Московский авиационный институт

(государственный технический университет)

Факультет прикладной математики

Кафедра вычислительной математики и программирования

КУРСОВАЯ РАБОТА

По курсам

«Процедуры и функции в качестве параметров»

I семестр

Задание 4

«Численные методы решения уравнений»

Студент: Чурилов С.Э.

Группа: М8О-103Б-20

Руководитель: Титов В.К.

Оценка:_____

Дата:_____

Москва, 2020

Содержание

Введение

В четвёртом задании курсового проекта необходимо составить программу на языке Си с процедурами решения трансцендентных алгебраических уравнений различными численными методами (итераций, Ньютона, дихотомии (половинного деления), хорд). Нелинейные уравнения необходимо оформить как параметры-функции, разрешив относительно неизвестной величины в случае необходимости. Следует применить каждую процедуру к решению трёх уравнений, заданных тремя строками таблицы, начиная с варианта с заданным номером (12). Если метод неприменим, дать математическое обоснование и графическую иллюстрацию.

Теоретические сведения

Рассматривается уравнение вида $F(x) = 0$. Предполагается, что функция $F(x)$ достаточно гладкая, монотонная на этом отрезке и существует корень уравнения $x^* \in [a; b]$. На отрезке $[a; b]$ ищется приближённое значение x с точностью ε, т.е. такое, что $|x - x^*| < \varepsilon$.

Различные численные методы предъявляют разные требования к функции $F(x)$, обладают различной скоростью сходимости и поведением.

Метод дихотомии (половинного деления)

Будем считать, что корень t функции $f(x)=0$ отделён на отрезке $[a,b]$. Задача заключается в том, чтобы найти и уточнить этот корень методом половинного деления. Другими словами, требуется найти приближённое значение корня с заданной точностью ε.

Пусть функция f непрерывна на отрезке $[a,b]$, $f(a) \cdot f(b) < 0$ и $t \in [a,b]$ – единственный корень уравнения $f(x)=0,\ a \le t \le b$. Поделим отрезок $[a,b]$ пополам. Получим точку $c = \frac{a+b}{2},\ a<c<b$ и два отрезка $[a,c],\ [c,b]$.

- Если $f(c)=0$, то корень найден ( ).$t = c$

- Если нет, то из двух полученных отрезков $[a,c]$ и $[c,b]$ надо выбрать один $[a_1;b_1]$, такой что $f(a_1)\cdot f(b_1)<0$, т.е.
  - $[a_1;b_1]=[a,c]$, если $f(a)\cdot f(c)<0$ или
  - $[a_1;b_1]=[c,b]$, если $f(c)\cdot f(b)<0$.

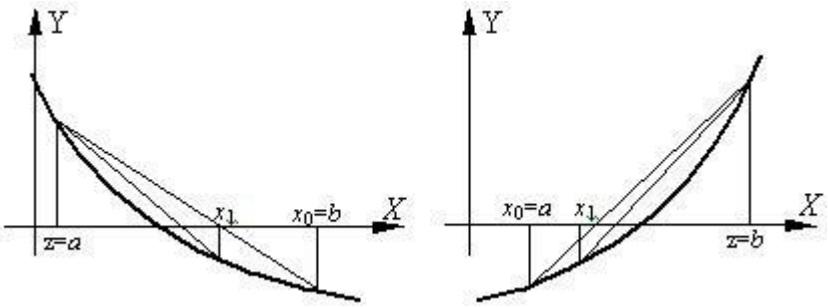
Новый отрезок $[a_1;b_1]$ делим пополам. Получаем середину этого отрезка $c_1=\frac{a_1+b_1}{2}$ и так далее. Для того, чтобы найти приближённое значение корня с точностью до $\varepsilon > 0$, необходимо остановить процесс половинного деления на таком шаге n, на котором $|b_n-c_n|<\epsilon$ и вычислить $x=\frac{a_n+b_n}{2}$. Тогда можно взять $t\approx x$

Метод хорд

Недостаток деления отрезка строго пополам проистекает от того, что он использует лишь знак функции, игнорируя отклонение (абсолютную величину). Но очевидно, что чем меньше (по абсолютной величине) значение функции, тем ближе мы находимся к корню. Метод хорд предлагает делить отрезок в точке, отстоящей от краев отрезка пропорционально абсолютному значению функции на краях. (Название "метод хорд" происходит от того, что точка деления является пересечением отрезка $[(X_{left},\ F(X_{left})),\ (X_{right},\ F(X_{right}))]$ – хорды - с осью абсцисс.)

Метод основан на замене функции $f(x)$ на каждом шаге поиска хордой, пересечение которой с осью X дает приближение корня. При этом в процессе поиска семейство хорд может строиться:

1. при фиксированном левом конце хорд, т.е. $z=a$, тогда начальная точка $x_0=b$;
2. при фиксированном правом конце хорд, т.е. $z=b$, тогда начальная точка $x_0=a$;



В результате итерационный процесс схождения к корню реализуется рекуррентной формулой:

1. для случая а): $x_{n+1} = x_n - \frac{f(x_n)}{f(x_n)-f(a)}(x_n - a)$;
2. для случая б): $x_{n+1} = x_n - \frac{f(x_n)}{f(x_n)-f(b)}(x_n - b)$;

Процесс поиска продолжается до тех пор, пока не выполнится условие $|x_{n+1} - x_n| \le \varepsilon$ или $|h| \le \varepsilon$.

Метод итераций

Идея метода заключается в замене исходного уравнения $F(x) = 0$ уравнением вида $x = f(x)$. Достаточное условие сходимости метода: $|f'_{(k)}| < 1$, $x \in [a; b]$. Начальное приближение корня: $x^{(0)} = (a + b) / 2$ (середина исходного отрезка). Итерационный процесс: $x^{(k+1)} = f(x^{(k)})$. Условие окончания: $|x^{(k)} - x^{(k+1)}| < \varepsilon$. Приближённое значение корня: $x^* \approx x^{(\text{конечное})}$.

Метод Ньютона

Метод Ньютона является частным случаем метода итераций. Метод обладает квадратичной сходимостью. В случае решения задач оптимизации предполагается, что функция $f(x)$ дважды непрерывно дифференцируема. Условие сходимости: $|F(x) \cdot F''(x)| < (F'(x))^2$ на отрезке $[a; b]$. Итерационный процесс $x^{(k+1)} = x^{(k)} - F(x^{(k)})/F'(x^{(k)})$.

Решение задачи

Для написания программного кода требуется предварительно «подготовить» функции. Найти производные, выразить функции в виде $x = f(x)$.

1) Функция F(x) = e^x + $\ln(x)$ – 10x. Заданный отрезок – [3,4]

$e^x + \frac{1}{x}$ – 10 – первая производная;

ln(10x – $ln(x)$) – функция в виде x = f(x);

2) Функция F(x) = $cosx$- $e^{\frac{-x^2}{2}}$ + x - 1. Заданный отрезок – [1,2]

-$sinx$ + x * $e^{\frac{-x^2}{2}}$ + 1 – первая производная;

1 - $cosx$ + $e^{\frac{-x^2}{2}}$ – функция в виде x = f(x);

3) Функция F(x) = 1 - x + $sinx$ - $ln(1 + x)$. Заданный отрезок – [1,1.5]

-1 + $cosx$ - $\frac{1}{(1+x)}$ – первая производная;

1 + $sinx$ - $ln(1 + x)$– функция в виде x = f(x);

В необходимую процедуру поиска корня подается сама функция, у которой нужно найти корень и значения концов самого отрезка. Далее осуществляется поиск корня, пока он не будет найден точно или с какой-то погрешностью. Описание функций в программном коде (возвращаемый тип всех функций – double):

Имя, аргументы	Назначение
f1 (double x)	Значение функции от x (изначальная функция 1 варианта)
f1p (double x)	Значение первой производной функции от x (первая производная функции 1 варианта)
F1 (double x)	Значение функции в виде x = f(x) (изначальной функции 1 варианта)

f2 (double x)	Значение функции от x (изначальная функция 2 варианта)
f2p (double x)	Значение первой производной функции от x (первая производная функции 2 варианта)
F2 (double x)	Значение функции в виде x = f(x) (изначальной функции 2 варианта)
f3 (double x)	Значение функции от x (изначальная функция 3 варианта)
f3p (double x)	Значение первой производной функции от x (первая производная функции 3 варианта)
F3 (double x)	Значение функции в виде x = f(x) (изначальной функции 3 варианта)
dichotomy (double f(double x), double a, double b)	Нахождение корня методом дихотомии (половинного деления)
iteration (double f(double x), double fd(double y), double a, double b)	Нахождение корня методом итераций (в качестве первого аргумента передаётся указатель на функцию, реализующую изначальную в виде $x = f(x)$).
newton (double f(double x), double fd(double y), double a, double b)	Нахождение корня методом Ньютона
hord (double f(double x), double a, double b)	Нахождение корня методом хорд

Эпсилон задаётся равным 0.00001, что позволяет обеспечить приемлемую точность. В коде программы сначала происходит подключение заголовочных файлов, затем объявление переменной eps (эпсилон). После чего идут процедуры функций и методов решения. В теле функции main происходит вызов соответствующих функций и печать итоговой информации. Границы отрезков передаются в функции в виде аргументов

Протокол

```
leninware@leninware-VirtualBox:~$ cat myinfo
**************************************************
* ФИО: Чурилов Сергей Эдуардович                 *
* Группа: М8О-103Б-20                             *
* E-mail: churilov.ser1204@gmail.com              *
* КП: 4                                           *
* Номер по списку: 29                            *
* Номер варианта: 1                              *
**************************************************
leninware@leninware-VirtualBox:~$ cat kurs4.c
#include <stdio.h>
#include <math.h>
const double eps = 0.00001;

double f1(double x) {
    return exp(x) + log(x) - 10 * x;
}
double F1(double x) {
    return log(10 * x - log(x));
}
double f1p(double x) {
    return exp(x) + 1/x - 10;
}

double f2(double x) {
    return cos(x) - exp(-x * x * 0.5) + x - 1;
}
double F2(double x) {
    return 1 - cos(x) + exp(-x * x * 0.5);
}
double f2p(double x) {
    return -sin(x) + exp(-x * x * 0.5) * x + 1;
}

double f3(double x) {
    return 1 - x + sin(x) - log(1 + x);
}
double F3(double x) {
    return 1 + sin(x) - log(1 + x);
}
double f3p(double x) {
    return -1 + cos(x) - 1/(1 + x);
}
// Dichotomy
double Dichotomy(double f(double x), double a, double b) {
    double c;
    while (fabs(a - b) > eps) {
        c = (a + b) / 2;
        if (f(a) * f(c) > 0)
            a = c;
        else b = c;
    }
    return c;
}
// Iteration
double Iteration(double f(double x), double a, double b) {
    double x = (a + b) / 2;
    double x1 = x + 1;
    while (fabs(x - x1) > eps) {
```

```
        x1 = x;
        x = f(x);
    }
    return x;
}
// Newton
double Newton(double f(double x), double fd(double y), double a,
double b) {
    double x = (a + b) / 2;
    double x1 = x - (f(x) / fd(x));
    while (fabs(x1 - x) > eps) {
        x = x1;
        x1 = x - (f(x) / fd(x));
    }
    return x;
}
//Hord
double Hord(double f(double x), double a, double b) {
    double c;
    while (fabs(a - b) > eps) {
        c = a - f(a) * (b - a) / (f(b) - f(a));
        a = b;
        b = c;
    }
    return c;
}
int main() {
    printf("e^x + ln(x) - 10x = 0\n");
    printf("Newton => %.4f\n", Newton(f1, f1p, 3, 4));
    printf("Iteration => %.4f\n", Iteration(F1, 3, 4));
    printf("Dihotomia => %.4f\n", Dichotomy(f1, 3, 4));
    printf("Hordes => %.4f\n\n", Hord(f1, 3, 4));



    printf("cos(x) - e^(-(X^2) / 2) + x - 1 = 0\n");
    printf("Newton => %.4f\n", Newton(f2, f2p, 1, 2));
    printf("Iteration => %.4f\n", Iteration(F2, 1, 2));
    printf("Dihotomia => %.4f\n", Dichotomy(f2, 1, 2));
    printf("Hordes => %.4f\n\n", Hord(f2, 1, 2));

    printf("1 - x + sin(x) - ln(1 + x) = 0\n");
    printf("Newton => %.4f\n", Newton(f3, f3p, 1, 1.5));
    printf("Iteration => %.4f\n", Iteration(F3, 1, 1.5));
    printf("Dihotomia => %.4f\n", Dichotomy(f3, 1, 1.5));
    printf("Hordes => %.4f\n", Hord(f3, 1, 1.5));

    return 0;
}
leninware@leninware-VirtualBox:~$ gcc -g -o kurs4 kurs4.c -lm
leninware@leninware-VirtualBox:~$ ./kurs4

e^x + ln(x) - 10x = 0
Newton=> 3.52650
Iteration => 3.52650
Dihotomia => 3.52650
Hordes=> 3.52650

cos(x) - e^(-(X^2) / 2) + x - 1 = 0
Newton=> 1.08944
Iteration => 1.08945
Dihotomia => 1.08944
```

```
Hordes=> 1.08944

1 - x + sin(x) - ln(1 + x) = 0
Newton=> 1.14744
Iteration => 1.14744
Dihotomia => 1.14744
Hordes=> 1.14744
leninware@leninware-VirtualBox:~$
```

Заключение

Задание номер четыре курсового проекта выполнено. Были найдены производные функций, преобразованы в необходимой форме. В программе реализованы все требуемые методы для решения трансцендентных алгебраических уравнений. Обеспечена приемлемая точность полученных ответов. Работу можно считать завершённой.