МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

 Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Лабораторная работа

«Решение нелинейного уравнения»

по дисциплине

Вычислительная математика

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Суркова А.С.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сухоруков В.А.

19-ИВТ-3

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2021

Оглавление

[Цель работы 3](#_Toc65673629)

[Вариант задания на лабораторную работу 3](#_Toc65673630)

[Краткие теоретические сведения и описание алгоритма работы программы в виде блок схем 3](#_Toc65673631)

[1. Деление отрезка пополам 3](#_Toc65673632)

[2. Метод хорд 4](#_Toc65673633)

[3. Метод Ньютона 5](#_Toc65673634)

[4. Метод простой итерации 6](#_Toc65673635)

[Расчётные данные (таблицы) 7](#_Toc65673636)

[1. Таблицы, построенные с использованием Excel 7](#_Toc65673637)

[Метод биекции 7](#_Toc65673638)

[Метод хорд 7](#_Toc65673639)

[Метод Ньютона 7](#_Toc65673640)

[Метод простой итерации 8](#_Toc65673641)

[2. Таблицы, построенные в ходе работы разработанной программы 8](#_Toc65673642)

[Метод биекции 8](#_Toc65673643)

[Метод хорд 8](#_Toc65673644)

[Метод Ньютона 9](#_Toc65673645)

[Листинг разработанной программы 10](#_Toc65673646)

[Результаты работы программы 16](#_Toc65673647)

[Решение методом биекции 16](#_Toc65673648)

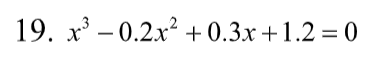
[Решение методом хорд 17](#_Toc65673649)

[Метод Ньютона 17](#_Toc65673650)

[Решение методом простых итераций 18](#_Toc65673651)

Цель работы: Закрепление знаний и умений по нахождению решений нелинейных уравнений различными способами.

# Вариант задания на лабораторную работу

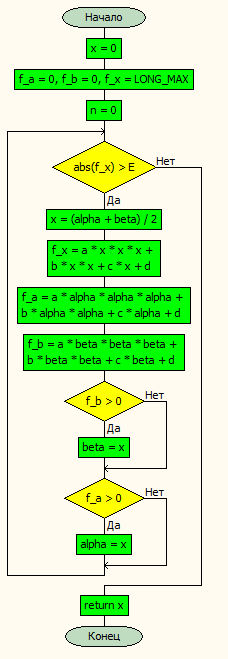


# Краткие теоретические сведения и описание алгоритма работы программы в виде блок схем

1. Деление отрезка пополам (метод биекции) — это численный метод нахождения (одного) решения x (с заданной точностью ε) нелинейного уравнения вида ***f(x) = 0***.

Суть метода деления отрезка пополам состоит в разбиении отрезка [a, b] (при условии ***f(a)f(b) <0***) на два отрезка, определении знака функции ***f(x)*** в середине отрезка (a + b)/2 и выборе отрезка, на котором функция меняет знак и содержит решение.

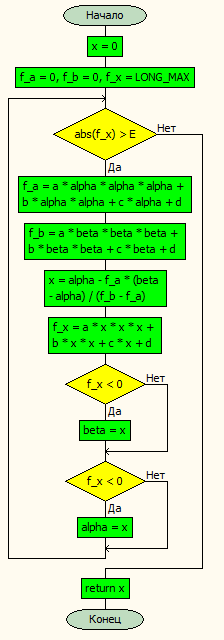
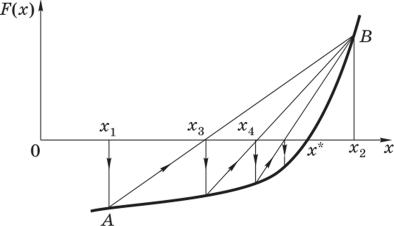
Деление отрезка продолжается до достижения необходимой точности решения ε.



1. Метод хорд (Метод секущих) один из методов решения нелинейных уравнений и основан на последовательном сужении интервала, содержащего единственный корень уравнения ***f(x) = 0***. Итерационный процесс выполняется до того момента, пока не будет достигнута заданная точность ε.

В отличие от метода половинного деления, метод хорд предлагает, что деление рассматриваемого интервала будет выполняться не в его середине, а в точке пересечения хорды с осью абсцисс (ось - Х). Следует отметить, что под хордой понимается отрезок, который проведен через точки рассматриваемой функции по концам рассматриваемого интервала. Рассматриваемый метод обеспечивает более быстрое нахождение корня, чем метод половинного деления, при условии задания одинакового рассматриваемого интервала.

Геометрически метод хорд эквивалентен замене кривой ***y=f(x)*** хордой, проходящей через точки ***(a, f(a))*** и ***(b, f(b))***



1. Метод Ньютона (метод касательных)

Геометрически итерационный процесс метода Ньютона означает замену на *k*-той итерации графика функции *y=f(x)* на касательную к этой функции в точке ***(x(k) , f(x(k)))*** (в связи с этим метод также иногда называется методом касательных). Уравнение касательной имеет вид

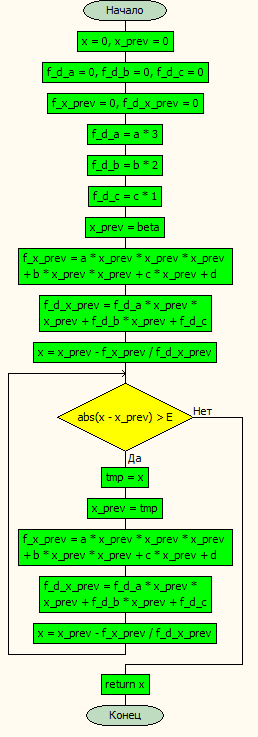
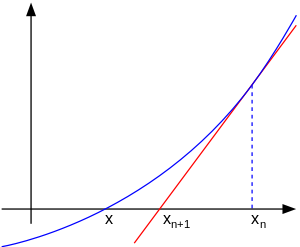
***y=f '(x(k))(x-x(k))+f(x(k))***

Найдем точку пересечения с осью OX этой касательной (вместо функции ***y=f(x)***), что соответствует нахождению решения линейного уравнения:

***f '(x(k))(x-x(k))+ f(x(k))=0***

вместо нелинейного ***f(x)=0***.

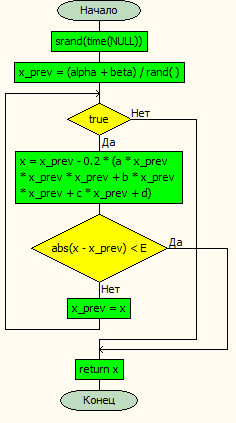
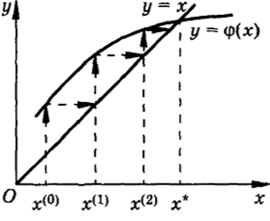
Выражая *x*, получаем: ***x = x(k) - f(x(k))/f '(x(k))≡ x(k+1)***



1. Метод простой итерации- один из простейших численных методов решения уравнений. Метод основан на принципе сжимающего отображения.

Идея метода простой итерации состоит в том, чтобы уравнение ***f(x) = 0*** привести к эквивалентному уравнению ***x = φ(x)*** так, чтобы отображение ***φ(x)*** было сжимающим. Если это удаётся, то последовательность итераций ***x(i+1) =* φ(*x(i)*)** сходится.

В предложенном варианте ***f(x)=x3-0.2x2+0.3x+1.2***.В качестве сжимающей функции была выбрана функция ***g(x)=x-0.2(x3-0.2x2+0.3x+1.2)***, которая может не подойди к другим вариантам.



# Расчётные данные (таблицы)

## Таблицы, построенные с использованием Excel

### Метод биекции

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| n | an | bn | xn | f(an) | f(bn) | f(xn) | |an-bn| |
| 0 | -1 | 0 |  | -0,3 | 1,2 |  | 1 |
| 1 | -1 | -0,5 | -0,5 | -0,3 | 1,2 | 0,875 | 0,5 |
| 2 | -1 | -0,75 | -0,75 | -0,3 | 0,875 | 0,440625 | 0,25 |
| 3 | -1 | -0,875 | -0,875 | -0,3 | 0,440625 | 0,114453 | 0,125 |
| 4 | -0,9375 | -0,875 | -0,9375 | -0,3 | 0,114453 | -0,08101 | 0,0625 |
| 5 | -0,9375 | -0,90625 | -0,90625 | -0,08101 | 0,114453 | 0,019574 | 0,03125 |
| 6 | -0,92188 | -0,90625 | -0,92188 | -0,08101 | 0,019574 | -0,02999 | 0,015625 |
| 7 | -0,91406 | -0,90625 | -0,91406 | -0,02999 | 0,019574 | -0,00503 | 0,007813 |
| 8 | -0,91406 | -0,91016 | -0,91016 | -0,00503 | 0,019574 | 0,007317 | 0,003906 |
| 9 | -0,91406 | -0,91211 | -0,91211 | -0,00503 | 0,007317 | 0,001155 | 0,001953 |
| 10 | -0,91309 | -0,91211 | -0,91309 | -0,00503 | 0,001155 | -0,00193 | 0,000977 |
|  |  |  | -0,9126 |  |  | -0,00039 |  |

### Метод хорд

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | xn | an | bn | |f(xn)| | f(an) | f(bn) |
| 0 |  | -1 | 0 |  | -0,3 | 1,2 |
| 1 | -0,8 | -1 | -0,8 | 0,32 | -0,3 | 0,32 |
| 2 | -0,90323 | -1 | -0,90323 | 0,029002 | -0,3 | 0,029002 |
| 3 | -0,91176 | -1 | -0,91176 | 0,00227 | -0,3 | 0,00227 |
|  | -0,91242 |  |  | 0,000175 |  |  |

### Метод Ньютона

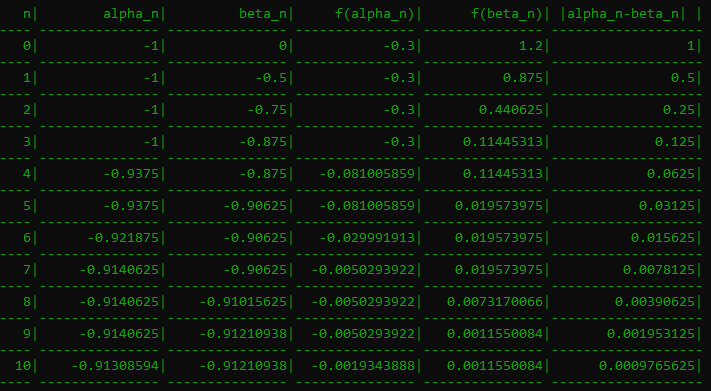
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | xn | f(xn) | f'(xn) | |xn-x(n-1)| |
| 0 | 0 |  |  |  |
| 1 | -4 | 1,2 | 0,3 | 4 |
| 2 | -2,65331 | -67,2 | 49,9 | 1,3466934 |
| 3 | -1,77777 | -19,6834 | 22,48143 | 0,8755392 |
| 4 | -1,24558 | -5,58398 | 10,49248 | 0,5321888 |
| 5 | -0,98581 | -1,41644 | 5,45263 | 0,2597719 |
| 6 | -0,91707 | -0,24813 | 3,609768 | 0,0687376 |
| 7 | -0,91249 | -0,01459 | 3,189875 | 0,004575 |
| 8 | -0,91247 | -6,2E-05 | 3,162935 | 1,95E-05 |

### Метод простой итерации

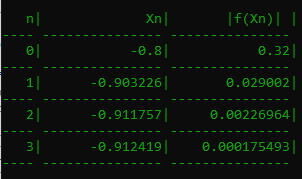
|  |  |  |
| --- | --- | --- |
| n | xn | |xn-x(n-1)| |
| 0 | -0,03 |  |
| 1 | -0,26816 | 0,2381586 |
| 2 | -0,48534 | 0,21717752 |
| 3 | -0,66393 | 0,17859349 |
| 4 | -0,78793 | 0,12399975 |
| 5 | -0,85799 | 0,07005647 |
| 6 | -0,89074 | 0,03275578 |
| 7 | -0,90421 | 0,01347213 |
| 8 | -0,9094 | 0,0051856 |
| 9 | -0,91134 | 0,0019398 |
| 10 | -0,91206 | 0,00071755 |

## Таблицы, построенные в ходе работы разработанной программы

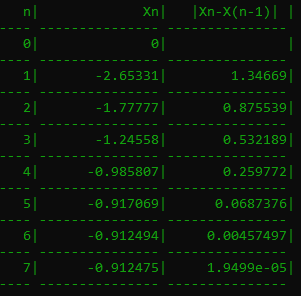
### Метод биекции



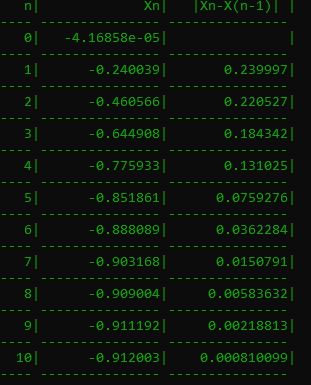
### Метод хорд



### Метод Ньютона



Метод простой итерации



# Листинг разработанной программы

#include<iostream>

#include<iomanip>

using namespace std;

double const E = 0.001;

/\*Функция поиска отрезка для нахождения корня

Параметры функции:

1)\*alpha,\*beta - указатели на переменные, содержащие границы отрезка, на котором ищется корень

2)a, b, c, d - параметры полинома a\*x^3+b\*x^2+c\*x+d

Принцип работы:

1)Присваиваем границам значения -100 и -99 (минимально возможные в рамках данной задачи)

2)Находим значения функции в точках alpha и beta

3)Если при умножении значений функции знак произведения отрицательный, то найдены нужные значения и функция завершается, иначе увеличиваем значение переменных alpha и beta на 1

4)Пункты 2)-3) выполняются пока не найдены нужные значения или граница не дошла до предельно допустимой в рамках данной задачи (99 и 100)

\*/

void find\_section(double \*alpha, double \*beta, double a, double b, double c, double d) {

double f\_a,f\_b;

\*alpha = -100;

\*beta = -99;

for (\*alpha; \*alpha < 100; (\*alpha)++) {

f\_a = a \* (\*alpha) \* (\*alpha) \* (\*alpha) + b \*(\*alpha)\*

(\*alpha) + c \* (\*alpha) + d;

f\_b = a \* (\*beta) \* (\*beta) \* (\*beta) + b \* (\*beta) \*

(\*beta) + c \* (\*beta) + d;

if (f\_a \* f\_b < 0) { break; }

(\*beta)++;

}

}

/\*Метод биекции

Параметры функции:

1)alpha,beta - границы отрезка, на котором ищется корень

2)a, b, c, d - параметры полинома a\*x^3+b\*x^2+c\*x+d

Принцип работы:

1)Находится середина отрезка [alpha,beta]

2)Вычисляется значение полинома в середине отрезка и на его концах

3)Если знак значения полинома в середине отрезка совпадает со знаком значения на одном из концов, то значение этого конца заменяется на значение середины отрезка

4)Пункты 1)-3) выполняются пока модуль значения полинома в середине отрезка больше погрешности E

\*/

double bisection(double alpha,double beta,double a,double b,double c,double d) {

double x = 0;

double f\_a = 0,f\_b=0,f\_x=LONG\_MAX;

int n = 0;

cout<<"\n\u001B[32m" << setw(5) << "n|" << setw(16)

<< "alpha\_n|" << setw(16) << "beta\_n|" << setw(16)

<< "f(alpha\_n)|" << setw(16) << "f(beta\_n)|" << setw(20)

<< "|alpha\_n-beta\_n| |\n"

<< "---- --------------- --------------- --------------- --------------- -------------------\n";

while (abs(f\_x) > E) {

x = (alpha + beta) / 2;

f\_x = a \* x \* x \* x + b \* x \* x + c \* x + d;

f\_a = a \* alpha \* alpha \* alpha + b \* alpha \* alpha

+ c \* alpha + d;

f\_b = a \* beta \* beta \* beta + b \* beta \* beta

+ c \* beta + d;

cout << setw(4) << n<<"|" << setw(15)

<< setprecision(8) << alpha << "|" << setw(15)

<< setprecision(8) << beta << "|" << setw(15)

<< setprecision(8) << f\_a << "|" << setw(15)

<< setprecision(8) << f\_b << "|" << setw(18)

<< setprecision(8) << abs(alpha - beta) << "|"

<< "\n"

<< "---- --------------- --------------- --------------- --------------- -------------------\n";

n++;

if (f\_x \* f\_b > 0) { beta = x; }

if (f\_x \* f\_a > 0) { alpha = x; }

}

return x;

}

/\*Метод хорд

Параметры функции:

1)alpha, beta - границы отрезка, на котором ищется корень

2)a, b, c, d - параметры полинома a \* x ^ 3 + b \* x ^ 2 + c \* x + d

Принцип работы:

1)Вычисляется значение полинома на концах отрезка

2)Вычисляется точка пересечения хорды с осью абсцисс

3)Вычисляется значение полинома в найденной точке

4)Если знак значения полинома в найденной точке совпадает со знаком значения на одном из концов, то значение другого конца заменяется на значение найденной точки

5)Пункты 1) - 4) выполняются пока модуль значения полинома в текущей больше погрешности E

\*/

double chords(double alpha, double beta, double a, double b, double c, double d) {

double x = 0;

double f\_a = 0, f\_b = 0, f\_x = LONG\_MAX;

int n = 0;

cout << "\n\u001B[32m" << setw(5) << "n|" << setw(16)

<< "Xn|" << setw(17) << "|f(Xn)| |\n"

<< "---- --------------- ---------------\n";

while (abs(f\_x) > E) {

f\_a = a \* alpha \* alpha \* alpha + b \* alpha \* alpha

+ c \* alpha + d;

f\_b = a \* beta \* beta \* beta + b \* beta \* beta

+ c \* beta + d;

x = alpha - f\_a \* (beta - alpha) / (f\_b - f\_a);

f\_x = a \* x \* x \* x + b \* x \* x + c \* x + d;

cout << setw(4) << n << "|" << setw(15) << x

<< "|" << setw(15) << f\_x<<"|\n"

<< "---- --------------- ---------------\n";

n++;

if (f\_a \* f\_x < 0) { beta = x; }

if (f\_b \* f\_x < 0) { alpha = x; }

}

return x;

}

/\*Метод Ньютона

Параметры функции :

1)alpha, beta - границы отрезка, на котором ищется корень

2)a, b, c, d - параметры полинома a \* x ^ 3 + b \* x ^ 2 + c \* x + d

Принцип работы :

1)Вычисляются значения коэффициентов производной

2)Вычисляются значения полинома и его производной в точке beta

3)Вычисляется значение следующей точки

4)Вычисляются значения полинома и его производной в текущей точке

5)Пункты 3) - 4) выполняются пока модуль разности предыдущей и текущей точек больше погрешности E

\*/

double Newtone(double alpha, double beta, double a, double b, double c, double d) {

double x=0,x\_prev=0;

double f\_d\_a = 0, f\_d\_b = 0, f\_d\_c = 0;

double f\_x\_prev = 0,f\_d\_x\_prev =0;

int n = 0;

f\_d\_a = a \* 3;

f\_d\_b = b \* 2;

f\_d\_c = c \* 1;

x\_prev = beta;

cout << "\n\u001B[32m" << setw(5) << "n|" << setw(16)

<< "Xn|" << setw(17) << "|Xn-X(n-1)| |\n"

<< "---- --------------- ---------------\n"

<< setw(4) << n << "|" << setw(15) << x\_prev << "|"

<< setw(17)<< "|\n"

<< "---- --------------- ---------------\n";

f\_x\_prev = a \* x\_prev \* x\_prev \* x\_prev + b \* x\_prev

\* x\_prev + c \* x\_prev + d;

f\_d\_x\_prev = f\_d\_a \* x\_prev \* x\_prev + f\_d\_b \* x\_prev

+ f\_d\_c;

x = x\_prev - f\_x\_prev / f\_d\_x\_prev;

while (abs(x - x\_prev) > E) {

double tmp = x;

x\_prev = tmp;

f\_x\_prev = a \* x\_prev \* x\_prev \* x\_prev + b \* x\_prev

\* x\_prev + c \* x\_prev + d;

f\_d\_x\_prev = f\_d\_a \* x\_prev \* x\_prev + f\_d\_b \* x\_prev

+ f\_d\_c;

x = x\_prev - f\_x\_prev / f\_d\_x\_prev;

n++;

cout << setw(4) << n << "|" << setw(15) << x << "|"

<< setw(15)<<abs(x-x\_prev) << "|\n"

<< "---- --------------- ---------------\n";

}

return x;

}

/\*Метод простой итерации

Параметры функции :

1)alpha, beta - границы отрезка, на котором ищется корень

2)a, b, c, d - параметры полинома a \* x ^ 3 + b \* x ^ 2 + c \* x + d

Принцип работы :

1)Случайно генерируется первая точка, принадлежащую отрезку [alpha,beta]

2)Вычисляется значение следующей точки, по формуле

g(x)=x-0.2\*(a \* x ^ 3 + b \* x ^ 2 + c \* x + d)

3)Пункт 2) выполняется пока модуль разности предыдущей и текущей точек больше погрешности E

\*/

double iterations(double alpha, double beta, double a, double b, double c, double d) {

double x,x\_prev;

int n = 0;

srand(time(NULL));

x\_prev= (alpha + beta)/ rand();

cout << "\n\u001B[32m" << setw(5) << "n|" << setw(16)

<< "Xn|" << setw(17) << "|Xn-X(n-1)| |\n";

cout << "---- --------------- ---------------\n"

<< setw(4) << n << "|" << setw(15) << x\_prev << "|"

<< setw(17) << "|\n"

<< "---- --------------- ---------------\n";

while (true) {

x = x\_prev - 0.2 \* (a \* x\_prev \* x\_prev \* x\_prev + b

\* x\_prev \* x\_prev + c \* x\_prev + d);

n++;

cout << setw(4) << n << "|" << setw(15) << x << "|"

<< setw(15) << abs(x - x\_prev) << "|\n"

<< "---- --------------- ---------------\n";

if (abs(x - x\_prev) < E) { break; }

x\_prev = x;

}

return x;

}

Int main () {

Set locale (LC\_ALL, "RUS");

Double a = 0, b = 0, c = 0, d = 0;

Double alpha = -2, beta = 0;

Double x = 0;

char metod;

//Запрос исходных данных

cout <<«\u001B [33mДанная программа предназначена для нахождения корней полинома третьей степени.\n Введите параметры a, b, c, d полинома вида a\*x^3+b\*x^2+c\*x+d=0\n"

Cout<<"\t\u001B [36mВведите а ";

с>> a;

cout <<«\tВведите b ";

с>> b;

Cout << "\tВведите c ";

Cin >> c;

Cout << "\tВведите d ";

Cin >> d;

//Нахождение отрезка для поиска корней

find\_section(&alpha, &beta, a, b, c, d);

cout << "\n\u001B[33mОтрезок [alpha,beta], на котором”

<<“ищется корень найден.\n alpha="

<< alpha << " beta=" << beta << "\n";

//Выбор метода нахождения корней

cout << "\nВыберите метод решения уравнения:\n"

<< "\u001B[36m\t{1} - Метод бисекции\n"

<< "\t{2} - Метод хорд\n"

<< "\t{3} - Метод Ньютона\n"

<< "\t{4} - Метод простой итерации\n\n";

cin >> metod;

switch (metod){

case '1':

x=bisection(alpha, beta, a, b, c, d);

Cout << "\nВремя, затраченное на нахождение корня = "

<< (double) (clock () - start) / CLOCKS\_PER\_SEC

<<" секунд\n";

break;

Case '2':

x = chords (alpha, beta, a, b, c, d);

Cout << "\nВремя, затраченное на нахождение корня = "

<< (double) (clock () - start) / CLOCKS\_PER\_SEC

<<" секунд\n";

Break;

Case '3':

x = Newtone (alpha, beta, a, b, c, d);

cout << "\nВремя, затраченное на нахождение корня = "

<< (double) (clock () - start) / CLOCKS\_PER\_SEC

<<" секунд\n";

break;

case '4':

x = iterations (alpha, beta, a, b, c, d);

cout << "\nВремя, затраченное на нахождение корня = "

<< (double) (clock () - start) / CLOCKS\_PER\_SEC

<<" секунд\n";

break;

}

//Вывод результата

cout << "\n\u001B[33mКорень полинома " << x<<"\n";

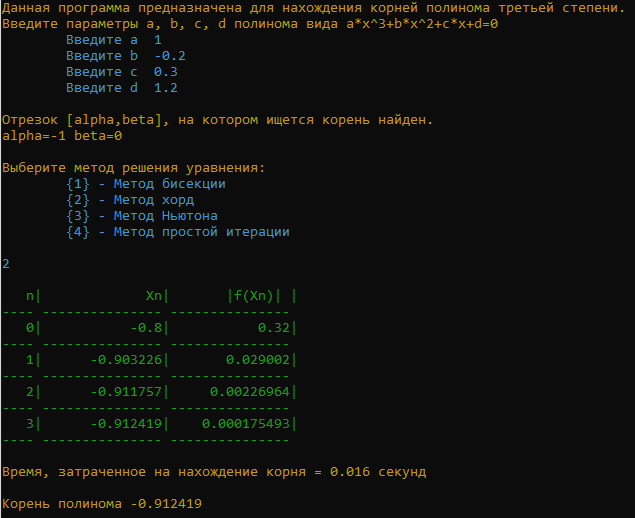
return 0;

}

# Результаты работы программы

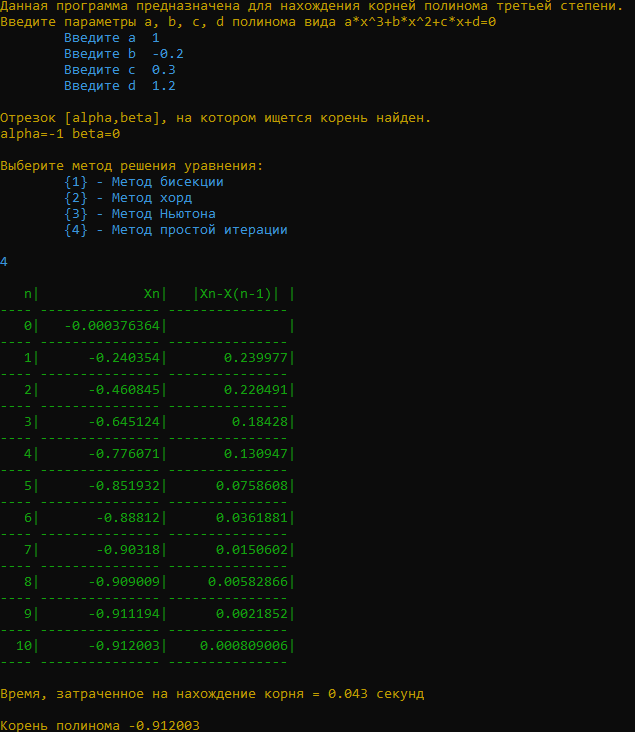
### Решение методом биекции

### Решение методом хорд



### Метод Ньютона

### Решение методом простых итераций



# Вывод

В ходе лабораторной работы были изучены и реализованы 4 метода решения нелинейного уравнения. Самым быстро действенным оказался метод хорд, самым долгим – метод биекции.