**Санкт-Петербургский национальный исследовательский университет ИТМО**

# Вычислительная математика

# Лабораторная работа №2
# Численное решение нелинейных уравнений и систем

Автор:

*Ненов Владислав Александрович*

*Вариант 5*

*Группа №P32082*

Преподаватель:

*Екатерина Алексеевна Машина*

**Санкт-Петербург**
**2023**

# Цель работы

Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов. Лабораторная работа состоит из двух частей: вычислительной и программной.

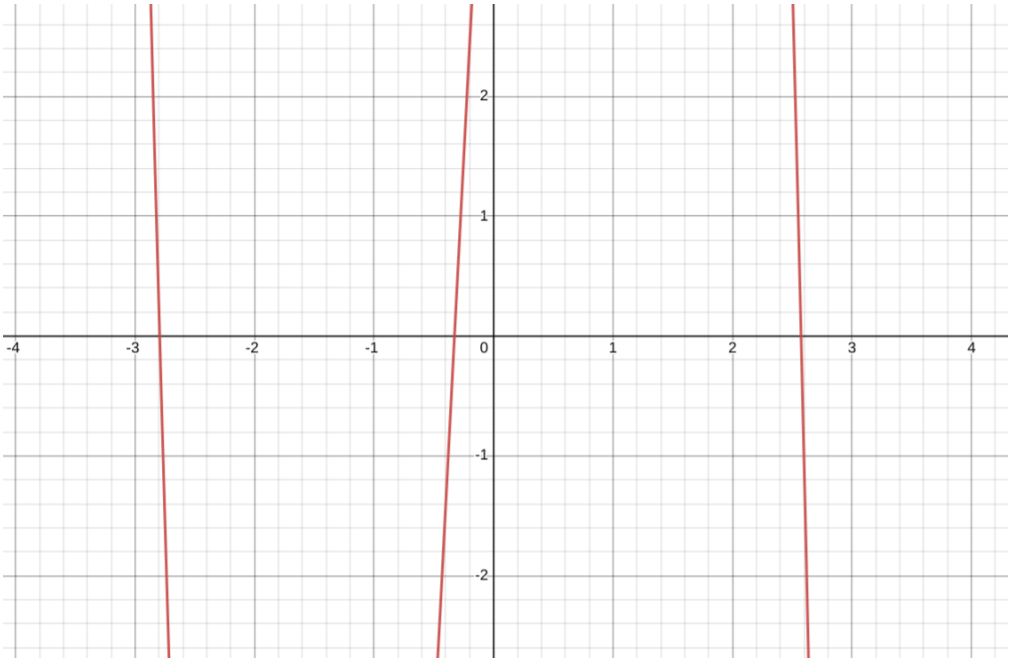
# 1) Вычислительная реализация задачи

## Задание

1. Отделить корни заданного нелинейного уравнения графически (вид уравнения представлен в табл. 6)
2. Определить интервалы изоляции корней.
3. Уточнить корни нелинейного уравнения (см. табл. 6) с точностью ε=10^-2.
4. Используемые методы для уточнения каждого из 3-х корней многочлена представлены в таблице 7.
5. Вычисления оформить в виде таблиц (1-5), в зависимости от заданного метода. Для всех значений в таблице удержать 3 знака после запятой.
5.1 Для метода половинного деления заполнить таблицу 1.
5.2 Для метода хорд заполнить таблицу 2.
5.3 Для метода Ньютона заполнить таблицу 3.
5.4 Для метода секущих заполнить таблицу 4.
5.5 Для метода простой итерации заполнить таблицу 5.
6. Заполненные таблицы отобразить в отчете.

## Выполнение

$$-2{,}7x^3 - 1{,}48x^2 + 19{,}23x + 6{,}35$$



### Левый корень

Метод простой итерации.
Возьмем интервал изоляции корня – [-3, -2]
Условия сходимости не выполняются! $|\varphi(x)| > 1$ на интервале.

### Центральный корень

Метод секущих.
Возьмем интервал изоляции корня – [-1, 0]

| *Итерация* | $x_{k-1}$ | $x_k$ | $x_{k+1}$ | $f(x_{k+1})$ | $\lvert x_k - x_{k+1}\rvert$ |
|---|---|---|---|---|---|
| 1 | -1 | 0 | -0.353 | -0,496 | 0,647 |
| 2 | 0 | -0.353 | -0,324 | 0.059 | 0.029 |
| 3 | -0.353 | -0,324 | -0,327 | 0 | 0.003 |

Ответ: x = -0.327.

### Правый корень

Метод хорд.
Возьмем интервал изоляции корня – [2, 3]

| *Итерация* | $a$ | $b$ | $x$ | $f(a)$ | $f(b)$ | $f(x)$ | $\lvert x_k - x_{k+1}\rvert$ |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 2,438 | 17,290 | -22,180 | 5,308 | 0,438 |
| 2 | 2 | 2,438 | 2,632 | 17,290 | 5,308 | -2,523 | 0,194 |
| 3 | 2,632 | 2,438 | 2,570 | -2,523 | 5,308 | 0,182 | 0,063 |
| 4 | 2,570 | 2,438 | 2,574 | 0,182 | 5,308 | -0,014 | 0,005 |

Ответ: x = 2.574.

# 2) Программная реализация задачи

## Задание

**Для нелинейных уравнений:**
1. Все численные методы (см. табл. 8) должны быть реализованы в виде отдельных подпрограмм/методов/классов.
2. Пользователь выбирает уравнение, корень/корни которого требуется вычислить (3-5 функций, в том числе и трансцендентные), из тех, которые предлагает программа.

3. Предусмотреть ввод исходных данных (границы интервала/начальное приближение к корню и погрешность вычисления) из файла или с клавиатуры по выбору конечного пользователя.
4. Выполнить верификацию исходных данных. Необходимо анализировать наличие корня на введенном интервале. Если на интервале несколько корней или они отсутствуют – выдавать соответствующее сообщение. Программа должна реагировать на некорректные введенные данные.
5. Для методов, требующих начальное приближение к корню (методы Ньютона, секущих, хорд с фиксированным концом), выбор начального приближения (a или b) вычислять в программе.
6. Для метода простой итерации проверять достаточное условие сходимости метода на введенном интервале.
7. Предусмотреть вывод результатов (найденный корень уравнения, значение функции в корне, число итераций) в файл или на экран по выбору конечного пользователя.
8. Организовать вывод графика функции, график должен полностью отображать весь исследуемый интервал (с запасом).

**Для систем нелинейных уравнений:**
1. Пользователь выбирает предлагаемые программой системы двух нелинейных уравнений (2-3 системы).
2. Организовать вывод графика функций.
3. Начальные приближения ввести с клавиатуры.
4. Для метода простой итерации проверить достаточное условие сходимости.
5. Организовать вывод вектора неизвестных: $x1$ , $x2$.
6. Организовать вывод количества итераций, за которое было найдено решение.
7. Организовать вывод вектора погрешностей.
8. Проверить правильность решения системы нелинейных уравнений.

## Выполнение

На языке Java

**Метод Простых итераций для систем**

```
public class SimpleSystemIterMethodForSystems extends SystemSolveMethod{

    public SimpleSystemIterMethodForSystems(double accuracy, MultiArgFunction[]
function) {
        super(accuracy, function);
    }

    @Override
    protected SystemEquationResult[] calculate(double[] initApprox) {
        double[] pastResult = initApprox.clone();
        double[] xData = new double[functions.length];
        double[] diffs = new double[functions.length];
        boolean run = true;
        int iterations = 0;
        while (run) {
            for (int i = 0; i < functions.length; i++) {
                xData[i] = functions[i].calculate(pastResult);
                double derSum = 0;
                for (int j = 0; j < functions.length; j++) {
                    derSum += Math.abs(functions[i].derivative(pastResult, j));
                }
                if (derSum >= 1)
                    return new SystemEquationResult[] { new SystemEquationResult("Метод не
сходится!") };
            }

            run = false;
            for (int i = 0; i < functions.length; i++) {
                diffs[i] = Math.abs(xData[i]-pastResult[i]);
                if (diffs[i] > accuracy)
                    run = true;
            }
            iterations++;
            pastResult = xData.clone();
        }
        SystemEquationResult[] results = new SystemEquationResult[xData.length];
        for (int i = 0; i < results.length; i++) {
            results[i] = new SystemEquationResult(xData[i], iterations,  diffs[i]);
        }
        return results;
    }
}
```

**Метод хорд**

```
public class ChordMethod extends EqSolveMethod {

    public ChordMethod(double accuracy, Function function) {
        super(accuracy, function);
    }

    @Override
    public EquationResult calculate(double a, double b) {
        double prX;
        double x = a;
        int iters = 0;
        do {
            prX = x;
            x = (a*function.calculate(b) - b* function.calculate(a)) / (function.calculate(b)-
function.calculate(a));
            iters++;
            if (function.calculate(a) * function.calculate(x) < 0)
                a = x;
            else
                b = x;
        } while (Math.abs(x - prX) > accuracy && Math.abs(a-b) > accuracy &&
Math.abs(function.calculate(x)) > accuracy);
        return new EquationResult(x, iters, function);
    }
}
```

**Метод секущих**

```
public class SecantMethod extends NewtonMethod{
    public SecantMethod(double accuracy, Function function) {
        super(accuracy, function);
    }

    @Override
    public EquationResult calculate(double a, double b) {
        double prX, x;
        if (function.calculate(a)*function.secondDerivative(a) > 0) {
            prX = a;
            x = b;
        }
        else {
            prX = b;
            x = a;
        }
        int iters = 0;
        do {
            double tmp = x;
            x = x - (x-prX)/(function.calculate(x)-function.calculate(prX)) * function.calculate(x);
            prX = tmp;
            iters++;
        } while (Math.abs(x-prX) > accuracy && Math.abs(function.calculate(x)) > accuracy);
        return new EquationResult(x, iters, function);
    }
}
```