Московский авиационный институт

(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

1 семестр

Курсовой проект

По курсу «Вычислительные системы»

Задание IV

Выполнил: Ляпин И. А.

Группа: М8О-104Б-22

Руководитель: Потенко М. А.

Оценка:_____

Дата:_____

Москва, 2022

Содержание

Цель работы:

Составить программу на языке Си с процедурами решения трансцендентных алгебраических уравнений различными численными методами (итераций, Ньютона и половинного деления — дихотомии). Нелинейные уравнения оформить как параметры-функции, разрешив относительно неизвестной величины в случае необходимости. Применить каждую процедуру к решению двух уравнений, заданных двумя строками таблицы, начиная с варианта с заданным номером. Если метод неприменим, дать математическое обоснование и графическую иллюстрацию, например, с использованием gnuplot.

Ход работы

Для решения необходимо запрограммировать 3 метода поиска корней на языке Си, а именно: Метод дихотомии, метод итераций, метод Ньютона. Все методы, представленные в виде функций нашей программы будут производить вычисления с некоторой точностью, то есть приближенные значения. Для задания данной точности воспользуемся значением машинного эпсилона. После достижения данной точности алгоритм можно считать максимально точным.

Опишем используемые методы при нахождении приближенного значения корная уравнения:

Метод Ньютона:

Метод Ньютона применим, если известно, что функция на заданном отрезке имеет один корень, причем первая и вторая производные на этом отрезке определены, непрерывны и сохраняют постоянные знаки. Возьмем x_0 - середину отрезка [a,b] и проведем в точке P0 $\{x_0, f(x_0)\}$ графика функции касательную к кривой y=f(x) до пересечения с осью Ox. Абсциссу x_1 точки пересечения можно взять в качестве приближенного значения корня. Проведя касательную через новую точку P_1 $\{x_1, f(x_1)\}$ и находя точку ее пересечения с осью Ox, получим второе приближение корня x_2 . Получившаяся последовательность сходится, если $|f(x) \cdot f''(x)| < (f'(x))^2$.

Метод Дихотомии:

Если на отрезке [a, b] существует корень уравнения, то значения функции на концах отрезка имеют разные знаки: F(a) * F(b) < 0. Метод заключается в делении отрезка пополам и его сужении в два раза на каждом шаге итерационного процесса в зависимости от знака функции в середине отрезка. Уравнение имеет действительные корня, если функция является непрерывной, и значения на концах отрезка имеют разные знаки. Путем последовательного деления отрезка пополам,

отбирая половину, которой корень принадлежит, мы будем производить сближение к корню. Метод можно считать завершенным, если достигнута заданная точность.

Метод итераций:

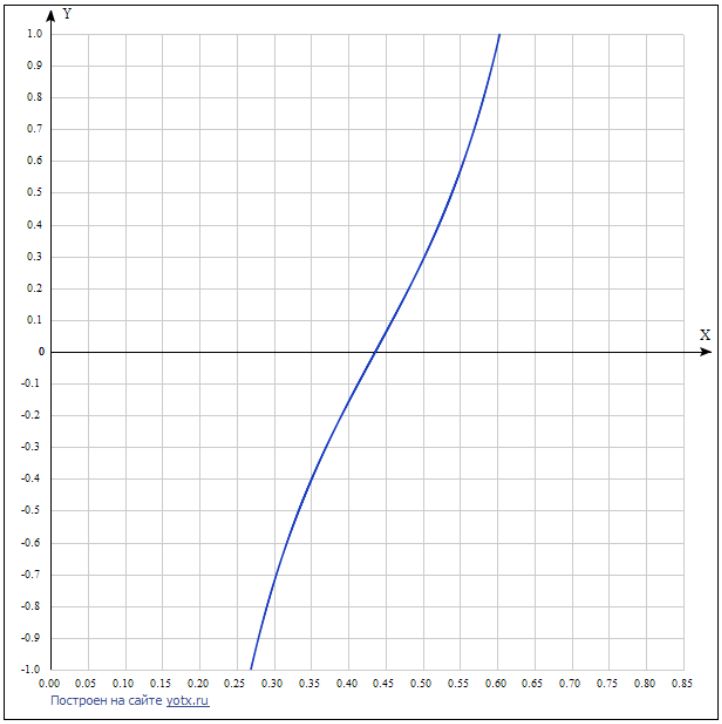
Уравнение f(x)=0 преобразуем в виде x = F(x). Выберем на заданном отрезке его середину x0 в качестве начального приближения и построим последовательность: $x_1=F(x_0)$, $x_2=F(x_1)$, ..., $x_n= F(x_n\text{-}1)$. Процесс итераций сходится если $|f'(x)| < 1$ на отрезке, и увеличивая n, можно получить приближение, сколь угодно мало отличающееся от истинного значения корня.

Варианты задания:

19	$x - \frac{1}{3 + \sin 3.6x} = 0$	[0, 0.85]	итераций	0.2624
20	$0{,}1x^2 - x \ln x = 0$	[1, 2]	Ньютона	1.1183

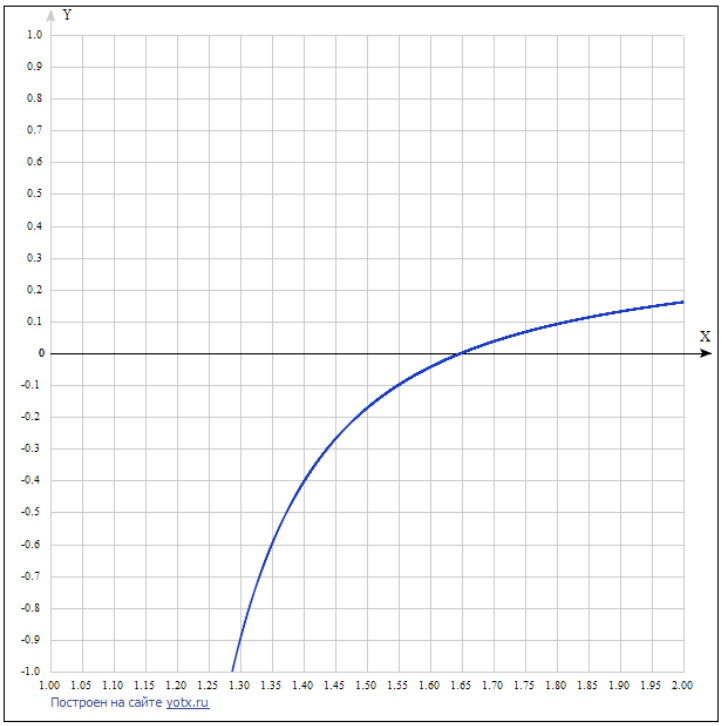
Следует проверить применимость функций в методе итераций до начала работы программы:

$$f_1^I(x) = \frac{-6 * cos(3.6x)}{5 * sin(3.6x)^2} :$$



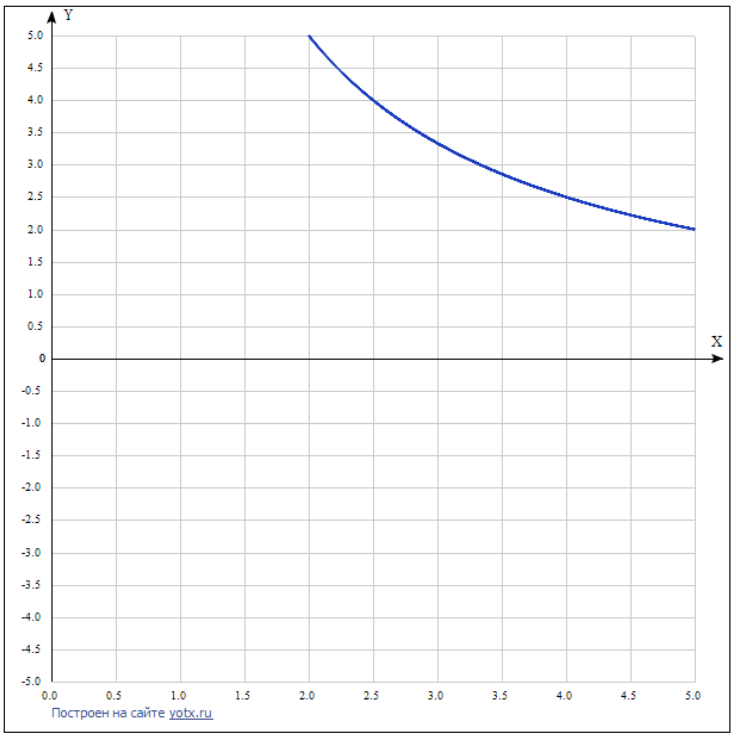
Как можно заметить, функция является достаточно определимой, чтобы найти корень уравнения, так как пользуясь обычным методом вычисления корня, можно убедиться, что он принадлежит данному графику.

$$f_2^I(x) = \frac{2 * ln(x) * x - x}{10 * ln(x)^2}:$$



В данном случае ситуация обратная, корень уравнения однозначно не принадлежит графику, функция в точке корня больше единицы по модулю, следовательно метод итераций неприменим.

Причем если мы выразим x по-другому, то получим функцию $f_2(x) = 10 * ln(x)$, её производная равна $f_2^I(x) = \frac{10}{x}$, которая явно не принадлежит указанному промежутку в задании :



Остальные методы можно проверить напрямую в коде программы.

Алгоритм

Для удобства вынесем всевозможные функции как отдельные.

Опишем функции методов и получение машинного эпсилона:

```
double FEps (void) // функция для вычисления машинного эпсилона
{
    double eps = 1.0;
    while (eps + 1.0 > 1.0) {
        eps /= 2.0;
    }
    return eps;
}
```

Данная функция возвращает значение машинного эпсилона.

Машинный эпсилон— числовое значение, меньше которого невозможно задавать относительную точность для любого алгоритма, возвращающего вещественные числа. Абсолютное значение «машинного эпсилон» зависит от разрядности сетки применяемой ЭВМ, типа (разрядности) используемых при расчетах чисел, и от принятой в конкретном трансляторе структуры представления вещественных чисел (количества бит, отводимых на мантиссу и на порядок).Формально машинный эпсилон обычно определяют как минимальное из чисел ε, для которого 1+ε>1 при машинных расчетах с числами данного типа. Альтернативное определение — максимальное ε, для которого справедливо равенство 1+ε=1.

Метод итераций:

```
double Iterations (double a, double b, double EPS, double f(double), double
f_diff(double)) // производим метод итераций
{
    double x; // заведем аргумент x
    int counterIter = 0; // заводим счетчик итераций, чтобы дать верхнюю границу
    x = (a + b) / 2; // присваиваем x значение середины отрезка

    if (fabs(f_diff(x)) < 1 && (fabs(f_diff(x / 1.5)) < 1)) { // проверяем на
сходимость метода, то есть производная от функции, вычленной корнем должна быть
< 1 по модулю
        while (fabs(f(x) - x) > EPS && counterIter < 100) { // заводим цикл,
пока абсолютная разница не будет < некторого машинного эпсилона или кол-во
итераций = 100
            x = f(x);
            counterIter++;
        }
    } else { // если метод невыполним, то завершаем функцию
        return 0;
    }
    return x; // выводим значение корня
}
```

В данную функцию подаются значения концов отрезка, значение машинного эпсилона, функция (зависит от варианта), производная предыдущей функции.

Внутри неё описан метод итераций из методичных указаний, но необходимо завести счетчик итераций, чтобы избежать возможного бесконечного цикла, а также следует провести проверку значения производный в более меньшем значении, чем середина отрезка, чтобы показать в случае чего неприменимость метода. Если метод неприменим, возвращаем 0 для удобства.

Метод Ньютона:

```
double Newton (double a, double b, double EPS, double f(double), double
f_2(double), double f_2_diff (double)) // создаем функцию, вычисл. корень по
Ньютону
{
    double x;
    x = (a + b) / 2; // значение середины отрезка
    if (fabs(f(x) * f_2_diff(x)) <= pow(f_2(x), 2)) { // проверка на выполнение
метода
        while (fabs(f(x) / f_2(x)) > EPS) { // цикл, пока значение деление не
стало меньшим или равным eps
            x -= f(x) / f_2(x); // вычитаем из x
        }
    } else { // выводим 0 , если метод неприменим
        return 0;
    }
    return x; // выводим корень уравнения
}
```

Аналогично методу итераций аргументами функции являются значения начала и конца отрезка, эпсилон, функция(зависит от варианта), её производная и вторая производная. Метод внутри функции соответсвует раннее описанному методу Ньютона. Если метод неприменим, то возвращаем значение 0, аналогично методу итераций и следующему методу Дихотомии.

Метод Дихотомии:

```
double Dikhotomy (double a, double b, double EPS, double f(double))
{
    double x;
    int counter = 0; // заводим счетчик итераций цикла, чтобы определить
верхнюю границу
    if (f(a) * f(b) < 0) { // проверка выполения метода
        while (fabs(a - b) > EPS && counter < 100) { // цикл пока сближение
небудет меньшим или равным eps
            x = (a + b) / 2; // придаем аргументы значение середины отрезка в
момент итерации
            if (f(a) * f((a + b) / 2) > 0) { // условие выполнения
                a = (a + b) / 2; // проводим сближение
                b = b;
            }
            if (f(b) * f((a + b) / 2) > 0) { // условие выполнения
                b = (a + b) / 2; // проводим сближение
                a = a;
            }
            counter++;
        }
    } else { // если метод невыполним - выводим 0, завершая ошибку
        return 0;
    }
    return (a + b) / 2; // выводим значение корня
}
```

Аргументы функции соответствуют аргументам функции метода итераций, но значение производной в данном случае не требуется. Аналогично методу итераций заведем счетчик итераций, чтобы избежать бесконечного цикла. Функция также полностью соответствует описанию данного метода ранее.

В функции int main объявим и инициализируем значения начал и концов отрезков, значение машинного эпсилона и будем выводить значения корней уравнения в виде таблицы:

```
int main (int argc, const char * argv[])
{
    double EPS = FEps(); // получаем значение машинного эпсилона
    double a_1 = 0; // начало отрезка для первой функции
    double b_1 = 0.85; // конец отрезка для первой функции
    double a_2 = 1; // начало отрезка для второй функции
    double b_2 = 2; // конец отрезка для второй функции
    printf("-------------------------------------------------------------------------------
\n"); // выводим таблицей значения корня для первой функции
    printf("|                          Уравнение x - 1 / (3 + sin(3.6x) = 0                 |\n");
    printf("|-------------------------------------------------------------------------------|\n");
    printf("|       Метод дихотомии     |      Метод Ньютона     |      Метод Итераций      |\n");
    printf("|-------------------------------------------------------------------------------|\n");
    printf("|       %.12f      |      %.12f    |      %.12f       |\n", Dikhotomy(a_1, b_1, EPS,
Func1), Newton (a_1, b_1, EPS, Func1, Func1_diff, Func1_diff2), Iterations(a_1, b_1, EPS,
Root_Func1, Root_Func1_diff));
    printf("|-------------------------------------------------------------------------------|\n");
    printf("\n");
    printf("-------------------------------------------------------------------------------
\n"); // выводим таблицей значения корня для второй функции
    printf("|                          Уравнение 0,1x^2 - xlnx = 0                          |\n");
    printf("|-------------------------------------------------------------------------------|\n");
    printf("|       Метод дихотомии     |      Метод Ньютона     |      Метод Итераций      |\n");
    printf("|-------------------------------------------------------------------------------|\n");
    printf("|       %.12f      |      %.12f    |      %.12f       |\n", Dikhotomy(a_2, b_2, EPS,
Func2), Newton (a_2, b_2, EPS, Func2, Func2_diff, Func2_diff2), Iterations(a_2, b_2, EPS,
Root_Func2, Root_Func2_diff));
    printf("|-------------------------------------------------------------------------------|\n");

    return 0;
}
```

Также внутри функции main добавим проверку на выполнимость методов, следующей конструкцией:

```
if (Iterations(a_1, b_1, EPS, Root_Func1, Root_Func1_diff) == 0) { // если
какой-то метод неприменим - выведем уведомление об этом
     printf("Метод итераций неприменим\n");
}
if (Dikhotomy(a_1, b_1, EPS, Func1) == 0) {
     printf("Метод дихотомии неприменим\n");
}
if (Newton(a_1, b_1, EPS, Func1, Func1_diff, Func1_diff2) == 0) {
     printf("Метод Ньютона неприменим\n");
}
```

Если в таблице будет видно, что применение невозможно, то программа выведет уведомление об этом.

Список переменных

Название переменной	Описание
double a_1	Начало отрезка [a, b] первой функции
double b_1	Конец отрезка [a, b] первой функции

Название переменной	Описание
double a_2	Начало отрезка [a, b] второй функции
double b_2	Конец отрезка [a, b] второй функции
double EPS	Значение машинного эпсилона

Список функций

Название функции	Описание
double FEps	Функция, вычисляющая значение машинного эпсилона
double Func1	Первая функция, вычисляющая значение обычным методом от аргумента
double Func2	Вторая функция, вычисляющая значение обычным методом от аргумента
double Func1_diff	Производная от первой функции
double Func2_diff	Производная от второй функции
double Func1_diff2	Вторая производная от первой функции
double Func2_diff2	Вторая производная от второй функции
double Root_Func1	Функция 1, вычленной корнем
double Root_Func2	Функция 2, вычленной корнем
double Root_Func1_diff	Производная от функции 1, вычленной корнем
double Root_Func2_diff	Производная от функции 2, вычленной корнем
Double Iterations	Функция, описывающая метод итераций
Double Newton	Функция, описывающая метод Ньютона
Double Dichotomy	Функция, описывающая метод Дихотомии

Общие сведения о программе

Программное обеспечение	Xcode
Аппаратное обеспечение	Macbook Pro 14` M1 Pro
Операционная система	macOS Montera
Язык программирования	C
Кол-во строк программы	67
Местонахождение файла	/Users/Ivan/Desktop/labs/kp4
Способ вызова	Компиляция в Xcode

Код программы

```c
/*
    Ляпин Иван Алексеевич
    М8О-104Б-22
    КП 4
    Вариант 19, 20
*/

#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <string.h>
#include <stdbool.h>

double FEps (void) // функция для вычисления машинного эпсилона
{
    double eps = 1.0;
    while (eps + 1.0 > 1.0) {
        eps /= 2.0;
    }
    return eps;
}
double Func1 (double x) // первая функция
{
    return x - 1 / (3 + sin(3.6*x)) ;
}
double Func2 (double x) // вторая функция
{
    return ((pow(x, 2) * 0.1) - (x * log(x)));
}
double Func1_diff (double x) // производная первой функции
{
    return 1 + (18 * cos(3.6 * x) / 5 * pow(3 + sin(3.6 * x), 2));
}
double Func2_diff (double x) // производная второй функции
{
    return (0.2 * x - log(x) - 1);
}
double Func1_diff2 (double x) // производная первой функции
{
    return (-3240 * sin(3.6 * x) - 1944 - 324 * sin(3.6 * x) * pow(cos(3.6 * x),
2)) / (25 * pow(3  + sin(3.6 * x), 4));
}
double Func2_diff2 (double x) // производная второй функции
{
    return 0.2 - (1 / x);
}
double Root_Func1 (double x) // функция 1, вычленная корнем
{
    return 1 / (3 + sin(3.6 * x));
}
double Root_Func2 (double x) // функция 2, вычленная корнем
{
    return (0.1 * pow(x, 2)) / log(x);
}
double Root_Func1_diff(double x) { // производная от функции 1, вычленной корнем
    return (-6 * cos(3.6 * x)) / (5 * pow(sin(3.6 * x), 2));
}
double Root_Func2_diff(double x) { // производная от функции 2, вычленной корнем
    return (2 * x * log(x) - x) / (10 * pow(log(x), 2));
}
```

```
double Iterations (double a, double b, double EPS, double f(double), double
f_diff(double)) // производим метод итераций
{
    double x; // заведем аргумент x
    int counterIter = 0; // заводим счетчик итераций, чтобы дать верхнюю границу
    x = (a + b) / 2; // присваиваем x значение середины отрезка

    if (fabs(f_diff(x)) < 1 && (fabs(f_diff(x / 1.5)) < 1)) { // проверяем на
сходимость метода, то есть производная от функции, вычленной корнем должна быть
< 1 по модулю
        while (fabs(f(x) - x) > EPS && counterIter < 100) { // заводим цикл,
пока абсолютная разница не будет < некторого машинного эпсилона или кол-во
итераций = 100
            x = f(x);
            counterIter++;
        }
    } else { // если метод невыполним, то завершаем функцию

        return 0;

    }
    return x; // выводим значение корня
}
double Newton (double a, double b, double EPS, double f(double), double
f_2(double), double f_2_diff (double)) // создаем функцию, вычисл. корень по
Ньютону
{
    double x;
    x = (a + b) / 2; // значение середины отрезка
    if (fabs(f(x) * f_2_diff(x)) <= pow(f_2(x), 2)) { // проверка на выполнение
метода
        while (fabs(f(x) / f_2(x)) > EPS) { // цикл, пока значение деление не
стало меньшим или равным eps
            x -= f(x) / f_2(x); // вычитаем из x
        }
    } else { // выводим 0 , если метод неприменим

        return 0;

    }
    return x; // выводим корень уравнения
}
double Dikhotomy (double a, double b, double EPS, double f(double))
{
    double x;
    int counter = 0; // заводим счетчик итераций цикла, чтобы определить
верхнюю границу
    if (f(a) * f(b) < 0) { // проверка выполения метода
        while (fabs(a - b) > EPS && counter < 100) { // цикл пока сближение
небудет меньшим или равным eps
            x = (a + b) / 2; // придаем аргументы значение середины отрезка в
момент итерации
            if (f(a) * f((a + b) / 2) > 0) { // условие выполнения
                a = (a + b) / 2; // проводим сближение
                b = b;
            }
            if (f(b) * f((a + b) / 2) > 0) { // условие выполнения
                b = (a + b) / 2; // проводим сближение
                a = a;
            }
            counter++;
        }
    } else { // если метод невыполним - выводим 0, завершая ошибку

        return 0;
```

```
    }
    return (a + b) / 2; // выводим значение корня
}
int main (int argc, const char * argv[])
{
    double EPS = FEps(); // получаем значение машинного эпсилона
    double a_1 = 0; // начало отрезка для первой функции
    double b_1 = 0.85; // конец отрезка для первой функции
    double a_2 = 1; // начало отрезка для второй функции
    double b_2 = 2; // конец отрезка для второй функции

printf("-------------------------------------------------------------------------
----------\n"); // выводим таблицей значения корня для первой функции
    printf("|                            Уравнение x - 1 / (3 + sin(3.6x) = 0
|\n");

printf("|-----------------------------------------------------------------------
---------|\n");
    printf("|       Метод дихотомии     |      Метод Ньютона     |      Метод
Итераций       |\n");

printf("|-----------------------------------------------------------------------
---------|\n");
    printf("|       %.12f      |      %.12f    |      %.12f       |\n",
Dikhotomy(a_1, b_1, EPS, Func1), Newton (a_1, b_1, EPS, Func1, Func1_diff,
Func1_diff2), Iterations(a_1, b_1, EPS, Root_Func1, Root_Func1_diff));

printf("|-----------------------------------------------------------------------
---------|\n");
    printf("\n");
    if (Iterations(a_1, b_1, EPS, Root_Func1, Root_Func1_diff) == 0) { // если
какой-то метод неприменим - выведем уведомление об этом
        printf("Метод итераций неприменим\n");
    }
    if (Dikhotomy(a_1, b_1, EPS, Func1) == 0) {
        printf("Метод дихотомии неприменим\n");
    }
    if (Newton(a_1, b_1, EPS, Func1, Func1_diff, Func1_diff2) == 0) {
        printf("Метод Ньютона неприменим\n");
    }

printf("-------------------------------------------------------------------------
----------\n"); // выводим таблицей значения корня для второй функции
    printf("|                            Уравнение 0,1x^2 - xlnx = 0
|\n");

printf("|-----------------------------------------------------------------------
---------|\n");
    printf("|       Метод дихотомии     |      Метод Ньютона     |      Метод
Итераций       |\n");

printf("|-----------------------------------------------------------------------
---------|\n");
    printf("|       %.12f      |      %.12f    |      %.12f       |\n",
Dikhotomy(a_2, b_2, EPS, Func2), Newton (a_2, b_2, EPS, Func2, Func2_diff,
Func2_diff2), Iterations(a_2, b_2, EPS, Root_Func2, Root_Func2_diff));

printf("|-----------------------------------------------------------------------
---------|\n");
    if (Iterations(a_2, b_2, EPS, Root_Func2, Root_Func2_diff) == 0) { // если
какой-то метод неприменим - выведем уведомление об этом
        printf("Метод итераций неприменим\n");
    }
    if (Dikhotomy(a_2, b_2, EPS, Func2) == 0) {
        printf("Метод дихотомии неприменим\n");
    }
```

```
    if (Newton(a_2, b_2, EPS, Func2, Func2_diff, Func2_diff2) == 0) {
        printf("Метод Ньютона неприменим\n");
    }
    return 0;
}
```

Результаты работы программы

```
-------------------------------------------------------------------------------
|                        Уравнение x - 1 / (3 + sin(3.6x) = 0                 |
|-----------------------------------------------------------------------------|
|       Метод дихотомии      |      Метод Ньютона      |      Метод Итераций      |
|-----------------------------------------------------------------------------|
|       0.262441465119       |      0.262441465119     |      0.262441465119      |
|-----------------------------------------------------------------------------|

-------------------------------------------------------------------------------
|                        Уравнение 0,1x^2 - xlnx = 0                          |
|-----------------------------------------------------------------------------|
|       Метод дихотомии      |      Метод Ньютона      |      Метод Итераций      |
|-----------------------------------------------------------------------------|
|       1.118325591590       |      1.118325591590     |      0.000000000000      |
|-----------------------------------------------------------------------------|
Метод итераций неприменим
Program ended with exit code: 0
```

Вывод

Исходя из результатов программы, можно выделить факт правильной её работы и правильности вычисляемых методов решения уравнений, таким образом данные методы являются общезначимыми и актуальными в работе за компьютером. Например можно запрограммировать калькулятор или любое подобное приложение, вычисляющее какие-то математические выражения, тем самым облегчить себе последующие вычисления. Например если программист работает в компании и ему нужно написать калькулятор, решающий уравнения, то применяя опыт работы данного задания курсового проекта, программисту будет проще понять с чего начинать. Также в данном задании я смог воспользоваться знаниями из курса линейной алгебры и аналитической геометрии, так как описанные методы являются неотъемлемой частью данного раздела математики. Тем самым найти корни уравнения можно и вручную, но используя навыки программирования, получится сэкономить значительное количество времени.

Справочник

1 - https://ru.wikipedia.org/wiki/Машинный_ноль

2 - https://ru.wikipedia.org/wiki/Метод_итерации

3 - https://ru.wikipedia.org/wiki/Метод_Ньютона

4 - https://ru.wikipedia.org/wiki/Дихотомия