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

Решение систем нелинейных уравнений

Выполнил: студент группы 253504

Фроленко Кирилл Юрьевич

Проверил: Анисимов Владимир Яковлевич

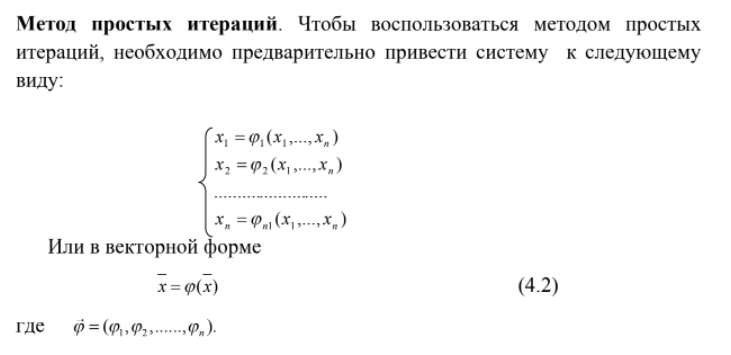
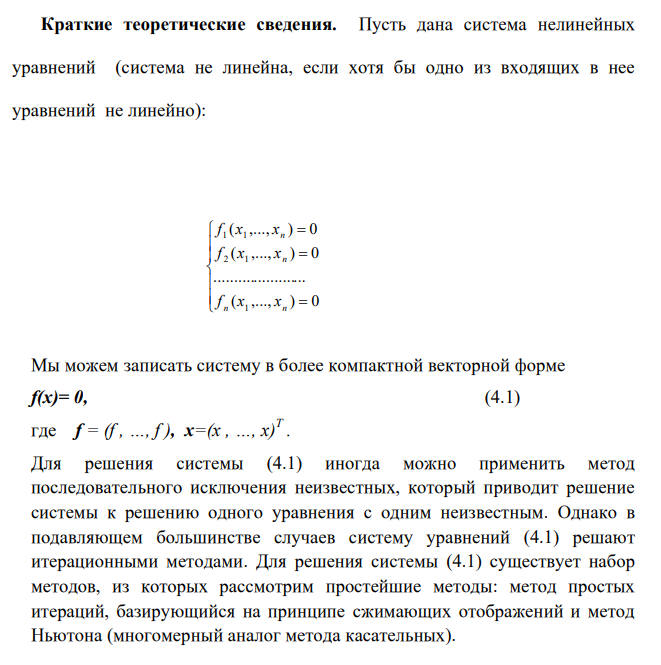
Минск 2023

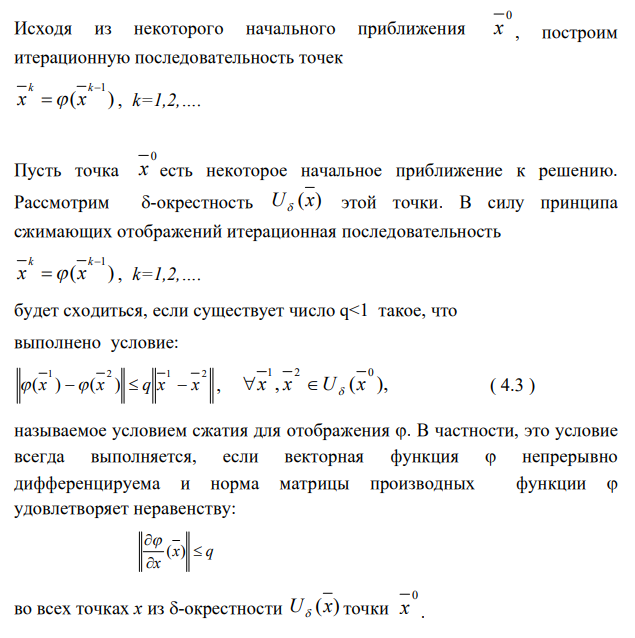
**Вариант 28**

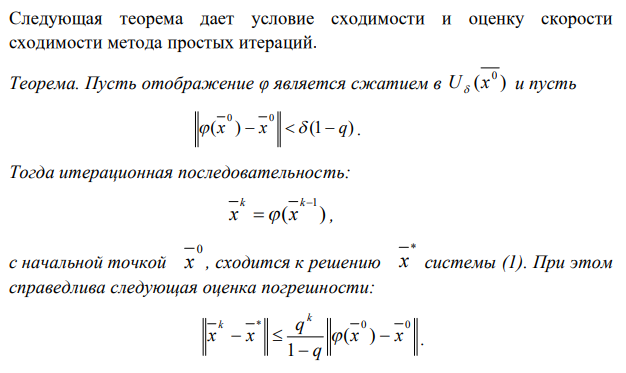
**Цели выполнения задания**

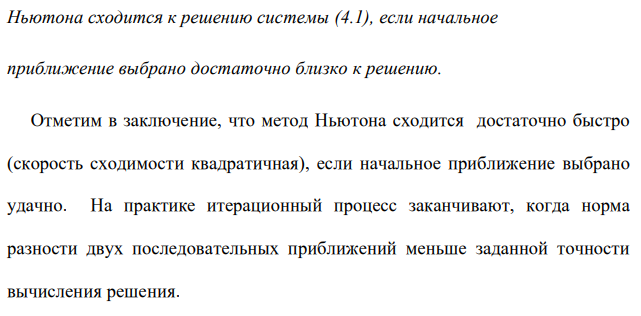
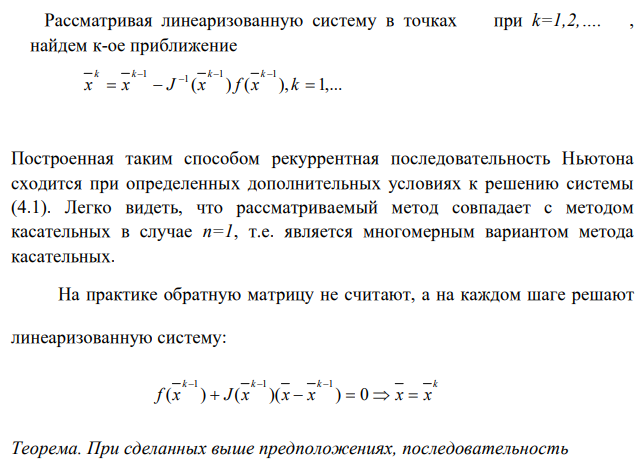
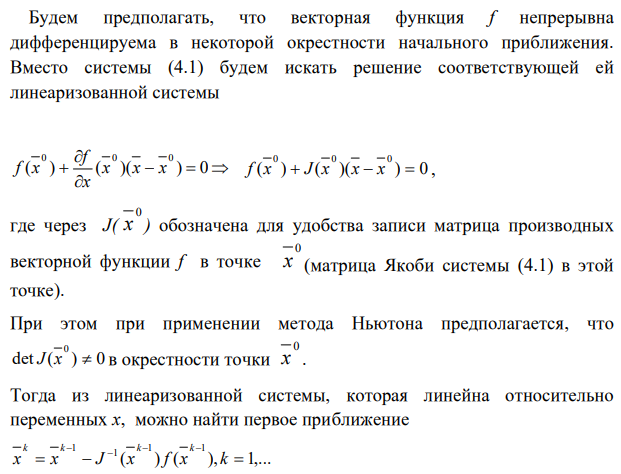
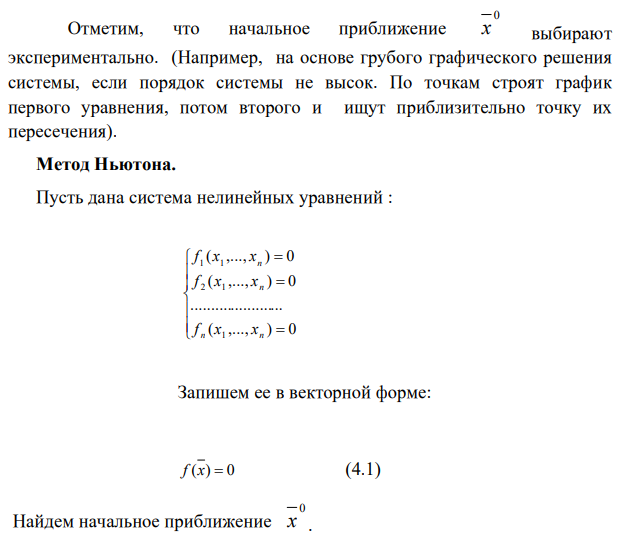
* Изучить численное решение систем нелинейных уравнений методами простых итераций и Ньютона.
* Провести отделение корней.
* Построить и запрограммировать алгоритмы методов
* Численно решить тестовое задание.
* Сравнить трудоемкость методов.

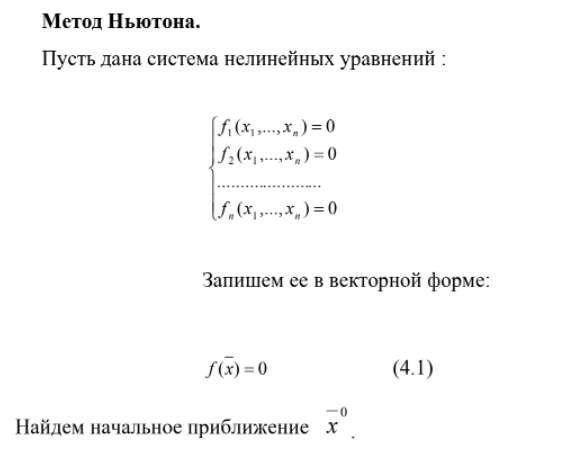
**Краткие теоретические сведения.**

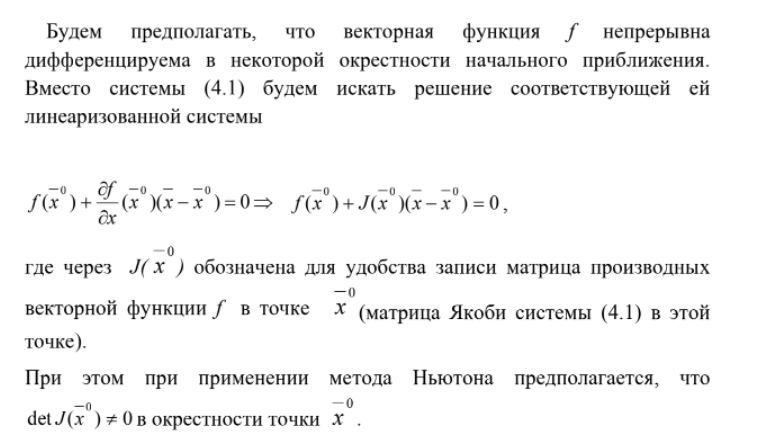


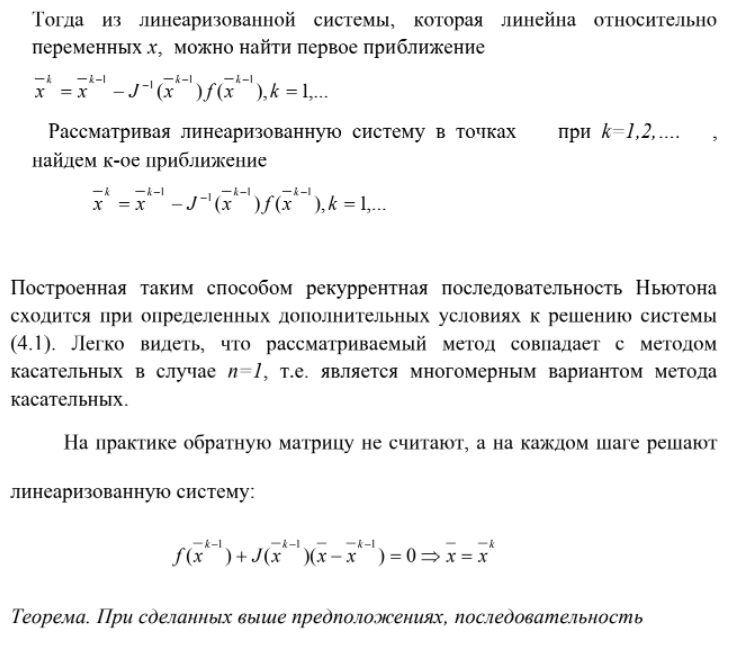


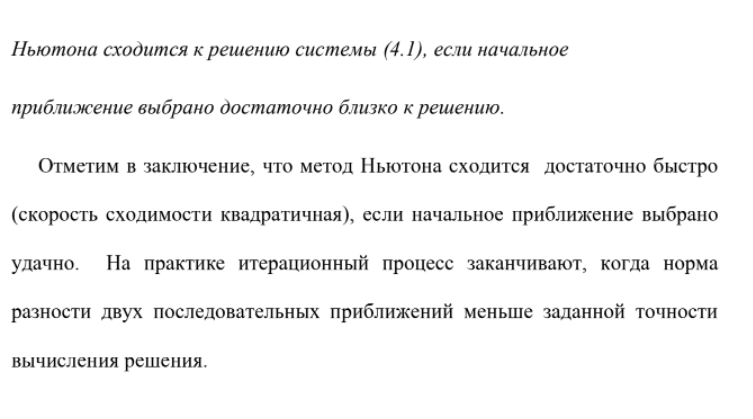




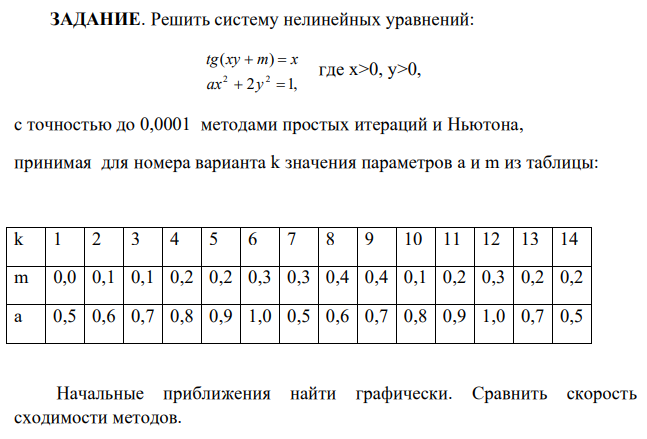




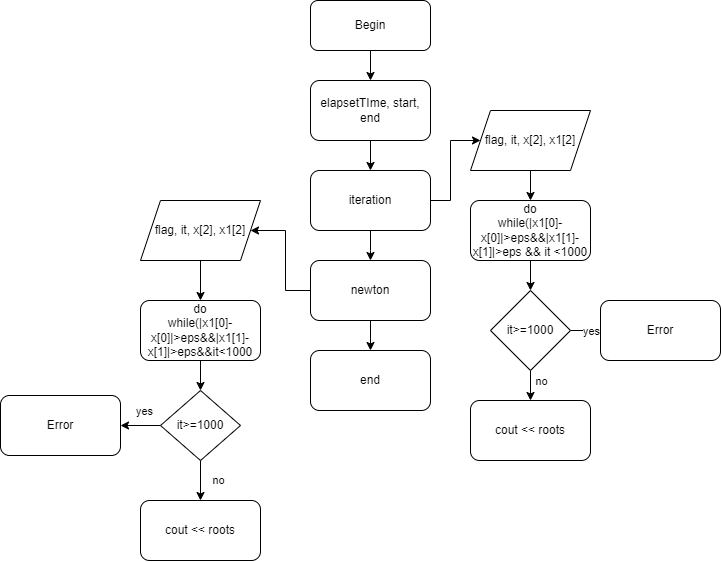




**Задание.**

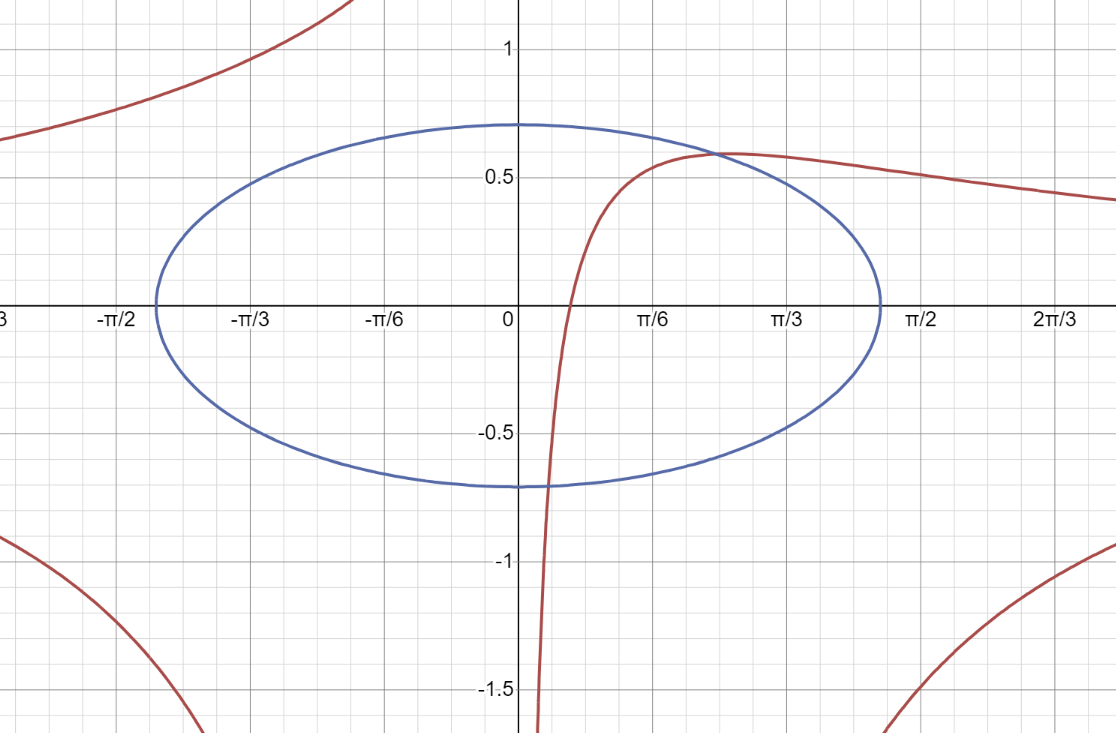
****

**Блок схема алгоритма:**



**Отделение корней:**

Определим начальное приближение, построив графики функций системы и найдя их точку пересечения



Из графика видно, что решения близки к (0.1, -0.7) и (0.8, 0.6) и их можно взять за начальные приближения.

Методы требуют преобразования над системой, поэтому приведем ее к виду

x = tg(x\*y + 2/10)

y = sqrt((1 - x2/2)/2)

для метода простых итераций и к виду

tg(xy+2/10) – x=0

x2/2+2\*y2-1=0

для метода Ньютона.

**Код программы:**

#include <bits/stdc++.h>

#define ll long long

#define ld long double

using namespace std;

const ld m = 0.2;

const ld a = 0.5;

ld pribl1X = 0.8;

ld pribl1Y = 0.6;

ld pribl2X = 0.1;

ld pribl2Y = -0.7;

void iteration() {

ll it = 0;

ld x[2] = { pribl1X, pribl1Y };

ld x1[2] = { pribl1X, pribl1Y };

cout << "Начальное приближение: ";

cout << x1[0] << " " << x1[1] << "\n";

do {

it++;

x[0] = x1[0];

x[1] = x1[1];

x1[0] = tan(x[0] \* x[1] + m);

x1[1] = sqrt(0.5 - a \* x[0] \* x[0] / 2);

} while (abs(x1[0] - x[0]) > 0.00001 && abs(x1[1] - x[1]) > 0.00001);

cout << "Полученное решение: ";

cout << x1[0] << " " << x1[1] << "\n";

cout << "Количество итераций: " << it << "\n";

it = 0;

x[0] = pribl2X; x[1] = pribl2Y; x1[0] = pribl2X; x1[1] = pribl2Y;

cout << "Начальное приближение: ";

cout << x1[0] << " " << x1[1] << "\n";

do {

it++;

x[0] = x1[0];

x[1] = x1[1];

x1[0] = tan(x[0] \* x[1] + m);

x1[1] = sqrt(0.5 - a \* x[0] \* x[0] / 2);

} while (abs(x1[0] - x[0]) > 0.00001 && abs(x1[1] - x[1]) > 0.00001);

cout << "Полученное решение: ";

cout << x1[0] << " " << x1[1] << "\n";

cout << "Количество итераций: " << it << "\n";

return;

}

void newton() {

ll it = 0;

ld x[2] = { pribl1X, pribl1Y };

ld x1[2] = { pribl1X, pribl1Y };

cout << "Начальное приближение: ";

cout << x1[0] << " " << x1[1] << "\n";

ld J[2][2] = { {0,0},{0,0} };

ld f1, f2;

do {

it++;

x[0] = x1[0];

x[1] = x1[1];

J[0][0] = x[1] / (cos(x[0] \* x[1] + m) \* cos(x[0] \* x[1] + m)) - 1;

J[0][1] = x[0] / (cos(x[0] \* x[1] + m) \* cos(x[0] \* x[1] + m));

J[1][0] = 2 \* a \* x[0];

J[1][1] = 4 \* x[1];

f1 = tan(x[0] \* x[1] + m) - x[0]; //J[0][0]\*dx+J[0][1]\*dy=-f1

f2 = a \* x[0] \* x[0] + 2 \* x[1] \* x[1] - 1; //J[1][0]\*dx+J[1][1]\*dy=-f2

ld deltay = (f2 \* J[0][0] / J[1][0] - f1) / (J[0][1] - J[1][1] \* J[0][0] / J[1][0]);

ld deltax = (-f2 - J[1][1] \* deltay) / J[1][0];

x1[0] = x[0] + deltax;

x1[1] = x[1] + deltay;

} while (abs(x1[0] - x[0]) > 0.00001 && abs(x1[1] - x[1]) > 0.00001);

cout << "Полученное решение: ";

cout << x1[0] << " " << x1[1] << "\n";

cout << "Количество итераций: " << it << "\n";

it = 0;

x[0] = pribl2X; x[1] = pribl2Y; x1[0] = pribl2X; x1[1] = pribl2Y;

cout << "Начальное приближение: ";

cout << x1[0] << " " << x1[1] << "\n";

do {

it++;

x[0] = x1[0];

x[1] = x1[1];

J[0][0] = x[1] / (cos(x[0] \* x[1] + m) \* cos(x[0] \* x[1] + m)) - 1;

J[0][1] = x[0] / (cos(x[0] \* x[1] + m) \* cos(x[0] \* x[1] + m));

J[1][0] = 2 \* a \* x[0];

J[1][1] = 4 \* x[1];

f1 = tan(x[0] \* x[1] + m) - x[0]; //J[0][0]\*dx+J[0][1]\*dy=-f1

f2 = a \* x[0] \* x[0] + 2 \* x[1] \* x[1] - 1; //J[1][0]\*dx+J[1][1]\*dy=-f2

ld deltay = (f2 \* J[0][0] / J[1][0] - f1) / (J[0][1] - J[1][1] \* J[0][0] / J[1][0]);

ld deltax = (-f2 - J[1][1] \* deltay) / J[1][0];

x1[0] = x[0] + deltax;

x1[1] = x[1] + deltay;

} while (abs(x1[0] - x[0]) > 0.00001 && abs(x1[1] - x[1]) > 0.00001);

cout << "Полученное решение: ";

cout << x1[0] << " " << x1[1] << "\n";

cout << "Количество итераций: " << it << "\n";

}

int main() {

system("chcp 65001");

cout << "Метод итераций : \n";

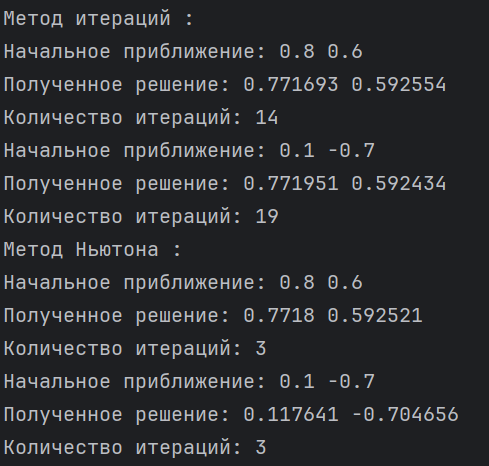
iteration();

cout << "Метод Ньютона : \n";

newton();

}

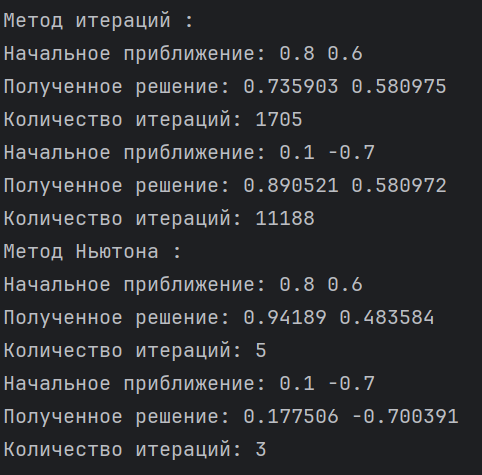
**Результаты**



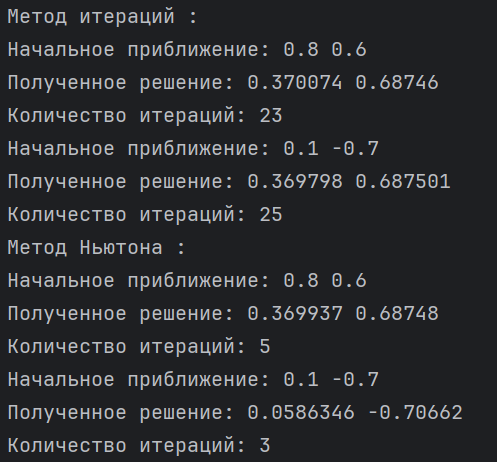
**Оценка**

Возьмем коэффициенты m1 = 0.3, a1 = 0.6 и m2 = 0.1, a2 = 0.4 и оставим те же начальные приближения. Тогда:

1. При m1 = 0.3, a1 = 0.6 получаем



2) При m2 = 0.4, a2 = 1.1 получаем

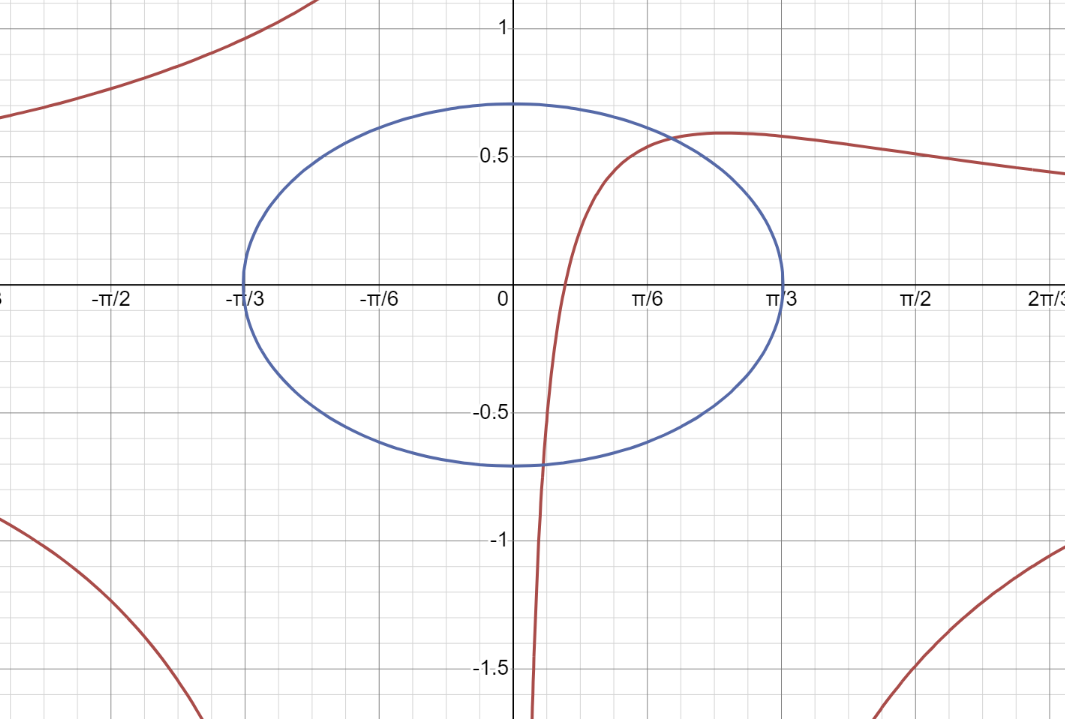


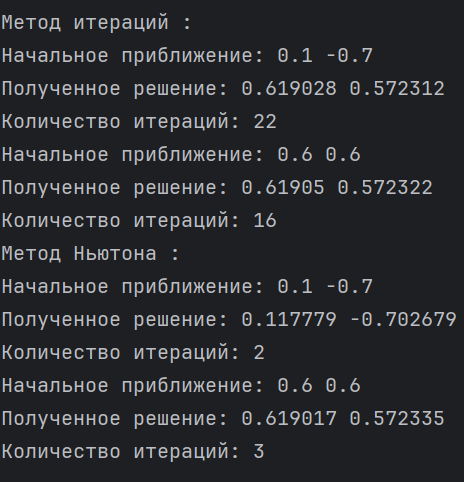
Можно сделать вывод, что точности 10-4 достигнуть не удалось, однако при изменение исходные значений на 0.0001 точность достижима.

**Тестовые примеры**

Тестовый пример 1:

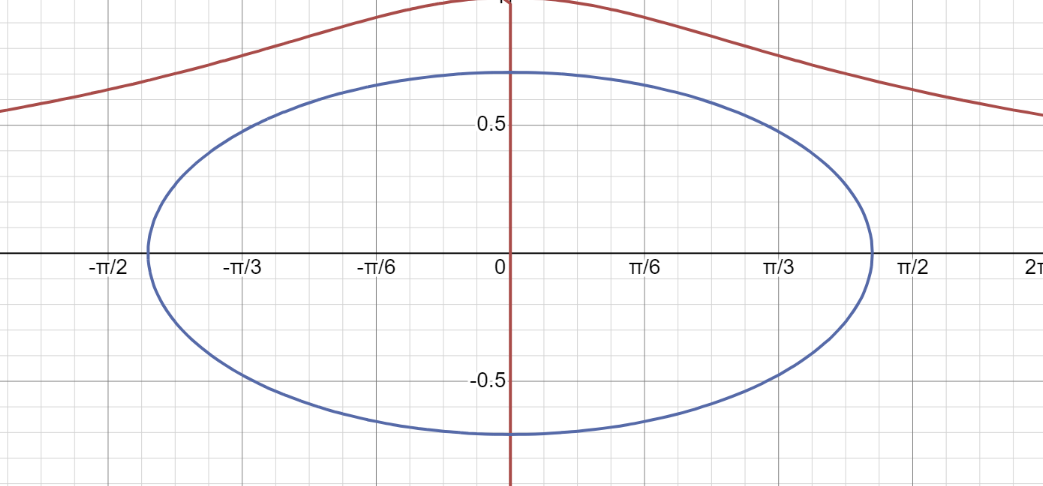
m = 0.2, a = 0.9. Начальные приближения: {0.1;-0.7},{0.6;0.6}.

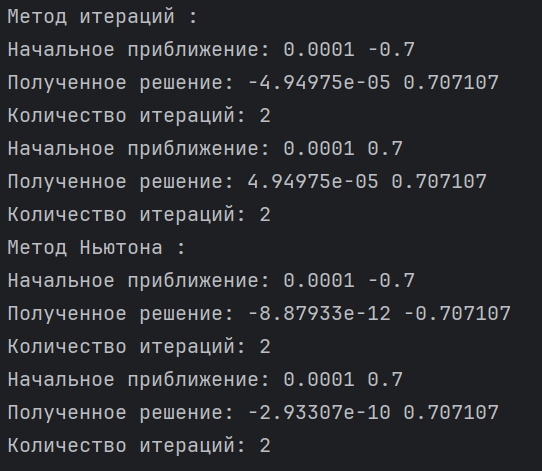




Тестовый пример 2:

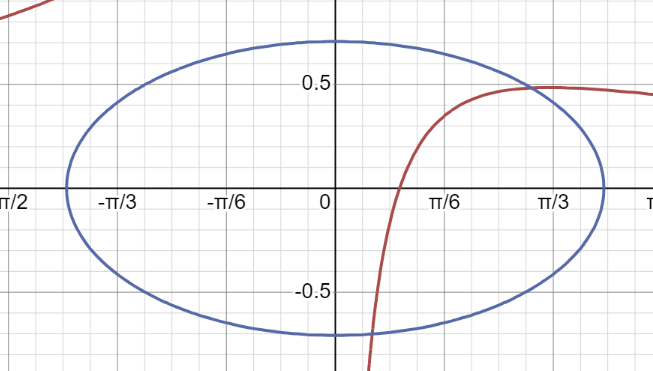
m = 0.0, a = 0.5. Начальные приближения: {0.0001;-0.7},{0.0001;0.7}.

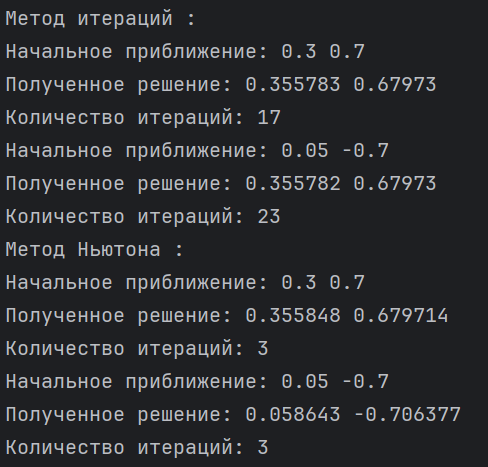




Тестовый пример 3:

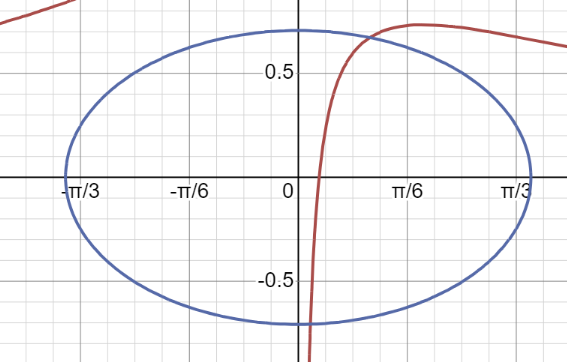
m = 0.1, a = 0.6. Начальные приближения: {0.3;0.7},{0.05;-0.7}.

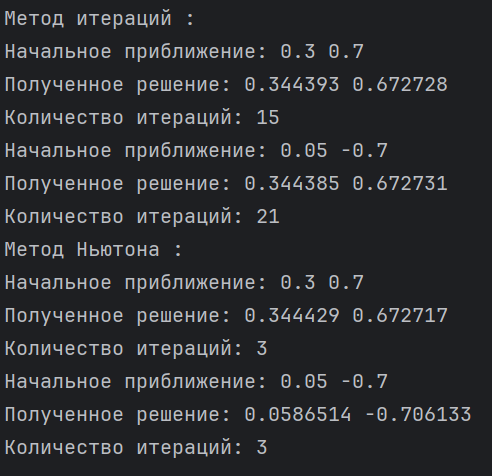




Тестовый пример 4:

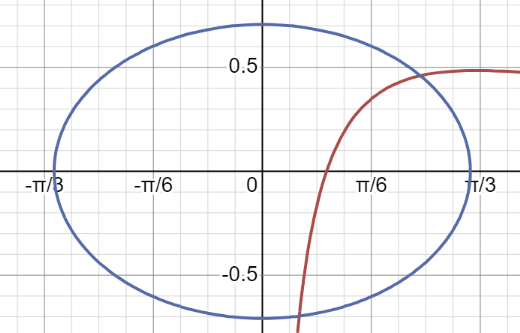
m = 0.1, a = 0.8. Начальные приближения: {0.3;0.7},{0.05;-0.7}.

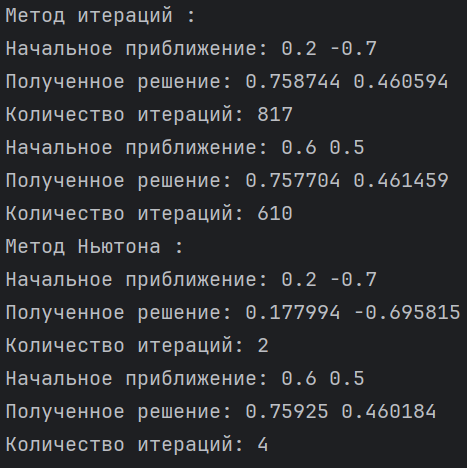




Тестовый пример 5:

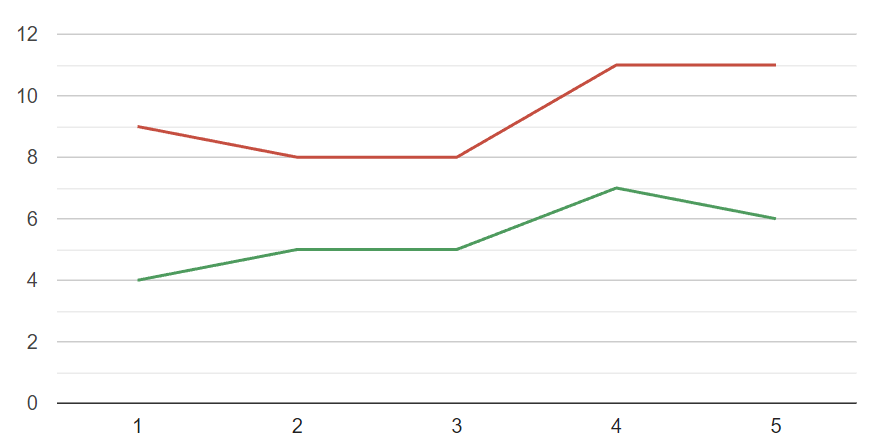
m = 0.3, a = 1.0. Начальные приближения: {0.2;-0.7},{0.6;0.5}.





**Сравнение трудоемкости методов**

Проведем сравнение временных затрат на выполнение каждого метода. Для сравнения временных затрат будем использовать данные из тестовых примеров. Временную зависимость изобразим на графике (красный – метод итераций, зеленый – метод Ньютона).



**Метод простых итераций (или метод последовательных приближений):**

Точность и сходимость: Этот метод имеет линейную сходимость, что означает, что он может потребовать большого количества итераций для достижения высокой точности решения. В некоторых случаях может потребоваться сильно увеличить число итераций для достижения желаемой точности.

Трудоемкость и вычислительные затраты: Метод простых итераций обычно менее требователен к затратам по сравнению с методом Ньютона, так как он не требует вычисления якобиана (матрицы производных) и обращения матрицы. Вместо этого он обновляет значения переменных по формулам, что обычно менее вычислительно интенсивно.

**Метод Ньютона:**

Точность и сходимость: Метод Ньютона имеет квадратичную сходимость, что означает, что он быстро сходится к решению при близких начальных приближениях. Однако при дальних начальных приближениях может потребоваться больше итераций.

Трудоемкость и вычислительные затраты: Метод Ньютона обычно более требователен к затратам, поскольку требует оценку якобиана и решение системы линейных уравнений на каждой итерации. Это может быть особенно затратным в случае больших систем уравнений.

**Вывод:**

В результате лабораторной работы были изучены методы решения систем нелинейных уравнений – метод простых итераций и метод Ньютона. Так же была построена блок-схема алгоритма и реализован программный продукт на языке программирования C++, который позволяет решать системы нелинейных уравнений данными методами. Была проанализирована трудоемкость каждого метода и сделаны соответствующие выводы.