**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №5**

**по дисциплине «Вычислительная Математика»**

**Тема: Метод Ньютона**

| | | |
|---|---|---|
| Студент гр. 8381 | ___________________ | СергеевА.Д. |
| Преподаватель | ___________________ | Щеголева Н.В. |

Санкт-Петербург

2019

**Цель работы.**

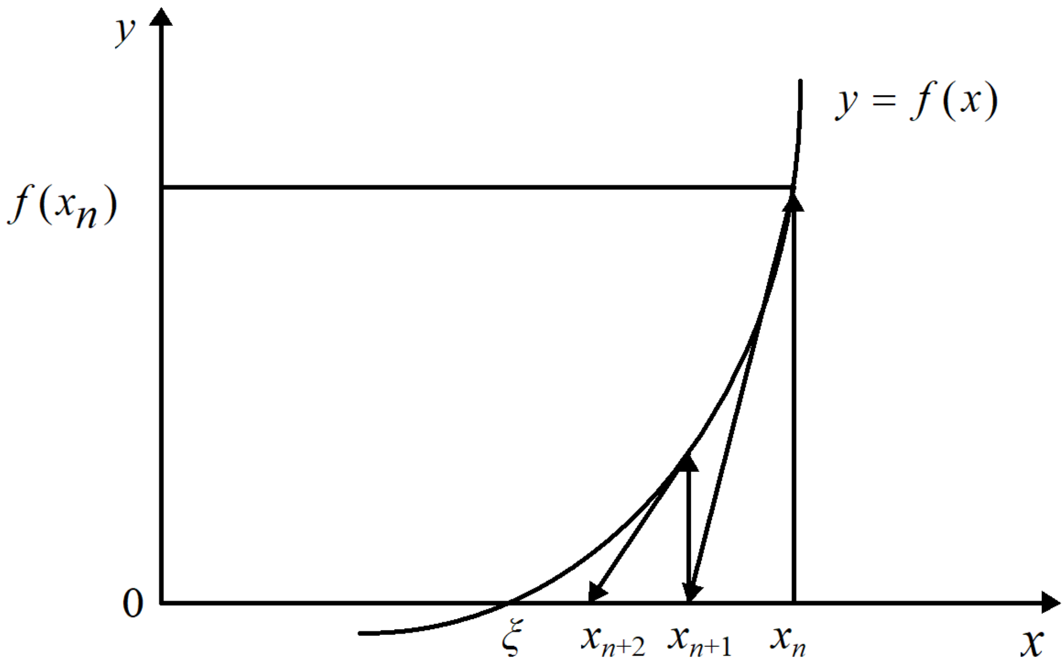
Формирование практических навыков нахождения корней алгебраических и трансцендентных уравнений методом Ньютона.

**Краткое изложение основных теоретических понятий.**

В случае, когда известно хорошее начальное приближение решения уравнения $f(x)=0$, эффективным методом повышения точности является метод Ньютона (касательных). Он состоит в построении итерационной последовательности $x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$, сходящейся к корню уравнения $f(x)=0$. Достаточные условия сходимости метода формулируются теоремой.

***Теорема***. *Пусть* $f(x)$ *определена и дважды дифференцируема на* $[a,b]$*, на котором функция* $f(x)$ *меняет знак, а производные* $f'(x)$, $f''(x)$ *сохраняют знак. Тогда, исходя из начального приближения* $x_0\in[a,b]$ *, удовлетворяющего неравенству* $f(x_0)f''(x_0)>0$ *, можно построить последовательность* $x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}, n=1,2,3,4,\dots$ *, сходящуюся к единственному на* $[a,b]$ *решению* $\xi$ *уравнения* $f(x)=0$ .

Метод Ньютона допускает простую геометрическую интерпретацию, представленную на рисунке. Если через точку с координатами $(x_n;f(x_n))$ провести касательную, то абсцисса точки пересечения этой касательной с осью $OX$ будет очередным приближением $x_n+1$ корня уравнения $f(x)=0$ .




Для оценки погрешности n-го приближения корня предлагается пользоваться неравенством: $|\xi - x_n| \leqslant \frac{M_2}{2m_1}|x_n - x_{n-1}|^2$ , где $M_2$ – наибольшее значение модуля второй производной $|f''(x)|$ на отрезке $[a,b]$ ; $m_1$ – наименьшее значение модуля первой производной $|f'(x)|$ на отрезке $[a,b]$ . Таким образом, если $|x_n - x_{n-1}| < \varepsilon$ , то $|\xi - x_n| \leqslant \frac{M_2 \varepsilon^2}{2m_1}$ . Это означает, что при хорошем начальном приближении процесс сходится очень быстро (имеет место квадратическая сходимость). Из указанного следует, что при необходимости нахождения корня с точностью ε итерационный процесс можно прекращать, когда $|x_n - x_{n-1}| < \varepsilon_0$ , где $\varepsilon_0 = \sqrt{\frac{2m_1\varepsilon}{M_2}}$ .

Рассмотрим один шаг итераций. Если на $(n-1)$ -м шаге очередное приближение $x_n - 1$ не удовлетворяет условию окончания процесса, то вычисляются величины $f(x_{n-1})$ , $f'(x_{n-1})$ и следующие приближение корня $x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$ . При выполнении условия остановки, описанного выше, величина $x_n$ принимается за приближенное значение корня ξ, вычисленное с точностью ε.

**Постановка задачи с кратким описанием порядка выполнения работы.**

Используя метод Ньютона, требуется найти корень функции $y(x)=arctg(x)-\ln(x)$ с заданной точностью ε и проверить этот метод на скорость сходимости и обусловленность.

Для выполнения работы было принято решение использовать язык программирования *java*. Был модифицирован метод *NEWTON*, предоставленный на сайте *moevm*, т. к. из-за особенностей выбранного языка перестало быть возможным получение информации о количестве итераций метода, а также появилась необходимость расчёта корня с моделированием помех во входных данных и записи результата в файл для удобства последующей обработки. В его сигнатуру были добавлены *boolean isWrong*, *PrintStream PS* и *double delta*. Внутри функции был добавлен выбор между вызовами метода *f(x)* и *round(f(x), delta)* в зависимости от параметра *isWrong*, который означает, производится вычисление с моделированием помех или без него. Информация о количестве итераций печатается в файл при помощи потока *PrintStream*.

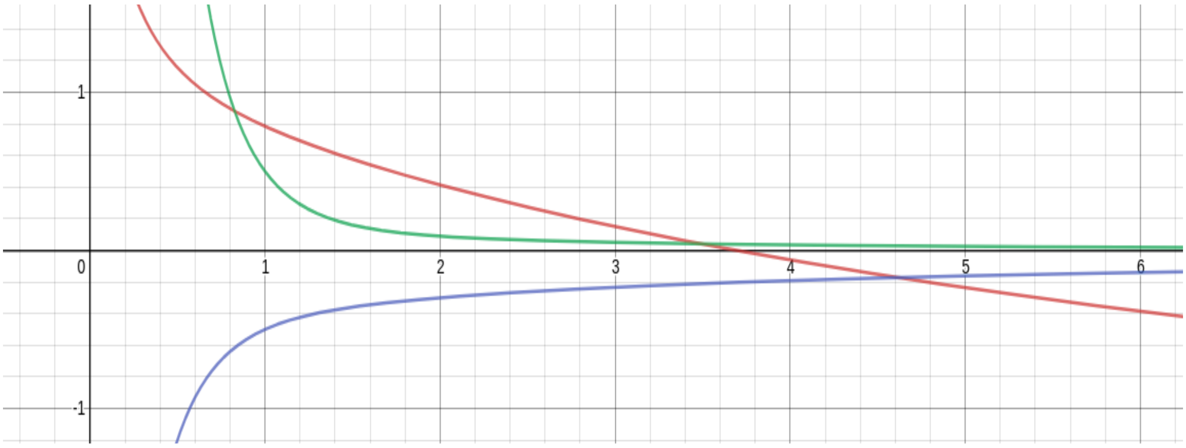
Также мной были написаны методы *collectVariabiles* и *executeResearch*, первый из которых собирает данные об условиях задачи, введённые пользователем или содержащиеся в файле *input.txt*, а второй производит измерения и расчёты для этих условий с разными параметрами, обращаясь к методу *horda*, и записывает их в файл *out.txt*. Исходный код программы содержится в Приложении А.

**Графическое или аналитическое решение уравнения.**

При помощи метода бисекции, описанного ранее, взятого с высокой точностью (ε < $10^{-6}$), был вычислен корень функции $f(x)=0$ при $x=3.6925856854554855\ldots$ .

Затем уравнение $y(x)=arctg(x)-\ln(x)$ я также решил графически, используя онлайн-калькулятор *desmos*.

Таким образом был отделён корень уравнения $f(x)=0$ , т. е. был найден отрезок $[a,b]$ , на котором функция $f(x)$ удовлетворяет условиям теоремы Больцано-Коши.




Также на отрезке $[2,5]$ заданная функция дифференцируема дважды и удовлетворяет условиям (непрерывна и монотонна) применения метода Ньютона (производные $f'(x)$ , $f''(x)$ сохраняют знак).

Была выбрана точка $x_0 \in [a,b]$ , например, точкой $x_0$ станет 3, для которой $f(x_0)f''(x_0)>0$ , т. к. $f(3)>0.15$ и $f''(3)>0.051$ .

Были оценены $m_1=\min\limits_{x\in[a,b]}|f'(x)|$ , $m_1=f'(5)=0.162$ и $M_2=\max\limits_{x\in[a,b]}|f''(x)|$ , $M_2=f''(2)=0.09$ .

В процессе работы программы автоматически вычисляется $\varepsilon_0$ исходя из значения ε по описанной выше формуле.

**Необходимые графики и таблицы с краткими выводами.**

По результатам работы программы была построена таблица. В ней представлены зависимость числа итераций функции *NEWTON* от заданной точности ε, а также от выбранного отрезка, на котором производятся вычисления. Также в таблице представлены сведения об устойчивости метода к ошибкам входных данных, которые были смоделированы с использованием функции *round*, округляющей значения функции с заданной точностью δ.

Для проверки обусловленности задачи было найдено число обусловленности $V_\delta = \frac{|x-x'|}{\delta}$, где $x$ — найденный ранее корень уравнения, $x = 3.6925856854554855\ldots$, а $x'$ — корень, найденный с использованием моделирования погрешностей во входных данных. Будем считать, что задача плохо обусловлена, когда число обусловленности в пять или более раз больше, чем $\frac{\varepsilon}{\delta}$.

| Корень | $x_0$ | ε | Итерации | δ | Корень с помехами | $\frac{\varepsilon}{\delta}$ | $V_\delta$ |
|---|---|---|---|---|---|---|---|
| 3.68305 | 2 | 0.1 | 2 | 0.0001 | 3.68316 | 1000 | 94.28768, хорошо |
| 3.69258 | 2 | 0.01 | 3 | 0.0001 | 3.69252 | 100 | 0.64549, хорошо |
| 3.69258 | 2 | 0.001 | 3 | 0.0001 | 3.69252 | 10 | 0.64549, хорошо |
| 3.69258 | 2 | 0.0001 | 3 | 0.0001 | 3.69252 | 1 | 0.64549, хорошо |
| 3.69259 | 2 | 0.00001 | 4 | 0.0001 | 3.69252 | 0.1 | 0.64549, плохо |

**Скорость схождения метода, сравнение с методом бисекции и хорд.**

Из полученных данных видно, что с уменьшением коэффициента ε, число итераций и точность корня растёт, а также, что при увеличении δ уменьшается точность и обусловленность выходных данных при низкой требуемой точности ε. Аналогично из данной таблицы подтверждается вывод о том, что количество итераций зависит от требуемой точности и растёт квадратично. Таким образом, теоретические результаты совпадают с экспериментальными данными.

Практические исследования показали, что для выбранной точности метод Ньютона требует в среднем в 4 раза меньше итераций, чем метод бисекций и в 2 раза меньше, чем метод хорд. Это соответствует теоретическим сведениям о том, что метод Ньютона быстрее обоих этих методов.

Если сравнить обусловленность этих методов, окажется, что, исходя из экспериментальных данных, метод Ньютона обусловлен примерно так же хорошо, как метод бисекции и в ряде случаев лучше, чем метод хорд.

**Общий вывод по проделанной работе.**

В результате проделанной работы, был сделан вывод о том, что число итераций метода Ньютона возрастает с увеличением точности выходных данных, а обусловленность этого метода прямо пропорциональна точности исходных данных и обработано пропорциональна точности вычисления корня.

**Приложение А: исходный код программы.**

**Файл Lab3.java:**

```
package classes;

import java.io.*;

/**
 * Function number 24 is:
 *      f(x) = arctg(x) - ln(x)
 *      f'(x) = 1/(x^2 + 1) - 1/x
 *
 */

public class Lab3 {
    private static final String OVERPATH = "/home/alex/IdeaProjects/labs";

    public static void main(String [] args) {
        try {
            collectVariables(false);
        } catch (FileNotFoundException fnfe) {
            System.out.println("NO INPUT FILE FOUND!");
            fnfe.printStackTrace();
        } catch (Exception e) {
            System.out.println(":/");
            e.printStackTrace();
        }
    }

    public static void collectVariables(boolean isUI) throws Exception {
         BufferedReader in = new BufferedReader(isUI ? new InputStreamReader(System.in) :
new FileReader(OVERPATH + "/buffer/input.txt"));

        int scale = 1;
        double epsilon = 0.000001, trueAnswer = 0;

        if (isUI) {
            System.out.println("The function is: f(x) = arctg(x) - ln(x)");
            System.out.println("It has the only root\n");
            System.out.println("Choose the Epsilon between 0.1 and 0.000001:");

            try {
                String input1 = in.readLine();
                epsilon = Double.parseDouble(input1);
            } catch (NumberFormatException e) {
                System.out.println("You've written not a number");
                return;
            } catch (IOException e) {
                System.out.println("Some error occurs");
                return;
            }
```

```
            if ((epsilon < 0.000001) || (epsilon > 0.1)) {
                System.out.println("Epsilon is not between 0.1 and 0.000001");
                return;
            }

            System.out.println("\nChoose the scale of research from 1 to 3:");

            try {
                String input2 = in.readLine();
                scale = Integer.parseInt(input2);
            } catch (NumberFormatException e) {
                System.out.println("You've written not a number");
                return;
            } catch (IOException e) {
                System.out.println("Some error occurs");
                return;
            }
            if ((scale < 1) || (scale > 3)) {
                System.out.println("Epsilon is not between 1 and 3");
                return;
            }

            System.out.println("\nType the correct answer for wrong answer checking:");

            try {
                String input3 = in.readLine();
                trueAnswer = Double.parseDouble(input3);
            } catch (NumberFormatException e) {
                System.out.println("You've written not a number");
                return;
            } catch (IOException e) {
                System.out.println("Some error occurs");
                return;
            }
        } else {
                PrintStream out = new PrintStream(new FileOutputStream(OVERPATH +
"/buffer/out.txt"));
            trueAnswer = Double.parseDouble(in.readLine());
            for (int i = 0; i < 5; i++) {
                String input = in.readLine();
                int delimPos = input.indexOf(':');
                scale = Integer.parseInt(input.substring(0, delimPos));
                epsilon = Double.parseDouble(input.substring(delimPos + 1));
                executeResearch(scale, epsilon, trueAnswer, 0.0001, out);
            }
            out.flush();
            out.close();
            return;
        }

        executeResearch(scale, epsilon, trueAnswer, 0.0001, System.out);
```

```
        in.close();
    }

    public static void executeResearch(int scale, double epsilon, double trueAnswer, double delta, PrintStream PS) {
        PS.println("\nINITIALIZED RESEARCH FOR:\nScale = " + scale + "\nEpsilon = " + epsilon + "\nTrue answer = " + trueAnswer + "\nDelta = " + delta + "\n");
        PS.println("Research in progress...");

        double answer;
        try {
            answer = Support.NEWTON(scale, epsilon, PS, false, delta);
        } catch (Support.WrongParameterException e) {
            PS.println("Research terminated with following exception.");
            return;
        }
        PS.println("Research ended! The answer is: " + answer + "\n");

        double wrongAnswer;
        try {
            wrongAnswer = Support.NEWTON(scale, epsilon, PS, true, delta);
        } catch (Support.WrongParameterException e) {
            PS.println("Research terminated with following exception.");
            return;
        }
        PS.println("The wrong answer is: " + wrongAnswer + "\n");

        double ED = epsilon / delta;
        double Vdel = Math.abs(trueAnswer - wrongAnswer) / delta;
        PS.println("Epsilon/Delta is: " + ED);
        PS.println("V(delta) is: " + Vdel);
        PS.println("Decision: " + ((Vdel < ED * 5) ? ("GOOD") : ("BAD")) + "\n");
        PS.println("\n\n\n");
    }
}
```

### Файл Support.java:

```
package classes;

import java.io.PrintStream;

public class Support {
    public static final double MINIMAL_DELTA = 1E-9;

    public static class WrongParameterException extends Exception {
        private WrongParameterException(String message) {
            super(message);
        }
    }
```

```
public static double round(double X, double Delta) throws WrongParameterException {
    if (Delta <= MINIMAL_DELTA) {
        throw new WrongParameterException("Точность округления слишком мала: " +
Delta + "\n");
    }

    if (X > 0.0) {
        return Delta * (long) (X / Delta + 0.5);
    } else {
        return Delta * (long) (X / Delta - 0.5);
    }
}


public static double f(double x) {
    return Math.atan(x) - Math.log(x);
}

public static double f1(double x) {
    return 1/(x*x + 1) - 1/x;
}

public static double NEWTON(double X, double Eps, PrintStream PS, boolean isWrong,
double delta) throws WrongParameterException {
    double Y, Y1, DX, Eps0;
    int N = 0;

    double m1 = 0.162,  // наименьшее значение модуля 1-ой производной
           M2 = 0.09;  // наибольшее значение модуля 2-ой производной

    Eps0 = Math.sqrt(2 * m1 * Eps / M2);

    do {
        Y = isWrong ? round(f(X), delta) : f(X);

        if (Y == 0.0) {
            return X;
        }

        Y1 = isWrong ? round(f1(X), delta) : f1(X);

        if (Y1 == 0.0) {
            throw new WrongParameterException("Производная обратилась в ноль");
        }

        DX = Y / Y1;
        X -= DX;
        N++;
    } while (Math.abs(DX) >= Eps0);

    PS.println("Iterations count: " + N);
```

```
        return X;
    }
}
```