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 фунукции в качестве параметров»

Группа	М8О-109Б-22
Студент	Адамов А.А.
Преподаватель	Сысоев М.А.
Оценка	
Дата	10.12.2022

Москва, 2022

Постановка задачи

Составить программу на Си с процедурами решения трансцендентных алгебраических уравнений резличными 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 1:

Функция:

$$\boxed{e^x + \ln x - 10x = 0}$$

Отрезок, содержащий корень: [3, 4]

Метод Ньютона.

Вариант 28:

Функция:

$$\boxed{x - 2 + \sin\frac{1}{x} = 0}$$

Отрезок, содержащий корень [1.2, 2]

Метод итераций.

Теоретическая часть

Метод итераций

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

Достаточне 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: $x^{(k+1)}=f\left(x^{(k)}\right)$

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

Приближенное значения корня: $x^{s} \approx x^{(конечное)}$

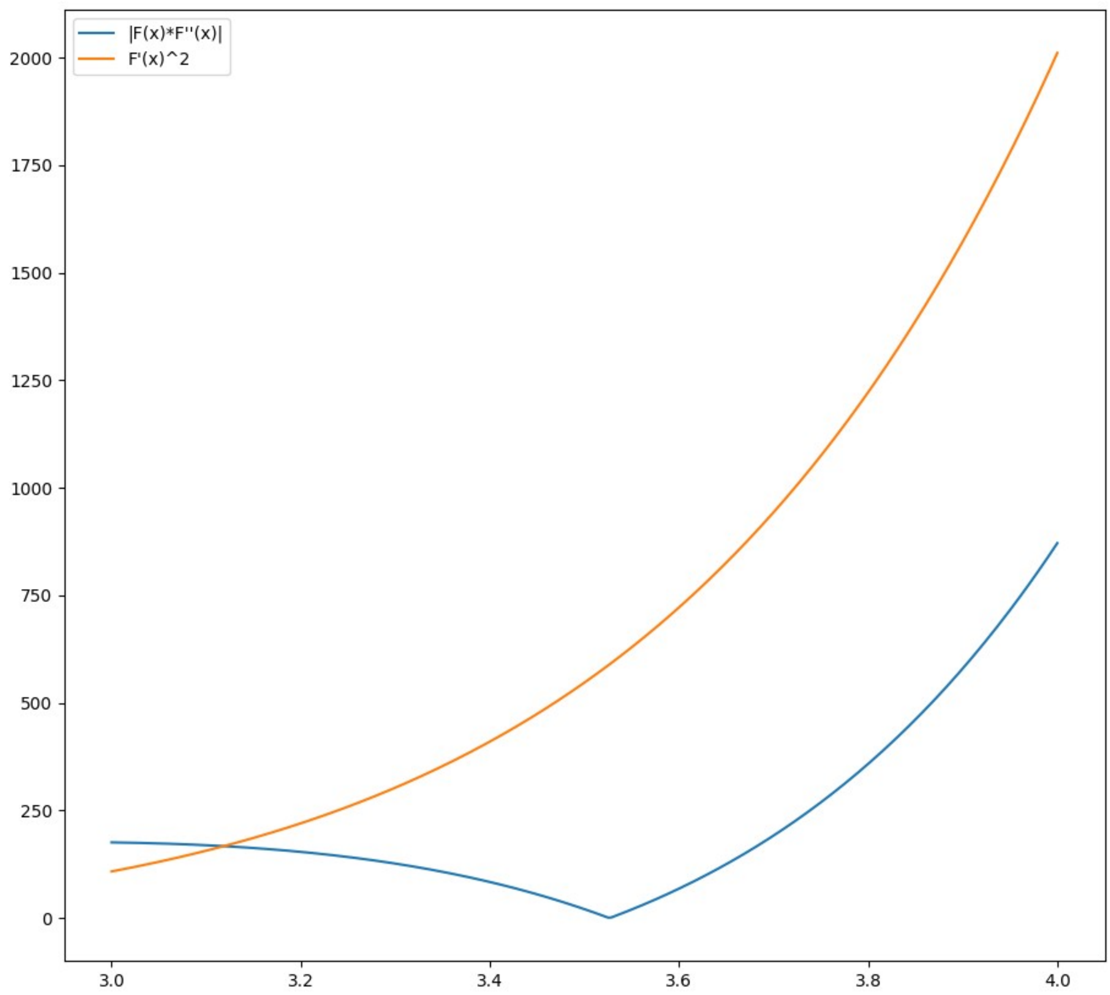
Метод Ньютона

Метод Ньютона является частным случаем метода итераций.

Условие сходимости метода: $|F(x) * F''(x)|<(F'(x))^2$

Итерационный процесс: $x^{(k+1)}=x^{(k)}-F\left(x^{(k)}\right)/F'\left(x^{(k)}\right)$

Посмотрим на график и заметим, что для нашей функции график $|F(x) * F''(x)|$ выше графика $(F'(x))^2$ на промежутке значений, близких к 3, потому метод не сходится.



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 h равное 1e-6:

$$\lim_{h \to 0} \left(\frac{f(x_0+h) - f(x_0)}{h} \right)$$

Описание алгоритма

Рассмотрим алгоритм решения.

Для начала с помощью функций is_newton_covergent и is_iterations_covergent проверим методы на сходимость. Для каждого сходящегося метода с его помощью вычислим корень соответсвующего уравнения. Затем подставим корни в уравнение для проверки.

Использованные в программе переменные

Название переменной	Тип переменной	Смысл переменной
a	long double	Начало отрезка
b	long double	Конец отрезка
root_numeric	long double	Корень первого уравнения, полученый с помощью численного дифференциирования
root_analytic	long double	Корень первого уравнения, полученый с помощью аналитической производной
root	long double	Корень двадцать восьмого уравнения
res	long double	Значение двадцать восьмого уравнения после подстановки корня
res_numeric	long double	Значение первого уравнения после подстановки корня, полученного с помощью численной производной
res_analytic	long double	Значение первого уравнения после подстановки корня, полученного с помощью аналитичской проивзодной

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_1(long double x) {
    // e^x + ln(x) - 10x = 0
    return expl(x) + logl(x) - 10*x;
}

long double func_1_deriative_1(long double x) {
    return expl(x) + (1/x) - 10;
}

long double func_1_deriative_2(long double x) {
    return expl(x) - (1/(x*x));
}

// 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_28(long double x) {
    // x = 2 - sin(1/x)
    return 2 - sinl(1/x);
}

long double func_28_deriative(long double x) {
    return cosl(1/x) * (1/(x*x));
}

// 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 = 3, b = 4;
    printf("Function 1:\n\texp(x) + ln(x) - 10x = 0\nMethod: newton.\n");
    if (is_newton_covergent(func_1, func_1_deriative_1, func_1_deriative_2, a,
b)) {
```

```
        printf("Method is covergent.\n");

        long double root_analytic = newton_analytic(func_1,
func_1_deriative_1, a, b);
        long double root_numeric = newton_numeric(func_1, 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_1(root_analytic);
        long double res_numeric = func_1(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 = 1.2;
    b = 2;
    printf("Function 28:\n\tx - 2 + sin(1/x) = 0\nMethod: iterations.\n");
    if (is_iteration_covergent(func_28, func_28_deriative, a, b)) {
        printf("Method is covergent.\n");

        long double root = iterations(func_28, 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.

Если метод сходится, вывести приближенный корень уравнения, а затем вывести значение уравнения, в который будет подставлен корень.

Протокол исполнения и тесты

Function 1:

exp(x) + ln(x) - 10x = 0

Method: newton.

Method is not covergent.

Function 28:

x - 2 + sin(1/x) = 0

Method: iterations.

Method is covergent.

Approximated root of the equation: 1.3076627156822268632

Inserting root in the equation: 0.0000000000000000000

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 дифференцирование https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5
2. Конечная разность https://en.wikipedia.org/wiki/Finite_difference
3. Методы численного дифференцирования функций http://aco.ifmo.ru/el_books/numerical_methods/lectures/glava1.html