МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н. Каразіна
Факультет комп’ютерних наук

Розрахунково-графічна робота

з дисципліни «крос-платформне програмування»

на тему: «Згадати все...»

Виконав:
Студент групи КС-23
Терещенко Є.Ю
Перевірив:
Канд. фіз.-мат. наук
Споров О. Є.

ХАРКІВ – 2023

ЗМІСТ

ВСТУП

В сучасному світі програмування, об'єктно-орієнтоване програмування (ООП) є одним з найбільш поширених підходів до розробки програмного забезпечення. В рамках курсу ООП на Java було розглянуто ряд принципів та понять, які дозволяють ефективно використовувати цей підхід для створення програм. Один з прикладів використання ООП може бути знайдено в розрахунково-графічній роботі, яку ми розглядатимемо.

У цій роботі було створено набір класів для представлення функцій однієї змінної, що задається різними шляхами. Це дозволило створити систему, яка може працювати з різними типами функцій, що задаються у різних форматах. Далі, систему було розширено можливостями зберігання функцій у різних структурах даних, зокрема у TreeMap та TreeSet. Це дозволило досягти оптимальної швидкодії та ефективності виконання операцій з функціями.

Крім цього, було додано графічний інтерфейс для користувача. Це дозволило зробити систему більш доступною та зручною у використанні. Завдяки графічному інтерфейсу користувач може легко створювати та редагувати функції, а також побудовувати графіки для їх відображення.

Мета роботи полягає в тому, щоб згадати основні відомості, вивчені в рамках курсу ООП на Java, та застосувати їх у практичній розрахунково-графічній роботі. Основною ціллю роботи є створення програм для роботи з функціями та програми із графічним інтерфейсом для побудування графіків. В результаті цієї роботи студенти зможуть поглибити свої знання про ООП та використати їх у практичному застосуванні.

СТИСЛИЙ ГЛОСАРІЙ

Функція є математичним поняттям, яке відображає відношення між елементами двох множин, де кожному елементу першої множини відповідає один і тільки один елемент другої множини. Головна властивість функції - її графік, який відображає геометричний образ функції.

Задати функцію означає встановити правило, за яким можна визначити значення функції для кожного значення незалежної змінної. Це важливо для подальшого використання функції в обчисленнях та моделюванні.

Похідна функції – це величина, яка характеризує швидкість зміни функції в кожній точці її графіка. Це поняття є основним в диференціальному численні та допомагає в розрахунках, де важлива швидкість зміни функції.

Інтерполяція – це метод знаходження проміжних значень функції, який використовується, коли наявний лише дискретний набір точок на графіку функції. Інтерполяція допомагає оцінити значення функції в тих точках, які не входять до початкового набору точок.

Аналітичний спосіб задання функції полягає у встановленні зв'язку між аргументом і значенням функції за допомогою математичної формули. Це дає можливість отримувати значення функції шляхом простих математичних операцій.

Табличний спосіб задання функції полягає у встановленні зв'язку між аргументом і значенням функції на основі дискретного набору точок. Цей метод часто використовується в практичних застосуваннях, де набір точок може бути отриманий експериментально або з інших джерел.

Графік функції – це геометричне відображення значень функції на координатній площині. На графіку функції можна визначити значення функції для будь-якого значення незалежної змінної та відобразити основні характеристики функції, такі як максимуми, мінімуми, точки перетину з осями координат та інші.

РОЗДІЛ 1 ОСНОВНА ЧАСТИНА

1.1 Теоретичні відомості

Математичні методи

Для того, щоб отримати значення похідної, можна використати чисельний метод диференціювання за допомогою центральної кінцево-разностної формули, яка базується на трьох точках:

$$f'(x)_i \approx \frac{f(x_i + 1) - f(x_i - 1)}{2 * h}, \quad (1.1)$$

де x_i – точка, в якій потрібно знайти похідну $x_{i\pm1} = x_{i\pm h}$;

h – крок, з яким обчислена похідна.

Для визначення значення h, при якому обчислена похідна досягає заданої точності ε, потрібно послідовно обчислити значення $D_n = f'(x_i)$ зі зменшуваними кроками h $(h_n = 0.1 * h_n - 1)$. Обчислення слід продовжувати, доки не буде виконана принаймні одна з нерівностей:

$$|D_n + 1 - Dn| \geq |D_n - D_n - 1| \text{ або } |D_n - D - n1| < \varepsilon, \quad (1.2)$$

де D_n – послідовність;

ε – задана точність.

Щоб знайти похідну таблично заданої функції, ми будуємо інтерполяційний поліном та обчислюємо його похідну за відповідним правилом.

1.1.1 Інтерполяційний поліном

Для ефективної роботи з функціями, які задані у вигляді таблиці, необхідно отримувати значення цих функцій в проміжках між вузлами таблиці. Цього можна досягти за допомогою методів інтерполяції.

Часто інтерполяційне завдання формулюється наступним чином: маємо функцію $y = f(x)$, відому в $N + 1$ точці (x_0, x_0), (x_1, x_1), (x_n, x_n), де

$y_k = f(x_k)$, а значення x_k лежать в інтервалі $[a, b]$ і задовольняють умови $a \le x_0 \in x_1 \in \cdots \in x_N \le b$. Необхідно знайти значення функції $f(x)$ в проміжках між цими точками табуляції на інтервалі $[a, b]$.

Якщо маємо точки (x_k, y_k), що відомі з високою точністю, то можна побудувати інтерполяційний поліном $P(x)$, який проходить через них. Значення інтерполяційного полінома $P(x)$ в точці x, що належить інтервалу $[x_0, x_N]$, називається значенням інтерполяції. Існує багато методів для побудови інтерполяційного полінома $P(x)$.

1.1.2 Інтерполяційний поліном Лагранжа

Жозеф Луї Лагранж, французький математик (1736 - 1813), запропонував метод знаходження інтерполяційного полінома $P(x)$, який базується на $N + 1$ точках (x_0, y_0), (x_1, y_1), ..., (x_k, y_k) і має ступінь не більше N. Згідно з цим методом, поліном може бути записаний у вигляді:

$$P_N = \sum_{k=0}^{N} y_k L_{n,k}(x), \qquad (1.3)$$

де P – інтерполяційний поліном;

N – точки (x_0, y_0), (x_1, y_1), ..., (x_k, y_k);

$L_{n,k}(x)$ – коефіцієнти полінома Лагранжа, заснованого на цих вузлах.

Вирази для коефіцієнтів $L_{n,k}(x)$ можуть бути записані у вигляді:

$$p_j(x) = k(x - x_1)(x - x_2) \ldots (x - x_{j-1})(x - x_{j+1}) \ldots$$

$$\ldots(x - x_n) = k \prod_{i=1,\, i \ne j}^{n} (x - x_i) = \prod_{i=1,\, i \ne j}^{n} \frac{(x - x_i)}{(x_j - x_i)} \qquad (1.4)$$

Для кожного фіксованого k коефіцієнти полінома Лагранжа $L_{N,k}(x)$ мають властивості: $L_{N,K}(x) = 1$, коли $j = k$, $L_{N,K}(x) = 0$, коли $i \neq k$.

Раніше для обчислення інтерполяції функцій приймали розділення задачі на два етапи. На першому етапі визначали коефіцієнти інтерполюючої функції, а на другому етапі використовували ці коефіцієнти для обчислення полінома. Однак на сьогоднішній день, зазвичай коефіцієнти обчислюються на кожному кроці обчислення полінома.

1.1.2 Пошук коренів рівняння виду: $f(x) = 0$

Ця процедура розбивається на два етапи. На першому етапі відбувається локалізація кореня, тобто визначається відрізок $[a, b]$, на якому досліджувана функція $f(x)$ має лише один корінь. Цей відрізок називається відрізком локалізації кореня x На другому етапі відбувається уточнення розташування кореня на відрізку локалізації.

Для точнішого визначення положення кореня можна використовувати метод січних, який є модифікованою версією методу Ньютона з заміною похідної на її чисельне наближення за допомогою різницевих рівнянь. Цей метод має два кроки, для обчислення нового наближення $x^{(k+1)}$ до кореня потрібно знати два попередні наближення $x^{(k)}$ та $x^{(k-1)}$. У даному методі нове наближення до кореня обчислюється за наступною формулою:

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k-1)} - x^{(k)}}{f(x^{(k-1)})) - f(x^{(k)})} f(x^{(k)}), \quad (1.5)$$

При обчисленні кореня рівняння ми вважатимемо, що корінь знайдений із достатньою точністю, якщо виконується нерівність:

$$|x^{(k+1)} - x^{(k)}| \in \varepsilon, \qquad (1.6)$$

де $x^{(k+1)}$ та $x^{(k)}$ – два послідовні наближення до кореня;

ε – точність, з якою ми хочемо знайти корінь.

1.2 Основна частина

Функції, які потрібно продиференціювати, можуть бути визначені різними способами, такими як за допомогою формули, таблиці значень, або рішення рівняння. Ці функції мають спільну властивість: за задання значення незалежної змінної x можна отримати значення функції $f(x)$. Щоб забезпечити можливість диференціювати будь-які функції однієї змінної, не залежно від способу їх створення, можна оголосити інтерфейс, який визначає, як можна обчислити результат, отримавши один аргумент. Такий інтерфейс (інтерфейс взаємодії) описує очікувану поведінку класу, не згадуючи конкретних дій. Інтерфейс може бути використаний в якості параметра функції чисельного диференціювання. Це дозволяє диференціювати будь-який об'єкт, який реалізує цей інтерфейс.

На початку буде створено інтерфейс `Evaluatable`, який забезпечуватиме можливість обчислення значення по одному аргументу для об'єкта класу, що реалізує цей інтерфейс.

Ми створимо клас `FFunction`, який імплементує інтерфейс та відображає одну з необхідних функцій для виконання обчислень згідно з формулою (1.7).

$$f(x) = exp(-x)^2 * sin(x), \qquad (1.7)$$

Наступний порядок дій:

- Набір класів створено для створення функції, яка визначається як набір точок, що зберігаються у файлі.

- Було створено класи для зберігання значень точок, зокрема абстрактний клас `Point`.
- Для зручності роботи з точкою в двовимірному просторі було створено похідний клас `Point2D`, який успадковує від класу `Point`.

Після цього будуть створені класи для виконання інтерполяції та обчислення таблично заданої функції для необхідного значення аргументу. Зокрема, будуть створені класи `Interpolator` (який є абстрактним і містить загальні методи для функцій інтерполяції даних та реалізує інтерфейс `Evaluatable`), `ListInterpolation` (спадкоємець класу `Interpolator`) та `FileListInterpolation` (спадкоємець класу `ListInterpolation`, що має методи для читання та запису інформації з файлу).

1.3 Розширення основної частини програми

В основній частині програми містяться класи, які призначені для зберігання даних функції, заданої у вигляді таблиці, у структурі типу ArrayList, а також запису цих даних до файлу.

Було розширено клас з метою зберігання даних функції, які задані таблично у структурі типу TreeSet. Для збереження цих даних до файлу в програмі використовувалися наступні класи:

- Для зчитування або запису текстових файлів використовуються класи `FileReader` та `FileWriter`.
- Клас `BufferedReader` призначений для буферизованого зчитування тексту з символьного потоку введення. Використання буфера допомагає підвищити продуктивність читання даних з потоку.
- Клас `PrintWriter` надає можливість форматованого друкування об'єктів до текстового потоку виводу.

- Клас `StringTokenizer` служить для розбиття рядка на частини, які називаються токенами. Токенізація полягає в розділенні послідовності символів на окремі частини.

Клас TreeSet <E> використовує деревову структуру для зберігання об'єктів у відсортованому порядку за зростанням і є загальним класом. Він успадковує клас `AbstractSet` і реалізує інтерфейс `NavigableSet`, який включає інтерфейс SortedSet.

Класи `TreeSetInterpolation` та `TreeMapInterpolation` були створені для зберігання даних функції, що задана таблично, у структурах даних TreeSet або TreeMap. Ці класи успадковують клас `Interpolator`, який реалізовує інтерфейс `Evaluatable`. Вони також зберігають об'єкти типу `Point2D`, які є координатами точок, отриманих під час обчислення функції. Цей підхід є аналогом до підходу, використаного у базовій частині програми зі структурою ArrayList.

Методи, що визначені для структур TreeSet та TreeMap, були перевизначені в класах `TreeSetInterpolation` та `TreeMapInterpolation`. Ось деякі з цих методів:

1. Метод `addPoint(Point2D pt)` – додає нову точку до структури;
2. Метод `numPoints()` – повертає кількість точок у структурі;
3. Метод `sort()` – сортує точки, але він є порожнім, оскільки в структурі TreeSet всі об'єкти зберігаються відсортовано, а в TreeMap всі об'єкти автоматично сортуються по зростанню їх ключів.
4. Метод `setPoint(int i, Point2D pt)` – встановлює значення точки за певним індексом;
5. Метод `getPoint(int i)` – повертає значення точки з певним індексом;
6. Метод `clear()` – видаляє всі точки зі структури;
7. Метод `removeLastPoint()` – видаляє останню точку зі структури;

У Лістингу А.2 наведено повний код реалізації структури типу TreeSet, а у Лістингу А.3 - структури типу TreeMap.

Для забезпечення запису та читання даних нової структури з файлу, ми створимо класи `FileTreeSetInterpolation` та `FileTreeMapInterpolation`. `FileTreeSetInterpolation` базується на структурі TreeSet та є нащадком класу `TreeSetInterpolation`, тоді як `FileTreeMapInterpolation` використовує структуру TreeMap та є нащадком класу `TreeMapInterpolation`. Обидва класи успадковують всі поля та методи від своїх батьківських класів та містять додаткові методи для читання та запису файлу.

1. `readFromFile(String fileName)` – метод для зчитування файлу;
2. `writeToFile(String fileName)` – метод для запису в файл.

У Лістингу А.1 наведено повний код реалізації структури типу TreeSet, а у Лістингу А.4 - структури типу TreeMap.

Наведені результати роботи для структури типу TreeSet та TreeMap у Лістингу Б.1, Лістингу Б.2, Лістингу Б.3 та Лістингу Б.4.

1.4 Обчислення аналітичної функції

Щоб реалізувати цю задачу була використана стороння бібліотека edu.hws.jcm.data, що є компонентом Java для математичних обчислень. Вона містить необхідні методи для обчислення функції, що задана як формула у вигляді строки.

Був створений клас `AnalyticalTool`, який наступні класи з бібліотеки:

1. Клас `Parser` – використовується для розбору виразу, представленого у вигляді рядка і для створення відповідного об'єкта типу Expression. Змінні додаються до об’єкту `Parser`;

2. Клас `Variable` – представляє змінну та дозволяє змінювати її значення, Expression, що представляє математичні вирази у вигляді формул;
3. Клас `Expression` – представляє математичні вирази у вигляді формул.

Аналітично задана функція вказується у вигляді рядка в процесі роботи програми. В програмній реалізації використовується меню, яке дозволяє користувачу обрати для вводу або функцію з параметром, або ні. Робота меню виконується за допомогою операторів управління switch-case.

Тобто програма має два пункти:

1. Функція з параметром – функція представляє змінну x та параметр a як об'єкти класу `Variable`, що дозволяє об'єкту типу `Parser` визначити константні або незмінні значення при додаванні за допомогою спеціальної функції. Клас `Expression` перетворює значення, повернене парсером у вираз. Далі йде процес математичного обчислення у циклі;
2. Функція без параметру – функція обчислення формули без параметру реалізована аналогічно, але без додавання у парсер параметру;

Повний код реалізації наведений у Лістінгу А.5.

1.5 Графічний інтерфейс

Для будування графіків функцій та роботи з графічним інтерфейсом та вікнами в реалізації цієї задачі була використана бібліотека JFreeChart з відкритим вихідним кодом, яка включає в себе компоненти AWT і Swing [2].

Був створений клас `MainFrame`, який наслідується від класу `JFrame`. У конструкторі без параметрів обробляються компоненти фрейму, отримані в результаті значення передаються для побудови графіку. Графік створюється у методі `createChart()`. Повний код реалізації наведений у Лістингу А.6.

Інтерфейс містить дві кнопки: кнопка `Calculate`, що будує графік та кнопка Exit, за допомогою якої можна вийти з програми. Є поля вводу, що дозволяє користувачу вказати саму функцію, початкову та кінцеву межу функції, а також крок, з яким функція буде змінюватися. Функцію програма читає у форматі строки. Інтерфейс програми наведений у Лістингу Б.7.

При коректному вводі даних програма відобразить графік. Для створення нового графіку не потрібно перезапускати програму - достатньо ввести необхідні дані у відповідні поля та натиснути на кнопку `Calculate`. Графік можна масштабувати, змінюючи кордони або значення кроку.

Приклад відображення графіку наведено у Лістингу Б.8.

ВИСНОВКИ

Під час виконання розрахунково-графічної роботи було отримано значний досвід у роботі з математичним матеріалом та бібліотекою JFreeChart. Було підтверджено та використано фундаментальні знання з курсу ООП, що допомогло реалізувати необхідний функціонал у програмі.

Окрім того, в процесі роботи було використано концепції похідних з математики для обчислення диференціалу функції. Була реалізована структура даних TreeSet для зберігання точок функції та їх інтерполяції.

Результатом роботи стала можливість візуалізувати дані та показати користувачу роботу функції на графіку. Завдяки цьому користувач може легко оцінити результат та зрозуміти, як функція поводиться в залежності від вхідних параметрів. Робота над проектом дозволила поглибити знання у програмуванні на мові Java та отримати практичні навички в роботі з математичним матеріалом.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Java Components for Mathematics. URL: http://math.hws.edu/javamath/.
2. Бібліотека JFreeChart. URL: https://www.jfree.org/jfreechart/.
3. Oracle Java SE Documentation. URL: https://docs.oracle.com/en/java/javase/index.html.
4. Java Tutorials. URL: https://docs.oracle.com/javase/tutorial/.
5. "Data Structures and Algorithms in Java" by Robert Lafore. ISBN-10: 0672324539.
6. "Java: A Beginner's Guide" by Herbert Schildt. ISBN-10: 1259589315.
7. "Java How To Program" by Paul Deitel and Harvey Deitel. ISBN-10: 0134743350.

ДОДАТОК А ПРИКЛАДИ КОДІВ

(без пакету consoleTasks)

Лістинг А.1 – Клас FileDataInterpolationTreeSet.

```
package own;

import consoleTasks.Point2D;
import java.io.*;
import java.util.Scanner;
import java.util.StringTokenizer;
public class FileDataInterpolationTreeSet extends
DataInterpolationTreeSet {
    public FileDataInterpolationTreeSet() {
        super();
    }
    public void readFromFile(String fileName) throws IOException {
        BufferedReader in = new BufferedReader(new
FileReader(fileName));
        String s = in.readLine();
        clear();
        while ((s = in.readLine()) != null) {
            StringTokenizer st = new StringTokenizer(s);
            double x = Double.parseDouble(st.nextToken());
            double y = Double.parseDouble(st.nextToken());
            addPoint(new Point2D(x, y));
        }
        in.close();
    }
    public void writeToFile(String fileName) throws IOException {
        File file = new File(fileName);
        if (!file.exists()) {
            file.createNewFile();
        }
```

```
        PrintWriter out = new PrintWriter(new
FileWriter(fileName));
        out.printf("%9s%25s\n", "x", "y");
        for (int i = 0; i < numPoints(); i++) {
            out.println(getPoint(i).getX() + "\t" +
getPoint(i).getY());
        }
        out.close();
    }
    public static void main(String[] args) {
        FileDataInterpolationTreeSet fun = new
FileDataInterpolationTreeSet();
        int num;
        double x;
        Scanner in = new Scanner(System.in);
        do {
            System.out.print("Кількість точок: ");
            num = in.nextInt();
        } while (num <= 0);
        for (int i = 0; i < num; i++) {
            x = 1.0 + (5.0 - 1.0) * Math.random();
            fun.addPoint(new Point2D(x, Math.sin(x)));
        }
        System.out.println("Інтерполяція по: " + fun.numPoints() +
" точкам");
        System.out.println("Не отсортований набір: ");
        for (int i = 0; i < fun.numPoints(); i++){
            System.out.println("Точка " + (i + 1) + ": " +
fun.getPoint(i));
        }
        System.out.println("Отсортований набір: ");
        System.out.println(" -----" + fun.numPoints() + "------");
        for (int i = 0; i < fun.numPoints(); i++){
```

```
            System.out.println("Точка " + (i + 1) + ": " +
fun.getPoint(i));
        }
        System.out.println("Мінімальне значення x: " +
fun.getPoint(0).getX());
        System.out.println("Максимальне значення x: " +
                fun.getPoint(fun.numPoints() - 1).getX());
        System.out.println("Зберігаємо у файл");
        try {
            fun.writeToFile("dataTreeSet.dat");
        } catch (IOException ex) {
            ex.printStackTrace();
            System.exit(-1);
        }
        System.out.println("Зчитуємо з файла");
        fun.clear();
        try {
            fun.readFromFile("dataTreeSet.dat");
        } catch (IOException ex) {
            ex.printStackTrace();
            System.exit(-1);
        }
        System.out.println("Дані з файлу: ");
        for (int i = 0; i < fun.numPoints(); i++) {
            System.out.println("Точка " + (i + 1) + ": " +
fun.getPoint(i));
        }
        System.out.println("Мінімальне значення x: " +
fun.getPoint(0).getX());
        System.out.println("Максимальне значення x: " +
                fun.getPoint(fun.numPoints() - 1).getX());
        x = 0.5 * (fun.getPoint(0).getX() +
fun.getPoint(fun.numPoints() - 1).getX());
```

```
        System.out.println("Занчення інтерполяції fun(" + x + ") =
" +
                fun.evalf(x));
        System.out.println("Точне значення sin(" + x + ") = " +
Math.sin(x));
        System.out.println("Абсолютна похибка = " +
Math.abs(fun.evalf(x) -
                Math.sin(x)));
        System.out.println("Готуємо дані для рахунку");
        fun.clear();
        for (x = 1.0; x <= 7.0; x += 0.1) {
            fun.addPoint(new Point2D(x, Math.sin(x)));
        }
        try {
            fun.writeToFile("TblFuncTreeSet.dat");
        } catch (IOException ex) {
            ex.printStackTrace();
            System.exit(-1);
        }
    }
}
```

Лістинг А.2 – Клас DataInterpolationTreeSet.

```
package own;

import consoleTasks.Interpolator;
import consoleTasks.Point2D;
import java.util.Iterator;
import java.util.TreeSet;
public class DataInterpolationTreeSet extends Interpolator {
    private TreeSet<Point2D> data = new TreeSet<>();
    public DataInterpolationTreeSet(TreeSet<Point2D> data) {
        this.data = data;
```

```
}
public DataInterpolationTreeSet() {
}
public void clear() {
    data.clear();
}
public int numPoints() {
    return data.size();
}
public void addPoint(Point2D pt) {
    data.add(pt);
}

public Point2D getPoint(int i) {
    if (i == 0) {
        return data.first();
    }
    int n = 0;
    Iterator<Point2D> it = data.iterator();
    Point2D curr = null;
    while (it.hasNext() && n <= i) {
        curr = it.next();
        n++;
    }
    return curr;
}
public void setPoint(int i, Point2D pt) {
    data.add(pt);
}
public void removeLastPoint() {
    data.remove(data.size() - 1);
}
public void sort() {
}
```

```
}
```

Лістинг А.3 – Клас DataInterpolationTreeMap.

```
package own;

import consoleTasks.Interpolator;
import consoleTasks.Point2D;
import java.util.Iterator;
import java.util.Map;
import java.util.TreeMap;
public class DataInterpolationTreeMap extends Interpolator {
    private TreeMap<Double, Double> data = new TreeMap<>();
    public DataInterpolationTreeMap(TreeMap<Double, Double> data)
{
        this.data = data;
    }
    public DataInterpolationTreeMap() {
    }
    public void clear() {
        data.clear();
    }
    public int numPoints() {
        return data.size();
    }
    public void addPoint(Point2D pt) {
        data.put(pt.getX(), pt.getY());
    }
    public Point2D getPoint(int i) {
        if (i < 0 || data.size() <= i) {
            throw new
IndexOutOfBoundsException("IndexOutOfBoundsException");
        }
        Map.Entry<Double, Double> e = null;
        Iterator<Map.Entry<Double, Double>> it =
data.entrySet().iterator();
```

```
        while (0 <= i--) {
            e = it.next();
        }
        double x = e.getKey();
        double y = e.getValue();
        return new Point2D(x, y);
    }
    public void setPoint(int i, Point2D pt) {
        data.put(pt.getX(), pt.getY());
    }
    public void removeLastPoint() {
        data.remove(data.size() - 1);
    }
    public void sort() {
    }
}
```

Лістинг А.4 – Клас FileDataInterpolationTreeMap.

```
package own;

import consoleTasks.Point2D;

import java.io.*;
import java.util.*;

public class FileDataInterpolationTreeMap extends
DataInterpolationTreeMap {
    public FileDataInterpolationTreeMap() {
        super();
    }
    public void readFromFile(String fileName) throws IOException {
        BufferedReader in = new BufferedReader(new
FileReader(fileName));
        String s = in.readLine();
```

```
        clear();
        while ((s = in.readLine()) != null) {
            StringTokenizer st = new StringTokenizer(s);
            double x = Double.parseDouble(st.nextToken());
            double y = Double.parseDouble(st.nextToken());
            addPoint(new Point2D(x, y));
        }
        in.close();
    }
    public void writeToFile(String fileName) throws IOException {
        File file = new File(fileName);
        if (!file.exists()) {
            file.createNewFile();
        }
        PrintWriter out = new PrintWriter(new
FileWriter(fileName));
        out.printf("%9s%25s\n", "x", "y");
        for (int i = 0; i < numPoints(); i++) {
            out.println(getPoint(i).getX() + "\t" +
getPoint(i).getY());
        }
        out.close();
    }
    public static void main(String[] args) {
        FileDataInterpolationTreeMap fun = new
FileDataInterpolationTreeMap();
        int num;
        double x;
        Scanner in = new Scanner(System.in);
        do {
            System.out.print("Кількість точок: ");
            num = in.nextInt();
        } while (num <= 0);
        for (int i = 0; i < num; i++) {
```

```
            x = 1.0 + (5.0 - 1.0) * Math.random();
            fun.addPoint(new Point2D(x, Math.sin(x)));
        }
        System.out.println("Інтерполяція по: " + fun.numPoints() +
" точкам");
        System.out.println("Не отсортований набір: ");
        for (int i = 0; i < fun.numPoints(); i++){
            System.out.println("Точка " + (i + 1) + ": " +
fun.getPoint(i));
        }
        System.out.println("Отсортований набір: ");
        System.out.println(" -----" + fun.numPoints() + "------");
        for (int i = 0; i < fun.numPoints(); i++){
            System.out.println("Точка " + (i + 1) + ": " +
fun.getPoint(i));
        }
        System.out.println("Мінімальне значення x: " +
fun.getPoint(0).getX());
        System.out.println("Максимальне значення x: " +
                fun.getPoint(fun.numPoints() - 1).getX());
        System.out.println("Зберігаємо у файл");
        try {
            fun.writeToFile("dataTreeMap.dat");
        } catch (IOException ex) {
            ex.printStackTrace();
            System.exit(-1);
        }
        System.out.println("Зчитуємо з файла");
        fun.clear();
        try {
            fun.readFromFile("dataTreeMap.dat");
        } catch (IOException ex) {
            ex.printStackTrace();
            System.exit(-1);
```

```
        }
        System.out.println("Дані з файлу: ");
        for (int i = 0; i < fun.numPoints(); i++) {
            System.out.println("Точка " + (i + 1) + ": " +
fun.getPoint(i));
        }
        System.out.println("Мінімальне значення x: " +
fun.getPoint(0).getX());
        System.out.println("Максимальне значення x: " +
                fun.getPoint(fun.numPoints() - 1).getX());
        x = 0.5 * (fun.getPoint(0).getX() +
fun.getPoint(fun.numPoints() - 1).getX());
        System.out.println("Занчення інтерполяції fun(" + x + ") =
" +
                fun.evalf(x));
        System.out.println("Точне значення sin(" + x + ") = " +
Math.sin(x));
        System.out.println("Абсолютна похибка = " +
Math.abs(fun.evalf(x) -
                Math.sin(x)));
        System.out.println("Готуємо дані для рахунку");
        fun.clear();
        for (x = 1.0; x <= 7.0; x += 0.1) {
            fun.addPoint(new Point2D(x, Math.sin(x)));
        }
        try {
            fun.writeToFile("TblFuncTreeMap.dat");
        } catch (IOException ex) {
            ex.printStackTrace();
            System.exit(-1);
        }
    }
}
```

Лістинг А.5 – Клас AnalyticalTool.

```
package own;

import edu.hws.jcm.data.Expression;
import edu.hws.jcm.data.Parser;
import edu.hws.jcm.data.Variable;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class AnalyticalTool {
    public static final BufferedReader in = new BufferedReader(new
            InputStreamReader(System.in));
    public static void main(String[] args) {
        boolean exit = false;
        String func;
        double from, to, step;
        int choice;
        while (!exit) {
            System.out.println("\n1. Функція з параметром");
            System.out.println("2. Функція без параметра");
            System.out.print("Вибір: ");
            choice = scanInteger();
            switch (choice) {
                case 1:
                    System.out.println("Введіть функцію: ");
                    func = scanLine();
                    do{
                        System.out.println("Від:");
                        from = scanDouble();
                        System.out.println("До:");
                        to = scanDouble();
                    } while (from >= to);
                    System.out.println("Крок: ");
```

```
                    step = scanDouble();
                    withParam(func, from, to, step);
                    break;
                case 2:
                    System.out.println("Введіть функцію: ");
                    func = scanLine();
                    do{
                        System.out.println("Від:");
                        from = scanDouble();
                        System.out.println("До:");
                        to = scanDouble();
                    } while (from >= to);
                    System.out.println("Крок: ");
                    step = scanDouble();
                    withoutParam(func, from, to, step);
                    break;
                default:
                    System.out.println("Error");
            }
        }
    }
       public static void withParam(String funStr, double from,
double to, double step)
    {
        Parser parser = new Parser(Parser.STANDARD_FUNCTIONS);
        System.out.println("1. Функція с параметром");
        Variable par = new Variable("a");
        Variable var = new Variable("x");
        parser.add(var);
        parser.add(par);
        Expression fun = parser.parse(funStr);
        Expression der = fun.derivative(var);
        System.out.println("f(x) = " + fun.toString());
        System.out.println("f'(x) = " + der.toString());
```

```
        par.setVal(1.0);
        for (double x = from; x <= to; x += step) {
            var.setVal(x);
               System.out.println(x + "\t" + fun.getVal() + "\t" +
der.getVal());
        }
    }
      public static void withoutParam(String funStr, double from,
double to, double
            step) {
        Parser parser = new Parser(Parser.STANDARD_FUNCTIONS);
        System.out.println("2. Функція без параметра");
        Variable var = new Variable("x");
        parser.add(var);
                      edu.hws.jcm.data.ExpressionProgram   funs   =
parser.parse(funStr);
        Expression ders = funs.derivative(var);
        System.out.println("f(x) = " + funs.toString());
        System.out.println("f'(x) = " + ders.toString());
        for (double x = from; x <= to; x += step) {
            var.setVal(x);
              System.out.println(x + "\t" + funs.getVal() + "\t" +
ders.getVal());
        }
    }
    public static int scanInteger(){
        int num = 0;
        boolean check = true;
        while (check){
            check = false;
            try {
                num = Integer.parseInt(in.readLine());
            } catch (NumberFormatException | IOException e){
                check = true;
```

```
                System.err.println("Incorrect number format");
            }
        }
        return num;
    }
    public static double scanDouble(){
        double num = 0;
        boolean check = true;
        while (check){
            check = false;
            try {
                num = Double.parseDouble(in.readLine());
            } catch (NumberFormatException | IOException e){
                check = true;
                System.err.println("Incorrect number format");
            }
        }
        return num;
    }
    public static String scanLine(){
        String check = "";
        try {
            check = in.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return check;
    }
}
```

Лістинг А.6 – Клас MainFrame.

```
package gui;
```

```
import edu.hws.jcm.data.Expression;
import edu.hws.jcm.data.ExpressionProgram;
import edu.hws.jcm.data.Parser;
import edu.hws.jcm.data.Variable;
import org.jfree.chart.ChartColor;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.XYPlot;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;

public class MainFrame extends JFrame {
    private final JPanel mainPanel;
    private JTextField textFieldFrom = null;
    private JTextField textFieldTo = null;
    private JTextField textFieldStep = null;
    private JTextField textFieldFunc = null;
    private XYSeries series = null;
    private XYSeries der = null;
    private double start;
    private double stop;
    private double step;

    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            try {
                MainFrame frame = new MainFrame();
                frame.setVisible(true);
```

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
    }

    public MainFrame() {

        setTitle("GetDerivative");
        setResizable(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 1000, 800);

        mainPanel = new JPanel();
        mainPanel.setBorder(new EmptyBorder(5, 20, 5, 20));
        mainPanel.setLayout(new BorderLayout(100, 5));
        setContentPane(mainPanel);

        JPanel panelButtons = new JPanel();
        panelButtons.setLayout(new FlowLayout(FlowLayout.CENTER,
5, 0));
        mainPanel.add(panelButtons, BorderLayout.NORTH);

        JButton btnNewButtonPlot = new JButton("Calculate");
        btnNewButtonPlot.setPreferredSize(new Dimension(250, 30));
        btnNewButtonPlot.addActionListener(e -> {
            start = Double.parseDouble(textFieldFrom.getText());
            stop = Double.parseDouble(textFieldTo.getText());
            step = Double.parseDouble(textFieldStep.getText());
            series.clear();
            der.clear();
            Parser parser = new Parser(Parser.STANDARD_FUNCTIONS);
```

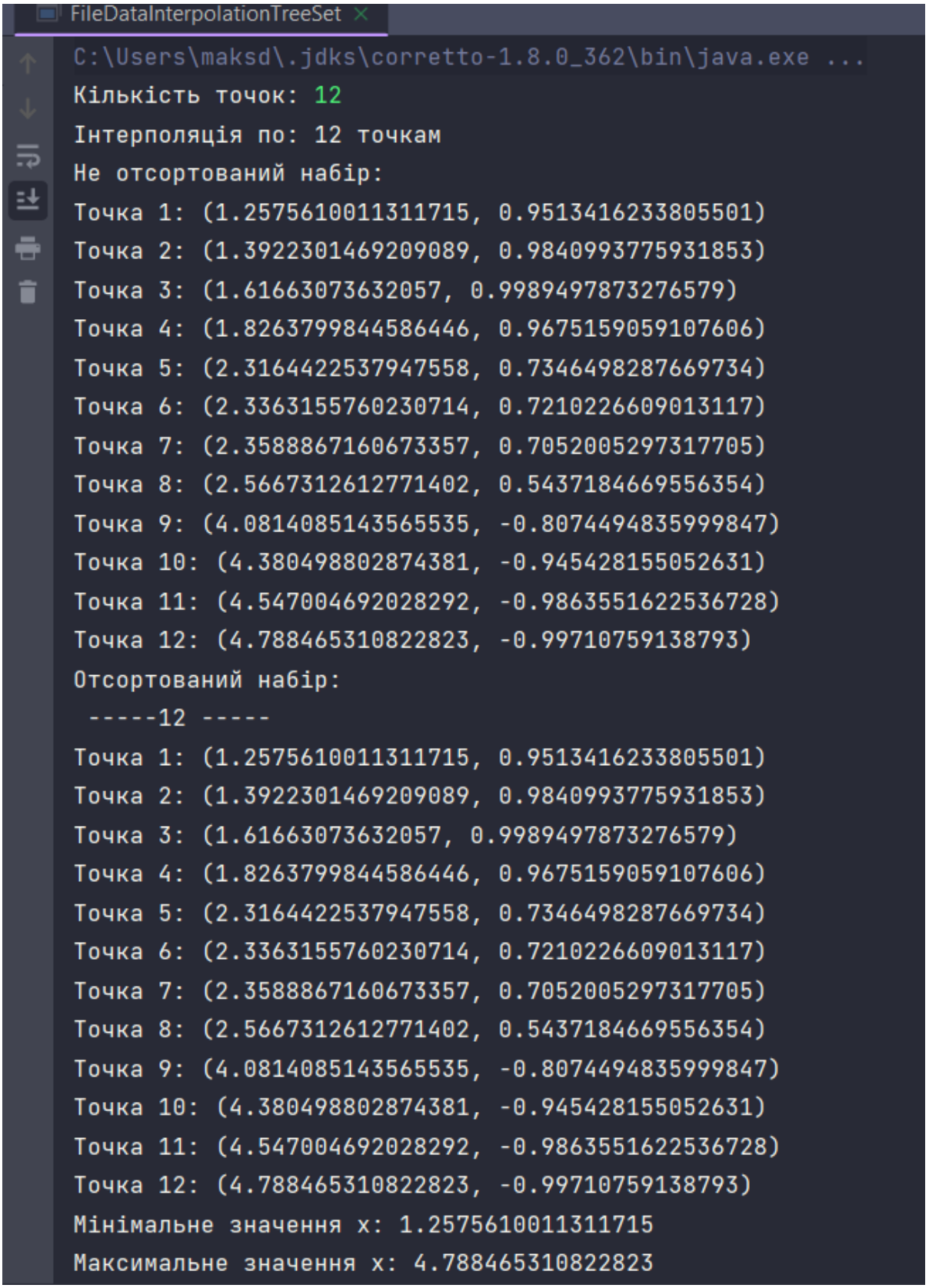
```
            Variable var = new Variable("x");
            Variable par = new Variable("a");
            String funStr =
String.valueOf(textFieldFunc.getText());
            parser.add(var);
            parser.add(par);
            ExpressionProgram funs = parser.parse(funStr);
            Expression ders = funs.derivative(var);
            double a = 0.5;
            par.setVal(a);
            for (double x = start; x < stop; x += step) {
                var.setVal(x);
                series.add(x, funs.getVal());
                der.add(x, ders.getVal());
            }
        });
        panelButtons.add(btnNewButtonPlot);

        JPanel panelData = new JPanel();
        panelData.setLayout(new FlowLayout(FlowLayout.CENTER, 20,
5));
        panelData.setPreferredSize(new Dimension(500, 100));
        mainPanel.add(panelData, BorderLayout.SOUTH);

        JLabel lblNewLabelfunc = new JLabel("f(x)= ");
        panelData.add(lblNewLabelfunc);

        textFieldFunc = new JTextField();
        textFieldFunc.setText("sin(x)/x");
        textFieldFunc.setColumns(15);
        panelData.add(textFieldFunc);

        JLabel lblNewLabelStart = new JLabel("From:");
        lblNewLabelStart.setPreferredSize(new Dimension(50, 50));
```

```
panelData.add(lblNewLabelStart);

textFieldFrom = new JTextField();
textFieldFrom.setText("-6");
textFieldFrom.setColumns(5);
panelData.add(textFieldFrom);

JLabel lblNewLabelStop = new JLabel("To:");
lblNewLabelStop.setPreferredSize(new Dimension(50, 50));
panelData.add(lblNewLabelStop);

textFieldTo = new JTextField();
textFieldTo.setText("6");
textFieldTo.setColumns(5);
panelData.add(textFieldTo);

JLabel lblNewLabelStep = new JLabel("Step:");
lblNewLabelStep.setPreferredSize(new Dimension(50, 50));

panelData.add(lblNewLabelStep);

textFieldStep = new JTextField();
textFieldStep.setText("0.1");
textFieldStep.setColumns(5);
panelData.add(textFieldStep);

JPanel panelChart = new JPanel();
panelChart.setLayout(new BorderLayout());
mainPanel.add(panelChart, BorderLayout.CENTER);

series = new XYSeries("f(x)");
der = new XYSeries("f'(x)");

XYSeriesCollection dataset = new XYSeriesCollection();
```

```
        dataset.addSeries(series);
        dataset.addSeries(der);

        JFreeChart chart = ChartFactory.createXYLineChart(
                "Function Graph",
                "x",
                "y",
                dataset
        );

        chart.getPlot().setBackgroundPaint(ChartColor.WHITE);
        chart.getPlot().setForegroundAlpha(0.8f);

        XYPlot plot = chart.getXYPlot();
        plot.setBackgroundPaint(Color.WHITE);
        plot.setDomainGridlinePaint(Color.LIGHT_GRAY);
        plot.setRangeGridlinePaint(Color.LIGHT_GRAY);

        ChartPanel chartPanel = new ChartPanel(chart);
        panelChart.add(chartPanel, BorderLayout.CENTER);
    }
}
```

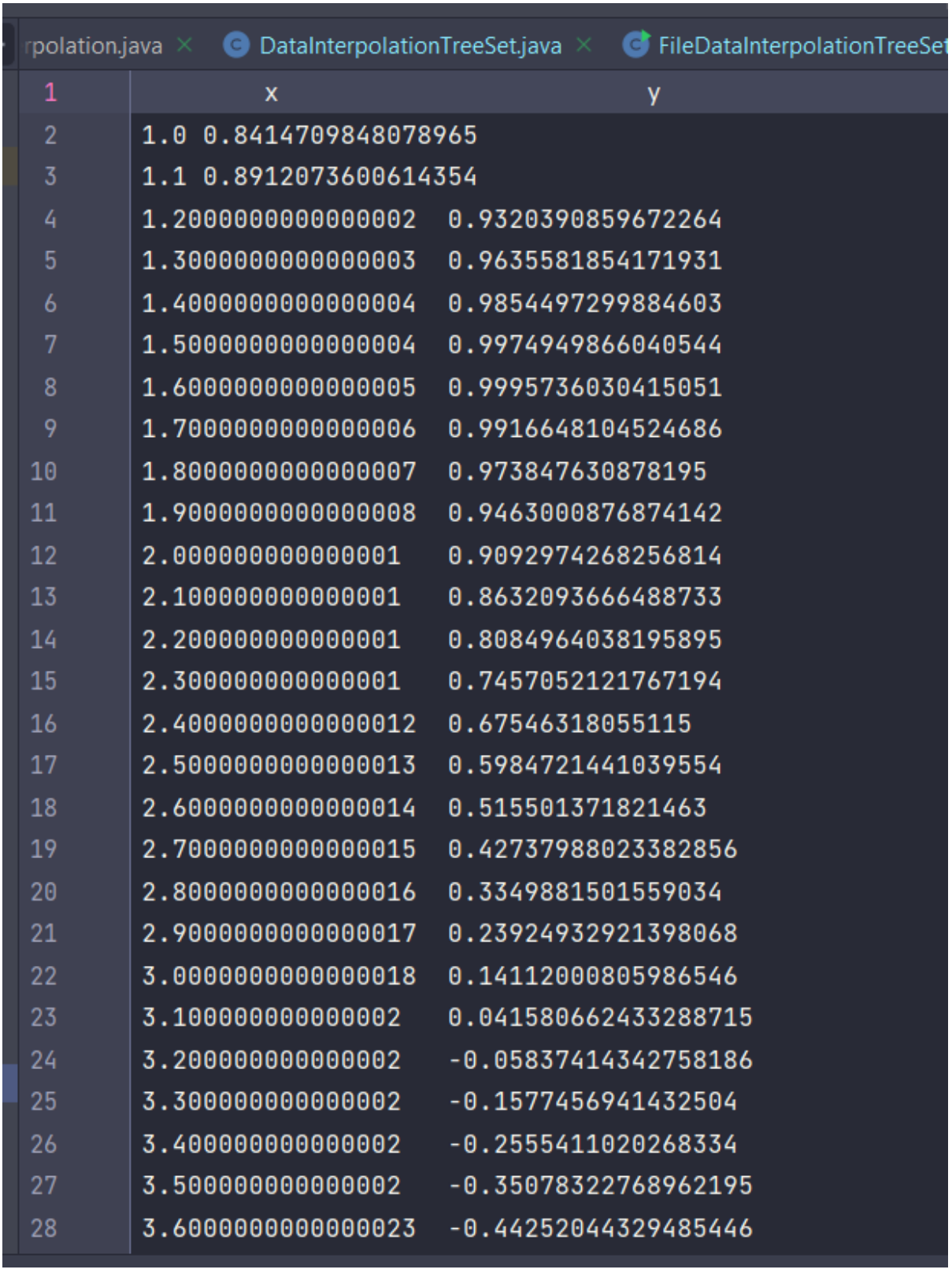
ДОДАТОК Б ЕКРАННІ ФОРМИ

Лістинг Б.1 – Результат роботи класу FileTreeSetInterpolation у консолі



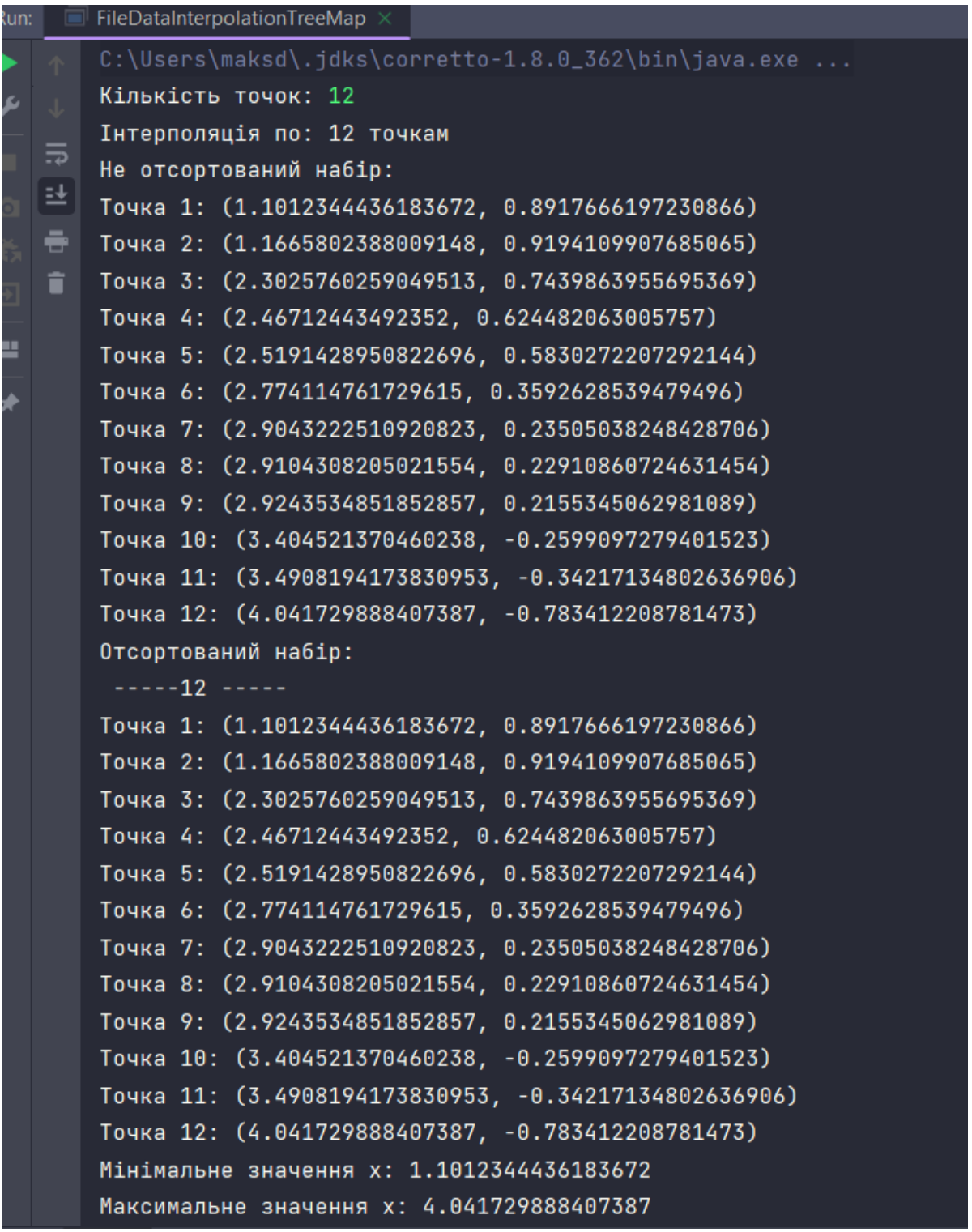
```
Мінімальне значення x: 1.2575610011311715
Максимальне значення x: 4.788465310822823
Зберігаємо у файл
Зчитуємо з файла
Дані з файлу:
Точка 1: (1.2575610011311715, 0.9513416233805501)
Точка 2: (1.3922301469209089, 0.9840993775931853)
Точка 3: (1.61663073632057, 0.9989497873276579)
Точка 4: (1.8263799844586446, 0.9675159059107606)
Точка 5: (2.3164422537947558, 0.7346498287669734)
Точка 6: (2.3363155760230714, 0.7210226609013117)
Точка 7: (2.3588867160673357, 0.7052005297317705)
Точка 8: (2.5667312612771402, 0.5437184669556354)
Точка 9: (4.0814085143565535, -0.8074494835999847)
Точка 10: (4.380498802874381, -0.945428155052631)
Точка 11: (4.547004692028292, -0.9863551622536728)
Точка 12: (4.788465310822823, -0.99710759138793)
Мінімальне значення x: 1.2575610011311715
Максимальне значення x: 4.788465310822823
Занчення інтерполяції fun(3.0230131559769973) = 0.11830179814685216
Точне значення sin(3.0230131559769973) = 0.11830179994669435
Абсолютна похибка = 1.7998421919518393E-9
Готуємо дані для рахунку

Process finished with exit code 0
```

Лістинг Б.2 – Результат роботи класу FileTreeSetInterpolation у файлі



	x	y
2	1.0	0.8414709848078965
3	1.1	0.8912073600614354
4	1.2000000000000002	0.9320390859672264
5	1.3000000000000003	0.9635581854171931
6	1.4000000000000004	0.9854497299884603
7	1.5000000000000004	0.9974949866040544
8	1.6000000000000005	0.9995736030415051
9	1.7000000000000006	0.9916648104524686
10	1.8000000000000007	0.973847630878195
11	1.9000000000000008	0.9463000876874142
12	2.000000000000001	0.9092974268256814
13	2.100000000000001	0.8632093666488733
14	2.200000000000001	0.8084964038195895
15	2.300000000000001	0.7457052121767194
16	2.4000000000000012	0.67546318055115
17	2.5000000000000013	0.5984721441039554
18	2.6000000000000014	0.515501371821463
19	2.7000000000000015	0.42737988023382856
20	2.8000000000000016	0.3349881501559034
21	2.9000000000000017	0.23924932921398068
22	3.0000000000000018	0.14112000805986546
23	3.100000000000002	0.041580662433288715
24	3.200000000000002	-0.05837414342758186
25	3.300000000000002	-0.1577456941432504
26	3.400000000000002	-0.2555411020268334
27	3.500000000000002	-0.35078322768962195
28	3.6000000000000023	-0.44252044329485446

Лістинг Б.3 – Результат роботи класу FileTreeMapInterpolation у консолі

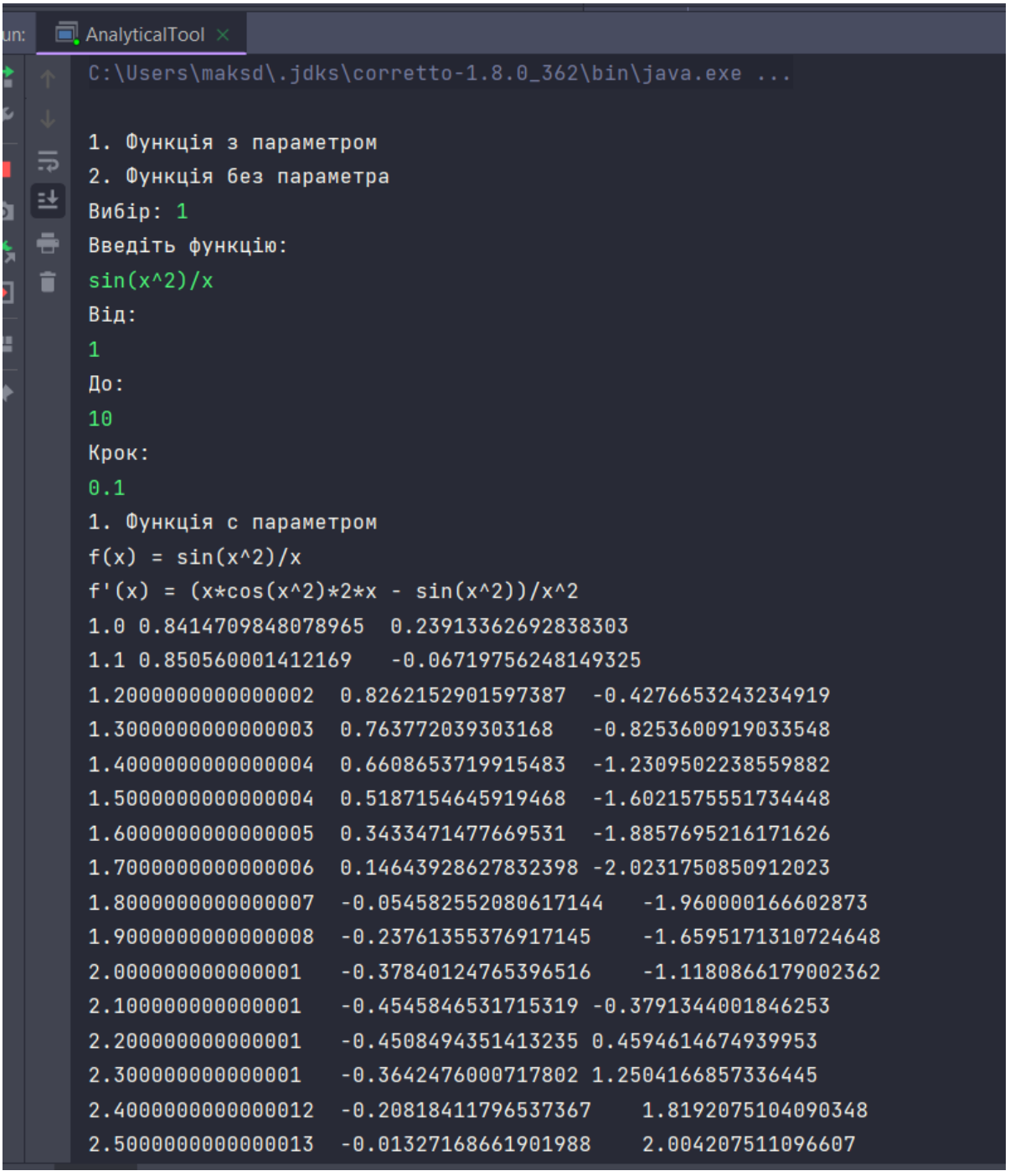


Лістинг Б.4 – Результат роботи класу FileTreeMapInterpolation у файлі

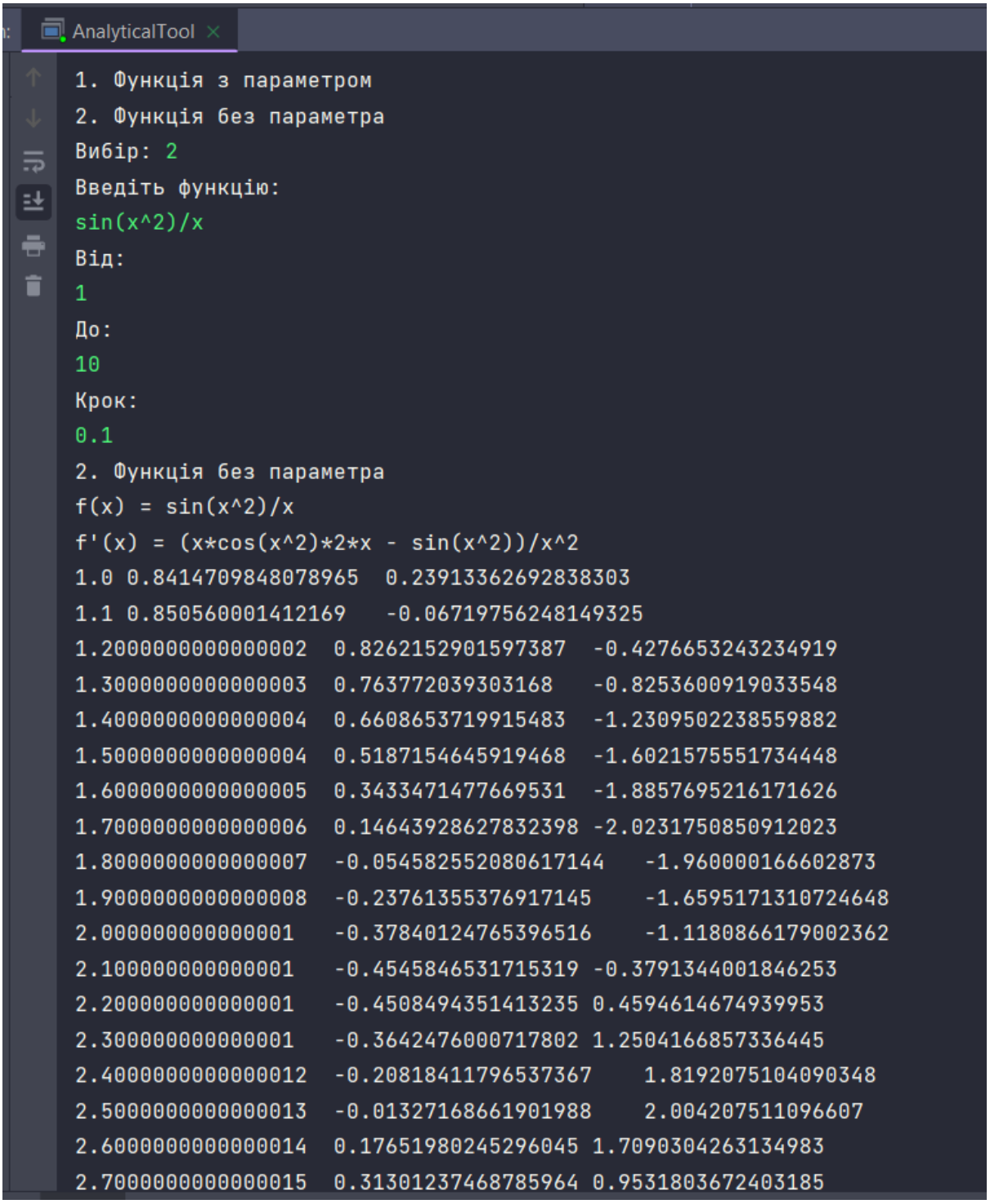
TblFuncTreeMap.dat

	x	y
2	1.0	0.8414709848078965
3	1.1	0.8912073600614354
4	1.2000000000000002	0.9320390859672264
5	1.3000000000000003	0.9635581854171931
6	1.4000000000000004	0.9854497299884603
7	1.5000000000000004	0.9974949866040544
8	1.6000000000000005	0.9995736030415051
9	1.7000000000000006	0.9916648104524686
10	1.8000000000000007	0.973847630878195
11	1.9000000000000008	0.9463000876874142
12	2.000000000000001	0.9092974268256814
13	2.100000000000001	0.8632093666488733
14	2.200000000000001	0.8084964038195895
15	2.300000000000001	0.7457052121767194
16	2.4000000000000012	0.67546318055115
17	2.5000000000000013	0.5984721441039554
18	2.6000000000000014	0.515501371821463
19	2.7000000000000015	0.42737988023382856
20	2.8000000000000016	0.3349881501559034
21	2.9000000000000017	0.23924932921398068
22	3.0000000000000018	0.14112000805986546
23	3.100000000000002	0.041580662433288715
24	3.200000000000002	-0.05837414342758186
25	3.300000000000002	-0.1577456941432504
26	3.400000000000002	-0.2555411020268334
27	3.500000000000002	-0.35078322768962195
28	3.6000000000000023	-0.44252044329485446
29	3.7000000000000024	-0.5298361409084953
30	3.8000000000000025	-0.611857890942721
31	3.9000000000000026	-0.6877661591839757

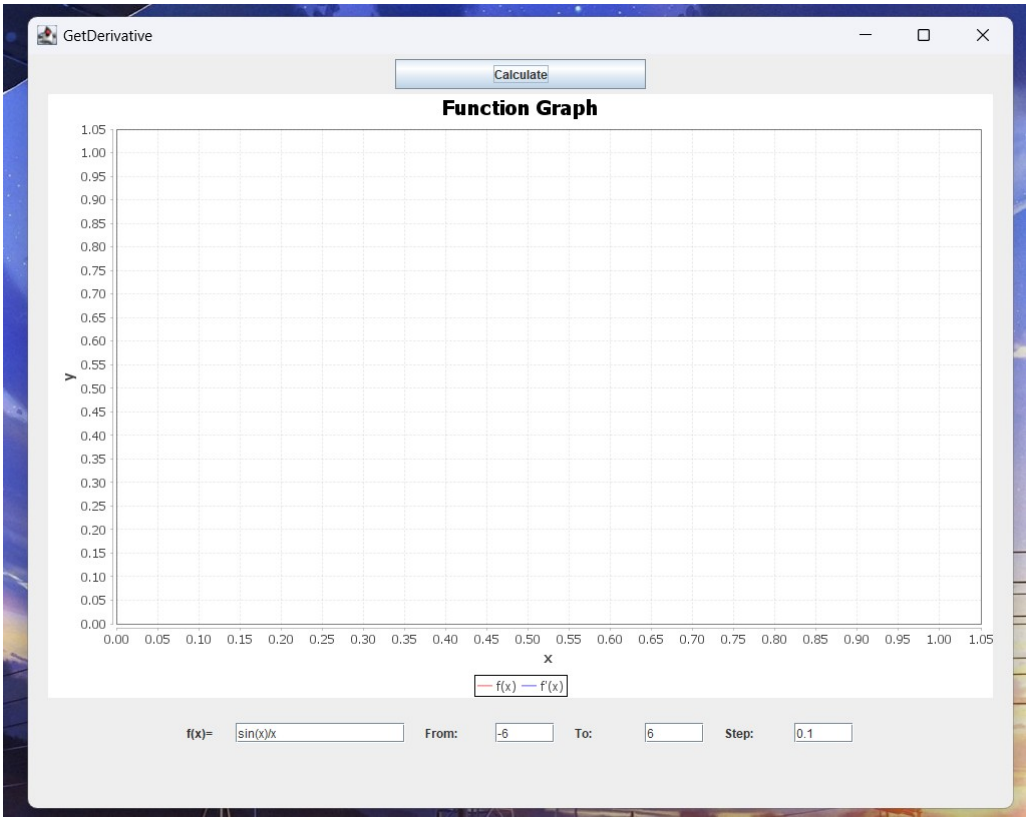
Лістинг Б.5 – Результат роботи класу AnalitEx з парметром (приклад 1)



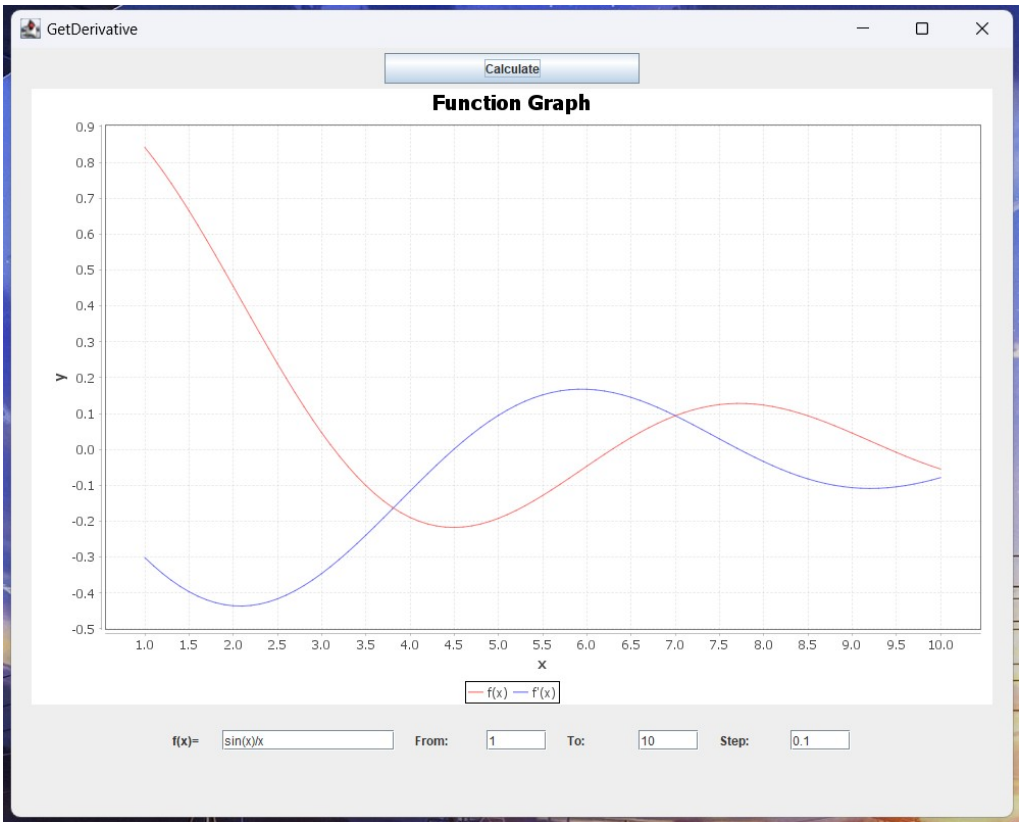
Лістинг Б.6 – Результат роботи класу AnalitEx без параметра(приклад 1)



Лістинг Б.7 – Інтерфейс користувача



Лістинг Б.8 – Результат роботи класу JFreeChartMainFrame(приклад 1)



Посилання на GidHub: https://github.com/zellii1/KPP.git