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»

«Застосування методів факторного аналізу в економіці»

«Factor analysis methods and its applications in economics»

Виконав: здобувач денної форми навчання
спеціальності 113 Прикладна математика
Освітня програма «Прикладна математика»
Давидов Кирило Анатолійович

Керівник: канд. фіз.-мат. наук, доц. Васильєв О. Б. _____
Рецензент: канд. техн. наук, доц. Мороз В. В.

Рекомендовано до захисту:

Протокол засідання кафедри

№ ___ від _____ 2025 р.

Завідувач кафедри

_____ _____

Захищено на засіданні ЕК № _____

Протокол № ___ від _____ 2025 р.

Оцінка _____ / _____ / _____

Голова ЕК

_____ _____

Одеса — 2025 р.

ЗМІСТ

ВСТУП

Актуальність теми дослідження

У сучасному світі багатовимірних даних одним із ключових завдань статистичного аналізу є зведення великих масивів інформації до більш керованої форми без втрати суттєвих характеристик досліджуваних явищ. Факторний аналіз посідає особливе місце серед методів багатовимірної статистики, оскільки дозволяє виявити приховані структури у даних та ідентифікувати латентні змінні, які пояснюють кореляційні зв'язки між спостережуваними показниками.

Метод головних факторів та метод мінімальних залишків є двома фундаментальними підходами до розв'язання проблеми факторного аналізу, кожен з яких має свої теоретичні основи та практичні переваги. Актуальність дослідження цих методів зумовлена їх широким застосуванням у психології, соціології, економіці, маркетингових дослідженнях та інших галузях, де необхідно аналізувати складні багатофакторні залежності.

Мета дослідження

Мета дослідження полягає у комплексному аналізі методу головних факторів та методу мінімальних залишків у факторному аналізі, порівнянні їх теоретичних основ, алгоритмічних особливостей та практичної ефективності.

Завдання дослідження

- проаналізувати теоретичні основи факторного аналізу та місце методу головних факторів і методу мінімальних залишків у системі методів багатовимірної статистики;
- дослідити математичний апарат та алгоритмічні особливості методу головних факторів;

- розглянути принципи функціонування методу мінімальних залишків та його обчислювальні аспекти;
- провести порівняльний аналіз переваг та недоліків досліджуваних методів;
- сформулювати рекомендації щодо вибору оптимального методу залежно від специфіки досліджуваних даних.

Об'єкт дослідження

Об'єкт дослідження – факторний аналіз як метод багатовимірної статистики для виявлення латентних структур у даних.

Предмет дослідження

Предмет дослідження – метод головних факторів та метод мінімальних залишків, їх математичні основи, алгоритмічні особливості та порівняльні характеристики.

Методи дослідження

У роботі використовуються методи математичної статистики, лінійної алгебри, чисельні методи оптимізації, а також методи статистичного моделювання та комп'ютерного експерименту для практичної апробації досліджуваних підходів.

Наукова новизна роботи

Наукова новизна роботи полягає у систематизації та поглибленому порівняльному аналізі методу головних факторів та методу мінімальних залишків, розробці практичних рекомендацій щодо їх застосування в залежності від характеристик вхідних даних та цілей дослідження.

Практична значущість дослідження

Практична значущість дослідження визначається можливістю використання отриманих результатів у прикладних дослідженнях, де застосовується факторний аналіз, а також у навчальному процесі при вивченні методів багатовимірної статистики.

РОЗДІЛ 1

ТЕОРЕТИЧНА ЧАСТИНА

1.1 Постановка задачі

Завданням дослідження є проведення порівняльного аналізу ефективності виробничо-господарської діяльності 40 підприємств(табл. 2.15) на основі комплексної оцінки ряду економічних показників. Для цього передбачено застосування методів факторного аналізу, зокрема методу головних факторів та методу мінімальних залишків. Ці методи дають змогу зменшити кількість вихідних показників шляхом виявлення прихованих структурних залежностей між ними та виділення найбільш впливових факторів, що визначають загальний рівень ефективності.

У якості об'єкта аналізу обрано такі показники:

- Продуктивність праці одного працівника;
- Рівень рентабельності обсягу продукції;
- Капіталовіддача основних засобів;
- Капіталоозброєність праці одного працівника;
- Рівень рентабельності основних засобів (капіталорентабельність);

Метою є скорочення кількості цих змінних шляхом їхньої трансформації у меншу кількість узагальнених факторів, що зберігають основну інформацію для подальшого порівняння підприємств за рівнем ефективності.

1.2 Факторний аналіз

Факторний аналіз — це метод багатовимірної статистики, що дає змогу дослідити структуру зв'язків між великою кількістю змінних і звести їх до меншого числа прихованих факторів (латентних змінних). Основна ідея полягає в тому, щоб замінити безліч взаємопов'язаних показників меншою кількістю факторів, які пояснюють спільну дисперсію вихідних змінних.

1.2.1 Мета та завдання факторного аналізу

Метою факторного аналізу є виявлення прихованих структур у даних, які впливають на спостережувані змінні, але самі безпосередньо не вимірюються. Завдяки цьому можна:

- зменшити розмірність даних;
- виділити головні напрямки змінності;
- групувати об'єкти за схожістю структури факторів;
- виявити надлишкові або слабоінформативні змінні;
- підвищити інтерпретованість результатів аналізу.

Факторний аналіз також широко використовується як інструмент попередньої обробки перед кластеризацією, регресійним аналізом або побудовою прогнозних моделей.

1.2.2 Історична довідка

Витоки факторного аналізу сягають початку XX століття. Засновником методу вважається британський психолог Чарльз Спірмен (Charles Spearman), який у 1904 році запропонував модель, що пояснювала результати тестування інтелектуальних здібностей за допомогою одного загального фактора ("g-фактора"). У подальші десятиліття метод розвивався завдяки внескам таких дослідників, як Терстоун (Thurstone), Гутман (Guttman), Гарман (Harman), Хотеллінг (Hotelling) та інших.

З розвитком обчислювальної техніки факторний аналіз став широко застосовуваним у різних науках — від психології до економіки та технічних дисциплін.

1.2.3 Сфери застосування

Факторний аналіз має широкий спектр застосувань, зокрема:

- в економіці — для аналізу ефективності підприємств, дослідження ринкових характеристик, оцінки фінансових ризиків;

- у соціології та психології — для побудови шкал тестування, виявлення структури особистісних якостей;
- у маркетингу — для вивчення поведінки споживачів, сегментації ринку;
- у медицині — для виявлення латентних причин захворювань;
- у техніці та виробництві — для оптимізації процесів та зменшення кількості контрольованих параметрів.

1.2.4 Основні підходи до факторного аналізу

У класичному факторному аналізі існує кілька підходів до побудови факторної моделі:

- метод головних факторів (Principal Factor Method);
- метод максимальної правдоподібності (Maximum Likelihood Method);
- метод мінімальних залишків (Method of Least Residuals);
- інші варіанти, зокрема alpha factoring, image factoring тощо.

У рамках цієї роботи розглядатимуться деякі з вищезазначених методів, зокрема метод головних факторів та метод мінімальних залишків, які дозволяють детально проаналізувати взаємозв'язки між економічними показниками підприємств і зробити висновки щодо структури ефективності їхньої діяльності.

1.3 Метод головних факторів

1.3.1 Загальний опис

Метод головних факторів (Principal Factor Method)[1][2][4] є одним із фундаментальних підходів у факторному аналізі, що використовується для виявлення латентних змінних, які пояснюють спільну варіацію між спостережуваними показниками. Цей метод виник як більш точна та математично обґрунтована альтернатива до центроїдного методу, запропонованого Л. Л. Тарстоуном у 1930-х роках. З розвитком обчислювальної техніки метод головних факторів набув широкого застосування в соціальних, психологічних

та економічних дослідженнях.

На відміну від центроїдного методу, який базується на наближених обчисленнях, метод головних факторів передбачає точне спектральне розкладання спеціально скоригованої кореляційної матриці, в якій на діагоналі замість одиниць розміщені оцінки спільності змінних. Таким чином, аналіз фокусується виключно на спільній дисперсії — тій частині варіації, що може бути пояснена факторами і не враховує унікальну та випадкову складову.

Основна перевага методу полягає в послідовному виділенні факторів, кожен з яких репрезентує максимально можливу частину залишкової спільної дисперсії. Це дозволяє ефективно зменшити розмірність даних і зберегти найважливішу інформацію про кореляційну структуру змінних. Хоча метод вимагає значних обчислювальних ресурсів, сучасні комп'ютерні засоби забезпечують його практичну реалізацію без суттєвих обмежень.

1.3.2 Підготовка даних

Перед застосуванням методу головних факторів необхідно здійснити ретельну підготовку вихідних даних. Цей процес включає кілька важливих етапів.

Формування кореляційної матриці

Для обчислення кореляційної матриці[1][2][4] використовується наступна формула. Нехай $\mathbf{X}$ є стандартизованою матрицею даних розмірності $n \times p$, де n — кількість спостережень, а p — кількість змінних. Тоді кореляційна матриця $\mathbf{R}$ розмірності $p \times p$ обчислюється як:

$$\mathbf{R} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}$$

де $\mathbf{X}^T$ — транспонована матриця стандартизованих даних.

Елементи кореляційної матриці r_{ij} представляють коефіцієнти кореляції

Пірсона між i-ою та j-ою змінними:

$$r_{ij} = \frac{1}{n-1} \sum_{k=1}^{n} x_{ki} x_{kj}$$

де x_{ki} та x_{kj} — стандартизовані значення i-ої та j-ої змінних для k-ого спостереження.

Діагональні елементи кореляційної матриці дорівнюють одиниці ($r_{ii} = 1$), оскільки кореляція змінної із самою собою завжди дорівнює одиниці, а матриця є симетричною ($r_{ij} = r_{ji}$). Після цього діагональні елементи змінюють на оцінку спільностей для отримання редукованої матриці

Оцінка спільностей

Одним з найпоширеніших методів оцінки спільностей є використання квадратів множинних кореляцій (Squared Multiple Correlations, SMC)[2][4]. Ці значення представляють квадрат множинної кореляції кожної змінної з усіма іншими змінними і, за певних умов, є нижніми границями невідомих справжніх спільностей. Для отримання SMC необхідно:

1) Обчислити обернену матрицю $\mathbf{R}^{-1}$ кореляційної матриці з одиницями на головній діагоналі
2) Взяти обернені значення діагональних елементів $\mathbf{R}^{-1}$
3) Відняти кожне таке обернене значення від 1.0

Математично це виражається як:

$$h_i^2 = 1 - \frac{1}{r^{ii}} \tag{1.1}$$

де r^{ii} – i-й діагональний елемент оберненої кореляційної матриці.

Проблеми з сингулярними матрицями

У деяких дослідженнях кореляційна матриця може бути сингулярною або близькою до сингулярної. В таких випадках неможливо обчислити обернену матрицю, що робить неможливим обчислення SMC. Це може

статися через високу мультиколінеарність між змінними або недостатній розмір вибірки[4].

1.3.3 Метод Хотеллінга

Ітераційний метод Хотеллінга[3][4] представляє сучасний підхід до реалізації методу головних факторів, який базується на ітераційному обчисленні власних значень та власних векторів кореляційної матриці. Цей метод є особливо ефективним для великих матриць і забезпечує високу точність обчислень.

Алгоритм ітераційного процесу

Ітераційний алгоритм Хотеллінга складається з наступних кроків:

1) **Ініціалізація**: Обчислення початкового наближення власного вектора шляхом сумування стовпців редукованої матриці та нормалізації результату.
2) **Степеневий метод**: На кожній ітерації k виконується наступна послідовність операцій:
   - Піднесення кореляційної матриці до степеня 2^k: $\mathbf{R}^{2^k}$
   - Обчислення вектора $\mathbf{p}$ шляхом множення матриці ($\mathbf{R}^{2^k}$) на попередній нормалізований вектор $\mathbf{S}^k$:

$$\mathbf{p}_k = (\mathbf{R}^{2^k})\mathbf{S}^k \tag{1.2}$$

     де $\mathbf{S}^k$ – нормалізований вектор з попередньої ітерації, $\mathbf{R}^{2^k}$ – кореляційна матриця в степені 2^k.
   - Нормалізація отриманого вектора відносно його максимального елемента:

$$\mathbf{v}_k = \frac{\mathbf{p}_k}{\max\limits_{1 \le i \le n} (|(p_k)_i|)} \tag{1.3}$$

     де $\max\limits_{1 \le i \le n} (|(p_k)_i|)$ — максимальний за абсолютною величиною елемент вектора; n — кількість показників (розмірність вектора); $(p_k)_i$ — i-й компонент вектора p_k.

3) **Прискорення збіжності**: Використання подвоєння степеня матриці на кожній ітерації для прискорення збіжності до домінуючого власного вектора.
4) **Обчислення факторних навантажень**: після досягнення збіжності обчислюється власне значення та відповідні факторні навантаження:

$$\lambda_j = \max_{1 \le i \le n} \left([\mathbf{R_h v_j}]_i \right) \tag{1.4}$$

$$(A_j)_i = \frac{\sqrt{\lambda}_j \, (\mathbf{v_j})_\mathbf{i}}{\|\mathbf{v_j}\|_2} \tag{1.5}$$

де R_h – редукована матриця; j – індекс фактору; i – індекс елементу вектору; $\max\limits_{1 \le i \le n} \left([\mathbf{R_h v_j}]_i \right)$ – максимальний елемент вектору.

Критерій збіжності

Ітераційний процес продовжується до тих пір, поки різниця між послідовними наближеннями власного вектора v не стане меншою за задану точність ε.

Дефляція матриці

Після знаходження першого фактора виконується дефляція матриці для виділення наступних факторів:

$$\mathbf{R}_{j+1} = \mathbf{R}_j - \mathbf{A}_j \mathbf{A}'_j \tag{1.6}$$

де $\mathbf{A}_j$ – вектор факторних навантажень j-го фактора.

1.3.4 Метод Якобі

Метод Якобі[2] є класичним алгоритмом для реалізації методу головних факторів через повну діагоналізацію редукованої матриці. Метод діагоналізації був розроблений математиком Карлом Якобі ще в минулому столітті і залишається ефективним способом одночасного знаходження всіх власних

значень та власних векторів симетричних матриць.

Принцип роботи методу

Процедура починається з пошуку матриці $\mathbf{B}_1$ такої, що:

$$\mathbf{B}_1' \mathbf{R_h} \mathbf{B}_1 = \mathbf{D}_1 \tag{1.7}$$

де $\mathbf{B}_1$ є ортогональною матрицею з ненульовими елементами в клітинках bii, b_{ij}, b_{ji} та b_{jj}, а також в інших діагональних клітинках. За винятком b_{ii} та b_{jj}, діагональні клітинки мають значення 1.0.
Ця матрична операція "анігілює"один позадіагональний елемент в початковій матриці $\mathbf{R_h}$, наближаючи її до остаточної діагональної форми. Для анігіляції обирається найбільший позадіагональний елемент матриці.

Обчислення елементів матриці трансформації

Елементи матриці $\mathbf{B}_i$ визначаються наступним чином:

$$bii = b_{jj} = \cos \phi_{ij} \tag{1.8}$$

$$b_{ij} = \sin \phi_{ij} \tag{1.9}$$

$$b_{ji} = -\sin \phi_{ij} \tag{1.10}$$

де кут ϕ_{ij} знаходиться з рівняння:

$$\tan 2\phi_{ij} = \frac{-2r_{ij}}{h_i^2 - h_j^2} \tag{1.11}$$

де r_{ij} — найбільший позадіагональний елемент редукованої матриці $\mathbf{R_h}$, h_i^2 та h_j^2 — відповідні діагональні елементи матриці R_h.

Ітераційний процес

Процес повторюється знову і знову, завжди з вибором найбільшого позадіагонального значення для анігіляції, незалежно від знаку. Кожен елемент, який анігілюється, вимагає додавання ще однієї матриці $\mathbf{B}_i$. Важливо від-

значити, що матричний елемент не залишається анігільованим назавжди – він повертається до ненульового значення при анігіляції інших елементів. Однак кожного разу, коли анігілюється матричний елемент, загальна сума квадратів позадіагональних елементів зменшується. Таким чином, шляхом ітерацій позадіагональні елементи можна зробити настільки близькими до нуля, наскільки дозволяє точність обчислень. В кінці результат виглядає наступним чином:

$$\mathbf{B}'_k \cdots \mathbf{B}'_3\mathbf{B}'_2\mathbf{B}'_1\mathbf{R_h}\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3 \cdots \mathbf{B}_k = \mathbf{D_k} \tag{1.12}$$

де k зазвичай є числом значно більшим за n.

Критерій збіжності

Після анігіляції кожного позадіагонального елемента проводиться перевірка, чи найбільший позадіагональний елемент що залишився в $\mathbf{D}_i$ більший за деяке обране значення ε, близьке до нуля. Якщо так, то виконується ще одна ітерація. Врешті-решт найбільший позадіагональний елемент матриці $\mathbf{D}_i$ стане меншим за ε, і в цей момент ітерації припиняються.

1.4 Метод мінімальних залишків

1.4.1 Загальний опис

Метод мінімальних залишків (англ. Minimum Residual Method або *MINRES*)[2][4] — це один із підходів до виділення факторів у факторному аналізі. На відміну від інших методів, таких як метод головних факторів чи метод максимальної правдоподібності, цей підхід дозволяє проводити факторизацію без попередньої оцінки спільностей (комунальностей) змінних. Тобто, він базується виключно на позадіагональних елементах матриці кореляцій.

Основна ідея методу полягає у послідовному мінімізації залишків між вихідною кореляційною матрицею та матрицею, що апроксимується факторним розкладом, шляхом поетапного виділення факторів. На кожному

кроці будується факторне навантаження, яке мінімізує суму квадратів позадіагональних залишків.

Цей підхід є особливо зручним у тих випадках, коли оцінити спільності важко або коли є ризик використання хибних оцінок, які можуть спотворити результат. При цьому, хоча спільності явно не задаються, вони можуть бути апроксимовані як суми квадратів навантажень, отриманих у процесі аналізу.

1.4.2 Підготовка даних

Підготовка даних до факторного аналізу методом мінімальних залишків включає кілька стандартних кроків[2][4]:

- **Стандартизація змінних** — вихідні змінні повинні бути приведені до однакової шкали, зазвичай зі середнім 0 та стандартним відхиленням 1.
- **Обчислення кореляційної матриці** R між усіма парами змінних. Це основна вхідна матриця для факторного аналізу.
- **Обнулення діагональних елементів матриці** R (тобто замість 1 на діагоналі записуються 0), що відповідає умові методу не використовувати оцінки комунальностей.

Після цього дані готові для ітераційного виділення факторів.

1.4.3 Опис методу

Процедура обчислення одного фактора методом мінімальних залишків полягає в наступному[2][4]:

1) **Початкова оцінка навантажень.** Створюється початковий вектор факторних навантажень A_0. Як перші наближення можна взяти найбільші за модулем значення в кожному стовпці матриці R, з урахуванням знака.
2) **Множення матриці кореляцій на вектор.** Обчислюється RA_0, тобто добуток матриці R (з нульовими діагоналями) на вектор A_0.
3) **Оновлення навантажень.** Для кожного елемента a_i нового вектора

A_1 розраховується:

$$a_i = \frac{\sum_{j \neq i} r_{ij} a_j}{\sum_{j \neq i} a_j^2} \tag{1.13}$$

де r_{ij} — елементи кореляційної матриці з нульовими діагоналями, a_j — елементи попереднього вектора факторних навантажень.

Цей вираз гарантує, що на кожному кроці ми отримуємо нові значення навантажень, які зменшують залишки.

4) **Середнє згладжування (метод коригування довжини).** Щоб забезпечити стабільну збіжність, застосовується спеціальна процедура згладжування:

$$A_{\text{new}} = A_2 \cdot \sqrt[4]{\frac{L_1^2}{L_2^2}}, \tag{1.14}$$

де L_1, L_2 — довжини векторів A_1, A_2

Це дозволяє уникнути ситуацій, коли ітерація «зациклюється» на векторах однакових за абсолютною величиною, але протилежних за знаком.

5) **Перевірка збіжності.** Якщо всі елементи нового вектора A_{new} близькі до попередніх (різниця менше деякого малого значення ε, наприклад, 0,0005), процес вважається збіжним і поточний вектор приймається як шуканий фактор.

6) **Оновлення залишкової матриці.** Виділений фактор віднімається з початкової матриці кореляцій:

$$R_{\text{new}} = R - AA' \tag{1.15}$$

де A — знайдений вектор факторного навантаження.

7) **Повторення процесу.** Для виділення наступного фактора процедура повторюється починаючи з кроку 1, використовуючи оновлену матрицю R_{new}.

Ітерації продовжуються до моменту, поки:

- або не буде досягнута задана кількість факторів,
- або наступний фактор не почне сходитися на вектори однакової довжини, але протилежних знаків — ознака того, що в матриці більше не залишилось значущої загальної дисперсії.

Важливо підкреслити, що отримані факторні навантаження можна використати для обчислення **оцінених спільностей** як суми квадратів навантажень по кожному рядку, однак сам метод *не вимагає їх попереднього задання*, на відміну від інших класичних підходів.

РОЗДІЛ 2

ПРАКТИЧНА ЧАСТИНА

2.1 Розробка мобільного додатку

2.1.1 Процес розробки

Для реалізації програмного забезпечення, що виконує факторний аналіз, було обрано мову програмування **Dart** у поєднанні з кросплатформовим фреймворком **Flutter**. Такий вибір зумовлений низкою переваг: високою швидкістю розробки, зручним інструментарієм для побудови графічного інтерфейсу користувача, а також можливістю розгортання додатку як на мобільних пристроях (Android, iOS), так і на настільних операційних системах (Windows, Linux, macOS).

Однак, варто зазначити, що екосистема Dart наразі не має повноцінних математичних бібліотек, подібних до `NumPy` в Python чи `Eigen` у C++. У зв'язку з цим, усі необхідні функціональні модулі для обробки числових даних, зокрема:

- операції над матрицями та векторами;
- обчислення кореляційної матриці;
- реалізація власних чисел та власних векторів;
- нормалізація даних;
- розрахунок спільностей;

були реалізовані з нуля. Такий підхід дозволив отримати глибше розуміння алгоритмів факторного аналізу та забезпечив повну контрольованість над точністю чисельних розрахунків.

Графічний інтерфейс користувача був реалізований із використанням засобів фреймворку Flutter. Він забезпечує побудову адаптивних та інтерактивних інтерфейсів завдяки декларативному стилю опису віджетів та підтримці «гарячого перезапуску» (hot reload), що значно пришвидшує процес тестування та налагодження. У застосунку реалізовано:

- введення початкових даних;
- відображення налаштування кожного методу;
- відображення кожного етапу розрахунку;

Таким чином, було створено повнофункціональний інструмент для виконання факторного аналізу, який поєднує точність чисельних методів із сучасним користувацьким інтерфейсом.

2.1.2 Демонстрація додатку

При вході в додаток нас зустрічає головна сторінка(рис. 2.1), де ми можемо внести дані для розрахунку і після цього перейти до використання методів факторного аналізу.

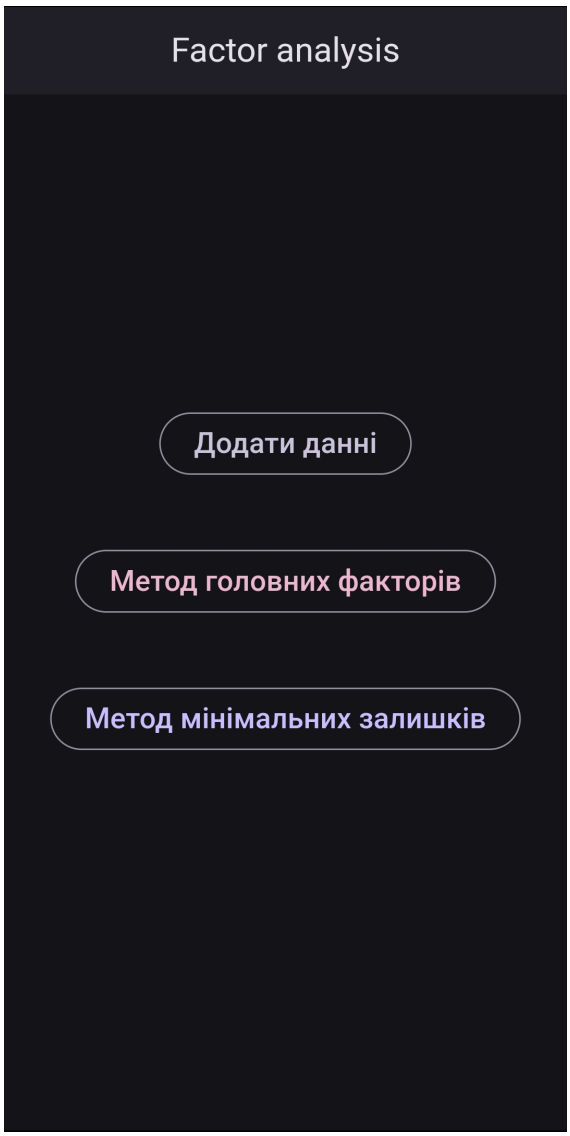


Рис. 2.1. Головна сторінка додатку

Переходимо до додавання даних. Нас зустрічає сторінка(рис. 2.2), де ми можемо внести кількість об’єктів, які ми плануємо аналізувати, а також кількість показників за якими ми будемо це робити(рис. 2.3). ”Назва об’єктів” відповідає за те, щоб замість, наприклад, ”Об’єкт №1” мати нумерацію з назвою, яку ми вказали. Але за необхідністю, користувач може змінити назву кожного об’єкту окремо, як і назву показника.

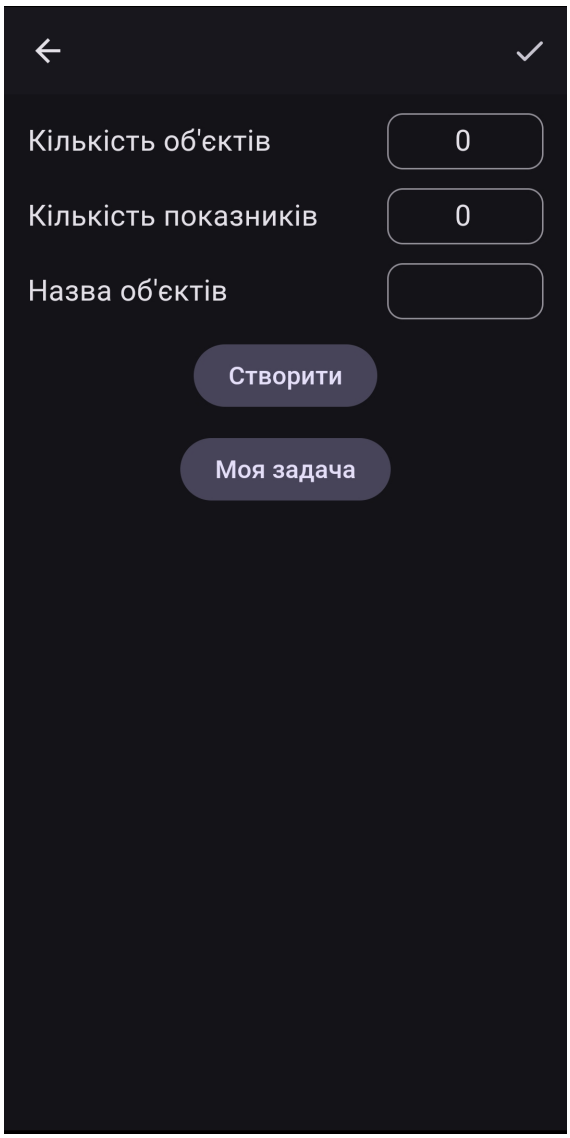


Рис. 2.2. Сторінка додавання даних

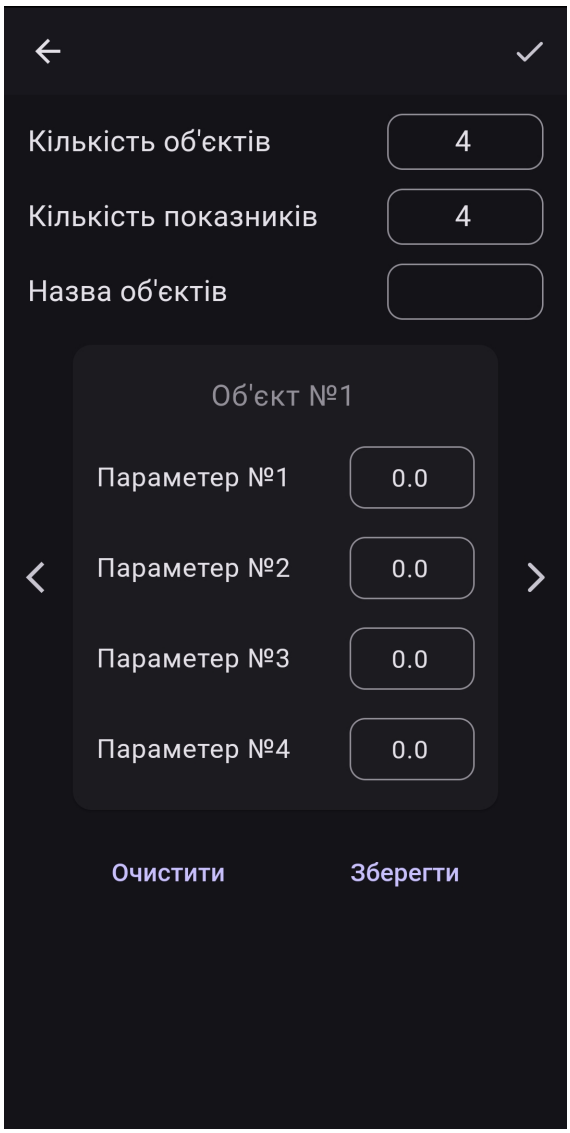


Рис. 2.3. Приклад додавання даних

Для розгляду задачі цієї роботи є кнопка ”Моя задача”, при натискані на яку, дані автоматично заповняться(рис. 2.4).

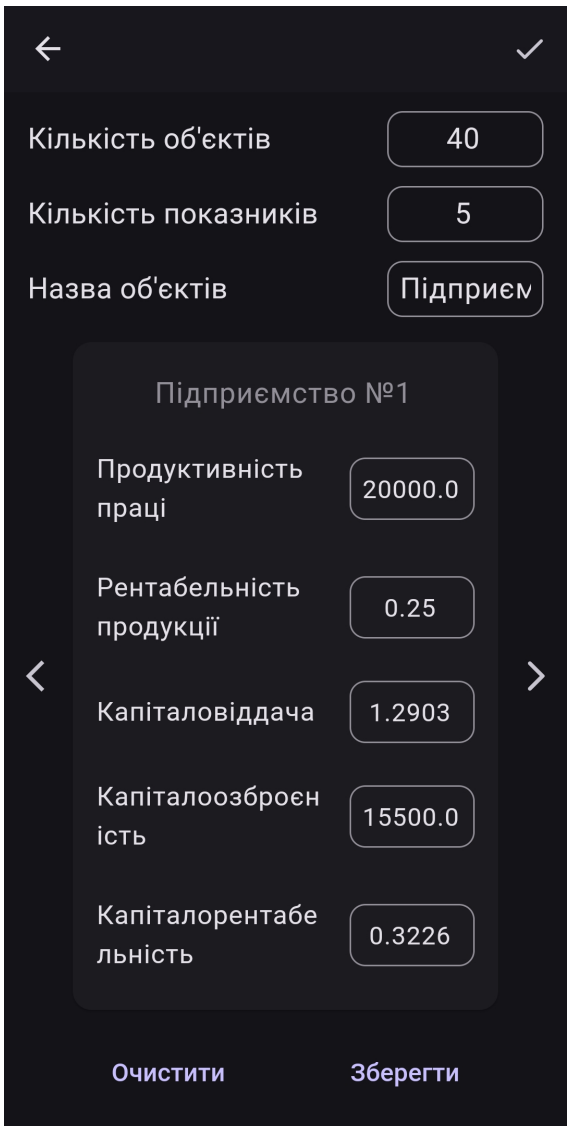


Рис. 2.4. Дані розглянутої задачі

Після того як дані заповнені, ми можемо перейти до налаштування методів.

Метод головних факторів має налаштування(рис. 2.5) точності розрахунків та переходу до наступного методу(використовується для методу Хотеллінгу), а також можна вибрати метод для розрахунків між методом Хотеллінгу та методом Якобі.

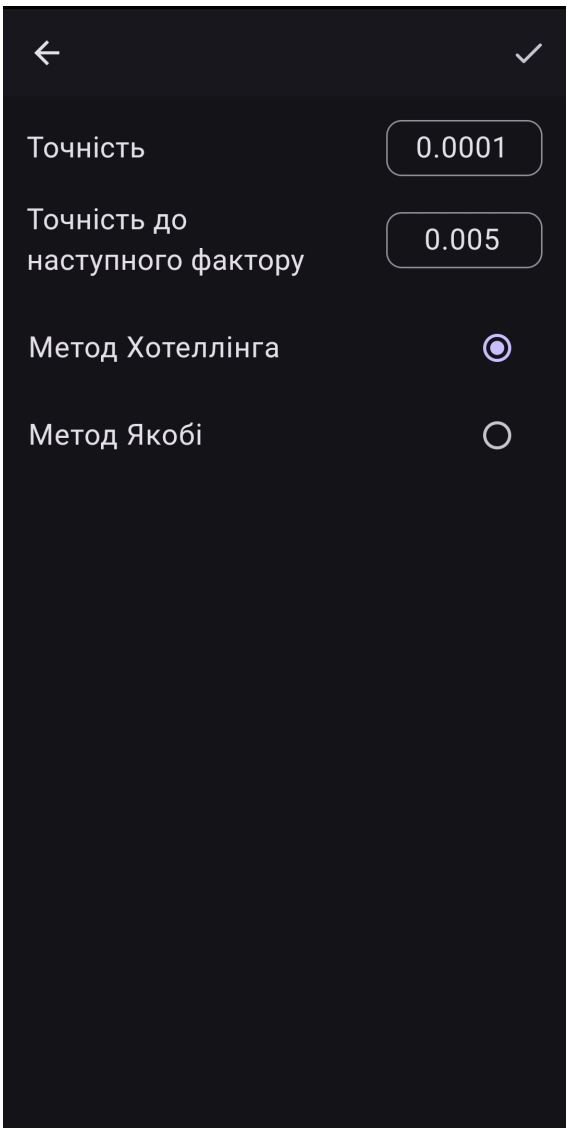


Рис. 2.5. Налаштування методу головних факторів

У метода мінімальних залишків інша сторінка налаштування(рис. 2.6). У нього можна вказати кількість факторів, які потрібно виділити, точність розрахунку, та кількість ітерацій.

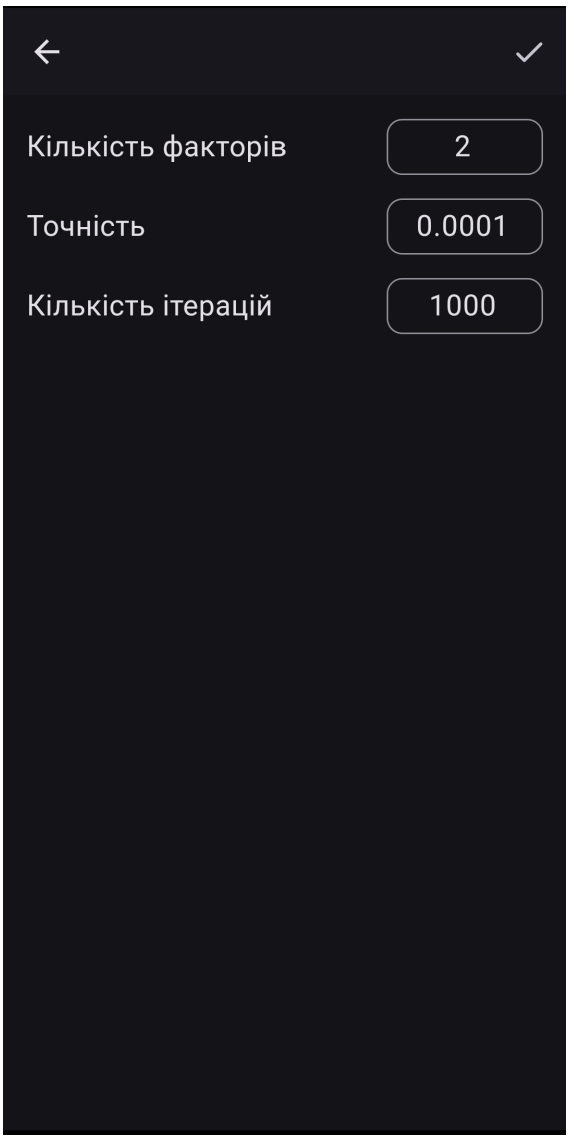


Рис. 2.6. Налаштування методу мінімальних залишків

Після налаштування ми можемо перейти на сторінку необхідного нам методу:

- Метод головних факторів з методом Хотеллінгу(рис. 2.7);
- Метод головних факторів з методом Якобі(рис. 2.8);
- Метод мінімальних залишків(рис. 2.9);

Етапи кожного методу будуть роглянуті далі.

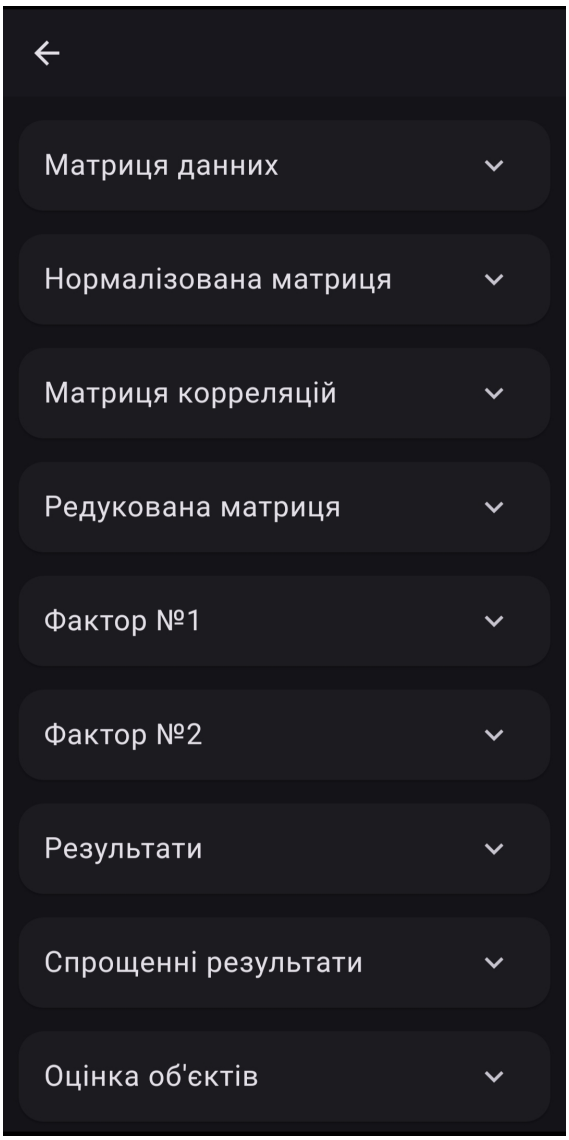


Рис. 2.7. Метод головних факторів з методом Хотеллінгу

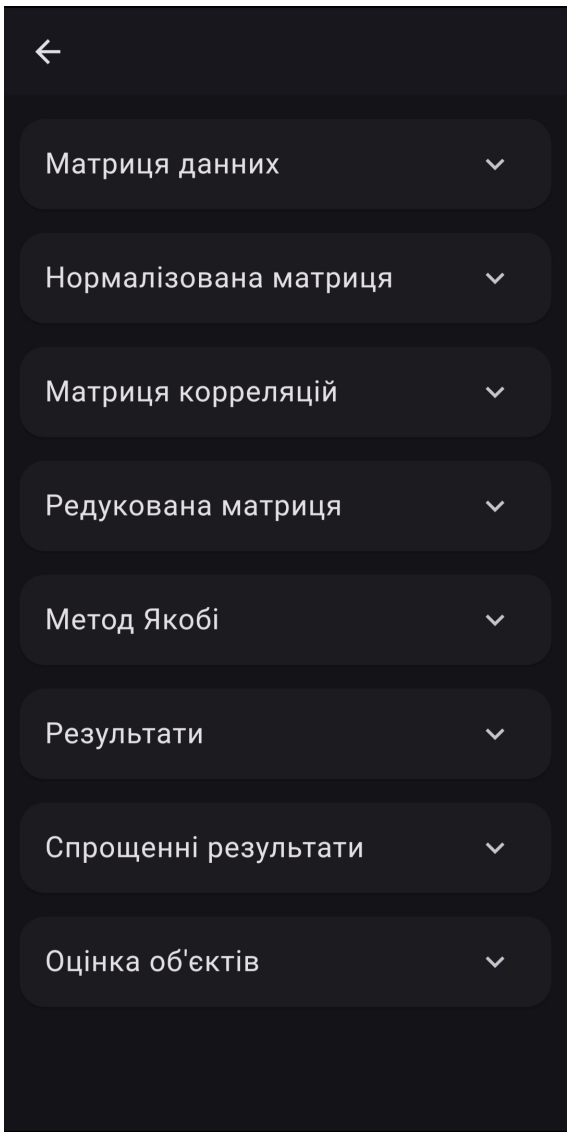


Рис. 2.8. Метод головних факторів з методом Якобі

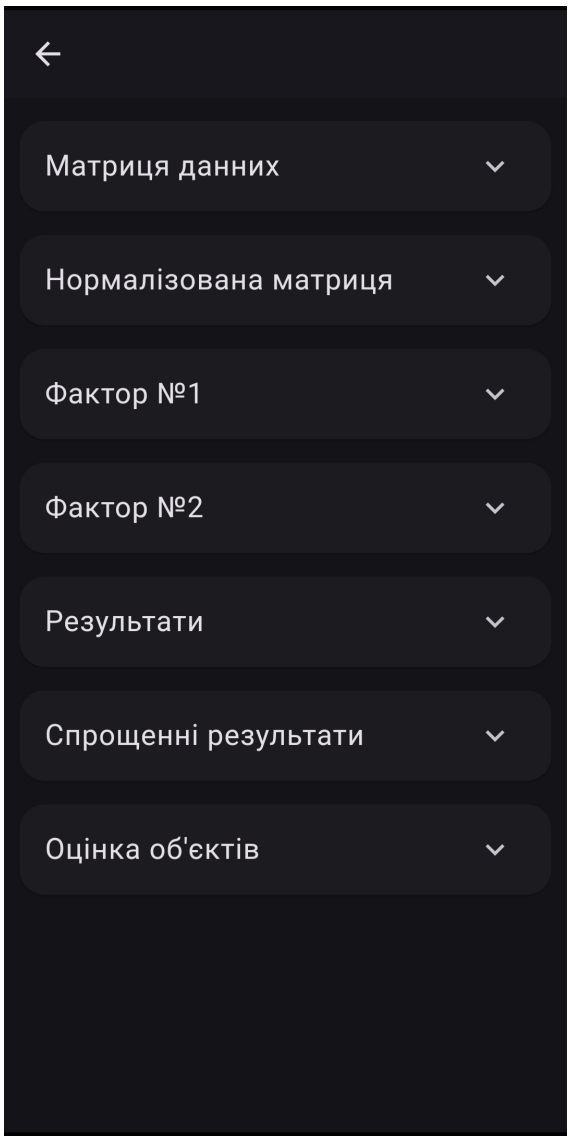


Рис. 2.9. Метод мінімальних залишків

2.2 Дослідження

2.2.1 Дані для задачі

Початкова матриця

Зводимо отримані дані до матриці 40 на 5(рис. 2.10):

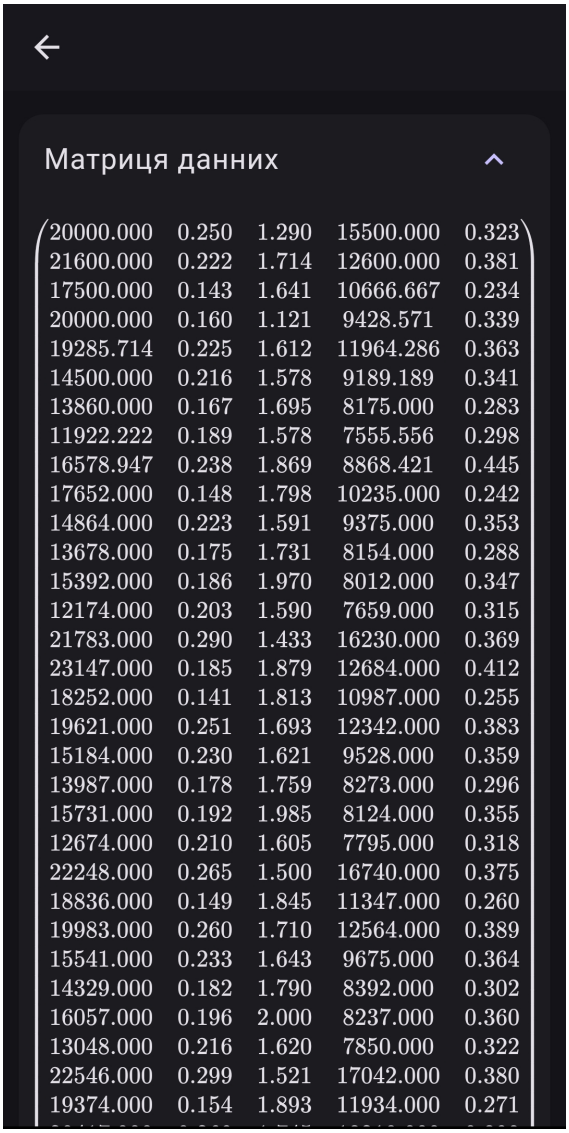


Рис. 2.10. Матриця даних у додатку

$$\begin{pmatrix}
20000.000 & 0.250 & 1.290 & 15500.000 & 0.323 \\
21600.000 & 0.222 & 1.714 & 12600.000 & 0.381 \\
17500.000 & 0.143 & 1.641 & 10666.667 & 0.234 \\
20000.000 & 0.160 & 1.121 & 9428.571 & 0.339 \\
19285.714 & 0.225 & 1.612 & 11964.286 & 0.363 \\
14500.000 & 0.216 & 1.578 & 9189.189 & 0.341 \\
13860.000 & 0.167 & 1.695 & 8175.000 & 0.283 \\
11922.222 & 0.189 & 1.578 & 7555.556 & 0.298 \\
16578.947 & 0.238 & 1.869 & 8868.421 & 0.445 \\
17652.000 & 0.148 & 1.798 & 10235.000 & 0.242 \\
14864.000 & 0.223 & 1.591 & 9375.000 & 0.353 \\
13678.000 & 0.175 & 1.731 & 8154.000 & 0.288 \\
15392.000 & 0.186 & 1.970 & 8012.000 & 0.347 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
14076.000 & 0.231 & 1.670 & 8554.000 & 0.336
\end{pmatrix} \tag{2.1}$$

Переглянути дані повністю можна у таблиці(табл. 2.15).

Нормалізація даних

Оскільки вихідні економічні показники мають різну природу та масштаби вимірювання, перед початком факторного аналізу виконується їх нормалізація(рис. 2.11). З цією метою використовується **Z-score нормалізація** (стандартизація), яка дозволяє привести кожну змінну до єдиної шкали зі середнім значенням 0 та стандартним відхиленням 1.

$$z_{ij} = \frac{x_{ij} - \bar{x}_j}{s_j} \tag{2.2}$$

де:

- z_{ij} — нормалізоване значення.
- x_{ij} — значення j-тої ознаки для i-го об'єкта;
- $\bar{x}_j$ — середнє значення j-тої ознаки;
- s_j — стандартне відхилення j-тої ознаки;

Такий підхід дозволяє забезпечити рівний внесок кожної змінної у подальший аналіз незалежно від її початкового масштабу. Це особливо важливо в контексті факторного аналізу, оскільки всі змінні повинні бути зіставними для коректного обчислення кореляційної матриці та подальшого виділення латентних факторів.

Рис. 2.11. Нормалізовані дані у додатку

$$\begin{pmatrix} 0.862 & 0.860 & -2.273 & 1.696 & -0.332 \\ 1.349 & 0.207 & 0.148 & 0.676 & 0.857 \\ 0.102 & -1.657 & -0.273 & -0.005 & -2.128 \\ 0.862 & -1.255 & -3.239 & -0.440 & 0.010 \\ 0.645 & 0.273 & -0.436 & 0.452 & 0.485 \\ -0.811 & 0.066 & -0.631 & -0.524 & 0.047 \\ -1.006 & -1.097 & 0.040 & -0.881 & -1.146 \\ -1.596 & -0.569 & -0.631 & -1.099 & -0.823 \\ -0.179 & 0.581 & 1.034 & -0.637 & 2.162 \\ 0.148 & -1.546 & 0.627 & -0.156 & -1.979 \\ -0.701 & 0.233 & -0.555 & -0.459 & 0.283 \\ -1.062 & -0.893 & 0.244 & -0.889 & -1.026 \\ -0.540 & -0.653 & 1.610 & -0.939 & 0.173 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -0.940 & 0.409 & -0.106 & -0.748 & -0.061 \end{pmatrix} \tag{2.3}$$

Переглянути дані повністю можна у таблиці(табл. 2.16).

Матриця кореляцій

Розраховуємо кореляційну матрицю так, як було описано у теоритичній частині(рис. 2.12).

Рис. 2.12. Кореляційна матриця

$$\begin{pmatrix} 1.000 & 0.396 & -0.178 & 0.900 & 0.422 \\ 0.396 & 1.000 & -0.329 & 0.598 & 0.778 \\ -0.178 & -0.329 & 1.000 & -0.339 & -0.058 \\ 0.900 & 0.598 & -0.339 & 1.000 & 0.373 \\ 0.422 & 0.778 & -0.058 & 0.373 & 1.000 \end{pmatrix} \tag{2.4}$$

2.2.2 Метод головних факторів

Редукована матриця

Як було сказано раніше, метод головних факторів потребує оцінку спільностей. Їх ми знаходимо методом квадрату множинних кореляцій. Після цього ми замінюємо одиниці у матриці кореляцій на спільності(рис. 2.13).

Рис. 2.13. Редукована матриця

$$\begin{pmatrix} 0.955 & 0.396 & -0.178 & 0.900 & 0.422 \\ 0.396 & 0.932 & -0.329 & 0.598 & 0.778 \\ -0.178 & -0.329 & 0.223 & -0.339 & -0.058 \\ 0.900 & 0.598 & -0.339 & 0.966 & 0.373 \\ 0.422 & 0.778 & -0.058 & 0.373 & 0.895 \end{pmatrix} \tag{2.5}$$

Метод Хотеллінга

Переходимо до ітерацій методу Хотеллінгу(код 2.1). Спочатку створимо таблицю з початковим наближенням власного вектору(рис. 2.14)(табл. 2.1).

Вихідна матриця

Ознака	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{X_5}$	$\mathbf{S_i^{(1)} = \sum_j r_{ij}}$	$\mathbf{a^{(1)} = \frac{S_i^{(1)}}{S_{max}}}$
X_1	0.955	0.396	-0.178	0.900	0.422	2.495	0.999
X_2	0.396	0.932	-0.329	0.598	0.778	2.376	0.951
X_3	-0.178	-0.329	0.223	-0.339	-0.058	-0.681	-0.273
X_4	0.900	0.598	-0.339	0.966	0.373	2.497	1.000
X_5	0.422	0.778	-0.058	0.373	0.895	2.410	0.965

Рис. 2.14. Початкове наближення

Ознака	X_1	X_2	X_3	X_4	X_5	$S_i^{(1)}$	$a^{(1)}$
X_1	0.955	0.396	-0.178	0.900	0.422	2.495	0.999
X_2	0.396	0.932	-0.329	0.598	0.778	2.376	0.951
X_3	-0.178	-0.329	0.223	-0.339	-0.058	-0.681	-0.273
X_4	0.900	0.598	-0.339	0.966	0.373	2.497	1.000
X_5	0.422	0.778	-0.058	0.373	0.895	2.410	0.965

Табл. 2.1. Початкове наближення

Далі йде ітераційний процес(рис. 2.15)(табл. 2.2), поки наближення власного вектору не стане менше заданої точності. Для першого вектору було необхідно 4 ітерації(рис. 2.16)(табл. 2.3).

Цикл №1

Ознака	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{X_5}$	$\mathbf{S_i^{(2)} = \sum_j r_{ij}}$	$\mathbf{p_i^{(2)} = R_h^1 S^{(1)}}$	$\mathbf{a^{(2)} = \frac{p_i}{p_{max}}}$	c
X_1	2.089	1.673	-0.670	2.184	1.434	6.710	6.710	0.931	
X_2	1.673	2.097	-0.698	1.893	1.831	6.795	6.795	0.943	
X_3	-0.670	-0.698	0.308	-0.782	-0.522	-2.364	-2.364	-0.328	
X_4	2.184	1.893	-0.782	2.354	1.558	7.207	7.207	1.000	
X_5	1.434	1.831	-0.522	1.558	1.727	6.029	6.029	0.836	

Рис. 2.15. Перший цикл

Ознака	X_1	X_2	X_3	X_4	X_5	$S_i^{(2)}$	$p_i^{(2)}$	$a^{(2)}$	$d = \lvert a^{(2)} - a^{(1)} \rvert$
X_1	2.089	1.673	-0.670	2.184	1.434	6.710	6.710	0.931	0.068
X_2	1.673	2.097	-0.698	1.893	1.831	6.795	6.795	0.943	0.008
X_3	-0.670	-0.698	0.308	-0.782	-0.522	-2.364	-2.364	-0.328	0.055
X_4	2.184	1.893	-0.782	2.354	1.558	7.207	7.207	1.000	0.000
X_5	1.434	1.831	-0.522	1.558	1.727	6.029	6.029	0.836	0.129

Табл. 2.2. Перший цикл

Цикл №4

Ознака	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{X_5}$	$\mathbf{S_i^{(5)} = \sum_j r_{ij}}$	$\mathbf{p_i^{(5)} = R_h^8 S^{(4)}}$	$\mathbf{a^{(5)} = \frac{p_i}{p_{max}}}$	c
X_1	2837379.709	2856476.122	-1035811.940	3078825.660	2475992.455	10212862.006	10212862.006	0.922	
X_2	2856476.122	2875701.199	-1042783.280	3099547.091	2492656.770	10281597.903	10281597.903	0.928	
X_3	-1035811.940	-1042783.280	378132.816	-1123954.039	-903884.168	-3728300.610	-3728300.610	-0.336	
X_4	3078825.660	3099547.091	-1123954.039	3340817.381	2686686.285	11081922.378	11081922.378	1.000	
X_5	2475992.455	2492656.770	-903884.168	2686686.285	2160634.000	8912085.342	8912085.342	0.804	

Рис. 2.16. Четвертий цикл

Ознака	X_1	X_2	X_3	X_4	X_5	$S_i^{(5)}$	$p_i^{(5)}$	$a^{(5)}$	$d = \lvert a^{(5)} - a^{(4)} \rvert$
X_1	2837379.709	2856476.122	-1035811.940	3078825.660	2475992.455	10212862.006	10212862.006	0.922	0.000
X_2	2856476.122	2875701.199	-1042783.280	3099547.091	2492656.770	10281597.903	10281597.903	0.928	0.000
X_3	-1035811.940	-1042783.280	378132.816	-1123954.039	-903884.168	-3728300.610	-3728300.610	-0.336	0.000
X_4	3078825.660	3099547.091	-1123954.039	3340817.381	2686686.285	11081922.378	11081922.378	1.000	0.000
X_5	2475992.455	2492656.770	-903884.168	2686686.285	2160634.000	8912085.342	8912085.342	0.804	0.000

Табл. 2.3. Четвертий цикл

Після цього ми отримуємо таблицю навантажень головного фактору(рис. 2.17)(табл. 2.4).

Навантаження головного фактора

Ознака	$\mathbf{U_1}$	$\beta_1 = \mathbf{R_h U_1}$	$\mathbf{A} = \frac{\mathbf{U_1}\sqrt{\lambda_1}}{\sqrt{\sum \mathbf{U_1^2}}}$
X_1	0.922	2.547	0.822
X_2	0.928	2.564	0.828
X_3	-0.336	-0.930	-0.300
X_4	1.000	2.764	0.892
X_5	0.804	2.223	0.718

Рис. 2.17. Навантаження головного фактору

Ознака	U_1	$\beta_1 = R_h U_1$	$A = \frac{U_1\sqrt{\lambda_1}}{\sqrt{\sum U_1^2}}$
X_1	0.922	2.547	0.822
X_2	0.928	2.564	0.828
X_3	-0.336	-0.930	-0.300
X_4	1.000	2.764	0.892
X_5	0.804	2.223	0.718

Табл. 2.4. Навантаження головного фактору

Ми розраховуємо матрицю залишкових коефіцієнтів(рис. 2.18) і бачимо, що вона має поки що великі значення, це означає, що потрібно виділити ще один фактор.

Матриця залишкових коеффіцієнтів

$$\begin{pmatrix} 0.279 & -0.285 & 0.069 & 0.166 & -0.169 \\ -0.285 & 0.246 & -0.080 & -0.141 & 0.184 \\ 0.069 & -0.080 & 0.133 & -0.071 & 0.158 \\ 0.166 & -0.141 & -0.071 & 0.169 & -0.268 \\ -0.169 & 0.184 & 0.158 & -0.268 & 0.380 \end{pmatrix}$$

Рис. 2.18. Матриця залишкових коефіцієнтів

$$\begin{pmatrix} 0.279 & -0.285 & 0.069 & 0.166 & -0.169 \\ -0.285 & 0.246 & -0.080 & -0.141 & 0.184 \\ 0.069 & -0.080 & 0.133 & -0.071 & 0.158 \\ 0.166 & -0.141 & -0.071 & 0.169 & -0.268 \\ -0.169 & 0.184 & 0.158 & -0.268 & 0.380 \end{pmatrix} \tag{2.6}$$

Повторюємо процес ітерацій, але тепер для початкового наближення вико-

ристовуємо матрицю залишкових коефіцієнтів(табл. 2.5)(табл. 2.6).

Ознака	X_1	X_2	X_3	X_4	X_5	$S_i^{(1)}$	$a^{(1)}$
X_1	0.279	-0.285	0.069	0.166	-0.169	0.060	0.211
X_2	-0.285	0.246	-0.080	-0.141	0.184	-0.076	-0.265
X_3	0.069	-0.080	0.133	-0.071	0.158	0.208	0.728
X_4	0.166	-0.141	-0.071	0.169	-0.268	-0.145	-0.507
X_5	-0.169	0.184	0.158	-0.268	0.380	0.286	1.000

Табл. 2.5. Початкове наближення

Ознака	X_1	X_2	X_3	X_4	X_5	$S_i^{(2)}$	$p_i^{(2)}$	$a^{(2)}$	$d = \lvert a^{(2)} - a^{(1)} \rvert$
X_1	0.219	-0.209	0.013	0.155	-0.197	-0.020	-0.020	-0.126	0.336
X_2	-0.209	0.202	-0.011	-0.149	0.188	0.021	0.021	0.131	0.397
X_3	0.013	-0.011	0.059	-0.041	0.074	0.093	0.093	0.597	0.131
X_4	0.155	-0.149	-0.041	0.153	-0.212	-0.095	-0.095	-0.609	0.102
X_5	-0.197	0.188	0.074	-0.212	0.304	0.156	0.156	1.000	0.000

Табл. 2.6. Перший цикл

Задану точність було досягнуто на п'ятому кроці(табл. 2.7).

Ознака	X_1	X_2	X_3	X_4	X_5	$S_i^{(6)}$	$p_i^{(6)}$	$a^{(6)}$	$d = \lvert a^{(6)} - a^{(5)} \rvert$
X_1	0.005	-0.005	-0.001	0.005	-0.006	-0.002	-0.002	-0.844	0.000
X_2	-0.005	0.005	0.001	-0.004	0.006	0.002	0.002	0.809	0.000
X_3	-0.001	0.001	0.000	-0.001	0.001	0.000	0.000	0.116	0.000
X_4	0.005	-0.004	-0.001	0.004	-0.006	-0.002	-0.002	-0.738	0.000
X_5	-0.006	0.006	0.001	-0.006	0.008	0.003	0.003	1.000	0.000

Табл. 2.7. П'ятий цикл

Отримали такі навантаження для другого фактору(табл. 2.8).

Ознака	U_1	$\beta_1 = R_h U_1$	$A = \frac{U_1\sqrt{\lambda_1}}{\sqrt{\sum U_1^2}}$
X_1	-0.844	-0.749	-0.465
X_2	0.809	0.718	0.446
X_3	0.116	0.103	0.064
X_4	-0.738	-0.655	-0.407
X_5	1.000	0.888	0.551

Табл. 2.8. Навантаження головного фактору

Можемо побачити, що матриця залишкових коефіцієнтів має малі значення, тому третій фактор виражати не потрібно

$$\begin{pmatrix} 0.063 & -0.077 & 0.098 & -0.023 & 0.087 \\ -0.077 & 0.048 & -0.109 & 0.040 & -0.062 \\ 0.098 & -0.109 & 0.129 & -0.045 & 0.123 \\ -0.023 & 0.040 & -0.045 & 0.004 & -0.044 \\ 0.087 & -0.062 & 0.123 & -0.044 & 0.077 \end{pmatrix} \tag{2.7}$$

Фінальні результати розглянемо далі.

Метод Якобі

Для методу Якобі також використовується редукована матриця. Як було сказано раніше, нам потрібна ортогональна(рис. 2.19) та діагональна матриці(рис. 2.20), які ми отримуємо після ітерацій до заданої точності(код 2.2).

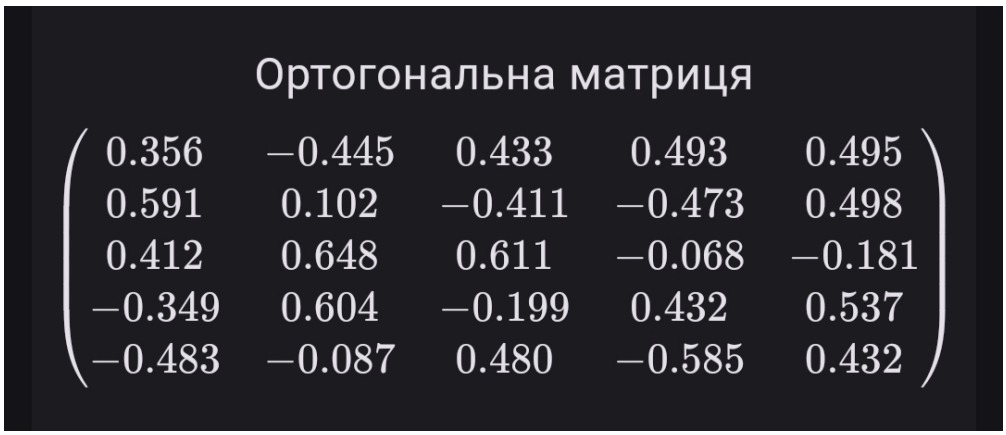


Рис. 2.19. Ортогональна матриця

$$\begin{pmatrix} 0.356 & -0.445 & 0.433 & 0.493 & 0.495 \\ 0.591 & 0.102 & -0.411 & -0.473 & 0.498 \\ 0.412 & 0.648 & 0.611 & -0.068 & -0.181 \\ -0.349 & 0.604 & -0.199 & 0.432 & 0.537 \\ -0.483 & -0.087 & 0.480 & -0.585 & 0.432 \end{pmatrix} \tag{2.8}$$

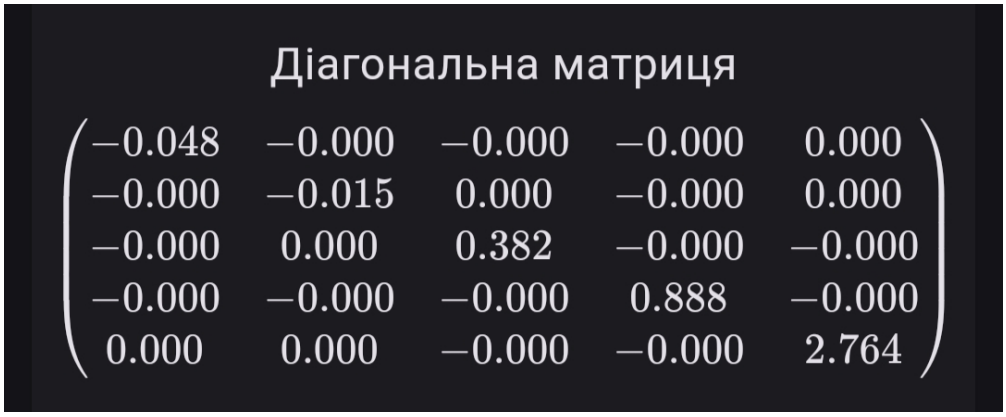


Рис. 2.20. Діагональна матриця

$$\begin{pmatrix} -0.048 & -0.000 & -0.000 & -0.000 & 0.000 \\ -0.000 & -0.015 & 0.000 & -0.000 & 0.000 \\ -0.000 & 0.000 & 0.382 & -0.000 & -0.000 \\ -0.000 & -0.000 & -0.000 & 0.888 & -0.000 \\ 0.000 & 0.000 & -0.000 & -0.000 & 2.764 \end{pmatrix} \tag{2.9}$$

Було отримано таку матрицю факторів(рис. 2.21). Остаточний результат та його аналіз буде описаний далі.

Матриця факторів

$$\begin{pmatrix} 0.000 & -0.000 & 0.268 & 0.465 & 0.822 \\ 0.000 & 0.000 & -0.254 & -0.446 & 0.828 \\ 0.000 & 0.000 & 0.378 & -0.064 & -0.300 \\ -0.000 & 0.000 & -0.123 & 0.407 & 0.892 \\ -0.000 & -0.000 & 0.297 & -0.551 & 0.718 \end{pmatrix}$$

Рис. 2.21. Матриця факторів

$$\begin{pmatrix} 0.000 & -0.000 & 0.268 & 0.465 & 0.822 \\ 0.000 & 0.000 & -0.254 & -0.446 & 0.828 \\ 0.000 & 0.000 & 0.378 & -0.064 & -0.300 \\ -0.000 & 0.000 & -0.123 & 0.407 & 0.892 \\ -0.000 & -0.000 & 0.297 & -0.551 & 0.718 \end{pmatrix} \tag{2.10}$$

2.2.3 Метод мінімальних залишків

Для методу мінімальних залишків(код 2.3) нам не потрібна редукована матриця, замість того ми замінюємо одиниці у матриці кореляцій на

нулі(рис. 2.22), та працюємо здебільшого з недіагональними елементами.

Рис. 2.22. Матриця кореляцій з нулями

$$\begin{pmatrix} 0.000 & 0.396 & -0.178 & 0.900 & 0.422 \\ 0.396 & 0.000 & -0.329 & 0.598 & 0.778 \\ -0.178 & -0.329 & 0.000 & -0.339 & -0.058 \\ 0.900 & 0.598 & -0.339 & 0.000 & 0.373 \\ 0.422 & 0.778 & -0.058 & 0.373 & 0.000 \end{pmatrix} \tag{2.11}$$

Після цього ми знаходимо вектори навантажень(рис. 2.23) доки не дійдемо до заданої точності або пройдемо задану кількість ітерацій.

$$A_0 = \begin{pmatrix} 0.900 \\ 0.778 \\ -0.339 \\ 0.900 \\ 0.778 \end{pmatrix}$$

$$A_1 = \begin{pmatrix} 0.766 \\ 0.759 \\ -0.309 \\ 0.905 \\ 0.611 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 0.766 \\ 0.758 \\ -0.309 \\ 0.905 \\ 0.611 \end{pmatrix}$$

Рис. 2.23. Вектори навантажень

$$A_0 = \begin{pmatrix} 0.900 \\ 0.778 \\ -0.339 \\ 0.900 \\ 0.778 \end{pmatrix} \tag{2.12}$$

$$A_1 = \begin{pmatrix} 0.766 \\ 0.759 \\ -0.309 \\ 0.905 \\ 0.611 \end{pmatrix} \tag{2.13}$$

Саме цей вектор є фінальними навантаженнями першого фактору.

$$A_2 = \begin{pmatrix} 0.766 \\ 0.758 \\ -0.309 \\ 0.905 \\ 0.611 \end{pmatrix} \tag{2.14}$$

Далі ми розраховуємо матрицю залишків(рис. 2.24)(рис. 2.25) та вектори навантажень(рис. 2.26) . Повторюємо процес, доки не досягнемо задану кількість факторів або вектори не почнуть розходитись.

Матриця корреляцій

$$\begin{pmatrix} 0.413 & -0.185 & 0.059 & 0.207 & -0.047 \\ -0.185 & 0.425 & -0.094 & -0.088 & 0.315 \\ 0.059 & -0.094 & 0.904 & -0.059 & 0.131 \\ 0.207 & -0.088 & -0.059 & 0.182 & -0.180 \\ -0.047 & 0.315 & 0.131 & -0.180 & 0.627 \end{pmatrix}$$

Рис. 2.24. Матриця залишків

$$\begin{pmatrix} 0.413 & -0.185 & 0.059 & 0.207 & -0.047 \\ -0.185 & 0.425 & -0.094 & -0.088 & 0.315 \\ 0.059 & -0.094 & 0.904 & -0.059 & 0.131 \\ 0.207 & -0.088 & -0.059 & 0.182 & -0.180 \\ -0.047 & 0.315 & 0.131 & -0.180 & 0.627 \end{pmatrix} \tag{2.15}$$

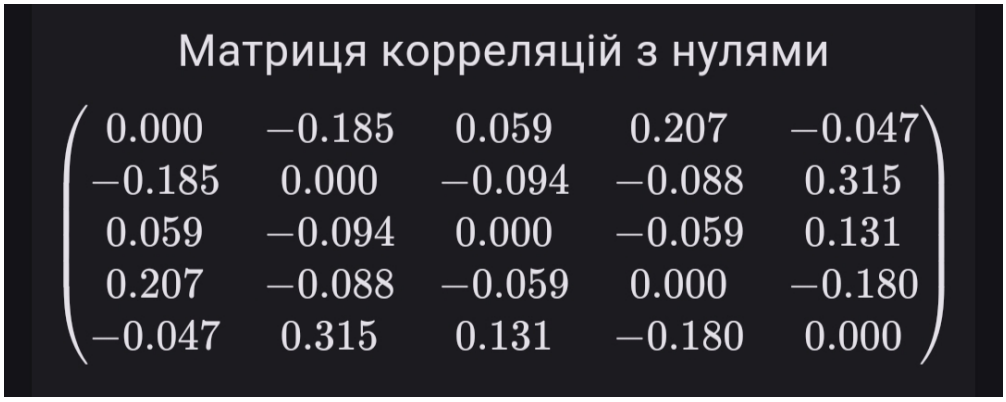


Рис. 2.25. Матриця залишків з нулями

$$\begin{pmatrix} 0.000 & -0.185 & 0.059 & 0.207 & -0.047 \\ -0.185 & 0.000 & -0.094 & -0.088 & 0.315 \\ 0.059 & -0.094 & 0.000 & -0.059 & 0.131 \\ 0.207 & -0.088 & -0.059 & 0.000 & -0.180 \\ -0.047 & 0.315 & 0.131 & -0.180 & 0.000 \end{pmatrix} \tag{2.16}$$

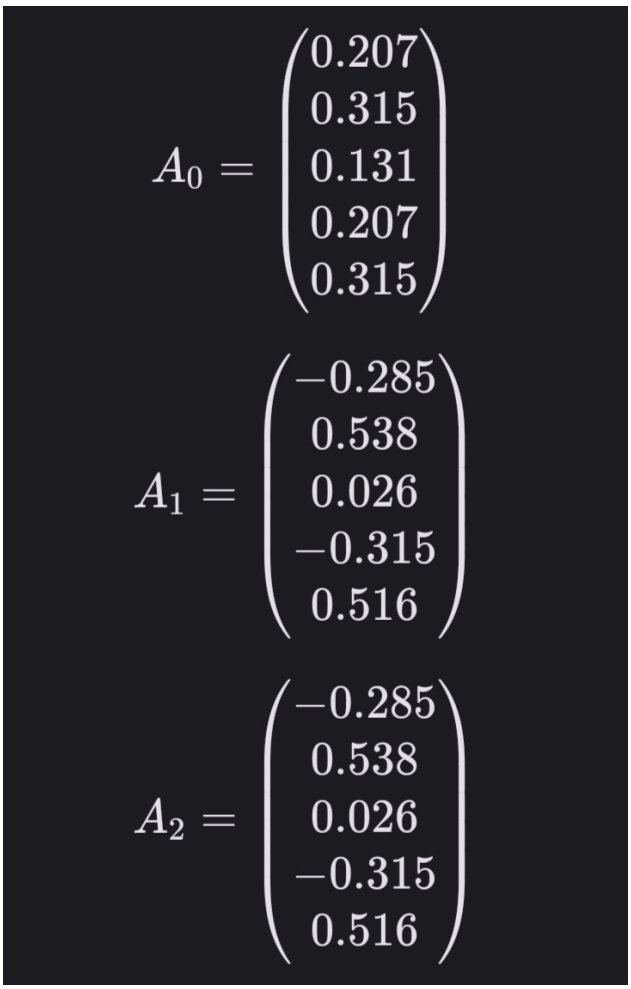


Рис. 2.26. Вектори навантажень

$$A_0 = \begin{pmatrix} 0.207 \\ 0.315 \\ 0.131 \\ 0.207 \\ 0.315 \end{pmatrix} \tag{2.17}$$

$$A_1 = \begin{pmatrix} -0.285 \\ 0.538 \\ 0.026 \\ -0.315 \\ 0.516 \end{pmatrix} \tag{2.18}$$

$$A_2 = \begin{pmatrix} -0.285 \\ 0.538 \\ 0.026 \\ -0.315 \\ 0.516 \end{pmatrix} \tag{2.19}$$

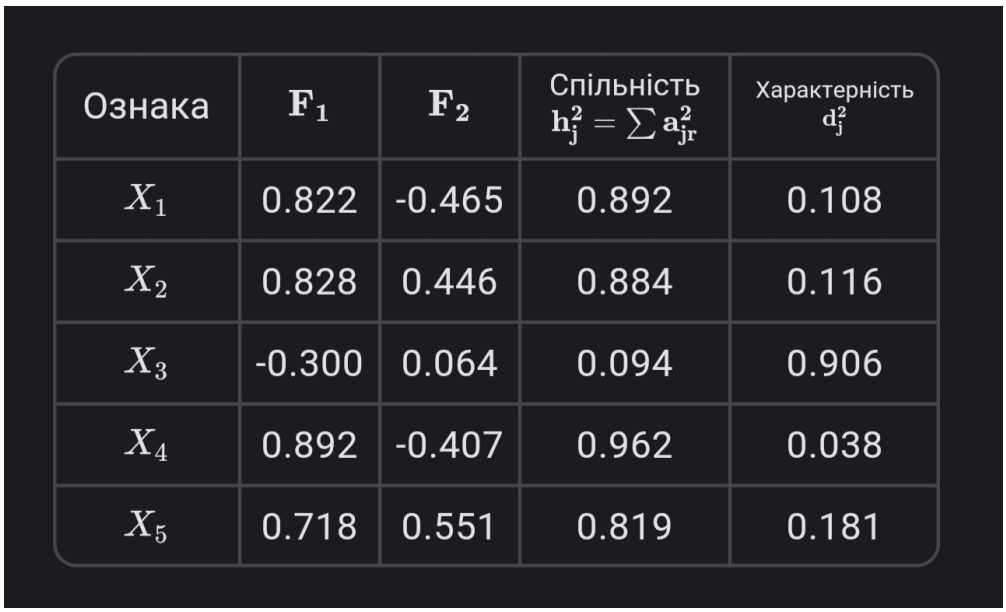
Подальший аналіз цього та інших методів проведено у наступній главі.

2.3 Аналіз результатів

2.3.1 Метод головних факторів

Метод Хотеллінга

З таблиць навантажень з методу Хотеллінгу ми отримали показники для двох факторів(рис. 2.27)(табл. 2.9).



Ознака	$\mathbf{F_1}$	$\mathbf{F_2}$	Спільність $\mathbf{h_j^2 = \sum a_{jr}^2}$	Характерність $\mathbf{d_j^2}$
X_1	0.822	-0.465	0.892	0.108
X_2	0.828	0.446	0.884	0.116
X_3	-0.300	0.064	0.094	0.906
X_4	0.892	-0.407	0.962	0.038
X_5	0.718	0.551	0.819	0.181

Рис. 2.27. Результати для методу головних факторів з методом Хотеллінгу

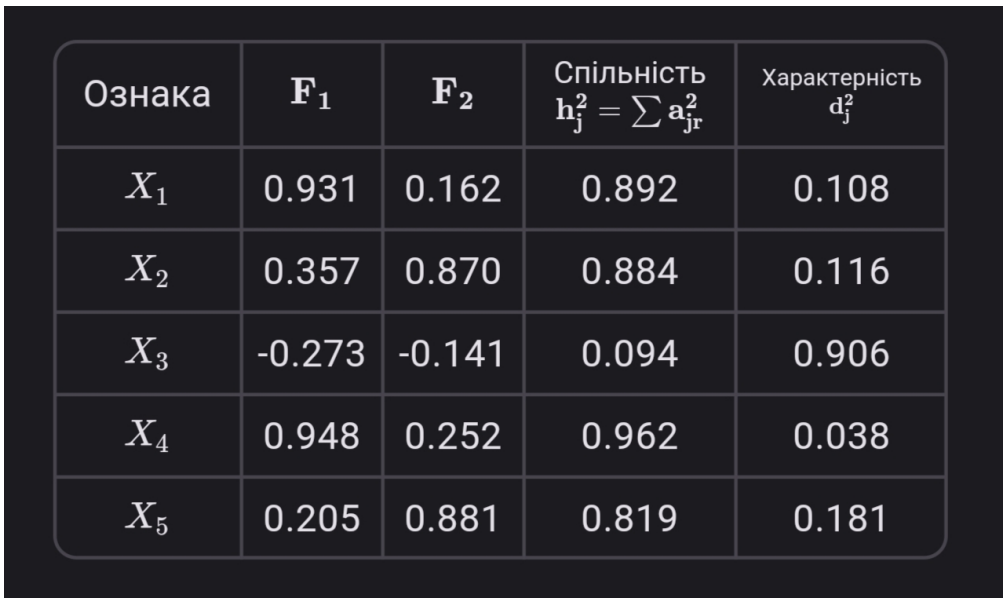
Ознака	F_1	F_2	Спільність $h_j^2 = \sum a_{jr}^2$	Характерність d_j^2
X_1	0.822	-0.465	0.892	0.108
X_2	0.828	0.446	0.884	0.116
X_3	-0.300	0.064	0.094	0.906
X_4	0.892	-0.407	0.962	0.038
X_5	0.718	0.551	0.819	0.181

Табл. 2.9. Результати для методу головних факторів з методом Хотеллінгу

Але такі результати не зручно інтерпретувати, оскільки факторні навантаження в початковій (неротованій) матриці мають складну структуру та не дозволяють чітко виділити змінні, що належать до окремих факторів. Тому для покращення інтерпретованості було використано метод ортогонального обертання векторів — **варімакс**(рис. 2.28)(табл. 2.10).

Метод варімакс[3][4] полягає в максимізації дисперсії квадратів факторних навантажень у межах кожного фактора. Іншими словами, він прагне зробити так, щоб кожна змінна мала якомога вище навантаження на один з

факторів і близьке до нуля — на решту. Це забезпечує чіткішу структуру факторів, коли кожен з них асоціюється з конкретною підгрупою змінних, що значно полегшує їх інтерпретацію. Варімакс є одним із найпоширеніших методів ортогонального обертання і зберігає незалежність (ортогональність) між факторами.



Ознака	$\mathbf{F_1}$	$\mathbf{F_2}$	Спільність $\mathbf{h_j^2 = \sum a_{jr}^2}$	Характерність $\mathbf{d_j^2}$
X_1	0.931	0.162	0.892	0.108
X_2	0.357	0.870	0.884	0.116
X_3	-0.273	-0.141	0.094	0.906
X_4	0.948	0.252	0.962	0.038
X_5	0.205	0.881	0.819	0.181

Рис. 2.28. Спрощені результати для методу Хотеллінгу

Ознака	F_1	F_2	Спільність $h_j^2 = \sum a_{jr}^2$	Характерність d_j^2
X_1	0.931	0.162	0.892	0.108
X_2	0.357	0.870	0.884	0.116
X_3	-0.273	-0.141	0.094	0.906
X_4	0.948	0.252	0.962	0.038
X_5	0.205	0.881	0.819	0.181

Табл. 2.10. Спрощені результати для методу Хотеллінгу

Можна побачити, що перший фактор навантажує «Продуктивність праці» та «Капіталоозброєність»; цей фактор можна назвати — *Продуктивність*. Другий фактор навантажує «Рентабельність продукції» та «Капіта-

лорентабельність», цей фактор — *Рентабельність*.
Далі знайдемо відсоток поясненої дbсперсії(коефіцієнт інформативності).

$$I = \frac{\sum \lambda}{k} \tag{2.20}$$

де I – пояснена дисперсія; λ – власні значення; k – кількість показників.

$\lambda_1 = 2.764$
$\lambda_2 = 0.888$

Відсоток поясненої дисперсії $= \frac{2.764+0.888}{5} = 73\%$

Ці фактори пояснюють 73% дисперсії, це говорить про те, що фактори здебільшого пояснюють модель.

Також, якщо ми просумуємо отримані спільності та поділимо їх на суму початкових спільностей, то отримаємо відсоток відтворення спільностей початкової редукованої матриці:

$$J = \frac{0.892 + 0.884 + 0.094 + 0.962 + 0.819}{0.955 + 0.932 + 0.223 + 0.966 + 0.895} = 0.897 \tag{2.21}$$

Відсоток відтворення спільностей початкової редукованої матриці становить 89.7%, що говорить о високом відтворенні спільностей.

Метод Якобі

Для результатів методу Якобі(рис. 2.29)(табл. 2.11) ми беремо не нульові вектори, та ті, які пояснюють більше 15% дисперсії. Також змінемо напрям вектору останього фактору, для більш зручного аналізу.

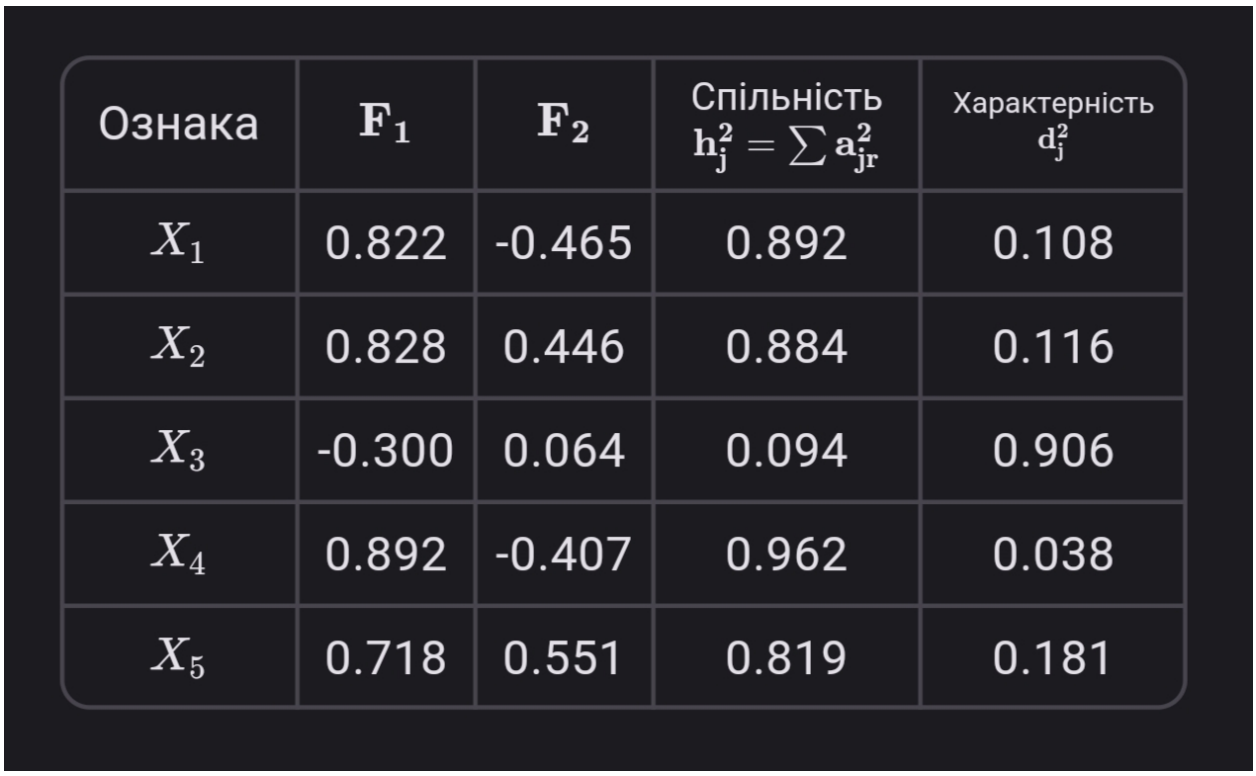
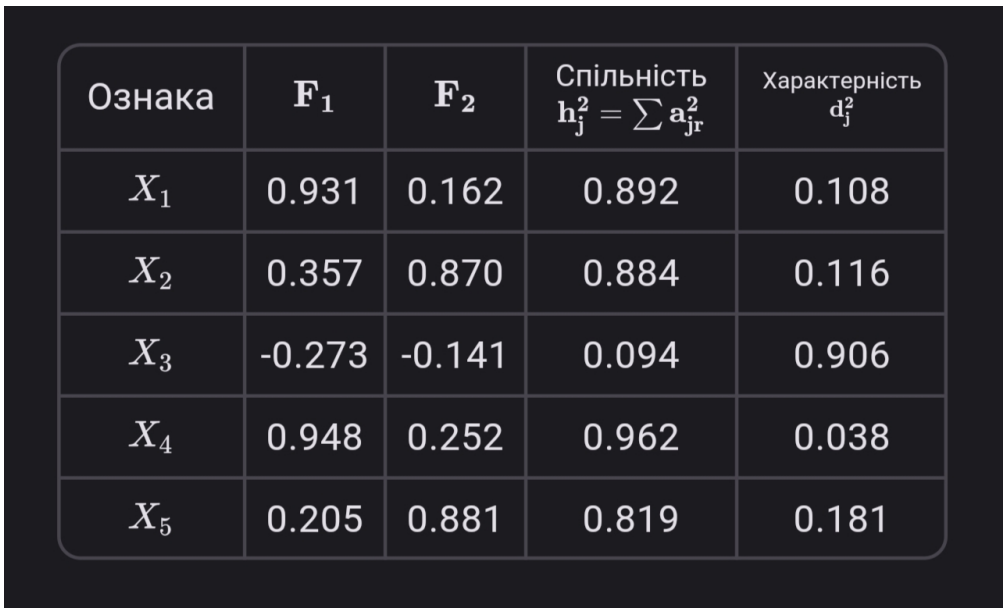


Рис. 2.29. Результати для методу головних факторів з методом Якобі

Ознака	F_1	F_2	Спільність $h_j^2 = \sum a_{jr}^2$	Характерність d_j^2
X_1	0.822	-0.465	0.892	0.108
X_2	0.828	0.446	0.884	0.116
X_3	-0.300	0.064	0.094	0.906
X_4	0.892	-0.407	0.962	0.038
X_5	0.718	0.551	0.819	0.181

Табл. 2.11. Результати для методу головних факторів з методом Якобі

Також спростимо значення завдяки варімакс(рис. 2.30)(табл. 2.12).



Ознака	$\mathbf{F_1}$	$\mathbf{F_2}$	Спільність $\mathbf{h_j^2 = \sum a_{jr}^2}$	Характерність $\mathbf{d_j^2}$
X_1	0.931	0.162	0.892	0.108
X_2	0.357	0.870	0.884	0.116
X_3	-0.273	-0.141	0.094	0.906
X_4	0.948	0.252	0.962	0.038
X_5	0.205	0.881	0.819	0.181

Рис. 2.30. Спрощені результати для методу Якобі

Ознака	F_1	F_2	Спільність $h_j^2 = \sum a_{jr}^2$	Характерність d_j^2
X_1	0.931	0.162	0.892	0.108
X_2	0.357	0.870	0.884	0.116
X_3	-0.273	-0.141	0.094	0.906
X_4	0.948	0.252	0.962	0.038
X_5	0.205	0.881	0.819	0.181

Табл. 2.12. Спрощені результати для методу Якобі

Можна побачити, що результати для двох методів зійшлися, що говорить про можливість використання будь якого з них. Також через це, аналіз для методу Якобі аналогічний методу Хотеллінга.

Оцінка підприємств за отриманими факторами

Через те, що метод Хотеллінга та метод Якобі дали однакові результати, оцінка підприємств актуальна для обох методів.

Помножимо матрицю нормалізованих даних на матрицю факторів, щоб отримати матрицю оцінок підприємств за продуктивністю та рентабельністю(рис. 2.31).

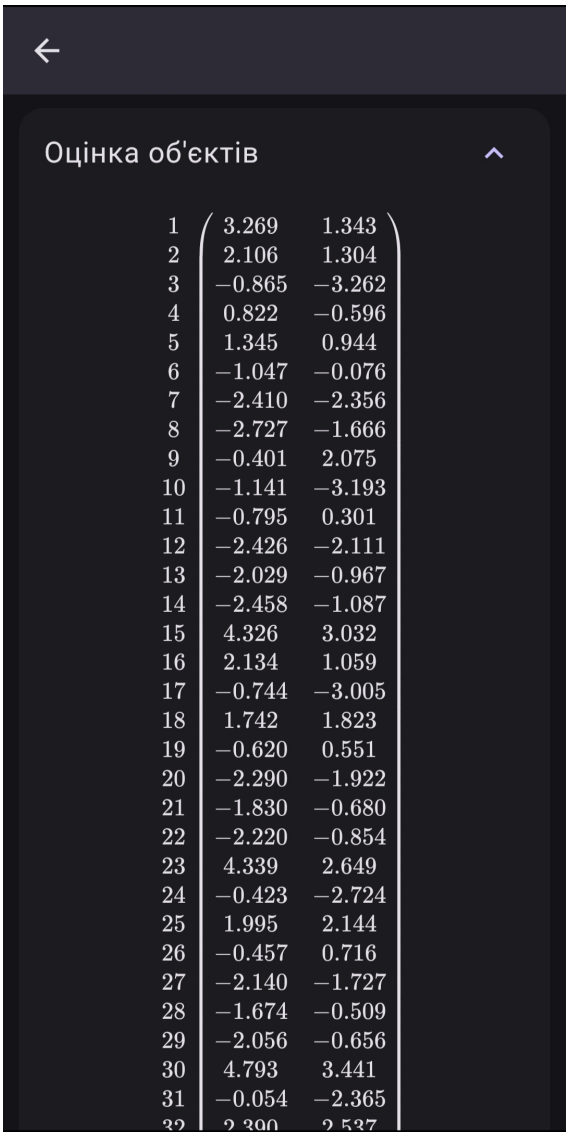


Рис. 2.31. Оцінка підприємств для методу головних факторів

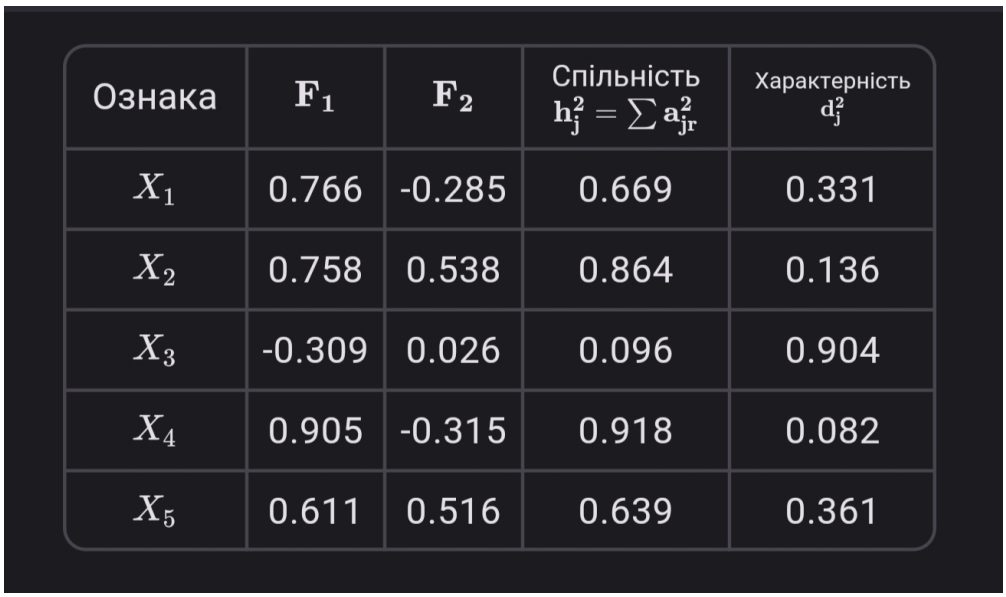
$$\begin{array}{c} 1 \\ \vdots \\ 30 \\ 31 \\ 32 \\ 33 \\ 34 \\ 35 \\ 36 \\ 37 \\ 38 \\ 39 \\ 40 \end{array} \begin{pmatrix} 3.269 & 1.343 \\ \vdots & \vdots \\ 4.793 & 3.441 \\ -0.054 & -2.365 \\ 2.390 & 2.537 \\ 0.040 & 1.139 \\ -1.743 & -1.316 \\ -1.688 & -0.321 \\ 4.824 & 3.116 \\ 2.688 & 2.893 \\ 0.351 & 1.407 \\ -1.503 & -1.054 \\ -1.421 & -0.024 \end{pmatrix} \tag{2.22}$$

Результати повністю можна переглянути у таблиці(табл. 2.17).

Підприємство номер 30 має найвищі показники для двох факторів.

2.3.2 Метод мінімальних залишків

З векторів навантажень ми отримали 2 вектори(рис. 2.32)(табл. 2.13).



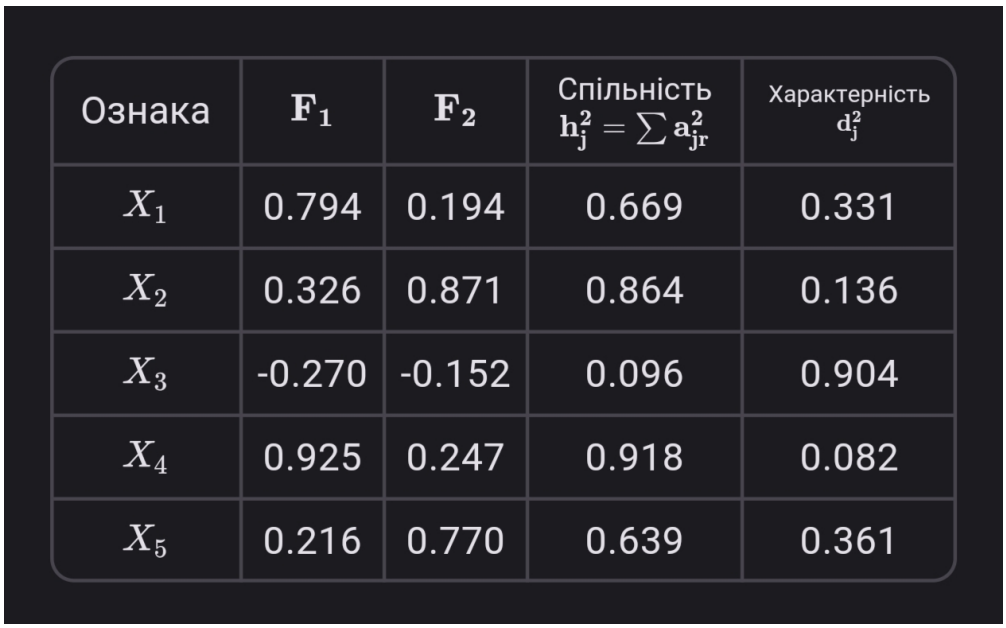
Ознака	$\mathbf{F_1}$	$\mathbf{F_2}$	Спільність $\mathbf{h_j^2 = \sum a_{jr}^2}$	Характерність $\mathbf{d_j^2}$
X_1	0.766	-0.285	0.669	0.331
X_2	0.758	0.538	0.864	0.136
X_3	-0.309	0.026	0.096	0.904
X_4	0.905	-0.315	0.918	0.082
X_5	0.611	0.516	0.639	0.361

Рис. 2.32. Результати для методу мінімальних залишків

Ознака	F_1	F_2	Спільність $h_j^2 = \sum a_{jr}^2$	Характерність d_j^2
X_1	0.766	-0.285	0.669	0.331
X_2	0.758	0.538	0.864	0.136
X_3	-0.309	0.026	0.096	0.904
X_4	0.905	-0.315	0.918	0.082
X_5	0.611	0.516	0.639	0.361

Табл. 2.13. Результати для методу мінімальних залишків

Спрощення даних завдяки варімакс(рис. 2.33)(табл. 2.14).



Ознака	$\mathbf{F_1}$	$\mathbf{F_2}$	Спільність $\mathbf{h_j^2 = \sum a_{jr}^2}$	Характерність $\mathbf{d_j^2}$
X_1	0.794	0.194	0.669	0.331
X_2	0.326	0.871	0.864	0.136
X_3	-0.270	-0.152	0.096	0.904
X_4	0.925	0.247	0.918	0.082
X_5	0.216	0.770	0.639	0.361

Рис. 2.33. Спрощені результати

Ознака	F_1	F_2	Спільність $h_j^2 = \sum a_{jr}^2$	Характерність d_j^2
X_1	0.794	0.194	0.669	0.331
X_2	0.326	0.871	0.864	0.136
X_3	-0.270	-0.152	0.096	0.904
X_4	0.925	0.247	0.918	0.082
X_5	0.216	0.770	0.639	0.361

Табл. 2.14. Спрощені результати

$\lambda_1 = 2.445$

$\lambda_2 = 0.737$

Відсоток поясненої дисперсії $= \frac{2.445+0.737}{5} = 63.73\%$

Як можна побачити, фактори навантажують ті самі показники, що й у методі головних факторів. Ці фактори пояснюють 63.73% дисперсії.

Оцінка підприємств за отриманими факторами

Для методу мінімальних залишків робимо ті ж самі операції, що для методу головних факторів(рис. 2.34).

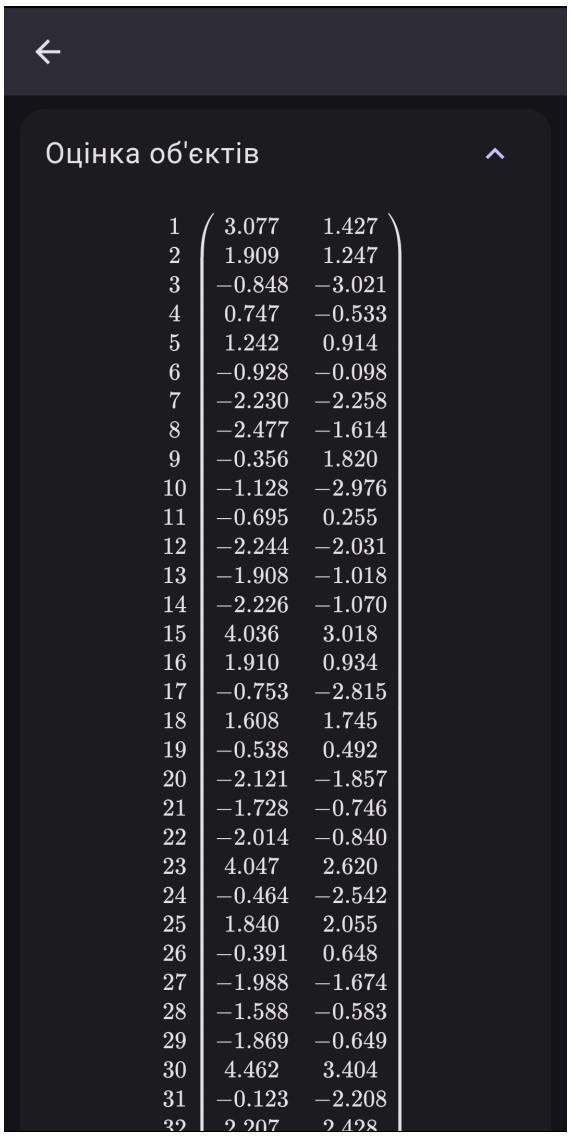


Рис. 2.34. Оцінка підприємств для методу мінімальних залишків

$$\begin{array}{c} 1 \\ \vdots \\ 30 \\ 31 \\ 32 \\ 33 \\ 34 \\ 35 \\ 36 \\ 37 \\ 38 \\ 39 \\ 40 \end{array} \begin{pmatrix} 3.077 & 1.427 \\ \vdots & \vdots \\ 4.462 & 3.404 \\ -0.123 & -2.208 \\ 2.207 & 2.428 \\ 0.065 & 1.056 \\ -1.625 & -1.281 \\ -1.531 & -0.329 \\ 4.492 & 3.061 \\ 2.479 & 2.772 \\ 0.346 & 1.314 \\ -1.407 & -1.028 \\ -1.290 & -0.042 \end{pmatrix} \tag{2.23}$$

Повні дані можно переглянути у таблиці(табл. 2.18).

Можна побачити, що для методу мінімальних залишків також, за значеннями факторів, перемагає підприємство під номером 30.

2.3.3 Порівняння результатів

Проведений порівняльний аналіз двох методів факторного аналізу дозволяє зробити наступні висновки щодо їх ефективності та практичного застосування.

Метод головних факторів продемонстрував вищу ефективність порівняно з методом мінімальних залишків, що підтверджується більшим відсотком інформативності(відсоток поясненої дисперсії) отриманих результатів. Цей показник свідчить про здатність методу краще пояснювати варіацію в до-

сліджуваних даних.

Метод мінімальних залишків, хоча і показав дещо нижчі результати за критерієм інформативності, має суттєві переваги в практичному застосуванні. Зокрема, він характеризується простішою реалізацією та не вимагає попередньої оцінки спільності змінних, що робить його більш доступним для практичного використання.

ВИСНОВКИ

У ході виконання дипломної роботи було реалізовано програмний інструмент для проведення факторного аналізу економічних показників діяльності підприємств. Основною метою дослідження було зменшення розмірності множини змінних та виявлення латентних факторів, що дозволяють адекватно описати основні напрями варіації в даних.

Для досягнення поставленої мети було використано метод головних факторів і метод мінімальних залишків. Проведена попередня обробка даних включала нормалізацію змінних за допомогою Z-score трансформації, що забезпечило порівнянність показників, а також побудову кореляційної матриці.

На основі обраних методів проведено виділення латентних факторів та обчислено факторні навантаження. Для покращення інтерпретованості результатів було застосовано метод ортогонального обертання варімакс, що дозволив отримати більш чітку структуру взаємозв'язків між змінними та факторами. Аналіз отриманих результатів показав, що більшість змінних мають високі значення спільностей, а отже добре пояснюються виділеними факторами.

Окрему увагу приділено реалізації програмного інструменту, розробленого з використанням мови програмування Dart та фреймворку Flutter. Графічний інтерфейс користувача забезпечує зручність взаємодії з даними та наочне представлення результатів аналізу.

Таким чином, поставлені завдання були успішно виконані: розроблено інструмент для багатовимірного статистичного аналізу, підтверджено ефективність застосування факторного підходу до аналізу економічної інформації, а також створено програмне забезпечення, придатне до практичного використання в аналітичних задачах.

СПИСОК ЛІТЕРАТУРИ

1. Яровий А. Т. Багатомірний статистичний аналіз / Яровий А. Т., Страхов Є. М. — Одеса:ОНУ, 2016. — 148 с.
2. Comry A. L. A first course in factor analysis / Comry A. L., Howard B. L. — New York, 1972. — 430 с.
3. Harman H. H. Factor analysis / Harman H. H., Holzinger K. J. — Chicago:The University of Chicago, 1941. — 417 с.
4. Harman H. H. Modern factor analysis — Chicago:The University of Chicago, 1976. — 487 с.

ДОДАТОК А

Таблиця даних задачі

Табл. 2.15. Показники підприємств

Підприємство	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{X_5}$
Підприємство 1	20000.0	0.2500	1.2903	15500.0	0.3226
Підприємство 2	21600.0	0.2222	1.7143	12600.0	0.3810
Підприємство 3	17500.0	0.1429	1.6406	10666.6667	0.2344
Підприємство 4	20000.0	0.1600	1.1212	9428.5714	0.3394
Підприємство 5	19285.7143	0.2250	1.6119	11964.2857	0.3627
Підприємство 6	14500.0	0.2162	1.5779	9189.1892	0.3412
Підприємство 7	13860.0	0.1667	1.6954	8175.0	0.2826
Підприємство 8	11922.2222	0.1892	1.5779	7555.5556	0.2985
Підприємство 9	16578.9474	0.2381	1.8694	8868.4211	0.4451
Підприємство 10	17652.0	0.1476	1.7982	10235.0	0.2417
Підприємство 11	14864.0	0.2233	1.5912	9375.0	0.3528
Підприємство 12	13678.0	0.1754	1.7310	8154.0	0.2885
Підприємство 13	15392.0	0.1856	1.9703	8012.0	0.3474
Підприємство 14	12174.0	0.2028	1.5897	7659.0	0.3146
Підприємство 15	21783.0	0.2901	1.4328	16230.0	0.3689
Підприємство 16	23147.0	0.1850	1.8792	12684.0	0.4125
Підприємство 17	18252.0	0.1407	1.8125	10987.0	0.2553
Підприємство 18	19621.0	0.2512	1.6934	12342.0	0.3827

Продовження таблиці 2.15

Підприємство	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{X_5}$
Підприємство 19	15184.0	0.2295	1.6215	9528.0	0.3594
Підприємство 20	13987.0	0.1781	1.7594	8273.0	0.2958
Підприємство 21	15731.0	0.1923	1.9847	8124.0	0.3549
Підприємство 22	12674.0	0.2102	1.6051	7795.0	0.3178
Підприємство 23	22248.0	0.2650	1.5001	16740.0	0.3754
Підприємство 24	18836.0	0.1487	1.8453	11347.0	0.2599
Підприємство 25	19983.0	0.2601	1.7095	12564.0	0.3891
Підприємство 26	15541.0	0.2327	1.6429	9675.0	0.3642
Підприємство 27	14329.0	0.1824	1.7896	8392.0	0.3016
Підприємство 28	16057.0	0.1958	1.9998	8237.0	0.3597
Підприємство 29	13048.0	0.2156	1.6204	7850.0	0.3221
Підприємство 30	22546.0	0.2987	1.5213	17042.0	0.3798
Підприємство 31	19374.0	0.1542	1.8927	11934.0	0.2714
Підприємство 32	20417.0	0.2689	1.7452	13210.0	0.3982
Підприємство 33	16258.0	0.2427	1.6734	10342.0	0.3725
Підприємство 34	14932.0	0.1928	1.8326	8901.0	0.3104
Підприємство 35	13597.0	0.2231	1.6538	8359.0	0.3297
Підприємство 36	22976.0	0.2748	1.5678	17492.0	0.3876
Підприємство 37	20836.0	0.2794	1.7624	13486.0	0.4043
Підприємство 38	16894.0	0.2482	1.6895	10591.0	0.3789

Продовження таблиці 2.15

Підприємство	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{X_5}$
Підприємство 39	15321.0	0.1997	1.8493	9128.0	0.3157
Підприємство 40	14076.0	0.2308	1.6697	8554.0	0.3359

ДОДАТОК Б

Таблиця нормалізованих даних задачі

Табл. 2.16. Нормалізовані показники підприємств

Підприємство	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{X_5}$
Підприємство 1	0.862	0.860	-2.273	1.696	-0.332
Підприємство 2	1.349	0.207	0.148	0.676	0.857
Підприємство 3	0.102	-1.657	-0.273	-0.005	-2.128
Підприємство 4	0.862	-1.255	-3.239	-0.440	0.010
Підприємство 5	0.645	0.273	-0.436	0.452	0.485
Підприємство 6	-0.811	0.066	-0.631	-0.524	0.047
Підприємство 7	-1.006	-1.097	0.040	-0.881	-1.146
Підприємство 8	-1.596	-0.569	-0.631	-1.099	-0.823
Підприємство 9	-0.179	0.581	1.034	-0.637	2.162
Підприємство 10	0.148	-1.546	0.627	-0.156	-1.979
Підприємство 11	-0.701	0.233	-0.555	-0.459	0.283
Підприємство 12	-1.062	-0.893	0.244	-0.889	-1.026
Підприємство 13	-0.540	-0.653	1.610	-0.939	0.173
Підприємство 14	-1.519	-0.249	-0.563	-1.063	-0.495
Підприємство 15	1.405	1.803	-1.459	1.953	0.611
Підприємство 16	1.820	-0.667	1.090	0.705	1.499
Підприємство 17	0.330	-1.708	0.709	0.108	-1.702
Підприємство 18	0.747	0.888	0.029	0.585	0.892

Продовження таблиці 2.16

Підприємство	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{X_5}$
Підприємство 19	-0.603	0.379	-0.382	-0.405	0.417
Підприємство 20	-0.967	-0.829	0.406	-0.847	-0.878
Підприємство 21	-0.437	-0.496	1.692	-0.899	0.326
Підприємство 22	-1.367	-0.075	-0.475	-1.015	-0.430
Підприємство 23	1.546	1.213	-1.075	2.132	0.743
Підприємство 24	0.508	-1.520	0.896	0.235	-1.609
Підприємство 25	0.857	1.098	0.121	0.663	1.022
Підприємство 26	-0.495	0.454	-0.259	-0.353	0.515
Підприємство 27	-0.863	-0.728	0.578	-0.805	-0.760
Підприємство 28	-0.338	-0.413	1.779	-0.859	0.423
Підприємство 29	-1.253	0.052	-0.388	-0.995	-0.342
Підприємство 30	1.637	2.005	-0.954	2.238	0.833
Підприємство 31	0.672	-1.391	1.167	0.441	-1.375
Підприємство 32	0.989	1.304	0.325	0.890	1.207
Підприємство 33	-0.276	0.689	-0.085	-0.119	0.684
Підприємство 34	-0.680	-0.484	0.824	-0.626	-0.580
Підприємство 35	-1.086	0.228	-0.197	-0.816	-0.187
Підприємство 36	1.768	1.443	-0.688	2.397	0.992
Підприємство 37	1.117	1.551	0.423	0.987	1.332
Підприємство 38	-0.083	0.818	0.007	-0.031	0.814

Продовження таблиці 2.16

Підприємство	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{X_5}$
Підприємство 39	-0.562	-0.322	0.919	-0.546	-0.472
Підприємство 40	-0.940	0.409	-0.106	-0.748	-0.061

ДОДАТОК В

Код для методу Хотеллінга

```
// Функція для отримання початкової зведеної матриці
// Обчислює суми по стовпцях і нормалізовані значення для початкового етапу
ReducedMatrix getInitialReducedMatrix(List<List<double>> reducedMatrix) {
  // Отримуємо кількість стовпців і рядків матриці
  final cols = reducedMatrix[0].length;
  final rows = reducedMatrix.length;

  // Створюємо списки для зберігання сум і нормалізованих значень
  final sumList = List.filled(cols, 0.0);
  final normalizedList = List.filled(cols, 0.0);

  // Обчислюємо суму елементів по кожному стовпцю
  for (int j = 0; j < cols; j++) {
    double sum = 0.0;
    for (int i = 0; i < rows; i++) {
      sum += reducedMatrix[i][j];
    }
    sumList[j] = sum;
  }

  // Знаходимо максимальну суму серед усіх стовпців
  final maxSum = sumList.reduce(max);

  // Нормалізуємо суми відносно максимальної суми
  for (int j = 0; j < cols; j++) {
    normalizedList[j] = sumList[j] / maxSum;
  }

  // Повертаємо об'єкт ReducedMatrix з усіма обчисленими значеннями
  return ReducedMatrix(
    reducedMatrix: reducedMatrix,     // Початкова матриця
    sum: sumList,                     // Суми по стовпцях
    normalizedSum: normalizedList,    // Нормалізовані суми
  );
}

```

```
// Функція для виконання одного циклу ітерації в методі Готеллінга
// Піднімає матрицю до заданого степеня і обчислює необхідні коефіцієнти
IterationCycle iterationLoop(
    List<List<double>> matrix,    // Вихідна матриця
    int power,                    // Степінь, до якого піднімається матриця
    List<double> prevS,           // Попередні суми з минулої ітерації
    List<double> prevA,           // Попередні нормалізовані коефіцієнти (альфа)
    ) {
  // Піднімаємо матрицю до заданого степеня
  final matrixPower = powerMatrix(matrix, power);

  // Піднімаємо матрицю до половинного степеня (для обчислення p)
  final halfPowerMatrix = powerMatrix(matrix, power ~/ 2);

  // Отримуємо розміри матриці
  final cols = matrixPower[0].length;
  final rows = matrixPower.length;

  // Обчислюємо суми по стовпцях для матриці в степені
  final sumList = List.filled(cols, 0.0);
  for (int j = 0; j < cols; j++) {
    for (int i = 0; i < rows; i++) {
      sumList[j] += matrixPower[i][j];
    }
  }

  // Перетворюємо попередній вектор S в матричну форму для множення
  final prevSMatrix = [prevS];

  // Обчислюємо вектор p: p = (R^(k/2))^T * S_(k-1)
  final p = transposeMatrix(multiplyMatrices(halfPowerMatrix, transposeMatrix(prevSMatrix)));
  final pVector = p[0]; // Витягуємо вектор з матриці

  // Знаходимо максимальний елемент вектора p
  final maxP = pVector.reduce(max);

  // Нормалізуємо вектор p відносно максимального елемента (отримуємо альфа)
  final normalizedList = List.generate(cols, (j) => pVector[j] / maxP);
```

```
75
76    // Обчислюємо різниці між поточними і попередніми значеннями альфа
77    // Це потрібно для перевірки збіжності алгоритму
78    final differenceList = List.generate(cols, (j) => (normalizedList[j] -
      ↪  prevA[j]).abs());
79
80    // Повертаємо результати поточної ітерації
81    return IterationCycle(
82      matrix: matrixPower,         // Матриця в степені
83      sum: sumList,                // Суми по стовпцях
84      p: pVector,                  // Вектор p
85      alpha: normalizedList,       // Нормалізовані коефіцієнти (альфа)
86      difference: differenceList,  // Різниці для перевірки збіжності
87    );
88  }
89
90  // Функція для знаходження першого залишкового коефіцієнта
91  // Виконує ітерації до досягнення заданої точності
92  List<IterationCycle> getResidualFirstCoefficient(
93      List<List<double>> matrix,           // Вихідна матриця
94      ReducedMatrix initialResult,         // Початкові дані
95      double accuracy,                     // Задана точність збіжності
96      ) {
97    List<IterationCycle> results = [];   // Список для зберігання результатів
      ↪  усіх ітерацій
98    var prevS = initialResult.sum;        // Початкові суми
99    var prevA = initialResult.normalizedSum; // Початкові нормалізовані
      ↪  значення
100   int power = 2;                        // Початковий степінь матриці
101
102   // Виконуємо ітерації до досягнення збіжності
103   while (true) {
104     // Виконуємо один цикл ітерації
105     final iteration = iterationLoop(matrix, power, prevS, prevA);
106
107     // Оновлюємо значення для наступної ітерації
108     prevS = iteration.sum;
109     prevA = iteration.alpha;
110
111     // Зберігаємо результат поточної ітерації
112     results.add(iteration);
```

```
113
114     // Подвоюємо степінь для наступної ітерації
115     power *= 2;
116
117     // Перевіряємо збіжність: чи всі різниці менші за задану точність
118     if (iteration.difference.every((e) => e <= accuracy)) {
119       break; // Якщо досягнута збіжність, виходимо з циклу
120     }
121   }
122
123   return results; // Повертаємо всі результати ітерацій
124 }
125
126 // Функція для обчислення основного факторного навантаження
127 // Знаходить власний вектор і факторні навантаження
128 MainFactorLoading getMainFactorLoading(
129     List<List<double>> matrix,    // Вихідна матриця
130     List<double> a,               // Вектор коефіцієнтів альфа
131     ) {
132
133   // Перетворюємо вектор a в матричну форму
134   final aMatrix = [a];
135
136   // Транспонуємо для отримання власного вектора (стовпця)
137   final eigenvectorMatrix = transposeMatrix(aMatrix);
138   final eigenvector = eigenvectorMatrix.map((row) => row[0]).toList();
139
140   // Обчислюємо бета-вектор:  = R * u (де u - власний вектор)
141   final betaMatrix = multiplyMatrices(matrix, eigenvectorMatrix);
142   final beta = betaMatrix.map((row) => row[0]).toList();
143
144   // Знаходимо максимальне власне значення (лямбда)
145   final lambda = beta.reduce(max);
146   final sqrtLambda = sqrt(lambda); // Квадратний корінь з лямбда
147
148   // Обчислюємо суму квадратів елементів власного вектора
149   double sumSquaredU = eigenvector
150       .map((x) => x * x)
151       .reduce((a, b) => a + b);
152   final sqrtSumSquaredU = sqrt(sumSquaredU); // Квадратний корінь з суми
    ↪  квадратів
```

```
153
154   // Обчислюємо вектор факторних навантажень
155   // Формула: A_i = (u_i * ) / (u_i²)
156   final factorLoadVector = eigenvector
157       .map((x) => (x * sqrtLambda) / sqrtSumSquaredU)
158       .toList();
159
160   // Повертаємо всі обчислені значення
161   return MainFactorLoading(
162     eigenvector: eigenvector,              // Власний вектор
163     beta: beta,                            // Бета-вектор
164     factorLoadVector: factorLoadVector,    // Вектор факторних навантажень
165   );
166 }
167
168 // Функція для обчислення другого залишкового коефіцієнта
169 // Віднімає реконструйовану матрицю від вихідної зведеної матриці
170 List<List<double>> getResidualSecondCoefficient(
171     List<double> A,                        // Вектор факторних навантажень
172     List<List<double>> reducedMatrix,      // Зведена матриця
173     ) {
174
175   // Перетворюємо вектор A в матричну форму
176   final AMatrix = [A];
177
178   // Обчислюємо реконструйовану матрицю: A^T * A
179   // Це матриця, яка відновлює частину вихідної матриці через перший фактор
180   final reconstructed = multiplyMatrices(transposeMatrix(AMatrix),
      ↪  AMatrix);
181
182   // Віднімаємо реконструйовану матрицю від зведеної матриці
183   // Отримуємо залишкову матрицю для наступного фактора
184   return subtractMatrices(reducedMatrix, reconstructed);
185 }
186
187 // Головна функція методу Готеллінга для факторного аналізу
188 // Послідовно виділяє фактори до досягнення заданої точності
189 List<HotellingApproach> hotellingApproach(
190     List<List<double>> reducedMatrix,                    // Початкова
       ↪  зведена матриця
191     double estimatesAccuracy,                            // Точність для
       ↪  збіжності оцінок
```

```
192     double residualSecondCoefficientAccuracy,           // Точність для
        ↪  залишкових коефіцієнтів
193     ) {
194   List<List<double>> R = reducedMatrix;  // Поточна матриця для обробки
195   List<HotellingApproach> results = [];  // Список результатів для кожного
      ↪  фактора
196
197   // Основний цикл виділення факторів
198   while (true) {
199     // Крок 1: Отримуємо початкову зведену матрицю
200     final reduced = getInitialReducedMatrix(R);
201
202     // Крок 2: Знаходимо перший залишковий коефіцієнт через ітерації
203     final iterationCycles = getResidualFirstCoefficient(R, reduced,
        ↪  estimatesAccuracy);
204
205     // Крок 3: Обчислюємо основне факторне навантаження
206     final mainLoading = getMainFactorLoading(R,
        ↪  iterationCycles.last.alpha);
207
208     // Крок 4: Обчислюємо другий залишковий коефіцієнт (нову залишкову
        ↪  матрицю)
209     final residualSecond =
        ↪  getResidualSecondCoefficient(mainLoading.factorLoadVector, R);
210
211     // Зберігаємо результати поточного фактора
212     results.add(HotellingApproach(
213       reducedMatrix: reduced,                    // Зведена матриця
214       iterationCycles: iterationCycles,          // Результати ітерацій
215       mainFactorLoading: mainLoading,            // Факторні навантаження
216       residualSecondCoefficient: residualSecond, // Залишкова матриця
217     ),);
218
219     // Оновлюємо матрицю для наступної ітерації
220     R = residualSecond;
221
222     // Перевіряємо умови зупинки:
223     // 1. Чи всі елементи залишкової матриці менші за задану точність
224     // 2. Чи не перевищено максимальну кількість факторів (розмір матриці)
225     if (residualSecond.expand((e) => e).every((e) => e <=
        ↪  residualSecondCoefficientAccuracy) ||
```

```
226         results.length >= reducedMatrix.length) {
227       break; // Зупиняємо процес виділення факторів
228     }
229   }
230
231   // Повертаємо список усіх виділених факторів
232   return results;
233 }
```

Код 2.1. Метод Хотеллінга

ДОДАТОК Г

Код для методу Якобі

```
// Функція для діагоналізації симетричної матриці методом Якобі
// Знаходить власні значення та власні вектори матриці
Jacobi jacobiDiagonalization(
    List<List<double>> A,           // Вхідна симетрична матриця
    {double epsilon = 1e-10,        // Точність збіжності (за замовчуванням 1e-10)
     int maxIterations = 100000}    // Максимальна кількість ітерацій
    ) {

  int n = A.length;  // Розмір матриці (n x n)

  // Створюємо копію вхідної матриці D, яка буде поступово діагоналізуватися
  List<List<double>> D = List.generate(n, (i) => List.from(A[i]));

  // Створюємо одиничну матрицю B, яка буде накопичувати власні вектори
  List<List<double>> B = List.generate(n, (i) => List.generate(n, (j) => i == j ? 1.0 : 0.0));

  // Основний ітераційний цикл методу Якобі
  for (int iter = 0; iter < maxIterations; iter++) {
    // Ініціалізуємо змінні для пошуку максимального недіагонального елемента
    int p = 0, q = 1;           // Індекси максимального елемента
    double maxVal = 0.0;        // Значення максимального елемента

    // Шукаємо максимальний за модулем недіагональний елемент матриці D
    for (int i = 0; i < n; i++) {
      for (int j = i + 1; j < n; j++) {  // Розглядаємо тільки верхню трикутну частину
        if (D[i][j].abs() > maxVal) {
          maxVal = D[i][j].abs();
          p = i;  // Рядок максимального елемента
          q = j;  // Стовпець максимального елемента
        }
      }
```

```
32      }
33
34      // Перевіряємо умову збіжності
35      // Якщо максимальний недіагональний елемент менший за epsilon, то
        ↪  алгоритм збігся
36      if (maxVal < epsilon) break;
37
38      // Обчислюємо кут повороту phi для обнулення елемента D[p][q]
39      // Формула:  = 0.5 * arctan(2*D[p][q] / (D[q][q] - D[p][p]))
40      double phi = 0.5 * atan2(2 * D[p][q], D[q][q] - D[p][p]);
41
42      // Обчислюємо косинус і синус кута повороту
43      double c = cos(phi);
44      double s = sin(phi);
45
46      // Створюємо матрицю повороту Якобі J
47      // Це одинична матриця з модифікованими елементами в позиціях (p,p),
        ↪  (q,q), (p,q), (q,p)
48      List<List<double>> J = List.generate(n, (i) => List.generate(n, (j) =>
        ↪  i == j ? 1.0 : 0.0));
49      J[p][p] = c;   // cos()
50      J[q][q] = c;   // cos()
51      J[p][q] = s;   // sin()
52      J[q][p] = -s;  // -sin()
53
54      // Застосовуємо перетворення подібності: D = J^T * D * J
55      // Це перетворення обнулює елемент D[p][q] і зберігає власні значення
56      D = multiplyMatrices(transposeMatrix(J), multiplyMatrices(D, J));
57
58      // Накопичуємо власні вектори: B = B * J
59      // Стовпці матриці B поступово стають власними векторами
60      B = multiplyMatrices(B, J);
61    }
62
63    // Повертаємо результат: B містить власні вектори, D містить власні
      ↪  значення на діагоналі
64    return Jacobi(B: B, D: D);
65  }
66
67  // Функція для побудови факторної матриці на основі результатів
    ↪  діагоналізації Якобі
```

```
// Створює матрицю факторних навантажень для факторного аналізу
FactorMatrix constructFactorMatrix(Jacobi jacobi) {
  List<List<double>> B = jacobi.B;  // Матриця власних векторів
  List<List<double>> D = jacobi.D;  // Діагональна матриця власних значень
  int n = B.length;                 // Розмір матриці

  // Створюємо матрицю A для факторних навантажень
  List<List<double>> A = List.generate(n, (_) => List.filled(n, 0.0));

  // Обчислюємо факторні навантаження
  for (int i = 0; i < n; i++) {      // Для кожного рядка (змінної)
    for (int j = 0; j < n; j++) {    // Для кожного стовпця (фактора)
      // Обчислюємо квадратний корінь з власного значення
      // Якщо власне значення від'ємне, використовуємо 0 (для стабільності)
      double sqrtLambda = D[j][j] > 0 ? sqrt(D[j][j]) : 0.0;

      // Факторне навантаження = власний вектор * (власне значення)
      // Формула: A[i][j] = B[i][j] * [j]
      A[i][j] = B[i][j] * sqrtLambda;
    }
  }

  // Повертаємо структуру з усіма обчисленими матрицями:
  // B - власні вектори, D - власні значення, A - факторні навантаження
  return FactorMatrix(B: B, D: D, A: A);
}
```

Код 2.2. Метод Якобі

ДОДАТОК Ґ

Код для методу мінімальних залишків

```
// Провайдер стану для управління методом мінімальних залишків (MinRes)
// Використовує StateNotifier для керування списком результатів факторного аналізу
class MinResNotifier extends StateNotifier<List<MinResClass>> {
  // Конструктор ініціалізує порожній список результатів
  MinResNotifier() : super([]);

  // Функція для скидання стану до початкового (порожнього списку)
  void reset() {
    state = [];
  }

  // Основна функція для вилучення наступного фактора методом мінімальних залишків
  void extractNextFactor(List<List<double>> inputCorrelationMatrix, {
    double epsilon = 0.0005,        // Точність збіжності алгоритму
    int maxIterations = 100,        // Максимальна кількість ітерацій
  }) {
    final n = inputCorrelationMatrix.length;  // Розмір кореляційної матриці

    // Визначаємо поточну матрицю для обробки:
    // - Якщо це перший фактор, використовуємо вихідну кореляційну матрицю
    // - Якщо не перший, обчислюємо залишкову матрицю після вилучення попередніх факторів
    final currentMatrix = state.isEmpty
        ? inputCorrelationMatrix
        : _computeResidual(inputCorrelationMatrix, state);

    // Створюємо матрицю з нульовою діагоналлю (обнуляємо діагональні елементи)
    // Це потрібно для методу мінімальних залишків, щоб не враховувати автокореляції
    final zeroDiagonalMatrix = List.generate(n, (i) =>
        List.generate(n, (j) => i == j ? 0.0 : currentMatrix[i][j]));

```

```
31      // Обчислюємо початковий вектор A0
32      // Для кожного стовпця знаходимо елемент з максимальним абсолютним
        ↪ значенням
33      final A0 = List.generate(n, (j) {
34        double maxAbs = 0.0;    // Максимальне абсолютне значення
35        double value = 0.0;     // Відповідне значення з знаком
36        for (int i = 0; i < n; i++) {
37          final r = zeroDiagonalMatrix[i][j];
38          if (r.abs() > maxAbs) {
39            maxAbs = r.abs();
40            value = r;
41          }
42        }
43        return value;
44      });
45
46      // Ініціалізуємо вектори для ітераційного процесу
47      List<double> A1 = List.filled(n, 0.0);      // Результат першої
        ↪ ітерації
48      List<double> A2 = List.filled(n, 0.0);      // Результат другої
        ↪ ітерації
49      List<double> A0Current = List.from(A0);     // Поточний робочий вектор
50
51      // Основний ітераційний цикл методу мінімальних залишків
52      for (int iteration = 0; iteration < maxIterations; iteration++) {
53
54        // Перша ітерація: обчислюємо A1
55        // Для кожного елемента i обчислюємо зважену суму інших елементів
56        for (int i = 0; i < n; i++) {
57          double numerator = 0.0;      // Чисельник: сума добутків
58          double denominator = 0.0;    // Знаменник: сума квадратів
59          for (int j = 0; j < n; j++) {
60            if (j == i) continue;      // Пропускаємо діагональний елемент
61            numerator += zeroDiagonalMatrix[i][j] * A0Current[j];
62            denominator += A0Current[j] * A0Current[j];
63          }
64          A1[i] = numerator / denominator;
65        }
66
67        // Друга ітерація: обчислюємо A2 аналогічно до A1, але використовуючи
          ↪ A1
```

```
68        for (int i = 0; i < n; i++) {
69          double numerator = 0.0;
70          double denominator = 0.0;
71          for (int j = 0; j < n; j++) {
72            if (j == i) continue;
73            numerator += zeroDiagonalMatrix[i][j] * A1[j];
74            denominator += A1[j] * A1[j];
75          }
76          A2[i] = numerator / denominator;
77        }
78
79        // Обчислюємо норми векторів A1 та A2
80        final norm1 = _vectorNorm(A1);
81        final norm2 = _vectorNorm(A2);
82
83        // Обчислюємо коефіцієнт масштабування k для стабілізації алгоритму
84        // Формула: k = (||A1||² / ||A2||²)^(1/4)
85        final k = pow(norm1 * norm1 / (norm2 * norm2), 0.25).toDouble();
86
87        // Масштабуємо A2 для наступної ітерації
88        A0Current = _scaleVector(A2, k);
89
90        // Перевіряємо збіжність: обчислюємо різниці між A1 та A2
91        final diff = _vectorDifference(A1, A2);
92
93        // Якщо всі різниці менші за epsilon, алгоритм збігся
94        if (diff.every((d) => d < epsilon)) {
95          // Створюємо результат поточного фактора
96          final result = MinResClass(
97            correlationMatrix: currentMatrix,              // Поточна
               ↪ кореляційна матриця
98            correlationMatrixWithZeros: zeroDiagonalMatrix, // Матриця з
               ↪ нульовою діагоналлю
99            A0: A0,                                        // Початковий
               ↪ вектор
100           A1: A1,                                        // Результат
               ↪ першої ітерації
101           A2: A2,                                        // Результат
               ↪ другої ітерації (фінальний фактор)
102         );
103
```

```
104         // Додаємо результат до стану
105         state = [...state, result];
106         return;
107       }
108
109       // Перевіряємо на патологічний випадок: вектори мають однакову
          ↪  величину, але протилежні знаки
110       final equalMagnitude = List.generate(n, (i) =>
111       (A1[i].abs() - A2[i].abs()).abs() < epsilon);
112       final oppositeSigns = List.generate(n, (i) => A1[i] * A2[i] < 0);
113
114       // Якщо всі елементи мають однакову величину та протилежні знаки,
          ↪  алгоритм не може збігтися
115       if (equalMagnitude.every((x) => x) && oppositeSigns.every((x) => x))
          ↪  {
116         throw StateError('Фактор не може бути вилучений - вектори
            ↪  протилежні за знаком');
117       }
118     }
119
120     // Якщо перевищено максимальну кількість ітерацій без збіжності
121     throw StateError('Перевищено максимальну кількість ітерацій
      ↪  ($maxIterations)');
122   }
123
124   // Функція для вилучення заданої кількості факторів послідовно
125   void extractFactors(List<List<double>> inputCorrelationMatrix, {
126     required int count,              // Кількість факторів для вилучення
127     double epsilon = 0.0005,         // Точність збіжності
128     int maxIterations = 100,         // Максимальна кількість ітерацій для
      ↪  кожного фактора
129   }) {
130     // Послідовно вилучаємо фактори
131     for (int i = 0; i < count; i++) {
132       try {
133         extractNextFactor(
134           inputCorrelationMatrix,
135           epsilon: epsilon,
136           maxIterations: maxIterations,
137         );
138       } on StateError catch (e) {
```

```
139         // Якщо виникла помилка, зупиняємо процес і виводимо повідомлення
140         print('Вилучення зупинено на кроці ${state.length + 1}:
            ↪ ${e.message}');
141         break;
142       }
143     }
144   }
145
146   // Приватна функція для обчислення залишкової матриці
147   // Віднімає внесок усіх попередньо вилучених факторів від вихідної
      ↪ матриці
148   List<List<double>> _computeResidual(
149       List<List<double>> originalMatrix,    // Вихідна кореляційна матриця
150       List<MinResClass> previousResults     // Результати попередніх
          ↪ факторів
151       ) {
152     final n = originalMatrix.length;
153
154     // Створюємо копію вихідної матриці
155     final residual = List.generate(n, (i) =>
156     List<double>.from(originalMatrix[i]));
157
158     // Для кожного попереднього фактора віднімаємо його внесок
159     for (final res in previousResults) {
160       final A = res.A2;  // Використовуємо фінальний вектор фактора
161
162       // Обчислюємо зовнішній добуток A * A^T (внесок фактора в кореляційну
          ↪ матрицю)
163       final contribution = _outerProduct(A, A);
164
165       // Віднімаємо внесок від залишкової матриці
166       for (int i = 0; i < n; i++) {
167         for (int j = 0; j < n; j++) {
168           residual[i][j] -= contribution[i][j];
169         }
170       }
171     }
172     return residual;
173   }
174
175   // Обчислення евклідової норми вектора
```

```
176   // Формула: ||v|| = (v_i²)
177   double _vectorNorm(List<double> v) =>
178       sqrt(v.fold(0.0, (sum, x) => sum + x * x));
179
180   // Масштабування вектора на скалярне значення
181   // Повертає новий вектор, де кожен елемент помножений на scalar
182   List<double> _scaleVector(List<double> v, double scalar) =>
183       v.map((x) => x * scalar).toList();
184
185   // Обчислення абсолютних різниць між відповідними елементами двох
      ↪  векторів
186   // Повертає вектор |a_i - b_i| для всіх i
187   List<double> _vectorDifference(List<double> a, List<double> b) =>
188       List.generate(a.length, (i) => (a[i] - b[i]).abs());
189
190   // Обчислення зовнішнього добутку двох векторів
191   // Повертає матрицю M[i][j] = a[i] * b[j]
192   List<List<double>> _outerProduct(List<double> a, List<double> b) =>
193       List.generate(a.length, (i) =>
194           List.generate(b.length, (j) => a[i] * b[j]));
195 }
196
197 // Провайдер для Riverpod, який створює та управляє MinResNotifier
198 // Дозволяє використовувати метод мінімальних залишків у додатку
199 final minResProvider = StateNotifierProvider<MinResNotifier,
    ↪  List<MinResClass>>(
200       (ref) => MinResNotifier(),
201 );
```

Код 2.3. Метод мінімальних залишків

ДОДАТОК Д

Таблиця оцінок підприємств для методу головних факторів

Табл. 2.17. Таблиця оцінок підприємств для методу головних факторів

Підприємство	$\mathbf{F_1}$	$\mathbf{F_2}$
Підприємство 1	3.269	1.343
Підприємство 2	2.106	1.304
Підприємство 3	-0.865	-3.262
Підприємство 4	0.822	-0.596
Підприємство 5	1.345	0.944
Підприємство 6	-1.047	-0.076
Підприємство 7	-2.410	-2.356
Підприємство 8	-2.727	-1.666
Підприємство 9	-0.401	2.075
Підприємство 10	-1.141	-3.193
Підприємство 11	-0.795	0.301
Підприємство 12	-2.426	-2.111
Підприємство 13	-2.029	-0.967
Підприємство 14	-2.458	-1.087
Підприємство 15	4.326	3.032
Підприємство 16	2.134	1.059
Підприємство 17	-0.744	-3.005
Підприємство 18	1.742	1.823

Продовження таблиці 2.17

Підприємство	$\mathbf{F_1}$	$\mathbf{F_2}$
Підприємство 19	-0.620	0.551
Підприємство 20	-2.290	-1.922
Підприємство 21	-1.830	-0.680
Підприємство 22	-2.220	-0.854
Підприємство 23	4.339	2.649
Підприємство 24	-0.423	-2.724
Підприємство 25	1.995	2.144
Підприємство 26	-0.457	0.716
Підприємство 27	-2.140	-1.727
Підприємство 28	-1.674	-0.509
Підприємство 29	-2.056	-0.656
Підприємство 30	4.793	3.441
Підприємство 31	-0.054	-2.365
Підприємство 32	2.390	2.537
Підприємство 33	0.040	1.139
Підприємство 34	-1.743	-1.316
Підприємство 35	-1.688	-0.321
Підприємство 36	4.824	3.116
Підприємство 37	2.688	2.893
Підприємство 38	0.351	1.407

Продовження таблиці 2.17

Підприємство	$\mathbf{F_1}$	$\mathbf{F_2}$
Підприємство 39	-1.503	-1.054
Підприємство 40	-1.421	-0.024

ДОДАТОК Е

Таблиця оцінок підприємств для методу мінімальних залишків

Табл. 2.18. Таблиця оцінок підприємств для методу мінімальних залишків

Підприємство	$\mathbf{F_1}$	$\mathbf{F_2}$
Підприємство 1	3.077	1.427
Підприємство 2	1.909	1.247
Підприємство 3	-0.848	-3.021
Підприємство 4	0.747	-0.533
Підприємство 5	1.242	0.914
Підприємство 6	-0.928	-0.098
Підприємство 7	-2.230	-2.258
Підприємство 8	-2.477	-1.614
Підприємство 9	-0.356	1.820
Підприємство 10	-1.128	-2.976
Підприємство 11	-0.695	0.255
Підприємство 12	-2.244	-2.031
Підприємство 13	-1.908	-1.018
Підприємство 14	-2.226	-1.070
Підприємство 15	4.036	3.018
Підприємство 16	1.910	0.934
Підприємство 17	-0.753	-2.815
Підприємство 18	1.608	1.745

Продовження таблиці 2.18

Підприємство	$\mathbf{F_1}$	$\mathbf{F_2}$
Підприємство 19	-0.538	0.492
Підприємство 20	-2.121	-1.857
Підприємство 21	-1.728	-0.746
Підприємство 22	-2.014	-0.840
Підприємство 23	4.047	2.620
Підприємство 24	-0.464	-2.542
Підприємство 25	1.840	2.055
Підприємство 26	-0.391	0.648
Підприємство 27	-1.988	-1.674
Підприємство 28	-1.588	-0.583
Підприємство 29	-1.869	-0.649
Підприємство 30	4.462	3.404
Підприємство 31	-0.123	-2.208
Підприємство 32	2.207	2.428
Підприємство 33	0.065	1.056
Підприємство 34	-1.625	-1.281
Підприємство 35	-1.531	-0.329
Підприємство 36	4.492	3.061
Підприємство 37	2.479	2.772
Підприємство 38	0.346	1.314

Продовження таблиці 2.18

Підприємство	$\mathbf{F_1}$	$\mathbf{F_2}$
Підприємство 39	-1.407	-1.028
Підприємство 40	-1.290	-0.042

Давидов К.А.