МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Воронежский государственный технический университет»

(ФГБОУ ВО «ВГТУ», ВГТУ)

**Факультет** информационных технологий и компьютерной безопасности

**Кафедра** систем управления и информационных технологий в строительстве

Отчет по лабораторной работе №3

по дисциплине: «Информатика»

Выполнил студент: Рыженков Евгений

Группа: бИСТ-221

Руководитель: доцент, к.т.н. Ефимова О.Е.

Работа защищена « » 2023г.

С оценкой

(подпись)

Лабораторная работа № 3

«Решение нелинейных алгебраических уравнений»

Вариант 23

Задания для лабораторной работы:

**Задания Варианта 23:**

**Задание**

Самостоятельно рассмотреть метод секущих. Составить алгоритм и блок схему метода секущих. Объяснить, чем метод секущих отличается от метода касательных, и что у них общего. Самостоятельно рассмотреть комбинированный метод хорд и касательных. Объяснить специфику применения комбинированного метода и касательных.

Используя методы нахождения корней нелинейного уравнения, найти все корни уравнения в соответствии с индивидуальным вариантом с заданной точностью:

|  |
| --- |
|  |
|  |

**Нахождения корня нелинейного уравнения методом Секущих.**

1. Задаем начальный интервал [a, b] и погрешность расчета. Начальный шаг - 1.

2. Проверяем условие существование корня на заданном интервале (график должен пересекать ось X): f(a) \* f(b) < 0.

Если корни есть, то вычисляем значения функции f(a) и f(b). Если нет, то выводим сообщение, что отрезок не является интервалом изоляции корня.

3. Вычисляем значения функции f(a) и f(b).

4. Для нахождения точки пересечения с осью абсцисс используем уравнение (рис. 1), где x(k) =b, а x(k-1) = a.

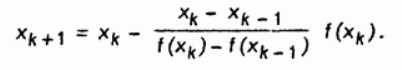


Рис. 1 - Итерационная формула для метода секущих.

После этого задаём новый интервал неопределенности [b, x] (a = b, b = x), на котором находится корень уравнения.

5. Процесс поиска продолжается до тех пор, пока не выполнится условие |b-a| <= ε. Если заданная точность не достигается, то продолжаем итерационный процесс n += 1 и вычисляем значения функции f(a) и f(b), иначе выводим результат.

6. Вывод результатов.

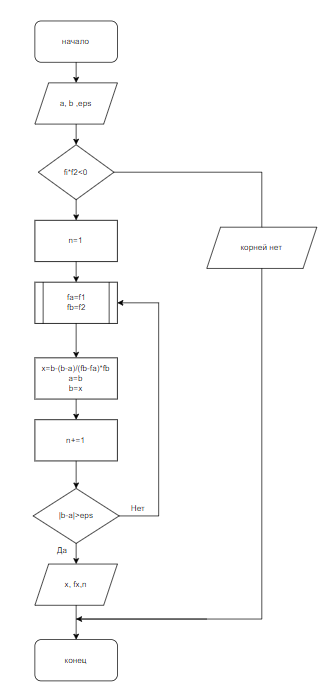


Рис. 2 - Блок-схема метода секущих.

Метод секущих — модификация метода Ньютона, в котором производная заменена на секущую. Секущая — прямая, проходящая через две точки на графике функции. В данном методе процесс итераций состоит в том, что в качестве приближений корню уравнения принимаются последовательные значения точек пересечения секущей с осью абсцисс приближение к корню происходит с двух сторон.

Метод секущих является двухшаговым, то есть новое приближение Описание: x^{k+1} определяется двумя предыдущими итерациями Описание: x^k и Описание: x^{k-1}. В методе необходимо задавать *два* начальных приближения Описание: x^0 и Описание: x^1.

Метод секущих несколько уступает методу Ньютона в скорости сходимости, одна не требует вычисления производной левой части уравнения.

По алгоритму метод секущих близок к методу хорд, однако в отличие от последнего начальные приближения в методе секущих могут располагаться как с разных сторон от корня, так и с одной стороны; кроме того, при уточнении корня не повторяются знаки функции f(х).

В комбинированном методе хорд и касательных мы разбиваем отрезки [a; b] (при условии f(a)f(b) <0) на три отрезка с помощью хорды и касательной и выборе нового отрезка от точки пересечения хорды с осью абсцисс до точки пересечения касательной с осью абсцисс, на котором функция меняет знак и содержит решение.

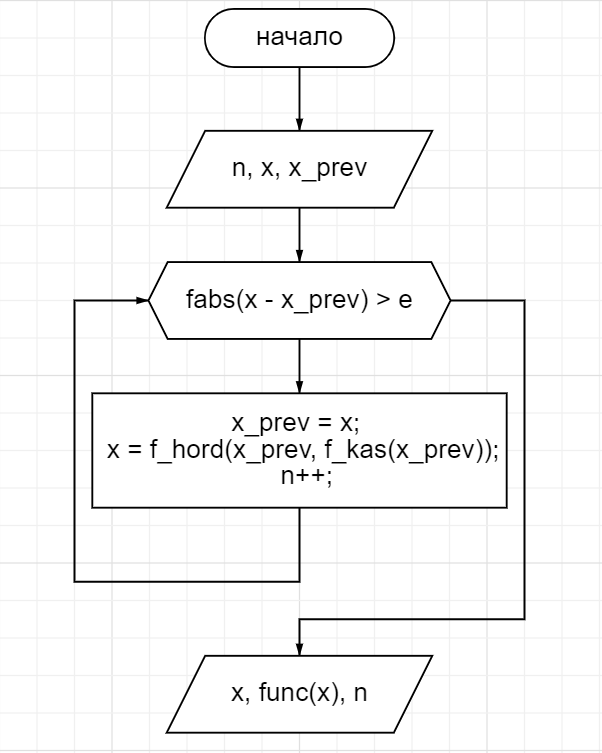


Рисунок 3 – Блок-схема комбинированного метода хорд и касательных

Построение хорд и касательных продолжается до достижения необходимой точности решения ε. Блок-схема комбинированного метода хорд и касательных показана на (рисунке 3).

**Результат выполнения заданий**

|  |
| --- |
|  |
| Рисунок 4 – результат выполнения метода половинного деления |

|  |
| --- |
|  |
| Рисунок – 5 результат выполнения метода Хорд |

|  |
| --- |
|  |
| Рисунок 6 – результат выполнения метода Ньютона |

|  |
| --- |
|  |
| Рисунок 7 – результат выполнения метода простых итераций |

|  |
| --- |
|  |
| Рисунок 8 – результат выполнения метода секущих |

|  |
| --- |
|  |
| Рисунок 9 – результат выполнения метода хорд и касательных |

**Вывод:** Проанализировав результаты работы программ, мы можем сделать вывод, что наиболее эффективным методом решения нелинейных алгебраических уравнений является метод секущих, так же малое количество итераций имеют методы: комбинированный метод хорд и касательных, метод Ньютона, метод хорд. Самый медленный - метод последовательных приближений, в сравнении с другими методами это очень неэффективный метод.

**Листинг программы на языке с++**

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <math.h>

#include <cmath>

using namespace std;

double metod1\_dichotomy(double a, double b, double eps);

double metod2\_hord(double a, double b, double eps);

double metod3\_Newton(double a, double b, double eps);

double metod4\_simple\_iteration(double a, double b, double x0, double eps);

double metod5\_sowing(double a, double b, double x0, double eps);

double f\_hord(double x1, double x2);

double f\_kas(double x0);

double komb(double a, double b, double e);

double func(double x);

double func1(double x);

double func2(double x);

int main() {

setlocale(LC\_ALL, "RUS");

double a, b, eps = 0.000001, x0;

unsigned int choice = 0;

while (choice != 7) {

cout << "Выберите пункт меню: " << endl;

cout << "1. Метод Деления Отрезка Пополам;" << endl;

cout << "2. Метод Хорд;" << endl;

cout << "3. Метод Ньютона;" << endl;

cout << "4. Метод последовательных приближений;" << endl;

cout << "5. Метод секущих;" << endl;

cout << "6. Комбинированным методом Хорд и Касательных;" << endl;

cout << "7. Выход" << endl;

cout << "Ваш выбор: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Введите значение 'a': ";

cin >> a;

cout << "Введите значение 'b': ";

cin >> b;

cout << "Введите значение 'eps': ";

cin >> eps;

if (func(a) \* func(b) > 0) {

cout << "На этом отрезке корней нет!" << endl;

}

else {

metod1\_dichotomy(a, b, eps);

}

break;

case 2:

cout << "Введите значение 'a': ";

cin >> a;

cout << "Введите значение 'b': ";

cin >> b;

cout << "Введите значение 'eps': ";

cin >> eps;

if (func(a) \* func(b) < 0) {

metod2\_hord(a, b, eps);

}

else {

cout << "На этом отрезке корней нет!" << endl;

}

break;

case 3:

cout << "Введите значение 'a': ";

cin >> a;

cout << "Введите значение 'b': ";

cin >> b;

cout << "Введите значение 'eps': ";

cin >> eps;

if (func(a) \* func(b) < 0) {

metod3\_Newton(a, b, eps);

}

else {

cout << "На этом отрезке корней нет!" << endl;

}

break;

case 4:

cout << "Введите значение 'a': ";

cin >> a;

cout << "Введите значение 'b': ";

cin >> b;

cout << "Введите значение 'eps': ";

cin >> eps;

cout << "Введите значение 'x0': ";

cin >> x0;

metod4\_simple\_iteration(a, b, x0, eps);

break;

case 5:

cout << "Введите значение 'a': ";

cin >> a;

cout << "Введите значение 'b': ";

cin >> b;

cout << "Введите значение 'eps': ";

cin >> eps;

cout << "Введите значение 'x0': ";

cin >> x0;

metod5\_sowing(a, b, x0, eps);

break;

case 6:

cout << "Введите значение 'a': ";

cin >> a;

cout << "Введите значение 'b': ";

cin >> b;

cout << "Введите значение 'eps': ";

cin >> eps;

komb(a, b, eps);

break;

case 7:

break;

default:

cout << "Выбрано недопустимое значение!" << endl;

system("pause");

system("cls");

break;

}

}

return 0;

}

double metod1\_dichotomy(double a, double b, double eps) {

unsigned int n = 1;

double t, f2, x, f\_res;

while (fabs(b - a) >= eps) {

t = (a + b) / 2;

f2 = func(t);

if (func(a) \* f2 < 0) {

b = t;

}

else {

a = t;

}

n += 1;

if (fabs(b - a) <= eps) {

x = (a + b) / 2;

cout << "Корень уравнения = " << x << "; Значение функции = " << func(x) << "; Кол-во проходов функции = " << n << endl;

return x;

}

}

return 0;

}

double metod2\_hord(double a, double b, double eps) {

double f1 = func(a), f2 = func2(a), x, z, fz, h = a;

unsigned n = 1;

if (f1 \* f2 > 0) {

x = b;

z = a;

}

else {

x = a;

z = b;

}

fz = func(z);

while (fabs(h) >= eps) {

f1 = func(x);

h = ((x - z) \* f1) / ((f1 - fz));

x = x - h;

n += 1;

if (fabs(h) <= eps) {

cout << "Корень уравнения = " << x << "; Значение функции = " << func(x) << "; Кол-во проходов функции = " << n << endl;

}

}

return x;

}

double metod3\_Newton(double a, double b, double eps) {

double f = func(b), f2 = func2(b), x, h = a, f1;

unsigned n = 1;

if (f \* f2 > 0) {

x = b;

}

else {

x = a;

}

while (fabs(h) >= eps) {

f = func(x);

f1 = func1(x);

h = f / f1;

x = x - h;

n += 1;

if (fabs(h) <= eps) {

cout << "Корень уравнения = " << x << "; Значение функции = " << func(x) << "; Кол-во проходов функции = " << n << endl;

}

}

return x;

}

double metod4\_simple\_iteration(double a, double b, double x0, double eps) {

double x = x0, f;

double xn = x;

unsigned int n = 1;

do {

x = xn;

xn = x - func(x) / (0.25 + 1 / x);

n += 1;

} while (fabs(xn - x) >= eps && xn >= a && xn <= b);

if (xn < a || xn > b) {

cout << "На этом отрезке корней нет!" << endl;

}

else {

cout << "Корень уравнения = " << x << "; Значение функции = " << func(x) << "; Кол-во проходов функции = " << n << endl;

}

return xn;

}

double metod5\_sowing(double a, double b, double x0, double eps) {

int n = 0;

double x1 = x0 - func(x0) / ((func(x0) - func(a)) / (x0 - a));

double x2 = x1 - func(x1) / ((func(x1) - func(x0)) / (x1 - x0));

double x = x2;

double xn = x;

do {

x = xn;

xn = x - func(x) / ((func(x) - func(x1)) / (x - x1));

x1 = x;

n++;

} while (fabs(xn - x) >= eps);

cout << "Корень уравнения = " << x << "; Значение функции = " << func(x) << "; Кол-во проходов функции = " << n << endl;

return x;

}

double f\_hord(double x1, double x2) {

double y1 = func(x1);

double y2 = func(x2);

return x2 - y2 \* (x2 - x1) / (y2 - y1);

}

double f\_kas(double x0) {

double y0 = func(x0);

double dy0 = 2 \* x0;

return x0 - y0 / dy0;

}

double komb(double a, double b, double e) {

int n = 0;

double x = a;

double x\_prev = b;

while (fabs(x - x\_prev) > e) {

x\_prev = x;

x = f\_hord(x\_prev, f\_kas(x\_prev));

n++;

}

cout << "Корень уравнения = " << x << "; Значение функции = " << func(x) << "; Кол-во проходов функции = " << n << endl;

return x;

}

double func(double x) {

return pow(log(x), 2) + (0.25 \* x) - 3;

}

double func1(double x) {

return (func(x + 0.1) - func(x - 0.1)) / (2 \* 0.1);

}

double func2(double x) {

return (func(x + 0.1) - 2 \* func(x) + func(x - 0.1)) / (0.1 \* 0.1);

}