Metody Numeryczne - Zadanie XVI

Mateusz Kamiński

December 2025

1 Wstęp

W zadaniu należało rozwiązać układ równań opisany elipsą oraz okręgiem.

2 Opis

Celem znalezienia więcej niż jednego pierwiastka, wybrano cztery potencjalne punkty, bliskie możliwych pierwiastków.

2.1 Równanie

Układ równań ma postać

$$\begin{cases} 2x^2 + y^2 = 2 \\ \left(x - \frac{1}{2}\right)^2 + (y-1)^2 = \frac{1}{4} \end{cases}$$

2.2 Metoda Newtona dla układu równań

Rozważamy układ równań nieliniowych:

$$g(x) = \begin{pmatrix} 2X^2 + Y^2 - 2 \\ \left(X - \frac{1}{2}\right)^2 + (Y-1)^2 - \frac{1}{4} \end{pmatrix}$$

Iteracyjna metoda Newtona przebiega w następujących krokach:

1. Początkowe przybliżenie: wybieramy wektor startowy $[x_0, y_0]$).
2. Obliczenie Jacobiana:

$$J(x) = \begin{pmatrix} 4X & 2Y \\ 2(X-0.5) & 2(Y-1) \end{pmatrix}$$

3. Obliczenie przyrostu δ rozwiązując układ liniowy:

$$J\left(x^{(k)}\right)\delta = g\left(x^{(k)}\right)$$

4. Aktualizacja przybliżenia:

$$x^{(k+1)} = x^{(k)} - \delta x^{(k)}$$

5. Warunek stopu

$$\|x^{(k+1)} - x^{(k)}\| < \varepsilon$$

3 Implementacja

```
import numpy as np
from numpy.typing import NDArray

vector = NDArray[np.float64]
matrix = NDArray[np.float64]
num = np.float64

def g(x: vector) -> vector:
    X = x[0]
    Y = x[1]

    return np.array([
        2*X**2 + Y**2 - 2, # f1(x)
        (X - 0.5)**2 + (Y - 1)**2 - 0.25], # f2(x)
        dtype=num)

def jacobian(x: vector) -> matrix:
    # Jacobian policzony analitycznie!
    X = x[0]
    Y = x[1]
    return np.array([
        [4*X, 2*Y],
        [2*(X-0.5), 2*(Y-1)]
    ], dtype=num)

def newton_iteration(x: vector) -> vector:
    J = jacobian(x)
    F = g(x)
```

```
    # J * delta = -F
    delta = np.linalg.solve(J, F)
    return x - delta

def newton(x: vector, tolerance: float = 1e-10, iterations: int = 50) ->
tuple[vector, int]:

    for i in range(iterations):
        x_next = newton_iteration(x)

        if np.linalg.norm(x_next - x) < tolerance:
            return x_next, i + 1

        x = x_next

    return x, iterations


def main():
    # punkty startowe obejmujące wszystkie możliwe pierwiastki
    guesses = [
        np.array([0, 0.5]),
        np.array([0, 1.5]),
        np.array([1, 0.5]),
        np.array([1, 1.5])
    ]

    roots = []

    for x0 in guesses:
        root, iterations = newton(x0)
        # sprawdzenie unikalności
        if not any(np.allclose(root, r, atol=1e-8) for r in roots):
            roots.append(root)
            print(f"Znaleziono pierwiastek x = {root} w {iterations}
iteracjach")

if __name__ == "__main__":
    main()
```

4 Wyniki

$$(x_0, y_0) = (0.18639893, 1.38942826)$$
$$(x_1, y_1) = (0.8791207, 0.67401305)$$

Liczba iteracji potrzebna do znalezienia pierwiastków to odpowiednio: [9, 5]