**Московский авиационный институт**

(национальный исследовательский университет)

**Факультет № 8 «Прикладная математика и информатика»**

**Кафедра 806 «Вычислительная математика и программирование»**

КУРСОВОЙ ПРОЕКТ

по дисциплине «Вычислительные системы»

1 семестр

на тему “Процедуры и функции в качестве параметров”

|  |  |
| --- | --- |
| Студент: | Потапов Е.Д. |
| Группа: | М8О-109Б-22 |
| Преподаватель: | Сысоев М. |
| Подпись: |  |
| Оценка: |  |

Москва, 2020

**Постановка задачи**

Составить программы на языке Си с процедурами решения трансцендентных алгебраических уравнений различными численными методами (итераций, Ньютона и половинного деления — дихотомия). 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*.

Вариант 14

Уравнение

Отрезок [1, 2]

Приближённое значение корня 1.0769

Вариант 15

Уравнение

Отрезок [1, 2]

Приближённое значение корня 1.2388

**Теоретическая часть**

**Метод дихотомии (половинного деления)** — простейший численный метод для решения нелинейных уравнений вида f(x)=0. Он подразумевает только непрерывности функции на данном отрезке. Нa концах этого отрезка функция должна быть разных знаков: Из непрерывности следует, что на отрезке существует хотя бы один корень уравнения. Нужно найти значение xM середины отрезка Вычислим значение функции f(xM) в середине отрезка. Если значения функции в середине отрезка и на левой границе разные , то нужно переместить правую границу в середину отрезка, иначе левую границу в середину отрезка. Затем нужно повторить алгоритм начиная с вычисления значения xM  Алгоритм заканчивается тогда, когда f(xM)=0 либо xL=xR.

**Метод итераций** —ещё один простой численный метод решения уравнений. Метод основан на принципе сжимающего отображения, который применительно к численным методам в общем виде так же может называться методом простой итерации. Идея состоит в замене исходного уравнения f(x)=0 на эквивалентное ему x=φ(x). При чём должно выполнятся условие сходимости |φ’(x)|<1 на всём отрезке [a, b]. Итерации начинаются со значения xM середины отрезка. Однако φ(x) может выбрано неоднозначно. Сохраняет корни уравнения такое преобразование: Здесь λ0 – постоянная, которая не зависит от количества шагов. В данном случае мы возьмём , что приводит к простому методу одной касательной и имеет условие сходимости . Тогда итерационный процесс выглядит так: .

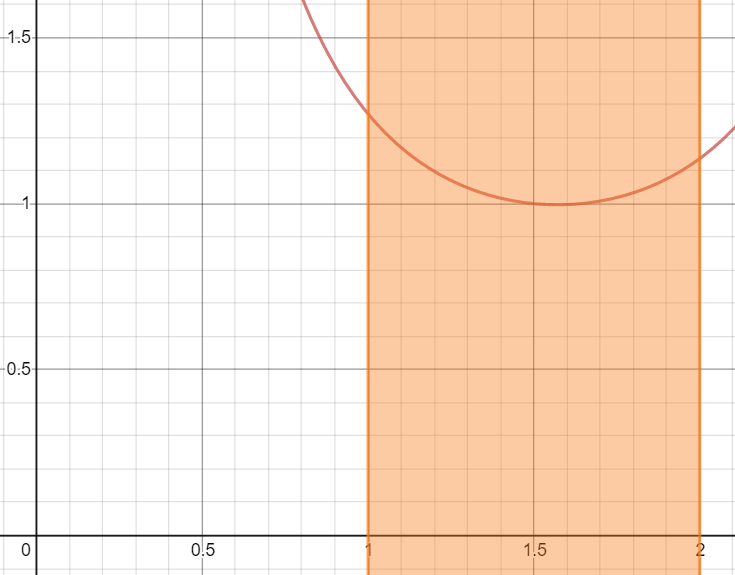
**Метод Ньютона** — итерационный численный метод нахождения корня заданной функции, который является частным случаем метода простых итераций. А именно за λ0 берётся значение производной в каждой новой точке. Тогда итерационный процесс имеет вид Условие окончания итераций и начальное значение абсолютно такие же, как и в методе итерации. Условие сходимости:

**ПРОВЕРКА НА УСЛОВИЕ СХОДИМОСТИ:**

Функции непрерывны на данном отрезке [1;2], следовательно, метод дихотомии подходит для решения двух уравнений.

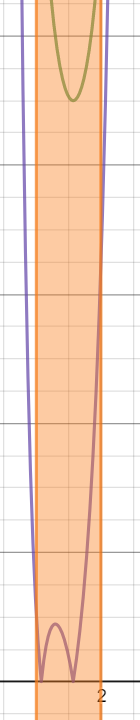
1) Возьмем производную от функции +1. λ=f'(xM)=3.010057

Проверим условия сходимости для метода итераций, построив графики производных сжимающих отображений:



На графике видно , что условие сходимости функция удовлетворяет методу итераций, потому что на интервале [1;2] функция .

Проверим условия сходимости для метода Ньютона, построив графики левых и правых частей неравенства .



- синий цвет

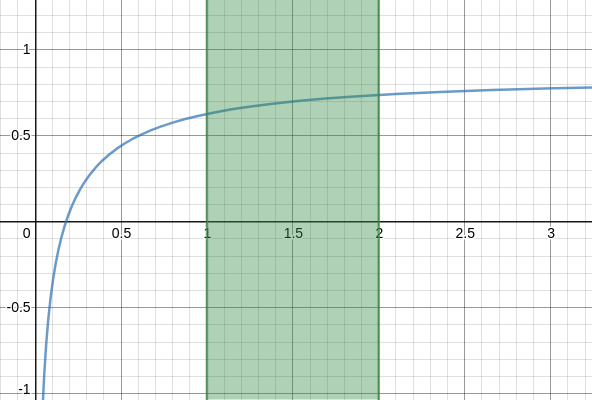
- зелёный цвет

Уравнение удовлетворяет условию сходимости метода Ньютона.

2) Возьмём производную от второй функции

λ=f'(xM)= -0.8367

Проверим условия сходимости для метода итераций, построив графики производных сжимающих отображений:

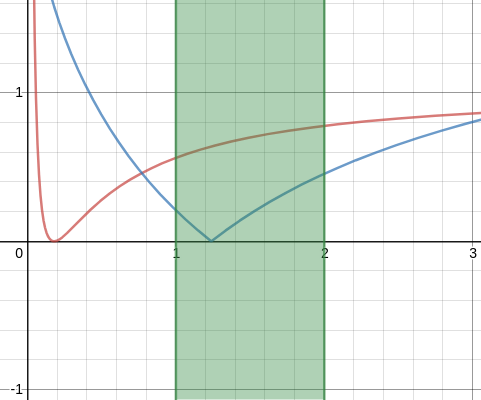


На графике видно , что условие сходимости функция удовлетворяет методу итераций, потому что на интервале [1;2] функция .

Проверим условия сходимости для метода Ньютона, построив графики левых и правых частей неравенства :

- синий цвет

- красный цвет

****

Из графика следует , что Метод Ньютона применим к этому уравнению.

**АЛГОРИТМ ВЫПОЛНЕНИЯ ПРГРАММЫ**

Первым делом программа считает машинный эпсилон. Вычисление эпсилона ЭВМ было расписано в 3-ей части КП. В этом курсовом проекте я дал эпсилон тип double для максимальной точности. Вся программа выполняется через функции , поэтому я создаю функции для расчёта уравнений по аругменту х:

double func(double x) {

return tan(x/2.0) - 1.0 / tan(x/2.0) + x;

}

double func\_1(double x) {

return 0.4 + atan(sqrt(x)) - x;

}

Чтобы эти функции можно было использовать в другой функции, то надо в качестве аргумента подать нужную функцию. Так для нахождения корня уравнения по методу дихотомии нужно брать среднее значение от суммы на концах отрезка , и если функция от левого края отрезка умноженная на функцию от среднего значения между краями отрезками:

double dhtm(double a, double b) {

double x = (a+b)/2;

double x\_prev = 1.0;

double eps = machine\_eps(1.0);

int n = 1;

while (func(x) != 0 && fabs(x - x\_prev) > eps) {

x\_prev = x;

if (sign(x) == sign(a)) {

a = x;

} else {

b = x;

}

x = (a+b)/2;

n++;

}

return x;

}

Для метода итераций надо было посчитать производную от уравнения и не зависящую от шага переменную, которая равняется 1/f“(x). Алгоритм , по которому ищется корень, очень простой: . Алгоритм выполняется пока выражение не станет истинной. Для этой функции тоже придётся использоваться в качестве аргумента функцию с параметром:

double itter(double a, double b) {

double x = 1.0;

double x\_prev = (a+b)/2;

double eps = machine\_eps(1.0);

int n = 1;

if (fabs(f1(x\_prev)) >= 1 ) {

return 0;

}

while (true) {

x = func\_itter(x\_prev);

if (fabs(x - x\_prev) < eps) {

break;

} else {

x\_prev = x;

}

}

return x;

}

Для метода Ньютона я использую формулу из прошлого метода с одним различием: вместо используется отношение функции уравнение и функции производной. Тут уже нужны две функции-параметра:

double newtn(double a, double b) {

double x = a;

double x\_prev = (a+b)/2;

double eps = machine\_eps(1.0);

int n = 1;

char s[] = "method isn't available";

if (fabs(func(x)\*f2(x)) > (f1(x)\*f1(x))) {

return 0;

}

while (n < 100) {

n++;

x = func\_newtn(x\_prev);

if (fabs(x - x\_prev) < eps) {

break;

} else {

x\_prev = x;

}

}

return x;

}

**КОД ПРОГРАММЫ**

#include <stdio.h>

#include <math.h>

#include <stdbool.h>

#include <stdlib.h>

double func(double x);

double f1(double x);

double f2(double x);

double sign(double x);

double machine\_eps(double eps);

double dhtm(double a, double b);

double func\_itter(double x\_prev);

double itter(double a, double b);

double newtn(double a, double b);

double func\_newtn(double x\_prev);

// tg(x/2)-ctg(x/2)+x=0

// x = ctg(x/2)-tg(x/2)

//x = x\_prev - func(x\_prev)/f1(x\_prev);

//

// 0,4+arctg(sqrt(x))-x

// x = 0,4+arctg(sqrt(x))

double func\_1(double x);

double f1\_1(double x);

double f2\_1(double x);

double dhtm\_1(double a, double b);

double sign\_1(double x);

double func\_itter\_1(double x\_prev);

double itter\_1(double a, double b);

double newtn\_1(double a, double b);

double func\_newtn\_1(double x\_prev);

void write();

int main() {

write();

}

void write() {

printf("dichotomy \t iteration\t Newton's\n");

printf("%.4lf \t\t", dhtm(1.0, 2.0));

printf("%.4lf \t\t", itter(1.0, 2.0));

printf("%.4lf", newtn(1.0, 2.0));

printf("\n");

printf("%.4lf \t\t", dhtm\_1(1.0, 2.0));

printf("%.4lf \t\t", itter\_1(1.0, 2.0));

printf("%.4lf", newtn\_1(1.0, 2.0));

}

// Hewthon 1 func

double newtn(double a, double b) {

double x = a;

double x\_prev = (a+b)/2;

double eps = machine\_eps(1.0);

int n = 1;

char s[] = "method isn't available";

if (fabs(func(x)\*f2(x)) > (f1(x)\*f1(x))) {

return 0;

}

while (n < 100) {

n++;

x = func\_newtn(x\_prev);

if (fabs(x - x\_prev) < eps) {

break;

} else {

x\_prev = x;

}

}

return x;

}

//Newthon 2 func

double newtn\_1(double a, double b) {

double x = a;

double x\_prev = (a+b)/2;

double eps = machine\_eps(1.0);

int n = 1;

if (fabs(func\_1(x)\*f2\_1(x)) > (f1\_1(x)\*f1\_1(x))) {

return 0;

}

while (true) {

n++;

x = func\_newtn\_1(x\_prev);

if (fabs(x - x\_prev) < eps) {

break;

} else {

x\_prev = x;

}

}

return x;

}

double func\_newtn(double x\_prev) {

double x = x\_prev - func(x\_prev)/f1(x\_prev);

return x;

}

double func\_newtn\_1(double x\_prev) {

double x = x\_prev - func\_1(x\_prev)/f1\_1(x\_prev);

return x;

}

//iter 1 func

double itter(double a, double b) {

double x = 1.0;

double x\_prev = (a+b)/2;

double eps = machine\_eps(1.0);

int n = 1;

if (fabs(f1(x\_prev)) >= 1 ) {

return 0;

}

while (true) {

x = func\_itter(x\_prev);

if (fabs(x - x\_prev) < eps) {

break;

} else {

x\_prev = x;

}

}

return x;

}

//inet 2 func

double itter\_1(double a, double b) {

double x = a;

double x\_prev = (a+b)/2;

double eps = machine\_eps(1.0);

int n = 1;

if (fabs(f1\_1(x\_prev)) >= 1 ) {

return 0;

}

while (true) {

x = func\_itter\_1(x\_prev);

if (fabs(x - x\_prev) < eps) {

break;

} else {

x\_prev = x;

}

}

return x;

}

double func\_itter(double x\_prev) {

double x = cos(x\_prev/2)/sin(x\_prev/2)-tan(x\_prev/2);

return x;

}

double func\_itter\_1(double x\_prev) {

double x = 0.4+atan(sqrt(x));

return x;

}

//dihotomia 1 func

double dhtm(double a, double b) {

double x = (a+b)/2;

double x\_prev = 1.0;

double eps = machine\_eps(1.0);

int n = 1;

while (func(x) != 0 && fabs(x - x\_prev) > eps) {

x\_prev = x;

if (sign(x) == sign(a)) {

a = x;

} else {

b = x;

}

x = (a+b)/2;

n++;

}

return x;

}

//dihotomia 2 func

double dhtm\_1(double a, double b) {

double x = (a+b)/2;

double x\_prev = a;

double eps = machine\_eps(1.0);

int n = 1;

while (func\_1(x) != 0 && fabs(x - x\_prev) > eps) {

x\_prev = x;

if (sign\_1(x) == sign\_1(a)) {

a = x;

} else {

b = x;

}

x = (a+b)/2;

n++;

}

return x;

}

double machine\_eps(double eps) {

while (1.0 + eps/2.0 > 1)

eps /= 2.0;

return eps;

}

double sign(double x) {

if ((tan(x/2)-cos(x/2)/sin(x/2)+x)> 0) {

return true;

} else if ((tan(x/2)-cos(x/2)/sin(x/2)+x)< 0) {

return false;

} else return false;

}

double sign\_1(double x) {

if ((0.4+atan(sqrt(x))-x) > 0) {

return true;

} else if ((0.4+atan(sqrt(x))-x) < 0) {

return false;

} else return false;

}

double func(double x) {

return -1/tan(x/2)+tan(x/2)+x;

}

double func\_1(double x) {

return 0.4+atan(sqrt(x))-x;

}

double f1(double x) {

return 1+1/(2\*pow(cos(x/2), 2))+1/(2\*pow(sin(x/2), 2));

}

double f1\_1(double x) {

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

}

double f2(double x) {

return -cos(x)/(2\*pow(cos(x/2), 3)\*pow(sin(x/2), 3));

}

double f2\_1(double x) {

return (3\*x+1)/((sqrt(x))\*pow((2\*sqrt(x)+2\*x\*sqrt(x)), 2));

}

**ПРОТОКОЛ**

**epsilon = 0.00000000000000022204**

**--------------------------------------**

**dichotomy iteration Newton's**

**1.0769 0.0000 1.0769**

**1.2388 1.2388 1.2388**

**--------------------------------------**

**Вывод**

В ходе курсового проекта я описал идеи и принципы трёх численных методов по нахождению корня уравнения: дихотомии, итераций и Ньютона. В начале КП я проверил условия сходимости данных уравнений методам и провел нужные вычисления для использования методов. Составил алгоритм решения уравнений, на основе которого составлена программа на языке Си.

**Список литературы**

1. Метод бисекции [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Метод_бисекции>
2. Метод простой итерации [Электронный ресурс] – URL: [https://ru.wikipedia.org/wiki/М](https://ru.wikipedia.org/wiki/Метод_бисекции)етод\_простой\_итерации
3. Метод Ньютона [Электронный ресурс] – URL: [https://ru.wikipedia.org/wiki/Метод\_](https://ru.wikipedia.org/wiki/Метод_бисекции)Ньютона