Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

ФКТиУ, кафедра Вычислительной техники

Лабораторная работа №2

по дисциплине
«Вычислительная математика»
Вариант: 1аб

Выполнил: Студент группы Р3233
Сабитов Д.Т.

Преподаватель:
Перл О. В.

Санкт-Петербург
2022 г.

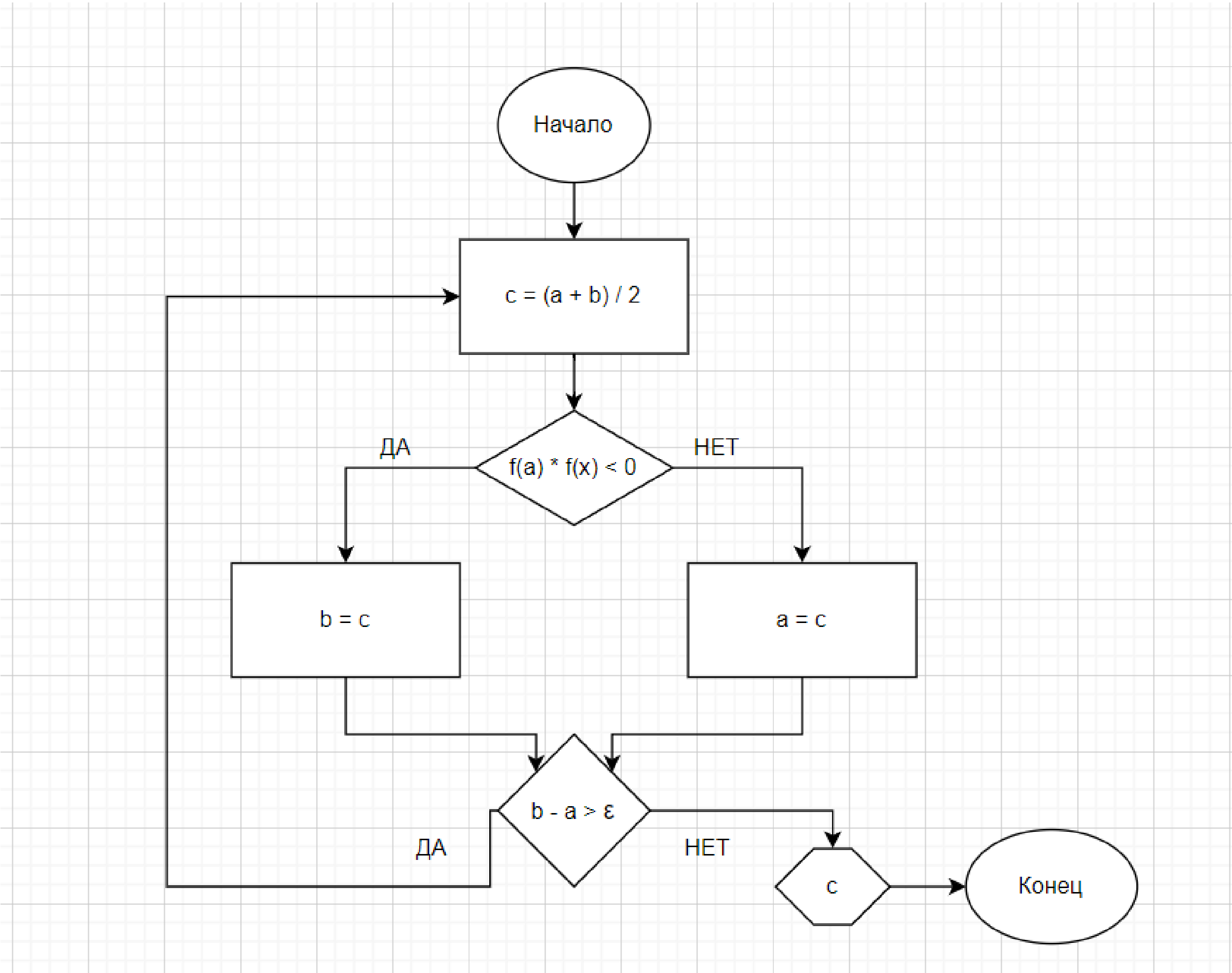
Метод деления пополам

Описание метода

Метод применим для численного решения уравнения f(x) = 0 для вещественной переменной x, где f-непрерывная функция, определенная на интервале [a, b] и где f(a) и f(b) имеют противоположные знаки. Метод заключается в сужении промежутка, на котором функция имеет разный знак. Начальный отрезок выбираем так, чтобы произведение значений функции на концах было меньше 0: f(a) * f(b) < 0.

В каждой итерации отрезок делится на 2: (a + b) / 2. Для дальнейшего деления выбираем половину так: если f(a) * f(x) < 0, то b = x. Если f(a) * (x) > 0, то это ненужная нам половина, значит, a = x. Если f(a) * f(x) = 0, то x – искомый корень. Вычисления завершаются при достижении заданной точности.

Блок-схема



Листинг метода

```
public static double bisectionMethod(double a, double b, double eps, int
equation) throws NotExistingEquationException {
    checkIntervalArguments(a, b, equation);
    double c;
    while (b - a > eps){
        c = (a + b) / 2;
        if (getEquation(equation, a) * getEquation(equation, c) < 0){
```

```
            b = c;
        }else if (getEquation(equation, a) * getEquation(equation, c) > 0){
            a = c;
        }else{
            return a;
        }
    }
    return a;
}
```

Расчетные формулы метода

$$c = \frac{b^{n-1}+a^{n-1}}{2}, \text{ где n-номер итерации}$$

$$\text{если } f(a^{n-1})*f(c)\leq 0, \text{ то } b^n=c,\ a^n=a^{n-1}$$

$$\text{если } f(b^{n-1})*f(c)\leq 0, \text{ то } a^n=c,\ b^n=b^{n-1}$$

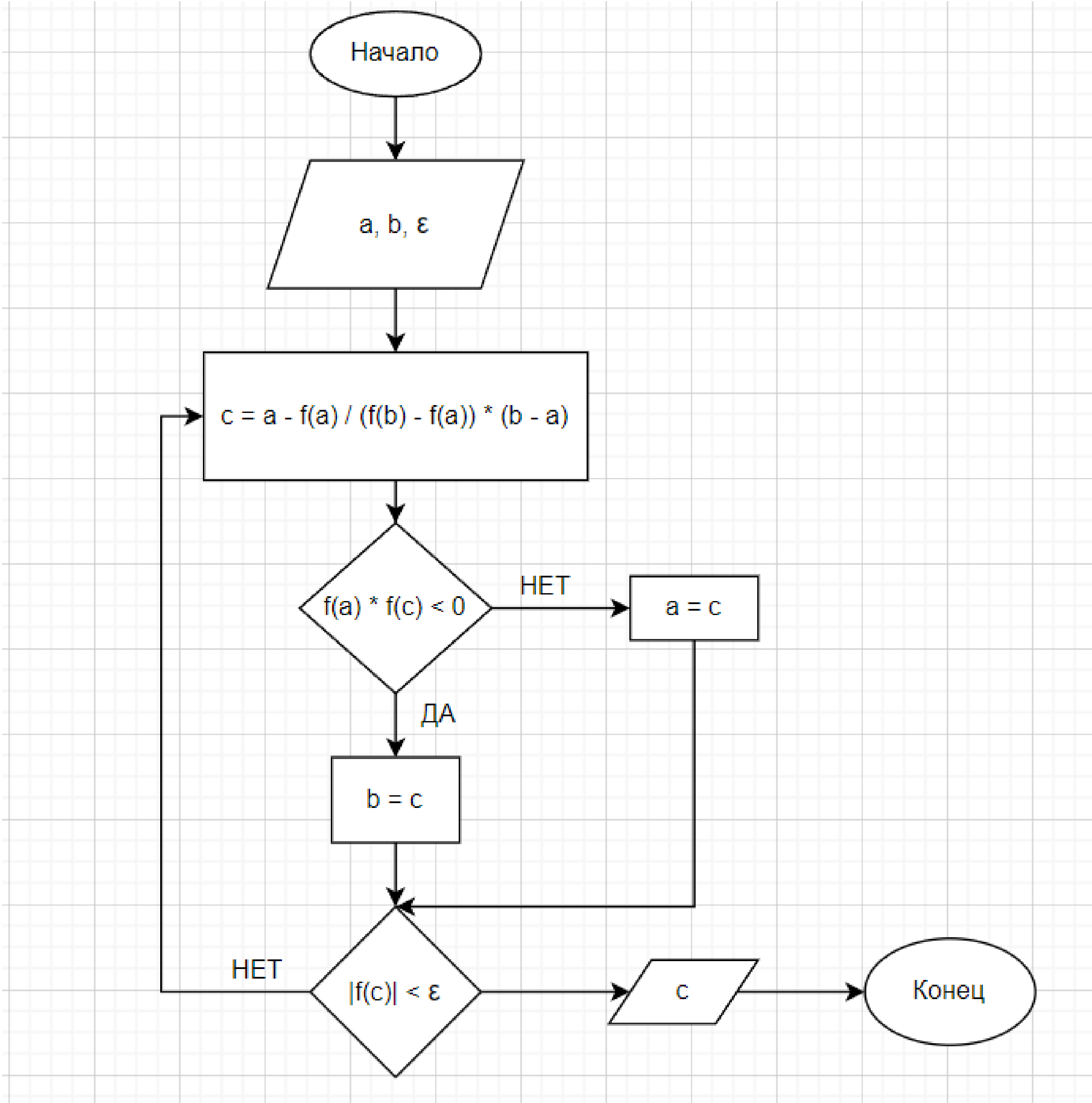
Метод хорд:

Описание метода

Суть метода хорд состоит в разбиении отрезка [a; b] (при условии f(a) * f(b) < 0) на два отрезка с помощью хорды и выборе нового отрезка от точки пересечения хорды с осью абсцисс до неподвижной точки, на котором функция меняет знак и содержит решение, причём подвижная точка приближается к ε-окрестности решения.

Новое значение для подвижной точки вычисляется по формуле x = a – f(a) * (b - a) / (f(b) – f(a)). Вычисления завершаются, когда абсолютная разница между значением текущей и новой подвижных точек становится меньше заданной точности.

Блок-схема



Листинг метода

```
public static double chordMethod(double a, double b, double eps, int
equation) throws NotExistingEquationException {
        checkIntervalArguments(a, b, equation);
        double temp;
        double diff = 1;
        double c = 0;
        while (Math.abs(diff) > eps){
            if (getEquation(equation, a) * getEquation(equation,
makeNextValue(a, b, equation)) < 0) {
                temp = b;
                b = makeNextValue(a, b, equation);
                diff = temp - b;
                c = b;
            } else {
                temp = a;
                a = makeNextValue(a, b, equation);
                diff = temp - a;
```

```
                c = a;
            }
        }
        return c;
        }

    private static double makeNextValue(double a, double b, int equation)
throws NotExistingEquationException {
        return (a - ((b - a) / (getEquation(equation, b) -
getEquation(equation, a))) * getEquation(equation, a));
    }
}
```

Расчетные формулы метода

$$c = a - \frac{f(a)}{f(b) - f(a)} \cdot (b - a)$$

Метод Ньютона

Описание метода

Идея метода состоит в том, чтобы начать с первоначального предположения, затем аппроксимировать функцию ее касательной линией и, наконец, вычислить x-перехват этой касательной линии. Этот x-перехват обычно будет лучшим приближением к корню исходной функции, чем первое предположение, и метод может быть повторен.

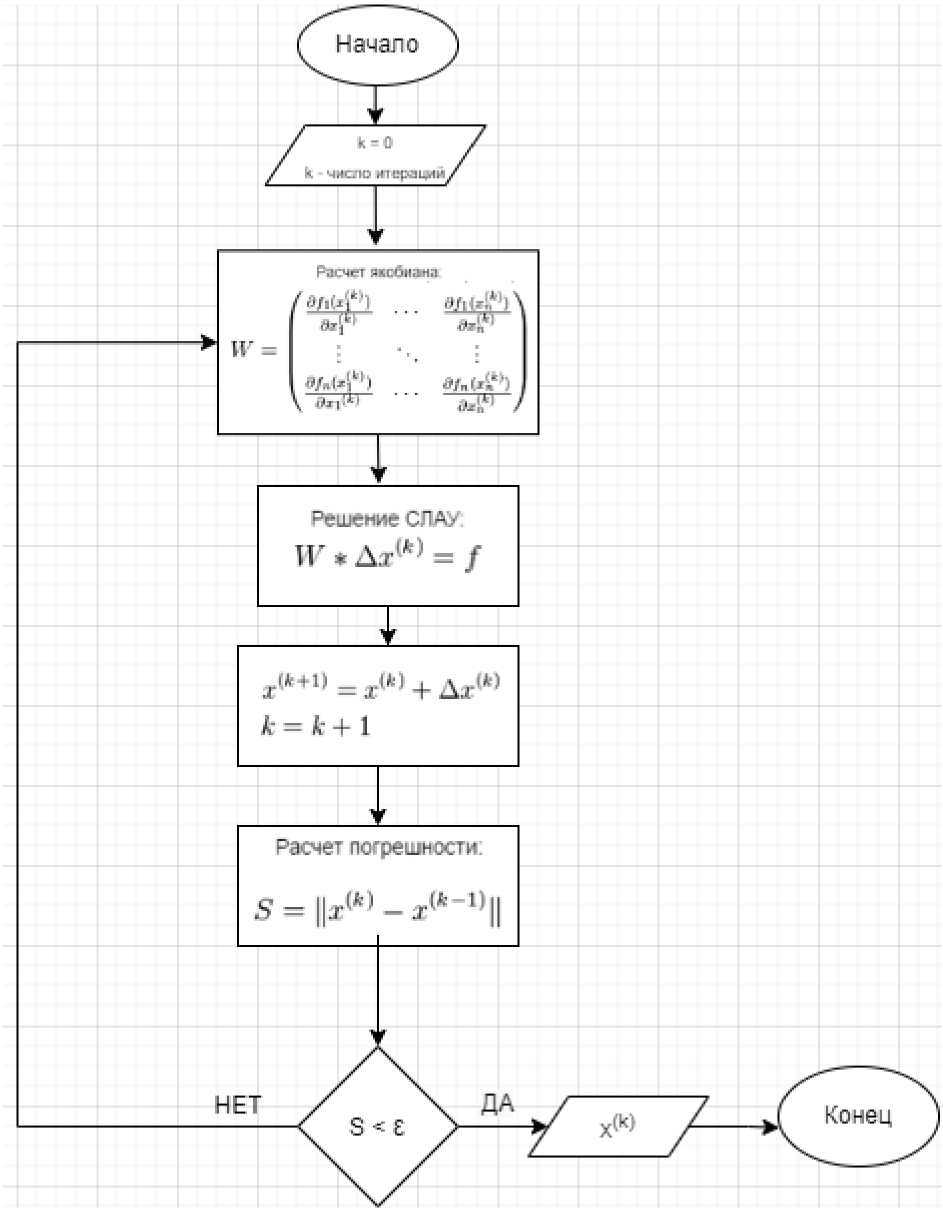
Если касательная к кривой f(x) при $x = x_n$ пересекает ось x при x_{n+1}, то наклон равен $f'(x_n) = \frac{f(x_n)}{x_n - x_{n+1}}$

Решение для x_{n+1} дает

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Мы начинаем процесс с некоторого произвольного начального значения x_0. Метод обычно сходится при условии, что это начальное предположение достаточно близко к неизвестному нулю и что $f'(x_0) \neq 0$.

Блок-схема



Листинг метода

```
public static double[] newtonMethod(int equationNumber, double x, double y,
double eps) throws NotExistingEquationException {
    double[][] jacoby;
    double[] answers = new double[2];
    double[] differs;
    double[] columns = new double[3];
    AbstractEquationSystem equationSystem =
getEquationSystems(equationNumber);
    columns[0] =equationSystem.getFirstEquation(x, y);
    columns[1] =equationSystem.getSecondEquation(x, y);
    double[] temp = {columns[0] + 2 * eps, columns[1] + 2 * eps};
    for (int i = 1; Math.abs(columns[0] - temp[0]) > eps &&
Math.abs(columns[1] - temp[1]) > eps; i++) {
        temp = Arrays.copyOf(columns, columns.length);
        jacoby = calculateJacoby(temp, equationSystem);
        answers[0] = -equationSystem.getFirstEquation(temp[0], temp[1]);
        answers[1] = -equationSystem.getSecondEquation(temp[0], temp[1]);
```

```
        differs = Gauss.getUnknownColumn(jacoby, answers);
        columns[0] += differs[0];
        columns[1] += differs[1];
        columns[2] += i;
    }

    return columns;
}

public static double[][] calculateJacoby(double[] columns,
AbstractEquationSystem system){
    double[][] jacoby = new double[2][2];
    jacoby[0][0] = system.getDerivativeFirstX(columns[0], columns[1]);
    jacoby[0][1] = system.getDerivativeFirstY(columns[0], columns[1]);
    jacoby[1][0] = system.getDerivativeSecondX(columns[0], columns[1]);
    jacoby[1][1] = system.getDerivativeSecondY(columns[0], columns[1]);
    return jacoby;
}
```

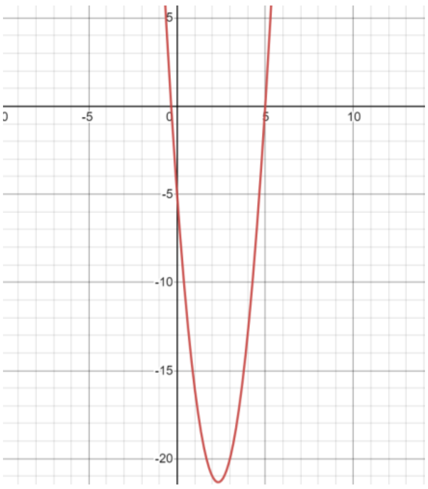
Расчетные формулы метода

$$J\left(x^{(k)}\right)\Delta x^{(k+1)} = -f\left(x^{(k)}\right)$$

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k+1)}$$

Результаты работы:

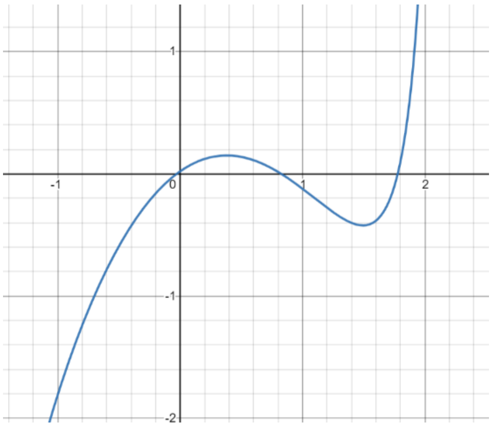
1)




```
Enter the beginning of the interval:
Enter the value: 0
Enter the end of the interval:
Enter the value: 5,5
Enter the accuracy:
Enter the value: 0,0001

Bisection solution result: 4.9999847412109375
Chord solution result: 4.99999728066797
The difference between two methods: 1.2539457032723078E-5
```

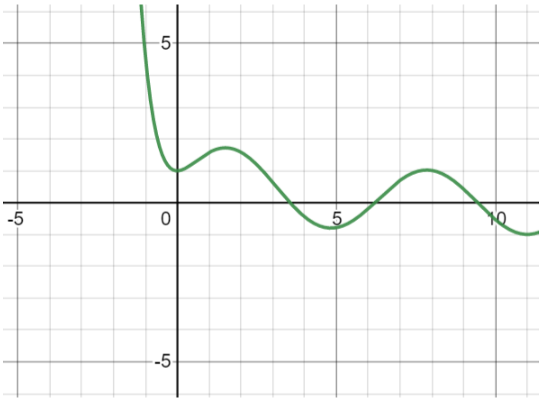
2)



```
Enter the beginning of the interval:
Enter the value: -0,54
Enter the end of the interval:
Enter the value: 0,23
Enter the accuracy:
Enter the value: 0,001

Bisection solution result: -0.027919921875000017
Chord solution result: -0.027138314624613358
The difference between two methods: 7.816072503866595E-4
```

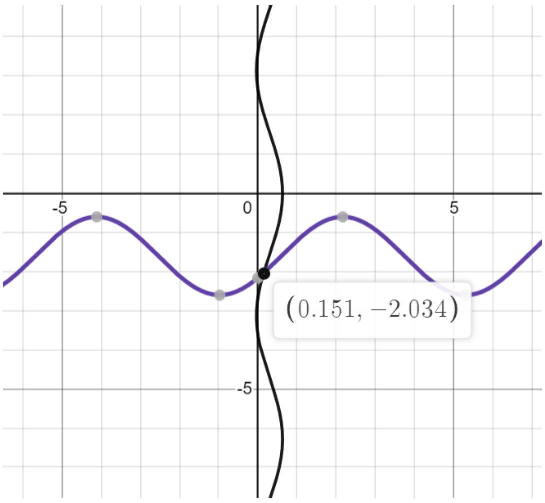
3)



```
Enter the beginning of the interval:
Enter the value: 1,35
Enter the end of the interval:
Enter the value: 5
Enter the accuracy:
Enter the value: 0,00001

Bisection solution result: 3.5441924095153814
Chord solution result: 3.544193138344842
The difference between two methods: 7.288294607832313E-7
```

4)




```
Enter the initial approximation for x:
Enter the value: -4
Enter the initial approximation for y:
Enter the value: 4
Enter the accuracy:
Enter the value: 0,00001

Newton method solution result:
x: 0.15105719263637626
y: -2.034013345171546
Number of iterations: 28.0
```

Вывод

В результате проделанной лабораторной работы я рассмотрел решения нелинейных уравнений методом деления пополам и методом хорд, а также решение систем нелинейных уравнений методом Ньютона.

В отличие от метода деления пополам, в методе хорд отрезок делится не пополам, пропорционально отношению f(a) / f(b). Из-за этого некоторые уравнения могут решаться быстрее методом хорд, чем методом деления пополам.

К недостаткам метода Ньютона следует отнести его локальность, поскольку он гарантированно сходится при произвольном стартовом приближении только, если везде выполнено условие , в противной ситуации сходимость есть лишь в некоторой окрестности корня, а также к недостаткам можно отнести необходимость вычисления производных на каждом шаге.