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 253505
Снежко Максим Андреевич

Проверил: Анисимов Владимир Яковлевич

Минск 2023

Вариант 10

Цель работы

1) Изучить методы численного решения нелинейных уравнений (метод половинного деления (биссекции), метод хорд, метод Ньютона)

2) Составить программу численного решения нелинейных уравнений методами половинного деления (биссекции), хорд, Ньютона

3) Проверить правильность работы программы на тестовых примерах

4) Численно решить нелинейное уравнение заданного варианта

5) Сравнить число итераций, необходимого для достижения заданной точности вычисления разными методами

6) Составить алгоритм решения СЛАУ указанными методами, применимый для организации вычислений на ЭВМ;

7) Составить программу решения СЛАУ по разработанному алгоритму;

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 x_i и использовании следующих теорем математического анализа:

1. *Если функция $y=f(x)$ непрерывна на отрезке $[a,b]$ 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 $[a,b]$, $f(a)f(b) < 0$ 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 $f(x_k)$ и $f(x_{k+1})$ имеют разные знаки, то это означает, что на интервале (x_k,x_{k+1}) уравнение имеет по крайней мере один действительный корень нечетной кратности (точнее – нечетное число корней). Выявить по таблице корень четной кратности очень сложно.

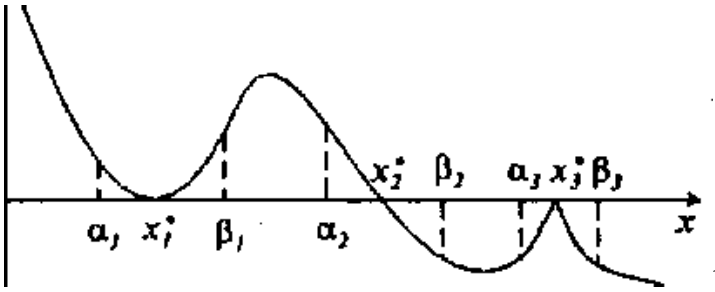


рис.1

На рис.1 представлены три наиболее часто встречающиеся ситуации:

а) кратный корень: $f'(x^*)=0$, $f(a_1)^* f(b_1) > 0$;

б) простой корень: $f'(x^*)=0$, $f(a_2)^* f(b_2) < 0$;

в) вырожденный корень: $f'(x^*)$ не существует, $f(a_3)^* f(b_3)>0$.

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 [a, 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 $f_0(x)$, $f_1(x)$, $f_2(x)$, ..., $f_m(x)$, где

$f_0(x) = f(x)$,

$f_1(x) = f'(x)$,

$f_i(x)$ = *остаток от деления* $f_{i-2}(x)$ *на* $f_{i-1}(x)$, *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 *x,* 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 $\varphi_1(x)=\varphi_2(x)$ таким образом, чтобы графики функций $y=\varphi_1(x)$ и $y=\varphi_2(x)$ были достаточно просты. Абсциссы точек пересечения этих графиков и будут корнями уравнения.

Допустим, что искомый корень уравнения отделен, т.е. найден отрезок *[a, 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 x_n, сходящуюся к искомому корню уравнения. Начальное приближение x_0 выбирают на отрезке [a, b], продолжают вычисления, пока не выполнится неравенство $| x_{n-1} – x_n | < \varepsilon$, и считают, что x_n 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 x_n, сходящаяся к пределу *x**, имеет скорость сходимости порядка α, если при . При $\alpha=1$ сходимость называется линейной, при $1<\alpha<2$ – сверхлинейной, при $\alpha=2$ – 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 $x=\varphi(x)$. Это можно сделать многими способами, например, положив $\varphi(x)=x+\blacklozenge(x)f(x)$, где $\blacklozenge(x)$ – произвольная непрерывная знакопостоянная функция. Выбираем некоторое начальное приближение x_0 и вычисляем дальнейшие приближения по формуле

$$x_k = \varphi(x_{k-1}),\ k=1,2,\ldots$$

Метод простых итераций не всегда обеспечивает сходимость к корню уравнения. Достаточным условием сходимости этого метода является выполнение неравенства $¿\varphi'(x)\vee \le q<1$ на отрезке, содержащем корень и все приближения x_n. Метод имеет линейную скорость сходимости и справедливы следующие оценки:

$$\left|x_n - x^{\ddot{\omega}}\right| < \frac{q}{1-q}\left|x_n - x_{n-1}\right|, \text{ если } \varphi'(x) > 0$$

$$\left|x_n - x^{\ddot{\omega}}\right| < \left|x_n - x_{n-1}\right|, \text{ если } \varphi'(x) < 0$$

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

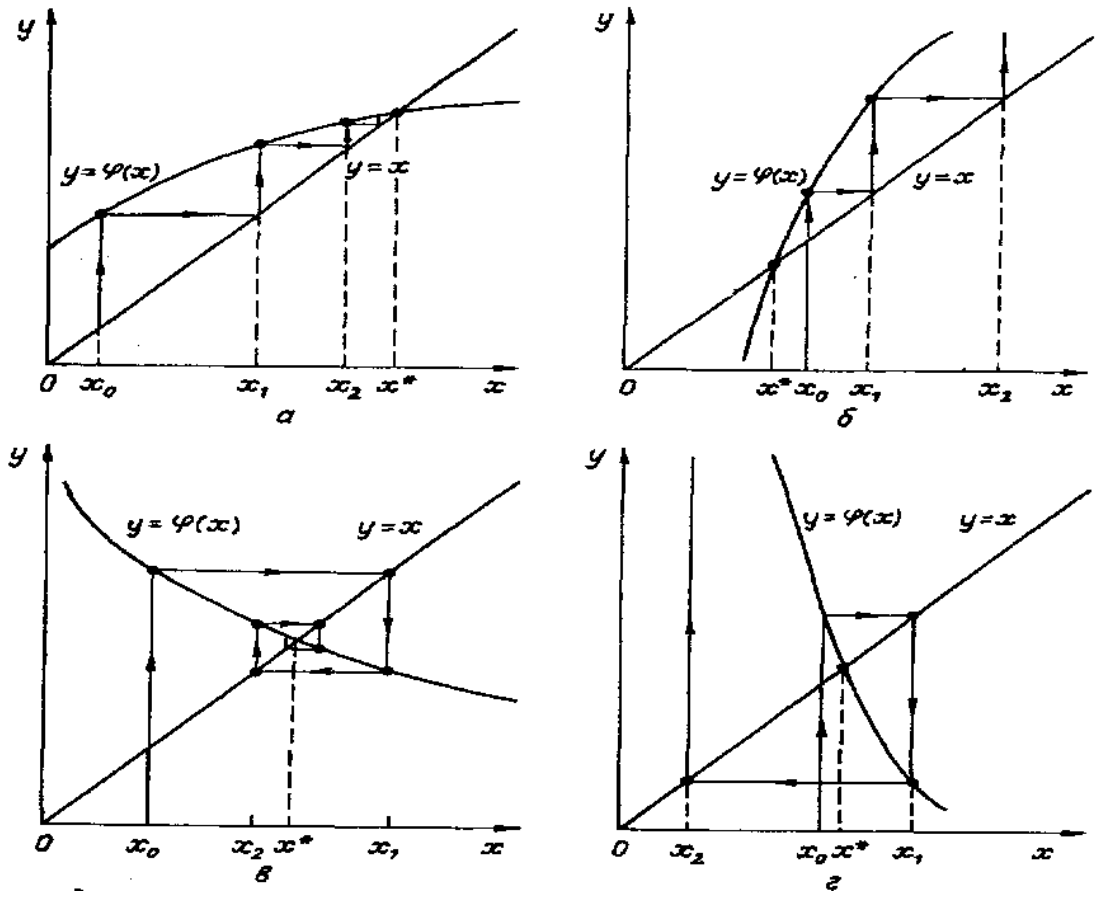


Рис. 2. Метод простых итераций: а - односторонний сходящийся процесс; б - односторонний расходящийся процесс; в - двухсторонний сходящийся процесс; г - двухсторонний расходящийся процесс

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

Метод хорд. Пусть дано уравнение $f(x) = 0$, $a \le x \le b$, 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).

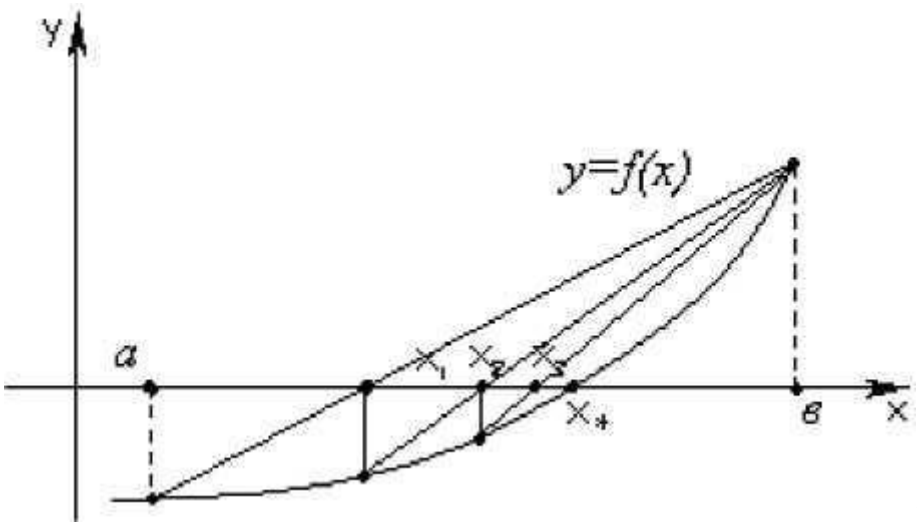


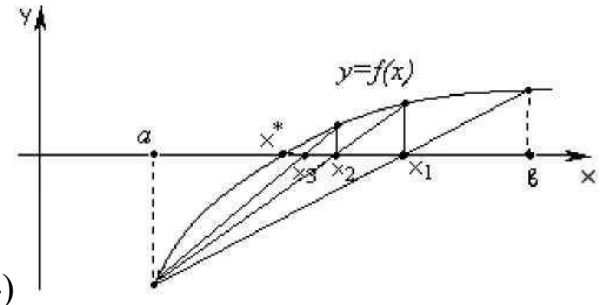
рис.3

Заменим график функции хордой (прямой), проходящей через точки $M_0(a, f(a))$ и $M_1(b, f(b))$. Уравнение прямой, проходящей через две заданные точки, можно записать в виде $\frac{y-y_1}{y_2-y_1}=\frac{x-x_1}{x_2-x_1}$ В нашем случае получим:

$\frac{y-f(a)}{f(b)-f(a)}=\frac{x-a}{b-a}$ Найдем точку пересечения $x_1=a-\frac{f(a)}{f(b)-f(a)}*(b-a)$ Теперь возьмем интервал (x_1,b) в качестве исходного и повторим вышеописанную процедуру (см. рис. 3). Получим $x_2=x_1-\frac{f(x_1)}{f(b)-f(x_1)}*(b-x_1)$ Продолжим процесс. Каждое последующее приближение вычисляется по рекуррентной формуле

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(b)-f(x_{n-1})}\cdot(b-x_{n-1}) \quad n=1,2,\ldots \ , \qquad (3.1)$$

$x_0 = a$.



Если же функция вогнута (см. рис. 4)

уравнение прямой соединяющей точки $M_0(a, f(a))$ и $M_1(b, f(b))$ запишем в виде $\frac{y-f(b)}{f(a)-f(b)}=\frac{x-b}{a-b}$ Найдем точку пересечения хорды с осью Ох:

$x_1 = b - \frac{f(b)}{f(a) - f(b)} * (a - b)$ Теперь возьмем интервал (a, x_1) в качестве исходного и найдем точки пересечения хорды, соединяющей точки $(a, f(a))$ и $(x_1, f(x_1))$, с осью абсцисс (см. рис. 4). Получим $x_2 = x_1 - \frac{f(x_1)}{f(a) - f(x_1)} * (a - x_1)$ Повторяя данную процедуру, получаем рекуррентную формулу:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(a) - f(x_{n-1})} \cdot (a - x_{n-1}) \quad n = 1,2,.. \qquad (3.2)$$

$x_0 = b$.

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 $f(b) f''(b) > 0$ для построения рекуррентной последовательности применяются формулы (3.1), а в случае, когда $f(a) f''(a) > 0$ применяют формулы (3.2).

Метод Ньютона (касательных). Для начала вычислений требуется задание одного начального приближения x_0, последующие приближения вычисляются по формуле

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, f'(x_n) \neq 0$$

Метод имеет квадратичную скорость сходимости для простого корня, но очень чувствителен к выбору начального приближения. При произвольном начальном приближении итерации сходятся, если всюду $|f(x) f''(x)| < (f'(x))^2$, в противном случае сходимость будет только при x_0, достаточно близком к корню. Существует несколько достаточных условий сходимости. Если производные $f'(x)$ и $f''(x)$ сохраняют знак в окрестности корня, рекомендуется выбирать x_0 так, чтобы $f(x) f''(x) > 0$ Если, кроме этого, для отрезка $[a,b]$, содержащего корень, выполняются условия $\left|\frac{f(a)}{f'(a)}\right| < b - a, \left|\frac{f(b)}{f'(b)}\right| < b - a,$ то метод сходится для любых $a \le x_0 \le b$.

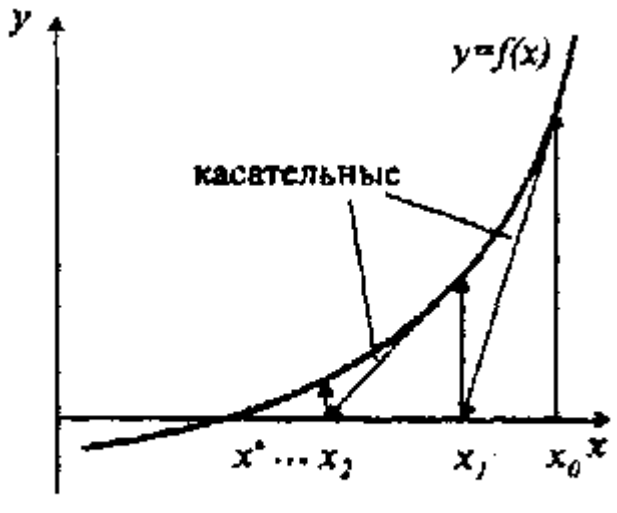


Рис. 5.4

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

1) Используя теорему Штурма определить число корней уравнения:

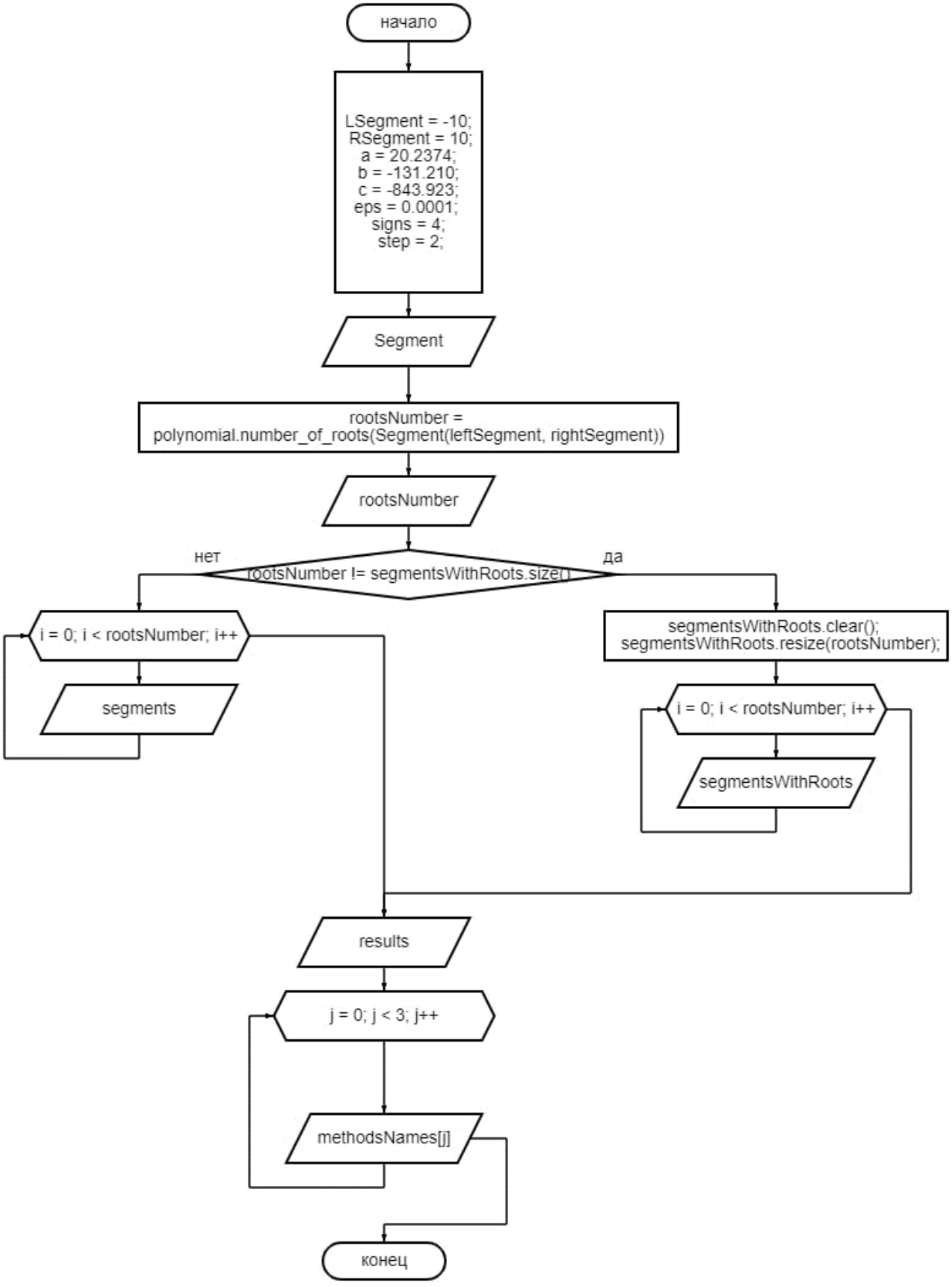
   $\mathbf{x^3+ax^2+bx+c=0}$ на отрезке [-10,10]. Значения коэффициентов уравнения взять из таблицы.

2) Отделить все корни, лежащие на данном отрезке.

3) Вычислить наименьший из корней сначала методом половинного деления, а за затем методом хорд и методом Ньютона. Сравнить число необходимых итераций в обоих методах. Точность до 0.0001.

a	b	c	№ вариан-та
-6,4951	-31,2543	23,1782	10

Алгоритм решения



Программная реализация

```
#include <iostream>
#include <vector>
#include <exception>
#include <iomanip>
#include <map>

using namespace std;

#define MAX_ITERATIONS 1000

struct Result
{
	double rootOfEq;
	int iterations;
	Result(double root = 0, int iterations = 0)
	{
		this->rootOfEq = root;
		this->iterations = iterations;
	}
};

struct Segment
{
	double a;
	double b;
	Segment(double a = 0, double b = 0)
	{
		this->a = a;
		this->b = b;
	}
};

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> val(4);
		val[0] = f0(x);
		val[1] = f1(x);
		val[2] = f2(x);
```

```
		val[3] = f3(x);
		int count = 0;
		for (int i = 0; i < 3; i++)
		{
			if (val[i] * val[i + 1] < 0)
			{
				++count;
			}
		}
		return count;
	}
public:
	Polynomial(double a, double b, double c)
	{
		this->a = a;
		this->b = b;
		this->c = c;
	}
	int number_of_roots(Segment segment)
	{
		return N(segment.a) - N(segment.b);
	}
	vector<Segment> segment_with_roots(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 (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, double eps)
	{
		if (!(f0(segment.a) * f0(segment.b) < 0) ||	// f(a) * f(b)
меньше нуля
			number_of_roots(segment) != 1)			// только 1
корень на промежутке
		{
			return Result(-1, -1);
		}
		int iterations = 1;
		double left = segment.a;
		double right = segment.b;
		double middle = (left + right) / 2;
		while (abs(f0(middle)) > eps && iterations < MAX_ITERATIONS)
		{
			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, double eps)
{
        if (!(f0(segment.a) * f0(segment.b) < 0) ||       // f(a) * f(b)
меньше нуля
                number_of_roots(segment) != 1)                          //
только 1 корень на промежутке
        {
                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 Newthon_method(Segment segment, double eps)
{

        if (number_of_roots(segment) != 1)    // только 1 корень на промежутке
        {
                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);
        }
};

int main()
{
        cout.setf(ios_base::fixed);
        double leftSegment = -10;
        double rightSegment = 10;
        double a = 20.2374;
        double b = -131.210;
        double c = -843.923;
        Polynomial polynomial(a, b, c);

        //eps
        double eps = 0.0001;
        //Количество знаков после запятой
        int signs = 4;
        //Отделять корни табличным методом с шагом
        double step = 2;

        cout << "Segment: [" << setprecision(signs) << leftSegment << ", "
        //Вывод отрезка
                << setprecision(signs) << rightSegment << "]\n";

        int rootsNumber = polynomial.number_of_roots(Segment(leftSegment,
rightSegment));
        cout << "Roots: " << rootsNumber << "\n\n";
                //Кол-во корней

        vector<Segment> segmentsWithRoots =
polynomial.segment_with_roots(Segment(leftSegment, rightSegment), step);

        if (rootsNumber != segmentsWithRoots.size())
        {                                               //Если обнаружен корень чётной
кратности, то нужно отделить корни вручную
                cout <<
"*******************************************************\nThe root of even
multiplicity is found. \n";
                cout << "It is necessary to separate the roots manually. \n\n";
        //нужно отделить корни вручную
                segmentsWithRoots.clear();
                segmentsWithRoots.resize(rootsNumber);
                for (int i = 0; i < rootsNumber; i++)
                {
                        cout << "A segment with a root " << i + 1 << ": \n";
        //Отрезок с корнем
                        cin >> segmentsWithRoots[i].a >> segmentsWithRoots[i].b;
                }
        }
        else
        {
                cout << "Segments: \n";
                                //Отрезки
```

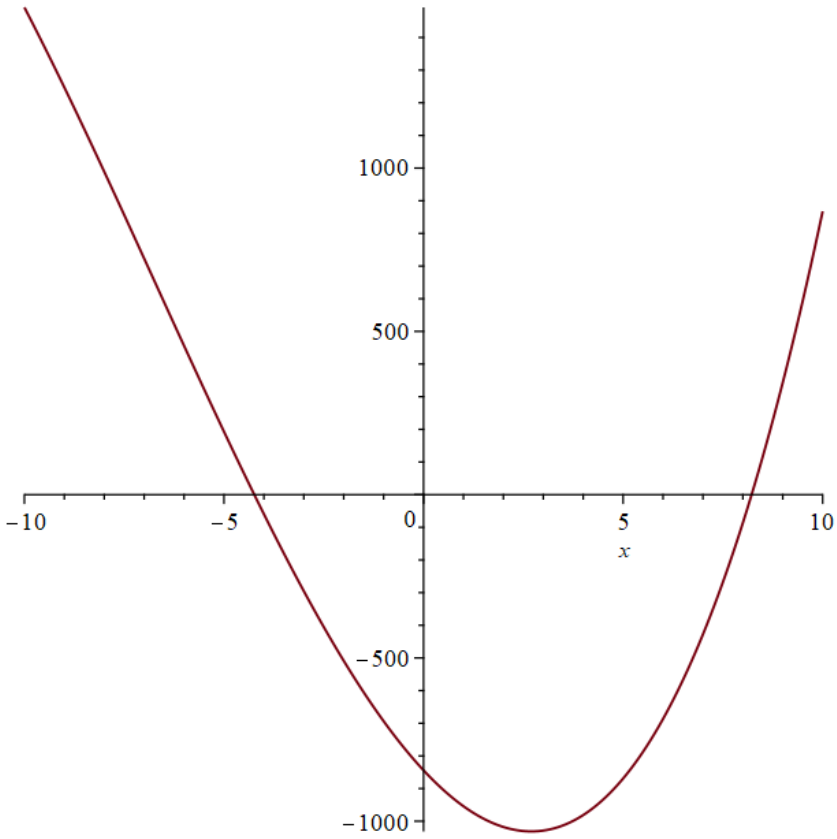
```
		for (const auto segment : segmentsWithRoots)
		{
			cout << "[" << setprecision(signs) << segment.a << ", " << segment.b << "]\n";
		}
	}
	cout << "\n************************************************************\n";
	map<int, string> methodsNames({ make_pair(0, "The method of half division: "), //Метод половинного деления
		make_pair(1, "The chord method: "),		//Метод хорд
		make_pair(2, "Newton 's Method: ") });		//Метод Ньютона

	map<string, vector<Result>> results;
	for (int i = 0; i < rootsNumber; i++)
	{
		results["The method of half division: "].push_back(polynomial.half_division(segmentsWithRoots[i], eps));
		results["The chord method: "].push_back(polynomial.chrod_method(segmentsWithRoots[i], eps));
		results["Newton 's Method: "].push_back(polynomial.Newthon_method(segmentsWithRoots[i], eps));
	}
	for (int i = 0; i < rootsNumber; i++)
	{
		cout << "\nSegment [" << 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 << "The solution by this method is not possible on this segment \n"; // Решение этим методом на этом отрезке невозможно
			else
				cout << setw(signs + 3) << right << setprecision(signs) << results[methodsNames[j]][i].rootOfEq
				<< ". Number of iterations: " << results[methodsNames[j]][i].iterations << "\n";	// Кол-во итераций
		}
	}
}
```

Полученные результаты программы

Выбрать Консоль отладки Microsoft Visual Studio

```
Segment: [-10.0000, 10.0000]
Roots: 2

Segments:
[-6.0000, -4.0000]
[8.0000, 10.0000]

************************************************************

Segment [-6.0000, -4.0000]:
The method of half division: -4.2400. Number of iterations: 21
        The chord method: -4.2400. Number of iterations: 4
        Newton 's Method: -4.2400. Number of iterations: 3

Segment [8.0000, 10.0000]:
The method of half division:  8.2191. Number of iterations: 20
        The chord method:  8.2191. Number of iterations: 6
        Newton 's Method:  8.2191. Number of iterations: 4
```

$$plot\left(x^3 + 20.2374\,x^2 - 131.210\,x - 843.923, x = -10\,..10\right)$$



$$fsolve\left(x^3 + 20.2374\,x^2 - 131.210\,x - 843.923 = 0, x = -10\,..10\right);\ Digits := 5$$

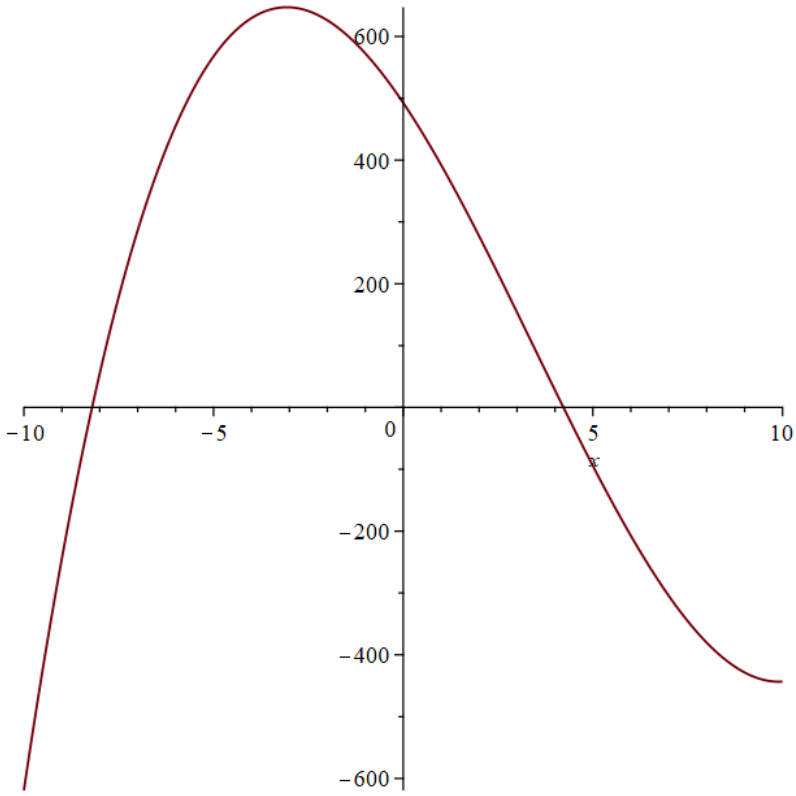
$$-4.2400,\ 8.2192$$

Тестовые примеры

a = -10.2374, b = -91.2105, c = 492.560

```
Segment: [-10.0000, 10.0000]
Roots: 2

Segments:
[-10.0000, -8.0000]
[4.0000, 6.0000]

************************************************************

Segment [-10.0000, -8.0000]:
The method of half division: -8.2027. Number of iterations: 21
        The chord method: -8.2027. Number of iterations: 6
        Newton 's Method: -8.2027. Number of iterations: 3

Segment [4.0000, 6.0000]:
The method of half division:  4.2240. Number of iterations: 19
        The chord method:  4.2240. Number of iterations: 3
        Newton 's Method:  4.2240. Number of iterations: 3
```

$plot\left(x^3 - 10.2374 \cdot x^2 - 91.2105 \cdot x + 492.560, x = -10 \ldots 10\right)$



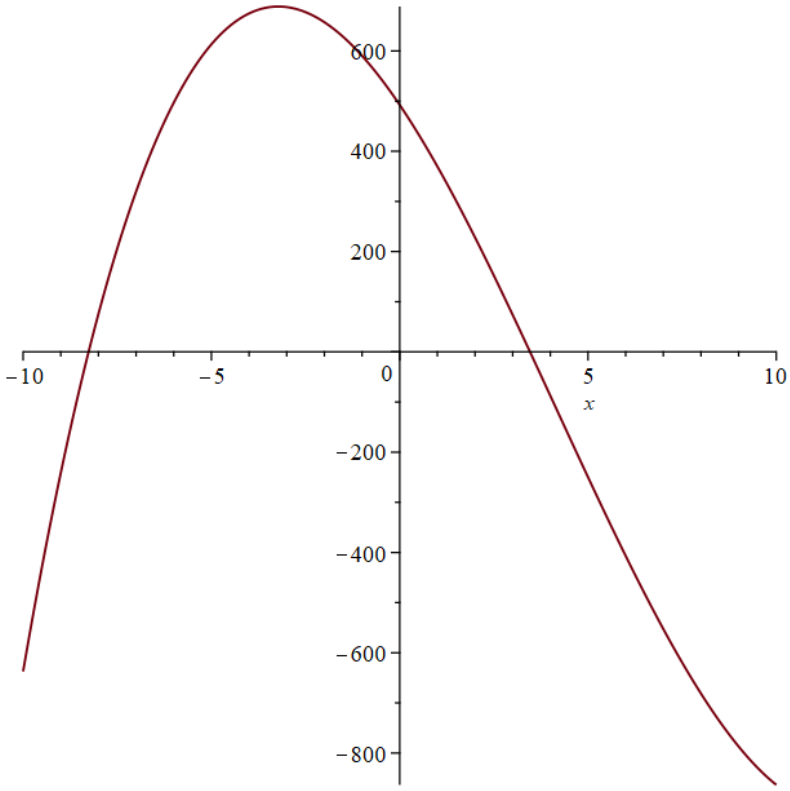
$fsolve\left(x^3 - 10.2374 \cdot x^2 - 91.2105 \cdot x + 492.560 = 0, x = -10 \ldots 10\right); Digits := 5$

$-8.2028, 4.2240$

a = -12.43, b = 111.312, c = 492.560

```
Segment: [-10.0000, 10.0000]
Roots: 2

Segments:
[-10.0000, -8.0000]
[2.0000, 4.0000]

************************************************************

Segment [-10.0000, -8.0000]:
The method of half division: -8.2612. Number of iterations: 22
        The chord method: -8.2612. Number of iterations: 6
        Newton 's Method: -8.2612. Number of iterations: 3

Segment [2.0000, 4.0000]:
The method of half division:  3.4602. Number of iterations: 20
        The chord method:  3.4602. Number of iterations: 3
        Newton 's Method:  3.4602. Number of iterations: 3
```

$plot(x^3 - 12.43 \cdot x^2 - 111.312 \cdot x + 492.560, x = -10 \ldots 10)$



$fsolve(x^3 - 12.43 \cdot x^2 - 111.312 \cdot x + 492.560 = 0, x = -10 \ldots 10)$; $Digits := 5$

$-8.2612, 3.4603$

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. В ходе работы были рассмотрены 5 функций, имеющих несколько корней на заданном промежутке (-10, 10), и в ходе решения результат был проверен с помощью графика, что дает понять, что реализованные методы успешно справляются с решением нелинейных уравнений.