МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

“Московский Авиационный Институт

(Национальный Исследовательский Университет)”

Факультет информационных технологий и прикладной математики

Кафедра 806 “Вычислительная математика и программирование”

Курсовая работа

по курсу “Вычислительные системы”

1 семестр

Задание 4. Процедуры и функции в качестве параметров

Студент: Цирулев Н.В.

Группа: М8О-108Б-22,

№ по списку 22

Руководитель: Сахарин Н.А.

Дата: 09.01.23

Оценка:

Москва, 2023

ОГЛАВЛЕНИЕ

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.

22	$\arccos x - \sqrt{1 - 0{,}3x^3} = 0$	[0, 1]	итераций	0.5629

Рисунок 1: 22 вариант

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, служащим 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 $\lim\limits_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$. В теле основной программы делаем только вызов подпрограмм и вывод.

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

ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

- Язык и система программирования: GNU C
- Местонахождение файлов \\wsl.localhost\Ubuntu\home\hackerman\
- Способ вызова и загрузки:

  gcc cw.c -Wall -std=c18 -pedantic -lm

  ./a.out

ОПИСАНИЕ ПЕРЕМЕННЫХ И ФУНКЦИЙ

Таблица 1.А. Описание переменных

Имя	Тип	Описание
a	ld	Левая граница отрезка
b	ld	Права граница отрезка

Таблица 1.Б. Описание функций

Название функции	Выходной тип	Входные параметры	Описание
machineeps	ld	отсутствуют	Функция, считающая машинное эпсилон
dx	ld	pFunc f, ld x	Функция, считающая дифференциал в данной точке
sign	ld	ld x	Возвращает 1, если знак x положительный, -1, если отрицательный и 0, если x близок к нулю.
dht_method	ld	pFunc f, ld a, ld b	Находит корень функции методом дихотомии
iter_method	ld	pFunc f, ld a, ld b	Находит корень функции методом итераций
newton_method	ld	pFunc f, ld a, ld b	Находит корень функции методом Ньютона
func	ld	ld x	Считает значение функции в данной точке

ПРОТОКОЛ

hackerman@WARMACHINE_mini:~$ cat cp4.c

```
#include <stdio.h>
#include <math.h>

typedef long double ld;
typedef ld(*pFunc)(ld);

ld compute_epsilon(){
ld eps = 1;
while (1 < 1 + eps)
eps /= 2;
return eps;
}
ld dx(pFunc f, ld x) {
ld eps = compute_epsilon() * 2;
return (f(x + eps / 2) - f(x - eps / 2)) / eps;
}

ld dht_method(pFunc f, ld a, ld b) {
ld x = (a + (b - a) / 2);
while (fabsl(a - b) > fmax(sqrt(compute_epsilon()) * fabs(x), compute_epsilon())) {
x = (a + (b - a) / 2);
if(f(a) * f(x) > 0) a = x;
else b = x;
}
return x;
}

ld iter_method(pFunc f, ld a, ld b) {
ld x = (a + (b - a) / 2);
x = f(x);
if (fabsl(dx(f, x)) < 1) {
while(fabsl(f(x) - x) > fmax(sqrt(compute_epsilon()) * fabs(x), compute_epsilon())) {
x = f(x);
```

```
}
}
return x;
}

ld newton_method(pFunc f, ld a, ld b) {
ld x = (a + (b - a) / 2);
while(fabsl(f(x) / dx(f, x)) > fmax(sqrt(compute_epsilon()) * fabs(x), compute_epsilon())){
x = x - f(x) / (dx(f, x));
}
return x;
}

ld func1(ld x) {
return acosl(x) - sqrtl(1 - 0.3 * powl(x, 3));
}

ld funx1(ld x) {
return cos(sqrtl(1 - 0.3 * powl(x, 3)));
}

ld func2(ld x) {
return 3*x - 4*log(x) - 5;
}

ld funx2(ld x) {
return (4*log(x) + 5) / 3;
}

int main() {
ld a = 0, b = 1;
printf("dichotomy method result for func1: %Lf\n", dht_method(func1, a, b));
printf("iteration method result for func1: %Lf\n", iter_method(funx1, a, b));
printf("newton method result for func1: %Lf\n", newton_method(func1, a, b));
a = 2, b = 4;
printf("dichotomy method result for func2: %Lf\n", dht_method(func2, a, b));
printf("iteration method result for func2: %Lf\n", iter_method(funx2, a, b));
printf("newton method result for func2: %Lf\n", newton_method(func2, a, b));
```

```
}
hackerman@WARMACHINE_mini:~$ gcc cp4.c -Wall -std=c18 -pedantic -lm
hackerman@WARMACHINE_mini:~$ ./a.out
dichotomy method result for func1: 0.562926
iteration method result for func1: 0.562926
newton method result for func1: 0.562926
dichotomy method result for func2: 3.229959
iteration method result for func2: 3.229959
newton method result for func2: 3.229959
hackerman@WARMACHINE_mini:~$
```

ЗАКЛЮЧЕНИЕ

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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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

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