МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский Авиационный Институт»
(Национальный Исследовательский Университет)**

Институт№8: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Курсовой проект
по курсу фундаментальная информатика 1 семестра
Задание 3. Процедуры и функции в качестве параметров

Студент: Калюжный М.С.

Группа: М8О-108Б-22

Преподаватель: Сахарин Н.А.

Подпись:

Оценка:

Москва 2022

СОДЕРЖАНИЕ

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.

11 вариант задания:

Уравнение: $e^x + (1 + e^{2x})^{0.5} - 2 = 0$
Отрезок, содержащий корень: [-1; 0]
Базовый метод: дихотомии
Приближенное значение корня: -0.2877

12 вариант задания:

Уравнение: $\ln(x) - x + 1.8 = 0$
Отрезок, содержащий корень: [2; 3]
Базовый метод: итераций
Приближенное значение корня: 2.8459

ОБЩИЙ МЕТОД РЕШЕНИЯ

Каждое уравнение решаем 3 методами: итераций, дихотомии и Ньютона.

Метод дихотомии

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 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 $a^{(0)} = a, b^{(0)} = 0$. Далее вычисления проводятся по формулам: $a^{(k+1)} = \frac{a^k + b^k}{2}, b^{(k+1)} = b^k$, если $F(a^k) * F(\frac{a^k + b^k}{2}) > 0$; или по формулам:$a^{(k+1)} = a^k, b^{(k+1)} = \frac{a^k + b^k}{2}$, если $F(b^k) * F(\frac{a^k + b^k}{2}) > 0$.

До тех пор, пока не будет выполнено условие $|a^k - b^k| < \varepsilon$, процесс будет выполняться.

Приближенное значение корня к моменту окончания итерационного процесса получается следующим образом$x \approx \frac{(a^{(\text{конечное})} + b^{(\text{конечное})})}{2}$.

Метод Итераций

Идея метода заключается в замене исходного уравнения F(x) = 0 уравнением вида $x = f(x)$. Достаточное условие сходимости данного метода $|f'(x)| < 1, x \in [a, b]$. Это условие необходимо проверить перед началом решения задачи, так как функция f(x) может быть выбрана неоднозначно, причем в случае неверного выбора указанной функции, метод расходится. Достаточно неплохим выбором является функция $x - sign(dx(x)) * F(x)$ где $dx(x)$ – производная функции $F(x)$.

Начальное приближение корня: $x^0 = \frac{(a+b)}{2}$ (середина исходного отрезка).

Итерационный процесс: $x^{(k+1)} = f(x^k)$.

Условие окончания: $|x^k - x^{(k-1)}| < \varepsilon$.

Метод Ньютона

Метод Ньютона является частным случаем метода итераций.

Условие сходимости метода: $|F(x) * F''(x)| < (F'(x))^2$на отрезке $[a, b]$.

Итерационный процесс: $x^{(k+1)} = x^k - \frac{F(x^k)}{F\prime(x^k)}$.

Метод обладает квадратичной сходимостью. Модификацией метода является метод хорд и касательных. Также метод Ньютона может быть использован для решения задач оптимизации, в которых требуется определить ноль первой производной либо градиента в случае многомерного пространства.

В начале программы составляем функции для решения уравнения определённым методом и функции, служащие им аргументом. Для каждой формулы необходимо запрограммировать исходную функцию. Производную в точку можно принять за дифференциал от функции в данной точке, который считается по формуле $\lim_{\Delta x \to 0} \frac{f(x+\Delta x)-f(x)}{\Delta x}$. В теле основной программы делаем только вызов подпрограмм и вывод.

Для удобства и лаконичности в прогрмаме с помощью оператора typedef введен тип pFunc, который расшифровывается как pointer function и ld, который расшифровывается long double.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

ОС семейства: UNIX

Наименование: Ubuntu

Версия: Ubuntu 22.04 LTS

Интерпретатор команд: bash

Версия: 5.1-6ubuntu1

Компилятор: gcc

ОПИСАНИЕ ПЕРЕМЕННЫХ И ФУНКЦИЙ

Имя	Тип	Описание
epsilon	ld	Машинное эпсилон
a1	ld	Левая граница 1го отрезка
b1	ld	Правая граница 1го отрезка
a2	ld	Левая граница 2го отрезка
b2	ld	Права граница 2го отрезка

Таблица 1. Описание переменных

Название функции	Выходной тип	Входные параметры	Описание
machineeps	long double	отсутствуют	Функиця, считающая машинное эпсилон
dx1(dx2)	long double	pFunc f, long double x	Функция, считающая дифференциал в данной точке
sign	long double	long double x	Возвращает 1, если знак x положительный, -1, если отрицательный и 0, если x близок к нулю.
dichotomy1 (dichotomy2)	long double	pFunc f, long double a, long double b	Находит корень функции методом дихотомии
iteration1 (iteration2)	long double	pFunc f, long double a, long double b	Находит корень функции методом итераций
newton1 (newton2)	long double	pFunc f, long double a, long double b	Находит корень функции методом Ньютона
func1	long double	long double x	Считает значение 1 функции в данной точке
func2	long double	long double x	Считает значение 2 функции в данной точке

Таблица 2. Описание функций

ПРОТОКОЛ

```
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <assert.h>
#include <stdbool.h>

typedef double(*pFunc)(double);

double machineeps();
double dx1(double x);
double dx2(double x);
double sign(double x);
double dichotomy1(pFunc f, double a, double b);
double iteration1(pFunc f, double a, double b);
double newton1(pFunc f, double a, double b);
double dichotomy2(pFunc f, double a, double b);
double iteration2(pFunc f, double a, double b);
double newton2(pFunc f, double a, double b);
double func1(double x);
double func2(double x);


int main(){
   double a1 = -1.0, b1 = 0.0;
   double a2 = 2.0, b2 = 3.0;
   printf("dichotomy method result 11 variant: %f\n", dichotomy1(func1, a1,
b1));
```

```
    printf("iteration method result 11 variant: %f\n", iteration1(func1, a1, b1));
    printf("newton method result 11 variant: %f\n", newton1(func1, a1, b1));
    printf("\n");
    printf("dichotomy method result 12 variant: %lf\n", dichotomy2(func2, a2,
b2));
    printf("iteration method result 12 variant: %f\n", iteration2(func2, a2, b2));
    printf("newton method result 12 variant: %f\n", newton2(func2, a2, b2));
    return 0;
}

double machineeps() {
    double x = 1.0;
    double eps = 1.0;
    while (x + eps / 2.0 != x)
        eps /= 2.0;
    return eps * 2.0;
}

double dx1(double x){
    return expl(x) + 1 / (2 * sqrtl(1 + expl(2 * x))) * expl(2 * x) * 2;
}

double dx2(double x){
    return ((1 / x) - 1.0);
}

double sign(double x){
    double epsilon = machineeps();
    return x > epsilon ? 1.0 : x < -epsilon ? -1.0 : 0.0;
```

```
}

double dichotomy1(pFunc f, double a, double b){
    double epsilon = machineeps();
    double x = a + (b - a) / 2;
    while (fabsl(a-b) > epsilon){
        x = a + (b - a) / 2;
        if(f(a) * f(x) > 0.l) a = x;
        else b = x;
    }
    return x;
}

double iteration1(pFunc f, double a, double b){
    double epsilon = machineeps();
    double x = a + (b - a) / 2;
    while(fabsl(f(x)) > epsilon){
        x = x - f(x)*sign(dx1(x));
    }
    return x;
}

double newton1(pFunc f, double a, double b){
    double epsilon = machineeps();
    double x = a + (b - a) / 2;
    while(fabsl(f(x)/ dx1(x)) > epsilon){
        x = x - f(x)/dx1(x);
    }
    return x;
```

```
}

double dichotomy2(pFunc f, double a, double b) {
    double epsilon = machineeps();
    double x = a + (b - a) / 2;
    while (fabsl(a-b) > epsilon){
        x = a + (b - a) / 2;
        if(f(a) * f(x) > 0.l) a = x;
        else b = x;
    }
    return x;
}

double iteration2(pFunc f, double a, double b){
    double epsilon = machineeps();
    double x = a + (b - a) / 2;
    while(fabsl(f(x)) > epsilon){
        x = (x - f(x)*sign(dx2(x)));
    }
    return x;
}

double newton2(pFunc f, double a, double b){
    double epsilon = machineeps();
    double x = a + (b - a) / 2;
    while(fabsl(f(x)/ dx2(x)) > epsilon){
        x = x - f(x)/dx2(x);
    }
    return x;
```

```
}

double func1(double x){
   return expl(x) + sqrtl(1.0 + expl(2 * x)) - 2.0;
}

double func2(double x){
   return logl(x) - x + 1.8;
}
```

mimik@mimik-VirtualBox:~$ gcc kp41.c -lm -Wall -pedantic -std=c99 -o c4.out && ./c4.out

```
dichotomy method result 11 variant: -0.287682
iteration method result 11 variant: -0.287682
newton method result 11 variant: -0.287682

dichotomy method result 12 variant: 2.845868
iteration method result 12 variant: 2.845868
newton method result 12 variant: 2.845868
```

ВЫВОД

Данное задание курсового проекта показывает суть некоторых численных методов и их практическое применение для вычисления приближенного значения корней, однако ни один из вышеприведённых методов нельзя назвать идеальным, так как требуется заранее определённые границы поиска искомого корня и при увеличении параметра точности затраты по времени растут крайне быстро. Также были использованы универсальные функции, которые принимают в качестве аргументов указатели на другие функции. Это решение позволяет избежать дублирования кода.

ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1) Математический энциклопедический словарь. — М.: «Сов. энциклопедия », 1988. — С. 847. [20.01.23]

2) Волков Е. А. Численные методы. — М. : Физматлит, 2003. [20.01.23]

3) Максимов Ю. А. Алгоритмы линейного и дискретного программирования – М.: МИФИ, 1980. [20.01.23]