповненого даними моделей.

3.3 Опис класів

Нижче в таблиці 3.1 наводиться опис усіх класів розробленої бібліотеки.

Таблиця 3.1 – Опис класів

|  |  |
| --- | --- |
| Назва класу | Опис |
| SolveHelper | Клас, що містить методи для розв’язання задачі дробно-лінійного програмування в умовах невизначенності |
| RandProblemGenerator | Клас, що містить методи для генерування задач дробно-лінійного програмування на основі задланих меж значень параметрів та коефіцієнтів |
| Constraint | Клас, що описує сутність “Обмеження” |
| NoOptimumException | Клас, що описує помилку у випадку коли задача не має розв’язку |
| Y0IsNullException | Клас, що описує помилку у випадку коли y0 = 0 |
| OptDirectionEnum | Перечислення, що представляє набір значень, для опису напрямку оптимізації |
| SymbolEnum | Перечислення, що представляє набір значень, для опису символів “>=”, “=” та “<=” |
| MyEnumExtensions | Клас, що дозволяя розширити функціонал класу SymbolEnum. |

3.4 Специфікація функцій

Нижче в таблиці 3.2 наводиться опис основних функцій розробленої бібліотеки.

Таблиця 3.2 – Опис основних функцій

|  |  |
| --- | --- |
| Функція | Опис |
| SolveProblem(List<List<double>>numerators, List<double>denominator,List<Constraint>\_constraints, List<double> ls, List<double> ws, OptDirectionEnum \_optDirection) | Функція, що розв’язує задачу дробно-лінійного програмування в умовах невизначенності. |
| FindDeltas(List<double>xs,List<List<double>>numerators, List<double> denominator, List<double> fOpts) | Функція, що обчислює дельти. |
| Solve(List<Constraint>\_constraints,List<double>\_function, OptDirectionEnum \_optDirection, int xCount, int zCount) | Функція, що що розв’язує задачу дробно-лінійного програмування. |
| GenerateProblem(List<Tuple<double,double>>coefBounds, List<Tuple<double, double>> bBounds, int constraintCount, string optDirectionString, List<Tuple<double, double>> lsBounds, List<Tuple<double, double>> wsBounds) | Функція, що генерує задачу дробно-лінійного програмування на основі введених обмежень на значення зміни вхідних даних. |

4 Оцінка адекватності моделі

у цьому розділі наводяться скріншоти, що показують приклад розв’язання задачі з розділу 1.3.2 та приклад розв’язання рандомно згенерованої задачі у ПЗ. Порівнявши дані отримані в цьому розділі і в розділі 1.3.2 ми можемо зробити висновок, що модель є правильною і ПП працює коректно.

4.1 Приклади розв’язання задачі у ПП

4.1.1 Задача з пункту «Приклад розв’язання задачі у недетермінованій постановці»

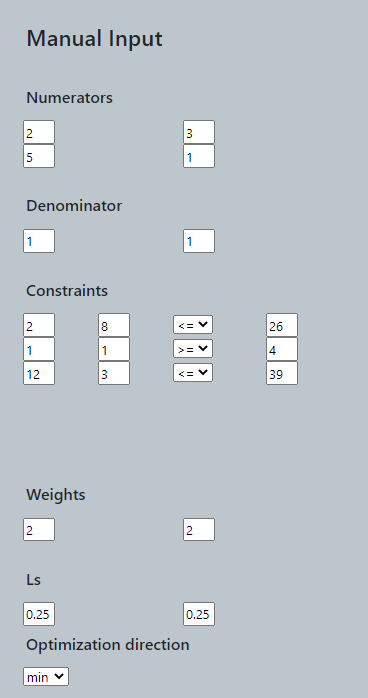


Рисунок 4.1 – Приклад введення вхідних даних задачі

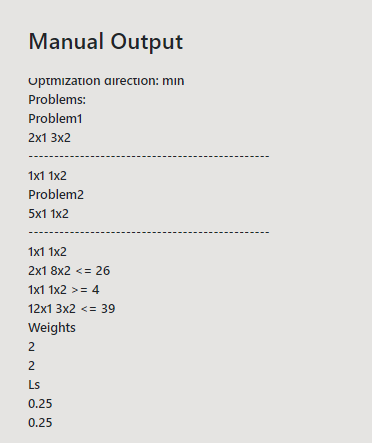


Рисунок 4.2 – Приклад виведення вхідних даних

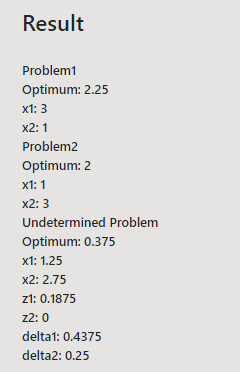


Рисунок 4.3 – Приклад виведення розв’язку

4.1.2 Задача згенерована випадково

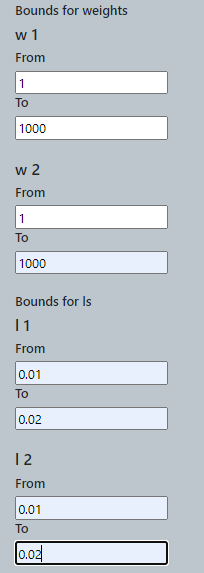
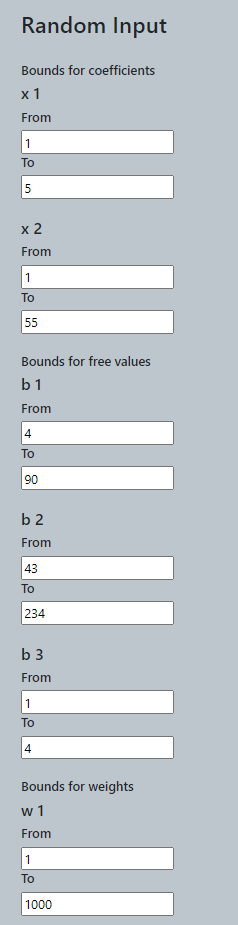


Рисунок 4.4 – Приклад введення меж для значень вхідних даних задачі

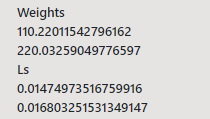
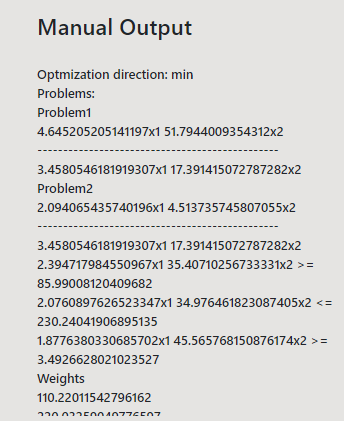


Рисунок 4.5 – Приклад виведення згенерованої умови задачі

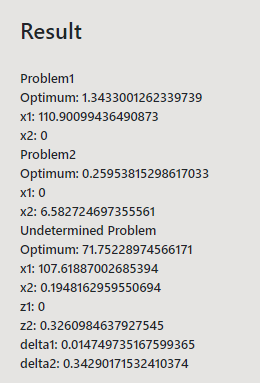


Рисунок 4.6 – Приклад виведення розв’язку

5. Організація експериментів з моделлю

5.1 План експериментів

Метою експериментів є дослідження впливу вхідних даних на поведінку моделі. Спочатку випадково генерується умова задачі дробно-лінійного програмування в умовах невизначенності враховуючи обмеження на значення величин. Далі змінюючи один з вхідних параметрів, ми можемо прослідкувати характер зміни вихідних параметрів. Кожен експеримент буде проведений декілька разів для достовірності результатів, а також як для задачі на мінімум так і на максимум.

5.1.1 Експеримент типу 1

Присвоюємо . Змінюємо значення до 1 та до 0.001 відповідно з кроком 0.001. Таким чином маємо можливість спостерігати як змінюются при зміні

5.1.2 Експеримент типу 2

Присвоюємо . Змінюємо значення до 1000 та до 1 відповідно з кроком 1. Таким чином маємо можливість спостерігати як змінюются при зміні

5.1.3 Експеримент типу 3

Присвоюємо . Змінюємо значення до 1000 та до 1 відповідно з кроком 1. Таким чином маємо можливість спостерігати як змінюются при зміні

5.2. Аналіз і оцінка результатів

На рисунках 5.1 та 5.2 можемо бачити залежність вихідних величин від l. – величина, що показує наскільки ми повинні “посунутися” у випадку якщо не задовольняється обмеження (6). Тож бачимо, що при збільшенні величини l зменшується та як видно на рисунку 5.2 при зменшенні величини l збільшується . Також можемо зробити висновок, що при деяких значеннях існує інтервал при якому значення обох дорівнює нулю.

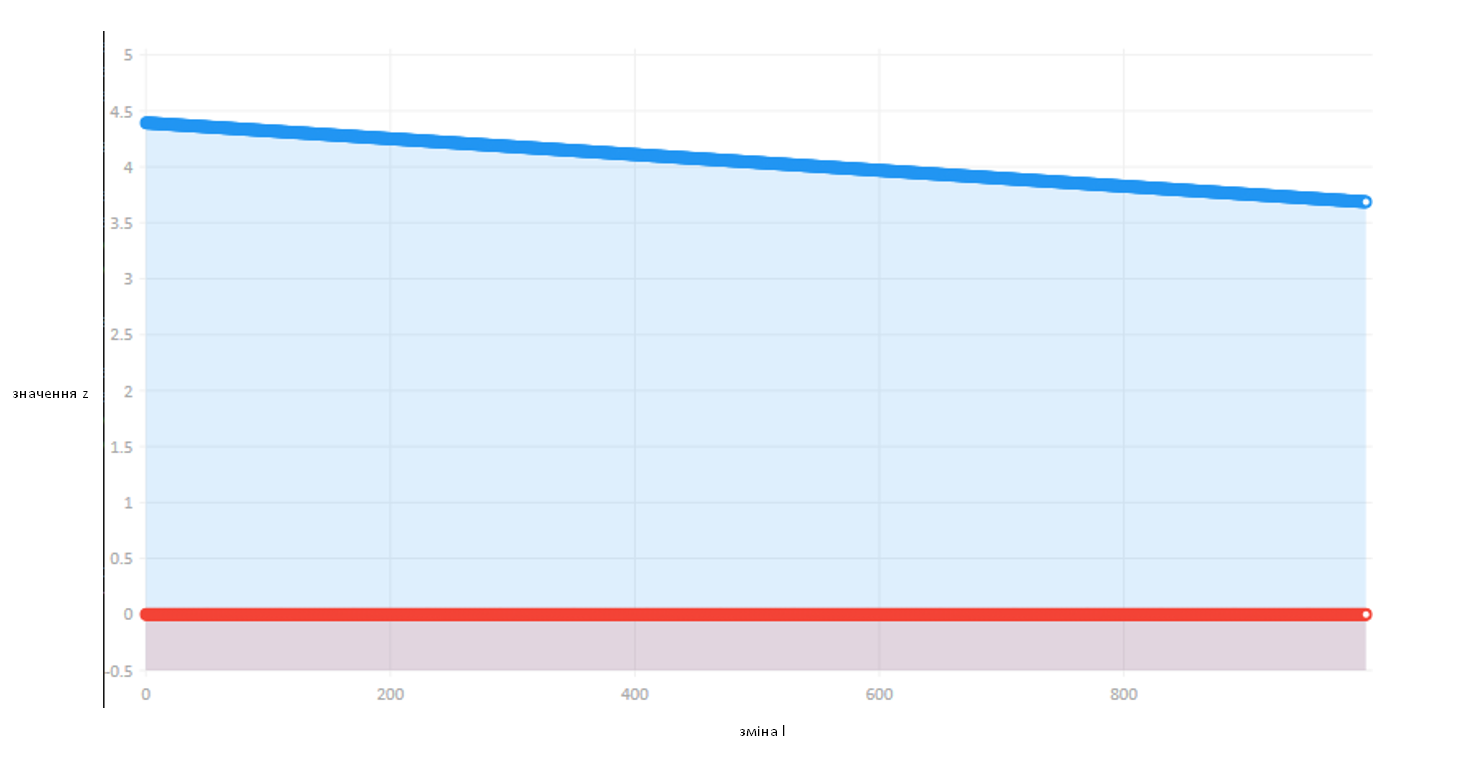


Рисунок 5.1 – приклад результатів експерименту типу 1 у випадку задачі на максимум.

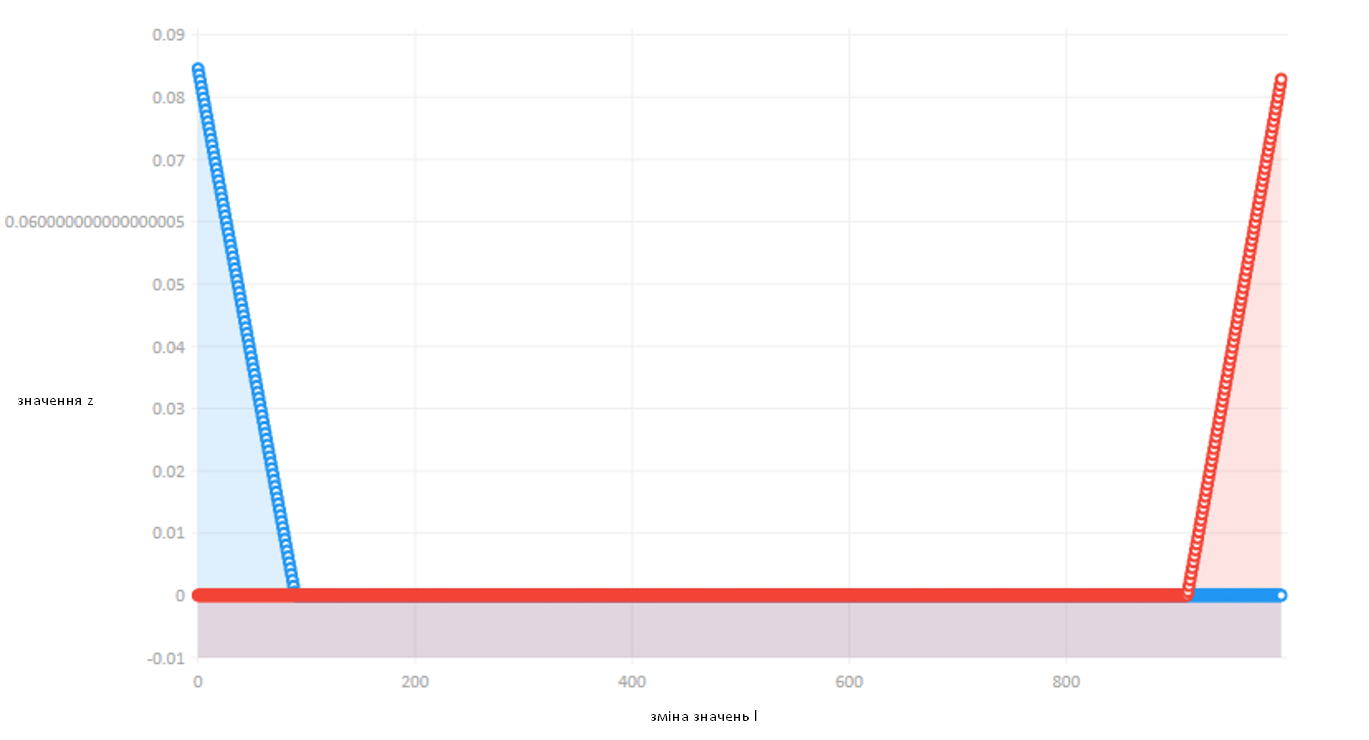


Рисунок 5.2 – приклад результатів експерименту 1 у випадку задачі на мінімум.

На рисунках 5.3 та 5.4 можемо бачити залежність вихідних величин від . – величина, що показує наскільки ми повинні “посунутися” у випадку якщо не задовольняється обмеження (6). Тож бачимо, що при збільшенні величини зменшується та при зменшенні величини збільшується .

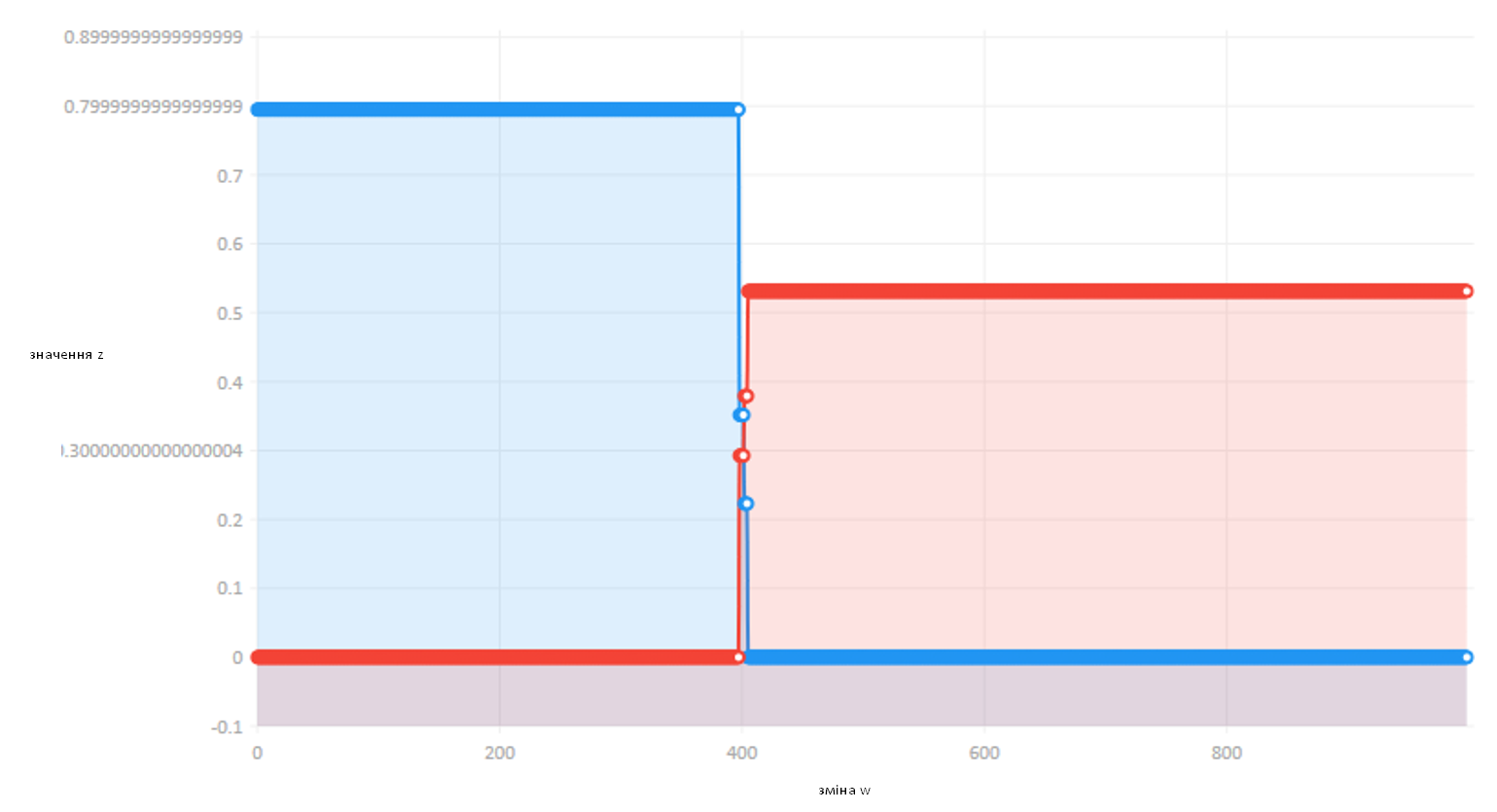


Рисунок 5.3 – приклад результатів експерименту типу 2 у випадку задачі на максимум.

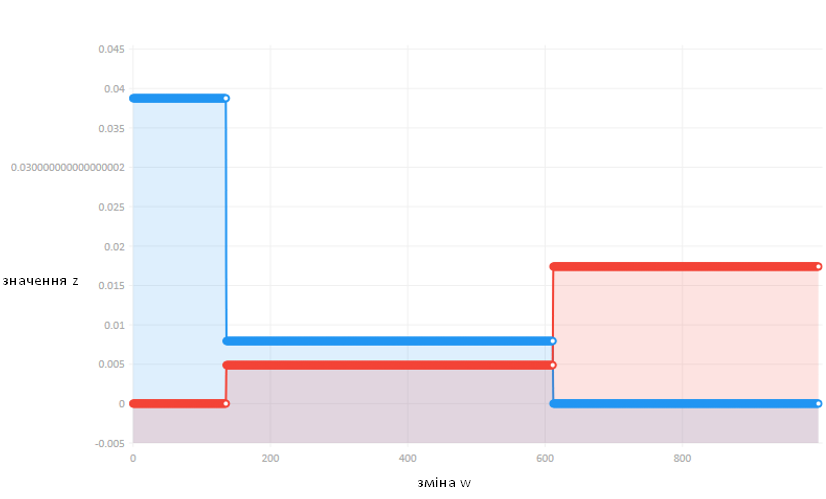


Рисунок 5.4 – приклад результатів експерименту 2 у випадку задачі на мінімум.

На рисунках 5.5 та 5.6 можемо бачити залежність вихідних величин від . – величина, що показує різницю між оптимальним значенням цільової функції та тим значенням, що ми отримуємо при пошуку компромісного рішення при розв’язанні ЗДЛПУН. Тож бачимо, що при збільшенні величини w зменшується та при зменшенні величини збільшується .

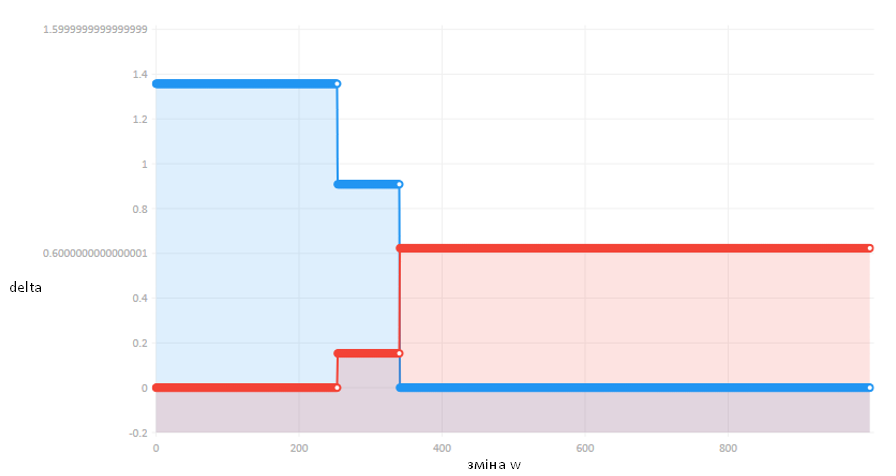


Рисунок 5.5 – приклад результатів експерименту 3 у випадку задачі на максимум.

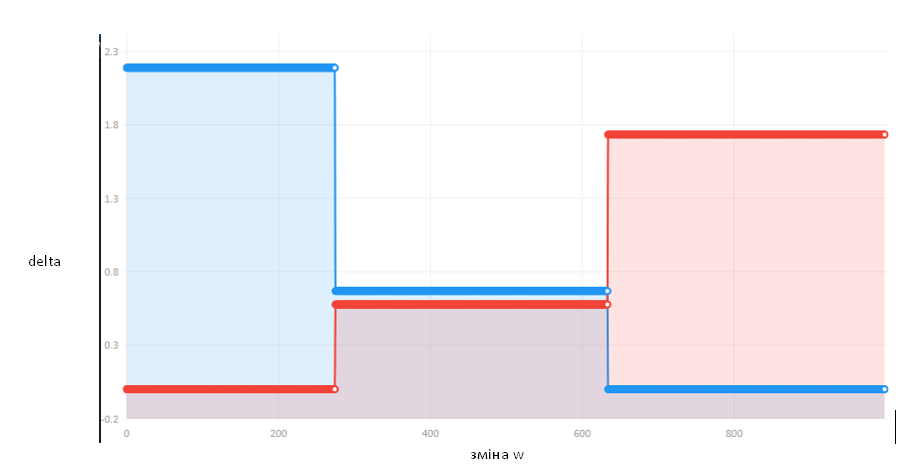


Рисунок 5.6 – приклад результатів експерименту 3 у випадку задачі на мінімум

ВИСНОВКИ

Була створено програмне забезпечення для вирішення проблеми дробно-лінійного програмування в умовах невизначеності та проведені дослідження щодо впливу вхідних даних на поведінку моделі в цілому. На основі єкспериментів, ми отримали графіки, з яких в свою чергу ми можемо зробити висновок, що зміна вихідних параметрів моделі має інтервальний характер – існують деякі значення вхідних даних при якому задача різко змінює вихідні дані. Також, бачимо, що при деякому значенні вхідних даних досягається задоволення критерію А () і в такому випадку нема необхідності використовувати критерій B.

ПЕРЕЛІК ПОСИЛАНЬ

1. Pavlov A.A. Optimization for one class of combinatorial problems under uncertainty. *Адаптивні системи автоматичного управління*. 2019. **1**. № 34. С. 81–89. doi: 10.20535/1560-8956.1.2019.178233.
2. Pavlov A.A. Combinatorial optimization under uncertainty and formal models of expert estimation. *Вісник Національного технічного університету «ХПІ».* 2019. № 1. С. 3–7. [[doi](https://doi.org/10.20998/2079-0023.2019.01.01)](https://doi)[: 10.20998/2079-0023.2019.01.01](https://doi.org/10.20998/2079-0023.2019.01.01).
3. Павлов А.А., Жданова Е.Г. Транспортная задача в условиях неопределенности // Проблемы управления и информатики. – 2020. – № 2 – С.34-45.
4. Павлов А.А., Жданова Е.Г. Задача дробно-линейного программирования в условиях неопределенности.
5. Г. Вагнер. Основы исследования операций, том 2. C.381.
6. И. Л. Акулич. Математическое программирование в примерах и задачах. 1986. С. 11-12.

ДОДАТОК. ЛІСТИНГ ПРОГРАМИ

Клас SolveHelper

public class SolveHelper : ISolveHelper

{

public Tuple<List<double>, List<List<double>>, List<List<double>>, List<double>> SolveProblem(List<List<double>> numerators, List<double> denominator, List<Constraint> \_constraints, List<double> ls, List<double> ws, OptDirectionEnum \_optDirection)

{

var fs = new List<double>();

var xs = new List<List<double>>();

var zs = new List<List<double>>();

var deltas = new List<double>();

var constraints = ConvertToDualConstraints(\_constraints, denominator);

for (int i = 0; i < numerators.Count; i++)

{

List<double> func = new List<double>

{

0

};

func.AddRange(numerators[i]);

try

{

Tuple<double, List<double>, List<double>> problem1 = Solve(

constraints,

func,

\_optDirection,

denominator.Count,

0);

fs.Add(problem1.Item1);

xs.Add(problem1.Item2);

zs.Add(problem1.Item3);

}

catch (NoOptimumException ex)

{

throw ex;

}

catch (Y0IsNullException ex)

{

throw ex;

}

}

List<double> fun = new List<double>();

for (int i = 0; i < denominator.Count + 1; i++)

{

fun.Add(0);

}

fun.AddRange(ws);

try

{

Tuple<double, List<double>, List<double>> problem = Solve(

ConvertToDualConstraintsWithZ(constraints, numerators, ls, fs, \_optDirection),

fun,

OptDirectionEnum.min,

denominator.Count,

ls.Count);

fs.Add(problem.Item1);

xs.Add(problem.Item2);

zs.Add(problem.Item3);

deltas = FindDeltas(problem.Item2, numerators, denominator, fs);

}

catch (NoOptimumException ex)

{

throw ex;

}

catch (Y0IsNullException ex)

{

throw ex;

}

return new Tuple<List<double>, List<List<double>>, List<List<double>>, List<double>>(fs, xs, zs, deltas);

}

private List<double> FindDeltas(List<double> xs, List<List<double>> numerators, List<double> denominator, List<double> fOpts)

{

var deltas = new List<double>();

double denominatorValue = 0.0;

for (int k = 0; k < denominator.Count; k++)

{

denominatorValue += (denominator[k] \* xs[k]);

}

for (int i = 0; i < numerators.Count; i++)

{

double numeratorValue = 0.0;

for (int j = 0; j < numerators[i].Count; j++)

{

numeratorValue += (numerators[i][j] \* xs[j]);

}

deltas.Add((numeratorValue / denominatorValue) - fOpts[i]);

}

return deltas;

}

private List<Constraint> ConvertToDualConstraintsWithZ(List<Constraint> \_constraints, List<List<double>> numerators, List<double> ls, List<double> fOpts, OptDirectionEnum optDirection)

{

List<Constraint> constraintStructs = new List<Constraint>();

for (int i = 0; i < \_constraints.Count; i++)

{

for (int j = 0; j < numerators.Count; j++)

{

\_constraints[i].Coefficients.Add(0);

}

constraintStructs.Add(\_constraints[i]);

}

for (int i = 0; i < numerators.Count; i++)

{

var coefs = new List<double>();

coefs.Add(0);

coefs.AddRange(numerators[i]);

for (int j = 0; j < numerators.Count; j++)

{

if (i == j)

{

if (optDirection == OptDirectionEnum.min)

{

coefs.Add(-1);

}

else

{

coefs.Add(1);

}

}

else

{

coefs.Add(0);

}

}

if (optDirection == OptDirectionEnum.min)

{

constraintStructs.Add(new Constraint(coefs, SymbolEnum.LessOrEqual, ls[i] + fOpts[i]));

}

else

{

constraintStructs.Add(new Constraint(coefs, SymbolEnum.MoreOrEqual, fOpts[i] - ls[i]));

}

}

return constraintStructs;

}

private List<Constraint> ConvertToDualConstraints(List<Constraint> \_constraints, List<double> denominators)

{

List<Constraint> constraintStructs = new List<Constraint>();

for (int i = 0; i < \_constraints.Count; i++)

{

List<double> coefs = new List<double>();

coefs.Add(\_constraints[i].FreeValue \* (-1));

for (int j = 0; j < \_constraints[i].Coefficients.Count; j++)

{

coefs.Add(\_constraints[i].Coefficients[j]);

}

constraintStructs.Add(new Constraint(coefs, \_constraints[i].SymbolEnum, 0));

}

var coefs2 = new List<double>();

coefs2.Add(0);

coefs2.AddRange(denominators);

constraintStructs.Add(new Constraint(coefs2, SymbolEnum.Equal, 1));

return constraintStructs;

}

private Tuple<double, List<double>, List<double>> Solve(List<Constraint> \_constraints, List<double> \_function, OptDirectionEnum \_optDirection, int xCount, int zCount)

{

Solver = Solver.CreateSolver("GLOP");

List<Variable> variables = new List<Variable>();

//add y

for (int i = 0; i < \_function.Count; i++)

{

variables.Add(solver.MakeNumVar(0.0, double.PositiveInfinity, "y" + i));

}

List<Google.OrTools.LinearSolver.Constraint> constraints = CreateConstraints(solver, \_constraints);

SetConstraints(constraints, \_constraints, variables);

//setup function

Objective = solver.Objective();

for (int i = 0; i < \_function.Count; i++)

{

objective.SetCoefficient(variables[i], \_function[i]);

}

if (\_optDirection == OptDirectionEnum.max)

{

objective.SetMaximization();

}

else

{

objective.SetMinimization();

}

Solver.ResultStatus resultStatus = solver.Solve();

// Check that the problem has an optimal solution.

if (resultStatus != Solver.ResultStatus.OPTIMAL)

{

throw new NoOptimumException();

//Console.WriteLine("The problem does not have an optimal solution!");

}

if (variables[0].SolutionValue() == 0)

{

throw new Y0IsNullException();

}

List<double> ys = new List<double>();

for (int i = 0; i < xCount + 1; i++)

{

ys.Add(variables[i].SolutionValue());

}

List<double> zs = new List<double>();

for (int i = 0; i < zCount; i++)

{

zs.Add(variables[xCount + 1 + i].SolutionValue());

}

return new Tuple<double, List<double>, List<double>>(solver.Objective().Value(), ConvertToX(ys), zs);

}

private List<Google.OrTools.LinearSolver.Constraint> CreateConstraints(Solver solver, List<Constraint> \_constraints)

{

//create constraints

List<Google.OrTools.LinearSolver.Constraint> constraints = new List<Google.OrTools.LinearSolver.Constraint>();

for (int i = 0; i < \_constraints.Count; i++)

{

Google.OrTools.LinearSolver.Constraint c;

SymbolEnum = \_constraints[i].SymbolEnum;

switch (symbolEnum)

{

case SymbolEnum.Equal: { c = solver.MakeConstraint(\_constraints[i].FreeValue, \_constraints[i].FreeValue); constraints.Add(c); break; }

case SymbolEnum.LessOrEqual: { c = solver.MakeConstraint(double.NegativeInfinity, \_constraints[i].FreeValue); constraints.Add(c); break; }

case SymbolEnum.MoreOrEqual: { c = solver.MakeConstraint(\_constraints[i].FreeValue, double.PositiveInfinity); constraints.Add(c); break; }

}

}

return constraints;

}

private void SetConstraints(List<Google.OrTools.LinearSolver.Constraint> constraints, List<Constraint> constraints2, List<Variable> variables)

{

for (int i = 0; i < constraints.Count; i++)

{

for (int j = 0; j < variables.Count; j++)

{

constraints[i].SetCoefficient(variables[j], constraints2[i].Coefficients[j]);

}

}

}

private List<double> ConvertToX(List<double> ys)

{

List<double> xs = new List<double>();

for (int i = 0; i < ys.Count - 1; i++)

{

xs.Add(ys[i + 1] / ys[0]);

}

return xs;

}

}

}

Клас RandProblemGenerator

public class RandProblemGenerator

{

public Tuple<List<List<double>>, List<double>, List<Constraint>, List<double>, List<double>, OptDirectionEnum> GenerateProblem(List<Tuple<double, double>> coefBounds, List<Tuple<double, double>> bBounds, int constraintCount, string optDirectionString, List<Tuple<double, double>> lsBounds, List<Tuple<double, double>> wsBounds)

{

//define optimization direction

OptDirectionEnum optDirection;

switch (optDirectionString)

{

case "min": { optDirection = OptDirectionEnum.min; break; }

case "max": { optDirection = OptDirectionEnum.max; break; }

default: { optDirection = GenerateOptDirection(); break; }

}

//define ws

List<double> ws = new List<double>();

for (int i = 0; i < wsBounds.Count; i++)

{

ws.Add(DoubleRand(wsBounds[i].Item1, wsBounds[i].Item2));

}

//define ls

List<double> ls = new List<double>();

for (int i = 0; i < lsBounds.Count; i++)

{

ls.Add(DoubleRand(lsBounds[i].Item1, lsBounds[i].Item2));

}

//define numerators

List<List<double>> numerators = new List<List<double>>();

for (int i = 0; i < wsBounds.Count; i++)

{

List<double> c = new List<double>();

for (int j = 0; j < coefBounds.Count; j++)

{

c.Add(DoubleRand(coefBounds[j].Item1, coefBounds[j].Item2));

}

numerators.Add(c);

}

//define denominator

List<double> denominator = new List<double>();

for (int i = 0; i < coefBounds.Count; i++)

{

denominator.Add(DoubleRand(coefBounds[i].Item1, coefBounds[i].Item2));

}

//define constraints

List<Constraint> constraints = new List<Constraint>();

for (int i = 0; i < constraintCount; i++)

{

List<double> constrCoefs = new List<double>();

for (int j = 0; j < coefBounds.Count; j++)

{

constrCoefs.Add(DoubleRand(coefBounds[j].Item1, coefBounds[j].Item2));

}

double freeValue = DoubleRand(bBounds[i].Item1, bBounds[i].Item2);

SymbolEnum symbol = GenerateSymbolEnum();

constraints.Add(new Constraint(constrCoefs, symbol, freeValue));

}

return new Tuple<List<List<double>>, List<double>, List<Constraint>, List<double>, List<double>, OptDirectionEnum>(numerators, denominator, constraints, ls, ws, optDirection);

}

private OptDirectionEnum GenerateOptDirection()

{

Random = new Random();

int value = random.Next(2);

if (value == 0)

return OptDirectionEnum.min;

else

return OptDirectionEnum.max;

}

private SymbolEnum GenerateSymbolEnum()

{

Random = new Random();

int value = random.Next(3);

if (value == 0)

return SymbolEnum.LessOrEqual;

else

if (value == 1)

return SymbolEnum.Equal;

else

return SymbolEnum.MoreOrEqual;

}

private double DoubleRand(double min, double max)

{

Random = new Random();

return min + (random.NextDouble() \* (max - min));

}

}