Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра Систем Штучного Інтелекту

Звіт

до розрахунково-графічної роботи

з дисципліни

Чисельні методи

на тему:

Варіант №24 (Завдання для варіанту №8)

Виконав: студент КН-217

Ратушняк Денис

Прийняла: доцент каф. СШІ

Мочурад Л. І.

Львів – 2022

Мета роботи:

Реалізація чисельних методів з використанням системи автоматизації математичних та науково-технічних розрахунків.

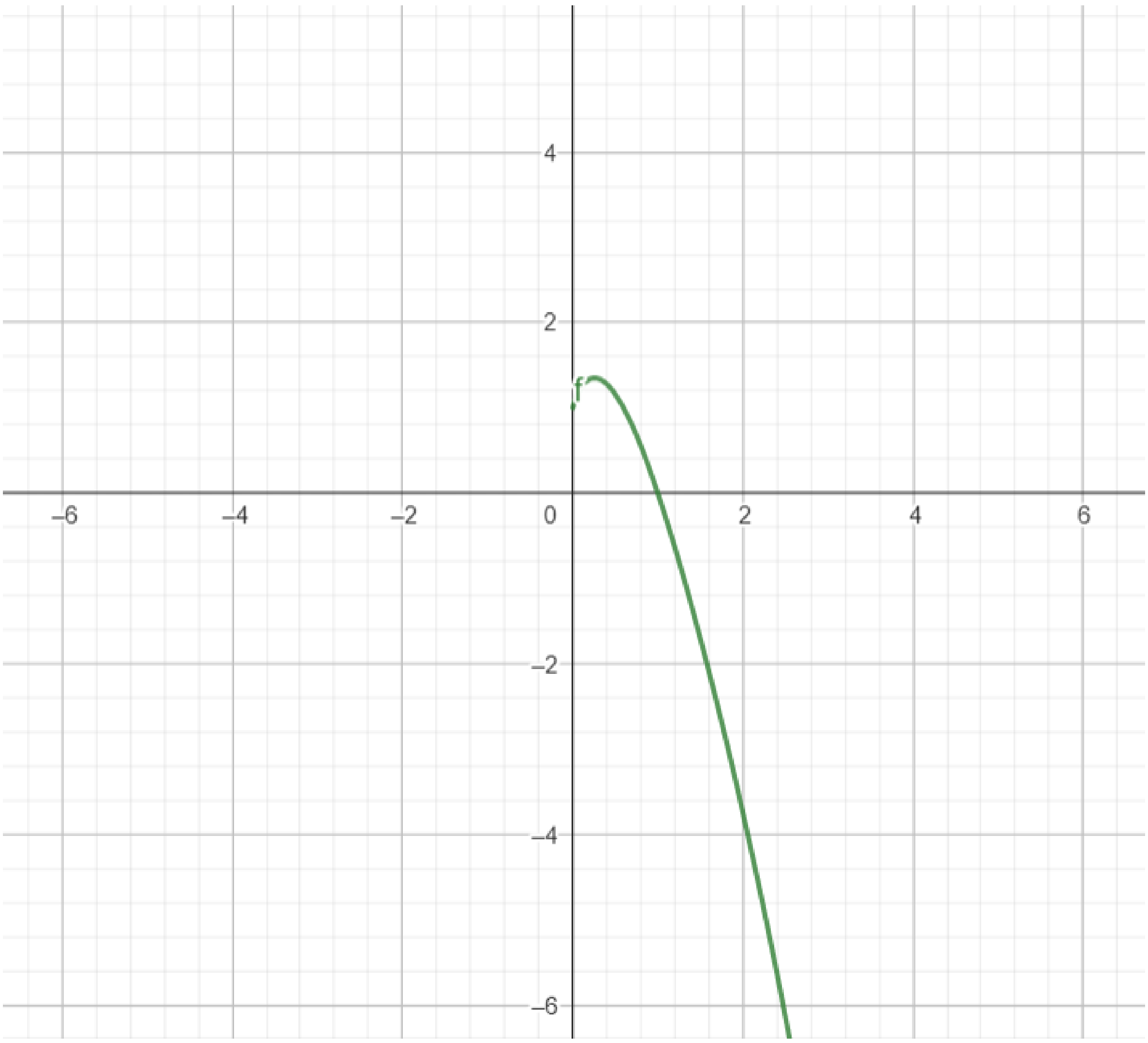
Завдання №1.

Завдання 1. Розв'язати функціональне рівняння двома заданими методами (табл. 2). $\varepsilon = 10^{-5}$. Порівняти результати, отримані в кожному з методів. Пояснити (у відповіді), яку роль грає виконання попереднього етапу в заданих методах – етапу відокремлення коренів.

8	дотичних	хорд	$x^{-x} - x^2 = 0$	[-10, -6]

Функція x ^ (-x) невизначена на проміжку (-∞, 0) тому інтервал розв'язання візьмемо як [0.0001, 10].

Графік цієї функції за допомогою онлайн сервісу geogebra:



Код розв’язку завдання:

```
#include <bits/stdc++.h>
using namespace std;

typedef long double ld;
typedef long long ll;
typedef vector< vector<ld > > matrix;

ld eps;
ld a,b,total_seg;

ld f(ld x)
{
    return pow(x, -x) - x * x;
}

ld deriv(ld x, ld eps)
{
    ld h = 0.5;
    ld d0 = 0;
    ld d = (f(x + h) - f(x - h)) / (2 * h);
    while(fabs(d - d0) > eps)
    {
        d0 = d;
        h *= 0.5;
        d = (f(x + h) - f(x - h)) / (2 * h);
    }
    return d;
}

ld secondderiv(ld x, ld eps)
{
    ld h = 0.5;
    ld d0 = 0;
    ld d = (f(x + h) - 2 * f(x) + f(x - h)) / (h * h);
    while(fabs(d - d0) > eps)
    {
        d0 = d;
        h *= 0.5;
        d = (f(x + h) - 2 * f(x) + f(x - h)) / (h * h);
    }
    return d;
}


void chords(ld a, ld b)
{
    ll steps = 0;
      ld x = 0;

      if(deriv(a, eps) * secondderiv(b, eps) < 0) swap(a, b);

      while(1)
      {
                x = a - (f(a) * (b - a)) / (f(b) - f(a));
                steps++;
                if (abs(a - x) < eps)
                          break;
                a = x;
      }
```

```
        cout << fixed << setprecision(10) << "CHORDS METHOD ->   x = " << x << " Iterations: " << steps << "\n";
        return;
}


void tangents(ld a, ld b){

      ll steps = 0;
      ld x;
      if(deriv(a, eps) * secondderiv(a, eps) > 0) x = a;
      else x= b;
      do {
            x = x - f(x) / deriv(x, eps);
            steps += 1;
      }
      while(fabs(f(x)) >= eps);

      cout << fixed << setprecision(10) << "TANGENTS METHOD ->   x = " << x << " Iterations: " << steps << "\n";
}

bool input()
{
      cin >> a >> b >> total_seg;
      eps = 1e-5;
      if(total_seg <= 0)
      {
            cout   << "total segments must be greater than 0\n";
            return 0;
      }
      if(a > b)
      {
            cout   << "a cannot be greater than b, try again\n";
            return 0;
      }
      if(a <= 0)
      {
            cout   << "a cannot be less or equal than 0 due to x^(-x), try again \n";
            return 0;
      }
      return 1;
}

int main()
{
      while(!input());

      ld len = (b - a) / total_seg;
      ld a1 = a;
      ld b1 = a1 + len;
ll total = 0;
      for(ll k = 0; k < total_seg; k++)
      {
            if(f(a1) * f(b1) < 0) {
                  chords(a1, b1);
                  tangents(a1, b1);
total++;
            }
            a1 += len;
            b1 += len;
      }
        if(total == 0) cout << "No roots were found\n";
```

```
    return 0;
}
```

Хід роботи:

Етап відокремлення коренів відіграє дуже велику роль в розв'язку рівнянь.
Обидва методи(хорд і дотичних) знаходять єдиний корінь рівняння на заданому проміжку, тож якщо на проміжку буде більше 1 кореня то алгоритм не знайде усі корені рівняння.
Протестуємо роботу програми на заданому проміжку [0.0001, 10] і перевіримо розв'язок за допомогою numpy і scipy.

```
0.0001 10 1000
CHORDS METHOD ->  x = 1.0000000000 Iterations: 2
TANGENTS METHOD ->  x = 1.0000000042 Iterations: 2
```

Перше число відповідає за ліву межу, друге за праву, третє за кількість відрізків, на яких шукати корінь.

```
import numpy as np
from scipy.optimize import brentq
f = lambda x: np.power(x, -x) - x * x
x = brentq(f, 0.0001, 10)
print("numpy + spicy -> x = ", x)
 numpy + spicy -> x =  0.9999999999999999
```

Можемо бачити, що розв'язки збігаються з заданою похибкою. $\varepsilon = 10^{-5}$

Протестуємо роботу програм на проміжку де немає кореня.

```
10 15 10000
No roots were found
```

```
Traceback (most recent call last):
  File "B:\pythonProject\task1.py", line 4, in <module>
    x = brentq(f, 10, 15)
  File "C:\Users\denis\.pyenv\pyenv-win\versions\3.10.7\
    r = _zeros._brentq(f, a, b, xtol, rtol, maxiter, arg
ValueError: f(a) and f(b) must have different signs
```

Завдання №2:

Завдання 2. Розв‘язати систему лінійних алгебраїчних рівнянь методом LU–розкладу та одним із ітераційних методів (за варіантом з табл. 3). Точність розв‘язків у ітераційному методі $\varepsilon = 10^{-2}$. Перевірити результати розв‘язку. Порівняти отримані розв‘язки.

8	Зейделя	$\begin{cases} 5x_1 + 2x_2 + 4x_3 = 9 \\ -x_1 + 3x_2 - 6x_3 = 8 \\ 7x_1 - x_2 - 3x_3 = 7 \end{cases}$	$x^0 = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$

Код розв’язку завдання:

```
#include <bits/stdc++.h>
using namespace std;

typedef long double ld;
typedef long long ll;
typedef vector< vector<ld > > matrix;

ld eps;
void print(vector<ld> &A, ll cnt = 5)
{
    cout << fixed << setprecision(cnt);
    for(int i = 0; i < A.size(); ++i) cout << A[i] << " ";
    cout << "\n";
    cout << fixed << setprecision(0);
}

void print(matrix &A, ll cnt=5)
{
    ll n = A.size();
    cout << fixed << setprecision(cnt);
    for(int i = 0; i < n; ++i)
    {
        for(int j = 0; j < A[i].size(); ++j) cout << A[i][j] << " ";
        cout << "\n";
    }
    cout << fixed << setprecision(0);
    cout << "\n";
}

matrix operator * (const matrix &A, const matrix &B)
{
    ll n = A.size();
    matrix res(n, vector<ld>(n, 0.0));
    for(int k = 0; k < n; ++k)
        for(int i = 0; i < n; ++i)
            for(int j = 0; j < n; ++j)
                res[i][j] += A[i][k] * B[k][j];
    return res;
}

ld euclidnorm(vector<ld> &guess, vector<ld> &ans){
    ld norm = 0;
    for(int i = 0; i < guess.size(); ++i)
        norm += (guess[i] - ans[i]) * (guess[i] - ans[i]);
```

```
    return sqrt(norm);
}


void seidel(matrix &A, vector<ld> &B)
{
    ll n = A.size();
    vector<ld> Y(n, 0);
    vector<ld> x(n, 0);
    x[0] = 1;
    x[1] = 1;
    x[2] = 1;
    vector<ld> y(n, 0);
    vector<ld> ybefor(n, 0);
    ll steps = 0;

    while(1)
    {
        for(int i = 0; i < n; ++i) ybefor[i] = y[i];
        for(int i = 0; i < n; i++)
        {
            y[i] = (B[i] / A[i][i]);
            for(int j = 0; j < n; j++)
            {
                if(j == i) continue;
                y[i] = y[i] - ((A[i][j] / A[i][i]) * x[j]);
                x[i] = y[i];
            }

        }

        ld norm = euclidnorm(y, ybefor);
        steps ++;

        if(steps % 1000000 <= 2) {
            cout << "Step: " << steps << setprecision(5) << " norm = "<< norm << " current answer: \n";
            print(y);
        }
        //cout << "\n";

        if(norm < eps) break;
    }
    cout << "SEIDEL METHOD ->    x = "; print(y);
    cout << "Iterations: " << steps << "\n";
}

bool is_determinant_zero(matrix A)
{
    ll n = A.size();
    matrix AE;
    AE.resize(n);

    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++ j)
            AE[i].emplace_back(A[i][j]);

    for(int k = 0; k < n; ++k)
    {
        int row_max_el = k;

        /// finding maximum element
        for(int i = k + 1; i < n; ++i)
```

```
            if(fabs(AE[i][k]) > fabs(AE[row_max_el][k])) row_max_el = i;

        /// if maximum element goes to 0 -> inverse matrix does not exist
        if(fabs(AE[row_max_el][k]) < eps) return true;


        /// change of 2 rows for optimization of division by the maximum element
        if(k != row_max_el)
            for(int j = k; j < n; ++j) swap(AE[k][j], AE[row_max_el][j]);


        /// forming zeros under main diagonal
        for(int i = k + 1; i < n; ++i)
        {
            ld mik = -AE[i][k]/AE[k][k];

            AE[i][k] = 0;
            for(int j = k + 1; j < n; ++j) AE[i][j] += mik * AE[k][j];
        }
    }

    for(int i = 0; i < n; ++i)
    {
        if(fabs(AE[i][i]) < eps) return true;
    }
    return false;

}

bool check_valid(matrix &A, ll &num)
{
    matrix B;
    ll n = A.size();
    B.resize(1);
    B[0].resize(1);
    B[0][0] = A[0][0];
    vector<ld> add(1);
    num = 0;
    if(fabs(A[0][0]) < eps) return false;
    for(int i = 1; i < n; ++i)
    {
        for(int num = 0; num < i; ++num) B[num].emplace_back(A[num][i]);
        for(int j = 0; j < i; ++j) add[j] = A[i][j];
        add.push_back(A[i][i]);
        B.push_back(add);
        if(is_determinant_zero(B)) {
            num = i;
            return false;
        }
    }
    return true;
}

void decompose_LU(matrix &A, matrix &L, matrix &U)
{
    ll n = A.size();
    vector<ld> null(n, 0);
    L.resize(n, vector<ld>(n, 0));
    U.resize(n, vector<ld>(n, 0));
    for(int s = 0; s < n; s++)
    {
        for(int j = s; j < n; j++)
```

```
        {
            U[s][j] = A[s][j];
            for(int k = 0; k < s; k++) U[s][j] -= L[s][k] * U[k][j];
        }
        for(int i = s; i < n; i++)
        {
            L[i][s] = A[i][s];
            for(int k = 0; k < s; k++) L[i][s] -= (L[i][k] * U[k][s]);
            L[i][s] /= U[s][s];
        }
    }
}


void LY_eq_B(matrix &L, vector<ld> &Y, vector<ld> &B)
{
    ll n = L.size();
    for(int i = 0; i < n; ++ i)
    {
        ld h = 0;
        for(int j = 0; j < i; ++j)
        {
            h += L[i][j] * Y[j];
        }
        Y[i] = B[i] - h;
    }
}

void UX_eq_Y(matrix &U, vector<ld> &X, vector<ld> &Y)
{
    ll n = U.size();
    for(int i = n - 1; i >= 0; -- i)
    {
        ld h = 0;
        for(int j = n - 1; j > i; --j)
        {
            h += U[i][j] * X[j];
        }
        X[i] = (Y[i] - h)/U[i][i];
    }
}


void LU(matrix &A, vector<ld> &B)
{
    ll n = A.size();
    ll num;
    if(!check_valid(A, num))
    {
        cout << "Cannot do LU decomposition, because " << num << "th leading principal minors is equal to 0\n";
        return;
    }

    matrix L,U;
    decompose_LU(A, L, U);
    cout << "L:\n" ;print(L);
    cout << "U:\n" ;print(U);
    vector<ld> Y(n);
    vector<ld> X(n);
    LY_eq_B(L, Y, B);
    UX_eq_Y(U, X, Y);
    cout << "LU METHOD -> x = ";
```

```
	print(X);

}

bool check_DD(matrix &A){
	ll n = A.size();
	bool if_one = 0;
	for(int i = 0; i < n; ++ i){
		ld sum = 0;
		for(int j = 0; j < n; ++ j){
			sum += fabs(A[i][j]);
		}
		sum -= fabs(A[i][i]);
		if(fabs(A[i][i]) > sum) if_one = 1;
		if(fabs(A[i][i]) < sum) return false;
	}
	return if_one;
}

int main()
{
	eps = 1e-2;
	matrix A;
	vector<ld> B,X;
	A = {{5, 2, 4}, {-1, 3, -6}, {7, -1, -3}};
	B = {9, 8, 7};

	//A = {{10, 2, 2}, {1, 15, 1}, {1, 1, 23}};
	//B = {1, 1, 1};

	cout << "A:\n";print(A);
	cout << "B:\n";print(A);

	LU(A, B);
	if(check_DD(A)) seidel(A, B);
	else {
		cout << "Matrix isnt diagonally dominant so Seidel Method may not coverge" << "\n";
		return 0;
	}
	return 0;
}
```

Хід роботи:

Запустимо програму для заданого рівняння та перевіримо розв’язок через numpy:

```
A:
5.00000 2.00000 4.00000
-1.00000 3.00000 -6.00000
7.00000 -1.00000 -3.00000

B:
5.00000 2.00000 4.00000
-1.00000 3.00000 -6.00000
7.00000 -1.00000 -3.00000

L:
1.00000 0.00000 0.00000
-0.20000 1.00000 0.00000
1.40000 -1.11765 1.00000

U:
5.00000 2.00000 4.00000
0.00000 3.40000 -5.20000
0.00000 0.00000 -14.41176

LU METHOD -> x = 1.17143 2.31429 -0.37143
Matrix isnt diagonally dominant so Seidel Method may not coverge
```

```
a = np.array([[5, 2, 4],
              [-1, 3, -6],
              [7, -1, -3]])

b = np.array([9, 8, 7])
x = np.linalg.solve(a, b)
print(x)
```

```
 [ 1.17142857  2.31428571 -0.37142857]
```

Можемо бачити, що Метод LU розкладу працює правильно відповідно до рішення з numpy для заданої точності ε = 10 ^ (-2). Також задана матриця не є діагонально переважаючою, тому метод Зейделя не гарантовано збігається(так і відбулося в мому випадку, для заданої матриці метод не збігся).

Протестуємо додатково на іншій системі рівнянь:

```
A:
10.00000 2.00000 2.00000
1.00000 15.00000 1.00000
1.00000 1.00000 23.00000

B:
10.00000 2.00000 2.00000
1.00000 15.00000 1.00000
1.00000 1.00000 23.00000

L:
1.00000 0.00000 0.00000
0.10000 1.00000 0.00000
0.10000 0.05405 1.00000

U:
10.00000 2.00000 2.00000
0.00000 14.80000 0.80000
0.00000 0.00000 22.75676

LU METHOD -> x = 0.08076 0.05879 0.03741
Step: 1 norm = 0.30577 current answer:
-0.30000 0.02000 0.05565
Step: 2 norm = 0.38711 current answer:
0.08487 0.05730 0.03730
SEIDEL METHOD ->  x = 0.08108 0.05877 0.03740
Iterations: 3
```

```
a = np.array([[10, 2, 2],
              [1, 15, 1],
              [1, 1, 23]])

b = np.array([1, 1, 1])
x = np.linalg.solve(a, b)
print(x)
```

```
[0.0807601  0.0587886  0.03741093]
```

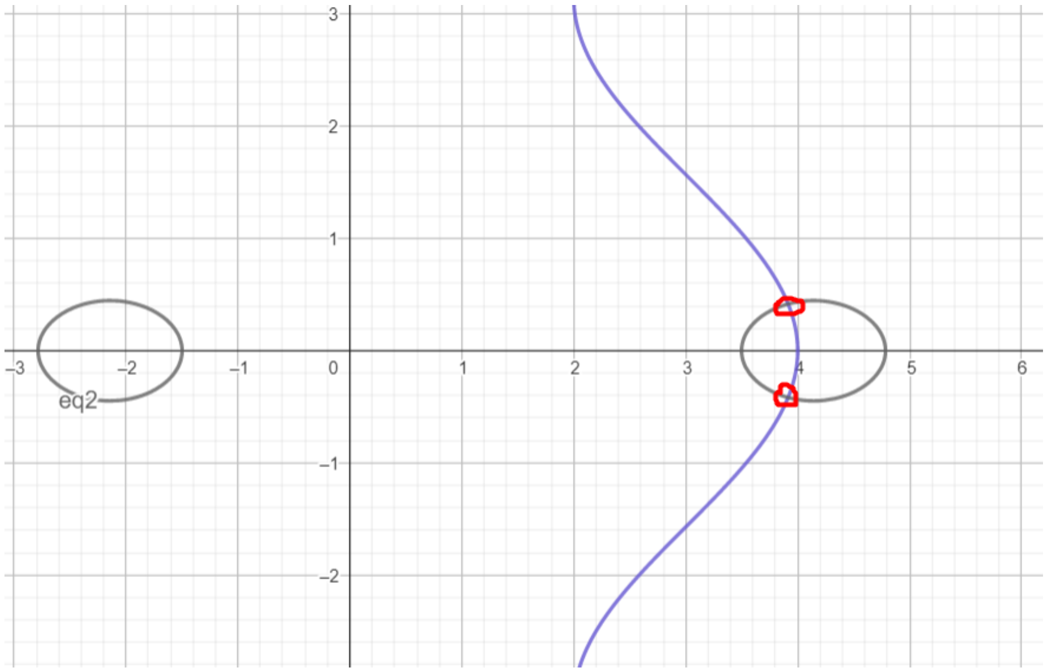
Бачимо, що метод LU розкладу та метод Зейделя працюють правильно відповідно до рішення з numpy для заданої точності ε = 10 ^ (-2).

Завдання №3

Завдання 3. Розв‘язати систему нелінійних рівнянь (СНР) вказаним за варіантом методом (табл. 3). Похибка результатів має бути не більше $\varepsilon = 10^{-2}$. Перевірити достовірність отриманих розв‘язків ($\boldsymbol{F(X_n) = 0}$).

8	Зейделя	$\begin{cases} \cos(x-1) + y^2 = -0.8 \\ x - \cos(y) = 3 \end{cases}$

Графік двох функцій за допомогою онлайн сервісу geogebra(червоним показано шукані розв’язки):



Код розв’язку завдання:

```
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
typedef long long ll;
typedef vector< vector<ld > > matrix;
ld eps;
int main()
{
    ld x,y;
    cout << "Input first guess x and y\n";
    cin >> x >> y;
    vector<ld> X,Y;
    X.push_back(x);
    Y.push_back(y);
    eps = 1e-2;
    ll steps = 0;
```

```
    while(1){
        ld oldx = X.back();
        ld oldy = Y.back();
        steps++;
        x = 3 + cos(oldy);
        y = sqrt(-0.8 - cos(oldx - 1));
        X.push_back(x);
        Y.push_back(y);
        if(steps > 2 && steps % 2 == 1 && fabs(x - X[X.size()-3]) < eps) break;
        if(steps > 2 && steps % 2 == 0 && fabs(y - Y[Y.size()-3]) < eps) break;
    }
    if(ll(X.size()) % 2 == 0){
        x = X.back();
        y = acos(x - 3);
    }
    else {
        y = Y.back();
        x = acos(-0.8 - y * y)+ 1;
    }
    cout << "SEIDEL METHOD TO SOLVE NONLINEAR SOE - > x1 = " << fixed << setprecision(5) << x << " y1 = " << y << "\n";
    cout << "x2 = " << fixed << setprecision(5) << x << " y2 = " << -y << "\n";
    cout << "First f(x1, y1) = 0 -> " << fixed << setprecision(5) << (cos(x - 1) + y * y + 0.8) << "\n";
    cout << "Second f(x1, y1) = 0 -> " << fixed << setprecision(5) << (x - cos(y) - 3) << "\n";
    cout << "First f(x2, y2) = 0 -> " << fixed << setprecision(5) << (cos(x - 1) + y * y + 0.8) << "\n";
    cout << "Second f(x2, y2) = 0 -> " << fixed << setprecision(5) << (x - cos(-y) - 3) << "\n";
    return 0;
}
```

Хід роботи:

Запустимо програму для заданої системи рівняннь та перевіримо розв’язок через numpy і scipy:

```
Input first guess x and y
0 0
SEIDEL METHOD TO SOLVE NONLINEAR SOE - > x1 = 3.91499 y1 = 0.41532
x2 = 3.91499 y2 = -0.41532
First f(x1, y1) = 0 -> -0.00194
Second f(x1, y1) = 0 -> 0.00000
First f(x2, y2) = 0 -> -0.00194
Second f(x2, y2) = 0 -> 0.00000
```

```
from scipy.optimize import fsolve
def eq(p):
    x, y = p
    return np.cos(x - 1) + y ** 2 + 0.8, x - np.cos(y) - 3


x, y = fsolve(eq, (1, 1))

print(x, y)
print(eq((x, y)))
 3.9141361504017196 -0.4174243703704107
 (0.0, 0.0)
```

Можемо бачити, що метод Зейделя для розв'язування системи нелінійних алгебраїчних рівнянь працює правильно відповідно до рішення з numpy + scipy для заданої точності ε = 10 ^ (-2). В цьому випадку рішення з numpy + scipy не знайшло другий корінь.

Завдання 4:

Завдання 4. Побудувати інтерполяційний поліном Лагранжа для функції, що задана таблично. Побудувати графік заданих точок та отриманого поліному Лагранжа.

8	***x***	3	4	5	6	7
	y	12	-1	5	8	3

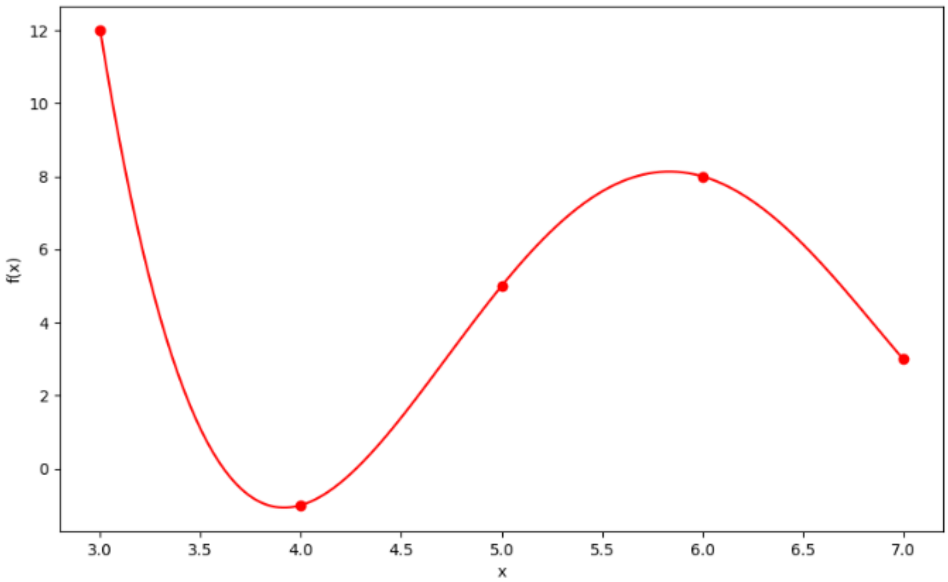
Код розв'язку завдання:

```
import numpy as np
import scipy
from matplotlib import pyplot as plt


def lag(x, arr_x, arr_y):
    res = 0
    n = len(arr_x)
    for i in range(n):
        cur = arr_y[i]
        for j in range(n):
            if i == j:
                continue
            cur *= (x - arr_x[j]) / (arr_x[i] - arr_x[j])

        res += cur
    return res


x_given = [3, 4, 5, 6, 7]
y_given = [12, -1, 5, 8, 3]

x = np.linspace(3, 7, 1000)
y_new = lag(x, x_given, y_given)

plt.figure(figsize=(10, 6))
plt.xlabel('x')
plt.ylabel('f(x)')

plt.plot(x, y_new, 'r')
plt.plot(x_given, y_given, 'ro')
plt.show()
```

Хід роботи:

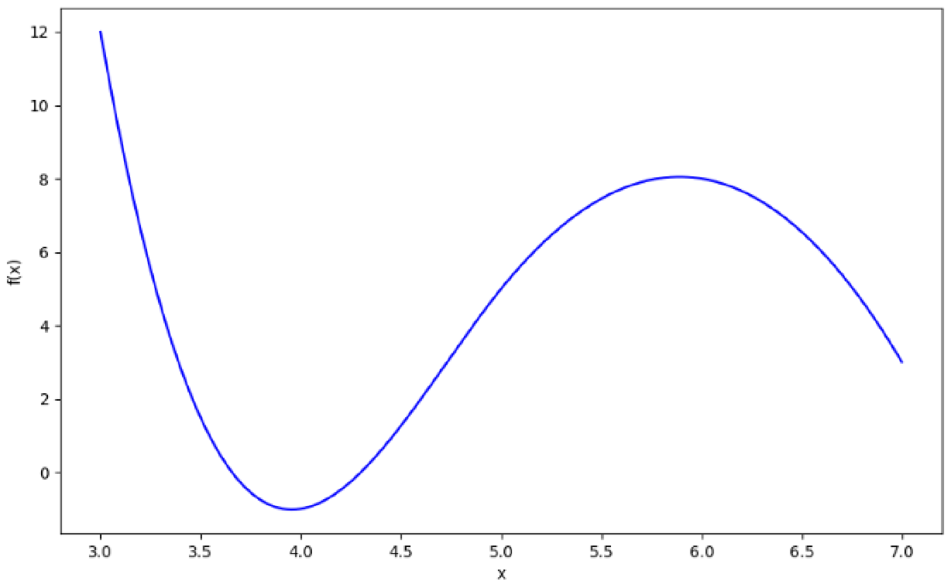
Запустимо програму для заданого набору точок та перевіримо розв'язок через numpy і scipy:

Графік отриманий методом:

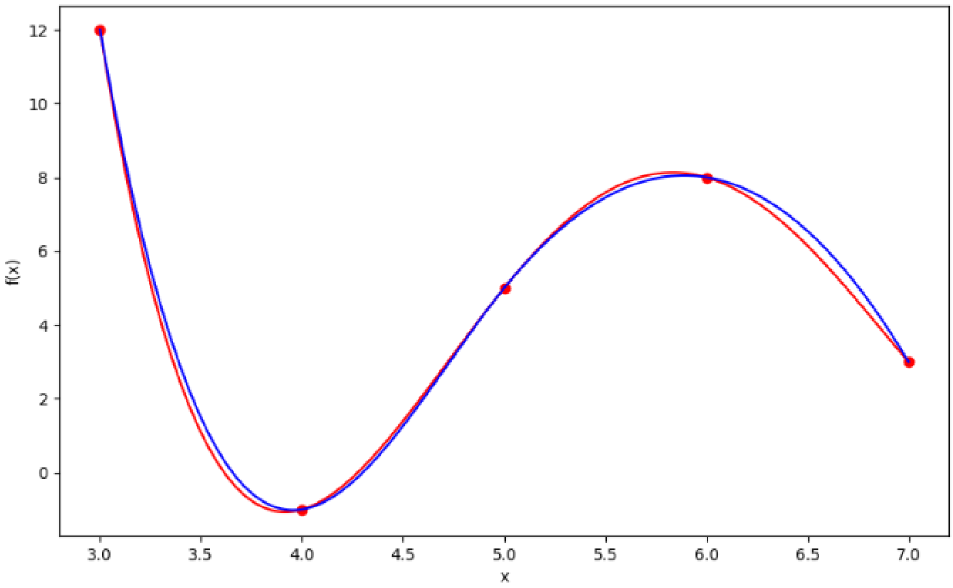


Графік отриманий numpy + spicy:

```
x_given = [3, 4, 5, 6, 7]
y_given = [12, -1, 5, 8, 3]

x = np.linspace(3, 7, 1000)

plt.figure(figsize=(10, 6))
plt.xlabel('x')
plt.ylabel('f(x)')

f = scipy.interpolate.interp1d(x_given, y_given, kind = 'cubic')
Y = f(x)
plt.plot(x, Y, 'b')
plt.show()
```



Поєднаємо 2 графіки щоб порівняти розв’язки:



Можемо бачити, що графіки майже ідентичні.

Висновок

Я закріпив вміння та навички роботи з такими чисельними методами: метод дотичних, хорд, LU-розклад, Зейделя, Зейделя для СНР, інтерполяції Лагранжа, та перевірив їх з використанням системи автоматизації математичних та науково-технічних розрахунків, і підтвердив коректність роботи цих методів.