Московский Авиационный Институт

(Национальный исследовательский университет)

**Факультет информационных технологий и прикладной математики**

**Кафедра №806 Вычислительная математика и программирование**

# Курсовой проект

**по курсам  
«Фундаментальная информатика», «Архитектура компьютера и информационных систем»  
I семестр**

**Задание 3**

**Вещественный тип. Приближенные вычисления.**

**Табулирование функций.**

Студент: Ибрагимов Д. М.

Группа: М8О-108Б-22

Руководитель: Сахарин Н. С.

Оценка:

Дата:

Подпись преподавателя:

СОДЕРЖАНИЕ

Задание 3

Вариант 3

Общий метод решения 4

Оборудование 5

Программное обеспечение 5

Функциональное назначение 6

Описание логической структуры 6

Описание переменных, констант и подпрограмм 6

Протокол 9

Входные данные 12

Выходные данные 12

Заключение 13

## Задание

Составить программу на языке Си с процедурами решения трансцендентных алгебраических уравнений различными способами (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.

## Вариант:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Уравнение | Отрезок, содержащий корень | Базовый метод | Приближенное значение корня |
| 9 |  | [2, 3] | итераций | 2.0267 |
| 10 |  | [0.4, 1] | Ньютона | 0.6533 |

## Общий метод решения

Вычисление приближенного значений функций при помощи метода дихотомии, метода итераций и метода Ньютона.

Рассматривается уравнение вида *F(x)= 0*. Предполагается, что функция *F(x)* достаточно гладкая, монотонная на этом отрезке и существует единственный корень уравнения *x\* ∈ [a, b]*. на отрезке *[a, b]* ищется приближенное решение x с точностью *𝜀*, т.е. такое, что *|x - x\*| < 𝜀*.

Метод дихотомии - деление отрезка пополам с учётом того, что знак функции на концах отрезка должен быть разным: *F(a) \* F(b) < 0*. До тех пор, пока длина отрезка не будет меньше значения *𝜀*, процесс деления будет выполняться. Приближенное значение корня к моменту окончания итерационного процесса будет находиться примерно в середине заданного отрезка.

Метод итераций заключается в замене исходного уравнения *F(x) = 0* уравнением *f(x) = x*. Начальным приближенным значением корня является середина заданного отрезка *x(0) = (a + b)/2*. Итерационный процесс имеет вид: *x(k+1) = f(x(k))*. Процесс выполняется пока *|x (k+1) - x(k-1)| < 𝜀*

Метод Ньютона - частный случай метода итераций. Итерационный процесс представляет собой: *x(k+1) = x(k) – F(x(k))/F’(x(k))*.

**Оборудование**

|  |  |
| --- | --- |
| Процессор | AMD Ryzen 5 5600H (12) @ 3.600GHz |
| ОП | 32GiB 3200 MHz LPDDR4 |
| SSD | 512 GiB |
| Монитор | 27-дюймовый (1920 х 1080) |
| Графика | AMD Radeon™ RX 6600M 2177 MHz 8GiB GDDR6 |

Таблица 2 – оборудование

**Программное обеспечение**

|  |  |
| --- | --- |
| Операционная система семейства | Microsoft Windows 11 Pro |
| Компилятор | GNU Compiler Collection |
| Текстовый редактор | Visual Studio Code версия 1.73.0 |

Таблица 3 – программное обеспечение

## Функциональное назначение

Программа предназначена для высокоточного вычисления вещественного значения трансцедентных функций в алгебраической форме с использованием ряда Тейлора и при помощи встроенных программных функций библиотеки языка Си.

## Описание логической структуры

Программа получает на вход заданный отрезок, находит значение уравнения *F(x) = 0* различными численными методами и выводит полученный корень уравнения.

## Описание переменных, констант и подпрограмм

|  |  |  |
| --- | --- | --- |
| Функция | Входные аргументы | Описание |
| epsilon | double x | Функция для подсчета машинного ε |
| F9, F10 | double x | Вычисляет значение входной функции, подставляя значение x |
| F9\_x, F10\_x | double x | Функция, вычисляющая выраженный x |
| F9\_first\_derivative, F10\_first\_derivative | double x | Функция, вычисляющая первую производную от заданного уравнения |
| F9\_second\_derivative, F10\_second\_derivative | double x | Функция, вычисляющая вторую производную от заданного уравнения |
| F9\_x\_first\_derivative, F10\_x\_first\_derivative | double x | Функция, вычисляющая первую производную от уравнения, в котором выражен x |
| dichotomy | double F(double), double a, double b | Функция, вычисляющая значение уравнения F(x) = 0 методом дихотомии |
| iterations | double F\_x(double), double F\_x\_first\_derivative(double), double a, double b | Функция, вычисляющая значение уравнения F(x) = 0 методом итераций |
| newton | double F(double), double F\_first\_derivative(double), double F\_second\_derivative(double), double a, double b | Функция, вычисляющая значение уравнения F(x) = 0 методом Ньютона |

Таблица 1. Описание функций программы

|  |  |
| --- | --- |
| Переменная | Значение |
| double abs\_eps | Абсолютный эпсилон |
| double relative\_eps | Относительный эпсилон |
| double a, b | Границы отрезка |
| long double x | Значение аргумента функции |

Таблица 2. Описание переменных

## Протокол

Код программы:

#include <stdio.h>

#include <math.h>

typedef double dbl;

dbl epsilon() {

    dbl eps = 1.0;

    while (1 + eps / 2.0 != 1) eps /= 2.0;

    return eps;

}

dbl F9(dbl x) {

    return x\*x - log(1 + x) - 3;

}

dbl F9\_ot\_x(dbl x) {

    return (log(1 + x) + 3)/x;

}

dbl F9\_ot\_x\_first\_derivative(dbl x) {

    return (-2\*x - (1 + x)\*log(1+x) - 3)/(x\*x + x\*x\*x);

}

dbl F9\_first\_derivative(dbl x) {

    return 2\*x - 1/(1+x);

}

dbl F9\_second\_derivative(dbl x) {

    return (2\*x\*x + 4\*x + 3)/((1+x)\*(1+x));

}

dbl F10(dbl x) {

    return 2\*x\*sin(x) - cos(x);

}

dbl F10\_ot\_x(dbl x) {

    return 1/(2\*tan(x));

}

dbl F10\_ot\_x\_first\_derivative(dbl x) {

    return (-1/(sin(x)\*sin(x)))/2;

}

dbl F10\_first\_derivative(dbl x) {

    return 3\*sin(x) + 2\*x\*cos(x);

}

dbl F10\_second\_derivative(dbl x) {

    return 5\*cos(x) - 2\*x\*sin(x);

}

dbl dichotomy(dbl (\*F)(dbl), dbl a, dbl b, dbl relative\_eps, dbl abs\_eps) {

    dbl x = (a + b) / 2;

    if (F(a) \* F(b) >= 0){

        return NAN;

    }

    while (fabs(a - b) > fmax(relative\_eps \* fabs(x), abs\_eps)) {

        x = (a + b) / 2;

        if (F(x) \* F(a) < 0) {

            b = x;

        }

        else {

            a = x;

        }

    }

    return x;

}

dbl iterations(dbl (\*F\_x)(dbl), dbl (\*F\_x\_first\_derivative)(dbl), dbl a, dbl b, dbl relative\_eps, dbl abs\_eps) {

    dbl x = (a + b) / 2;

    if (fabs(F\_x\_first\_derivative(x)) >= 1) {

        printf("\n x - %lf; f -%lf debug printami\n", x, F\_x\_first\_derivative(x));

        return NAN;

    }

    while (fabs(F\_x(x) - x) >= fmax(relative\_eps \* fabs(x), abs\_eps)) x = F\_x(x);

    return x;

}

dbl newton(dbl (\*F)(dbl), dbl (\*F\_first\_derivative)(dbl), dbl (\*F\_second\_derivative)(dbl), dbl a, dbl b, dbl relative\_eps, dbl abs\_eps) {

    dbl x = (a + b / 2);

    if (fabs(F(x) \* F\_second\_derivative(x)) >= (F\_first\_derivative(x) \* F\_first\_derivative(x))) {

        return NAN;

    }

    while (fabs(F(x) / F\_first\_derivative(x)) > fmax(relative\_eps \* fabs(x), abs\_eps)) {

        x -= F(x) / F\_first\_derivative(x);

    }

    return x;

}

void result(dbl d, dbl i, dbl n) {

   if (d != NAN) printf("Dichotomy method: %.10f\n", d);

   else printf("The dichotomy method isn't suitable\n");

   if (i != NAN) printf("Iterations method: %.10f\n", i);

   else printf("The iterations method isn't suitable\n");

   if (n != NAN) printf("Newton's method: %.10f\n", n);

   else printf("The Newton's method isn't suitable\n");

int main() {

    dbl abs\_eps = epsilon();

    dbl relative\_eps = sqrt(abs\_eps);

    dbl a9 = 2, b9 = 3, a10 = 0.4, b10 = 1;

    dbl d9 = dichotomy(F9, a9, b9, relative\_eps, abs\_eps);

    dbl i9 = iterations(F9\_ot\_x, F9\_ot\_x\_first\_derivative, a9, b9, relative\_eps, abs\_eps);

    dbl n9 = newton(F9, F9\_first\_derivative, F9\_second\_derivative, a9, b9, relative\_eps, abs\_eps);

    printf("Machine epsilon for long double = %.16e\n", abs\_eps);

    printf(" x^2 - ln(1 + x) - 3 = 0 \n");

    result(d9, i9, n9);

    printf("\n\n");

    dbl d10 = dichotomy(F10, a10, b10, relative\_eps, abs\_eps);

    dbl i10 = iterations(F10\_ot\_x, F10\_ot\_x\_first\_derivative, a10, b10, relative\_eps, abs\_eps);

    dbl n10 = newton(F10, F10\_first\_derivative, F10\_second\_derivative, a10, b10, relative\_eps, abs\_eps);

    printf("Machine epsilon for long double = %.16e\n", abs\_eps);

    printf(" 2\*x\*sin(x) - cos(x) = 0 \n");

    result(d10, i10, n10);

    return 0;

}

## Входные данные

Отсутствуют.

**Выходные данные**

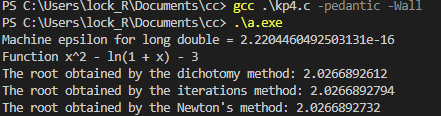
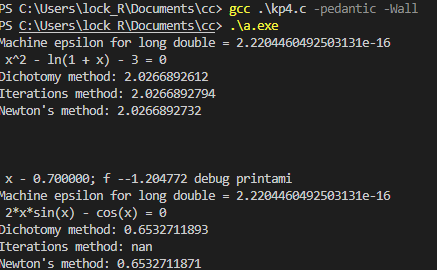
****

Рис.1 – выходные данные

## Заключение

## В результате выполнения данного курсового проекта были получены навыки работы с процедурами и функциями в качестве параметров. Было изучено вычисление машинного эпсилона и различных численных методов, таких как: метод дихотомии (половинного деления), метод итераций и метод Ньютона. Оценивая полученные данные, можно сказать, что каждый из методов неидеален, так как для поиска корня необходимо знать точные границы отрезка. Также при увеличении точности вычислений затраты по времени быстро увеличиваются.

## Благодаря использованию универсальных функций, которые принимают в качестве аргументов указатели на другие функции, удалось избежать дублирования кода.