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

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: **«ПСЕВДО-ОБЕРНЕНА МАТРИЦЯ ТА ЇЇ ЗАСТОСУВАННЯ У ПОБУДОВІ ОПТИМАЛЬНОГО ПОРТФЕЛЯ ІНВЕСТОРА»**

Виконала: студентка 4-го року навчання
освітньої програми «Прикладна математика»,
спеціальності 113 Прикладна математика

Бурдим Анастасія Андріївна

Керівник: Щестюк Н. Ю.
кандидат фіз.-мат. наук, ст. викладач

Рецензент:

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК_____
(підпис)

«_____» _____ 20___р.

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
студентці 4-го курсу, факультету інформатики
Бурдим Анастасії Андріївні

Тема: «Псевдо-обернена матриця та ii застосування у побудові оптимального портфеля інвестора»

Зміст кваліфікаційної роботи:

Анотація

1. Вступ
2. Огляд основних означень та тверджень, що пов’язані з псевдо-оберненою матрицею
3. Складання алгоритму і написання програми для побудови оптимального портфелю Марковиця. Використання 2 модифікацій
4. Застовування псевдо-оберненої матриці для випадку сингулярності. Складання алгоритмиу. Ілюстрація на прикладах. Написання програми
5. Застосування доробки до реальних фінансових даних і складання оптимального портфеля.
6. Список літератури

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

Графік підготовки кваліфікаційної роботи до захисту

Графік узгоджено «_____» _____ 2024р.

№ з/п	Перелік робіт	Термін виконання етапу	Підпис наукового керівника	Дата ознайомлення наукового керівника	Примітка
1.	Отримання теми кваліфікаційної роботи.	11.10.2024			
2.	Ознайомлення з темою кваліфікаційної роботи.	16.10.2024			
3.	Розробка плану та структури роботи.	16.10.2024			
4.	Робота з науковою літературою, опис основних означень. Написання вступу та анотації.	7.11.2024			
5.	Написання програм.	15.12.2024			
6.	Робота над текстовим оформленням та практичним застосуванням.	18.02.2025			
7.	Виступ на XIII Всеукраїнській науковій конференції молодих математиків.	09.05.2025			
8.	Попередній захист кваліфікаційної роботи.	22.05.2025			
9.	Захист кваліфікаційної роботи.	06.06.2025			

Науковий керівник _____
(ПІБ)

Виконавець кваліфікаційної роботи _____
(ПІБ)

Зміст

Анотація **3**

1 Вступ **4**

2 Огляд основних означень та теоретичних положень, що пов'язані з псевдо-оберненою матрицею **6**

2.1 Основні означення лінійної алгебри та псевдо-обернених матриць 6

2.2 Методи обчислення псевдо-оберненої матриці 11

2.3 Ілюстративні приклади знаходження матриці Мура–Пенроуза використовуючи запропонований алгоритм 12

3 Алгоритм і реалізація програми для побудови оптимального портфеля Марковиця для сингулярних даних **16**

3.1 Теоретичні основи портфеля Марковиця 16

3.2 Розробка алгоритму побудови портфеля Марковіца(2 модифікації) 19

3.3 Ілюстративні приклади оптимізації портфеля Марковіца для двох формулювань 22

4 Практичне застосування для реальних фінансових даних **30**

4.1 Збір та попередня обробка фінансових даних 30

4.2 Оцінка параметрів 32

4.3 Оптимізація портфеля у базовому випадку та у випадку сингулярності 34

5 Стохастичне представлення очікуваної дохідності та дисперсії **40**

5.1 Теорема про стохастичне представлення обраних величин 40

5.2 Проведення симуляції V^{+}_{GMV} та R^{+}_{GMV} на реальних фінансових даних 41

Додаток А. Програмний код **45**

Список літератури **55**

Анотація

Метою цієї кваліфікаційної роботи є дослідження теоретичних і практичних аспектів використання псевдо-обернених матриць (матриць Мура-Пенроуза) у побудові оптимального портфеля інвестора Марковіца. У роботі розглядаються основні означення та властивості псевдо-обернених матриць, а також методи їх обчислення. Особливу увагу приділено розробці алгоритму побудови оптимального портфеля Марковиця, включаючи реалізацію 2 модифікацій формулювання задачі (класичний варіант та додаткове формулювання).

У процесі роботи були досліджені не тальки базові випадки, а і ситуації, коли матриця коваріацій була сингулярною (або майже сингулярною), що може стати проблемою при роботі з реальними фінансовими даними (у разі нестачі історичних даних чи високої кореляції між активами). У такому разі було розроблено алгоритм із застосуванням псевдо-обернених матриць.
Практичну частину роботи присвячено застосуванню розроблених алгоритмів до реальних фінансових даних для чотирьох компаній (Amazon, Netflix, Tesla і Apple) за період одного року. На основі історичних даних цих компаній було проведено кілька варіантів оптимізацій (у випадку додатньовизначеності матриці коваріацій, з додаванням обмеження на додатність ваг оптимального портфеля та у випадку сингулярності коваріаційної матриці). Отримані результати було проаналізовано, і запропоновано рекомендації щодо використання псевдо-обернених матриць на практиці. Також додатково було проведено симуляцію аби дослідити стохастичне представлення очікуваної дохідності та дисперсії портфеля у випадку сингулярності коваріаційної матриці.

1 Вступ

Теорія оптимального портфеля є однією з основ сучасної фінансової математики, започаткованою Г. Марковіцем у 1952 році[1]. Ця теорія має на меті досягнення найкращого співвідношення між очікуваною прибутковістю та ризиком портфеля. Інвестори можуть мінімізувати ризик за заданого рівня очікуваного доходу, що має надзвичайне значення у реальних умовах фінансового ринку [1, 2].

Проте у випадках, коли матриця коваріації активів є сингулярною, класичні методи аналізу не можуть бути використані. При роботі з реальними фінансовими даними таке травляється за високої коваріації між активами чи при нестачі історичної інформації. У таких ситуаціях застосування псевдообернених матриць, наприклад, матриць Мура-Пенроуза [3], є ефективним рішенням , адже класична інверсія матриці є неможливою. Це дозволяє не лише подолати проблеми, пов'язані з обмеженістю даних, але й удосконалювати портфельний аналіз в умовах високої вимірності [2].

Оскільки фінансові ринки наразі характеризуються нестабільністю та великою кількістю фінансових інструментів, що часто веде до сильно корельованих або навіть лінійно залежних активів, це дослідження фокусується на методах оптимізації портфлеля саме у випадку сингулярних або майже сингулярних матриць коваріації.

Крім того, сучасні інвестиційні платформи й алгоритмічні системи потребують стабільних обчислювальних методів, які можуть працювати навіть у випадках недостатньої кількості даних або у високовимірних просторах, видаючи коректні результати для інвесторів. З огляду на це, методи на основі псевдообернених матриць, зокрема оберненої за Муром-Пенроузом, стають все більш затребуваними як у наукових дослідженнях, так і в реальних прикладних задачах [12].

Таким чином, дослідження ефективного застосування псевдо-обернених матриць до побудови оптимального портфеля інвестора Марковіца є актуальним як у теоретичному, так і в практичному аспектах.

Метою даної кваліфікаційної роботи є дослідження властивостей псевдообернених матриць та їх застосування у побудові оптимального портфеля

інвестора Марковіца для реальних фінансових даних.

Для досягнення мети поставлено такі завдання:

1. Огляд теоретичних основ псевдо-обернених матриць(матриць Мура-Пенроуза) та їх властивостей. Огляд допоміжних означень, пов'язаних з матрицею Мура-Пенроуза.

2. Розробка та реалізація алгоритму побудови оптимального портфеля Марковіца (використання двох модифікацій формулювання - класичного та додаткового).

3. Модифікація алгоритму для випадків сингулярності з використанням псевдо-обернених матриць.

4. Практичне застосування алгоритмів до реальних фінансових даних та аналіз отриманих результатів.

5. Стохастичне представлення очікуваної дохідності та дисперсії оптимального портфеля при сингулярній матриці коваріації . Проведення симуляції для аналізу розподілу на основі реальних фінансових даниї.

Робота базується на фундаментальних результатах Г. Марковиця [1], ідеях сучасних підходів до роботи з сингулярними матрицями [2] та використання у цьому випадку матриць Мура-Пенроуза [3], а також використанні обчислювальних методів для фінансового аналізу. Робота складається з п'яти розділів. У першому подано вступ та обґрунтування теми. У другому розглянуто теоретичні основи розкладу Холеського та матриці Мура–Пенроуза. У третьому сформульовано та реалізовано класичну задачу оптимізації портфеля Марковиця та модифіковану. У четвертому проведено прикладне моделювання на основі реальних фінансових даних. У п'ятому виконано стохастичний аналіз очікуваної дохідності та дисперсії портфеля.

2 Огляд основних означень та теоретичних положень, що пов'язані з псевдо-оберненою матрицею

2.1 Основні означення лінійної алгебри та псевдо-обернених матриць

Розглянемо основні означення лінійної алгебри, які стануть у нагоді для подальшого розуміння алгоритму. Почнемо з того, що псевдо-обернена матриця (обернена у сенсі Мура-Пенроуза) є найбільш загальновизнаним узагальненням оберненої матриці (див., наприклад, [3]). Наедемо базові поняття, необхідні для розуміння псевдо-обернених матриць та їх застосувань у портфельному аналізі ([3]):

Означення 1. *Псевдо-обернена матриця* (за Муром-Пенроузом) для матриці $A \in \mathbb{R}^{m\times n}$ — це матриця A^+, яка задовольняє такі умови [2]:

1. **Рефлексивність:**
$$AA^+A = A$$
2. **Симетрія:**
$$A^+AA^+ = A^+$$
3. **Гермітові властивості:**
$$(AA^+)^T = AA^+$$
$$(A^+A)^T = A^+A$$

Матриця Мура-Пенроуза може бути використана у випадках, коли матриця є сингулярною або не є квадратною. У випадку якщо матриця A може бути обернена стандартним способом, її псевдообернена матриця збігається з цією оберненою, а саме:

$$A^+ = A^{-1}[3]$$

Для того, аби розуміти, для яких матриць маємо застосовувати алгоритм знаходження оберненої Мура-Пенроуза наведемо означення сингулярної матриці.

Означення 2 Сингулярна матриця — це така матриця, визначник якої дорівнює нулю:

$$\det A = 0.$$

Також у ході роботи було проведено перевірку на додатньовизначеність матриць. Якщо матриця виявлялась додатньовизначеною, вона мала позитивний визначник і в такому випадку до неї не застосовувався алгоритм знаходження матриці Мура-Пенроуза. Якщо ж вона не проходила перевірку на додатньовизначеність, застосовувався алгоритм знаходження псевдо-оберненої матриці.

Означення 3 *Додатно визначена матриця:* Матриця A є додатно визначеною, якщо для всіх ненульових векторів x виконується нерівність

$$x^\top Ax > 0.$$

Додатно визначена матриця має дійсні та додатні власні значення, а також всі її головні мінори (визначники головних підматриць) є додатними. З цього означення також випливає, що всі її діагональні елементи є додатними. [4]

Для того, аби знайти розв'язок рівняння $Ax = b$, де матриця A є додатно визначеною, можна використати розклад Холеського (В оригіналі The Cholesky Decomposition), що має вигляд:

$$A = L^\top L,$$

де L — верхня трикутна матриця. Цей розклад існує тоді і тільки тоді, коли матриця A є симетричною та додатно визначеною. В такому разі, спроба виконати розклад Холеського [4] є перевіркою, чи є матриця додатно визначеною. Якщо розклад виконати неможливо, матриця не є додатно визанченою.

В мові програмування Python перевірка на додатньовизначеність матриці може бути виконана за допомогою бібліотеки numpy та методу з цієї бібліотеки. Функція для обчислення розкладу Холеського у Python:

```
np.linalg.cholesky(matrix)
```

Якщо декомпозиція може бути виконана - матриця є додатньовизначеною. В ході виконання практичної роботи цю перевірку на мові Python було виконано

і вручну, слідуючи алгоритму. Розглянемо симетричну, додатно визначену матрицю A:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

Потрібно знайти нижньотрикутну матрицю L, таку що:

$$A = LL^{\top}.$$

Матриця L матиме вигляд:

$$L = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix}$$

Елементи L обчислюються за такими формулами:

Діагональні елементи:

$$l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}, \quad \text{для } j = 1, \ldots, n.$$

Обчислення всіх інших елементів:

$$l_{ij} = \frac{1}{l_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right), \quad \text{для } i = j+1, \ldots, n.$$

Застосовуючи ці рівняння, можна виконати розклад Холеського [5] для будь-якої симетричної додатно визначеної матриці.

Важливо зазначити [6], що якщо матриця A є додатно визначеною, то вона має обернену, і її визначник є невід'ємним:

$$\det A > 0$$

В такому разі алгоритм знаходження оберненої матриці Мура-Пенроуза застосовуватися не буде і класичної інверсії буде достатньо.

Приклад 1. Розглянемо ілюстративний приклад симетричної, але не додатновизначеної матриці розміру 3×3, в якої визначник $= 0$:

```
import numpy as np
matrix = np.array([
    [1, 2, 3],
    [2, 4, 6],
    [3, 6, 9]
])
#Manual method output
Matrix is not positive definite.
Negative value encountered at L[1][1].
#Built-in method
#np.linalg.cholesky(matrix):
print(is_positive_definite(matrix))
determinant = np.linalg.det(matrix)
print(determinant)
#Output:
False
0.0
```

Спочатку у коді виведено результат для перевірки додатньовизначеності вручну (див додаток 1), що також показує в якому саме індексі було виявлено негативне значення. Після цього використовується згаданий раніше готовий метод для перевірки. Таким чином було перевірено декомпозицію Холеського, прописану вручну. Результати збігаються з вбудованим методом в бібліотеці numpy для кількох прикладів.

Також можемо побачити, що для матриці A неможливо знайти обернену, адже її визначник дорівнює нулю. Тому у випадку сингулярності маємо застосовувати алгоритм для знаходження матриці Мура-Пенроуза.

Перевірка матриць на додатньовизначеність у роботі була використана для того, аби визначати для яких матриць буде використаний класичний алгоритм застосування знаходження оберненої матриці, а для яких - псевдооберненої.

У подальших розділах ці означення будуть використані для побудови алгоритмів оптимізації портфеля Марковиця, включаючи випадки сингулярних матриць коваріації.

На початку дослідження було реалізовано алгоритм перевірки додатньої визначеності та знаходження оберненої матриці на мові Python. У разі виявлення того, що вхідна матриця була додатньовизначеною, була застосована базова інверсія використовуючи алгоритм Гауса. Розглянемо короткий опис першої програми. Алгоритм складається з двох основних функцій: `is_positive_definite` та `is_positive_definite_and_inverse`. Функція `is_positive_definite` реалізовувала розклад Холеського (Cholesky decomposition) описаний вище вручну. Вона перевіряє, чи задана матриця є симетричною(якщо ж ні, негативна відповідь видається одразу), а далі послідовно будує нижню трикутну матрицю L, для якої виконується умова $A = LL^T$. При цьому, якщо на будь-якому кроці на діагоналі L виникає від'ємне або нульове значення, матриця не вважається додатньо визначеною. Функція `is_positive_definite_and_inverse` викликає перевірку на додатню визначеність, а у випадку позитивного результату — виконує обчислення оберненої матриці методом Гаусса-Жордана. Для цього створюється розширена матриця $(A \mid I)$, де I — одинична матриця того ж порядку, що і A. Після приведення A до одиничної матриці, друга частина містить шукану A^{-1}.

Приклад. Наведемо приклад використання алгоритму для симетричної матриці:

$$A = \begin{bmatrix} 4 & 2 & 2 \\ 2 & 3 & 1 \\ 2 & 1 & 3 \end{bmatrix}.$$

Результатом виконання функції є підтвердження додатньої визначеності та наступна обернена матриця (обчислена вручну методом Ґаусса):

$$A^{-1} = \begin{bmatrix} 0.5 & -0.5 & -0.0 \\ -0.5 & 1.5 & -0.5 \\ -0.0 & -0.5 & 0.5 \end{bmatrix}.$$

Для перевірки правильності обчислень додатково було виконано множення AA^{-1} та порівняння з одиничною матрицею. Також було здійснено порівняння з вбудованою функцією `numpy.linalg.inv` для підтвердження результату.

Основна перевага такого підходу — можливість ручної перевірки умов додатньої визначеності з відповідною інтерпретацією результатів.

Запропонований алгоритм може бути використаний у практичних задачах аналізу коваріаційних матриць, що і знадобиться нам далі у роботі.

2.2 Методи обчислення псевдо-оберненої матриці

Використовується декілька алгоритмів знаходження псевдо-оберненої матриці. Розглянемо один із них та опишемо покроково. У цій роботі матриця Мура-Пенроуза для матриці A знаходиться за допомогою алгоритму SVD - Singular Value Decomposition або Сингулярного Представлення Матриці. Введемо означення для цього розкладу.

Алгоритм Сингулярного Представлення матриці (SVD) [8] - це метод, що дозволяє представити матрицю як набір лінійних компонент, що відображають її внутрішню структурну суть.Також цей алгоритм допомагає знайти матрицю Мура-Пероуза для даної матриці.

Розглянемо цей алгоритм послідовно.

Спершу виконується розклад матриці A (розмірності $m \times n$) на добуток трьох матриць:

$$A = U\Sigma V^T$$

де:

- U — ортогональна матриця розмірності $m \times m$, стовпці якої є лівими сингулярними векторами матриці A.
- Σ — діагональна матриця розмірності $m \times n$ з сингулярними значеннями матриці A на головній діагоналі.
- V — ортогональна матриця розмірності $n \times n$, стовпці якої є правими сингулярними векторами матриці A.

Після цього псевдообернена матриця A^+ визначається як:

$$A^+ = V\Sigma^+ U^T$$

де:

- Σ^+ — псевдообернена діагональна матриця, яка отримується з Σ шляхом заміни кожного ненульового елемента σ_i на $\frac{1}{\sigma_i}$ і транспонуванням, якщо Σ не квадратна.

- U — ортогональна матриця $m \times m$, ліві сингулярні вектори A.

- Σ — діагональна матриця $m \times n$ з сингулярними значеннями A.

- V — ортогональна матриця $n \times n$, праві сингулярні вектори A. [7]

2.3 Ілюстративні приклади знаходження матриці Мура–Пенроуза використовуючи запропонований алгоритм

Приклад 2. Розглянемо ілюстративний приклад знаходження матриці Мура-Пенроуза вручну, використовуючи запропонований вище алгоритм SVD. Для прикладу візьмемо сингулярну матрицю розмірності 2×2. Нехай

$$A = \begin{pmatrix} 3 & 6 \\ -1 & -2 \end{pmatrix}$$

Для перевірки обчислимо визначник матриці A:

$$\det(A) = 3 \cdot (-2) - 6 \cdot (-1) = -6 + 6 = 0$$

$$\Rightarrow \det(A) = 0$$

Отже, матриця A є сингулярною, тому класичний алгоритм знаходження оберненої для неї не моеж бути застосовний.

Згідно з алгоритмом SVD, розклад матриці A має вигляд:

$$A = U\Sigma V^T$$

Сингулярні значення:

$$\sigma_1 = \sqrt{50}, \quad \sigma_2 = 0$$

$$\Sigma = \begin{pmatrix} \sqrt{50} & 0 \\ 0 & 0 \end{pmatrix}, \quad \Sigma^+ = \begin{pmatrix} \frac{1}{\sqrt{50}} & 0 \\ 0 & 0 \end{pmatrix}$$

Матриці U та V, впорядковані відповідно до сингулярних значень:

$$U = \begin{pmatrix} -0.9487 & -0.3162 \\ 0.3162 & -0.9487 \end{pmatrix}, \quad V = \begin{pmatrix} -0.4472 & -0.8944 \\ -0.8944 & 0.4472 \end{pmatrix}$$

Наступний крок — знайти псевдообернену:

$$A^+ = V\Sigma^+U^T$$

$$U^T = \begin{pmatrix} -0.9487 & 0.3162 \\ -0.3162 & -0.9487 \end{pmatrix}$$

$$\Sigma^+U^T = \begin{pmatrix} \frac{1}{\sqrt{50}} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} -0.9487 & 0.3162 \\ -0.3162 & -0.9487 \end{pmatrix} = \begin{pmatrix} -0.1341 & 0.0447 \\ 0 & 0 \end{pmatrix}$$

$$A^+ = \begin{pmatrix} -0.4472 & -0.8944 \\ -0.8944 & 0.4472 \end{pmatrix} \begin{pmatrix} -0.1341 & 0.0447 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0.06 & -0.02 \\ 0.12 & -0.04 \end{pmatrix}$$

Фінальна відповідь:

$$A^+ = \begin{pmatrix} 0.06 & -0.02 \\ 0.12 & -0.04 \end{pmatrix}$$

Тепер розгялнемо ілюстративний приклад знаходження матриці Мура–Пенроуза використовуючи запропонований алгоритм вже за допомогою програми написаної на Pythonz. Розглянемо ту саму сингулярну матрицю

$$A = \begin{pmatrix} 3 & 6 \\ -1 & -2 \end{pmatrix}$$

та знайдемо псевдо-обернену для неї використовуючи мову програмування Python, щоб запевнитись, що обрахунки вручну були проведені коректно. Для обчислення псевдооберненої матриці за допомогою Python використаємо бібліотеку `NumPy` та готову функцію `np.linalg.pinv`. Також для додаткової

перевірки вручну був написаний метод, що знаходить псевдообернену матрицю, використовуючи бібліотеку `NumPy` та готову функцію `np.linalg.svd`. Вона автоматично виконує перший крок алгоритму і розкладає задану матрицю на добуток трьох. Також написані допоміжні функції, що виконують усі наступні кроки алгоритму сингулярного представлення. В обох випадках виконання програм отримали наступні результати: **Оригінальна матриця:**

$$A = \begin{pmatrix} 3 & 6 \\ -1 & -2 \end{pmatrix}$$

Визначник матриці A:

$$\det(A) = 0.0$$

Псевдообернена матриця (використання функції `np.linalg.pinv`): (див. Додаток 1)

$$A^{+} = \begin{pmatrix} 0.06 & -0.02 \\ 0.12 & -0.04 \end{pmatrix}$$

Псевдообернена матриця ((використання функції `np.linalg.svd`)): (див. Додаток 1)

$$A^{+} = \begin{pmatrix} 0.06 & -0.02 \\ 0.12 & -0.04 \end{pmatrix}$$

Отже, у цьому розділі було розглянуто основні означення, що будуть використані у роботі з псевдо-оберненими матрицями та стануть у нагоді при розв'язку задачі оптимізації портфеля Марковіца у випадку сингулярності коваріаційної матриці. Також було розглянуто алгоритм знаходження псевдо-оберненої матриці та показано ілюстративний приклад того, як він працює. Приклад був обрахований вручну та за допомогою мови Python у двох модифікаціях. Збіжність результатів показує коректність відпрацювання алгоритму.

Отже, у другому розділі роботи було введено ключові поняття лінійної алгебри, що необхідні для подальшого використання у ході виконання дослідження : сингулярна матриця, додатно визначена матриця, псевдо-обернена матриця (матриця Мура–Пенроуза). Також розглянуто випадок, коли обернена та псевдо-обернена співпадають. Також було зазначено, що псевдо-обернена

матриця є узагальненням поняття оберненої матриці і використовується у випадках, коли звичайна обернена матриця не існує (наприклад, для сингулярних або прямокутних матриць). Було описано алгоритм перевірки матриці на додатню визначеність за допомогою розкладу Холеського, як вручну, так і за допомогою Додатково наведено реалізацію алгоритму визначення оберненої матриці для додатньо визначених симетричних матриць з використанням методу Гаусса. У випадку сингулярних матриць продемонстровано обраний алгоритм обчислення псевдо-оберненої матриці за допомогою сингулярного розкладу (SVD) матриці A на три інші матриці. В кінці також було показано ілюстративний приклад розрахунку псевдо-оберненої матриці вручну та підтверджено результат з використанням бібліотек `numpy.linalg.pinv` і `numpy.linalg.svd` у Python. Отже, отримані результати співпадають у всіх випадках, що свідчить про коректність реалізації алгоритмів та їх практичну придатність для подальших задач на реальних історичних даних. Підсумовуючи, отримані та розглянуті означення та алгоритми становлять фундамент для розв'язання задачі оптимізації портфеля Марковіца у випадках сингулярної коваріаційної матриці, що буде розглянуто у наступних розділах роботи.

3 Алгоритм і реалізація програми для побудови оптимального портфеля Марковиця для сингулярних даних

3.1 Теоретичні основи портфеля Марковиця

Оптимальний портфель був розглянутий в багатьох джерелах, і ця тема є досить відомою та активно досліджуваною.

Базуючись на вже існуючих роботах ([9],[10],[11]), введемо поняття інвестиційного портфелю, аби сформулювати задачі оптимізації. Інвестиційний портфель - збалансована сукупність різних фінансових інструментів, таких як акції, облігації, деривативи та інші активи, що перебувають під управлінням інвесторів або фінансових установ.

Основною метою формування портфеля є диверсифікація ризиків, яку ми хочемо отримати за рахунок комбінування активів та розподілу інвестицій у різні джерела. Також метою є знизити загальний рівень ризику інвестиційного портфеля без значної втрати очікуваної дохідності[9] . У другому розглянутому формулюванні, до прикладу, буде розглянуто ставлення інвестора до ризику як змінний параметр.

Розглянемо основні характеристики, які описують поведінку портфеля (див. [10],[11]): математичне сподівання (середнє значення доходності), дисперсія та коваріація доходностей активів. Математичне сподівання (mean) (E[X]) відображає середнє або найбільш типове значення доходності активу. Також на це значення будемо посилатися, як на "очікувану дохідність"Дисперсія (variance) (Var(X))дозволяє оцінити ступінь розсіювання доходностей відносно середнього значення, тобто рівень ризику, пов'язаний з інвестицією. Коваріація (covariance), у свою чергу, характеризує ступінь взаємозв'язку між доходностями різних активів у портфелі, що має важливе значення при формуванні збалансованого портфеля.

Далі при роботі з реальними фінансовими даними було обраховано оцінки усіх цих характеристик інвестиційного портфелю, а також було проведено оцінку матриці коваріації.

Для ефективного управління інвестиційним портфелем необхідно врахо-

вувати ці статистичні характеристики, які допомагають визначити не лише очікувану прибутковість, але й рівень ризику та взаємозалежність між окремими активами.

Переходячи до теорії портфелю Марковіца, у класичному варіанті задача оптимізації має на меті мінімізувати дисперсію, тобто ризик. При цьому має бути враховано, що дохідність залишиться більшою чи рівною таргетованій дохідності, що прописано в обмеженнях в умові задачі.

У роботі буде розглянуто два формулювання задачі оптимізації портфеля Марковіца. Розглянемо перший класичний варіант та введемо усі позначення [10]:

$$\min_x \quad \sigma_x^2 \quad \equiv \quad \min_x \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_{ij} x_i x_j$$

$$\text{s.t.} \quad \sum_{i=1}^{d} \mu_i x_i \geq r$$

$$\sum_{i=1}^{d} x_i = 1.$$

- $x = (x_1, x_2, \ldots, x_d)$ — вектор часток інвестованого капіталу у кожен із d активів.
- σ_x^2 — дисперсія дохідності портфеля (ризик портфеля).
- σ_{ij} — елемент матриці коваріацій, що показує взаємозв'язок між прибутковістю активів i та j.
- μ_x — очікувана дохідність портфеля.
- μ_i — очікувана дохідність окремого активу i.
- r — мінімально допустима очікувана дохідність портфеля.
- Умова $\sum_{i=1}^{d} x_i = 1$ введена для того щоб запевнитися, що всі інвестиції розподілені між активами (повна інвестованість капіталу).

Після цього розглянемо модифіковане формулювання задачі Марковіца: максимізація дохідності з урахуванням ризику

У даному формулюванні враховується як очікувана дохідність, так і ризик портфеля. Метою є максимізація дохідності з урахуванням ризику, зваженого на параметр схильності до ризику інвестора:

$$\max_x \left(\mu_x - \tau\sigma_x^2\right) \equiv \max_x \left(\sum_{i=1}^{d} \mu_i x_i - \tau \left(\sum_{i=1}^{d}\sum_{j=1}^{d} \sigma_{ij} x_i x_j\right)\right)$$

$$\text{s.t.} \quad \sum_{i=1}^{d} x_i = 1.$$

де:

- $x = (x_1, x_2, \ldots, x_d)$ — вектор часток капіталу, інвестованих у кожен із d активів;
- $\mu_x = \sum_{i=1}^{d} \mu_i x_i$ — очікувана дохідність портфеля;
- μ_i — очікувана дохідність активу i;
- $\sigma_x^2 = \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_{ij} x_i x_j$ — дисперсія дохідності портфеля;
- σ_{ij} — елемент матриці коваріацій, що показує зв'язок між активами i та j;
- τ — параметр схильності до ризику: що більшим є τ, то більшу вагу інвестор надає зменшенню ризику;
- умова $\sum_{i=1}^{d} x_i = 1$ гарантує повну інвестованість капіталу.

Варто зазначити, що модифікованому формулюванні задачі оптимізації портфеля Марковіца, [11] де максимізується функція $\mu_x - \tau\sigma_x^2$, детальніше розглянемо параметр τ, який називається *коефіцієнтом схильності до ризику* (в оригіналі це звучить як risk aversion parameter). Він відображає індивідуальні вподобання інвестора [16] щодо балансу між очікуваною дохідністю портфеля та його ризиком.

З математичної точки зору, параметр τ визначає, яку вагу інвестор надає ризику під час прийняття оптимізаційного рішення. Якщо $\tau = 0$, це означає, що інвестор ігнорує ризик і зосереджується виключно на максимізації дохідності — така поведінка характерна для повністю ризикованої (*risk-seeking*)

стратегії. Навпаки, високі значення τ вказують на сильну схильність до уникнення ризику: інвестор готовий жертвувати частиною потенційного прибутку заради зменшення волатильності портфеля.

У частині наукових джерел замість τ використовується позначення α. Хоча обидва параметри вводять компроміс між ризиком і дохідністю, їх інтерпретації можуть відрізнятися. Параметр τ зазвичай розглядається як *поведінкова характеристика інвестора*, яка відображає його індивідуальну толерантність до ризику. У той час як α часто використовується як *математичний параметр регуляризації* — він змінює вигляд ефективного фронту або допомагає стабілізувати чисельне розв'язання задачі оптимізації.

Завдяки параметру τ модель Марковіца набуває гнучкості: змінюючи його значення, можна адаптувати оптимальне портфельне рішення під різні типи інвесторів — від нейтральних до ризику до дуже обережних.

3.2 Розробка алгоритму побудови портфеля Марковіца(2 модифікації)

Розглянемо алгоритм побудови оптимального портфеля Марковіца для першого класичного формулювання:

$$\min_x \quad \sigma_x^2 \quad \equiv \quad \min_x \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_{ij} x_i x_j$$

$$\text{s.t.} \quad \sum_{i=1}^{d} \mu_i x_i \geq r$$

$$\sum_{i=1}^{d} x_i = 1.$$

Для того, аби мінімізувати таргетовану функцію нам знадобиться функція Лагранжа. Спершу маємо сформувати функцію Лагранжа з множниками Лагранжа u та v

$$L = \sum_{i=1}^{d} \sum_{j=1}^{d} \delta_{ij} x_i x_j - v \left(\sum_{i=1}^{d} \mu_i x_i - r \right) - u \left(\sum_{i=1}^{d} x_i - 1 \right)$$

Далі аби мінімізувати функціюL, беремо часткові похідні L по кожному x_i та прирівнюємо їх до нуля:

$$\frac{\partial L}{\partial x_i} = 0 \quad \text{для } i = 1, \ldots, d$$

У висновку матимемо d рівнянь.

$$2\sum_{j=1}^{d} \delta_{ij} x_j - v\mu_i - u = 0, \quad \text{for all } i = 1, \ldots, d$$

Щоб знайти оптимальний портфель [10], розв'язуємо ці $d + 2$ рівнянь для $d + 2$ змінних: $x_1, \ldots, x_d, u, v$.

Альтернативно розв'язання класичного формулювання задачі Марковіца має вигляд:

$$\underbrace{\begin{bmatrix} 2\sigma_{11} & 2\sigma_{12} & \cdots & 2\sigma_{1d} & -\mu_1 & -1 \\ 2\sigma_{21} & 2\sigma_{22} & \cdots & 2\sigma_{2d} & -\mu_2 & -1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 2\sigma_{d1} & 2\sigma_{d2} & \cdots & 2\sigma_{dd} & -\mu_d & -1 \\ \mu_1 & \mu_2 & \cdots & \mu_d & 0 & 0 \\ 1 & 1 & \cdots & 1 & 0 & 0 \end{bmatrix}}_{\mathbf{A}} \cdot \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ v \\ u \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ r \\ 1 \end{bmatrix}}_{\mathbf{b}}$$

Отже,

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ v \\ u \end{bmatrix} = \mathbf{A}^{-1}\mathbf{b}$$

Саме у такому варіанті розв'язку ми стикаємося з тим, що маємо знайти обернену матрицю, а отже у випадку сингулярності маємо застосовувати псевдо-обернену. Проте у роботі був зроблений акцент на друге формулювання для випадку сингулярності, адже тут випадок є тривіальнішим.

Розглянемо розв'язок для позитивновизначеної матриці для другого формулювання задачі оптимізації портфеля Марковіца.

$$\max_x \left(\mu_x - \tau\sigma_x^2\right) \equiv \max_x \left(\sum_{i=1}^{d} \mu_i x_i - \tau \left(\sum_{i=1}^{d}\sum_{j=1}^{d} \sigma_{ij} x_i x_j\right)\right)$$

$$\text{s.t.} \quad \sum_{i=1}^{d} x_i = 1.$$

Нижче наведено формули для обчислення оптимального вектора ваг портфеля у ввипадку коли коваріаційна матриця Σ є додатноозначеною [13]:

$$\mathbf{w}_{EU} = \frac{\Sigma^{-1}\mathbf{1}_k}{\mathbf{1}_k^\top \Sigma^{-1}\mathbf{1}_k} + \frac{1}{2\tau}\mathbf{R}\boldsymbol{\mu}, \tag{1}$$

де

$$\mathbf{R} = \Sigma^{-1} - \frac{\Sigma^{-1}\mathbf{1}_k\mathbf{1}_k^\top\Sigma^{-1}}{\mathbf{1}_k^\top\Sigma^{-1}\mathbf{1}_k}. \tag{2}$$

- $\mathbf{w}_{EU}$ — оптимальні ваги активів у портфелі за критерієм очікуваної корисності;
- Σ — коваріаційна матриця доходностей активів;
- Σ^{-1} — обернена матриця до Σ;
- $\mathbf{1}_k$ — k-вимірний вектор, заповнений одиницями;
- τ — коефіцієнт ризиконебажаності інвестора;
- $\mathbf{R}$ — матриця, що відображає скориговану структуру ризику;
- $\boldsymbol{\mu}$ — вектор математичних сподівань доходностей активів.

У ситуації, коли матриця Σ є сингулярною , замість оберненої матриці використовується псевдообернена матриця Мура-Пенроуза, позначена як Σ^+. Такий підхід було запропоновано Bodnar, T., Mazur, S., Nguyen у їхній науковій роботі[2]

$$\mathbf{w}_{EU} = \frac{\Sigma^{+}\mathbf{1}_k}{\mathbf{1}_k^\top \Sigma^{+}\mathbf{1}_k} + \frac{1}{2\tau}\mathbf{R}\boldsymbol{\mu},$$

$$\mathbf{R} = \Sigma^{+} - \frac{\Sigma^{+}\mathbf{1}_k\mathbf{1}_k^{\top}\Sigma^{+}}{\mathbf{1}_k^{\top}\Sigma^{+}\mathbf{1}_k}.$$

- $\mathbf{w}_{EU}$ — оптимальні ваги портфеля за критерієм максимізації очікуваної корисності.
- Σ^{+} — псевдообернена матриця Мура–Пенроуза для коваріаційної матриці Σ.
- $\mathbf{1}_k$ — k-вимірний вектор, заповнений одиницями.
- τ — коефіцієнт ризиконебажаності інвестора.
- $\mathbf{R}$ — матриця, що враховує скориговану структуру ризику.
- $\boldsymbol{\mu}$ — вектор очікуваних доходностей активів.

3.3 Ілюстративні приклади оптимізації портфеля Марковіца для двох формулювань

Розглянемо ілюстративний приклад розв'язання задачі оптимізації портфеля Марковіца у класичному формулюванні для двох активів (d=2). Будемо використовувати алгоритм запропонований вище.

Спочатку розглядаємо базові випадки, де матриця коваріацій є позитивно визначеною та не є сингулярною.

Приклад 3. Задамо середні значення, мінімальну очікувану дохідінсть портфеля та матрицю коваріації розмірністю 2x2

$$\mu_1 = 0.1, \quad \mu_2 = 0.08, \quad r = 0.09$$

$$\Sigma = \begin{bmatrix} 0.04 & 0.02 \\ 0.02 & 0.03 \end{bmatrix}$$

$$\delta_x^2 = \sum_{i=1}^{2}\sum_{j=1}^{2}\delta_{ij}x_i x_j = \delta_{11}x_1^2 + 2\delta_{12}x_1x_2 + \delta_{22}x_2^2$$

1. Формуємо функцію Лагранжа

$$L = 0.04x_1^2 + 2(0.02)x_1x_2 + 0.03x_2^2 \\ - v(0.1x_1 + 0.08x_2 - 0.09) - u(x_1 + x_2 - 1)$$

2. Знаходимо часткові похідні функції Лагранжа та прирівнюємо їх до нуля, аби мінімізувати функцію:

$$\frac{\partial L}{\partial x_1} = 2 \cdot 0.04x_1 + 2 \cdot 0.02x_2 - 0.1v - u = 0$$
$$\frac{\partial L}{\partial x_2} = 2 \cdot 0.03x_2 + 2 \cdot 0.02x_1 - 0.08v - u = 0$$
$$\frac{\partial L}{\partial v} = -0.1x_1 - 0.08x_2 + 0.09 = 0$$
$$\frac{\partial L}{\partial u} = -x_1 - x_2 + 1 = 0$$

3. Розв'язуємо отриману систему рівнянь для невідомих змінних:

$$\begin{cases} 0.08x_1 + 0.04x_2 - 0.1v - u = 0 \\ 0.06x_2 + 0.04x_1 - 0.08v - u = 0 \\ 0.1x_1 + 0.08x_2 = 0.09 \\ x_1 + x_2 = 1 \end{cases}$$

4. Отримали фінальний розв'язок:

$$x_1 = 0.5, \quad x_2 = 0.5$$
$$v = 0.5, \quad u = 0.01$$

Візьмемо ті самі вхідні дані:

$$\mu_1 = 0.1, \quad \mu_2 = 0.08, \quad r = 0.09$$
$$\Sigma = \begin{bmatrix} 0.04 & 0.02 \\ 0.02 & 0.03 \end{bmatrix}$$

Приклад 4. Використаємо мову програмування Python аби перевірити, чи результати були пораховані коректно.

Для цього використовуємо бібліотеку `NumPy` та готову функцію `np.linalg.solve`. Додавши модифікації було написано метод, що обраховує оптимальний портфель Марковіца для першого класичного формулювання.(див. Додаток 1) Програма приймає на вхід такі параметри:

1. вектор очікуваних прибутковостей активів;

2. коваріаційну матрицю прибутковостей;

3. задану цільову прибутковість портфеля.

Далі формулюється система лінійних рівнянь, що відповідає умовам першого класичного формулювання задачі Марковіца з двома обмеженнями:

1. сума ваг активів у портфелі дорівнює одиниці;

2. очікувана прибутковість портфеля дорівнює заданому значенню.

Система включає як змінні ваги активів, так і два множники Лагранжа, які враховують відповідно обмеження на прибутковість і на суму ваг. Матрична форма цієї системи автоматично будується у програмі, після чого розв'язується за допомогою раніше згаданого методу для розв'язку задач лінійної оптимізації з бібліотеки Numpy.

У результаті виконання дана програма повертає:

- оптимальні ваги активів у портфелі;

- значення множників Лагранжа;

- перевірку досягнутої прибутковості портфеля;

- обчислену дисперсію (ризик) портфеля.

У висновку маємо, що реалізований алгоритм дозволяє не тільки знайти оптимальне розподілення інвестицій, але й оцінити, наскільки обґрунтовані результати з точки зору поставлених обмежень. Це забезпечує додаткову надійність при побудові портфеля на основі реальних фінансових даних. Проте у даному дослідженні не буде використано оптимізації для першого варіанту

на реальних фінансових даних, хоча це можна розглянути у майбутніх дослідженнь. Тож у ході виконання програми на ілюстративному прикладі були отримані наступні результати:
Система лінійних рівнянь має такий матричний вигляд:

$$\begin{bmatrix} 0.08 & 0.04 & -0.1 & -1.0 \\ 0.04 & 0.06 & -0.08 & -1.0 \\ 0.10 & 0.08 & 0.0 & 0.0 \\ 1.00 & 1.00 & 0.0 & 0.0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ v \\ u \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.09 \\ 1.0 \end{bmatrix}$$

Розв'язок цієї системи дає оптимальні ваги портфеля та значення множників Лагранжа:

$$x_1 = 0.5, \quad x_2 = 0.5$$
$$u = 0.010000000000000003 \quad \text{(множник обмеження ваг)}$$
$$v = 0.49999999999999956 \quad \text{(множник обмеження дохідності)}$$

Оптимальні характеристики портфеля:

$$\text{Дохідність портфеля:} \quad r_p = 0.09$$
$$\text{Дисперсія портфеля:} \quad \sigma_p^2 = 0.0275$$

Бачимо, що пораховані результати вручну збігаються з результатами обрахованими в Python.
Приклад 5. Тепер розглянемо приклад оптимізації портфеля Марковіца вже у другому формулюванні з коефіцієнтом ставлення інвестора до ризику. Для ілюстрації було обрано параметр 4000, хоча дослдідження були проведені на різних випадках. Також цей приклад ілюструє випадок сингулярності, адже матриця коваріації має нульовий визначник. Було продемонстровано алгоритм Sigular Value Decomposition та попроково показано необхідні калькуляції.

$$\mu_1 = 0.1$$

$$\mu_2 = 0.08$$

$$\Sigma = \begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix}$$

$$\tau = 4000$$

Обрахунок матриці Мура-Пенроуза для Σ:

$$\Sigma\Sigma^T = \begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix} = \begin{bmatrix} 20 & 10 \\ 10 & 5 \end{bmatrix}$$

$$\det\left(\begin{bmatrix} 20-\lambda & 10 \\ 10 & 5-\lambda \end{bmatrix}\right) = (20-\lambda)(5-\lambda) - 100 = \lambda^2 - 25\lambda = 0$$

$$\lambda_1 = 25, \quad \lambda_2 = 0$$

$$\sigma_1 = \sqrt{25} = 5, \quad \sigma_2 = \sqrt{0} = 0$$

$$\Sigma = \begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}, \quad \Sigma^+ = \begin{bmatrix} \frac{1}{5} & 0 \\ 0 & 0 \end{bmatrix}$$

$$(\Sigma\Sigma^T - 25I)\vec{u} = 0, \quad \begin{bmatrix} -5 & 10 \\ 10 & -20 \end{bmatrix} \Rightarrow \vec{u}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \hat{u}_1 = \frac{1}{\sqrt{5}} \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$(\Sigma\Sigma^T - 0I)\vec{u} = 0, \quad \begin{bmatrix} 20 & 10 \\ 10 & 5 \end{bmatrix} \Rightarrow \vec{u}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \quad \hat{u}_2 = \frac{1}{\sqrt{5}} \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

$$U = \begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix}$$

$$v_1 = \frac{1}{\sigma_1}\Sigma^T u_1 = \frac{1}{5} \begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix} \cdot \frac{1}{\sqrt{5}} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \frac{1}{5\sqrt{5}} \begin{bmatrix} 5 \\ 10 \end{bmatrix} = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$v_2 = \frac{1}{\sqrt{5}} \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

$$V = \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{-2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix}$$

$$V\Sigma^{+} = \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{-2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} \frac{1}{5} & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{5\sqrt{5}} & 0 \\ \frac{2}{5\sqrt{5}} & 0 \end{bmatrix}$$

$$U^{T} = \begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix}$$

$$\Sigma^{+} = V\Sigma^{+}U^{T} = \begin{bmatrix} \frac{1}{5\sqrt{5}} & 0 \\ \frac{2}{5\sqrt{5}} & 0 \end{bmatrix} \begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix}$$

$$\Sigma^{+} = \begin{bmatrix} \frac{2}{25} & \frac{1}{25} \\ \frac{4}{25} & \frac{2}{25} \end{bmatrix}$$

Обрахунок оптимального портфеля:

$$\Sigma^{+} = \begin{bmatrix} 0.08 & 0.04 \\ 0.16 & 0.08 \end{bmatrix}, \quad \mu = \begin{bmatrix} 0.1 \\ 0.08 \end{bmatrix}, \quad \mathbf{1}_k = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \tau = 4000$$

$$\Sigma^{+}\mathbf{1}_k = \begin{bmatrix} 0.08 & 0.04 \\ 0.16 & 0.08 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.12 \\ 0.24 \end{bmatrix}$$

$$\mathbf{1}_k^{\top}\Sigma^{+}\mathbf{1}_k = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 0.12 \\ 0.24 \end{bmatrix} = 0.36$$

$$\frac{\Sigma^{+}\mathbf{1}_k}{\mathbf{1}_k^{\top}\Sigma^{+}\mathbf{1}_k} = \frac{1}{0.36} \begin{bmatrix} 0.12 \\ 0.24 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \end{bmatrix}$$

$$\Sigma^{+}\mu = \begin{bmatrix} 0.08 & 0.04 \\ 0.16 & 0.08 \end{bmatrix} \begin{bmatrix} 0.1 \\ 0.08 \end{bmatrix} = \begin{bmatrix} 0.012 \\ 0.024 \end{bmatrix}$$

$$\frac{1}{2\tau} = \frac{1}{8000}$$

$$\frac{1}{8000} \cdot \Sigma^{+}\mu = \frac{1}{8000} \cdot \begin{bmatrix} 0.012 \\ 0.024 \end{bmatrix} = \begin{bmatrix} 1.5 \times 10^{-6} \\ 3 \times 10^{-6} \end{bmatrix}$$

$$w_{EU} = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \end{bmatrix} + \begin{bmatrix} 1.5 \times 10^{-6} \\ 3 \times 10^{-6} \end{bmatrix} = \begin{bmatrix} 0.3330015 \\ 0.666003 \end{bmatrix}$$

Приклад 6. Розглянемо той самий приклад, але вже використовуючи програму, написану на Python, аби перевірити коректність результатів, обрахованих вручну. Код реалізації алгоритму можна побачити у додатку.

У загальному програма на Python для дургого формулювання оптимізації портфеля Марковіца реалізує наступні кроки:

1. Обчислюється псевдообернена матриця коваріацій Σ^+ з використанням функції `np.linalg.pinv`, що реалізує стабільний чисельний метод на основі вже раніше згаданого сингулярного розкладу (SVD) згідно документації PythonNumpy.

2. Знаходиться перший компонент оптимального портфеля — портфель з мінімальною дисперсією (мінімально-ризиковий портфель), який не враховує очікувані прибутковості. Він обчислюється за формулою, яка була наведена у теоретичнійі частині розділу:

$$x^{(1)} = \frac{\Sigma^+ \mathbf{1}}{\mathbf{1}^\top \Sigma^+ \mathbf{1}},$$

де $\mathbf{1}$ — вектор одиниць, що відповідає числу активів.

3. Обчислюється другий компонент — поправка на ризик-аверсію, яка враховує очікувані прибутковості активів та ступінь небажання інвестора до ризику τ:

$$x^{(2)} = \frac{1}{2\tau}\left(\Sigma^+ - \frac{\Sigma^+ \mathbf{1}\mathbf{1}^\top \Sigma^+}{\mathbf{1}^\top \Sigma^+ \mathbf{1}}\right)\mu.$$

Цей доданок враховує відхилення від мінімального ризику в напрямку очікуваних прибутковостей. Саме тут ми можемо регулювати наш результат у залежності від вподобань певного інвестора щодо ризику.

4. Остаточний портфель є сумою цих двох компонент:

$$x = x^{(1)} + x^{(2)}.$$

В результаті виконання алгоритму ми отримуємо вектор x — оптимальні ваги активів у портфелі, які водночас враховують мінімізацію ризику, очікувану прибутковість та ризик-аверсію інвестора. У цьому випадку ніяких

обмежень на додатність доданків ще додано не було. Але у ході роботи з реальними даними ситуація зміниться.

Додатково, для наглядності результату та покроковості обчислень виводяться допоміжні результати, такі як:

- псевдообернена матриця Σ^{+};
- компоненти $x^{(1)}$ і $x^{(2)}$ окремо;
- фінальний вектор оптимальних ваг x.

Отже перейдемо до тестування програми на ілюстратичному прикладі.

Візьмемо ті самі вхідні дані:

$$\mu_1 = 0.1, \quad \mu_2 = 0.08, \quad \tau = 4000$$

$$\Sigma = \begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix}$$

Отримаємо наступні результати:

Determinant of $\Sigma = 0.0$

Inverse of Σ $(\Sigma^{-1}) =$

$$\begin{bmatrix} 0.08 & 0.04 \\ 0.16 & 0.08 \end{bmatrix}$$

First Term (Minimum Variance Portfolio) $=$

$$\begin{bmatrix} 0.33333333 & 0.66666667 \end{bmatrix}$$

Second Term (Risk-Adjusted Return Component) $=$

$$\begin{bmatrix} 1 \cdot 10^{-7} & 2 \cdot 10^{-7} \end{bmatrix}$$

Optimal Portfolio Weights $=$

$$\begin{bmatrix} 0.33333343 & 0.66666687 \end{bmatrix}$$

Бачимо, що результати обох обчислень збігаються і оптимальний портфель для ілюстративного прикладу має вигляд:

$$\begin{bmatrix} 0.33333343 & 0.66666687 \end{bmatrix}$$

4 Практичне застосування для реальних фінансових даних

4.1 Збір та попередня обробка фінансових даних

Перейдемо до реалізації практичної частини. Для того, аби застосувати описаний алгоритм побудови оптимального портфелю було використано реальні історичні дані про ціни акцій чотирьох великих компаній, що представлені на біржі NASDAQ: **Netflix**, **Amazon**, **Tesla** та **Apple**. Дані було завантажено у форматі CSV з офіційного сайту біржі NASDAQ, після чого проведено попередню обробку у середовищі Python.

Попередня обробка включала:

- імпорт цін закриття акцій;
- усунення пропущених значень;
- вирівнювання часових рядів по датах;
- приведення до одного формату та масштабу;
- візуалізацію динаміки цін.

Розглянемо візуалізації, аби розуміти загальну ситуацію. Нижче подано графіки цін закриття акцій кожної з компаній у часовому розрізі (за період одного року, дані були вивантаженя на початку квітня 2025):

Рис. 1: Динаміка цін акцій компанії Netflix. Видно чіткий ріст у першій половині періоду з наступним поступовим спадом і коливаннями.

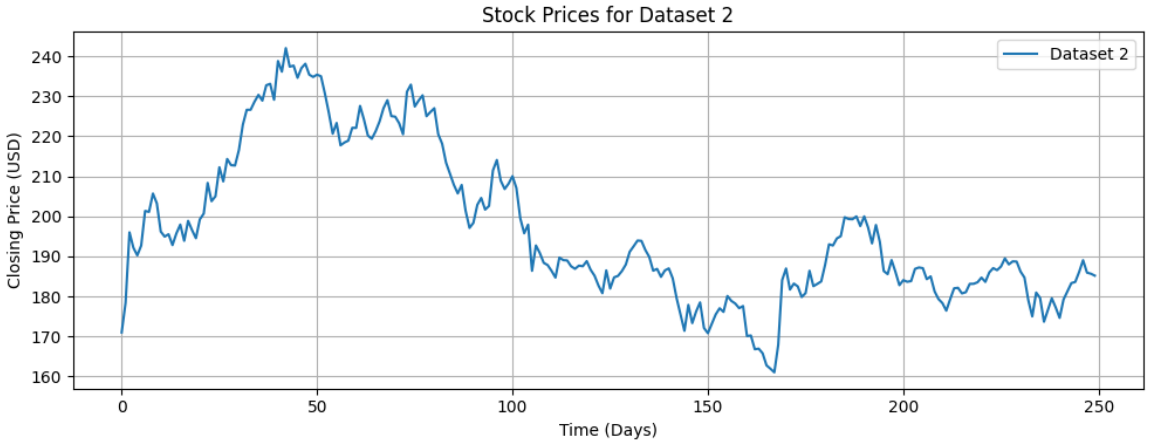

Рис. 2: Ціни акцій компанії Amazon. Спостерігається значна волатильність з падінням після піку — характерна для нестабільного періоду.

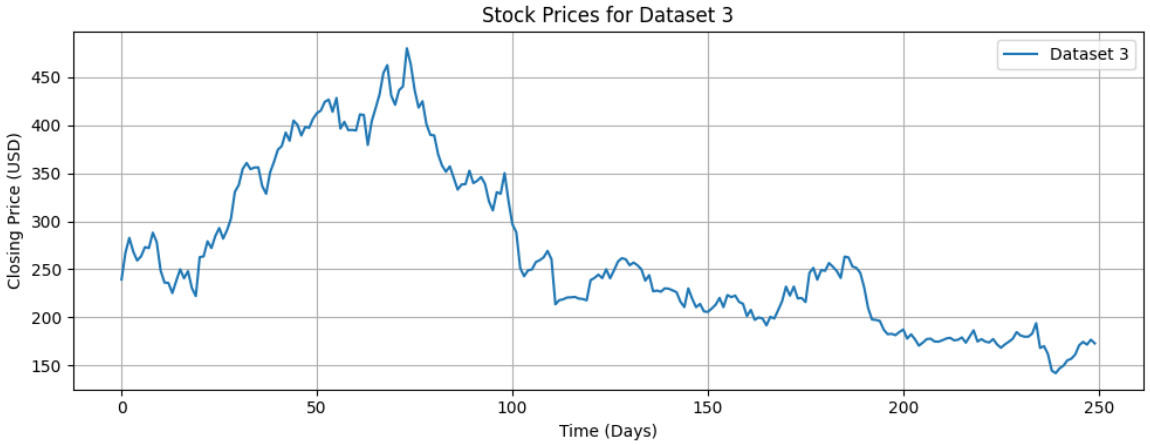

Рис. 3: Ціни акцій компанії Tesla. Протягом періоду акції демонстрували кілька хвиль зростання та зниження, зберігаючи високий рівень змін.

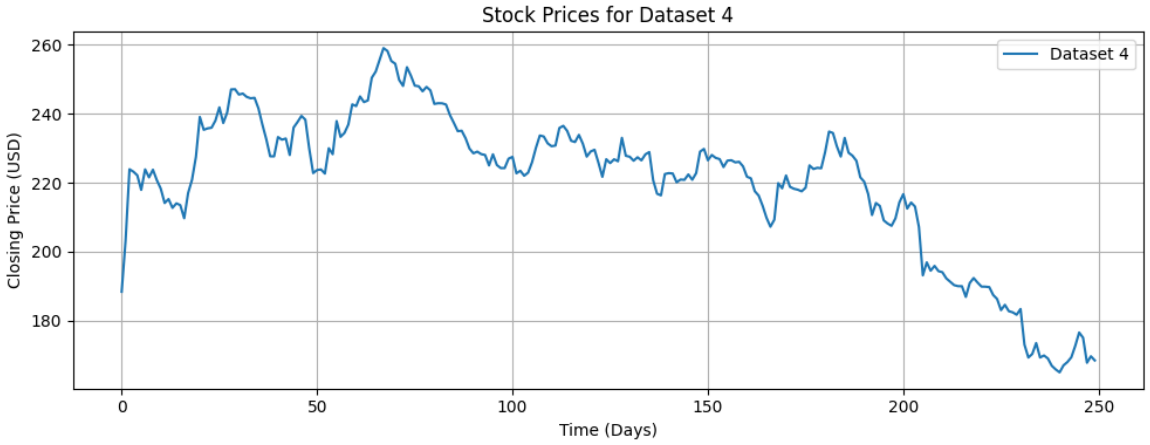

Рис. 4: Динаміка акцій компанії Apple. Відносно стабільні зміни з тенденцією до зниження у другій половині періоду.

Ці графіки дозволяють оцінити характер змін цін акцій та волатильність кожної компанії за один рік. Це знадобилося у подальшому аналізі для формування оптимального інвестиційного портфеля та для інтерпритації отриманих результатів.

4.2 Оцінка параметрів

Після збору та попередньої обробки фінансових даних наступним кроком була оцінка ключових параметрів, які необхідні для побудови оптимального портфеля Марковіца. Зокрема, це очікувані значення дохідностей (математичне сподівання), коефіцієнти кореляції Пірсона між активами, а також коваріаційна матриця.

Почнемо з обрахунків оцінок середніх значень (очікуваної дохідності).

Для кожного активу було розраховано середнє значення на основі щоденних цін закриття. Були отримані наступні результати:

- Середнє значення для dataset 1 (Netflix): **769.64**
- Середнє значення для dataset 2 (Amazon): **196.77**
- Середнє значення для dataset 3 (Tesla): **265.62**
- Середнє значення для dataset 4 (Apple): **220.06**

У векторному вигляді в Python це виглядало так:

$$\boldsymbol{\mu} = \begin{bmatrix} 769.64404 \\ 196.77032 \\ 265.6184 \\ 220.06492 \end{bmatrix}$$

Далі розглянемо оцінки коефіцієнтів кореляції, аби розуміти кореляції між даними та можливі потенційні мультиколінеарності, які власне і можуть спричинити випадок сингулярності коваріаційної матриці.

Для оцінки лінійної залежності між парами активів було обчислено коефіцієнти кореляції Пірсона. [14] Формула для розрахунку кореляції між активами i та j виглядає наступним чином [15]:

$$\hat{\rho}_{i,j} = \frac{\sum_{t=1}^{N}(x_t^i - \bar{X}_i)(x_t^j - \bar{X}_j)}{\sqrt{\sum_{t=1}^{N}(x_t^i - \bar{X}_i)^2 \sum_{t=1}^{N}(x_t^j - \bar{X}_j)^2}}$$

де $\mathrm{cov}(X, Y)$ — коваріація між рядами X та Y, а σ_X і σ_Y — їх стандартні відхилення. Для обчислення цього коефіцієнта у ході роботи був реалізований алгоритм мовою Python. Програма дозволяла завантажити декілька CSV-файлів з даними про ціни закриття фінансових активів. На попередньому етапі дані очищуються: з цін видаляються символи долара, значення конвертуються у числовий формат та усуваються пропущені значення.

Після цього для кожного ряду обчислюється середнє значення ціни. Далі ряди обрізаються до однакової довжини, що забезпечує коректність порівняння.

Основною метою є обчислення коефіцієнта кореляції Пірсона між кожною парою часових рядів, який визначає лінійний зв'язок між двома змінними. Результатом роботи програми ми отримали виведення значень кореляційних коефіцієнтів для кожної пари рядів.
Розглянемо результати розрахунків:

- Кореляція між dataset 1 і dataset 2: **0.821**
- Кореляція між dataset 1 і dataset 3: **0.796**
- Кореляція між dataset 1 і dataset 4: **0.675**
- Кореляція між dataset 2 і dataset 3: **0.895**
- Кореляція між dataset 2 і dataset 4: **0.597**
- Кореляція між dataset 3 і dataset 4: **0.756**

Як видно з результатів, активи демонструють досить високі значення позитивної кореляції, що свідчить про наявність спільних ринкових тенденцій. Від'ємних значень не було виявлено.

Нарешті перейдемо до оцінки коваріаційної матриці. Коваріаційна матриця є ключовим елементом в моделі Марковіца, оскільки вона відображає не

лише ризик кожного окремого активу (через дисперсію), але й спільні ризики (через коваріацію). Також вона необхідна для розв'язку задачі оптимізації. Для побудови коваріаційної матриці використовувалась вбудована функція `np.cov` у Python, бібліотека numpy.

Python-код:

```
cov_matrix = np.cov(data_matrix)
```

Отримана оцінка матриці коваріацій має наступний вигляд:

$$\Sigma = \begin{bmatrix} 19646.23 & 2202.97 & 9318.26 & 2036.24 \\ 2202.97 & 366.27 & 1430.73 & 245.93 \\ 9318.26 & 1430.73 & 6969.40 & 1358.97 \\ 2036.24 & 245.93 & 1358.97 & 463.27 \end{bmatrix}$$

Ця матриця буде використана на наступному етапі побудови оптимального портфеля.

4.3 Оптимізація портфеля у базовому випадку та у випадку сингулярності

Розглянемо оптимізацію портфеля на реальних фінансових даних. На цьому етапі, використовуючи раніше оцінені параметри (а саме вектор середніх значень доходностей та коваріаційну матрицю), було здійснено побудову оптимального інвестиційного портфеля Марковіца. Було розглянуто декілька варіацій. Також варто зазначити, що оптимізація була попередньо проведена для різних наборів даних (за шість місяців, за рік на двох різних наборах даних).

Розглянемо результати оцінок матрицб коваріацій за різні періоди та моделювання випадків сингулярності для них.

Період 1: 6 місяців (Netflix, Apple, Tesla)

Для першого періоду, що охоплює шість місяців, матриця коваріації Σ оцінюється за історичними даними трьох активів у вигляді:

$$\Sigma = \begin{bmatrix} 10111.9983 & 924.8702 & 7710.6236 \\ 924.8702 & 118.4363 & 746.9015 \\ 7710.6236 & 746.9015 & 6890.7104 \end{bmatrix}$$

Для моделювання випадку сингулярної матриці, третій рядок був заданий як лінійна комбінація перших двох:

$$\Sigma_{\text{singular}} = \begin{bmatrix} 10111.9983 & 924.8702 & 7710.6236 \\ 924.8702 & 118.4363 & 746.9015 \\ (0.5 \cdot 10111.9983 + 0.5 \cdot 924.8702) & (0.5 \cdot 924.8702 + 0.5 \cdot 118.4363) & (0.5 \cdot 7710.6236 + 0.5 \cdot 746.9015) \end{bmatrix}$$

Це призводить до того, що третій рядок є залежним від перших двох, що робить матрицю сингулярною і дає можливість проілюструвати застосування матриці Мура-Пенроуза.

Період 2: 1 рік (Netflix, Amazon, Apple, Tesla)

Для більш тривалого періоду (один рік) із залученням чотирьох активів, отримано таку оцінку матриці коваріації:

$$\Sigma = \begin{bmatrix} 11597.1930 & 1661.3038 & 2257.5196 & 7857.4475 \\ 1661.3038 & 297.7284 & 294.8522 & 1208.2153 \\ 2257.5196 & 294.8522 & 666.9957 & 1621.2992 \\ 7857.4475 & 1208.2153 & 1621.2992 & 6362.0946 \end{bmatrix}$$

Аналогічно, для створення сингулярного випадку, четвертий рядок був змодельований як зважена лінійна комбінація перших трьох:

$$\Sigma_{\text{singular}} = \begin{bmatrix} 11597.1930 & 1661.3038 & 2257.5196 & 7857.4475 \\ 1661.3038 & 297.7284 & 294.8522 & 1208.2153 \\ 2257.5196 & 294.8522 & 666.9957 & 1621.2992 \\ 0.5 \cdot 11597.1930 + 0.3 \cdot 1661.3038 + 0.2 \cdot 2257.5196 & 0.5 \cdot 1661.3038 + 0.3 \cdot 297.7284 + 0.2 \cdot 294.8522 & 0.5 \cdot 2257.5196 + 0.3 \cdot 294.8522 + 0.2 \cdot 666.9957 & 0.5 \cdot 7857.4475 + 0.3 \cdot 1208.2153 + 0.2 \cdot 1621.2992 \end{bmatrix}$$

У результаті четвертий рядок стає лінійно залежним від перших трьох, що знову ж таки призводить до втрати повного рангу матриці.

Надалі буде розглянуто фінальний варіант оптимізації (за рік, дані вивантаженя були на початку квітня 2025).

$$\Sigma = \begin{bmatrix} 19646.23 & 2202.97 & 9318.26 & 2036.24 \\ 2202.97 & 366.27 & 1430.73 & 245.93 \\ 9318.26 & 1430.73 & 6969.40 & 1358.97 \\ 2036.24 & 245.93 & 1358.97 & 463.27 \end{bmatrix}$$

Для початку розгялнемо загальну формулу оптимізації портфеля у випадку сингулярності.

На відміну від класичного варіанта, де матриця коваріацій додатно визначена, ми використовуємо більш загальний підхід, запропонований у вже згаданій роботі Bodnar, Mazur, Nguyen [2], що включає використання псевдооберненої матриці Мура-Пенроуза. Важливо зазначити, що на відміну від попередньої формули для розв'язку зараз ми будемо використовувати оцінки параметрів, які відповідно матимуть інші позначення.
Отже, формула для знаходження вектора ваг оптимального портфеля має вигляд:

$$\widehat{\mathbf{w}}_{EU}^{+} = \frac{\mathbf{S}^{+}\mathbf{1}_k}{\mathbf{1}_k^{\top}\mathbf{S}^{+}\mathbf{1}_k} + (2\tau)^{-1}\widehat{\mathbf{R}}^{+}\bar{\mathbf{x}},$$

де

$$\widehat{\mathbf{R}}^{+} = \mathbf{S}^{+} - \frac{\mathbf{S}^{+}\mathbf{1}_k\mathbf{1}_k^{\top}\mathbf{S}^{+}}{\mathbf{1}_k^{\top}\mathbf{S}^{+}\mathbf{1}_k}.$$

Тут:

- $\widehat{\mathbf{w}}_{EU}^{+}$ — оптимальні ваги активів;
- $\mathbf{S}^{+}$ — оцінка псевдооберненої коваріаційної матриці;
- $\mathbf{1}_k$ — вектор одиниць розмірності k;
- τ — параметр ризик-аверсії, що дорівнює $\tau = \alpha/2$ (в оригінальній літературі використано позначення α);
- $\bar{\mathbf{x}}$ — вектор середніх доходностей;
- $\widehat{\mathbf{R}}^{+}$ — коригована матриця ризику на основі вибірки.

Бачимо, що матриця коваріації у формулі тепер позначена $\mathbf{S}$, бо це вже значення-оцінка.

Розглянемо оптимізації портфеля для реальних фінансоих даних.Першою була реалізована базова версія оптимізації для додатно визначеної матриці

коваріацій без накладання додаткових обмежень на ваги активів. У результаті обчислень були отримані оптимальні ваги, проте серед них виявилися від'ємні значення.
Отримані результати:

$$\texttt{Optimal Portfolio Weights:} \quad [-0.0478 \quad 0.8498 \quad -0.1773 \quad 0.3753]$$

Зважаючи на отримання від'ємних ваг у попередньому кроці, наступним етапом було здійснено оптимізацію з додатковими обмеженнями, які забороняють від'ємні значення ваг. Це було реалізовано в програмі Python із застосуванням відповідних обмежень.
Оскільки у класичній моделі Марковиця не передбачено жорстких обмежень на знак ваг активів, в оптимальному розв'язку можуть виникати як додатні, так і від'ємні значення(з чим ми і стикнулися на практиці з реальними фінансовими даними). Від'ємні ваги інтерпретуються як короткі позиції (short selling), однак у багатьох практичних сценаріях короткий продаж є недоступним або небажаним. Тому постає необхідність модифікації розв'язку так, щоб ваги портфеля були иільк додатними для зручності практичного застосування програми.

У рамках цієї роботи було реалізовано простий і ефективний спосіб забезпечення додатності ваг після первинного розв'язання задачі оптимізації. Алгоритм полягає у наступному:

1. Усі від'ємні значення ваг обнуляються:

$$x[x < 0] = 0$$

2. Отриманий вектор нормалізується так, щоб сума ваг дорівнювала одиниці:

$$x = \frac{x}{\sum x}$$

Цей підхід забезпечує виконання умов:

$$x_i \geq 0 \quad \forall i, \quad \sum_{i=1}^{n} x_i = 1$$

Альтернативно розглядалася також інша модифікація, яка зберігає напрямок початкового розподілу, коригуючи його з урахуванням відкинутих від'ємних ваг. Однак у фінальній версії роботи було обрано саме більш просту нормалізацію після обнулення негативних значень, оскільки вона виявилася стабільнішою у чисельних експериментах та легко інтерпретується з фінансової точки зору. Обидві версії можна переглянути у додатку з кодом.

Таким чином, отриманий портфель задовольняє практичним обмеженням інвестора: всі інвестиції є не від'ємними, а їх загальна сума становить 100% капіталу.

$$\texttt{Optimal Weights (Non-negative):} \quad [0 \quad 0.6937 \quad 0 \quad 0.3063]$$

Як видно, у портфелі залишилися лише два активи, і серед значень тепер немає від'ємних.

Далі було розглянуто випадок, коли коваріаційна матриця є сингулярною. Це може статися, наприклад, у ситуаціях, коли деякі активи мають лінійну залежність. У нашому випадку це було змодельовано повторенням одного з рядків у матриці Σ:

$$\Sigma = \begin{bmatrix} 19646.23 & 2202.97 & 9318.26 & 2036.24 \\ 2202.97 & 366.27 & 1430.73 & 245.93 \\ 9318.26 & 1430.73 & 6969.40 & 1358.97 \\ 19646.23 & 2202.97 & 9318.26 & 2036.24 \end{bmatrix}$$

Так як маємо ситуацію сингулярності, визначник дорівнює нулю, а тому класичне обернення неможливе. Замість цього було використано псевдообернену матрицю Мура-Пенроуза.

Результати оптимізації у випадку сингулярності:

$$\texttt{Optimal Portfolio Weights:} \quad [-0.0940 \quad 3.8600 \quad -0.0816 \quad -2.6844]$$

Цей результат має обмежене практичне застосування, оскільки містить великі від'ємні ваги, проте з математичної точки зору демонструє застосування узагальненого підходу до побудови портфеля навіть у складних умовах.

Отже, для реальних фінансових даних компаній Netflix, Amazon, Tesla та Apple:

- Знайдено оптимальний портфель Марковитця w = $\begin{bmatrix} -0.0478 \\ 0.8498 \\ -0.1773 \\ 0.3753 \end{bmatrix}$;, де ваги з плюсом означають купити, а з мінусом - продати.
- Побудовано портфель шляхом додавання обмежень w = $\begin{bmatrix} 0 \\ 0.6937 \\ 0 \\ 0.3063 \end{bmatrix}$;
- Побудовано портфель для сингулярної кореляційної матриці

$$\mathbf{w} = \begin{bmatrix} -0.0940 \\ 3.8600 \\ -0.0816 \\ -2.6844 \end{bmatrix}$$

5 Стохастичне представлення очікуваної дохідності та дисперсії

5.1 Теорема про стохастичне представлення обраних величин

Перейдемо до теореми про стохастичне представлення та симуляції на реальних фінансових даних. У ході виконання роботи було досліджено стохастичне представлення, яке є важливим етапом аналізу оцінок оптимального портфеля у випадку, коли коваріаційна матриця є сингулярною. Цей підхід дозволяє не лише побудувати оцінки, але й описати та візуально побачити їх розподіли. Розглянемо теорему про стохастичне представлення. (Bodnar, Mazur, Nguyen [2])

Нехай $\mathbf{x}_1, \ldots, \mathbf{x}_n$ — незалежні однаково розподілені випадкові вектори з розподілом $\mathcal{N}_k(\mu, \Sigma)$, при цьому $k \geq n$, а ранг матриці Σ дорівнює $r < n$. Тоді стохастичні представлення для оцінок очікуваної дохідності $\widehat{R}^+_{EU}$ та дисперсії $\widehat{V}^+_{EU}$ мають вигляд:

$$\widehat{R}^+_{EU} \stackrel{d}{=} R^+_{GMV} + \alpha^{-1}\frac{(n-1)(r-1)}{n(n-r+1)}\xi + \sqrt{\frac{1}{n}\left(1 + \frac{r-1}{n-r+1}\xi\right)}\sqrt{V^+_{GMV}}z_0,$$

$$\widehat{V}^+_{EU} \stackrel{d}{=} \frac{V^+_{GMV}}{n-1}\eta + \alpha^{-2}\frac{(n-1)(r-1)}{n(n-r+1)}\xi,$$

де:

- $\xi \sim \mathcal{F}_{r-1,n-r+1}$ — розподіл Фішера,
- $\eta \sim \chi^2_{n-r}$ — хі-квадрат розподіл,
- $z_0 \sim \mathcal{N}(0,1)$ — стандартне нормальне розподілення.

Ці випадкові величини є незалежними між собою. Згідно з теоремою [2], оцінки $\widehat{R}^+_{EU}$ та $\widehat{V}^+_{EU}$ є випадковими величинами, що залежать як від структури дисперсій-коваріацій, так і від кількості спостережень.

- R^+_{GMV} — очікувана дохідність глобального портфеля з мінімальною дисперсією (GMV);
- $V^+_{GMV} = \frac{1}{\mathbf{1}_k^\top \Sigma^+ \mathbf{1}_k}$ — дисперсія GMV-портфеля;
- α — коефіцієнт ризик-аверсії (в коді обчислення може бути виражений через $\tau = \alpha/2$).

5.2 Проведення симуляції V^+_{GMV} та R^+_{GMV} на реальних фінансових даних

У рамках моделювання було зібрано вибірку з трьох компаній: Netflix, Apple і Tesla (за період = шість місяців), на основі якої побудовано наступну матрицю коваріацій:

$$\Sigma = \begin{bmatrix} 10111.998 & 924.870 & 7583.999 \\ 924.870 & 118.436 & 693.653 \\ 7710.624 & 746.901 & 5782.968 \end{bmatrix}$$

Ця матриця є близькою до сингулярної, що дозволяє застосовувати її для даного стохастичного представлення.

Гістограми оцінок

На основі багаторазових симуляцій за вказаними стохастичними формулами були побудовані гістограми оцінок $\widehat{R}^+_{EU}$ та $\widehat{V}^+_{EU}$ Була реалізована програма на Python. Програма приймає наступні параметри:

- коваріаційну матрицю активів Σ;
- обсяг вибірки n (кількість спостережень);
- кількість факторів r, що відповідає кількості оцінюваних параметрів;
- параметр ризик-аверсії α;
- кількість симуляцій N (за замовчуванням — 1000).

Алгоритм виконує наступні кроки:

1. Обчислюється дисперсія глобального мінімуму ризику (GMV portfolio) з використанням псевдооберненої матриці:

$$V_{GMV}^{+} = \left(\mathbf{1}^{\top}\Sigma^{+}\mathbf{1}\right)^{-1}.$$

2. Для кожної симуляції генеруються випадкові змінні згідно з теоретичними розподілами:

   - $\xi \sim F_{r-1,\,n-r+1}$ — випадкова змінна з розподілом Фішера;
   - $\eta \sim \chi^2_{n-r}$ — хі-квадрат розподіл;
   - $z_0 \sim \mathcal{N}(0,1)$ — стандартна нормальна змінна.

3. На основі аналітичних формул оцінок з літератури [2] розраховуються:

$$\widehat{R}_{EU}^{+} = V_{GMV}^{+} + \frac{(n-1)(r-1)}{\alpha(n-r+1)}\xi + \sqrt{\frac{1}{n}\left(1 + \frac{(r-1)}{n-r+1}\xi\right)}\sqrt{V_{GMV}^{+}}z_0,$$

$$\widehat{V}_{EU}^{+} = \frac{V_{GMV}^{+}}{n-1}\eta + \frac{(n-1)(r-1)}{\alpha^2 n(n-r+1)}\xi.$$

4. Зібрані значення зберігаються в масиви для подальшої візуалізації.

5. Побудова двох гістограм: одна — для емпіричних значень очікуваної дохідності $\widehat{R}_{EU}^{+}$, інша — для дисперсій $\widehat{V}_{EU}^{+}$. Візуалізація дозволяє оцінити варіативність та асиметрію оцінок залежно від параметрів моделі.

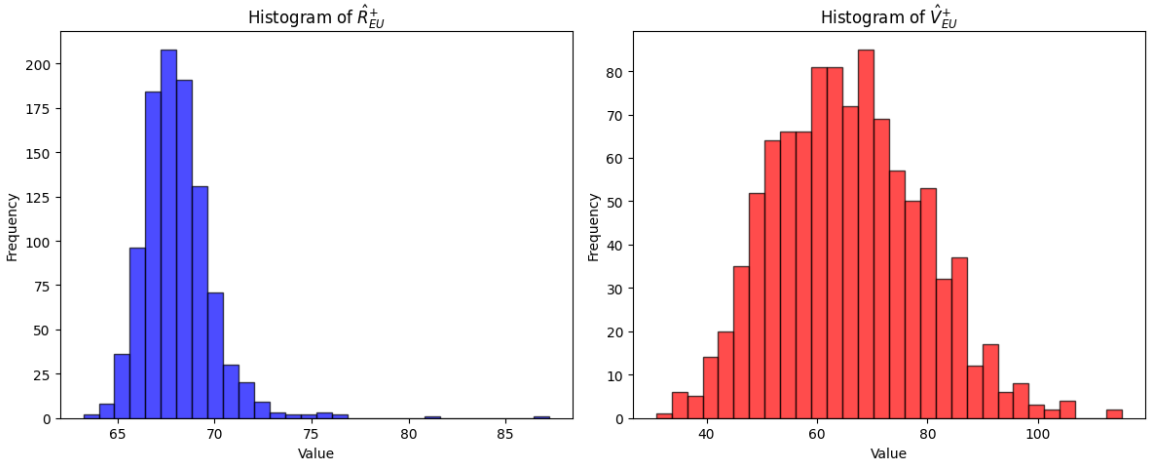

Рис. 5: Гістограми оцінок очікуваної дохідності $\widehat{R}_{EU}^{+}$ (зліва) та дисперсії $\widehat{V}_{EU}^{+}$ (справа)

Аналіз результатів симуляції та висновки

- Бачимо, що гістограма $\widehat{R}^{+}_{EU}$ має симетричну, дзвоноподібну форму, що свідчить про наближеність до нормального розподілу.

- Для оцінки $\widehat{V}^{+}_{EU}$ гістограма є асиметричною з правим хвостом, що відповідає теоретичному розподілу, сформованому через χ^2 та F-розподіли.

- Такі результати підтверджують теоретичні висновки про стохастичну природу оцінок, які базуються не лише на точкових значеннях, але і на варіативності, зумовленій випадковістю вибірки.

Отже, у цьому розділі було проведено симуляцію стохастичного представлення, що дає змогу глибше зрозуміти поведінку оцінок оптимального портфеля у реальних ринкових умовах, де дані часто є обмеженими, а коваріаційні матриці — сингулярними. Для практичного застосування такі дослідження є важливими, адже вони дають додаткову інформацію.

Висновки

У ході дослідження були досягнуті всі попердньо поставлені завдання. Основною метою було дослідити властивості псевдо-обернених матриць(або матриць Мура-Пенроуза) та продемонструвати їх застосування у задачах оптимізації інвестиційного портфеля Марковіца.Окрема увага була приділена випадку, коли матриця коваріацій виявлялась сингулярною або ж близткою до сингулярної. Таке акцентування уваги обгрунтовується практичним застосуванням даного підходу для вирішення реальної проблеми у випадку роботи з історичними фінансовими даними.

Результати виконання дослідження:

1. Розглянуто алгоритм і приклади застосування (вручну і на Python) розкладу Холеського для визначення додатньо-визначеної матриці.

2. Розглянуто алгоритм і приклади обчислення матриці Мура–Пенроуза за допомогою сингулярного розкладу (SVD) початкової матриці на три інші матриці.

3. Розглянуто алгоритми та ілюстративні приклади для двох формулювань задачі оптимізації портфеля Марковиця

4. Побудовано декілька оптимізаційних портфелів для компаній Netflix, Amazon, Tesla та Apple

5. Проведено стохастичне представлення двох вибраних випадкових величин — очікуваної дохідності та дисперсії портфеля.

Наступним кроком для подальших досліджень є інтеграція методу Мура-Пенроуза з підходами до оцінки ризику, зокрема з методологією Value at Risk (VaR). Таке поєднання може відкрити нові можливості для точнішого моделювання ризиків портфеля в умовах сингулярних або нестабільних даних, особливо в контексті управління ризиками на нестійких ринках.

Додаток А. Програмний код

А.1 Перевірка матриці на додатньовизначеність

```
import numpy as np
#MAIn SCRIPT!
def is_positive_definite(matrix):
    """
    Check if a given square matrix is positive definite using Cholesky decomposition.
    """
    # Check for symmetry
    if not np.allclose(matrix, matrix.T):
        return False, "Matrix is not symmetric."

    n = matrix.shape[0]
    try:
        # Create an empty lower triangular matrix
        L = np.zeros_like(matrix)

        for i in range(n):
            for j in range(i + 1): # Only fill lower triangular part
                if i == j: # Diagonal elements
                    sum_diagonal = sum(L[i][k]**2 for k in range(j))
                    value = matrix[i][i] - sum_diagonal
                    if value <= 0:
                        return False, f"Matrix is not positive definite. Negative value encountered at L[{i}][{i}]."
                    L[i][j] = np.sqrt(value)
                else: # Off-diagonal elements
                    sum_off_diag = sum(L[i][k] * L[j][k] for k in range(j))
                    L[i][j] = (matrix[i][j] - sum_off_diag) / L[j][j]

        return True, "Matrix is positive definite."
    except Exception as e:
        return False, f"Error occurred: {e}"

# Example usage
matrix = np.array([
    [1, 2, 3],
    [2, 4, 6],
    [3, 6, 9]
])

result, message = is_positive_definite(matrix)
print(message)
import numpy as np

def is_positive_definite(matrix):
    try:
        # Attempt Cholesky decomposition
        np.linalg.cholesky(matrix)
        return True
    except np.linalg.LinAlgError:
        return False

# Example usage
matrix = np.array([
    [1, 2, 3],
```

```
    [2, 4, 6],
    [3, 6, 9]
])

print(is_positive_definite(matrix)) # Output: True
determinant = np.linalg.det(matrix)
print(determinant)
```

A.2 Обрахунок псевдо-оберненої матриці

```
import math
import numpy as np

def compute_pseudo_inverse_svd(matrix):
    """
    Computes the Moore-Penrose pseudo-inverse of a given matrix using Singular Value Decomposition (SVD).

    Parameters:
        matrix (list of list of floats): The input matrix to be pseudo-inverted.

    Returns:
        list of list of floats: The pseudo-inverse of the input matrix.
    """
    # Helper function to compute transpose of a matrix
    def transpose(mat):
        return [[mat[j][i] for j in range(len(mat))] for i in range(len(mat[0]))]

    # Helper function to multiply two matrices
    def mat_mult(mat1, mat2):
        return [[sum(a * b for a, b in zip(row, col)) for col in zip(*mat2)] for row in mat1]

    # Step 1: Perform Singular Value Decomposition (SVD)
    def svd_manual(mat):
        mat = np.array(mat)
        u, s, vh = np.linalg.svd(mat, full_matrices=False)
        return u, s, vh

    # Step 2: Compute pseudo-inverse of the singular value matrix
    def pseudo_inverse_singular_values(singular_values, tol=1e-10):
        return [1 / sv if sv > tol else 0 for sv in singular_values]

    # Step 3: Combine to form pseudo-inverse
    matrix_np = np.array(matrix)
    u, s, vh = svd_manual(matrix_np)
    s_inv = np.diag(pseudo_inverse_singular_values(s))

    # Reconstruct pseudo-inverse A^+ = V^T * S^+ * U^T
    pseudo_inverse = np.dot(vh.T, np.dot(s_inv, u.T))
    return pseudo_inverse.tolist()

# Example usage
if __name__ == "__main__":
    # Define a sample matrix
    #A = [[2, 4], [1, 2]]
    A = [[3, 6], [-1, -2]]
   # A = [[4, 2, 2], [2, 3, 1], [2, 1, 3]]
```

```
    # Compute the pseudo-inverse using SVD
    A_pseudo_inverse = compute_pseudo_inverse_svd(A)

    print("Original Matrix:")
    for row in A:
        print(row)

    print("\nPseudo-Inverse Matrix:")
    for row in A_pseudo_inverse:
        print(row)
import numpy as np

# Define the singular matrix A
#A_singular = np.array([
# [0.0004, 0.0003, 0.00035],
# [0.0003, 0.0005, 0.0004],
# [0.00035, 0.0004, 0.000375] # Linear combination of the first two rows/columns
#])
#A_singular = np.array(
# [[2, 4], [1, 2]])
A_singular = np.array(
    [[3, 6], [-1, -2]])
# Calculate the determinant
det_A = np.linalg.det(A_singular)

# Print the determinant
print("Determinant of matrix A:", det_A)
pseudo_inverse = np.linalg.pinv(A_singular)
print("Pseudoinverse of matrix A:",pseudo_inverse)
```

A.3 Оптимізація портфеля Марковіца у класичному формулюванні

```
import numpy as np

def lagrange_portfolio_optimization(mu, Sigma, target_return):
    """
    Optimizes portfolio weights using the Lagrange multiplier method.

    Args:
        mu (np.array): Expected returns for assets.
        Sigma (np.array): Covariance matrix of asset returns.
        target_return (float): Target return for the portfolio.

    Returns:
        weights (np.array): Optimal portfolio weights.
        lagrange_multipliers (tuple): Values of u and v (Lagrange multipliers).
    """
    d = len(mu) # Number of assets

    # Create matrices for the linear system
    A = np.zeros((d + 2, d + 2))
    b = np.zeros(d + 2)
```

```
    # Fill in the covariance matrix part (2 * Sigma)
    A[:d, :d] = 2 * Sigma

    # Add the Lagrange multiplier components
    A[:d, d] = -mu # -v * mu part
    A[:d, d + 1] = -1 # -u part
    A[d, :d] = mu # Sum(mu_i * x_i) = r
    A[d + 1, :d] = 1 # Sum(x_i) = 1
    print(A)
    # Right-hand side for target return and weight sum
    b[d] = target_return
    b[d + 1] = 1
    print(b)
    # Solve the linear system
    solution = np.linalg.solve(A, b)

    # Extract portfolio weights and Lagrange multipliers
    weights = solution[:d]
    lagrange_multipliers = (solution[d+1], solution[d])

    return weights, lagrange_multipliers

# Input Data
mu = np.array([0.1, 0.08]) # Expected returns for each asset
Sigma = np.array([
    [0.04, 0.02],
    [0.02, 0.03]
]) # Covariance matrix
target_return = 0.09 # Desired portfolio return

# Optimize the portfolio
optimal_weights, (u, v) = lagrange_portfolio_optimization(mu, Sigma, target_return)

# Display results
print("Optimal Portfolio Weights:", optimal_weights)
print("Lagrange Multiplier u (weight constraint):", u)
print("Lagrange Multiplier v (return constraint):", v)

# Validate Results
portfolio_variance = optimal_weights.T @ Sigma @ optimal_weights
portfolio_return = mu @ optimal_weights
print("Portfolio Variance:", portfolio_variance)
print("Portfolio Return:", portfolio_return)
```

A.4 Оптимізація портфеля Марковіца у другому формулюванні

```
import numpy as np

# Inputs
#Data for Netflix, Apple and Tesla (6 months)
#Sigma = np.array([
# [10111.99833112, 924.87024263, 7710.62359481],
# [924.87024263, 118.43629114, 746.90147421],
# [7710.62359481, 746.90147421, 6890.71035732]
```

```
#])

# Constructing a matrix where the third row is a linear combination of the first two (based on Netflix, Apple
    and Tesla (6 months))
#Sigma = np.array([
# [10111.99833112, 924.87024263, 7710.62359481],
# [924.87024263, 118.43629114, 746.90147421],
# [0.5*10111.99833112 + 0.5*924.87024263, 0.5*924.87024263 + 0.5*118.43629114, 0.5*7710.62359481 +
    0.5*746.90147421]
#])

#A correlation matrix for (Netflix)1Y, (Amazon)1Y, (Apple)1Y, (Tesla)1Y
#Sigma = np.array([
# [11597.19304614, 1661.30380793, 2257.51964983, 7857.44746527],
# [1661.30380793, 297.72835567, 294.85220833, 1208.21528825],
# [2257.51964983, 294.85220833, 666.99570252, 1621.2992109 ],
# [7857.44746527, 1208.21528825, 1621.2992109 , 6362.09456258]
#])

# Constructing the modified matrix where the fourth row is a linear combination of the first three
#Sigma = np.array([
# [11597.19304614, 1661.30380793, 2257.51964983, 7857.44746527],
# [1661.30380793, 297.72835567, 294.85220833, 1208.21528825],
# [2257.51964983, 294.85220833, 666.99570252, 1621.2992109],
# [0.5*11597.19304614 + 0.3*1661.30380793 + 0.2*2257.51964983,
# 0.5*1661.30380793 + 0.3*297.72835567 + 0.2*294.85220833,
# 0.5*2257.51964983 + 0.3*294.85220833 + 0.2*666.99570252,
# 0.5*7857.44746527 + 0.3*1208.21528825 + 0.2*1621.2992109]
#])
#Sigma = np.array([
# [19646.23270048, 2202.96775854, 9318.25876834, 2036.239968],
# [2202.96775854, 366.26548986, 1430.73080533, 245.93092814],
# [9318.25876834, 1430.73080533, 6969.39617976, 1358.97427055],
# [2036.239968, 245.93092814, 1358.97427055, 463.27231104]
#])
mu = np.array([769.64404, 196.77032, 265.6184, 220.06491999999997]) # Expected returns for Netflix, Apple and
    Tesla (6 months)

Sigma = np.array([
    [19646.23270048, 2202.96775854, 9318.25876834, 2036.239968],
    [2202.96775854, 366.26548986, 1430.73080533, 245.93092814],
    [9318.25876834, 1430.73080533, 6969.39617976, 1358.97427055],
    [19646.23270048, 2202.96775854, 9318.25876834, 2036.239968] # Duplicate of the first row
])

#mu = np.array([0.1, 0.08]) # Expected returns for each asset
#Sigma = np.array([
# [0.04, 0.02],
# [0.02, 0.03]
#])

det = np.linalg.det(Sigma)
print("Determinant of Sigma:", det)


#mu = np.array([764.9979199999999, 230.51456,289.49368]) # Expected returns for Netflix, Apple and Tesla (6
    months)
mu = np.array([689.7190800000001, 188.0075, 209.70308000000003, 238.36283999999998]) # Expected returns
tau = 4000 # Risk-aversion parameter
```

```
ones = np.array([1, 1, 1, 1]) # Vector of ones
#ones = np.array([1, 1]) # Vector of ones


#Not real data with singular cov matrix
#Sigma = np.array([
# [0.0004, 0.0003, 0.00035],
# [0.0003, 0.0005, 0.0004],
# [0.00035, 0.0004, 0.000375]
#])
#mu = np.array([0.12, 0.08, 0.06]) # Expected returns


# Step 1: Inverse or Pseudoinverse of Sigma
Sigma_inv = np.linalg.pinv(Sigma)
#Sigma_inv = np.linalg.inv(Sigma)

# Step 2: Compute the first term (minimum variance portfolio)
Sigma_inv_ones = Sigma_inv @ ones
ones_T_Sigma_inv_ones = ones.T @ Sigma_inv @ ones
first_term = Sigma_inv_ones / ones_T_Sigma_inv_ones

# Step 3: Compute the second term (risk-adjusted return)
#np.outer is a function in NumPy that computes the outer product of two vectors.
R = Sigma_inv - np.outer(Sigma_inv_ones, Sigma_inv_ones) / ones_T_Sigma_inv_ones
second_term = (1 / (2 * tau)) * R @ mu

# Step 4: Combine both terms
x = first_term + second_term

#Additional Constraint test
# Ensure non-negative weights with iterative adjustment
#negative_sum = np.sum(x[x < 0]) # Total sum of the negative elements
#x[x < 0] = 0 # Any weight that was negative is set to 0

#if np.sum(x) > 0: # Avoid division by zero
    #x += (negative_sum / np.sum(x)) * x # Redistribute the removed weights

#alternative approach
x[x < 0] = 0 # Remove negatives
x = x / np.sum(x) # Normalize to sum to 1

# Display results
print("Inverse of Sigma (Sigma_inv):")
print(Sigma_inv)
print("\nFirst Term (Minimum Variance Portfolio):")
print(first_term)
print("\nSecond Term (Risk-Adjusted Return Component):")
print(second_term)
print("\nOptimal Portfolio Weights:")
print(x)
```

A.5 Обрахунок коефіцієнта кореляції Пірсона та виведення графіків цін акцій компаній

```python
from google.colab import files
import pandas as pd
import numpy as np

# Upload files
print("Please upload your datasets.")
uploaded = files.upload()

# Function to load and process data
def load_and_process_data(file_name):
    # Load the dataset
    data = pd.read_csv(file_name)

    # Print the first few rows (head) of the dataset
    print(f"\nHead of dataset '{file_name}':")
    print(data.head()) # Print the first 5 rows

    # Remove dollar signs and convert to numeric
    data['Close/Last'] = data['Close/Last'].replace({'\$': ''}, regex=True).astype(float)

    # Ensure the 'Close/Last' column is numeric
    data['Close/Last'] = pd.to_numeric(data['Close/Last'], errors='coerce')

    # Drop rows with missing values in 'Close/Last' after conversion
    data = data.dropna(subset=['Close/Last'])

    # Return the closing prices as an array
    return data['Close/Last'].values

# List to store the closing price arrays
prices = []

# Load and process each dataset, dynamically
for file_name in uploaded.keys():
    closing_prices = load_and_process_data(file_name)
    prices.append(closing_prices)

# Find the minimum length among all datasets
min_length = min(len(prices[0]), *[len(p) for p in prices[1:]])

# Trim each dataset to the determined minimum length
prices = [p[:min_length] for p in prices]

# Compute and print the mean value for each dataset
for i, price in enumerate(prices):
    mean_value = np.mean(price)
    print(f'Mean value for dataset {i+1}: {mean_value}')

# Compute the Pearson correlation coefficient for each pair of datasets
for i in range(len(prices)):
    for j in range(i + 1, len(prices)):
        correlation = np.corrcoef(prices[i], prices[j])[0, 1]
        print(f'Pearson Correlation Coefficient between dataset {i+1} and dataset {j+1}: {correlation}')

# Compute and print the covariance matrix
# Stack the datasets to create a 2D array where each row represents a dataset
data_matrix = np.vstack(prices)
cov_matrix = np.cov(data_matrix)
```

```
print("\nCovariance Matrix:")
print(cov_matrix)
```

A.6 Симуляція наведеної теореми

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import f, chi2, norm

def simulation(cov_matrix, n, r, alpha, num_simulations=1000):
    """
    Simulates the theorem using the provided covariance matrix needed amount of times and visualizes results.
    """
    k = cov_matrix.shape[0]

    # Compute GMV Variance (assuming 1_k is a vector of ones of size k)
    ones_k = np.ones((k, 1))
    V_GMV_plus = 1 / (ones_k.T @ np.linalg.pinv(cov_matrix) @ ones_k)[0, 0]

    R_EU_plus_values = []
    V_EU_plus_values = []

    for _ in range(num_simulations):
        # Generate random variables
        xi = f.rvs(dfn=r-1, dfd=n-r+1) # F-distributed
        eta = chi2.rvs(df=n-r) # Chi-square distributed
        z0 = norm.rvs() # Standard normal

        # Compute \hat{R}^{+}_{EU}
        term1 = alpha**(-1) * ((n-1)*(r-1)) / (n-r+1) * xi
        term2 = np.sqrt((1/n) * (1 + (r-1)/(n-r+1) * xi)) * np.sqrt(V_GMV_plus) * z0
        R_EU_plus = V_GMV_plus + term1 + term2

        # Compute \hat{V}^{+}_{EU}
        V_EU_plus = (V_GMV_plus / (n-1)) * eta + alpha**(-2) * ((n-1)*(r-1)) / (n*(n-r+1)) * xi

        R_EU_plus_values.append(R_EU_plus)
        V_EU_plus_values.append(V_EU_plus)

    # Plot histograms
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.hist(R_EU_plus_values, bins=30, alpha=0.7, color='b', edgecolor='black')
    plt.title(r"Histogram of $\hat{R}_{EU}^{+}$")
    plt.xlabel("Value")
    plt.ylabel("Frequency")

    plt.subplot(1, 2, 2)
    plt.hist(V_EU_plus_values, bins=30, alpha=0.7, color='r', edgecolor='black')
    plt.title(r"Histogram of $\hat{V}^{+}_{EU}$")
    plt.xlabel("Value")
    plt.ylabel("Frequency")

    plt.tight_layout()
```

```
    plt.show()

#Data for Netflix, Apple and Tesla (6 months)
#cov_matrix_example = np.array([
# [10111.99833112, 924.87024263, 7710.62359481],
# [924.87024263, 118.43629114, 746.90147421],
# [7710.62359481, 746.90147421, 6890.71035732]
#])

cov_matrix_example = np.array([
    [10111.99833112, 924.87024263, 10111.99833112 * 0.75],
    [ 924.87024263, 118.43629114, 924.87024263 * 0.75],
    [ 7710.62359481, 746.90147421, 7710.62359481 * 0.75]
])
print(np.linalg.det(cov_matrix_example))

n_example = 50
r_example = 2
alpha_example = 1.5

simulation(cov_matrix_example, n_example, r_example, alpha_example)
```

listings xcolor

Список літератури

1. Markowitz, H. *Portfolio Selection.* Journal of Finance, Vol. 7, No. 1 (1952), pp. 77–91.

2. Bodnar, T.; Mazur, S.; Nguyen, H. *Estimation of Optimal Portfolio Compositions for Small Sample and Singular Covariance Matrix.* Örebro University School of Business, Working Paper 15/2022. ISSN 1403-0586.

3. Stoer, J.; Bulirsch, R. *Introduction to Numerical Analysis.* 3rd ed. Berlin / New York: Springer-Verlag, 2002. ISBN 978-0-387-95452-3.

4. University of Chicago. *Lecture 8: Principal Components and Singular Value Decomposition.* Available at: `https://www.stat.uchicago.edu/~lekheng/courses/302/notes8.pdf`

5. SciCoding. *Cholesky Decomposition – In-Depth Guide for Scientists and Engineers.* Available at: `https://scicoding.com/cholesky-decomposition-in-depth-guide-for-scientists-and-engineers/`

6. Emory University. *Section 8.3 – Numerical Linear Algebra Lecture Notes.* Available at: `https://math.emory.edu/~lchen41/teaching/2020_Fall/Section_8-3.pdf`

7. MIT Mathematics. *Lecture Notes on Singular Value Decomposition.* 2016 IAP Course 18.095. Available at: `https://math.mit.edu/classes/18.095/2016IAP/lec2/SVD_Notes.pdf`

8. ScienceDirect. *Singular Value Decomposition – Topics in Mathematics.* Available at: `https://www.sciencedirect.com/topics/mathematics/singular-value-decomposition`

9. ResearchGate. *The Significance of Portfolio Management in Investment and Financial Decisions.* Available at: `https://www.researchgate.net/publication/379892080_The_Significance_of_Portfolio_Management_in_Investment_and_Financial_Decisions`

10. Lecture Notes. *Introduction to Portfolio Theory – Stochastic Financial Math.*

11. Li, G. *Portfolio Optimization and Risk Analysis in Financial Markets.* Advances in Economics, Management and Political Sciences, Vol. 61 (2023), pp. 236–246. DOI: 10.54254/2754-1169/61/20231273.

12. Ben-Israel, A.; Greville, T. N. E. *Generalized Inverses: Theory and Applications.* 2nd ed. New York: Springer, 2003 (CMS Books in Mathematics). ISBN 978-0-387-00293-4. DOI: https://doi.org/10.1007/b97366.

13. Barry, C. B. *Portfolio Analysis under Uncertain Means, Variances, and Covariances.* Journal of Finance, Vol. 29 (1974), pp. 515–522.

14. GeeksforGeeks. *Cross-Correlation Analysis in Python.* Available at: https://www.geeksforgeeks.org/cross-correlation-analysis-in-python/ [Accessed: 26 May 2025].

15. Newcastle University. *Strength of Correlation.* Available at: https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/strength-of-correlation.html [Accessed: 26 May 2025].

16. Stanford University. *Lecture Notes: Risk-Averse Portfolio Optimization.* EE365 – Modern Convex Optimization. Available at: https://stanford.edu/class/ee365/lectures/risk_averse.pdf [Accessed: 26 May 2025].