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

Катедра
інформатики та програмної інженерії
(повна назва катедри, циклової комісії)

КУРСОВА РОБОТА

з дисципліни «Основи програмування»
(назва дисципліни)
на тему: «Розв'язання систем нелінійних рівнянь»

Студента 1 курсу, групи ІП-11
Лесіва Владислава Ігоровича

Спеціальности 121 «Інженерія програмного забезпечення»

Керівник Головченко Максим Миколайович
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: _____

Національна оцінка _____

Члени комісії

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ - 2022 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Катедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ІПЗ"

Курс 1 Група ІП-11 Семестр 2

ЗАВДАННЯ

на курсову роботу студента

Лесіва Владислава Ігоровича

(прізвище, ім'я, по батькові)

1. Тема роботи «Розв'язання систем нелінійних рівнянь»

2. Строк здачі студентом закінченої роботи 12.06.2022

3. Вихідні дані до роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

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

6. Дата видачі завдання 10.02.2022

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	10.02.2022	
2.	Підготовка ТЗ	02.05.2022	
3.	Пошук та вивчення літератури з питань курсової роботи	03.05.2022	
4.	Розробка сценарію роботи програми	04.05.2022	
6.	Узгодження сценарію роботи програми з керівником	04.05.2022	
5.	Розробка (вибір) алгоритму рішення задачі	04.05.2022	
6.	Узгодження алгоритму з керівником	04.05.2022	
7.	Узгодження з керівником інтерфейсу користувача	05.05.2022	
8.	Розробка програмного забезпечення	06.05.2022	
9.	Налагодження розрахункової частини програми	06.05.2022	
10.	Розробка та налагодження інтерфейсної частини програми	07.05.2022	
11.	Узгодження з керівником набору тестів для контрольного прикладу	25.05.2022	
12.	Тестування програми	26.05.2022	
13.	Підготовка пояснювальної записки	05.06.2022	
14.	Здача курсової роботи на перевірку	12.06.2022	
15.	Захист курсової роботи	15.06.2022	

Студент _____

(підпис)

Керівник _____ Головченко М. М.

(підпис) (прізвище, ім'я, по батькові)

"___" _____20__р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 90 сторінок, 26 рисунків, 21 таблиця, 2 посилання.

Об’єкт дослідження: задача розв’язання систем нелінійних рівнянь.

Мета роботи: дослідження методів розв’язання систем нелінійних рівнянь, створення програмного забезпечення для ефективного розв’язання таких систем.

Вивчено методи розв’язання систем нелінійних рівнянь: метод простих ітерацій (Якобі) та метод Гауса-Зейделя. Приведені змістовні постановки задач, їх індивідуальні математичні моделі, а також описано детальний процес розв’язання кожної з них.

Виконана програмна реалізація ітеративного алгоритму розв’язання систем нелінійних рівнянь методами Якобі та Гауса-Зейделя.

ЗМІСТ

ВСТУП

Дана робота присвячена розробці програмного забезпечення для розв’язання систем нелінійних рівнянь з використанням об’єктно-орієнтованого програмування. Задача програмного забезпечення полягає в текстовому та графічному відображенні розв’язку системи нелінійних рівнянь розмірности 2 та запису його у файл за потреби.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що буде знаходити рішення для заданої системи нелінійних рівнянь наступними методами:

а) метод простої ітерації (Якобі);

б) метод Гауса-Зейделя;

Вхідними даними для даної роботи є обраний вид системи, обраний метод розв’язання, мінімальна точність обчислення і сама система з двох нелінійних рівнянь, яка задана у відповідному вигляді:

1) Для системи алгебраїчних нелінійних рівнянь:

$$\begin{cases} a{x_1}^5 - b{x_2}^2 + c = 0 \\ d{x_1}^3 + e{x_1}^2 + f{x_2}^3 = 0 \end{cases};$$

2) Для системи тригонометричних нелінійних рівнянь:

$$\begin{cases} a(\cot x_1)^3 - b\sin(cx_2) = 0 \\ d\cos x_1 - e\tan x_2 = 0 \end{cases};$$

3) Для системи трансцендентних нелінійних рівнянь:

$$\begin{cases} a\log_2 x_1 - b\cos x_2 - c = 0 \\ d2^{x_1} + e\log_2 x_2 = 0 \end{cases};$$

де a, b, c, d, e, f – коефіцієнти, x_1, x_2 – шукані значення (рішення системи). Програмне забезпечення повинно обробляти коефіцієнти для систем нелінійних рівнянь розмірности 2.

Вихідними даними для даної роботи є два дійсні числа, що є розв’язками даної системи, які виводяться на екран, а також точність обчислення знайдених розв’язків. Програмне забезпечення повинно виводити графік системи розмірности 2 у разі існування її розв’язку. Якщо система не має розв’язків або їх нескінченна кількість, то програма повинна видати відповідне повідомлення.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Систему з 2 нелінійних рівнянь з 2 невідомими можна задати наступним чином:

$$f_k(x_1, x_2) = 0, \qquad k = 1, 2; \tag{2.1}$$

де f_k – задане рівняння, x_1, x_2 – розв'язки системи.

Якщо система має розв'язок, то його можна знайти одним із наступних методів.

2.1. Метод простих ітерацій (Якобі)

Сутність методу Якобі полягає в тому, що система рівнянь 2.1 перетворюється до еквівалентного вигляду [1]:

$$x_i = g_i(x_1, x_2), i = 1, 2;$$

Виберімо початкове наближення ${x_i}^{(0)}$

Тоді наступні наближення отримаємо за формулою:

$${x_i}^{(k+1)} = g_i\left({x_1}^{(k)}, {x_2}^{(k)}\right)$$

Позначимо через M_S матрицю Якобі.

$$M_s = \begin{pmatrix} \dfrac{dg_1\,({x_1}^{(s)}, {x_2}^{(s)})}{dx_1} & \dfrac{dg_1\,({x_1}^{(s)}, {x_2}^{(s)})}{dx_2} \\ \dfrac{dg_2\,({x_1}^{(s)}, {x_2}^{(s)})}{dx_1} & \dfrac{dg_2\,({x_1}^{(s)}, {x_2}^{(s)})}{dx_2} \end{pmatrix}$$

Тоді достатня умова збіжности ітераційного процесу буде така:

$$M_s * M_{s-1} * \ldots * M_0 \to \infty \text{ для } s \to \infty$$

Умова завершення ітераційного процесу Якобі при досягненні точности ε у спрощеній формі має вигляд:

$$\max_i \left\| {x_i}^{(k+1)} - {x_i}^{(k)} \right\| \le \varepsilon$$

2.2. Метод Гауса-Зейделя

Сутність методу Гауса-Зейделя полягає в тому, що система рівнянь 2.1 перетворюється до еквівалентного вигляду [2]:

$$x_i = g_i(x_1, x_2), i = 1, 2;$$

Виберімо початкове наближення ${x_i}^{(0)}$

Тоді наступні наближення отримаємо за формулами:

$${x_1}^{(k+1)} = g_1\left({x_1}^{(k)}, {x_2}^{(k)}\right)$$

$${x_2}^{(k+1)} = g_2\left({x_1}^{(k+1)}, {x_2}^{(k)}\right)$$

Позначимо через M_S матрицю Якобі.

$$M_s = \begin{pmatrix} \dfrac{dg_1\left({x_1}^{(s)}, {x_2}^{(s)}\right)}{dx_1} & \dfrac{dg_1\left({x_1}^{(s)}, {x_2}^{(s)}\right)}{dx_2} \\ \dfrac{dg_2\left({x_1}^{(s)}, {x_2}^{(s)}\right)}{dx_1} & \dfrac{dg_2\left({x_1}^{(s)}, {x_2}^{(s)}\right)}{dx_2} \end{pmatrix}$$

Тоді достатня умова збіжности ітераційного процесу буде така:

$$M_s * M_{s-1} * \ldots * M_0 \to \infty \text{ для } s \to \infty$$

Умова завершення ітераційного процесу Якобі при досягнені точности ε у спрощеній формі має вигляд:

$$\max_i \left\| {x_i}^{(k+1)} - {x_i}^{(k)} \right\| \leq \varepsilon$$

3 ОПИС АЛГОРИТМІВ

Перелік всіх основних змінних та їхнє призначення наведено в таблиці 3.1.

Таблиця 3.1 – Основні змінні та їхні призначення

Змінна	Призначення
ε	Точність, з якою необхідно знайти корені розв’язків
$par1$	Масив коефіцієнтів першого рівняння
$par2$	Масив коефіцієнтів другого рівняння
$precision$	Обчислювальна точність
k	Кількість ітерацій
boo	Ознака збіжности методу (false – метод не сходиться, true – метод сходиться)

3.1 Загальний алгоритм

1. ПОЧАТОК
2. Зчитати обраний вигляд системи.
3. Зчитати метод розв’язання системи.
4. Зчитати точність обчислення ε.
5. Зчитати початкове наближення для змінних розв’язків.
6. Зчитати масиви коефіцієнтів першого та другого рівнянь:
   - 6.1. Цикл проходу по полях першого рівняння:
     - 6.1.1. ЯКЩО поточний елемент у полі – вірно записане число, ТО записати його в відповідну комірку $par1$. ІНАКШЕ видати повідомлення про помилку та перейти до пункту 12.
   - 6.2. Цикл проходу по полях другого рівняння:

6.2.1. ЯКЩО поточний елемент у полі – вірно записане число, ТО записати його в відповідну комірку $par2$. ІНАКШЕ видати повідомлення про помилку та перейти до пункту 12.

7. Обробити дані на предмет окремих випадків коефіцієнтів і за такої умови розв'язати систему.
8. ЯКЩО не мали місце окремі випадки, ТО обробити дані згідно алгоритму обраного методу (пункт 3.2)
9. Побудувати та вивести графік системи.
10. Вивести рішення системи.
11. Записати систему та її рішення у файл.
12. КІНЕЦЬ

3.2 Алгоритм розв'язку обраним методом

1. ПОЧАТОК
2. Присвоюємо початкові значення для масивів розв'язків, в кожному з яких перший елемент – попередній розв'язок, другий – розв'язок шуканий на даному етапі.
3. Обчислюємо початковий визначник матриці Якобі (**convergence)**.
4. ЦИКЛ поки поточна точність більша за необхідну ($precision > \varepsilon$) і кількість ітерацій не перевищує 200 ($k <= 200$):
   4.1. Відповідно до обраного методу й типу рівнянь знаходимо розв'язок ${x_1}^k$ на даному етапі, підставляючи раніше знайдені розв'язки.
   4.2. Знаходимо розв'язок ${x_2}^k$ на даному етапі, підставляючи попередньо знайдений ${x_2}^{k-1}$ та залежно від методу попередній ${x_1}^{k-1}$ або поточний ${x_1}^k$.
      4.2.1. ЯКЩО обраний метод Якобі, підставляємо для

знаходження розв’язок $x_1{}^{k-1}$, знайдений у попередній ітерації. ІНАКШЕ підставляємо розв’язок, знайдений у поточній ітерації $x_1{}^{k}$.

4.3. Обчислюємо точність на даній ітерації $precision = \max(|x_1[1] - x_1[0]|, |x_2[1] - x_2[0]|)$.

4.4. На місце попередньо знайдених елементів присвоюємо елементи, знайдені на поточній ітерації.

4.5. Обчислюємо визначник матриці Якобі і перевіряємо процес на збіжність **(convergence).**

4.5.1. ЯКЩО процес не збіжний ($!\,boo$), ТО ПЕРЕРВАТИ ЦИКЛ.

4.6. Інкрементуємо змінну кількости ітерацій.

5. Повертаємо значення знайдених розв’язків та точність розв’язання.

6. КІНЕЦЬ

3.3 Алгоритм перевірки процесу на збіжність.

1. ПОЧАТОК
2. Знаходимо визначник матриці Якобі 2x2.
   2.1. Знаходимо добуток елементів на головній діагоналі.
   2.2. Від знайденого віднімаємо добуток елементів на побічній діагоналі.
3. Домножуємо знайдений визначник на число, знайдене у попередній ітерації.
4. ЯКЩО число на поточній ітерації менше або рівне за число з попередньої ітерації, ТО ПОВЕРНУТИ: це число й True. ІНАКШЕ ПОВЕРНУТИ це число й False.
5. КІНЕЦЬ.

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Діаграма класів програмного забезпечення.

Діаграма класів розробленого програмного забезпечення наведена на рисунку 4.1.

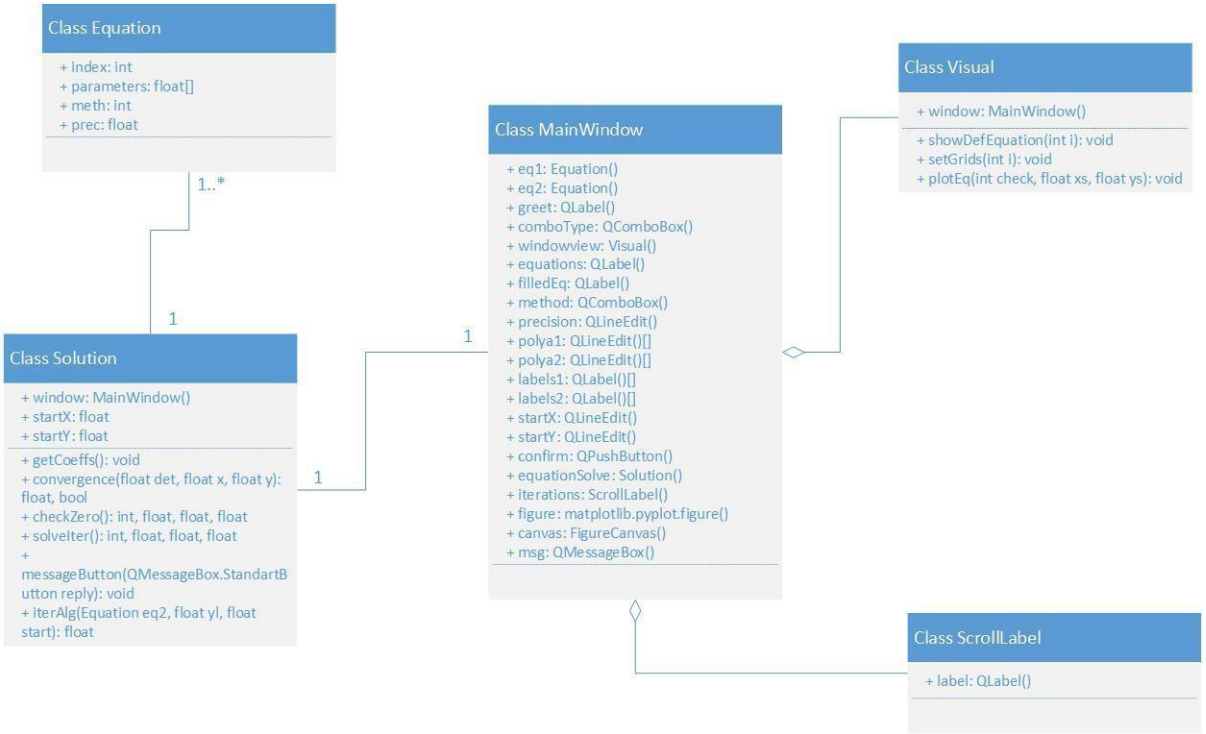


Рисунок 4.1 – Діаграма класів

4.2 Опис методів частин програмного забезпечення

4.2.1 Стандартні методи

У таблиці 4.1 наведено стандартні методи, використані під час розробки програмного забезпечення.

Таблиця 4.1 – Стандартні методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Модуль
1	QScrollArea	setWidgetResizeable	Визначає, чи повинна QScrollArea змінювати розмір віджету перегляду.	bool resizeable: якщо True, область автоматично змінюватиме розмір віджета; False – розмір незмінний	-	PyQt6.QtCore
2	QScrollArea	setWidget	Встановлює віджет області прокручування.	QWidget w: стає дочірнім до області прокручування.	-	PyQt6.QtCore
3	QLabel	setWordWrap	Встановлює налаштування перенесення слів.	bool on: якщо True – слова переносяться, False – не переносяться	-	PyQt6.QtCore
4	QLabel	setText	Встановлює текст текстового поля	string a0: текст.	-	PyQt6.QtCore

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Модуль
5	QLabel	text	Повертає текст з текстового поля	-	String text: наявний текст	PyQt6.QtCore
6	QLayout	addWidget	Додає віджет у кінець макета.	QWidget a0: віджет, що додається; int stretch = 0: коефіцієнт розтягнення; AlignmentFlag alignment = QtAlignment(): вирівнювавння	-	PyQt6.QtCore
7	QLayout	addLayout	Встановлює макет в сітці	QLayout layout: макет, що встановлюється; int stretch: коефіцієнт розтягнення.	-	PyQt6.QtCore

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Модуль
8	QWidget	setWindowTitle	Встановлює заголовок вікна	string a0: заголовок вікна	-	PyQt6.QtCore
9	QWidget	setFixedSize	Встановлює фіксований розмір віджета	QSize a0: заданий розмір	-	PyQt6.QtCore
10	QWidget	setLayout	Встановлює заданий макет на віджет	QLayout layout: макет, що встановлюється	-	PyQt6.QtCore
11	QWidget	show	Показує віджет і його нащадків	-	-	PyQt6.QtCore
12	QWidget	hide	Приховує віджет	-	-	PyQt6.QtCore
13	QComboBox	addItems	Додає кожен елемент по черзі до списку вже існуючих елементів.	string[] texts: список з елементами.	-	PyQt6.QtCore

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Модуль
14	QComboBox	currentIndexChanged	Передає сигнал, якщо індекс списку змінений користувачем або програмно.	int index	-	PyQt6.QtCore
15	QComboBox	currentIndex	Повертає індекс вибраного елемента	-	int index	PyQt6.QtCore
16	QLineEdit	setPlaceholderText	Встановлює текст заповнювача для рядка редагування.	string a0: текст, що буде встановлено	-	PyQt6.QtCore
17	QLineEdit	displayText	Повертає введений текст	-	string text: введений текст	PyQt6.QtCore
18	QAbstractButton	setCheckable	Визначає, чи можна натиснути на кнопку.	bool a0: True – можна натиснути; False – не можна	-	PyQt6.QtCore

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Модуль
19	QAbstractButton	clicked	Передає сигнал, якщо кнопка натиснута.	bool checked;	-	PyQt6.QtCore
20	QObject	connect	З’єднує переданий сигнал з виконуваною функцією	Function: функція, яка буде виконуватися	-	PyQt6.QtCore
21	QMessageBox	setIcon	Встановлює піктограму вікна	QMessageBox.Icon a0: необхідна піктограма	-	PyQt6.QtCore
22	QMessageBox	setText	Встановлює текст вікна повідомлення	string a0: текст.	-	PyQt6.QtCore
23	QMessageBox	setInformativeText	Встановлює інформативний текст	string a0: текст.	-	PyQt6.QtCore
24	QMessageBox	exec	Показує повідомлення як діалогове вікно	-	-	PyQt6.QtCore

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Модуль
25	QMainWindow	setCentralWidget	Встановлює даний віджет як центральний віджет головного вікна.	QWidget widget: віджет, що встановлюється	-	PyQt6.QtCore
26	-	figure	Створює нову фігуру або активує існуючу	-	Figure figure: створена фігура	matplotlib.pyplot
27	Figure	clear	Очищує створену фігуру	-	-	matplotlib.pyplot
28	Figure	add_subplot	Додає осі до малюнка	Int nrows: кількість рядків; int ncols: кількість стовпців; int index: позиція на сітці	-	matplotlib.pyplot

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Модуль
29	-	plot	Будує графік	Float[] x, float[] y: координати	-	matplotlib.pyplot
30	-	grid	Налаштовує лінії сітки	-	-	matplotlib.pyplot
31	-	legend	Налаштовує легенду графіку	-	-	matplotlib.pyplot
32	FigureCanvasQTAgg	draw	Розташовує графік на віджеті	-	-	matplotlib.backends.backend_qt5agg
33	-	abs	Повертає модуль числа	Float x: задане число	Float a: модуль числа x	Вбудована
34	-	sin	Повертає синус числа	Float x: задане число	Float a: синус числа x	math

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Модуль
35	-	atan	Повертає арктангенс числа	Float x: задане число	Float a: арктангенс числа x	math
36	-	cos	Повертає косинус числа	Float x: задане число	Float a: косинус числа x	math

4.2.2 Користувацькі методи

У таблиці 4.2 наведено користувацькі методи, створені під час розробки програмного забезпечення.

Таблиця 4.2 – Користувацькі методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	Visual	showDefEquation	Показує загальний вигляд системи і викликає подальші дії	int i: індекс обраного типу рівнянь з випадного списку.	-

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
2	Visual	setGrids	Форматує поля для вводу і допоміжний текст для них.	int i: індекс обраного типу рівнянь з випадного списку.	-
3	Visual	plotEq	Будує графік для систем, що мають розв’язок.	int check: показує, чи система розв’язана за окремим принципом або загальним методом; float xs, float ys: розв’язки системи.	-
4	Solution	getCoeffs	Зчитує усі введені дані, виводить вигляд заповненої системи, викликає подальші дії.	-	-

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
5	Solution	convergence	Обчислює визначник матриці Якобі й перевіряє, чи процес збіжний на даній ітерації.	Float det: число на попередній ітерації, що визначає збіжність; float x, float y: наближення на даній ітерації.	Float temp (float det): число, що визначає збіжність на даній ітерації; bool b: True, якщо процес збіжний
6	Solution	checkZero	Перевіряє рівняння на можливість окремого розв’язання (наприклад, коли той чи інший коефіцієнт дорівнює 0)	-	Const int 1: показує, що рівняння розв’язалося неітераційно; float sol[0], float sol[1]: розв’язки системи; const int 0: точність, адже розв’язано неітераційно.

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
7	Solution	solveIter	Розв’язує систему обраним ітераційним методом.	-	Const int 0: показує, що система розв’язалася ітераційно; float x1[0], float x2[0]: розв’язки системи; float precision: точність розв’язання
8	Solution	messageButton	Записує розв’язок у файл.	QMessageBox. StandartButton reply: відповідь на діалогове вікно.	-
9	Solution	iterAlg	Ітераційно обчислює розв’язок кубічного рівняння, коли інший розв’язок знайдено неітераційно	Equation eq2: рівняння; float yl: другий розв’язок; float start: початкове наближення	sol: відповідь на розв’язок – або безліч “b”,або немає “0”, або є і це значення типу float

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 План тестування

У цьому пункті відбуватиметься тестування розробленого додатку, що розв’язує системи нелінійних рівнянь з двома змінними. Додаток бере на вхід вид рівнянь, коефіцієнти, точність, метод розв’язання та початкове наближення. Необхідно переконатися, що у будь-якому випадку введення даних додаток працює коректно, виводячи розв’язок на екран та показуючи графік або ж, виводячи відповідне повідомлення, аварійно не завершує свою роботу.

Відповідно до плану сформовано таку стратегію тестування:

а) Тестування правильности введених значень.

1) Тестування під час введення некоректних символів.
2) Тестування під час введення замалих та завеликих значень.
3) Тестування за відсутности введення деяких значень.

б) Тестування коректної роботи під час введення систем, коефіцієнти яких можуть дорівнювати нулю.

1) Тестування роботи програми з нульовим значенням коефіцієнта біля першої змінної у першому рівнянні.
2) Тестування роботи програми з нульовим значенням коефіцієнта біля другої змінної у другому рівнянні.
3) Тестування роботи програми з нульовими значеннями коефіцієнтів біля першої змінної в першому рівнянні й другої змінної в другому рівнянні.

в) Тестування коректности роботи методів 1, 2.

1) Перевірка коректности роботи методу 1 (Якобі).
2) Перевірка коректности роботи методу 2 (Гауса-Зейделя).

г) Тестування коректности роботи методів 1,2 з дійсними коефіцієнтами.

1) Перевірка коректности роботи методу 1 (Якобі).

2) Перевірка коректности роботи методу 2 (Гауса-Зейделя).

д) Тестування коректної роботи під час введення систем, для яких методи 1, 2 не є збіжними.

1) Перевірка правильности результату.

е) Тестування побудови графіків.

1) Перевірка коректности графіку, коли система розв’язана ітеративно.

2) Перевірка коректности графіку, коли коефіцієнти біля змінних, що виражаються, дорівнюють нулю, але розв’язок існує.

3) Перевірка виведення, коли числового розв’язку не існує.

ж) Тестування запису у файл.

1) Перевірка запису, коли обрано записати у файл.

2) Перевірка реакції програми, коли обрано не записувати у файл.

5.2 Приклади тестування

Нижче подані тести, що були виконані відповідно до стратегії тестування програми. У тесті можна побачити мету, різні вхідні дані, схему проведення, очікуваний та фактичний результат, що задовільняє очікування. Результати тестування подані в таблицях 5.1 – 5.16 та на рисунках 5.1 – 5.16.

Таблиця 5.1 - Приклад роботи програми при введенні некоректних символів

Мета тесту	Перевірити можливість введення некоректних даних
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Алгебраїчні; Метод Якобі; 0.0001; 1 e 344; r @3 4; 23; 2
Схема проведення тесту	Обрання видів рівнянь і методів, введення точности і наближень, поелементне заповнення коефіцієнтів.
Очікуваний результат	Повідомлення про помилку формату даних
Стан програми після проведення випробувань	Видано помилку «Переконайтеся, що: -введені коефіцієнти, початкові наближення та точність - це цілі або дійсні числа у форматі '0.0'».

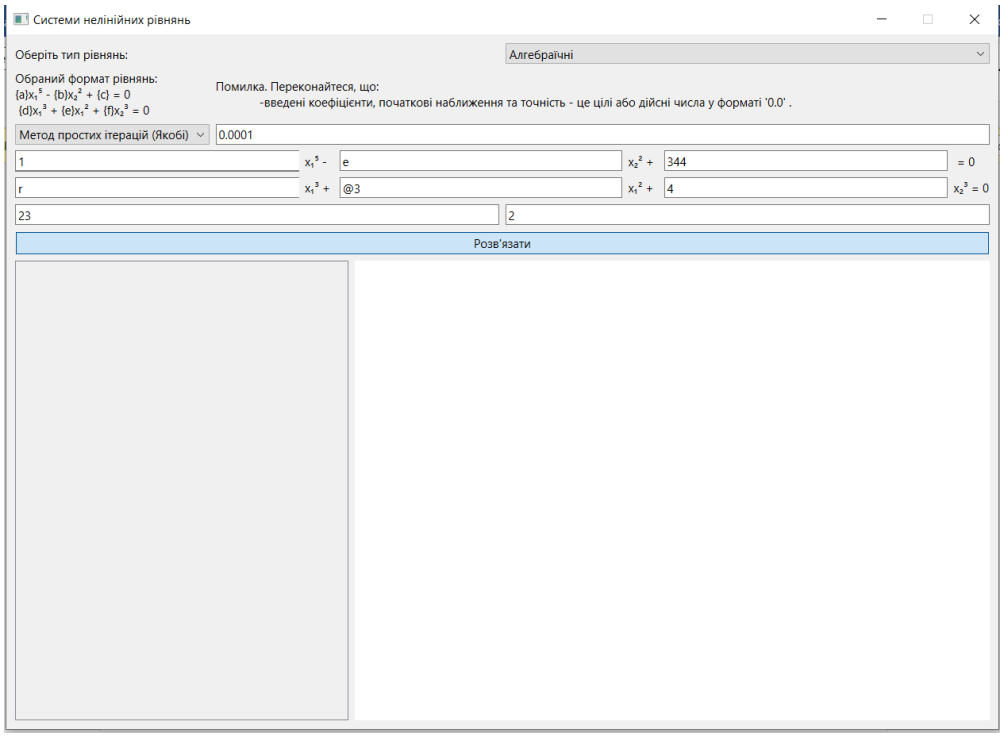


Рисунок 5.1 – Введення некоректних символів

Таблиця 5.2 - Приклад роботи програми при введенні завеликих значень.

Мета тесту	Перевірити можливість введення замалих чи завеликих значень
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Трансцендентні; Метод Якобі; 0.0001; 1323 452 3444; 32325 56765; 23453256; 2435
Схема проведення тесту	Обрання видів рівнянь і методів, введення точности і наближень, поелементне заповнення коефіцієнтів.
Очікуваний результат	Повідомлення про помилку переповнення.
Стан програми після проведення випробувань	Видано помилку «Помилка переповнення. Спробуйте зменшити значення початкового наближення».

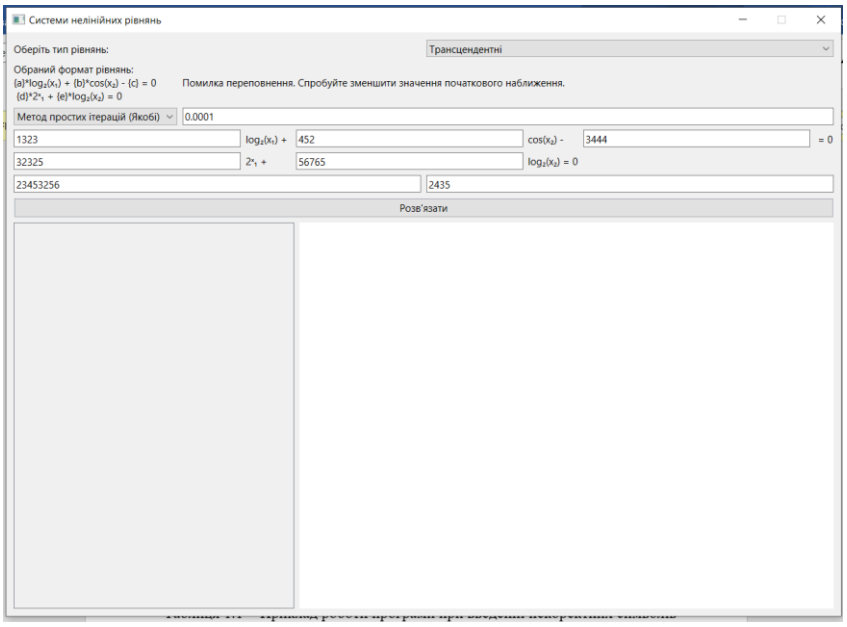


Рисунок 5.2 – Введення завеликих значень

Таблиця 5.3 - Приклад роботи програми за відсутности введення деяких значень.

Мета тесту	Перевірити можливість введення некоректних даних
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Алгебраїчні; Метод Якобі; 0.0001; 45 344; 323 7; 234; 243
Схема проведення тесту	Обрання видів рівнянь і методів, введення точности і наближень, поелементне заповнення коефіцієнтів, за винятком декількох полів.
Очікуваний результат	Повідомлення про помилку формату даних
Стан програми після проведення випробувань	Видано помилку «Переконайтеся, що: -введені коефіцієнти, початкові наближення та точність - це цілі або дійсні числа у форматі '0.0'».

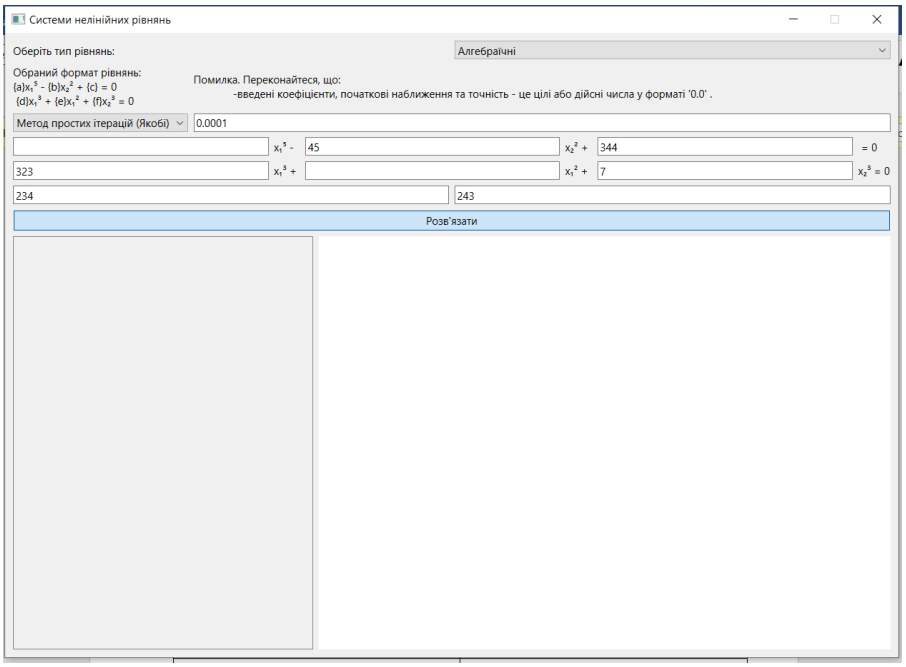


Рисунок 5.3 – Відсутні деякі значення

Таблиця 5.4 - Приклад роботи програми при нульовому значенні коефіцієнта біля першої змінної у першому рівнянні.

Мета тесту	Перевірити можливість введення нульового коефіцієнта біля змінної, що виражається.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Алгебраїчні; Метод Якобі; 0.0002; 12 2 3; 4 5 0; 12; 2;
Схема проведення тесту	Обрання видів рівнянь і методів, введення точности і наближень, поелементне заповнення коефіцієнтів з коефіцієнтом біля виразної змінної другого рівняння 0.
Очікуваний результат	Розв’язок системи або повідомлення про відсутність\безліч розв’язків
Стан програми після проведення випробувань	Видано розв’язок, позначено точку розв’язку на графіку.

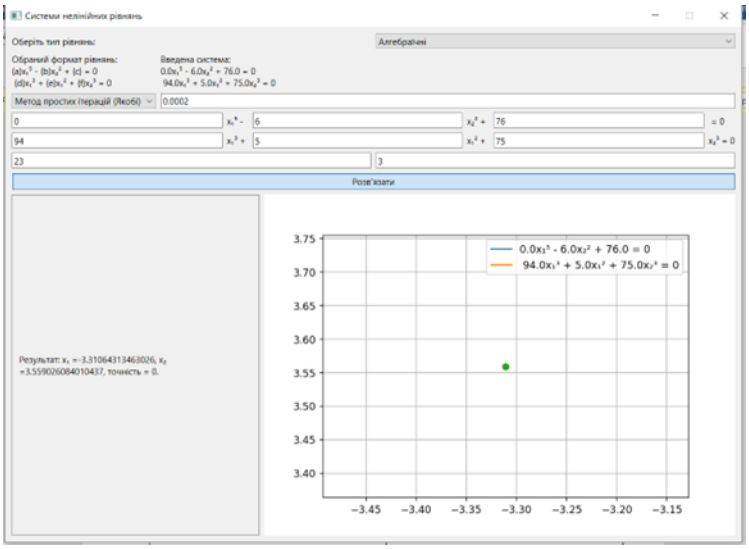


Рисунок 5.4 – Нуль біля першої змінної

Таблиця 5.5 - Приклад роботи програми при нульовому значенні коефіцієнтабіля другої змінної у другому рівнянні.

Мета тесту	Перевірити можливість введення нульового коефіцієнта біля змінної, що виражається.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Алгебраїчні; Метод Якобі; 0.0002; 0 6 76; 94 5 75; 23; 3
Схема проведення тесту	Обрання видів рівнянь і методів, введення точности і наближень, поелементне заповнення матриці коефіцієнтів з коефіцієнтом біля виразної змінної першого рівняння 0.
Очікуваний результат	Розв’язок системи або повідомлення про відсутність\безліч розв’язків.
Стан програми після проведення випробувань	Видано розв’язок, позначено точку розв’язку на графіку.

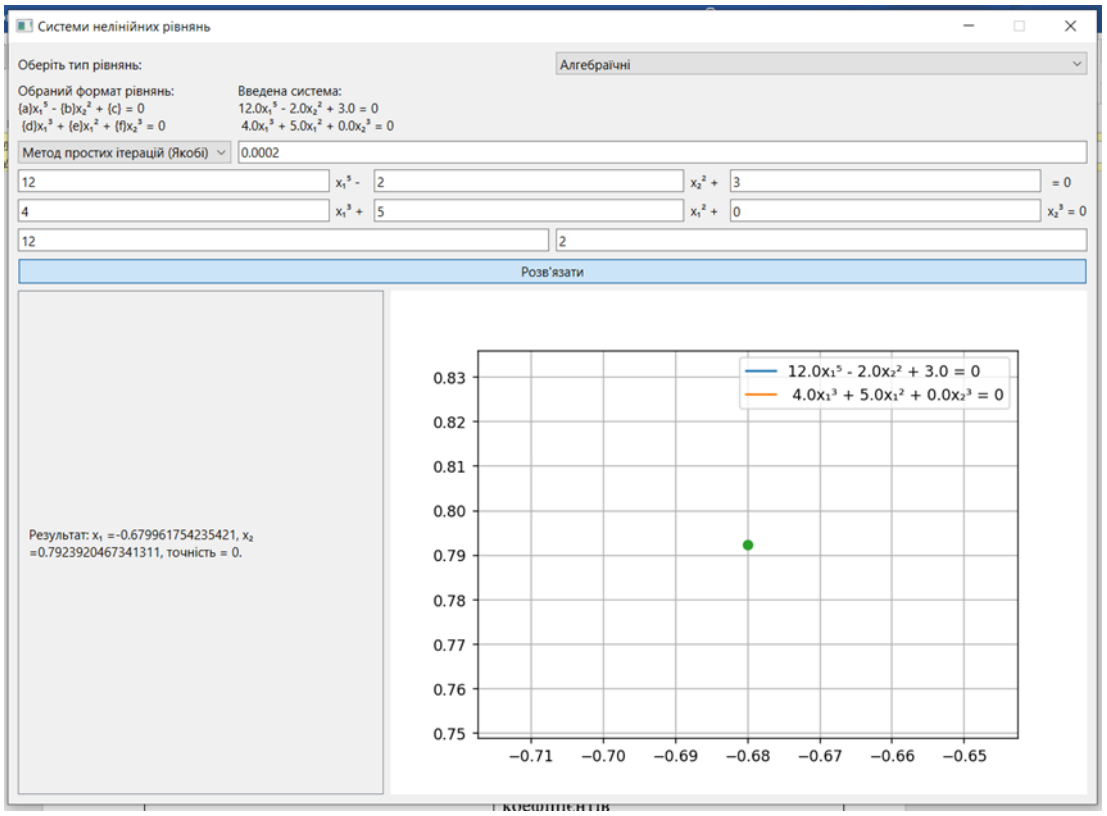


Рисунок 5.5 – Нуль біля другої змінної

Таблиця 5.6 - Приклад роботи програми при нульовому значенні коефіцієнтів біля першої змінної у першому рівнянні й другої змінної у другому рівнянні.

Мета тесту	Перевірити можливість введення нульових значень коефіцієнтів біля змінних в обоїрівняннях.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Алгебраїчні; Метод Якобі; 0.0002; 0 -6 76; 94 4 0; 23; 3
Схема проведення тесту	Поелементне заповнення коефіцієнтів
Очікуваний результат	Розв’язок системи або повідомлення про відсутність\безліч розв’язків
Стан програми після проведення випробувань	Видано повідомлення «система розв’язків не має».

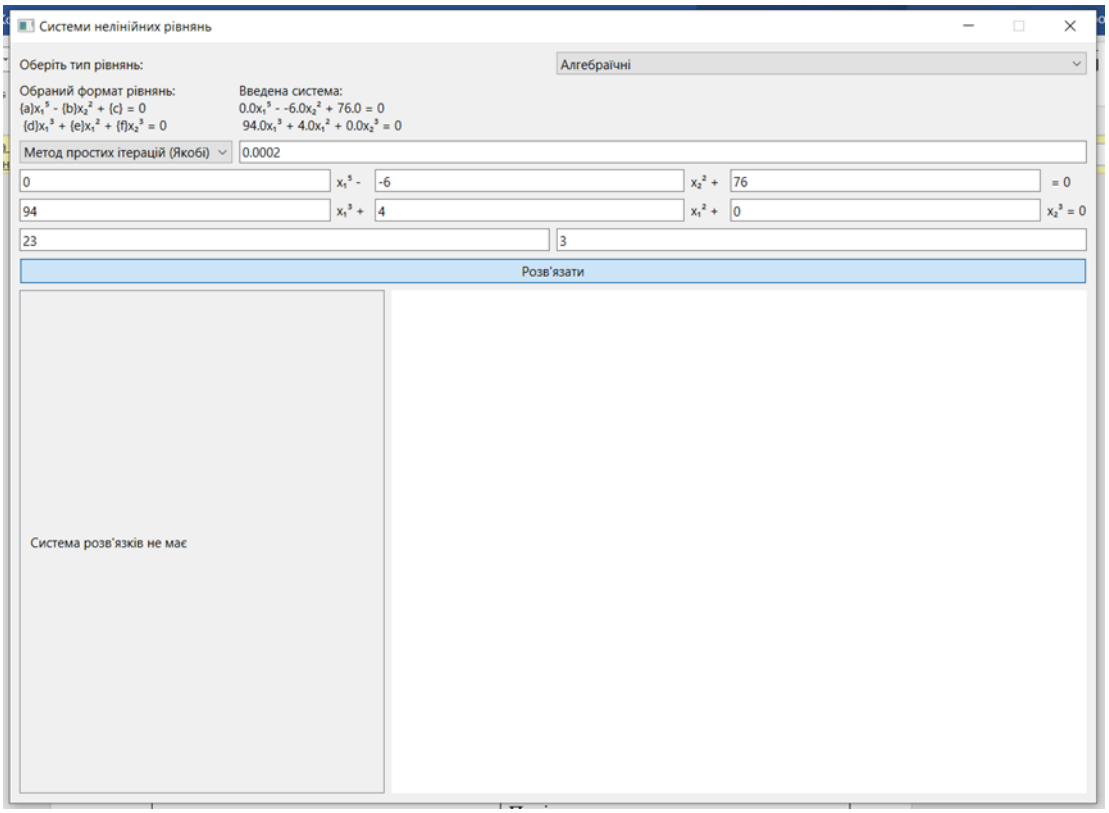


Рисунок 5.6 – Нуль біля обох змінних

Таблиця 5.7 - Приклад роботи програми під час розв’язування методом 1 (Якобі)

Мета тесту	Перевірити працездатність методу 1.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Тригонометричні; Метод Якобі; 0.0002; 113 6 76; 9 4; 23; 33
Схема проведення тесту	Поелементне заповнення коефіцієнтів
Очікуваний результат	Розв’язок системи або повідомлення про відсутність\безліч розв’язків
Стан програми після проведення випробувань	Систему розв’язано, видано ітерації й розв’язок, побудовано графік.

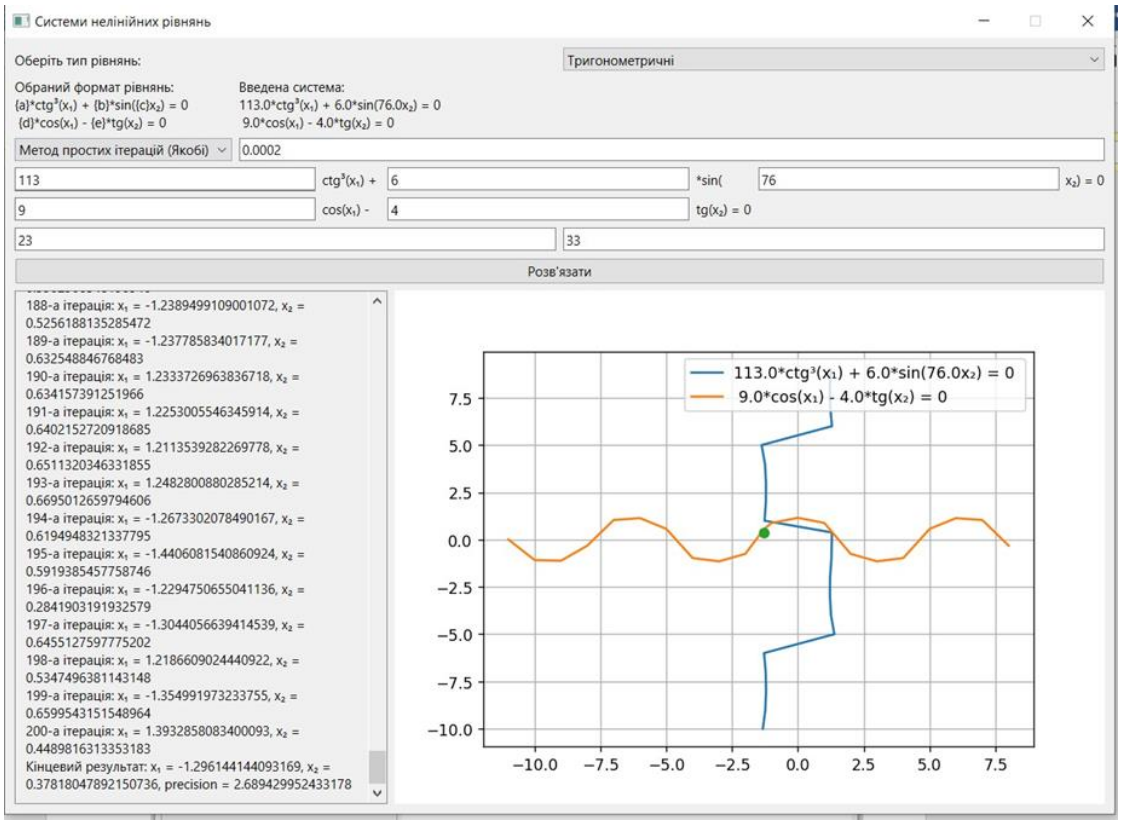


Рисунок 5.7 – Розв’язання методом Якобі

Таблиця 5.8 - Приклад роботи програми під час розв’язування методом 2 (Гауса-Зейделя)

Мета тесту	Перевірити працездатність методу 2.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Тригонометричні; Метод Гауса-Зейделя; 0.0002; 113 6 76; 9 4; 23; 33
Схема проведення тесту	Поелементне заповнення коефіцієнтів
Очікуваний результат	Розв’язок системи або повідомлення про відсутність\безліч розв’язків
Стан програми після проведення випробувань	Систему розв’язано, видано ітерації й розв’язок, побудовано графік.

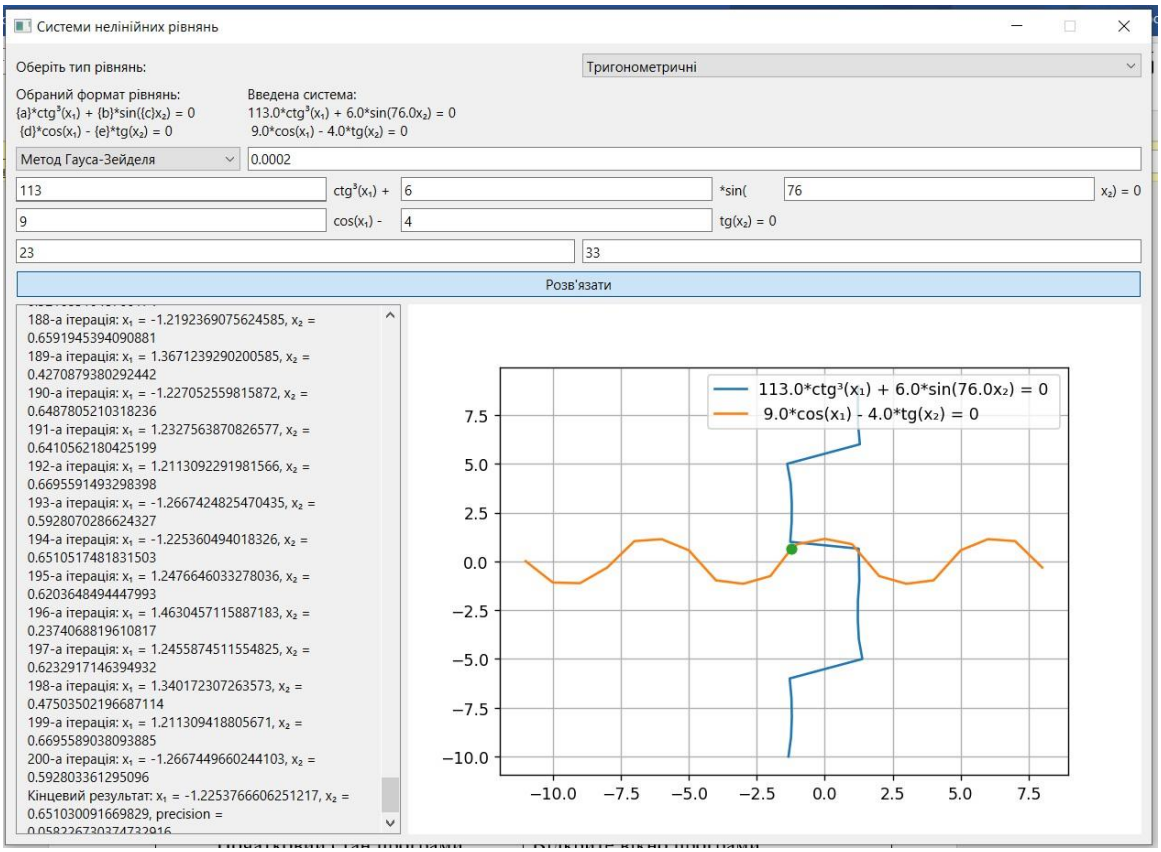


Рисунок 5.8 – Розв’язання методом Гауса-Зейделя

Таблиця 5.9 - Приклад роботи програми під час розв’язування методом 1 (Якобі)з дійсними коефіцієнтами.

Мета тесту	Перевірити працездатність методу 1 з дійсними коефіцієнтами.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Трансцендентні; Метод Якобі; 0.0002; 11.5 3.243 7.8; 9.2 4.3; 2.3; 6.5
Схема проведення тесту	Поелементне заповнення коефіцієнтів
Очікуваний результат	Розв’язок системи або повідомлення про відсутність\безліч розв’язків
Стан програми після проведення випробувань	Систему розв’язано, видано ітерації й розв’язок, побудовано графік.

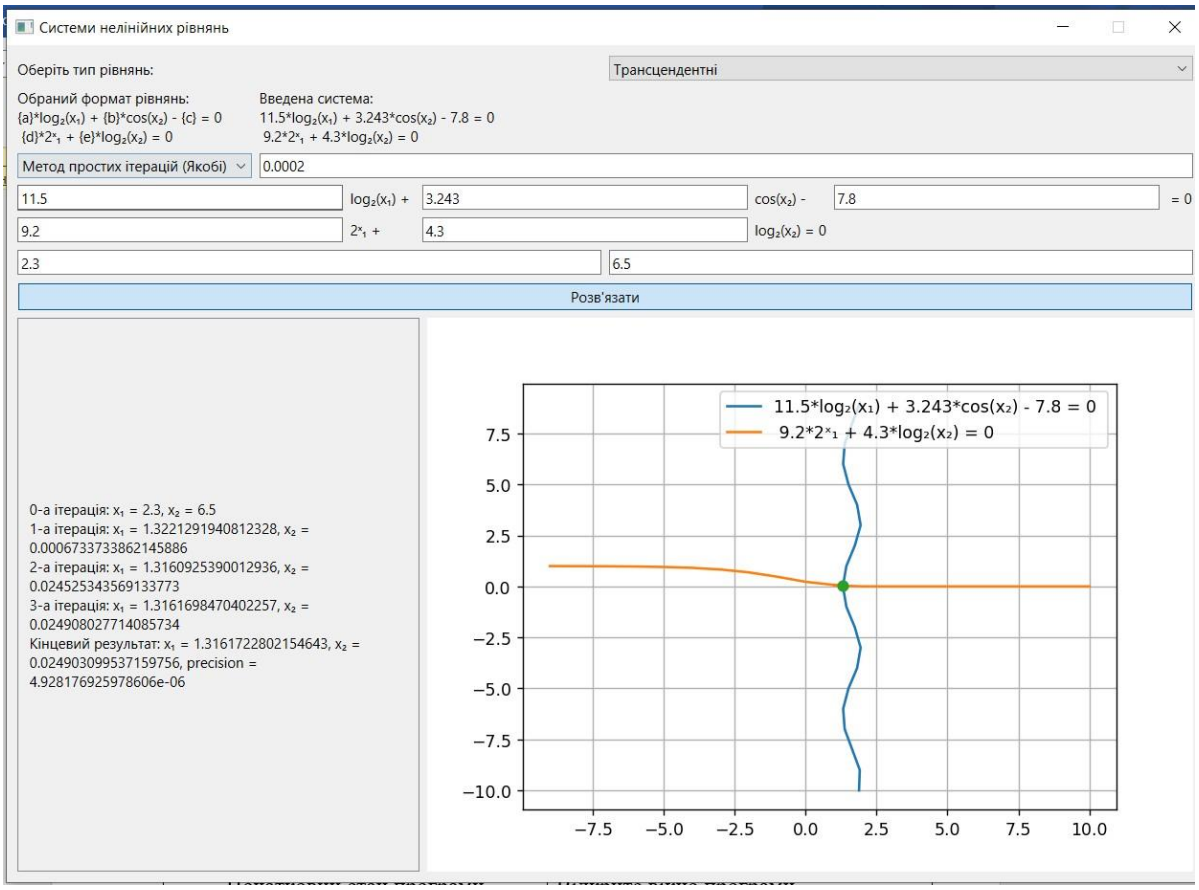


Рисунок 5.9 – Розв’язання методом Якобі з дійсними коефіцієнтами

Таблиця 5.10 - Приклад роботи програми під час розв’язування методом 2 (Гауса-Зейделя) з дійсними коефіцієнтами.

Мета тесту	Перевірити працездатність методу 2 з дійсними коефіцієнтами.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Трансцендентні; Метод Гауса-Зейделя; 0.0002; 11.5 3.243 7.8; 9.2 4.3; 2.3; 6.5
Схема проведення тесту	Поелементне заповнення коефіцієнтів
Очікуваний результат	Розв’язок системи або повідомлення про відсутність\безліч розв’язків
Стан програми після проведення випробувань	Систему розв’язано, видано ітерації й розв’язок, побудовано графік.

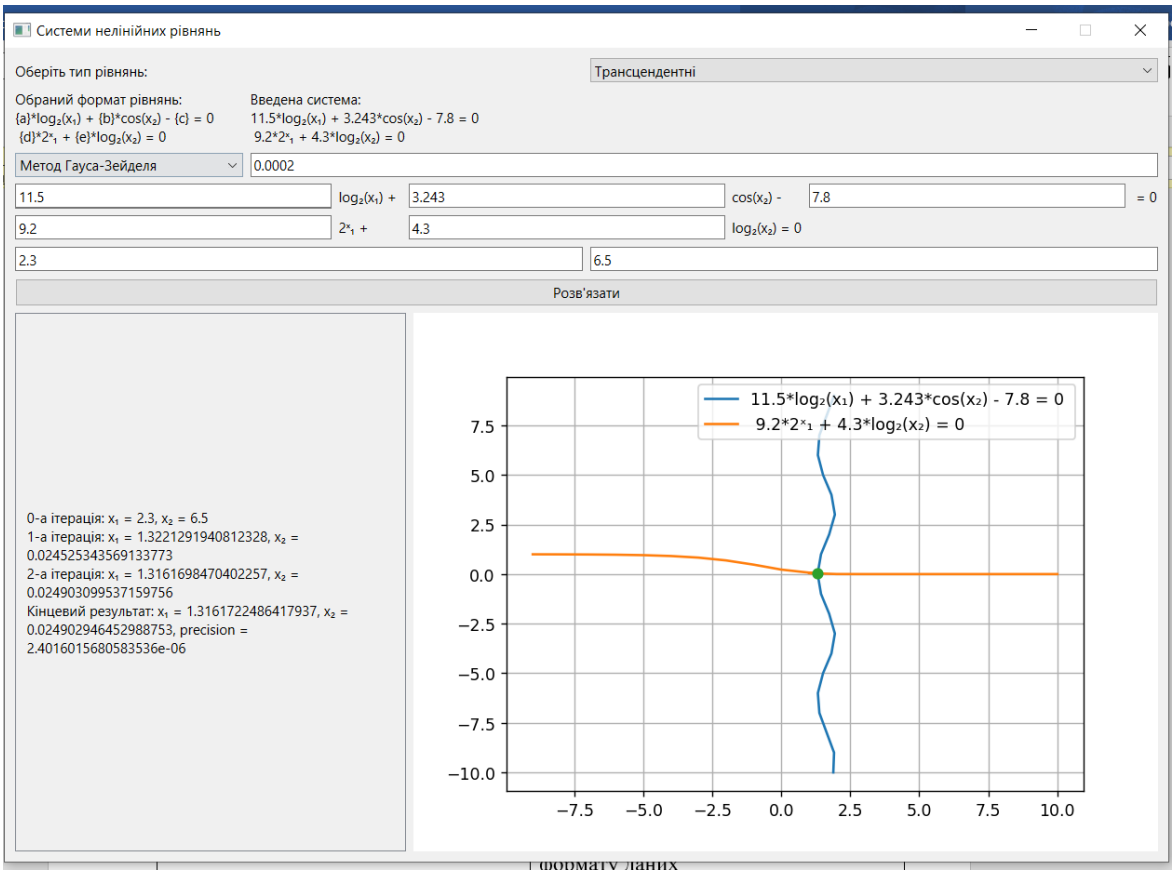


Рисунок 5.10 – Розв’язання методом Гауса-Зейделя з дійсними коефіцієнтами

Таблиця 5.11 - Приклад роботи програми під час введення систем, для яких методи не є збіжними.

Мета тесту	Перевірити працездатність програми, коли методи не є збіжними
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Тригонометричні; Метод Якобі; 0.0002; 11 32 7.8; 9.2 4.3; 23; 6.5
Схема проведення тесту	Поелементне заповнення коефіцієнтів
Очікуваний результат	Розв’язок системи або повідомлення про відсутність\безліч розв’язків
Стан програми після проведення випробувань	Видано повідомлення «Процес не збіжний, неможливо знайти розв’язок»

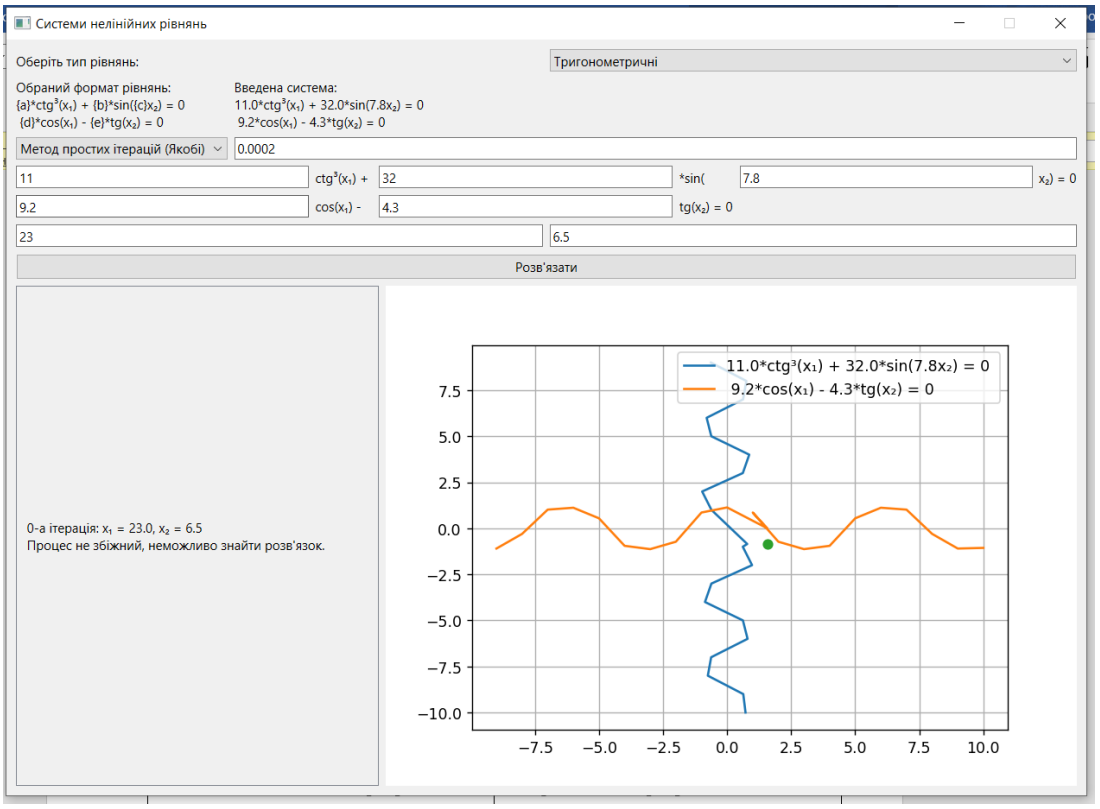


Рисунок 5.11 – Метод не збіжний

Таблиця 5.12 - Приклад роботи програми, коли графік будується після ітеративного розв’язання.

Мета тесту	Перевірити правильність побудови графіку, коли система розв’язана ітеративно.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Тригонометричні; Метод Якобі; 0.0002; 113 6 76; 9 4; 23; 33
Схема проведення тесту	Поелементне заповнення коефіцієнтів
Очікуваний результат	Побудований коректний графік відповідно до розв’язку – повноцінний графік.
Стан програми після проведення випробувань	Систему розв’язано, видано ітерації й розв’язок, побудовано коректний графік.

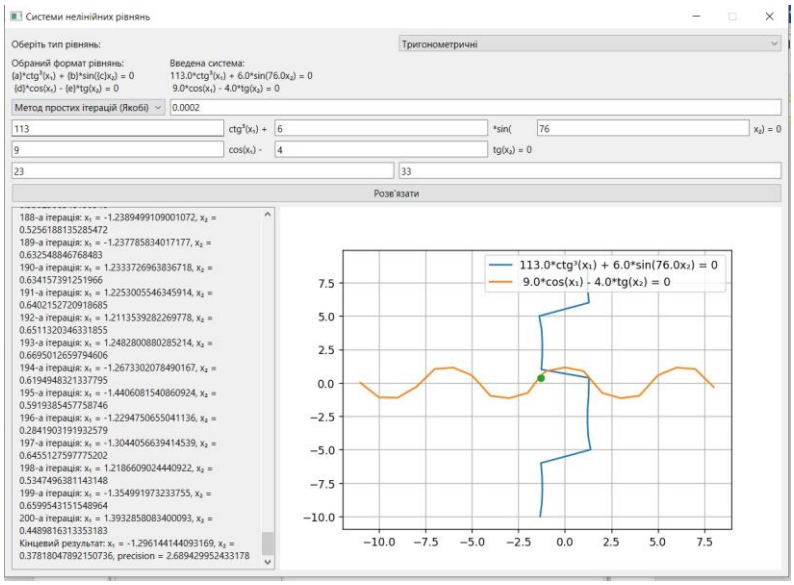


Рисунок 5.12 – Графік після ітеративного розв’язання

Таблиця 5.13 - Приклад роботи програми, коли графік будується післянеітеративного розв’язання

Мета тесту	Перевірити правильність побудови графіку, коли система розв’язана ітеративно.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Алгебраїчні; Метод Якобі; 0.0002; 0 6 76; 94 5 75; 23; 3
Схема проведення тесту	Обрання видів рівнянь і методів, введення точности і наближень, поелементне заповнення матриці коефіцієнтів з коефіцієнтом біля виразної змінної першого рівняння 0.
Очікуваний результат	Побудований коректний графік відповідно до розв’язку – точка на графіку.
Стан програми після проведення випробувань	Видано розв’язок, позначено точку розв’язку на графіку.

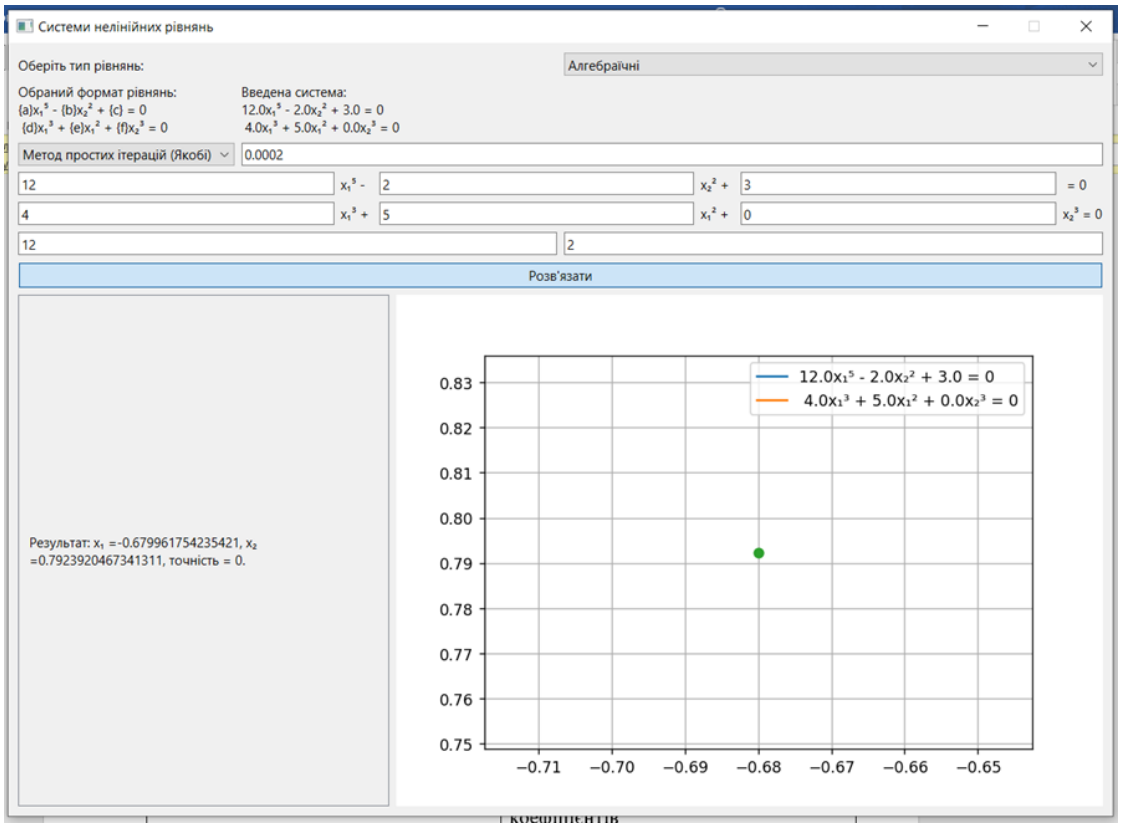


Рисунок 5.13 – Графік після неітеративного розв’язання

Таблиця 5.14 - Приклад роботи програми, коли графік будується за відсутности розв’язку.

Мета тесту	Перевірити правильність графіку, коли розв’язку не існує.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Алгебраїчні; Метод Якобі; 0.0002; 0 -6 76; 94 4 0; 23; 3
Схема проведення тесту	Поелементне заповнення коефіцієнтів
Очікуваний результат	Відсутність графіку.
Стан програми після проведення випробувань	Видано повідомлення «система розв’язків не має», графік відсутній.

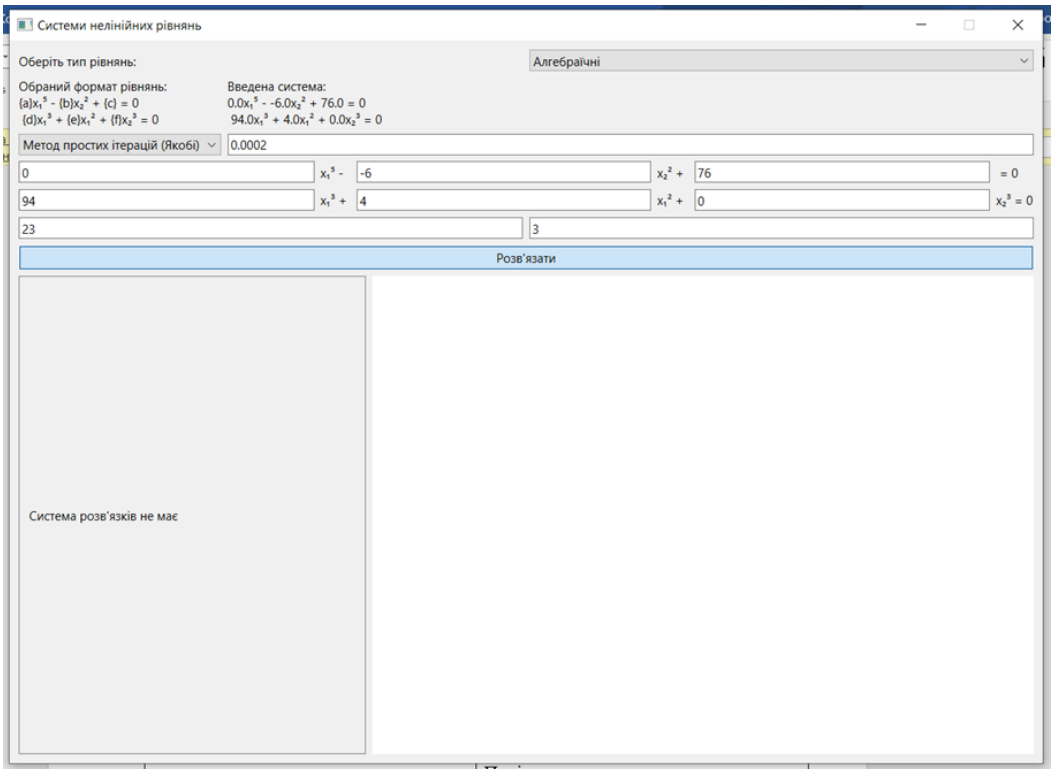


Рисунок 5.14 – Графіку немає

Таблиця 5.15 - Приклад роботи програми, коли обрано записати розв’язок у файл.

Мета тесту	Перевірити правильність запису розв’язку у файл.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Алгебраїчні; Метод Якобі; 0.0002; 1 2 3; 4 5 6; 1; 1;
Схема проведення тесту	Поелементне заповнення коефіцієнтів
Очікуваний результат	Записано систему, ітерації та розв’язок у файл.
Стан програми після проведення випробувань	Записано систему, ітерації та розв’язок у файл.

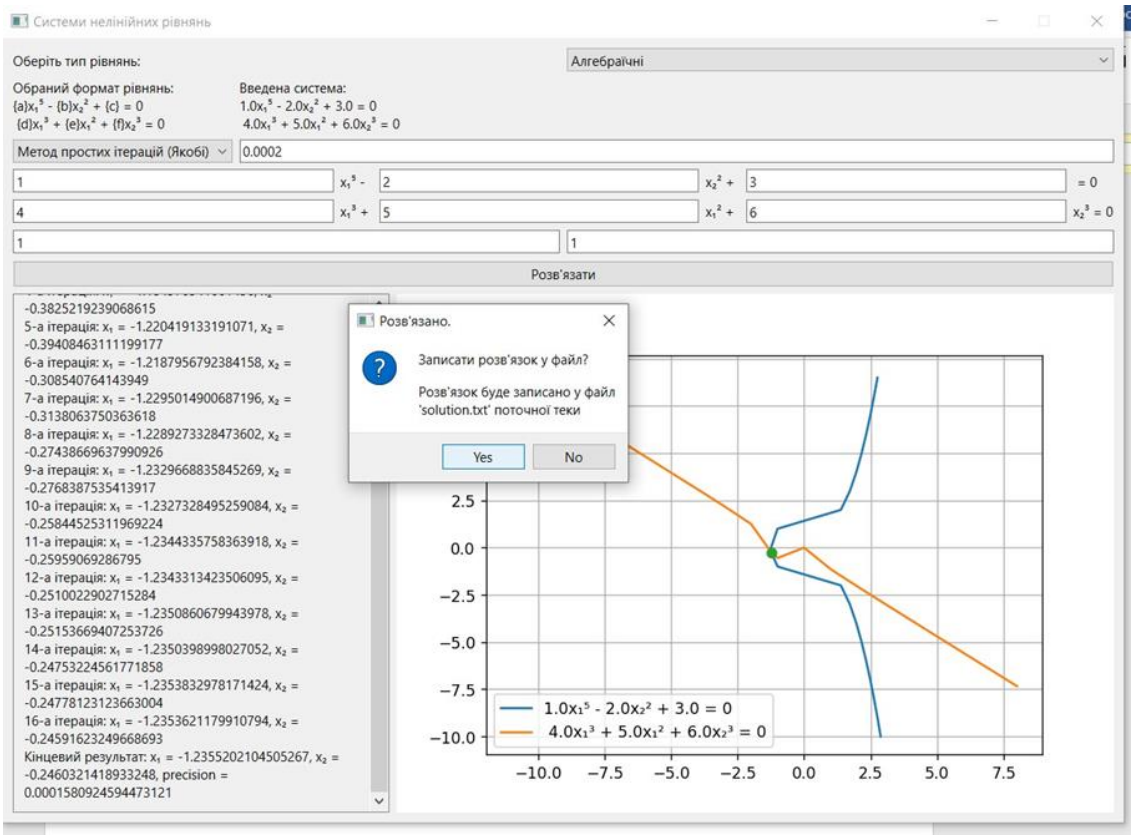


Рисунок 5.15 – Вікно запису у файл

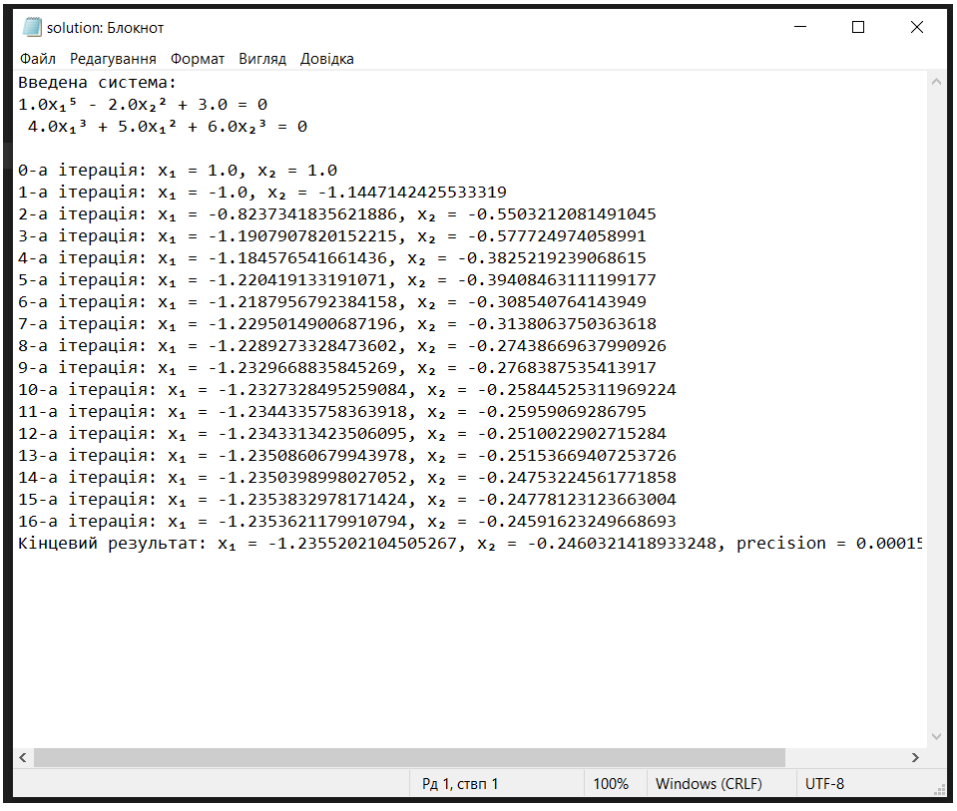


Рисунок 5.16 – Файл після обраного запису

Таблиця 5.16 - Приклад роботи програми, коли обрано не записувати розв’язок у файл.

Мета тесту	Перевірити правильність без запису розв’язку у файл.
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Алгебраїчні; Метод Якобі; 0.0002; 1 2 3; 4 5 6; 1; 1;
Схема проведення тесту	Поелементне заповнення коефіцієнтів
Очікуваний результат	У файлі залишається розв’язок попередньої системи, яку було обрано записати
Стан програми після проведення випробувань	У файлі залишається розв’язок попередньої системи, яку було обрано записати

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1 Робота з програмою

Після запуску виконавчого файлу з розширенням *.exe, відкривається головне вікно програми (Рисунок 6.1).

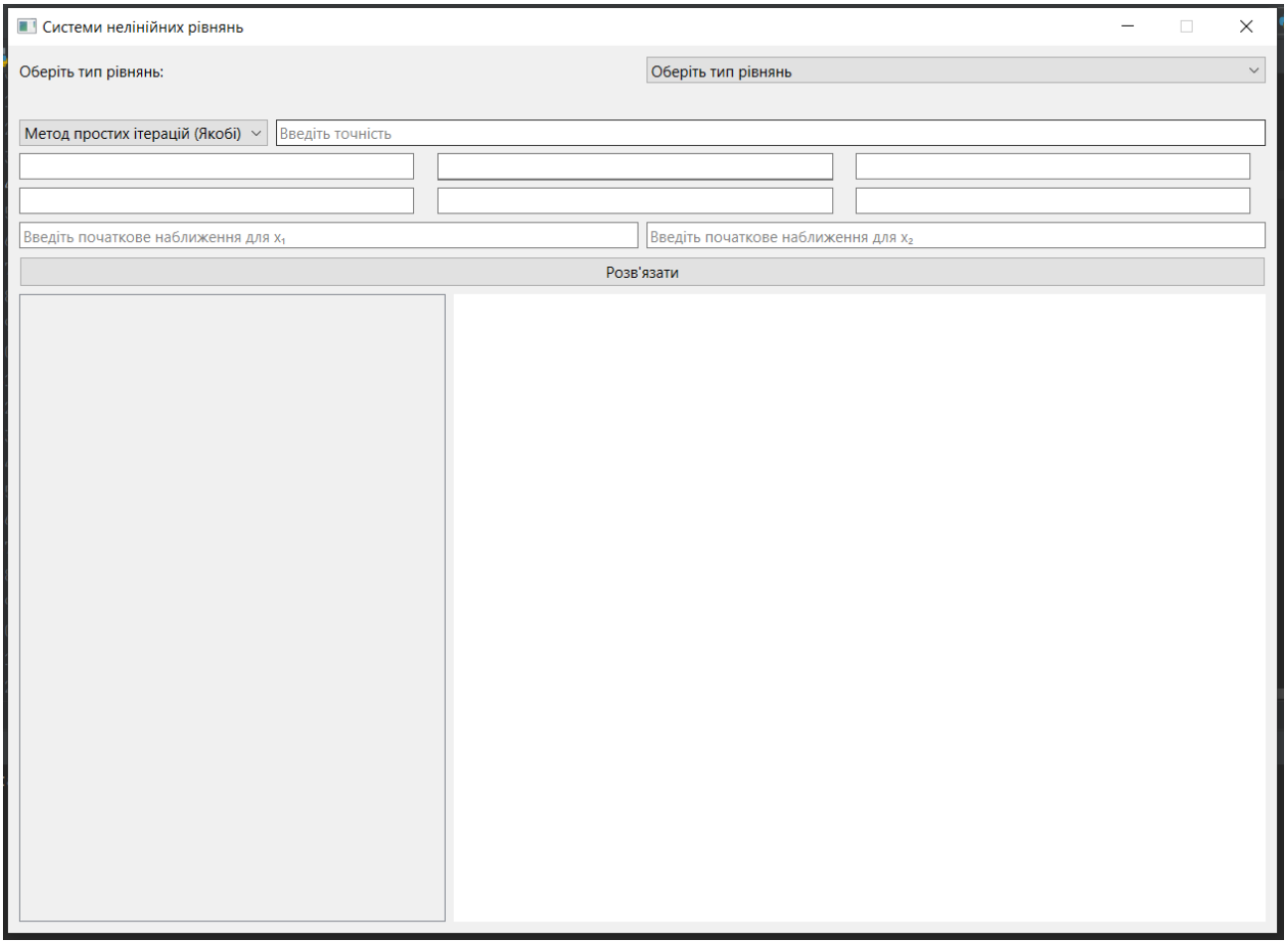


Рисунок 6.1 – Головне вікно програми

Далі за допомогою випадного списку з назвою «Оберіть тип рівнянь» шляхом натиску на цей список і обрання відповідного типу, необхідно обрати тип рівнянь, що буде оброблятися програмою (рисунок 6.2):

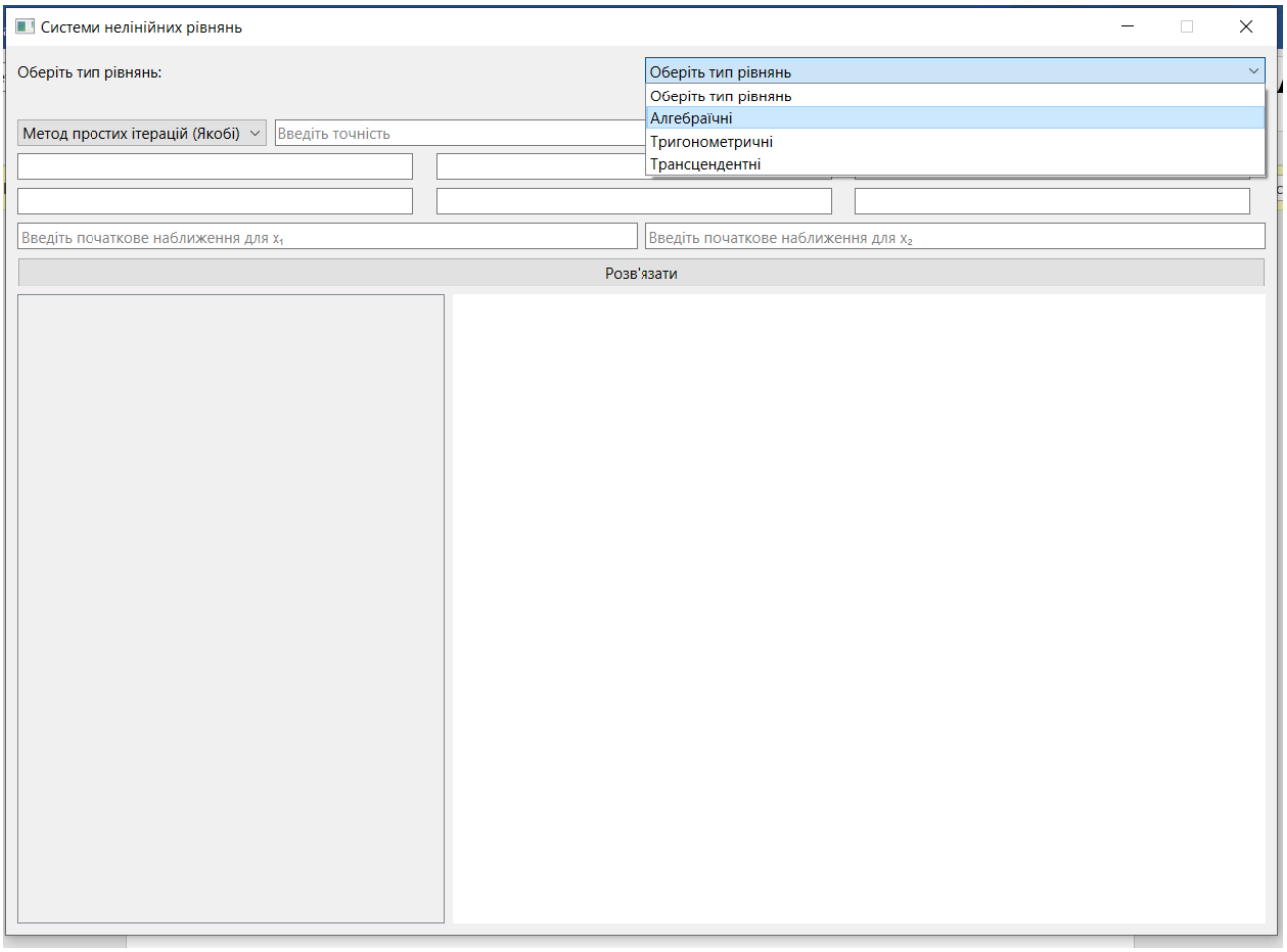


Рисунок 6.2 – Вибір необхідного типу рівнянь

Після обрання типу рівнянь у вікні відображається обраний тип рівнянь, поля для введення коефіцієнтів, а також відповідні змінні біля цих полів. Далі шляхом натиску на список з методами, необхідно обрати метод, що буде оброблятися програмою (за замовчуванням це метод простих ітерацій (Якобі)), а також ввести усі необхідні значення у поля: точність, коефіцієнти, початкові наближення (рисунок 6.3).

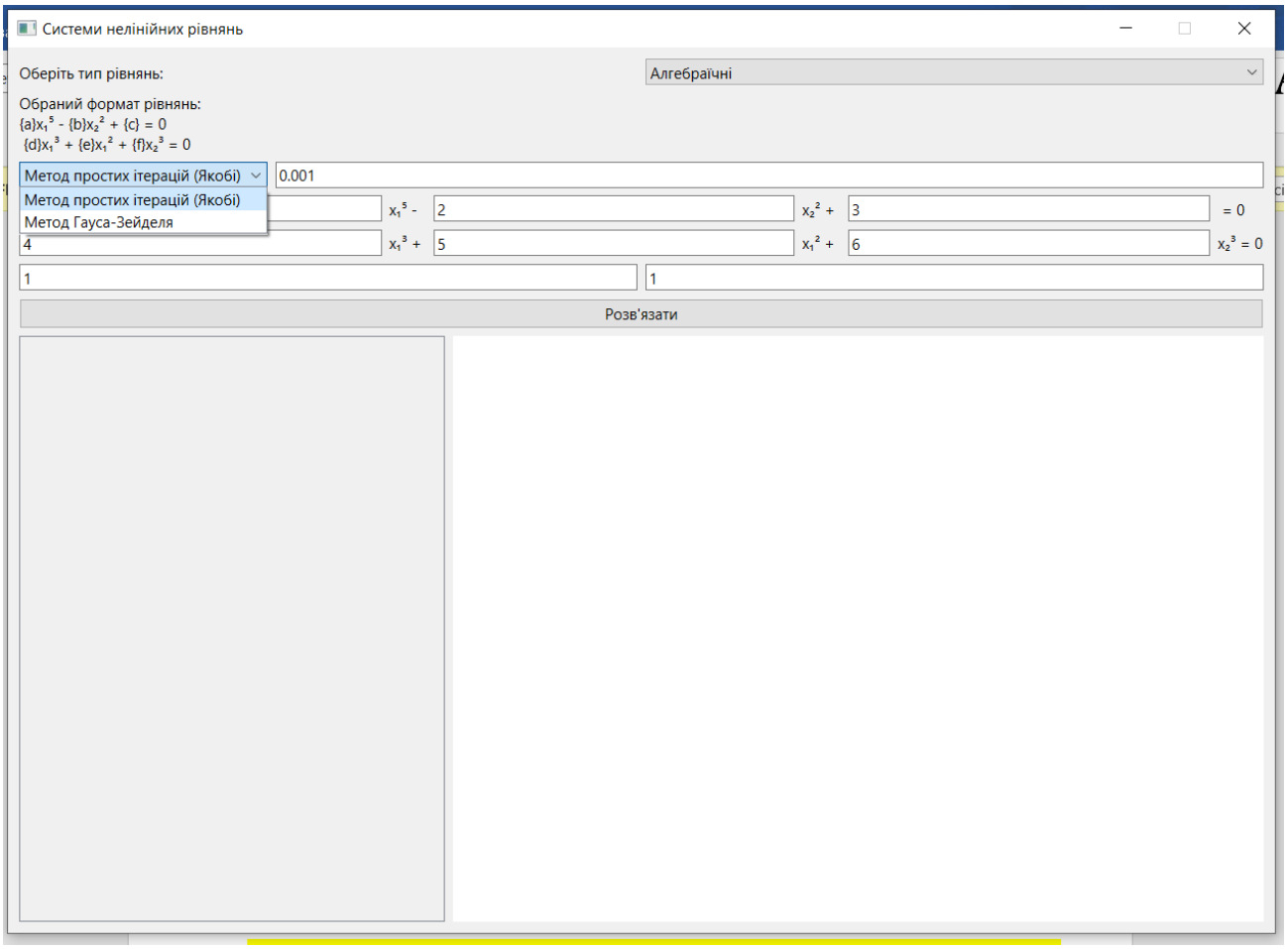


Рисунок 6.3 – Вибір необхідного методу і введення значень.

Для того, аби розв’язати введену систему, необхідно натиснути кнопку «Розв’язати», після чого на екран виведеться застереження, яке необхідно виправити, щоб розв’язати систему, або ж розв’язок з ітераціями й побудованим графіком. Також буде створене вікно, яке запитає про необхідність запису розв’язку у файл (рисунок 6.4).

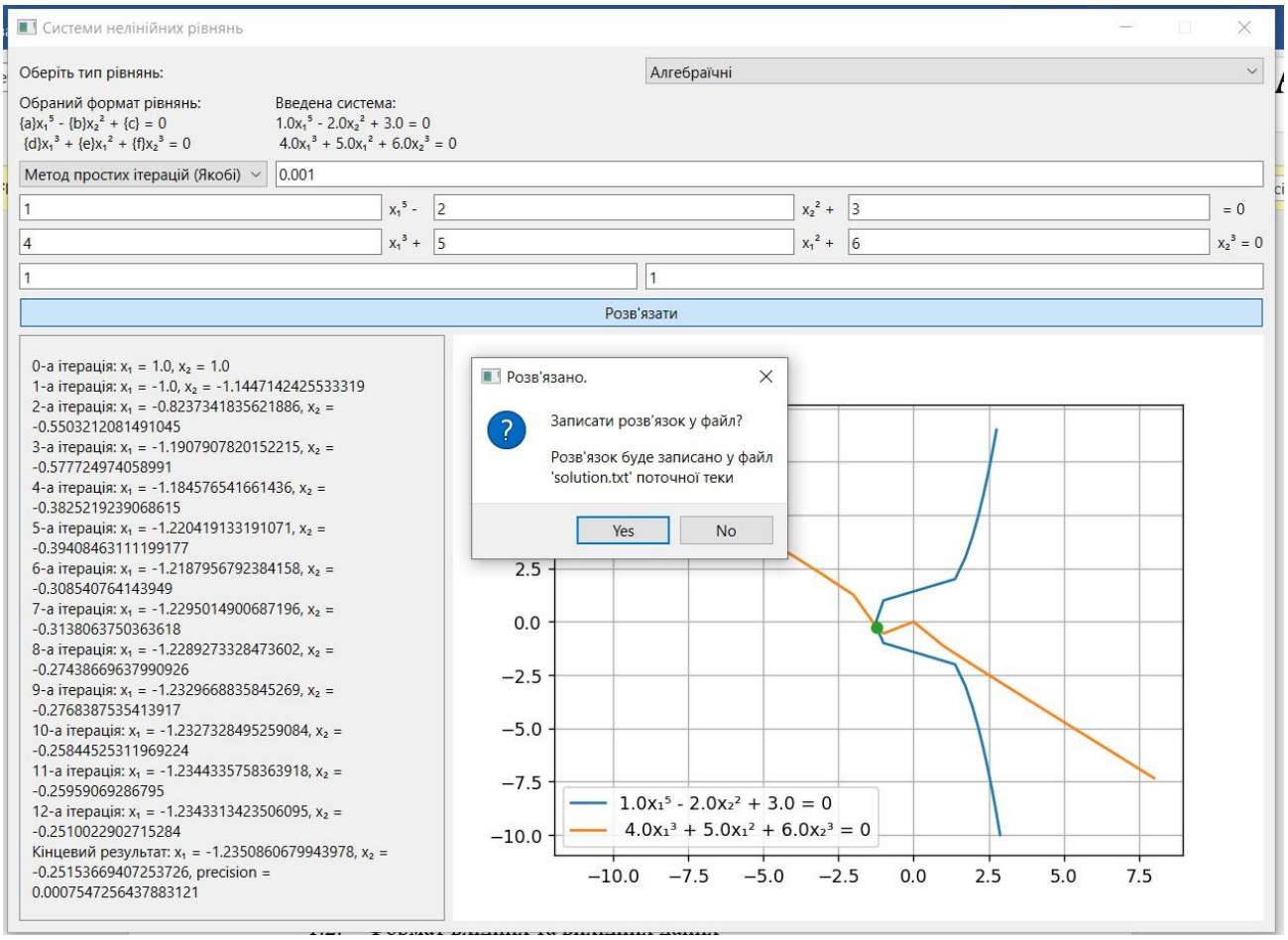


Рисунок 6.4 – Розв’язання і вікно щодо запису у файл.

Якщо буде обрано запис у файл, програма занесе розв’язок системи до файлу, «solution.txt» теки, у якій знаходиться виконуваний файл програми, та перейде у стан очікування. Інакше програма просто перейде у стан очікування.

6.2 Формат вхідних та вихідних даних

Користувачем на вхід програми подається система нелінійних рівнянь у вигляді коефіцієнтів рівнянь відповідного типу, що обирається користувачем безпосередньо перед введенням коефіцієнтів. Також користувач обирає метод розв’язання, точність обчислення, а також початкові наближення для шуканих змінних.

Результатом виконання програми є розв’язок заданої системи нелінійних рівнянь, який подається як значення першої змінної, значення другої змінної та точність розв’язання, або ж повідомлення, що дана система не має розв’язків, маєбезліч розв’язків або не сходиться для обраного методу.

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® XP/Windows Vista/Windows 7/ Windows 8/Windows 10 (з останніми обновленнями)	Windows 7/ Windows 8/Windows 10 (з останніми обновленнями)
Процесор	Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz	Intel® Pentium® D або AMD Athlon™ 64 X2
Оперативна пам'ять	256 MB RAM (для Windows® XP) / 1 GB RAM (для Windows Vista/Windows 7/ Windows 8/Windows 10)	2 GB RAM
Відеоадаптер	Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог)	

Продовження таблиці 6.1

	Мінімальні	Рекомендовані
Дисплей	800x600	1024x768 або краще
Прилади введення	Клавіатура, комп’ютерна миша	

7 АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ

Головною задачею курсової роботи була реалізація програми для розв’язання системи нелінійних рівнянь наступними методами: простих ітерацій (Якобі), Гауса-Зейделя.

Критичні ситуації у роботі програми виявлені не були. Під час тестування було виявлено, що більшість помилок виникало тоді, коли користувачем вводилися не числові вхідні дані. Тому всі дані, які вводить користувач, ретельно перевіряються на валідність і лише потім подаються на обробку програмі.

Для перевірки та доведення достовірности результатів виконання програмного забезпечення скористаюся MS Excel:

а) Метод Якобі.

Результат виконання методу Якобі наведено на рисунку 7.1:

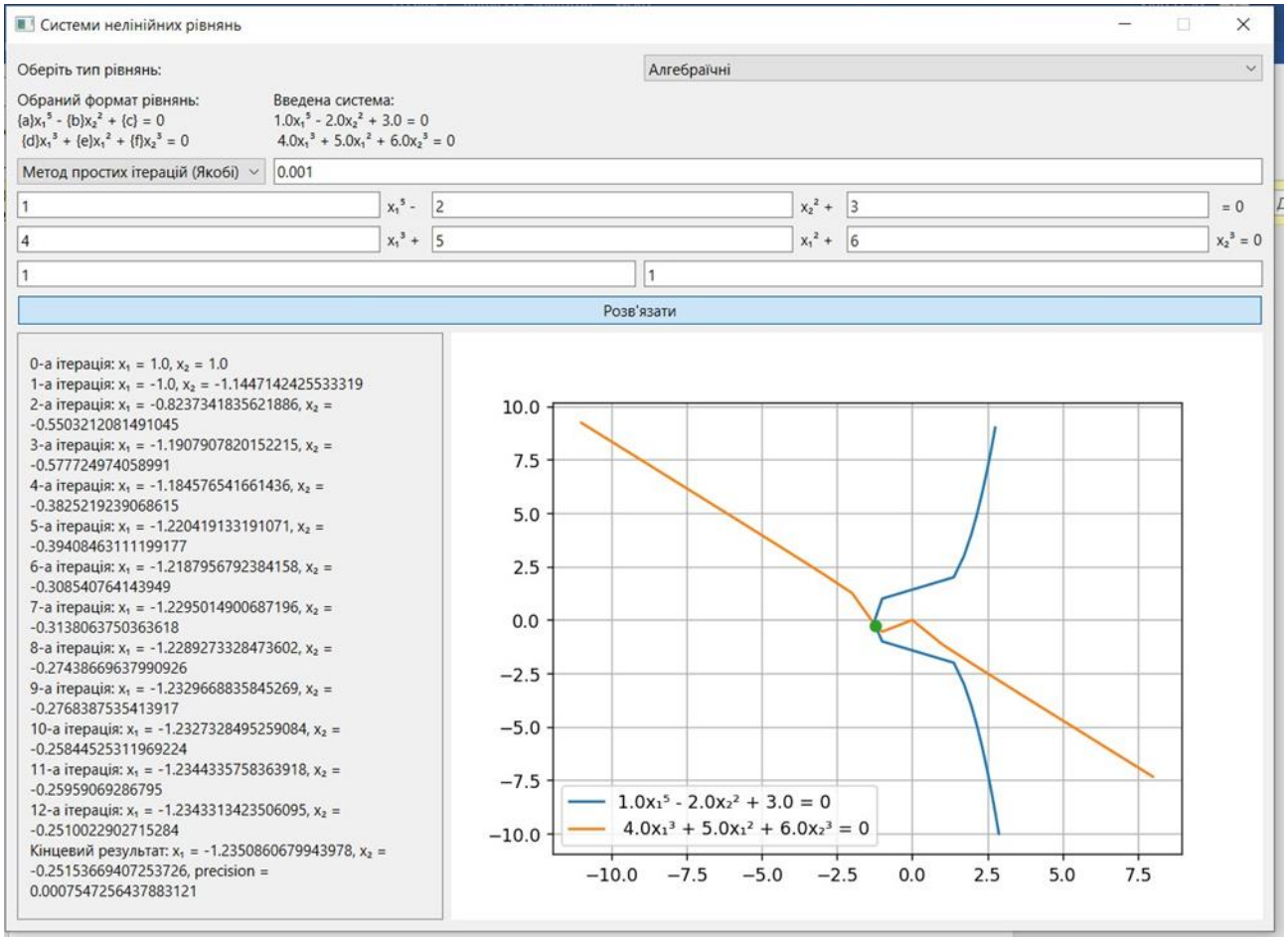
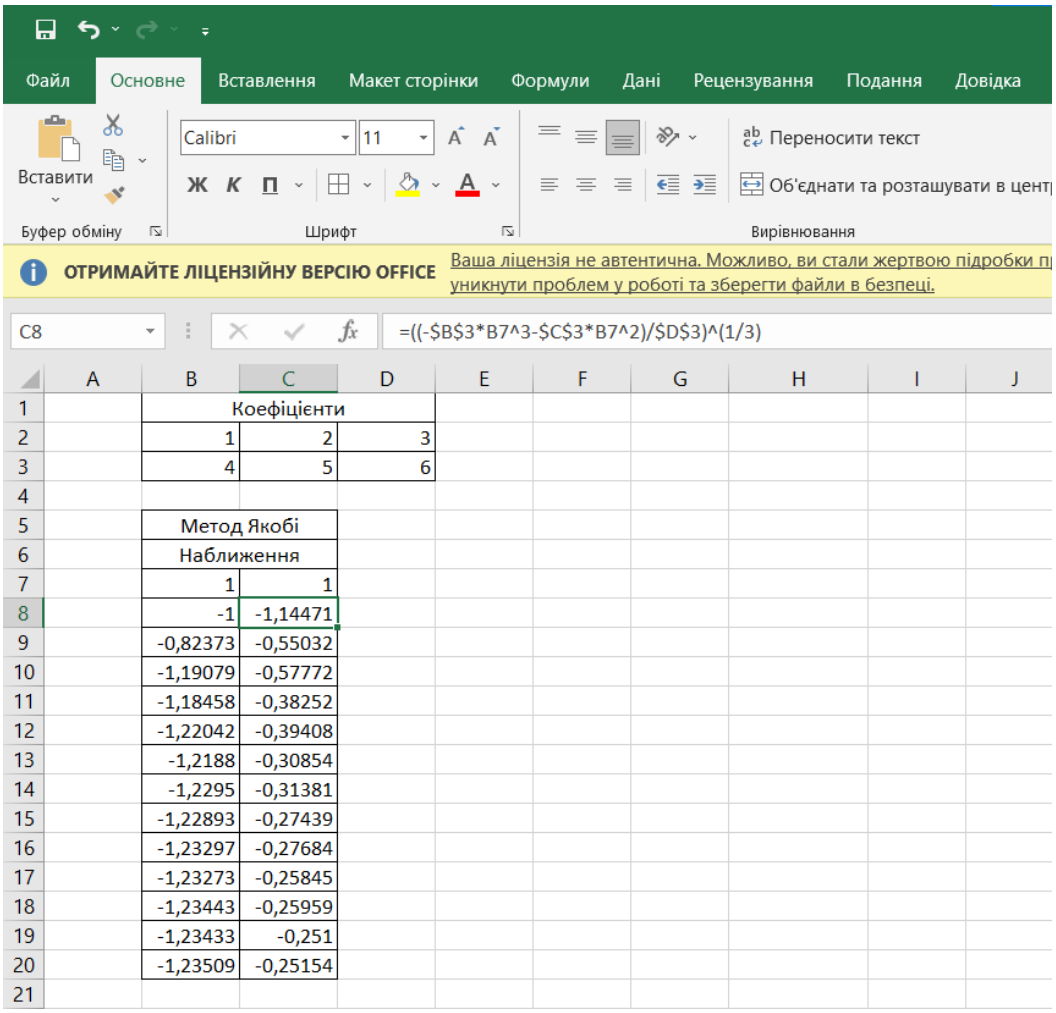


Рисунок 7.1 – Результат виконання методу Якобі

Оскільки результат виконання збігається з результатом в MS Excel (рисунок 7.2), то даний метод працює вірно.



	A	B	C	D	E	F	G	H	I	J
1		Коефіцієнти								
2		1	2	3						
3		4	5	6						
4										
5		Метод Якобі								
6		Наближення								
7		1	1							
8		-1	-1,14471							
9		-0,82373	-0,55032							
10		-1,19079	-0,57772							
11		-1,18458	-0,38252							
12		-1,22042	-0,39408							
13		-1,2188	-0,30854							
14		-1,2295	-0,31381							
15		-1,22893	-0,27439							
16		-1,23297	-0,27684							
17		-1,23273	-0,25845							
18		-1,23443	-0,25959							
19		-1,23433	-0,251							
20		-1,23509	-0,25154							
21										

Рисунок 7.2 – Перевірка методу Якобі в MS Excel 2019

б) Метод Гауса-Зейделя.

Результат виконання методу Гауса-Зейделя наведено на рисунку 7.3:

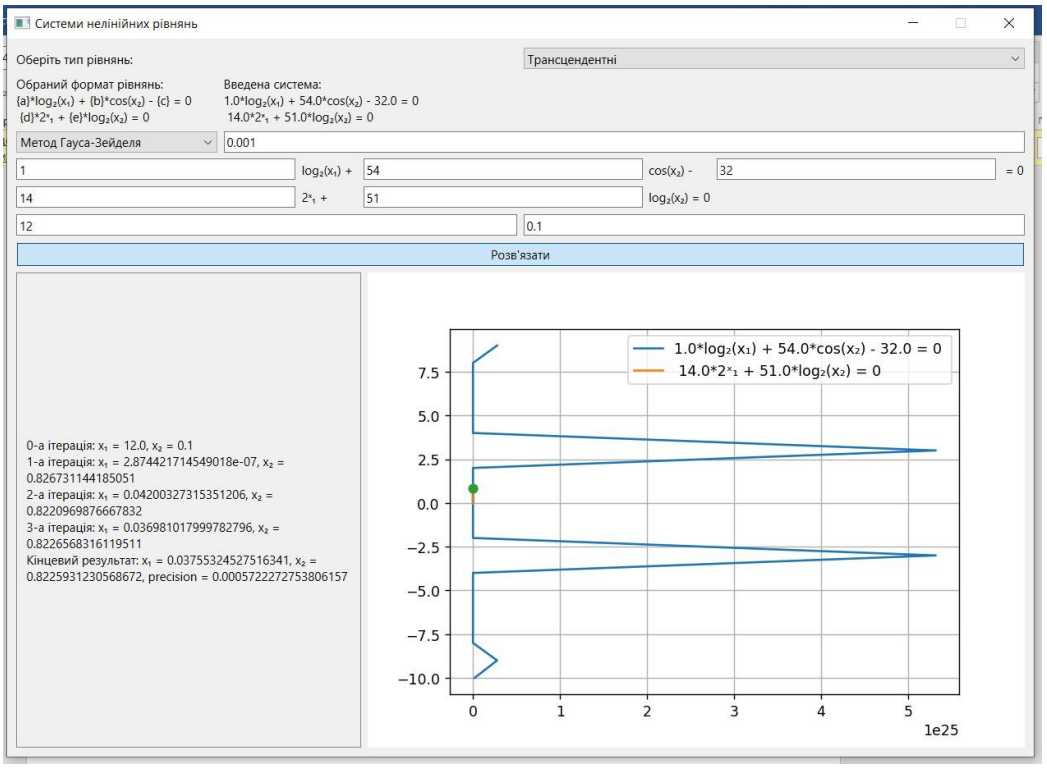


Рисунок 7.3 – Результат виконання методу Якобі

Оскільки результат виконання збігається з результатом в MS Excel (рисунок 7.4), то даний метод працює вірно.

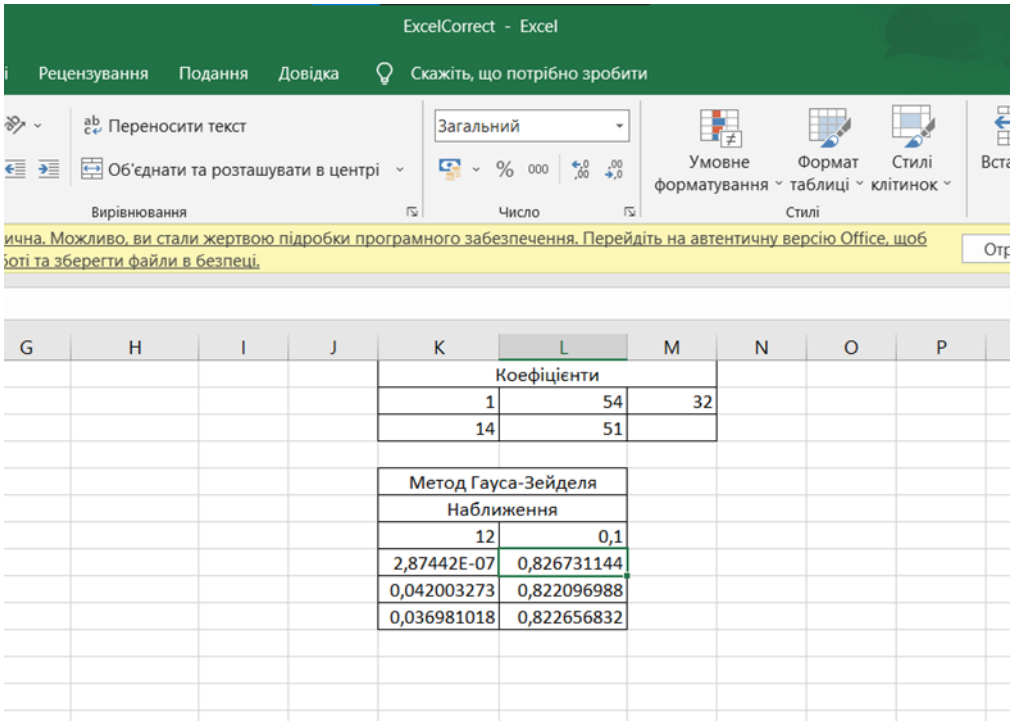


Рисунок 7.4 – Перевірка методу Гауса-Зейделя в MS Excel 2019

Таблиця 7.1 – Тестування ефективности методів.

Візьмемо рівняння кожного виду, для яких процес є збіжним, і порівняємо методи Якобі та Гауса-Зейделя за кількістю ітерацій:

1) Для алгебраїчних рівнянь: точність розв'язання $\varepsilon = 0.000001$, початкові наближення $x_1^{(0)} = 1$, $x_2^{(0)} = 2$.

$$\begin{cases} 12{x_1}^5 - 13{x_2}^2 + 14 = 0 \\ 14{x_1}^3 + 44{x_1}^2 + 3{x_2}^3 = 0 \end{cases};$$

2) Для тригонометричних рівнянь: точність розв'язання $\varepsilon = 0.000001$, початкові наближення $x_1^{(0)} = 1$, $x_2^{(0)} = 2$.

$$\begin{cases} 11(\cot x_1)^3 - 15\sin(14x_2) = 0 \\ 1\cos x_1 - 4\tan x_2 = 0 \end{cases};$$

3) Для трансцендентних рівнянь: точність розв'язання $\varepsilon = 0.000001$, початкові наближення $x_1^{(0)} = 12$, $x_2^{(0)} = 2$.

$$\begin{cases} 11\log_2 x_1 - 15\cos x_2 - 34 = 0 \\ 1 * 2^{x_1} + 44\log_2 x_2 = 0 \end{cases};$$

Вид рівнянь	Параметри тестування	Метод	
		Якобі	Гауса-Зейделя
Алгебраїчні	Кількість ітерацій	27	14
Тригонометричні	Кількість ітерацій	38	20
Трансцендентні	Кількість ітерацій	32	18

Візуалізація результатів таблиці 7.1 наведено на рисунку 7.5:

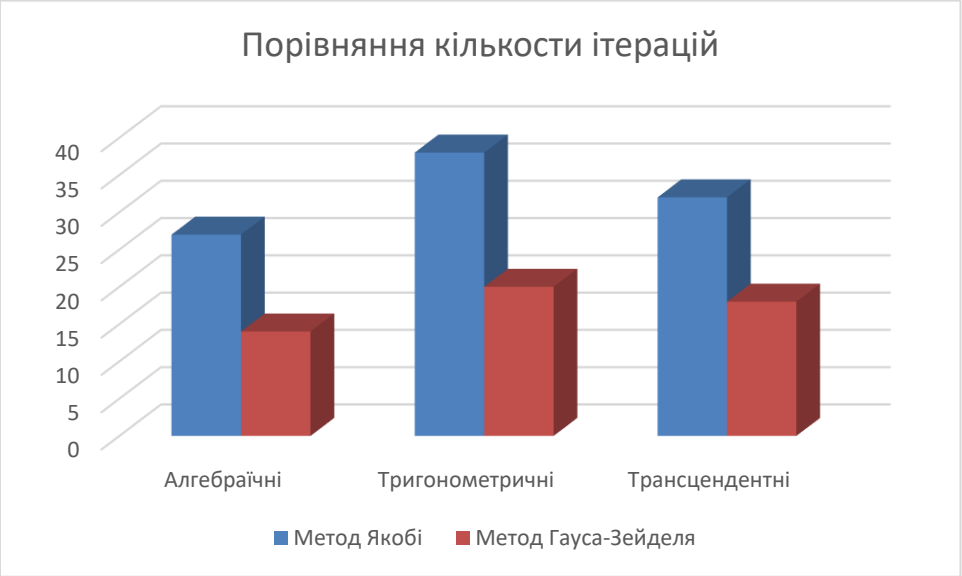


Рисунок 7.5 – Діаграма залежности кількости ітерацій від обраного методу.

За результатами тестування можна зробити такі висновки:

а) Розглянуті методи дозволяють знаходити розв’язки систем нелінійних рівнянь розглянутих видів.

б) З розглянутих методів найоптимальнішим для практичного використання є метод Гауса-Зейделя, оскільки він виконується найшвидше та має кращу збіжність.

ВИСНОВОК

Отже, було досліджено методи Якобі та Гауса-Зейделя, створено алгоритм для обчислення розв’язків систем нелінійних рівнянь, створено програмне забезпечення на його основі з використанням об’єктно-орієнтованого програмування та протестовано його.

ПЕРЕЛІК ПОСИЛАНЬ

1. Методи розв’язування систем нелінійних рівнянь. // Studfiles.

URL: https://studfile.net/preview/5581806/page:7

2. Метод Зейделя. // Studfiles.

URL: https://studfile.net/preview/5581806/page:8/

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Катедра
інформатики та програмної інженерії

Затвердив
Керівник Головченко Максим
Миколайович
«___»_____2022 р.

Виконавець:
Студент Лесів Владислав Ігорович
«___»_____2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання курсової роботи
на тему: «Розв'язання систем нелінійних рівнянь»
з дисципліни:
«Основи програмування»

Київ 2022

1. *Мета*: Метою курсової роботи є розробка ефективного програмногозабезпечення для розв’язання систем нелінійних рівнянь.
2. *Дата початку роботи*: «_____»_____202_ р.
3. *Дата закінчення роботи*: «___»_____202_ р.
4. *Вимоги до програмного забезпечення*.
   1) Функціональні вимоги:
      – Можливість обрати загальний вигляд системи нелінійних рівнянь;
      – Можливість задавати систему нелінійних рівнянь;
      – Можливість перевіряти на коректність введення даних і видавати повідомлення у разі некоректности введення;
      – Можливість обирати метод розв’язання системи нелінійних рівнянь (простої ітерації (Якобі) або Гауса-Зейделя);
      – Можливість розв’язання системи нелінійних рівнянь методом простої ітерації (Якобі);
      – Можливість розв’язання системи нелінійних рівнянь методом Гауса-Зейделя;
      – Можливість графічного розв’язання системи нелінійних рівнянь у випадку розмірности 2x2;
      – Можливість збереження результатів розв’язання у файлі;
      – Можливість відображення статистичних та\або аналітичних даних для подальшого аналізу ефективности алгоритму.
   2) Нефункціональні вимоги:
      – Можливість запуску програмного забезпечення за допомогою засобів операційної системи Windows версій 7 та вище;

– Можливість удосконалення розробленого програмного забезпечення за допомогою засобів мови програмування Python 3;

– 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 проєктування 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 …… 202_ 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

main.py

```
import sys
from interfacee import *


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec()


if __name__ == "__main__":
    main()
```

interfacee.py

```
from PyQt6.QtCore import *
from solution import *
from visual import *
from scrolllabel import *
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas


class MainWindow(QMainWindow):
```

"""

Клас головного вікна програми.

Атрибути

----

equation1 : Equation

Перше рівняння системи.

equation2 : Equation

Друге рівняння системи.

greetLabel : QLabel

Текстове поле "Оберіть тип рівнянь"

combotype : QComboBox

Випадний список для обрання типу рівнянь

windowView : Visual

Атрибут для обробки взаємодії користувача з вікном.

equations : QLabel

Текстове поле для виведення загального виду системи

filledEq : QLabel

Текстове поле для виведення введеної системи

method : QComboBox

Випадний список для обрання методу розв'язання

precision : QLineEdit

Поле для введення точности розв'язання

polya1 : QLineEdit[]

Список полів для введення коефієцінтів 1 рівняння

polya2 : QLineEdit[]

Список полів для введення коефієцінтів 2 рівняння

labels1 : QLabel[]

Список текстових полів для змінних біля коефіцієнтів 1 рівняння

labels2 : QLabel[]

Список текстових полів для змінних біля коефіцієнтів 2 рівняння

startX : QLineEdit

Поле для введення початкового наближення першої змінної

startY : QLineEdit

Поле для введення початкового наближення другої змінної

confirm : QPushButton

Кнопка для розв'язання системи.

equationSolve : Solution

Атрибут для розв'язання системи і взаємодії його з вікном

iterations : ScrollLabel

Текстове поле з прокручуванням для виведення ітерацій й кінцевого розв'язку

figure : Figure

Фігура, де буде намальований графік системи

canvas : FigureCanvas

Віджет полотна, де буде розташований графік

msg : QMessageBox

Вікно з питанням про запис у файл

Методи

"""

def __init__(self):

"""

Створює усі елементи вікна, розташовує їх, надає їм функціонал.

Аргументи:

```
        ---------
            немає.
        """

        super().__init__()

        self.equation1 = None
        self.equation2 = None

        self.setWindowTitle("Системи нелінійних рівнянь")

        self.greetLabel = QLabel()
        self.greetLabel.setText("Оберіть тип рівнянь:")

        self.comboType = QComboBox()
        self.comboType.addItems(["Оберіть тип рівнянь", "Алгебраїчні",
"Тригонометричні", "Трансцендентні"])
        self.windowView = Visual(self)

        # Під'єднуємо змінну обраного елементу в списку до виклику відповідного
методу в класі Visual

self.comboType.currentIndexChanged.connect(self.windowView.showDefEquation)

        self.equations = QLabel()
        self.filledEq = QLabel()

        self.method = QComboBox()
        self.method.addItems(["Метод простих ітерацій (Якобі)", "Метод Гауса-
Зейделя"])
```

```
        self.precision = QLineEdit()
        self.precision.setPlaceholderText("Введіть точність")

        self.polya1 = [QLineEdit() for i in range(3)]
        self.labels1 = [QLabel() for i in range(3)]
        self.polya2 = [QLineEdit() for i in range(3)]
        self.labels2 = [QLabel() for i in range(3)]

        self.startX = QLineEdit()
        self.startY = QLineEdit()
        self.startX.setPlaceholderText("Введіть початкове наближення для x₁")
        self.startY.setPlaceholderText("Введіть початкове наближення для x₂")

        self.confirm = QPushButton("Розв'язати")
        self.confirm.setCheckable(True)
        self.equationSolve = Solution(self)

        # Під'єднумо натискання кнопки до виклику відповідного методу в класі
Solution
        self.confirm.clicked.connect(self.equationSolve.getCoeffs)

        self.iterations = ScrollLabel()

        self.figure = plt.figure()
        self.canvas = FigureCanvas(self.figure)

        self.msg = QMessageBox()
        self.msg.setIcon(QMessageBox.Icon.Question)
```

```
        self.msg.setText("Записати розв'язок у файл?")
        self.msg.setInformativeText("Розв'язок буде записано у файл 'solution.txt'
поточної теки")
        self.msg.setWindowTitle("Розв'язано.")
        self.msg.setStandardButtons(QMessageBox.StandardButton.Yes |
QMessageBox.StandardButton.No)

        # Макети для того, щоб зібрати усі віджети так, як вони мають знаходитися
        outerLayout = QVBoxLayout()
        chooseLayout = QHBoxLayout()
        eqLayout = QGridLayout()
        gridLayout = QGridLayout()
        startLayout = QHBoxLayout()
        enterLayout = QVBoxLayout()
        downLayout = QHBoxLayout()

        chooseLayout.addWidget(self.greetLabel)
        chooseLayout.addWidget(self.comboType)

        eqLayout.addWidget(self.equations, 0, 0)
        eqLayout.addWidget(self.filledEq, 0, 1)
        eqLayout.addWidget(self.method, 1, 0)
        eqLayout.addWidget(self.precision, 1, 1)

        for i in range(3):
            gridLayout.addWidget(self.polya1[i], 0, i * 2)
            gridLayout.addWidget(self.labels1[i], 0, i * 2 + 1)
        for i in range(3):
            gridLayout.addWidget(self.polya2[i], 1, i * 2)
            gridLayout.addWidget(self.labels2[i], 1, i * 2 + 1)
```

```
        startLayout.addWidget(self.startX)
        startLayout.addWidget(self.startY)

        # Встановлюється фіксований розмір вікна
        self.setFixedSize(QSize(1000, 700))
        enterLayout.addLayout(chooseLayout)
        enterLayout.addLayout(eqLayout)
        enterLayout.addLayout(gridLayout)
        enterLayout.addLayout(startLayout)
        enterLayout.addWidget(self.confirm)
        downLayout.addWidget(self.iterations)
        downLayout.addWidget(self.canvas)
        outerLayout.addLayout(enterLayout)
        outerLayout.addLayout(downLayout)

        container = QWidget()
        container.setLayout(outerLayout)

        self.setCentralWidget(container)
```

scrolllabel.py

```
from PyQt6.QtWidgets import *


class ScrollLabel(QScrollArea):
    """
    Клас поля з прокручуванням
```

```
---

Атрибути
--------
label   : Qlabel
    Текстове поле, яке має прокручуватися.

Методи
------
"""

def __init__(self):
    """
    Ініціалізує створення текстового поля з прокручуванням.

    Аргументи
    ---------
        немає.
    """

    # Ініціалізуємо віджет області з прокручуванням
    QScrollArea.__init__(self)

    # Віджет автоматично змінює розмір відповідно до умов
    self.setWidgetResizable(True)

    content = QWidget(self)
    self.setWidget(content)
```

```
        lay = QVBoxLayout(content)

        # Створюємо текстове поле і дозволяємо перенос слів
        self.label = QLabel(content)
        self.label.setWordWrap(True)
        lay.addWidget(self.label)
```

visual.py

```
from math import *


class Visual:
    """
    Клас, призначений для взаємодії з користувачем у графічному інтерфейсі

    ---

    Атрибути
    --------
    windowGiven : MainWindow
        Вікно, за яким відбуватиметься взаємодія.

    Методи
    ------
    showDefEquation(i):
        Виводить у вікно формат рівнянь відповідно до того, який обраний в випадному списку.

    setGrids(i)
```

Виводить на екран потрібні поля для введення коефіцієнтів і змінні біля них.

plotEq(check, xs, ys)

Будує графік розв'язаної системи і виводить його на екран.

"""

def __init__(self, windowGiven):

"""

Конструктор класу.

Аргументи

---

windowGiven : MainWindow

Вікно, з яким буде взаємодія

"""

self.window = windowGiven

def showDefEquation(self, i):

"""

Виводить у вікно формат рівнянь відповідно до того, який обраний в випадному списку.

Аргументи

---

i : int

Індекс обраного типу рівнянь у випадному списку.

Повертає

```
        --------
        None
        """

        text = "Обраний формат рівнянь:\n"
        if i == 1:
            text += "{a}x₁⁵ - {b}x₂² + {c} = 0 \n {d}x₁³ + {e}x₁² + {f}x₂³ = 0"
        elif i == 2:
            text += "{a}*ctg³(x₁) + {b}*sin({c}x₂) = 0 \n {d}*cos(x₁) - {e}*tg(x₂) = 0"
        elif i == 3:
            text += "{a}*log₂(x₁) + {b}*cos(x₂) - {c} = 0 \n {d}*2^x₁ + {e}*log₂(x₂) = 0"

        # Задаємо текст відповідному текстовому полю
        self.window.equations.setText(text)

        # Викликаємо наступний метод
        self.setGrids(i)

    def setGrids(self, i):
        """
        Виводить на екран потрібні поля для введення коефіцієнтів і змінні біля них.

        Аргументи
        ---------
        i    : int
            Індекс обраного типу рівнянь у випадному списку.

        Повертає
        --------
```

```
        None
        """

        # У тригон. і трансц. рівняннях нема 6-го коефіцієнту, тому приховуємо його.
        if i == 1:
            self.window.labels1[0].setText("x₁⁵ - ")
            self.window.labels1[1].setText("x₂² + ")
            self.window.labels1[2].setText(" = 0")
            self.window.labels2[0].setText("x₁³ + ")
            self.window.labels2[1].setText("x₁² + ")
            self.window.polya2[2].show()
            self.window.labels2[2].show()
            self.window.labels2[2].setText("x₂³ = 0")
        elif i == 2:
            self.window.labels1[0].setText("ctg³(x₁) + ")
            self.window.labels1[1].setText("*sin(")
            self.window.labels1[2].setText("x₂) = 0")
            self.window.labels2[0].setText("cos(x₁) - ")
            self.window.labels2[1].setText("tg(x₂) = 0")
            self.window.polya2[2].hide()
            self.window.labels2[2].hide()
        elif i == 3:
            self.window.labels1[0].setText("log₂(x₁) + ")
            self.window.labels1[1].setText("cos(x₂) -")
            self.window.labels1[2].setText(" = 0")
            self.window.labels2[0].setText("2^x₁ + ")
            self.window.labels2[1].setText("log₂(x₂) = 0")
            self.window.polya2[2].hide()
            self.window.labels2[2].hide()
```

```
def plotEq(self, check, xs, ys):
    """
    Будує графік розв'язаної системи і виводить його на екран.

    Аргументи
    ---------
    check    : int
        Показує, розв'язано окремим випадком чи загальним (ітераційно).
    xs   : float
        Перший розв'язок системи.
    ys   : float
        Другий розв'язок системи.

    Повертає
    --------
    None
    """

    # Виділяємо коефіцієнти рівнянь
    a, b, c = self.window.equation1.parameters
    d, e, f = self.window.equation2.parameters
    x11, x21, x12, x22 = [], [], [], []

    '''Якщо розв'язано ітераційно, то на проміжку
    10 в обидва боки від розв'язку шукаємо значення функцій і
    збіерігаємо їх в окремі списки'''
    '''Якщо ж розв'язано неітераційно, то перевіряємо,
    чи має система числовий розв'язок:
    якщо так, то маємо точку перетину на графіку'''
```

```
if check == 0:
    x21 = list(range(int(ys) - 10, 10 + int(ys)))
    x21.insert(len(x21) // 2, ys)
    x12 = list(range(int(xs) - 10, 10 + int(xs)))
    x12.insert(len(x12) // 2, xs)
    if self.window.equation1.index == 1:
        for i in x21:
            w = -1 if ((b * i ** 2 - c) / a) < 0 else 1
            x11.append(w * abs((b * i ** 2 - c) / a) ** (1 / 5))
        for i in x12:
            w = -1 if ((-d * i ** 3 - e * i ** 2) / f) < 0 else 1
            x22.append(w * abs((-d * i ** 3 - e * i ** 2) / f) ** (1 / 3))
    elif self.window.equation1.index == 2:
        if 0 in x21:
            x21.remove(0)
        for i in x21:
            w = -1 if (-b * sin(c * i) / a) < 0 else 1
            x11.append(atan(1 / (w * abs(-b * sin(c * i) / a) ** (1 / 3))))
        x22 = [atan((d * cos(i)) / e) for i in x12]
    elif self.window.equation1.index == 3:
        x11 = [2 ** ((c - b * cos(i)) / a) for i in x21]
        x22 = [2 ** ((-d * 2 ** i) / e) for i in x12]
elif type(xs) == int or type(xs) == float and type(ys) == int or type(ys) == float:
    x11, x21 = xs, ys
    x12, x22 = xs, ys
self.window.figure.clear()

'"Якщо розв'язок існує, то малюємо окремо графіки і позначаємо
окремо їх точку перетину"'
```

```
        if x11 != []:
            ax = self.window.figure.add_subplot(111)
            labels = (self.window.filledEq.text()[17:]).split("\n")
            ax.plot(x11, x21, label=labels[0])
            ax.plot(x12, x22, label=labels[1])
            ax.plot(xs, ys, marker="o")
            ax.grid()
            ax.legend()

        self.window.canvas.draw()
```

solution.py

```
from math import *
from PyQt6.QtWidgets import *
from equation import *


class Solution:
    """
    Клас, призначений для розв'язування введеної системи рівнянь.

    ---

    Атрибути
    --------
    window : MainWindow
        Вікно, за яким відбуватиметься взаємодія з інтерфейсом.
    startY  : float
        Початкове наближення другої змінної.
```

startX : float

Початкове наближення першої змінної.

Методи

---

getCoeffs()

Зчитує усі введені значення, створює об'єкти рівнянь, викликає подальші дії:

розв'язання, будування графіку, запис у файл.

convergence(det, x, y)

Обраховує визначник матриці Якобі для поточної ітерації і число, що відповідає

за збіжність ітераційного процесу.

checkZero()

Перевіряє, чи не потрібно обчислити систему окремо від загального ітераційного процесу,

і, якщо так, то обчислює. Інакше переходить до звичайного розв'язання.

solveIter()

Розв'язує систему ітеративно відповідно до обраного методу.

messageButton(reply)

Якщо обрано, записує розв'язок системи у файл

iterAlg(equation2, yl, start)

Обчислює ітеративно змінну в кубічному рівнянні у випадку, коли інша змінна відома.

"""

```
    def __init__(self, windowGiven):
        """
        Конструктор класу.

        Аргументи
        ---------
            немає.
        """

        self.window = windowGiven
        self.startY = None
        self.startX = None

    def getCoeffs(self):
        """
        Зчитує усі введені значення, створює об'єкти рівнянь, викликає подальші
дії:
        розв'язання, будування графіку, запис у файл.

        Аргументи
        --------
            немає.

        Повертає
        --------
        None
        """

        self.window.filledEq.setText("")
```

```
        try:
            ind = self.window.comboType.currentIndex()
            meth = self.window.method.currentIndex()
            prec = float(self.window.precision.displayText())
            a = [float(self.window.polya1[i].displayText()) for i in range(3)]
            b = [self.window.polya2[i].displayText() for i in range(3)]

            self.startX, self.startY = float(self.window.startX.displayText()),
float(self.window.startY.displayText())
            if ind == 1:
                b = [float(i) for i in b]
            elif ind != 0:
                b = [float(i) for i in b[:2]]
                b.append(0)
            self.window.equation1, self.window.equation2 = Equation(ind, a, meth, prec),
Equation(ind, b, meth, prec)
            s = self.window.equations.text()
            s = s.replace("{a}", str(self.window.equation1.parameters[0]))
            s = s.replace("{b}", str(self.window.equation1.parameters[1]))
            s = s.replace("{c}", str(self.window.equation1.parameters[2]))
            s = s.replace("{d}", str(self.window.equation2.parameters[0]))
            s = s.replace("{e}", str(self.window.equation2.parameters[1]))
            if ind == 1:
                s = s.replace("{f}", str(self.window.equation2.parameters[2]))
            self.window.filledEq.setText("Введена система:\n" + s[24:])
            check, x1, x2, press = self.checkZero()
            self.window.windowView.plotEq(check, x1, x2)
            reply = self.window.msg.exec()
            self.messageButton(reply)
        except (ValueError, TypeError):
```

```
            self.window.filledEq.setText(
                "Помилка. Переконайтеся, що: \n\t-введені коефіцієнти, початкові наближення та точність - це цілі або "
                "дійсні числа у форматі '0.0' .")
        except ZeroDivisionError:
            self.window.filledEq.setText("Помилка. Ділення на 0.")
        except OverflowError:
            self.window.filledEq.setText("Помилка переповнення. Спробуйте зменшити значення початкового наближення.")

    def convergence(self, det, x, y):
        """
        Обраховує визначник матриці Якобі для поточної ітерації і число, що відповідає
        за збіжність ітераційного процесу.

        Аргументи
        --------
            det    : float
                Число, що відповідає за збіжність, обчислене на попередній ітерації.
            x    : float
                Поточне наближення для першої змінної.
            y    : float
                Поточне наближення для другої змінної.

        Повертає
        --------
            temp (det)    : float
                Обчислене число, що відповідає за збіжність.
            a    : bool
```

```
        Вираз, що показує, збіжний процес (True) чи ні (False) на даній
ітерації.
        """

        a, b, c = self.window.equation1.parameters
        d, e, f = self.window.equation2.parameters
        try:
            if self.window.equation1.index == 1:
                w = -1 if ((b * y ** 2 - c) / a) < 0 else 1
                temp = (2 * b * y) / (5 * w * (abs(a * (b * y ** 2 - c) ** 4) ** (1 / 5)))
                w = -1 if ((-d * x ** 3 - e * x ** 2) / f) < 0 else 1
                temp *= ((3 * d * x + 2 * e) * w * abs(((-d * x ** 3) - e * x ** 2) / f) ** (1 /
3)) / (
                        3 * d * x ** 2 + 3 * e * x)
            elif self.window.equation1.index == 2:
                w = -1 if (b * sin(c * y) / a) < 0 else 1
                temp = (a * b * c * cos(c * y)) / (
                        (3 * b ** 2 * sin(c * y) ** 2 + 3 * a ** 2) * atan(w * abs(b * sin(c *
y)) / a) ** (-2 / 3))
                temp *= ((-e * d * sin(x)) / (e ** 2 + d ** 2 + (cos(x)) ** 2))
            else:
                temp = 2 ** ((c - b * cos(y)) / a) * log(2) * (b * sin(y) / a)
                temp *= (2 ** (x - (d * 2 ** x / e)) * log(2) ** 2 * (-d / e))
            temp *= -1
            temp *= det
            if abs(temp) <= abs(det):
                return temp, True
            else:
                return temp, False
```

```
        except:
            return det, True

    def checkZero(self):
        """
        Перевіряє, чи не потрібно обчислити систему окремо від загального
ітераційного процесу,
        і, якщо так, то обчислює. Інакше переходить до звичайного розв'язання.

        Аргументи
        ---------
            немає.

        Повертає
        --------
            1    : const int
                Число, що означає, що розв'язано окремо від загального методу.
            sol[0]   : float
                Перший розв'язок системи.
            sol[1]   : float
                Другий розв'язок системи.
            0    : const int
                Точність розв'язання, окремим методом дорівнює 0.
        """

        a, b, c = self.window.equation1.parameters
        d, e, f = self.window.equation2.parameters
        sol = [0, 0]
        check = 0
        if self.window.equation1.index == 1:
```

```
if a == 0:
    check = 1
    if b == 0:
        if c == 0:
            sol = ["b", "b"]
        else:
            sol = ["0", "0"]
    else:
        if c / b >= 0:
            x2 = sqrt(c / b)
            x1 = self.iterAlg(self.window.equation2, x2, self.startX)
            sol = [x1, x2]
        else:
            sol = ["0", "0"]
if check == 0:
    if f == 0:
        check = 1
        if e == 0 and d == 0:
            sol = ["b", "b"]
        else:
            if e == 0 and d != 0 or e != 0 and d == 0:
                sol = [0, 0]
            else:
                sol = [self.iterAlg(self.window.equation2, 0, self.startX), 0]
                try:
                    if (a * sol[0] ** 5 + c) / b >= 0:
                        sol[1] = sqrt((a * sol[0] ** 5 + c) / b)
                    else:
                        sol[1] = "0"
                except TypeError:
```

```
                    sol[1] = "0"
elif self.window.equation1.index == 2:
    if a == 0:
        check = 1
        if b == 0 or b != 0 and c == 0:
            sol = ["b", "b"]
        else:
            sol = [pi / 2, 0]
    if check == 0:
        if e == 0:
            check = 1
            if d == 0:
                sol = ["b", "b"]
            else:
                sol = [pi / 2, 0]
elif self.window.equation1.index == 3:
    if a == 0:
        check = 1
        if b == 0:
            if c == 0:
                sol = ["b", "b"]
            else:
                sol = ["0", "0"]
        else:
            if -1 <= c / b <= 1:
                sol = [0, acos(c / b)]
                if sol[1] > 0 and -e * log2(sol[1]) / d > 0:
                    sol[0] = log2(-e * log2(sol[1]) / d)
            else:
                sol = ["0", "0"]
```

```
            if check == 0:
                if e == 0:
                    check = 1
                    if d == 0:
                        sol = ["b", "b"]
                    else:
                        sol = ["0", "0"]
        if check == 1:
            if sol[0] == "0" or sol[1] == "0":
                text = "Система розв'язків не має"
            elif sol[0] != "b" and sol[1] != "b":
                text = "Результат: x₁ =" + str(sol[0]) + ", x₂ =" + str(sol[1]) + ", точність =
0."
            else:
                text = "Система має безліч розв'язків"
            self.window.iterations.label.setText(text)
            return 1, sol[0], sol[1], 0
        else:
            return self.solveIter()

    def solveIter(self):
        """
        Розв'язує систему ітеративно відповідно до обраного методу.

        Аргументи
        ---------
            немає.

        Повертає
        --------
```

```
    0    : const int
        Число, що означає, що розв'язано ітеративним методом.
    x1[0]   : float
        Перший розв'язок системи.
    x2[0]   : float
        Другий розв'язок системи.
    precision    : float
        Точність розв'язання.
"""

text = ""
precision = self.window.equation1.prec + 1
a, b, c = self.window.equation1.parameters
d, e, f = self.window.equation2.parameters
x1, x2 = [self.startX, 0], [self.startY, 0]
k = 0
det, boo = self.convergence(1, x1[0], x2[0])
el = self.window.equation1.meth
while precision >= abs(self.window.equation1.prec) and k <= 200:
    if self.window.equation1.index == 1:
        w = -1 if ((b * x2[0] ** 2 - c) / a) < 0 else 1
        x1[1] = w * abs((b * x2[0] ** 2 - c) / a) ** (1 / 5)
        w = -1 if ((-d * x1[el] ** 3 - e * x1[el] ** 2) / f) < 0 else 1
        x2[1] = w * abs((-d * x1[el] ** 3 - e * x1[el] ** 2) / f) ** (1 / 3)
    elif self.window.equation1.index == 2:
        if k == 0:
            x1[1] = pi / 2
        else:
            w = -1 if (-b * sin(c * x2[0]) / a) < 0 else 1
            x1[1] = atan(1 / (w * abs(-b * sin(c * x2[0]) / a) ** (1 / 3)))
```

```
            x2[1] = atan((d * cos(x1[el])) / e)
        elif self.window.equation1.index == 3:
            x1[1] = 2 ** ((c - b * cos(x2[0])) / a)
            x2[1] = 2 ** ((-d * 2 ** x1[el]) / e)
        text += (str(k) + "-а ітерація: x₁ = " + str(x1[0]) + ", x₂ = " + str(x2[0]) + "\n")
        self.window.iterations.label.setText(text)
        precision = max(abs(x1[1] - x1[0]), abs(x2[1] - x2[0]))
        x1[0], x2[0] = x1[1], x2[1]
        det, boo = self.convergence(det, x1[0], x2[0])
        if not boo:
            kin = "Процес не збіжний, неможливо знайти розв'язок."
            break
        k += 1

    if boo:
        kin = ("Кінцевий результат: x₁ = " + str(x1[0]) + ", x₂ = " + str(x2[0]) + ",
precision = " + str(
            precision))
    text += kin
    self.window.iterations.label.setText(text)
    return 0, x1[0], x2[0], precision

  def messageButton(self, reply):
    """
    Якщо обрано, записує розв'язок системи у файл

    Аргументи
    ---------
    reply    : QMessageBox.StandartButton
        Обрана кнопка у вікні запитання.
```

```
        Повертає
        --------
        None.
        """

        if reply == QMessageBox.StandardButton.Yes:
            text = self.window.filledEq.text()
            text += "\n\n"
            text += self.window.iterations.label.text()
            with open("solution.txt", "w", encoding="utf-8") as file:
                file.write(text)

    def iterAlg(self, equation2, yl, start):
        """
        Обчислює ітеративно змінну в кубічному рівнянні у випадку, коли інша змінна відома.

        Аргументи
        ---------
        equation2  : Equation
            Рівняння, яке треба розв'язати.
        yl   : float
            Друга змінна, вже відома.
        start    : float
            Початкове наближення для розв'язання.

        Повертає
        --------
            x    : float||string
```

```
        Значення обчисленої змінної.
"""


d, c, f = equation2.parameters
if d == 0:
    if e == 0:
        if f * yl ** 3 == 0:
            return "b"
        else:
            return "0"
    else:
        if (-f * yl ** 3) / e >= 0:
            return sqrt((-f * yl ** 3) / e)
        else:
            return "0"
elif e == 0:
    w = -1 if (-f * yl ** 3) / d < 0 else 1
    return w * abs((f * yl ** 3) / d) ** (1 / 3)
else:
    x = [start, 0]
    k = 0
    prec = equation2.prec + 1
    while prec > equation2.prec and k <= 100:
        w = -1 if (-f * yl ** 3 - e * x[0] ** 2) / d < 0 else 1
        x[1] = w * abs((-f * yl ** 3 - e * x[0] ** 2) / d) ** (1 / 3)
        prec = abs(x[1] - x[0])
        x[0] = x[1]
        k += 1
    return x[0]
```

equation.py

```
class Equation:
    """
    Клас, призначений для зберігання даних, пов'язаних з рівняннями.

    ---

    Атрибути
    --------

    index : int
        Індекс, що відповідає за тип рівнянь.
    parameters  : float[]
        Список коефіцієнтів у цьому рівнянні.
    meth  : int
        Індекс, що відповідає за метод розв'язання.
    prec    : float
        Точність розв'язання.

    Методи
    ------
    """

    def __init__(self, index, parameter, method, precision):
        """
        Конструктор класу.
```

```
Аргументи
---------
index : int
    Індекс, що відповідає за тип рівнянь.
parameter  : float[]
    Список коефіцієнтів у цьому рівнянні.
method  : int
    Індекс, що відповідає за метод розв'язання.
precision    : float
    Точність розв'язання.
"""

self.index = index
self.parameters = parameter
self.meth = method
self.prec = precision
```