

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА
з дисципліни «Основи програмування»
на тему: «Обернення матриці»

Студент 1 курсу, групи ІП-24
Новиков Гліб Костянтинович
Спеціальності 121 «Інженерія програмного
забезпечення»

Керівник к.т.н., доц. Муха І. П.
(посада, вчене звання, науковий
ступінь, прізвище та ініціали)

Кількість балів: _____
Національна оцінка _____

Члени комісії

ст. вик. Вітковська І. І.

(підпис)

(посада, вчене звання, науковий ступінь,
прізвище та ініціали)

ас. Вовк Є. А.

(підпис)

(посада, вчене звання, науковий ступінь,
прізвище та ініціали)

Київ - 2023 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрямок "ПІЗ"

Курс 1 Група ПІ-24

Семестр 2

ЗАВДАННЯ

на курсову роботу студента

Новикова Гліба Костянтиновича

(прізвище, ім'я, по батькові)

1. Тема роботи «Обернення матриці»

2. Строк здачі студентом закінченої роботи 04 червня 2023 року

3. Вихідні дані до роботи ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ, ДОДАТОК Б ТЕКСТ ПРОГРАМНОГО КОДУ

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)
Постановка задачі; теоретичні відомості; опис алгоритмів; опис програмного забезпечення;
тестування програмного забезпечення; інструкція користувача; аналіз результатів.

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

6. Дата видачі завдання 12 лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	12.02.2023	
2.	Підготовка ТЗ	25.03.2023	
3.	Пошук та вивчення літератури з питань курсової роботи	27.03.2023	
4.	Розробка сценарію роботи програми	30.03.2023	
6.	Узгодження сценарію роботи програми з керівником	31.03.2023	
5.	Розробка (вибір) алгоритму рішення задачі	03.04.2023	
6.	Узгодження алгоритму з керівником	10.04.2023	
7.	Узгодження з керівником інтерфейсу користувача	11.04.2023	
8.	Розробка програмного забезпечення	20.04.2023	
9.	Налагодження розрахункової частини програми	25.04.2023	
10.	Розробка та налагодження інтерфейсної частини програми	30.04.2023	
11.	Узгодження з керівником набору тестів для контрольного прикладу	01.05.2023	
12.	Тестування програми	02.05.2023	
13.	Підготовка пояснювальної записки	25.05.2023	
14.	Здача курсової роботи на перевірку	04.06.2023	
15.	Захист курсової роботи	08.06.2023	

Студент _____
(підпис)

Керівник _____
(підпис)

Вітковська І. І.
(прізвище, ім'я, по батькові)

" 12 " _____ лютого _____ 2023 р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 40 сторінок (74 із додатками), 14 рисунки, 19 таблиць, 4 посилання.

Мета роботи: розробка якісного програмного забезпечення, яке реалізує задачу з обернення матриці методами окаймлення та Шульца.

Вивчено методи обернення матриці.

Виконана програмна реалізація алгоритмів окаймлення та Шульца.

МАТРИЦЯ, КВАДРАТНА МАТРИЦЯ, ОБЕРНЕННЯ МАТРИЦІ, ВИЗНАЧНИК, МЕТОД ОКАЙМЛЕННЯ, МЕТОД ШУЛЬЦА, МНОЖЕННЯ МАТРИЦЬ, ОДИНИЧНА МАТРИЦЯ.

ЗМІСТ

ВСТУП	5
1 ПОСТАНОВКА ЗАДАЧІ.....	6
2 ТЕОРЕТИЧНІ ВІДОМОСТІ.....	7
3 ОПИС АЛГОРИТМІВ.....	10
3.1. Загальний алгоритм	10
3.2. Алгоритм методу окаймлення	12
4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	14
4.1. Діаграма класів програмного забезпечення.	14
4.2. Опис методів частин програмного забезпечення	16
4.2.1. Користувацькі методи	16
4.2.2. Стандартні методи	17
5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	20
5.1. План тестування	20
5.2. Приклади тестування	21
6 ІНСТРУКЦІЯ КОРИСТУВАЧА	29
7 АНАЛІЗ РЕЗУЛЬТАТІВ	36
ВИСНОВКИ.....	40
ПЕРЕЛІК ПОСИЛАНЬ.....	41
ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ.....	42
ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ	45

ВСТУП

У сучасному світі, де обробка великих обсягів даних є невід'ємною складовою багатьох наукових, технічних та інженерних завдань, розробка ефективних алгоритмів обчислення та обернення матриць має велике значення. Матриці використовуються в таких різноманітних галузях, як комп'ютерна графіка, статистика, машинне навчання, фізика, фінанси та багато інших. Пошук швидких та надійних методів обернення матриць є актуальною задачею, яка привертає увагу багатьох дослідників та розробників програмного забезпечення.

Необхідність розробки програм для обернення матриць методом Шульца та методом окаймлення виникає з багатьох причин. По-перше, обернена матриця є важливою математичною структурою, яка використовується для вирішення систем лінійних рівнянь, знаходження розв'язків диференціальних рівнянь та багатьох інших задач. По-друге, широке застосування матриць у різних галузях науки та технологій створює потребу у швидких та ефективних алгоритмах обернення. Розробка програмних продуктів з графічним інтерфейсом для користувача сприяє зручності використання цих алгоритмів та дозволяє широкому колу фахівців застосовувати їх у своїх дослідженнях та роботі.

У рамках цієї курсової роботи будуть розроблені програми для обернення матриць методом Шульца та методом окаймлення з графічним інтерфейсом для користувача. Програми будуть написані на мові програмування C++. Перш ніж реалізувати алгоритми, буде проведений огляд наукової літератури та досліджень з цієї теми для отримання більшого розуміння методів обернення матриць. Після цього будуть створені програмні модулі, що реалізують обидва методи обернення матриць та їх графічний інтерфейс. Програми будуть протестовані на різних наборах тестових даних, щоб перевірити їх точність та ефективність.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, задачею якого є обернення квадратної матриці наступними методами:

- а) метод окаймлення;
- б) метод Шульца.

Вхідними даними для даної роботи є матриця заданої розмірності елементами якої є дійсні числа. Матриця має такий вигляд:

$$A_{n \times n} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

Де A – матриця заданої розмірності, n – розмірність матриці, a – дійсне число, що є елементом матриці.

Вихідними даними для даної роботи є обернена до початкової матриця, що виводиться на екран. Програмне забезпечення повинно відображати аналіз методів розв’язання та зберігати вхідні та вихідні дані у текстовий файл.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Квадратну матрицю можна представити у наступному вигляді:

$$A_{n \times n} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix},$$

Де A – матриця заданої розмірності, n – розмірність матриці, a – дійсне число, що є елементом матриці.

Інші позначення для матриці: A , A_n , $A_{n \times n}$. Замість літери A може використовуватись будь-яка інша літера латинського алфавіта великого регістру.

Визначник або детермінант – це число, вираз складений за певним законом з n^2 елементів квадратної матриці. Одна з найважливіших характеристик квадратних матриць. Для квадратної матриці розміру $n \times n$ визначник є многочленом степеня n від елементів матриці, і є сумою добутків елементів матриці зі всіма можливими комбінаціями різних номерів рядків і стовпців (в кожному із добутків є рівно по одному елементу з кожного рядка і кожного стовпця). Кожному добутку приписується знак плюс чи мінус, в залежності від парності перестановки номерів. Позначення: ΔA , $\det A$,

$$\det(A) \text{ або } \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}.$$

Невиродженою називається квадратна матриця, визначник якої не дорівнює 0: $\det(A) \neq 0$.

Множення матриць – бінарна операція, що здійсненна тільки в тому випадку, якщо число стовпців в першому співмножнику дорівнює числу рядків у другому, і утворює нову матрицю, яка називається добутком матриць.

Добуток матриць AB складається з усіх можливих комбінацій скалярних добутків вектор-рядків матриці A і вектор-стовпців матриці B .

Одинична матриця – квадратна матриця розміру n , елементи головної діагоналі якої дорівнюють одиниці, а всі інші елементи дорівнюють нулю. Позначення: $I, I_n, I_{n \times n}, E, E_n, E_{n \times n}$.

Транспонована матриця – матриця, що виникає з матриці A в результаті унарної операції транспонування: заміни рядків матриці на стовпчики. Позначення: A^T .

Оберненою до квадратної матриці A називають таку невироджену матрицю, що добуток початкової та оберненої матриці дорівнює одиничній матриці. Позначення: $A^{-1}, \frac{1}{A}$. Замість літери A може використовуватись будь-яка інша літера латинського алфавіта великого регістру.

Основна властивість оберненої матриці:

$$AA^{-1} = A^{-1}A = I_n,$$

де I_n одинична матриця розмірності $n \times n$.

Отримати обернену матрицю можна наступними методами:

2.1. Метод окаймлення

Нехай дана невироджена матриця A розмірністю $n \times n$.

Сутність методу окаймлення наступна. Розглянемо матриці A_1, A_2, \dots, A_n , де $A_1 = a_{11}, A_2 = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \dots, A_1 = A$. Тобто, кожна наступна матриця виходить із попередньої «окаймленням» попередньої матриці рядком і стовпцем матриці A . На кожному кроці $k = 1, 2, \dots, n$ будується матриця A_k^{-1} , обернена до A_k . При чому, якщо матрицю A_k представити в формі:

$$A_k = \begin{pmatrix} A_{k-1} & U_k \\ V_k & a_{kk} \end{pmatrix},$$

де:

$$V_k = (a_{k1}, a_{k2}, \dots, a_{k,k-1}), U_k = (a_{1k}, a_{2k}, \dots, a_{k-1,k})^T,$$

то обернена до неї матриця A_k^{-1} матиме вигляд:

$$A_k^{-1} = \begin{pmatrix} B_{k-1}^{-1} & r_k \\ q_k & \frac{1}{a_k} \end{pmatrix},$$

де a_k – число:

$$a_k = a_{kk} - V_k * A_{k-1}^{-1} * U_k;$$

r_k – стовпчик:

$$r_k = -\frac{1}{a_k} * A_{k-1}^{-1} * U_k;$$

q_k – рядок:

$$q_k = -\frac{1}{a_k} * V_k * A_{k-1}^{-1};$$

B_{k-1}^{-1} – матриця:

$$B_{k-1}^{-1} = A_{k-1}^{-1} - (A_{k-1}^{-1} * U_k) * V_k.$$

Таким чином, для побудови A_k^{-1} використовується A_{k-1}^{-1} , що була побудована на попередньому кроці. На кроці n отримуємо $A_n^{-1} = A^{-1}$.

Для A_1 , $A_1^{-1} = \frac{1}{a_{11}}$.

2.2 Метод Шульца

Цей ітеративний метод складається з декількох кроків, які повторюються поки точність для елементів оберненої матриці не буде задовільною.

По-перше, необхідно задати початкове наближення оберненої матриці $U^{(0)}$; ε – мале додатне число.

Крок 2 – обчислення $\psi^{(k)}$:

$$\psi^{(k)} = E - A \cdot U^{(k)}.$$

Крок 3 – вирахувати $\|\psi^{(k)}\|$. Якщо $\|\psi^{(k)}\| \leq \varepsilon$, завершити процес і покласти $A^{-1} = U^{(k)}$. Інакше перейти до наступного кроку.

Крок 4 – знайти наступне наближення:

$$U^{(k+1)} = U^{(k)} \cdot \{E + \psi^{(k)} + [\psi^{(k)}]^2 + \dots + [\psi^{(k)}]^m\},$$

покласти $k = k + 1$ і перейти до кроку 2.

3 ОПИС АЛГОРИТМІВ

Перелік основних змінних та їхнє призначення наведено в таблиці 3.1.

Таблиця 3.1 – Основні змінні та їх призначення

Змінна	Призначення
size	Розмірність матриці
matrix	Початкова матриця A
prev	Матриця A_{k-1}
prev_reversed	Матриця A_{k-1}^{-1}
a	Число a_k
v	Рядок V_k
u	Стовпчик U_k
r	Стовпчик r_k
q	Рядок q_k
b	Матриця B_{k-1}^{-1}
reversed	Обернена до початкової матриця A^{-1}
epsilon	Близьке до нуля додатне число – точність обчислення
strange_suma	Корінь із суми квадратів усіх елементів матриці
E	Одинична матриця
psi	Наближення для обчислення оберненої матриці

3.1. Загальний алгоритм

1. ПОЧАТОК

2. Ініціалізація інтерфейсу програми.

3. ЯКЩО користувач обрав ручне введення, ТО:

3.1. Перейти до вікна ручного введення.

3.2. Зчитати елементи матриці.

3.3. ЯКЩО усі елементи матриці валідні, ТО:

3.3.1. Перейти до вікна показу оберненої матриці.

3.3.2. ЯКЩО обраний метод окаймлення, ТО:

3.3.2.1. Перейти до обробки даних згідно алгоритму методу окаймлення (підрозділ 3.2).

3.3.3. ЯКЩО обраний метод Шульца, ТО:

3.3.3.1. Перейти до обробки даних згідно алгоритму методу Шульца (підрозділ 3.3)

3.3.4. Вивести на екран початкову матрицю.

3.3.5. Вивести на екран обернену матрицю.

3.3.6. ЯКЩО користувач натиснув кнопку інформації про алгоритм, ТО:

3.3.6.1. Вивести інформацію про алгоритм.

3.3.7 ЯКЩО користувач натиснув кнопку запису до файлу, ТО:

3.3.7.1. Записати початкову та обернену матриці до файлу.

3.3.8 Якщо користувач натиснув на кнопку «Готово», ТО:

3.3.8.1. Перейти до пункту 2.

3.4 ІНАКШЕ:

3.4.1 Повідомити про помилку та перейти до пункту 3.1.

4. ЯКЩО користувач обрав генерацію матриці, ТО:

4.1. Згенерувати матрицю.

4.2 ЯКЩО згенерована матриця є невиродженою, ТО:

4.2.1. Перейти до вікна показу оберненої матриці.

4.2.2. ЯКЩО обраний метод окаймлення, ТО:

3.3.2.1. Перейти до обробки даних згідно алгоритму методу окаймлення (підрозділ 3.2).

4.2.3. ЯКЩО обраний метод Шульца, ТО:

4.2.3.1. Перейти до обробки даних згідно алгоритму методу Шульца (підрозділ 3.3)

4.2.4. Вивести на екран початкову матрицю.

4.2.5. Вивести на екран обернену матрицю.

4.2.6. ЯКЩО користувач натиснув кнопку інформації про алгоритм, ТО:

4.2.6.1. Вивести інформацію про алгоритм.

4.2.7 ЯКЩО користувач натиснув кнопку запису до файлу, ТО:

4.2.7.1. Записати початкову та обернену матриці до файлу.

4.2.8 Якщо користувач натиснув на кнопку «Готово», ТО:

4.2.8.1. Перейти до пункту 2.

4.3 ІНАКШЕ:

4.3.1. Перейти до пункту 4.1.

5. ЯКЩО натиснуто кнопку закриття програми, ТО:

5.1. КІНЕЦЬ.

3.2. Алгоритм методу окаймлення

1. ПОЧАТОК

2. ЯКЩО ($size == 1$), ТО:

2.1 ПОВЕРНУТИ ($1/matrix$).

3. Прирівняти $prev$ до $matrix$ розміром, зменшену на 1.

4. Прирівняти $prev_reversed$ до поверненого значення від рекурсивно застосованого алгоритму методу окаймлення до $prev$.

5. Прирівняти v до останнього рядка $matrix$ без останнього елемента.

6. Прирівняти u до останнього стовпчика $matrix$ без останнього елемента.

7. Обчислити значення a за формулою: $a = (\text{останній елемент matrix}) - (v * \text{prev_reversed} * u)$.
8. Обчислити значення q за формулою: $q = (-1/a) * v * \text{prev_reversed}$.
9. Обчислити значення r за формулою: $r = (-1/a) * \text{prev_reversed} * u$.
10. Обчислити b за формулою: $b = \text{prev_reversed} - ((\text{prev_reversed} * u) * q)$.
11. Обчислити значення reversed за формулою: $\text{reversed} = \text{матриця } b \text{ доповнена рядком } r, \text{ стовпцем } q \text{ і числом } a$.
12. ПОВЕРНУТИ reversed .
13. КІНЕЦЬ.

3.3 Алгоритм методу Шульца

1. ПОЧАТОК
2. Обрахувати транспоновану матрицю matrix .
3. Обрахувати початкове наближення як добуток початкової матриці matrix та транспонованої матриці matrix .
4. Обрахувати strange_suma як корінь суми квадратів усіх елементів матриці.
5. Знайти значення оберненої матриці reversed як частку транспонованої матриці matrix і strange_suma .
4. Обрахувати ψ за формулою: $\psi = E - \text{matrix} * \text{reversed}$.
5. ЦИКЛ ПОКИ ($\text{strange_suma} > \text{epsilon}$):
 - 5.1. Обрахувати strange_suma як корінь суми квадратів усіх елементів ψ .
 - 5.2. Обчислити reversed за формулою: $\text{reversed} = (\text{reversed}) * (E + \psi)$;
 - 5.3. Обрахувати ψ за формулою: $\psi = E - \text{matrix} * \text{reversed}$.
6. ПОВЕРНУТИ reversed .
7. КІНЕЦЬ.

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Діаграма класів програмного забезпечення.

На рис. 4.1 зображена діаграма класів програми. На ній зображені наступні класи:

- `matrix` – матриця розмірністю m на n , що містить усі методи та операції, які можна здійснювати над матрицями.
- `MainWindow` – відображення головного вікна-інтерфейсу, у якому міститься вибір розміру, способу задання, та методу обернення матриці.
- `InputWindow` – відображення вікна-інтерфейсу для введення елементів матриці.
- `MatrixInversion` – відображення вікна-інтерфейсу, в якому присутні початкова та обернена матриці, можливість запису в файл та аналіз алгоритму.

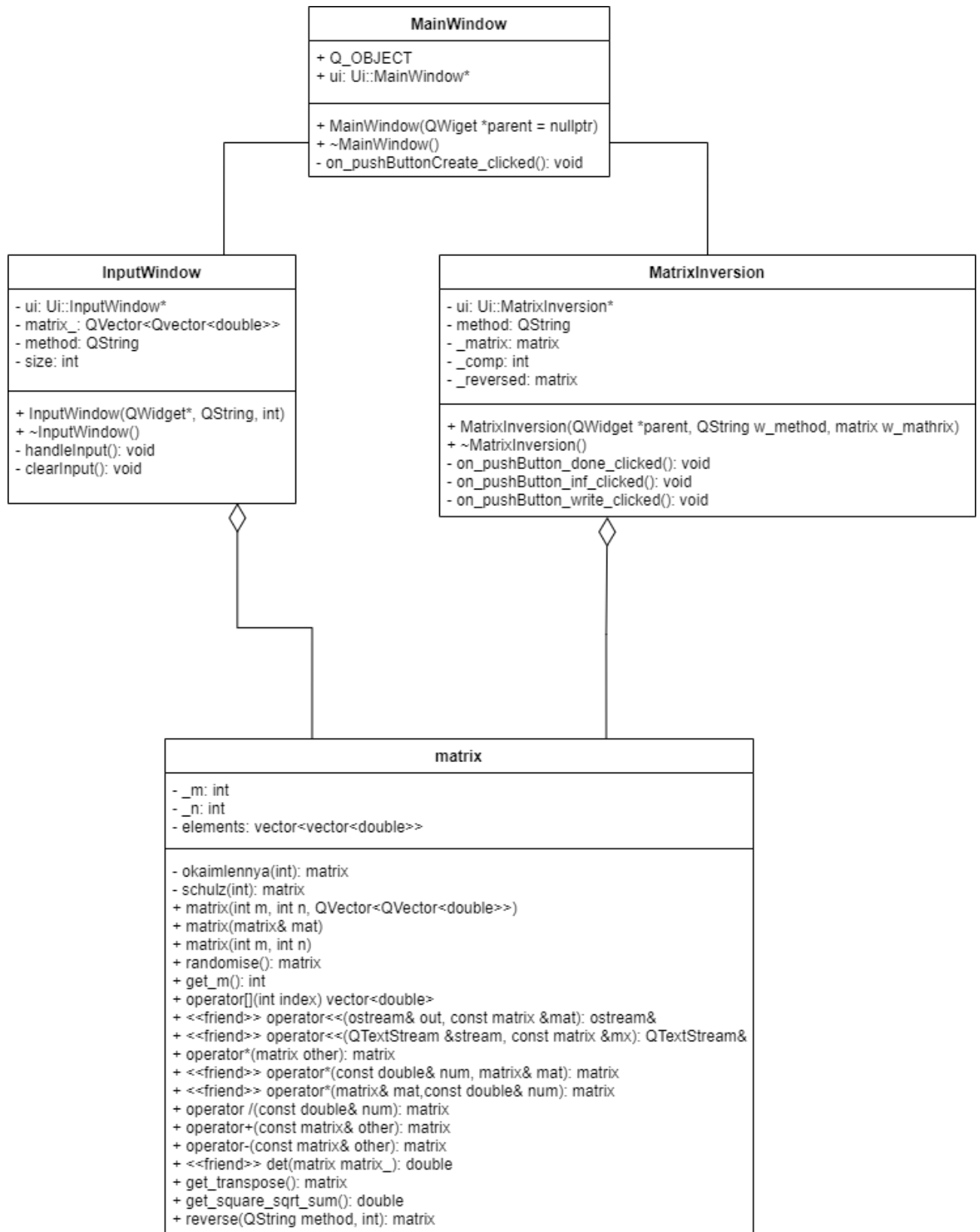


Рисунок 4.1 – Діаграма класів

4.2. Опис методів частин програмного забезпечення

4.2.1. Користувацькі методи

У таблиці 4.1 наведено користувацькі методи, які були розроблені спеціально для реалізації логіки програми.

Таблиця 4.1 – користувацькі методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
1	matrix	okaimlennya	Обрахунок оберненої матриці методом окаймлення	comp – не обов'язковий параметр, для підрахунку практичної складності	Обернена матриця	matrix.h
2	matrix	schulz	Обрахунок оберненої матриці методом Шульца	comp – не обов'язковий параметр, для підрахунку практичної складності	Обернена матриця	matrix.h
3	matrix	randomise	Заповнення матриці випадковими елементами	-	Матриця з випадково згенерован ими елементам и	matrix.h
4	matrix	det	Обчислення визначника матриці	Матриця matrix	Дійсне число – визначник матриці	matrix.h

Продовження таблиці 4.1

№ п/ п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовни й файл
5	matrix	get_transpose	Транспонування матриці	-	Транспонована матриця	matrix.h
6	matrix	get_square_sqrt_sum	Отримати корінь суми квадратів елементів матриці	-	Дійсне число	matrix.h
7	matrix	reverse	Обернути матрицю обраним методом	method – метод знаходження оберненої матриці comp – не обов'язковий параметр, для підрахунку практичної складності методу		matrix.h
8	matrix	get_m	Отримати кількість рядків у матриці	-	Ціле число – кількість рядків у матриці	matrix.h

4.2.2. Стандартні методи

У таблиці 4.2 наведено стандартні методи, реалізація яких наявна в стандартних та зовнішніх бібліотеках та не є власною реалізацією.

Таблиця 4.2 – стандартні методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
1	string	length	Отримати довжину рядка	-	Довжина рядка	string
4	string	c_str	Перетворення рядка типу std::string на рядок типу char*	-	Рядок типу char*	string
3	vector	size	Пошук кількості елементів у векторі	-	Ціле значення кількості елементів	vector
4	QString	toStdString	Перетворення рядка типу QString на рядок типу std::string	-	Рядок типу std::string	QString
5	QString	toInt	Перетворення рядка типу QString на число типу int	-	Ціле число	QString
6	QString	toDouble	Перетворення рядка типу QString на число типу double	-	Число з плаваючою крапкою	QString
7	QMainWindow	show	Показує вікно користувачу	-	-	QMainWindow
8	QMainWindow	hide	Ховає вікно від користувача	-	-	QMainWindow

Продовження таблиці 4.2

№ п/ п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрі в	Заголовний файл
9	QRadioButton	isChecked	Перевіряє чи натиснута кнопка	-	Логічне значення	QRadioButton
10	QTableWidget	setRowCount	Установлює кількість рядків у таблиці	Ціле число	-	QTableWidget
11	QTableWidget	setColumnCount	Установлює кількість стовпців у таблиці	Ціле число	-	QTableWidget
12	QTableWidget	setFixedSize	Установлює фіксований розмір таблиці	Ціле число – ширина, ціле число – висота	-	QTableWidget
13	QTableWidget	setGeometry	Установлює розташуванн я таблиці у вікні	Два цілих числа – координат и x та y	-	QTableWidget
14	QDialog	setFixedSize	Установлює фіксований розмір вікна	Ціле число – ширина, ціле число – висота	-	QDialog
15	QLabel	move	Переміщує напис на вказану позицію	Два цілих числа – координат и x та y	-	QLabel

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1. План тестування

Цей план тестування програмного забезпечення розроблений з метою перевірки функціональності та стабільності роботи програмного забезпечення. Він визначає стратегію тестування, цілі, завдання та методи, які будуть використовуватись під час процесу тестування. Тестування буде мануального типу з ручною розробкою тестових випадків. Тестування програмного забезпечення на виключні ситуації та основний функціонал програми було проведено за наступним планом.

а) Тестування кнопки створення матрицю.

- 1) Тестування кнопки створення матриці із обраними розміром матриці, ручним введенням і обернення методом окаймлення.
- 2) Тестування кнопки створення матриці із обраними розміром матриці, ручним введенням і обернення методом Шульца.
- 3) Тестування кнопки створення матриці із обраними розміром матриці, генерацією матриці і обернення методом окаймлення.
- 4) Тестування кнопки створення матриці із обраними розміром матриці, генерацією матриці і обернення методом Шульца.

б) Тестування на введення даних при ручному введенні елементів матриці.

- 1) Тестування натиснення кнопки обрахування при пустих значеннях в елементах масиву.
- 2) Тестування натиснення кнопки обрахувати при вводі некоректних символів у поля для елементів масиву.
- 3) Тестування натиснення кнопки обрахувати при вводі з великих значень у поля для елементів масиву.
- 4) Тестування натиснення кнопки обрахувати при введенні малих значень у поля для елементів масиву.
- 5) Тестування натиснення кнопки обрахувати з введеною виродженою матрицею розмірності 3 наступного вигляду:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

6) Тестування натиснення кнопки обрахувати з введеною невиродженою матрицею розмірності 3 наступного вигляду:

$$\begin{pmatrix} -1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

7) Тестування натиснення кнопки очистити.

г) Тестування кнопки запису даних у файл.

г) Тестування кнопки аналізу алгоритму обернення матриці.

5.2. Приклади тестування

Для перевірки очікуваних та реальних результатів поведінки згідно з планом тестування протестуємо коректність та стабільність роботи програмного забезпечення. У таблиця 5.1, 5.2, 5.3, 5.4 наведені приклади тестування кнопки створення матриці. У таблицях 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11 наведені приклади тестування на введення даних при ручному введенні елементів матриці. У таблиці 5.12 наведено тестування кнопки запису у файл. У таблиці 5.13 наведено тестування кнопки аналізу алгоритму обернення матриці.

Таблиця 5.1 – Тестування кнопки створення матриці із обраними розміром матриці, ручним введенням і обернення методом окаймлення

Мета тесту	Перевірити функціональність меню вибору
Початковий стан програми	Відкрите вікно програми
Вхідні дані	тип створення: ручне введення та метод обернення: окаймлення
Схема проведення тесту	Обрати певний розмір матриці, обрати тип створення: ручне введення та метод обернення: окаймлення
Очікуваний результат	Відкриється вікно введення елементів матриці
Стан програми після проведення випробувань	Відкрите вікно для введення елементів матриці

Таблиця 5.2 – Тестування кнопки створення матриці із обраними розміром матриці, ручним введенням і обернення методом Шульца

Мета тесту	Перевірити функціональність меню вибору
Початковий стан програми	Відкрите вікно програми
Вхідні дані	тип створення: ручне введення та метод обернення: метод Шульца
Схема проведення тесту	Обрати певний розмір матриці, обрати тип створення: ручне введення та метод обернення: метод Шульца
Очікуваний результат	Відкриється вікно введення елементів матриці
Стан програми після проведення випробувань	Відкрите вікно для введення елементів матриці

Таблиця 5.3 – Тестування кнопки створення матриці із обраними розміром матриці, генерацією матриці і обернення методом окаймлення

Мета тесту	Перевірити функціональність меню вибору
Початковий стан програми	Відкрите вікно програми
Вхідні дані	тип створення: генерація, метод обернення: окаймлення
Схема проведення тесту	Обрати певний розмір матриці, обрати тип створення: генерація та метод обернення: окаймлення
Очікуваний результат	Відкриється вікно зі згенерованою та оберненою матрицею методом окаймлення
Стан програми після проведення випробувань	Відкрите вікно зі згенерованою та оберненою матрицею методом окаймлення

Таблиця 5.4 – Тестування кнопки створення матриці із обраними розміром матриці, генерацією матриці і обернення методом Шульца

Мета тесту	Перевірити функціональність меню вибору
Початковий стан програми	Відкрите вікно програми
Вхідні дані	тип створення: генерація, метод обернення: метод Шульца
Схема проведення тесту	Обрати певний розмір матриці, обрати тип створення: генерація та метод обернення: метод Шульца
Очікуваний результат	Відкриється вікно зі згенерованою та оберненою матрицею методом Шульца
Стан програми після проведення випробувань	Відкрите вікно зі згенерованою та оберненою матрицею методом Шульца

Таблиця 5.5 – Тестування натиснення кнопки обрахування при пустих значеннях в елементах масиву

Мета тесту	Перевірити функціональність вікна введення елементів матриці
Початковий стан програми	Відкрите вікно для ручного введення даних
Вхідні дані	-
Схема проведення тесту	Натиснути кнопку обрахувати
Очікуваний результат	З'явиться повідомлення про помилку
Стан програми після проведення випробувань	З'явився напис червоним кольором «Будь-ласка введіть ЧИСЛА»

Таблиця 5.6 – Тестування натиснення кнопки обрахувати при вводі некоректних символів у поля для елементів масиву

Мета тесту	Перевірити функціональність вікна введення елементів матриці
Початковий стан програми	Відкрите вікно для ручного введення даних
Вхідні дані	Символи що не є цифрами
Схема проведення тесту	Увести у поля необхідні вхідні дані та натиснути кнопку обрахувати
Очікуваний результат	З'явиться повідомлення про помилку
Стан програми після проведення випробувань	З'явився напис червоним кольором «Будь-ласка введіть ЧИСЛА»

Таблиця 5.7 – Тестування натиснення кнопки обрахувати при вводі завеликих значень у поля для елементів масиву

Мета тесту	Перевірити функціональність вікна введення елементів матриці
Початковий стан програми	Відкрите вікно для ручного введення даних
Вхідні дані	Числа більше за 2^{16}
Схема проведення тесту	Увести у поля необхідні вхідні дані та натиснути кнопку обрахувати
Очікуваний результат	З'явиться повідомлення про помилку
Стан програми після проведення випробувань	З'явився напис червоним кольором «Введено занадто велике число»

Таблиця 5.8 – Тестування натиснення кнопки обрахувати при введенні замалих значень у поля для елементів масиву

Мета тесту	Перевірити функціональність вікна введення елементів матриці
Початковий стан програми	Відкрите вікно для ручного введення даних
Вхідні дані	Числа менше за -2^{16}
Схема проведення тесту	Увести у поля необхідні вхідні дані та натиснути кнопку обрахувати
Очікуваний результат	З'явиться повідомлення про помилку
Стан програми після проведення випробувань	З'явився напис червоним кольором «Введено занадто велике число»

Таблиця 5.9 – Тестування натиснення кнопки обрахувати з введеною виродженою матрицею розмірності 3

Мета тесту	Перевірити функціональність вікна введення елементів матриці
Початковий стан програми	Відкрите вікно для ручного введення даних
Вхідні дані	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
Схема проведення тесту	Увести у поля необхідні вхідні дані та натиснути кнопку обрахувати
Очікуваний результат	З'явиться повідомлення про помилку
Стан програми після проведення випробувань	З'явився напис червоним кольором «Оберненої матриці не існує»

Таблиця 5.10 – Тестування натиснення кнопки обрахувати з введеною невиродженою матрицею розмірності 3

Мета тесту	Перевірити функціональність вікна введення елементів матриці
Початковий стан програми	Відкрите вікно для ручного введення даних
Вхідні дані	$\begin{pmatrix} -1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
Схема проведення тесту	Увести у поля необхідні вхідні дані та натиснути кнопку обрахувати
Очікуваний результат	Відкриється наступне вікно з введеною та оберненою матрицею обраним методом
Стан програми після проведення випробувань	Відкрите вікно з введеною та оберненою матрицею обраним методом

Таблиця 5.11 – Тестування натиснення кнопки очистити

Мета тесту	Перевірити функціональність вікна введення елементів матриці
Початковий стан програми	Відкрите вікно для ручного введення даних
Вхідні дані	Деякі введені дані у поля для елементів матриці
Схема проведення тесту	Натиснути кнопку очистити
Очікуваний результат	Поля для введення елементів матриці стануть пустими
Стан програми після проведення випробувань	Відкрите вікно для ручного введення даних з пустими полями для введення елементів матриці

Таблиця 5.12 – Тестування кнопки запису даних у файл

Мета тесту	Перевірити функціональність вікна введення елементів матриці
Початковий стан програми	Відкрите вікно показу початкової та оберненої матриці
Вхідні дані	-
Схема проведення тесту	Натиснути кнопку записати у файл і задати шлях і назву файлу у який запишеться матриця
Очікуваний результат	Початкова і обернена матриці запишуться у файл
Стан програми після проведення випробувань	Початкова і обернена матриці записані у файл, а замість кнопки записати у файл надпис «Інформацію успішно записано у файл»

Таблиця 5.13 – Тестування кнопки аналізу алгоритму обернення матриці

Мета тесту	Перевірити функціональність вікна введення елементів матриці
Початковий стан програми	Відкрите вікно показу початкової та оберненої матриці
Вхідні дані	-
Схема проведення тесту	Натиснути кнопку отримати інформацію про алгоритм
Очікуваний результат	З'явиться інформація про використаний алгоритм обернення матриці
Стан програми після проведення випробувань	З'явилося інформаційне вікно з детальним описом алгоритму та його практичною складністю

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1. Робота з програмою

Після запуску застосунку відкривається вікно для вибору типу створення матриці, її розміру та методу обертання (рисунок 6.1).

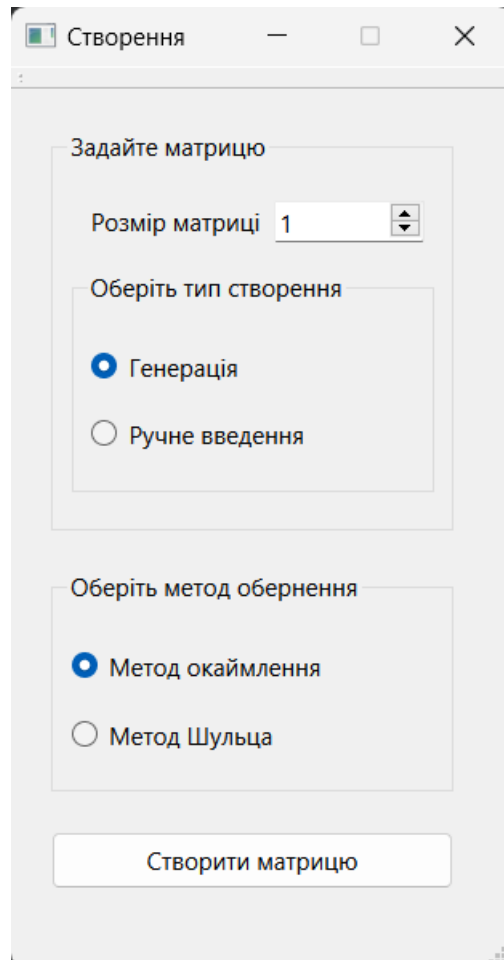


Рисунок 6.1 – Вікно для вибору типу створення матриці, її розміру та методу обертання

Одразу після запуску програми за умовчанням обрано розмір матриці, тип створення та метод обертання, тож користувач може без зайвих кроків натиснути на кнопку «Створити матрицю», або ж може обрати інший метод обертання, інший тип створення чи інший розмір матриці (до 10) і також натиснути кнопку «Створити матрицю».

Якщо було обрано генерацію, то відкриється нове вікно, у якому будуть відображені згенерована випадковим чином і обернена матриці, розмір яких

залежить від обраного раніше розміру, а метод обертання від обраного раніше методу (рисунк 6.2, рисунок 6.3).

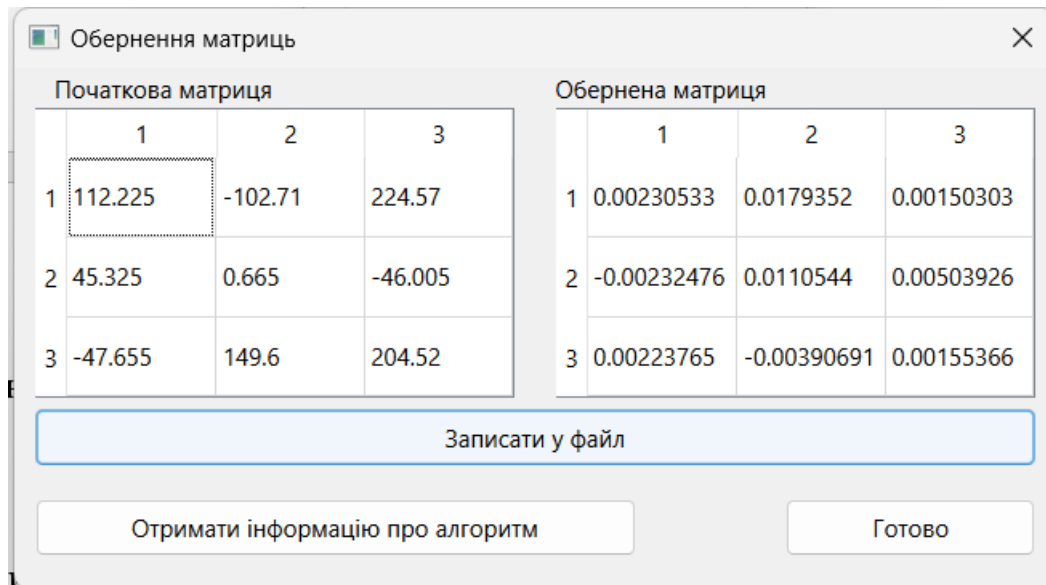


Рисунок 6.2 – Вікно показу згенерованої та оберненої матриці розміром 3 × 3

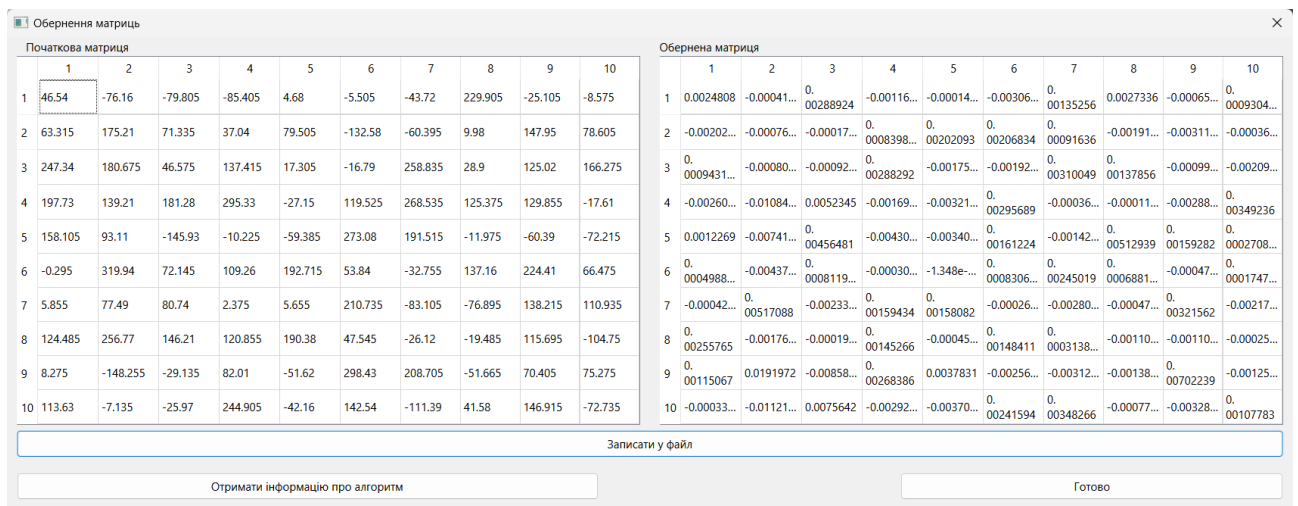


Рисунок 6.3 – Вікно показу згенерованої та оберненої матриці розміром 10 × 10

Також у вікні присутні три кнопки «Записати у файл», «Отримати інформацію про алгоритм» та «Готово».

Після натискання на кнопку «Записати у файл» відкриється провідник для вибору шляху та введення назви файлу (рисунк 6.4). Якщо ви вийдете з провідника, не зробивши запис, то у вікні з початковою та оберненою матрицею нічого не зміниться, а якщо ви успішно запишете матрицю у файл, то кнопка «Записати у файл» пропаде, а на її місці з'явиться напис зеленого кольору «Інформацію успішно записано у файл» (рисунк 6.5).

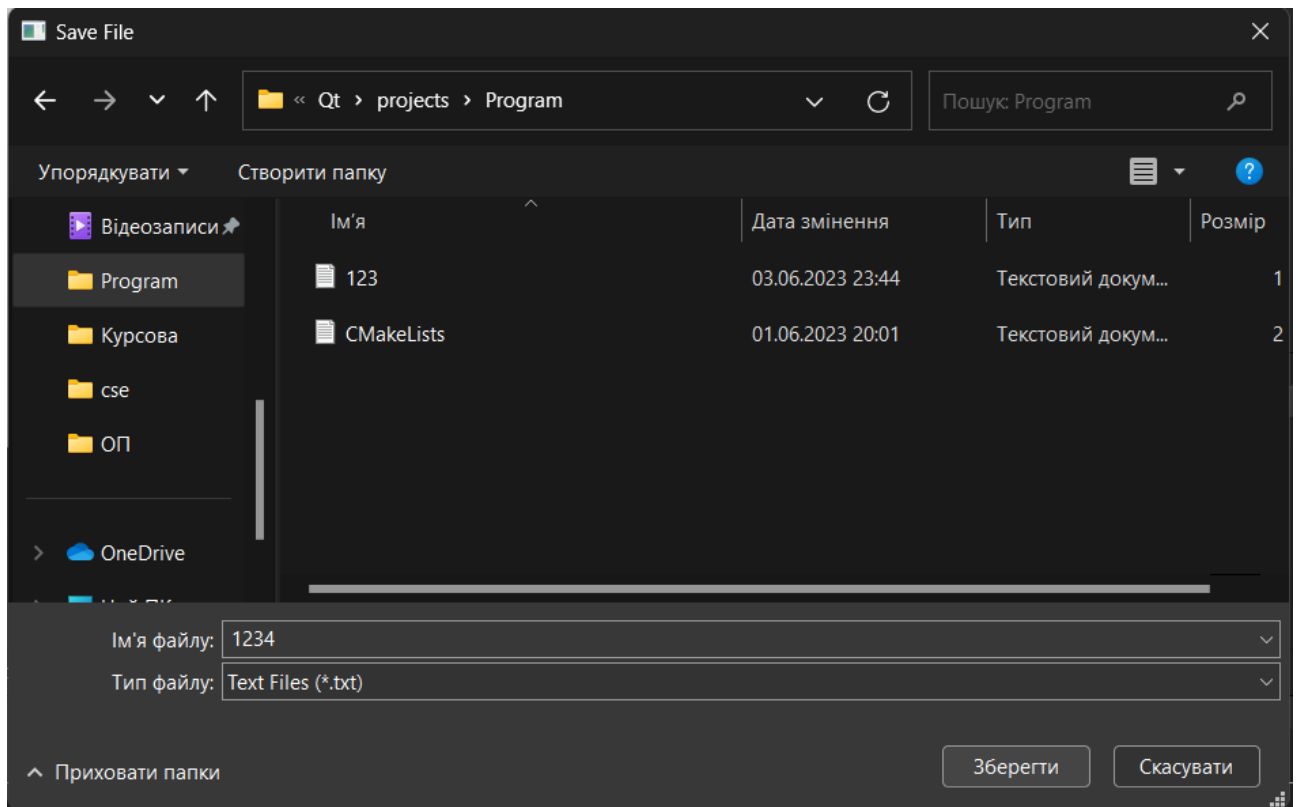


Рисунок 6.4 – Провідник для вибору шляху та введення назви файлу

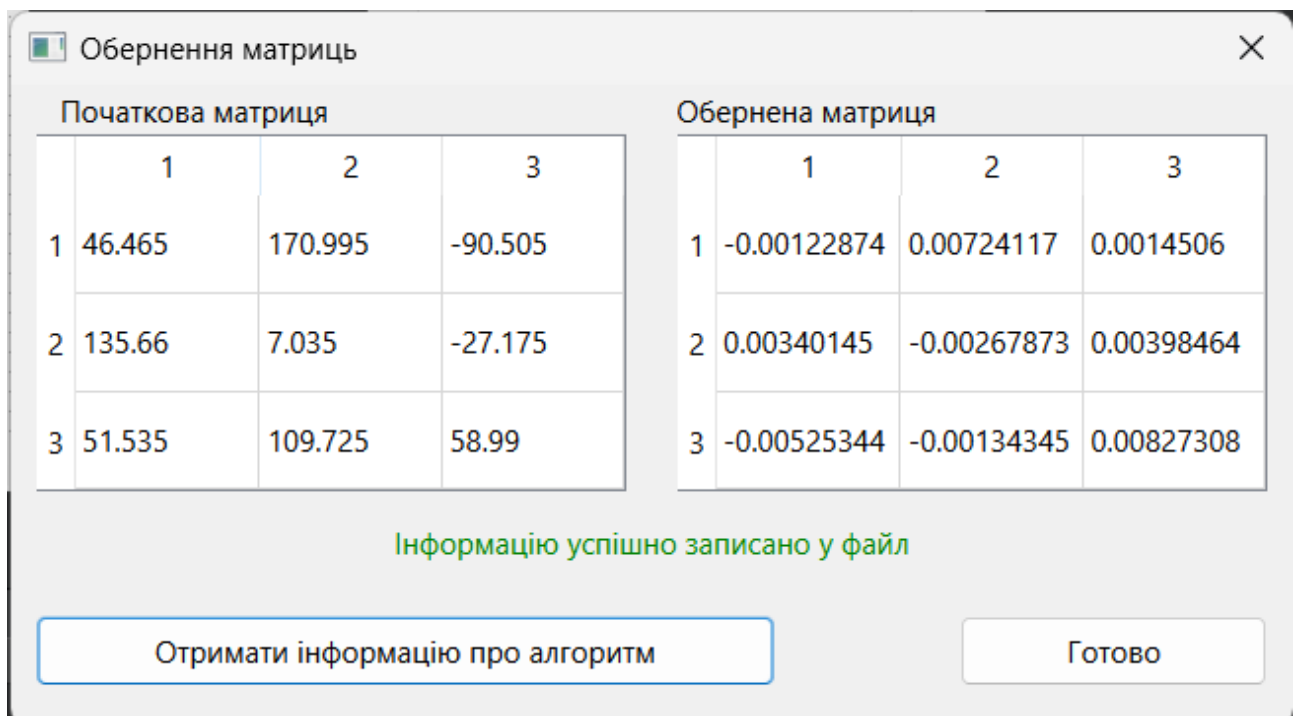


Рисунок 6.5 – Вікно програми після успішного запису у файл

Після натискання на кнопку «Отримати інформацію про алгоритм» відкриється інформаційне вікно з детальним описом обраного раніше

алгоритму, а також практична складність – кількість ітерацій, що пройшов алгоритм для обернення даної початкової матриці (рисунок 6.6).

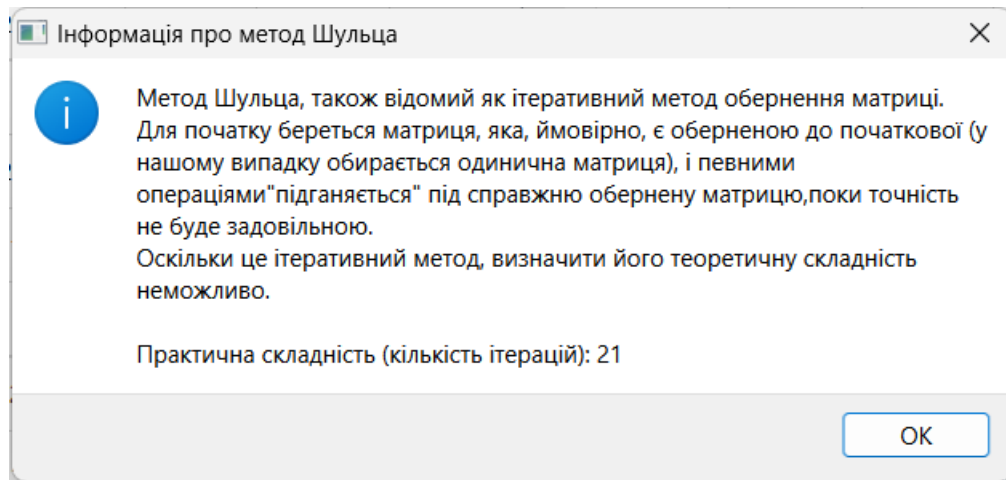


Рисунок 6.6 – Інформаційне вікно про обраний метод обернення

Після натискання на кнопку «Готово» дане вікно закривається і знову стає доступним для взаємодії початкове вікно для вибору типу створення матриці, її розміру та методу обертання, яке було зображене на рисунку 6.1.

Якщо у початковому вікні було обрано ручне введення, то відкривається вікно для введення елементів матриці (рисунок 6.7).

Заповнення матриць

Вставка елементів матриці

	1	2	3	4	5
1					
2					
3					
4					
5					

Очистити Обрахувати

Рисунок 6.7 – вікно для введення елементів матриці 5×5

У вікні присутні поля для введення елементів матриці та дві кнопки: «Очистити» і «Обрахувати».

При натисканні на кнопку «Очистити» все, що було введено в поля видаляється.

Якщо у полях для елементів матриці будуть введені будь-які невалідні дані, то при натисканні на кнопку «Обрахувати» з'явиться повідомлення про помилку (приклад на рисунку 6.8).

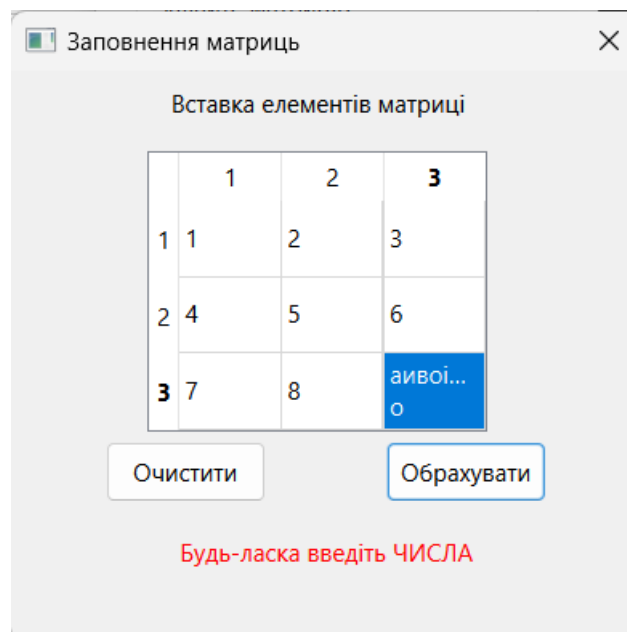


Рисунок 6.8 – Приклад повідомлення про невірно введені дані

Так само виведеться повідомлення, якщо введена матриця є виродженою і не може бути обернена.

Якщо ж усі введені дані будуть валідні, то при натисканні на кнопку «Обрахувати» відкриється те саме вікно, що і при виборі генерації матриці випадковим чином, як на рисунку 6.2, тільки замість випадково згенерованої матриці на місці початкової буде відображатись введена вручну матриця (рисунок 6.9).

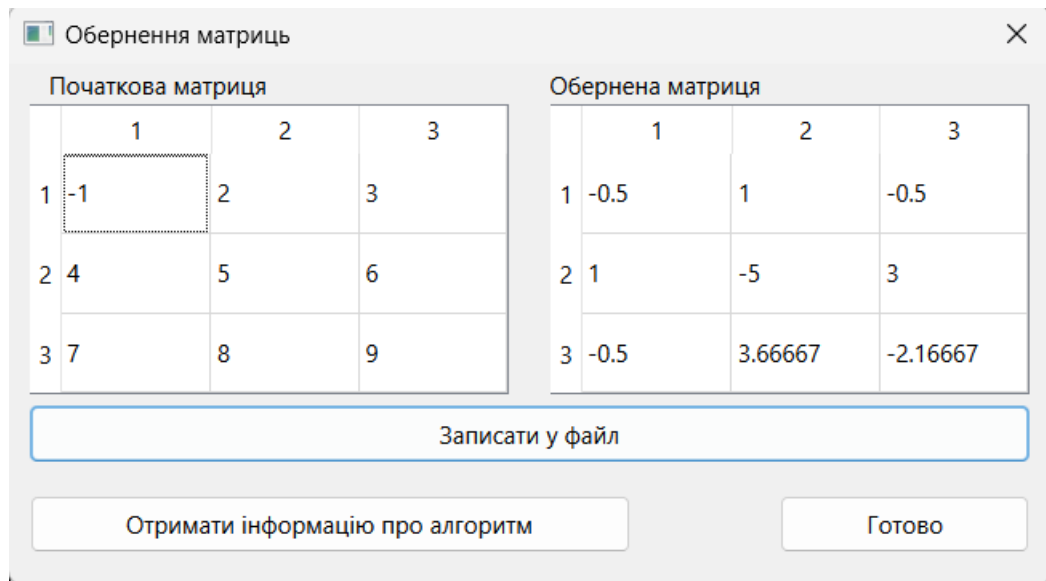


Рисунок 6.9 – Вікно показу введеної вручну і оберненої до неї матриць

6.2 Формат вхідних та вихідних даних

Користувач обирає розмір матриці від 1 до 10, один із двох способів створення матриці та один із двох методів її обернення. Якщо було обрано спосіб створення матриці вручну, то користувач має ввести n^2 (n – обрана розмірність матриці) дійсних чисел від, що не менше ніж $-1e5$ і не більше $1e5$. Також знаків після коми має бути не більше ніж 15.

Результатом програми є обернена обраним методом матриця.

6.3 Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.1.

Таблиця 6.1 – Системні вимоги програмного забезпечення

	Мінімальні	Рекомендовані
Операційна система	Windows 10 (з останніми оновленнями)	Windows 10 (з останніми оновленнями)
Процесор	Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz	Intel® Pentium® D або AMD Athlon™ 64 X2
Оперативна пам'ять	1 GB RAM	4 GB RAM
Відеоадаптер	Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог)	
Дисплей	1024x768	1920x1080 або краще
Прилади введення	Клавіатура, комп'ютерна миша	
Додаткове програмне забезпечення	Microsoft .Net Framework 4.5.2 або вище	

7 АНАЛІЗ РЕЗУЛЬТАТІВ

Метою курсової роботи була розробка якісного програмного забезпечення, яке реалізує задачу з обернення матриці методами окаймлення та Шульца.

Під час тестування програми критичних ситуацій виявлено не було.

Для перевірки правильності знаходження оберненої матриці скористаємось онлайн-застосунком onlinemschool.com:

а) Метод окаймлення

Результат роботи методу окаймлення для матриці розмірністю 3×3 представлено на рисунку 7.1, а результат обчислення onlinemschool.com на рисунку 7.2.

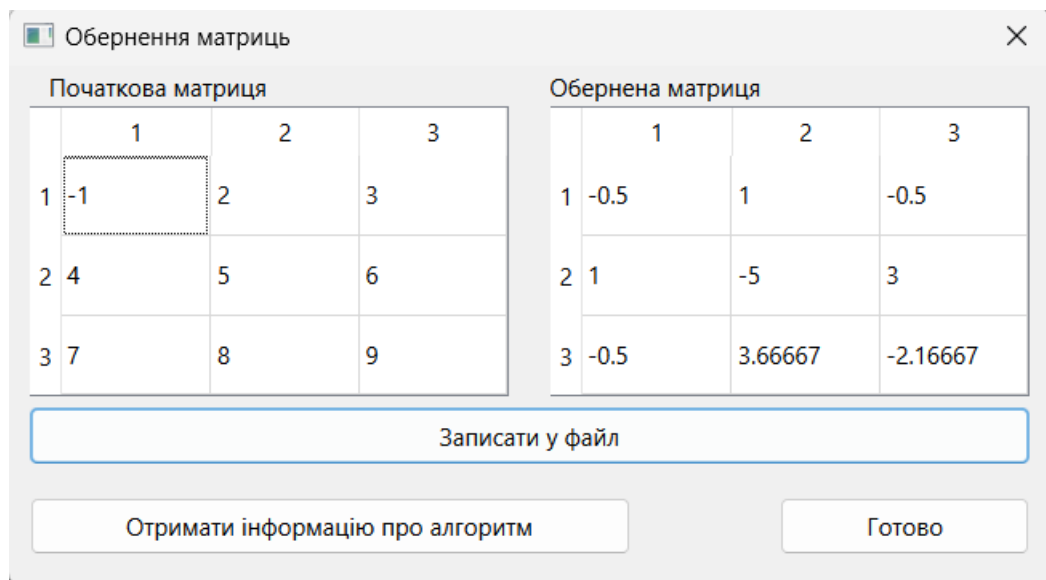


Рисунок 7.1 – Результат роботи методу окаймлення для матриці розмірністю 3×3

Відповідь:

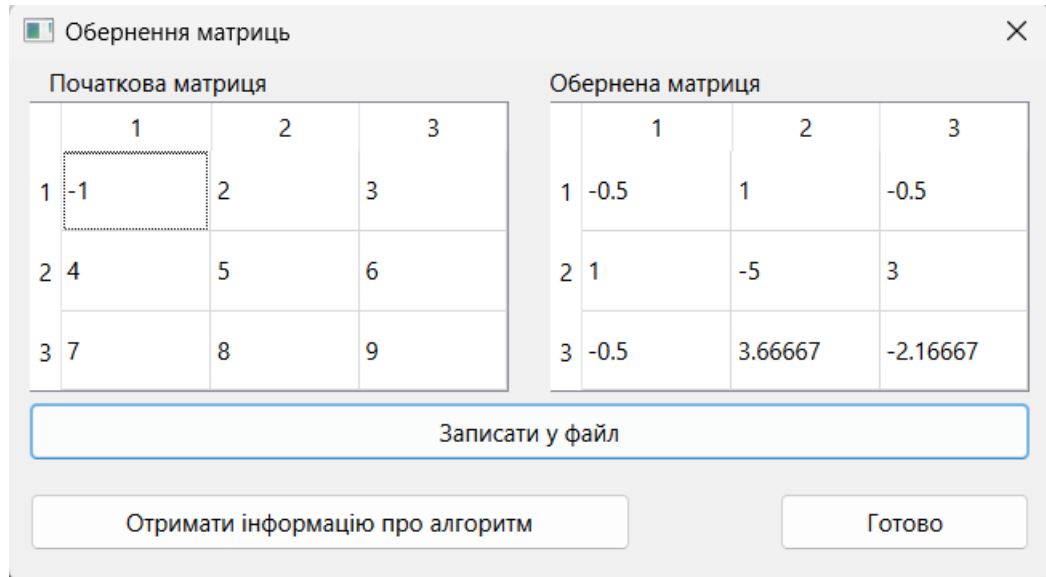
$$A^{-1} = \begin{pmatrix} -0.5 & 1 & -0.5 \\ 1 & -5 & 3 \\ -0.5 & 3\frac{2}{3} & -2\frac{1}{6} \end{pmatrix}$$

Рисунок 7.2 – Результат обчислення з сайту onlinemschool.com

Оскільки результат виконання програми збігається з результатом онлайн-застосунку, можна зробити висновок, що метод працює правильно.

б) Метод Шульца

Результат роботи методу Шульца для матриці розмірністю 3×3 представлено на рисунку 7.3.



	1	2	3
1	-1	2	3
2	4	5	6
3	7	8	9

	1	2	3
1	-0.5	1	-0.5
2	1	-5	3
3	-0.5	3.66667	-2.16667

Рисунок 7.3 – Результат роботи методу Шульца для матриці розмірністю 3×3

Як видно із попередніх рисунків, результати роботи обох методів збігаються. Результати роботи програми є достовірними.

Для проведення тестування ефективності програми було згенеровано випадковим чином матриці розмірність від 1 до 10 елементів. Результати тестування ефективності алгоритмів обернення матриці наведено в таблицях 7.1 і 7.2.

Таблиця 7.1 – Тестування ефективності методу окаймлення

Розмірність системи	Параметри тестування	Теоретична складність	Практична складність
1	Кількість ітерацій	1	1
2	Кількість ітерацій	16	46
3	Кількість ітерацій	81	196
4	Кількість ітерацій	256	539
5	Кількість ітерацій	652	1205
6	Кількість ітерацій	1296	2346
7	Кількість ітерацій	2401	4146
8	Кількість ітерацій	4096	6819
9	Кількість ітерацій	6561	10609
10	Кількість ітерацій	10000	15790

Таблиця 7.2 – Тестування ефективності методу Шульца

Розмірність системи	Параметри тестування	Теоретична складність	Практична складність
1	Кількість ітерацій	-	10
2	Кількість ітерацій	-	367
3	Кількість ітерацій	-	852
4	Кількість ітерацій	-	1630
5	Кількість ітерацій	-	5022
6	Кількість ітерацій	-	7112
7	Кількість ітерацій	-	12616
8	Кількість ітерацій	-	16470
9	Кількість ітерацій	-	26350
10	Кількість ітерацій	-	36764

На рисунку 7.4 зображено порівняльний графік для отриманих результатів.

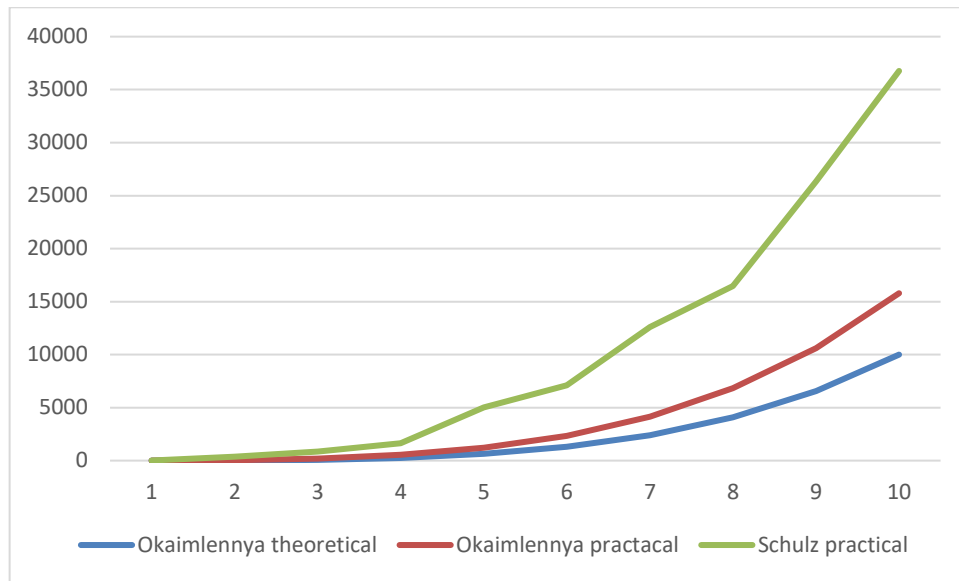


Рисунок 7.4 – Графік для представлення результатів тестування ефективності алгоритмів

За результатами тестування можна зробити такі висновки:

- а) Обидва методи дозволяють знаходити обернену матрицю.
- б) Метод окаймлення є надійнішим, оскільки не є ітераційним і має фіксовану кількість операцій залежну від розмірів.
- в) Алгоритм Шульца загалом має змінну практичну часову складність, яка залежить від того, наскільки точно було визначено початкове припущення.
- г) В середньому, складність алгоритму Шульца більша, ніж у методу окаймлення.
- г) Практична складність методу окаймлення трохи більша за теоретичну.

ВИСНОВКИ

У результаті даної курсової роботи було розроблено програмне забезпечення на мові програмування C++, яке реалізує обернення матриці методами окаймлення і Шульца. Головною метою цієї роботи було створення якісного програмного забезпечення, яке забезпечує вирішення задачі обернення матриці за допомогою обраних методів.

Курсова робота включала в себе сім розділів, які систематично розкривали постановку задачі, теоретичні відомості про методи обернення матриці, опис алгоритмів, програмного забезпечення та його графічного інтерфейсу. Для перевірки точності та ефективності програмного забезпечення було проведено тестування на різних наборах тестових даних.

Аналіз результатів тестування показав, що обидва методи (окаймлення і Шульца) здатні ефективно знаходити обернену матрицю. Проте, метод окаймлення виявився надійнішим, оскільки він не є ітераційним і має фіксовану кількість операцій, залежну від розмірів матриці. Алгоритм Шульца, у свою чергу, має змінну практичну часову складність, що залежить від точності початкового припущення. У середньому, складність алгоритму Шульца більша, ніж у методу окаймлення.

Результати роботи підтверджують доцільність розробки програмного забезпечення для обернення матриць методами окаймлення і Шульца. Розроблене програмне забезпечення може бути використане в різних областях, де важливим завданням є обернення матриць, забезпечуючи точність та ефективність обчислень.

ПЕРЕЛІК ПОСИЛАНЬ

1. Итерационный метод Шульца нахождения обратной матрицы. Математический форум Math Help Planet. URL: <https://mathhelpplanet.com/static.php?p=iteratsionnyi-metod-shultsa-nakhozhdeniya-obratnoi-matritsy>.
2. Golub G. H., Van Loan C. F. Matrix computations. Baltimor : The Johns Hopkins University Press, 1983. 723 p. URL: https://twiki.cern.ch/twiki/pub/Main/AVFedotovHowToRootTDecompQRH/Golub_VanLoan.Matr_comp_3ed.pdf.
3. Need some facts about Newton-Schulz iterative method and its application to sparse matrices. Mathematics Stack Exchange. URL: <https://math.stackexchange.com/questions/1058476/need-some-facts-about-newton-schulz-iterative-method-and-its-application-to-spar>.
4. Westlake J. A handbook of numerical matrix inversion and solution of linear equations. New York : Wiley, 1968.
5. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, Introduction to Algorithms, MIT Press, 1990.

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

Затвердив

Керівник Головченко М.М.

« » 202_ р.

Виконавець:

Студент Новиков Гліб Костянтинович

« » 202_ р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Обернення матриці»

з дисципліни:

«Основи програмування»

Київ 2023

1. *Мета:* Метою курсової роботи є розробка якісного програмного забезпечення, яке реалізує задачу з обернення матриці методами окаймлення та Шульца.

2. *Дата початку роботи:* «12» лютого 2023 р.

3. *Дата закінчення роботи:* «31» травня 2023 р.

4. *Вимоги до програмного забезпечення.*

1) Функціональні вимоги:

- можливість введення розмірності матриці за допомогою розробленої самостійно екранної форми;
- можливість введення елементів матриці за допомогою розробленої самостійно екранної форми;
- можливість генерації матриці разом з елементами;
- можливість обирати метод розв’язання задачі обернення матриці;
- розв’язання задачі обраним методом;
- можливість відображення результатів;
- графічне представлення одержаних результатів;
- збереження результатів у файл;
- можливість відображати аналітичні дані стосовно складності алгоритмів.

2) Нефункціональні вимоги:

- можливість запускати програму на базі операційних систем windows 10, windows 11;
- Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 2015 - Розробка технічної документації.

5. Стадії та етапи розробки:

- 1) Об'єктно-орієнтований аналіз предметної області задачі (до __.__.202_р.)
- 2) Об'єктно-орієнтоване проектування архітектури програмної системи (до __.__.202_р.)
- 3) Розробка програмного забезпечення (до __.__.202_р.)
- 4) Тестування розробленої програми (до __.__.202_р.)
- 5) Розробка пояснювальної записки (до __.__.202_р.).
- 6) Захист курсової роботи (до 08.06.2023 р.).

6. Порядок контролю та приймання. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ

*Тексти програмного коду програмного забезпечення
вирішення задачі обернення матриць*

(Найменування програми (документа))

CD-RW

(Вид носія даних)

28 арк, 96 Кб

(Обсяг програми (документа), арк.,

студента групи ІІІ-24 І курсу

Новикова Г. К.

main.cpp

```
#include "mainwindow.h"
#include <QApplication>
#include<QLabel>
//#include <QtWidgets>
#include <QWidget>
#include <QTableWidget>
#include <QApplication>
#include <QTableWidget>
#include <QVBoxLayout>
#include <QHeaderView>

int main(int argc, char *argv[])
{
    srand(time(nullptr));
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("Створення");
    w.show();
    return a.exec();
}
```

matrix.h

```
#pragma once
#include <iostream>
#include <vector>
#include <QTextStream>

using namespace std;
```

```

class matrix{
    vector<vector<double>> _elements;
    int _m, _n;

    matrix okaimlennya(int &comp);
    matrix schulz(int &comp);
public:
    matrix(int m,int n, double **mat);
    matrix(int m, int n, QVector<QVector<double>> vec);
    matrix(matrix& mat);
    matrix(int m,int n);

    matrix& randomise();

    int get_m();

    vector<double>& operator[](int index) ;

    friend std::ostream& operator<<(std::ostream& out, const matrix &mat);
    friend QTextStream &operator<<(QTextStream &stream, const matrix &mx);

    matrix operator*( matrix other);
    friend matrix operator*(const double& num, matrix& mat);
    friend matrix operator*(matrix& mat, const double& num);
    matrix operator/(const double& num);
    matrix operator+(const matrix& other);
    matrix operator-(const matrix& other);

    friend double det(matrix matrix_);
    friend double gauge_det(matrix matrix_);

```



```
matrix get_transpose();
double get_square_sqrt_sum();
```

```
matrix reverse(QString method, int &comp);
};
```

matrix.cpp

```
#include <cstdlib>
#include <ctime>
#include <string>
#include <iostream>
#include <vector>
#include "matrix.h"
#include <math.h>
#include <QString>
#include <QVector>
```

```
using namespace std;
```

```
matrix::matrix(int m,int n,double **mat){
    _m = m;
    _n = n;
    for (int i = 0; i < _m; i++){
        vector<double> line;
        for (int j = 0; j < _n; j++)
            line.insert(line.end(), mat[i][j]);
        _elements.insert(_elements.end(), line);
    }
}
```

```

matrix::matrix(matrix& mat) {
    _m = mat._m;
    _n = mat._n;
    for (int i = 0; i < _m; i++){
        vector<double> line;
        for (int j = 0; j < _n; j++)
            line.insert(line.end(), mat[i][j]);
        _elements.insert(_elements.end(), line);
    }
}

```

```

matrix::matrix(int m, int n) {
    _m = m;
    _n = n;
    for (int i = 0; i < _m; i++){
        vector<double> line;
        for (int j = 0; j < _n; j++)
            line.insert(line.end(), 0);
        _elements.insert(_elements.end(), line);
    }
}

```

```

matrix::matrix(int m, int n, QVector<QVector<double>> vec){
    _m = m;
    _n = n;
    for (int i = 0; i < _m; i++){
        vector<double> line;
        for (int j = 0; j < _n; j++)
            line.insert(line.end(), vec[i][j]);
        _elements.insert(_elements.end(), line);
    }
}

```

```

    }
}

```

```

matrix& matrix::randomise() {

    for (int i = 0; i < _m; i++)
        for(int j = 0; j < _n; j++)
            _elements[i][j] = (rand()/100.0 - rand()/200.0);
    return *this;
}

```

```

int matrix::get_m(){
    return _m;
}

```

```

std::ostream& operator<<(std::ostream& out, const matrix &mat){
    for (int i = 0; i<mat._m; i++){
        for (int j = 0; j < mat._n; j++){
            out<<mat._elements[i][j]<<" ";
        }
        out<<"\n";
    }
    return out;
}

```

```

QTextStream &operator<<(QTextStream &stream, const matrix &mx) {
    int columnWidth = 11; // ШИрина стовпця
    QString horizontalLine = QString("-").repeated(mx._m * columnWidth + mx._m +
1); // Горизонтальна лінія

```

```

for (int i = 0; i < mx._m; i++) {
    stream << horizontalLine << '\n';
    for (int j = 0; j < mx._m; j++) {
        stream << "|" << QString("%1").arg(mx._elements[i][j], columnWidth, ' ');
    }
    stream << "|" << '\n';
}
stream << horizontalLine << '\n';
return stream;
}

```

```

vector<double>& matrix::operator[](int index){
    return _elements[index];
}

```

```

matrix matrix::operator*(matrix other) {
    if (_n != other._m)
        return matrix(0, 0);

    matrix result(_m, other._n);

    for (int i = 0; i < result._m; i++){
        for (int j = 0; j < result._n; j++){
            result[i][j] = 0;
            for (int k = 0; k < _n; k++)
                result[i][j] += (*this)[i][k]*other[k][j];
        }
    }

    return result;
}

```

```
}
```

```
matrix operator*(const double &num, matrix &mat) {
    matrix result(mat._m, mat._n);

    for (int i = 0; i < mat._m; i++)
        for (int j = 0; j < mat._n; j++)
            result[i][j] = num*mat._elements[i][j];

    return result;
}
```

```
matrix operator*( matrix &mat, const double &num) {
    matrix result(mat._m, mat._n);

    for (int i = 0; i < mat._m; i++)
        for (int j = 0; j < mat._n; j++)
            result[i][j] = num*mat._elements[i][j];

    return result;
}
```

```
matrix matrix::operator/(const double &num) {
    matrix result(_m, _n);

    for (int i = 0; i < _m; i++)
        for (int j = 0; j < _n; j++)
            result[i][j] = _elements[i][j]/num;

    return result;
}
```

```
}
```

```
matrix matrix::operator+(const matrix &other) {
```

```
    if ((_m != other._m) or (_n != other._n))
```

```
        return matrix(0, 0);;
```

```
    matrix result(_m, _n);
```

```
    for (int i = 0; i < _m; i++)
```

```
        for (int j = 0; j < _n; j++)
```

```
            result[i][j] = (*this)[i][j] + other._elements[i][j];
```

```
    return result;
```

```
}
```

```
matrix matrix::operator-(const matrix &other) {
```

```
    if ((_m != other._m) or (_n != other._n))
```

```
        return matrix(0, 0);;
```

```
    matrix result(_m, _n);
```

```
    for (int i = 0; i < _m; i++)
```

```
        for (int j = 0; j < _n; j++)
```

```
            result[i][j] = (*this)[i][j] - other._elements[i][j];
```

```
    return result;
```

```
}
```

```
double det(matrix matrix_){
```

```
    double determinant = 0;
```

```

matrix submatrix(matrix_._m-1, matrix_._n-1);
if (matrix_._m == 1)
    return matrix_[0][0];
if (matrix_._m == 2)
    return ((matrix_[0][0] * matrix_[1][1]) - (matrix_[1][0] * matrix_[0][1]));
else {
    for (int x = 0; x < matrix_._m; x++) {
        int subi = 0;
        for (int i = 1; i < matrix_._m; i++) {
            int subj = 0;
            for (int j = 0; j < matrix_._m; j++) {
                if (j == x)
                    continue;
                submatrix[subi][subj] = matrix_[i][j];
                subj++;
            }
            subi++;
        }
        determinant = determinant + (pow(-1, x) * matrix_[0][x] * det(submatrix));
    }
}
return determinant;
}

```

```

double gause_det(matrix matrix_) {
    double determinant = 1;
    matrix copy_matrix = matrix_;

    for (int i = 0; i < matrix_._m; ++i){

```

```

    for (int j = i+1; j < matrix_._m; ++j){
        double store = copy_matrix[j][i];
        for (int k = i; k < matrix_._n; ++k) {
            copy_matrix[j][k] -= (copy_matrix[i][k]*store)/copy_matrix[i][i];
        }
    }
    determinant *= copy_matrix[i][i];
}

return determinant;
}

matrix matrix::get_transpose() {
    matrix result(_n,_m);
    for (int i =0; i<_m; i++)
        for (int j = 0; j<_n; j++)
            result[j][i] = (*this)[i][j];
    return result;
}

double matrix::get_square_sqrt_sum() {
    double suma = 0;
    for (int i = 0; i < _m; i++)
        for (int j = 0; j < _n; j++)
            suma += (*this)[i][j]*(*this)[i][j];
    return sqrt(suma);
}

matrix matrix::okaimlennya(int &comp) {
    comp++;

```



```

if (_n != _m)
    return matrix(0, 0);

if (_n == 1){
    matrix reversed(1, 1);
    reversed[0][0] = 1 / (*this)[0][0];
    return reversed;
}

matrix prev(_m-1, _n-1);
for (int i = 0; i<_m-1; i++)
    for (int j = 0; j<_n-1; j++) {
        double d = (*this)[i][j];
        prev[i][j] = d;
        comp++;
    }

matrix prev_reversed(_m - 1, _n - 1);
prev_reversed = prev.okaimlennya(comp);

matrix v(1, _n-1);
for (int i = 0; i<_n-1; i++) {
    v[0][i] = (*this)[_m - 1][i];
    comp++;
}

matrix u(_m-1, 1);
for (int i = 0; i<_n-1; i++) {
    u[i][0] = (*this)[i][_n - 1];
    comp++;
}

```

```

}

double a = (*this)[_m-1][_n-1] - (v * prev_reversed * u)[0][0];
comp+=_m*_m*_m;

matrix r = (-1/a) * prev_reversed * u;
matrix q = (-1/a) * v * prev_reversed;
matrix b = prev_reversed - (prev_reversed * u) * q;
comp+=4*_m*_m*_m;

matrix reversed(_m, _n);

for (int i = 0 ; i < _m-1; i++) {
    reversed[_m-1][i] = q[0][i];
    reversed[i][_m-1] = r[i][0];
    for (int j = 0; j < _n - 1; j++) {
        reversed[i][j] = b[i][j];
        comp++;
    }
}
reversed[_m-1][_n-1] = 1/a;

return reversed;
}

matrix matrix::schulz(int &comp) {

    const double epsilon = 1e-9;

    if (_m!=_n)

```

```

    return matrix(0,0);

    matrix ct = this->get_transpose();
    comp+=_m*_m;
    matrix c1 = (*this) * ct;
    comp+=_m*_m*_m;

    double strange_suma = c1.get_square_sqrt_sum();
    comp+=_m*_m;

    matrix reversed = ct / strange_suma;
    comp+=_m*_m;

    matrix E(_m,_n);
    for (int i = 0; i < _m; i++)
        E[i][i] = 1;
    comp += _m;

    matrix psi = E - (*this)*reversed;
    comp+=_m*_m*_m;

    while (strange_suma > epsilon){
        strange_suma = psi.get_square_sqrt_sum();
        comp+=_m*_m;

        reversed = (reversed)*(E+psi);
        comp+=_m*_m*_m;

        psi = E - ((*this)*reversed);
        comp+=_m*_m*_m;
    }

```

```

        comp++;
    }

    return reversed;
}

matrix matrix::reverse(QString method, int &comp) {
    if (method == "schulz")
        return this->schulz(comp);
    else if (method == "okaimlennya")
        return this->okaimlennya(comp);
    else
        return *this;
}

```

mainwindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{

```

Q_OBJECT

public:

MainWindow(QWidget *parent = nullptr);

~MainWindow();

private slots:

void on_pushButtonCreate_clicked();

private:

Ui::MainWindow *ui;

};

#endif // MAINWINDOW_H

mainwindow.cpp

#include "mainwindow.h"

#include "../ui_mainwindow.h"

#include "matrixinversion.h"

#include "inputwindow.h"

#include "matrix.h"

//#include <QMessageBox>

//#include <QDebug>

MainWindow::MainWindow(QWidget *parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

```
MainWindow::~MainWindow()
```

```
{
    delete ui;
}
```

```
void MainWindow::on_pushButtonCreate_clicked()
```

```
{
    QString method;
    if(ui->rbFirstMethod->isChecked())
        method = "okaimlennya";
    else
        method = "schulz";

    if(ui->rbGenerateInput->isChecked())
    {
        int size = ui->spinBox->value();
        matrix m(size, size);
        while (!det(m.randomise()));
        MatrixInversion window(this, method, m);
        window.setWindowTitle("Обернення матриць");
        window.setModal(true);
        window.exec();
    }
    else
    {
        InputWindow window(this, method, ui->spinBox->value());
    }
}
```

```

        window.setWindowTitle("Заповнення матриць");
        window.setModal(true);
        window.exec();
    }

}

inputwindow.h

#ifndef INPUTWINDOW_H
#define INPUTWINDOW_H

#include <QDialog>
#include <QLineEdit>
#include <QGridLayout>
#include <QTableWidget>
#include <QStandardItemModel>
#include <QMessageBox>
#include "matrixinversion.h"
#include "ui_inputwindow.h"
#include <QStatusBar>

#define MAX_PRECISION 10
#define MAX_VALUE 1e5

#define MIN_LINE_EDIT_WIDTH 50 // Minimum width for the QLineEdit widgets
#define MIN_LINE_EDIT_HEIGHT 30

namespace Ui {
class InputWindow;
}

```

```
class InputWindow : public QDialog
```

```
{
    Q_OBJECT
```

```
public:
```

```
    InputWindow(QWidget*, QString, int);
    ~InputWindow();
```

```
private slots:
```

```
    void handleInput(); // submitButton clicked
    void clearInput(); // clearButton clicked
```

```
private:
```

```
    Ui::InputWindow *ui;
    QVector<QVector<double>> matrix_;
    QString method;
    int size;
```

```
};
```

```
#endif // InputWindow_H
```

InputWindow.cpp

```
#include "inputwindow.h"
```

```
#include "ui_inputwindow.h"
```

```
#include <QStatusBar>
```

```
InputWindow::InputWindow(QWidget *parent, QString w_method, int w_size) :
```

```
    QDialog(parent), ui(new Ui::InputWindow), method(w_method), size(w_size)
```

```
{
```

```
    ui->setupUi(this);
```



```

this ->setFixedSize(180 + size * 45, 180 + size * 35);

ui->label->move(20 + size * 20, 0);

ui->matrixWidget->setGeometry(68, 40, 35 + size * 45, 35 + size * 35);
ui->matrixWidget->setColumnCount(size);
ui->matrixWidget->setRowCount(size);
ui->matrixWidget->horizontalHeader()-
>setSectionResizeMode(QHeaderView::Stretch);
ui->matrixWidget->verticalHeader()-
>setSectionResizeMode(QHeaderView::Stretch);

ui->clearButton->setFixedSize(80, 30);
ui->clearButton->move(26 + size * 7, 80 + size * 35);

ui->submitButton->setFixedSize(80, 30);
ui->submitButton->move(82 + size * 35, 80 + size * 35);

ui->statusBar->move(0, 120 + size * 35);
ui->statusBar->setFixedSize(180 + size * 45, 30);
ui->statusBar->setStyleSheet("color: red;");

connect(ui->submitButton,                &QPushButton::clicked,        this,
&InputWindow::handleInput);
connect(ui->clearButton,                  &QPushButton::clicked,        this,
&InputWindow::clearInput);
}

InputWindow::~~InputWindow()

```

```
{
    delete ui;
}
```

```
void InputWindow::handleInput()
```

```
{
    matrix_.clear();
    matrix_.resize(size);
```

```
    matrix* new_matrix;
```

```
    try{
        for (int row = 0; row < size; row++)
        {
            matrix_[row].resize(size);
```

```
        for (int column = 0; column < size; column++)
        {
```

```
            QTableWidgetItem* item = ui->matrixWidget->item(row, column);
            bool isValueNumber = false;
            double value = item ? item->text().toDouble(&isValueNumber) : 0.0;
```

```
            if (isValueNumber)
            {
                string valueString = item->text().StdString();
                size_t decimalPos = valueString.find('.');
                int precision = valueString.length() - decimalPos - 1;
```

```

    if (abs(value) > MAX_VALUE)
    {
        throw QString("Введено занадто велике число");
        return;
    }
    else if (precision > MAX_PRECISION)
    {
        throw QString("Введено забагато знаків після коми");
        return;
    }

    matrix_[row][column] = value;
}
else
{
    throw QString("Будь-ласка введіть ЧИСЛА");
    return;
}
}

new_matrix = new matrix(size, size, matrix_);
double determinant = det(*new_matrix);
if (fabs(determinant) < 1e-7 || isnan(determinant)) // close to zero
    throw QString("Оберненої матриці не існує");
}

catch(const QString& err){
    ui->statusBar->setText(err);
    return;
}

```

```

    }

    hide();
    MatrixInversion window(this, method, *new_matrix);
    window.setWindowTitle("Обернення матриць");
    window.setModal(true);
    window.exec();
    delete new_matrix;
}

```

```

void InputWindow::clearInput()
{
    ui->matrixWidget->clear();
    matrix_.clear();
}

```

matrixinversion.h

```

#ifndef MATRIXINVERSION_H
#define MATRIXINVERSION_H

```

```

#include <QDialog>
#include "matrix.h"

```

```

namespace Ui {
class MatrixInversion;
}

```

```

class MatrixInversion : public QDialog
{
    Q_OBJECT

```

public:

```
    explicit MatrixInversion(QWidget *parent = nullptr, QString w_method = "",
matrix w_matrix = matrix(0,0));
    ~MatrixInversion();
```

private slots:

```
    void on_pushButton_done_clicked();

    void on_pushButton_inf_clicked();

    void on_pushButton_write_1_clicked();
```

private:

```
    Ui::MatrixInversion *ui;
    QString method;
    matrix _matrix;
    int _comp;
    matrix _reversed;
};
```

```
#endif // MATRIXINVERSION_H
```

matrixinversion.cpp

```
#include "matrixinversion.h"
#include "ui_matrixinversion.h"
#include <QTableWidget>
#include "matrix.h"
#include <QStandardItemModel>
#include <QMessageBox>
#include <QFile>
#include <QFileDialog>
```

```

MatrixInversion::MatrixInversion(QWidget *parent, QString w_method, matrix
w_matrix) :
    QDialog(parent), method(w_method), _matrix(w_matrix), _reversed(w_matrix),
    ui(new Ui::MatrixInversion)
{
    ui->setupUi(this);

    this ->setFixedSize(160+_matrix.get_m()*120, 155+_matrix.get_m()*35);

    ui->source_matrix_widget->setGeometry(10,    20,    60+_matrix.get_m()*60,
40+_matrix.get_m()*35);
    ui->source_matrix_widget->setColumnCount(_matrix.get_m());
    ui->source_matrix_widget->setRowCount(_matrix.get_m());
    for (int i = 0; i < _matrix.get_m(); i++) {
        for (int j = 0; j < _matrix.get_m(); j++) {
            QTableWidgetItem *item = new
QTableWidgetItem(QString::number(_matrix[i][j]));
            item->setFlags(item->flags() & ~Qt::ItemIsEditable); // Вимкнення флага
редагованості
            ui->source_matrix_widget->setItem(i, j, item);
            //delete item;
        }
    }
    ui->source_matrix_widget->horizontalHeader()-
>setSectionResizeMode(QHeaderView::Stretch);
    ui->source_matrix_widget->verticalHeader()-
>setSectionResizeMode(QHeaderView::Stretch);

```

```

_comp = 0;
_reversed = _matrix.reverse(w_method, _comp);

ui->reversed_matrix_widget->setGeometry(90+_matrix.get_m()*60,      20,
60+_matrix.get_m()*60, 40+_matrix.get_m()*35);
ui->reversed_matrix_widget->setColumnCount(_matrix.get_m());
ui->reversed_matrix_widget->setRowCount(_matrix.get_m());
for (int i = 0; i < _matrix.get_m(); i++) {
    for (int j = 0; j < _matrix.get_m(); j++) {
        QTableWidgetItem *item = new
QTableWidgetItem(QString::number(_reversed[i][j]));
        item->setFlags(item->flags() & ~Qt::ItemIsEditable);
        ui->reversed_matrix_widget->setItem(i, j, item);
    }
}
ui->reversed_matrix_widget->horizontalHeader()-
>setSectionResizeMode(QHeaderView::Stretch);
ui->reversed_matrix_widget->verticalHeader()-
>setSectionResizeMode(QHeaderView::Stretch);

ui->label_source->move(20, 0);
ui->label_reversed->move(90+_matrix.get_m()*60, 0);

ui->pushButton_write_1->move(10, 65+_matrix.get_m()*35);
ui->pushButton_write_1->setFixedWidth(60+_matrix.get_m()*120+80);

ui->pushButton_inf->move(10, 110+_matrix.get_m()*35);
ui->pushButton_inf->setFixedWidth(210+(_matrix.get_m()-1)*45);

```

```

    ui->pushButton_done->move(145+_matrix.get_m()*80,
110+_matrix.get_m()*35);
    ui->pushButton_done->setFixedWidth(5+_matrix.get_m()*40);

    ui->statusBar->move(10, 65+_matrix.get_m()*35);
    ui->statusBar->setFixedWidth(60+_matrix.get_m()*120+80);
    ui->statusBar->setStyleSheet("color: green;");
}

```

```

MatrixInversion::~MatrixInversion()

```

```

{
    delete ui;
}

```

```

void MatrixInversion::on_pushButton_done_clicked()

```

```

{
    ui->~MatrixInversion();
    MatrixInversion::close();
}

```

```

void MatrixInversion::on_pushButton_inf_clicked()

```

```

{
    if (method == "schulz")
        QMessageBox::information(this, "Інформація про метод Шульца", "Метод
Шульца, також відомий як ітеративний метод "
                                "обернення матриці.\n"
                                "Для початку береться матриця, яка,
ймовірно, є "

```



```

        випадку обирається "
        "оберненою до початкової (у нашому
        "одинична матриця), і певними
        операціями"
        "\"підганяється\" під справжню
        обернену матрицю,"
        "поки точність не буде задовільною.\n"
        "Оскільки це ітеративний метод,
        визначити "
        "його теоретичну складність
        неможливо.\n\n"
        "Практична складність (кількість
        ітерацій): "+QString(to_string(_comp).c_str());
        else if (method == "okaimlennya")
            QMessageBox::information(this, "Інформація про метод окаймлення",
            "Метод окаймлення - метод обернення матриці, "
            "який отримує обернену матрицю із
            попередньої "
            "матриці меншої розмірності,
            шляхом \"окаймлення\" "
            "її стовпцем та рядком.\n "
            "Загалом, метод повторюється
            рекурсивно, поки "
            "розмірність матриці не
            дорівнюватиме 1, адже для "
            "такої матриці можна легко
            обчислити обернену, просто "
            "обернувши її перший і єдиний
            елемент.\n"

```

```

        "Теоретична складність  $O(n^4)$ \n"
    ""

        "Практична складність:

"+QString(to_string(_comp).c_str());
}

void MatrixInversion::on_pushButton_write_1_clicked()
{
    QString fileName = QFileDialog::getSaveFileName(this, "Save File", "", "Text Files (*.txt)"); // Діалогове вікно вибору файлу для збереження
    QFile file(fileName);

    if (file.open(QIODevice::WriteOnly | QIODevice::Text))
    {
        QTextStream stream(&file);
        stream << "Вхідна матриця\n" << _matrix;
        stream << "Визначник матриці: " +
        QString(std::to_string(det(_matrix)).c_str());

        stream << "Обернена матриця\n" << _reversed;
        stream << "\nЧасова складність методу " + method + ": " +
        QString(std::to_string(_comp).c_str());

        file.close();
        //QMessageBox::information(this, "Успіх", "Матрицю успішно записано у файл.");
        ui->statusBar->setText("Інформацію успішно записано у файл");

        ui->pushButton_write_1->hide();
    }
}

```

}

}

