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»

Курсовая работа
по курсу «Фундаментальная
информатика» I семестр
Задание 4
«Процедуры и функции в качестве параметров»

Группа	М8О-109Б-22
Студент	Гиголаев А.А.
Преподаватель	Сысоев М.А.
Оценка	
Дата	

Москва, 2022

Постановка задачи

Составить программу на 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.

Вариант 22:

Функция:

$$\arccos x - \sqrt{1 - 0{,}3x^3} = 0$$

Отрезок содержащий корень: [0, 1]

Вариант 23:

Функция:

$$3x - 4\ln x - 5 = 0$$

Отрезок содержащий корень: [2, 4]

Теоретическая часть

Метод итераций

Идея метода заключается в замене исходного уравнения $F(x)=0$ уравнением вида $x=f(x)$.

Достаточное условие сходимости 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.

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

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

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

Приближенное значение корня: $x^{*}\approx x^{(конечное)}$.

Метод дихотомии (половинного деления)

Очевидно, что если на отрезке $[a,b]$ существует корень уравнения, то значения функции на концах отрезка имеют разные знаки: $F(a)\cdot F(b)<0$. 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.

Итерационный процесс строится следующим образом: за начальное приближение принимаются границы исходного отрезка $a^{(0)}=a$, $b^{(0)}=b$. Далее вычисления проводятся по формулам: $a^{(k+1)}=(a^{(k)}+b^{(k)})/2$, $b^{(k+1)}=b^{(k)}$, если $F(a^{(k)})\cdot F((a^{(k)}+b^{(k)})/2)>0$; или по формулам: $a^{(k+1)}=a^{(k)}$, $b^{(k+1)}=(a^{(k)}+b^{(k)})/2$, если $F(b^{(k)})\cdot F((a^{(k)}+b^{(k)})/2)>0$.

Процесс повторяется до тех пор, пока не будет выполнено условие окончания $\left|a^{(k)}-b^{(k)}\right|<\varepsilon$.

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

Численное дифференцирование — Так как возможности компьютера не позволяют проводить вычисления с бесконечно малыми, для расчетов будем брать просто очень маленькие значения. Так, для вычисления производной через предел возьмем prib равное 1e-6

Описание алгоритма

Делаем функцию для высчитывания корня методом дихотомии. После чего выводим его значение. Аналогично поступаем и для метода итераций, но для него отдельно выгодно будет сделать проверку.

Использованные в программе переменные

Название переменной	Тип переменной	Смысл переменной
prib	long double	Маленькое значение
step	long double	Шаг для проверки
a	long double	Левая граница отрезка
b	long double	Правая граница отрезка
x1	long double	Следующее значение x

Исходный код программы:

```
#include <stdio.h>
#include <float.h>
#include <math.h>
// Compute first deriative numerically
long double derive(long double (*f)(long double), long double x0) {
    long double h = 1e-6;
    long double res = (f(x0 + h) - f(x0)) / h;
    return res;
}
// Compute second deriative numerically
long double derive_2(long double (*f)(long double), long double x0) {
    long double h = 1e-6;
    long double res = (f(x0 + h) - 2*f(x0) + f(x0 - h)) / (h*h);
    return res;
}
long double func_23(long double x) {
 // e^x + ln(x) - 10x = 0
 return 3*x - 4 * logl(x) - 5;
}
long double func_23_deriative_1(long double x) {
 return 3 - 4/x;
}
long double func_23_deriative_2(long double x) {
 return 4/powl(x, 2);
}
// Returns 1 if method is covergent, otherwise 0
int is_newton_covergent(
    long double (*f)(long double),
    long double (*deriative_1)(long double),
    long double (*deriative_2)(long double),
    long double a,
    long double b) {
    int flag = 1;
    long double step = (b-a)/1000;
 for (long double x=a; x<=b; x+=step) {
    if (fabsl(f(x) * derive_2(f, x)) >= powl(derive(f, x), 2)) {
        flag = 0;
    }
    if (fabsl(f(x) * deriative_2(x)) >= powl(deriative_1(x), 2)) {
        flag = 0;
    }
 }
 return flag;
}
// Computes newton method using analytic deriative
long double newton_analytic(
 long double (*f)(long double),
 long double (*deriative)(long double),
 long double a,
 long double b) {
 long double x = (a+b)/2;
 long double x_next = x - f(x) / deriative(x);
 while (fabsl(x_next - x) >= LDBL_EPSILON) {
```

```
 x = x_next;
  x_next = x - f(x) / deriative(x);
 }
 return x_next;
}
// Computes newton method using numeric deriative
long double newton_numeric(
 long double (*f)(long double),
 long double a,
 long double b) {
 long double x = (a+b)/2;
 long double x_next = x - f(x) / derive(f, x);
 while (fabsl(x_next - x) >= LDBL_EPSILON) {
 x = x_next;
 x_next = x - f(x) / derive(f, x);
 }
 return x_next;
}
long double func_22(long double x) {
    return cosl(sqrtl(1-0.3*powl(x, 3)));
}

long double func_22_hands(long double x) {
    return (9*sqrtl(10)*powl(x, 2)*sinl(sqrtl( (10-3*powl(x, 3))/(10) )))/(20*sqrtl(10-3*powl(x, 3)));
}
// Returns 1 if method is covergent, otherwise 0
int is_iteration_covergent(
 long double (*f)(long double),
 long double (*deriative)(long double),
 long double a,
 long double b) {
 int flag = 1;
 long double step = (b-a)/1000;
 for (long double x=a; x<=b; x+=step) {
 if (derive(f, x) >= 1 || deriative(x) >= 1) {
 flag = 0;
 }
 }
 return flag;
}
// Computes iterations method
long double iterations(
 long double (*f)(long double),
 long double a,
 long double b) {
 long double x = (a+b)/2;
 long double x_next = f(x);
 while (fabsl(x_next - x) >= LDBL_EPSILON) {
 x = x_next;
 x_next = f(x);
 }
 return x_next;
}
int main() {
 long double a = 2, b = 4;
```

```
 printf("Function 23\nMethod: newton.\n");
 if (is_newton_covergent(func_23, func_23_deriative_1, func_23_deriative_2, a, b)) {
    printf("Method is covergent.\n");
 long double root_analytic = newton_analytic(func_23, func_23_deriative_1, a, b);
 long double root_numeric = newton_numeric(func_23, a, b);
 printf("Approximated root of the equation:\n");
 printf("\tAnalytic: %.19Lf\n\tNumeric: %.19Lf\n", root_analytic, root_numeric);
 long double res_analytic = func_23(root_analytic);
 long double res_numeric = func_23(root_numeric);
 printf("Inserting root in the equation:\n");
 printf("\tAnalytic: %.19Lf\n\tNumeric: %.19Lf\n", res_analytic, res_numeric);
 } else {
 printf("Method is not covergent.\n");
 }
 putchar('\n');
 a = 0;
 b = 1;
 printf("Function 22\nMethod: iterations.\n");
 if (is_iteration_covergent(func_22, func_22_hands, a, b)) {
 printf("Method is covergent.\n");
 long double root = iterations(func_22, a, b);
 printf("Approximated root of the equation: %.19Lf\n", root);
 long double res = root - 2 + sinl(1/root);
 printf("Inserting root in the equation: %.19Lf\n", res);
 } else {
 printf("Method is not covergent.\n");
 }
 return 0;
}
```

Входные данные

Нет

Выходные данные

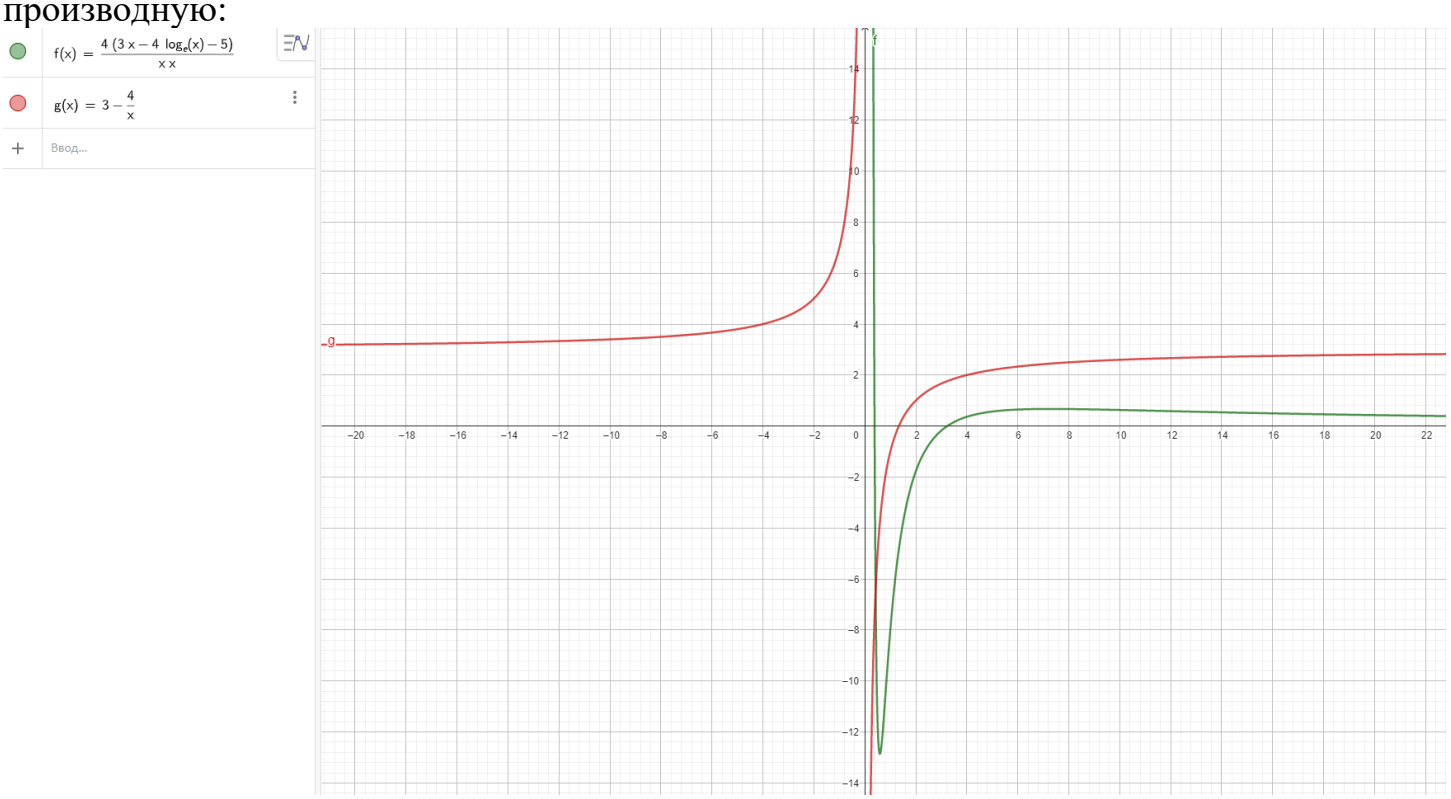
Программа должна вывести для каждого уравнения, сходится метод или нет. Вслучае, если сходится, вывести его значение.

Тест №1

```
Function 23
Method: newton.
Method is not covergent.

Function 22
Method: iterations.
Method is covergent.
Approximated root of the equation: 0.56292595285803870 41
Inserting root in the equation: -0.45814278003243338 53
```

Уравнение из задания 23 не оказалось сходящимся по Ньютону. Это могут подтвердить графики первой производной и функции, умноженной на вторую производную:



Вывод

В работе описаны и использованы различные численные методы для решения трансцендентных алгебраических уравнений. Даны обоснования сходимости и расходимости тех или иных методов. Имплементирована функция вычисления производной от заданной функции в точке. На основе алгоритма составлена программа на языке Си, сделана проверка

полученных значений путем подстановки. Работа представляется довольно полезной для понимания принципов работы численных методов и способов их имплементации.

Список литературы

1. Численное дифференецирование – URL:

   Численное дифференцирование — Википедия (wikipedia.org)

2. Конечная разность – URL:

   Численное дифференцирование — Википедия (wikipedia.org)