**Московский авиационный институт**

**(Национальный исследовательский университет)**

**Факультет прикладной математики и физики**

**Курсовой проект**

**по курсу**

**«Фундаментальная информатика»**

**I семестр**

**Задание 4**

|  |  |
| --- | --- |
| Студент:  Группа:  Руководитель:  Оценка:  Дата: | Хасанов Д. Р.  М8О-113Б-21  Довженко А. А. |

**Москва**

**2021г.**

**Задача**

Составить программу на языке Си с процедурами решения трансцендентных алгебраических уравнений различными численными методами (итераций, Ньютона и 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 вариантом с заданным номером. Если метод не применим дать математическое обоснование и графическую иллюстрацию.

Вариант 27(28)

; отрезок, содержащий корень [1; 3]

; отрезок, содержащий корень [1.2; 2]

**Решение**

Описание методов:

1. Метод половинного деления (дихотомии)

Очевидно, что если на отрезке существует корень уравнения, то значения функции на концах отрезка имеют разные знаки . Метод заключается в делении отрезка пополам и его сужения в два раза на каждом шаге итерационного процесса в зависимости от знака функции в середине отрезка.

Итерационный процесс строится следующим образом: за начальное приближение принимаются границы исходного отрезка . Далее, вычисления проводятся по формулам: , если ; или по формулам: , , если . Процесс повторяется до тех пор, пока не будет выполнено условие окончания . Приближенное значение корня к моменту окончания итерационного процесса получается следующим образом .

Функция непрерывна на отрезке и на концах его принимает значения разных знаков: ;

Функция непрерывна на отрезке и на концах его принимает значения разных знаков: ;

1. Метод итераций

Идея метода заключается в замене исходного уравнения уравнением вида . Достаточное условие сходимости метода: Это условие необходимо проверить перед началом решения задачи, так как функция может быть выбрана неоднозначно, причём в случае неверного выбора указанной функции метод расходится.

Начальное приближение корня: (середина исходного отрезка).

Итерационный процесс: .

Условие окончания: .

Приближённое значение корня: .

Для преобразования в используется уравнение вида , где – некоторая постоянная, знак которой совпадает со знаком производной в некоторой окрестности корня.

Условие сходимости метода итераций:

Найдем для функции

Условие сходимости выполняется.

Найдем для функции

Условие сходимости выполняется.

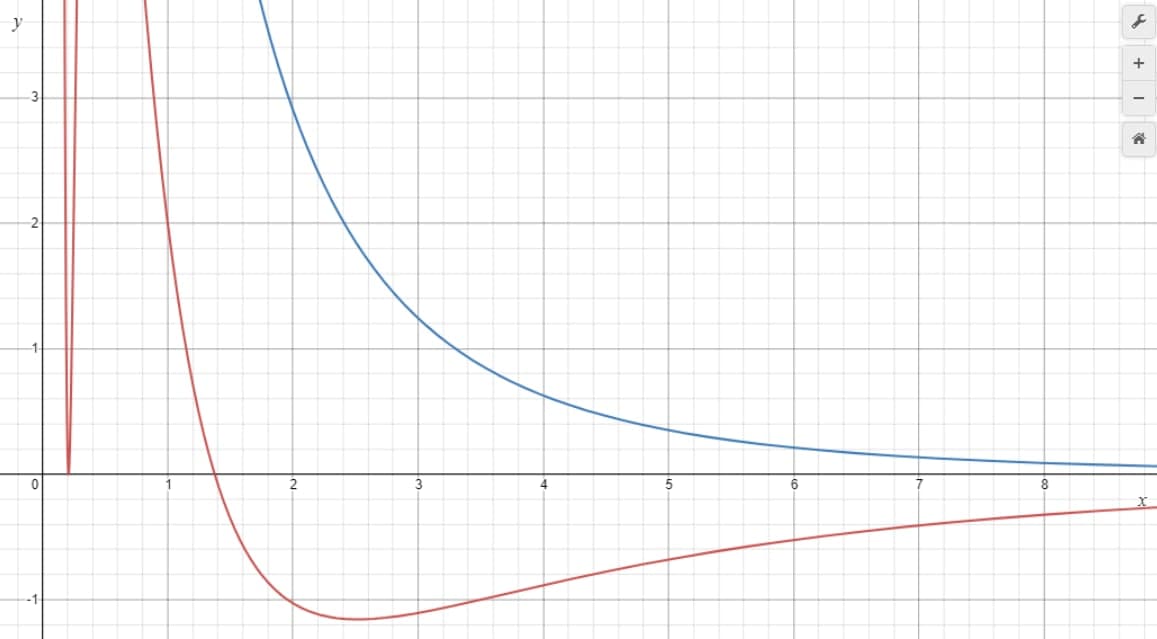
1. Метод Ньютона

Метод Ньютона — это итерационный численный метод нахождения корня(нуля) заданной функции. Метод был впервые предложен Исааком Ньютоном. Поиск решения осуществляется путём построения последовательных. Метод обладает квадратичной сходимостью и является частным случаем метода итераций. Условие сходимости метода: на отрезке . Итерационный процесс: .

Найдем производные данных функций:

Вариант 27:

| на отрезке

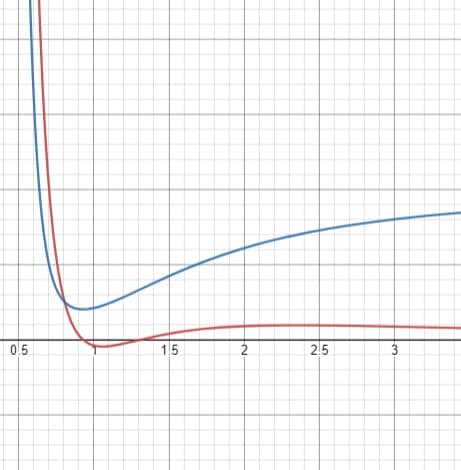


(синим - , коричневым - )

Условие сходимости выполняется.

Вариант 28:

| на отрезке



(синим - , красным - )

Условие сходимости выполняется.

**Листинг программного кода**

#include <stdio.h>

#include <math.h>

#include <float.h>

#include <locale.h>

double Function\_1(double x) {

return sin(log(x)) - cos(log(x)) + 2.0 \* log(x);

}

double Function\_iter\_1(double x) {

return x - 0.5 \* (sin(log(x)) - cos(log(x)) + 2.0 \* log(x));

}

double Function\_2(double x) {

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

}

double Function\_iter\_2(double x) {

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

}

double derivative\_1(double x) {

return (cos(log(x)) + sin(log(x)) + 2.0) / x;

}

double derivative\_2(double x) {

return 1.0 - cos(1.0 / x) / (x \* x);

}

double dichotomy(double a, double b, double(\*form)(double x), double eps)

{

double x = 0.0;

while (fabs(a - b) > eps) {

x = (a + b) / 2.0;

if (form(a) \* form(x) > 0) {

a = x;

} else {

b = x;

}

}

return x;

}

double newton(double a, double b, double(\*form)(double x), double(\*derivative)(double x), double eps)

{

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

double x\_last = 0;

while (fabs(x\_now - x\_last) > eps) {

x\_last = x\_now;

x\_now -= form(x\_now) / derivative(x\_now);

}

return x\_now;

}

double iteration(double a, double b, double(\*iter\_form)(double x), double eps)

{

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

double x\_last = 0.0;

while (fabs(x\_now - x\_last) > eps) {

x\_last = x\_now;

x\_now = iter\_form(x\_now);

}

return x\_now;

}

int main(void)

{

setlocale(LC\_ALL, "Rus");

double eps = 1.0;

while (1.0 + eps / 2.0 > 1) {

eps /= 2.0;

}

printf("\t|--------------------------------------------------------------------------------------|\n");

printf("\t|%20s| %20s| %20s| %21s\n", "Метод", "Дихотомии ", "Итераций ", "Ньютона |");

printf("\t|--------------------------------------------------------------------------------------|\n");

printf("\t|%20s| %20.16f| %20.16f| %20.16f|\n", "Вариант 27", dichotomy(1.0, 3.0, Function\_1, eps), iteration(1.0, 3.0, Function\_iter\_1, eps), newton(1.0, 3.0, Function\_1, derivative\_1, eps));

printf("\t|--------------------------------------------------------------------------------------|\n");

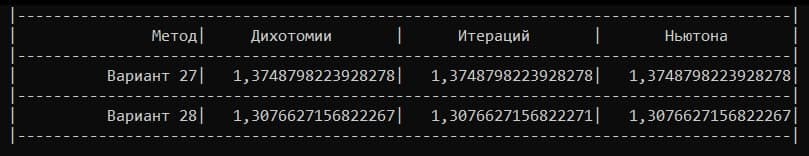
printf("\t|%20s| %20.16f| %20.16f| %20.16f|\n", "Вариант 28", dichotomy(1.2, 2.0, Function\_2, eps), iteration(1.2, 2.0, Function\_iter\_2, eps), newton(1.2, 2.0, Function\_2, derivative\_2, eps));

printf("\t|--------------------------------------------------------------------------------------|\n");

return 0;

}

**Результат работы программы**



**Заключение**

После генерации таблицы значений заданной функции можно увидеть, что значения корней уравнений, вычисленных разными способами, совпадают до 14-15 знака после запятой. Из-за того, что методы вычисления корней имеют различную «скорость» приближения к точному значению корня на данном отрезке, и один алгоритм достигает точки остановки быстрее другого, а также из-за того, что вещественные числа имеют диапазон представления в памяти компьютера, это неизбежно приводит к тому, что в вычислениях в окрестности границ данного диапазона возникают погрешности. Отсюда и небольшие различия в значениях корней.

Понятно, что 3 приведенных метода поиска корня функции не очень эффективны. В первую очередь из-за того, что нам нужно знать отрезок, на котором находится корень. Также метод итераций, например, может и не сработать для произвольной функции.