**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 ИУ5-13Б |  | преподаватель каф. ИУ5 |
| Лачина Екатерина |  | Аксенова М.В. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2023 г.

1. Задача

**Задание:**

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.

**Указания**

Численному решению уравнения

f(x) = 0 (1)

должно предшествовать хотя бы грубое исследование вопросов существования и положения корней.

Итерационные методы

Заданное уравнение f(x) = 0 приводят к виду

x = ϕ(x). (2)

Выбирая некоторое начальное приближение Х0,  вычисляют последовательные приближения

Хj+1 = ϕ(Xj), (j=0, 1, 2, …).

Сходимость таких приближений к искомому решению Х требует отдельного исследования. Сходимость зависит прежде всего от вида функции, а также от начального приближения. (В данной лабораторной работе такие исследования не делаются, но в пункте 3 задания приведена функция, для которой решения методом Ньютона и методом простой итерации расходятся.) Для того, чтобы программа нахождения корней этими методами не зацикливалась, следует ограничивать максимальное число итераций Nmax, например, Nmax < 100000.

Возможны различные способы приведения уравнения (1) к виду (2).

Простая итерация

Хj+1 = Xj - f(Xj).

Метод Ньютона

Хj+1 = Xj - f(Xj)/ f ′(Xj).

Метод половинного деления

Для использования этого метода нужно задать границы интервала на оси абсцисс, содержащего ровно один корень [xl, xr] и требуемую точность вычислений.

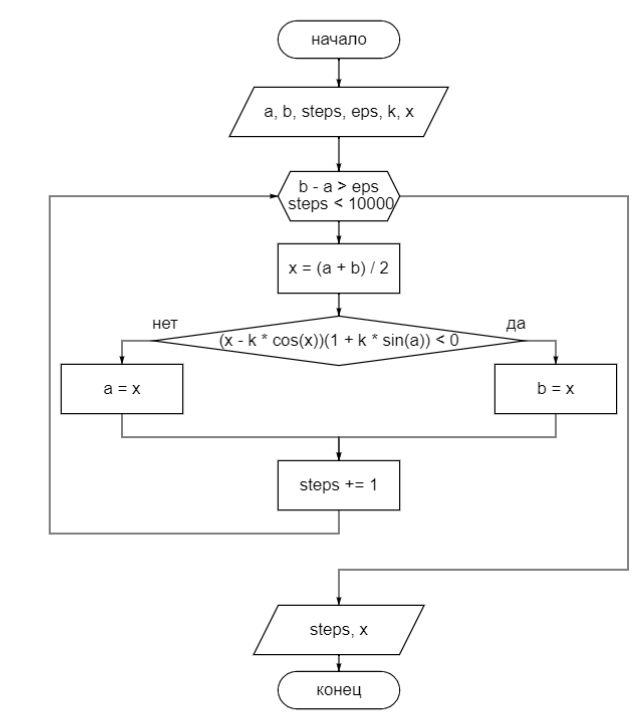
Суть метода заключается в следующем. Выбирают Х на середине интервала [xl, xr] и определяют f(X). Если f(X) < eps, то середина интервала считается корнем функции, иначе корень ищется на том интервале из двух полученных, для которого значения функции на концах имеют разные знаки.

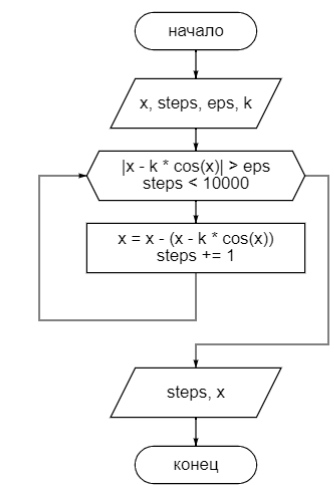
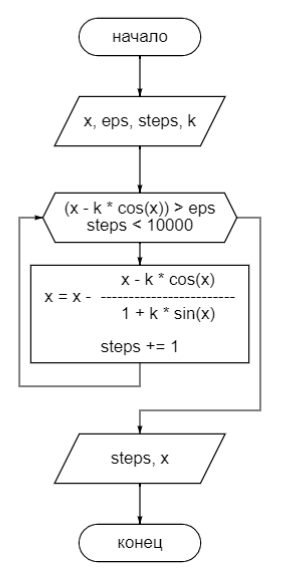
1. Разработка алгоритма

Для решения задачи я создаю многофайловый проект, включающий следующие файлы: **file.h –** заголовочный файл, где я подключаю прототипы функций. **special.cpp –** файл, где прописаны все функции. **laba4.cpp** – файл, который вызывает функции и выводит на консоль результат.

Переменные.

|  |  |  |
| --- | --- | --- |
| Функция типа **double** | Переменные типа **double** | Переменные типа **int** |
| **f1 –** выражение функции | x – корень | k – коэффициент при косинусе |
| **f2 –** производная моей функции | x – корень | k – коэффициент при синусе |
| **simpleiteration –** метод простых итераций | x – корень  eps – эпсилон | steps – счетчик шагов выполнения функции  k – коэффициент (для **f1, f2**) |
| Функция типа **double** | Переменные типа **double** | Переменные типа **int** |
| **binpoisk –** метод деления отрезка пополам (бинарный поиск) | a – левая граница  b – правая граница  x – корень  eps – эпсилон | speps – счетчик шагов выполнения функции  k – коэффициент (для **f1, f2**) |
| **newton –** метод Ньютона | x – корень  eps – эпсилон | steps – счетчик шагов выполнения функции  k – коэффициент (для **f1, f2**) |



1. Листинг

**file.h**

#include <iomanip>

#include <iostream>

double simpleiteration(double x, double eps, int steps, int k);

double binpoisk(double a, double b, double eps, int steps, int k);

double newton(double x, double eps, int steps, int k);

**special.cpp**

#include <cmath>

#include <iostream>

double f1(double x, int k) { //выражение функции

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

}

double f2(double x, int k) { //производная моей функции

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

}

double simpleiteration(double x, double eps, int steps, int k) {

steps = 0;

x = 0;

while (fabs(f1(x, k)) > eps && steps < 10000) {

x = x - f1(x, k);

steps++;

}

std::cout << steps << std::endl;

return x;

}

double binpoisk(double a, double b, double eps, int steps, int k) {

steps = 0;

double x = 0;

while (b - a > eps && steps < 10000) {

x = (a + b) / 2;

if (f1(x, k) \* f1(a, k) < 0) {

b = x;

}

else {

a = x;

}

steps++;

}

std::cout << steps << std::endl;

return x;

}

double newton(double x, double eps, int steps, int k) {

steps = 0;

x = 0;

while (fabs(f1(x, k)) > eps && steps < 10000) {

x = x - f1(x, k) / f2(x, k);

steps++;

}

std::cout << steps << std::endl;

return x;

}

**laba4.cpp**

#include <iostream>

#include <cmath>

#include "file.h"

#include "special.cpp"

int main(){

system("color F0");

double eps = 0.000001;

int steps=0;

int k = 1;

double x = 0;

//(1)

std::cout << "double eps = 0.000001; k = 1" << std::endl;

std::cout << "steps\_simple: " << std::endl;

double root\_simple = simpleiteration(0, eps, steps, k);

std::cout << "root\_simple: " << root\_simple << std::endl;

std::cout << "steps\_binpoisk: " << std::endl;

double root\_binpoisk = binpoisk(0, 1, eps, steps, k);

std::cout << "root\_binpoisk: " << root\_binpoisk << std::endl;

std::cout << "steps\_newton: " << std::endl;

double root\_newton = newton(1, eps, steps, k);

std::cout << "root\_newton: " << root\_newton << std::endl;

std::cout << "\n";

//(2)

eps = 0.00000001;

std::cout << "eps = 0.00000001; k = 1" << std::endl;

std::cout << "steps\_simple: " << std::endl;

root\_simple = simpleiteration(0, eps, steps, k);

std::cout << "root\_simple: " << std::setprecision(8) << root\_simple << std::endl;

std::cout << "steps\_binpoisk: " << std::endl;

root\_binpoisk = binpoisk(0, 1, eps, steps, k);

std::cout << "root\_binpoisk: " << std::setprecision(9) << root\_binpoisk << std::endl;

std::cout << "steps\_newton: " << std::endl;

root\_newton = newton(1, eps, steps, k);

std::cout << "root\_newton: " << std::setprecision(8) << root\_newton << std::endl;

std::cout << "\n";

//(3)

eps = 0.000001;

double root = 0;

k = 10;

std::cout << "eps = 0.000001; k = 10" << std::endl;

std::cout << "steps\_simple: " << std::endl;

root = simpleiteration(0, eps, steps, k);

std::cout << "root (x - 10cos(x) = 0) by simple: " << std::setprecision(7) << root << std::endl;

std::cout << "steps\_binpoisk: " << std::endl;

root = binpoisk(0, 5, eps, steps, k);

std::cout << "root (x - 10cos(x) = 0) by binpoisk: " << std::setprecision(7) << root << std::endl;

std::cout << "steps\_newton: " << std::endl;

root = newton(1, eps, steps, k);

std::cout << "root (x - 10cos(x) = 0) by newton: " << std::setprecision(7) << root << std::endl;

system ("pause");

return 0;

}

4) Тестирование

