**Федеральное государственное образовательное бюджетное учреждение  
высшего образования  
«Финансовый университет при Правительстве РФ»**

**КОЛЛЕДЖ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ**

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКИХ РАБОТ № 3 - 7

По дисциплине: Численные методы в программировании

«Методы нахождения корней уравнения»

Студент: Адещенко К.Р.

Группы: 3ПКС-316

Дата: 15.01.19\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Семенихина А. В*.*

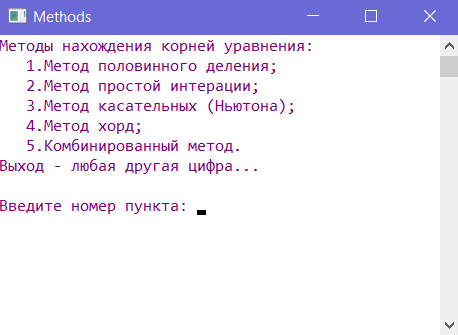
Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

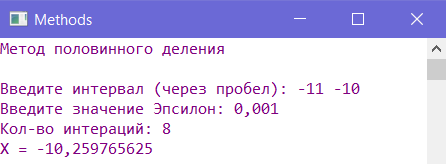
Москва 2018

Результат работы программы:

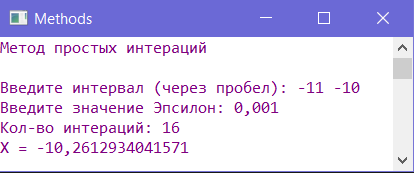
1. Меню программы



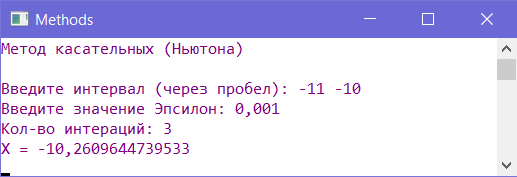
1. Метод половинного деления



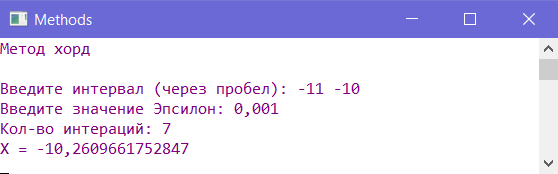
1. Метод простой итерации



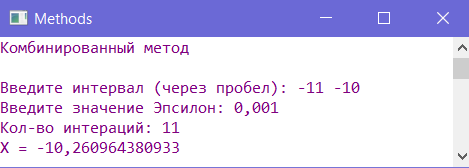
1. Метод касательных



1. Метод хорд



1. Комбинированный метод



Листинг программы:

//

// main.cpp

// lab\_1

//

// Created by hpowlows on 15/01/2019.

// Copyright © 2019 Kaparray. All rights reserved.

//

#include <iostream>

#include <vector>

#include <cmath>

#include <boost/regex.hpp>

#include <iostream> // std::cout

#include <string> // std::string, std::stod

using namespace std;

// Combined

double FunCombined(double x)

{

return pow(x, 4) - 3 \* pow(x, 2) + 75 \* x - 10000;

}

double FunCombined1(double x)

{

return 4 \* pow(x, 3) - 6 \* x + 75;

}

double FunCombined2(double x)

{

return 12 \* pow(x, 2) - 6;

}

double CalculateCombined(double a, double b, double eps, int n)

{

do

{

if (FunCombined(a) \* FunCombined2(a) < 0) //Условие начальной точки для метода хорд

{

a = a + (b - a) / (FunCombined(a) - FunCombined(b)) \* FunCombined(a); //формулы расчета по методу хорд

b = b - FunCombined(b) / FunCombined1(b);

}

if (FunCombined(a) \* FunCombined2(a) > 0) //Условие начальной точки для метода касательных

{

a = a - FunCombined(a) / FunCombined1(a); //формулы расчета по методу касательных

b = b + (b - a) / (FunCombined(b) - FunCombined(a)) \* FunCombined(b);

}

if (FunCombined(a) == 0) return a;

if (FunCombined(b) == 0) return b;

n++;

} while (abs(b - a) > eps);

return (a + b) / 2.0;

}

// Chord ////////////////////////////

double FunChord(double x)

{

return pow(x, 4) - 3 \* pow(x, 2) + 75 \* x - 10000;

}

double FunChord1(double x)

{

return 4 \* pow(x, 3) - 6 \* x + 75;

}

double FunChord2(double x)

{

return 12 \* pow(x, 2) - 6;

}

double CalculateChord(double a, double b, double eps, int n)

{

double x = a, xl;

if (FunChord2(a) > 0)

{

x = a;

do

{

xl = x;

x = xl - FunChord(xl) / (FunChord(b) - FunChord(xl)) \* (b - xl);

n++;

} while (abs(x - xl) >= eps);

}

else

{

x = b;

do

{

xl = x;

x = xl - FunChord(xl) / (FunChord(xl) - FunChord(a)) \* (xl - a);

n++;

} while (abs(x - xl) >= eps);

}

return x;

}

/////////////////////////////////////////////////////////////

// Tangents //

double FunTangents(double x)

{

return pow(x, 4) - 3 \* pow(x, 2) + 75 \* x - 10000;

}

double FunTangents1(double x)

{

return 4 \* pow(x, 3) - 6 \* x + 75;

}

double FunTangents2(double x)

{

return 12 \* pow(x, 2) - 6;

}

double CalculateTangents(double a, double b, double eps, int n)

{

double x, xl;

if (FunTangents(a) \* FunTangents2(a) > 0) x = a;

else x = b;

do

{

xl = x;

x = xl - FunTangents(xl) / FunTangents1(xl);

n++;

}

while (abs(x - xl) >= eps);

return x;

}

/////////////////////////////////////////////////////////////

// Simple\_Iteration //

double FunSimple\_Iteration(double x)

{

return pow(x, 4) - 3 \* pow(x, 2) + 75 \* x - 10000;

}

double FunSimple\_Iteration1(double x)

{

return 4 \* pow(x, 3) - 6 \* x + 75;

}

double CalculateSimple\_Iteration(double a, double b, double eps, int n)

{

double l, xl, x;

x = a;

l = -2 / FunSimple\_Iteration1(x);

do

{

xl = x;

x = x + l \* FunSimple\_Iteration(x);

n++;

} while (abs(x - xl) > eps);

return x;

}

/////////////////////////////////////////////////////////////

// Half\_Division //

double FunHalf\_Division(double x)

{

return pow(x, 4) - 3 \* pow(x, 2) + 75 \* x - 10000;

}

double CalculateHalf\_Division(double a, double b, double eps, int n)

{

double x = (a + b) / 2;

if (FunHalf\_Division(a) \* FunHalf\_Division(x) < 0)

b = x;

else

a = x;

if ((b - a) > 2 \* eps && FunHalf\_Division(x) != 0)

{

n++;

return CalculateHalf\_Division(a, b, eps, n);

}

else

return x;

}

/////////////////////////////////////////////////////////////

static void input(double a, double b, double eps)

{

string \*users = NULL;

cout << "Введите интервал (через пробел): ";

string text; cin >> text;

users = new string[boost::split(text, users, boost::is\_any\_of(" "))];

a = std::stod(users[0]);

b = std::stod(users[1]);

cout << "Введите значение Эпсилон: ";

cin >> eps;

}

static void output(double x, int n)

{

cout << "Кол-во интераций: " << n << "\nX = " << x;

system("cls");

}

int main() {

int num, n = 0;

double a = 0, b = 0, eps = 0, x = 0;

input\_next:

cout << "Методы нахождения корней уравнения:\n 1.Метод половинного деления;\n 2.Метод простой интерации;\n" <<

" 3.Метод касательных (Ньютона);\n 4.Метод хорд;\n 5.Комбинированный метод.\nВыход - любая другая цифра...\n";

do

{

cout << "Введите номер пункта: ";

} while (cin >> num);

switch (num)

{

case 1: x = CalculateHalf\_Division(a, b, eps, n); break;

case 2: x = CalculateSimple\_Iteration(a, b, eps, n); break;

case 3: x = CalculateTangents(a, b, eps, n); break;

case 4: x = CalculateChord(a, b, eps, n); break;

case 5: x = CalculateCombined(a, b, eps, n); break;

}

input(a, b, eps); // input data here

output(x, n); // output

goto input\_next;

return 0;

}