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 №2

З дисципліни «Комп’ютерне моделювання та оптимізація»

**Комп’ютерне розв’язання рівнянь та їх систем**

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. Відділеня коренів здійснити побудовою графіків функції з використанням математичних пакетів. Абсолютна похибка має дорівнювати 0,01. Знайти точне значення розв’язку задачі за допомогою математичних бібліотек та порівняти його зі значенням, отриманим в результаті роботи розробленого програмного забезпечення.

1. Знайти розв’язок рівняння =0 (де g – остання цифра у номері студентського квитка, а k – передостання) методами ділення навпіл, хорд та дотичних (Ньютона).
2. Знайти розв’язок рівняння =0 (де g – остання цифра у номері студентського квитка, а k – передостання) методом ітерацій.
3. Знайти розв’язок СЛАР методом: Гауса - для парних g, Крамера - для непарних g = 3:



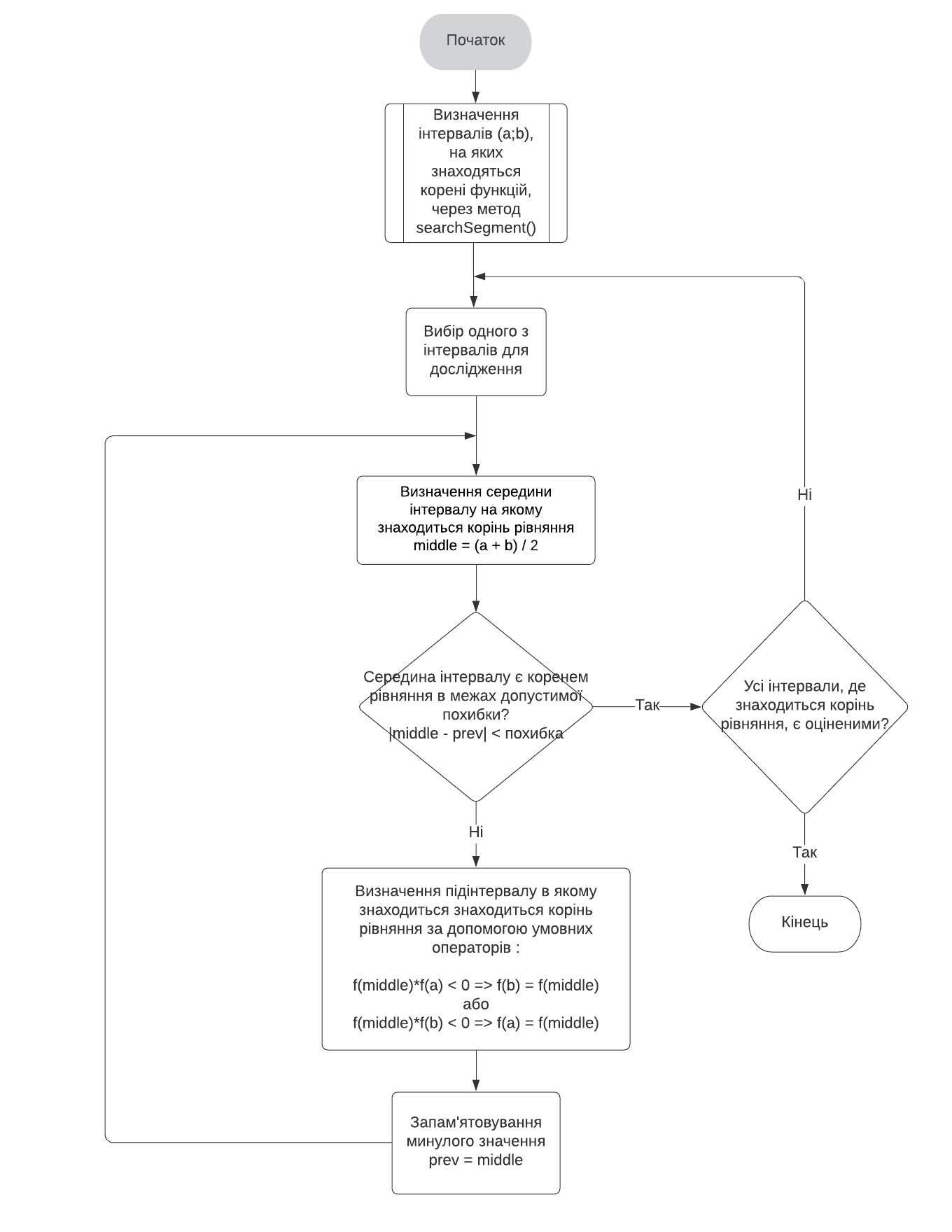
де g – остання цифра у номері студентського квитка, а k – передостання.

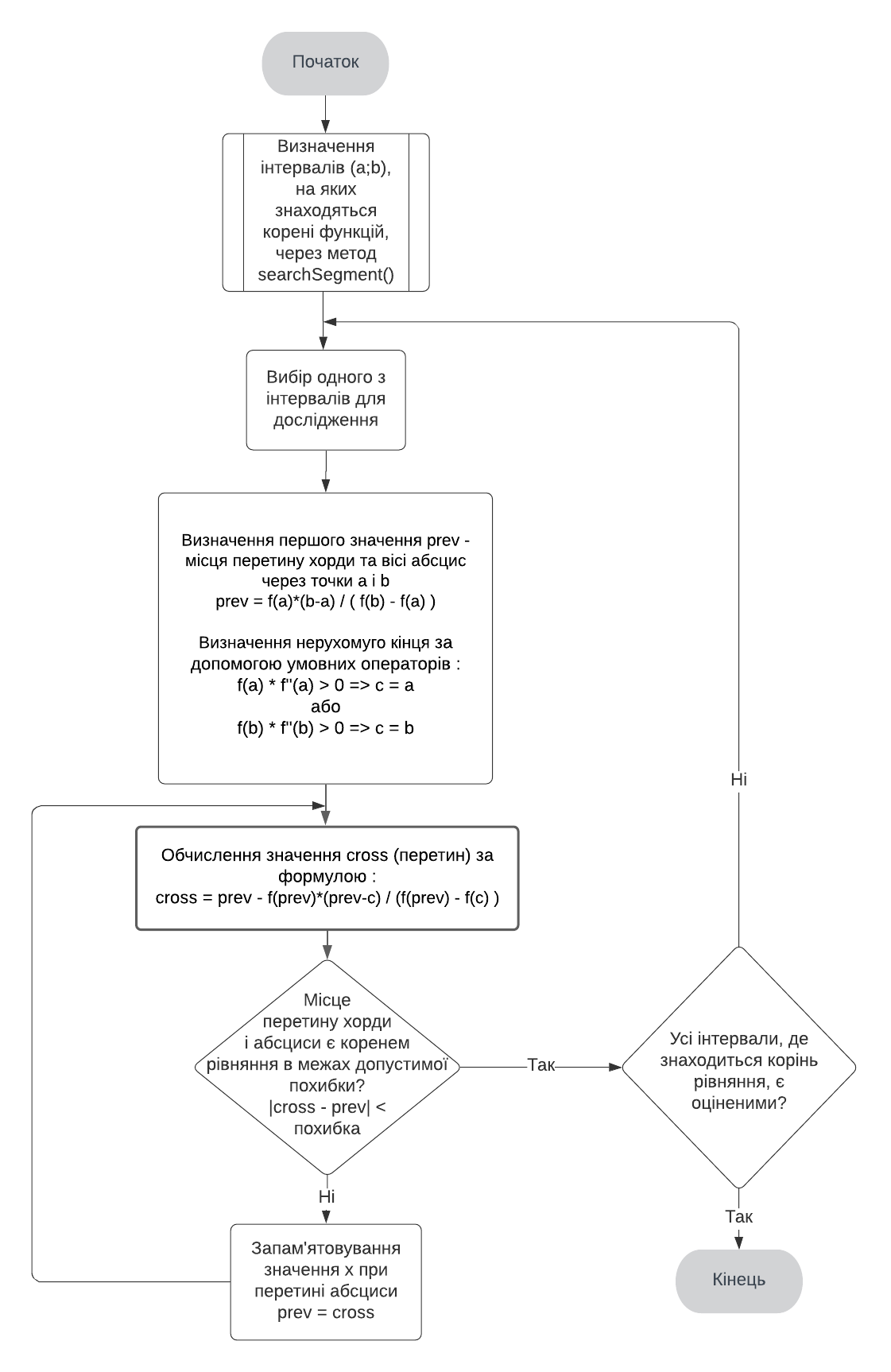
1. Знайти з точністю 0,01 розв’язок системи нелінійних рівнянь методами Ньютона та Зейделя:

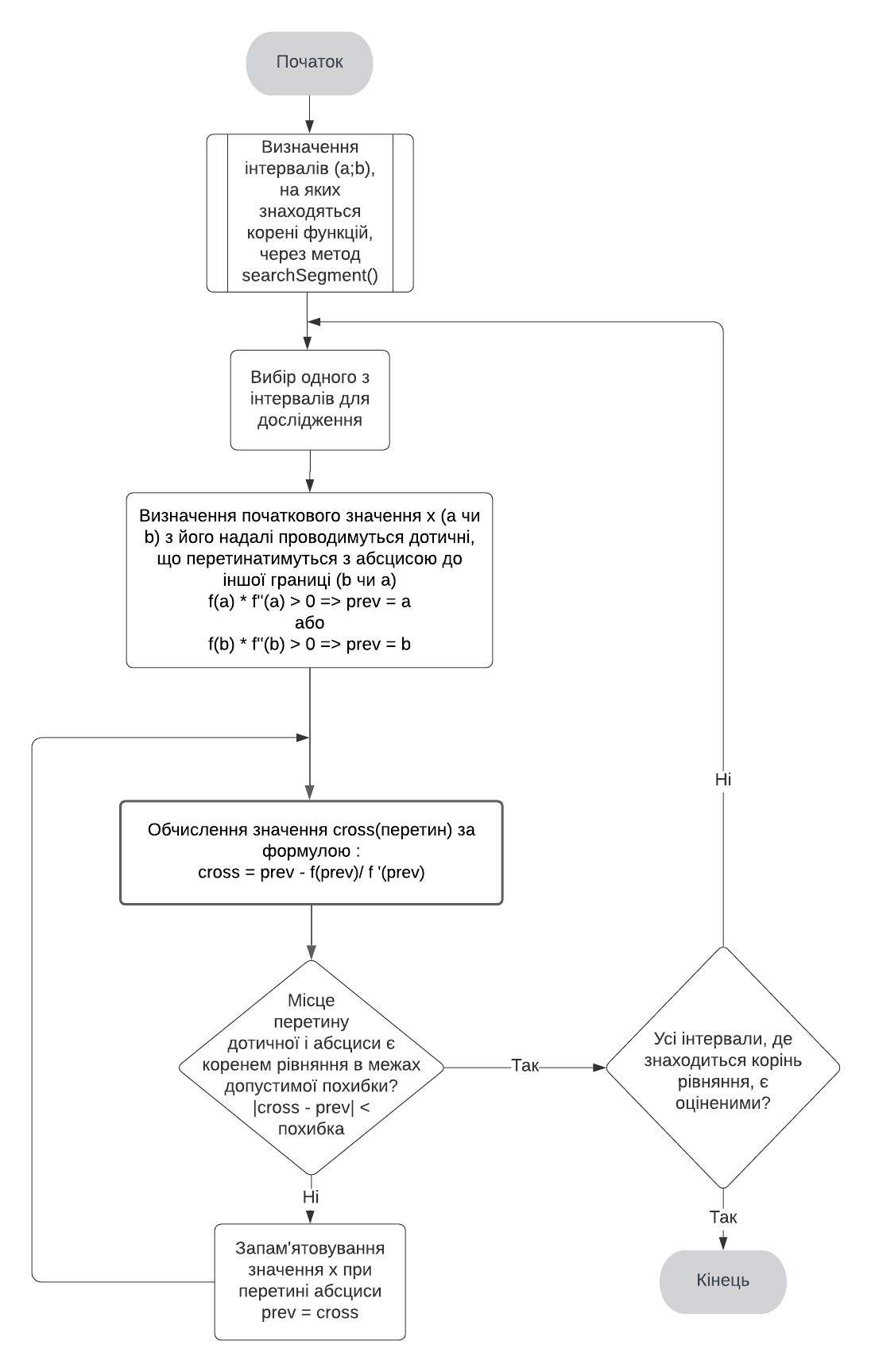


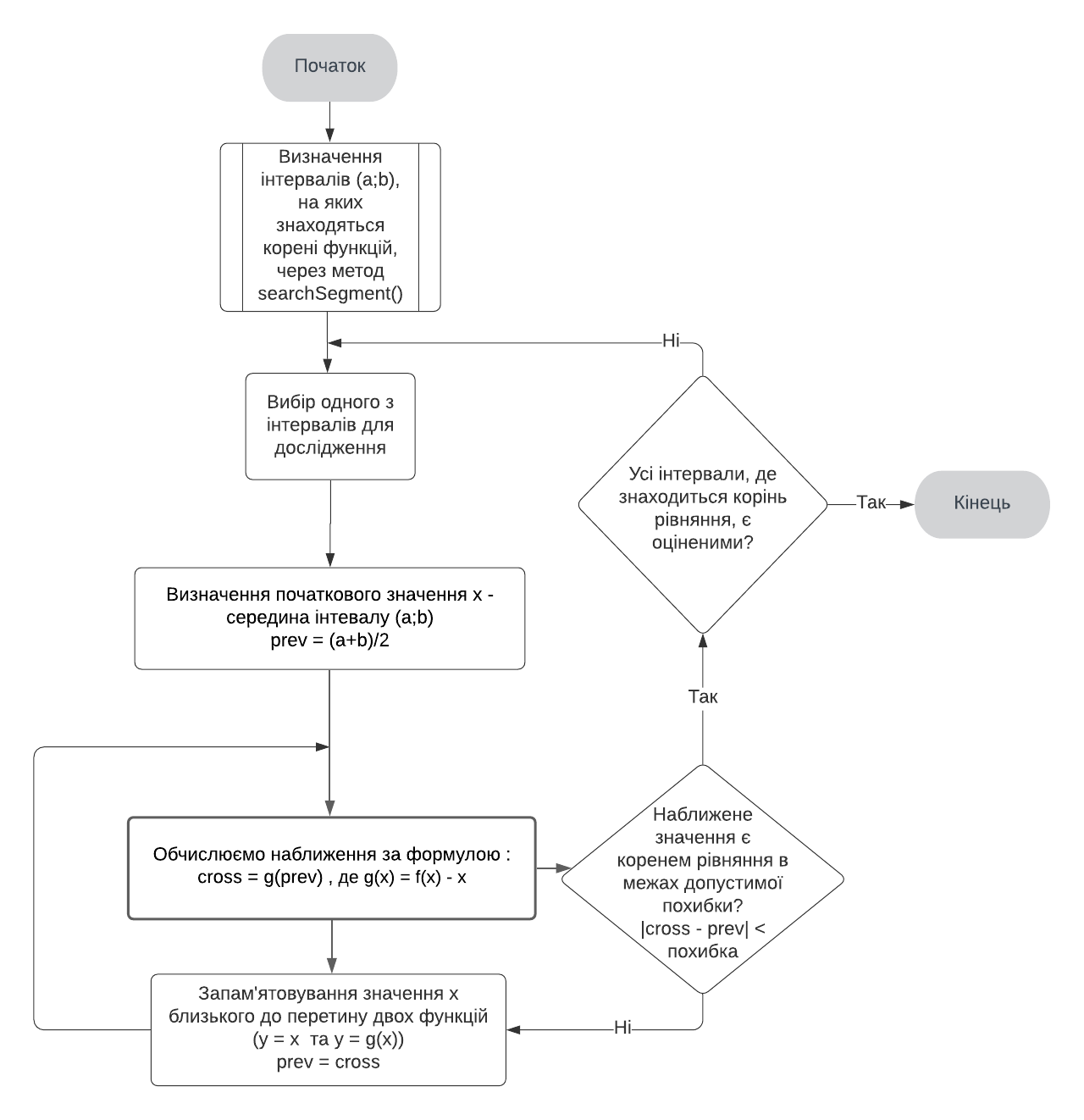
де g – остання цифра у номері студентського квитка, а k – передостання. Для пошуку початкового наближення використати побудову графіків рівнянь.

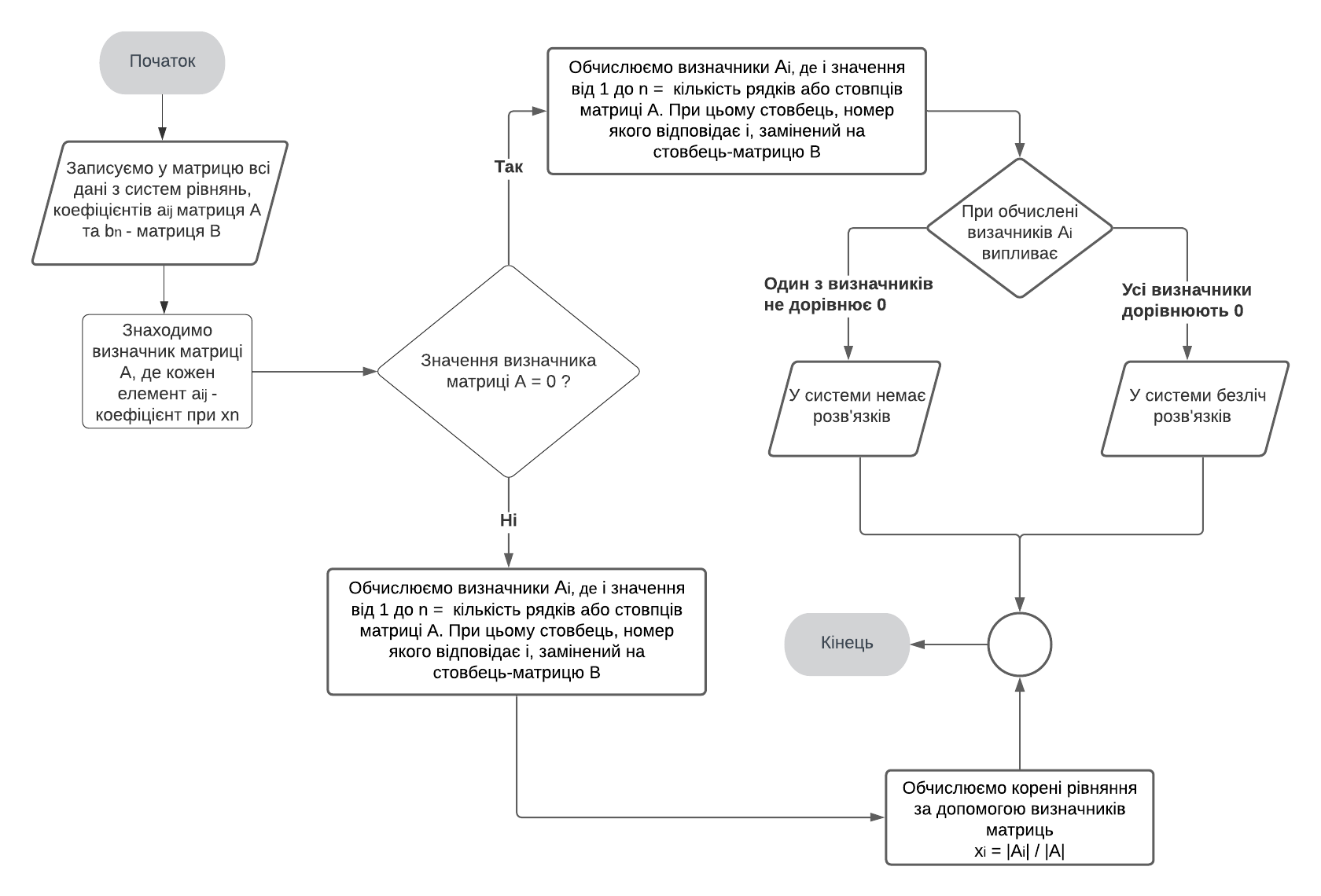
**Алгоритми**

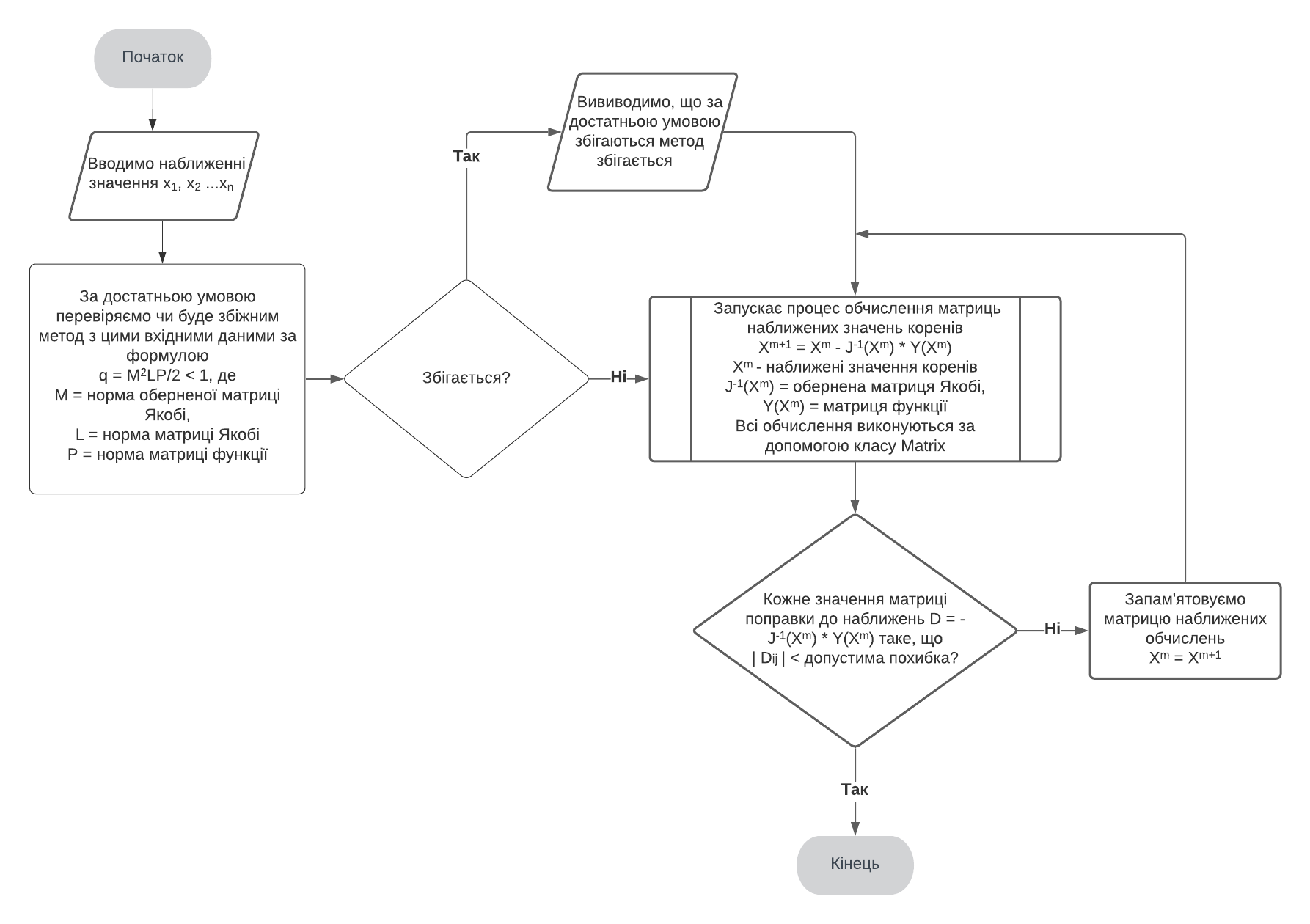
**Метод половинного ділення**

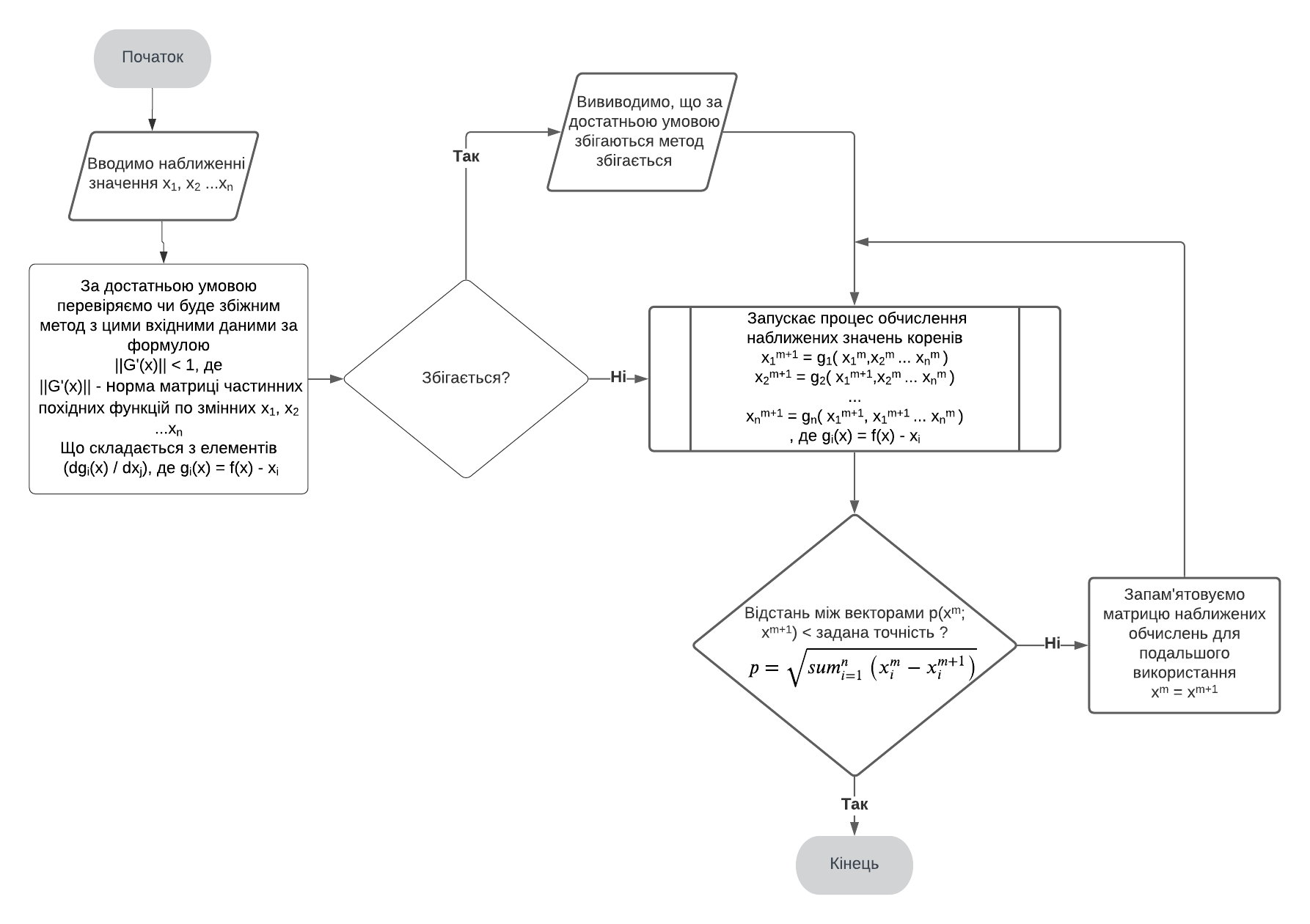
**Метод хорд**

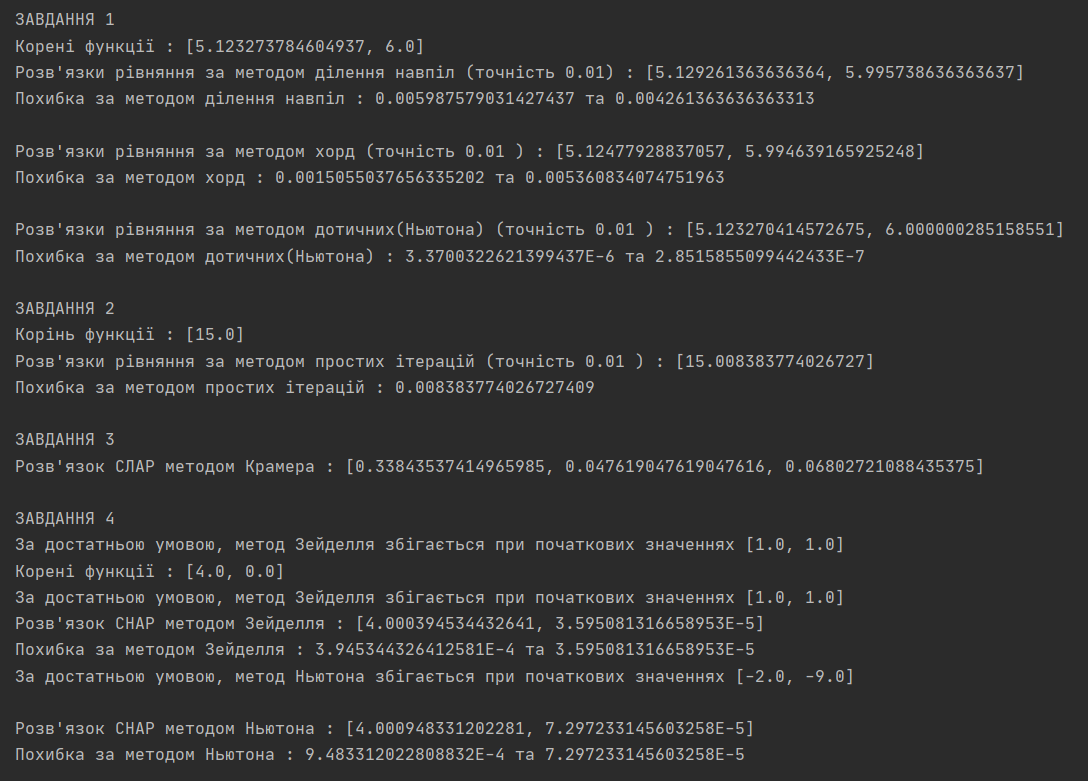
**Метод дотичних (Ньютона)**

**Метод простих ітерацій**

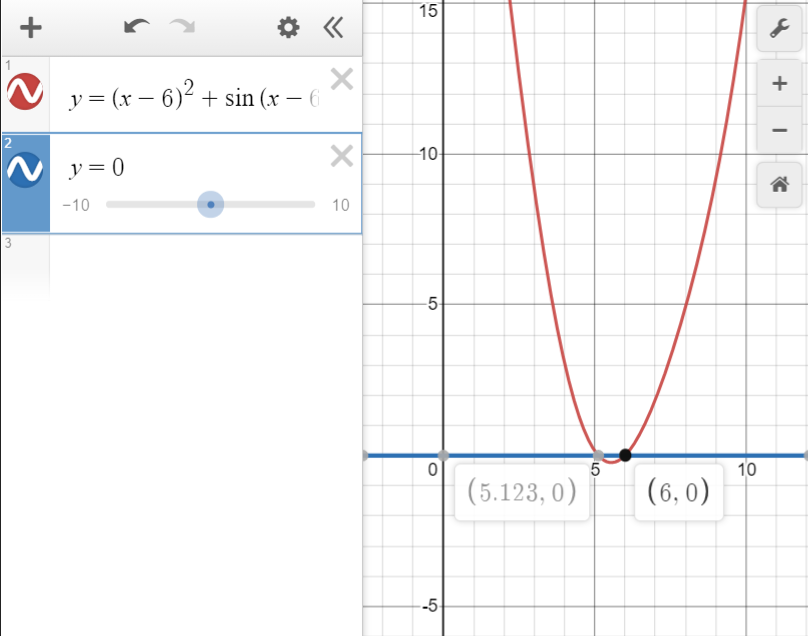
**Метод Крамера**

**Метод Ньютона**

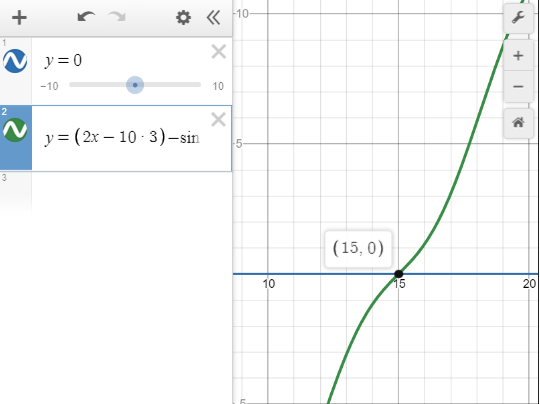
**Метод Зейделя**

**Результати роботи**

Графік для **Завдання 1**

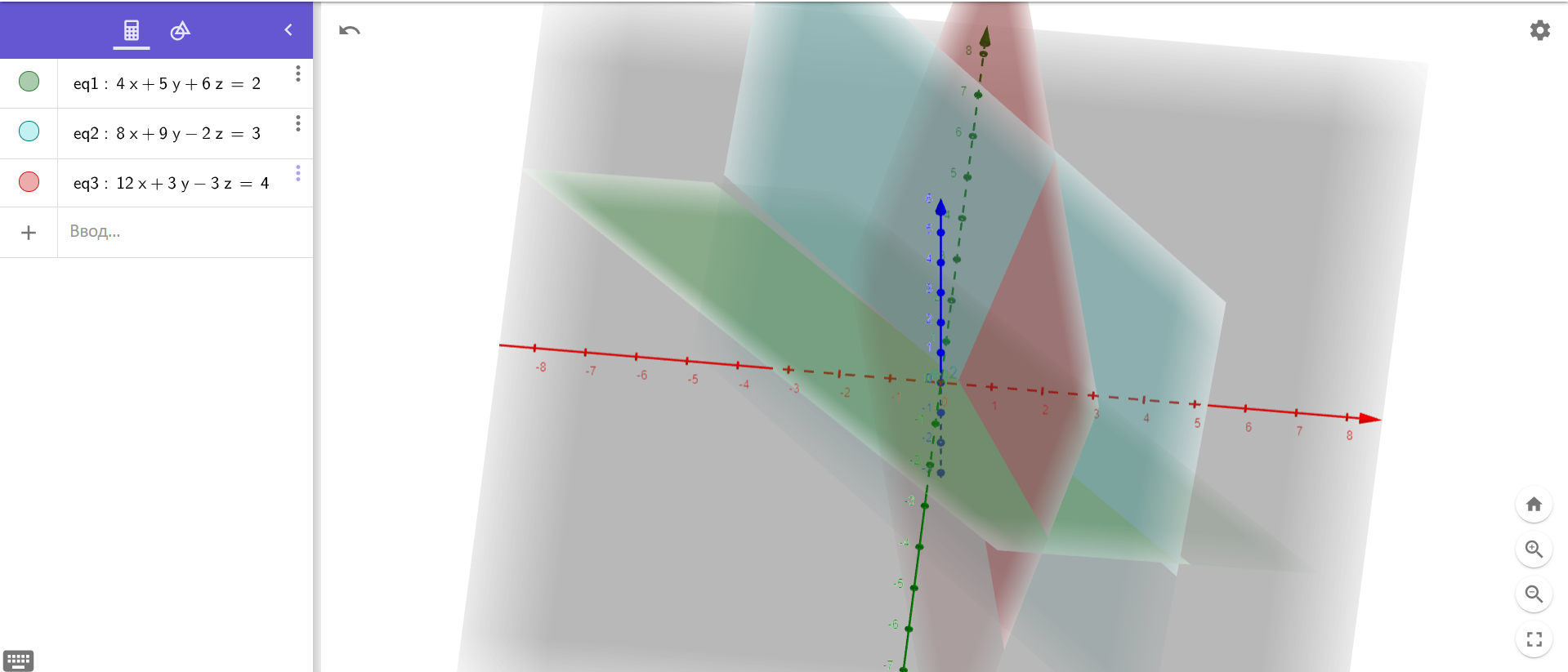
y = (x-6)^2 + sin(x-6) = 0

Корені : х = 5.123 та х = 6

Графік для **Завдання 2**

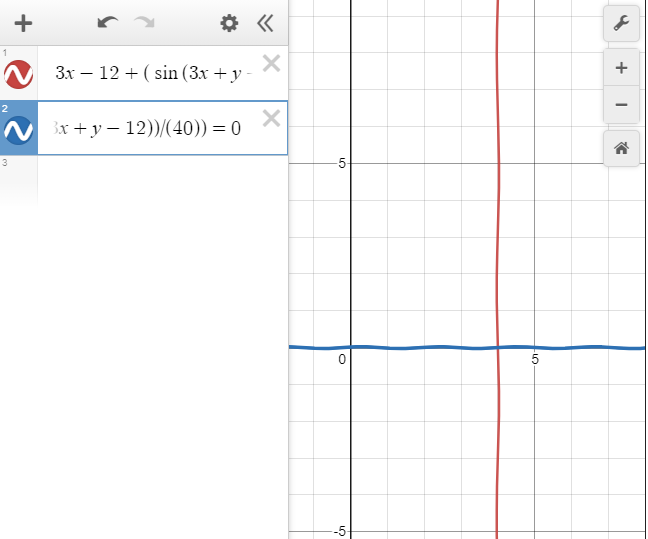
y = (2x-10\*3) – sin (x – 10 \* 3/2)

Корінь x = 15

Графік для **Завдання 3**

Корені рівняння х = 0.33, у = 0.047, z= 0.068

Графік для **Завдання 4**

Перетин функцій знаходиться у точці (4; 0). Тобто x = 4, y = 0 – корені рівняння

**Лістинг**

Main.java

package lab2;  
  
  
import lab2.MethodsNonlinear.\*;  
import lab2.MethodsNonolinearSystem.NewtonMethod;  
import lab2.MethodsNonolinearSystem.SeidelMethod;  
import lab2.MethodsSLAR.KramerMethod;  
  
import java.util.Arrays;  
import java.util.function.Function;  
  
import static java.lang.Math.*abs*;  
import static java.lang.Math.*pow*;  
  
  
public class Main {  
  
  
 public static void main(String[] args) {  
  
 System.*out*.println("\nЗАВДАННЯ 1");  
  
  
 double[] natural = DivisionMethod.*divisionMethod*(0, 10, 0.5, *pow*(0.01, 10), 3);  
 double[] calculate;  
  
 System.*out*.println("Корені функції : " + Arrays.*toString*(natural));  
  
 ///////////////Ділення навпіл  
 System.*out*.println("Розв'язки рівняння за методом ділення навпіл (точність 0.01) : "  
 + Arrays.*toString*(calculate = DivisionMethod.*divisionMethod*(0, 10, 1, 0.01, 3)));  
  
 System.*out*.println("Похибка за методом ділення навпіл : " + *abs*(calculate[0] - natural[0])  
 + " та " + *abs*(calculate[1] - natural[1]));  
  
 ////////////////Хорд  
 System.*out*.println("\nРозв'язки рівняння за методом хорд (точність 0.01 ) : "  
 + Arrays.*toString*(calculate = ChordsMethod.*chordsMethod*(0, 10, 1, 0.01, 3)));  
  
 System.*out*.println("Похибка за методом хорд : " + *abs*(calculate[0] - natural[0])  
 + " та " + *abs*(calculate[1] - natural[1]));  
  
 ////////////////Дотичних(Ньютона)  
 System.*out*.println("\nРозв'язки рівняння за методом дотичних(Ньютона) (точність 0.01 ) : "  
 + Arrays.*toString*(calculate = TangentialMethod.*tangentialMethod*(0, 10, 1, 0.01, 3)));  
  
 System.*out*.println("Похибка за методом дотичних(Ньютона) : " + *abs*(calculate[0] - natural[0])  
 + " та " + *abs*(calculate[1] - natural[1]));  
  
 */\*\* ///////////////////////////////////////////////////////////////////////////////////////////////////////// \*/* System.*out*.println("\nЗАВДАННЯ 2");  
  
 double[] segment = new double[]{14, 16};  
  
 System.*out*.println((IterationMethod.*convergence*(segment) ? "Метод простих ітерацій збіжний на " :  
 "Метод простих ітерацій НЕ збіжний на ") + Arrays.*toString*(segment));  
  
 System.*out*.println("Корінь функції : "  
 + Arrays.*toString*(natural = IterationMethod.*iterationMethod*(0, 20, 1, *pow*(0.01, 10), 2)));  
  
 System.*out*.println("Розв'язки рівняння за методом простих ітерацій (точність 0.01 ) : "  
 + Arrays.*toString*(calculate = IterationMethod.*iterationMethod*(0, 20, 1, 0.01, 2)));  
  
 System.*out*.println("Похибка за методом простих ітерацій : " + *abs*(calculate[0] - natural[0]));  
  
  
 */\*\* ///////////////////////////////////////////////////////////////////////////////////////////////////////// \*/* System.*out*.println("\nЗАВДАННЯ 3");  
  
  
 System.*out*.println("Розв'язок СЛАР методом Крамера : "  
 + Arrays.*toString*(KramerMethod.*kramerMethod*(MyFunctions.*matrixTask3*())));  
  
  
 */\*\* ///////////////////////////////////////////////////////////////////////////////////////////////////////// \*/* System.*out*.println("\nЗАВДАННЯ 4");  
  
 Function<double[], Double>[][] functions = new Function[3][2];  
 functions[0][0] = MyFunctions::*functionTask4X*;  
 functions[0][1] = MyFunctions::*functionTask4Y*;  
 functions[1][0] = MyFunctions::*functionTask4XderX*;  
 functions[1][1] = MyFunctions::*functionTask4XderY*;  
 functions[2][0] = MyFunctions::*functionTask4YderX*;  
 functions[2][1] = MyFunctions::*functionTask4YderY*;  
  
 System.*out*.println("Корені функції : "  
 + Arrays.*toString*(natural = SeidelMethod.*seildelMethod*(new double[]{1, 1}, *pow*(0.01, 10), functions)));  
  
  
 System.*out*.println("Розв'язок СНАР методом Зейделля : "  
 + Arrays.*toString*(calculate = SeidelMethod.*seildelMethod*(new double[]{1, 1}, 0.01, functions)));  
  
 System.*out*.println("Похибка за методом Зейделля : " + *abs*(calculate[0] - natural[0])  
 + " та " + *abs*(calculate[1] - natural[1]));  
  
 functions[0][0] = MyFunctions::*functionTask4F1*;  
 functions[0][1] = MyFunctions::*functionTask4F2*;  
 functions[1][0] = MyFunctions::*functionTask4F1derX*;  
 functions[1][1] = MyFunctions::*functionTask4F1derY*;  
 functions[2][0] = MyFunctions::*functionTask4F2derX*;  
 functions[2][1] = MyFunctions::*functionTask4F2derY*;  
  
  
 System.*out*.println("\nРозв'язок СНАР методом Ньютона : "  
 + Arrays.*toString*(calculate = NewtonMethod.*newtonMethod*(new double[]{-2, -9}, 0.01, functions)));  
  
 System.*out*.println("Похибка за методом Ньютона : " + *abs*(calculate[0] - natural[0])  
 + " та " + *abs*(calculate[1] - natural[1]));  
 }  
}

MyFunctions.java

package lab2;  
  
  
import static java.lang.Math.\*;  
  
public class MyFunctions {  
  
 public static final int *G* = 3; //3  
 public static final int *K* = 2; //2  
  
 public static double functionTask1(double x) {  
 return *pow*(x - *G* \* *K*, 2) + *sin*(x - *G* \* *K*);  
 }  
  
 public static double functionTask1der1(double x) {  
 return 2 \* (x - *G* \* *K*) + *cos*(x - *G* \* *K*);  
 }  
  
 public static double functionTask1der2(double x) {  
 return 2 - *sin*(x - *G* \* *K*);  
 }  
  
 public static double functionTask2(double x) {  
 return (*K* \* x - 10 \* *G*) - *sin*(x - (double) (10 \* *G*) / *K*);  
 }  
  
 public static double functionTask2fi(double x) {  
 return (10 \* *G* + *sin*(x - (double) (10 \* *G*) / *K*)) / *K*;  
 }  
  
 public static double[][] matrixTask3() {  
 return new double[][]{  
 {*G* + 1, *G* + 2, *G* + 3, *K*},  
 {2 \* (*G* + 1), *G* + 6, *G* - 5, *K* + 1},  
 {3 \* (*G* + 1), *G*, -3, *K* + 2}};  
 }  
  
 //G(X,Y) = X || Y  
  
 public static double functionTask4X(double[] x) {  
 return (4 \* *G* - (*sin*((*K* + 1) \* x[0] + x[1] - 4 \* *G*)) / 10) / (*K* + 1);  
 }  
  
 public static double functionTask4Y(double[] x) {  
 return (*sin*((*K* + 1) \* x[0] + x[1] - 4 \* *G*)) / (10 \* (*G* + 1));  
 }  
  
 public static double functionTask4XderX(double[] x) {  
 return -(*cos*((*K* + 1) \* x[0] + x[1] - 4 \* *G*)) / 10;  
 }  
  
 public static double functionTask4XderY(double[] x) {  
 return -*cos*((*K* + 1) \* x[0] + x[1] - 4 \* *G*) / (10 \* *K* + 10);  
 }  
  
 public static double functionTask4YderX(double[] x) {  
 return (*K* + 1) \* *cos*((*K* + 1) \* x[0] + x[1] - 4 \* *G*) / (10 \* (*G* + 1));  
 }  
  
 public static double functionTask4YderY(double[] x) {  
 return *cos*((*K* + 1) \* x[0] + x[1] - 4 \* *G*) / (10 \* (*G* + 1));  
 }  
  
  
 //F(X,Y) = 0  
  
 public static double functionTask4F1(double[] x) {  
 return (*K* + 1) \* x[0] - 4 \* *G* - (*sin*((*K* + 1) \* x[0] + x[1] - 4 \* *G*)) / 10;  
 }  
  
 public static double functionTask4F2(double[] x) {  
 return x[1] - *sin*((*K* + 1) \* x[0] + x[1] - 4 \* *G*) / (10 \* (*G* + 1));  
 }  
  
 public static double functionTask4F1derX(double[] x) {  
 return (*K* + 1) + (*cos*((*K* + 1) \* x[0] + x[1] - 4 \* *G*) \* (*K* + 1)) / 10;  
 }  
  
 public static double functionTask4F1derY(double[] x) {  
 return *cos*((*K* + 1) \* x[0] + x[1] - 4 \* *G*) / 10;  
 }  
  
 public static double functionTask4F2derX(double[] x) {  
 return -(*cos*((*K* + 1) \* x[0] + x[1] - 4 \* *G*) \* (*K* + 1)) / (10 \* (*G* + 1));  
 }  
  
 public static double functionTask4F2derY(double[] x) {  
 return 1 - *cos*((*K* + 1) \* x[0] + x[1] - 4 \* *G*) / (10 \* (*G* + 1));  
 }  
  
  
}

SearchSegment.java

package lab2;  
  
  
import java.util.function.UnaryOperator;  
  
public class SearchSegment {  
  
 public static double[][] searchSegment(double start, double end, double step,  
 int num, UnaryOperator<Double> function) {  
  
 if (step <= 0 || start + step > end) try {  
 throw new Exception("Error in Searching segments");  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
  
 double[][] segments = new double[num][2];  
  
 for (int i = 0; i < num; i++) {  
  
 segments[i] = null;  
  
 for (double j = start; j < end; j = j + step) {  
  
  
 if (function.apply(j) \* function.apply(j + step) < 0) {  
 segments[i] = new double[]{j, j + step};  
 break;  
 }  
 if (function.apply(j + step) == 0) {  
 segments[i] = new double[]{j, j + step};  
 j -= step;  
 step /= 1.1;  
 }  
  
 }  
 //System.out.println(Arrays.toString(segments[i]));  
  
 if (segments[i] == null) {  
 double[][] result = new double[i][2];  
 System.*arraycopy*(segments, 0, result, 0, i);  
 return result;  
 }  
 start = segments[i][1];  
 }  
  
  
  
 return segments;  
  
 }  
  
// public static double[] searchSegment(double start, double end, double step, UnaryOperator<Double> function) {  
//  
// for (double i = start; i < end; i = i + step) {  
// if (function.apply(i) \* function.apply(i + step) <= 0) {  
// return new double[]{i, i + step};  
// }  
// }  
//  
// return null;  
// }  
  
  
}  
  
*/\*\*  
 \* if (function.apply(j) == 0) ...  
 \* if (function.apply(j + step) == 0) ...  
 \* <p>  
 \* Даний функціонал спрощений,  
 \* оскільки суть лабораторної роботи полягає у використані обчислювальних методів  
 \* для знаходження коренів функції з певною точністю,  
 \* а не у вгадуванні коренів (див. код вище).  
 \* <p>  
 \* P.S. Особисто в мене, завдання 2 не мало б жожного сенсу.  
 \*/*

Matrix.java

package lab2;  
  
  
public class Matrix {  
  
 public static double[][] TransponateMatrix(double[][] arr) {  
  
 double[][] arrT = new double[arr[0].length][arr.length];  
 for (int i = 0; i < arr.length; i++)  
 for (int j = 0; j < arr[0].length; j++)  
 arrT[j][i] = arr[i][j];  
  
 return arrT;  
 }  
  
 public static double[][] MultiplyMatrix(double[][] arr1, double[][] arr2) {  
  
 double[][] arr = new double[arr1.length][arr2[0].length];  
  
 for (int i = 0; i < arr1.length; i++)  
 for (int j = 0; j < arr2[0].length; j++) {  
 arr[i][j] = 0;  
 for (int k = 0; k < arr2.length; k++)  
 arr[i][j] += arr1[i][k] \* arr2[k][j];  
  
 }  
  
  
 return arr;  
 }  
  
 public static double DeterminateMatrix(double[][] arr) {  
  
 if (arr.length != arr[0].length) try {  
 throw new Exception("Matrix isn't square!");  
 } catch (Exception e) {  
 e.printStackTrace();  
 return 0;  
 }  
  
 if (arr.length == 1) return arr[0][0];  
 int l = arr.length - 1;  
 double num = 0;  
  
 for (int i = 0, j = 0; i < arr.length; i++) {  
  
  
 double[][] temp = new double[l][l];  
  
 for (int k = 0, n = 0; n < l \* l; k++) {  
 if (!(k / arr.length == i || k % arr.length == j))  
 temp[n / l][n++ % l] = arr[k / arr.length][k % arr.length];  
 }  
  
 double d = arr[i][j] \* *DeterminateMatrix*(temp);  
 if ((i + j) % 2 == 1) d \*= -1;  
 num += d;  
  
 }  
  
 return num;  
 }  
  
 public static double[][] ReversedMatrix(double[][] arr) {  
  
 double det = *DeterminateMatrix*(arr);  
  
 if (det == 0) {  
 System.*out*.println("Визначник дорівнює нулю - матриця вироджена.");  
 System.*exit*(-1);  
 }  
  
 double detR = 1 / det;  
  
 double[][] arrT = *TransponateMatrix*(arr);  
  
 double[][] arrR = *AllyMatrix*(arrT);  
  
 return *MultiplyMatrix*(arrR, detR);  
 }  
  
 public static double[][] MultiplyMatrix(double[][] arr, double det) {  
  
 double[][] returned = new double[arr.length][arr[0].length];  
  
 for (int i = 0; i < arr.length; i++)  
 for (int j = 0; j < arr[0].length; j++)  
 returned[i][j] = det \* arr[i][j];  
  
 return returned;  
  
 }  
  
 public static double[][] AllyMatrix(double[][] arr) {  
  
 double[][] arrA = new double[arr.length][arr.length];  
  
 for (int i = 0; i < arr.length; i++)  
 for (int j = 0; j < arr.length; j++) {  
  
 int l = arr.length - 1; // 2  
 double[][] temp = new double[l][l];  
  
 for (int k = 0, n = 0; n < l \* l; k++) {  
 if (!(k / arr.length == i || k % arr.length == j))  
 temp[n / l][n++ % l] = arr[k / arr.length][k % arr.length];  
 }  
  
 arrA[i][j] = *DeterminateMatrix*(temp);  
 if ((i + j) % 2 == 1) arrA[i][j] \*= -1;  
  
 }  
 return arrA;  
 }  
  
 public static double[][] MatrixForKramer(double[][] fullMatrix, int replace) {  
  
  
 int col = fullMatrix[0].length;  
 int row = fullMatrix.length;  
  
 if (col <= replace) try {  
 throw new Exception("Column is out of border!");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 double[][] matrix = new double[row][row];  
  
 if (col == replace + 1) {  
 for (int i = 0; i < row; i++) {  
 System.*arraycopy*(fullMatrix[i], 0, matrix[i], 0, row);  
 }  
 } else {  
 for (int i = 0; i < row; i++) {  
 System.*arraycopy*(fullMatrix[i], 0, matrix[i], 0, replace);  
 System.*arraycopy*(fullMatrix[i], col - 1, matrix[i], replace, 1);  
 System.*arraycopy*(fullMatrix[i], replace + 1, matrix[i], replace+1, row - replace - 1 );  
 }  
 }  
  
// for (int i = 0; i < matrix.length; i++)  
// System.out.println(Arrays.toString(matrix[i]));  
  
 return matrix;  
  
 }  
  
 public static double[][] SubtractMatrix(double[][] arr1, double[][] arr2){  
  
 if (arr1.length != arr2.length || arr1[0].length != arr2[0].length) return null;  
  
 double arr[][] = new double[arr1.length][arr1[0].length];  
  
 for (int i = 0 ; i < arr1.length; i++){  
 for (int j = 0 ; j < arr1[0].length; j++){  
 arr[i][j] = arr1[i][j] - arr2[i][j];  
 }  
 }  
 return arr;  
 }  
  
 public static double NormaByRow(double[][] arr){  
 double max = arr[0][0];  
  
 for (int i = 1; i < arr.length; i++) {  
 double sum = 0;  
 for (int j = 0; j < arr[0].length; j++) {  
 sum += Math.*abs*(arr[i][j]);  
 }  
 if (sum > max) max = sum;  
 }  
  
 return max;  
 }  
  
  
}

ChordsMethod.java

package lab2.MethodsNonlinear;  
  
import lab2.MyFunctions;  
  
import static java.lang.Math.*abs*;  
import static lab2.MyFunctions.*functionTask1*;  
import static lab2.MyFunctions.*functionTask1der2*;  
import static lab2.SearchSegment.*searchSegment*;  
  
public class ChordsMethod {  
  
  
 public static double moveTo(double[] segment) {  
  
 if (*functionTask1*(segment[0]) \* *functionTask1der2*(segment[0]) > 0) return segment[0];  
 else if (*functionTask1*(segment[1]) \* *functionTask1der2*(segment[1]) > 0) return segment[1];  
  
 return 0 ;  
 }  
  
 public static double first(double[] segment){  
 return segment[0] - (*functionTask1*(segment[0]) \* (segment[1] - segment[0])) /  
 (*functionTask1*(segment[1]) - *functionTask1*(segment[0]));  
 }  
  
 public static double findСrossing(double prev, double moveTo, double error) {  
  
 double cross = prev - (*functionTask1*(prev) \* (prev - moveTo)) /  
 (*functionTask1*(prev) - *functionTask1*(moveTo));  
  
 if (*abs*(cross - prev) <= error) return cross;  
  
 return *findСrossing*(cross, moveTo, error);  
 }  
  
 public static double[] chordsMethod(double start, double end, double step, double error, int num) {  
  
 double[][] segments = *searchSegment*(start, end, step, num, MyFunctions::*functionTask1*);  
 double[] result = new double[segments.length];  
  
  
 for (int i = 0; i < segments.length; i++) {  
 result[i] = *findСrossing*( *first*(segments[i]), *moveTo*(segments[i]), error);  
 }  
  
 return result;  
  
 }  
  
}

DivisionMethod.java

package lab2.MethodsNonlinear;  
  
import lab2.MyFunctions;  
  
import static java.lang.Math.*abs*;  
import static lab2.MyFunctions.*functionTask1*;  
import static lab2.SearchSegment.*searchSegment*;  
  
public class DivisionMethod {  
  
  
 public static double divideSegment(double[] segment, double prev , double error) {  
  
 double middle = (segment[0] + segment[1]) / 2;  
  
 if (*abs*(*functionTask1*(middle) - *functionTask1*(prev) ) <= error) return middle;  
  
 else if (*functionTask1*(segment[0]) \* *functionTask1*(middle) < 0) segment[1] = middle;  
  
 else if (*functionTask1*(segment[1]) \* *functionTask1*(middle) < 0) segment[0] = middle;  
  
 return *divideSegment*(segment , middle , error);  
 }  
  
 public static double[] divisionMethod(double start, double end, double step , double error , int num) {  
  
 double[][] segments = *searchSegment*(start, end, step, num, MyFunctions::*functionTask1*);  
  
 double[] result = new double[segments.length];  
  
 for (int i = 0; i < segments.length ; i++) {  
 result[i] = *divideSegment*(segments[i] , 0 , error);  
 }  
  
 return result;  
  
 }  
  
}

IterationMethod.java

package lab2.MethodsNonlinear;  
  
import lab2.MyFunctions;  
  
import static java.lang.Math.*abs*;  
import static lab2.MyFunctions.*functionTask2fi*;  
import static lab2.SearchSegment.*searchSegment*;  
  
  
public class IterationMethod {  
 public static double first(double[] segment) {  
 return (segment[0] + segment[1]) / 2;  
 }  
  
 public static double simpleIteration(double prev, double error) {  
  
 double cross = *functionTask2fi*(prev);  
  
 if (*abs*(cross - prev) <= error) return cross;  
  
 return *simpleIteration*(cross, error);  
 }  
  
 public static double[] iterationMethod(double start, double end, double step, double error, int num) {  
  
  
 double[][] segments = *searchSegment*(start, end, step, num, MyFunctions::*functionTask2*);  
  
 double[] result = new double[segments.length];  
  
 for (int i = 0; i < segments.length; i++) {  
 result[i] = *simpleIteration*(*first*(segments[i]), error);  
 }  
  
 return result;  
 }  
  
 public static boolean convergence(double[] segment) {  
  
 for (double v : segment) {  
 double value = *abs*(MyFunctions.*functionTask2fiDer1*(v));  
 if ( value >= 1 || value == 0 ) return false;  
 }  
  
 return true;  
 }  
}

TangentialMethod.java

package lab2.MethodsNonlinear;  
  
import lab2.MyFunctions;  
  
import static java.lang.Math.*abs*;  
import static lab2.MyFunctions.\*;  
import static lab2.SearchSegment.*searchSegment*;  
  
public class TangentialMethod {  
  
 public static double moveFrom(double[] segment) {  
  
 if (*functionTask1*(segment[0]) \* *functionTask1der2*(segment[0]) > 0) return segment[0];  
  
 else if (*functionTask1*(segment[1]) \* *functionTask1der2*(segment[1]) > 0) return segment[1];  
  
 return 0;  
 }  
  
 public static double tangential(double prev, double error) {  
  
 double cross = prev - *functionTask1*(prev) / *functionTask1der1*(prev);  
  
 if (*abs*(cross - prev) <= error) return cross;  
  
 return *tangential*(cross, error);  
 }  
  
 public static double[] tangentialMethod(double start, double end, double step, double error, int num) {  
  
 double[][] segments = *searchSegment*(start, end, step, num, MyFunctions::*functionTask1*);  
  
 double[] result = new double[segments.length];  
  
 for (int i = 0; i < segments.length; i++) {  
 result[i] = *tangential*(*moveFrom*(segments[i]), error);  
 }  
  
 return result;  
  
 }  
  
  
}

NewtonMethod

package lab2.MethodsNonolinearSystem;  
  
import lab2.Matrix;  
  
import java.util.Arrays;  
import java.util.function.Function;  
  
import static java.lang.Math.*abs*;  
import static java.lang.Math.*pow*;  
  
public class NewtonMethod {  
  
 public static double[] newtonIteration(double[] x, double error, Function<double[], Double>[][] functions) {  
  
  
 double[][] y = new double[x.length][1];  
 double[][] jacobi = new double[x.length][x.length];  
  
 for (int i = 0; i < x.length; i++) {  
 y[i][0] = functions[0][i].apply(x);  
 for (int j = 0; j < x.length; j++) {  
 jacobi[i][j] = functions[i + 1][j].apply(x);  
 }  
 }  
  
 double[][] jacobiR = Matrix.*ReversedMatrix*(jacobi);  
  
  
 double[][] diff = Matrix.*MultiplyMatrix*(jacobiR, y);  
  
 double[][] colX = new double[x.length][1];  
 for (int i = 0; i < x.length; i++) colX[i][0] = x[i];  
  
 double[][] result = Matrix.*SubtractMatrix*(colX, diff);  
  
 boolean end = true;  
  
  
 for (int i = 0; i < result.length; i++) {  
 if (*abs*(diff[i][0]) > error) {  
 end = false;  
 break;  
 }  
 }  
 if (end) return Matrix.*TransponateMatrix*(result)[0];  
 return *newtonIteration*(Matrix.*TransponateMatrix*(result)[0], error, functions);  
  
  
 }  
  
 public static double[] newtonMethod(double[] x, double error, Function<double[], Double>[][] functions) {  
  
 if(*convergence*(x, functions))  
 System.*out*.println("За достатньою умовою, метод Ньютона збігається при початкових значеннях "  
 + Arrays.*toString*(x));  
  
 return *newtonIteration*(x, error, functions);  
  
 }  
  
 public static boolean convergence(double[] x, Function<double[], Double>[][] functions) {  
  
  
 double[][] y = new double[x.length][1];  
 double[][] jacobian = new double[x.length][x.length];  
  
 for (int i = 0; i < x.length; i++) {  
 y[i][0] = functions[0][i].apply(x);  
 for (int j = 0; j < x.length; j++) {  
 jacobian[i][j] = functions[i + 1][j].apply(x);  
 }  
 }  
  
 double[][] jacobianR = Matrix.*ReversedMatrix*(jacobian);  
  
 double L = Matrix.*NormaByRow*(jacobian);  
 double M = Matrix.*NormaByRow*(jacobianR);  
 double P = Matrix.*NormaByRow*(y);  
  
 return (*pow*(M, 2) \* L \* P / 2 < 1);  
 }  
  
  
}

SeidelMethod.java

package lab2.MethodsNonolinearSystem;  
  
import java.util.Arrays;  
import java.util.function.Function;  
  
  
import static java.lang.Math.*pow*;  
import static java.lang.Math.*sqrt*;  
  
public class SeidelMethod {  
  
  
 public static double[] approximate(double[] x, double error, Function<double[], Double>[][] functions) {  
  
 double[] y = new double[x.length];  
 double[] copyX = new double[x.length];  
 System.*arraycopy*(x, 0, copyX, 0, x.length);  
  
  
 double r = 0;  
 for (int i = 0; i < x.length; i++) {  
 y[i] = functions[0][i].apply(copyX);  
 copyX[i] = y[i];  
 r += *pow*(x[i] - y[i], 2);  
  
 }  
 if (*sqrt*(r) < error) return y;  
 return *approximate*(y, error, functions);  
 }  
  
 public static double[] seildelMethod(double[] x, double error, Function<double[], Double>[][] functions) {  
  
 if (*convergence*(x, functions))  
 System.*out*.println("За достатньою умовою, метод Зейделля збігається при початкових значеннях "  
 + Arrays.*toString*(x));  
  
 return *approximate*(x, error, functions);  
  
 }  
  
 public static boolean convergence(double[] x, Function<double[], Double>[][] functions) {  
 for (int i = 1; i < functions.length; i++) {  
 double sum = 0;  
 for (int j = 0; j < functions[0].length; j++) {  
 sum += Math.*abs*(functions[i][j].apply(x));  
 }  
 if (sum > 1) return false;  
  
 }  
 return true;  
 }  
  
}

KramerMethod.java

package lab2.MethodsSLAR;  
  
import lab2.Matrix;  
  
public class KramerMethod {  
 public static double[] kramerMethod(double[][] fullMatrix) {  
 int col = fullMatrix[0].length;  
 int row = fullMatrix.length;  
  
 double[] deters = new double[row + 1];  
 double[] result = new double[row];  
 double[][] matrix;  
  
 matrix = Matrix.*MatrixForKramer*(fullMatrix, col - 1);  
 deters[0] = Matrix.*DeterminateMatrix*(matrix);  
  
 //System.out.println(deters[0]);  
  
 if (deters[0] == 0) {  
 boolean InfinitySolutions = true;  
 for (int i = 0; i < row ; i++) {  
 matrix = Matrix.*MatrixForKramer*(fullMatrix, i);  
 deters[i + 1] = Matrix.*DeterminateMatrix*(matrix);  
 if(deters[i+1] != 0) {  
 InfinitySolutions = false;  
 break;  
 }  
  
 }  
 System.*out*.println( InfinitySolutions ? "Безліч розв'язків" : "Немає розв'язків" );  
 return null;  
  
 } else for (int i = 0; i < row; i++) {  
  
 matrix = Matrix.*MatrixForKramer*(fullMatrix, i);  
 deters[i + 1] = Matrix.*DeterminateMatrix*(matrix);  
  
 result[i] = deters[i + 1] / deters[0];  
  
 //System.out.println(deters[i+1]);  
  
 }  
  
 return result;  
  
 }  
  
  
}

**Висновки**

Під час цієї лабораторної роботи були вивчені різні методи для пошуку коренів функції, а саме : метод половинного ділення, метод хорд, метод дотичних, метод простих ітерацій. Для визначення коренів систем лінійних алгебраїчних рівнянь (СЛАР) освоєний прямий(точний) метод Крамера. Для визначення коренів систем транспарентних рівнянь були використані ітераційні методи Ньютона та Зейделя.