

Черкасов А. А-06-19 Вариант 24

Условие задачи

Двумя методами для N значений погрешности ϵ (0,1; 0,01; 0,001; ... 10^{-N} , $1 \leq N \leq 10$) вычислить значение корня для двух заданных функций (№ и №+3 по табл.2) на отрезке [A, B] (0,2) и вывести их в виде таблицы

№	Функция	Методы
24	$\frac{1}{x\sqrt{x+0,3} + e^{-x} + 1/7} - x;$ 0,73153	1) Метод деления отрезка пополам 2) метод хорд
27	$\cos x - e^{-(x+1)^2} + \frac{1}{9} - x;$ 0,77997	1) Метод деления отрезка пополам 2) метод секущих

Метод деления пополам

На каждом шаге отрезок уменьшается вдвое. В начале каждой итерации находим середину нового отрезка [a, b]. Затем следует определить, с какой стороны от середины отрезка x находится корень x^* . Для этого достаточно сравнить знаки $f(x)$ и $f(b)$ или знаки $f(x)$ и $f(a)$. Если знаки $f(x)$ и $f(b)$ не совпадают, то это означает, что $f(x)$ пересекает ось x на правом полуотрезке [x, b]. Заменяем a на x. Если же знаки $f(x)$ и $f(b)$ совпадают, то $f(x)$ пересекает ось x на левом полуотрезке [a, x]. Заменяем b на x. Итак, в результате выполнения итерации отрезок [a, b] как и прежде, содержит единственный корень, но его длина стала меньше в два раза. Вычисления следует прекратить, если на очередном шаге длина отрезка [a, b] станет меньше ϵ

Метод секущих

Метод секущих, в отличие от метода Ньютона, использует не одно, а два начальных приближения, которые мы обозначим соответственно x_1 и x_2 , чтобы найти третье x_3 . За начальные значения x_1 и x_2 берутся соответственно a и b.

Третья точка находится как точка пересечения отрезка, проведенного между точками $(x_1, f(x_1))$ и $(x_2, f(x_2))$ по формуле:
$$x_3 = x_1 - \frac{f(x_1)}{\frac{f(x_2) - f(x_1)}{x_2 - x_1}}$$

Из трех приближений к корню оставим два последних (отбрасываем самое старое x_1). В методе секущих это делается по следующему правилу: $x_1 = x_2$; $x_2 = x_3$. После чего можно заново искать третье значение по формуле (4).

Выполнение итераций можно прекратить при выполнении условия: $|x_3 - x_2| < \epsilon$ а полученное значение приближения x_3 взять в качестве искомого значения корня.

Метод хорд

Метод хорд Этот метод вместе с методом бисекции относится к методам дихотомии – как отрезок делится на две части, но на этот раз неравных. Точка деления отрезка находится как точка пересечения отрезка,

проведенного между точками $(a, f(a))$ и $(b, f(b))$, с осью OX по формуле: $x = a - \frac{f(a)}{\frac{f(b)-f(a)}{b-a}}$

Новые значения a и b вычисляются так же, как и в методе бисекции – в зависимости от знаков на границах новых отрезков $[a, x]$ и $[x, b]$. При выполнении итераций по методу хорд может оказаться, что к корню приближается только левая или только правая граница отрезка $[a, b]$. Поэтому в качестве меры близости к корню здесь следует применить величину перемещения границы при очередной итерации, которая равна: $x - a$, если корень справа от x и перемещаем a , $d = b - x$, если корень слева от x и перемещаем b .

Необходимая точность будет достигнута при выполнении после очередной итерации неравенства: $|d| < \varepsilon$ Полученное значение приближения x надо взять в качестве искомого значения корня.

Результат работы программы

Отрезок $[0.1, 1.9]$

Функция 24

Метод Точность	Деление пополам		Метод хорд	
	Корень	Итерации	Корень	Итерации
0.1	0.7	5	0.7	3
0.5	0.6	2	0.7	3
0.01	0.73	8	0.73	4
0.05	0.75	6	0.73	3
0.001	0.732	11	0.732	4
0.005	0.729	9	0.732	4
0.0001	0.7315	15	0.7315	4
0.0005	0.7315	12	0.7315	4
0.00001	0.73153	18	0.73153	4
0.00005	0.73152	16	0.73153	4

Функция 27

Метод Точность	Деление пополам		Метод секущих	
	Корень	Итерации	Корень	Итерации
0.1	0.8	5	Нет корня	3
0.5	0.6	2	0.7	2
0.01	0.78	8	Нет корня	3
0.05	0.80	6	Нет корня	3
0.001	0.779	11	Нет корня	3
0.005	0.779	9	Нет корня	3
0.0001	0.7800	15	Нет корня	3
0.0005	0.7798	12	Нет корня	3
0.00001	0.77996	18	Нет корня	3
0.00005	0.77997	16	Нет корня	3

Отрезок [0.7,0.8]

Функция 24

Метод Точность	Деление пополам		Метод хорд	
	Корень	Итерации	Корень	Итерации
0.1	0.8	1	0.7	1
0.5	0.8	1	0.7	1
0.01	0.73	4	0.73	2
0.05	0.72	2	0.73	1
0.001	0.732	7	0.732	2
0.005	0.734	5	0.732	2
0.0001	0.7315	10	0.7315	2
0.0005	0.7316	8	0.7315	2
0.00001	0.73152	14	0.73153	3
0.00005	0.73149	11	0.73153	2

Функция 27

Метод Точность	Деление пополам		Метод секущих	
	Корень	Итерации	Корень	Итерации
0.1	0.8	1	0.8	1
0.5	0.8	1	0.8	1
0.01	0.78	4	0.78	2
0.05	0.78	2	0.78	1
0.001	0.780	7	0.780	2
0.005	0.778	8	0.780	2
0.0001	0.7800	10	Нет корня	3
0.0005	0.7801	8	Нет корня	3
0.00001	0.77997	14	Нет корня	3
0.00005	0.77993	11	Нет корня	3

Вывод: на основе таблицы можно сказать, что самым лучшим методом из рассмотренных является метод хорд – наиболее точный и быстрый метод.

Метод секущих можно считать самым наихудшим – не всегда находит корень, работать может с маленьким отрезком или с небольшой точностью.

Код

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.15)
project(Lab8)
```

```
set(CMAKE_CXX_STANDARD 17)
```

```
add_executable(Lab8 main.cpp)
```

main.cpp

```
#include <cstdio>
#include <cmath>
#include <cstdlib>
#include "LabFunc.h"
```

```
int main() {
    system("chcp 65001"); //Поддержка кириллицы Входной файл UTF-8
    printf("Введите значения A, B\n");
    int n;
    double a,b;
    try {
        scanf("%lf", &a);
        if (a<=0) throw 1; if (a>=2) throw 2; // Проверка на аномалии с выдачей кода ошибки
        scanf("%lf", &b);
        if (b<=0) throw 3; if (b>=2) throw 4; // Проверка на аномалии с выдачей кода ошибки
        if (b<a) throw 5;
        fflush(stdin);
        printf("Введите количество итераций\n");
        scanf("%d", &n); if (n<1) throw 6; if (n>10) throw 7; // Проверка на аномалии с выдачей
кода ошибки
        double e[n];
        for (int i = 0; i < n; i++) {
            printf("Точность[%d]= ", i + 1);
            scanf("%lf", &e[i]);
            if (floor(e[i]*pow(10,10))==0) throw 8 + i;
            fflush(stdin);
        }

        printf("Для первой функции:\n");
        TableStart(24, a, b); //Начало таблцы
        for (int i = 0; i < n; i++) { //Поиск корня
            int n1 = 0;
            int n2 = 0;
            double x1 = FindRootDiv(a, b, e[i], funcF, &n1); //Деление пополам
            double x2 = FindRootChord(a, b, e[i], funcF, &n2); //Метод хорд
            TableCell(e[i], x1, n1, x2, n2); //Вывод в таблицу
        }
        printf("Для второй функции:\n");
        TableStart(27, a, b); //Начало таблцы
    }
```

```

for (int i = 0; i < n; i++) { //Поиск корня
    int n1 = 0;
    int n2 = 0;
    double x1 = FindRootDiv(a, b, e[i], funcS, &n1); //Деление пополам
    double x2 = FindRootSec(a, b, e[i], funcS, &n2); //Метод хорд
    TableCell(e[i], x1, n1, x2, n2); //Вывод в таблицу
}
}
catch (int errNum) { //Ловим код ошибки
    switch (errNum){ //Обрабатываем
        default:
            errNum-=7;
            printf("Точность[%d] слишком маленькая\n",errNum);
            break;
        case 1:
            printf("\A\' должна быть больше 0\n");
            break;
        case 2:
            printf("\A\' должна быть меньше 2\n");
            break;
        case 3:
            printf("\B\' должна быть больше 0\n");
            break;
        case 4:
            printf("\B\' должна быть меньше 2\n");
            break;
        case 5:
            printf("\A\' не должна быть больше \B\'\n");
            break;
        case 6:
            printf("\N\' должна быть больше 1\n");
            break;
        case 7:
            printf("\N\' должна не превосходить 10\n");
            break;
    }
}
printf("-----\nPress ENTER"); getc(stdin);
return 0;
}

```

LabFunc.h

typedef double (*func)(double); //Указатель на функцию типа double с принимаемым значением double

```
double funcF(double x) //Функция N24
{
    return (
        (double)1/(x*sqrt(x+0.3)+exp(-x)+(double)1/(double)7)-x
    );
}
```

```
double funcS(double x) //Функция N27
{
    return (
        cos(x)-exp(-pow(x+(double)1,2))+(double)1/(double)9-x
    );
}
```

```
double FindRootDiv(double a, double b, double e, func f,int *nIter) //Метод деления отрезка
{
    double x,FA,FB,FX;
    bool flag = true; //Проверка на наличие корня на отрезке
    do //Цикл с постусловием
    {
        x = (b+a)/(double)2; //Середина отрезка
        FA = f(a); //F(a) левый край
        FB = f(b); //F(b) правый край
        FX = f(x); //F(X)
        if (FX*FB<0) a = x; //Сравнение знаков F(x) и F(b)
        else b = x;
        (*nIter)++;
        if (FA*FB>0) flag=false; //Если знак одинаковый - нет корня
    } while (fabs(a-b)>=e and FX!=0 and flag); //Ищем, пока не пересечем Oх или пока
    отрезок не будет меньше точности(погрешности)
    if (!flag) return -1;
    return x;
}
```

```
double FindRootChord(double a, double b, double e, func f,int *nIter) //Метод хорд
{
    double x,FA,FB,FX,d;
    bool flag = true; //Проверка на наличие корня на отрезке
    do //Цикл с постусловием
    {
        FA = f(a); //F(a) левый край
        FB = f(b); //F(b) правый край
        x = a - (FA*(b-a)/(FB-FA));
        FX = f(x); //F(X)
        if (FX*FB<0) { //Сравнение знаков F(x) и F(b)
            d = fabs(a-x);
            a = x;
        }
    } while (d>e and FX!=0 and flag);
    return x;
}
```

```

    }
    else {
        d = fabs(b-x);
        b = x;
    }
    (*nIter)++;
    if (FA*FB>0) flag=false; //Если знак одинаковый - нет корня
}
while(d>=e and FX!=0 and flag); //Ищем, пока не пересечем Ох или пока один из
отрезков не будет меньше точности(погрешности)
if (!flag) return -1;
return x;
}

```

```

double FindRootSec(double a, double b, double e, func f,int *nIter) //Метод секущей
{
    double x1,x2,x3,FL,FR,FX;
    x1=a; x2=b;
    bool flag = true; //Проверка на наличие корня на отрезке
    do //Цикл с постусловием
    {
        FL = f(x1); //Левый край
        FR = f(x2); //Правый край
        x3 = x1 - FL/(FR-FL)*(x2-x1);
        FX = f(x3);
        x1=x2;x2=x3; //Переходим к следующему отрезку
        (*nIter)++;
        if (FL*FR>0) flag=false; //Если знак одинаковый - нет корня
    }
    while(fabs(x2-x1)>=e and FX!=0 and flag); //Ищем, пока не пересечем Ох или пока
отрезок не будет меньше точности(погрешности)
    if (!flag) return -1;
    return x3;
}

```

```

void TableStart(int n,double a,double b) //Личное удобство для вывода таблицы
{
    char* fname;
    if (n==24) fname =(char*)"    Метод хорд    ";
    else fname=(char*)"    Метод секущих    ";
    printf("Отрезок [%.5f, %.5f]\n",a,b);
    printf("|Функция N%d | Деление отрезка   |%s|\n",n,fname);
    printf("|Точность   |Корень   |Итерации |Корень   |Итерации |\n");
}

```

```

void TableCell(double e,double x1,int n1, double x2, int n2) //Личное удобство для вывода
таблицы
{
    int z = ceil(fabs(log(e)/log(10)));
    char* r1 = new char[z+2];
    char* r2 = new char[z+2];
    if (x1==-1) r1 = (char*)"Нет корня   ";
}

```

```
else sprintf(r1,"%.*f",z,x1);
if (x2==-1) r2 = (char*)"Нет корня ";
else sprintf(r2,"%.*f",z,x2);
printf("|%12.*f|%12s|%9d|%12s|%9d|\n",z,e,r1,n1,r2,n2);
delete []r1; delete []r2;
}
```