МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
"Национальный исследовательский университет ИТМО"

ФАКУЛЬТЕТ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
Кафедра проектирования и безопасности компьютерных систем

ЛАБОРАТОРНАЯ РАБОТА №3
по дисциплине
"ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА"

по теме:
«Приближенное решение уравнений»

Выполнил: Нгуен Ле Минь
Группа: N3251
Преподаватель: Гришенцев А.Ю.

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург
2021

1 Задание

Задание 3.1 : Графическое решение уравнений

Графически определить верхние и нижние границы корней уравнений, проверить справедливость теоремы об отделении корней.

$5x^4 - 2x^3 + 3x^2 + x - 4 = 0$
$x^5 - 3x^4 + 7x^2 + x - 8 = 0$

Задание 3.2 : Метод половинного деления

Методом половинного деления найти хотя бы один вещественный корень уравнения, решение снабдить графиками функций и указать найденные корни :

$2x^3 + x^2 - 7 = 0;$
$5cos(3x) + 0,5x = 2, x \in [0; 2\pi];$
$x^5 - 2x^4 + 6x^2 + 2x - 4 = 0$
$x^3 - 0,2x^2 - 0,2x - 1,2 = 0$
$ln(|x^3|+1) + x^3 = 2$

Произвести оценку вычислительной сложности метода. Посчитать число итераций для решения уравнения с заданной точностью.

Задание 3.3 : Метод хорд

Методом хорд найти хотя бы один вещественный корень уравнения, решение снабдить графиками функций и указать найденные корни:

$5^x\sqrt{8^{x-1}} - 189 = 0$
$x^3 - x^2 + 2x - 5 = 0$
$2lg(x^2) - 5lg^2x - 4 = 0$
$2sin(2x) - cos(3x) = 0,5; x \in [0; 2\pi]$
$2x^3 - 7x^2 - 7x - 2,5 = 0$

Произвести оценку вычислительной сложности метода. Посчитать число итераций для решения уравнения с заданной точностью.

Задание 3.4 : Метод Ньютона

Методом Ньютона найти хотя бы один вещественный корень уравнения, решение снабдить графиками функций и указать найденные корни:

$2lg(x) - cos(x) = 0; x \in (0; 4\pi]$
$2x^3 - 5x^2 - 1 = 0;$
$2sin^3(2x) - cos(x) = 0; x \in [0; \pi)$
$x^5 - 3x^4 + 8x^2 + 2x - 7 = 0$
$0,5x^2 + 5cos(2x) - 2 = 0; x \in [-\pi; \pi]$
Произвести оценку вычислительной сложности метода. Посчитать число итераций для решения уравнения с заданной точностью.

Задание 3.5 : Метод итерации

Методом итераций найти хотя бы один вещественный корень уравнения, решение снабдить графиками функций и указать найденные корни:
$cosx = 0.1, x \in (0; 4\pi]$
$x^3 + x = 1000$
$x^5 - x^4 - x^2 - x - 5 = 0$
$x^3 - x - 1 = 0$
$\ln x + x = 2,25$
Произвести оценку вычислительной сложности метода. Посчитать число итераций для решения уравнения с заданной точностью.

Задание 3.6 : Метод Ньютона для комплексных корней

Методом Ньютона с заданной точностью найти хотя бы один комплекснозначный корень следующих уравнений:
$4z^4 + 2z^2 + 1,3 = 0$
$z^2 + 2,71 = 0$
$2e^z + \sqrt{2} = 0$
Произвести оценку вычислительной сложности метода. Посчитать число итераций для решения уравнения с заданной точностью.

Задание 3.7 : Метод Бернулли решения алгебраических уравнений

Методом Бернулли с заданной точностью найти хотя бы один корень следующих уравнений.
$5x^4 - 2x^3 + 3x^2 - x - 4 = 0$
$x^5 + 5x^4 - 5 = 0$
Произвести оценку вычислительной сложности метода. Посчитать число итераций для решения уравнения с заданной точностью.

2 Теория

3.1 : Графическое решение уравнений

Если функция $f(x)$ непрерывна на отрезке $[a, b]$ и на концах отрезка принимает ненулевые значения разных знаков, то на интервале (a, b) найдется по крайней мере одна точка ξ в которой $f(\xi) = 0$.

3.2 : Метод половинного деления

Если функция $f(x)$ непрерывна на промежутке $[a; b]$ и произведение значений функций на концах отрезка отрицательное, то в этом промежутке есть хотя бы один корень

3.3 : Метод хорд

Метод хорд представляет собой итерационный численный метод приближённого нахождения корня уравнения. Мы берем отрезок $[a; b]$ так, чтобы $f(a)*f(b)$, проводим прямую, пересекающую Ось абсцисс в точке x_0, и заменяем одну из крайних точек на x_0. Повторяя эти действия мы приближаемся к корню.

3.4 : Метод Ньютона

Пусть дано уравнение $f(x) = 0$, где $f(x)$ определено и непрерывно в некотором конечном или бесконечном интервале $x \in [a; b]$. Всякое значение ξ, обращающее функцию $f(x)$ в нуль, то есть такое, что $f(\xi) = 0$ называется корнем уравнения или нулем функции $f(x)$. Число ξ называется корнем k-ой кратности, если при $x = \xi$ вместе с функцией $f(x)$ обращаются в нуль ее производные до $(k - 1)$ порядка включительно: $f(\xi) = f'(\xi) = \ldots = f'^{(k-1)}(\xi) = 0$. Однократный корень называется простым. Приближенное нахождение корней уравнения складывается из двух этапов:

1. Отделение корней, то есть установление интервалов $[\alpha_i,\beta_i]$, в которых содержится один корень уравнения.

1.1 : $f(a) * f(b) < 0$ т.е. значения функции на его концах имеют противоположные знаки

1.2 : $f'(x)$ сохраняет постоянный знак, т.е. функция монотонна (эти два условия достаточны, но НЕ необходимы) для единственности корня на искомом отрезке).

1.3 : $f'(x)$ сохраняет постоянный знак, т.е. функция выпукла вверх, либо – вниз.

2. Уточнение приближенных корней, то есть доведение их до заданной точности.

3.5 :Метод итерации

Пусть дано уравнение $f(x) = 0$, где $f(x)$ - непрерывная функция, и требуется определить его вещественные корни.

Заменим уравнение равносильным уравнением : $x = \phi(x)$ (2)

Выберем каким-либо способом грубо приближенное значение корня x_0 и подставим его в правую часть уравнения (2). Тогда получим некоторое число : $x_1 = \phi(x_0)$ (3)

Подставляя теперь в правую часть равенства (3) вместо x_0 число x_1 получим новое число $x_2 = \phi(x_1)$. Повторяя этот процесс, будем иметь последовательность чисел : $x_n = \phi(x_{n-1}(n = 1, 2, ...)$ (4)

Если эта последовательность – сходящаяся, т.е. существует предел $\xi = \lim_{n\to+\infty} x_n$ то, переходя к пределу в равенстве (4) и предполагая функцию $\phi(x)$ непрерывной, найдем:

$\lim_{n\to+\infty} x_n = \phi(\lim_{n\to+\infty} x_{n-1})$ или $x = \phi(x)$ (5)

Таким образом, предел x является корнем уравнения (2) и может быть вычислен по формуле (4) с любой степенью точности.

Процесс итерации сходится, если $|\phi'(x)| < 1$

Условие достаточности приближения : $|x_n - x_{n-1}| < \frac{1-q}{q} * e$, где e - заданная точность, q - правильная дробь, такая что $|\phi'(x)| \leq q < 1$

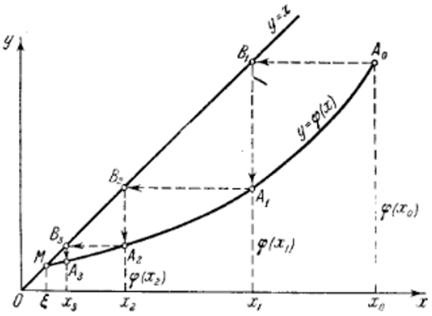


Рис. 1

3.6: Метод Ньютона для комплексных корней

Классический метод Ньютона или касательных заключается в том, что если x_n -некоторое приближение к корню x уравнения $f(x) = 0$, то следующее приближение определяется как корень касательной к функции $f(x)$, проведённой в точке x_n.

Уравнение касательной к функции $f(x)$ в точке x_n имеет вид :

$$f'(x_j) = \frac{y - f(x_n)}{x - x_n}$$

В уравнении касательной положим $y = 0$ и $x = x_{n+1}$

Тогда алгоритм последовательных вычислений в методе Ньютона состоит в следующем :

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

Сходимость метода касательных квадратичная, порядок сходимости равен 2

3.7: Метод Бернулли решения алгебраических уравнений

Пусть дано алгебраическое уравнение:

$a_0x^n + a_1x^{n-1} + ... + a_n = 0 (a_0 \neq 0)$

корни которого $x_1, x_2, \ldots, x_n$ различны

На основе коэффициентов a_k $(k = 0, 1, \ldots, n)$ построим так называемое конечно-разностное уравнение:

$a_0y_{n+i} + a_1y_{n+i-1} + ... + a_ny_i = 0$ $(i = 0, 1, 2, ...)$ которое представляет собой рекуррентное соотношение, связывающее любые, следующие друг за другом, $n+1$ членов бесконечной последовательности $y_0, y_1, y_2, ..., y_i, ...$

Последовательность $y_i = f(i) (i = 0, 1, 2, ...)$, члены которой удовлетворяют конечно-разностному уравнению, называется решением этого уравнения. Для построения решения y_i достаточно задать n его начальных значений $y_0, y_1, ..., y_{n-1}$

Теорема : Пусть алгебраическое уравнение имеет единственный наибольший по модулю корень x_1. Тогда отношение двух последовательных членов y_{i+1} и y_i решения конечно-разностного уравнения стремится, вообще говоря, к пределу, равному x_1, то есть : $\lim_{i\to+\infty} \frac{y_{i+1}}{y_i} = x_1$

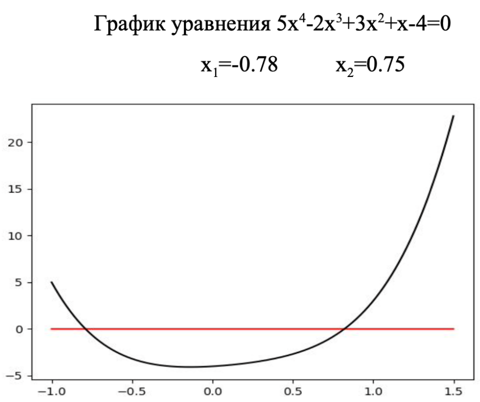
3 Программа

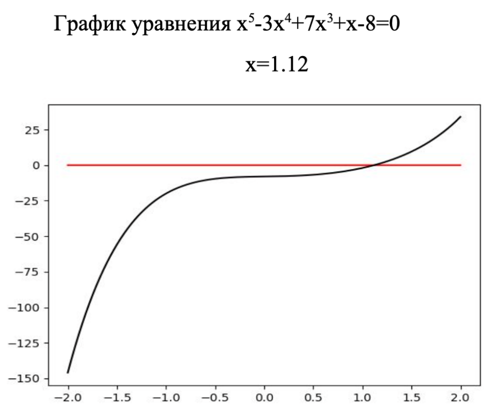
3.1 : Графическое решение уравнений

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 3
```

```
 */
#include <iostream>
#include <cmath>
#include "matplotlibcpp.h" /*Using this header to draw graphs*/
#include <vector>
using namespace std;
namespace plt = matplotlibcpp;
double miN = 0;
double maX = 3;
int points = 5000;
double root = /* Root of the equation*/;

double f(double x){
    return /*function*/;
}
double df(double x){
    return /*derivative function*/;
}

int main() {
    double fake_root1, fake_root2;
    double step = (maX - miN) / points;
    std::vector<double> x(points);
    for (int i = 0; i < points; i++)
        x.at(i) = miN + i * step;
    plt::plot(x, [](int d) { return 0; }, "r-", x, [](double d) { return f(d); }, "k-");
    plt::show();
    cout << "Enter the lower bound of the root according to the graph: ";

    cin >> fake_root1;
    cout << "Enter the upper bound of the root on the graph: ";
    cin >> fake_root2;
    cout << "Does the root approximation theorem hold?";
    if (fabs((fake_root1 + fake_root2) / 2 - root) <= abs(f((fake_root1 + fake_root2) / 2)) / abs(
        std::cout << "True" << std::endl;
    else
        std::cout << "False" << std::endl;
    return 0;
}
```

Результаты программы :



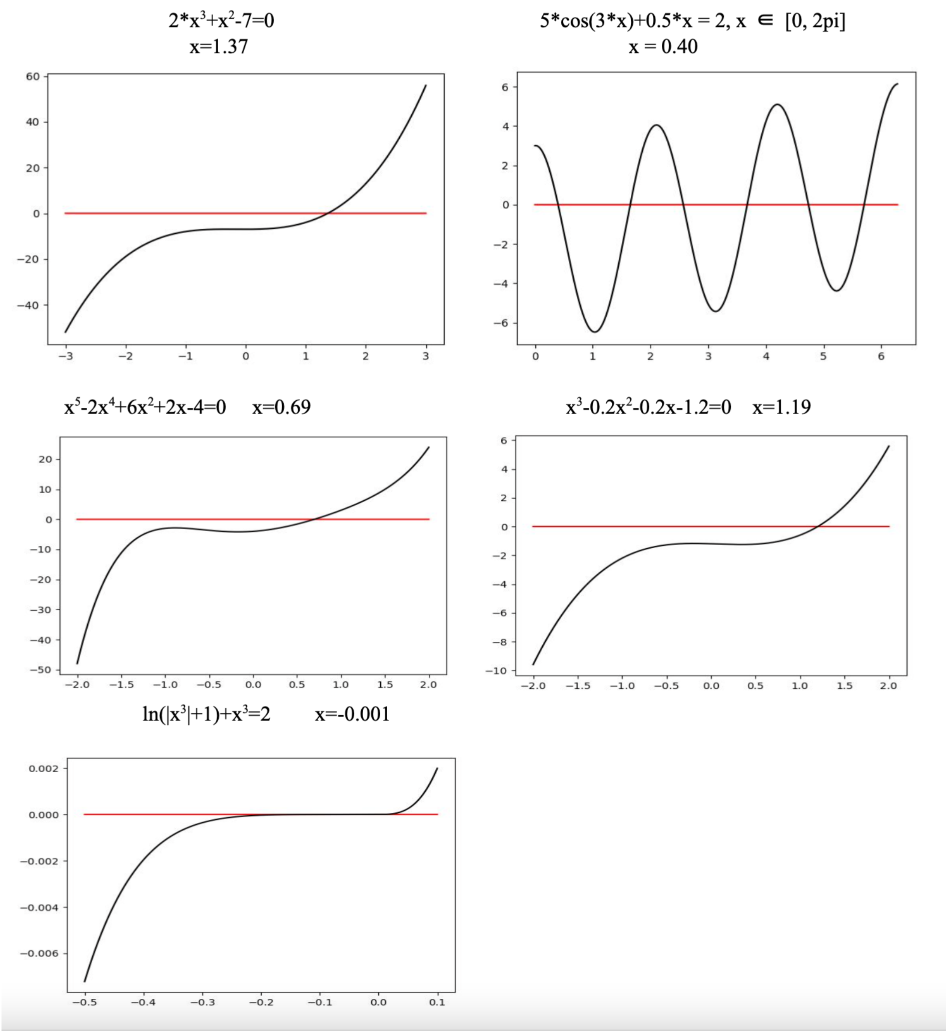


Вывод : В этом задании мы графически определили корень уравнения и проверили справедливость теоремы об отделении корней.

3.2 : Метод половинного деления

```cpp
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 3
 */
#include <iostream>
#include <cmath>
#include <vector>
#include "matplotlibcpp.h"
using namespace std;
namespace plt = matplotlibcpp;
double err = 0.001;
double miN = -10, maX = 10;
int points = 5000;
double f(double x){
    return /*function*/;
}
void divide(double min, double max, double err){ double mid;
    for(int i = 0; i < 1000; i++){
        mid = (min+max)/2; if(f(min) == 0){
            cout << "Root of this equation: " << min << endl;
            return;
        }
        else if(f(max) == 0){
            cout << "Root of this equation: " << max << endl;
            return;
        }
        else if(f(min)*f(mid) < 0)
            max = mid;
        else if(f(mid)*f(max) < 0)
            min = mid;
        else{
            cout << "It's a bad interval or no roots " << endl;
            return;
        }
        if(max-min < err){
            cout << "Root of this equation: " << (min+max)/2 << endl; return;
        }
    }
    cout << "Calculation limit exceeded " << endl;
}

int main() {
    divide(miN, maX, err);
    double step = (maX - miN)/points;
    std::vector<double> x(points);
    for(int i=0; i<points; i++)
        x.at(i) = miN + i*step;
    plt::plot(x, [](int d){return 0;}, "r-", x, [](double d) { return f(d); }, "k-"); plt::show();
    return 0;
}
```

Результаты программы :

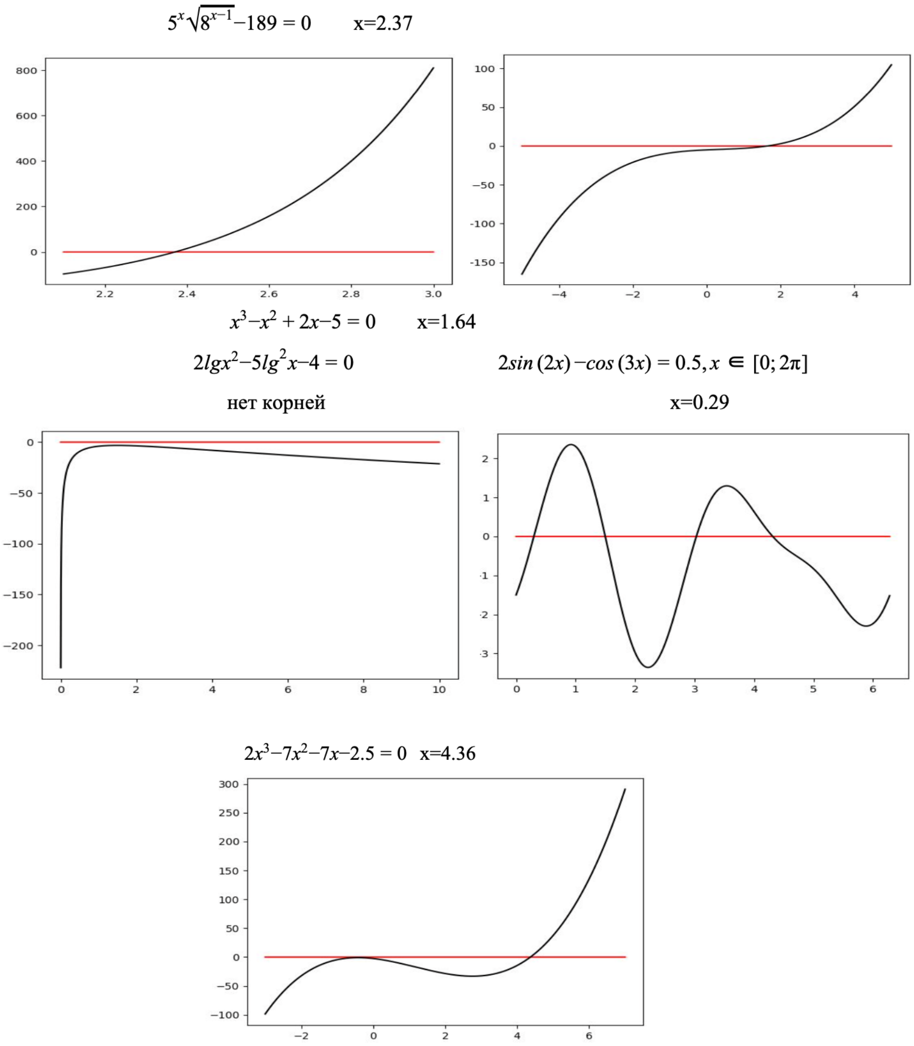


Вывод : В этом задании мы научились находить корень уравнения с помощью метода половинного деления.

3.3 : Метод половинного деления

```cpp
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 3
 */
#include <iostream>
#include <cmath>
#include <vector>
#include "matplotlibcpp.h"
using namespace std;

namespace plt = matplotlibcpp;

double err = 0.001;
double miN = 0, maX = 10;
int points = 5000;
double f(double x){
    return /*function*/;
}

void hord(double min, double max, double err) {
    if (f(min) * f(max) > 0) {
        cout << "It's a bad interval !!!" << endl;
        return;
    }
    for(int i = 0; i < 100; i++){
        min = max - (max - min) * f(max) / (f(max) - f(min));
        max = min + (min - max) * f(min) / (f(min) - f(max));
        if(fabs(max - min) < err){
            cout << "Roots of this equation are : " << max <<endl;
            return;
        }
    }
    cout << "Calculation limit exceeded !" << endl;
}

int main(){
    hord(miN, maX, err);
    double step = (maX - miN)/points;
    std::vector<double> x(points);
    for(int i = 0; i < points; i++)
        x.at(i) = miN + i*step;
    plt::plot(x, [](int d){return 0;}, "r-", x, [](double d) { return f(d); }, "k-"); plt::show();
    return 0;
}
```

Результаты программы :



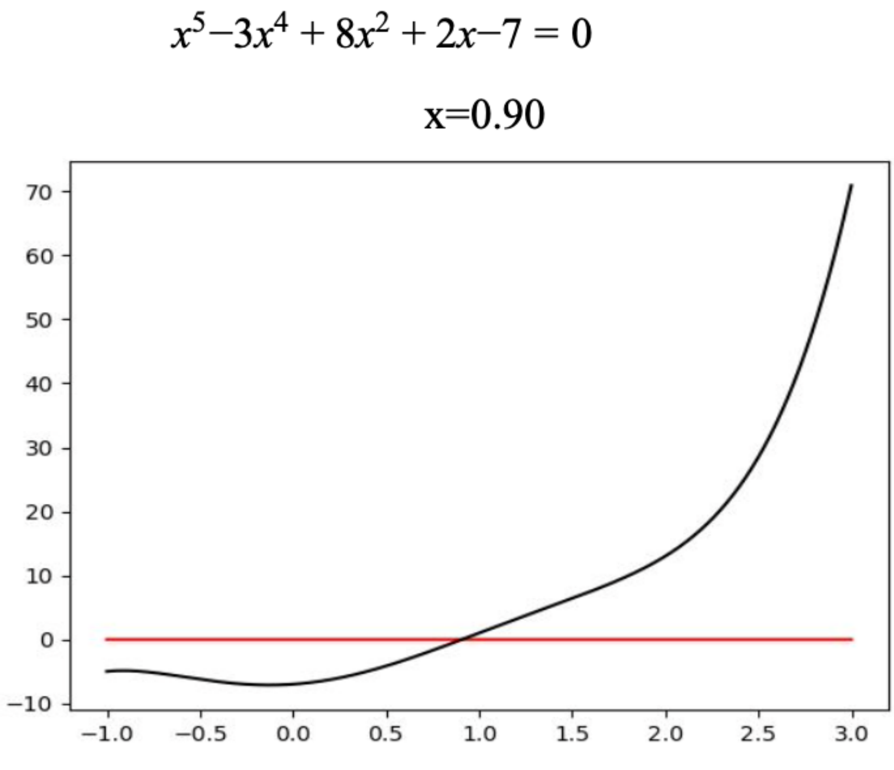
Вывод : В этом задании мы научились находить корень уравнения с помощью метода хорд.

3.4 : Метод Ньютона

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 3
 */
#include <iostream>
#include <cmath>
#include <vector>
#include "matplotlibcpp.h"

using namespace std;

namespace plt = matplotlibcpp;
double err = 0.001;
double x_0 = 1;
int points = 50001;
double miN = -1;
double maX = 3;

double f(double x){
    return /*function*/;
};
```

```
double df(double x){
    return /*function*/;
}

void newton(double x0, double err){
    double x1 = x0 - f(x0)/df(x0);
    for(int i = 0; i < 1000; i++){
        x0 = x1;
        x1 = x0 - f(x0)/df(x0);
        if(fabs(x0 - x1) < err){
            cout << "Roots of this equation are : " << x1 << endl;
            return;
        }
    }
    cout << "Calculation limit exceeded !" << endl;
}

int main(){
    newton(x_0, err);
    double step = (maX - miN)/points;
    std::vector<double> x(points);
    for(int i=0; i<points; i++)
        x.at(i) = miN + i*step;
    plt::plot(x, [](int d){return 0;}, "r-", x, [](double d) { return f(d); }, "k-"); plt::show();
    return 0;
}
```

Результаты программы : Мы выбираем уравнение $x^5 - 3x^4 + 8x^2 + 2x - 7 = 0$

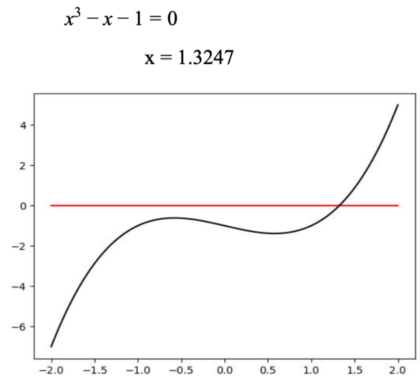


Вывод : В этом задании мы научились находить корень уравнения с помощью метода Ньютона.

3.5 : Метод итераций

```cpp
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 3
 */
#include <iostream>
#include <cmath>
#include <vector>
#include "matplotlibcpp.h"

using namespace std;
namespace plt = matplotlibcpp;

double err = 0.001;
double x_0 = 1.2;
int points = 5000;
double miN = 0;
double maX = 5;

double f(double x){
    return /*function*/;
}

double ff(double x){
    return /*function*/;
}

void iter(double x, double err){
    double x0; int iter = 0;
    do{
        x0 = x;
        x = ff(x); iter++;
    } while (fabs(x0 - x) > err && iter<30000);
    cout << "Root of this equation is: "<< x << endl;
}

int main(){
    iter(x_0, err);
    double step = (maX - miN)/points;
    std::vector<double> x(points);
    for(int i=0; i<points; i++) {
        x.at(i) = miN + i * step;
    }
    plt::plot(x, [](int d){return 0;}, "r-", x, [](double d) { return f(d); }, "k-"); plt::show();
    return 0;
}
```

Результаты программы : Мы выбираем уравнение $x^3 - x - 1 = 0$



Вывод : В этом задании мы научились реализовывать программу для нахождения корня уравнения методом итераций.

3.6 : Метод Ньютона для комплексных чисел

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 3
 */
#include <iostream>
#include <cmath>
#include <complex>

using namespace std;

typedef complex<double> comp;

double eps;

comp fx1(comp z) {
    return 4.0* pow(z, 4) + 2.0 * pow(z, 2) + 1.3;
}
comp dfx1(comp z) {
    return 16.0* pow(z, 3) + 4.0 * z;
}

comp fx2(comp z) {
    return pow(z, 2) + 2.71;
}
comp dfx2(comp z) {
    return 2.0* z;
}

comp fx3(comp z) {
    return 2.0* exp(z) + sqrt(2.0);
}
comp dfx3(comp z) {
    return 2.0* exp(z);
}

comp solve(comp fz(comp), comp dfz(comp), comp z0) {
    comp z1 = z0 - fz(z0) / dfz(z0);
    int iterations = 0;

    while (abs(z1.real() - z0.real()) > eps || abs(z1.imag() - z0.imag()) > eps) {
        iterations++;
        z0 = z1;
```

```
        z1 = z0 - fz(z0) / dfz(z0);
    }

    return z1;
}

int main() {
    comp x0 = 1.0 + 1.0i;
    int number;
    cout << "Please enter the number of equation " << endl;
    cin >> number;

    cout << "Enter the margin of error:  " << endl;
    cin >> eps;

    switch (number) {
        case 1: {
            cout << "x = " << solve(fx1, dfx1, x0) << endl;
            break;
        }
        case 2: {
            cout << "x = " << solve(fx2, dfx2, x0) << endl;
            break;
        }
        case 3: {
            cout << "x = " << solve(fx3, dfx3, x0) << endl;
            break;
        }
    }

    return 0;
}
```

Результаты программы :

```
Please enter the number of equation
2
Enter the margin of error:
0.0000001
x = (7.30465e-17,1.64621)
```

Вывод : Мы научились применять метод Ньютона для решения комплекснозначных уравнений с использованием средств языка C++.

3.7 : Метод Ньютона для комплексных чисел

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 3
 */
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

typedef vector<double> polynome;
const double eps = 0.000001;

double x(int num, int max, polynome& coeffs) {
```

```
    double cases[] = {0.0, 0.0, 0.0, 0.0, 1.0};
    if (num <= max) {
        return cases[num];
    }
    double retval = 0;
    for (int j = 1; j < coeffs.size(); j++) {
        retval += -coeffs[j] * x(num - j, max, coeffs);
    }
    return retval;
}

double calc(double y, polynome& coeffs) {
    int n = coeffs.size();
    double res = 0.0;
    for (int i = 0; i < n; i++) {
        res += pow(y, n-i-1)*coeffs[i];
    }
    return res;
}

polynome f = {1.0, 5.0, 0.0, 0.0, 0.0, -5.0};

double bernoulli(polynome& f) {
    int n = f.size()-1;
    int k = 0;
    double approx = x(n,n-1,f)/x(n-1,n-1,f);
    while (abs(calc(approx, f)) > eps) {
        k++;
        approx = x(n+k,n-1,f)/x(n+k-1,n-1,f); }
    cout << "Iterations: " << k << endl;
    return approx;
}

int main() {
    cout << bernoulli(f) << endl;
    return 0;
}
```

Результаты программы :

```
/Users/nguyenminh/CLionProjects/lab_computational/cmake-build-debug/lab37
Iterations: 12
-4.99195
```

Вывод : В ходе выполнения задания была освоена реализация метода Бернулли на языке C++ для решения алгебраических уравнений.