**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №4**

**по дисциплине «Вычислительная Математика»**

**Тема: Метод хорд**

Студент гр. 8381 _____ СергеевА.Д.

Преподаватель _____ Щеголева Н.В.

Санкт-Петербург

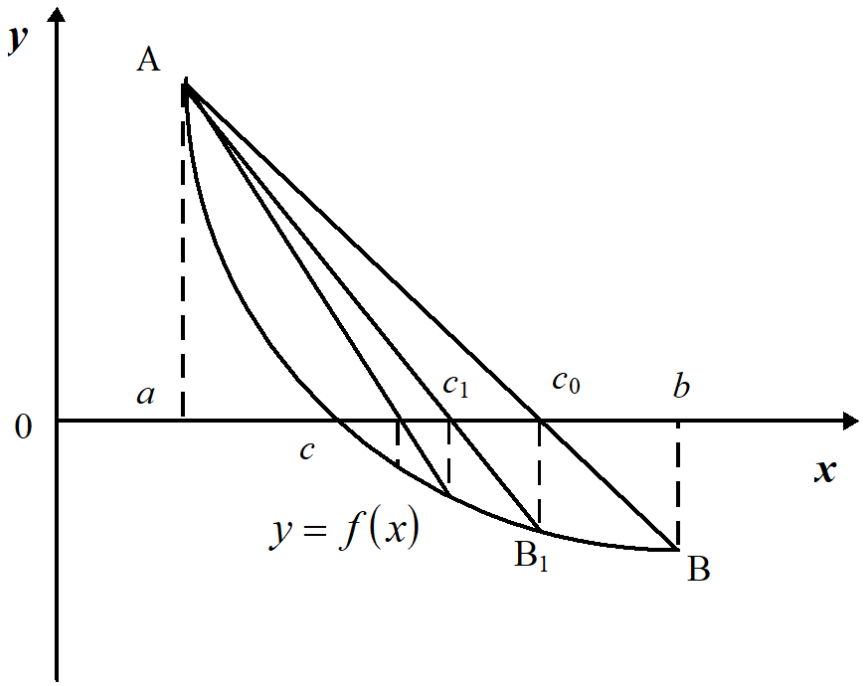
2019

**Цель работы.**

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

**Краткое изложение основных теоретических понятий.**

Пусть найден отрезок $[a,b]$, на котором функция $f(x)$ меняет знак. Предположим, $f(a)>0$, а $f(b)<0$. В методе хорд в качестве приближений к корню берутся значения $c_0, c_1, \ldots$ как на рисунке:




Находится уравнение хорды $AB$: $\frac{y-f(a)}{f(b)-f(a)}=\frac{x-a}{b-a}$.

Для точки пересечения $AB$ с осью абсцисс $(y=0, x=c_0)$ получится уравнение: $c_0=a-\frac{(b-a)\cdot f(a)}{f(b)-f(a)}$.

Далее сравниваются знаки величин $f(a)$ и $f(c_0)$. Для случая, показанного на рисунке, корень находится в интервале $(a,c_0)$, так как $f(a)>0$ и $f(c_0)<0$. В

следующей итерации находится следующее приближение — точка $c_1$, являющаяся пересечением хорды $AB_1$ с осью абсцисс, где точка $B_1=f(c_0)$.

Алгоритм выполняется до тех пор, пока значение $f(c_n)$ не станет по модулю меньше заданного числа ε.

Метод хорд в ряде случаев даёт более быструю сходимость итерационного процесса, причём успех его применения, как и метода бисекции, гарантирован.

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

Используя метод хорд, 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 *horda*, 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*, *double delta* и *PrintStream PS*. 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 выводится в поток *PS*.

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 в предыдущей лабораторной работе, взятого с очень большой 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. Заданная функция удовлетворяет условиям (непрерывна и монотонна) применения метода хорд, как на отрезке $[3,4]$, так и на отрезке $[1,6]$.

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

По результатам работы программы была построена таблица. В ней представлены зависимость числа итераций функции *horda* 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 | Отрезок | ε | Итерации | δ | Корень с помехами | $\frac{\varepsilon}{\delta}$ | $V_\delta$ |
|---|---|---|---|---|---|---|---|
| 3.71326 | [3;4] | 0.1 | 1 | 0.0001 | 3.71313 | 1000 | 205.48501, хорошо |
| 3.71326 | [3;4] | 0.01 | 1 | 0.0001 | 3.71313 | 100 | 205.48501, хорошо |
| 3.69399 | [3;4] | 0.001 | 2 | 0.0001 | 3.69376 | 10 | 11.74869, хорошо |
| 3.69268 | [3;4] | 0.0001 | 3 | 0.0001 | 3.69238 | 1 | 2.06819, хорошо |
| 3.69259 | [3;4] | 0.00001 | 4 | 0.0001 | 3.69238 | 0.1 | 2.06818, плохо |
| 3.88957 | [1;6] | 0.1 | 2 | 0.0001 | 3.88964 | 1000 | 1970.52440, хорошо |
| 3.71083 | [1;6] | 0.01 | 4 | 0.0001 | 3.71085 | 100 | 182.65584, хорошо |
| 3.69429 | [1;6] | 0.001 | 6 | 0.0001 | 3.69429 | 10 | 17.81088, хорошо |
| 3.69275 | [1;6] | 0.0001 | 8 | 0.0001 | 3.69265 | 1 | 0.66722, хорошо |
| 3.69258 | [1;6] | 0.00001 | 9 | 0.0001 | 3.69265 | 0.1 | 0.66722, плохо |

**Скорость схождения метода и сравнение его с предыдущим (методом бисекции).**

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 линейно. Таким образом, теоретические результаты совпадают с экспериментальными данными.

Практические исследования показали, что метод хорд требует в среднем в 2 раза меньше итераций, чем метод бисекций. Это соответствует теоретическим сведениям о том, что метод хорд может быть быстрее метода бисекций.

Если сравнить обусловленность этих методов, окажется, что, исходя из экспериментальных данных, метод бисекции в ряде случаев лучше обусловлен, чем метод хорд.

**Общий вывод по проделанной работе.**

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

От длины отрезка, к которому применяется метод хорд, его точность практически не зависит.

## 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";
    private static final double [] HIGHEST_LIMIT = {4.0, 5.0, 6.0};
    private static final double [] LOWEST_LIMIT  = {3.0, 2.0, 1.0};

    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 < 10; 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.horda(LOWEST_LIMIT[scale - 1], HIGHEST_LIMIT[scale - 1], epsilon, false, delta, PS);
            } 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.horda(LOWEST_LIMIT[scale - 1], HIGHEST_LIMIT[scale - 1], epsilon, true, delta, PS);
            } 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 * 2) ? ("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 horda(double Left, double Right, double Eps, boolean isWrong,
double delta, PrintStream PS) throws WrongParameterException {
        double fLeft = isWrong ? round(f(Left), delta) : f(Left);
        double fRight = isWrong ? round(f(Right), delta) : f(Right);
        double X, Y;

        if (fLeft * fRight > 0.0) {
            throw new WrongParameterException("Знаки границ интервала не различны: " +
fLeft + " " + fRight + "\n");
        }

        if (Eps <= 0.0) {
            throw new WrongParameterException("Заданная точность меньше нуля: " + Eps +
"\n");
        }

        int N = 0;

        if (fLeft == 0.0) {
            PS.println("Number of iterations: " + N);
            return Left;
        }

        if (fRight == 0.0) {
            PS.println("Number of iterations: " + N);
            return Right;
        }

        do {
```

```
            X = Left - (Right - Left) * fLeft / (fRight - fLeft);
            Y = isWrong ? round(f(X), delta) : f(X);

            if (Y == 0.0) {
               PS.println("Number of iterations: " + N);
               return X;
            }

            if (Y * fLeft < 0.0) {
               Right = X;
               fRight = Y;
            } else {
               Left = X;
               fLeft = Y;
            }

            N++;
         } while (Math.abs(Y) >= Eps);

         PS.println("Number of iterations: " + N);
         return X;
      }
   }
```