МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НУАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
ІНСТИТУТ МАТЕМАТИКИ І ФУНДАМЕНТАЛЬНИХ НАУК
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ

КУРСОВА РОБОТА
З ДИСЦИЛІНИ «ЧИСЕЛЬНІ МЕТОДИ ЧАСТИНА 2»
НА ТЕМУ:

«Метод Ньютона та модифікований метод Ньютона розв’язування систем нелінійних рівнянь»

Виконав: студент групи ПМ-41
Дудяк Михайло
Перевірив: доцент Пізюр Я. В.

Львів 2016

Досліджено застосування методу Ньютона (та його модифікованої версії) розв’язування системи нелінійних рівнянь. Написано програму на мові C++ (з використанням бібліотеки Eigen) для розв’язування систем нелінійних рівнянь 2 та 3 порядку з заданими похибкою та початковим наближенням.

Зміст

Вступ

У зв’язку з розвитком нової обчислювальної техніки сучасна інженерна практика все частіше і частіше зустрічається з математичними задачами, точний розв’язок яких отримати складно або неможливо. В таких випадках зазвичай застосовують ті чи інші методи наближених обчислень. Саме тому наближені і чисельні методи математичного аналізу набули широкого розвитку та неабиякого значення для сучасної математичної науки.

В цій курсовій роботі розглянуто відомий метод Ньютона та його модифікацію, застосовані для розв’язування систем нелінійних рівнянь. Розв’язок останніх – одна зі складних задач обчислювальної математики. Складність полягає в тому, щоб визначити, чи має система корені і, якщо так, то скільки.

В розділі «Додатки» наведено вихідний код на C++ програми, яка демонструє роботу вищезгаданих методів.

Теоретичні відомості

Постановка задачі

Дано систему з n нелінійних рівнянь з n невідомими:

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

де $f_i(x_1, x_2 \dots x_n)$: $\mathbb{R}^n \to \mathbb{R}, i = \overline{1,n}$ – нелінійні функції, визначені та неперервні в деякій області $G \subset \mathbb{R}^n$, або в векторному вигляді:

$$F(x) = 0 \quad (2)$$

де $x = (x_1, x_2 \dots x_n)^T, F(x) = \big(f_1(x), f_2(x) \dots f_n(x)\big)^T$.

Необхідно знайти такий вектор $x^* = (x_1^*, x_2^* \dots x_n^*)^T$, який при підстановці в систему (1) перетворює кожне рівняння в тотожність.

Примітки:

1. Проблема розв'язування системи (1) постає при розв'язуванні багатьох прикладних задач, наприклад, пошуку безумовного екстремуму функції багатьох змінних за допомогою необхідних умов, при застосуваннінеявних методів інтегрування звичайних диференціальних рівнянь і т.д.
2. Задачу пошуку комплексних коренів $f(z) = 0$ можна звести до проблеми розв'язуваннядвох рівнянь з двома невідомими. Для цього слід покласти $z = x + iy$ і виділити дійсну та уявну частини функції:

$$f(z) = u(x, y) + iv(x, y) = 0 \quad (3)$$

   Звідси отримуємо систему $\begin{cases} \mathrm{u(x,y)} = 0 \\ \mathrm{v(x,y)} = 0 \end{cases}$, яку можна розв'язати одним з методів, описаних в цій роботі.
3. Застосування розглядуваних надалі методів вимагає знаходження початкового наближення $x^{(0)}$. У випадку $n = 2$ це можна зробити графічно, визначивши наближено координати точки перетину кривих, описаних рівняннями $f_1(x_1, x_2) = 0$ і $f_2(x_1, x_2) = 0$.
4. Задачу розв'язування системи (1) можна звести до задачі пошуку мінімуму функції $\Psi(x) = \sum_{i=1}^{n} f_i(x_1, \dots, x_n)$. Оскільки функція $\Psi(x)$ невід'ємна, її мінімальне значення, рівне нулю, досягається в точці x^* , яка і є розв'язком системи (1). Для

пошуку мінімуму функції $\Psi(x)$ можна застосувати різноманітні методи пошуку безумовного екстремуму функції багатьох змінних(першого, другого, нульового порядків).

Метод Ньютона розв'язування систем нелінійних рівнянь

Цей метод застосовується для розв'язання систем виду (1) або (2). Формула для знаходження розв'язку є узагальненням формули методу Ньютона у одновимірному випадку:

$$x^{(k+1)} = x^{(0)} - \frac{f\left(x^{(k)}\right)}{f'(x^{(k)})}, k = 0,1,2, \dots \quad (4)$$

Виглядає вона наступним чином:

$$x^{(k+1)} = x^{(k)} - J^{-1}\left(x^{(k)}\right) - F\left(x^{(k)}\right), k = 0,1,2, \dots \quad (5)$$

де $J(x) = \begin{pmatrix} \frac{\partial f_1(x)}{\partial x_1} & \dots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \dots & \frac{\partial f_n(x)}{\partial x_n} \end{pmatrix}$ – матриця Якобі.

Оскільки процес обчислення оберненої матриці є трудомістким, перетворимо (5) наступним чином:

$$\Delta x^{(k)} = -J^{-1}\left(x^{(k)}\right) \cdot F\left(x^{(k)}\right), k = 0,1,2, \dots \quad (6)$$

де $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$ – поправка до поточного наближення $x^{(k)}$.

Помножимо останній вираз зліва на матрицю Якобі $J(x^{(k)})$:

$$J\left(x^{(k)}\right) \cdot \Delta x^{(k)} = -J\left(x^{(k)}\right)J^{-1}\left(x^{(k)}\right)F\left(x^{(k)}\right) = -F\left(x^{(k)}\right), k = 0,1,2, \dots \quad (7)$$

В результаті отримуємо систему лінійних алгебраїчних рівнянь відносно поправки $\Delta x^{(k)}$. Після її отримання обчислюється наступне наближення $x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$.

Алгоритм застосування методу Ньютона

1. Задати початкове наближення $x^{(0)}$ і мале додатнє число ϵ (точність). Покласти $k = 0$.
2. Розв'язати систему лінійних алгебраїчних рівнянь відносно поправки $\Delta x^{(k)}$:

$$J\left(x^{(k)}\right) \cdot \Delta x^{(k)} = -F(x^{(k)}) \quad (8)$$

3. Обчислити наступне наближення: $x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$.

4. Якщо $\Delta^{(k+1)} = \max_i \left| x_i^{(k+1)} - x_i^{(k)} \right| \leq \epsilon$, завершити процес і покласти $x^* \cong x^{(k+1)}$. Якщо ж $\Delta^{(k+1)} > \epsilon$, то покласти $k = k + 1$ і перейти до пункту №2.

Достатні умови збіжності методу Ньютона для систем

Теорема 1 (про достатні умови збіжності методу Ньютона):

Нехай функція $F(x)$ неперервно диференційовна на відкритій опуклій множині $G \subset \mathbb{R}^n$. Припустимо, що існують $x^* \in \mathbb{R}^n$ і $r, \beta > 0$, такі, що $N(x^*, r) \subset G, F(x^*) = 0$ та існує $J^{-1}(x^*)$, причому $\|J^{-1}(x^*)\| \leq \beta$ і $J(x) \in Lip_\gamma(N(x^*, r))$. Тоді існує $\epsilon > 0$ таке, що для будь-якого $x^{(0)} \in N(x^*, \epsilon)$ послідовність $x^{(1)}, x^{(2)}, \ldots$, породжена співвідношенням (5), збігається до x^* і задовільняє нерівність

$$\left\| x^{(k+1)} - x^* \right\| \leq \beta \cdot \gamma \cdot \left\| x^{(k)} - x^* \right\|, k = 0,1,2, \ldots \quad (9)$$

Тут використано наступні позначення: $N(x, r) = \{\bar{x} \in \mathbb{R}^n : |\bar{x} - x^*| < r\}$; запис $J(x) \in Lip_\gamma(N(x^*, r))$ значить, що $J(x)$ неперервна за Ліпшицем, де γ – константа Ліпшиця, тобто

$$\|J(y) - J(x)\| \leq \gamma \cdot \|y - x\| \; \forall x, y \in N(x^*, r). \quad (10)$$

Примітки:

1. Теорема 1 свідчить про локальну квадратичну збіжність методу Ньютона.
2. До недоліків методу Ньютона слід віднести:
   - Необхідність вибору досить точного початкового наближення;
   - Відсутність глобальної збіжності для багатьох задач;
   - Необхідність обчислення матриці Якобі на кожній ітерації;
   - Необхідність розв'язування на кожній ітерації системи лінійних рівнянь, яка може бути погано обумовленою.
3. Перевагою методу є квадратична збіжність з хорошого початкового наближення за умови невиродженості матриці Якобі.

Модифікований метод Ньютона

В цьому методі, на відміну від звичайного методу Ньютона, пошук оберненої матриці відбувається лише один раз в початковій точці $x^{(0)}$:

$$x^{(k+1)} = x^{(k)} - J^{-1}\left(x^{(0)}\right) - F\left(x^{(k)}\right), k = 0,1,2, \ldots \quad (11)$$

Варто зазначити, що при розв'язуванні одновимірного випадку $f(x) = 0$ модифікованим методом Ньютона похідну функції обчислюють також лише раз в початковій точці.

Методика розв’язування задачі аналогічна до застосовуваної у попереднього випадку, де замість (8) використовується система

$$J\left(x^{(0)}\right) \cdot \Delta x^{(k)} = -F\left(x^{(k)}\right), k = 0,1,2, \ldots \quad (12)$$

Матриця цієї системи – $J(x^{(0)})$ – залишається сталою на кожній ітерації, що дає змогу прискорити розв'язання системи за допомогою LU-розкладу її матриці.

Очевидно, в загальному випадку збіжність модифікованого методу гірша в порівнянні зі звичайним методом Ньютона.

Пояснення до програмної реалізації

Програмна реалізація складається з п’яти файлів, опис яких наведено нижче, а вихідний код – в розділі «Додатки».

Coursework.cpp – файл головної програми. Тут задано дві задачі (2 і 3 порядків) матрицею функцій F та матрицею, яка задає якобіан(в коді її позначено **W**), які містять λ-функції наступного вигляду:

```
[](const Vector2d &x) {return x[0] + x[1] - 3; }
```

Наведений приклад реалізує одне з рівнянь системи 2 порядку:

$$f_i(x_1, x_2) = x_1 + x_2 - 3$$

Функції для якобіану задаються аналогічно.

Також в цьому файлі визначені початкові наближення та точність для кожної з задач в наступному вигляді (приклад для системи 2 порядку):

```
double eps = 0.0001;//!< Точність, з якою проводяться обчислення
Vector2d x0;
x0 << 5.0, 2.0; //!< Початкове наближення
```

Newton.h – файл з описом класу, що реалізує метод Ньютона.

NewtonModified.h – файл з описом класу, що реалізує модифікований метод Ньютона.

util.h – файл з описом допоміжних функцій, які використовуються в обчисленнях.

Також програма використовує вільну бібліотеку Eigen, в якій описано відповідні класи для задання матриць, векторів, а також алгоритми факторизації та розв'язування систем лінійних алгебраїчних рівнянь.

Висновки

В курсовій роботі проілюстровано використання методу Ньютона (звичайного та модифікованого), високу швидкість його збіжності та зручність використання модифікованого методу на практиці завдяки можливості оптимізувати процес розв’язування СЛАР. Також реалізовано прикладну програму, яка дозволяє розв’язувати системи нелінійних рівнянь другого та третього порядків. За допомогою незначних модифікацій програму можна адаптувати під розв’язування рівнянь вищих порядків, проте при цьому не гарантується належне форматування виводу.

Список літератури

1. Бахвалов Н. С., Жидков Н. П., Кобельков Г. М. Чисельні методи: Навч. посібник. — Москва: Наука, 1987. — (ст. 323-329)
2. Самарський А.А., Гулін А.В. Чисельні методи. Москва: Наука, 1989. — (ст. 203-224)

Додатки

Coursework.cpp

```
// Coursework.cpp : Defines the entry point for the console application.
//

// Vanilla includes:
#include "stdafx.h"
#include <iostream>
#include <chrono>

// In-project includes
#include "util.h"
#include "Newton.h"
#include "NewtonModified.h"
#include <iomanip>

// Namespaces
using namespace std;
using namespace chrono;

//static IOFormat CleanFmt(StreamPrecision, 0, ", ", ", ", "", "", "(", ")");

// Main function
int main()
{
	system("@chcp 65001 > NUL");
	setlocale(LC_ALL, "Ukrainian");
	{
		Matrix<dfun2, 2, 1> F; //!< Матриця функцій, що задають систему рівнянь(у вигляді f(x)=0)
		cout << "#-------------------------------------------------------------------------------------------------#" << endl;
		wcout << L"| Задана система рівнянь №1 |" << endl;
		cout << "#-------------------------------------------------------------------------------------------------#" << endl;
		wcout << L"| x_1 + x_2 - 3 = 0 |" << endl;
		wcout << L"| x_1^2 + x_2^2 - 9 = 0 |" << endl;
		cout << "#-------------------------------------------------------------------------------------------------#" << endl;
		F << [](const Vector2d &x) {return x[0] + x[1] - 3; },
			[](const Vector2d &x) {return x[0] * x[0] + x[1] * x[1] - 9; };
		Matrix<dfun2, 2, 2> W; //!< Якобіан цієї системи
```

```
                                        /**
                                        * 1,    1
                                        * 2x_1, 2x_2
                                        **/
          W << [](const Vector2d &x) {return 1; }, [](const
Vector2d &x) {return 1; },
               [](const Vector2d &x) {return 2 * x[0]; }, [](const
Vector2d &x) {return 2 * x[1]; };
          double eps = 0.000000001;//!< Точність, з якою
проводяться обчислення
          Vector2d x0;
          x0 << 5.0, 2.0; //!< Початкове наближення
          wcout << L"| Точність обчислення:  " << setw(10) << eps
<< "                                                              |"
<< endl;
          stringstream ss;
          ss << x0.format(CleanFmt);
          string s = ss.str();
          s.resize(29);
          wcout << L"| Початкове наближення:       "; cout << s;
wcout << L"                                          |" << endl;
          cout << "#-----------------------------------------------
-------------------------------------------------#" << endl;
          cout << endl << endl;
          Newton<2> N;
          NewtonModified<2> N1;
          for (Newton<2>* n : { &N, static_cast<Newton<2>*>(&N1)
})
          {
               n->F = F;
               n->W = W;
               n->eps = eps;
               n->x0 = x0;
               n->debug = false;
               high_resolution_clock::time_point       t1       =
high_resolution_clock::now();
               n->main();
               high_resolution_clock::time_point       t2       =
high_resolution_clock::now();
               auto duration = duration_cast<microseconds>(t2 -
t1).count();
               wcout << L"| Час виконання: " << setw(7) << duration
<<                    L"                              мікросекунд
|" << endl;
               cout << "#-----------------------------------------
-----------------------------------------------------#"    <<
endl;
               cout << endl << endl;
          }
     }
     {
```

```
        Matrix<dfun3, 3, 1> F; //!< Матриця функцій, що задають
систему рівнянь(у вигляді f(x)=0)
        cout << "#-----------------------------------------
-----------------------------------------------#" << endl;
        wcout   <<   L"|   Задана   система   рівнянь       №2
|" << endl;
        cout << "#-----------------------------------------
-----------------------------------------------#" << endl;
        wcout  <<  L"|   x_1^2  +   x_2^2  +   x_3^2  -  1  =  0
|" << endl;
        wcout  <<  L"|  2x_1^2  +   x_2^2  -  4x_3           =  0
|" << endl;
        wcout  <<  L"|  3x_1^2  -  4x_2     +   x_3^2        =  0
|" << endl;
        cout << "#-----------------------------------------
-----------------------------------------------#" << endl;
        F << [](const Vector3d &x) {return x[0] * x[0] + x[1] *
x[1] + x[2] * x[2] - 1; },
             [](const Vector3d &x) {return 2 * x[0] * x[0] +
x[1] * x[1] - 4 * x[2]; },
             [](const Vector3d &x) {return 3 * x[0] * x[0] - 4
* x[1] + x[2] * x[2]; };
        Matrix<dfun3, 3, 3> W; //!< Якобіан цієї системи
                               /**
                               * 2x_1,     2x_2,    2x_3
                               * 4x_1,     2x_2,     -4
                               *    6x_1,  -4,       2x_3
                               **/
        W << [](const Vector3d &x) {return 2 * x[0]; }, [](const
Vector3d &x) {return 2 * x[1]; }, [](const Vector3d &x) {return 2
* x[2]; },
             [](const Vector3d &x) {return 4 * x[0]; }, [](const
Vector3d &x) {return 2 * x[1]; }, [](const Vector3d &x) {return -
4; },
             [](const Vector3d &x) {return 6 * x[0]; }, [](const
Vector3d &x) {return -4; }, [](const Vector3d &x) {return 2 * x[2];
};
        double   eps   =   0.000000001;//!<   Точність,   з   якою
проводяться обчислення
        Vector3d x0;
        x0 << 0.5, 0.5, 0.5; //!< Початкове наближення
        wcout << L"| Точність обчислення:  " << setw(10) << eps
<< "                                                    |"
<< endl;
        stringstream ss;
        ss << x0.format(CleanFmt);
        string s = ss.str();
        s.resize(29);
        wcout << L"| Початкове наближення:       "; cout << s;
wcout << L"                                    |" << endl;
```

```
          cout << "#-----------------------------------------
----------------------------------------------#" << endl;
          cout << endl << endl;
          Newton<3> N;
          NewtonModified<3> N1;
          for (Newton<3>* n : { &N, static_cast<Newton<3>*>(&N1)
})
          {
               n->F = F;
               n->W = W;
               n->eps = eps;
               n->x0 = x0;
               n->debug = false;
               high_resolution_clock::time_point       t1       =
high_resolution_clock::now();
               n->main();
               high_resolution_clock::time_point       t2       =
high_resolution_clock::now();
               auto duration = duration_cast<microseconds>(t2 -
t1).count();
               wcout << L"| Час виконання: " << setw(7) << duration
<<                    L"                              мікросекунд
|" << endl;
               cout << "#-------------------------------------
-----------------------------------------------------#"    <<
endl;
               cout << endl << endl;
          }
     }
     system("pause");
    return 0;
}
```

Newton.h

```
#pragma once
#include "util.h"

#include <ostream>
#include <iostream>
#include <iomanip>

using namespace std;

static IOFormat CleanFmt(StreamPrecision, 0, ", ", ", ", ", ", "", "",
"(", ")");

template<int N>
class Newton
{
public:
     Newton() : debug(false), eps(0.001)
```

```
	{
	}
	virtual void main() const
	{
		int k = 0;
		Matrix<double, N, 1> x1;
		Matrix<double, N, 1> x0 = this->x0;
		//cout << x0 << endl;
		Matrix<double, N, 1> F1 = -apply<N, 1>(F, x0);
		if (debug) {
			cout << F1 << endl;
			cout << endl;
		}
		Matrix<double, N, N> W1 = apply<N, N>(W, x0);
		if (debug) {
			cout << W1 << endl;
			cout << endl;
			cout << W1.fullPivLu().solve(F1) << endl << endl;
		}
		x1 = x0 + W1.fullPivLu().solve(F1);//x0 + deltaX
		cout << "#-----------------------------------------------
-----------------------------------------------#" << endl;
		wcout	<<	L"|	Метод	Ньютона
|" << endl;
		cout << "#------------#-------------------------------
----#-----------------------------------------------#" << endl;
		wcout << L"| № Ітерації | Розв'язок (X)                  |
Нев'язка(Z)                                  |" << endl;
		cout << "#------------#-------------------------------
----#-----------------------------------------------#" << endl;
		//wcout << L"# Ітерація " << k << ":\t\t\t#" << endl;
		stringstream ss;
		ss << x1.format(CleanFmt);
		string s = ss.str();
		ss.str("");
		ss.clear();
		ss << (apply<N, 1>(F, x1)).format(CleanFmt);
		string zs = ss.str();
		s.resize(34);
		zs.resize(45);
		cout << "|	0 | " << s << " | " << zs << " |"
<< endl;
		if (debug)
		{
			cout << delta<N>(x1, x0) << endl;
		}
		Matrix<double, N, 1> zero;
		for (int i = 0; i < N; i++)zero[i] = 0;
		//cout << endl;
		while (delta<N>(x1, x0) >= eps || delta<N>(apply<N,
1>(this->F, x1), zero) >= eps)
```

```
          {
               //cout << delta<N>(apply<N, 1>(this->F, x1), zero)
<< endl;
               k++;
               //wcout << L"Ітерація " << k << ":" << endl;

               x0 = x1;
               F1 = -apply<N, 1>(F, x0);
               if (debug) {
                    cout << F1 << endl;
                    cout << endl;
               }
               W1 = apply<N, N>(W, x0);
               if (debug) {
                    cout << W1 << endl;
                    cout << endl;
                    cout << W1.fullPivLu().solve(F1) << endl <<
endl;
               }
               x1 = x0 + W1.fullPivLu().solve(F1);//x0 + deltaX
               if (debug) {
                    double relative_error = (W1*x1 - F1).norm() /
F1.norm(); // norm() is L2 norm
                    wcout   <<   L"Відносна   похибка:   "   <<
relative_error << endl;
               }
               ss.str("");
               ss.clear();
               ss << x1.format(CleanFmt);
               s = ss.str();
               ss.str("");
               ss.clear();
               ss << (apply<N, 1>(F, x1)).format(CleanFmt);
               zs = ss.str();
               s.resize(34);
               zs.resize(45);
               cout << "| " << setw(10) << k << " | " << s << " |
" << zs << " |" << endl;
               //cout << x1.format(CleanFmt) << endl;
               if (debug) {
                    cout << delta<N>(x1, x0) << endl << endl;
               }
               //cout << endl;
          }
          cout << "#------------#-------------------------------
----#-------------------------------------------#" << endl;
          wcout << L"| Кінцевий результат: ";
          cout << s << "                                          |"
<< endl;
          cout << "#-------------------------------------------
-------------------------------------------------#" << endl;
```

```
            //cout << x1.format(CleanFmt) << endl;
        }

        virtual ~Newton()
        {
        }

        Matrix<function<double(const Matrix<double, N, 1>&)>, N, 1>
F;
        Matrix<function<double(const Matrix<double, N, 1>&)>, N, N>
W;
        Matrix<double, N, 1> x0;
        bool debug;
        double eps;
};
```

NewtonModified.h

```
#pragma once
#include "Newton.h"
#include <iostream>
#include <iomanip>

using namespace std;


template<int N>
class NewtonModified :
        public Newton<N>
{
public:
        NewtonModified(): debug(false), eps(0.001)
        {
        }

        virtual void main() const override
        {
                int k = 0;
                Matrix<double, N, 1> x1;
                Matrix<double, N, 1> x0 = this->x0;
                //cout << x0 << endl;
                Matrix<double, N, 1> F1 = -apply<N, 1>(this->F, x0);
                if (debug) {
                        cout << F1 << endl;
                        cout << endl;
                }
                Matrix<double, N, N> W1 = apply<N, N>(this->W, x0);
                auto W_lu = W1.fullPivLu();
                if (debug) {
                        cout << W1 << endl;
                        cout << endl;
                        cout << W_lu.solve(F1) << endl << endl;
```

```
          }
          x1 = x0 + W1.fullPivLu().solve(F1);//x0 + deltaX
          cout << "#-----------------------------------------
----------------------------------------------#" << endl;
          wcout    <<    L"|    Модифікований    метод    Ньютона
|" << endl;
          cout << "#------------#-------------------------------
----#----------------------------------------------#" << endl;
          wcout << L"| № Ітерації | Розв'язок (X)                 |
Нев'язка(Z)                                    |" << endl;
          cout << "#------------#-------------------------------
----#----------------------------------------------#" << endl;
          //wcout << L"# Ітерація " << k << ":\t\t\t#" << endl;
          stringstream ss;
          ss << x1.format(CleanFmt);
          string s = ss.str();
          ss.str("");
          ss.clear();
          ss << (apply<N, 1>(this->F, x1)).format(CleanFmt);
          string zs = ss.str();
          s.resize(34);
          zs.resize(45);
          cout << "|           0 | " << s << " | " << zs << " |"
<< endl;
          if (debug)
          {
               cout << delta<N>(x1, x0) << endl;
          }
          Matrix<double, N, 1> zero;
          for (int i = 0; i < N; i++)zero[i] = 0;
          //cout << endl;
          while (delta<N>(x1, x0) >= eps || delta<N>(apply<N,
1>(this->F, x1),zero ) >= eps)
          {
               /*cout << delta<N>(apply<N, 1>(this->F, x1), zero)
<< endl;*/
               k++;
               //wcout << L"Ітерація " << k << ":" << endl;
               x0 = x1;
               F1 = -apply<N, 1>(this->F, x0);
               if (debug) {
                    cout << F1 << endl;
                    cout << endl;
               }
               if (debug) {
                    cout << W1 << endl;
                    cout << endl;
                    cout << W_lu.solve(F1) << endl << endl;
               }
               x1 = x0 + W_lu.solve(F1);//x0 + deltaX
               if (debug) {
```

```
                    double relative_error = (W1*x1 - F1).norm() /
F1.norm(); // norm() is L2 norm
                    wcout   <<   L"Відносна   похибка:   "   <<
relative_error << endl;
               }
               ss.str("");
               ss.clear();
               ss << x1.format(CleanFmt);
               s = ss.str();
               ss.str("");
               ss.clear();
               ss << (apply<N, 1>(this->F, x1)).format(CleanFmt);
               zs = ss.str();
               s.resize(34);
               zs.resize(45);
               cout << "| " << setw(10) << k <<" | " << s << " |
" << zs << " |" << endl;
               if (debug) {
                    cout << delta<N>(x1, x0) << endl << endl;
               }
               //cout << endl;
          }
          cout << "#------------#------------------------------
----#-----------------------------------------------#" << endl;
          wcout << L"| Кінцевий результат: ";
          cout << s << "                                          |"
<< endl;
          cout << "#------------------------------------------
-----------------------------------------------#" << endl;
     }

     ~NewtonModified()
     {
     }

     //Matrix<function<double(const Matrix<double, N, 1>&)>, N, 1>
F;
     //Matrix<function<double(const Matrix<double, N, 1>&)>, N, N>
W;
     //Matrix<double, N, 1> x0;
     bool debug;
     double eps;
};
```

util.h

```
#pragma once

// Eigen or boost includes:
#include <Eigen/Core>
#include <Eigen/Dense>
```

```
// Namespaces
using namespace Eigen;

// Type definitions
typedef std::function<double(const Vector2d&)> dfun2;
typedef std::function<double(const Vector3d&)> dfun3;


// Function definitions
template <int n, int k>
Matrix<double,            n,            k>            apply(const
Matrix<std::function<double(const Matrix<double, n, 1>&)>, n, k>
&F, const Matrix<double, n, 1> &x)
{
     Matrix<double, n, k> ret;
     for (int i = 0; i < n; i++)
     {
          for (int j = 0; j < k; j++)
          {
               ret(i,j) = F(i,j)(x);
          }
     }
     return ret;
}

template <int n>
double delta(const Matrix<double, n, 1> &x1, const Matrix<double,
n, 1> &x0)
{
     double maxdiff = 0;
     for (int i = 0; i < n; i++)
     {
          double delta = abs(x1[i] - x0[i]);
          if (delta > maxdiff) maxdiff = delta;
     }
     return maxdiff;
}
```