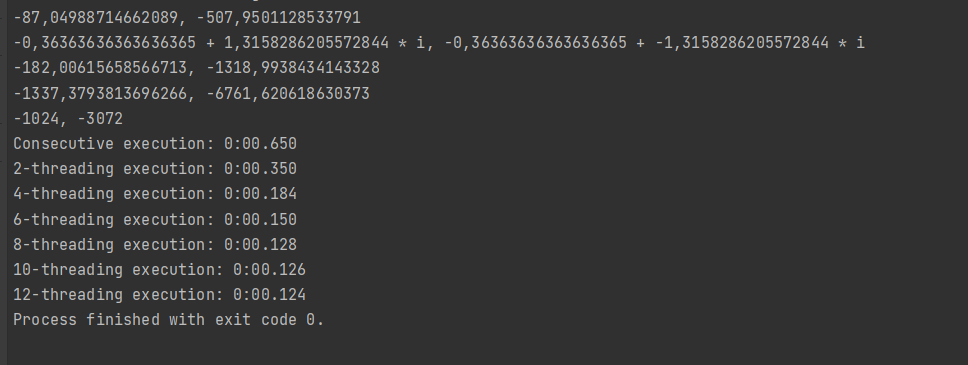
# Лабораторная работа 1

Программа решает 2 500 000 квадратных уравнений со случайными коэффициентами выполняя программу последовательно и одновременно в нескольких потоках (2, 4, 6, 8, 10, 12). Программа выводит время, понадобившееся для каждого рассчета, также выводит решения пяти первых квадратных уравнений.

## Вывод:



## Код программы:

**Program.cs**

internal static class Program

{

private static void Main()

{

var numbers = NumberGenerator.GenerateNumbersArray();

ConsecutiveExecution(numbers);

ChooseThreadingExecution(numbers, 2);

ChooseThreadingExecution(numbers, 4);

ChooseThreadingExecution(numbers, 6);

ChooseThreadingExecution(numbers, 8);

ChooseThreadingExecution(numbers, 10);

ChooseThreadingExecution(numbers, 12);

}

private static void ChooseThreadingExecution(int[][] numbers, int threads)

{

var parallelOptions = new ParallelOptions

{

MaxDegreeOfParallelism = threads

};

Console.Write($"\n{threads}-threading execution: ");

MultiThreadingExecution(numbers, parallelOptions);

}

private static void MultiThreadingExecution(int[][] numbers, ParallelOptions parallelOptions)

{

var stopwatch = new Stopwatch();

stopwatch.Start();

Parallel.For(

0,

numbers.Length,

parallelOptions,

i =>

{

var quadraticEquationSolver = new QuadraticEquationSolver(

numbers[i][0],

numbers[i][1],

numbers[i][2]

);

quadraticEquationSolver.Solve();

});

var timeTaken = stopwatch.Elapsed;

Console.Write(timeTaken.ToString(@"m\:ss\.fff"));

}

private static void ConsecutiveExecution(int[][] numbers)

{

var stopwatch = new Stopwatch();

stopwatch.Start();

for (var i = 0; i < 5; i++)

{

var quadraticEquationSolver = new QuadraticEquationSolver(

numbers[i][0],

numbers[i][1],

numbers[i][2]

);

quadraticEquationSolver.SolveAndOutputAnswer();

}

Console.Write("Consecutive execution: ");

for (var i = 5; i < numbers.Length; i++)

{

var quadraticEquationSolver = new QuadraticEquationSolver(

numbers[i][0],

numbers[i][1],

numbers[i][2]

);

quadraticEquationSolver.Solve();

}

stopwatch.Stop();

var timeTaken = stopwatch.Elapsed;

Console.Write(timeTaken.ToString(@"m\:ss\.fff"));

}

**NumberGenerator.cs**

public class NumberGenerator

{

private const int MinValue = 1;

private const int MaxValue = 100;

private const int ArrayCount = 2500000;

private const int NumbersInArrayCount = 3;

public static int[][] GenerateNumbersArray()

{

var randNum = new Random();

var result = new List<int[]>();

for (var i = 0; i < ArrayCount; i++)

{

var numbersList = new List<int>();

for (var j = 0; j < NumbersInArrayCount; j++)

{

numbersList.Add(randNum.Next(MinValue, MaxValue));

}

result.Add(numbersList.ToArray());

}

return result.ToArray();

}

**QuadraticEquationSolver.cs**

public class QuadraticEquationSolver

{

private double a;

private double b;

private double c;

public QuadraticEquationSolver(double a, double b, double c)

{

this.a = a;

this.b = b;

this.c = c;

}

public void Solve()

{

var discriminant = GetDiscriminant();

if (discriminant >= 0)

{

var realSolutions = GetRealSolutions(discriminant);

}

else

{

var imaginarySolutions = GetImaginarySolutions(discriminant);

}

}

public void SolveAndOutputAnswer()

{

var discriminant = GetDiscriminant();

if (discriminant >= 0)

{

var realSolutions = GetRealSolutions(discriminant);

Console.WriteLine(realSolutions.Item1 + ", " + realSolutions.Item2);

}

else

{

var imaginarySolutions = GetImaginarySolutions(discriminant);

Console.WriteLine(imaginarySolutions.Item1 + " + " + imaginarySolutions.Item2 + " \* i, " + imaginarySolutions.Item1 + " + " + imaginarySolutions.Item3 + " \* i ");

}

}

private (double, double) GetRealSolutions(double discriminant)

{

var x1 = (-b + Math.Sqrt(discriminant)) / 2 \* a;

var x2 = (-b - Math.Sqrt(discriminant)) / 2 \* a;

return (x1, x2);

}

private (double, double, double) GetImaginarySolutions(double discriminant)

{

var x1 = -b / (2 \* a);

discriminant = -discriminant ;

var x2 = Math.Sqrt(discriminant) / (2 \* a);

var x3 = -Math.Sqrt(discriminant) / (2 \* a);

return (x1, x2, x3);

}

private double GetDiscriminant()

{

return Math.Pow(b, 2) - 4 \* a \* c;

}

}