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 153503

Кончик Денис Сергеевич

Проверил: Анисимов Владимир Яковлевич

Минск 2022

**CОДЕРЖАНИЕ**

1. ЦЕЛЬ

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

3. АЛГОРИТМ РЕШЕНИЯ

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

5. ТЕСТОВЫЕ РЕЗУЛЬТАТЫ

6. ЗАДАНИЕ

7. ВЫВОД

# **ЦЕЛЬ**

1. Изучить методы отделения корней – табличный и графический, теорему Штурма для определения числа корней на заданном промежутке.
2. Изучить итерационные методы уточнения корня – метод половинного деления, метод хорд, метод Ньютона.
3. Составить алгоритмы для решения уравнений вида :
   1. Табличный метод отделения корней.
   2. Определение числа корней на заданном промежутке (т. Штурма).
   3. Уточнение корня: методы половинного деления, хорд, Ньютона.
4. Составить программу решения соответствующих уравнений по составленным алгоритмам.
5. Решить тестовые примеры и задание, проверить правильность работы программы. Сравнить число необходимых итераций в трех методах уточнения корней.

# **КРАТКИЕ ТЕОРИТИЧЕСКИЕ 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.

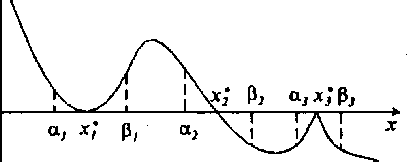


Рис. 1

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.

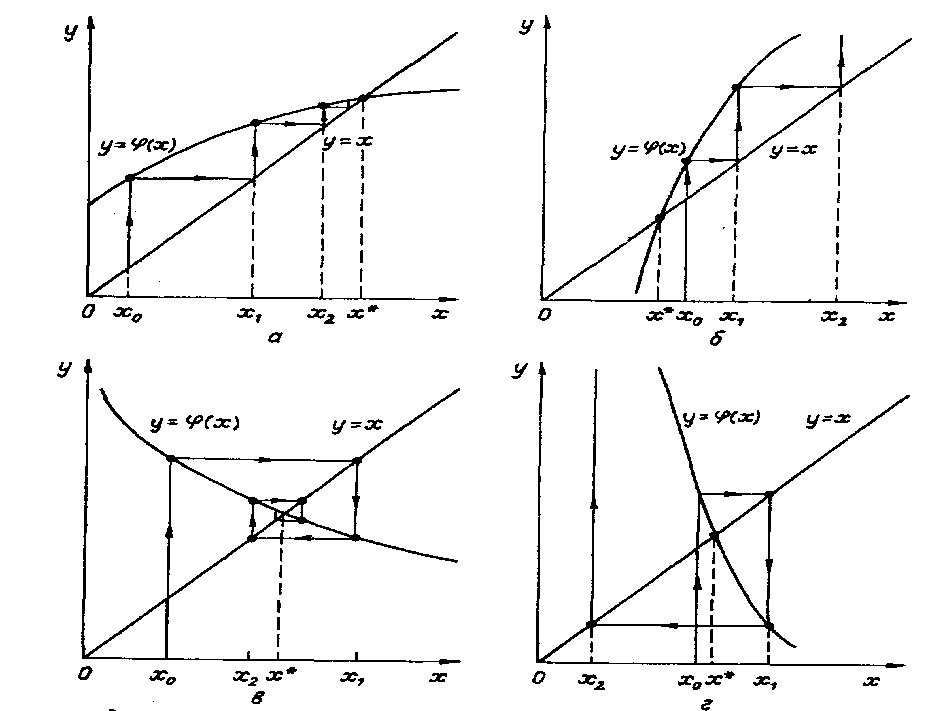


Рис. 2. Метод простых итераций: а - односторонний сходящийся процесс; б - односторонний расходящийся процесс; в - двухсторонний сходящийся процесс; г - двухсторонний расходящийся процесс

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).

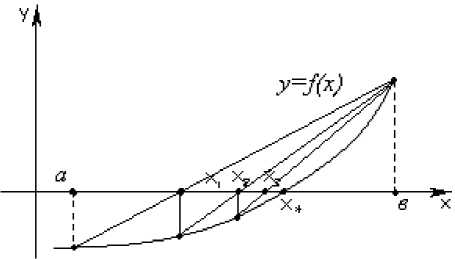


Рис. 3

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 хорды с осью Ox. Полагая у = 0, получаем из предыдущего уравнения: . 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 (3.1)

,

.

Если же функция вогнута (см. рис. 4),

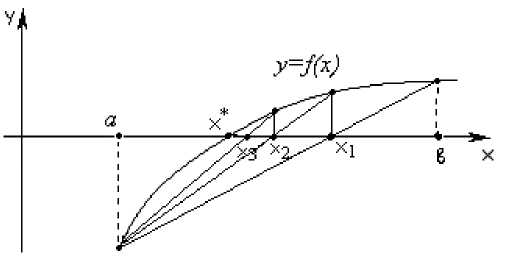


Рис. 4

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:

(3.2)

,

.

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.

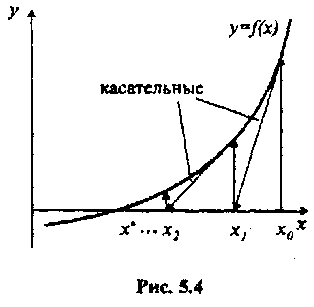


Рис. 5

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.

# **АЛГОРИТМЫ**

**1. Табличный метод отделения корней**



**2. Определение числа корней на заданном промежутке**



**3. Уточнение корня: метод половинного деления.**

**4. Уточнение корня: метод хорд.**



**5. Уточнение корня: метод Ньютона.**



# **ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

#include <iostream>

#include <vector>

#include <exception>

#include <iomanip>

#include <map>

#include "Polynomial.h"

#include "Result.h"

using namespace std;

// x^3 + ax^2 + cx + d = 0

int main() {

setlocale(LC\_ALL, "Rus");

cout.setf(ios\_base::fixed);

double LSegment = -10;

double RSegment = 10;

double a = 6.0951;

double b = -35.3942;

double c = -25.7283;

Polynomial polynomial(a, b, c);

cout << "Считать корни с точностью eps = ";

double eps; cin >> eps;

cout << "Выводить знаков после запятой в корнях: ";

int signs = 0; cin >> signs;

cout << "Отделять корни табличным методом с шагом: ";

double step = 0; cin >> step;

cout << "Отрезок: [" << setprecision(signs) << LSegment << ", "

<< setprecision(signs) << RSegment << "]\n";

int rootsNumber = polynomial.RootsNumber(Segment(LSegment, RSegment));

cout << "Корней: " << rootsNumber << "\n\n";

vector<Segment> segmentsWithRoots = polynomial.SegmentsWithRoots(Segment(LSegment, RSegment), step);

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

cout << "Обнаружен корень четной кратности. \n";

cout << "Необходимо отделить корни вручную. \n\n";

segmentsWithRoots.clear();

segmentsWithRoots.resize(rootsNumber);

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

cout << "Отрезок с корнем " << i + 1 << ": \n";

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

}

}

else {

cout << "Отрезки: \n";

for (const auto segment : segmentsWithRoots)

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

}

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

make\_pair(1, "Метод хорд: "),

make\_pair(2, "Метод Ньютона: ") });

map<string, vector<Result>> results;

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

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

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

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

}

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

cout << "\nОтрезок [" << setprecision(signs) << segmentsWithRoots[i].a

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

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

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

if (results[methodsNames[j]][i].iterations == -1)

cout << "решение данным методом на данном отрезке невозможно \n";

else

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

<< ". Итераций: " << results[methodsNames[j]][i].iterations << "\n";

}

}

}

#pragma once

#include "Segment.h"

#include "Result.h"

#include <cmath>

#include <vector>

using namespace std;

#define MAX\_ITERATIONS 1000

class Polynomial {

private:

double a;

double b;

double c;

double f0(double x) {

return pow(x, 3) + a \* pow(x, 2) + b \* x + c;

}

double f1(double x) {

return 3 \* pow(x, 2) + 2 \* a \* x + b;

}

double f2(double x) {

return ((2.0 / 9.0) \* pow(a, 2) \* x - (2.0 / 3.0) \* b \* x + (1.0 / 9.0) \* a \* b - c);

}

double f3(double x) {

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

double denominator = pow(a \* a - 3 \* b, 2);

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

}

int N(double x) {

vector<double> values(4);

values[0] = f0(x);

values[1] = f1(x);

values[2] = f2(x);

values[3] = f3(x);

int counter = 0;

for (int i = 0; i < 3; i++) {

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

++counter;

}

return counter;

}

public:

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

this->a = a;

this->b = b;

this->c = c;

}

int RootsNumber(Segment segment) {

return N(segment.a) - N(segment.b);

}

vector<Segment> SegmentsWithRoots(Segment segment, double step) {

vector<Segment> segments;

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

if (RootsNumber(segments[i]) != 1) {

for (const auto& item : SegmentsWithRoots(segments[i], step / 2))

upd\_segments.push\_back(item);

}

else {

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

}

}

return upd\_segments;

}

Result HalfDivision(Segment segment, double eps) {

if (!(f0(segment.a) \* f0(segment.b) < 0) ||

RootsNumber(segment) != 1)

return Result(-1, -1);

int iterations = 1;

double left = segment.a;

double right = segment.b;

double mid = (left + right) / 2;

while (abs(f0(mid)) > eps && iterations < MAX\_ITERATIONS) {

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

right = mid;

else

left = mid;

mid = (left + right) / 2;

++iterations;

}

return Result(mid, iterations);

}

Result ChordMethod(Segment segment, double eps) {

if (!(f0(segment.a) \* f0(segment.b) < 0) ||

RootsNumber(segment) != 1)

return Result(-1, -1);

int iterations = 1;

double Xn\_prev = 0;

double Xn\_curr = 0;

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 < MAX\_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 < MAX\_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 NewtonMethod(Segment segment, double eps) {

if (RootsNumber(segment) != 1)

return Result(-1, -1);

int iterations = 1;

double Xn\_prev = 0;

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

Xn\_prev = segment.b;

else

Xn\_prev = segment.a;

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

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

Xn\_prev = Xn\_curr;

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

++iterations;

}

return Result(Xn\_curr, iterations);

}

};

struct Result {

double root;

int iterations;

Result(double root = 0, int iterations = 0) {

this->root = root;

this->iterations = iterations;

}

};

struct Segment {

double a;

double b;

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

this->a = a;

this->b = b;

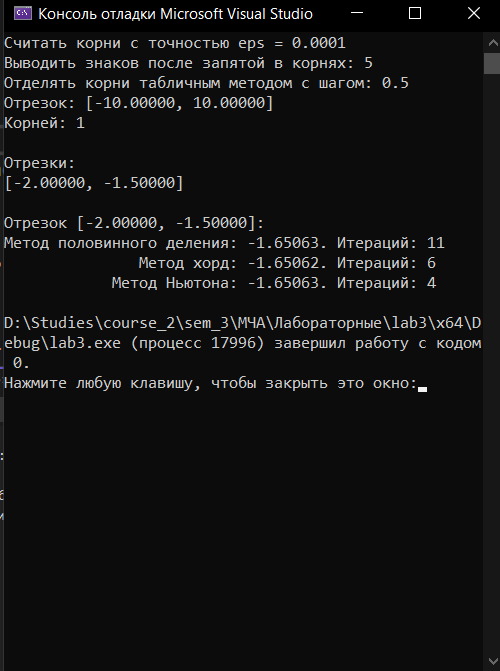
}

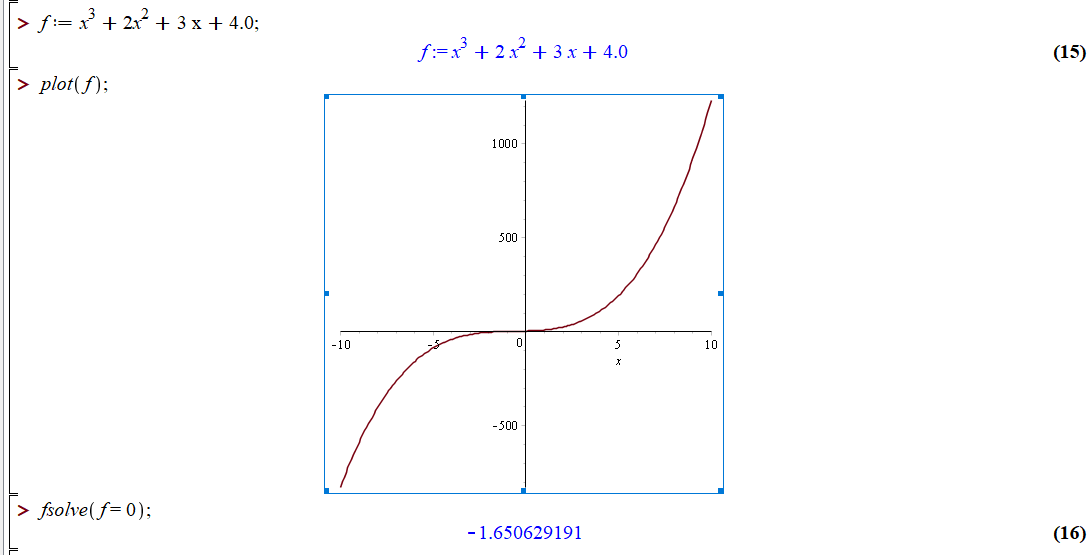
};

# **ТЕСТОВЫЕ ПРИМЕРЫ**

**Тестовый пример 1.**

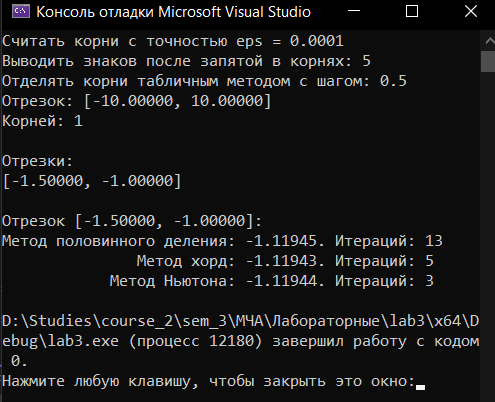
Вычислить корни уравнения на промежутке [-10,10] с точностью 0.0001.



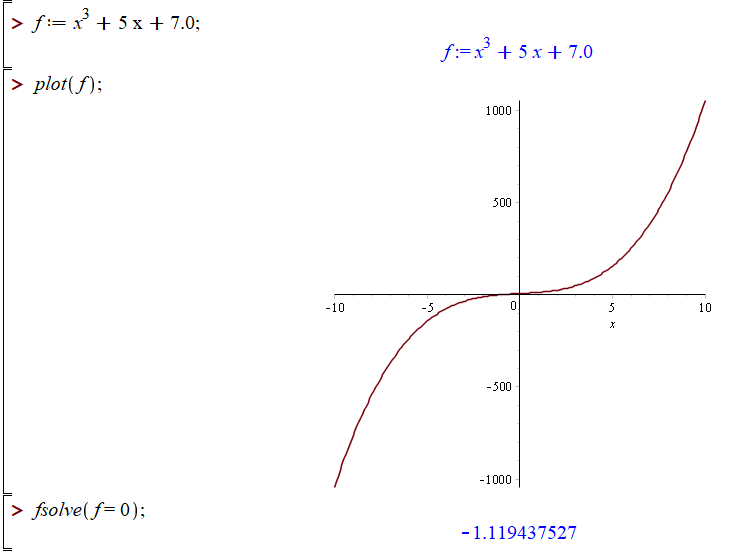
Проверка результата в Maple:

**Тестовый пример 2.**

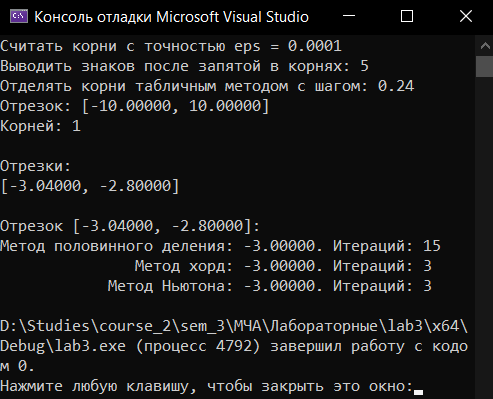
Вычислить корни уравнения на промежутке [-10,10] с точностью 0.0001.

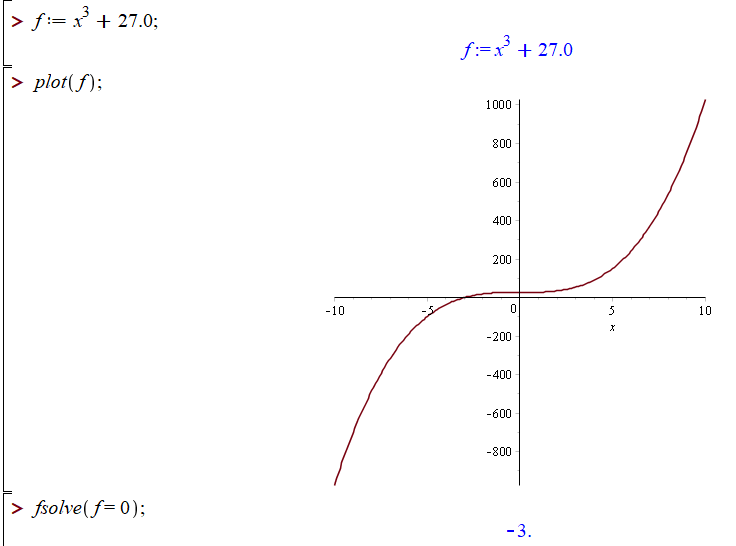


Проверка результата в Maple:



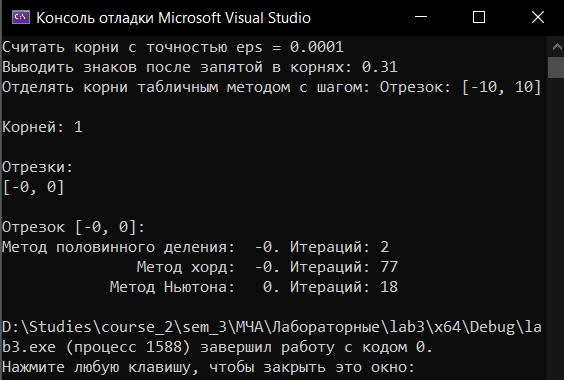
**Тестовый пример 3.**

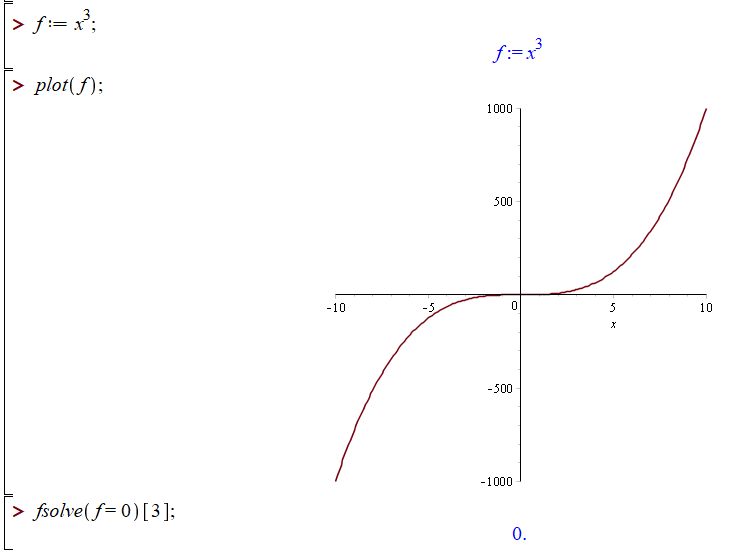
Вычислить корни уравнения на промежутке [-10,10] с точностью 0.0001.

Проверка результата в Maple:

**Тестовый пример 4.**

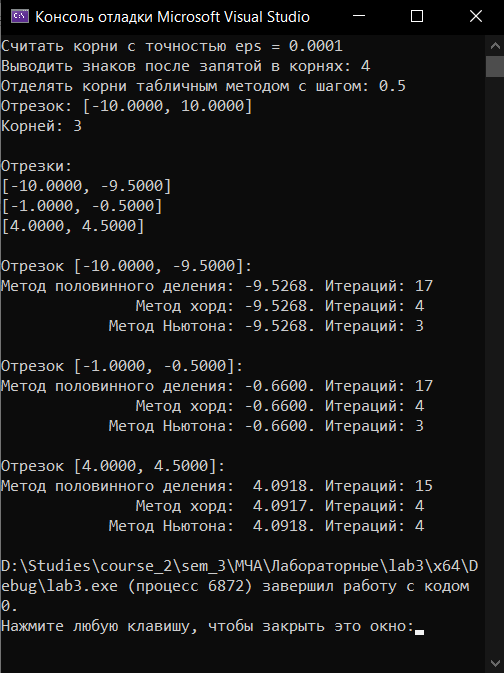
Вычислить корни уравнения на промежутке [10,10] с точностью 0.0001.



Проверка результата в Maple:

# **ЗАДАНИЕ**

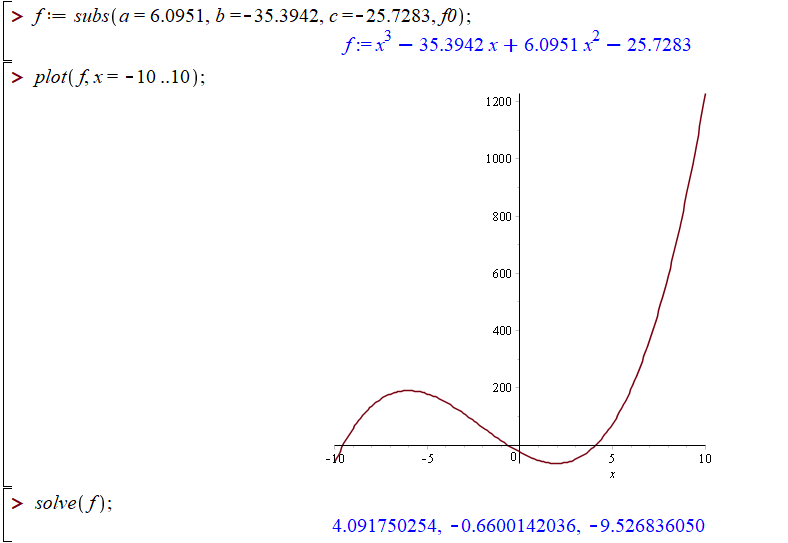
Вариант 12.

a = 6.0951, b = -35.3942,

c = -25.7283

В данном случае метод половинного деления требудет большее количество ресурсов по сравнению с остальными методами из-за большего числа итераций.

Проверка результата в Maple:



# **ВЫВОД**

Таким образом, в ходе выполнения лабораторной работы были изучены методы численного решения нелинейных уравнений (метод половинного деления, метод хорд, метод Ньютона), составлен алгоритм и программа численного решения нелинейных уравнений, проверена правильность работы программы на тестовых примерах, численно решено нелинейное уравнение заданного варианта, определено количество итераций, необходимых для достижения заданной точности вычисления разными методами.

Можно сделать вывод о том, что оптимальным методом уточнения корней в решенных примерах является метод Ньютона.