Laboratorium 7

Równania nieliniowe

Bartłomiej Szubiak

23.04.2024

Zadania

Zad 1

Napisz iteracje wg metody Newtona do rozwiązywania każdego z następujących równań nieliniowych:
(a) x cos(x) =1;
(b) x3 - 5x - 6 = 0;
(c) e-x = x--2-1.

Aby policzyć pierwiastki wykorzystam metodę Newtona-Raphsona

Iteracja Newtona dana jest wzorem:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Oszacuje przedziały takie, że dla miejsca zerowego z tego przedziału zawężę przedział poszukiwania tak aby funkcja zmieniała znak, nie zawierała ekstremów (ze względu na pochodna = 0), oraz aby była ona wypukła lub wklęsła na tym przedziale.

Dla wszystkich obliczeń przyjmę dokładność 10^{-6} oraz maksymalną ilość iteracji na 1000

Dodatkowo uruchomię algorytm dwukrotnie na każdym przypadku rozpoczynając od lewego i prawego krańca przedziału.

Wykorzystam kod Python:

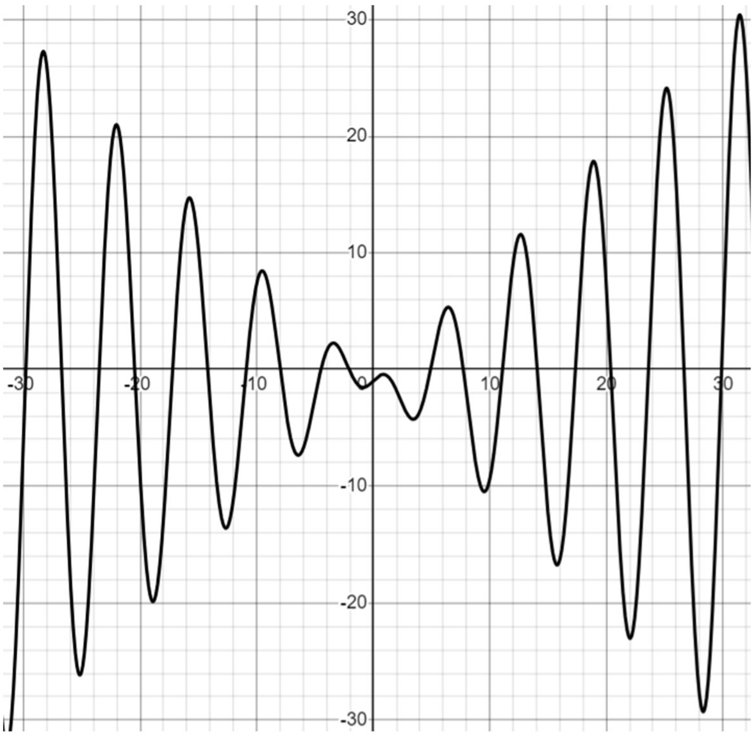
```
def newton_method(f, df, x0, tol=1e-10, max_iter=1000):
    x = x0
    for n in range(max_iter):
        f_x = f(x)
        if abs(f_x) < tol:
            print(f'Znaleziono rozwiązanie po {n+1} iteracjach.')
            return x
        df_x = df(x)
        if df_x == 0:
            print('ERROR Zero derivative. No solution found.')
```

```
        return None
      x = x - f_x / df_x
    print('WARNING No solution found within tolerance.')
    return x

def find_root_in_interval(f, df, interval):
    print(f"Przedział: {interval}")
    for initial_guess in interval:
        print(f"Przybliżenie początkowe: {initial_guess}")
        root = newton_method(f, df, initial_guess)
        print(f"Miejsce zerowe:", root)
    print('\n')
```

a) $\boldsymbol{x \quad cos\, x = 1}$

Zmieniamy postać równania na f (x) = 0 , przerzucamy wszystkie elementy na lewą stronę równania:

$f(x) = x\ cos\ x\ - 1$, i tą funkcje będziemy przyrównywać do 0 w poszukiwaniu pierwiastków.



Rys 1.1.1: Wykres funkcji $x\ cos\ x\ - 1$

Widzimy, ze funkcja jest okresowa i ma nieskończenie wiele pierwiastków.

Oblicze tylko wartości wszystkich pierwiastków z przedziału [-6,6]

Przedziały poszukiwania 3 miejsc zerowych:

1. [-5,-4]
2. [-3,-1]
3. [4,6]

Dla każdego sprawdzamy warunek: f(a) · f(b) < 0 gdzie a, b to krańce przedziału w którym szukamy miejsca zerowego.

1. f(-5) = −2.418, < 0
   f(-4) = 1.615 , > 0
   f(-5) · f(-4) to iloczyn liczby ujemnej z dodatnią więc jest < 0
2. f(-3) = 1.97, > 0
   f(-1) = -1.54 , < 0
   f(-3) · f(-1) to iloczyn liczby ujemnej z dodatnią więc jest < 0
3. f(4) = -3.615, < 0
   f(6) = 4.761 , > 0
   f(4) · f(6) to iloczyn liczby ujemnej z dodatnią więc jest < 0

Warunek jest spełniony w każdym przypadku, zatem funkcja na pewno ma pierwiastki w przedziałach [-5,-4], [-3,-1], [4,6].

$f(x) = x\, cos\, x\; - 1$

$f'(x) = -\mathrm{x} \cdot \sin(\mathrm{x}) + \cos(\mathrm{x})$

$f''(x) = -2\sin(\mathrm{x}) - \mathrm{x} \cdot \cos(\mathrm{x})$

Wyniki algorytmu:

Przedział: [-5, -4]
Przybliżenie początkowe: -5
Znaleziono rozwiązanie po 5 iteracjach.
Miejsce zerowe: -4.487669603341089
Przybliżenie początkowe: -4
Znaleziono rozwiązanie po 6 iteracjach.
Miejsce zerowe: -4.487669603341088

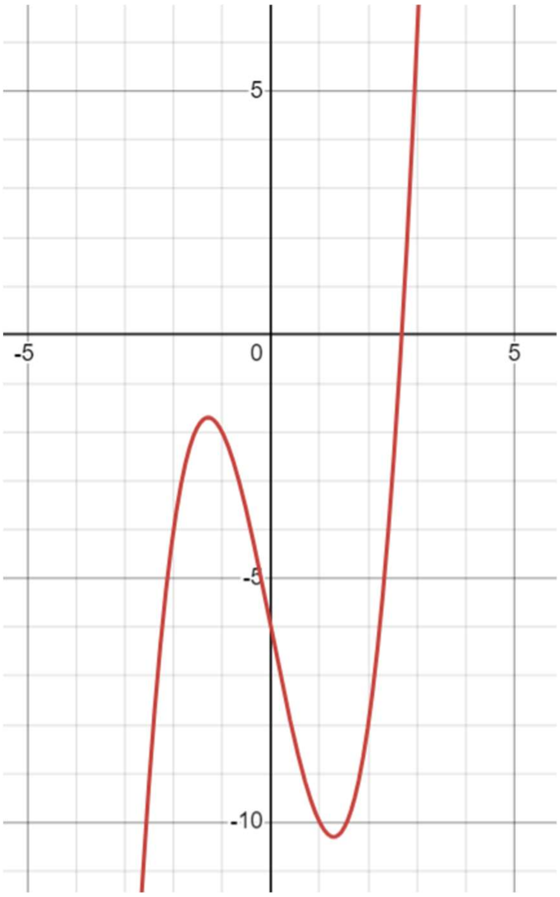
Przedział: [-3, -1]
Przybliżenie początkowe: -3
Znaleziono rozwiązanie po 6 iteracjach.
Miejsce zerowe: -2.0739328090912252
Przybliżenie początkowe: -1
Znaleziono rozwiązanie po 7 iteracjach.
Miejsce zerowe: -2.073932809091215

Przedział: [4, 6]
Przybliżenie początkowe: 4
Znaleziono rozwiązanie po 6 iteracjach.
Miejsce zerowe: 4.917185925287132
Przybliżenie początkowe: 6
Znaleziono rozwiązanie po 6 iteracjach.
Miejsce zerowe: 4.917185925287132

Kod python dla tego przypadku:

```
def f(x):
    return x * np.cos(x) - 1

def df(x):
    return -x * np.sin(x) + np.cos(x)

def ddf(x):
    return -2 * np.sin(x) - x * np.cos(x)

x_intervals = [[-5, -4], [-3, -1], [4, 6]]
for interval in x_intervals:
    find_root_in_interval(f, df, interval)
```

b) $\boldsymbol{x^3 - 5x - 6 = 0}$



Rys 1.2.1: Wykres funkcji $x^3 - 5x - 6$

Z powyższego wykresu odczytujemy przebieg funkcji $f(x) = x^3 - 5x - 6$ aby znaleźć przedział w którym znajduje się pierwiastek równania $f(x) = 0$.

Przedziałem tym jest m.in. [2,4].

Sprawdzamy warunek: f(a) · f(b) < 0 gdzie a, b to krańce przedziału w którym szukamy miejsca zerowego.

f(2) = −8, < 0

f(4) = 38 , > 0

f(2) · f(4) to iloczyn liczby ujemnej z dodatnią więc jest < 0

Warunek jest spełniony, zatem funkcja na pewno ma pierwiastek w przedziale [2,4]

Aby policzyć pierwiastek wykorzystam metode Newtona-Raphsona

$f(x) = x^3 - 5x - 6$

$f'(x) = 3x^2 - 5$

$f''(x) = 6x$

Wyniki algorytmu:

Przedział: [2, 4]
Przybliżenie początkowe: 2
Znaleziono rozwiązanie po 6 iteracjach.
Miejsce zerowe: 2.6890953236426456
Przybliżenie początkowe: 4
Znaleziono rozwiązanie po 6 iteracjach.
Miejsce zerowe: 2.6890953236398376

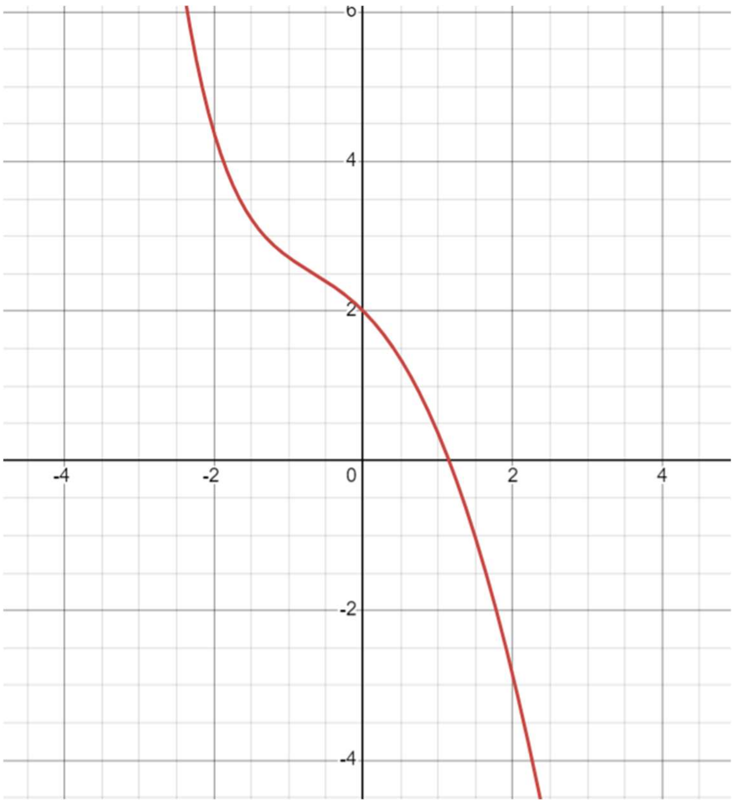
Kod python dla tego przypadku:

```
def f(x):
   return x**3 - 5*x - 6

def df(x):
   return 3*x**2 - 5

find_root_in_interval(f, df, [2, 4])
```

c) $\mathbf{e^{-x}} = \boldsymbol{x^2} - \mathbf{1}$

Zmieniamy postać równania na f (x) = 0 , przerzucamy wszystkie elementy na lewą stronę równania:

$f(x) = e^{-x} - x^2 + 1$, i tą funkcje będziemy przyrównywać do 0 w poszukiwaniu pierwiastków.



Rys 1.3.1: Wykres funkcji $f(x) = e^{-x} - x^2 + 1$

Z powyższego wykresu odczytujemy przebieg funkcji $f(x) = e^{-x} - x^2 + 1$ aby znaleźć przedział w którym znajduje się pierwiastek równania $f(x) = 0$.

Przedziałem tym jest m.in. [0,2].

Sprawdzamy warunek: f(a) · f(b) < 0 gdzie a, b to krańce przedziału w którym szukamy miejsca zerowego.

f(0) = 2, > 0

f(2) = -2,865, < 0

f(0) · f(2) to iloczyn liczby ujemnej z dodatnią więc jest < 0

Warunek jest spełniony, zatem funkcja na pewno ma pierwiastek w przedziale [0,2]

Po raz kolejny do obliczenia miejsca zerowego używam Newtona-Raphsona.

$$f(x) = e^{-x} - x^2 + 1$$
$$f'(x) = -e^{-x} - 2x$$
$$f''(x) = e^{-x} - 2$$

Wyniki algorytmu:

Przedział: [0, 2]
Przybliżenie początkowe: 0
Znaleziono rozwiązanie po 7 iteracjach.
Miejsce zerowe: 1.1477576321447434
Przybliżenie początkowe: 2
Znaleziono rozwiązanie po 6 iteracjach.
Miejsce zerowe: 1.1477576321447434

Kod python dla tego przypadku:

```
import numpy as np

def f(x):
    return np.exp(-x) - x**2 + 1

def df(x):
    return -np.exp(-x) - 2*x

def ddf(x):
    return np.exp(-x) - 2

find_root_in_interval(f, df, [0, 2])
```

Wnioski:
Dla powyższych przykładów wynik został znaleziony dość szybko, nie zależnie z jakiego punkty zaczynaliśmy

Zad 2

(a) Pokaż, że iteracyjna metoda $x_{k+1} = \dfrac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$ matematycznie jest równoważna z metodą siecznych przy rozwiązywaniu skalarnego nieliniowego równania f (x) = 0.

Dla równania skalarnego gdzie mnożenie jest przemienne aby uzyskać powyższy wzór wystarczy wychodząc od równania z metody siecznych kilka przekształceń algebraicznych

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_{k-1}) - f(x_k)}$$

$$x_{k+1} = x_k - \frac{f(x_k)x_k - f(x_k)x_{k-1}}{f(x_{k-1}) - f(x_k)}$$

$$x_{k+1} = \frac{x_k\big(f(x_{k-1}) - f(x_k)\big) - f(x_k)x_k + f(x_k)x_{k-1}}{f(x_{k-1}) - f(x_k)}$$

$$x_{k+1} = \frac{x_k f(x_{k-1}) - f(x_k)x_{k-1}}{f(x_{k-1}) - f(x_k)}$$

Co daje nam szukany zapis.

(b) Jeśli zrealizujemy obliczenia w arytmetyce zmiennoprzecinkowej o skończonej precyzji, jakie zalety i wady ma wzór podany w podpunkcie (a), w porównaniu ze wzorem dla metody siecznych podanym poniżej?

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$$

Wzór A:

$$x_{k+1} = \frac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$$

Wzór B:

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$$

Zalety wzoru A względem B:

- zawiera on 2 mnożenia 2 odejmowania i 1 dzielenie, gdzie wzór B zawiera 3 odejmowania i 1 mnożenie i 1 dzielenie, (wzór A ma 1 mnożenie kosztem odejmowania) odejmowania bardzo małych liczb mogą prowadzić do efektu „Catastrophic cancellation"
- Unikanie pośrednich obliczeń: Wzór A wykonuje bezpośrednie obliczenie nowej wartości x_{k+1} bez konieczności najpierw obliczania różnicy wartości funkcji i skalowania przez różnicę $x_k - x_{k-1}$. To może prowadzić do zmniejszenia błędów zaokrągleń w pewnych scenariuszach, gdzie wartości $f(x_k)$ i $f(x_{k-1})$ są bliskie sobie.
- Numeryczna stabilność dla bliskich wartości funkcji: Jeśli wartości $f(x_k)$ i $f(x_{k-1})$ są bardzo bliskie sobie, standardowa metoda siecznych może doświadczyć problemów z dzieleniem przez małą wartość (prawie zero), co prowadzi do dużych błędów. Wzór alternatywny teoretycznie może lepiej radzić sobie w takich sytuacjach, choć i tutaj dzielenie przez małą wartość pozostaje ryzykiem.

Wady wzoru A względem B:

- jest bardziej skomplikowanym wzorem nie widać skąd bezpośrednio się wywodzi
- złożoność obliczeniowa jeżeli mnożenia są droższe obliczeniowo od odejmowania
- akumulacja błędów zaokrągleń

Podsumowanie

Oba wzory mają swoje zalety i wady, a wybór odpowiedniego może zależeć od specyfiki problemu, wrażliwości na błędy zaokrągleń, oraz od wartości, które są przetwarzane. Alternatywny wzór może być korzystny w przypadkach, gdy różnice wartości funkcji są bardzo małe, ale należy być świadomym ryzyka dzielenia przez wartości bliskie zero, co może powodować niestabilności numeryczne.

Zad 3

Zapisz iteracje Newtona do rozwiązywania następującego układu równań nieliniowych.

$x_1^2 + x_1x_2^3 = 9$
$3x_1^2\, x_2 - x_2^3 = 4$

Oznaczmy $x := x_1$, $y := x_2$, oraz przerzućmy wszystkie składniki na lewą stronę wtedy:

$$x^2 + xy^3 - 9 = 0$$
$$3x^2y - y^3 - 4 = 0$$

Co można przedstawić również jako:

$$g\,(x, y) = 0$$
$$h\,(x, y) = 0$$

Metoda Newtona dla funkcji wielu zmiennych jest uogólnieniem tej metody dla funkcji jednej zmiennej. W metodzie stycznych wykorzystywaliśmy wzór: $\mathrm{f(x)} \approx \mathrm{f(X_0) + f'(X_0)(X - X_0)}$.

Jeśli przyjmiemy $\mathrm{X} \approx \mathrm{X_0}$ otrzymamy: $\mathrm{X = X_0 - f(X_0)/f'\,(X_0)}$

Dla funkcji wielu zmiennych możemy f przybliżyć równaniem podobnym:

$$\mathrm{f(X) \approx f(X_0) + Df(X_0) \cdot (X - X_0)}, \quad \text{gdzie } \mathrm{X} \in \mathrm{R^n} \text{ oraz:}$$

$$Df(x_0) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Załóżmy, że f(X) = 0 wtedy nasze równanie przyjmie postać:

$$0 \; = \; \mathrm{f(X)} \;\; \approx \;\; \mathrm{f(X_0) + Df(X_0)(X - X_0)}$$

Jeśli macierz Df jest odwracalna to możemy wyliczyć z tego równania x:

$$\mathrm{X = X_0 - [Df(X_0)]^{-1} f(X_0)}$$

Otrzymujemy następujące równanie rekurencyjne:

$$\mathrm{X}_{n+1} = \mathrm{X_n} - [Df(\mathrm{X_n})]^{-1} f(\mathrm{X}_n)$$

Generuje ono ciąg który dąży do miejsca zerowego funkcji f:

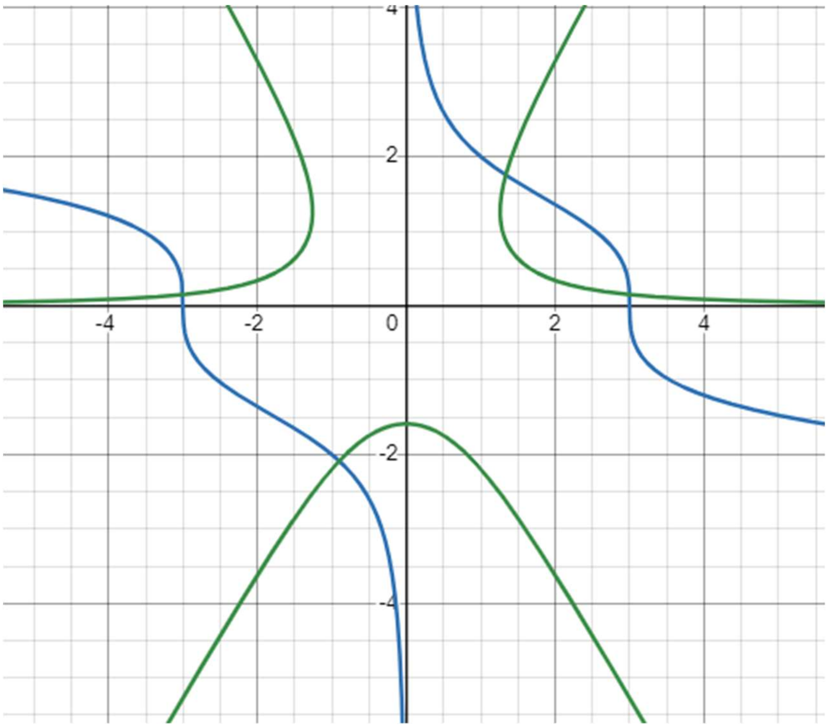
$$f(x, y) = \begin{bmatrix} g\,(x, y) \\ h\,(x, y) \end{bmatrix} = \begin{bmatrix} x^2 + xy^3 - 9 \\ 3x^2y - y^3 - 4 \end{bmatrix}$$

Obliczmy różniczke f'(x,y):

$$Df(x, y) = \begin{bmatrix} 2x + y^3 & 3x \cdot y^2 \\ 6y \cdot x & 3x^2 - 3y^2 \end{bmatrix}$$

Równanie sprowadza się do postaci:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - \begin{bmatrix} 2x + y^3 & 3x \cdot y^2 \\ 6y \cdot x & 3x^2 - 3y^2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} x^2 + xy^3 - 9 \\ 3x^2y - y^3 - 4 \end{bmatrix}$$



Rysunek 3.1: Równanie $x^2 + xy^3 - 9 = 0$ na tle równania $3x^2y - y^3 - 4 = 0$

Z wykresu możemy wywnioskować, że pierwiastki będą w:

1. [-3.5 , -2.5] x [-0.5 , 0.5]
2. [3.5 , 2.5] x [-0.5 , 0.5]
3. [-1.5 , -0.5] x [-2.5 , -1.5]
4. [1, 1.5] x [1.5 , 2]

Wyniki: Dla dokładności 10^{-10} , maksymalnej ilości iteracji 100:

Punkt startowy: [-3.5, -0.5]
Znaleziono rozwiązanie po 6 iteracjach.
Rozwiązanie: [-3.00162489 0.14810799]

Punkt startowy: [3.5, -0.5]
Znaleziono rozwiązanie po 6 iteracjach.
Rozwiązanie: [2.99836535 0.14843098]

Punkt startowy: [-1.5, -2.5]
Znaleziono rozwiązanie po 6 iteracjach.
Rozwiązanie: [-0.90126619 -2.08658759]

Punkt startowy: [1, 1.5]

Znaleziono rozwiązanie po 6 iteracjach.
Rozwiązanie: [1.33635538 1.7542352]

Kod Python:

```
import numpy as np

# Definicja funkcji
def F(X):
    x , y = X
    return np.array([
        x**2 + x*y**3 - 9,
        3*x**2*y - y**3 - 4
    ])

# Definicja Jacobianu
def jacobian(X):
    x , y = X
    return np.array([
        [2*x + y**3, 3*x*y**2],
        [6*x*y, 3*x**2 - 3*y**2]
    ])

# Metoda Newtona
def newton_method(F, jacobian, X_0, tol=1e-10, max_iter=100):
    X_n = np.array(X_0)
    for i in range(max_iter):
        J = jacobian(X_n)
        Fx = F(X_n)
        try:
            # Obliczenie kroku metody Newtona
            delta_x = np.linalg.solve(J, -Fx)
        except np.linalg.LinAlgError:
            print("Błąd macierzy Jacobiego. Może być osobliwa.")
            return None
        X_n = X_n + delta_x
        if np.linalg.norm(delta_x) < tol:
            print(f'Znaleziono rozwiązanie po {i+1} iteracjach.')
            return X_n
    print('Nie znaleziono rozwiązania w dopuszczalnej liczbie iteracji.')
    return None

starting_points = [
    [-3.5,-0.5],
    [3.5,-0.5],
    [-1.5,-2.5],
    [1,1.5]
]

for X in starting_points:
```

```
print(f"Punkt startowy: {X}")
solution = newton_method(F, jacobian, X)
print("Rozwiązanie:", solution)
print('\n')
```

Wnioski:
Wynik został znaleziony dość szybko, pomimo większego wymiaru.

Bibliografia:

wykład dr inż. Katarzyna Rycerz

https://www.desmos.com/calculator?lang=pl

https://home.agh.edu.pl/~funika/mownit/lab7/RF.pdf