**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Основы информатики»

Отчет по лабораторной работе №4

«Нахождение корней нелинейного уравнения»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-11 |  | преподаватель каф. ИУ5 |
| Михалёв Ярослав |  | Козлов А.Д. |
| Подпись и дата: |  | Подпись и дата: |
|  |  |  |

Москва, 2021 г.

Постановка задачи

1. Найти корень уравнения

x - cos(x) = 0

простой итерацией, половинным делением и методом Ньютона с погрешностью eps<0.000001 и для каждого из трех методов определить количество шагов алгоритма.

2. Выполнить п.1 для eps < 0.00000001.

3. Выполнить п.1 для уравнения

x – 10cos(x) = 0

и объяснить результаты.

Разработка алгоритма

Метод итерации

Описание входных и выходных данных:

1. double start – левая граница
2. double end – правая граница
3. double eps – точность
4. double f(x) – функция
5. int iter – количество итераций
6. double x – корень уравнения f(x)

Метод Ньютона

Описание входных и выходных данных:

1. double x – окрестность поиска корней
2. double eps – точность
3. double f(x) – функция
4. double df(x) – производная функции
5. int iter – количество итераций
6. double x – корень уравнения f(x)

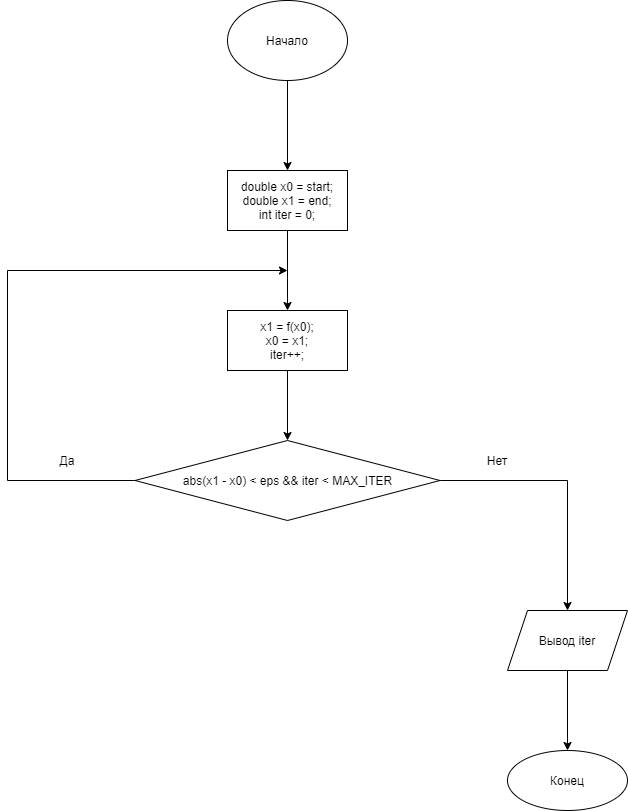
Метод половинного деления

Описание входных и выходных данных:

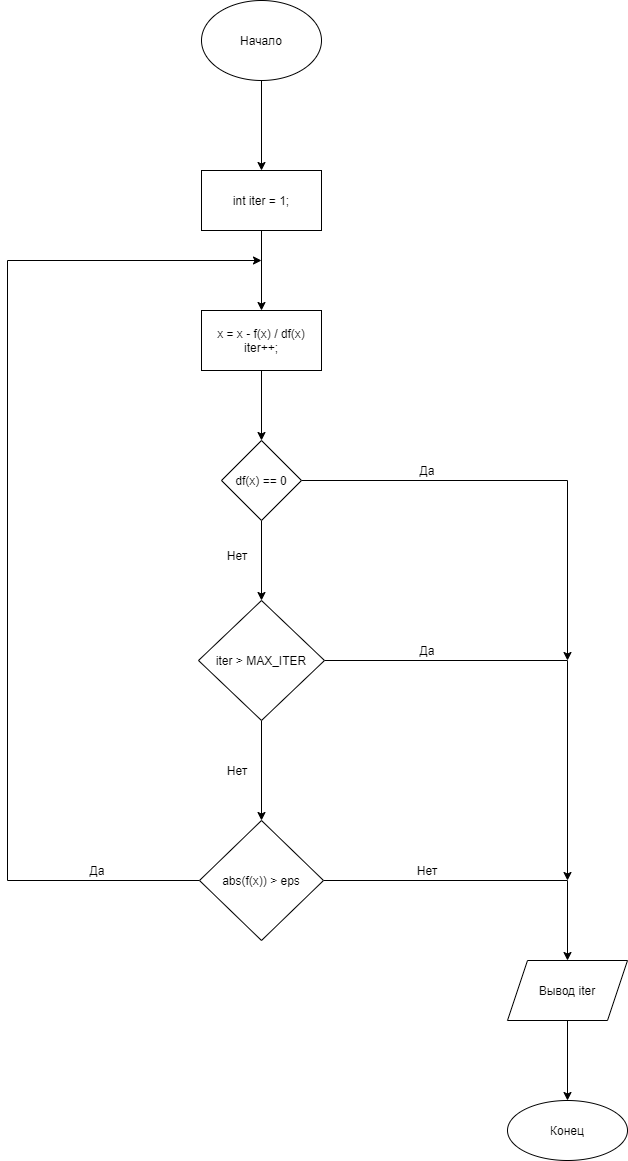
1. double x0 – левая граница
2. double x1 – правая граница
3. eps – точность
4. double f(x) – функция
5. int iter – количество итераций
6. double x – корень уравнения f(x)

Схема алгоритма

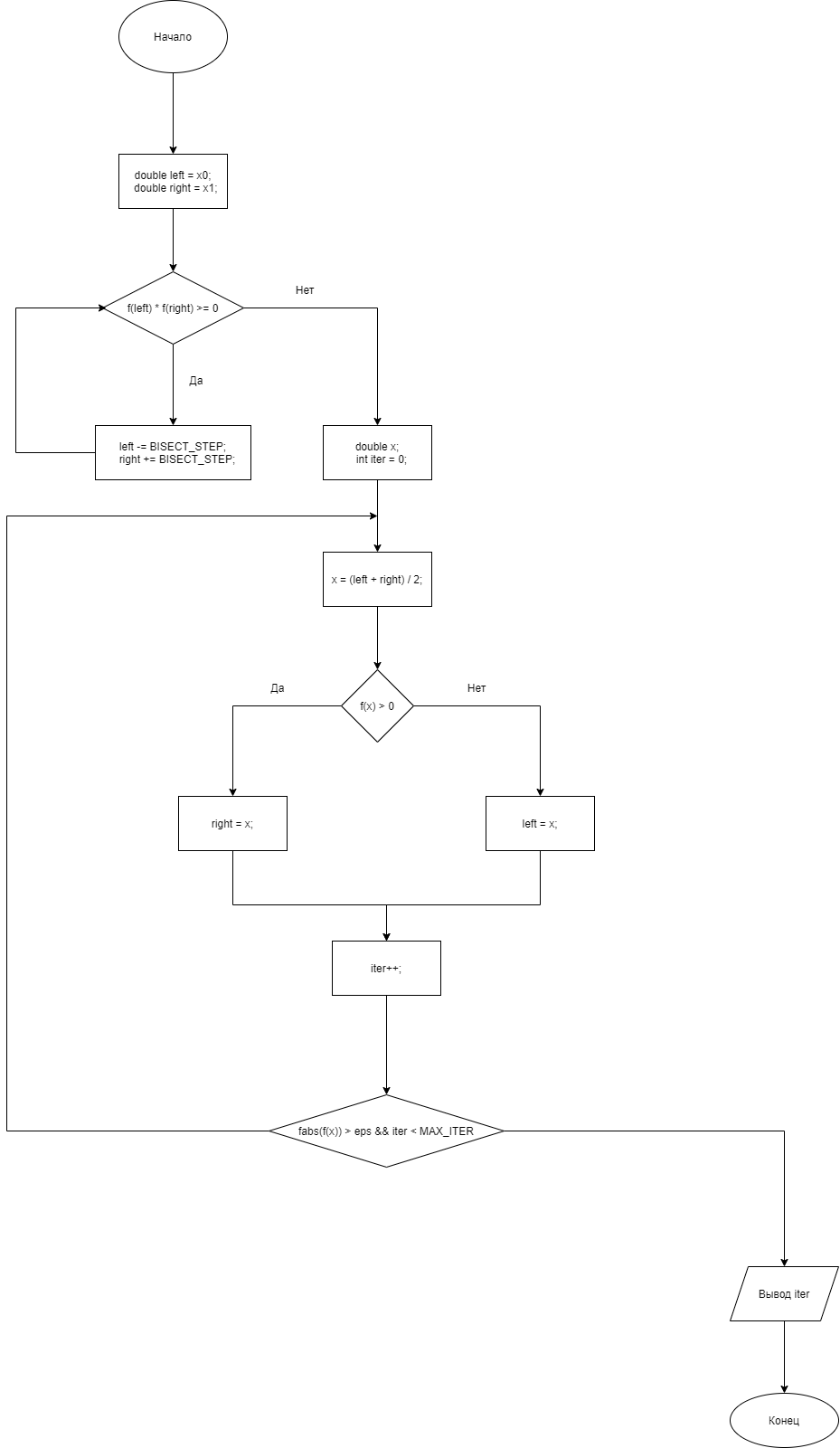
Iteration.cpp



Newton.cpp



Bisection.cpp



Текст программы

Main.cpp

#include <iostream>

#include <iomanip>

#include <cmath>

#include "BruteForce.h"

#include "Iteration.h"

#include "Newton.h"

#include "Bisection.h"

**using** **namespace** std**;**

#define precision1 7

#define eps1 0.000001

#define precision2 9

#define eps2 0.00000001

double f1**(**double x**)**

**{**

**return** x **-** cos**(**x**);**

**}**

double f2**(**double x**)**

**{**

**return** x **-** 10 **\*** cos**(**x**);**

**}**

double g1**(**double x**)**

**{**

**return** cos**(**x**);**

**}**

double g2**(**double x**)**

**{**

**return** 10 **\*** cos**(**x**);**

**}**

double df1**(**double x**)**

**{**

**return** 1 **+** sin**(**x**);**

**}**

double df2**(**double x**)**

**{**

**return** 1 **+** 10 **\*** sin**(**x**);**

**}**

int main**()**

**{**

system**(**"chcp 1251 > nul"**);**

cout **<<** "--- Метод итераций ---" **<<** endl**;**

cout **<<** setprecision**(**precision1**)** **<<** "x1 = " **<<** Iteration**(**0.0**,** 1.0**,** eps1**,** g1**)** **<<** endl**;**

cout **<<** setprecision**(**precision2**)** **<<** "x1 = " **<<** Iteration**(**0.0**,** 1.0**,** eps2**,** g1**)** **<<** endl**;**

cout **<<** setprecision**(**precision1**)** **<<** "x2 = " **<<** Iteration**(**1.0**,** 2.0**,** eps1**,** g2**)** **<<** endl**;**

cout **<<** "\n\n\n\n"**;**

cout **<<** "--- Метод Ньютона ---" **<<** endl**;**

cout **<<** setprecision**(**precision1**)** **<<** "x1 = " **<<** Newton**(**1.0**,** eps1**,** f1**,** df1**)** **<<** endl**;**

cout **<<** setprecision**(**precision2**)** **<<** "x1 = " **<<** Newton**(**1.0**,** eps2**,** f1**,** df1**)** **<<** endl**;**

cout **<<** setprecision**(**precision1**)** **<<** "x2 = " **<<** Newton**(**1.0**,** eps2**,** f2**,** df2**)** **<<** endl**;**

cout **<<** "\n\n\n\n"**;**

cout **<<** "--- Метод половинного деления ---" **<<** endl**;**

cout **<<** setprecision**(**precision1**)** **<<** "x1 = " **<<** Bisection**(**0.0**,** 1.0**,** eps1**,** f1**)** **<<** endl**;**

cout **<<** setprecision**(**precision2**)** **<<** "x1 = " **<<** Bisection**(**0.0**,** 1.0**,** eps2**,** f1**)** **<<** endl**;**

cout **<<** setprecision**(**precision1**)** **<<** "x2 = " **<<** Bisection**(**1.0**,** 2.0**,** eps1**,** f2**)** **<<** endl**;**

**return** 0**;**

**}**

Iteration.cpp

#include "Iteration.h"

#include <iostream>

#include <cmath>

**using** **namespace** std**;**

#define MAX\_ITER 5000000

double Iteration**(**double start**,** double end**,** double eps**,** double**(**f**(**double x**)))**

**{**

double x0 **=** start**;**

double x1 **=** end**;**

int iter **=** 0**;**

**do**

**{**

x1 **=** f**(**x0**);**

x0 **=** x1**;**

iter**++;**

**}** **while** **(**abs**(**x1 **-** x0**)** **<** eps **&&** iter **<** MAX\_ITER**);**

cout **<<** "Количество итераций " **<<** iter **<<** endl**;**

**return** x1**;**

**}**

Newton.cpp

#include "Newton.h"

#include <math.h>

#include <iostream>

**using** **namespace** std**;**

#define MAX\_ITER 5000000

double Newton**(**double x**,** double eps**,** double**(**f**(**double**)),** double**(**df**(**double**)))**

**{**

int iter **=** 1**;**

**do**

**{**

x **=** x **-** f**(**x**)** **/** df**(**x**);**

iter**++;**

**if** **(**df**(**x**)** **==** 0**)**

**break;**

**if** **(**iter **>** MAX\_ITER**)**

**break;**

**}** **while** **(**abs**(**f**(**x**))** **>** eps**);**

cout **<<** "Количество итераций " **<<** iter **<<** endl**;**

**return** x**;**

**}**

Bisection.cpp

#include "Bisection.h"

#include <iostream>

**using** **namespace** std**;**

#define BISECT\_STEP 0.05

#define MAX\_ITER 5000000

double Bisection**(**double x0**,** double x1**,** double eps**,** double**(**f**(**double**)))**

**{**

double left **=** x0**;**

double right **=** x1**;**

**while** **(**f**(**left**)** **\*** f**(**right**)** **>=** 0**)** **{**

left **-=** BISECT\_STEP**;**

right **+=** BISECT\_STEP**;**

**}**

double x**;**

int iter **=** 0**;**

**do** **{**

x **=** **(**left **+** right**)** **/** 2**;**

**if** **(**f**(**x**)** **>** 0**)**

right **=** x**;**

**else**

left **=** x**;**

iter**++;**

**}** **while** **(**fabs**(**f**(**x**))** **>** eps **&&** iter **<** MAX\_ITER**);**

cout **<<** "Количество итераций " **<<** iter **<<** endl**;**

**return** x**;**

**}**

Анализ результатов

Метод итераций

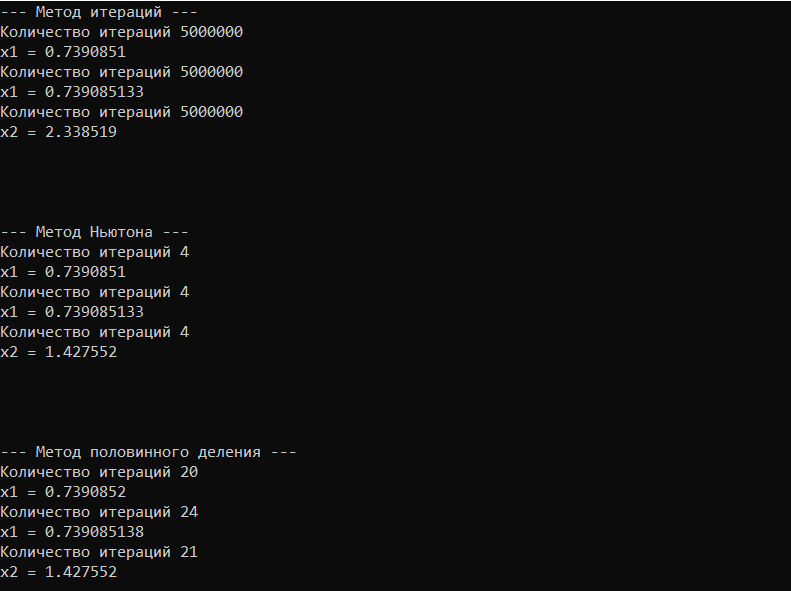
|  |  |  |  |
| --- | --- | --- | --- |
| Отрезок | Точность | Количество итераций | Корень |
| [0,1] | e-7 | 5000000 | 0.7390851 |
| [0,1] | e-9 | 5000000 | 0.739085133 |
| [-10000, 10000] | e-7 | 5000000 | 0.7390851 |
| [-10000, 10000] | e-9 | 5000000 | 0.739085133 |

Метод Ньютона

|  |  |  |  |
| --- | --- | --- | --- |
| Окрестность | Точность | Количество итераций | Корень |
| 1 | e-7 | 4 | 0.7390851 |
| 1 | e-9 | 4 | 0.739085133 |
| -10000 | e-7 | 81 | 0.7390851 |
| -10000 | e-9 | 81 | 0.739085133 |

Метод бисекции

|  |  |  |  |
| --- | --- | --- | --- |
| Отрезок | Точность | Количество итераций | Корень |
| [0,1] | e-7 | 20 | 0.7390852 |
| [0,1] | e-9 | 24 | 0.739085138 |
| [-10000, 10000] | e-7 | 34 | 0.7390848 |
| [-10000, 10000] | e-9 | 40 | 0.739085135 |



Проанализировав результаты программы, можно сказать, что самым эффективным алгоритмом для решения нахождение корней нелинейного уравнения является Метод Ньютона

Вывод

Я научился использовать функции