Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление системного и прикладного программного обеспечения

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

курса «Вычислительная математика»

по теме: «Решение нелинейных уравнений и систем нелинейных уравнений»

Вариант № 21

Выполнил студент:

Тюрин Иван Николаевич

группа: P32131

Преподаватель:

Малышева Т. А.,

Бострикова Д. К.

Санкт-Петербург, 2023 г.

Содержание

Лабораторная работа № 2 Решение нелинейных уравнений и систем нелинейных уравнений

1. Задание варианта № 21

, , ,

Для вычислительной части: метод хорд (1), метод половинного деления (2), метод простой итерации (3) и функция

$$1,8 \cdot x^3 - 2,47 \cdot x^2 - 5,53 \cdot x + 1,539.$$

Для программной части: метод ньютона, метод хорд, метод простой итерации для уравнения и системы уравнений.

, , ,

2. Цель работы

Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного равнения/системы нелинейных уравнений, выполнить программную реализацию методов.

3. Вычислительная часть работы

3.1. Подготовка к вычислениям

Отсделилили корни заданого нелинейного уравнения графически с помощью программы Desmos. График функции на интервале $[-5; 5]$ можно видеть на рисунке 1.1.

Получили следующие результаты:

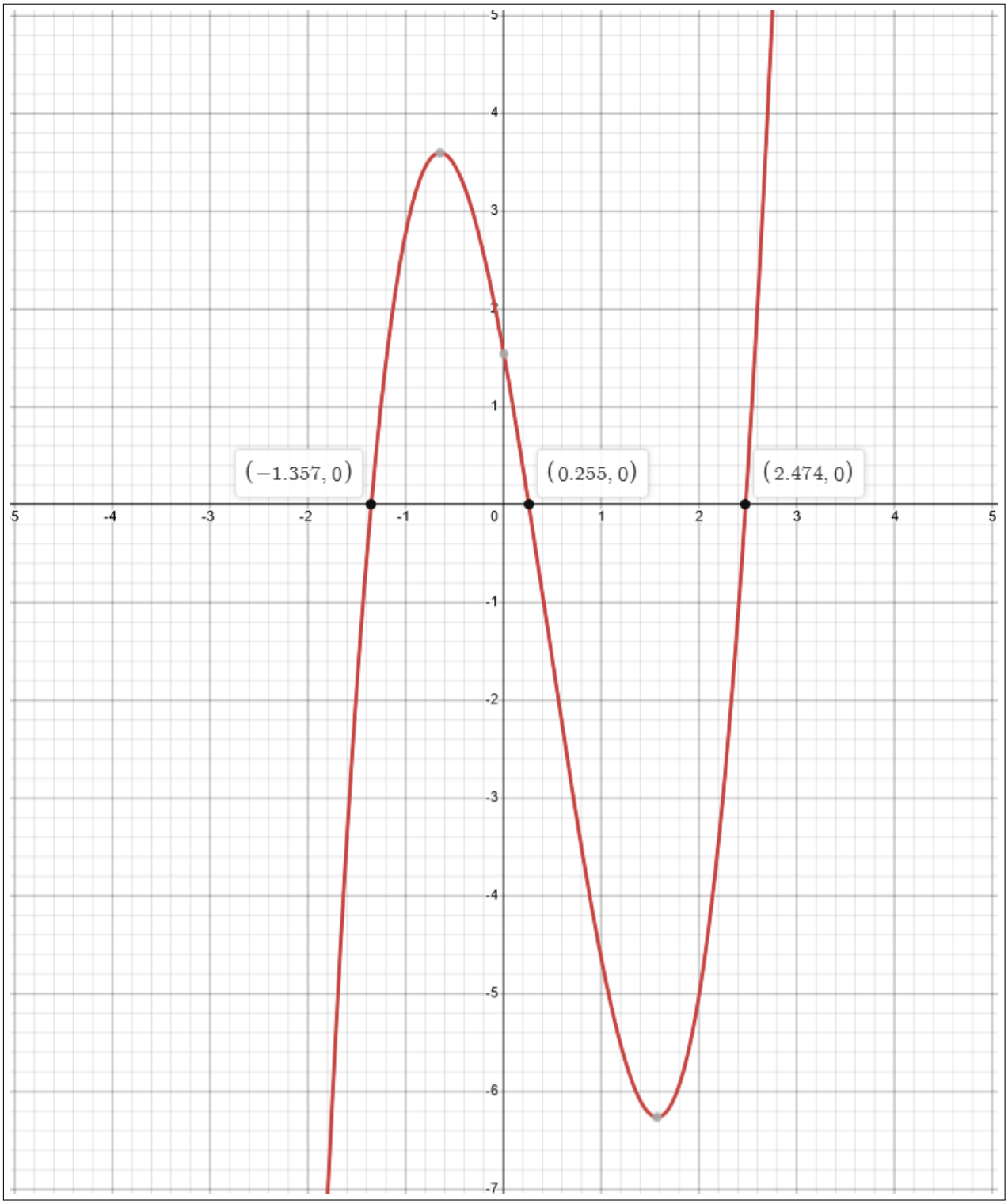

Рис. 1.1: График данной функции на промежутке $[-5; 5]$

k	a	b	x	$f(a)$	$f(b)$	$f(x)$	$\lvert x_{k+1} - x_k \rvert$
1,0000	-1,4000	-1,3000	-1,3545	-0,4994	0,5991	0,0242	-
2,0000	-1,4000	-1,3545	-1,3566	-0,4994	0,0242	0,0009	0,00210
3,0000	-1,4000	-1,3566	-1,3567	-0,4994	0,0009	0,0000	0,00008
4,0000	-1,4000	-1,3567	-1,3567	-0,4994	0,0000	0,0000	0,00000

Таблица 1.1: Нахождение первого корня уравнения методом хорд

k	a	b	x	$f(a)$	$f(b)$	$f(x)$	$\lvert a - b \rvert$
1,0000	0,2000	0,3000	0,2500	0,3486	-0,2937	0,0302	0,1000
2,0000	0,2500	0,3000	0,2750	0,0302	-0,2937	-0,1311	0,0500
3,0000	0,2500	0,2750	0,2625	0,0302	-0,1311	-0,0503	0,0250
4,0000	0,2500	0,2625	0,2563	0,0302	-0,0503	-0,0100	0,0125
5,0000	0,2500	0,2563	0,2531	0,0302	-0,0100	0,0102	0,0062

Таблица 1.2: Нахождение второго корня уравнения методом половинного деления

- первый корень $x_1 \in [-1,4;\ -1,3]$,
- второй корень $x_2 \in [0,2;\ 0,3]$,
- третий корень $x_3 \in [2,4;\ 2,5]$.

3.2. Вычисления

Используя табличный процессор Google Sheets вычислили приближенные значения корней уравнения.

1. Вычисление первого корня методом хорд можно видеть в табдице 1.1.
2. Вычисление второго корня методом половинного деления можно видеть в табдице 1.2.
3. Вычисление третьего корня методом простой итерации можно видеть в табдице 1.3.

Важно при этом уточнить, что при вычислении третьего корня методом простой итерации была использована функция приближения полученная после выражения переменной из старшего члема многочлена: $x_{i+1} = \sqrt[3]{(2,47 \cdot x_i^2 + 5,53 \cdot x_i + 1,539)/1,8}$.

k	x_k	$x_{k+1} = \sqrt[3]{...}$	$f(x_{k+1})$	$\lvert x_{k+1} - x_k \rvert$
1	-2	-1,147393101	1,913301856	0,8526068992
2	-1,147393101	-1,370381379	-0,1536172866	0,2229882787
3	-1,370381379	-1,355062491	0,01841095071	0,01531888879
4	-1,355062491	-1,356916747	-0,002166862467	0,001854255854

Таблица 1.3: Нахождение третьего корня уравнения методом простой итерации

4. Программная часть работы

4.1. Листинг программы

Основную часть программной реализации на языке программировани Kotlin можно посмотреть в листинге 1.1. Весь код представлен в личном репозитории [1].

```
class ChordSolvingMethod(
    private val function: (Double) -> Double
) : SolvingMethod<ClosedRange<Double>> {

    override fun nextApproximation(current: ClosedRange<Double>):
    ClosedRange<Double> {
        val startValue = function(current.start)
        val endValue = function(current.endInclusive)
        require(startValue * endValue < 0) { /* ... */ }

        val middle = current.start - (current.endInclusive - current.start
    ) * startValue / (endValue - startValue)
        val middleValue: Double = function(middle)

        return if (startValue * middleValue < 0) {
            current.start..middle
        } else if (endValue * middleValue < 0) {
            middle..current.endInclusive
        } else {
            middle..middle
        }
    }

}

class NewtonSolvingMethod @Throws(IllegalArgumentException::class)
    constructor(
    range: ClosedRange<Double>,
    private val function: (Double) -> Double,
    private val derivative: (Double) -> Double = function.derivative
) : SolvingMethod<Double> {

    init { /* ... */ }

    @Throws(IllegalStateException::class)
    override fun nextApproximation(current: Double): Double {
        val tan = derivative(current)

```

```
        check(tan != 0.0) { /* ... */ }

        val h = function(current) / tan
        val x = current - h
        return x
    }

    companion object {
        @Throws(IllegalArgumentException::class)
        fun initialApproximation(
            range: ClosedRange<Double>,
            function: (Double) -> Double,
            firstDerivative: (Double) -> Double = function.derivative,
            secondDerivative: (Double) -> Double = firstDerivative.
    derivative
        ): Double {
            return if (function(range.start) * secondDerivative(range.
    start) > 0) range.start
            else if (function(range.endInclusive) * secondDerivative(range
    .endInclusive) > 0) range.endInclusive
            else throw IllegalArgumentException("Function value and it's 2
    nd derivative value must have same sign and not be zero at least in one
     bound of range")
        }
    }

}

class SimpleIterationSolvingMethod(
    private val approximationFunction: (Double) -> Double,
) : SolvingMethod<Double> {

    override fun nextApproximation(current: Double): Double {
        return approximationFunction(current)
    }

    companion object {
        fun approximationFunctionFrom(
            range: ClosedRange<Double>,
            stepToCheck: Double = 1e-2,
            function: (Double) -> Double,
            derivative: (Double) -> Double = function.derivative,
        ): Pair<(Double) -> Double, (Double) -> Double> {
            val lambda = -1 / maxAbsDerivativeValue(
                range, stepToCheck, derivative
            )

            return Pair({ x: Double -> x + lambda * function(x) },  // phi
    (x)
                { x: Double -> 1 + lambda * derivative(x) }         // d(
    phi(x))/dx
            )
        }

        fun testConvergenceCondition(
            range: ClosedRange<Double>, stepToCheck: Double = 1e-2,
    derivative: (Double) -> Double
        ): Boolean {
            return maxAbsDerivativeValue(range, stepToCheck, derivative) <
     1
```

```
87          }
88
89          private fun maxAbsDerivativeValue(
90              range: ClosedRange<Double>, stepToCheck: Double = 1e-2,
    derivative: (Double) -> Double
91          ): Double {
92              var maxAbsDerivativeValue = 1.0
93
94              for (x in range step stepToCheck) {
95                  maxAbsDerivativeValue = max(maxAbsDerivativeValue, abs(
    derivative(x)))
96              }
97
98              return maxAbsDerivativeValue
99          }
100     }
101 }
102
103 class SimpleIterationSystemSolvingMethod(
104     private val approximationFunction: List<(List<Double>) -> Double>,
105 ) : SolvingMethod<List<Double>> {
106
107     init { /* ... */ }
108
109     override fun nextApproximation(current: List<Double>): List<Double> {
110         require(approximationFunction.size == current.size) {
111             "Invalid current approximation dimension. By agreement, the
    dimension of the domains of equations must be equal to $dim."
112         }
113
114         return approximationFunction(current)
115     }
116
117     companion object {
118         const val dim: Int = 2
119
120         fun approximationFunctionFrom(
121             range: List<ClosedRange<Double>>,
122         ): (List<Double>) -> Double {
123             TODO("Not yet implemented")
124         }
125
126         fun testConvergenceCondition(
127             range: List<ClosedRange<Double>>,
128             stepsToCheck: List<Double> = listOf(1e-2, 1e-2),
129             jacobianMatrix: List<List<(List<Double>) -> Double>>
130         ): Boolean {
131             require(range.size == dim) {
132                 "Invalid domain ranges dimension. By agreement, the
    dimension of the domains of equations must be equal to $dim."
133             }
134
135             var maxDerivativeValueInRange = 0.0
136
137             for (x1 in range[0] step stepsToCheck[0]) {
138                 for (x2 in range[1] step stepsToCheck[1]) {
139                     var maxDerivativeValueInMatrix = 0.0
140
141                     jacobianMatrix.map { derivative ->
142                         maxDerivativeValueInMatrix =
```

```
143                                 max(maxDerivativeValueInMatrix, derivative(
    listOf(x1, x2)).reduce { acc, d ->
144                                     acc + abs(d)
145                                 })
146                         }
147
148                         maxDerivativeValueInRange = max(
149                             maxDerivativeValueInRange,
    maxDerivativeValueInMatrix
150                         )
151                     }
152                 }
153                 return maxDerivativeValueInRange < 1
154             }
155         }
156 }
```

Листинг 1.1: Реализация на языке программирования Kolin основной логики методов решения нелинейных уравнений

4.2. Примеры и результаты работы программы

В утилите реализована возможность ввода данных через файл специального формата даннных CON — производной от JSON с отличием, что в нем все запятые заменены на точки с запятыми и все точки заменены на запятые; нужен он для удовлетворения дополнительным требованиям практика по использованию запятых в качестве разделителя при вводе и выводе данных [2]. Внешний вид пользовательского приложения можно увидеть на скриншотах

- пользовательский интерфейс при отображении решения уравнения 1.2,
- при отображении ошибки 1.3,
- при отображении решения уравнения системы уравнений 1.4,

.

5. Вывод

В ходе выполнения данной лабораторной работы углубили понимание работы методов решения нелинейных уравнений, реализовали на языке Kotlin требуемое приложение с графическим интерфейсом для их решения и самостоятельно вычислити решение уравнения.

Выяснили на практике, что разные методы по разному точны и имеют определенные сферы применения. Реализовать базовые методы решения

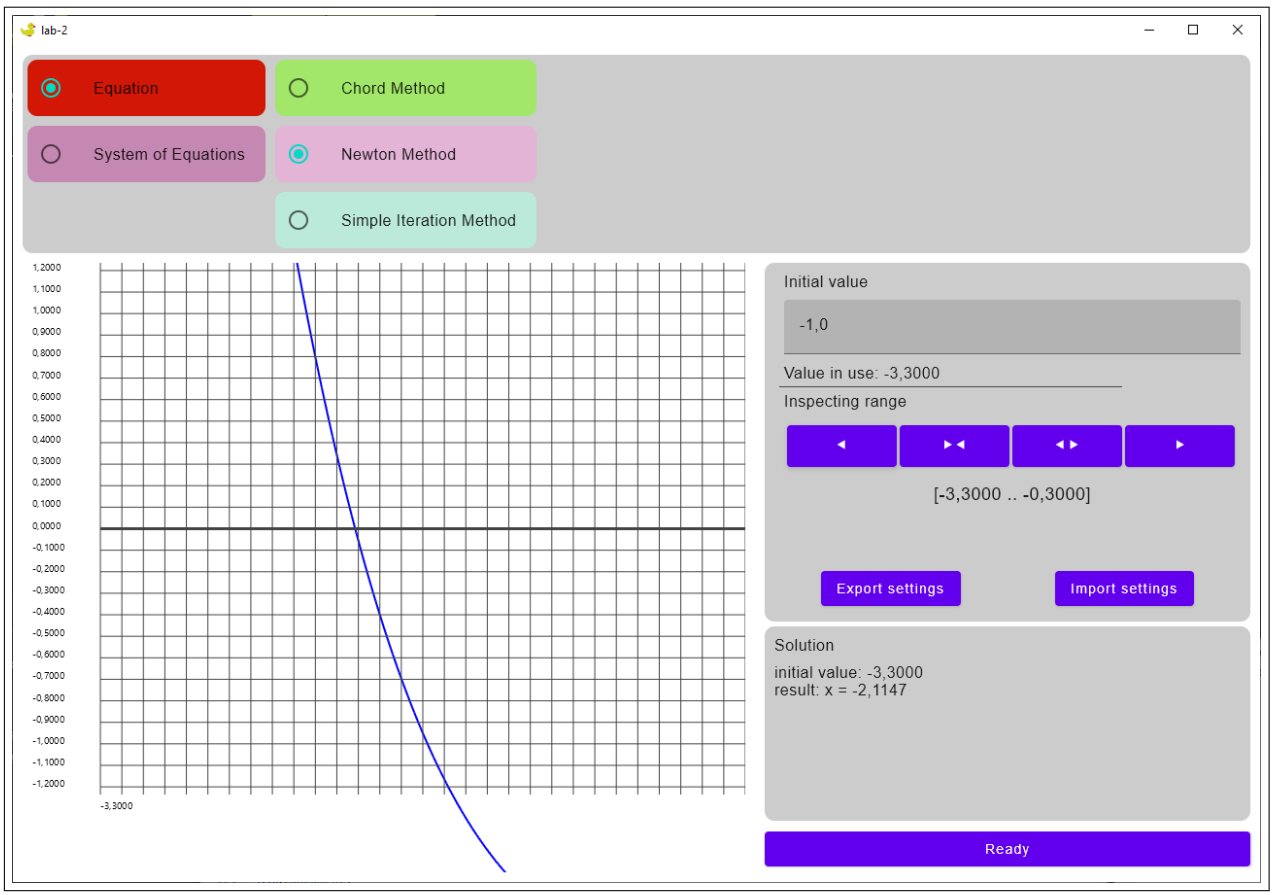

Рис. 1.2: Пользовательский интейфейс при решении уравнения

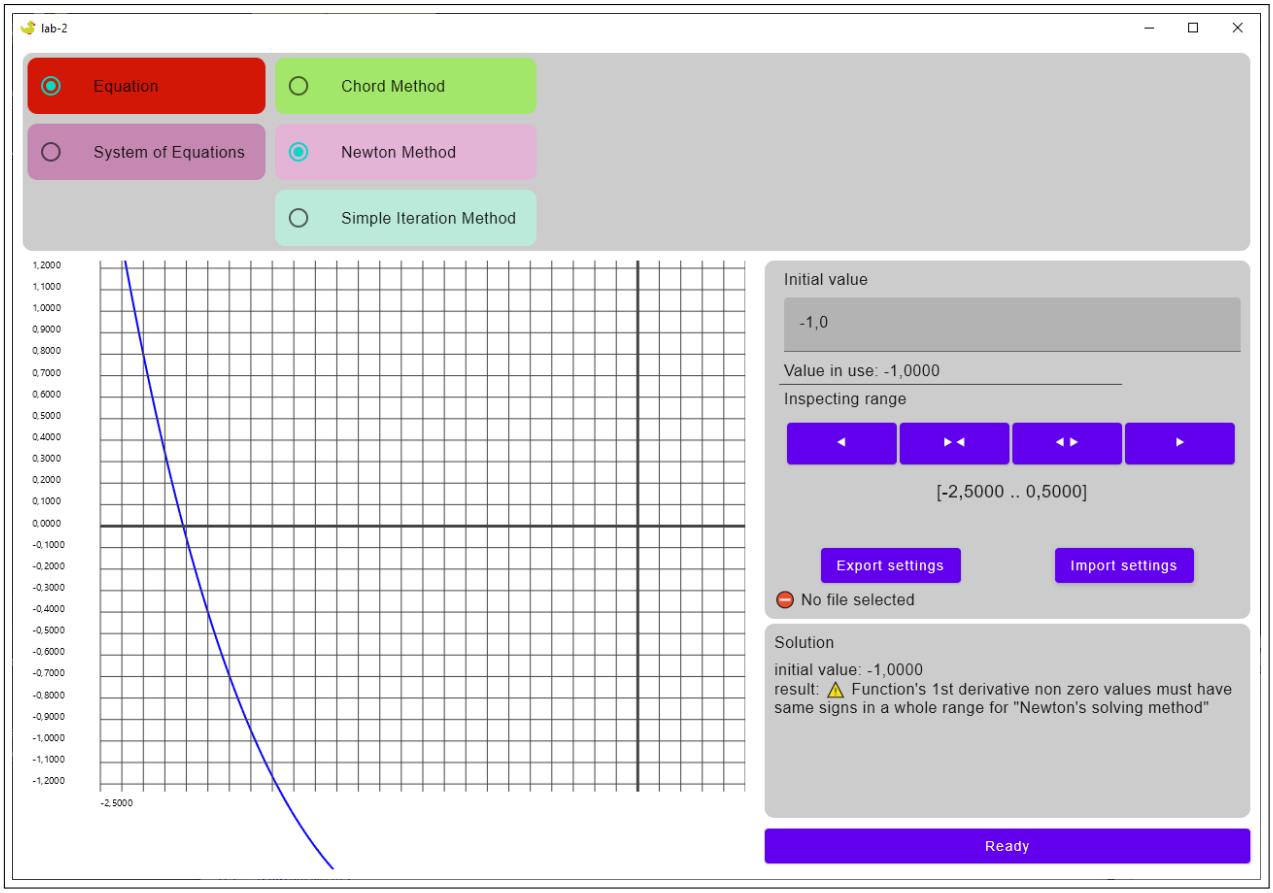

Рис. 1.3: Пользовательский интейфейс при возникновении ошибки

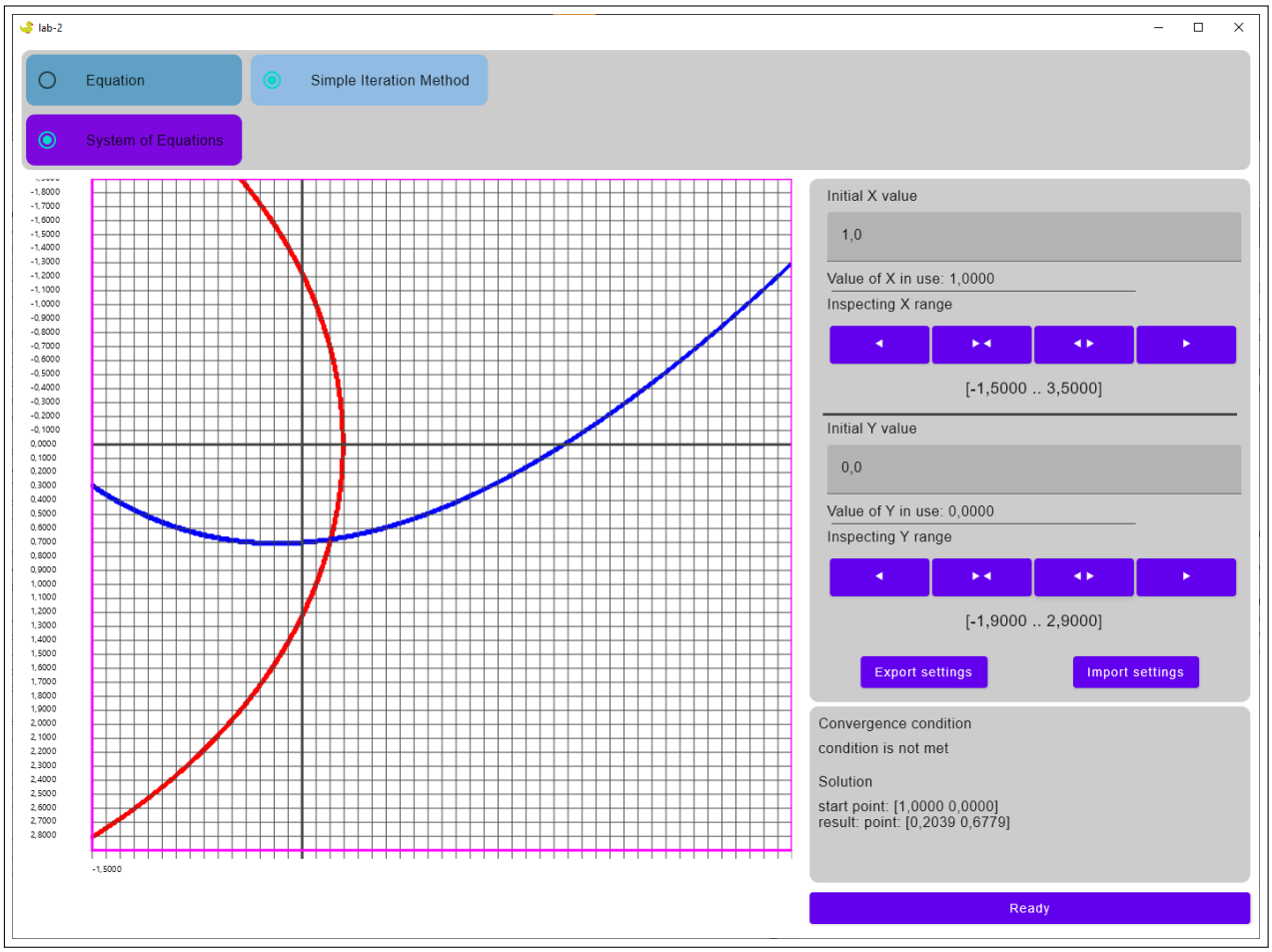

Рис. 1.4: Пользовательский интейфейс при решении системы уравнений

нелинейных уравнений оказалось довольно легко. Так же поняли, что реализовать графическое приложение гораздо сложнее, чем реализовать аналогичное консольное и что это задание не соответствует направленности курса «Вычислительная математика». Получили большой опыт в создании десктомных приложений с помощью Compose Multiplatform на языке программирования Kotlin.

Литература

[1] Ссылка на личный репозиторий GitHub: https://github.com/e1turin/itmo-comp-math/tree/main/lab-2

[2] Ссылка на страницу из Википедии, где говорится, что нет формального требования использовать десятичным разделителем в числах именно запятую («,») и приводятся ссылки на государственные стандарты. https://ru.wikipedia.org/wiki/Р¥РҳСҒСЅСЂҷСЊР,,СЌРє_СҐРөРҷРұРҳРњРҷСЂҳРњСЈ#Р¥РҳСҒСЅСЂҷСЊР,,СЌРҳ_СҐРөРҷРұРҳРњРҷСЂҳРњРҷСЂҳРњРҷ_Рў_СҒСЂСҐРөР,,РөСЖ_Рҷ_СЅРҷСЌРәРөСЖ.