Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**ЗВІТ**

**Про виконання лабораторної роботи № 7**

**«Чисельні методи розв’язування систем нелінійних рівнянь»**

**з дисципліни «Чисельні методи»**

**Лектор:**

доцент кафедри ПЗ

Мельник Н.Б.

**Виконав:**

студ. групи ПЗ-18

Юшкевич А.І.

**Прийняв:**

проф. каф. ПЗ

Гавриш В.І.

«\_\_\_» \_\_\_\_\_\_ 2023 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2023

**Тема роботи:** Чисельні методи розв’язування систем нелінійних рівнянь.

**Мета роботи:** Ознайомлення на практиці з методом ітерацій та методом Ньютона розв’язування систем нелінійних рівнянь.

**Теоретичні відомості**

**Метод простої ітерації**

Нехай дана система нелінійних рівнянь виду

 (1)

де   – неперервно-диференційні функції.

Одним із найбільш простих алгоритмів її розв’язування є метод простої ітерації. Систему (1) запишемо у вигляді

 (2)

або у векторному вигляді .

Ітераційна послідовність будується за формулою

,  (3)

де - початкове наближення, яке має бути задано.

Достатньою умовою збіжності ітераційного процесу є виконання умови

, (4)

де - матриця з елементами  (= - норма матриці ) для довільного  із області визначення розв’язку. Відображення  називають стискуючим, якщо для двох довільних елементів  та  виконується умова

,

де коефіцієнт стискання  задовольняє нерівність 

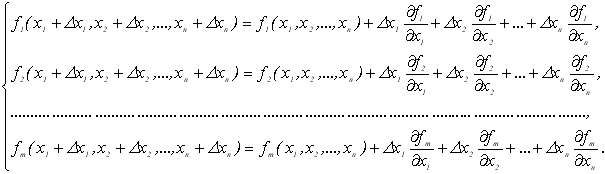
**Метод Ньютона**

Алгоритм методу базується на розкладі кожної функції системи в околі точки з координатами в ряд Тейлора.

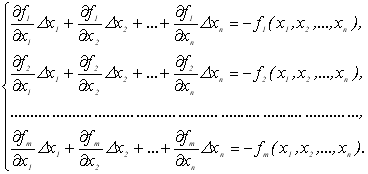
http://posibnyky.vntu.edu.ua/met/m1_t1_lecture4_src/m1_t1_lecture4_image468.png

члени рядів вищих порядків (http://posibnyky.vntu.edu.ua/met/m1_t1_lecture4_src/m1_t1_lecture4_image470.png тощо).

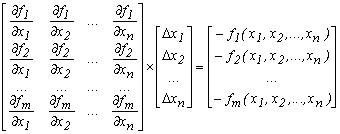
1. Початкова система буде мати вигляд:

 (4)

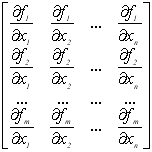
2. Припустимо, що прирости http://posibnyky.vntu.edu.ua/met/m1_t1_lecture4_src/m1_t1_lecture4_image474.png вибрані таким чином, що точки з координатами http://posibnyky.vntu.edu.ua/met/m1_t1_lecture4_src/m1_t1_lecture4_image476.png є коренями даної системи рівнянь з заданим степенем наближення http://posibnyky.vntu.edu.ua/met/m1_t1_lecture4_src/m1_t1_lecture4_image478.png. Тоді ліву частину рівнянь системи (2) можна прирівняти до нуля, тобто система рівнянь (2) буде мати вигляд:

     (5)

Або в матричній формі система (5) буде мати вигляд:

     (6)

де

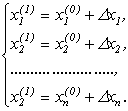


– матриця Якобі .

3. В результаті таких перетворень система рівнянь може розглядатися як система лінійних алгебраїчних рівнянь відносно http://posibnyky.vntu.edu.ua/met/m1_t1_lecture4_src/m1_t1_lecture4_image485.png. В такому випадку, якщо врахувати, що заданий вектор х початкових наближень виду:

http://posibnyky.vntu.edu.ua/met/m1_t1_lecture4_src/m1_t1_lecture4_image486.png,

можливо розв'язувати систему відносно вектора приросту http://posibnyky.vntu.edu.ua/met/m1_t1_lecture4_src/m1_t1_lecture4_image488.png, та знайти розв'язок системи, як сума попереднього значення та вектора http://posibnyky.vntu.edu.ua/met/m1_t1_lecture4_src/m1_t1_lecture4_image489.png:

     (7)

Дану задачу можна розв'язати з будь-якої точки, вибравши вектор початкових наближень.

4. Процес розв’язання системи нелінійних рівнянь ітераційний та буде продовжуватись до тих пір, поки всі координати вектору приростів не стануть меншими за абсолютною величиною від заданої похибки http://posibnyky.vntu.edu.ua/met/m1_t1_lecture4_src/m1_t1_lecture4_image127.png, тобто http://posibnyky.vntu.edu.ua/met/m1_t1_lecture4_src/m1_t1_lecture4_image493.png.

**Індивідуальне завдання**

1. Ознайомитись з теоретичними відомостями.

2. Розв'язати систему нелінійних рівнянь з точністю eps = 10-3 методом ітерацій та методом Ньютона.

**Варіант 22**



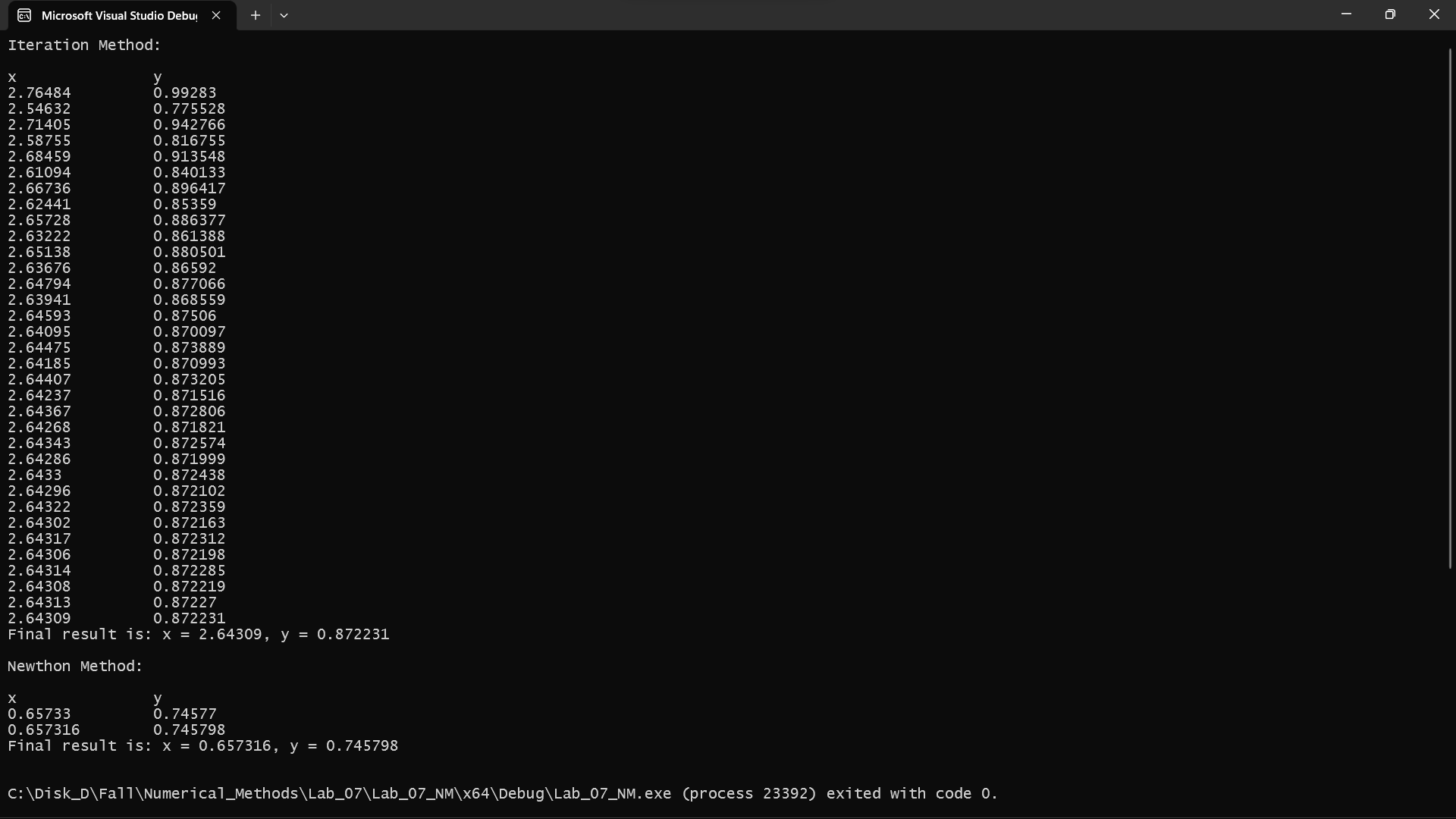


Рис.1. Результат виконання програми

**Код програми**

CNumericalMethods.h:

#pragma once

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

struct SResult {

double x;

double y;

};

class CNumericalMethods

{

public:

CNumericalMethods() = delete;

CNumericalMethods(double x, double y);

CNumericalMethods(double x, double y, double epsilon);

SResult Iteration();

SResult Newthon();

private:

double m\_epsilon;

const double number\_of\_equasions{ 2 };

SResult m\_result;

double Func1(double y) const;

double Func2(double x) const;

double GetDerivativeByX(double x) const;

double GetDerivativeByY(double y) const;

double FByXY(double x, double y) const;

double GByXY(double x, double y) const;

double FByXDerivative(double x, double y) const;

double FByYDerivative(double x, double y) const;

double GByXDerivative(double x, double y) const;

double GByYDerivative(double x, double y) const;

bool IsProcessConvergent(double first\_derivative, double second\_derivative) const;

bool GetCloserIteration();

bool GetCloserNewthon(vector<vector<double>>& matrix\_j, vector<double> f);

void SetTranspondedJNewthon(vector<vector<double>>& matrix\_j, double x, double y) const;

void SetTranspondedJ(vector<vector<double>>& matrix\_j, double x, double y) const;

double GetDeterminant(vector<vector<double>>& matrix\_j) const;

vector<double> MultiplyMatrixAndColumn(const vector<vector<double>> matrix, const vector<double> column) const;

};

CNumericalMethods.cpp:

#include "CNumericalMethods.h"

CNumericalMethods::CNumericalMethods(double x, double y) {

m\_result.x = x;

m\_result.y = y;

m\_epsilon = 0.0001;

}

CNumericalMethods::CNumericalMethods(double x, double y, double epsilon) : m\_epsilon(epsilon) {

m\_result.x = x;

m\_result.y = y;

}

double CNumericalMethods::Func1(double y) const {

return 2 + cos(y);

}

double CNumericalMethods::Func2(double x) const{

return 0.8 - cos(x - 1);

}

double CNumericalMethods::GetDerivativeByX(double x) const{

return sin(x-1);

}

double CNumericalMethods::GetDerivativeByY(double y) const{

return -sin(y);

}

bool CNumericalMethods::IsProcessConvergent(double first\_derivative, double second\_derivative) const {

return first\_derivative < 1 && second\_derivative < 1;

}

bool CNumericalMethods::GetCloserIteration() {

SResult previous = m\_result;

m\_result.x = Func1(m\_result.y);

m\_result.y = Func2(m\_result.x);

cout << m\_result.x << "\t\t" << m\_result.y << endl;

return fabs(m\_result.x - previous.x) + fabs(m\_result.y - previous.y) < m\_epsilon;

}

SResult CNumericalMethods::Iteration() {

if (!IsProcessConvergent(GetDerivativeByX(m\_result.x), GetDerivativeByY(m\_result.y))) {

m\_result.x = NAN;

m\_result.y = NAN;

}

else {

while (!GetCloserIteration());

}

return m\_result;

}

void CNumericalMethods::SetTranspondedJ(vector<vector<double>>& matrix\_j, double x, double y) const {

matrix\_j[0][0] = -1;

matrix\_j[0][1] = -GetDerivativeByY(y);

matrix\_j[1][0] = -GetDerivativeByX(x);

matrix\_j[1][1] = -1;

}

void CNumericalMethods::SetTranspondedJNewthon(vector<vector<double>>& matrix\_j, double x, double y) const {

matrix\_j[0][0] = GByYDerivative(x, y);

matrix\_j[0][1] = -FByYDerivative(x, y);

matrix\_j[1][0] = -GByXDerivative(x, y);

matrix\_j[1][1] = FByXDerivative(x, y);

}

double CNumericalMethods::GetDeterminant(vector<vector<double>>& matrix\_j) const {

return matrix\_j[0][0] \* matrix\_j[1][1] - (matrix\_j[0][1] \* matrix\_j[1][0]);

}

vector<double> CNumericalMethods::MultiplyMatrixAndColumn(const vector<vector<double>> matrix, const vector<double> column) const {

vector<double> local\_column(column.size());

double sum = 0;

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

for (int j = 0; j < matrix[0].size(); j++) {

local\_column[i] += matrix[i][j] \* column[j];

}

}

return local\_column;

}

bool CNumericalMethods::GetCloserNewthon(vector<vector<double>>& matrix\_j, vector<double> f) {

SResult previous = m\_result;

SetTranspondedJNewthon(matrix\_j, m\_result.x, m\_result.y);

double determinant = GetDeterminant(matrix\_j);

f[0] = FByXY(m\_result.x, m\_result.y);

f[1] = FByXY(m\_result.x, m\_result.y);

for (int i = 0; i < matrix\_j.size(); i++) {

for (int j = 0; j < matrix\_j.size(); j++) {

matrix\_j[i][j] \*= (1/determinant);

}

}

vector<double> j\_and\_f = MultiplyMatrixAndColumn(matrix\_j, f);

m\_result.x = m\_result.x - j\_and\_f[0];

m\_result.y = m\_result.y - j\_and\_f[1];

cout << m\_result.x << " \t" << m\_result.y << endl;

return fabs(m\_result.x - previous.x) + fabs(m\_result.y - previous.y) < m\_epsilon;

}

SResult CNumericalMethods::Newthon() {

vector<vector<double>> matrix\_j(number\_of\_equasions, vector<double>(number\_of\_equasions));

vector<double> f(number\_of\_equasions);

if (!IsProcessConvergent(GetDerivativeByX(m\_result.x), GetDerivativeByY(m\_result.y))) {

m\_result.x = NAN;

m\_result.y = NAN;

}

else {

while (!GetCloserNewthon(matrix\_j, f));

}

return m\_result;

}

double CNumericalMethods::FByXY(double x, double y) const {

return sin(x + y) - 1.5 \* x;

}

double CNumericalMethods::GByXY(double x, double y) const {

return x \* x + y \* y - 1;

}

double CNumericalMethods::FByXDerivative(double x, double y) const{

return cos(x + y) - 1.5;

}

double CNumericalMethods::FByYDerivative(double x, double y) const{

return cos(x + y);

}

double CNumericalMethods::GByXDerivative(double x, double y) const{

return 2 \* x;

}

double CNumericalMethods::GByYDerivative(double x, double y) const{

return 2 \* y;

}

Lab\_07\_NM.cpp:

#include <iostream>

#include "CNumericalMethods.h"

int main()

{

double epsilon = 0.0001;

cout << "Iteration Method:\n\nx\t\ty\n";

CNumericalMethods iter(2.5, 0.7, epsilon);

SResult result = iter.Iteration();

cout << "Final result is: x = " << result.x << ", y = " << result.y << "\n\n";

cout << "Newthon Method:\n\nx\t\ty\n";

CNumericalMethods newthon(0.65, 0.76, epsilon);

result = newthon.Newthon();

cout << "Final result is: x = " << result.x << ", y = " << result.y << "\n\n";

}

**Висновки**

У результаті виконання лабораторної роботи визначено дійсні корені нелінійної системи алгебраїчних рівнянь з заданою точністю 10-3 методом ітерацій та методом Ньютона. Розв’язок отриманий з заданою точністю, методом простої ітерації за 5 кроки, а за методом Ньютона за 4 кроки.