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.

$\sum$ = ____ ___________

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

$$\begin{cases} f_1(x_1, x_2, ..., x_n) = 0, \\ f_2(x_1, x_2, ..., x_n) = 0, \\ ... \\ f_n(x_1, x_2, ..., x_n) = 0. \end{cases} \quad (1)$$

де $f_1, f_2, ..., f_n$ – неперервно-диференційні функції.

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

$$\begin{cases} x_1 = \varphi_1(x_1, x_2, ..., x_n), \\ x_2 = \varphi_2(x_1, x_2, ..., x_n), \\ ..............................., \\ x_n = \varphi_n(x_1, x_2, ..., x_n) \end{cases} \quad (2)$$

або у векторному вигляді $X = \phi(X)$.

Ітераційна послідовність будується за формулою

$$X^{(k+1)} = \phi(X^{(k)}), \qquad k = 0,1,2,..., \quad (3)$$

де $X^{(0)}$ - початкове наближення, яке має бути задано.

Достатньою умовою збіжності ітераційного процесу є виконання умови

$$\|M\| \le q < 1 \quad , \quad (4)$$

де $M$ - матриця з елементами $m_{ij} = \dfrac{\partial \varphi_i}{\partial x_j}$ ($\|M\| = \max\limits_j \sum\limits_{i=1}^{n} \left| \dfrac{\partial \varphi_i(x)}{\partial x_j} \right|$ - норма матриці $M$

) для довільного $X$ із області визначення розв'язку. Відображення $\overline{\varphi(X)}$ називають стискуючим, якщо для двох довільних елементів $X_1$ та $X_2$ виконується умова

$$\left\| \overline{\varphi(X_1)} - \overline{\varphi(X_2)} \right\| \le q \|X_1 - X_2\|,$$

де коефіцієнт стискання $q$ задовольняє нерівність $0 < q < 1$.

## Метод Ньютона

Алгоритм методу базується на розкладі кожної функції системи в околі точки з координатами $X = \{x_1, x_2, \dots, x_n\}$ в ряд Тейлора.

$$f_i(x_1 + \Delta x_1, x_2 + \Delta x_2, \dots, x_n + \Delta x_n) = f_i(x_1, x_2, \dots, x_n) + \Delta x_1 \frac{\partial f_i}{\partial x_1} + \Delta x_2 \frac{\partial f_i}{\partial x_2} + \dots + \Delta x_n \frac{\partial f_i}{\partial x_n} +$$

члени рядів вищих порядків ($f', f''$ тощо).

1. Початкова система буде мати вигляд:

$$\begin{cases} f_1(x_1 + \Delta x_1, x_2 + \Delta x_2, \dots, x_n + \Delta x_n) = f_1(x_1, x_2, \dots, x_n) + \Delta x_1 \frac{\partial f_1}{\partial x_1} + \Delta x_2 \frac{\partial f_1}{\partial x_2} + \dots + \Delta x_n \frac{\partial f_1}{\partial x_n}, \\ f_2(x_1 + \Delta x_1, x_2 + \Delta x_2, \dots, x_n + \Delta x_n) = f_2(x_1, x_2, \dots, x_n) + \Delta x_1 \frac{\partial f_2}{\partial x_1} + \Delta x_2 \frac{\partial f_2}{\partial x_2} + \dots + \Delta x_n \frac{\partial f_2}{\partial x_n}, \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots, \\ f_m(x_1 + \Delta x_1, x_2 + \Delta x_2, \dots, x_n + \Delta x_n) = f_m(x_1, x_2, \dots, x_n) + \Delta x_1 \frac{\partial f_m}{\partial x_1} + \Delta x_2 \frac{\partial f_m}{\partial x_2} + \dots + \Delta x_n \frac{\partial f_m}{\partial x_n}. \end{cases} \quad (4)$$

2. Припустимо, що прирости $\Delta x_i$ вибрані таким чином, що точки з координатами $x_1 + \Delta x_1, x_2 + \Delta x_2, \dots, x_n + \Delta x_n$ є коренями даної системи рівнянь з заданим степенем наближення $\varepsilon$. Тоді ліву частину рівнянь системи (2) можна прирівняти до нуля, тобто система рівнянь (2) буде мати вигляд:

$$\begin{cases} \frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 + \dots + \frac{\partial f_1}{\partial x_n}\Delta x_n = -f_1(x_1, x_2, \dots, x_n), \\ \frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 + \dots + \frac{\partial f_2}{\partial x_n}\Delta x_n = -f_2(x_1, x_2, \dots, x_n), \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots, \\ \frac{\partial f_m}{\partial x_1}\Delta x_1 + \frac{\partial f_m}{\partial x_2}\Delta x_2 + \dots + \frac{\partial f_m}{\partial x_n}\Delta x_n = -f_m(x_1, x_2, \dots, x_n). \end{cases} \quad (5)$$

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

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \times \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \dots \\ \Delta x_n \end{bmatrix} = \begin{bmatrix} -f_1(x_1, x_2, \dots, x_n) \\ -f_2(x_1, x_2, \dots, x_n) \\ \dots \\ -f_m(x_1, x_2, \dots, x_n) \end{bmatrix} \quad (6)$$

де

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

– матриця Якобі .

3. В результаті таких перетворень система рівнянь може розглядатися як система лінійних алгебраїчних рівнянь відносно $\Delta x_i$. В такому випадку, якщо врахувати, що заданий вектор x початкових наближень виду:

$$\begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \dots \\ x_n^{(0)} \end{bmatrix},$$

можливо розв'язувати систему відносно вектора приросту $\Delta x_i$, та знайти розв'язок системи, як сума попереднього значення та вектора $\Delta x_i$:

$$\begin{cases} x_1^{(1)} = x_1^{(0)} + \Delta x_1, \\ x_2^{(1)} = x_2^{(0)} + \Delta x_2, \\ \dots\dots\dots\dots\dots\dots\dots\dots, \\ x_2^{(1)} = x_n^{(0)} + \Delta x_n. \end{cases} \quad (7)$$

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 $\varepsilon$, тобто $|\Delta x| \le \varepsilon$.

## 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**

№ 22. 1) $\begin{cases} \cos(x-1) + y = 0{,}8; \\ x - \cos y = 2 \end{cases}$ 2) $\begin{cases} \sin(x+y) - 1{,}5x = 0; \\ x^2 + y^2 = 1. \end{cases}$

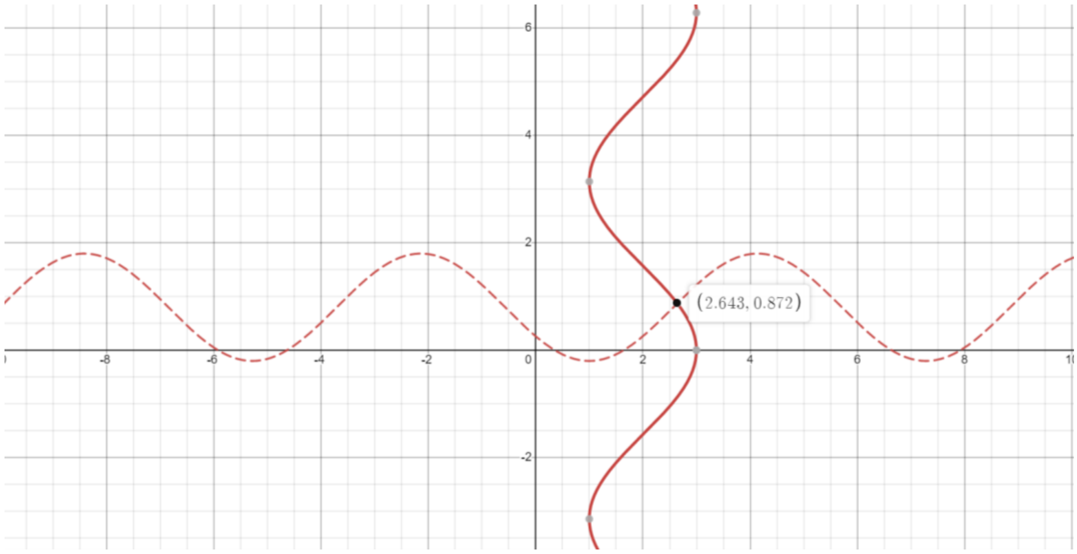


Рис. 1. Перша система рвінянь

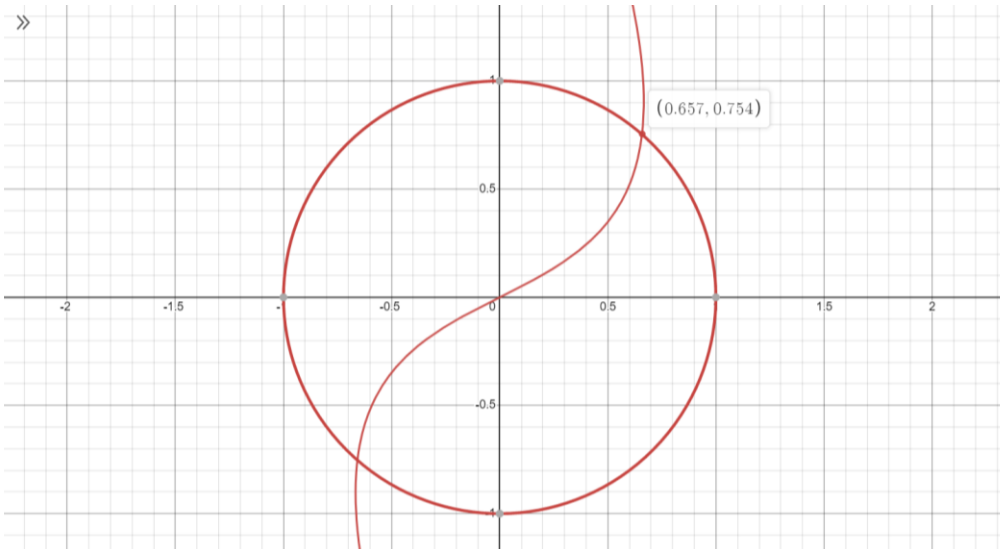


Рис. 2. Друга система рівнянь

```
Iteration Method:

x               y
2.76484         0.99283
2.54632         0.775528
2.71405         0.942766
2.58755         0.816755
2.68459         0.913548
2.61094         0.840133
2.66736         0.896417
2.62441         0.85359
2.65728         0.886377
2.63222         0.861388
2.65138         0.880501
2.63676         0.86592
2.64794         0.877066
2.63941         0.868559
2.64593         0.87506
2.64095         0.870097
2.64475         0.873889
2.64185         0.870993
2.64407         0.873205
2.64237         0.871516
2.64367         0.872806
2.64268         0.871821
2.64343         0.872574
2.64286         0.871999
2.6433          0.872438
2.64296         0.872102
2.64322         0.872359
2.64302         0.872163
2.64317         0.872312
2.64306         0.872198
2.64314         0.872285
2.64308         0.872219
2.64313         0.87227
2.64309         0.872231
Final result is: x = 2.64309, y = 0.872231

Newthon Method:

x               y
0.65733         0.74577
0.657316        0.745798
Final result is: x = 0.657316, y = 0.745798
```

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.