Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

**Кваліфікаційна робота**

освітній ступінь – бакалавр

на тему: **«Оптимізація мережі ланцюгів постачання за умов невизначеності»**

Виконав: студент 4-го року навчання
освітньої програми «Прикладна математика»,
спеціальності 113 Прикладна математика

Дубов Ілья Володимирович

Керівник: Чорней Р. К.
Зав.кафедри математики, доцент,
кандидат фіз.-мат. наук

Рецензент:

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК_____
(підпис)

«_____» _____ 2025р.

Київ - 2025

Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

ЗАТВЕРДЖУЮ
Зав.кафедри математики,
доцент, кандидат фіз.-мат. наук
_____ *Чорней Р.К.*
(підпис)
“____”_____ 2025

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
для кваліфікаційної роботи
студенту 4-го курсу, факультету інформатики
Дубову Ільї Володимировичу

**Тема:** «Оптимізація мережі ланцюгів постачання за умов невизначеності»

**Зміст кваліфікаційної роботи:**

Анотація
Терміни і позначення
Вступ
1. Лінійне програмування
2. Стохастичне програмування
3. Приклади і розв’язки задач
Висновки
Список літератури

Дата видачі “____”_____ 2025 Керівник _____
(підпис)
Завдання отримав _____
(підпис)

# Графік підготовки кваліфікаційної роботи до захисту

Графік узгоджено «_____» _____ 2025р.

| № з/п | Перелік робіт | Термін виконання етапу | Підпис наукового керівника | Дата ознайомлення наукового керівника | Примітка |
|---|---|---|---|---|---|
| 1. | Отримання теми кваліфікаційної роботи. | 17.09.2024 | | | |
| 2. | Розробка плану та структури роботи. | 27.09.2024 | | | |
| 3. | Робота з науковою літературою, опис основних означень. Написання вступу та анотації. | 21.10.2024 | | | |
| 4. | Дослідження задач стохастичного лінійного програмування. | 01.02.2025 | | | |
| 5. | Робота над текстовим оформленням теоретичної частини та одержаних результатів. | 18.02.2025 | | | |
| 6. | Попередній аналіз кваліфікаційної роботи. Виправлення помилок. | 04.04.2025 | | | |
| 7. | Попередній захист кваліфікаційної роботи. | 23.05.2025 | | | |
| 8. | Захист кваліфікаційної роботи. | 05.06.2025 | | | |

Науковий керівник _____
*(ПІБ)*

Виконавець кваліфікаційної роботи _____
*(ПІБ)*

# Зміст

# Анотація

Метою даної кваліфікаційної роботи було дослідження класу оптимізаційних задач за умов невизначеності. Розглянуто основну теоретичну базу від найтривіальніших задач детермінованого лінійного програмування, до більш абстрактних моделей з використанням теорії вибору за умов невизначеності.

Для моделювання реальних задач з великою кількістю вхідних даних (обмежень) було використано програмне забезпечення Python, а саме код бібліотеки SciPy. Переважна більшість розрахунків ведеться саме програмованим шляхом.

Розглянуто приклади оптимізації задач в різних галузях людської діяльності з використанням спільного підходу до визначення розв'язку.

# Терміни і позначення

- **LP** — лінійне програмування (linear programming)
- **SP** — стохастичне програмування (stochastic programming)
- **SLP** — стохастичне лінійне програмування (stochastic linear programming)
- **B&B** — алгоритм розгалуження і розмежування (Branch and Bound)
- **VaR** — вартість під ризиком (Value at Risk)
- **CVaR** — умовна вартість під ризиком (conditional Value at Risk)

# Вступ

У сучасному світі оптимізаційні моделі є рушійним фактором прийняття рішень у різних сферах людської діяльності, наприклад в економіці, логістиці, наданні послуг, розбудові бізнесу, тощо. Рішення щодо зменшення витрат, збільшення прибутку, мінімізації ризків тощо, можуть прийматися чи не на щоденній основі. Причому кількість даних, які залучені у прийняття оптимального рішення, може вимірюватися різними порядками. Окрім самих вхідних даних, проблему ще можуть описувати різноманітні події навколишнього середовища, які створюють значний вплив на поставлену задачу, але не зазнають впливу зі сторони того, хто приймає рішення. Враховуючи дані фактори, постає необхідність сформулювати певну математичну теорію, яка дозволить класифікувати проблеми оптимізації і пошуку оптимального рішення.

Найбільш тривіальною моделлю є лінійне програмування (LP), яке досить добре описує задачі пошуку оптимального розв'язку за наданих умов. Така модель є детермінованою - тобто такою, чий стан ми можемо передбачити, і вона не зазнає впливу з боку природи. Детермінованість дозволяє досить швидко і легко знаходити розв'язки до цілого класу проблем, але водночас, не покриває повністю потреби сучасного світу.

Один з методів, які більше описують реальні умови, — стохастичне лінійне програмування (SLP). SLP використовується переважно у випадках, коли система може зазнавати значного впливу з боку природи. Параметри моделі мають стохастичний (випадковий) характер: ми заздалегідь не знаємо які умови будуть в майбутньому, але знаємо ймовірнісний розподіл цих умов. Такий підхід дозволяє більш реалістично оцінювати проблему і шляхи її оптимізації.

SLP не є чітко-визначеною моделлю, бо є більш орієнтованою на конкретні ситуації. Це можна дослідити у постанові цільової функції, а саме яким чином звести вплив ймовірнісних чинників природи до реальних, прикладних даних. У рамках роботи наведені приклади з різними критеріями, але детальніше розглядається критерій **Баєса-Лапласа**, що формує задачу LP.

Окрім звичних чисельно-дробових даних, можливе використання виклю-

чно цілочисельних, або булевих параметрів. Іноді задачі потребують дискретного характеру, тобто такі, що мають на меті знайти оптимальну кількість **штук** певної одиниці. В такому випадку окрім наявних обмежень значень зверху-знизу, має ще бути обмеження на тип самого числа, наприклад ціле значення. У випадку коли рішення може бути або прийнято, або не прийнято, можливе позначення через 1 або 0 відповідного рішення, іншими словами - булеве значення.

У роботі також розглянуто теоретичну основу для знаходження розв'язків задач, а також зручніші варіанти обрахунків за допомогою програмного забезпечення мовою Python, та бібліотеки SciPy.

Тож питання знаходження розв'язку оптимізаційної задачі є досить великою темою, яка потребує спеціально побудованої теорії, а також програмного забезпечення для аналізу отриманих результатів.

# 1 Лінійне програмування

## 1.1 Історія та актуальність

Лінійне програмування не має такої великої історії порівняно з іншими галузями математики, головним чином через труднощі вирішення більшості проблем без допомоги комп'ютерних обчислень. Одну з перших спроб забезпечити життєздатне вирішення проблем лінійного програмування зробив Джозеф Фур'є, який опублікував метод у 1827 році. Однак його алгоритм був неефективним для великих систем, оскільки для його виконання потрібен був принаймні експоненціальний час, тобто для вирішення задач із сотнею чи більше змінними все одно знадобиться неможлива кількість часу. Було зроблено кілька інших спроб знайти рішення, але вони мали обмежений успіх. Основні розробки в цій галузі не будуть зроблені до середини 20 століття.

Найбільш відомий і значний внесок у область лінійного програмування зробив Джордж Данціг у 1947 році, коли він розробив симплекс-метод для розв'язування задач лінійного програмування, працюючи військовим радником у Пентагоні [4]. Симплекс-метод був першим практичним алгоритмом, який мав ефективне рішення задач лінійного програмування. Однак на момент його створення ще існували обмеження щодо здатності вирішувати великі проблеми. Насправді, одна з перших реалізацій алгоритму у великій системі вимагала 8 годин подачі карток у картковий програмований калькулятор [1]. Незважаючи на ці обмеження, симплекс-алгоритм дозволив людям вирішувати задачі лінійного програмування набагато ефективніше, ніж будь-який попередній метод. Досить цікаво, що назва лінійне програмування має мало спільного з реальними комп'ютерними програмами, і насправді походить від військового використання слова «програма» для позначення їх матеріально-технічного забезпечення та планів розгортання [3].

За роки з моменту, коли воно було вперше запропоновано в 1947 році автором (у зв'язку з діяльністю військового планування), лінійне програмування та його численні розширення набули широкого застосування. В академічних колах вчені, які займаються прийняттям рішень (дослідники операцій і вчені з менеджменту), а також численні аналітики, математики та економісти написали сотні книг і незліченну кількість статей на цю тему.

Алан Дж. Хоффман (1953) представив перший приклад того, що Данціг (1963) вважав за краще називати кружлянням, але загальновідоме як їзда на велосипеді [8]. Тут відбувається послідовність ітерацій без зміни значення цільової функції (у цьому випадку метод не збігається). Циклування може відбуватися, якщо є рішення, яке відповідає більш ніж одному вибору набору основних змінних, ситуація називається виродженням. Данциг (1951) запропонував різні методи уникнення їзди на велосипеді, які виявилися успішними. Протягом цього періоду відбулися дві додаткові події, які мали фундаментальний вплив на майбутнє LP.

В 1984 році була стаття Н. Кармаркара [7]. Кармаркар використовував проективні перетворення, щоб продемонструвати поліноміальну часову межу для LP, яка була не тільки набагато кращою, а також відповідала обчислювальному підходу, який був застосовним на практиці. Стаття Кармаркара призвела до надзвичайного буму теоретичної роботи в лінійному програмуванні та суміжних областях, яка багато в чому продовжується і до цього дня.

Щодо обчислювальної сторони, AT&T розробила систему KORBX, що виявилося значною мірою невдалою спробою комерційного використання прориву Кармаркара. Однак у той же час дослідники швидко визнали зв'язок між теоретичним внеском Кармаркара та більш ранньою роботою Фіакко та МакКорміка щодо методів логарифмічного бар'єру. Це усвідомлення врешті-решт призвело до розробки класу алгоритмів, відомих як первинно-подвійні лог-бар'єрні алгоритми. Ці результати добре задокументовані щодо обчислювальної сторони в роботі Люстіга, Марстена та Шанно [5], які розробили код OB1 FORTAN, що реалізує ранні версії цього алгоритму логарифмічного бар'єру. Цей код був загальнодоступним приблизно в 1991 році, і разом з удосконаленнями, які відбувалися в цей же період із симплексними алгоритмами в таких кодах, як CPLEX і OSL, це означало кінець коду KORBX. Хоча сам OB1 також не був комерційно успішним, він, тим не менш, був провідним кодом бар'єрних алгоритмів свого часу та викликав величезний інтерес.

Період близько 1990 року був надзвичайно активним періодом у LP. Робота Кармаркара стимулювала ще більше інтересу до LP, як з теоретичної, так і з обчислювальної сторони. Це не тільки призвело до кращого розуміння та покращених реалізацій бар'єрних алгоритмів, але також призвело до

відродження інтересу до симплексних алгоритмів і певною мірою відповідає за деякі з ранніх розробок у коді CPLEX LP, вперше випущеному в 1988 році. Приблизно в той же час IBM також випустила свій код OSL, визначену заміну для MPSX/370, розробленого в основному Джоном Форрестом і Джоном Томліном. Ці два коди CPLEX і OSL були домінантними кодами LP на початку 1990-х років і включали реалізації первинних і подвійних симплексних алгоритмів, а також, зрештою, бар'єрних алгоритмів. Для коду CPLEX багато з цих розробок задокументовано в [2].

У сьогоденні лінійне програмування є більш корисним, ніж будь-коли раніше, для вирішення завдань управління та логістики завдяки легкому доступу до великих обсягів даних. У поєднанні з вдосконаленими алгоритмами та вдосконаленнями обчислювальної потужності лінійне програмування здатне вирішувати більші та складніші проблеми.

Основною метою лінійного програмування є оптимізація лінійної цільової функції, яка може полягати в максимізації або мінімізації певної величини. Ця цільова функція підпорядкована набору лінійних нерівностей або рівнянь, відомих як обмеження. Змінні в обмеженнях повинні бути невід'ємними, що відображає той факт, що від'ємні кількості ресурсів або продуктів не мають сенсу в більшості сценаріїв реального світу.

Наприклад, виробник може захотіти визначити найкраще поєднання продуктів для виробництва, яке максимізує прибуток без перевищення доступних ресурсів, таких як робоча сила, матеріали та обладнання. У цьому випадку цільова функція представляє прибуток, який необхідно максимізувати, а обмеження представляють обмеження ресурсів. Використання лінійного програмування дає найкращий можливий результат у певній ситуації.

## 1.2 Постановка задачі лінійного програмування

Лінійне програмування (LP) — це метод математичної оптимізації, яка використовується для максимізації або мінімізації лінійної цільової функції з урахуванням набору обмежень лінійної рівності та нерівності. Метод широко використовується в різних галузях, таких як економіка, техніка, дослідження операцій і наука про менеджмент, щоб знайти найкращий можливий результат за обмежених ресурсів.

**Означення 1.1** *Задачею лінійного програмування (LP) називають оптимізаційну задачу наведену нижче:*

$$z = c^T x \to \min(\max) \tag{1}$$

$$Ax \leq b \tag{2}$$

$$x_i \geq 0$$

Зазначимо, що розмірності векторів $x$ та $b$ не обов'язково мають бути однаковими. В загальному вигляді:

$$x = (x_1, x_2, \dots, x_n)$$

$$b = (b_1, b_2, \dots, b_p)$$

Матриця $A$, відповідно, матиме розмірність $p \times n$.

Компоненти лінійного програмування включають:

1. **Цільова функція** $z(x)$: це лінійна функція, яку необхідно оптимізувати (максимізувати або мінімізувати). Вона представляє кількість, яку потрібно максимізувати або мінімізувати, наприклад прибуток, вартість, час тощо.

2. **Змінні рішення** $x_i$: це змінні, які представляють вибір або рішення, які необхідно прийняти. Наприклад, встановити кількість ресурсів для постачання даному споживачу, визначити потужність виробництва, побудувати певну кількість текстильних фабрик, тощо. Мета розв'язку задачі полягає в знаходженні значень даних зміних, які оптимізують цільову функцію.

3. **Обмеження** $Ax \leq b$: це обмеження щодо змінних рішень, які мають бути задоволені. Обмеження можуть бути виражені у вигляді лінійних рівнянь або нерівностей. Вони описують обмеження, накладені наявними ресурсами, обмеженнями потужності, вимогами попиту тощо.

4. **Допустима область**: це набір усіх можливих комбінацій змінних рішень, які задовольняють усі обмеження. Він визначається перетином обмежень.

5. **Оптимальне рішення**: це найкраще можливе рішення, яке максимізує або мінімізує цільову функцію, задовольняючи всі обмеження.

Лінійне програмування забезпечує систематичний і ефективний підхід до прийняття рішень у ситуаціях, коли ресурси обмежені, а цілі необхідно оптимізувати. Нижче наведено типи задач лінійного програмування:

- *Виробничі проблеми*: У цих задачах оцінюється кількість одиниць різних предметів, які має виробляти та продавати компанія, коли кожен продукт вимагає заданої кількості робочої сили, машино-годин, робочих годин на одиницю продукту, складських площ на одиницю продукції тощо, щоб максимізувати прибуток.

  Виробничі проблеми передбачають максимізацію рівня виробництва або чистого прибутку виготовленої продукції, яка може вимірювати доступний робочий простір, кількість робітників, машино-годин, використані пакувальні матеріали, необхідну сировину, ринкову вартість продукту та інші фактори. Вони зазвичай використовуються в промисловому секторі, щоб передбачити майбутнє збільшення капіталу компанії з часом.

- *Проблеми з дієтою*: У цих завданнях оцінюється, скільки компонентів або поживних речовин має містити дієта, щоб знизити вартість бажаної дієти, гарантуючи мінімальну кількість кожного вітаміну.

  Як випливає з назви, проблеми, пов'язані з дієтою, можна вирішити, вживаючи більш конкретну їжу, яка багата основними поживними речовинами та може сприяти прийняттю певного плану дієти. Знайти набір страв,

який задовольнить набір щоденних харчових потреб за найменшу суму грошей, є метою проблеми дієти.

- *Транспортні проблеми*: У цих проблемах створюється графік транспортування, щоб знайти найбільш економічно ефективний метод транспортування продукту з різних заводів на різні ринки. Вивчення транспортних маршрутів або того, як предмети з різних джерел виробництва транспортуються на різні ринки з метою мінімізації загальних витрат на транспортування, пов'язане з транспортними труднощами. Аналіз таких викликів має вирішальне значення для великих фірм з кількома виробничими підрозділами та широкою базою клієнтів.

- *Задачі на оптимальне призначення*: Ця проблема стосується виконання компанією певного завдання шляхом вибору певної кількості працівників для виконання завдання протягом необхідного періоду часу, припускаючи, що кожна особа працює лише на одній роботі. Прикладами таких проблем є, наприклад, планування та управління подіями у великих організаціях.

- *Фінансові проблеми*: У цих проблемах фігурують питання максимізації прибутків з інвестицій, кредитів, боргів, тощо; або мінімізація ризику фінансових угод. Нерідко цей клас задач містить фактори невизначеності, які будуть розглянуті пізніше. Однією з таких проблем є, наприклад, оптимальний вибір активів у портфель інвестора, враховуючи ROI та ризик втрати.

| **Типи задач лінійного програмування** | **Цільова функція** | **Обмеження** |
|---|---|---|
| *Виробнича проблема* | Прибуток з продажу товару | Вартість пакувальних матеріалів, години роботи, тощо |
| *Проблеми з дієтою* | Вартість спожитої їжі | Визначена потреба в харчуванні |
| *Транспортна проблема* | Вартість транспортування | Унікальна модель попиту та пропозиції |
| *Задачі на оптимальне призначення* | Загальна кількість виконаних завдань | Час роботи кожного працівника та кількість працівників |
| *Фінансові проблеми* | Річний прибуток з інвестицій | Початковий капітал, вартість одного активу, ROI кожного активу, фактори ризику, тощо |

Табл. 1: Обмеження та цільова функція кожної задачі LP

## 1.3 Методи розв'язку задачі LP

Історично виникло 2 типи методів розв'язання оптимізаційних задач LP: *метод вершин* і *метод внутрішньої точки*. Другий іноді називають *«бар'єрним методом»*.

- *Метод вершин*

  Суть таких методів полягає у поступовому ітераційному обходженні граничних точок перетинів принаймні двох обмежень. Оптимальне значення цільової функції має бути досягнуто в одній з таких точок. Найбільш розповсюдженим методом вершин є класичний симплекс-метод. Існують також його варіації, наприклад двоїстий симплекс-метод, який є ідентичним до симплекс-методу, але використовується для розв'язання двоїстої задачі LP. Хоч і винайдення симплекс-методу стало революцією в розділі задач оптимізації, в найгіршому випадку він демонструє експоненційний час виконання. Саме цю проблему вирішує другий метод.

- *Метод внутрішньої точки (бар'єрний метод)*

  Дані методи демонструють поліноміальний час, що є значно швидше, ніж експоненціальний час симплекс-методу. Також, на відміну від попереднього методу, метод внутрішньої точки оперує значеннями всередині множини обмежень. Суть алгоритму полягає у визначенні деякої бар'єрної функції, яка зростає до нескінченності при наближенні точки до границі множини обмежень. До бар'єрної функції додається параметр $\mu$, що контролює її вплив.

  Нехай дано задачу LP (1) і обмеження (2). Для бар'єрного методу сформулюємо бар'єрну функцію:

$$\Phi(x) = -\sum_{i=1}^{p} \ln |b_i - A_i x| \tag{3}$$

  , де $A_i$ — $i$-ий рядок матриці $A$.

  Очевидно, що при наближенні $A_i x \to b_i$, логарифм швидко зростатиме до нескінченності, що і моделює ефект «бар'єру»[5]. Тоді початкова задача (1) може бути записана у наступному вигляді:

$$z(x, \mu) = c^T x + \mu\Phi(x) \to \min \tag{4}$$
$$\mu \in (0, 1)$$

Варто зазначити, що цільова функція мінімізується. Якщо умови задачі вимагають максимізації, то дуже просто перейти до задачі мінімізації замінивши вектор $c$ на $c' = -c$. Бар'єрна функція при цьому залишається незмінною.

Алгоритм передбачає наступні кроки:

1. Обрати деяку початкову точку в середині множини обмежень (2)
2. Обрати параметр $\mu$ і закон його почергового зменшення $\mu_{k+1} = g(\mu_k)$
3. Ітераційно знаходити мінімуми цільової функції (наприклад методом Ньютона, чи градієнтним спуском), зменшуючи кожен раз значення $\mu$.

$g(\mu)$ можна обирати довільним чином, але з вимогами:

$$\mu_{k+1} < \mu_k$$
$$\mu \in (0, 1)$$

Приклади таких способів:

– Лінійний: $g(\mu) = \alpha\mu, \alpha \in (0, 1)$
– Лінійний відносно кількості ітерацій: $g(\mu) = \frac{1}{k}\mu$
– Експоненційний: $g(\mu) = \mu e^{-\beta k}$

В подальших розрахунках та прикладах будемо застосовувати саме цей метод, а також ПО Python і бібліотеку SciPy.

**Приклад 1.1** *Ферма виготовляє щоденно 100 кг кормової суміші, в склад якої входять 5 інгредієнтів. 100 кг кормової суміші має містити щонайменше 20% білка, 3% клітковини та 1100 МДж енергії. Кожен інгредієнт має унікальну харчову цінність, яка наведена в таблиці:*

| **Інгредієнт** | **Вартість ($)** | **Білок (г)** | **Клітковина (г)** | **Енергія (МДж)** |
|---|---|---|---|---|
| *Кукурудзяне борошно* | 0.25 | 80 | 20 | 14 |
| *Соєвий шрот* | 0.4 | 440 | 60 | 12 |
| *Пшеничні висівки* | 0.2 | 160 | 100 | 10 |
| *Меляса* | 0.15 | 30 | 20 | 9 |
| *Насіння соняшника* | 0.3 | 300 | 80 | 11 |

Табл. 2: Харчова цінність і вартість кожного інгредієнта на 1 кг

*Скільки кілограм кожного інгредієнту треба закупити фермі, і яка їхня загальна мінімальна вартість?*

Розв'яжемо задачу у Python:

```
import scipy.optimize as opt
import numpy as np

A = np.array([
        [80, 440, 160, 30, 300],
        [20, 60, 100, 20, 80],
        [14, 12, 10, 9, 11],
])

b = np.array([20_000, 3_000, 1_100])

A_eq = np.array([[1,1,1,1,1]])
b_eq = np.array([100])

```

```
c = np.array([0.25, 0.4, 0.2, 0.15, 0.3])

res = opt.linprog(c, -A, -b, A_eq, b_eq, method='highs-ipm')
print("Result: ", res.x, "\nFunction value: ", res.fun)

# Result:  [15.625  0.    46.875  0.    37.5  ]
# Function value:  24.53125
```

Отже, фермі необхідно закупити 15.625 кг кукурудзяного борошна, 46.875 кг пшеничних висівок та 37.5 кг насіння соняшника на загальну суму у 24.53\$.

Звернемо увагу, що матриця $A$ та вектор $b$ встановлюють обмеження **знизу**, коли функція linprog приймає тільки обмеження **зверху**. Тому довелось помножити обидві структури на $-1$, щоб задовільнити цю умову.

## 1.4 Цілочисельні розв'язки. Алгоритм Branch and Bound (B&B)

Попередньо розглядалася теорія LP для дійсних чисел. Але нерідко виникає потреба знайти розв'язки у цілих числах. Наприклад, коли потрібно обраховувати кількість контейнерів, штук товару, одиниць наданої послуги, транспортувальних засобів, тощо. Очевидно, що дробова частина розв'язку не матиме ніякого сенсу. Водночас, просте видалення дробової частини з розв'язку може і дасть наближений результат, але не гарантуватиме оптимальність. Розглянемо приклад:

**Приклад 1.2**

$$z = 3x + 2y$$

$$2x + y \leq 7$$

$$x + 2y \leq 7$$

$$x, y \in \mathbb{Z}^+ \cup \{0\}$$

$$\max(z) = ?$$

Розв'яжемо задачу графічно.

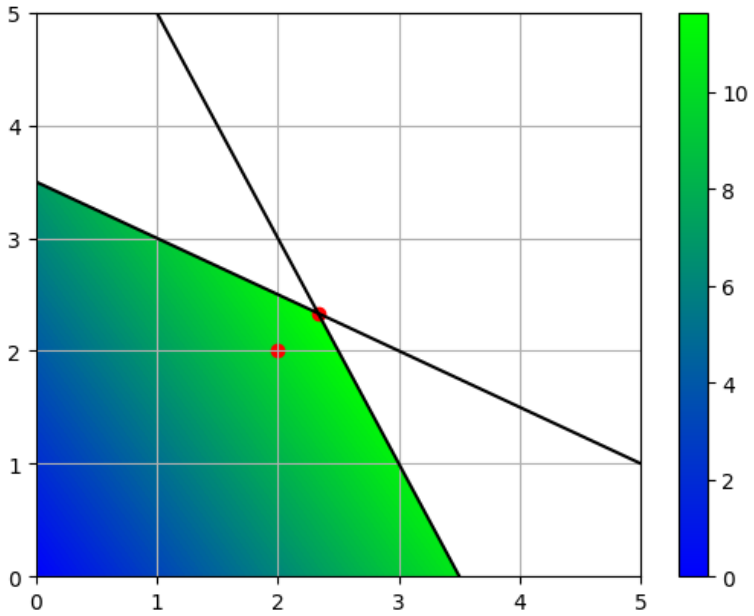



Рис. 1: Множина розв'язків і значення цільової функції

Як бачимо, точка $\left(\frac{7}{3}, \frac{7}{3}\right)$ дійсно максимізує цільову функцію $z = \frac{35}{3}$. Опустимо дробові частки розв'язку, і отримаємо точку $(2, 2)$, яка теж позначена на графіку. В цій точці цільова функція має значення $z = 10$, що досить близько до реального значення. Проте, можна помітити, що точка $(3, 1)$ теж належить області обмежень і надає значення цільовій функції $z = 11$, що є краще, аніж $(2, 2)$, а в даному прикладі є ще і оптимальним значенням. Саме тому постала необхідність сформувати алгоритм розв'язку задачі LP в цілих числах. Найбільш розповсюдженим методом є метод *розгалуження та розмежування (англ. Branch and Bound - B&B)*

**Означення 1.2** *Алгоритм розгалуження та розмежування (B&B) - це широко використовуваний метод отримання розв'язку оптимізаційних задач у цілих числах, який полягає у рекурсивній побудові дерева розв'язків з новими обмеженнями, і повторному розв'язку задачі до моменту, коли буде досягнуто розв'язку в цілих числах.*

Алгоритм можна описати наступним чином:

1. Розв'язати задачу без обмеження на цілі значення (релаксація). Якщо отриманий результат вийшов цілозначним - ігноруємо наступні кроки.

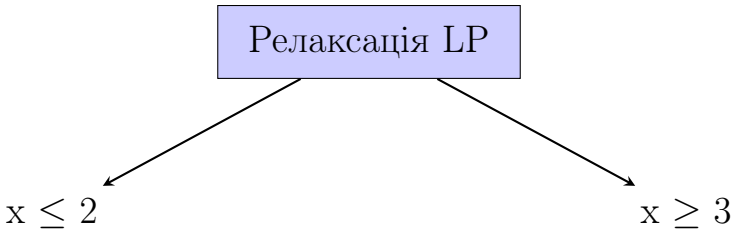
2. Для найбільшого нецілого значення змінної $x_k = p$ зробити розгалуження на 2 гілки, які представляють нові обмеження: $x_k \leq \lfloor p \rfloor$ та $x_k \geq \lceil p \rceil$

3. Розв'язати задачу з новими обмеженнями для одного випадку. Якщо отримано розв'язок не в цілих числах, повторити п. 2. Інакше, зафіксувати значення цільової функції та розв'язок і виконати ті самі обрахунки для другого випадку. Ігнорувати подальші кроки, якщо розв'язку не існує.

Розв'яжемо приклад 1.2 використовуючи алгоритм B&B і продемонструємо відповідні дерева. Всі обрахунки будуть аналогічними і проводитимуться у Python. Наведемо знаходження розв'язку для початкової задачі:

```
A = np.array([
        [2, 1],
        [1, 2],
])
b = np.array([7, 7])
c = np.array([-3, -2])

res = opt.linprog(c, A, b, method='highs-ipm')
print("Result: ", res.x, "\nFunction value: ", res.fun)

# Result:  [2.33333333 2.33333333]
# Function value:  -11.666666666666666
```

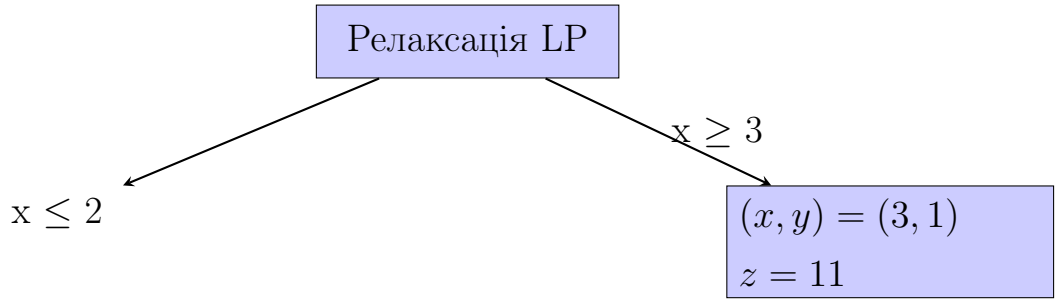
Після релаксації отриманий розв'язок $(x, y) = (\frac{7}{3}, \frac{7}{3})$. Значення цільової функції від'ємне, оскільки бібліотека вирішує задачу мінімізації. Відповідно, щоб досягти максимізації, цільова функція була помножена на $-1$. Тим не менш, результат повністю прогнозований. Нехай щодо змінної $x$ буде використано розгалуження. Маємо 2 гілки: $x \leq 2$ та $x \geq 3$, які стають новими додатковими обмеженнями. Зобразимо дерево:



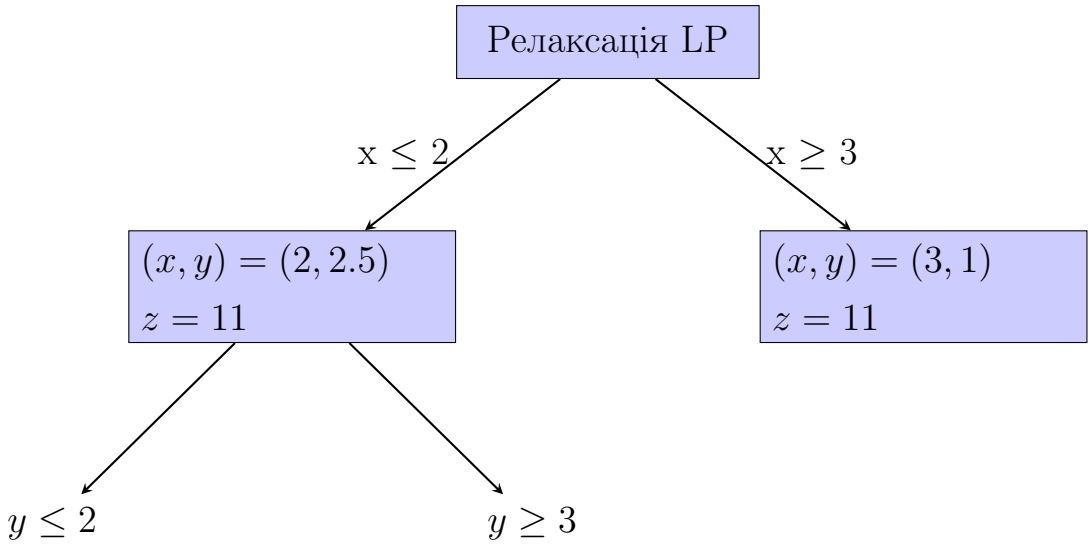

Розв'язуємо для правої гілки. Додається умова $x \geq 3$, яка рівносильна умові $-x \leq -3$.

```
A = np.array([
        [2, 1],
        [1, 2],
        [-1, 0],
])
b = np.array([7, 7, -3])
c = np.array([-3, -2])

res = opt.linprog(c, A, b, method='highs-ipm')
print("Result: ", res.x, "\nFunction value: ", res.fun)

# Result:  [3. 1.]
# Function value:  -11.0
```

Отриманий результат - це цілі числа $(x, y) = (3, 1)$ і значення $z = 11$. Отже, один з потенційний розв'язків знайдений. Вважатимемо дану гілку дерева завершеною. Оновимо дерево:



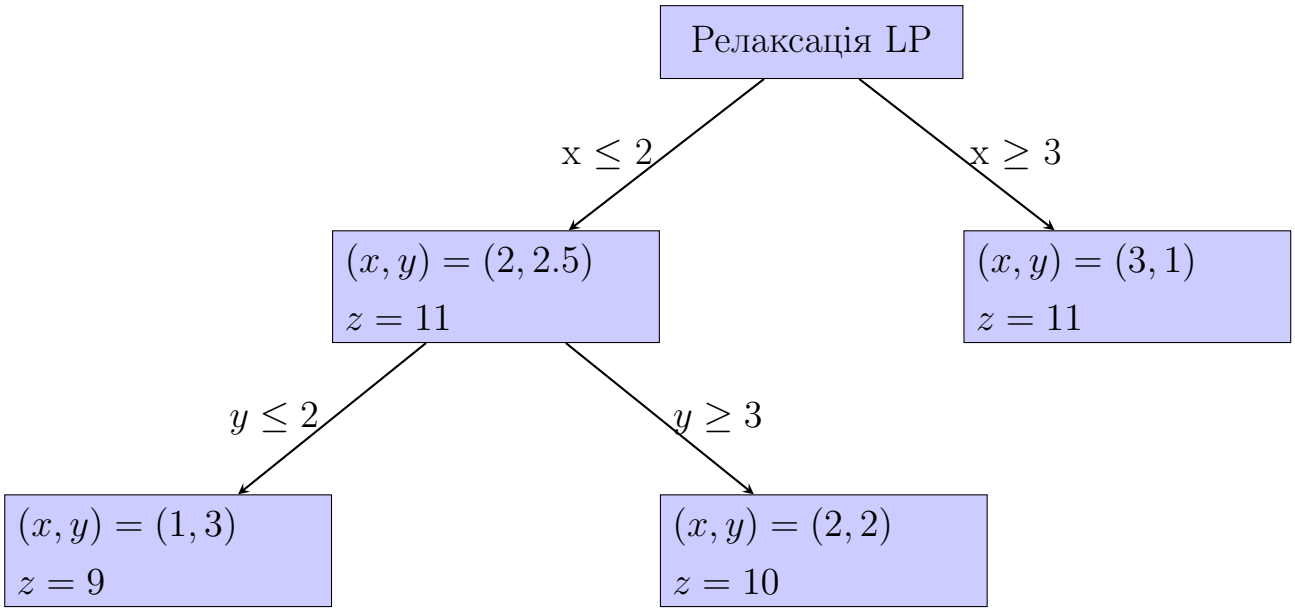

Аналогічно розв'яжемо для $x \leq 2$ і одразу побудуємо дерево:




Отримано ціле значення для $x$, але дробове для $y$. Тому робимо розгалуження для $y \leq 2$ та $y \geq 3$.

- $y \geq 3$: $(x, y) = (1, 3)$; $z = 9$
- $y \leq 2$: $(x, y) = (2, 2)$; $z = 10$

На обидвох гілках отримано цілі значення, а отже, задача завершена. Побудуємо кінцеве дерево. Відповідь до задачі буде знаходитись на одному з листків дерева. Обираємо той, де цільова функція має найбільше значення (або найменше, якщо задача мінімізації). В даному випадку $(x, y) = (3, 1)$ є **оптимальним розв'язком задачі в цілих числах**.




Аналогічний результат можна отримати у SciPy, використавши параметр integrality=1:

```
A = np.array([
        [2, 1],
        [1, 2],
])
b = np.array([7, 7])
c = np.array([-3, -2])

res = opt.linprog(c, A, b, method='highs', integrality=1)
print("Result: ", res.x, "\nFunction value: ", res.fun)

# Result:  [3. 1.]
# Function value:  -11.0
```

# 2 Стохастичне програмування

## 2.1 Постановка задачі стохастичного програмування

Раніше розглядалися моделі *детермінованого* LP. Вважалося, що модель має справджуватися у будь-якому випадку, і не містить в собі інші сценарії розвитку подій. Наприклад, у *транспортній задачі* досліджувалися оптимальні варіанти перевезення товару з метою максимізації прибутку. Проте нехтувалися такі можливі сценарії, як несправність транспортного засобу, затримка достави з подальшими штрафами, бракованість товару, тощо. Очевидно, що подібні ситуації не є рідкісними винятками, а скоріше природними факторами, які трапляються випадково і не залежать від постачальника чи логістичної компанії. Також очевидно, що просто нехтувати ними не є коректним шляхом вирішення проблеми, особливо коли ризики занадто високі (наприклад аварія під час перевезення нафтової продукції може спричинити екологічні проблеми та високі збитки на відновлення). Тож ризики теж треба враховувати.

Впродовж років постала необхідність сформувати математичну теорію для подібних задач, яка отримала назву *стохастичне програмування* [6]. Дамо означення ключовим термінам задач стохастичного програмування.

**Означення 2.1** *Стохастичне програмування (SP) – це метод прийняття оптимальних рішень в умовах ризику.*

**Означення 2.2** *Ризик - абстрактне поняття, що описує можливі сценарії розвитку подій, кожна з яких має свої обмеження і створює вплив на кінцевий результат.*

Ризик може бути представлений у векторі ймовірностей розвитку різних сценаріїв:

$$P = (p_1, p_2, ..., p_s) \tag{5}$$

$$\sum_{i=1}^{s} p_i = 1 \tag{6}$$

Сценарії формують ймовірнісний *простір сценаріїв* $\Omega$. Один із сценаріїв обов'язково настане. При настанні сценарію модель стає детермінованою.

**Означення 2.3** *Стохастичний критерій — функція $g(x, \Omega)$, яка, відповідно до контексту задачі, на основі ймовірнісного простору сценаріїв, описує предмет оптимізації.*

Іншими словами, стохастичний критерій — це фактично «що має бути оптимізовано».

Кожен сценарій ($\omega \in \Omega$) представлений унікальними обмеженнями ($f(x)$). Поєднання кожного сценарію $\omega_i$ з відповідними обмеженнями $f_i(x)$, вибір стохастичного критерію $g(x, \Omega)$, і відповідної цільової функції $z(x)$ повністю формують задачу SP:

$$z(x) = z(x, g(x, \Omega)) \tag{7}$$

$$\forall \omega \in \Omega : f_\omega(x) = 0 \tag{8}$$

$$\min(z) - ?$$

**Приклад 2.1** *Магазин продає деякий товар і щомісяця замовляє його у виробника. Якщо товару недостатньо, магазин несе витрати у розмірі 10\$ за одиницю товару. Якщо товару забагато, то магазин має витрачати 2\$ на одиницю товару за його утримання. Можливі значення попиту: 10, 20, або 30 одиниць товару. Заздалегідь невідомо, який в наступному місяці очікується попит. Нехай $x$ — змінна, яка означає кількість товару, яку необхідно придбати для наступного місяця. Яка модель допоможе мінімізувати витрати магазину в наступному місяці?*

Маємо:

$$\Omega = \{10, 20, 30\}$$

Означимо функцію витрат:

$$f_\omega(x) = 10 \cdot \max(0, x - \omega) + 2 \cdot \max(0, \omega - x)$$

$$x \in \mathbb{Z}^+$$

Використовуючи *мінімаксний* критерій, можемо означити цільову функцію:

$$z(x) = \max_{\omega \in \Omega}(f_\omega(x))$$

Відповідну цільову функцію необхідно мінімізувати:

$$\min(z)$$

Ця модель не використовує ймовірнісний розподіл сценаріїв, оскільки завжди мінімізує витрати в найгіршому випадку. Такий підхід є доволі корисним у заздалегідь програшних ситуаціях, проте в цьому випадку, інший критерій теж може бути застосований. Наведемо приклад, де мінімізація стосується очікуваного значення ризику. Нехай задано ймовірнісний розподіл станів природи:

$$p_\omega = (0.3, 0.1, 0.6)$$

Функція витрат:

$$f_\omega(x) = \begin{cases} 10(x - \omega), & x > \omega \\ 2(\omega - x), & x < \omega \\ 0 & x = \omega \end{cases}$$

Середні очікувані витрати:

$$z(x) = \mathbb{E}[f(x)] = \sum_{i=1}^{3} p_i f_i(x) = 0.3f_1(x) + 0.1f_2(x) + 0.6f_3(x)$$

$$\min(z) - ?$$

Дана модель мінімізує середні очікувані витрати шляхом використання математичного сподівання і його мінімізації. Такий критерій також має назву «критерій **Баєса-Лапласа**».

## 2.2 Стохастичні критерії

Стохастичне програмування тісно пов'язане з теорією вибору. Оцінка ризику є ключовим фактором у побудові задачі SP, та отриманні об'єктивного, сприятливого розв'язку. Неправильно оцінений ризик може призвести до величезних втрат, а разом і з тим — до різного типу наслідків (фінансові витрати, жертви цивільного населення, ризики забруднення довкілля, інші типи ризиків). Для того, щоб об'єктивно оцінити ситуацію та запропонувати стохастичну модель, існують *критерії*. Їхня роль полягає у математичному означенні ризику, для прийняття одного, зваженого рішення перед настанням сценарію.

**Означення 2.4** *Функція втрат $f(x, \omega)$ — це функція, яка описує втрати певної величини, за настання сценарію $\omega$*

Критерій поєднує функції втрат $f(x, \omega)$ на ймовірнісному просторі $\Omega$, дозволяючи отримати одну функцію, яку необхідно оптимізувати. Одні критерії потребують мінімізації, інші максимізації. Розглянемо основні стохастичні критерії, їхнє застосування, переваги і недоліки.

1. **Баєса-Лапласа (очікувані втрати)**

$$u = \mathbb{E}[f(x, \omega)] = \sum_{\omega} p(\omega) f(x, \omega)$$

$$\min\{u\}$$

   Критерій дозволяє оцінити очікувані втрати за всіх сценаріїв. Він також може бути означеним для неперервної випадкової величини $\omega$:

$$u = \int p(\omega) f(x, \omega) d\omega$$

   Основною перевагою критерію є лінійність, що свідчить про можливе формування задачі LP. З недоліків: за низької ймовірності даного сценарію втрати можуть бути занадто великими.

2. **Мінімаксний (критерій Вальда)**

$$u = \max_{\omega} f(x, \omega)$$

$$\min\{u\}$$

Критерій описує найбільші втрати при будь-якому сценарію. Він орієнтований на випадки, коли ціна ризику занадто висока, і найбільш песимістичний сценарій є досить ймовірним. Основний його недолік — це занадто велика обережність, яка в багатьох випадках не є об'єктивним фактором прийняття рішення (наприклад, якщо найбільш песимістичний сценарій має дуже низьку ймовірність, то, можливо, не варто використовувати цей критерій).

3. **Очікувана дисперсія**

$$u = \mathrm{Var}[f(x,\omega)] = \sum_{\omega} p(\omega) \left( f(x,\omega) - \sum_{\omega} p(\omega) f(x,\omega) \right)^2$$

$$\min\{u\}$$

Цей критерій забезпечує мінімальне відхилення від очікуваного значення, що дозволяє позбутися малоймовірних викидів, як у випадку критерію Баєса-Лапласа. Задача оптимізації, в якій використано цей критерій, лінійні обмеження та лінійні цільові функції (для кожного сценарію), є задачею *квадратичного програмування*. Також може бути записаний для неперервного випадку:

$$u = \int p(\omega) \left( f(x,\omega) - \mathbb{E}[f(x,\omega)] \right)^2 d\omega$$

4. **Вартість під ризиком (Value-at-Risk, VaR)**

$$u_\alpha = \mathrm{VaR}_\alpha[f(x,\omega)] = f_k(x,\omega), \text{ де } k = \min \left\{ j \in \{1,2,...,|\Omega|\} \;\middle|\; \sum_{i}^{j} p(i) \geq \alpha \right\}$$

$$\min\{u_\alpha\}$$

Value-at-Risk є деяким узагальненням мінімаксного критерію. Замість мінімізації найбільш песимістичного сценарію, мінімізується перша функція втрат, кумулятивна ймовірність якої перевищує певну межу $\alpha$. Кумулятивна ймовірність вираховується для відсортованих значень $f(x,\omega)$ від меншого до більшого. Тут $\alpha$ — це фактично квантиль, після якого

втрати ігноруються. Таким чином цей критерій є менш радикальним ніж мінімаксний, оскільки ігнорує високі малоймовірні втрати, і фокусується на більш реалістичних сценаріях. Вибір параметру $\alpha$ залежить від контексту задачі, а також розподілу сценаріїв. Для малої кількості сценаріїв (до 5-ти) цей критерій може не мати сенсу, і мінімаксний може бути кращим вибором.

5. **Умовна вартість під ризиком (Conditional Value-at-Risk, CVaR)**

$$u_\alpha = \mathrm{CVaR}_\alpha[f(x,\omega)] = \mathbb{E}\left[f(x,\omega) \;\middle|\; f(x,\omega) \geq \mathrm{VaR}_\alpha(f(x,\omega)\right]$$

$$\min\{u_\alpha\}$$

Даний критерій замість конкретного песимістичного сценарію розглядає очікуване значення всіх песимістичних сценаріїв, рівень яких встановлюється параметром $\alpha$. Добре використовувати, коли наявно багато песимістичних сценаріїв, за межею квантиля $\alpha$.

6. **Модальний**

$$u = f(x,\omega^*), \text{ де } \omega^* = \arg\max_\omega\{p(\omega)\}$$

$$\min\{u\}/\max\{u\}$$

Модальний критерій повертає найбільш ймовірний сценарій, який необхідно оптимізувати. В залежності від контексту, може бути як мінімізація (коли стосується ризику), так і максимізація (коли ризику не стосується). Основним недоліком є ігнорування всіх сценаріїв, що дуже рідко може бути коректним шляхом вирішення проблеми. Критерій може бути використаний для формулювання задачі LP.

7. **Критерій Гурвіца**

$$u = \alpha \max_\omega\{f(x,\omega)\} + (1-\alpha)\min_\omega\{f(x,\omega)\}$$

$$\min\{u\}$$

Критерій Гурвіца дозволяє балансувати між найгіршим і найкращим сценаріями. Ігнорується ймовірнісний розподіл, відповідно і найбільш ймовірні сценарії.

8. **Критерій Севіджа**

$$u = \max_{\omega} \{f(x, \omega) - \min\{f(x, \omega)\}\}$$

$$\min\{u\}$$

Даний критерій повертає максимальний «програш» за всіх сценаріїв, з метою його мінімізації. Поняття «програш» в даному випадку означає різницю між функцією втрат та найменшим можливим її значенням. Теж ігнорується ймовірнісний розподіл, що не завжди може адекватно описувати проблему.

9. Інші критерії

Існує ще багато інших критеріїв, які мають своє застосування в прикладних задачах теорії вибору та оптимізації. Більш того, кожна задача може потребувати дослідження на декількох критеріях, містити їхні комбінації, або ж вимагати нового специфічного критерію, який не є загально означеним в літературі. Наприклад, може бути сформований наступний критерій:

$$u = \lambda \text{CVaR}_{\alpha}[f(x, \omega)] + (1 - \lambda)\mathbb{E}[f(x, \omega)]$$

$$\min\{u\}$$

Такий критерій є поєднанням критеріїв CVaR (враховує найгірші втрати, але ігнорує очікувані) та Баєса-Лапласа (враховує очікувані, але ігнорує найгірші). Він може бути застосованим, коли пріоритет мають очікувані витрати, але від найгірших (припустимо 5%) сценаріїв теж треба захиститися. Аналогічно можна комбінувати існуючі критерії для пошуку нових.

## 2.3 Задача лінійного стохастичного програмування (SLP)

Як було показано раніше, задача стохастичного програмування SP обов'язково має визначатися через критерій вибору, який є частиною цільової функції (або представляє собою цільову функцію). Критерій може бути довільним, але єдиним лінійним критерієм, який враховує розподіл ймовірностей, є критерій Баєса-Лапласа. Щоб досягти лінійності у задачі SP, надалі використовуватимемо саме цей критерій. Враховуючи це, можемо сформулювати задачу стохастичного лінійного програмування.

Нехай дано простір сценаріїв $\Omega$. Для кожного $\omega \in \Omega$ сформуємо детерміновану задачу LP:

$$z_\omega = c_\omega^T x \tag{9}$$

$$Ax \leq b \tag{10}$$

$$x_i \geq 0$$

Зазначимо, що просто оптимізувати $z_\omega$ не достатньо, оскільки так втрачається сенс стохастичної моделі. Отримані функції $z_\omega$ мають стати частиною критерію, а отже, і деякої цільової функції $z$. В якості критерію буде використано мат. сподівання, а отримана функція $z$ буде лінійною.

$$z = \mathbb{E}(z_\omega) = \sum_{i=1}^{|\Omega|} p_i c_\omega^T x \tag{11}$$

$$Ax \leq b$$

$$\min(z) - ?$$

Отримана модель може описувати ситуацію, де ризикове рішення приймається зараз, а результат буде відомим у майбутньому. При цьому змінні рішення і обмеження єдині для всіх сценаріїв $\omega$. Отримана задача (11) є звичайною задачею LP.

Насправді, ця модель не враховує декілька важливих речей, наприклад унікальні обмеження для кожного сценарію. Також нерідко постає необхідність приймати рішення як до настання стохастичного сценарію, так і після. Модель (11) не надає такої можливості. Задачу можна модифікувати, щоб врахувати такі деталі.

По-перше, вважатимемо змінні $x$ рішенням, яке має бути прийнято миттєво, до настання випадкової події.

По-друге, розглядатимемо незалежні набори змінних для кожної реалізації $\omega$. Це рішення, які приймаються після настання сценарію. Їхні обмеження $A_\omega x_\omega \leq b_\omega$ матимуть вплив лише на змінні реалізації $\omega$ та змінні початкового рішення $x$.

Модель можна описати у наступному вигляді:

$$z(x) = c^T x + \mathbb{E}[Q(x, x_\omega)] \tag{12}$$

$$Ax \leq b \tag{13}$$

$$x \geq 0$$

$$\min(z) - ?$$

де $Q(x, x_\omega)$ — це розв'язок до конкретної реалізації $\omega$:

$$Q(x, x_\omega) = q_w^T x_\omega \tag{14}$$

$$B_\omega x + C_\omega x_\omega \leq h_\omega \tag{15}$$

$$x_\omega \geq 0$$

**Означення 2.5** *Вищезазначена модель називається* ***двоетапною задачею стохастичного лінійного програмування (two-stage SLP)***.

Кожна задача двоетапного стохастичного лінійного програмування має еквівалент у вигляді детермінованої задачі LP [6]. Для моделі (12) еквівалентна задача LP може бути записана наступним чином:

$$\begin{aligned}
\min\{z(x) = \; & c^T x_0 + p_1 q_1^T x_1 + p_2 q_2^T x_2 + \ldots + p_n q_n^T x_n\} \\
& Ax_0 \leq b \\
& B_1 x_0 + C_1 x_1 \leq h_1 \\
& B_2 x_0 + C_2 x_2 \leq h_2 \\
& \vdots \\
& B_n x_0 + C_n x_n \leq h_n \\
& x_0, x_1, \ldots, x_n \geq 0
\end{aligned} \tag{16}$$

# 3 Приклади і розв'язки задач

## 3.1 Проблема логістичних сполучень

**Приклад 3.1** *Гуманітарна організація готується до природної катастрофи у місцевому регіоні, в якому знаходиться 7 міст. Оцінюється, що ураган точно зруйнує міста A і B (ймовірність 75%); можливо ще зруйнує міста C, D та E (ймовірність 20%); або всі міста (ймовірність 5%). Люди, які проживають в цих містах, потребуватимуть допомоги протягом 10 днів, допоки будуть виконуватися роботи по відновленню міст. Розраховується надати наступні засоби першої допомоги: аптечки, сухі пайки, бутельовану воду, набори гігієни, набори дитячого харчування, та набори гігієни для людей похилого віку. Для їх забезпечення організація планує орендувати до 5-ти складських приміщень неподалік регіону, де будуть запаковані всі засоби для подальшої відправи в міста. У випадку, якщо засобів не буде вистачати, організація зможе доставити необхідну кількість з центрального складу, але за більшими тарифами. Засоби відправляються партіями по 100 штук одного засобу зі складу до міста. Враховуючи вищезазначені умови, та наведені нижче табличні дані, мінімізуйте очікувані витрати організації. Визначте, які складські приміщення має орендувати організація, яким чином вони мають бути заповнені, і яким чином налаштувати доставу партій засобів першої допомоги при кожному сценарію. Якими будуть витрати організації у кожному випадку?*

| Місто | Населення |
|---|---|
| A | 10000 |
| B | 12000 |
| C | 5000 |
| D | 20000 |
| E | 8500 |
| F | 9200 |
| G | 15400 |

Табл. 3: Населення міст

| Вікові категорії | | Потреби на 1 душу населення на 1 день | | | | | |
|---|---|---|---|---|---|---|---|
| Вікова група | Відсоток населення | Аптечки | Сухі пайки | Бутельована вода | Набори гігієни | Набори дитячого харчування | Набори гігієни для людей похилого віку |
| 0-4 | 5% | 0.02 | 0 | 2 | 0.05 | 0.5 | 0 |
| 5-14 | 17.5% | 0.04 | 0.3 | 3 | 0.08 | 0 | 0 |
| 15-64 | 65% | 0.05 | 0.33 | 4 | 0.1 | 0 | 0 |
| 65+ | 12.5% | 0.07 | 0.3 | 3.5 | 0 | 0 | 0.2 |

Табл. 4: Розподіл потреб по віковим категоріям

| Склади | Ціна перевезення однієї партії до міста ($) | | | | | | | Ціна оренди на 1 день ($) | Ємність (одиниці засобів) |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | | |
| 1 | 50 | 40 | 35 | 35 | 25 | 30 | 45 | 1000 | 800 000 |
| 2 | 45 | 45 | 30 | 40 | 50 | 55 | 50 | 700 | 600 000 |
| 3 | 47 | 49 | 58 | 60 | 35 | 40 | 35 | 900 | 720 000 |
| 4 | 25 | 40 | 60 | 70 | 65 | 40 | 55 | 350 | 320 000 |
| 5 | 55 | 35 | 40 | 45 | 48 | 35 | 65 | 490 | 440 000 |
| Центральний | 250 | 200 | 190 | 210 | 280 | 275 | 290 | - | - |

Табл. 5: Інформація про складські приміщення

| | |
|---|---|
| Аптечка | 15$ |
| Сухий пайок | 10$ |
| Вода | 0.5$ |
| Набір гігієни | 5$ |
| Набір дитячого харчування | 7$ |
| Набір гігієни для людей похилого віку | 12.5$ |

Табл. 6: Вартості засобів першої допомоги

Для початку введемо основні позначення:

- Індекси
  - $i$ - індекс засобу першої допомоги (1-6).
  - $j$ - індекс складу (1-5).
  - $j^*$ - індекс складу разом з центральним (1-6).
  - $s$ - індекс міста (1-7).
  - $k$ - сценарії (1-3).
- Змінні
  - $x_{ij}$ - кількість засобу $i$ на складі $j$.
  - $w_j$ - булеве значення. 1 якщо треба орендувати складське приміщення, інакше 0.
  - $y^k_{ij^*s}$ - кількість партій засобу $i$, які треба доставити зі складу $j^*$ до міста $s$ за сценарію $k$.
- Параметри
  - $S_s$ - населення міста $s$.
  - $f_j$ - ціна оренди складу $j$.
  - $C_j$ - ємність складу $j$.
  - $c_{j^*s}$ - вартість достави зі складу $j^*$ до міста $s$.
  - $p_i$ - вартість одиниці засобу $i$.
  - $E_i$ - потреби засобу $i$ на душу населення. Оцінюється як математичне сподівання згідно з табличними даними.

Простором сценаріїв буде 3 випадки руйнувань:

$$\Omega = \{\omega_1, \omega_2, \omega_3\}$$

$$\begin{aligned} \omega_1 &= \{1, 2\} & p_1 &= 0.75 \\ \omega_2 &= \{1, 2, 3, 4, 5\} & p_2 &= 0.2 \\ \omega_3 &= \{1, 2, 3, 4, 5, 6, 7\} & p_3 &= 0.05 \end{aligned}$$

Рішення приймаються у 2 етапи. Перший етап включає в себе оренду складських приміщень, закупівлю необхідної кількості засобів першої допомоги та розподіл між складами. Другий етап — це проблема достави засобів до міст, а також додаткові витрати в разі відсутності достатньої кількості засобів на складах. Всі змінні мають бути невід'ємні, в подальшому ігноруватимемо запис цього обмеження.

Сформуємо перший етап:

$$z = \sum_j w_j \sum_i p_i x_{ij} + 10 \sum_j w_j f_j + \mathbb{E}[Q_k] \tag{17}$$

$$\sum_i x_{ij} \leq C_j \tag{18}$$

$$\min(z) - ?$$

Можна помітити, що перший доданок у цільовій функції не є лінійним $\sum_j w_j \sum_i p_i x_{ij}$, бо набори змінних $x_{ij} w_j$ утворюють добуток. Цю задачу можна розглядати як задачу квадратичного програмування, або ж, в силу булевих значень змінних $w_j$, як сукупність задач LP, де змінні приймають конкретні булеві значення. Далі буде використано саме цей підхід.

Тепер сформуємо другий етап задачі:

$$Q_k = \sum_{s \in \omega_k} \sum_i \left( \sum_j w_j c_{js} y^k_{ijs} + c_{6s} y^k_{i6s} + 100 p_i y^k_{i6s} \right) \tag{19}$$

$$100 \sum_{j^*} y^k_{ij^*s} \geq 10 \cdot S_s E_i \tag{20}$$

$$x_{ij} - 100 \sum_{s \in \omega_k} y^k_{ijs} \geq 0 \tag{21}$$

У першому виразі обмежень фігурує параметр $E_i$. Його можна обрахувати як математичне сподівання засобу $i$. Наприклад для аптечки $E_1 = 0.05 \cdot 0.02 + 0.175 \cdot 0.04 + 0.65 \cdot 0.05 + 0.125 \cdot 0.07 = 0.04925$. Тоді на, припустимо, 10000 мешканців очікувані потреби складатимуть $10000 E_1 = 492.5$.

Вираз (19) включає в себе витрати по трьом категоріям:

1. Транспортні витрати для кожного засобу від кожного складу (якщо він орендований) до кожного міста.

2. Транспортні витрати для кожного засобу від центрального складу до кожного міста.

3. Вартість кожної партії засобів, завезених з центрального складу.

Обмеження (20) зобов'язує задовольнити потреби кожного міста у кожному засобі, враховуючи той факт, що достава проходить партіями по 100 штук.

Обмеження (21) не дозволяє транспортувати більшу кількість засобів, аніж зберігається на складі.

Тепер розв'яжемо задачу у Python. Як і зазначалося раніше, розглянемо всі варіанти для булевих змінних, відповідно матимемо 32 задачі. Відповідь оберемо ту, цільова функція якої має найменше значення. Усі табличні дані винесемо у **csv** файли.

Імпортуємо бібліотеки:

```
import scipy.optimize as opt
from itertools import product
import pandas as pd
import numpy as np
```

Підготуємо вхідні параметри:

- Населення міст

```
cities_raw = pd.read_csv('data/city_data.csv')
cities = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
cities_population = cities_raw['population'].to_numpy()
```

- Очікувані потреби засобів першої допомоги на душу населення

```
supply_needs = pd.read_csv('data/supply_needs.csv')
supply_needs['Percentage'] = supply_needs['Percentage']/100

supply_types = ['first-aid-kits','dry-rations','bottled-water','hygiene-
    kits','child-food-kits','elderly-hygiene-kits']
supply_types_estimated_needs = np.array([
    supply_needs['Percentage'].transpose() @ supply_needs[k]
    for k in supply_types
])
```

- Вартості одиниць засобів

```
supply_prices_raw = pd.read_csv('data/supply_prices.csv')
supply_prices = np.array([
    supply_prices_raw[supply_prices_raw['supply'] == s]['price'].
       to_numpy()[0]
    for s in supply_types
])
```

- Дані складських приміщень

```
warehouse_data_raw = pd.read_csv('data/warehouses_data.csv')

rentable_warehouse_data = warehouse_data_raw[warehouse_data_raw['
    warehouse'] != '6-central']
warehouse_rent_prices = rentable_warehouse_data['rent'].to_numpy()
warehouse_capacities = rentable_warehouse_data['capacity'].to_numpy()
delivery_prices = warehouse_data_raw[cities].to_numpy()
```

- Решта параметрів

```
T = 10 # Repair time
N = 100 # Supplies in 1 portion
K = 3 # Number of scenarios

# Scenario probabilities set
Omega = [np.arange(0,2), np.arange(0,5), np.arange(0,7)]
p = [0.75, 0.2, 0.05]

total_supply_types = len(supply_types)
total_warehouses = len(rentable_warehouse_data)
```

Сформуємо всі можливі варіанти комбінацій булевих змінних:

```
warehouse_permutations = np.array(
    list(product([0, 1], repeat=total_warehouses))
)
```

На даному етапі усі дані завантажені, і можна приступити до реалізації самого алгоритму. Спочатку сформуємо цільову функцію у вигляді вектора всіх змінних, далі сформуємо обмеження. Усі розв'язки запишемо у змінну **solutions** для подальшого дослідження.

```
solutions = []
for warehouse_setup in warehouse_permutations:
    # --- PREPARE VARIABLES ---
    # Indicies of the warehouses that are rented
```

```
    w = np.where(warehouse_setup == 1)[0]
    w_with_central = np.append(w, 5)

    # All supply variables
    x_labels = [(i,j)
        for i in range(total_supply_types)
        for j in w
    ]

    # All delivery variables
    y_labels = [(k,i,j,s)
        for k in range(K)
        for i in range(total_supply_types)
        for j in w_with_central
        for s in Omega[k]
    ]

    # Variables-indicies mapper
    label_to_index = {
        **{('x', i, j):
            idx for idx, (i, j) in enumerate(x_labels)},
        **{('y', k, i, j, s):
            idx + len(x_labels) for idx, (k, i, j, s) in enumerate(y_labels)
               }
    }
    index_to_label = {
        idx: ('x', i, j) for idx, (i, j) in enumerate(x_labels)
    }
    index_to_label.update({
        idx + len(x_labels): ('y', k, i, j, s)
            for idx, (k, i, j, s) in enumerate(y_labels)
    })

    # Objective function
    total_vars = len(label_to_index)
    c = np.zeros(total_vars)

    # --- OBJECTIVE FUNCTION ---
    # Total rental price to be included after the solution
    total_rental_price = T * np.sum(warehouse_rent_prices[w])

    # First stage. Initial supplies cost
    for (i, j) in x_labels:
        idx = label_to_index[('x', i, j)]
        c[idx] += supply_prices[i]

```

```
    # Second stage. Delivery cost
    for (k, i, j, s) in y_labels:
        idx = label_to_index[('y', k, i, j, s)]
        if j == 5:
            c[idx] += p[k] * (delivery_prices[j][i] + N * supply_prices[i])
        else:
            c[idx] += p[k] * delivery_prices[j][i]

    # --- CONSTRAINTS ---
    A_ub = []
    b_ub = []

    # First stage. Capacity constraints
    for j in w:
        row = np.zeros(total_vars)
        for i in range(total_supply_types):
            idx = label_to_index[('x', i, j)]
            row[idx] = 1
        A_ub.append(row)
        b_ub.append(warehouse_capacities[j])

    # Second stage. Fullfillment of needs
    for k in range(K):
        for s in Omega[k]:
            for i in range(total_supply_types):
                row = np.zeros(total_vars)
                for j in w_with_central:
                    idx = label_to_index[('y', k, i, j, s)]
                    row[idx] = N
                A_ub.append(-row) # Invert due to linprog
                b_ub.append(
                    -T * supply_types_estimated_needs[i] * cities_population
                        [s]
                ) # Invert due to linprog

    # Second stage. Stocked supplies restriction
    for k in range(K):
        for i in range(total_supply_types):
            for j in w:
                row = np.zeros(total_vars)
                idx = label_to_index[('x', i, j)]
                row[idx] = -1
                for s in Omega[k]:
                    idx = label_to_index[('y', k, i, j, s)]
                    row[idx] = N
                A_ub.append(row)
```

```
                b_ub.append(0)


    if (np.array(A_ub).shape[0] == 0):
        A_ub = np.zeros((1, total_vars))

    if (np.array(b_ub).shape[0] == 0):
        b_ub = np.zeros(1)

    res = opt.linprog(
        c, A_ub=A_ub, b_ub=b_ub, method='highs', integrality=1
    )
    solutions.append({
        'status': res.success,
        'fun': res.fun + total_rental_price,
        'w': w,
        'x': { index_to_label[i]: val for i, val in enumerate(res.x) }
    })
```

Даний цикл згенерує список всіх розв'язків. Знайдемо мінімальний серед всіх:

```
best_func = np.inf
best_sol = None

for i, sol in enumerate(solutions):
    if (sol['fun'] < best_func):
        best_func = sol['fun']
        best_sol = sol
```

Для аналізу розв'язку спочатку поглянемо на булеві значення:

```
best_sol['w']
# array([0, 1, 3], dtype=int64)
```

Відповідь на одне з запитань маємо: **організація має орендувати перший, другий та четвертий склади**.

Проаналізуємо змінні рішення першого етапу:

```
first_stage_raw = {k: v for k, v in best_sol['x'].items() if k[0] == 'x'}
first_stage_decision = pd.DataFrame()
for ((_, i, j), v) in first_stage_raw.items():
    if (v == 0):
        continue
    first_stage_decision = pd.concat([first_stage_decision, pd.DataFrame({
        'supply': [i],
        'supply_type': [supply_types[i]],
```

```
        'warehouse': [j],
        'amount': [v]
    })], ignore_index=True)

first_stage_decision
```

| supply | supply_type | warehouse | amount |
|---|---|---|---|
| 0 | first-aid-kits | 3 | 11000 |
| 1 | dry-rations | 3 | 67100 |
| 2 | bottled-water | 0 | 771000 |
| 2 | bottled-water | 1 | 600000 |
| 3 | hygiene-kits | 0 | 18000 |
| 4 | child-food-kits | 0 | 5500 |
| 5 | elderly-hygiene-kits | 0 | 5500 |

Звідси отримуємо відповідь на друге запитання. **Організація має придбати засоби першої допомоги і розподілити їх наступним чином**:

- 11,000 аптечок. Усі в четвертому складі.
- 67,100 сухих пайків. с в етвертому склад.
- 1,371,000 бутлів води. 771,000 в першому складі, решта у другому.
- 18,000 наборів гігієни. Усі в першому складі.
- 5,500 наборів дитячого харчування. Усі в першому складі.
- 5,500 наборів гігієни для людей похилого віку. Усі в першому складі.

Означимо допоміжну функцію для отримання результатів для кожного сценарію:

```
def get_scenario_df(sol, k):
    x_vars = { key: v
        for key, v in sol['x'].items()
        if key[0] == 'x'
    }
    y_vars = { key: v
        for key, v in sol['x'].items()
        if key[0] == 'y' and key[1] == k
```

```
    }

    x_values = np.sum(v * supply_prices[i]
        for (var, i, j), v in x_vars.items()
    )
    rental_price = T * np.sum(warehouse_rent_prices[sol['w']])
    y_values = np.sum(v * delivery_prices[j][i]
        for (var, scenario, i, j, s), v in y_vars.items()
        if j != 5
    )
    y_values_central = np.sum(
        v * (delivery_prices[j][i] + N * supply_prices[i])
        for (var, scenario, i, j, s), v in y_vars.items()
        if j == 5
    )
    total_value = x_values + rental_price + y_values + y_values_central

    scenario_raw = { key: v
        for key, v in sol['x'].items()
        if key[0] == 'y' and key[1] == k
    }
    scenario_decision = pd.DataFrame()
    for ((_, k, i, j, s), v) in scenario_raw.items():
        if (v == 0):
            continue
        scenario_decision = pd.concat([scenario_decision, pd.DataFrame({
            'supply': [i],
            'supply_type': [supply_types[i]],
            'warehouse': [j],
            'city': [s],
            'amount': [v]
        })], ignore_index=True)

    return (scenario_decision, total_value, x_values + rental_price)
```

Функція повертає значення змінних конкретного сценарію, загальні витрати для даного сценарію, а також витрати на першому етапі прийняття рішення.

- Для першого сценарію маємо:

```
(scenario1_decision, value1, first_stage_value) = get_scenario_df(
    best_sol, 0)
print(value1, first_stage_value)
# 2030195.0 1739250.0
```

Витрати становлять **2 030 195$**, якщо справджується **перший** сценарій.

- Для другого:

```
(scenario2_decision, value2, first_stage_value) = get_scenario_df(
    best_sol, 1)
print(value2, first_stage_value)
# 5739740.0 1739250.0
```

Витрати становлять **5 739 740$**, якщо справджується **другий** сценарій.

- Для третього:

```
(scenario3_decision, value3, first_stage_value) = get_scenario_df(
    best_sol, 2)
print(value3, first_stage_value)
# 9313900.0 1739250.0
```

Витрати становлять **9 313 900$**, якщо справджується **третій** сценарій.

Також можемо перевірити коректність отриманих значень. Від загальної суми витрат віднімемо суму витрат на першому етапі. Далі, згідно з означенням двоетапної задачі SLP, обрахуємо очікувані витрати на другому етапі, і звіримо відповідь:

$$2,030,195 - 1,739,250 = 290,945$$

$$5,739,740 - 1,739,250 = 4,000,490$$

$$9,313,900 - 1,739,250 = 7,574,650$$

$$1,739,250 + 290,945 \cdot 0.75 + 4,000,490 \cdot 0.2 + 7,574,650 \cdot 0.05 = 3,316,289.25$$

Звіримо отримане значення з очікуваним:

```
best_sol['fun']
# 3136289.25
```

Значення однакові. Отже, інтерпретація задачі та її математичне формулювання відповідають дійсності. Зазначимо, що чим менш ймовірний сценарій, тим більше становитимуть витрати. При цьому найбільш ймовірний сценарій потребує витрат менше, аніж очікувані витрати, навіть за умов мінімізації. Саме ця різниця і є запобіжним фактором у випадку, коли найбільш ймовірний сценарій не справджується.

На завершення, наведемо розв'язок у змінних $y$ для першого сценарію:

```
scenario1_decision
```

| supply | supply_type | warehouse | city | amount |
|---|---|---|---|---|
| 0 | first-aid-kits | 3 | 0 | 50 |
| 0 | first-aid-kits | 3 | 1 | 60 |
| 1 | dry-rations | 3 | 0 | 305 |
| 1 | dry-rations | 3 | 1 | 366 |
| 2 | bottled-water | 0 | 0 | 2058 |
| 2 | bottled-water | 1 | 0 | 1605 |
| 2 | bottled-water | 1 | 1 | 4395 |
| 3 | hygiene-kits | 0 | 0 | 82 |
| 3 | hygiene-kits | 0 | 1 | 98 |
| 4 | child-food-kits | 0 | 0 | 25 |
| 4 | child-food-kits | 0 | 1 | 30 |
| 5 | elderly-hygiene-kits | 0 | 0 | 25 |
| 5 | elderly-hygiene-kits | 0 | 1 | 30 |

- З четвертого складу буде відправлено **50 партій аптечок** до міста A та **60 партій** до міста B.
- З четвертого складу буде відправлено **305 партій сухих пайків** до міста A та **366 партій** до міста B.
- З першого складу буде відправлено **2058 партій бутельованої води** до міста A. З другого складу буде відправлено **1605 партій** до міста A і **4395 партій** до міста B.
- З першого складу **82 партії наборів гігієни** будуть відправлені до міста A і **98** до міста B.
- З першого складу буде відправлено **25 партій наборів дитячого харчування** до міста A і **30 партій** до міста B.
- З першого складу буде відправлено **25 партій наборів гігієни для людей похилого віку** до міста A і **30 партій** до міста B.

Аналогічно можна отримати результати для другого і третього сценаріїв:

```
scenario2_decision
```

| supply | supply_type | warehouse | city | amount |
|---|---|---|---|---|
| 0 | first-aid-kits | 3 | 0 | 50 |
| 0 | first-aid-kits | 3 | 2 | 25 |
| 0 | first-aid-kits | 3 | 4 | 35 |
| 0 | first-aid-kits | 5 | 1 | 60 |
| 0 | first-aid-kits | 5 | 3 | 99 |
| 0 | first-aid-kits | 5 | 4 | 7 |
| 1 | dry-rations | 3 | 1 | 62 |
| 1 | dry-rations | 3 | 3 | 609 |
| 1 | dry-rations | 5 | 0 | 305 |
| 1 | dry-rations | 5 | 1 | 304 |
| 1 | dry-rations | 5 | 2 | 153 |
| 1 | dry-rations | 5 | 4 | 259 |
| 2 | bottled-water | 0 | 3 | 5720 |
| 2 | bottled-water | 0 | 4 | 1990 |
| 2 | bottled-water | 1 | 1 | 4395 |
| 2 | bottled-water | 1 | 3 | 1605 |
| 2 | bottled-water | 5 | 0 | 3663 |
| 2 | bottled-water | 5 | 2 | 1832 |
| 2 | bottled-water | 5 | 4 | 1124 |
| 3 | hygiene-kits | 0 | 0 | 82 |
| 3 | hygiene-kits | 0 | 2 | 41 |
| 3 | hygiene-kits | 0 | 3 | 57 |
| 3 | hygiene-kits | 5 | 1 | 98 |
| 3 | hygiene-kits | 5 | 3 | 106 |
| 3 | hygiene-kits | 5 | 4 | 70 |
| 4 | child-food-kits | 0 | 0 | 20 |
| 4 | child-food-kits | 0 | 2 | 13 |
| 4 | child-food-kits | 0 | 4 | 22 |
| 4 | child-food-kits | 5 | 0 | 5 |
| 4 | child-food-kits | 5 | 1 | 30 |
| 4 | child-food-kits | 5 | 3 | 50 |
| 5 | elderly-hygiene-kits | 0 | 0 | 25 |
| 5 | elderly-hygiene-kits | 0 | 1 | 30 |
| 5 | elderly-hygiene-kits | 5 | 2 | 13 |
| 5 | elderly-hygiene-kits | 5 | 3 | 50 |
| 5 | elderly-hygiene-kits | 5 | 4 | 22 |

```
scenario3_decision
```

| supply | supply_type | warehouse | city | amount |
|---|---|---|---|---|
| 0 | first-aid-kits | 3 | 1 | 60 |
| 0 | first-aid-kits | 3 | 3 | 50 |
| 0 | first-aid-kits | 5 | 0 | 50 |
| 0 | first-aid-kits | 5 | 2 | 25 |
| 0 | first-aid-kits | 5 | 3 | 49 |
| 0 | first-aid-kits | 5 | 4 | 42 |
| 0 | first-aid-kits | 5 | 5 | 46 |
| 0 | first-aid-kits | 5 | 6 | 76 |
| 1 | dry-rations | 3 | 3 | 609 |
| 1 | dry-rations | 3 | 5 | 62 |
| 1 | dry-rations | 5 | 0 | 305 |
| 1 | dry-rations | 5 | 1 | 366 |
| 1 | dry-rations | 5 | 2 | 153 |
| 1 | dry-rations | 5 | 4 | 259 |
| 1 | dry-rations | 5 | 5 | 219 |
| 1 | dry-rations | 5 | 6 | 469 |
| 2 | bottled-water | 0 | 3 | 7325 |
| 2 | bottled-water | 0 | 5 | 385 |
| 2 | bottled-water | 1 | 0 | 1183 |
| 2 | bottled-water | 1 | 2 | 1832 |
| 2 | bottled-water | 1 | 5 | 2985 |
| 2 | bottled-water | 5 | 0 | 2480 |
| 2 | bottled-water | 5 | 1 | 4395 |
| 2 | bottled-water | 5 | 4 | 3114 |
| 2 | bottled-water | 5 | 6 | 5641 |
| 3 | hygiene-kits | 0 | 1 | 54 |
| 3 | hygiene-kits | 0 | 6 | 126 |
| 3 | hygiene-kits | 5 | 0 | 82 |
| 3 | hygiene-kits | 5 | 1 | 44 |
| 3 | hygiene-kits | 5 | 2 | 41 |
| 3 | hygiene-kits | 5 | 3 | 163 |
| 3 | hygiene-kits | 5 | 4 | 70 |
| 3 | hygiene-kits | 5 | 5 | 75 |
| 4 | child-food-kits | 0 | 2 | 10 |
| 4 | child-food-kits | 0 | 4 | 22 |
| 4 | child-food-kits | 0 | 5 | 23 |
| 4 | child-food-kits | 5 | 0 | 25 |
| 4 | child-food-kits | 5 | 1 | 30 |
| 4 | child-food-kits | 5 | 2 | 3 |
| 4 | child-food-kits | 5 | 3 | 50 |
| 4 | child-food-kits | 5 | 6 | 39 |
| 5 | elderly-hygiene-kits | 0 | 0 | 25 |
| 5 | elderly-hygiene-kits | 0 | 2 | 7 |
| 5 | elderly-hygiene-kits | 0 | 5 | 23 |
| 5 | elderly-hygiene-kits | 5 | 1 | 30 |
| 5 | elderly-hygiene-kits | 5 | 2 | 6 |
| 5 | elderly-hygiene-kits | 5 | 3 | 50 |
| 5 | elderly-hygiene-kits | 5 | 4 | 22 |
| 5 | elderly-hygiene-kits | 5 | 6 | 39 |

На даному етапі можна вважати задачу розв'язаною.

# Висновки

У цій роботі було досліджено тему задач оптимізації за умов невизначеності. Був наведений логічний теоретичний ланцюжок побудови задачі стохастичного лінійного програмування, який складається з теорії детермінованого лінійного програмування, використання теорії вибору у виборі стохастичного критерію, та зведенні стохастичної моделі до детерміністичного аналогу.

Робота містила 2 основні частини:

1. Постановку задачі лінійного програмування, методи її розв'язку, алгоритм цілочисельного програмування B&B та приклади реалізації.

2. Теорію оптимізації за умов невизначеності, вибір критерію оптимізації, теорію лінійної стохастичної моделі та детерміністичного аналогу для двоетапної моделі.

На основі розглянутої теорії було побудовано математичну модель проблеми логістичних сполучень під час природної катастрофи та показано її практичне застосування з отриманням конкретних результатів для кожного сценарію розвитку подій. Також був наведений код алгоритму розв'язку задачі з відповідними поясненнями деталей реалізації. Отримані результати підтверджують, що стохастичне лінійне програмування є потужним і структурним інструментом для оптимізації в задачах, де наявний фактор невизначеності.

Майбутні дослідження можуть містити моделі нелінійного програмування, стохастичного програмування з нелінійними критеріями, використання універсальних підходів для розв'язку будь-яких типів задач оптимізації, а також поєднання стохастичних задач оптимізації з іншими галузями, зокрема машинного навчання та нейромереж.

# Література


[1] Stormy Attaway. *Matlab: A Practical Introduction to Programming and Problem Solving*. Elsevier, 2009. URL: https://dl.icdst.org/pdfs/files3/375826d75004252629466f2157b47197.pdf.

[2] ROBERT E. BIXBY. Solving real-world linear programs:a decade and more of progress. *Operations Research*, 2002. URL: https://pubsonline.informs.org/doi/epdf/10.1287/opre.50.1.3.17780.

[3] GEORGE B. DANTZIG. Linear programming. *Operations Research*, 2002. URL: https://courses.cs.duke.edu/spring07/cps296.2/papers/LinearProgramming_article.pdf.

[4] Joe Holley. Vanguard mathematician george dantzig dies. *Washington Post*, 2005. URL: https://supernet.isenberg.umass.edu/photos/gdobit.html.

[5] David F. Shanno Irvin J. Lustig, Roy E. Marsten. Interior point methods for linear programming: Computational state of the art. 1994. URL: https://pubsonline.informs.org/doi/pdf/10.1287/ijoc.6.1.1.

[6] François Louveaux John R. Birge. *Introduction to Stochastic Programming*. Springer, 2011. URL: https://books.google.com.ua/books?hl=ru&lr=&id=Vp0Bp8kjPxUC&oi=fnd&pg=PR1&ots=q7BN-VdbDv&sig=GwlD8aoOSan7P5Wv6WoeASuSls8&redir_esc=y#v=onepage&q&f=false.

[7] Narendra Karmarkar. A new polynomial-time algorithm for linear programming-ii. *AT&T Bell Laboratories*, 1984. URL: https://www.researchgate.net/publication/228057795_A_New_Polynomial-Time_Algorithm_for_Linear_Programming-II.

[8] Sasirekha Vinjamuri Saul I. Gass. Cycling in linear programming problems. *ORSA Journal on Computing*, 2002. URL: /https://web.ist.utl.pt/~ist11038/CD_Casquilho/LP2004Comp&OR_GassVinjamuri.pdf.