Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы численного анализа

**ОТЧЁТ**

к лабораторной работе

на тему

Численное решение нелинейных уравнений

Выполнил: студент группы 253504

Фроленко Кирилл Юрьевич

Проверил: Анисимов Владимир Яковлевич

Минск 2023

Оглавление

[**Цели выполнения задания** 3](#_30j0zll)

[**Краткие теоретические сведения** 4](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.1fob9te)

[**Задание** 11](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.3znysh7)

**Алгоритм решения**12

[**Программная реализация** 1](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.2et92p0)3

[**Полученные результаты** 1](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.tyjcwt)7

[**Оценка** 1](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.3dy6vkm)8

**Тестовые Примеры**  [1](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.3dy6vkm)9

[**Выводы** 23](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.1t3h5sf)

Вариант 28

# **Цели выполнения задания**

* Изучить методы численного решения нелинейных уравнений - методов бисекции, хорд, простой итерации, релаксации, метода Ньютона и его модификаций
* Исследовать скорость сходимости итерационных процедур
* Изучить метод Эйткена ускорения сходимости
* Сравнить число итераций, необходимого для достижения заданной точности вычисления разными методами

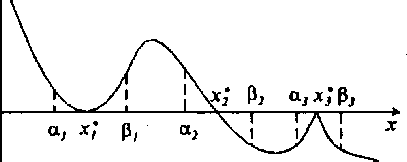
# 

# **Краткие теоретические сведения**

Численное решение нелинейного уравнения *f(x)=0* заключается в вычислении с заданной точностью значения всех или некоторых корней уравнения и распадается на несколько задач: *во-первых,* надо исследовать количество и характер корней (вещественные или комплексные, простые или кратные), *во-вторых,* определить их приближенное расположение, т.е. значения начала и конца отрезка, на котором лежит только один корень, *в- третьих,* выбрать интересующие нас корни и вычислить их с требуемой точностью. Вторая задача называется ***отделением корней****.* Решив ее, по сути дела, находят приближенные значения корней с погрешностью, не превосходящей длины отрезка, содержащего корень. Отметим два простых приема отделения действительных корней уравнения - *табличный* и *графический.* Первый прием состоит в вычислении таблицы значений функции *f(x)* в заданных точках *xt* и использовании следующих теорем математического анализа:

1. *Если функция y=f(x) непрерывна на отрезке [а,b] и f(a)f(b)<0, то внутри отрезка [a,b] существует по крайней мере один корень уравнения f(x)=0.*
2. *Если функция y=f(x) непрерывна на отрезке [а,b], f(a)f(b) < 0 и f '(x) на интервале (a,b) сохраняет знак, то внутри отрезка [a,b] существует единственный корень уравнения f(x)=0.*

Таким образом, если при некотором *k* числа *f(xk)* и *f(xk+1)* имеют разные знаки, то это означает, что на интервале *(xk,xk+1)* уравнение имеет по крайней мере один действительный корень нечетной кратности (точнее – нечетное число корней). Выявить по таблице корень четной кратности очень сложно.

рис.1

На рис.1 представлены три наиболее часто встречающиеся ситуации:

а) кратный корень: *f '(x\*)=0, f(a1)\* f(b1) > 0;*

б) простой корень: *f '(x\*)=0, f(a2)\* f(b2) < 0;*

в) вырожденный корень: *f '(x\*)* не существует, *f(a3)\* f(b3)>0.*

Как видно из рис.1, в первых двух случаях значение корня совпадает с точкой экстремума функции и для нахождения таких корней рекомендуется использовать методы поиска минимума функции.

Для определения числа корней на заданном промежутке используется Теорема Штурма: Если *f(x)* является многочленом и уравнение *f(x)=0* не имеет кратных корней на промежутке [а, b], то число корней этого уравнения, лежащих на таком промежутке, совпадает с числом *N(a) – N(b)*, где функция *N* определяется следующим образом.

Строим ряд Штурма *f0*(*x*), *f1*(*x*)*, f2*(*x*), *…,* *fm*(*x*), где

*f0*(*x*) = *f(x),*

*f1*(*x*) = *f* '*(x),*

*fi*(*x*) = *остаток от деления fi-2*(*x*) *на fi-1(x), взятый с обратным знаком*

Функция *N(x)* определяется как число перемен знака в ряде Штурма, если подставить в функции ряда значение *x*

Для отделения корней можно использовать график функции *y=f(x).* Корнями уравнения являются те значения *х,* при которых график функции пересекает ось абсцисс. Построение графика функции даже с малой точностью обычно дает представление о расположении и характере корней уравнения (иногда позволяет выявить даже корни четной кратности). Если построение графика функции *y=f(x)* вызывает затруднение, следует преобразовать исходное уравнение к виду *φ1(х)=φ2(х)* таким образом, чтобы графики функций *у=φ1(х)* и *у=φ2(х)* были достаточно просты. Абсциссы точек пересечения этих графиков и будут корнями уравнения.

Допустим, что искомый корень уравнения отделен, т.е. найден отрезок *[а, b],* на котором имеется только один корень уравнения. Для вычисления корня с требуемой точностью ε обычно применяют какую-либо итерационную процедуру ***уточнения корня****,* строящую числовую последовательность значений *xn,* сходящуюся к искомому корню уравнения. Начальное приближение *х0* выбирают на отрезке [а, b], продолжают вычисления, пока не выполнится неравенство | *xn-1* – *xn* | < ε, и считают, что *xn* есть корень уравнения, найденный с заданной точностью. Имеется множество различных методов построения таких последовательностей и выбор алгоритма – весьма важный момент при практическом решении задачи. Немалую роль при этом играют такие свойства метода, как простота, надежность, экономичность, важнейшей характеристикой является его *скорость сходимости.* Последовательность *хп,* сходящаяся к пределу *x\**, имеет скорость сходимости порядка α, если при . При α=1 сходимость называется линейной, при 1<α<2 – сверхлинейной, при α=2 – квадратичной. С ростом α алгоритм, как правило, усложняется и условия сходимости становятся более жесткими. Рассмотрим наиболее распространенные итерационные методы уточнения корня.

**Метод простых итераций**. Вначале уравнение *f(x)=0* преобразуется к эквивалентному уравнению вида *х=φ(х).* Это можно сделать многими способами, например, положив *φ(х)=х+♦(x)f(x)*, где *♦(х)* – произвольная непрерывная знакопостоянная функция. Выбираем некоторое начальное приближение *х0* и вычисляем дальнейшие приближения по формуле

*хk= φ(хk-1), k=1,2, ...*

Метод простых итераций не всегда обеспечивает сходимость к корню уравнения. Достаточным условием сходимости этого метода является выполнение неравенства  на отрезке, содержащем корень и все приближения *хп.* Метод имеет линейную скорость сходимости и справедливы следующие оценки:





Метод имеет простую геометрическую интерпретацию: нахождение корня уравнения *f(х)=0* равносильно обнаружению неподвижной точки функции *х= φ(х),* т.е. точки пересечения графиков функций *у= φ(х)* и *у=х.* Если производная *φ'(х)<0,* то последовательные приближения колеблются около корня, если же производная *φ'(х)>0,* то последовательные приближения сходятся к корню монотонно.

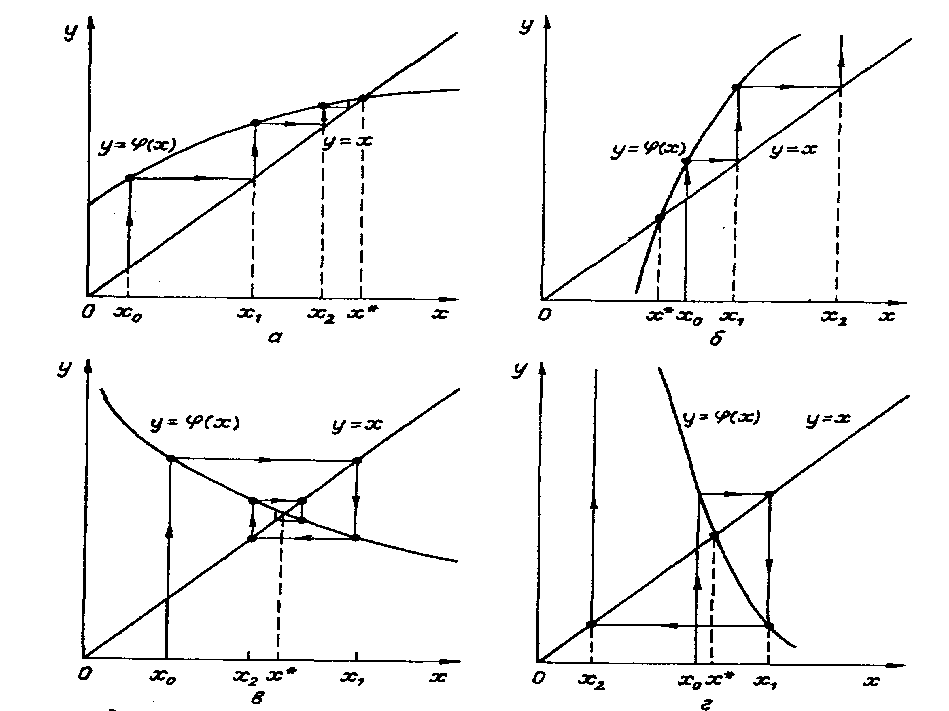
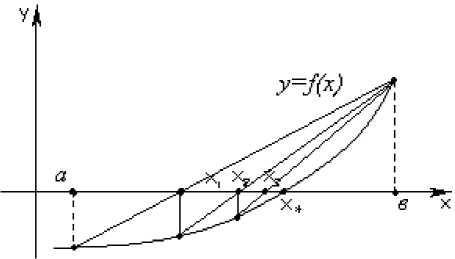


Рис. 2. Метод простых итераций: а - односторонний сходящийся процесс; б - односторонний расходящийся процесс; в - двухсторонний сходящийся процесс; г - двухсторонний расходящийся процесс

Рассмотрим процесс графически (рис. 2). Из графиков видно, что при *φ'(x)<0* и при *φ'(x)>0* возможны как сходящиеся, так и расходящиеся итерационные процессы. Скорость сходимости зависит от абсолютной величины производной *φ(х).* Чем меньше *|φ'(х)|* вблизи корня, тем быстрее сходится процесс.

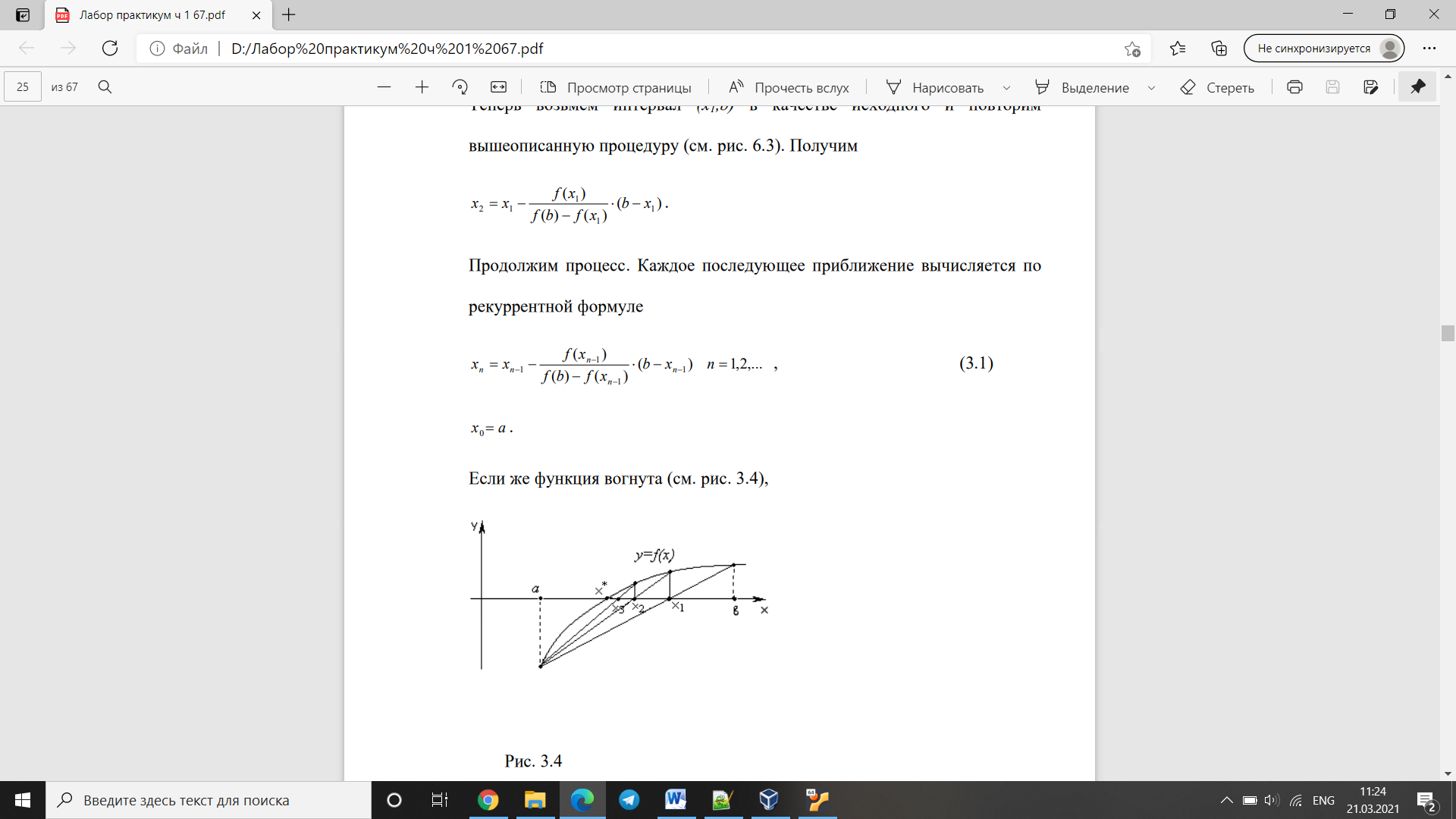
**Метод хорд**. Пусть дано уравнение *f(x) =* 0, *a* ≤ *x* ≤*b*, где *f(x)* – дважды непрерывно дифференцируемая функция. Пусть выполняется условие *f(a)\*f(b)<*0 и проведено отделение корней, то есть на данном интервале *(a, b)* находится один корень уравнения. При этом, не ограничивая общности, можно считать, что *f(b)>0.*

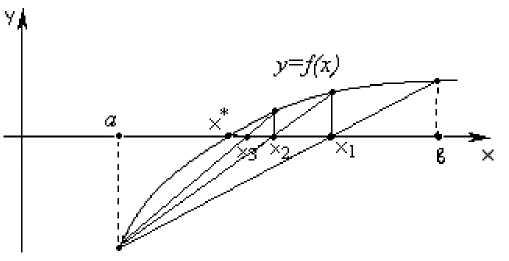
Пусть функция *f* выпукла на интервале *(a, b)* (см. рис. 3).

рис.3

Заменим график функции хордой (прямой), проходящей через точки *M0(a, f(a))* и *M1(b, f(b))*. Уравнение прямой, проходящей через две заданные точки, можно записать в виде  В нашем случае получим:

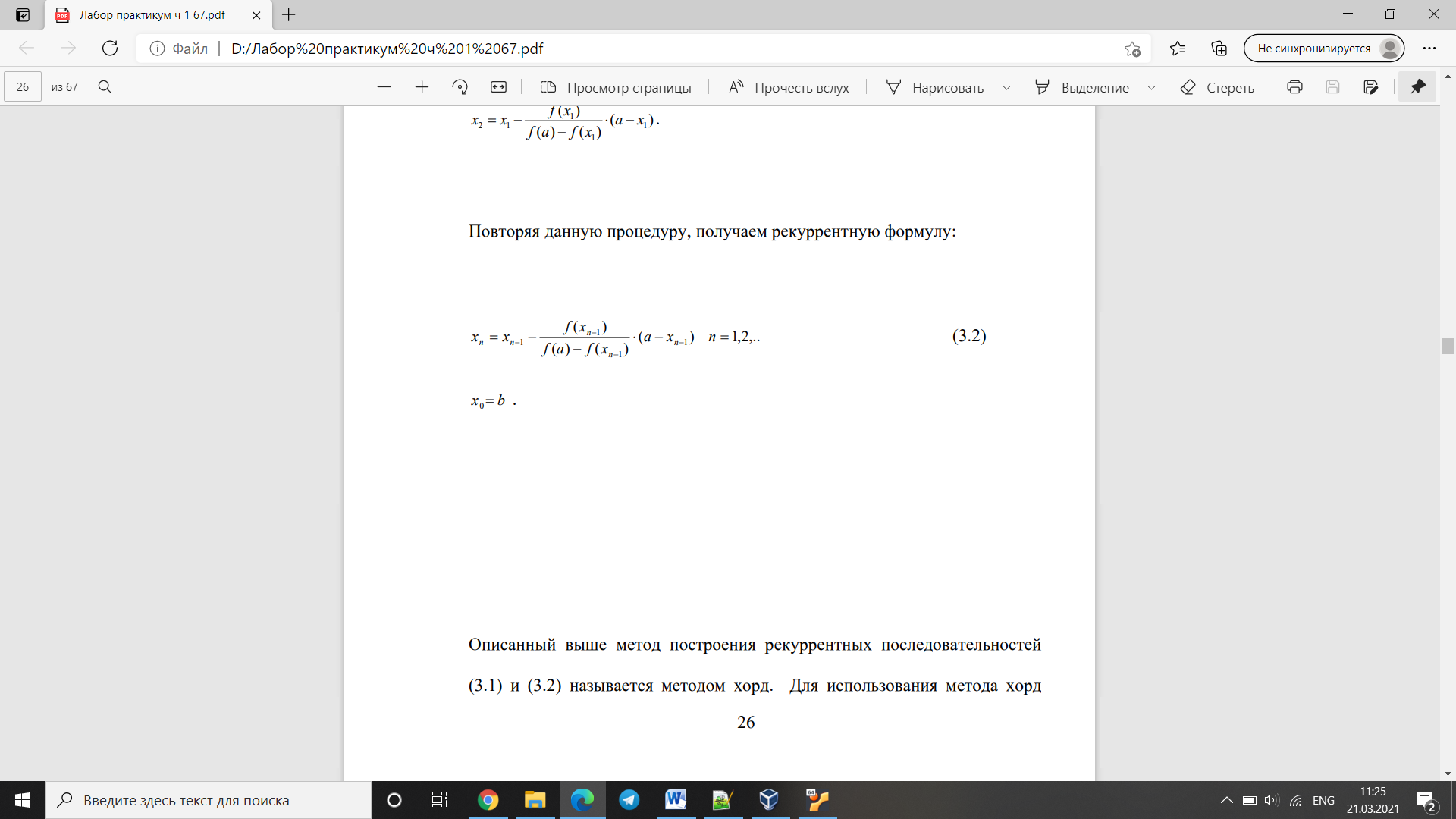
 Найдем точку пересечения  Теперь возьмем интервал *(x1,b)* в качестве исходного и повторим вышеописанную процедуру (см. рис. 3). Получим  Продолжим процесс. Каждое последующее приближение вычисляется по рекуррентной формуле



Если же функция вогнута (см. рис. 4)

уравнение прямой соединяющей точки *M0(a, f(a))* и *M1(b, f(b))* запишем в виде  Найдем точку пересечения хорды с осью Ox:

 Теперь возьмем интервал *(a, x1)* в качестве исходного и найдем точки пересечения хорды, соединяющей точки *(a, f(a))* и *(x1, f(x1)),* с осью абсцисс (см. рис. 4). Получим  Повторяя данную процедуру, получаем рекуррентную формулу:



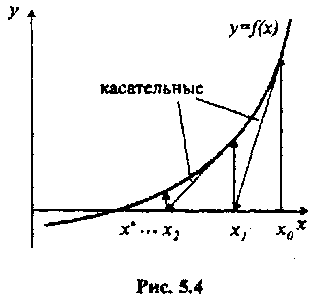
Описанный выше метод построения рекуррентных последовательностей (3.1) и (3.2) называется методом хорд. Для использования метода хорд нужно было бы предварительно найти точки перегиба и выделить участки, на которых функция не меняет характер выпуклости.

Однако на практике поступают проще: в случае  для построения рекуррентной последовательности применяются формулы (3.1), а в случае, когда  применяют формулы (3.2).

**Метод Ньютона** (**касательных**). Для начала вычислений требуется задание одного начального приближения *x0,* последующие приближения вычисляются по формуле

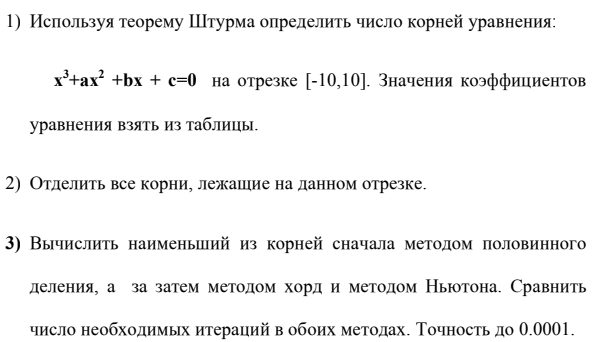


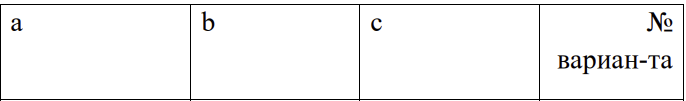
Метод имеет квадратичную скорость сходимости для простого корня, но очень чувствителен к выбору начального приближения. При произвольном начальном приближении итерации сходятся, если всюду , в противном случае сходимость будет только при *x0,* достаточно близком к корню. Существует несколько достаточных условий сходимости. Если производные  и  сохраняют знак в окрестности корня, рекомендуется выбирать *x0* так, чтобы  Если, кроме этого, для отрезка *[a,b],* содержащего корень, выполняются условия  то метод сходится для любых *a ≤ x0 ≤ b.*

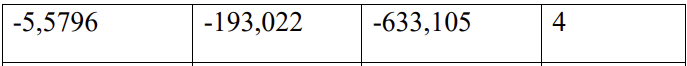
рис.5

Метод Ньютона получил также второе название *метод касательных* благодаря геометрической иллюстрации его сходимости, представленной на рис. 5. Метод Ньютона позволяет находить как простые, так и кратные корни. Основной его недостаток – малая область сходимости и необходимость вычисления производной.

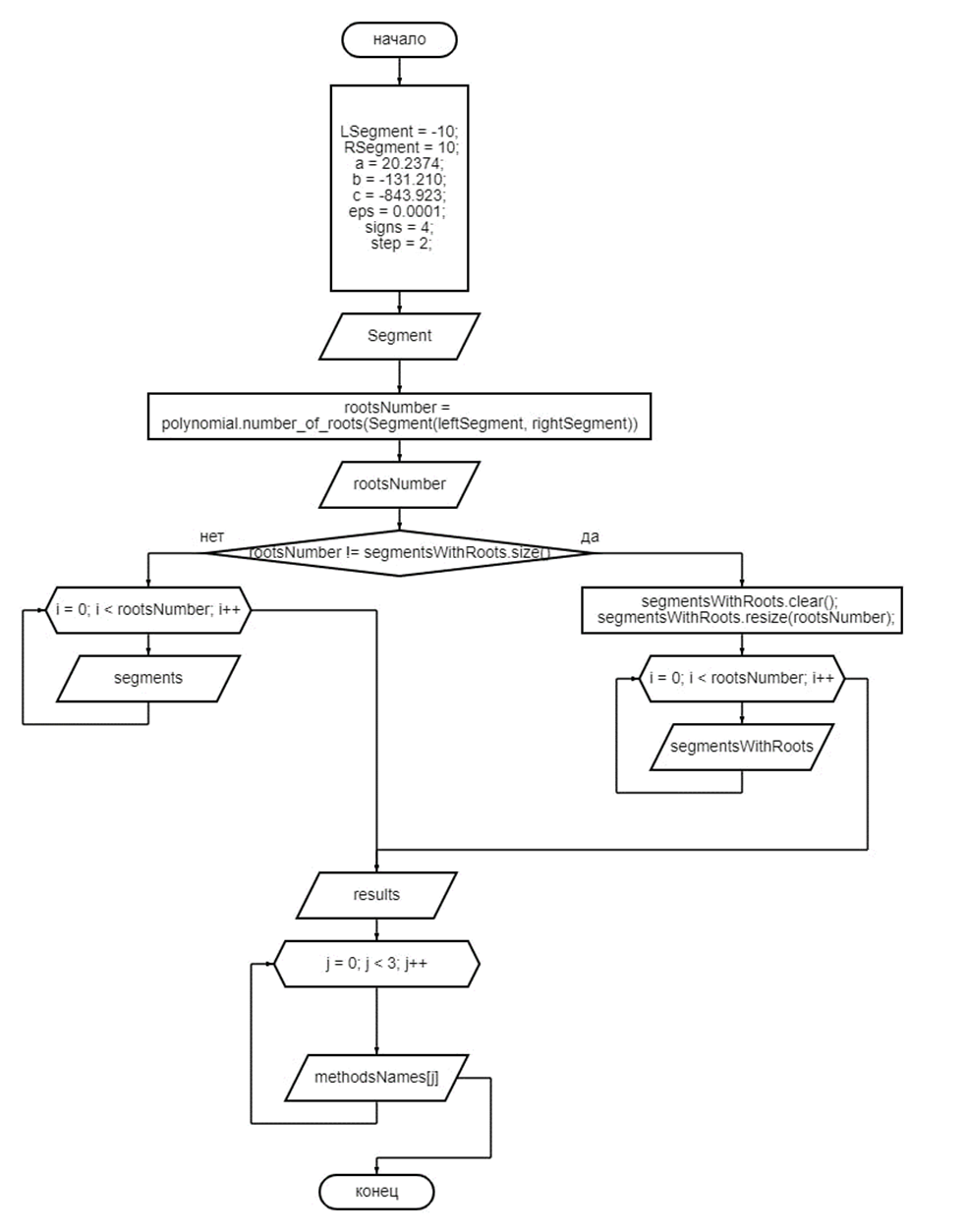
**Задание**

****

****

****

# **Алгоритм решения**



# 

# **Программная реализация**

#include <bits/stdc++.h>

using namespace std;

#define ITERATIONS 1000

#define ll long long

#define ld long double

using namespace std;

ll binpow(ll a, ll n) {

if (n == 0)

return 1;

if (n % 2 == 1)

return binpow(a, n - 1) \* a;

else {

ll b = binpow(a, n / 2);

return b \* b;

}

}

struct Result {

ld rootOfEq;

ll iterations;

Result(ld root = 0, ll iterations = 0) {

this->rootOfEq = root;

this->iterations = iterations;

}

};

struct Segment {

ld a;

ld b;

Segment(ld a = 0, ld b = 0) {

this->a = a;

this->b = b;

}

};

class Polynomial {

private:

ld a;

ld b;

ld c;

//Задание 1

ld f0(ld x) { return binpow(x, 3) + a \* pow(x, 2) + b \* x + c; }

ld f1(ld x) { return 3 \* binpow(x, 2) + 2 \* a \* x + b; }

//Задание 2

ld f2(ld x) { return ((2.0 / 9.0) \* binpow(a, 2) \* x - (2.0 / 3.0) \* b \* x + (1.0 / 9.0) \* a \* b - c); }

//Задание 3

ld f3(ld x) {

ld numerator = 4 \* pow(a, 3) \* c - pow(a \* b, 2) - 18 \* a \* b \* c + 4 \* pow(b, 3) + 27 \* pow(c, 2);

ld denominator = binpow(a \* a - 3 \* b, 2);

return -(9.0 / 4.0) \* (numerator / denominator);

}

// Метод Штурма

ll N(ld x) {

vector<ld> val(4);

val[0] = f0(x);

val[1] = f1(x);

val[2] = f2(x);

val[3] = f3(x);

ll count = 0;

for (ll i = 0; i < 3; i++)

if (val[i] \* val[i + 1] < 0)

count++;

return count;

}

public:

Polynomial(ld a, ld b, ld c) {

this->a = a;

this->b = b;

this->c = c;

}

ll number\_of\_roots(Segment segment) { return N(segment.a) - N(segment.b); }

vector<Segment> segment\_with\_roots(Segment segment, ld step) {

vector<Segment> segments;

for (ld x = segment.a; x < segment.b; x += step)

if (f0(x) \* f0(x + step) < 0)

segments.emplace\_back(x, x + step);

vector<Segment> upd\_segments;

for (ll i = 0; i < segments.size(); i++) {

// Если есть корни четной кратности

if (number\_of\_roots(segments[i]) != 1)

for (const auto& item : segment\_with\_roots(segments[i], step / 2))

upd\_segments.push\_back(item);

else

upd\_segments.push\_back(segments[i]);

}

return upd\_segments;

}

Result half\_division(Segment segment, ld eps) {

// f(a) \* f(b) меньше нуля или только 1 корень на промежутке

if (!(f0(segment.a) \* f0(segment.b) < 0) or number\_of\_roots(segment) != 1)

return Result(-1, -1);

ll iterations = 1;

ld left = segment.a;

ld right = segment.b;

ld middle = (left + right) / 2;

while (abs(f0(middle)) > eps && iterations < ITERATIONS) {

//f(a)\*f((a+b)/2)

if (f0(left) \* f0(middle) < 0)

{

right = middle;

}

else

{

left = middle;

}

middle = (left + right) / 2;

iterations++;

}

return Result(middle, iterations);

}

Result chrod\_method(Segment segment, ld eps) {

// f(a) \* f(b) меньше нуля или только 1 корень на промежутке

if (!(f0(segment.a) \* f0(segment.b) < 0) or number\_of\_roots(segment) != 1)

return Result(-1, -1);

ll iterations = 1;

ld Xn\_prev = 0;

ld Xn\_curr = 0;

// выбор уравнения(f(b)\*f''(b))

if (f0(segment.b) \* (2 \* a + 6 \* segment.b) > 0) {

Xn\_prev = segment.a;

Xn\_curr = Xn\_prev - (f0(Xn\_prev) / (f0(segment.b) - f0(Xn\_prev))) \* (segment.b - Xn\_prev);

while (fabs(Xn\_curr - Xn\_prev) > eps && iterations < ITERATIONS) {

Xn\_prev = Xn\_curr;

Xn\_curr = Xn\_prev - (f0(Xn\_prev) / (f0(segment.b) - f0(Xn\_prev))) \* (segment.b - Xn\_prev);

iterations++;

}

}

else {

Xn\_prev = segment.b;

Xn\_curr = Xn\_prev - (f0(Xn\_prev) / (f0(segment.a) - f0(Xn\_prev))) \* (segment.a - Xn\_prev);

while (fabs(Xn\_curr - Xn\_prev) > eps && iterations < ITERATIONS) {

Xn\_prev = Xn\_curr;

Xn\_curr = Xn\_prev - (f0(Xn\_prev) / (f0(segment.a) - f0(Xn\_prev))) \* (segment.a - Xn\_prev);

iterations++;

}

}

return Result(Xn\_curr, iterations);

}

Result Newthon\_method(Segment segment, ld eps) {

// только 1 корень на промежутке

if (number\_of\_roots(segment) != 1)

return Result(-1, -1);

ll iterations = 1;

ld Xn\_prev = 0;

// выбор Xn

if (f0(segment.b) >= 0)

Xn\_prev = segment.b;

else

Xn\_prev = segment.a;

// Xn - (f(Xn)/f'(Xn))

ld Xn\_curr = Xn\_prev - f0(Xn\_prev) / f1(Xn\_prev);

while (fabs(Xn\_curr - Xn\_prev) > eps && iterations < ITERATIONS) {

Xn\_prev = Xn\_curr;

// Xn+1 = Xn - (f(Xn)/f'(Xn))

Xn\_curr = Xn\_prev - f0(Xn\_prev) / f1(Xn\_prev);

iterations++;

}

if (segment.a < 0 and segment.b < 0 and Xn\_curr>0)

Xn\_curr \*= -1;

return Result(Xn\_curr, iterations);

}

};

int main() {

system("chcp 65001");

ld leftSegment = -10;

ld rightSegment = 10;

ld a = -5.5797;

ld b = -193.023;

ld c = -633.106;

Polynomial polynomial(a, b, c);

//Eps

ld eps = 0.0001;

//Количество знаков после запятой

ll signs = 10;

//Отделять корни табличным методом с шагом

ld step = 1;

cout << "Первое задание:" << endl;

//Вывод отрезка

cout << "Отрезок: [" << setprecision(signs) << leftSegment << ", " << setprecision(signs) << rightSegment << "]" << endl;

ll rootsNumber = polynomial.number\_of\_roots(Segment(leftSegment, rightSegment));

//Кол-во корней

cout << "Количество корней: " << rootsNumber << endl << endl;

vector<Segment> segmentsWithRoots = polynomial.segment\_with\_roots(Segment(leftSegment, rightSegment), step);

cout << "Второе задание:" << endl;

//Если обнаружен корень чётной кратности, то нужно отделить корни вручную

if (rootsNumber != segmentsWithRoots.size()) {

cout << endl << "Обнаружен корень четной кратности." << endl;

//Нужно отделить корни вручную

cout << "Необходимо отделить корень вручную." << endl << endl;

segmentsWithRoots.clear();

segmentsWithRoots.resize(rootsNumber);

for (ll i = 0; i < rootsNumber; i++) {

//Отрезок с корнем

cout << "Отрезок с корнем " << i + 1 << ": " << endl;

cin >> segmentsWithRoots[i].a >> segmentsWithRoots[i].b;

}

}

else {

if (!segmentsWithRoots.empty()) {

//Отрезки

cout << "Отрезки: \n";

for (const auto segment : segmentsWithRoots)

cout << "[" << setprecision(signs) << segment.a << ", " << segment.b << "]" << endl;

}

else {

cout << "Нет корней на отрезке [" << leftSegment << ", " << rightSegment << "]";

}

}

cout << endl;

cout << "Третье задание:";

if (rootsNumber) {

//Метод половинного деления

map<ll, string> methodsNames({ make\_pair(0, "Метод половинного деления: "),

//Метод хорд

make\_pair(1, "Метод хорд: "),

//Метод Ньютона

make\_pair(2, "Метод Ньютона: ") });

map<string, vector<Result>> results;

for (ll i = 0; i < rootsNumber; i++) {

results["Метод половинного деления: "].push\_back(polynomial.half\_division(segmentsWithRoots[i], eps));

results["Метод хорд: "].push\_back(polynomial.chrod\_method(segmentsWithRoots[i], eps));

results["Метод Ньютона: "].push\_back(polynomial.Newthon\_method(segmentsWithRoots[i], eps));

}

for (ll i = 0; i < rootsNumber; i++) {

cout << endl << "Отрезок [" << setprecision(signs) << segmentsWithRoots[i].a

<< ", " << setprecision(signs) << segmentsWithRoots[i].b << "]: " << endl;

for (ll j = 0; j < 3; j++) {

cout << setw(27) << right << methodsNames[j];

cout << setw(signs + 3) << right << setprecision(signs) << results[methodsNames[j]][i].rootOfEq

// Кол-во итераций

<< " Количество итераций: " << results[methodsNames[j]][i].iterations << endl;

}

}

}

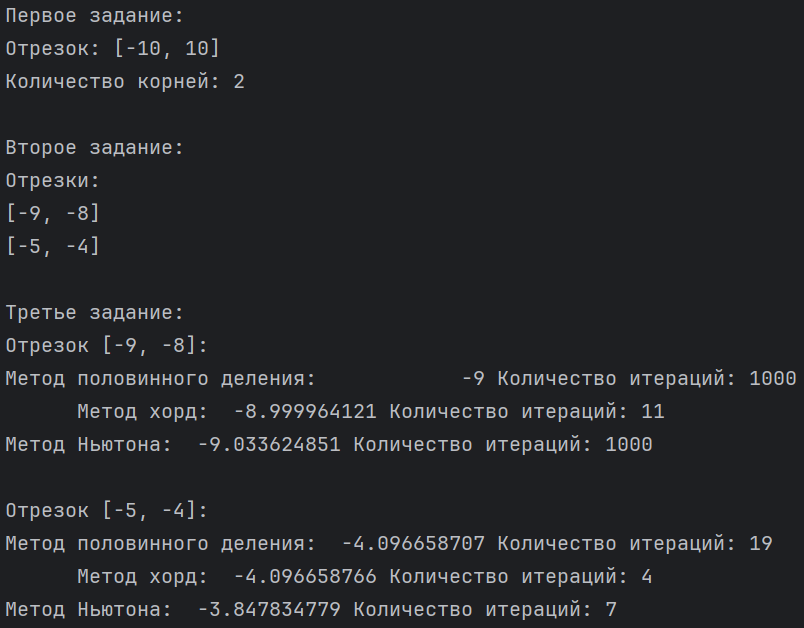
else {

cout << endl << "Нет корней на отрезке [" << leftSegment << ", " << rightSegment << "]" << endl;

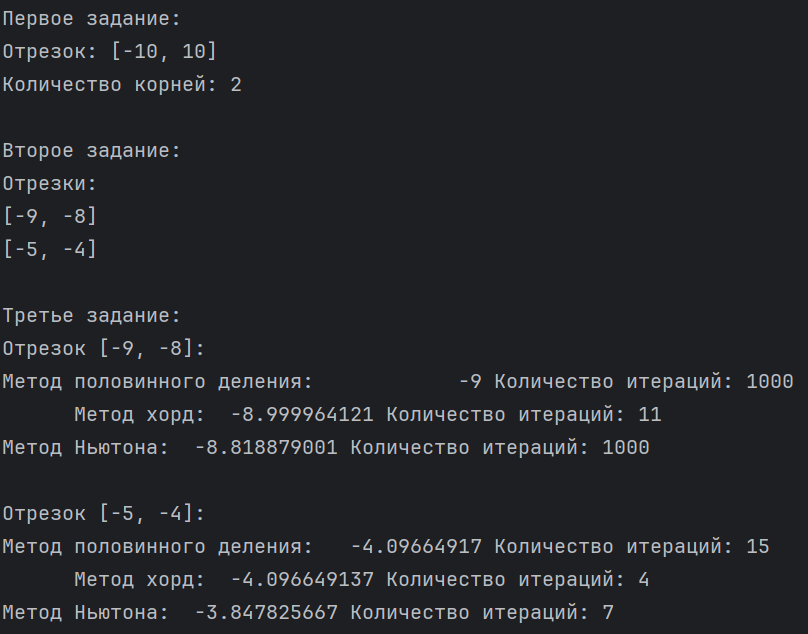
}

}

**Полученные результаты программы**

Если учесть, что числа при вводе заданы точно то: 

А теперь возмемем приближенные значения в соответствии с вариантом:



# **Оценка**

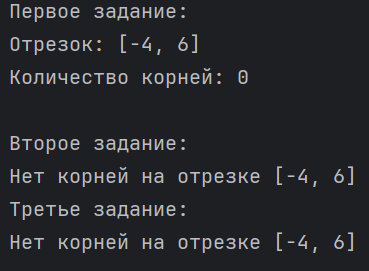
Оценим точность наших измерений и сделаем вывод:

Точность в 0.0001 достигнута во всех методах, при вводе приближенных значений.

**Тестовые примеры**

**Пример 1**

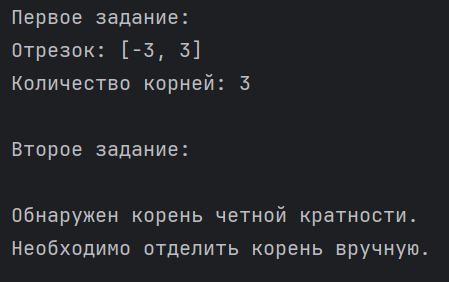
a = 20.2374, b = -131.2100, c = -843.9230 отрезок: [-4,6]:

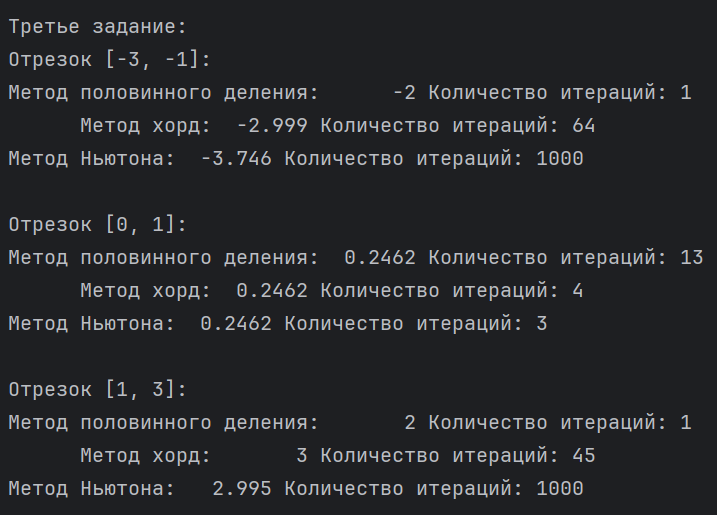


**Пример 2**

a = -0.2500, b = -4.0000, c = 1.0000 отрезок: [-3,3]:

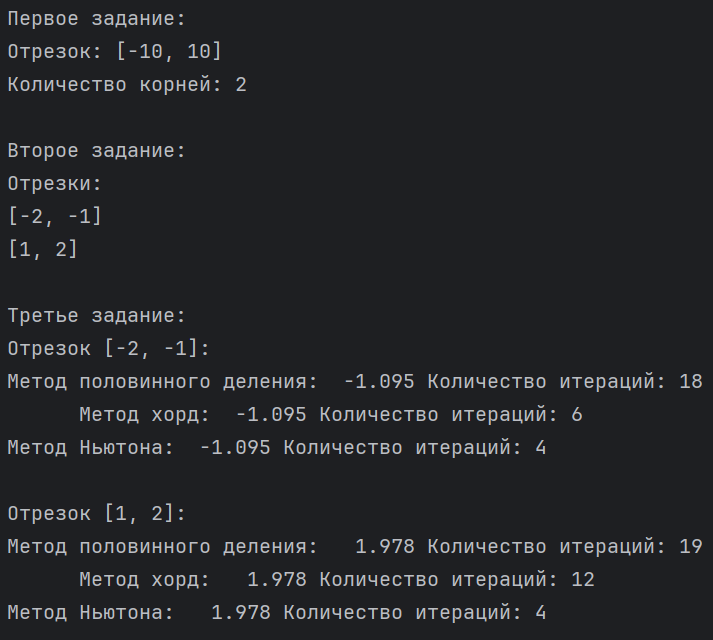
(-3,-1; 0,1; 1,3)





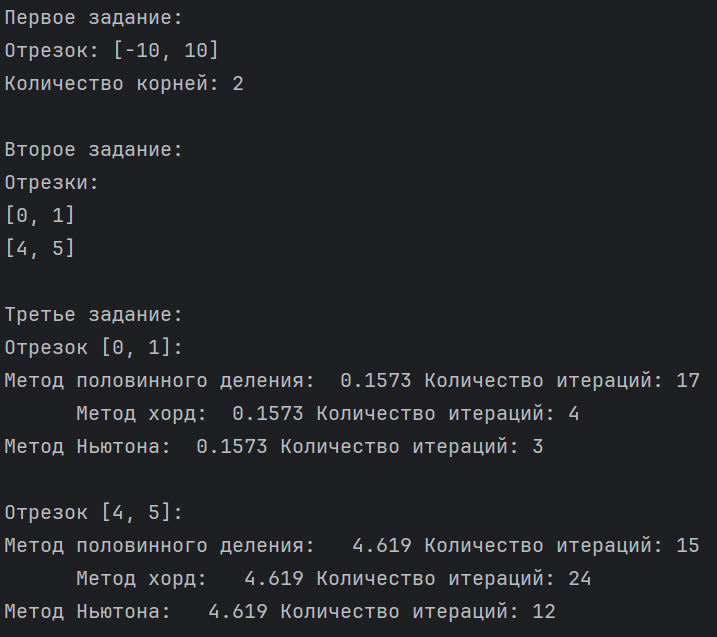
**Пример 3**

a = 57.2345, b = -51.2100, c = -123.7230 отрезок: [-10,10]:



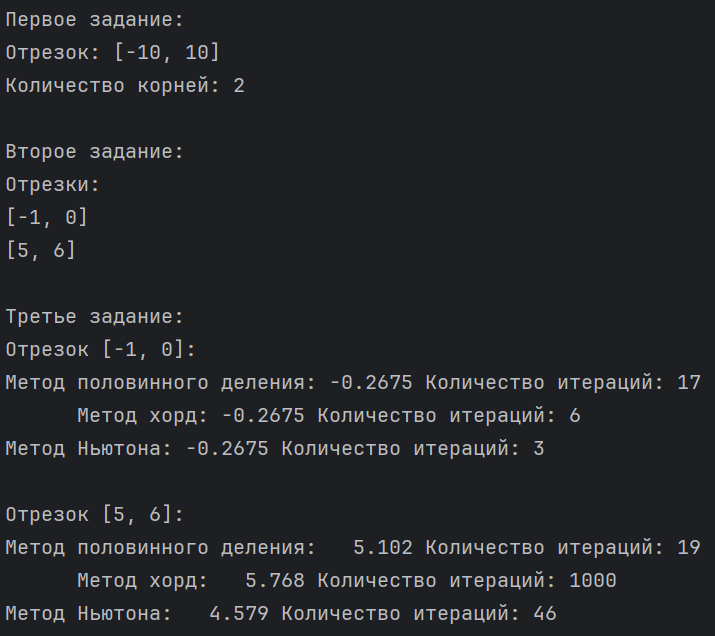
**Пример 4**

a = 15.7845, b = -89.7345, c = 13.7230 отрезок: [-10,10]:

****

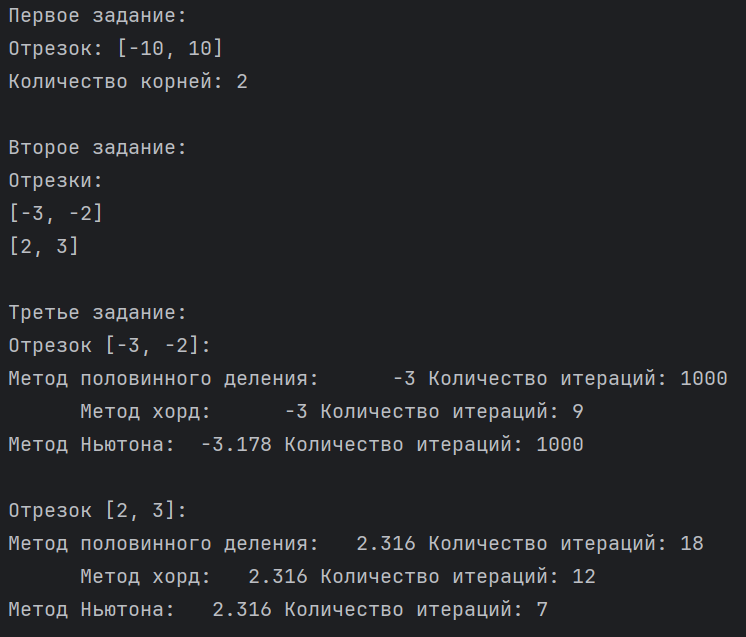
**Пример 5**

a = -19.2310, b = 69.7005, c = 20.0230 отрезок: [-10,10]:



**Пример 6**

a = -10.9991, b = -12.7595, c = 80.5320 отрезок: [-10,10]:

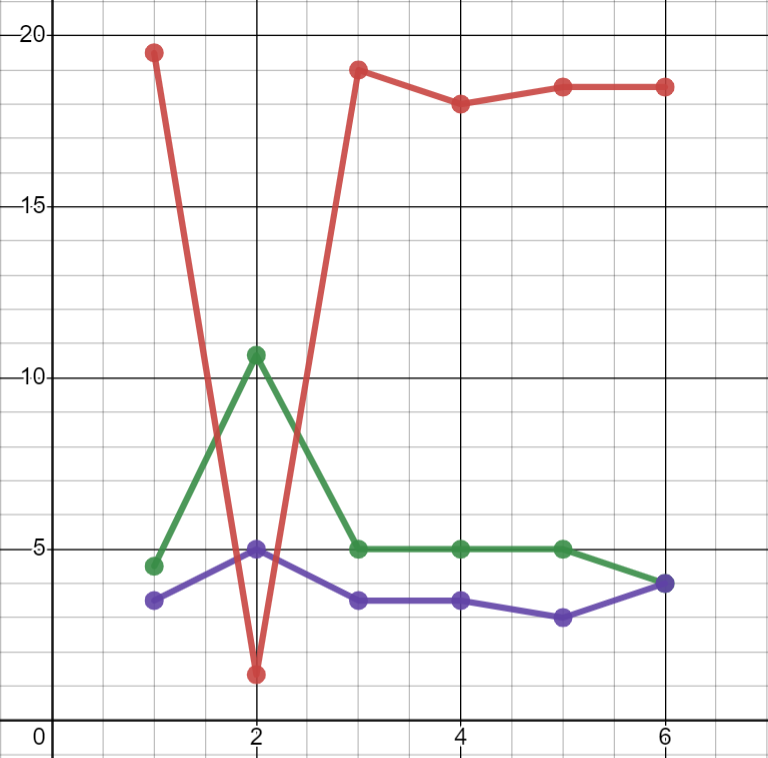


**Оценка эффективности методов**

**Подсчет среднего числа итераций каждого из методов**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Название метода** | **Номер теста** | | | | | | **Среднее количество итераций** |
| **1** | **2** | **3** | **4** | **5** | **6** |
| **Метод половинного деления** | **20/19** | **1/2/1** | **20/18** | **17/19** | **19/18** | **19/18** | **14,69** |
| **Метод хорд** | **4/5** | **13/4/15** | **6/4** | **4/6** | **6/4** | **4/4** | **6,07** |
| **Метод Ньютона** | **3/4** | **8/2/5** | **4/3** | **3/4** | **3/3** | **5/3** | **3,84** |

**Графики средних значений количества итераций для каждого из методов**

****

Где красный – метод половинного деления, зеленый – метод хорд, фиолетовый – метод Ньютона.

**Выводы**

Таким образом, в ходе выполнения лабораторной работы были изучены методы численного решения нелинейных уравнений (метод простой итерации, метод хорд, метод Биссекции, метод Ньютона), исследована скорость сходимости итерационных процедур, составлена программа численного решения нелинейных уравнений методами бисекции, хорд, Ньютона, проверена правильность работы программы на тестовых примерах, численно решено нелинейное уравнение заданного варианта, сравнены количества итераций, необходимых для достижения заданной точности вычисления разными методами.

Оптимальным способом численного решения нелинейных уравнений является применение метода Ньютона, так скорость сходимости в этом методе почти всегда квадратичная, однако нельзя забывать об ограничениях, которые присутствуют в методах, преобразовании исходного уравнения(Метод простых итераций) и определении начального приближения(Метод Ньютона), что не нужно для метода половинного деления. В ходе работы были рассмотрены 6 функций, имеющих несколько корней на заданном отрезке (-10, 10), и в ходе решения результат был проверен с помощью графика, что дает понять, что реализованные методы успешно справляются с решением нелинейных уравнений.