

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

Лабораторна робота № 3
з курсу «Чисельні методи»
з теми «Методи розв'язання нелінійних систем»
Варіант № 9

Виконав студент 2 курсу групи КА-02
Романович Володимир Володимирович
перевірила старший викладач
Хоменко Ольга Володимирівна

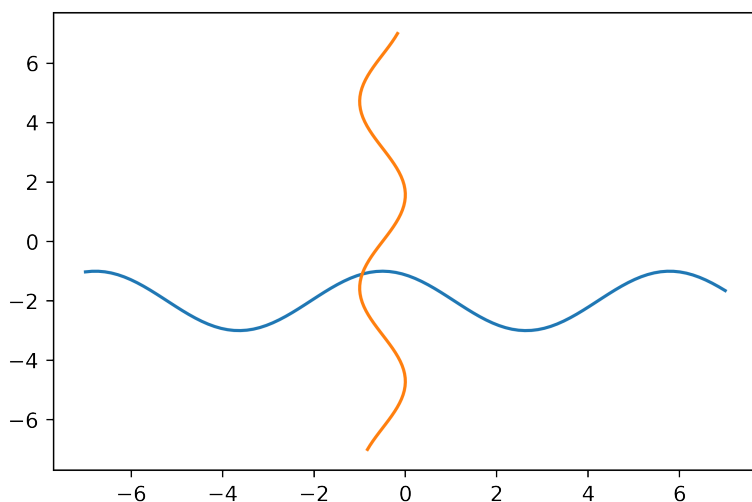
Задача 1

$$\begin{cases} \cos(x + 0.5) - y = 2 \\ \sin y - 2x = 1 \end{cases}$$

Запишемо систему у вигляді $\vec{x} = \Phi(\vec{x})$:

$$\begin{cases} x = \frac{1}{2}(\sin y - 1) \\ y = \cos(x + 0.5) - 2 \end{cases} \implies \vec{x} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(\sin y - 1) \\ \cos(x + 0.5) - 2 \end{pmatrix} = \Phi(\vec{x})$$

Побудуємо графік для знаходження початкового наближення:



Перевіримо достатню умову збіжності

Нехай $G = \{|x + 1| \leq 0.2, |y + 1| \leq 0.2\}$

$$\frac{\partial \varphi_1}{\partial x} = 0, \quad \frac{\partial \varphi_1}{\partial y} = \frac{1}{2} \cos y$$

$$\frac{\partial \varphi_2}{\partial x} = -\sin(x + 0.5), \quad \frac{\partial \varphi_2}{\partial y} = 0$$

$$\left| \frac{\partial \varphi_1}{\partial x} \right| + \left| \frac{\partial \varphi_2}{\partial x} \right| = |\sin(x + 0.5)| \leq |\sin(-1.2 + 0.5)| \leq 0.65 < 1$$

$$\left| \frac{\partial \varphi_1}{\partial y} \right| + \left| \frac{\partial \varphi_2}{\partial y} \right| = \left| \frac{1}{2} \cos y \right| \leq 0.5 < 1$$

Отже якщо послідовні наближення не будуть виходити за G то ітераційний процес збіжний і розв'язок єдиний.

Запрограмувавши метод простих ітерацій отримали розв'язок:

Iter	x	y	Δ
0	-1.000000	-1.000000	
1	-0.920735	-1.122417	0.122417
2	-0.950576	-1.087211	0.035206
3	-0.942667	-1.099803	0.012592
4	-0.945559	-1.096387	0.003416
5	-0.944781	-1.097630	0.001243
6	-0.945065	-1.097295	0.000335
7	-0.944989	-1.097417	0.000122
8	-0.945016	-1.097384	0.000033
9	-0.945009	-1.097396	0.000012
10	-0.945012	-1.097393	0.000003

Також взявши інші початкові наближення ми отримуємо такий же результат (з точністю до ε), але можливо з більшою кількістю ітерацій. Наприклад наведемо таблицю з початковим наближенням (500, -500):

Iter	x	y	Δ
0	500.000000	-500.000000	
1	-0.266114	-2.551389	500.266114
2	-0.778265	-1.027227	1.524162
3	-0.927934	-1.038467	0.149669
4	-0.930813	-1.090175	0.051708
5	-0.943354	-1.091374	0.012540
6	-0.943631	-1.096682	0.005308
7	-0.944849	-1.096801	0.001218
8	-0.944876	-1.097324	0.000523
9	-0.944995	-1.097336	0.000119
10	-0.944998	-1.097387	0.000051
11	-0.945010	-1.097388	0.000012
12	-0.945010	-1.097393	0.000005

Отже отримали розв'язок $(x, y) = (-0.94501; -1.09739)$

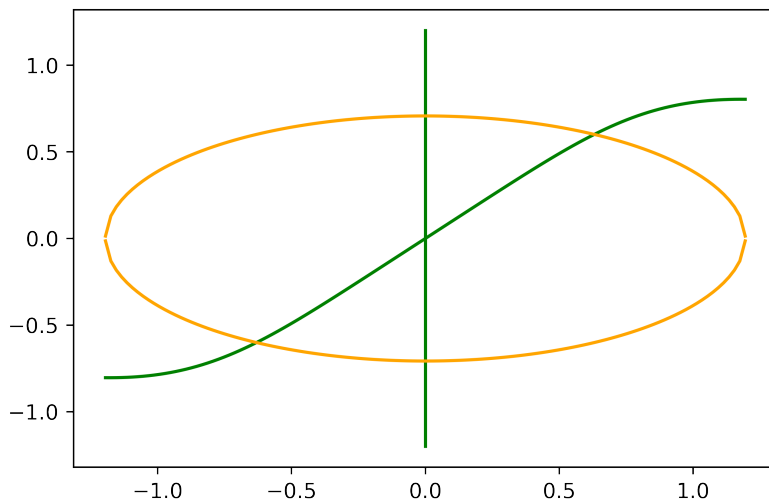
Задача 2

Розв'язати систему рівнянь $\begin{cases} tg(xy) = x^2 \\ 0.7x^2 + 2y^2 = 1 \end{cases}$ спрощеним методом Ньютона.
 $f_1(\vec{x}) = tg(xy) - x^2 = 0, \quad f_2(\vec{x}) = 0.7x^2 + 2y^2 - 1 = 0$

$$W = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix} = \begin{pmatrix} \frac{y}{\cos^2(xy)} - 2x & \frac{x}{\cos^2(xy)} \\ 1.4x & 4y \end{pmatrix}$$

$$F = \begin{pmatrix} tg(xy) - x^2 \\ 0.7x^2 + 2y^2 - 1 \end{pmatrix}$$

Побудуємо графік для знаходження початкового наближення:



Бачимо що графіки мають 4 точки перетину, оберемо для цих точок початкові наближення: (0.6, 0.6), (0, 0.7), (-0.6, -0.6), (0, -0.7)

Запрограмувавши спрощений метод Ньютона, отримали такі розв'язки:

(0.63103, 0.60053) (0, 0.70711) (-0.63103, -0.60053) (0, -0.70711)

Для розв'язку (0.63103, 0.60053) наведемо таблицю із значеннями x, y, Δ на кожній ітерації:

Iter	x	y	Δ
0	0.600000	0.600000	
1	0.632322	0.600354	0.032322
2	0.630903	0.600546	0.001419
3	0.631037	0.600525	0.000134
4	0.631024	0.600527	0.000013
5	0.631025	0.600527	0.000001

Задавши початкове наближення $(100, 200)$ ми не отримали розв’язок за 999 ітерацій та, скоріш за все, метод не є збіжним при такому початковому наближенні.

Також була проведена перевірка за допомогою функції `fsolve()` та обчисленням $F(\vec{x}_*)$, де \vec{x}_* – знайдений розв’язок. Отримали що з точністю ϵ наш розв’язок дорівнює розв’язку з `fsolve()` та $F(\vec{x}_*) = \vec{0}$

Висновки

Мовою програмування Python ми реалізували методи простих ітерацій та спрощений метод Ньютона для розв’язку нелінійних систем рівнянь, побудувавши графік знайшли початкові наближення та підставивши ці наближення в функції, отримали правильний(перевірений декількома способами) розв’язок. У випадку методу простих ітерацій із зміною початкового наближення відповідь не змінилась, проте збільшилась кількість ітерацій, а у випадку спрощеного методу Ньютона, через те що він тільки локально збіжний, ми не змогли отримати розв’язок з наближенням $(100, 200)$. Це не дивно, оскільки область збіжності цього методу є досить малою, і потрібно обирати початкове наближення близьке до розв’язку, як ми це і зробили після побудови графіку.

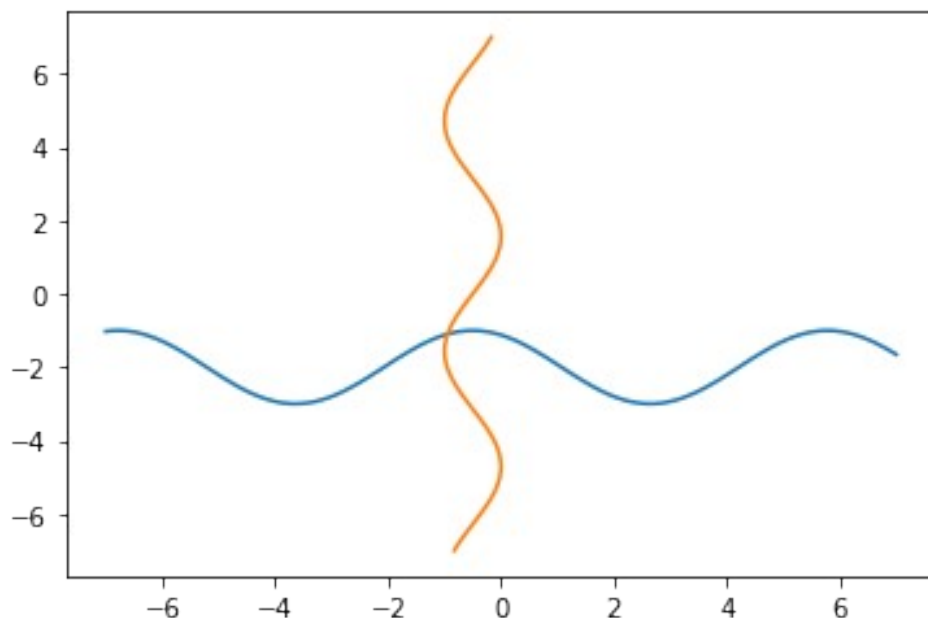
Лістинг програми наведений знизу

Побудуємо графіки цих функцій

```
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.linspace(-7, 7, 100)
y1 = np.cos(x + 0.5) - 2
y = np.linspace(-7, 7, 100)
x2 = 0.5 * (np.sin(y) - 1)

plt.plot(x, y1)
plt.plot(x2, y)
plt.savefig('plot.png', dpi=500)
plt.show()
```



Бачимо, що перетин кривих є близьким до точки $(-1, -1)$, отже її і візьмемо як початкове наближення

Запишемо систему у вигляді $\begin{cases} x = \frac{1}{2} (\sin(y) - 1) \\ y = \cos(x + 0.5) - 2 \end{cases}$, отже $\vec{x} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2} (\sin(y) - 1) \\ \cos(x + 0.5) - 2 \end{pmatrix} = \Phi(\vec{x})$

```
from scipy.optimize import fsolve
import pandas as pd
```

```
def phi_1(x_vec):
    """Takes (x,y) vector and returns 0.5(sin(y)-1)"""
    (x, y) = x_vec
    return 0.5 * (np.sin(y) - 1)
```

```

def phi_2(x_vec):
    """Takes (x,y) vector and returns cos(x + 0.5) - 2"""
    (x,y) = x_vec
    return np.cos(x+0.5) - 2

def iterative_solve(Phi_vec, starting_value_vec, epsilon=0.00001,
print_table = False):
    """Takes vector-function  $\Phi$  and starting value, solves  $x=\Phi(x)$ 
equation, returns x vector and print latex code table if chosen to"""
    df = pd.DataFrame({"x": [starting_value_vec[0]], "y":
[starting_value_vec[1]], "delta": [" "]})
    prev_x_vec = starting_value_vec.copy()
    current_x_vec = [None, None]
    iteration = 0
    while True:
        iteration += 1
        for i in range(len(prev_x_vec)):
            current_x_vec[i] = Phi_vec[i](prev_x_vec)
        delta = max([abs(x) for x in np.subtract(prev_x_vec,
current_x_vec)])

        df.loc[iteration] = [current_x_vec[0], current_x_vec[1],
delta]

        if delta < epsilon:
            if print_table:
                print(df.style.to_latex())
            return current_x_vec
        else:
            prev_x_vec = current_x_vec.copy()

```

```

Phi_vector = [phi_1, phi_2]
# x = iterative_solve(Phi_vector, [-1,-1], print_table=True)
# print(iterative_solve(Phi_vector, [5, 100]))
# print(iterative_solve(Phi_vector, [0, 0]))
# print(iterative_solve(Phi_vector, [500, -500], print_table=True))

```

```

\begin{tabular}{lrrl} Iter & x & y & \Delta \\ 0 & -1.000000 & -1.000000 & \ 1 & -0.920735 \\ & -1.122417 & 0.122417 & \ 2 & -0.950576 & -1.087211 & 0.035206 \\ & \ 3 & -0.942667 & -1.099803 & 0.012592 \\ & \ 4 & -0.945559 & -1.096387 & 0.003416 \\ & \ 5 & -0.944781 & -1.097630 & 0.001243 \\ & \ 6 & -0.945065 & -1.097295 & 0.000335 \\ & \ 7 & -0.944989 & -1.097417 & 0.000122 \\ & \ 8 & -0.945016 & -1.097384 & 0.000033 \\ & \ 9 & -0.945009 & -1.097396 & 0.000012 \\ & \ 10 & -0.945012 & -1.097393 & 0.000003 \\ \end{tabular}

```

Напишемо функцію, щоб записувати нецілі числа з 5 знаками після коми (більше нам не потрібно, оскільки точність 0.00001)

```

def format_float(flt):
    """Function to format floats so only 5 digits after decimal is

```

```

shown"""
    if flt is int:
        return flt
    else:
        return float("{:.5f}".format(flt))

```

Перевіримо наш розв'язок використовуючи fsolve() з scipy.optimize, та порахувавши $\vec{x} - \Phi(\vec{x})$

```

def F1(x_vec):
    (x, y) = x_vec
    f1 = 0.5 * (np.sin(y) - 1) - x
    f2 = np.cos(x+0.5) - 2 - y
    return [f1, f2]

result = fsolve(F1, (-1, -1))
print("SciPy result:", [format_float(num) for num in result])

x = iterative_solve(Phi_vector, [-1, -1])
print("Our result:", [format_float(num) for num in x])
x_minus_Phi = np.subtract(x, [Phi_vector[0](x), Phi_vector[1](x)])
print("x-Phi(x):", [format_float(num) for num in x_minus_Phi])

SciPy result: [-0.94501, -1.09739]
Our result: [-0.94501, -1.09739]
x-Phi(x): [-0.0, 0.0]

```

Бачимо, що з точністю до ϵ наш розв'язок рівний розв'язку fsolve(), та $\vec{x} - \Phi(\vec{x}) = \vec{0}$

\begin{center} \Large Задача 2 \end{center}

$$\begin{cases} tg(xy) = x^2 \\ 0.7x^2 + 2y^2 = 1 \end{cases}$$

Для початку побудуємо графік

```

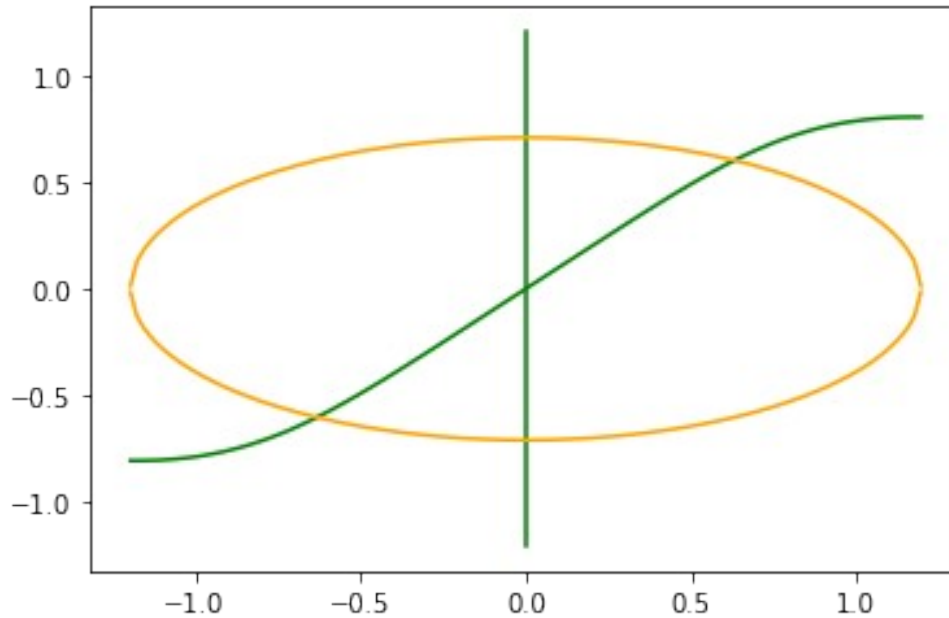
x = np.linspace(-1.195, 1.195, 120)
y1 = np.arctan(x**2)/x
y = np.linspace(-1.2, 1.2, 120)
x2 = 0*x
y3 = np.sqrt(1-0.7*x**2) / np.sqrt(2)
y4 = -y3

plt.plot(x, y1, color="green")
plt.plot(x2, y, color="green")
plt.plot(x, y3, color="orange")
plt.plot(x, y4, color="orange")

```



```
plt.savefig("plot2.png", dpi=500)
plt.show()
```



Бачимо що система має 4 розв'язки, в точках, близьких до (0.6,0.6), (0,0.7), (-0.6,-0.6), (0,-0.7). Саме такі точки будемо брати як початкове наближення. Знайдемо тепер матрицю Якобі та матрицю F

$$W = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix} = \begin{pmatrix} \frac{y}{\cos^2(xy)} - 2x & \frac{x}{\cos^2(xy)} \\ 1.4x & 4y \end{pmatrix}$$

$$F = \begin{pmatrix} tg(xy) - x^2 \\ 0.7x^2 + 2y^2 - 1 \end{pmatrix}$$

Далі шукаємо розв'язок використовуючи ітераційну формулу

$$x^{(k+1)} = x^{(k)} - W^{-1}(x^{(0)}) \cdot F(x^{(k)})$$

```
def W(x_vec):
```

```
    """Takes vector and returns Jacobi matrix of specific shown above
    function"""
```

```
    (x,y) = x_vec
    df1dx = y/(np.cos(x * y))**2 - 2*x
    df1dy = x/(np.cos(x * y))**2
    df2dx = 1.4 * x
    df2dy = 4 * y
    return np.array([
        [df1dx, df1dy],
        [df2dx, df2dy]
```

```

    ])

def F(x_vec):
    """Takes vector x_vec and returns F(x_vec), where F is a vector-
    function shown above"""
    (x,y) = x_vec
    f1 = np.tan(x * y) - x ** 2
    f2 = 0.7 * x**2 + 2 * y**2 - 1
    return [f1, f2]

def newton_method(W, F, starting_value_vec, epsilon=0.00001,
print_table=False):
    """Takes [x0,y0] starting value and returns [x,y] solution of
    system found by newton method"""
    df = pd.DataFrame({"x": [starting_value_vec[0]], "y":
[starting_value_vec[1]], "delta": [" "]})
    prev_x_vec = list(starting_value_vec)
    W_inv_0 = np.linalg.inv(W(prev_x_vec))
    iteration = 0
    while True:
        if iteration > 999:
            raise RuntimeError
        iteration += 1
        #  $x^{k+1} = x^k - W^{-1} * F(x^k)$ 
        current_x_vec = np.subtract(np.array([prev_x_vec]), W_inv_0 @
F(prev_x_vec))
        delta_vec = np.subtract(prev_x_vec, current_x_vec)[0] #
Result of np.subtract is [[delta1...deltan]] so delta_vec is
[delta1...deltan]
        delta = max([abs(d) for d in delta_vec])
        df.loc[iteration] = [current_x_vec[0][0], current_x_vec[0][1],
delta]
        if delta < epsilon:
            if print_table:
                print(df.style.to_latex())
            return current_x_vec[0]
        else:
            prev_x_vec = current_x_vec[0]

print(newton_method(W, F, [0.6,0.6], print_table=True))
print(newton_method(W, F, [0, 0.7]))
print(newton_method(W, F, [-0.6,-0.6]))
print(newton_method(W, F, [0, -0.7]))

\begin{tabular}{lrrl}
& x & y & delta \\
0 & 0.600000 & 0.600000 & \\
1 & 0.632322 & 0.600354 & 0.032322 \\
2 & 0.630903 & 0.600546 & 0.001419 \\
3 & 0.631037 & 0.600525 & 0.000134
\end{tabular}

```

```
4 & 0.631024 & 0.600527 & 0.000013 \\
5 & 0.631025 & 0.600527 & 0.000001 \\
\end{tabular}
```

```
[0.63102545 0.6005268 ]
[0.          0.70710678]
[-0.63102545 -0.6005268 ]
[ 0.          -0.70710678]
```

Отримали розв'язки (0.63102545, 0.6005268), (0, 0.70710678), (-0.63102545, -0.6005268), (0, -0.70710678). Для розв'язку з початковим наближенням (0.6, 0.6) наведемо таблицю:

```
\begin{tabular}{cccc} Iter & x & y & Δ \\ 0 & 0.600000 & 0.600000 & \\ 1 & 0.632322 & 0.600354 & 0.032322 \\ 2 & 0.630903 & 0.600546 & 0.001419 \\ 3 & 0.631037 & 0.600525 & 0.000134 \\ 4 & 0.631024 & 0.600527 & 0.000013 \\ 5 & 0.631025 & 0.600527 & 0.000001 \end{tabular}
```

Перевіримо ці розв'язки за допомогою функції fsolve із scipy.optimize та підрахувавши $F(\vec{x}_i)$

```
scipy_results = [fsolve(F, (0.6, 0.6)), fsolve(F, (0, 0.7)), fsolve(F, (-0.6, -0.6)), fsolve(F, (0, -0.7))]
scipy_results = [[format_float(num) for num in lst] for lst in
scipy_results] # Formatting results
print("SciPy results:", *scipy_results)
```

```
our_results = [newton_method(W, F, (0.6, 0.6)), newton_method(W, F, (0, 0.7)), newton_method(W, F, (-0.6, -0.6)), newton_method(W, F, (0, -0.7))]
F_x = [F(x_vec) for x_vec in our_results]
our_results = [[format_float(num) for num in lst] for lst in
our_results] # Formatting results
print("Our results:", *our_results)
F_x = [[format_float(num) for num in lst] for lst in F_x] #
Formatting results
print(*F_x)
```

```
SciPy results: [0.63103, 0.60053] [-0.0, 0.70711] [-0.63103, -0.60053]
[0.0, -0.70711]
Our results: [0.63103, 0.60053] [0.0, 0.70711] [-0.63103, -0.60053]
[0.0, -0.70711]
[-0.0, 0.0] [0.0, 0.0] [-0.0, 0.0] [-0.0, 0.0]
```

Бачимо що із заданою точністю наші розв'язки дорівнюють розв'язкам функції fsolve, та $F(\vec{x}_i) = \vec{0}$ Спробуємо взяти інші початкові наближення, наприклад (100, 200)

```
try:
    print(newton_method(W, F, [5, 5]))
```

```
except RuntimeError:
    print("Can't find a solution in 999 iterations")

/tmp/ipykernel_103363/2585968209.py:16: RuntimeWarning: overflow
encountered in double_scalars
    f1 = np.tan(x * y) - x ** 2
/tmp/ipykernel_103363/2585968209.py:16: RuntimeWarning: invalid value
encountered in tan
    f1 = np.tan(x * y) - x ** 2
/tmp/ipykernel_103363/2585968209.py:17: RuntimeWarning: overflow
encountered in double_scalars
    f2 = 0.7 * x**2 + 2 * y**2 - 1

Can't find a solution in 999 iterations
```

Бачимо що при такому початковому наближенні спрощений метод Ньютона не є збіжним