1

Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Автоматизированные системы обработки информации и управления»



**Отчет**

**Лабораторная работа № 4**

**«Численные методы решения нелинейных уравнений»**

**ИСПОЛНИТЕЛЬ:**

ФИО

Группа ИУ5-14Б

«фамилия»

"\_\_"\_\_\_\_\_\_\_\_\_\_\_201 г.

**ПРЕПОДАВАТЕЛЬ:**

Папшев И.С.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

"\_\_"\_\_\_\_\_\_\_\_\_\_\_201 г.

Москва 2018

* **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 (и вывести на экран) количество шагов алгоритма.   
   2. Выполнить п.1 для eps < 0.00000001.  
   3. Выполнить п.1 для уравнения  
   x – 10cos(x) = 0  
   и объяснить результаты.  
   Разрабатывайте программу так, чтобы пункты 1, 2 и 3 выполнялись бы для каждого из методов по одному алгоритму (для этого коэффициент при cos(x) сделайте переменной).

* **Цель работы**

- Программирование численных методов решения нелинейных уравнений.

- Сравнительный анализ методов простой итерации, половинного деления и метода Ньютона.

* **Разработка алгоритма**

Необходимо вычислить корень уравнения тремя разными методами: Методом простых итераций, Методом половинного деления(дихотомии) и Методом Ньютона. Выводить результаты следует сразу трёх методов чтобы была возможность сравнить количество шагов затраченное на нахождение корня и результаты с учётом введённой погрешности eps.  
Программу необходимо сделать «повторяющуюся» при помощи цикла do-while, дабы было удобно отлаживать и, опять же, сравнивать результаты с разной погрешностью.  
**(\*)Алгоритмы:  
*\* Метод простых итераций***

Метод заключается в последовательном приближении к искомому значению(корню). Выбирая некоторое начально

значение Х0 , вычисляются последовательные приближения.

За Х0 принимаем правую границу интервала **a**. Сходимость таких приближений к искомому решению Х требует отдельного исследования. Сходимость зависит прежде всего от вида функции, а также от начального приближения.

Для того, чтобы программа нахождения корней этими методами не зацикливалась, следует ограничивать максимальное число итераций Nmax, например, Nmax < 100.

i = 0; //метод простой итерации (k)

x = a;

f1 = a - (k\*cos(a));

f2 = b - (k\*cos(b));

if (((f1\*f2) < 0))

{

do

{

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

i++;

} while ((abs(x - (k\*cos(x))) > eps) && (i < 12));

cout << "(\*) Метод простой итерации: \n" << " \n";

cout << " Корень: " << setprecision(10) << x << endl;

cout << " Найден за " << i << " шаг(-а;-ов.) \n" << " \n";

}

else

{

cout << "(\*) Метод простой итерации: \n" << " \n";

cout << " Корней НЕТ, или возможен случай что x = 0 \n" << " \n";

}

***\* Метод половинного деления (дихотомии)***

Принцип метода кроется в его названии. Для нахождения корня уравнения на некотором интервале **[a,b]** подставляем за место X значения **a** и **с** – середины интервала **[a,b]**, и вычисляем произведение

**f(a) \* f(c)**. Если знак после умножения меняется на противоположный, тогда корень уравнения находится как раз таки между **a** и **с**. Тогда принимаем за левую границу интервала значение **с** вместо **b**. Если не меняется, то берём с за место правой границы **a**.

После проделанного повторяем действия с новым интервалом пока условие выхода из цикла не перестанет удовлетворять(abs(b - a) > eps).

i = 0; //метод половинного деления (k)

double z;

f1 = a - (k\*cos(a));

f2 = b - (k\*cos(b));

if (((f1\*f2) < 0))

{

do

{

c = (a + b) / 2;

if (((a - (k\*cos(a)))\*(c - (k\*cos(c))) > 0))

a = c;

else

b = c;

z = (a + b) / 2;

i++;

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

x = z;

cout << "(\*) Метод половинного деления: \n" << " \n";

cout << " Корень: " << setprecision(10) << x << endl;

cout << " Найден за " << i << " шаг(-а;-ов.)\n" << " \n";

}

else

{

cout << "(\*) Метод простой итерации: \n" << " \n";

cout << " Корней НЕТ, или возможен случай что x = 0 \n" << " \n";

}

***\* Метод Ньютона (метод касательной)***

Данный метод является самым простым, за одним исключением: требует вычисления производной.

Хj+1 = Xj - f(Xj)/ f ′(Xj)

//метод Ньютона (k)

i = 0;

x = a;

f1 = a - (k\*cos(a));

f2 = b - (k\*cos(b));

if (((f1\*f2) < 0))

{

do

{

x = x - ((x - (k\*cos(x))) / ((k\*sin(x)) + 1));

i++;

} while ((abs(x - (k\*cos(x)))) && (i < n));

cout << "(\*) Метод Ньютона: \n" << " \n";

cout << " Корень: " << setprecision(10) << x << endl;

cout << " Найден за " << i << " шаг(-a;-ов.) \n" << " \n";

}

else

{

cout << "(\*) Метод простой итерации: \n" << " \n";

cout << " Корней НЕТ, или возможен случай что x = 0 \n" << " \n";

}

Во всех этих способах коэффициент k необходим для выполнения задания ( k = 10). Для нахождения корня первоначально данной функции k = 1.

В коде этих численных методов также предусмотрен исход с отсутствием корней, или если X равен нулю.

((f1\*f2) < 0

* **Кодирование**

#include <iostream>

#include <iomanip>

#include <cmath>

using namespace std;

int main()

{

system("chcp 1251>nul ");

int j = 1, l = 0;

char exit1, exit2, koef;

double f1, f2;

cout << "Найти корень уравнения x-cos(x)\n 'Методом простой итерациии'\n 'Методом Ньютона'\n 'Методом половинного деления' \n";

do {

int i, k;

double x, a, b, c, eps;

cout << " \n" << j << "." << l << ")\n" << " \n";

cout << "Введите точность вычислений: ";

cin >> eps;

cout << "Введите границы на оси Ox (a,b): ";

cin >> a >> b;

cout << " \n";

if (a == b) //проверяем интервал

{

cout << "a не должно быть равно b, начните заного!";

system("pause");

return 0;

}

if (a > b)

{

cout << "a должно быть меньше b, начните заного!";

system("pause");

return 0;

} //проверяем интервал \*

i = 0; //метод Ньютона

x = a;

f1 = a - cos(a);

f2 = b - cos(b);

if (((f1\*f2) < 0))

{

do

{

x = x - ((x - cos(x)) / (sin(x) + 1));

i++;

} while (abs(x - cos(x)));

cout << "(\*) Метод Ньютона: \n" << " \n";

cout << " Корень: " << setprecision(10) << x << endl;

cout << " Найден за " << i << " шаг(-a;-ов.) \n" << " \n";

}

else

{

cout << "(\*) Метод простой итерации: \n" << " \n";

cout << " Корней НЕТ, или возможен случай что x = 0 \n" << " \n";

}

i = 0; //метод простой итерации

x = a;

f1 = a - cos(a);

f2 = b - cos(b);

if (((f1\*f2) < 0))

{

do

{

x = x - (x - cos(x));

i++;

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

cout << "(\*) Метод простой итерации: \n" << " \n";

cout << " Корень: " << setprecision(10) << x << endl;

cout << " Найден за " << i << " шаг(-а;-ов.) \n" << " \n";

}

else

{

cout << "(\*) Метод простой итерации: \n" << " \n";

cout << " Корней НЕТ, или возможен случай что x = 0 \n" << " \n";

}

i = 0; //метод половинного деления

double z;

f1 = a - cos(a);

f2 = b - cos(b);

if (((f1\*f2) < 0))

{

do

{

c = (a + b) / 2;

if (((a - cos(a))\*(c - cos(c)) > 0))

a = c;

else

b = c;

z = (a + b) / 2;

i++;

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

x = z;

cout << "(\*) Метод половинного деления: \n" << " \n";

cout << " Корень: " << setprecision(10) << x << endl;

cout << " Найден за " << i << " шаг(-а;-ов.)\n" << " \n";

}

else

{

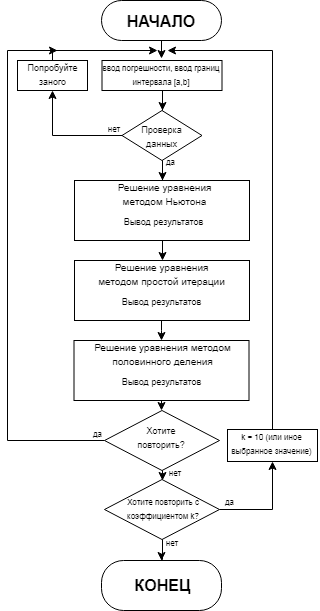
cout << "(\*) Метод простой итерации: \n" << " \n";

cout << " Корней НЕТ, или возможен случай что x = 0 \n" << " \n";

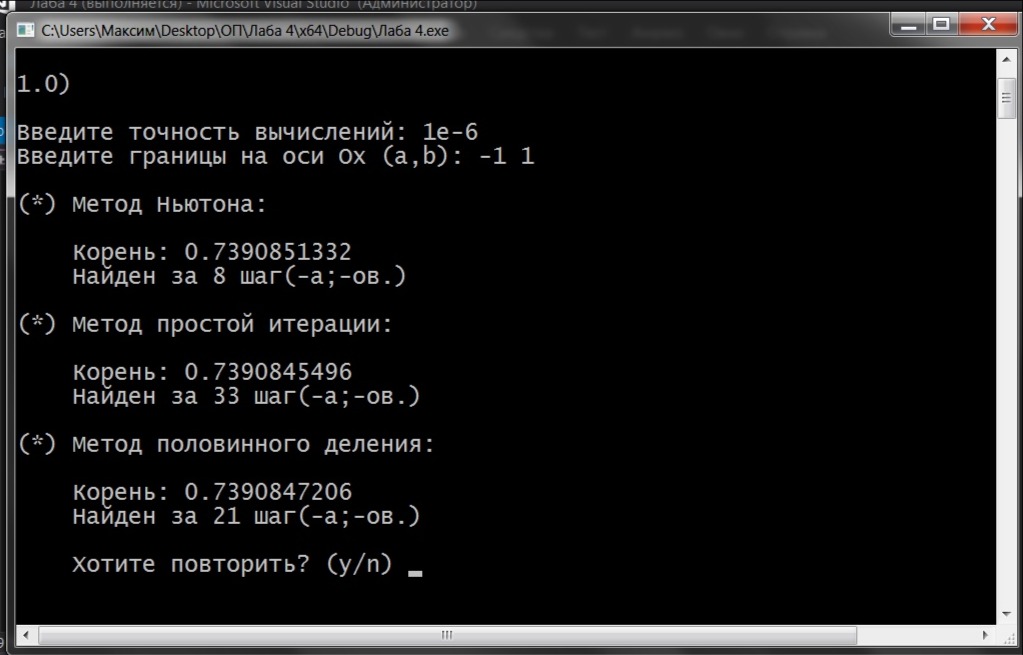
}

cout << " Хотите повторить? (y/n) ";

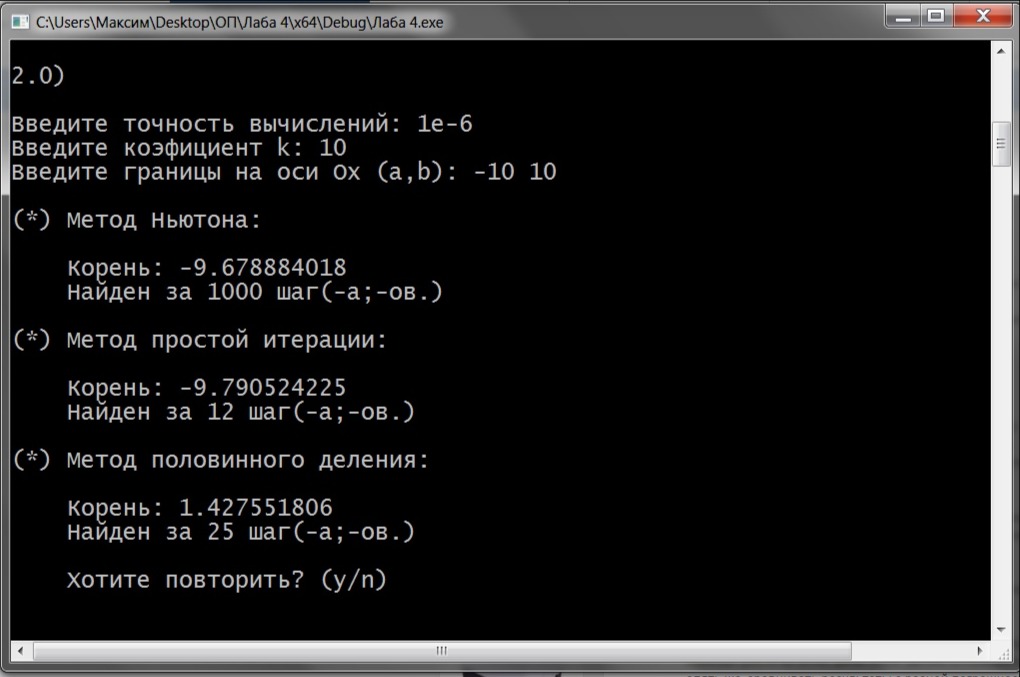
cin >> exit1;



* **Результаты**

****

При k = 10



Если на интервале не находится корней или X = 0

