Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра оптимального керування і економічної кібернетики

# Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

«Дослідження математичних моделей оптимізації портфелю цінних паперів»

«Research of mathematical models of securities portfolio optimization»

Виконала: здобувачка денної форми навчання
спеціальності 113 Прикладна математика
Освітня програма «Прикладна математика»
Казарян Лаура Артурівна

Керівник: кандидат фіз.-мат. наук, доц. Васильєв О. Б.
Рецензент: доктор фіз.-мат. наук, доц. Кічмаренко О. Д.

Рекомендовано до захисту:

Протокол засідання кафедри

№ ___ від _____2023 р.

Завідувач кафедри

_____ _____

(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК №_____

протокол № ____ від _____ 2023 р.

Оцінка_____/___ ___/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ _____

(підпис) (прізвище, ініціали)

Одеса – 2023

# Зміст

Вступ ..... 3
1 Оптимізація портфелю цінних паперів двох активів, зведенням до одновимірної мінімізації функції ..... 4
1.1 Задача мінімізації ризику при визначеному рівні прибутку ..... 6
1.2 Задача максимізації прибутку при визначеному рівні ризику ..... 7
1.3 Методи розв’язання ..... 8
1.3.1 Метод бісекції ..... 8
1.3.2 Метод січних ..... 9
1.3.3 Метод Ньютона ..... 10
1.4 Приклади розв’язання задач ..... 11
2 Задачі Г. Марковица з оптимізації портфелю цінних паперів методами безумовної мінімізації функцій ..... 17
2.1 Математичні моделі безумовної мінімізації для вирішення задач Г. Марковіца ..... 19
2.1.1 Задача мінімізації ризику при визначеному рівні прибутку ..... 19
2.1.2 Задача максимізації прибутку при визначеному рівні ризику ..... 22
2.2 Вирішення задачі безумовної мінімізації функції за допомогою виключення двох змінних ..... 23
2.3 Алгоритми методів безумовної мінімізації функцій, застосованих у дослідженні ..... 24
2.3.1 Метод спряжених градієнтів ..... 24
2.3.2 Метод Ньютона ..... 25
2.4 Порівняльний аналіз математичних моделей і методів безумовної мінімізації функцій на прикладах портфелів цінних паперів ..... 26
Висновки ..... 38
Додатки ..... 39
Література ..... 62

# Вступ

У сучасному світі, де фінансові ринки постійно зростають і розвиваються, оптимізація портфелю цінних паперів стає важливим завданням для інвесторів, фондів і фінансових установ. Завдяки високому рівню конкуренції та швидкому темпу змін, раціональне розподілення активів у портфелі є ключовим фактором успіху на фондовому ринку.

Оптимізація портфелю цінних паперів полягає у виборі найефективнішого способу розподілу активів з метою досягнення максимального прибутку або мінімального ризику.

Тема дослідження актуальна тому, що подібні дослідження допомагають інвесторам та фінансовим установам ефективно керувати своїми інвестиціями, забезпечуючи оптимальне поєднання ризику і прибутку. Застосування математичних моделей дозволяє враховувати різноманітні фактори, такі як ризик і дохідність, що дозволяє побудувати ефективні портфелі, що відповідають поставленим цілям. Математичні моделі дозволяють здійснювати аналіз різних варіантів розподілу активів та визначати оптимальний портфель, що відповідає конкретним фінансовим цілям та обмеженням інвестора.

Одним з основних викликів при оптимізації портфелю є пошук балансу між доходністю та ризиком. Інвесторам потрібно знайти оптимальну точку, де очікувана доходність портфелю максимізується при заданому рівні ризику. Це може бути складно досягти без використання математичних моделей, які дозволяють враховувати різні фактори та обмеження для побудови оптимального рішення.

Таким чином, дослідження математичних моделей оптимізації портфелю цінних паперів є актуальною та важливою темою, яка має практичне застосування у реальному фінансовому середовищі.

# Розділ 1

## 1 Оптимізація портфелю цінних паперів двох активів, зведенням до одновимірної мінімізації функції

Припустимо, що ми маємо суму грошей $M$, яку потрібно поділити між двома керованими фондами, котрі заявляють про ймовірну норму прибутку (у відсотках) $r_1$ та $r_2$. Якщо ми інвестуємо суми $y_1$ та $y_2$, то очікується, що загальний дохід становитиме:

$$R = \frac{r_1 y_1 + r_2 y_2}{M}\%. \tag{1.1}$$

Середня прибутковість $\overline{r_i}$ для кожного активу розраховується за формулою:

$$\overline{r_i} = \frac{\sum_{j=1}^{m} r_{ij}}{m}, \tag{1.2}$$

де $r_{ij}(i = 1,\ldots,n;\ j = 1,\ldots,m)$ позначає прибуток від активу $i$ за період часу $j$.

Таким чином, передбачуваний дохід портфеля:

$$R = \sum_{i=1}^{n} \overline{r_i} y_i \tag{1.3}$$

Цю формулу можна записати зручніше за допомогою матрично-векторного позначення

$$R = \overline{r}^T y, \tag{1.4}$$

де $\overline{r}$ позначає вектор-стовпець $\left(\overline{r_1}, \overline{r_2}, \ldots, \overline{r_n}\right)^T$, а $y-$ це вектор-стовпець, який складається з інвестованих часток $\left(y_1, y_2, \ldots, y_n\right)^T$.

Ризик, пов’язаний з конкретним портфелем, визначається за дисперсіями та коваріаціями, які можна розрахувати на основі історії доходів $r_{ij}$. Дисперсія актива $i$:

$$\sigma_i^2 = \frac{\sum_{j=1}^{m}\left(r_{ij} - \overline{r}_i\right)^2}{m}. \tag{1.5}$$

Коваріація активів $i$ та $k$:

$$\sigma_{ik} = \frac{\sum_{j=1}^{m}\left(r_{ij} - \overline{r}_i\right)\left(r_{kj} - \overline{r}_k\right)}{m} \tag{1.6}$$

Дисперсія портфеля, яка визначається інвестованими частками $y_1, y_2, \ldots, y_n$:

$$V = \sum_{i=1}^{n}\sigma_i^2 y_i^2 + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sigma_{ij} y_i y_j, \tag{1.7}$$

яка може бути використана як міра ризику портфеля.

Формулу (1.7) можна записати у вигляді:

$$V = y^T Q y, \tag{1.8}$$

де дисперсійно-коваріаційна матриця $Q$ позначається як:

$$Q = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \ldots & \sigma_{1n} \\ \sigma_{12} & \sigma_{22} & \ldots & \sigma_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ \ldots & \sigma_{ij} & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots \\ \sigma_{1n} & \sigma_{2n} & \ldots & \sigma_{nn} \end{pmatrix} \tag{1.9}$$

(використовуємо еквівалентне позначення $\sigma_{ii} = \sigma_i^2$).

Якщо ми розподілимо інвестиції на $n$ активів, і якщо $y_i$ є часткою, вкладеною в актив $i$, то значення $y_i$ визначають портфель. Оскільки всі інвестиції повинні бути розподілені поміж $n$ активів, інвестовані частки повинні задовольняти

$$S = \sum_{i=1}^{n} y_i = 1. \tag{1.10}$$

## 1.1 Задача мінімізації ризику при визначеному рівні прибутку

Основною проблемою при виборі портфеля є мінімізація ризику. Зафіксуємо цільове значення очікуваного прибутку $R_p$%, і, користуючись формулами (1.4), (1.8) і (1.10), отримаємо задачу

$$\begin{gathered}\text{Minimize} \quad V = y^T Q y \\ \text{за умови} \quad \sum_{i=1}^{n} \bar{r}_i y_i = R_p \quad \text{і} \quad \sum_{i=1}^{n} y_i = 1.\end{gathered} \tag{1.1.1}$$

Один із способів спробувати визначити портфель з найменшим ризиком при заданому рівні прибутку — це розглянути складену функцію, таку як:

$$F = V + \rho R = y^T Q y + \rho\left(\bar{r}^T y\right), \tag{1.1.2}$$

де перший доданок є величиною найменшого ризику, а другий — множником прибутку. Позитивна константа $\rho$ контролює баланс між ризиком та прибутковістю.

Зробимо модифіковану версію задачі (1.1.1), користуючись складеною функцією (1.1.2):

$$\text{Minimize} \quad F = y^T Q y + \frac{\rho}{R_p^2}\left(\sum_{i=1}^{n} \bar{r}_i y_i - R_p\right)^2 \quad \text{за умови} \quad \sum_{i=1}^{n} y_i = 1. \tag{1.1.3}$$

У мінімумі функції $F$ можна очікувати, що ризик $y^T Q y$ буде маленьким, а також, що значення прибутку $\bar{r}^T y$ буде близьким до значення $R_p$.

Таким чином, отримали лише одну умову в (1.1.3). Намагатимемося позбавитися її для більш легкого розв’язання задачі, тобто усунемо $y_i$ і отримаємо задачу мінімізації без обмежень, яка буде включати в себе лише $y_1, y_2, \ldots, y_n$.

У випадку з двома активами, ми маємо обмеження $y_1 + y_2 = 1$. Якщо ми позначимо $y_1$ як $x$, тоді $y_2 = 1 - y_1 = 1 - x$. Тепер можемо визначити

$$\alpha = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \beta = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{1.1.4}$$

З цього випливає, що $y = \alpha + \beta x,$ і задача (1.1.3) приймає наступний вигляд:

$$\text{Minimize} \quad F = (\alpha + \beta x)^T Q(\alpha + \beta x) + \frac{\rho}{R_p^2}\left(\bar{r}^T \alpha - R_p + \bar{r}^T \beta x\right)^2. \qquad (1.1.5)$$

## 1.2 Задача максимізації прибутку при визначеному рівні ризику

Ще однією проблемою є максимізація прибутку. Зафіксуємо прийнятний рівень ризику $V_a$, а потім максимізуємо очікувану прибутковість. Тоді, за допомогою формул (1.4), (1.8) і (1.10), отримуємо задачу:

$$\begin{gathered} \text{Maximize} \quad R = \bar{r}^T y \\ \text{за умови} \quad V = y^T Q y = V_a \quad \text{та} \quad \sum_{i=1}^{n} y_i = 1. \end{gathered} \qquad (1.2.1)$$

(1.2.1) можна представити як задачу умовної мінімізації:

$$\begin{gathered} \text{Minimize} \quad R = -\bar{r}^T y \\ \text{за умови} \quad V = y^T Q y = V_a \quad \text{та} \quad \sum_{i=1}^{n} y_i = 1. \end{gathered} \qquad (1.2.2)$$

Один із способів спробувати визначити портфель з «найбільшим прибутком при заданому прийнятному ризику» — це розглянути складену функцію, таку як:

$$F = -R + \rho V, \qquad (1.2.3)$$

де перший доданок є від'ємною величиною очікуваного доходу, а другий — множником ризику. Позитивна константа $\rho$ контролює баланс між прибутковістю та ризиком.

Зробимо модифіковану версію задачі (1.2.2), користуючись складеною функцією (1.2.3):

$$\text{Minimize} \quad F = -\bar{r}^T y + \frac{\rho}{V_a^2}\left(y^T Q y - V_a\right)^2 \quad \text{за умови} \quad \sum_{i=1}^{n} y_i = 1. \qquad (1.2.4)$$

Можна очікувати, що рішення (1.2.4) відбудеться, коли $\bar{r}^T y$ велике та значення $y^T Q y$ є близьким до $V_a$.

Таким чином, отримали лише одну умову в (1.2.4). Намагатимемося

позбавитися її для більш легкого розв'язання задачі. Для цього використаємо (1.1.4) і отримаємо функцію

$$\text{Minimize} \quad F = -\bar{r}^T\alpha - \left(\bar{r}^T\beta\right)x + \frac{\rho}{V_a^2}\left[\alpha^T Q\alpha + 2\left(\beta^T Q\alpha\right)x + \left(\beta^T Q\beta\right)x^2 - V_a\right]^2 \quad (1.2.5)$$

## 1.3 Методи розв'язання

### 1.3.1 Метод бісекції

Просто (але неефективно) оцінити найменше значення $F(x)$ у діапазоні $a < x < b$ можна за допомогою обчислення функції в багатьох точках $[a,b]$ і вибором однієї з найменшим значенням. Кожна ітерація використовує порівняння між значеннями функції в п'яти точках, щоб зменшити вдвічі розмір проміжку, що містить мінімум. Цей метод знайде мінімум, якщо його застосувати до функції $F(x)$, яка є унімодальною, тобто має лише один мінімум у діапазоні $[a,b]$.

*Визначення проміжку*

Знайдемо діапазон $a < x < b$, який містить мінімум функції $F(x)$. Цей метод використовує нахил $F'$, щоб визначити, чи слід шукати мінімум ліворуч чи праворуч від початкової точки $x_0$. Якщо похідна $F'\left(x_0\right)$ є позитивною, то для $x < x_0$, будуть знайдені менші значення функції, тоді як $F'\left(x_0\right) < 0$ означає, що нижчі значення $F$ мають місце, коли $x > x_0$. Алгоритм просто робить все більші і більші кроки в напрямку «вниз» до тих пір, поки функція не почне зростати, що буде вказувати на те, що мінімум локалізован на $[a,b]$.

***Знаходження*** $a$ ***і*** $b$, ***щоб оцінити локальний мінімум*** $F(x)$

Choose an initial point $x_0$ and a step size $\alpha(>0)$

Set $\delta = -\alpha \times sign\left(F'\left(x_0\right)\right)$

**Repeat for** $k = 0,1,2,\ldots$

$$x_{k+1} = x_k + \delta,\ \delta = 2\delta \quad (1.3.1.1)$$

**until** $F\left(x_{k+1}\right) > F\left(x_k\right)$

if $k = 0$ then set $a = x_0$ and $b = x_1$

if $k > 0$ then set $a = x_{k-1}$ and $b = x_{k+1}$

***Метод бісекції для мінімізації*** $F(x)$ ***на проміжку*** $[a,b]$

Set $x_a = a$ and $x_m = \frac{1}{2}(a+b)$

Calculate $F_a = F\left(x_a\right),\ F_b = F\left(x_b\right),\ F_m = F\left(x_m\right)$

**Repeat**

set $x_l = \frac{1}{2}\left(x_a + x_m\right),\ x_r = \frac{1}{2}\left(x_m + x_b\right)$

calculate $F_l = F\left(x_l\right)$ and $F = F\left(x_r\right)$

let $F_{\min} = \min\left\{F_a, F_b, F_m, F_l, F_r\right\}$

if $F_{\min} = F_a$ or $F_l$ then set $x_b = x_m,\ x_m = x_l,\ F_b = F_m,\ F_m = F_l$

else if $F_{\min}$=$F_m$ then set $x_a = x_l,\ x_b = x_r,\ F_a = F_l,\ F_b = F_r$

else if $F_{\min}$=$F_r$ or $F_b$ then set $x_a = x_m,\ x_m = x_r,\ F_a = F_m,\ F_m = F_r$

**until** $\left|x_b - x_a\right|$ is sufficiently small

Будемо стверджувати, що якщо $F(x)$ є унімодальною на $a \le x \le b$ з мінімумом в точці $x^*$, тоді кількість ітерацій, необхідних методу бісекції для того, щоб знайти $x^*$ в межах проміжку розміром менше ніж $10^{-s}$, дорівнює $K$:

$$\frac{\log_{10}(b-a)+s}{\log_{10}(2)} \tag{1.3.1.2}$$

## 1.3.2 Метод січних

Розглянемо ітераційний метод розв’язання $F'(x) = 0$. Це дозволить знайти локальний мінімум $F(x)$ за умови, якщо ми використовуємо його в області, де друга похідна $F''(x)$ залишається додатною. Підхід заснований на лінійній інтерполяції. Якщо оцінити $F'$ в двох точках $x = x_1$ і $x = x_2$, тоді обчислення

$$x = x_1 - \frac{F'\left(x_1\right)}{F'\left(x_2\right) - F'\left(x_1\right)}\left(x_2 - x_1\right) \tag{1.3.2.1}$$

дає оцінку точки, де $F'$ обертається в нуль. Формула (2.2.1) працює, якщо $F(x)$ є квадратичною функцією. Однак, якщо $F$ не є квадратичною, необхідно застосувати формулу інтерполяції ітераційно.

***Метод січної для розв'язання*** $F'(x)=0$

Choose $x_0$, $x_1$ as two estimates of the minimum of $F(x)$
**Repeat** for $k=0,1,2,\ldots$

$$x_{k+2}=x_k-\frac{F'(x_k)}{F'(x_{k+1})-F'(x_k)}(x_{k+1}-x_k)$$

**until** $\left|F'(x_{k+2})\right|$ is sufficiently small.

В цьому алгоритмі повторно використовується формула (2.2.1), на основі двох останніх розрахованих точок. Коли $k>1$, зазвичай ми будемо розраховувати $x_{k+2}$ користуючись $x_{k+1}$ разом з $x_k$ або $x_{k-1}$ згідно з однією із стратегій:

a. Обирається будь яка змінна з $x_k$ та $x_{k-1}$, яка дає найменше значення $|F'|$.
b. Обирається будь яка змінна з $x_k$ та $x_{k-1}$, яка дає значення $F'$ з протилежним знаком до $F'(x_{k+1})$.
c. Обирається будь яка змінна з $x_k$ та $x_{k-1}$, яка дає найменше значення $F$.

Стратегія (a) і (c) базуються на використанні точок, які здаються ближчими до мінімуму. Стратегія (b) може бути застосована тільки у тому випадку, якщо обрані нами початкові $x_0$ і $x_1$ дають різні знаки для $F'(x_0)$ та $F'(x_1)$.

## 1.3.3 Метод Ньютона

Цей метод шукає мінімум $F(x)$ користуючись першою і другою похідною. В своїй найпростішій формі цей метод має наступний вигляд:

***Метод Ньютона для мінімізації*** $F(x)$

Choose $x_0$ as an estimate of the minimum of $F(x)$
**Repeat** for $k=0,1,2,\ldots$

$$x_{k+1}=x_k-\frac{F'(x_k)}{F''(x_{k+1})} \tag{1.3.3.1}$$

**until** $\left|F'\left(x_{k+1}\right)\right|$ is sufficiently small.

Результат збіжності методу залежить від поведінки $F''\left(x\right)$, тому практичний алгоритм повинен містити гарантії від розбіжностей. Ми можемо використовувати (2.3.1) тільки у випадках, коли $F''\left(x\right)$ є чітко позитивною. Але навіть тоді потрібно перевіряти нові точки, утворені методом Ньютона, чи є вони «кращими» за старі. Ці ідеї включені в наступний алгоритм:

***Безпечний метод Ньютона для мінімізації*** $F(x)$ ***на*** $[a,b]$

Make a quess $x_0$ $\left(a < x_0 < b\right)$ for the minimum of $F(x)$
**Repeat** for $k = 0,1,2,\ldots$
if $F''(x_k) > 0$ then $\delta x = -F'\left(x_k\right)/F''\left(x_k\right)$
else $\delta x = -F'\left(x_k\right)$
if $\delta x < 0$ then $\alpha = \min\left(1,\ \left(a - x_k\right)/\delta x\right)$
if $\delta x > 0$ then $\alpha = \min\left(1,\ \left(b - x_k\right)/\delta x\right)$
**Repeat** for $\mathrm{j} = 0,1,\ldots$
$\alpha = 0.5^j \alpha$
**until** $F\left(x_k + \alpha\delta x\right) < F\left(x_k\right)$
Set $x_{k+1} = x_k + \alpha\delta x$
**until** $\left|F'\left(x_{k+1}\right)\right|$ is sufficiently small.

Окрім надання альтернативного вибору $\delta x$ коли $F'' \le 0$, безпечний алгоритм Ньютона включає в себе розмір кроку $\alpha$. Він обирається, спочатку, щоб запобігти виходу кроків корекції за межі проміжку $[a,b]$, а потім, за допомогою повторного ділення навпіл, щоб переконатися, що кожна нова точка має менше значення функції $F$, ніж попередня. Алгоритм завжди спочатку намагається виконати повний крок ($\alpha = 1$), отже, він може мати таку ж швидку кінцеву збіжність, що й основний метод Ньютона.

## 1.4 Приклади розв’язання задач

Розглянемо наступну таблицю:

| | Дохід % | | | | | |
|---|---|---|---|---|---|---|
| | січень | лютий | березень | квітень | травень | червень |
| актив 1 | 1,2 | 1,3 | 1,4 | 1,5 | 1,1 | 1,2 |
| актив 2 | 1,3 | 1 | 0,8 | 0,9 | 1,4 | 1,3 |

*Таблиця 1.4.1*

Спочатку розрахуємо середню прибутковість $\overline{r}$ за формулою (1.2):

$$\overline{r} = \begin{pmatrix} 1,2833 \\ 1,1167 \end{pmatrix}$$

За допомогою формул (1.5) і (1.6), сформуємо дисперсійно-коваріаційну матрицю (1.9):

$$Q = \begin{pmatrix} 0,0181 & -0,0281 \\ -0,0281 & 0,0514 \end{pmatrix}$$

Знаючи $\alpha$ і $\beta$ з (1.1.4), сформуємо функцію мінімізації (1.2.5):

$$F = -1,1167 - 0,1667x + \frac{\rho}{V_a^2}\left[0,0514 - 0,1589x + 0,1256x^2 - V_a^2\right]^2 \qquad (1.4.1)$$

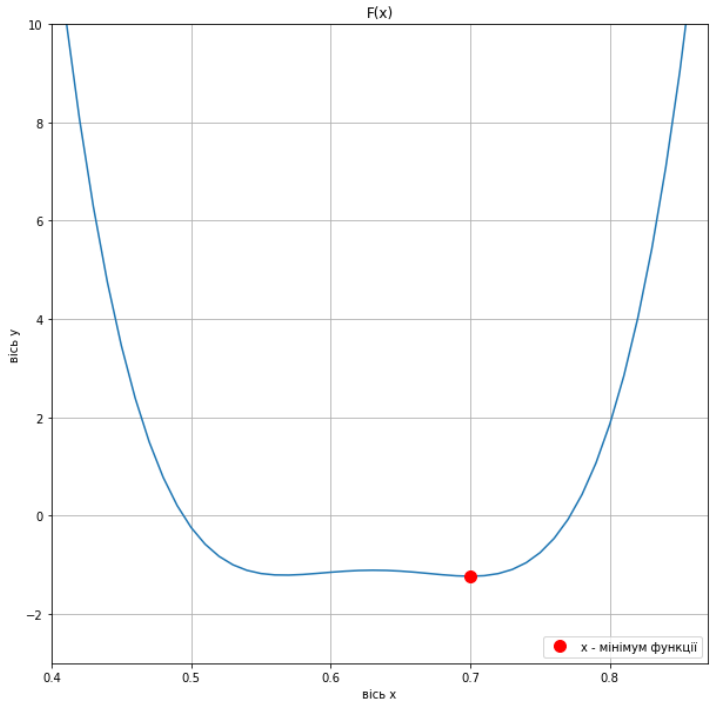
В якості прикладу, покладемо $\rho = 1$, $V_a = 0,00168$, $\alpha = 0,01$, $[a,b] = [0;\ 1]$.

Таблиця результатів обчислень кожного методу за допомогою програми, написаної на Python 3.8 (додаток А):

| метод | ітерації | $y_1$ | $y_2$ | $V$ | $R$ |
|---|---|---|---|---|---|
| Бісекції | 14 | 0,7 | 0,3 | 0,00168 | 1,23% |
| Січних | 10 | 0,7 | 0,3 | 0,00168 | 1,23% |
| Ньютона | 7 | 0,7 | 0,3 | 0,00168 | 1,23% |

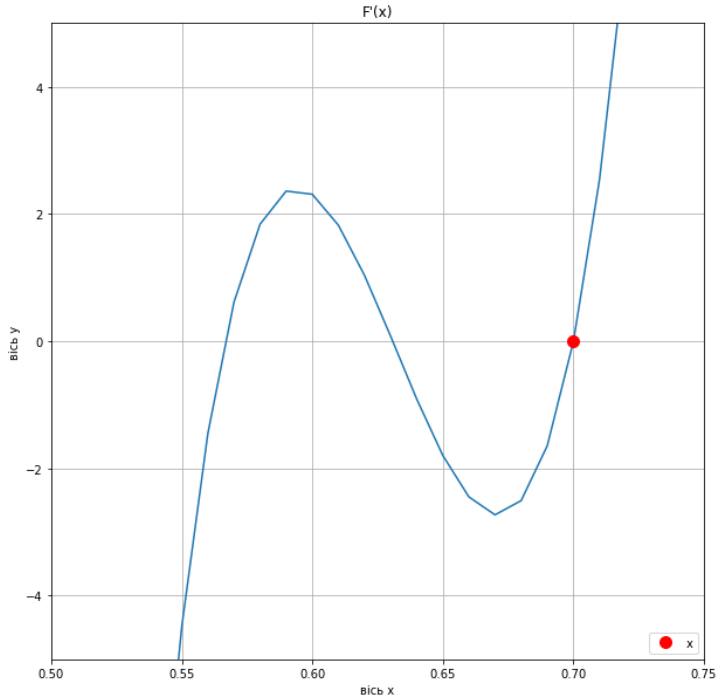
*Таблиця 1.4.2*

Графік функції (3.1):




*Мал. 1.4.1*

Графік похідної від функції (3.1):




*Мал. 1.4.2*

Ми бачимо, що всі три методи мінімізації знаходять однакове рішення, але використовують різну кількість ітерацій.

Роблячи висновки з порівняння приведених результатів в таблиці (3.2), важливо переконатися, що всі методи використовували однакові (або подібні)

початкові припущені рішення та критерії збіжності. У наведених вище результатах в методі бісекції і в методі січної були задані початкові значення $[a,b]=[0,1]$. Метод Ньютона було розпочато з $x=1$ (йому потрібна лише одна початкова точка). Ітерації бісекції завершуються, коли проміжок, що містить оптимум, зменшений до ширини, меншої за $10^{-4}$. Збіжність методів січної і Ньютона відбувається при градієнті $|F'(x)|<10^{-5}$. Тому робимо висновок, що метод Ньютона дійсно ефективніший, ніж метод січної, який, у свою чергу, кращий за метод бісекції.

Тепер, розглянемо іншу задачу

| | Дохід % | | | | |
|---|---|---|---|---|---|
| | день 1 | день 2 | день3 | день 4 | день 5 |
| Актив 1 | 0,15 | 0,26 | 0,18 | 0,04 | 0,06 |
| Актив 2 | 0,04 | -0,07 | -0,05 | 0,07 | 0,03 |

*Таблиця 1.4.3*

щоб побачити, чи буде поведінка методів схожою.

Функція мінімізації для таблиці 3.3:

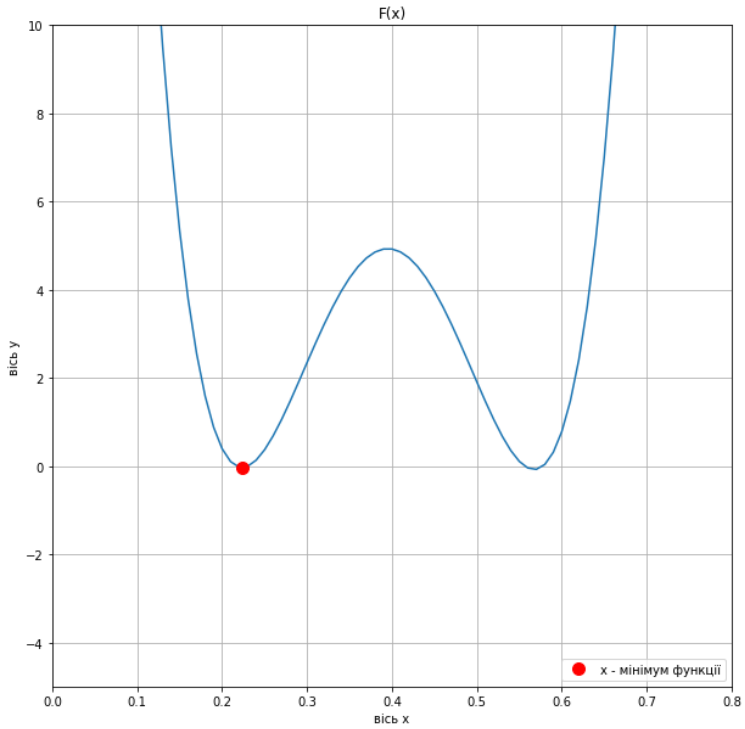
$$F=-0.004-0.134x+\frac{\rho}{V_a^2}\left[0.0029-0.0136x+0.0172x^2-V_a\right]^2 \qquad (1.4.2)$$

Покладемо $\rho=10,\ V_a=0,00072$, $\alpha=0,01$, $[a,b]=[0;\ 1]$ (додаток Б):

| метод | ітерації | $y_1$ | $y_2$ | $V$ | $R$ |
|---|---|---|---|---|---|
| Бісекції | 14 | 0,22 | 0,78 | 0,00072 | 0,034% |
| Січних | 6 | 0,22 | 0,78 | 0,00072 | 0,034% |
| Ньютона $(x_0=0)$ | 5 | 0,22 | 0,78 | 0,00072 | 0,034% |
| Ньютона $(x_0=1)$ | 7 | 0,564 | 0,436 | 0,00072 | 0,08% |

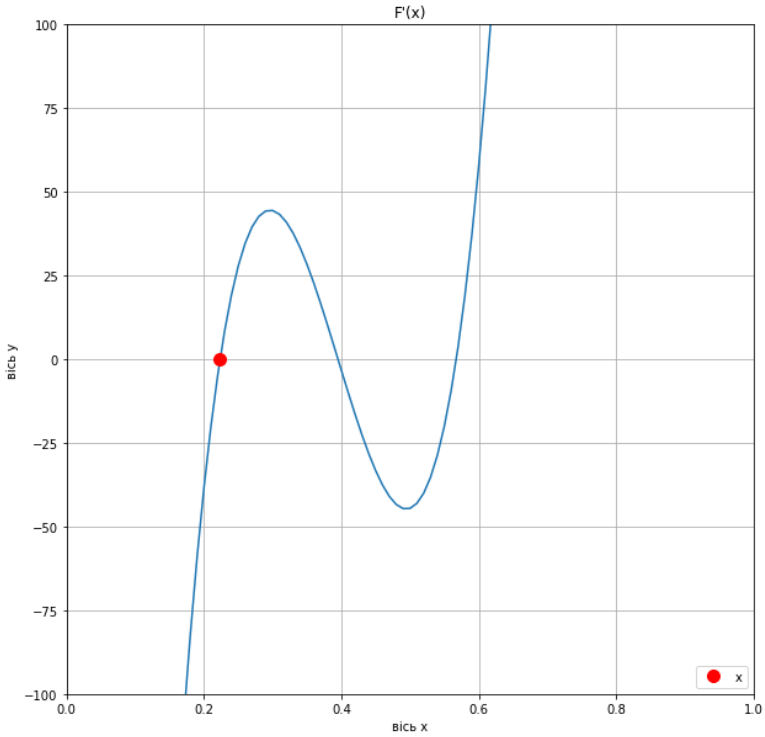
*Таблиця 1.4.4*

Графік функції (3.2):




*Мал. 1.4.3*

Графік похідної від функції (3.2):




*Мал. 1.4.4*

Перші три рядки таблиці 3.4 знову показують, що методи січної та Ньютона перевершують метод бісекції, знаходячи те саме рішення за меншу кількість ітерацій. Фактично, кількість кроків бісекції залежить лише від розміру

початкового проміжку та критерію збіжності, тоді як кількість ітерацій для методів січної та Ньютона може змінюватися від однієї задачі до іншої. Останній рядок таблиці 3.4 показує, що ми отримаємо інше рішення , якщо метод Ньютона розпочати з $x_0 = 1$ замість $x_0 = 0$. Це альтернативне рішення насправді краще, ніж рішення в перших трьох рядках таблиці, оскільки дохід $R$ є більшим. Проте, обидва розв'язки є дійсними локальними мінімумами (1.16), і тому можна сказати, що методам бісекції та січної «не пощастило» збігтися до меншого результату доходу.

# Розділ 2

## 2 Задачі Г. Марковица з оптимізації портфелю цінних паперів методами безумовної мінімізації функцій

Поки що ми мали справу лише з портфелями, що включають два активи, які призводять до задач оптимізації з однією змінною. Далі будемо розглядати портфелі з $n$ активами, які будуть призводити до задач оптимізації з $n$, $n-1$ і $n-2$ змінними.

Згадаємо позначення та термінологію для проблеми $n-$активів, що включає історію доходів за $m$ періодів часу. Якщо ми обчислили середню прибутковість $\bar{r}_i$ для кожного активу, а також дисперсії $\sigma_{ii}$ та коваріації $\sigma_{ij}$, тоді очікувана прибутковість портфеля, визначена інвестованими частками $y_1,\dots y_n$ визначається за формулою

$$R = \bar{r}^T y, \quad \text{де} \quad \bar{r} = \begin{pmatrix} \bar{r}_1 \\ \bar{r}_2 \\ .. \\ \bar{r}_n \end{pmatrix} \quad \text{і} \quad y = \begin{pmatrix} y_1 \\ y_2 \\ .. \\ y_n \end{pmatrix}. \tag{2.1}$$

Вираз для ризику можна записати як

$$V = y^T Q y, \tag{2.2}$$

де

$$Q = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \dots & \sigma_{1n} \\ \sigma_{12} & \sigma_{22} & \dots & \sigma_{2n} \\ \dots & \dots & \dots & \dots \\ \dots & \sigma_{ij} & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \sigma_{1n} & \sigma_{2n} & \dots & \sigma_{nn} \end{pmatrix} \tag{2.3}$$

Також згадаємо умову нормалізації $\sum_{i=1}^{n} y_i = 1$, яка може бути записана у вигляді

$$S = e^T y = 1, \quad \text{де} \quad e = \begin{pmatrix} 1 \\ 1 \\ .. \\ 1 \end{pmatrix}. \tag{2.4}$$

Ми можемо використати (2.4), щоб усунути $y_n$ і, отже, виразити прибутковість і ризик лише через $y_1, \ldots y_{n-1}$. Нехай $x$ є вектором-стовпцем з $x_i = y_i$, $(i = 1, \ldots, n-1)$. Ми також визначаємо $n$-вектор $\alpha$ і матрицю $B$ розмірністю $n \times (n-1)$ як

$$\alpha = \begin{pmatrix} 0 \\ 0 \\ .. \\ 0 \\ 1 \end{pmatrix} \quad \text{і} \quad B = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ .. & .. & .. & .. \\ 0 & 0 & \ldots & 1 \\ -1 & -1 & \ldots & -1 \end{pmatrix}. \tag{2.5}$$

Це розширення $n-$активів з (1.15), і легко побачити, що вони призводять до

$$y = \alpha + Bx. \tag{2.6}$$

Також, для подальшого розв'язання задач, введемо умови оптимальності. Умови, які характеризують мінімум неперервно-диференційованої функції $n-$змінних $F(x_1, \ldots x_n)$, виражаються через вектор перших частинних похідних

$$g = \left( \frac{\partial F}{\partial x_1}, \ldots, \frac{\partial F}{\partial x_n} \right)^T, \tag{2.7}$$

та через матрицю $G$ розмірністю $n \times n$ других частинних похідних, $(i, j)-$й елемент якої є

$$G_{ij} = \frac{\partial^2 F}{\partial x_i \partial x_j}. \tag{2.8}$$

**Означення.** Вектор $g$ з (2.7) називається *градієнтом* і може позначатися як $\nabla F$ (або іноді як $F_x$).

**Означення.** Матриця $G$ з (2.8) називається *матрицею Гессе* і може позначатися як $\nabla^2 F$ або $F_{xx}$.

Матриця Гессе завжди симетрична, коли $F$ є двічі неперервно-диференційованою функцією, оскільки

$$\frac{\partial^2 F}{\partial x_i \partial x_j} = \frac{\partial^2 F}{\partial x_j \partial x_i}.$$

**Означення.** Додатньо визначена симетрична матриця - це матриця, у якої усі власні значення є додатними. Еквівалентно, матриця А є додатньо визначеною тоді і тільки тоді, коли

$$x^T Ax > 0 \quad \forall x \neq 0. \tag{2.9}$$

**Теорема.** Якщо $F(x)$ є функцією $n-$змінних, градієнт і матриця Гессе якої задовольняють умовам

$$g\left(x^*\right) = 0 \quad \text{і} \quad G\left(x^*\right) \text{ є додатньо визначеною,} \tag{2.10}$$

Тоді точка $x^*$ є локальним мінімумом функції $F(x)$.

Саме друга з умов оптимальності (2.10) відрізняє мінімум від максимуму (або будь-якої іншої стаціонарної точки). З геометричної точки зору, додатна визначеність $G$ означає, що функція є опуклою поблизу мінімуму. Також, це гарантує, що $F\left(x^*\right) < F\left(x\right)$ для всіх $x$ у деякій, можливо, невеликій, області навколо $x^*$. Для деяких функцій $F(x)$ може існувати декілька точок $x^*$, які задовольняють (2.10). Всі вони є *локальними* мінімумами; і точка, яка дає найменше значення $F$, буде називатися *глобальним* мінімумом.

## 2.1 Математичні моделі безумовної мінімізації для вирішення задач Г. Марковіца

### 2.1.1 Задача мінімізації ризику при визначеному рівні прибутку

Ми вже розглядали цю задачу у розділі 1 для випадку з двома активами, і отримали задачу мінімізації (1.1.5). Використовуючи (2.1.5) і (2.1.6), отримаємо задачу для випадку з $n$ активами (**MinimumRisk1**):

$$\text{Minimize} \quad F = \left(\alpha + Bx\right)^T Q\left(\alpha + Bx\right) + \frac{\rho}{R_p^2}\left(\bar{r}^T \alpha - R_p + \bar{r}^T Bx\right)^2. \tag{2.1.1.1}$$

Таким чином, ми перейшли від задачі умовної мінімізації (1.1.1) до задачі безумовної мінімізації для $n-$активів (2.1.1.1). Така постановка дає нам можливість позбавитися або від однієї, або від двох змінних (як позбавлятися від двох змінних розглянемо далі).

Для того, щоб отримати вирази для градієнта та матриці Гессе функції $F$ з (2.1.1.1), позначимо

$$F = V + \bar{\rho}\left(R - R_p\right)^2,$$

де

$$V = (\alpha + Bx)^T Q(\alpha + Bx), \qquad \bar{\rho} = \frac{\rho}{R_p^2} \qquad \text{і} \qquad R = \bar{r}^T(\alpha + Bx).$$

Використовуючи підрядкові позначення для градієнтів, за допомогою ланцюгового правила диференціювання маємо:

$$F_x = V_x + 2\bar{\rho}\left(R - R_p\right)R_x,$$

де

$$V_x = 2B^T Q(\alpha + Bx) \qquad \text{і} \qquad R_x = B^T\bar{r}.$$

Аналогічно, використовуючи подвійні підрядкові індекси для позначення матриці Гессе:

$$F_{xx} = V_{xx} + 2\bar{\rho}\left(R - R_p\right)R_{xx} + 2\bar{\rho}R_x R_x^T.$$

Одразу видно, що $R_{xx}$ є нульовою матрицею, оскільки $R_x$ не залежить від $x$. Тому маємо:

$$F_{xx} = V_{xx} + 2\bar{\rho}R_x R_x^T,$$

де

$$V_{xx} = 2B^T QB$$

і $R_x R_x^T$ визначає матрицю першого рангу, $(i, j)-$й елемент якої задається

$$\left(R_x R_x^T\right)_{i,j} = \frac{\partial R}{\partial x_i}\frac{\partial R}{\partial x_j}.$$

Перейти від задачі умовної мінімізації (1.1.1) до задачі безумовної мінімізації для $n-$активів можна ще й наступним чином (**MinimumRisk2**):

$$\text{Minimize} \quad F(y)=\frac{\rho}{R_p^2}\left(R-R_p\right)^2+\rho\left(S-1\right)+V, \qquad (2.1.1.2)$$

де $V=y^TQy,\ R=\bar{r}^Ty$ і $S=e^Ty,$ і тому можна очікувати, що мінімум $F(y)$ буде знаходитися у точці, яка наближає розв'язок до (1.1.1). Значення $V$ буде малим і, коли $\rho$ достатньо велике, перші два члени у $F(y)$ будуть близькими до нуля, а отже, умови (1.3) і (1.10) будуть задоволені. Розв'язання (2.1.1.2) означає, що ми маємо справу безпосередньо з вкладеними частками $y,$ і ми уникаємо використання перетворень, пов'язаних з (2.5).

В доданку очікуваного прибутку в (2.1.1.1) (а також в (2.1.1.2)), ми могли б опустити знаменник $R_p^2.$ Тоді цільова функція матиме вигляд

$$\hat{F}(y)=\rho\left(R-R_p\right)^2+\rho(S-1)^2+V.$$

Тепер припустимо, що інвестовані частки $y_i$ такі, що $\left|R-R_p\right|\approx 0.1R_p$ і $|S-1|\approx 0.1.$ Можемо бачити, що відсоткові похибки в обох обмеженнях приблизно рівні (близько 10%). Якщо $R_p$ досить мале - скажімо, 0.1%, то перший член у $F(y)$ буде приблизно в сто разів меншим за другий. Це свідчить про те, що умова $S=1$ може мати непропорційний вплив на положення мінімуму $\hat{F}(y).$ Тому, масштабування доданку $F(y),$ який відповідає за очікувану прибутковість, робить внески дохідності та нормалізації краще збалансованими.

Для подальшого розв’язання задачі (2.1.1.2) нам знадобляться похідні від $F$ по змінним $y_1,\ldots,y_n.$ Покладемо $\bar{\rho}=\dfrac{\rho}{R_p^2}$ і тому

$$\frac{\partial F}{\partial y_i}=2\bar{\rho}\left(R-R_p\right)\frac{\partial R}{\partial y_i}+2\rho(S-1)\frac{\partial S}{\partial y_i}+\frac{\partial V}{\partial y_i}.$$

З означень $R,\ S$ та $V$ отримуємо

$$\frac{\partial R}{\partial y_i} = \bar{r}_i, \qquad \frac{\partial S}{\partial y_i} = 1 \qquad \text{і} \qquad \frac{\partial V}{\partial y_i} = 2\sum_{j-1}^{n} \sigma_{ij} y_j.$$

Таким чином, вектор градієнта $F_y$ функції в (2.1.1.2) має вигляд

$$F_y = 2\bar{\rho}\left(R - R_p\right)\bar{r} + 2\rho(S-1)e + 2Qy,$$

де $e$ це вектор $(1,\ 1,\ 1,\ldots,\ 1)^T$.

## 2.1.2 Задача максимізації прибутку при визначеному рівні ризику

Ця задача вже розглядалась у розділі 1 як задача безумовної мінімізації для двох активів (1.2.5). Узагальнюючи цю задачу для випадку $n-$активів, використовуючи (2.1.5) і (2.1.6), отримуємо **MaximumReturn1**:

$$\text{Minimize} \quad F = -\bar{r}^T\left(\alpha + Bx\right) + \frac{\rho}{V_a^2}\left[\left(\alpha + Bx\right)^T Q\left(\alpha + Bx\right) - V_a\right]^2. \qquad (2.1.2.1)$$

Для отримання виразів градієнта та матриці Гессе $F$, зручно записати

$$F = -R + \bar{\rho}\left(V - V_a\right)^2,$$

де $R = \bar{r}^T(\alpha + Bx)$, $\bar{\rho} = \dfrac{\rho}{V_a^2}$ і $V = (\alpha + Bx)^T Q(\alpha + Bx)$.

Градієнт $R$ це:

$$R_x = B^T\bar{r}$$

і матриця Гессе $R_{xx}$ є нулем, тому що $R$ лінійна. Градієнт і матриця Гессе для $V$ позначаються як:

$$V_x = 2B^T Q\alpha + 2B^T QBx$$

$$V_{xx} = 2B^T QB.$$

Використовуючи правило ланцюга, градієнт функції з (2.1.2.1) можна записати як:

$$F_x = -B^T\bar{r} + 2\bar{\rho}\left(V - V_a\right)V_x.$$

Вираз для матриці Гессе буде мати вигляд

$$F_{xx} = 2\bar{\rho}\left(V_x V_x^T + \left(V - V_a\right)V_{xx}\right),$$

де добуток $V_x V_x^T$ є матрицею першого рангу, $i,\ j-$й елемент якої задається формулою

$$\left(V_x V_x^T\right)_{i,j} = \frac{\partial V}{\partial x_i}\frac{\partial V}{\partial x_j}.$$

Перейти від задачі умовної мінімізації (1.2.2) до задачі безумовної мінімізації для $n-$активів можна іншим шляхом, таким як (**MaximumReturn2**):

$$\text{Minimize} \quad F(y) = \rho\left(S-1\right)^2 + \frac{\rho}{V_a^2}\left(V - V_a\right) - R \qquad (2.1.2.2)$$

де $V = y^T Q y,\quad R = \bar{r}^T y$ і $S = e^T y.$ Причини масштабування доданку ризику в (2.1.2.2) подібні до тих, що стосуються масштабування доданку очікуваного прибутку в (2.1.1.2).

## 2.2 Вирішення задачі безумовної мінімізації функції за допомогою виключення двох змінних

Вирішимо задачу мінімізації у термінах $y_1,\ldots,y_{n-2}$. Для цього позначимо

$$\gamma_1 = R_p - \sum_{i=1}^{n-2}\bar{r}_i y_i \quad \text{і} \quad \gamma_2 = 1 - \sum_{i=1}^{n-2} y_i \qquad (2.2.1)$$

тоді (1.3) і (1.10) означають

$$\bar{r}_{n-1} y_{n-1} + \bar{r}_n y_n = \gamma_1 \qquad (2.2.2)$$

$$y_{n-1} + y_n = \gamma_2. \qquad (2.2.3)$$

Помножуючи (2.3.3) на $\bar{r}_n$ і віднімаючи його від (2.3.2) отримуємо

$$\left(\bar{r}_{n-1} - \bar{r}_n\right) y_{n-1} = \gamma_1 - \bar{r}_n y_n \qquad (2.2.4)$$

і отже

$$y_{n-1} = \frac{\gamma_1 - \bar{r}_n \gamma_2}{\bar{r}_{n-1} - \bar{r}_n} = \tilde{\alpha}_{n-1} + \sum_{i=1}^{n-2}\tilde{\beta}_{n-1,i} y_i \qquad (2.2.5)$$

$$\text{де} \quad \tilde{\alpha}_{n-1} = \frac{R_p - \bar{r}_n}{\bar{r}_{n-1} - \bar{r}_n} \quad \text{і} \quad \tilde{\beta}_{n-1,i} = \frac{\bar{r}_n - \bar{r}_i}{\bar{r}_{n-1} - \bar{r}_n}. \qquad (2.2.6)$$

Крім того

$$y_n = \gamma_2 - y_{n-1} = \tilde{\alpha}_n + \sum_{i=1}^{n-2} \tilde{\beta}_{n,i} y_i \tag{2.2.7}$$

де $\tilde{\alpha}_n = 1 - \tilde{\alpha}_{n-1}$ і $\tilde{\beta}_{n,i} = -\left(1 + \tilde{\beta}_{n-1,i}\right)$. (2.2.8)

Якщо ми використовуємо ці вирази для усунення $y_{n-1}$ і $y_n$, функцію ризику можна вважати залежною лише від $y_1, \ldots, y_{n-2}$. Визначимо $n$-вектор $\tilde{\alpha}$ і $n \times (n-2)$ матрицю $\tilde{B}$ як

$$\tilde{\alpha} = \begin{pmatrix} 0 \\ 0 \\ .. \\ 0 \\ \tilde{\alpha}_{n-1} \\ \tilde{\alpha}_n \end{pmatrix} \qquad \tilde{B} = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ .. & .. & .. & .. \\ 0 & 0 & \ldots & 1 \\ \tilde{\beta}_{n-1,1} & \tilde{\beta}_{n-1,2} & \ldots & \tilde{\beta}_{n-1,n-2} \\ \tilde{\beta}_{n,1} & \tilde{\beta}_{n!2} & \ldots & \tilde{\beta}_{n,t,\cdot 2} \end{pmatrix}. \tag{2.2.9}$$

## 2.3 Алгоритми методів безумовної мінімізації функцій, застосованих у дослідженні

### 2.3.1 Метод спряжених градієнтів

Метод спряжених градієнтів є одним із найбільш уживаних градієнтних методів оптимізації функцій $n$ змінних. Формальний опис методу наведено нижче.

***Метод спряжених градієнтів для мінімізації F(x)***

Choose $x_0$ as an initial estimate of the solution
Calculate $g_0 = \nabla F\left(x_0\right)$. Set $p_0 = -g_0$
Repeat for $k = 0, 1, 2, \ldots$
    find $s$ by a perfect line search to minimize $F\left(x_k + sp_k\right)$
    set $x_{k+1} = x_k + sp_k$, $g_k = \nabla F\left(x_k\right)$
    if $k$ is not a multiple of $n$ then
        find $\beta$ and set $p_{k+1} = -g_{k+1} + \beta p_k$
    else
        set $p_{k+1} = -g_{k+1}$
until $\|g_{k+1}\|$ is sufficiently small.

Значення $s$ знаходиться з рівняння $p_k^T g_{k+1} = 0$. Значення $\beta$ знаходиться по різному, в залежності від того, з якою функцією ми маємо справу. Якщо функція квадратична, тоді використовується формула Флетчера-Рівса:

$$\beta = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}. \qquad (2.3.1.1)$$

Для інших випадків використовуємо формулу Полака-Ріб'єра:

$$\beta = \frac{g_{k+1}^T \left( g_{k+1} - g_k \right)}{g_k^T g_k}. \qquad (2.3.1.2)$$

Якщо функція квадратична, то формули (2.3.1.1) і (2.3.1.2) дадуть однакові значення $\beta$. Якщо функція не є квадратичною, тоді ці формули призведуть до різних напрямків пошуку.

## 2.3.2 Метод Ньютона

Цей метод базується на властивостях квадратичної функції, яка має сталу матрицю Гессе. Рішення буде мінімумом, якщо отримана матриця Гессе буде додатньо визначеною. Формальний опис методу Ньютона наведено нижче.

***Метод Ньютона***

Choose $x_0$ as an initial estimate of the minimum of $F(x)$
Repeat for $k = 0,1,2,\ldots$
    Set $g_k = \nabla F\left(x_k\right),\ G_k = \nabla^2 F\left(x_k\right)$.
    if $G_k$ is positive definite then obtain $p_k$ by solving $G_k p_k = -g_k$
    else set $p_k = -g_k$
    Set $x_{k+1} = x_k + p_k$
until $\left\|\nabla F\left(x_{k+1}\right)\right\|$ is sufficiently small.

Вектор $p_k = -G_k^{-1} g_k$ називається поправкою Ньютона. За сприятливих умов - тобто, коли матриця Гессе функції $F$ є додатньо визначеною - алгоритм Ньютона може бути дуже ефективним.

## 2.4 Порівняльний аналіз математичних моделей і методів безумовної мінімізації функцій на прикладах портфелів цінних паперів

### Приклад 1

Розглянемо задачу, яка базується на історії активів у таблиці 2.4.1 і передбачає пошук портфеля з максимальним прибутком для отримання ризику у розмірі $V_a = 0,003$. Використаємо метод спряжених градієнтів. Точність розрахунків $\varepsilon = 10^{-5}\sqrt{3}$, константа балансу між прибутковістю та ризиком $\rho = 1$.

| | Дохід % за період | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Актив 1 | 1,2 | 1,3 | 1,4 | 1,5 | 1,1 | 1,2 | 1,1 | 1,0 | 1,0 | 1,1 |
| Актив 2 | 1,3 | 1,0 | 0,8 | 0,9 | 1,4 | 1,3 | 1,2 | 1,1 | 1,2 | 1,1 |
| Актив 3 | 0,9 | 1,1 | 1,0 | 1,1 | 1,1 | 1,3 | 1,2 | 1,1 | 1,0 | 1,1 |

*Таблиця 2.4.1*

Розрахуємо середню прибутковість $\bar{r}$ за формулою (1.2):

$$\bar{r} \approx (1.2833,\ 1.1167,\ 1.0833)^T.$$

За допомогою формул (1.5) і (1.6), сформуємо дисперсійно-коваріаційну матрицю (1.9):

$$Q \approx \begin{pmatrix} 0.0181 & -0.02806 & -0.00194 \\ -0.02806 & 0.0514 & 0.00528 \\ -0.00194 & 0.00528 & 0.0147 \end{pmatrix}.$$

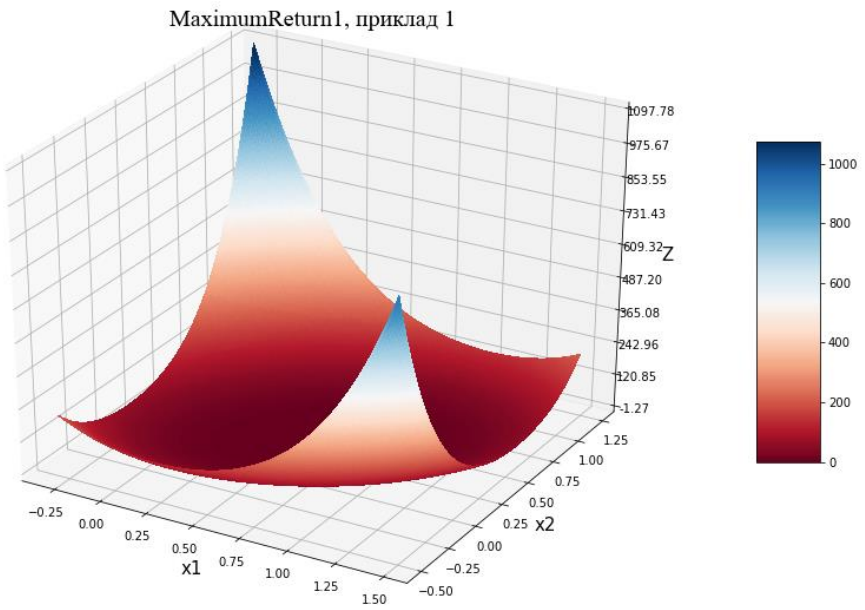
Вирішимо цю задачу, використовуючи модель (2.1.2.1) за допомогою виключення однієї змінної. Матриці $\alpha$ і $B$ мають вигляд (2.5) і розмірності 3 та $3 \times 2$. Елементи матриць розраховуються за формулами (2.3.6). Отримуємо

$$\alpha = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \end{pmatrix}.$$

Формуємо функцію для подальшої мінімізації **MaximumReturn1** (2.1.2.1), де $x = \left(x_1,\ x_2\right)^T$ :

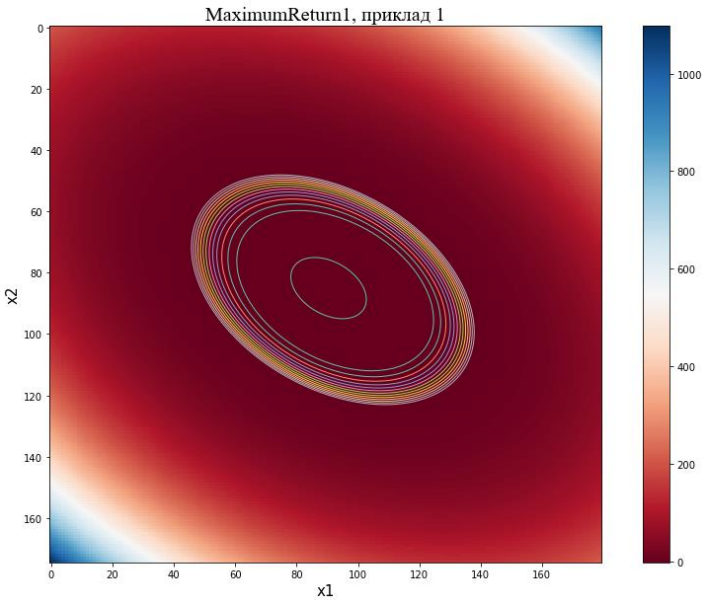
$$F(x) \approx 149.5x_1^4 - 272.25 \cdot x_1^3 \cdot x_2 - 271.269x_1^3 + 576.66 \cdot x_1^2 \cdot x_2^2 + 93.44 \cdot x_1^2 \cdot x_2 +$$
$$218.43x_1^2 - 412.23 \cdot x_1 \cdot x_2^3 - 270.9 \cdot x_1 \cdot x_2^2 + 52.5 \cdot x_1 \cdot x_2 - 86.55 \cdot x_1 + 342.7x_2^4 -$$
$$-232.5x_2^3 + 183.8x_2^2 - 48.98x_2 + 15.1$$

Графік цієї функції має наступний вигляд




*Мал. 2.4.1*

Вигляд мал. 2.4.1 зверху




*Мал. 2.4.2*

Обираємо початкове наближення $x_0 = \left(\frac{1}{3},\ \frac{1}{3},\ \frac{1}{3}\right)^T \approx (0.33,\ 0.33,\ 0.33)^T$. Вектор градієнту має вигляд

$$g \approx \begin{pmatrix} 597.96x_1^3 - 816.74 \cdot x_1^2 \cdot x_2 - 813.8x_1^2 + 1153.33 \cdot x_1 \cdot x_2^2 + 186.9 \cdot x_1 \cdot x_2 + \\ +436.9 \cdot x_1 - 412.23x_2^3 - 270.9x_2^2 + 52.5 \cdot x_2 - 86.728 \\ -272.247x_1^3 + 1153.33 \cdot x_1^2 \cdot x_2 + 93.4x_1^2 - 1236.7 \cdot x_1 \cdot x_2^2 - 541.83 \cdot x_1 \cdot x_2 + \\ +52.5 \cdot x_1 + 1371x_2^3 - 697.6 \cdot x_2^2 + 367.7 \cdot x_2 - 49 \end{pmatrix}.$$

Значення градієнту у початковій точці

$$g_0 = \nabla F(x_0) \approx \begin{pmatrix} -4.024 \\ 1.318 \end{pmatrix}$$

і, отже, напрямок пошуку від $x_0$ буде

$$p_0 = -g_0 \approx \begin{pmatrix} 4.024 \\ -1.318 \end{pmatrix}.$$

***1 ітерація***

$$x_1 = x_0 + sp_0 \approx \begin{pmatrix} 0.33 \\ 0.33 \end{pmatrix} + s \cdot \begin{pmatrix} 4.024 \\ -1.318 \end{pmatrix} = \begin{pmatrix} 4.024s + 0.33 \\ -1.318s + 0.33 \end{pmatrix}$$

$$g_1 = \nabla F(x_1) \approx \begin{pmatrix} 65418s^3 - 10605.6s^2 + 462.5s - 4.02 \\ -54151s^3 + 6953s^2 - 194.3s + 1.32 \end{pmatrix}.$$

Значення $s$ знаходимо з рівняння $p_0^T g_1 = 0$:

$$334644.7s^3 - 51846.35s^2 + 2117.53s - 17.9 = 0$$

$$\Rightarrow s \approx 0.093$$

Розв'язавши рівняння, отримуємо довжину кроку і нову точку

$$x_1 \approx \begin{pmatrix} 0.708 \\ 0.21 \end{pmatrix} \quad \text{де} \quad g_1 \approx \begin{pmatrix} -0.077 \\ -0.235 \end{pmatrix}.$$

Звідси можемо знайти $y = \begin{pmatrix} 0.708 \\ 0.21 \\ 1 - 0.708 - 0.21 \end{pmatrix} = \begin{pmatrix} 0.708 \\ 0.21 \\ 0.081 \end{pmatrix}$.

При таких значеннях $y$, $V \approx 0.0030474599$ (1.8), $R_p \approx 1.23177219$.

Використаємо $g_1$, щоб знайти напрямок пошуку для наступної ітерації. Спочатку розрахуємо значення $\beta$ за допомогою формули Полака-Ріб'єра (2.3.1.2), так як маємо справу з функцією четвертого степеня:

$$\beta = \frac{g_1^T\left(g_1 - g_0\right)}{g_0^T g_0} \approx \frac{0.06119}{17.93} \approx 0.0034$$

звідси

$$p_1 = -g_1 + \beta p_0 \approx \begin{pmatrix} 0.077 \\ 0.235 \end{pmatrix} + 0.0034 \cdot \begin{pmatrix} 4.024 \\ -1.318 \end{pmatrix} = \begin{pmatrix} 0.0907 \\ 0.2306 \end{pmatrix}$$

***2 ітерація***

Нова оцінка розв'язку

$$x_2 = x_1 + sp_1 \approx \begin{pmatrix} 0.0907s + 0.708 \\ 0.23s + 0.21 \end{pmatrix}$$

звідси

$$g_2 = \nabla F\left(x_2\right) \approx \begin{pmatrix} -0.59s^3 + 7.4s^2 - 8.697s - 0.077 \\ 12.829s^3 - 27.68s^2 + 14.48s - 0.235 \end{pmatrix}.$$

Довжина кроку $s$ задовольняє рівнянню $p_1^T g_2 = 0$:

$$2.9s^3 - 5.7s^2 + 2.55s - 0.06 = 0$$

$$\Rightarrow s \approx 1.3073$$

Розв'язавши рівняння, отримуємо довжину кроку і нову точку

$$x_2 \approx \begin{pmatrix} 0.827 \\ 0.5119 \end{pmatrix} \quad \text{де} \quad g_2 \approx \begin{pmatrix} -0.12 \\ 0.047 \end{pmatrix}.$$

Звідси можемо знайти $y = \begin{pmatrix} 0.827 \\ 0.5119 \\ -0.33899 \end{pmatrix}$.

При таких значеннях $y$, $V \approx 0.00303490459$ (1.8), $R_p \approx 1.26567681$.

Отже, при заданому рівні ризику $V_a = 0.003$, отримуємо $y_1 = 0.827$, $y_2 = 0.5119$, $y_3 = -0.33899$. При такому розподілі грошей, прибуток становитиме $R_p \approx 1.265$.

Розв'язавши цю задачу методом Ньютона (додаток В), порівняємо результати:

| $V=0.003$, $e=1.732\cdot10^{-5}$, $i=50$ | | **MaximumRetrurn1** | | |
|---|---|---|---|---|
| | | $\rho=1$ | $\rho=10$ | $\rho=100$ |
| Метод спряжених градієнтів | Risk = | 0.0030349 | 0.003003 | 0.0029997 |
| | Return = | 1.26567% | 1.26283% | 1.23106% |
| | Time = | 0.540228 s | 0.6359 s | 0.5639 s |
| | Iter = | 2 | 2 | 2 |
| Метод Ньютона | Risk = | 0.0030608 | 0.003006 | 0.00303 |
| | Return = | 1.27% | 1.269679% | 1.253469% |
| | Time = | 0.374997 s | 0.484811 s | 0.797438 s |
| | Iter = | 16 | 24 | 38 |

*Таблиця. 2.4.2*

Висновки:

1. Найкращий результат був отриманий методом Ньютона при значенні вагового множника $\rho=1$.
2. Найнижчий рівень доходу був отриманий методом спряжених градієнтів при $\rho=100$. Але при цьому обмеження задачі виконується точніше.
3. Метод Ньютона працює швидше ніж метод спряжених градієнтів, не дивлячись на більшу кількість ітерацій, та дає найліпші результати при будь якому $\rho$.

**Приклад 2**

Розглянемо задачу, яка базується на історії активів у таблиці 2.4.3 і передбачає пошук портфеля з мінімальним ризиком для отримання прибутку у розмірі $R_p=1.15\%$. Використаємо метод Ньютона. Точність розрахунків $\varepsilon=10^{-5}\sqrt{5}$, константа балансу між прибутковістю та ризиком $\rho=1$.

| | Дохід % за період | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Актив 1 | 1,2 | 1,3 | 1,4 | 1,5 | 1,1 | 1,2 | 1,1 | 1,0 | 1,0 | 1,1 |
| Актив 2 | 1,3 | 1,0 | 0,8 | 0,9 | 1,4 | 1,3 | 1,2 | 1,1 | 1,2 | 1,1 |
| Актив 3 | 0,9 | 1,1 | 1,0 | 1,1 | 1,1 | 1,3 | 1,2 | 1,1 | 1,0 | 1,1 |
| Актив 4 | 1,1 | 1,1 | 1,2 | 1,3 | 1,2 | 1,2 | 1,1 | 1,0 | 1,1 | 1,2 |
| Актив 5 | 0,8 | 0,75 | 0,65 | 0,75 | 0,8 | 0,9 | 1,0 | 1,1 | 1,1 | 1,2 |

*Таблиця 2.4.3*

Розрахуємо середню прибутковість $\overline{r}$ за формулою (1.2):

$$\overline{r} \approx (1.19,\ 1.13,\ 1.09,\ 1.15,\ 0.905)^T.$$

За допомогою формул (1.5) і (1.6), сформуємо дисперсійно-коваріаційну матрицю (1.9):

$$Q \approx \begin{pmatrix} 0.0249 & -0.0187 & -0.0011 & 0.0085 & -0.0219 \\ -0.0187 & 0.0321 & 0.0033 & -0.0035 & 0.0099 \\ -0.0011 & 0.0033 & 0.0109 & 0.0015 & 0.0035 \\ 0.0085 & -0.0035 & 0.0015 & 0.0065 & -0.0058 \\ -0.0219 & 0.0099 & 0.0035 & -0.0058 & 0.0307 \end{pmatrix}.$$

Вирішимо цю задачу, використовуючи модель (2.1.1.1) за допомогою виключення двох змінних. Матриці $\tilde{\alpha}$ і $\tilde{B}$ мають вигляд (2.2.9) і розмірності 5 та $5\times 3$. Елементи матриць розраховуються за формулами (2.2.6) і (2.2.8). Отримуємо

$$\tilde{\alpha} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad \tilde{B} \approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1.163 & -0.918 & -0.755 \\ 0.163 & -0.082 & -0.245 \end{pmatrix}.$$

Формуємо функцію для подальшої мінімізації **MinimumRisk1** (2.1.1.1), де $x = \left(x_1,\ x_2,\ x_3\right)^T$ :

$$F(x) \approx 0.00977x_1^2 - 0.0243 \cdot x_1 \cdot x_2 + 0.00045 \cdot x_1 \cdot x_3 - 1.6326 \cdot 10^{-5} \cdot x_1 + 0.0417x_2^2 +$$
$$+0.01 \cdot x_2 \cdot x_3 - 0.018x_2 + 0.01x_3^2 - 0.004x_3 + 0.0065$$

Обираємо початкове наближення $x_0 = \left(\frac{1}{5},\ \frac{1}{5},\ \frac{1}{5}\right)^T \approx \left(0.2,\ 0.2,\ 0.2\right)^T$. Вектор градієнту має вигляд:

$$\nabla F = g \approx \begin{pmatrix} 0.019x_1 - 0.024x_2 + 0.00045x_3 - 1.633 \cdot 10^{-5} \\ -0.024x_1 + 0.08x_2 + 0.01x_3 - 0.018 \\ 0.00045x_1 + 0.01x_2 + 0.02x_3 - 0.004 \end{pmatrix},$$

а матриця Гессе:

$$\nabla^2 F = G_0 \approx \begin{pmatrix} 0.019 & -0.024 & 0.00045 \\ -0.024 & 0.08 & 0.01 \\ 0.00045 & 0.01 & 0.02 \end{pmatrix}.$$

Значення градієнту у початковій точці є

$$\nabla F\left(x_0\right) = g_0 \approx \begin{pmatrix} -0.00088 \\ -0.004 \\ 0.0024 \end{pmatrix}.$$

***1 ітерація***

Так як матриця Гессе $G_0$ є додатньо визначеною (2.9), тому знайдемо $p_0$ з системи рівняннь $G_0 p_0 = -g_0$ :

$$\begin{pmatrix} 0.019 & -0.024 & 0.00045 \\ -0.024 & 0.08 & 0.01 \\ 0.00045 & 0.01 & 0.02 \end{pmatrix} \cdot \begin{pmatrix} p_{01} \\ p_{01} \\ p_{03} \end{pmatrix} \approx \begin{pmatrix} 0.00088 \\ 0.004 \\ -0.0024 \end{pmatrix}$$

$$\Rightarrow\ p_0 \approx \begin{pmatrix} 0.22 & 0.1369 & -0.19 \end{pmatrix}^T.$$

Знаходимо $x_1 = x_0 + p_0$ :

$$x_1 \approx \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \end{pmatrix} + \begin{pmatrix} 0.22 \\ 0.1369 \\ -0.19 \end{pmatrix} \approx \begin{pmatrix} 0.42 \\ 0.3369 \\ 0.01 \end{pmatrix}.$$

Значення градієнта у цій точці:

$$\nabla F\left(x_1\right) = g_1 \approx \begin{pmatrix} -3.303 \cdot 10^{-20} \\ 2.385 \cdot 10^{-18} \\ 3.253 \cdot 10^{-19} \end{pmatrix}.$$

$\left|g_1\right| < 2.236 \cdot 10^{-5} = \varepsilon \Rightarrow$ ми досягли заданої точності, тому точка $x_1$ найкраще мінімізує функцію $F(x)$. Отримуємо

$$y_1 \approx 0.42,\ y_2 \approx 0.34,\ y_3 \approx 0.01$$

Частки $y_4$ і $y_5$ знайдемо за формулами (2.2.5) і (2.2.7):

$$y_4 = \tilde{\alpha}_4 + \sum_{i=1}^{3} \tilde{\beta}_{4,i} y_i \qquad y_5 = \tilde{\alpha}_5 + \sum_{i=1}^{3} \tilde{\beta}_{5,i} y_i$$

$$y_4 \approx 1 + \left(-1.163 \cdot 0.42 - 0.918 \cdot 0.34 - 0.755 \cdot 0.01\right) \approx 0.2$$

$$y_5 \approx 0 + \left(0.163 \cdot 0.42 - 0.082 \cdot 0.34 - 0.245 \cdot 0.01\right) \approx 0.04$$

Тому, при вкладених частках

$$y_1 \approx 0.42,\ y_2 \approx 0.34,\ y_3 \approx 0.01,\ y_4 \approx 0.02,\ y_5 \approx 0.04$$

отримуємо мінімальний ризик $V \approx 0.00344635014$ при $R_p \approx 1.149$ (1.4).

Вирішимо цю задачу методом спряжених градієнтів (додаток Г) і порівняємо результати:

| $R_p = 1.15\%$, $e = 2.236 \cdot 10^{-5}$, $i = 3$ | | **MinimumRisk1** | | |
|---|---|---|---|---|
| | | $\rho = 1$ | $\rho = 10$ | $\rho = 100$ |
| Метод спряжених градієнтів | Risk = | 0.00344635014 | 0.003446350859 | 0.00344635802 |
| | Return = | 1.149999% | 1.149999% | 1.150…6% |
| | Time = | 0.2221324 s | 0.2 s | 0.2 s |
| | Iter = | 3 | 3 | 3 |
| Метод Ньютона | Risk = | 0.00344635014 | 0.003446350859 | 0.00344635802 |
| | Return = | 1.149999% | 1.149999% | 1.150…008% |
| | Time = | 0.0219447 s | 0.019654 s | 0.0169866 s |
| | Iter = | 1 | 1 | 1 |

*Таблиця 2.4.4*

Висновки:

1. Як можемо бачити, обидва методи показують непогані і майже однакові результати, за якими важко щось оцінювати. Показниками «якості» залишаються час роботи методів та кількість ітерацій. Тут кращим виявився метод Ньютона.
2. Найменший показник ризику в обох методах отримуємо при $\rho = 1$, а найбільший при $\rho = 100$.
3. Найбільшу точність дотримання обмеження маємо при $\rho = 100$.

## Приклад 3

Розглянемо задачу, яка базується на історії активів у таблиці 2.4.3 і передбачає пошук портфеля з максимальним прибутком для ризику у розмірі $V_a = 0,00375$. Точність розрахунків $\varepsilon = 10^{-5}\sqrt{5}$, константа балансу між прибутковістю та ризиком $\rho = 1$.

Для розв'язку використаємо модель **MaximumReturn2** (2.1.2.2). В даному випадку ми не можемо виключати змінні, тому функція буде мати 5 невідомих:

$$F(x) = -1.19x_1 - 1.13x_2 - 1.09x_3 - 1.15x_4 - 0.905x_5 + (x_1 + x_2 + x_3 + x_4 + x_5 - 1)^2 +$$

$$+17777.78\big(x_1(0.0249x_1 - 0.0187x_2 - 0.0011x_3 + 0.0085x_4 - 0.02195x_5) +$$

$$+x_2(-0.0187x_1 + 0.0321x_2 + 0.0033x_3 - 0.0035x_4 + 0.0099x_5) +$$

$$+x_3(-0.0011x_1+0.0033x_2+0.0109x_3+0.0015x_4+0.0035x_5)\ +$$
$$+\ x_4(0.0085x_1-0.0035x_2+0.0015x_3+0.0065x_4-0.0058x_5)+$$
$$+x_5(-0.02195x_1+0.0099x_3+0.0035x_3-0.0058x_4+0.0307x_5)-0.00375)^2$$

Приблизним розв'язком є:

$$y_1=0.41,\ \ y_2=0.34,\ \ y_3=0.01,\ \ y_4=0.21,\ \ y_5=0.02,$$

Знайдемо мінімум цієї функції (тобто максимум прибутку), використовуючи метод спряжених градієнтів (додаток Д) і метод Ньютона (додаток Е):

| $V=0.00375$, $e=2.236\cdot10^{-5}$, $i=50$ | | **MaximumRetrurn2** | | |
|---|---|---|---|---|
| | | $\rho=1$ | $\rho=10$ | $\rho=100$ |
| Метод спряжених градієнтів | Risk = | 0.004058 | 0.00377 | 0.00375133 |
| | Return = | 1.408735% | 1.117995% | 1.085267% |
| | Time = | 5.263 s | 5.2 s | 5.0997 s |
| | Iter = | 5 | 5 | 5 |
| Метод Ньютона | Risk = | 0.003867 | 0.003758957 | 0.00375088 |
| | Return = | 1.429672% | 1.182013% | 1.156722% |
| | Time = | 3.4 s | 5.74458 s | 7.55 s |
| | Iter = | 11 | 20 | 27 |

*Таблиця 2.4.5*

Висновки:

1. Найкращий результат за точністю виконання обмеження був отриманий методом Ньютона при значенні вагового множника $\rho=100$.
2. Найгірший результат за точністю виконання обмеження був отриманий методом спряжених градієнтів при $\rho=1$. Рівень ризику відхиляється від заданого найбільше.
3. Метод Ньютона працює швидше ніж метод спряжених градієнтів, не дивлячись на більшу кількість ітерацій, та дає найліпші результати при $\rho=10$ і $\rho=100$.

**Приклад 4**

Розглянемо задачу, яка базується на історії активів у таблиці 2.4.1 і передбачає пошук портфеля з мінімальним ризиком для заданого прибутку у розмірі

$R_p = 1.25\%$. Точність розрахунків $\varepsilon = 10^{-5}\sqrt{3}$, константа балансу між прибутковістю та ризиком $\rho = 1$.

Для розв'язку використаємо модель **Minimumrisk2** (2.1.1.2). В даному випадку (тобто для даної моделі), ми не можемо виключати змінні, тому функція буде мати 3 невідомі:

$$F(x) = 2.072x_1^2 + 3.778 \cdot x_1 \cdot x_2 + 3.7756 \cdot x_1 \cdot x_3 - 4.05328x_1 + 1.8495x_2^2 +$$
$$+3.559 \cdot x_2 \cdot x_3 - 3.7867x_2 + 1.7658x_3^2 - 3.7333x_3 + 2$$

Приблизним розв'язком є

$$y_1 = 0.76, \;\; y_2 = 0.41, \;\; y_3 = -0.18.$$

Порівняємо результати отримані за допомогою методу спряжених градієнтів і методу Ньютона:

| $R_p = 1.25\%$, $e = 1.732 \cdot 10^{-5}$, $i = 3$ | | **MinimumRisk2** | | |
|---|---|---|---|---|
| | | $\rho = 1$ | $\rho = 10$ | $\rho = 100$ |
| Метод спряжених градієнтів | Risk = | 0.00137 | 0.00172 | 0.001828 |
| | Return = | 1.23836% | 1.24709% | 1.249655% |
| | Time = | 0.2795 s | 0.280589 s | 0.330312 s |
| | Iter = | 3 | 3 | 3 |
| Метод Ньютона | Risk = | 0.00137 | 0.00172 | 0.001828 |
| | Return = | 1.23836% | 1.24709% | 1.249655% |
| | Time = | 0.0356 s | 0.02715 s | 0.0460288 s |
| | Iter = | 1 | 1 | 1 |

*Таблиця 2.4.6*

Висновки:

1. Найкращий результат був отриманий методами Ньютона і спряжених градієнтів при значенні вагового множника $\rho = 100$.
2. Найгірший результат був отриманий методами Ньютона і спряжених градієнтів при $\rho = 1$.
3. Метод Ньютона працює швидше ніж метод спряжених градієнтів.

**Порівняння моделей MinimumRisk1 і MinimumRisk2**

Модель **MinimumRisk1** краще підходить для аналізу історії активів, бо обидва методи мінімізації отриманої функції (метод спряжених градієнтів і метод Ньютона) дають приблизно однакові результати. Тому досить очевидно, як потрібно робити розподіл грошей між активами. Ця модель досить точно відповідає заданим умовам, а також час на виконання аналізу є меншим, ніж у моделі **MinimumRisk2**. Модель **MinimumRisk1** є безумовною задачею мінімізації з можливостю виключення однієї або двох змінних, а у **MinimumRisk2** наявні штрафні функції, тобто додаються незв'язки двох обмежень-рівностей з множниками штрафу $\rho$ (чим більше $\rho$, тим точніше виконуються обмеження). Як бачимо, ефективнішим виявилася стратегія виключення змінних.

**Порівняння моделей MaximumReturn1 і MaximumReturn2**

Обидві моделі досить точно відповідають заданим умовам, але більш точні та однозначні результати дає модель **MaximumReturn1**. Також ця модель використовує трішки менше часу на аналіз історії активів, ніж **MaximumReturn2**. Як і в аналогічному порівнянні моделей вище, кращою виявилася та задача безумовної мінімізації, яка дає можливість виключати змінні.

# Висновки

Основною метою цієї роботи був пошук найефективнішого способу розподілу активів з метою досягнення максимального прибутку або мінімального ризику. Це робилося за допомогою зведення історії прибутковості активів до задач безумовної мінімізації.

Розглянуто дві задачі Г. Марковіца – мінімізація ризику при заданому рівні прибутку і максимізація прибутку при заданому рівні ризику.

У першому розділі створювалися моделі для аналізу портфелю лише двох активів, розглядалися основні принципи побудови математичних моделей та методи їх розв’язання. Одними з можливих методів розв’язання є методи бісекції, січних та Ньютона. Метод Ньютона виявився найкращим, бо він є методом 2-го порядку і тому має тенденцію збігатися швидше.

У другому розділі розглядалися такі портфелі, які мають в собі $n-$активів. Кожну із задач Г. Марковіца ми призводили до задачі безумовної мінімізації двома шляхами – або за допомогою виключення змінних, або за допомогою введення штрафних функцій. Ефективнішим виявилися моделі **MinimumRisk1** і **MaximumReturn1**, які мають можливість виключення змінних. Кожна з отриманих моделей розв’язувалася двома методами – спряжених градієнтів і Ньютона. Обидва методи є стійкими, можуть працювати з функціями з будь-якою кількістю змінних, але все ж таки найефективнішим виявився метод Ньютона.

Як можемо бачити, для розв’язку поставлених задач краще використовувати моделі **MinimumRisk1** і **MaximumReturn1**, і знаходити оптимальний розподіл грошей між активами за допомогою метода Ньютона.

# Додатки

## Додаток А

Python 3.8

```
import numpy as np
import matplotlib.pyplot as plt
from sympy.abc import x
import sympy as sym

m = 2
n = 6
matrix = [[1.2, 1.3, 1.4, 1.5, 1.1, 1.2],
          [1.3, 1, 0.8, 0.9, 1.4, 1.3]]

ri = 0
r = []

# середня прибутковість для кожного активу
for i in range (m):
    for j in range (n):
        ri += matrix[i][j]
    r.append(round(ri/n, 4))
    ri=0

# формування дисперійно-коваріаційної матриці Q
sigma1 = []
sigma_i_squared = 0
for i in range (m):
    for j in range (n):
        sigma_i_squared += (matrix[i][j]-r[i])**2
    sigma1.append(round(sigma_i_squared/n, 4))
    sigma_i_squared = 0

sigma2 = []
sigma_ik = 0

for k in range (m):
    for j in range (n):
        if k!=i:
            sigma_ik += (matrix[i][j]-r[i])*(matrix[k][j]-r[k])
    sigma2.append(round(sigma_ik/n, 4))

alpha = [[0],
         [1]]
trans_alpha = np.transpose(alpha)
beta = [[1],
        [-1]]
trans_beta = np.transpose(beta)
Q = [[sigma1[0], sigma2[0]],
     [sigma2[1], sigma1[1]]]

rT_alpha = (np.dot(r, alpha))
rT_beta = np.dot(r, beta)
alphaT_Q_alpha = np.dot(np.dot(trans_alpha, Q), alpha)
betaT_Q_alpha = np.dot(np.dot(trans_beta, Q), alpha)*2
betaT_Q_beta = np.dot(np.dot(trans_beta, Q), beta)
```

```
Va = 0.00168 # заданий рівень ризику
ro = 1 # константа, яка контролює баланс між прибутковістю та ризиком.

# формуємо функцію мінімізації
def func(x):
    return (-rT_alpha - rT_beta*x + (ro/Va**2)*(alphaT_Q_alpha + betaT_Q_alpha*x
+ betaT_Q_beta*x*x - Va)**2)

new = [y[0] for y in func(x)]
simplified_func = new[0].cancel()

#знаходимо похідну від функції F(x)
fu=sym.diff(simplified_func)
deriv = sym.simplify(fu)


# задаємо проміжок
a=0
b=1

print("deriv", deriv)

# метод бісекції
print("Метод бісекції:")
x_a = a
x_b = b
x_m = 0.5 * (x_a + x_b)
Fa = simplified_func.subs(x, x_a)
Fb = simplified_func.subs(x, x_b)
Fm = simplified_func.subs(x, x_m)
while abs(x_b - x_a) >= 0.0001:
    x_l = 0.5 * (x_a + x_m)
    x_r = 0.5 * (x_m + x_b)
    Fl = simplified_func.subs(x, x_l)
    Fr = simplified_func.subs(x, x_r)
    Fmin = min(Fa, Fb, Fm, Fl, Fr)
    if Fmin == Fa or Fmin == Fl:
        x_b = x_m
        x_m = x_l
        Fb = Fm
        Fm = Fl
    elif Fmin == Fm:
        x_a = x_l
        x_b = x_r
        Fa = Fl
        Fb = Fr
    elif Fmin == Fr or Fmin == Fb:
        x_a = x_m
        x_m = x_r
        Fa = Fm
        Fm = Fr

f_a = simplified_func.subs(x, x_a)
f_b = simplified_func.subs(x, x_b)
f_m = simplified_func.subs(x, x_m)
f_l = simplified_func.subs(x, x_l)
f_r = simplified_func.subs(x, x_r)
minim = min(f_a, f_b, f_m, f_l, f_r)
```

```
print("Дохід R = " + str(abs(minim)) + "%")
print("x =", x_m)
print("Кількість ітерацій:", np.ceil((np.log10(b-a)+4)/np.log10(2)))

# метод січних
print()
print("Метод січних:")
x_0 = 0
x_1 = 1
itns = 10
print("Кількість ітерацій:", itns)
x_i = [x_0, x_1]
for k in range (itns):
    x_i.append(x_i[k]-(deriv.subs(x, x_i[k]))/(deriv.subs(x, x_i[k+1]) -
deriv.subs(x, x_i[k]))*(x_i[k+1]-x_i[k]))
    if abs(deriv.subs(x, x_i[k+2])) <= 0.0001:
        break

F_deriv = []
for i in range (len(x_i)):
    F_deriv.append(deriv.subs(x, x_i[i]))

for i in range (len(x_i)):
    if abs(F_deriv[i]) < abs(F_deriv[i-1]):
        itnA = i
print("Стратегія а:")
print("x =", x_i[itnA])
print("Дохід R = " + str(abs(simplified_func.subs(x, x_i[itnA]))) + "%")

for i in range (len(x_i)):
    if np.sign(F_deriv[i]) != np.sign(F_deriv[i-1]):
        itnB = i
print("Стратегія б:")
print("x =", x_i[itnB-2])
print("Дохід R = " + str(abs(simplified_func.subs(x, x_i[itnB-2]))) + "%")

for i in range (len(x_i)):
    if simplified_func.subs(x, x_i[i]) < simplified_func.subs(x, x_i[i-1]):
        itnC = i
print("Стратегія в:")
print("x =", x_i[itnC])
print("Дохід R = " + str(abs(simplified_func.subs(x, x_i[itnC]))) + "%")


# метод Н'ютона
#x_0 = 0
x_0 = 1
deriv2 = sym.diff(deriv)
x_k = [x_0]
iterations = 7
for k in range (iterations):
    if deriv2.subs(x, x_k[k]) > 0:
        delta_x = -deriv.subs(x, x_k[k])/deriv2.subs(x, x_k[k])
    else:
        delta_x = -deriv.subs(x, x_k[k])
    if delta_x < 0:
        alpha = min(1, (a-x_k[k])/delta_x)
    if delta_x > 0:
```

```
        alpha = min(1, (b-x_k[k])/delta_x)
    for j in range (iterations):
        if simplified_func.subs(x, x_k[k]+alpha*delta_x) >
simplified_func.subs(x, x_k[k]):
            alpha = (0.5**j) * alpha
    if abs(deriv.subs(x, x_k[k-1])) > 0.00001:
        x_k.append(x_k[k]+alpha*delta_x)
for i in range (iterations+1):
    if simplified_func.subs(x, x_k[i]) < simplified_func.subs(x, x_k[i-1]):
        itns = i
print()
print("Метод Н'ютона:")
print("x =", x_k[itns])
print("Кількість ітерацій:", iterations)
print("Дохід R = " + str(abs(simplified_func.subs(x, x_k[itns]))) + "%")
```

## Результат роботи програми:

```
Метод бісекції:
Дохід R = 1.23326036811204%
x = 0.700042724609375
Кількість ітерацій: 14.0

Метод січних:
Кількість ітерацій: 10
Стратегія а:
x = 0.700049377436735
Дохід R = 1.23326037246943%
Стратегія б:
x = 0.700049377436735
Дохід R = 1.23326037246943%
Стратегія в:
x = 0.700049377436735
Дохід R = 1.23326037246943%

Метод Н'ютона:
x = 0.700057339287547
Кількість ітерацій: 7
Дохід R = 1.23326036563822%
```

## Додаток Б

Python 3.8

```
import numpy as np
import matplotlib.pyplot as plt
from sympy.abc import x
import sympy as sym

m = 2
n = 5

matrix = [[0.15, 0.26, 0.18, 0.04, 0.06],
          [0.04, -0.07, -0.05, 0.07, 0.03]]

ri = 0
r = []

# середня прибутковість для кожного активу
for i in range (m):
    for j in range (n):
```

```
        ri += matrix[i][j]
    r.append(round(ri/n, 4))
    ri=0

# формування дисперійно-коваріаційної матриці Q
sigma1 = []
sigma_i_squared = 0
for i in range (m):
    for j in range (n):
        sigma_i_squared += (matrix[i][j]-r[i])**2
    sigma1.append(round(sigma_i_squared/n, 4))
    sigma_i_squared = 0

sigma2 = []
sigma_ik = 0

for k in range (m):
    for j in range (n):
        if k!=i:
            sigma_ik += (matrix[i][j]-r[i])*(matrix[k][j]-r[k])
    sigma2.append(round(sigma_ik/n, 4))

alpha = [[0],
         [1]]
trans_alpha = np.transpose(alpha)
beta = [[1],
        [-1]]
trans_beta = np.transpose(beta)
Q = [[sigma1[0], sigma2[0]],
     [sigma2[1], sigma1[1]]]

rT_alpha = (np.dot(r, alpha))
rT_beta = np.dot(r, beta)
alphaT_Q_alpha = np.dot(np.dot(trans_alpha, Q), alpha)
betaT_Q_alpha = np.dot(np.dot(trans_beta, Q), alpha)*2
betaT_Q_beta = np.dot(np.dot(trans_beta, Q), beta)
Va = 0.00072
ro = 10

# формуємо функцію мінімізації
def func(x):
    return (-rT_alpha - rT_beta*x + (ro/Va**2)*(alphaT_Q_alpha + betaT_Q_alpha*x
+ betaT_Q_beta*x*x - Va)**2)

new = [y[0] for y in func(x)]
simplified_func = new[0].cancel()

#знаходимо похідну від функції F(x)
fu=sym.diff(simplified_func)
deriv = sym.simplify(fu)


# задаємо проміжок
a=0
b=1

print("deriv", deriv)
```

```
# метод бісекції
print("Метод бісекції:")
x_a = a
x_b = b
x_m = 0.5 * (x_a + x_b)
Fa = simplified_func.subs(x, x_a)
Fb = simplified_func.subs(x, x_b)
Fm = simplified_func.subs(x, x_m)
while abs(x_b - x_a) >= 0.0001:
    x_l = 0.5 * (x_a + x_m)
    x_r = 0.5 * (x_m + x_b)
    Fl = simplified_func.subs(x, x_l)
    Fr = simplified_func.subs(x, x_r)
    Fmin = min(Fa, Fb, Fm, Fl, Fr)
    if Fmin == Fa or Fmin == Fl:
        x_b = x_m
        x_m = x_l
        Fb = Fm
        Fm = Fl
    elif Fmin == Fm:
        x_a = x_l
        x_b = x_r
        Fa = Fl
        Fb = Fr
    elif Fmin == Fr or Fmin == Fb:
        x_a = x_m
        x_m = x_r
        Fa = Fm
        Fm = Fr

f_a = simplified_func.subs(x, x_a)
f_b = simplified_func.subs(x, x_b)
f_m = simplified_func.subs(x, x_m)
f_l = simplified_func.subs(x, x_l)
f_r = simplified_func.subs(x, x_r)
minim = min(f_a, f_b, f_m, f_l, f_r)
print("Дохід R = " + str(abs(minim)) + "%")
print("x =:", x_m)
print("Кількість ітерацій:", np.ceil((np.log10(b-a)+4)/np.log10(2)))

# метод січних
print()
print("Метод січних:")
x_0 = 0
x_1 = 1
itns = 6
print("Кількість ітерацій:", itns)
x_i = [x_0, x_1]
for k in range (itns):
    x_i.append(x_i[k]-(deriv.subs(x, x_i[k]))/(deriv.subs(x, x_i[k+1]) -
deriv.subs(x, x_i[k]))*(x_i[k+1]-x_i[k]))
    if abs(deriv.subs(x, x_i[k+2])) <= 0.0001:
        break

F_deriv = []
for i in range (len(x_i)):
    F_deriv.append(deriv.subs(x, x_i[i]))
```

```
for i in range (len(x_i)):
    if abs(F_deriv[i]) < abs(F_deriv[i-1]):
        itnA = i
print("Стратегія a:")
print("x =", x_i[itnA])
print("Дохід R = " + str(abs(simplified_func.subs(x, x_i[itnA]))) + "%")

for i in range (len(x_i)):
    if np.sign(F_deriv[i]) != np.sign(F_deriv[i-1]):
        itnB = i

for i in range (len(x_i)):
    if simplified_func.subs(x, x_i[i]) < simplified_func.subs(x, x_i[i-1]):
        itnC = i
print("Стратегія в:")
print("x =", x_i[itnC])
print("Дохід R = " + str(abs(simplified_func.subs(x, x_i[itnC]))) + "%")


# метод Н'ютона
#x_0 = 0
x_0 = 1
deriv2 = sym.diff(deriv)
x_k = [x_0]
iterations = 7
for k in range (iterations):
    if deriv2.subs(x, x_k[k]) > 0:
        delta_x = -deriv.subs(x, x_k[k])/deriv2.subs(x, x_k[k])
    else:
        delta_x = -deriv.subs(x, x_k[k])
    if delta_x < 0:
        alpha = min(1, (a-x_k[k])/delta_x)
    if delta_x > 0:
        alpha = min(1, (b-x_k[k])/delta_x)
    for j in range (iterations):
        if simplified_func.subs(x, x_k[k]+alpha*delta_x) >
simplified_func.subs(x, x_k[k]):
            alpha = (0.5**j) * alpha
    if abs(deriv.subs(x, x_k[k-1])) > 0.00001:
        x_k.append(x_k[k]+alpha*delta_x)
for i in range (iterations+1):
    if simplified_func.subs(x, x_k[i]) < simplified_func.subs(x, x_k[i-1]):
        itns = i
print()
print("Метод Н'ютона:")
print("x =", x_k[itns])
print("Кількість ітерацій:", iterations)
print("Дохід R = " + str(abs(simplified_func.subs(x, x_k[itns]))) + "%")
```

## Результат роботи програми:

```
Метод бісекції:
Дохід R = 0.0339459935649984%
x =: 0.223541259765625
Кількість ітерацій: 14.0

Метод січних:
Кількість ітерацій: 6
```

```
Стратегія а:
x = 0.223528092665061
Дохід R = 0.0339461095053935%
Стратегія в:
x = 0.223528092665061
Дохід R = 0.0339461095053935%

Метод Н'ютона:
x = 0.567368146223986
Кількість ітерацій: 7
Дохід R = 0.0800206856996510%
```

**Додаток В**
Python 3.8

```
import numpy as np
import sympy as sym
from scipy.linalg import cholesky
import time

start_time = time.time()

n = 3
m = 6
matrix = [[1.2, 1.3, 1.4, 1.5, 1.1, 1.2],
          [1.3, 1, 0.8, 0.9, 1.4, 1.3],
          [0.9, 1.1, 1, 1.1, 1.1, 1.3]]
ro = 10 # константа, яка контролює баланс між прибутковістю та ризиком.
V_a = 0.003

# середня прибутковість для кожного активу
r = []
ri = 0
for i in range (n):
    for j in range (m):
        ri += matrix[i][j]
    r.append(round(ri/m, 4))
    ri = 0
print("r =", r)

# формування дисперійно-коваріаційної матриці Q

# формуємо діагональ матриці
sigma1 = []
sigma_i_squared = 0
for i in range (n):
    for j in range (m):
        sigma_i_squared += (matrix[i][j]-r[i])**2
    sigma1.append(round(sigma_i_squared/m, 4))
    sigma_i_squared = 0

sigma_12_21 = 0
sigma_23_32 = 0
sigma_13_31 = 0
for j in range (m):
    # елементи матриці Q_12 і Q_21
    sigma_12_21 += (matrix[0][j]-r[0])*(matrix[1][j]-r[1])
    # елементи матриці Q_13 і Q_31
```

```
    sigma_13_31 += (matrix[0][j]-r[0])*(matrix[2][j]-r[2])
    # елементи матриці Q_23 і Q_32
    sigma_23_32 += (matrix[1][j]-r[1])*(matrix[2][j]-r[2])

sigma_12_21_m = round(sigma_12_21/m, 5)
sigma_23_32_m = round(sigma_23_32/m, 5)
sigma_13_31_m = round(sigma_13_31/m, 5)


Q = [[sigma1[0], sigma_12_21_m, sigma_13_31_m],
     [sigma_12_21_m, sigma1[1], sigma_23_32_m],
     [sigma_13_31_m, sigma_23_32_m, sigma1[2]]]
print("Q =", Q)

alpha = [[0],
         [0],
         [1]]
B = [[1, 0],
     [0, 1],
     [-1, -1]]

# формуємо функцію мінімізації
x1 = sym.Symbol("x1")
x2 = sym.Symbol("x2")
x = [[x1], [x2]]
def func(x1, x2):
    #rT = np.transpose(r)
    B_x = np.dot(B, x)
    r_minus = np.dot(r, -1)
    alpha_Bx = alpha + B_x
    first = np.dot(r_minus, alpha_Bx)
    second = np.dot(np.dot(np.transpose(alpha + B_x), Q), (alpha + B_x))
    final = first + ro/(V_a**2) * (second - V_a)**2
    return final

new = [y[0] for y in func(x1, x2)]
simplified_func = new[0].cancel()
print()
print("F(x) =", simplified_func, "\n")

# Метод Ньютона

# почтакове наближення
x_i = [1/n, 1/n]

nn = len(x_i)

# знаходимо першу похідну від функції F(x) (градієнт)
def deriv1(func):
    deriv = []
    for i in range(nn):
        deriv.append(sym.diff(func, x[i][0]))
    return deriv

g_first = deriv1(simplified_func)
gg_first = deriv1(simplified_func)

# знаходимо другу похідну від функції F(x)
```

```
def deriv2(func):
    deriv = []
    for i in range(nn):
        for j in range(nn):
            deriv.append(sym.diff(func[i], x[j][0]))
    return deriv

g_second = deriv2(g_first)
gg_second = deriv2(g_first)

# знаходимо значення першої похідної у почтаковій точці (g_0)
grad_at_point_x0 = [[0] * nn] * nn
for i in range (nn):
    for j in range (nn):
        grad_at_point_x0[i][j] = g_first[j].subs(x[i][0], x_i[j])
        g_first[j] = grad_at_point_x0[i][j]
g_i = [grad_at_point_x0[0]]

# знаходимо значення другої похідної у почтаковій точці
grad_at_point1_x0 = [0]*nn*2
for i in range (nn):
    for j in range (nn*2):
        grad_at_point1_x0[j] = g_second[j].subs(x[i][0], x_i[i])
        g_second[j] = grad_at_point1_x0[j]

# формуємо матрицю G_0
G_0 = [[grad_at_point1_x0[0], grad_at_point1_x0[1]],
       [grad_at_point1_x0[2], grad_at_point1_x0[3]]]
G_i = G_0

# задаємо точність розрахунків
eps = 10**(-5)*np.sqrt(n)
print("accuracy =", eps, "\n")

for i in range (100):
    # визначаємо, чи є матриця G_i позитивно визначеною
    is_positive_definite = cholesky(np.array(G_i, dtype=float).any())

    if is_positive_definite:
        # вирішуємо рівняння G_i * p_i = -g_i
        g_minus = np.dot(g_i[i], -1)
        solve = np.linalg.solve(np.array(G_i, dtype=float), np.array(g_minus,
dtype=float))
        p_i = []
        p_i.append(solve)
    else:
        p_i = [np.dot(g_i[i], -1)]

    # знаходимо нову точку x_i
    xx = [0] * nn
    for j in range (nn):
        xx[j] = x_i[j] + (1)*p_i[0][j]
    x_i = xx

    # розраховуємо значення g_i
    gg = []
    for j in range (nn):
        gg.append(gg_first[j].subs([(x1, x_i[0]), (x2, x_i[1])]))
```

```
    g_i.append(gg)

    # знаходимо значення другої похідної у точці x_i
    grad_at_point_xn = []
    for j in range (nn*2):
        grad_at_point_xn.append(gg_second[j].subs([(x1, x_i[0]), (x2, x_i[1])]))

    # формуємо матрицю G_0
    G_i = [[grad_at_point_xn[0], grad_at_point_xn[1]],
           [grad_at_point_xn[2], grad_at_point_xn[3]]]


    # дивимося чи досягли ми заданої точності
    if (abs(g_i[i][0]) < eps and abs(g_i[i][1]) < eps):
        itern = i+1
        break

print("Answer")
print("ro =", ro)
y_i = [x_i[0], x_i[1], 1 - x_i[0] - x_i[1]]
print("y_i =", y_i)
Rp = simplified_func.subs([(x1, y_i[0]), (x2, y_i[1])])
V_i = np.dot(np.dot(y_i, Q), np.transpose(y_i))
print("Rp =", abs(Rp))
print("V =", V_i)
print("number of iterations =", itern)

end_time = time.time()
execution_time = end_time - start_time
print(f"Час виконання програми: {execution_time} секунд")
```

## Результат роботи програми:

```
r = [1.2833, 1.1167, 1.0833]
Q = [[0.0181, -0.02806, -0.00194], [-0.02806, 0.0514, 0.00528], [-0.00194,
0.00528, 0.0147]]

F(x) = 1494.91377777778*x1**4 - 2722.47111111111*x1**3*x2 -
2712.68977777778*x1**3 + 5766.63822222222*x1**2*x2**2 +
934.446222222222*x1**2*x2 + 2184.30044444444*x1**2 - 4122.30222222222*x1*x2**3 -
2709.14488888889*x1*x2**2 + 524.922666666666*x1*x2 - 865.48*x1 +
3427.43511111111*x2**4 - 2325.27466666667*x2**3 + 1838.424*x2**2 - 489.8734*x2 +
151.0167

accuracy = 1.7320508075688774e-05

Answer
ro = 10
y_i = [0.8570854280112548, 0.44925542692247933, -0.3063408549337341]
Rp = 1.26967994095611
V = 0.0030061683315668196
number of iterations = 24
Час виконання програми: 0.4063704013824463 секунд
```

**Додаток Г**
Python 3.8

```
import numpy as np
import sympy as sym
from decimal import Decimal, getcontext
import time

start_time = time.time()

getcontext().prec = 50

n = 5
m = 10
matrix = [[1.2, 1.3, 1.4, 1.5, 1.1, 1.2, 1.1, 1, 1, 1.1],
          [1.3, 1, 0.8, 0.9, 1.4, 1.3, 1.2, 1.1, 1.2, 1.1],
          [0.9, 1.1, 1, 1.1, 1.1, 1.3, 1.2, 1.1, 1, 1.1],
          [1.1, 1.1, 1.2, 1.3, 1.2, 1.2, 1.1, 1.0, 1.1, 1.2],
          [0.8, 0.75, 0.65, 0.75, 0.8, 0.9, 1, 1.1, 1.1, 1.2]]
Rp = 1.15 # очікуваний ріень доходу
ro = 10 # константа, яка контролює баланс між прибутковістю та ризиком

# середня прибутковість для кожного активу
r = []
ri = 0
for i in range (n):
    for j in range (m):
        ri += matrix[i][j]
    r.append(round(ri/m, 4))
    ri = 0
print("r =", r, "\n")

# формування дисперійно-коваріаційної матриці Q

# формуємо діагональ матриці
sigma1 = []
sigma_i_squared = 0
for i in range (n):
    for j in range (m):
        sigma_i_squared += (matrix[i][j]-r[i])**2
    sigma1.append(round(sigma_i_squared/m, 4))
    sigma_i_squared = 0

# формуємо решту елементів матриці Q
sigma_12_21 = 0
sigma_13_31 = 0
sigma_14_41 = 0
sigma_15_51 = 0
sigma_23_32 = 0
sigma_24_42 = 0
sigma_25_52 = 0
sigma_34_43 = 0
sigma_35_53 = 0
sigma_45_54 = 0
for j in range (m):
    # елементи матриці Q_12 і Q_21
    sigma_12_21 += (matrix[0][j]-r[0])*(matrix[1][j]-r[1])
    # елементи матриці Q_13 і Q_31
```

```
    sigma_13_31 += (matrix[0][j]-r[0])*(matrix[2][j]-r[2])
    # елементи матриці Q_14 і Q_41
    sigma_14_41 += (matrix[0][j]-r[0])*(matrix[3][j]-r[3])
    # елементи матриці Q_15 і Q_51
    sigma_15_51 += (matrix[0][j]-r[0])*(matrix[4][j]-r[4])

    # елементи матриці Q_23 і Q_32
    sigma_23_32 += (matrix[1][j]-r[1])*(matrix[2][j]-r[2])
    # елементи матриці Q_24 і Q_42
    sigma_24_42 += (matrix[1][j]-r[1])*(matrix[3][j]-r[3])
    # елементи матриці Q_25 і Q_52
    sigma_25_52 += (matrix[1][j]-r[1])*(matrix[4][j]-r[4])

    # елементи матриці Q_34 і Q_43
    sigma_34_43 += (matrix[2][j]-r[2])*(matrix[3][j]-r[3])
    # елементи матриці Q_35 і Q_53
    sigma_35_53 += (matrix[2][j]-r[2])*(matrix[4][j]-r[4])

    # елементи матриці Q_45 і Q_54
    sigma_45_54 += (matrix[3][j]-r[3])*(matrix[4][j]-r[4])

sigma_12_21_m = round(sigma_12_21/m, 4)
sigma_13_31_m = round(sigma_13_31/m, 4)
sigma_14_41_m = round(sigma_14_41/m, 4)
sigma_15_51_m = round(sigma_15_51/m, 5)
sigma_23_32_m = round(sigma_23_32/m, 4)
sigma_24_42_m = round(sigma_24_42/m, 4)
sigma_25_52_m = round(sigma_25_52/m, 4)
sigma_34_43_m = round(sigma_34_43/m, 4)
sigma_35_53_m = round(sigma_35_53/m, 4)
sigma_45_54_m = round(sigma_45_54/m, 4)

Q = [[sigma1[0], sigma_12_21_m, sigma_13_31_m, sigma_14_41_m, sigma_15_51_m],
     [sigma_12_21_m, sigma1[1], sigma_23_32_m, sigma_24_42_m, sigma_25_52_m],
     [sigma_13_31_m, sigma_23_32_m, sigma1[2], sigma_34_43_m, sigma_35_53_m],
     [sigma_14_41_m, sigma_24_42_m, sigma_34_43_m, sigma1[3], sigma_45_54_m],
     [sigma_15_51_m, sigma_25_52_m, sigma_35_53_m, sigma_45_54_m, sigma1[4]]]
print("Q =", Q, "\n")

# розраховуємо елементи матриць alpha та B
alpha4 = (Rp - r[4]) / (r[3] - r[4])
alpha5 = 1 - alpha4

beta_4i = []
for i in range(n-2):
    beta_4i.append((r[4] - r[i])/(r[3] - r[4]))

beta_5i = []
for i in range(n-2):
    beta_5i.append(-(1 + beta_4i[i]))

# формуємо матриці alpha та B
alpha = [[0],
         [0],
         [0],
         [alpha4],
         [alpha5]]
print("alpha =", alpha, "\n")
```

```
B = [[1, 0, 0],
     [0, 1, 0],
     [0, 0, 1],
     [beta_4i[0], beta_4i[1], beta_4i[2]],
     [beta_5i[0], beta_5i[1], beta_5i[2]]]
print("B =", B, "\n")

# формуємо функцію для подальшої мінімізації
x1 = sym.Symbol("x1")
x2 = sym.Symbol("x2")
x3 = sym.Symbol("x3")
x = [[x1], [x2], [x3]]

def func(x1, x2, x3):
    x = [[x1], [x2], [x3]]
    B_x = np.dot(B, x)
    alpha_Bx = alpha + B_x
    alpha_Bx_T = np.transpose(alpha_Bx)
    r_alpha = np.dot(r, alpha)
    r_Bx = np.dot(r, B_x)
    first = np.dot(np.dot(alpha_Bx_T, Q), alpha_Bx)
    return (first + ro / (Rp*Rp) * (r_alpha - Rp + r_Bx)**2)

new = [y[0] for y in func(x1, x2, x3)]
simplified_func = new[0].cancel()
print("F(x) =", simplified_func, "\n")

# Метод спяжених градієнтів

# почтакове наближення
x_i = [[1/n, 1/n, 1/n]]

nn = len(x_i[0])

def deriv(func):
    deriv = []
    for i in range(nn):
         deriv.append(sym.diff(func, x[i][0]))
         deriv[i] = sym.simplify(deriv[i])
    return deriv

# знаходимо похідну від функції F(x) (градієнт)
g = deriv(simplified_func)
gg = deriv(simplified_func)

# знаходимо значення градієнту у почтаковій точці
grad_at_point_x0 = [[0] * nn] * nn
for i in range (nn):
    for j in range (nn):
        grad_at_point_x0[i][j] = g[j].subs(x[i][0], x_i[0][j])
        g[j] = grad_at_point_x0[i][j]
g_i = [grad_at_point_x0[0]]

# знаходимо p0
p_i = [[]]
for i in range (nn):
    p_i[0].append(-g_i[0][i])
```

```
# задаємо точність розрахунків
eps = 10**(-5)*np.sqrt(n)
print("accuracy =", eps, "\n")

s = sym.Symbol("s")

for i in range (1, n):
    # знаходимо нову точку x_i
    xx = [0] * nn
    for j in range (nn):
        xx[j] = x_i[i-1][j] + p_i[i-1][j]*s
    x_i.append(xx)

    # знаходимо значення градієнту у точці x_i
    grad = []
    for j in range (nn):
        grad.append(sym.simplify(gg[j].subs([(x1, x_i[i][0]), (x2, x_i[i][1]),
(x3, x_i[i][2])])))
    g_i.append(grad)

    # знаходимо s вирішивши рівняння p_(i-1) * g_i = 0
    gi_T = np.transpose(g_i[i])
    eq0 = np.dot(p_i[i-1], gi_T)
    eq = sym.nsolve(eq0, s, -1)

    # розраховуємо значення x_i
    for j in range (nn):
        x_i[i][j] = x_i[i][j].subs(s, eq)

    # розраховуємо значення ризику
    V_i = simplified_func.subs([(x1, x_i[i][0]), (x2, x_i[i][1]), (x3,
x_i[i][2])])

    # розраховуємо значення прибутку
    y_4 = 0
    y_5 = 0
    y_i = [0]*n
    for j in range(n-2):
        y_i[j] = (x_i[i][j])
        y_4 += beta_4i[j] * x_i[i][j]
        y_5 += beta_5i[j] * x_i[i][j]
    y_i[n-2] = alpha4 + y_4
    y_i[n-1] = alpha5 + y_5

    rr = []
    yy_i = [0]*n
    for j in range (n):
        rr.append(Decimal(r[j]))
        yy_i[j] = Decimal(str(y_i[j]))
    R_i = np.dot(rr, yy_i)

    # розраховуємо значення g_i
    for j in range (nn):
        g_i[i][j] = g_i[i][j].subs(s, eq)

    if (float(i/n) != 0):
        gg_1_T = np.transpose(g_i[i])
```

```
        gg_0_T = np.transpose(g_i[i-1])
        beta = np.dot(g_i[i], gg_1_T) / np.dot(g_i[i-1], gg_0_T)
        g_minus = []
        for j in range (nn):
            g_minus.append(-g_i[i][j])
        p1 = list(g_minus + np.dot(beta, p_i[i-1]))
        p_i.append(p1)
    else:
        p_i.append(g_minus)


    # дивимося чи досягли ми заданої точності
    if (abs(g_i[i][0]) < eps and abs(g_i[i][1]) < eps and abs(g_i[i][2]) < eps):
        itern = i
        break

print("\n Answer")
print("ro =", ro)
for i in range (n):
    print("y" + str(i+1) + " = " + str(y_i[i]))
print("V =", V_i)
print("Rp =", R_i)
print("number of iterations =", itern)

end_time = time.time()
execution_time = end_time - start_time
print(f"Час виконання програми: {execution_time} секунд")
```

## Результат роботи програми:

```
r = [1.19, 1.13, 1.09, 1.15, 0.905]

Q = [[0.0249, -0.0187, -0.0011, 0.0085, -0.02195], [-0.0187, 0.0321, 0.0033, -
0.0035, 0.0099], [-0.0011, 0.0033, 0.0109, 0.0015, 0.0035], [0.0085, -0.0035,
0.0015, 0.0065, -0.0058], [-0.02195, 0.0099, 0.0035, -0.0058, 0.0307]]

alpha = [[0], [0], [0], [1.0], [0.0]]

B = [[1, 0, 0], [0, 1, 0], [0, 0, 1], [-1.1632653061224492, -0.9183673469387754,
-0.7551020408163271], [0.16326530612244916, -0.08163265306122458, -
0.24489795918367285]]

F(x) = 0.00977426072469804*x1**2 - 0.0243456892961266*x1*x2 +
0.000456809662640553*x1*x3 - 1.63265306122474e-5*x1 + 0.0417292794668888*x2**2 +
0.0106287380258226*x2*x3 - 0.0179918367346939*x2 + 0.0103226988754686*x3**2 -
0.00397551020408165*x3 + 0.0065

accuracy = 2.23606797749979e-05

 Answer
ro = 10
y1 = 0.420176225595182
y2 = 0.336896777826694
y3 = 0.00982214309458051
y4 = 0.194411853966814
y5 = 0.0386929995167289
V = 0.00344635006339475
Rp = 1.1499999999999992366504695995255169727045085892314
number of iterations = 3
Час виконання програми: 0.41980981826782227 секунд
```

**Додаток Д**
Python 3.8

```
import numpy as np
import sympy as sym
import time

start_time = time.time()

n = 5
m = 10
matrix = [[1.2, 1.3, 1.4, 1.5, 1.1, 1.2, 1.1, 1, 1, 1.1],
          [1.3, 1, 0.8, 0.9, 1.4, 1.3, 1.2, 1.1, 1.2, 1.1],
          [0.9, 1.1, 1, 1.1, 1.1, 1.3, 1.2, 1.1, 1, 1.1],
          [1.1, 1.1, 1.2, 1.3, 1.2, 1.2, 1.1, 1.0, 1.1, 1.2],
          [0.8, 0.75, 0.65, 0.75, 0.8, 0.9, 1, 1.1, 1.1, 1.2]]

V_a = 0.00375
ro = 10 # константа, яка контролює баланс між прибутковістю та ризиком

# середня прибутковість для кожного активу
r = [1.19, 1.13, 1.09, 1.15, 0.905]

# формування дисперійно-коваріаційної матриці Q

Q = [[0.0249, -0.0187, -0.0011, 0.0085, -0.02195],
     [-0.0187, 0.0321, 0.0033, -0.0035, 0.0099],
     [-0.0011, 0.0033, 0.0109, 0.0015, 0.0035],
     [0.0085, -0.0035, 0.0015, 0.0065, -0.0058],
     [-0.02195, 0.0099, 0.0035, -0.0058, 0.0307]]

alpha = [[0],
         [0],
         [0],
         [0],
         [1]]
B = [[1, 0, 0, 0],
     [0, 1, 0, 0],
     [0, 0, 1, 0],
     [0, 0, 0, 1],
     [-1, -1, -1, -1]]

x1 = sym.Symbol("x1")
x2 = sym.Symbol("x2")
x3 = sym.Symbol("x3")
x4 = sym.Symbol("x4")
x5 = sym.Symbol("x5")
y = [[x1], [x2], [x3], [x4], [x5]]
x = [[x1], [x2], [x3], [x4], [x5]]

def func(x1, x2, x3, x4, x5):
    y_T = np.transpose(y)
    e = [1, 1, 1, 1, 1]
    V = np.dot(np.dot(y_T, Q), y)
    R = np.dot(r, y)
    S = np.dot(e, y)
    F = ro * (S - 1)**2 + ro / (V_a*V_a) * (V - V_a)**2 - R
    return F
```

```
simplified_func = func(x1, x2, x3, x4, x5)[0][0]
print("F(x) =", simplified_func, "\n")

# Метод спяжених градієнтів

# почтакове наближення
x_i = [[1/n, 1/n, 1/n, 1/n, 1/n]]

nn = len(x_i[0])

def deriv(func):
    deriv = []
    for i in range(nn):
         deriv.append(sym.diff(func, x[i][0]))
         deriv[i] = sym.simplify(deriv[i])
    return deriv

# знаходимо похідну від функції F(x) (градієнт)
g = deriv(simplified_func)
gg = deriv(simplified_func)

# знаходимо значення градієнту у почтаковій точці
grad_at_point_x0 = [[0] * nn] * nn
for i in range (nn):
    for j in range (nn):
        grad_at_point_x0[i][j] = g[j].subs(x[i][0], x_i[0][j])
        g[j] = grad_at_point_x0[i][j]
g_i = [grad_at_point_x0[0]]

# знаходимо p0
p_i = [[]]
for i in range (nn):
    p_i[0].append(-g_i[0][i])

# задаємо точність розрахунків
eps = 10**(-5)*np.sqrt(n)
print("accuracy =", eps, "\n")

s = sym.Symbol("s")

for i in range (1, n+1):
    # знаходимо нову точку x_i
    xx = [0] * nn
    for j in range (nn):
        xx[j] = x_i[i-1][j] + p_i[i-1][j]*s
    x_i.append(xx)

    # знаходимо значення градієнту у точці x_i
    grad = []
    for j in range (nn):
        grad.append(sym.simplify(gg[j].subs([(x1, x_i[i][0]), (x2, x_i[i][1]),
(x3, x_i[i][2]), (x4, x_i[i][3]), (x5, x_i[i][4])])))
    g_i.append(grad)

    # знайдемо s вирішивши рівняння p_(i-1) * g_i = 0
    gi_T = np.transpose(g_i[i])
    eq0 = np.dot(p_i[i-1], gi_T)
    eq = sym.nsolve(eq0, s, 0.1)
```

```
    # розраховуємо значення x_i
    for j in range (nn):
        x_i[i][j] = x_i[i][j].subs(s, eq)

    # розраховуємо значення g_i
    for j in range (nn):
        g_i[i][j] = g_i[i][j].subs(s, eq)

    if (float(i/n) != 0):
        gg_1_T = np.transpose(g_i[i])
        gg_0_T = np.transpose(g_i[i-1])
        beta0 = gg_1_T - gg_0_T
        beta = np.dot(g_i[i], beta0) / np.dot(g_i[i-1], gg_0_T)
        g_minus = []
        for j in range (nn):
            g_minus.append(-g_i[i][j])
        p1 = list(g_minus + np.dot(beta, p_i[i-1]))
        p_i.append(p1)
    else:
        p_i.append(g_minus)

    itern = i

    # дивимося чи досягли ми заданої точності
    if (abs(g_i[i][0]) < eps and abs(g_i[i][1]) < eps and abs(g_i[i][2]) < eps
and abs(g_i[i][3]) < eps and abs(g_i[i][4]) < eps):
        break

y_i = []
print("Answer")
print("ro =", ro, "\n")
for i in range (nn):
    y_i.append(x_i[nn][i])
    print("y" + str(i+1) + " = " + str(y_i[i]))
Rp = simplified_func.subs([(x1, y_i[0]), (x2, y_i[1]), (x3, y_i[2]), (x4,
y_i[3]), (x5, y_i[4])])
print("\nRp =", abs(Rp))
V_i = np.dot(np.dot(y_i, Q), y_i)
print("V =", V_i)
print("amount of iterations =", itern)

end_time = time.time()
execution_time = end_time - start_time
print(f"Час виконання програми: {execution_time} секунд")
```

## Результат роботи програми:

```
F(x) = -1.19*x1 - 1.13*x2 - 1.09*x3 - 1.15*x4 - 0.905*x5 + 10*(x1 + x2 + x3 + x4
+ x5 - 1)**2 + 711111.111111111*(x1*(0.0249*x1 - 0.0187*x2 - 0.0011*x3 +
0.0085*x4 - 0.02195*x5) + x2*(-0.0187*x1 + 0.0321*x2 + 0.0033*x3 - 0.0035*x4 +
0.0099*x5) + x3*(-0.0011*x1 + 0.0033*x2 + 0.0109*x3 + 0.0015*x4 + 0.0035*x5) +
x4*(0.0085*x1 - 0.0035*x2 + 0.0015*x3 + 0.0065*x4 - 0.0058*x5) + x5*(-0.02195*x1
+ 0.0099*x2 + 0.0035*x3 - 0.0058*x4 + 0.0307*x5) - 0.00375)**2

accuracy = 2.23606797749979e-05

Answer
```

```
ro = 10

y1 = 0.138940810304849
y2 = 0.274742080141394
y3 = 0.233386050947762
y4 = 0.199450106885553
y5 = 0.191074373252768

Rp = 1.11799485611799
V = 0.00377222745951711
amount of iterations = 5
Час виконання програми: 5.3985819816589355 секунд
```

**Додаток Е**
Python 3.8

```
import sympy as sym
from scipy.linalg import cholesky
import time

start_time = time.time()

n = 5
m = 10
matrix = [[1.2, 1.3, 1.4, 1.5, 1.1, 1.2, 1.1, 1, 1, 1.1],
          [1.3, 1, 0.8, 0.9, 1.4, 1.3, 1.2, 1.1, 1.2, 1.1],
          [0.9, 1.1, 1, 1.1, 1.1, 1.3, 1.2, 1.1, 1, 1.1],
          [1.1, 1.1, 1.2, 1.3, 1.2, 1.2, 1.1, 1.0, 1.1, 1.2],
          [0.8, 0.75, 0.65, 0.75, 0.8, 0.9, 1, 1.1, 1.1, 1.2]]

V_a = 0.00375
ro = 10 # константа, яка контролює баланс між прибутковістю та ризиком

# середня прибутковість для кожного активу
r = [1.19, 1.13, 1.09, 1.15, 0.905]

# формування дисперійно-коваріаційної матриці Q
Q = [[0.0249, -0.0187, -0.0011, 0.0085, -0.02195],
     [-0.0187, 0.0321, 0.0033, -0.0035, 0.0099],
     [-0.0011, 0.0033, 0.0109, 0.0015, 0.0035],
     [0.0085, -0.0035, 0.0015, 0.0065, -0.0058],
     [-0.02195, 0.0099, 0.0035, -0.0058, 0.0307]]

alpha = [[0],
         [0],
         [0],
         [0],
         [1]]
B = [[1, 0, 0, 0],
     [0, 1, 0, 0],
     [0, 0, 1, 0],
     [0, 0, 0, 1],
     [-1, -1, -1, -1]]

x1 = sym.Symbol("x1")
x2 = sym.Symbol("x2")
x3 = sym.Symbol("x3")
x4 = sym.Symbol("x4")
x5 = sym.Symbol("x5")
```

```
y = [[x1], [x2], [x3], [x4], [x5]]
x = [[x1], [x2], [x3], [x4], [x5]]

def func(x1, x2, x3, x4, x5):
    y_T = np.transpose(y)
    e = [1, 1, 1, 1, 1]
    V = np.dot(np.dot(y_T, Q), y)
    R = np.dot(r, y)
    S = np.dot(e, y)
    F = ro * (S - 1)**2 + ro / (V_a*V_a) * (V - V_a)**2 - R
    return F
simplified_func = func(x1, x2, x3, x4, x5)[0][0]
print("F(x) =", simplified_func)

# задаємо точність розрахунків
eps = 10**(-5)*np.sqrt(n)
print("accuracy =", eps, "\n")

# Метод Ньютона

# почтакове наближення
x_i = [1/n, 1/n, 1/n, 1/n, 1/n]

nn = len(x_i)

# знаходимо першу похідну від функції F(x) (градієнт)
def deriv1(func):
    deriv = []
    for i in range(nn):
        deriv.append(sym.diff(func, x[i][0]))
    return deriv

g_first = deriv1(simplified_func)
gg_first = deriv1(simplified_func)

# знаходимо другу похідну від функції F(x)
def deriv2(func):
    deriv = []
    for i in range(nn):
        for j in range(nn):
            deriv.append(sym.diff(func[i], x[j][0]))
    return deriv

g_second = deriv2(g_first)
gg_second = deriv2(g_first)

# знаходимо значення першої похідної у почтаковій точці (g_0)
grad_at_point_x0 = [[0] * nn] * nn
for i in range (nn):
    for j in range (nn):
        grad_at_point_x0[i][j] = g_first[j].subs(x[i][0], x_i[j])
        g_first[j] = grad_at_point_x0[i][j]
g_i = [grad_at_point_x0[0]]

# знаходимо значення другої похідної у почтаковій точці
grad_at_point1_x0 = [0]*nn*5
for i in range (nn):
    for j in range (nn*5):
```

```
        grad_at_point1_x0[j] = g_second[j].subs(x[i][0], x_i[i])
        g_second[j] = grad_at_point1_x0[j]

# формуємо матрицю G_0
G_0 = [[grad_at_point1_x0[0], grad_at_point1_x0[1], grad_at_point1_x0[2],
grad_at_point1_x0[3], grad_at_point1_x0[4]],
       [grad_at_point1_x0[5], grad_at_point1_x0[6], grad_at_point1_x0[7],
grad_at_point1_x0[8], grad_at_point1_x0[9]],
       [grad_at_point1_x0[10], grad_at_point1_x0[11], grad_at_point1_x0[12],
grad_at_point1_x0[13], grad_at_point1_x0[14]],
       [grad_at_point1_x0[15], grad_at_point1_x0[16], grad_at_point1_x0[17],
grad_at_point1_x0[18], grad_at_point1_x0[19]],
       [grad_at_point1_x0[20], grad_at_point1_x0[21], grad_at_point1_x0[22],
grad_at_point1_x0[23], grad_at_point1_x0[24]]]
G_i = G_0

# задаємо точність розрахунків
eps = 10**(-5)*np.sqrt(n)
print("accuracy =", eps, "\n")

for i in range (50):
    # визначаємо, чи є матриця G_i позитивно визначеною
    is_positive_definite = cholesky(np.array(G_i, dtype=float).any())

    if is_positive_definite:
        # вирішуємо рівняння G_i * p_i = -g_i
        g_minus = np.dot(g_i[i], -1)
        solve = np.linalg.solve(np.array(G_i, dtype=float), np.array(g_minus,
dtype=float))
        p_i = []
        p_i.append(solve)
    else:
        p_i = [np.dot(g_i[i], -1)]

    # знаходимо нову точку x_i
    xx = [0] * nn
    for j in range (nn):
        xx[j] = x_i[j] + (1)*p_i[0][j]
    x_i = xx

    # розраховуємо значення g_i
    gg = []
    for j in range (nn):
        gg.append(gg_first[j].subs([(x1, x_i[0]), (x2, x_i[1]), (x3, x_i[2]),
(x4, x_i[3]), (x5, x_i[4])]))
    g_i.append(gg)

    # знаходимо значення другої похідної у точці x_i
    grad_at_point_xn = []
    for j in range (nn*5):
        grad_at_point_xn.append(gg_second[j].subs([(x1, x_i[0]), (x2, x_i[1]),
(x3, x_i[2]), (x4, x_i[3]), (x5, x_i[4])]))

    # формуємо матрицю G_0
    G_i = [[grad_at_point_xn[0], grad_at_point_xn[1], grad_at_point_xn[2],
grad_at_point_xn[3], grad_at_point_xn[4]],
           [grad_at_point_xn[5], grad_at_point_xn[6], grad_at_point_xn[7],
grad_at_point_xn[8], grad_at_point_xn[9]],
```

```
            [grad_at_point_xn[10], grad_at_point_xn[11], grad_at_point_xn[12],
grad_at_point_xn[13], grad_at_point_xn[14]],
            [grad_at_point_xn[15], grad_at_point_xn[16], grad_at_point_xn[17],
grad_at_point_xn[18], grad_at_point_xn[19]],
            [grad_at_point_xn[20], grad_at_point_xn[21], grad_at_point_xn[22],
grad_at_point_xn[23], grad_at_point_xn[24]]]

    # дивимося чи досягли ми заданої точності
    if (abs(g_i[i][0]) < eps and abs(g_i[i][1]) < eps and abs(g_i[i][2]) < eps
and abs(g_i[i][3]) < eps and abs(g_i[i][4]) < eps):
        itern = i+1
        break

print("Answer")
print("ro =", ro)
print("y_i =", xx)
Rp = simplified_func.subs([(x1, xx[0]), (x2, xx[1]), (x3, xx[2]), (x4, xx[3]),
(x5, xx[4])])
print("Rp =", abs(Rp))
V_i = np.dot(np.dot(xx, Q), xx)
print("V =", V_i)
print("amount of iterations =", itern)

end_time = time.time()
execution_time = end_time - start_time
print(f"Час виконання програми: {execution_time} секунд")
```

## Результат роботи програми:

```
F(x) = -1.19*x1 - 1.13*x2 - 1.09*x3 - 1.15*x4 - 0.905*x5 + 10*(x1 + x2 + x3 + x4
+ x5 - 1)**2 + 711111.111111111*(x1*(0.0249*x1 - 0.0187*x2 - 0.0011*x3 +
0.0085*x4 - 0.02195*x5) + x2*(-0.0187*x1 + 0.0321*x2 + 0.0033*x3 - 0.0035*x4 +
0.0099*x5) + x3*(-0.0011*x1 + 0.0033*x2 + 0.0109*x3 + 0.0015*x4 + 0.0035*x5) +
x4*(0.0085*x1 - 0.0035*x2 + 0.0015*x3 + 0.0065*x4 - 0.0058*x5) + x5*(-0.02195*x1
+ 0.0099*x2 + 0.0035*x3 - 0.0058*x4 + 0.0307*x5) - 0.00375)**2
accuracy = 2.23606797749979e-05

accuracy = 2.23606797749979e-05

Answer
ro = 10
y_i = [0.4444523553426469, 0.3536881275901129, 0.010103187865222506,
0.20026124599556241, 0.04441007363660277]
Rp = 1.18201289746015
V = 0.0037589570664479043
amount of iterations = 20
Час виконання програми: 6.481479167938232 секунд
```

## Література


1. Бейко І.В., Зінько П.М., Наконечний О.Г. Задачі, методи і алгоритми оптимізації. Рівне: НУВГП, 2011. 624 с.
2. Григорків В.С., Григорків М.В., Ярошенко О.І. Оптимізаційні методи та моделі. Чернівці : Чернівецький нац. ун-т, 2022. 440 с.
3. Кублікова Т.Б., Кубліков В.К. Інвестиції на ринку цінних паперів. Одеса: ОРІДУ НАДУ, 2006. 428 с.
4. Кузьмичов А.І. Оптимізаційні методи і моделі: практикум в EXCEL. Київ: ВПЦ АМУ, 2013. 438 с.
5. Bartholomew-Biggs M.Nonlinear Optimization with FinancialApplications. Kluwer Academic Publishers, 2005. 261 p.
6. Wenyu San, Ya-Xiang Yuan. Optimization Theory and Methods: Nonlinear Programming. Springer Science+Business Media, LLC, 2006. 687 p.