Лабораторная работа №2

17 Вариант

Имя: **Татарников Максим Станисловович**

Группа: А-07-22

Проверил:

Оценка:

Москва 2023

Содержание

1 Задача

1. Описать класс, представляющий нелинейное уравнение вида:

$$ax - cos(x) = 0$$

.

2. Описать метод, вычисляющий решение этого уравнения на заданном интервале методом деления пополам и выбрасывающий исключение в случае отсутствия корня.

3. Описать свойства для получения состояния объекта.

4. Написать программу, демонстрирующую все разработанные элементы класса.

5. Создать дочерний класс, реализующий метод вычисления решения этого уравнения на заданном интервале методом Ньютона.

2 Постановка задачи

Методы приближенных вычислений для уточнения корня Для того, чтобы найти значение корня функции $y = f(x)$ вовсе не обязательно искать обратную функцию $x = f^{-1}(y)$, существуют методы приближенного вычисления, которые позволяют вычислить корень с заданной точностью имея начальное приближение или отрезок.

2.1 Методы

2.1.1 Деление отрезка пополам

Рисунок 1 иллюстрирует метод половинного деления, который состоит в постепенном сужении отрезка поиска корня до заданной величины ε. На каждом шаге отрезок уменьшается вдвое.

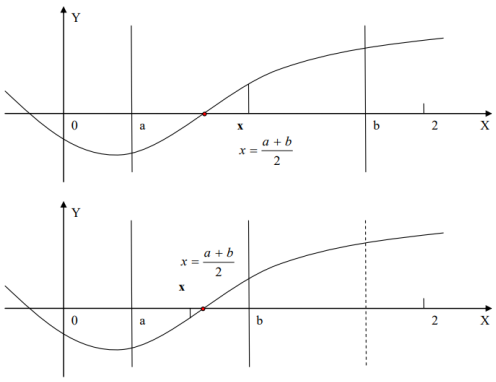


Рис. 1: Метод деление отрезка пополам

В начале каждой итерации находим середину нового отрезка $[a, b]$

$$x = \frac{a+b}{2}$$

Затем следует определить, с какой стороны от середины отрезка x находится корень $x*$. Для этого достаточно сравнить знаки $f(x)$ и $f(b)$ или знаки $f(x)$ и $f(a)$.

Если знаки $f(x)$ и $f(b)$ не совпадают, то это означает, что $f(x)$ пересекает ось x на правом полуотрезке $[x, b]$. Следовательно, корня нет на левом полуотрезке $[a, x]$, и этот полуотрезок

можно отбросить, то есть можно перенести левую границу a в среднюю точку x (заменить значение приближения a на значение x).

Если же знаки $f(x)$ и $f(b)$ совпадают, то $f(x)$ пересекает ось x на левом полуотрезке $[a, x]$ и, следовательно, корня нет на правом полуотрезке $[x, b]$, и этот полуотрезок можно отбросить, то есть можно перенести правую границу b в среднюю точку x (заменить значение приближения b на значение x).

Итак, в результате выполнения итерации отрезок $[a, b]$ как и прежде, содержит единственный корень, но его длина стала меньше в два раза.

Вычисления следует прекратить, если на очередном шаге длина отрезка $[a, b]$ станет меньше ε. Тогда с точностью ε любая точка этого отрезка – с точностью $\varepsilon/2$. Совпадение знаков $f(x)$ и $f(b)$ можно проверить, проверив неравенство

$$f(x) \cdot f(b) > 0$$

поскольку произведение двух чисел с одинаковыми знаками дает положительное значение.

2.2 Метод Ньютона (касательных)

Пусть имеется начальное приближение к корню, которое обозначим x_1. Например, можно взять середину отрезка $[a, b]$:

$$x_1 = \frac{a+b}{2}$$

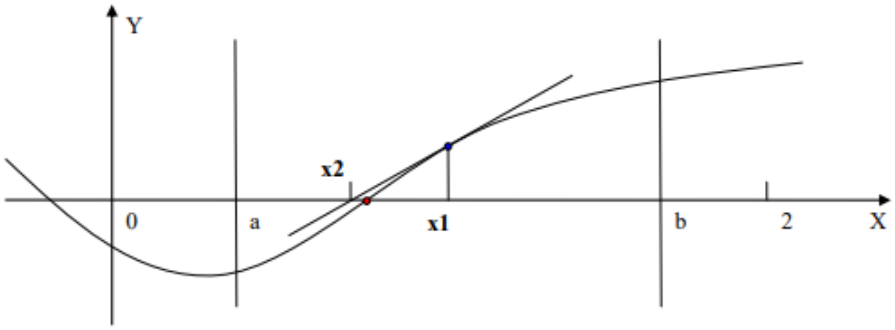


Рис. 2: Метод Ньютона (касательных)

Проведем касательную к графику $y = f(x)$ в точке с координатами $(x_1, f(x_1))$. Новое приближение к корню, которое мы будем называть следующим приближением, x_2 получим как точку пересечения этой касательной с осью абсцисс. Это правило приводит к следующей расчетной формуле:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

При соблюдении некоторых условий (они называются условиями сходимости), которые будут перечислены ниже, строго доказано, что приближение x_2 находится ближе к корню, чем приближение x_1. Теперь заменим значение начального приближения x_1 на значение только что полученного приближения x_2. Мы пришли к той же самой задаче, но теперь начальное приближение расположено ближе к корню, чем до его изменения на x_2. Каждое такое улучшение приближения к корню за счет вычисления следующего приближения называется итерацией. Сколько нужно выполнить итераций, чтобы нас могла устроить точность приближение x_2 к значению корня $x*$? Обычно считают, что требуемая точность достигнута, если после вычисления x_2 при выполнении очередной итерации соблюдается условие

$$|f(x_1)| < \varepsilon$$

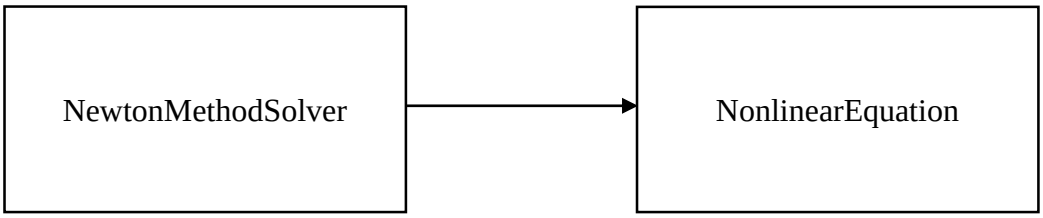
При выполнении этого неравенства итерационный процесс уточнения корня следует прекратить и в качестве искомого приближенного значения корня взять x_2. Смысл условий сходимости метода Ньютона состоит в том, что

1. Начальное приближение x_1, используемое при выполнении первой итерации, должно быть не слишком далеко от корня.

2. Производная $f'(x)$, которая находится в знаменателе, должна изменяется на отрезке $[a, b]$ не очень быстро и не обращаться в ноль ни в одной точке отрезка $[a, b]$.

Мы будем считать, что они выполняются. Метод Ньютона является наиболее быстрым среди численных методов вычисления корня функционального уравнения. На практике необходимая точность достигается буквально после выполнения нескольких (не более 10) итераций.

Диаграмма классов



Описание классов

NonlinearEquation:

Поля:

```
double a; // коэффициент перед X
double intervalStart; // начало интервала
double intervalEnd; // конец интервала
double epsilon; // точность
```

Конструктор и Деструктор:

```
NonlinearEquation(double coefficient, double start, double end, double eps) //
конструктор с параметрами
: a(coefficient), intervalStart(start), intervalEnd(end), epsilon(eps) {}

~NonlinearEquation() // деструктор без параметров {}
```

Методы:

```
double getA() const { // вызываем и получаем коэффициент}

double getIntervalStart() const { // вызываем и получаем начало отрезка}

double getIntervalEnd() const { //вызываем и получаем конец отрезка}

double getEpsilon() const { // вызываем и получаем точность}

double calculateEquationValue(double x) { // вызываем и получаем функцию}

double calculateEquationDerivative(double x) { // вызываем и получаем первую производную}

double solveByBisectionMethod() { // метод деления отрезка пополам}
```

NewtonMethodSolver:

Конструктор и Деструктор:

```
NewtonMethodSolver(double coefficient, double start, double end, double eps) //
конструктор с параметрами
        : NonlinearEquation(coefficient, start, end, eps) {}

~NewtonMethodSolver() // деструктор без параметров{}
```

Методы:

```
double solveByNewtonMethod(){}
```

Описание пользовательского интерфейса:

Функции пользователя:

1. Выбор метода решения

При входе в программу пользователь видит меню

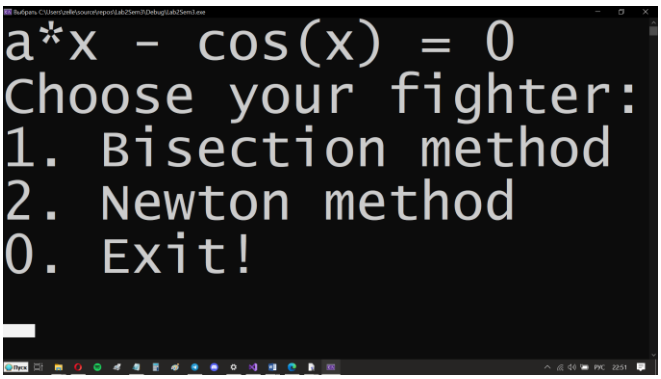


Рисунок 1 Выбор метода

С помощью ввода с клавиатуры пользователь выбирает метод решения
Если сделать выбор, которого нет в программе, программа выдаст следующее окно:

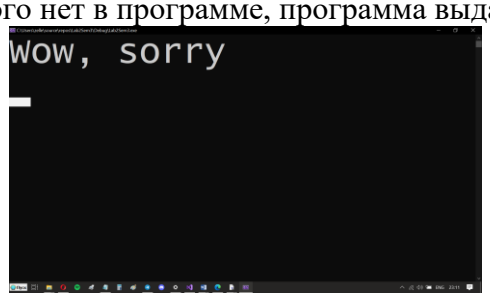


Рисунок 2 Ошибка ввода выбора метода

Нажав Enter, пользователь вернется на стартовый экран

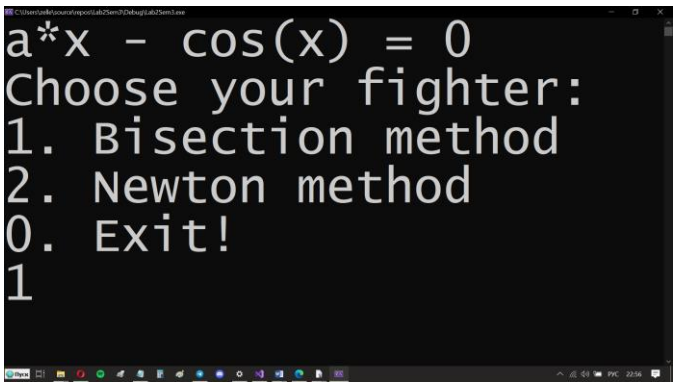


Рисунок 3 Ввод данных для выбора метода

Нажимая Enter, пользователь переходит на следующее окно

2. Ввод параметров

При вводе корректных значений выплывает следующее меню:
Ввод параметров происходит через ручной ввод (клавиатура). Пользователь вводит 4 параметра через пробел (коэффициент, начало отрезка, конец отрезка, точность)

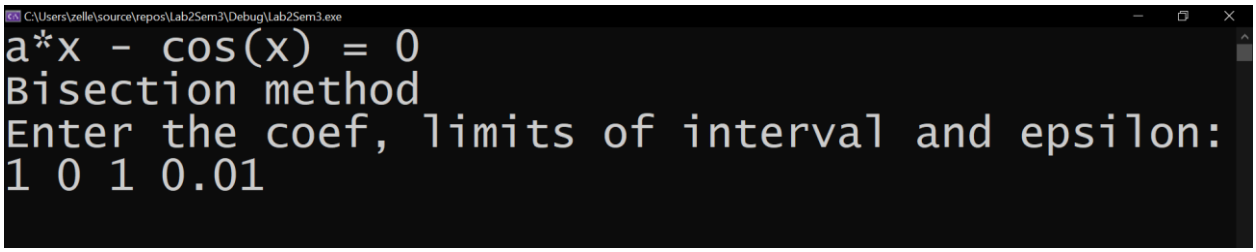


Рисунок 4 Ввод параметров

Нажимая Enter, пользователь получает результат

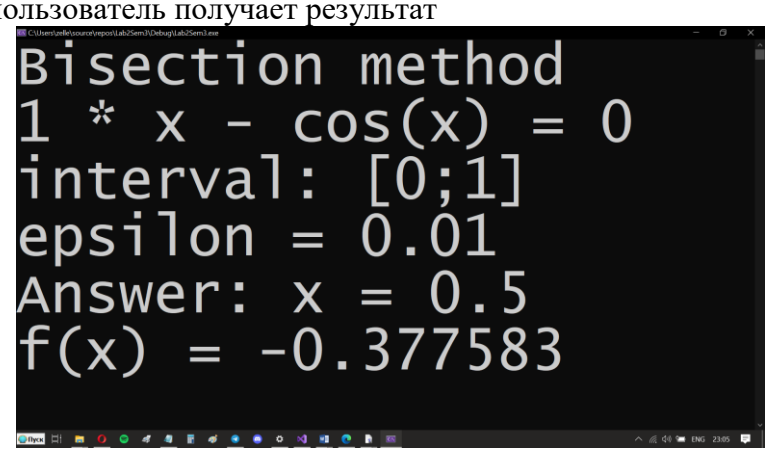


Рисунок 5 Результат

После нажатия Enter, пользователь переходит обратно на главный экран

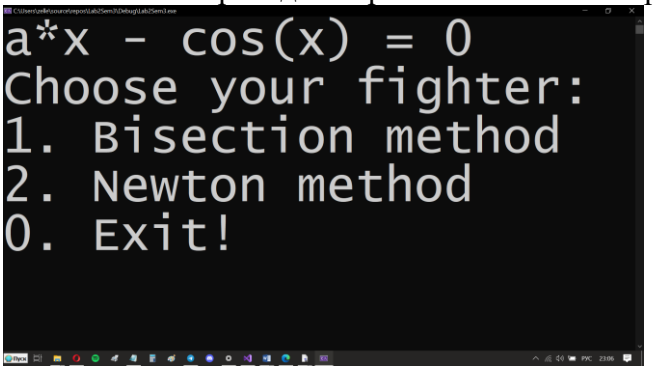


Рисунок 6 Переход на главный экран

И пользователь может заново выбрать метод, ввести данные и получить ответ или выйти из программы.

3. Выход из программы

При нажатии пользователем 0 и Enter – программа завершается

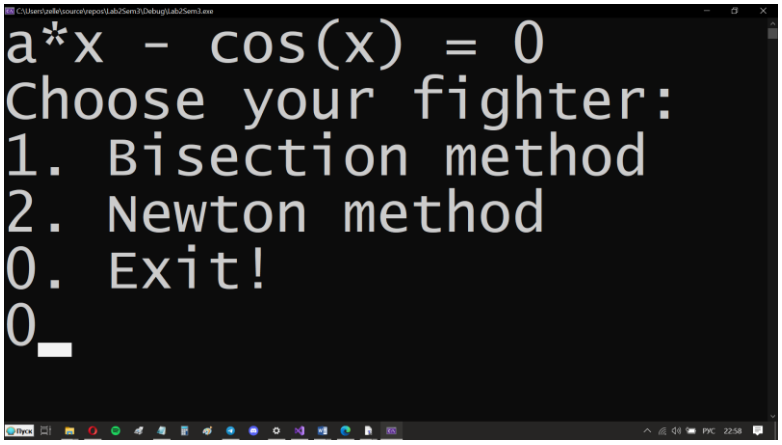


Рисунок 7 Ввод выхода из программы

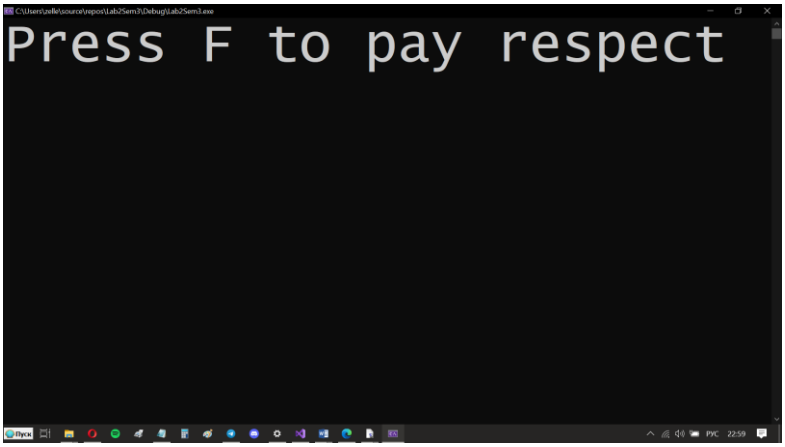


Рисунок 8 Выход из программы

При еще одном нажатии программа полностью завершается

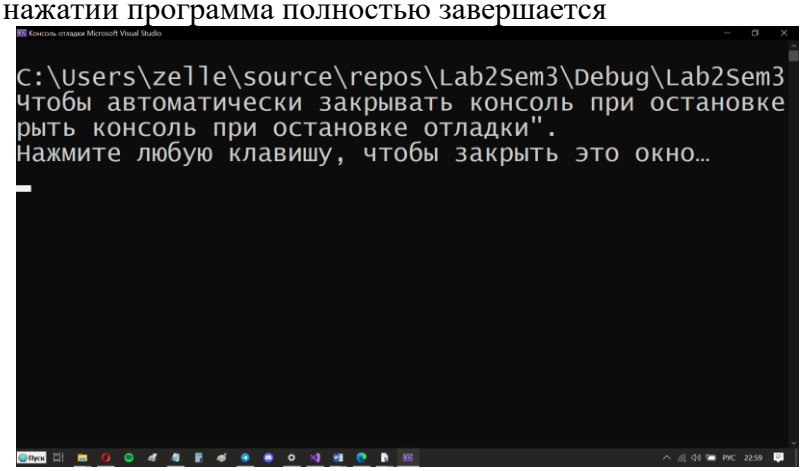


Рисунок 9 Полное завершение программы

Тесты

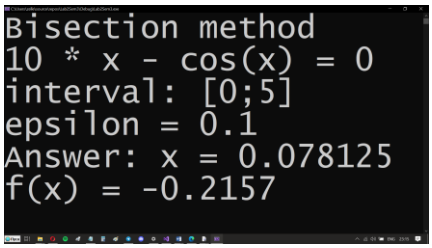


Рисунок 11 Тест 1

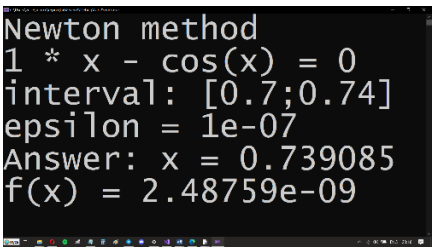


Рисунок 10 Тест 4

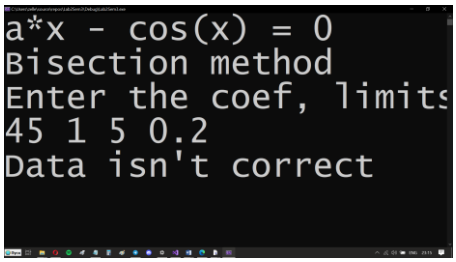


Рисунок 13 Тест 2 некорректный ввод данных

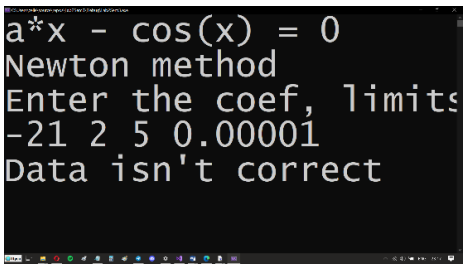


Рисунок 12 Тест 5 некорректные данные

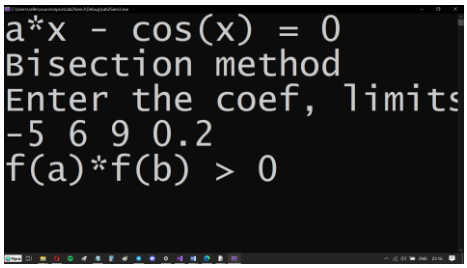


Рисунок 15 Тест 3 знак функций на концах отрезка

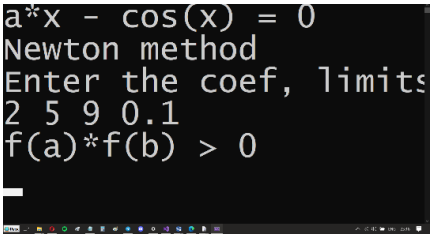


Рисунок 14 Тест 6 Знак функций на концах отрехка

Листинг

```
#include <conio.h> // getch
#include <iostream> // ввод вывод
#include <cmath> // математика

using namespace std; // чтобы не писать каждый раз std


class PosException { // класс исключения на знаки функций на концах
    public:
        void what() {
            cout << "f(a)*f(b) > 0\n";
        }
};

class CorrectDataExc { // исключение на некорректные данные
public:
    void what() {
        cout << "Data isn't correct\n";
    }
};

class NonlinearEquation { // родительский класс со всеми полями методами и тд
    double a; // коэффициент перед X
    double intervalStart; // начало интервала
    double intervalEnd; // конец интервала
    double epsilon; // точность

public:
    NonlinearEquation(double coefficient, double start, double end, double eps) //
конструктор с параметрами
        : a(coefficient), intervalStart(start), intervalEnd(end), epsilon(eps) {
    }

    ~NonlinearEquation() // деструктор без параметров
    {

    }

    double getA() const { // вызываем и получаем коэффициент
        return a;
    }

    double getIntervalStart() const { // вызываем и получаем начало отрезка
        return intervalStart;
    }

    double getIntervalEnd() const { //вызываем и получаем конец отрезка
        return intervalEnd;
    }

    double getEpsilon() const { // вызываем и получаем точность
        return epsilon;
    }

    double calculateEquationValue(double x) { // вызываем и получаем функцию
        return getA() * x - cos(x);
```

```
    }

    double calculateEquationDerivative(double x) { // вызываем и получаем первую производную
        return getA() + sin(x);
    }

    double solveByBisectionMethod() { // метод деления отрезка пополам
        double start = getIntervalStart(); // обозначаем старт
        double end = getIntervalEnd(); // обозначаем конец отрезка
        double root = 0.0; // наш корень

        if (fabs(getA()) >= 20 || fabs(getIntervalStart()) >= 20 || fabs(getIntervalEnd())
>= 20 || (getEpsilon() <= 0 && getEpsilon() >= 1)) throw CorrectDataExc(); // проверка, что
данные корректны иначе исключение
        else if (calculateEquationValue(start) * calculateEquationValue(end) > 0) throw
PosException(); // проверка на знак функции на концах отрезка иначе исключение
        else // иначе)
        {
            while ((end - start) > getEpsilon()) { // пока границы отрезка больше точности
крутим цикл
                double mid = (start + end) / 2.0; // находим середину отрезка

                if (calculateEquationValue(mid) < getEpsilon()) { // если значение функции
середины меньше точности, то говорим, что это корень и выходим из цикла
                    root = mid;
                    break;
                }
                else if (calculateEquationValue(mid) * calculateEquationValue(start) < 0) {
// если значения функции на концах разного знака, то сдвигаем одну часть отрезка
                    end = mid;
                }
                else { // иначе двигаем другую часть отрезка
                    start = mid;
                }
            }

            if (root != 0.0) { // если корень хоть раз изменился, то возвращаем его
                return root;
            }
            else {
                throw std::runtime_error("Equation does not have a root in the given
interval"); // иначе кидаем исключение
            }
        }
    }
};

class NewtonMethodSolver : public NonlinearEquation { // дочерний класс
public:
    NewtonMethodSolver(double coefficient, double start, double end, double eps) //
конструктор с параметрами
        : NonlinearEquation(coefficient, start, end, eps) {
    }

    ~NewtonMethodSolver() // деструктор без параметров
    {

    }

    double solveByNewtonMethod() { // решение методом ньютона
```

```
        if (fabs(getA()) >= 20 || fabs(getIntervalStart()) >= 20 || fabs(getIntervalEnd())
>= 20 || (getEpsilon() <= 0 && getEpsilon() >= 1)) // все как и в прошлом случае
        {
            throw CorrectDataExc();
        }
        double root = (getIntervalStart() + getIntervalEnd()) / 2.0;  // опять находим центр
        if (calculateEquationValue(getIntervalStart()) *
calculateEquationValue(getIntervalEnd()) > 0) // проверка на знак
            throw PosException();
        else
        {
            while (std::abs(calculateEquationValue(root)) > getEpsilon()) { // пока функция
больше точности
                root = root - calculateEquationValue(root) /
calculateEquationDerivative(root); // кайфуем с математики
            }
            return root; // возвращаем ответ
        }

    }
};

int main() { // главная функция

    double coefficient; // я не буду это комментировать...
    double Start;
    double End;
    double eps;

    int n=1; // переменная для case

    while (n != 0)
    {
        cout << "a*x - cos(x) = 0\n";
        cout << "Choose your fighter:\n";
        cout << "1. Bisection method\n";
        cout << "2. Newton method\n";
        cout << "0. Exit!\n";
        cin >> n;
        system("cls"); // чистим экран
        switch (n)
        {
        case 1:

        {
            cout << "a*x - cos(x) = 0\n";
            cout << "Bisection method\n";
            cout << "Enter the coef, limits of interval and epsilon:\n";
            cin >> coefficient >> Start >> End >> eps; // вводим параметры

            NonlinearEquation equation(coefficient, Start, End, eps); // оформляем
конструктор
            try {
                double root = equation.solveByBisectionMethod(); // получаем корень
                system("cls");
                cout << "Bisection method\n";
                cout << equation.getA() << " * x - cos(x) = 0 \ninterval: [" <<
equation.getIntervalStart() << ";" << equation.getIntervalEnd() << "] \nepsilon = " <<
equation.getEpsilon();
                cout << "\nAnswer: x = " << root << "\n";
            }
```

```
            catch (const std::runtime_error& e) { // если корень не найден
                cout << e.what();
            }
            catch (const PosException& e) { // если проблемы со знаком
                PosException exc;
                exc.what();
            }
            catch (const CorrectDataExc& e) { // если данные не правильны
                CorrectDataExc exc;
                exc.what();
            }
        }
            _getch();
            break;
        case 2:
        {
            cout << "a*x - cos(x) = 0\n";
            cout << "Newton method\n";
            cout << "Enter the coef, limits of interval and epsilon:\n";
            cin >> coefficient >> Start >> End >> eps; // заполняем данные

            NewtonMethodSolver eq(coefficient, Start, End, eps); // конструктор
            try {
                double root = eq.solveByNewtonMethod(); // получаем корень
                system("cls");
                cout << "Newton method\n";
                cout << eq.getA() << " * x - cos(x) = 0 \ninterval: [" <<
eq.getIntervalStart() << ";" << eq.getIntervalEnd() << "] \nepsilon = " << eq.getEpsilon();
                cout << "\nAnswer: x = " << root << "\n";
            }
            catch (const PosException& e) { // исключение раз
                PosException exc;
                exc.what();
            }
            catch (const CorrectDataExc& e) { // исключение два
                CorrectDataExc exc;
                exc.what();
            }
        }
            _getch();
            break;
        case 0:
            cout << "Press F to pay respect\n"; // Это на выход
            _getch();
            break;
        default:
            cout << "Wow, sorry\n"; // это если неправильно ввел цифру
            _getch();
            break;
        }
        system("cls"); // чистим экран
    }




    return 0; // конец)
}
```