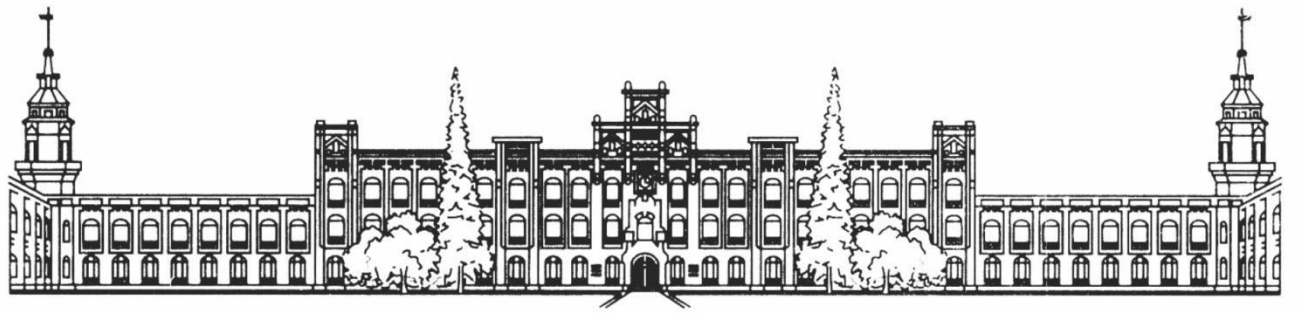


Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського» Інститут прикладного системного аналізу

Лабораторна робота №4

курсу «Чисельні методи 1»

з теми «Методи розв’язання нелінійних систем»

Варіант №9

Виконав студент 2 курсу групи КА-91
Косицький Вадим Вікторович
перевірила старший викладач
Хоменко Ольга Володимирівна

Київ-2021

Завдання 1

1. Розв'язати систему 1методом простих ітерацій. Для цього:
   - визначити початкове наближення, побудувавши графіки кривих системи;
   - перевірити достатні умови збіжності з детальним поясненням (задати область, в якій перевірити виконання умов збіжності, можна робити фото написаного і вставляти в звіт);
   - реалізувати метод простих ітерацій. Розв'язати систему з точністю $\varepsilon = 10^{-5}$.
2. Задати декілька інших початкових наближень (які не близькі до розв'язку) та з'ясувати як змінюється при цьому ітераційний процес.

Завдання 2

1. Розв'язати систему 2 методом Ньютона (або спрощеним методом Ньютона). Для цього

   - визначити початкове наближення, побудувавши графіки кривих системи;
   - реалізувати метод Ньютона (спрощений метод Ньютона). Якщо обрано метод Ньютона, за потреби можна використовувати функції linalg.solve та ін. Розв'язати систему з точністю $\varepsilon = 10^{-5}$.

2. Задати декілька інших початкових наближень (які не близькі до розв'язку) та з'ясувати як змінюється при цьому ітераційний процес.

№ 9. 1) $\begin{cases} \cos(x+0{,}5)-y=2; \\ \sin y-2x=1. \end{cases}$ 2) $\begin{cases} \operatorname{tg} xy=x^2; \\ 0{,}7x^2+2y^2=1. \end{cases}$

Текст програми:

Файл main.py (Для методу простих ітерацій)

```
from func import *

eps = 0.00001

#main
l = [float(i) for i in input("Enter x(0): ").split()]
x0 = np.asarray(l)
n, inputs, arguments = input_data()
Simple_iter(n, inputs, x0, arguments, eps)
```

Файл main.py (Для методу Н’ютона)

```
from func import *

eps = 0.00001

#main
l = [float(i) for i in input("Enter x(0): ").split()]
x0 = np.asarray(l)
n, inputs, arguments = input_data()
Newtone(n, inputs, x0, arguments, eps)
```

Файл func.py (Тут реалізовано усі потрібні алгоритми)

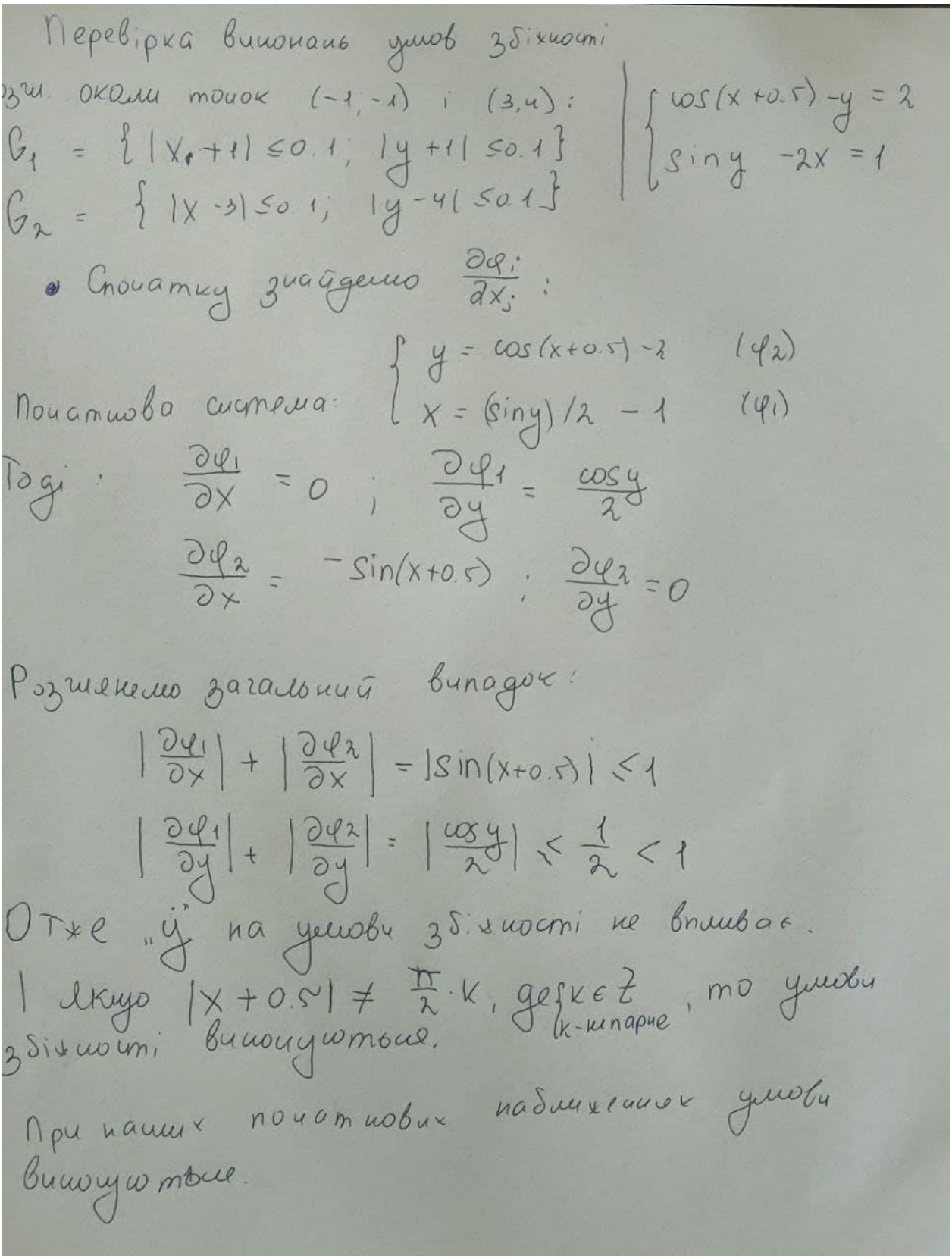
```
import numpy as np
import pandas as pd
import pathlib
from pathlib import Path
from math import *
from sympy import *

eps1 = 0.0001
def input_data():
    n = int(input("Enter number of equations: "))
    arguments = input("Enter arguments: ").split()
    print("Enter equations")
    inputs = [input() for i in range(n)]
    return n, inputs, arguments

def Solve_matrix(matrix, parametrs, arguments , n,m):
    L, L1= [], []
    for i in range(n):
        for j in range(m):
            L1.append(matrix[i][j].evalf(subs=dict(zip(arguments, parametrs))))
        L.append(L1.copy())
        L1.clear()
    M = np.array(L, dtype='float')
    return M

def Solve_vector(matrix, parametrs, arguments ,n):
    L =[]
    for i in range(n):
        L.append(matrix[i].evalf(subs=dict(zip(arguments, parametrs))))
    M = np.array(L.copy(), dtype='float')
    return M
```

```
def Newtone(n, inputs, x0,arguments, eps):
    print("Newtone method")
    solutions, deltas = [x0], [0]
    #Find F and W
    L, L1, K = [], [], []
    for i in range(n):
        for j in range(n):
             L1.append(sympify(inputs[i]).diff(arguments[j]))
        L.append(L1.copy())
        L1.clear()
        K.append(sympify(inputs[i]))
    W =np.array(L.copy())
    F = np.array(K.copy())
    L.clear(), K.clear()
    delta = np.linalg.solve(Solve_matrix(W, x0, arguments, n, n), -Solve_vector(F, x0,
arguments, n))
    solutions.append(( delta +x0).copy())
    deltas.append(np.linalg.norm(delta, ord = np.inf))
    while(deltas[-1] > eps):
        delta = np.linalg.solve(Solve_matrix(W, solutions[-1], arguments, n, n), -
Solve_vector(F, solutions[-1], arguments, n))
        solutions.append((delta + solutions[-1]).copy())
        deltas.append(np.linalg.norm(delta, ord=np.inf))
    Solution = np.c_[np.matrix(solutions), np.array(deltas)]
    col = arguments.copy()
    col.append('Δ')
    df = pd.DataFrame(Solution, columns=col)
    print(df)

    print("\nF(x*): ", Solve_vector(F,solutions[-1] , arguments, n))

def Simple_iter(n, inputs, x0, arguments, eps):
    print("Simple iteration method")
    solutions, deltas = [x0], [0]
    # Find F and W
    L, L1, K = [], [], []
    for i in range(n):
        for j in range(n):
            L1.append(sympify(inputs[i]).diff(arguments[j]))
        L.append(L1.copy())
        L1.clear()
        K.append(sympify(inputs[i]))
    W = np.array(L.copy())
    F = np.array(K.copy())
    L.clear(), K.clear()
    if (np.linalg.norm(Solve_matrix(W, solutions[-1], arguments, n, n), ord = np.inf) < 1):
print("\nVerification is done\n")
    else: print("\nVerification is not done\n")
    solutions.append(Solve_vector(F, solutions[-1], arguments, n))
    deltas.append(np.linalg.norm(solutions[-1]-solutions[-2], ord = np.inf))
    while(deltas[-1] > eps):
        solutions.append(Solve_vector(F, solutions[-1], arguments, n))
        deltas.append(np.linalg.norm(solutions[-1] - solutions[-2], ord=np.inf))
    Solution = np.c_[np.matrix(solutions), np.array(deltas)]
    col = arguments.copy()
    col.append('Δ')
    df = pd.DataFrame(Solution, columns=col)
    print(df)

    print("\nF(x*): ", Solve_vector(F, solutions[-1], arguments, n)-solutions[-1])
```

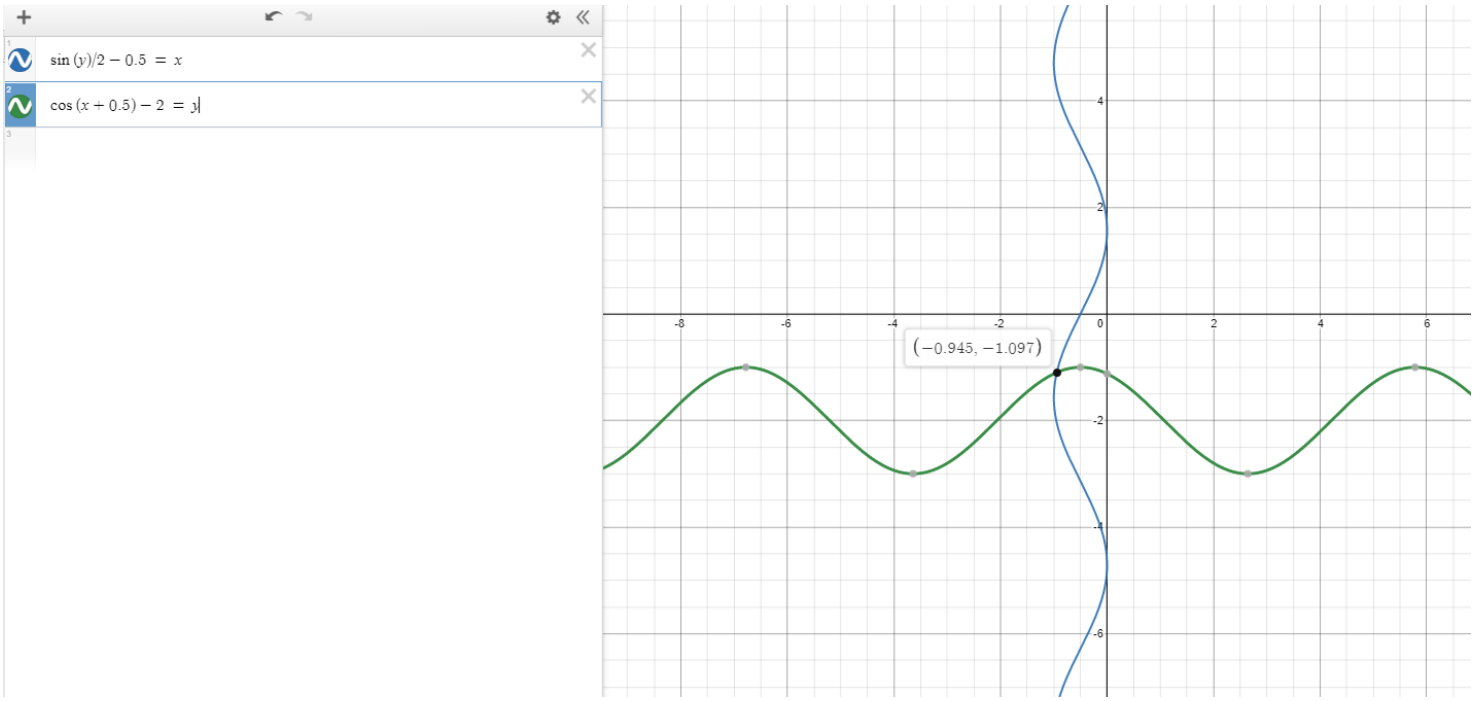
Результати роботи:

Розглянемо спочатку метод простих ітерацій.

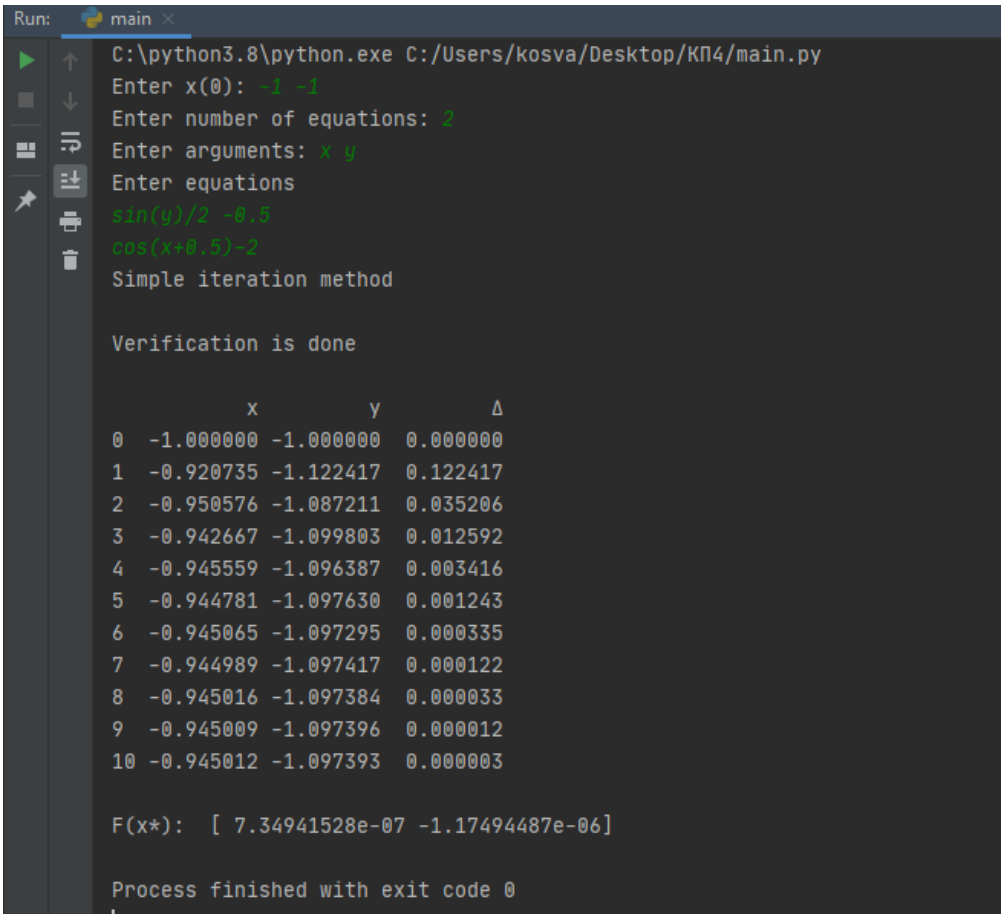
Для початку я наведу фотографію з перевіркою виконань умов збіжності



За допомогою алгоритмів Desmos, я побудував графіки даної системи, та знайшов приблизну точку перетину.



Тепер використовуючи таке наближення: (-1,-1), знайдемо розв’язок системи за допомогою моєї програми.



Зауваження про правила вводу: спочатку ми вводимо початкове наближення, потім кількість рівнянь в системі, потім вводимо змінні, які використовуємо в рівняннях, далі вводимо систему функцій φ, де порядок послідовності функцій збігається з порядком введених змінних.

Тобто маючи таку систему:

$$\begin{cases} \cos(x+0{,}5)-y=2; \\ \sin y-2x=1. \end{cases}$$

Я перетворюю її на еквівалентну виду:

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

Тобто в даному випадку на:

$$\begin{cases} x = \dfrac{\sin(y)}{2} - 0.5 \\ y = cos(x + 0.5) - 2 \end{cases}$$

І праві частини рівнянь, я і маю вводити до консолі, причому саме в порядку: x, а потім y, оскільки саме в такому порядку вводив змінні до консолі.

Тепер повернемося до нашого прикладу.

Бачимо, що після уведення даних, програма видає що верифікація виконуються (Verification is done) (Ця перевірка відрізняється від перевірки достатніх умов збіжності, вона налаштована на те, що підставляє точки наближення замість тих точок, де сума модулів максимальна і перевіряє чи ця сума менше за 1), після чого виводить таблицю з результатами роботи програми на кожній ітерації, допоки $\Delta = \max\left|x_i^{k+1} - x_i^k\right|$ не буде меншим за $\varepsilon = 0.00001$.

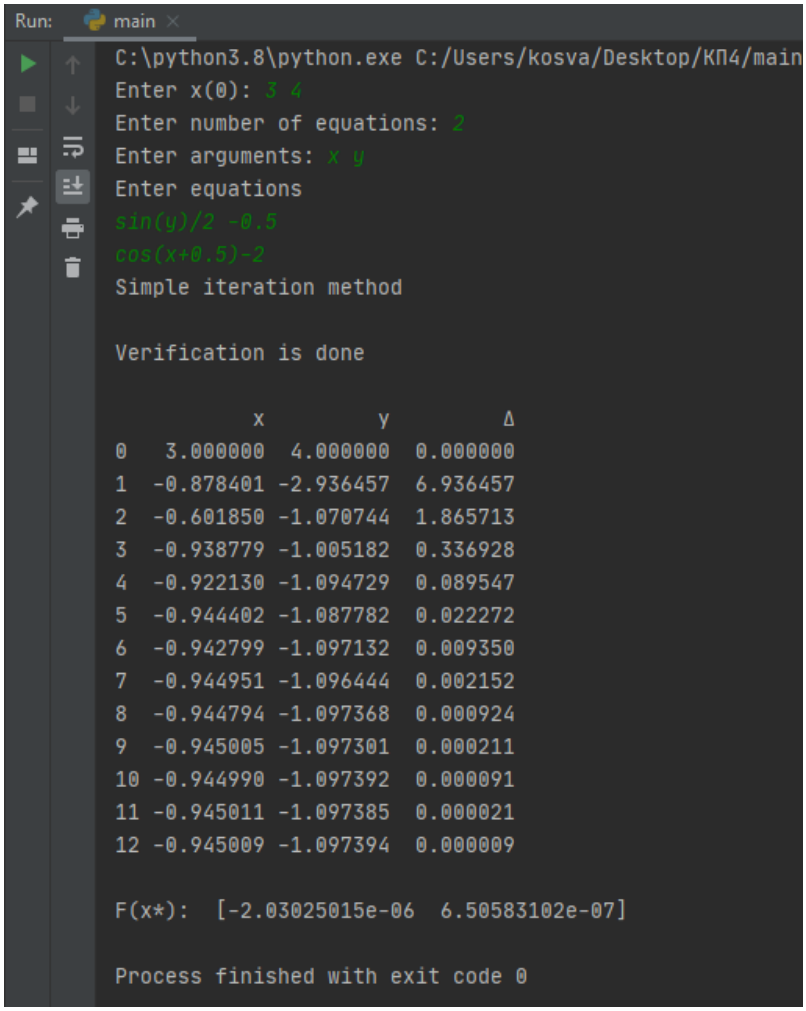
Достатні умови збіжності(на першій картинці) я перевіряв за формулою

$$max_{x \in G} max_i \sum_{j=1}^{n} \left| \frac{\partial \varphi_i(x)}{\partial x_j} \right| \leq q < 1$$

, де G – окіл початкового наближення.

Бачимо, що F(x*) містить в собі і справді дуже маленькі значення. Отже програма працює правильно.

Тепер спробуємо ввести інше початкове наближення: (3, 4)



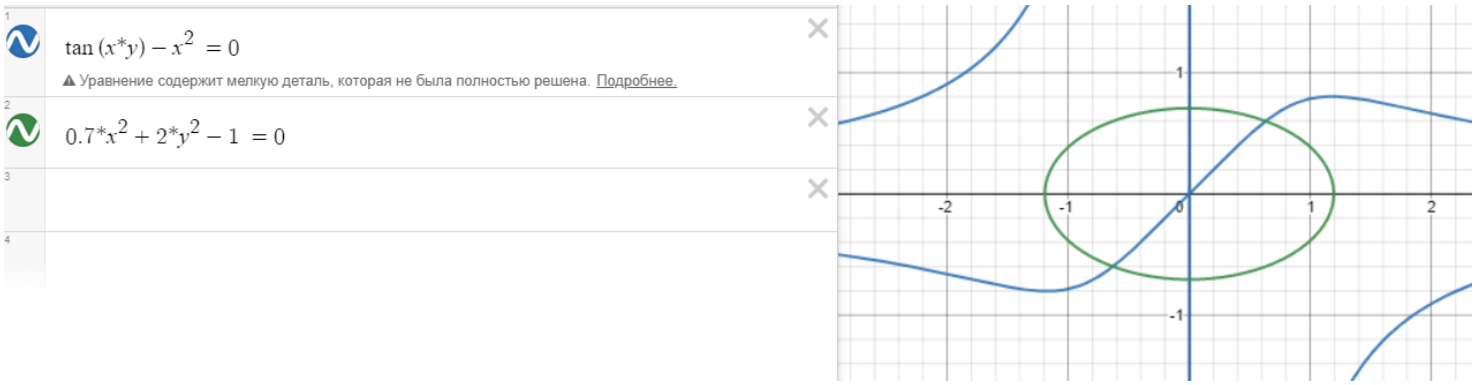
Бачимо, що навіть при віддаленому початковому наближені, верифікація (знову ж таки повторюсь, що перевірка не зовсім коректна, але дозволяє програмі робити зважені кроки) знову виконується, і програма все одно видає правильні результати.

Зауваження: причому виконання умов збіжності досить зрозуміле, оскільки фукнції sin та cos – обмежені 1, і якщо подивитися на похідні φ_i по x i y, то

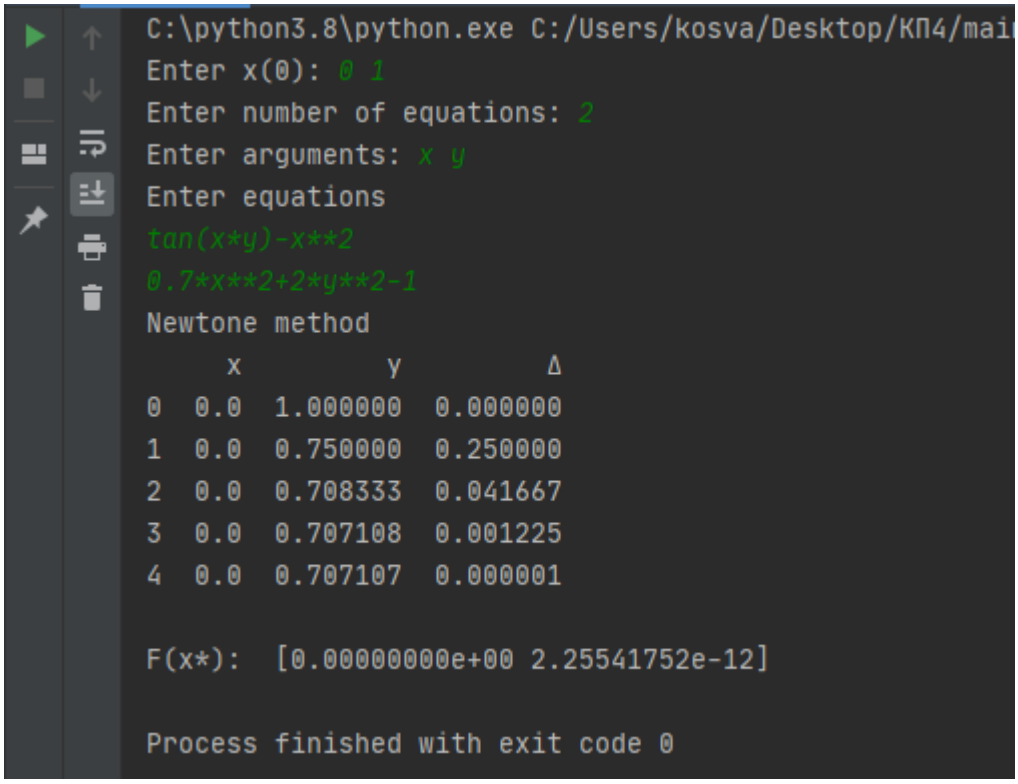
бачимо, що при підстановці будь якого початкового наближення, сума модулів цих функцій, завжди менше або рівне 1.

Тепер розглянемо приклади із застосуванням метода Н'ютона.

Знову скористаємось алгоритмом Desmos, для побудови потрібних графіків.



Тепер подивимося на роботу програми при початковому наближені: (0, 1)

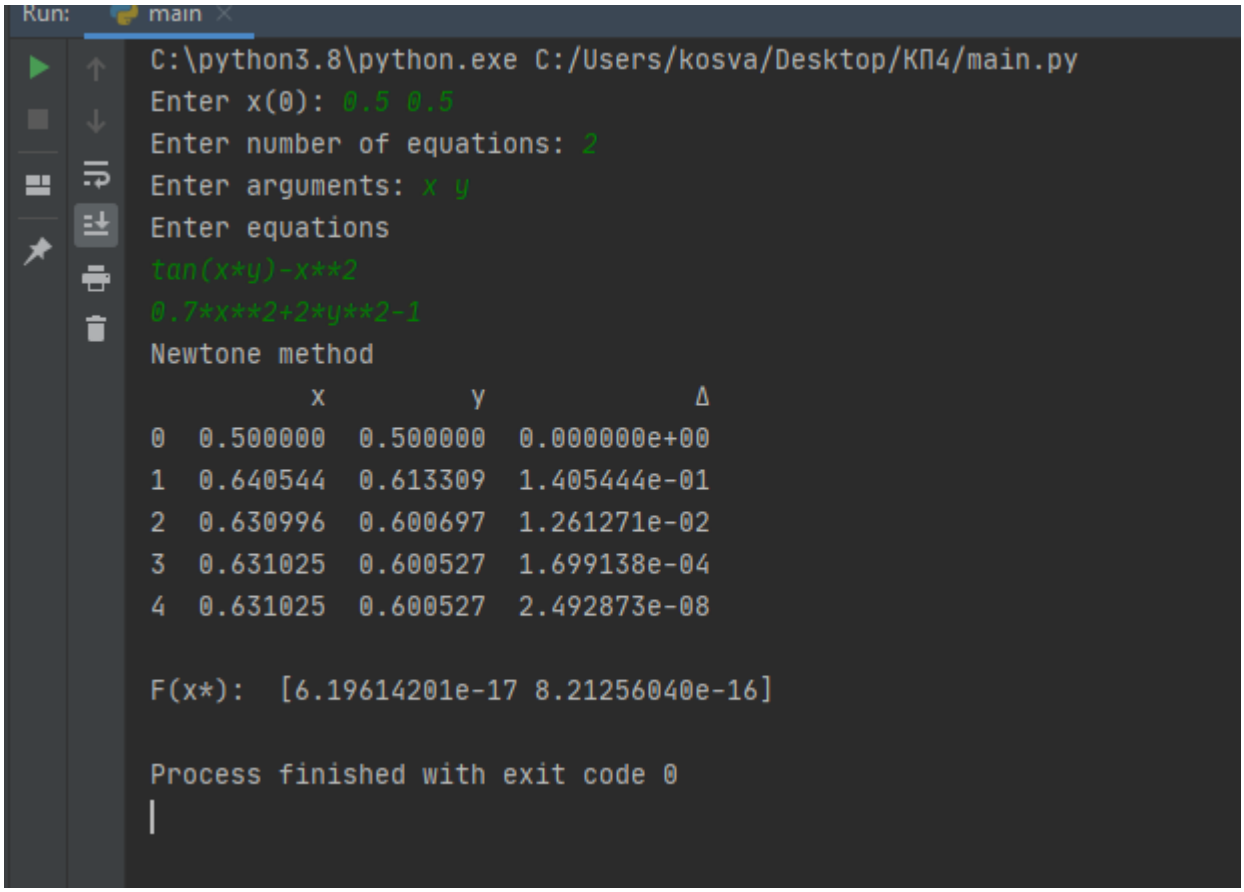


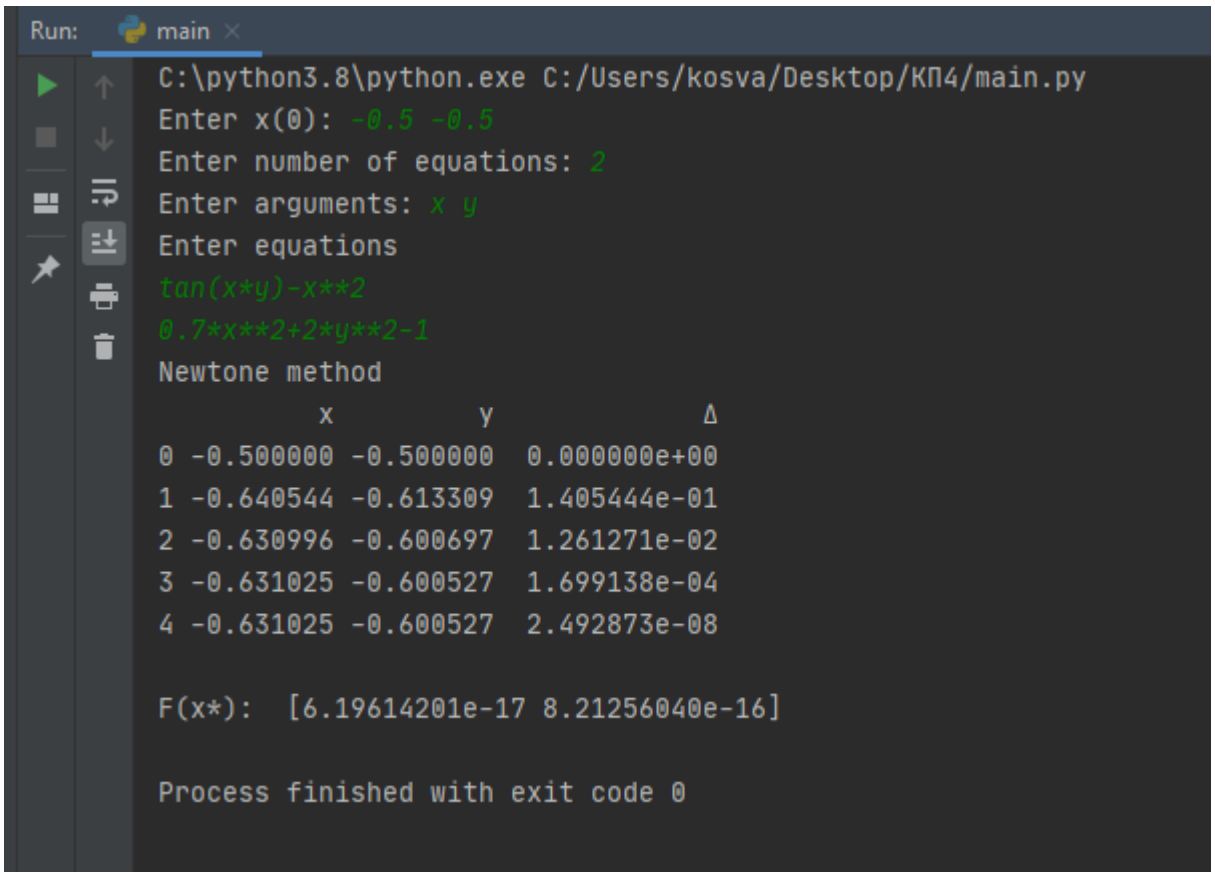
Бачимо, що дельта, і справді достатньо маленька, і відповідно далі бачимо, що при підстановці отриманих результатів, отримуємо досить маленькі значення (F(x*)), а тому можемо казати, що програма видає правильні результати.

Зауваження: правила вводу аналогічні минулому методу, тільки рівняння системи ми вводимо формату:

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

Але у нашої системи більше розв’язків, тому візьмемо і інші наближення.





```
Run:  main
C:\python3.8\python.exe C:/Users/kosva/Desktop/КП4/main.py
Enter x(0): 0 -1
Enter number of equations: 2
Enter arguments: x y
Enter equations
tan(x*y)-x**2
0.7*x**2+2*y**2-1
Newtone method
     x         y         Δ
0  0.0 -1.000000  0.000000
1  0.0 -0.750000  0.250000
2  0.0 -0.708333  0.041667
3  0.0 -0.707108  0.001225
4  0.0 -0.707107  0.000001

F(x*):  [0.00000000e+00 2.25541752e-12]

Process finished with exit code 0
```

Отже, бачимо, що беручи початкові наближення біля різних розв’язків, програма видає правильні відповідні розв’язки системи.

Тепер візьмемо точку, віддалену від усіх розв’язків.

```
Run:  main
C:\python3.8\python.exe C:/Users/kosva/Desktop/КП4/main.py
Enter x(0): 10 20
Enter number of equations: 2
Enter arguments: x y
Enter equations
tan(x*y)-x**2
0.7*x**2+2*y**2-1
Newtone method
                x          y             Δ
0   1.000000e+01  20.000000  0.000000e+00
1   1.983381e+01   7.416582  1.258342e+01
2   6.224145e+00   7.198382  1.360967e+01
3  -9.260200e+01  32.599931  9.882615e+01
4  -7.271728e+01  -9.955309  4.255524e+01
5  -3.240340e+01 -15.114450  4.031388e+01
6  -8.922222e+00 -13.035954  2.348118e+01
7   1.246666e+01 -10.592222  2.138888e+01
8   1.060319e+01  -3.519607  7.072615e+00
9   8.774760e+00   1.831314  5.350921e+00
10  3.872480e+00   1.915673  4.902280e+00
11 -1.028152e+00   3.185692  4.900632e+00
12 -1.115832e+00   1.603348  1.582344e+00
13 -1.829666e+00   0.647827  9.555212e-01
14 -7.115361e-01   0.910780  1.118130e+00
15 -3.381768e-01   0.734690  3.733593e-01
16 -8.929939e-02   0.720479  2.488774e-01
17 -1.020598e-02   0.708725  7.909341e-02
18 -1.654728e-04   0.707134  1.004051e-02
19 -4.495599e-08   0.707107  1.654278e-04
20 -3.320893e-15   0.707107  4.495599e-08

F(x*):  [-2.34822612e-15  1.39279014e-15]
```

Бачимо, що навіть в такому випадку, програма спрацьовує відмінно, правда кількість ітерацій значно зростає.

Висновок:

Виконуючи дану лабораторну роботу я навчився застосовувати чисельні методи розв’язання нелінійних систем. Я застосував 2 методи: метод простих ітерацій та метод Н’ютона.

У першому методі, побудувавши графік кривих системи та перевіривши достатні умови збіжності, я обрав початкове наближення: (-1, -1). Програма виконала верифікацію (відмінності верифікації і перевірки достатніх умов збіжностей я наводив раніше. Верифікація була пройдено успішно. Після чого за 10 ітерацій видало результат з заданою точністю

```
-0.945012 -1.097393
```

Потім взявши інше початкове наближення: (3, 4), достатні умови збіжності все одно були виконані, верифікація пройдено успішно, а результати роботи програми після 12 ітерацій, були вже з заданою точністю.

```
-0.945089 -1.097394
```

Задана точність $\varepsilon = 0.00001$

У другому методі, побудувавши графік кривих системи, я отримав, що система має 4 розв’язки. Першим початковим наближенням я обрав: (0, 1), потім (0.5, 0.5), потім (-0.5, -0.5), потім (0, -1). За кожного із цих наближень, програма видала результат з заданою точністю, причому в кожному випадку всього за 4 наближення.

Результати:

```
0.0  0.707107
```
при початковому наближені (0, 1)

```
0.631025  0.600527
```
при початковому наближені (0.5, 0.5)

```
-0.631025 -0.600527
```
при початковому наближені (-0.5, -0.5)

```
0.0 -0.707107
```
при початковому наближені (0, -1).

Також я взяв досить віддалене початкове наближення: (10, 20), за якого також отримав результат з заданою точністю:

```
-3.320893e-15   0.707107
```

Проте в цьому випадку, кількість ітерацій зросла до 20.

Кожен результат роботи програми за обох методів, я підставляв у відповідні рівняння, в яких результат мав прямувати до 0. Як бачимо, в усіх випадках F(x*) -і справді мав значення майже 0, що свідчить про правильність роботи моєї програми.

Зауваження: дана програма може працювати не тільки з заданими мені у завданні функціями, але і з будь якими іншими, задовільняючими правила вводу.