**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский Авиационный Институт»**

**(Национальный Исследовательский Университет)**

**Факультет №8 «Информационные технологии и прикладная математика»**

# Кафедра 806 «Прикладная математика и информатика»

Курсовой проект

по курсу «Вычислительные системы»

1 семестр

Задание 4

**Автор работы:**

студент 1 курса, группа М8О-103Б-20

Березнев Никита Вадимович

**Руководитель проекта:**

Севостьянов В.С.

**Дата сдачи:**

Москва, 2021

# Задача

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.*

# Решение

Всё решение сводится к тому, чтобы запрограммировать на языке Си три функции, ищущие корни заданных трансцендентных уравнений на заданных отрезках. Все функции, реализующие вычисление корней различными методами представляют собой различные итерационные процессы, в ходе которого последовательно вычисляется приближенные значения корня уравнения на заданном отрезке.

Основная проблема заключается в точности вычислений. Поэтому в решении данной задачи используется значение машинного эпсилон.

Как уже говорилось, машинное эпсилон — это максимальная относительная погрешность для конкретной процедуры округления, это такое числовое значение, меньше которого невозможно задавать относительную точность для любого алгоритма, возвращающего вещественные числа. В данной программе оно будет использоваться как признак того, что вычисленное значение уже достаточно точное, и дальнейшие вычисления уже не целесообразны т. к. дальнейшие числа являются одинаковыми с точки зрения машинной арифметики.

**Методы**

**1.** **Дихотомия или метод половинного деления.**

Очевидно, что 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 корни на заданном отрезке, если функция на нем непрерывна и её значения на концах отрезка имеют разные знаки. Последовательно будем делить отрезок пополам и выбирать ту половину, на которой лежит корень (то есть значения функции на концах которого имеют разные знаки), пока не будет достигнута заданная точность.

**2**. **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: x1=F(x0 ), x2=F(x1 ), ..., xn= F(xn-1 ). Процесс итераций сходится если |f’(x)| < 1 на отрезке, и увеличивая n, можно получить приближение, сколь угодно мало отличающееся от истинного значения корня.

**3.** **Метод Ньютона.**

Метод Ньютона можно применять, когда известно, что функция на заданном отрезке имеет один корень, причем первая и вторая производные на этом отрезке определены, непрерывны и сохраняют постоянные знаки. Возьмем x0 - середину отрезка [a,b] и проведем в точке P0 {x0 ,f(x0 )} графика функции касательную к кривой y=f(x) до пересечения с осью Ox. Абсциссу x1 точки пересечения можно взять в качестве приближенного значения корня. Проведя касательную через новую точку P1 {x1 ,f(x1 )} и находя точку ее пересечения с осью Ox, получим второе приближение корня x2 . Получившаяся последовательность сходится, если |f(x) · f ''(x)| < (f ’(x))2.

# Вычисление машинного эпсилон

Делим 1.0 пополам пока не получится так, что мы не можем отличить одно от другого. Если так случилось, значит, разница на предыдущем шаге и есть машинное эпсилон.

double eps(void)

{

double eps;

eps = 1;

while (1 + eps > 1) {

eps = eps / 2;

}

return eps;

}

**Описание функций**

double F\_Dix\_a\_1(double x) – Вычисляет значение функции 1 при заданном x

double F\_Dix\_ab\_1(double a, double b) – Вычисляет значение функции 1 при х, равном значению середины заданного отрезка [a; b] для функции

double Method\_Dixotomii\_1(double a, double b) – Вычисляет приближенное значение корня методом дихотомии для функции 1, заданной на отрезке [a; b]

double F\_Dix\_a\_2(double x) – Вычисляет значение функции 2 при заданном x

double F\_Dix\_ab\_2(double a, double b) – Вычисляет значение функции 2 при х, равном значению середины заданного отрезка [a; b] для функции

double Method\_Dixotomii\_2(double a, double b) – Вычисляет приближенное значение корня методом дихотомии для функции 2, заданной на отрезке [a; b]

double Method\_Iteracii\_1(double a, double b) – Вычисляет приближенное значение корня методом итераций для функции 1, заданной на отрезке [a; b]

double Method\_Iteracii\_1(double a, double b) – Вычисляет приближенное значение корня методом итераций для функции 1, заданной на отрезке [a; b]

double F\_x\_1(double x\_old) - Вычисляет значение функции 1 при заданном x

double F\_diff\_1\_x\_1(double x\_old) – Вычисляет значение производной первого порядка для функции 1

double F\_diff\_2\_x\_1(double x\_old) - Вычисляет значение производной второго порядка для функции 1

double Method\_Newton\_1(double a, double b) - Вычисляет приближенное значение корня методом Ньютона для функции 1, заданной на отрезке [a; b]

double F\_x\_2(double x\_old) - Вычисляет значение функции 2 при заданном x

double F\_diff\_1\_x\_2(double x\_old) – Вычисляет значение производной первого порядка для функции 2

double F\_diff\_2\_x\_2(double x\_old) - Вычисляет значение производной второго порядка для функции 2

double Method\_Newton\_2(double a, double b) - Вычисляет приближенное значение корня методом Ньютона для функции 2, заданной на отрезке [a; b]

int Line(void) – Функция создает разделительные линии в таблице в выводом результата

int main(void) – Главная функция

# Программа

#include <stdio.h>

#include <math.h>

double eps(void)

{

double eps;

eps = 1;

while (1 + eps > 1) {

eps = eps / 2;

}

return eps;

}

double mod(double a, double b)

{

double Out;

if (a > b) {

Out = a - b;

}

if (b >= a) {

Out = b - a;

}

return Out;

}

double F\_Dix\_a\_1(double x)

{

double Out;

Out = (3 \* x) - 14 + exp(x) - exp(-x);

return Out;

}

double F\_Dix\_ab\_1(double a, double b)

{

double Out;

double x;

x = (a + b) / 2;

Out = (3 \* x) - 14 + exp(x) - exp(-x);

return Out;

}

double Method\_Dixotomii\_1(double a, double b)

{

double z;

double x;

int n;

n = 0;

x = 0;

z = mod(a, b);

while ((z > eps()) && (n < 100)) {

if ((F\_Dix\_a\_1(a) \* F\_Dix\_ab\_1(a, b)) > 0) {

a = (a + b) / 2;

b = b;

}

if ((F\_Dix\_a\_1(b) \* F\_Dix\_ab\_1(a, b)) > 0) {

a = a;

b = (a + b) / 2;

}

z = mod(a, b);

n = n + 1;

x = (a + b) / 2;

}

return x;

}

double F\_Dix\_a\_2(double a)

{

double Out;

Out = sqrt(1 - a) - tan(a);

return Out;

}

double F\_Dix\_ab\_2(double a, double b)

{

double Out;

double x;

x = (a + b) / 2;

Out = sqrt(1 - x) - tan(x);

return Out;

}

double Method\_Dixotomii\_2(double a, double b)

{

int n;

double z;

double x;

n = 0;

x = 0;

z = mod(a, b);

while (z > eps() && n < 100) {

if ((F\_Dix\_a\_2(a) \* F\_Dix\_ab\_2(a, b)) > 0) {

a = (a + b) / 2;

b = b;

}

if ((F\_Dix\_a\_2(b) \* F\_Dix\_ab\_2(a, b)) > 0) {

a = a;

b = (a + b) / 2;

}

z = mod(a, b);

n = n + 1;

x = (a + b) / 2;

}

return x;

}

double Method\_Iteracii\_1(double a, double b)

{

double x\_old;

double x;

double z;

int n;

n = 0;

x\_old = (a + b) / 2;

z = -((3 \* exp(x\_old) + 1) / ((14 \* exp(x\_old)) - (3 \* x\_old \* exp(x\_old) + 1)));

if (z < 0) {

z = -1 \* z;

}

if (z < 1) {

x = log(14 - 3 \* x\_old + exp(-x\_old));

z = mod(x, x\_old);

while (z > eps() && n < 100) {

x\_old = x;

x = log(14 - 3 \* x\_old + exp(-x\_old));

z = mod(x, x\_old);

n = n + 1;

}

return x;

}

}

double Method\_Iteracii\_2(double a, double b)

{

double x\_old;

double x;

double z;

int n;

n = 0;

x\_old = (a + b) / 2;

z = -(1 / (2 \* sqrt(1 - x\_old) \* (2 - x\_old)));

if (z < 0) {

z = -1 \* z;

}

if (z < 1) {

x = atan(sqrt(1 - x\_old));

z = mod(x, x\_old);

while (z > eps() && n < 100) {

x\_old = x;

x = atan(sqrt(1 - x\_old));

z = mod(x, x\_old);

n = n + 1;

}

return x;

}

}

double F\_x\_1(double x\_old)

{

double F\_x;

F\_x = 3 \* x\_old - 14 + exp(x\_old) - exp(-x\_old);

return F\_x;

}

double F\_diff\_1\_x\_1(double x\_old)

{

double F\_diff\_x;

F\_diff\_x = (3 \* exp(x\_old) + exp(2 \* x\_old) + 1) / exp(x\_old);

return F\_diff\_x;

}

double F\_diff\_2\_x\_1(double x\_old)

{

double F\_x;

F\_x = (exp(2 \* x\_old) - 1) / exp(x\_old);

return F\_x;

}

double Method\_Newton\_1(double a, double b)

{

double x;

double x\_old;

double z;

int n;

n = 0;

x\_old = (a + b) / 2;

if (mod(F\_x\_1(x\_old), F\_diff\_2\_x\_1(x\_old)) < pow(F\_diff\_1\_x\_1(x\_old), 2)) {

x = x\_old - (F\_x\_1(x\_old) / F\_diff\_1\_x\_1(x\_old));

z = mod(x, x\_old);

while (z > eps() && n < 100) {

x\_old = x;

x = x\_old - (F\_x\_1(x\_old) / F\_diff\_1\_x\_1(x\_old));

z = mod(x, x\_old);

n = n + 1;

}

return x;

}

}

double F\_x\_2(double x\_old)

{

double F\_x;

F\_x = sqrt(1 - x\_old) - tan(x\_old);

return F\_x;

}

double F\_diff\_1\_x\_2(double x\_old)

{

double F\_x;

F\_x = - (1.0 / pow(cos(x\_old), 2)) - (1.0 / 2 \* sqrt(1 - x\_old));

return F\_x;

}

double F\_diff\_2\_x\_2(double x\_old)

{

double F\_x;

F\_x = - (2.0 \* sin(x\_old) / pow(cos(x\_old), 3)) - (1.0 / 4 \* sqrt(1 - x\_old) \* (1 - x\_old));

return F\_x;

}

double Method\_Newton\_2(double a, double b)

{

double x;

double x\_old;

double z;

int n;

n = 0;

x\_old = (a + b) / 2;

if (mod(F\_x\_2(x\_old), F\_diff\_2\_x\_2(x\_old)) < pow(F\_diff\_1\_x\_2(x\_old), 2)) {

x = x\_old - (F\_x\_2(x\_old) / F\_diff\_1\_x\_2(x\_old));

z = mod(x, x\_old);

while (z > eps() && n < 100) {

x\_old = x;

x = x\_old - (F\_x\_2(x\_old) / F\_diff\_1\_x\_2(x\_old));

z = mod(x, x\_old);

n = n + 1;

}

return x;

}

}

int Line(void)

{

int i;

printf("|");

for (i = 0; i < 152; i++) {

printf("-");

}

printf("|\n");

}

int main(void)

{

double a\_1;

double b\_1;

double a\_2;

double b\_2;

a\_1 = 1;

b\_1 = 3;

a\_2 = 0;

b\_2 = 1;

printf("Машинное эпсилон: %.20f\n\n", eps());

printf("%58s Function: 3x - 14 + exp(x) - exp(-x)\n", "");

Line();

printf("| Метод : %44s %14s| %42s %15s| %41s %15s|\n", "Метод Дихотомии:", "", "Метод Итераций:", "", "Метод Ньютона:", "");

Line();

printf("%9s %.42f | %.42f | %.42f |\n", "| Значение X:", Method\_Dixotomii\_1(a\_1, b\_1), Method\_Iteracii\_1(a\_1, b\_1), Method\_Newton\_1(a\_1, b\_1));

printf("%9s %.42f | %.42f | %.42f |\n", "| Проверка F:", F\_x\_1(Method\_Dixotomii\_1(a\_1, b\_1)), F\_x\_1(Method\_Iteracii\_1(a\_1, b\_1)), F\_x\_1(Method\_Newton\_1(a\_1, b\_1)));

Line();

printf("\n\n\n");

printf("%61s Function: sqrt(1 - x) - tan(x)\n", "");

Line();

printf("| Метод : %44s %14s| %42s %15s| %41s %15s|\n", "Метод Дихотомии:", "", "Метод Итераций:", "", "Метод Ньютона:", "");

Line();

printf("%9s %.42f | %.42f | %.42f |\n", "| Значение X:", Method\_Dixotomii\_2(a\_2, b\_2), Method\_Iteracii\_2(a\_2, b\_2), Method\_Newton\_2(a\_2, b\_2));

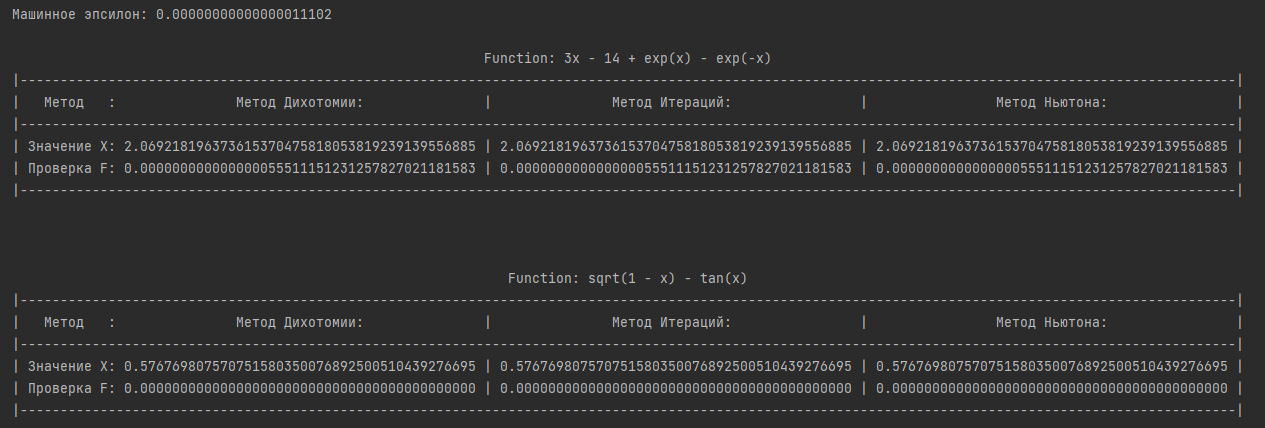
printf("%9s %.42f | %.42f | %.42f |\n", "| Проверка F:", F\_x\_2(Method\_Dixotomii\_2(a\_2, b\_2)), F\_x\_2(Method\_Iteracii\_2(a\_2, b\_2)), F\_x\_2(Method\_Newton\_2(a\_2, b\_2)));

Line();

return 0;

}

# Результат работы программы



# ЗАКЛЮЧЕНИЕ

После генерации таблицы значений заданной функции можно увидеть, что значения корней уравнений, вычисленных разными способами совпадают.

Также, как можно заметить, значения корней вычислены достаточно точно, о чем свидетельствуют значения функции с подставленным в нее корнем уравнения.

Существуют более совершенные способы решения задач, подобных данной, но вычисления корней с помощью данных методов могут быть использованы так же и в приложении простого калькулятора для решения подобных уравнений.