Міністерство освіти й науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

ЗВІТ

Про виконання лабораторної роботи №3

З дисципліни «Комп’ютерне моделювання та оптимізація»

**Комп’ютерне інтегрування та диференціювання**

TI-92 Черноусова Дениса

Перевірив проф. д.т.н. Шушура О. М.

Київ – 2021

g = 3

k = 2

g – остання цифра у номері студентського квитка, а k – передостання.

**Завдання**

Розробити алгоритми та програмне забезпечення для розв’язку наведеної задачі вказаними методами. Алгоритми представити у вигляді блок-схем або діаграм діяльності UML. Програмне забезпечення розробити на будь-якій сучасній мові програмування. Знайти точне значення розв’язку задачі за допомогою математичних бібліотек та порівняти його зі значеннями, отриманими в результаті роботи розробленого програмного забезпечення.

1. Для функції *f(x)=х3+x* знайти в точці х = g-2k-2 (де g – остання цифра у номері студентського квитка, а k – передостання) значення першої похідної функції з точністю 0,001 за допомогою послідовності приростів, центрованої різниці (порядку точності 2). Порівняти із точним значенням похідної функції, оцінити кількість ітерацій та точність по кожному методу.
2. Для функції *f(x)=х3+x* скласти таблицю значень, заповнивши таблицю 1. На основі складеної таблиці знайти в точці х = g-2k-2 значення першої похідної функції методом квадратичної інтерполяції. Порівняти із точним значенням похідної функції, оцінити точність.

Таблиця 1

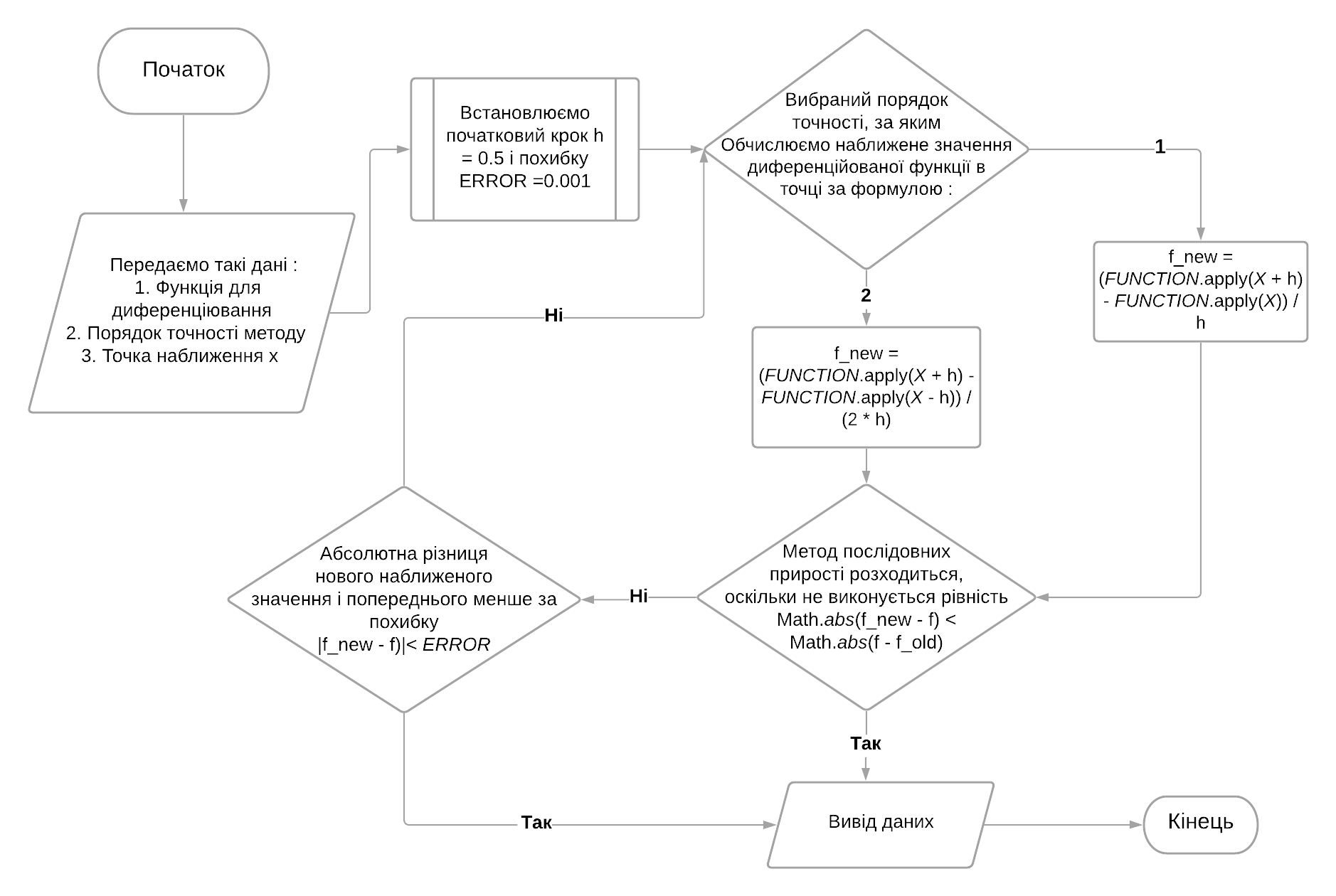
|  |  |  |  |
| --- | --- | --- | --- |
| ***x*** | ***g-2k-3*** | ***g-2k-1,5*** | ***g-2k*** |
| ***f(x)*** |  |  |  |

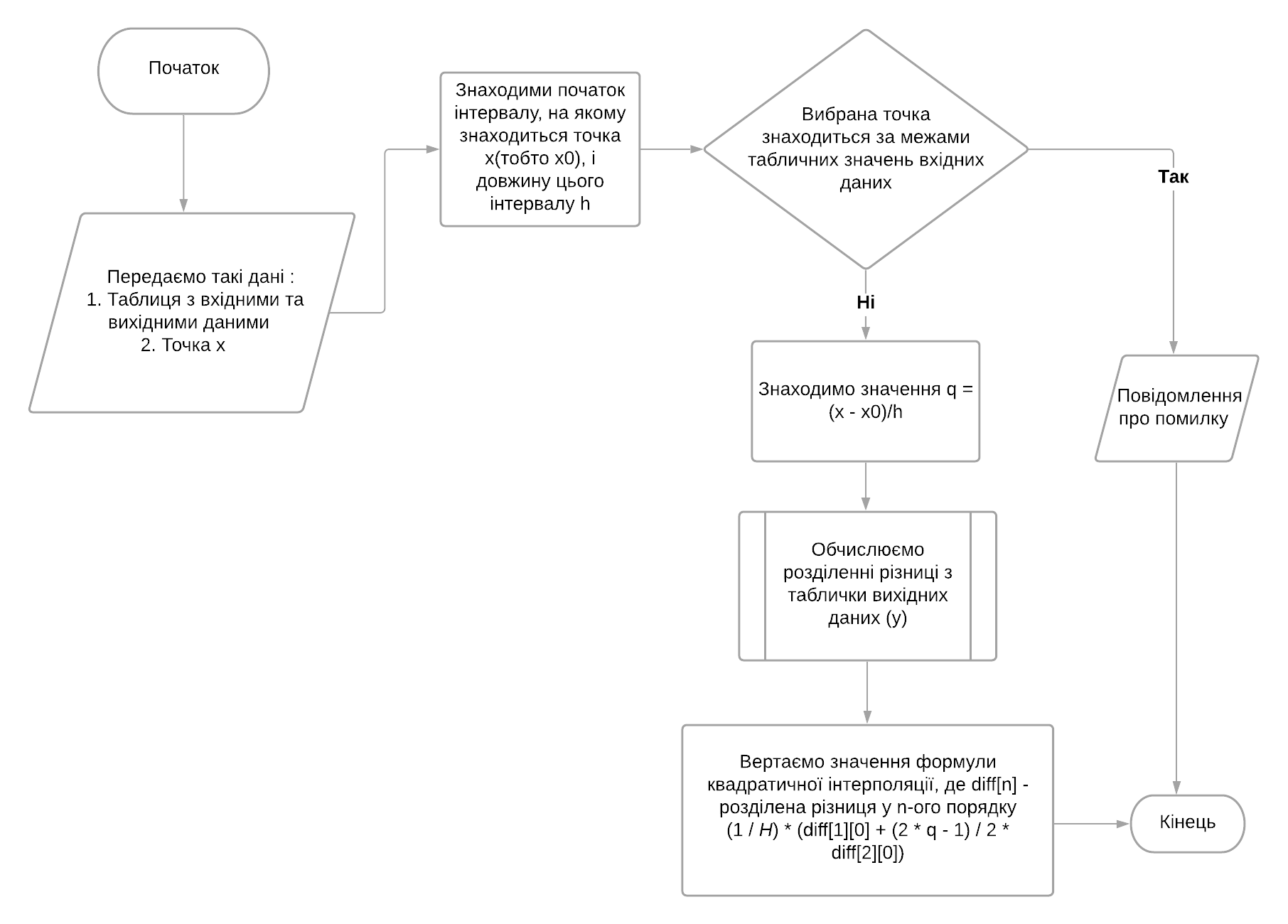
1. Знайти інтеграл:

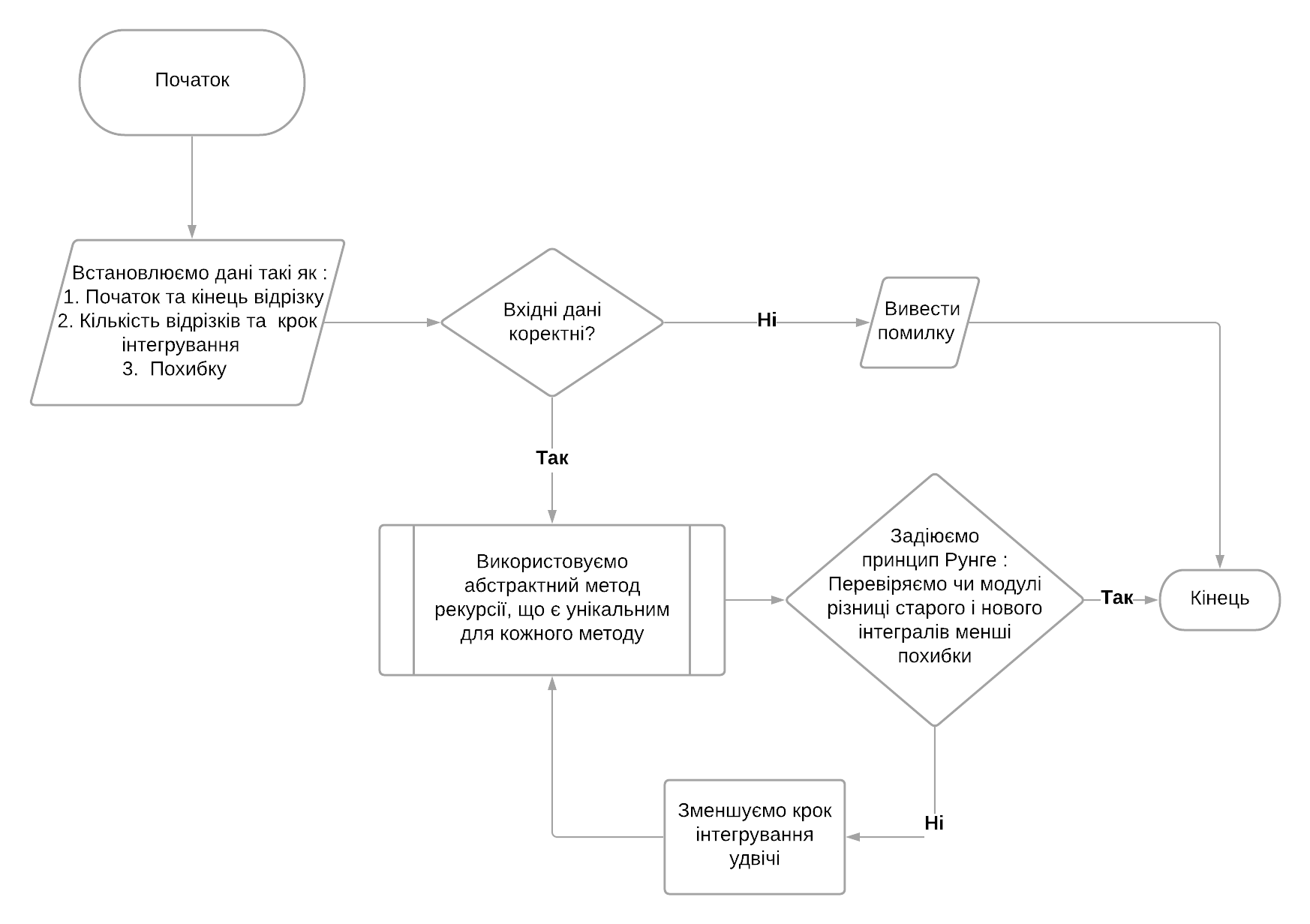


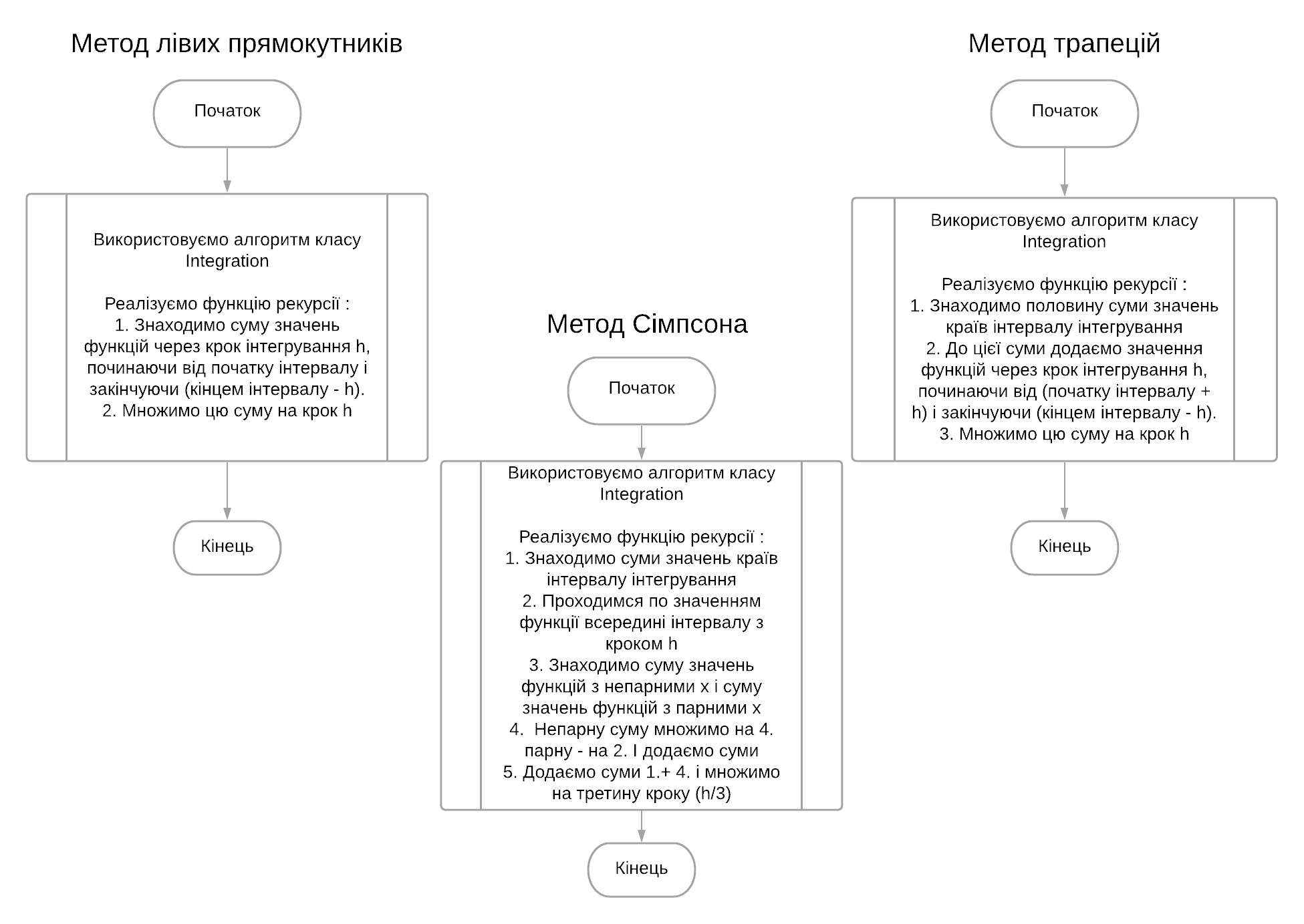
за допомогою методів лівих прямокутників, трапецій і Сімпсона з точністю 0,001. Для оцінювання точності використовувати принцип Рунге. Обчислити точне значення інтеграла і порівняти з отриманими наближеними значеннями, оцінити точність.

**Алгоритми**

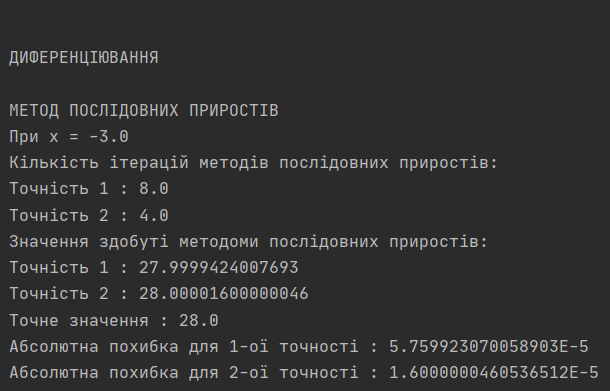
Метод послідовних приростів

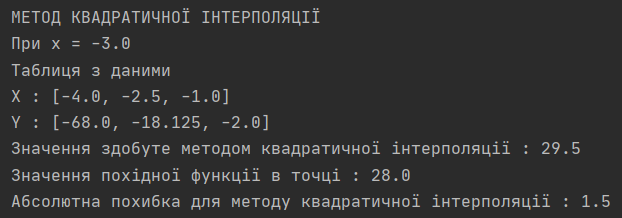
Метод квадратичної інтерполяції

Допоміжний клас Integration

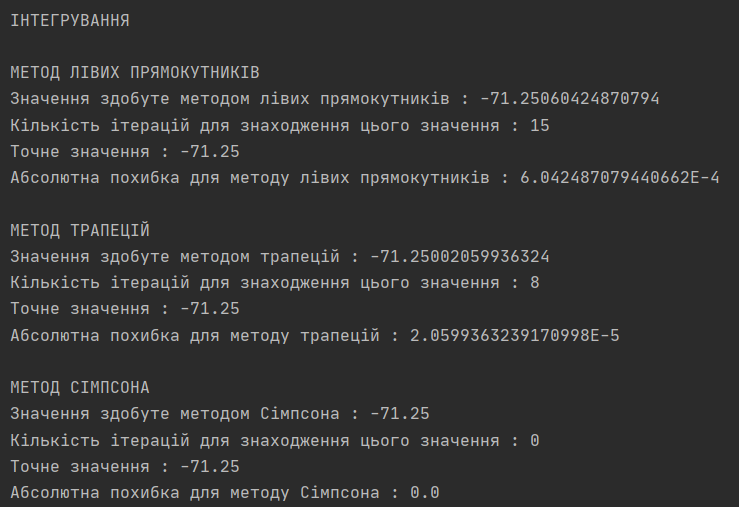
Методи інтегрування

**Результати роботи**

Завдання 1

Завдання 2

Завдання 3



**Лістинг**

Main.java

package Lab3;  
  
import Lab3.Differentiation.IncrementsSequence;  
import Lab3.Differentiation.QuadraticInterpolation;  
import Lab3.NumericalIntegration.LeftRectangles;  
import Lab3.NumericalIntegration.Simpson;  
import Lab3.NumericalIntegration.Trapezium;  
  
import java.util.Arrays;  
  
public class Main {  
  
 static int *G* = 3; //3  
 static int *K* = 2; //2  
  
 public static void Task1() throws Exception {  
  
 System.*out*.println("\n\nДИФЕРЕНЦІЮВАННЯ");  
  
 System.*out*.println("\nМЕТОД ПОСЛІДОВНИХ ПРИРОСТІВ");  
  
 double x = *G* - 2 \* *K* - 2;  
  
 System.*out*.println("При x = " + x);  
  
 double acc1 = IncrementsSequence.*incrementsSequence*(MyFunctions::*function1*, x, 1);  
 double num1 = IncrementsSequence.*ITERATIONS*;  
  
 double acc2 = IncrementsSequence.*incrementsSequence*(MyFunctions::*function1*, x, 2);  
 double num2 = IncrementsSequence.*ITERATIONS*;  
  
 double real = MyFunctions.*function1der1*(x);  
  
 System.*out*.println("Кількість ітерацій методів послідовних приростів:" +  
 "\nТочність 1 : " + num1 +  
 "\nТочність 2 : " + num2);  
  
 System.*out*.println("Значення здобуті методоми послідовних приростів:" +  
 "\nТочність 1 : " + acc1 +  
 "\nТочність 2 : " + acc2 +  
 "\nТочне значення : " + real);  
  
 System.*out*.println("Абсолютна похибка для 1-ої точності : " + Math.*abs*(acc1 - real) +  
 "\nАбсолютна похибка для 2-ої точності : " + Math.*abs*(acc2 - real));  
 }  
  
 public static void Task2() throws Exception {  
  
 System.*out*.println("\nМЕТОД КВАДРАТИЧНОЇ ІНТЕРПОЛЯЦІЇ");  
  
 double x = *G* - 2 \* *K* - 2;  
  
 System.*out*.println("При x = " + x);  
  
 double[] arrX = new double[]{*G* - 2 \* *K* - 3, *G* - 2 \* *K* - 1.5, *G* - 2 \* *K*};  
  
 double[] arrY = new double[arrX.length];  
  
 for (int i = 0; i < arrX.length; i++) {  
 arrY[i] = MyFunctions.*function1*(arrX[i]);  
 }  
  
 System.*out*.println("Таблиця з даними");  
 System.*out*.println("X : " + Arrays.*toString*(arrX));  
 System.*out*.println("Y : " + Arrays.*toString*(arrY));  
  
 double quadric = QuadraticInterpolation.*quadraticInterpolation*(arrX, arrY, x);  
  
 double real = MyFunctions.*function1der1*(x);  
  
 System.*out*.println("Значення здобуте методом квадратичної інтерполяції : " + quadric);  
  
 System.*out*.println("Значення похідної функції в точці : " + real);  
  
 System.*out*.println("Абсолютна похибка для методу квадратичної інтерполяції : " + Math.*abs*(quadric - real));  
 }  
  
 public static void Task3() throws Exception {  
  
  
 System.*out*.println("\n\nІНТЕГРУВАННЯ");  
  
 final double real = -71.25; // Значення взято з калькулятора інтегралів  
   
 ///////////////  
 System.*out*.println("\nМЕТОД ЛІВИХ ПРЯМОКУТНИКІВ");  
  
 double left = (new LeftRectangles()).method(*G* - 2 \* *K* - 3, *G* - 2 \* *K*, MyFunctions::*function1*);  
  
 System.*out*.println("Значення здобуте методом лівих прямокутників : " + left);  
  
 System.*out*.println("Кількість ітерацій для знаходження цього значення : " + LeftRectangles.*ITERATIONS*);  
  
 System.*out*.println("Точне значення : " + real);  
  
 System.*out*.println("Абсолютна похибка для методу лівих прямокутників : " + Math.*abs*(left - real));  
  
 ////////////////  
 System.*out*.println("\nМЕТОД ТРАПЕЦІЙ");  
  
 double trap = (new Trapezium()).method(*G* - 2 \* *K* - 3, *G* - 2 \* *K*, MyFunctions::*function1*);  
  
 System.*out*.println("Значення здобуте методом трапецій : " + trap);  
  
 System.*out*.println("Кількість ітерацій для знаходження цього значення : " + Trapezium.*ITERATIONS*);  
  
 System.*out*.println("Точне значення : " + real);  
  
 System.*out*.println("Абсолютна похибка для методу трапецій : " + Math.*abs*(trap - real));  
  
 ////////////////  
 System.*out*.println("\nМЕТОД СІМПСОНА");  
  
 double simp = (new Simpson()).method(*G* - 2 \* *K* - 3, *G* - 2 \* *K*, MyFunctions::*function1*);  
  
 System.*out*.println("Значення здобуте методом Сімпосна : " + simp);  
  
 System.*out*.println("Кількість ітерацій для знаходження цього значення : " + Simpson.*ITERATIONS*);  
  
 System.*out*.println("Точне значення : " + real);  
  
 System.*out*.println("Абсолютна похибка для методу Сімпосна : " + Math.*abs*(simp - real));  
  
 }  
  
 public static void main(String[] args) throws Exception {  
  
 *Task1*();  
 *Task2*();  
 *Task3*();  
  
 }  
}

MyFunctions.java

package Lab3;  
  
public class MyFunctions {  
  
 public static double function1(double x) {  
 return x + Math.*pow*(x, 3);  
 }  
  
 public static double function1der1(double x) {  
 return 1 + Math.*pow*(x, 2) \* 3;  
 }  
}

IncrementsSequence.java

package Lab3.Differentiation;  
  
import java.util.function.UnaryOperator;  
  
public class IncrementsSequence {  
  
 static double *ERROR* = 0.001;  
 static double *X*;  
 static UnaryOperator<Double> *FUNCTION*;  
 static double *H* = 0.5;  
 public static int *ITERATIONS* = 0;  
 static int *ACCURACY* = 0;  
  
 public static double recursion(double h, double f, double f\_old) throws Exception {  
  
 *ITERATIONS*++;  
  
 double f\_new;  
  
 if (*ACCURACY* == 2) f\_new = (*FUNCTION*.apply(*X* + h) - *FUNCTION*.apply(*X* - h)) / (2 \* h);  
 else if (*ACCURACY* == 1) f\_new = (*FUNCTION*.apply(*X* + h) - *FUNCTION*.apply(*X*)) / h;  
 else throw new Exception("No such accuracy!");  
  
 if (!*validation*(f\_new, f, f\_old))  
 throw new Exception("Метод послідовних приростів розходиться при значені x = " + *X*);  
  
 if (Math.*abs*(f\_new - f) < *ERROR*) return f\_new;  
  
 return *recursion*(h / 5, f\_new, f);  
  
  
 }  
  
 public static double incrementsSequence(UnaryOperator<Double> func, double x, int accuracy) throws Exception {  
 *ITERATIONS* = 0;  
 *FUNCTION* = func;  
 *ACCURACY* = accuracy;  
 *X* = x;  
 return *recursion*(*H*, 0, 0);  
 }  
  
 public static boolean validation(double f\_new, double f, double f\_old) {  
 if (f == f\_old && f\_old == 0) return true;  
 return Math.*abs*(f\_new - f) < Math.*abs*(f - f\_old);  
 }  
  
}

QuadraticInterpolation.java

package Lab3.Differentiation;  
  
import java.util.function.UnaryOperator;  
  
public class QuadraticInterpolation {  
  
 static double *H* = 0;  
  
  
 public static double getIntervalStart(double[] arrX, double x) throws Exception {  
  
 for (int i = 0; i < arrX.length - 1; i++) {  
 if (arrX[i] <= x && arrX[i + 1] > x) {  
 *H* = Math.*abs*(arrX[i + 1] - arrX[i]);  
 return arrX[i];  
 }  
 }  
 if (arrX[arrX.length - 1] == x) {  
 *H* = Math.*abs*(arrX[arrX.length - 1] - arrX[arrX.length - 2]);  
 return x;  
 }  
 throw new Exception("x = " + x + " is not in borders!");  
 }  
  
  
 public static double quadraticInterpolation(double[] arrX, double[] arrY, double x) throws Exception {  
  
 double x0 = *getIntervalStart*(arrX, x);  
  
 double q = (x - x0) / *H*;  
  
 // Розділені різниці  
 double[][] diff = new double[arrX.length][arrX.length];  
 System.*arraycopy*(arrY, 0, diff[0], 0, arrX.length);  
  
 for (int i = 1; i < arrX.length; i++)  
 for (int j = 0; j < arrX.length - i; j++)  
 diff[i][j] = diff[i - 1][j + 1] - diff[i - 1][j];  
  
  
 return (1 / *H*) \* (diff[1][0] + (2 \* q - 1) / 2 \* diff[2][0]);  
 }  
  
}

Integration.java

package Lab3.NumericalIntegration;  
  
import java.util.function.UnaryOperator;  
  
public abstract class Integration {  
  
 static UnaryOperator<Double> *FUNCTION*;  
 static final double *ERROR* = 0.001;  
 static final int *SEGMENTS* = 100;  
 public static int *ITERATIONS*;  
 static double *START*;  
 static double *END*;  
  
 public double method(double start, double end, UnaryOperator<Double> func) throws Exception {  
 if (*SEGMENTS* <= 0) throw new Exception("Number of segments has to be more 0");  
 if (start >= end) throw new Exception("Start has be more than End");  
 *ITERATIONS* = 0;  
  
 *FUNCTION* = func;  
 *START* = start;  
 *END* = end;  
  
  
 double h = (*END* - *START*) / *SEGMENTS*;  
  
 return recursion(h, 0);  
 }  
  
 // Використовуємо принцип Рунге  
 public double Runge(double integral, double prev, double h) {  
 *ITERATIONS*++;  
 if (Math.*abs*(integral - prev) < *ERROR*) return integral;  
 return recursion(h / 2, integral);  
 }  
  
 public abstract double recursion(double h, double prev);  
  
}

LeftRectangles.java

package Lab3.NumericalIntegration;  
  
import java.util.function.UnaryOperator;  
  
public class LeftRectangles extends Integration {  
  
 public double recursion(double h, double prev) {  
  
 double sum = 0;  
 for (double x = *START*; x < *END*; x += h) {  
 sum += *FUNCTION*.apply(x);  
 }  
 double integral = sum \* h;  
  
 // Використовуємо принцип Рунге  
 return Runge(integral,prev ,h);  
 }  
  
}

Simpson.java

package Lab3.NumericalIntegration;  
  
public class Simpson extends Integration {  
  
 @Override  
 public double recursion(double h, double prev) {  
  
 double sum = *FUNCTION*.apply(*START*) + *FUNCTION*.apply(*END*); // Чому так?  
  
 double even = 0; // парні  
 double odd = 0; // непарні  
  
 double x = *START* + h;  
 int i = 1;  
  
 while (x < *END*) {  
 System.*out*.println("1. i = " + i);  
 odd += *FUNCTION*.apply(x);  
 x += (h \* 2);  
 i += 2;  
 }  
  
 x = *START* + (h \* 2);  
 i = 2;  
  
 while (x < *END*) {  
 System.*out*.println("2. i = " + i);  
 even += *FUNCTION*.apply(x);  
 x += (h \* 2);  
 i += 2;  
 }  
  
 odd \*= 4; // добуток значень з непарними індексами  
 sum += odd;  
  
 even \*= 2; // добуток значень з парними індексами  
 sum += even;  
  
 double integral = sum \* h / 3;  
  
 return integral;  
 //return Runge(integral, prev, h);  
 }  
}

Trapezium.java

package Lab3.NumericalIntegration;  
  
  
  
public class Trapezium extends Integration{  
  
  
 public double recursion(double h, double prev) {  
  
 double sum = (*FUNCTION*.apply(*START*) + *FUNCTION*.apply(*END*)) / 2;  
  
 for (double x = *START* + h; x < *END*; x += h) {  
 sum += *FUNCTION*.apply(x);  
 }  
 double integral = sum \* h;  
  
 // Використовуємо принцип Рунге  
 return Runge(integral,prev ,h);  
 }  
}

**Висновки**

Під час цієї лабораторної роботи були вивчені методи, що дозволяють знайти значення похідної функції в точці такі як метод послідовних прирості та метод квадратичної інтерполяції. Для інтегрування функцій був створений абстрактний батьківський клас для методів: лівих прямокутників, трапецій та Сімпсона. Через абстрактний клас викликався реалізований принцип Рунге. Найефективніше себе проявив метод трапецій - найменша кількість операцій з прийнятним результатом.