Zadanie Numeryczne 15

Mateusz Wojtyna

1. Wstęp

Należało rozwiązać układ równań nieliniowych

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

Układ ten składa się z równania elipsy oraz okręgu. Rozwiązanie zostanie znalezione numerycznie przy użyciu wielowymiarowej metody Newtona.

2. Opis

Zdefiniujmy wektor zmiennych $\mathbf{x} = [x, y]^T$ oraz wektorową funkcję $\mathbf{g}(\mathbf{x})$, której miejsc zerowych szukamy:

$$\mathbf{g}(x, y) = \begin{bmatrix} 2x^2 + y^2 - 2 \\ (x - 0.5)^2 + (y-1)^2 - 0.25 \end{bmatrix}$$

Do zastosowania metody Newtona wymagane jest wyznaczenie jakobianu $\mathbf{J}(\mathbf{x})$:

$$\mathbf{J}(x, y) = \begin{bmatrix} \frac{\partial g_1}{\partial x} & \frac{\partial g_1}{\partial y} \\ \frac{\partial g_2}{\partial x} & \frac{\partial g_2}{\partial y} \end{bmatrix}$$

Obliczając poszczególne pochodne:

$$\frac{\partial g_1}{\partial x} = 4x, \quad \frac{\partial g_1}{\partial y} = 2y$$

$$\frac{\partial g_2}{\partial x} = 2(x - 0.5) = 2x - 1, \quad \frac{\partial g_2}{\partial y} = 2(y-1) = 2y - 2$$

Otrzymujemy postać jakobianu użytą w programie:

$$\mathbf{J}(x, y) = \begin{bmatrix} 4x & 2y \\ 2x - 1 & 2y - 2 \end{bmatrix}$$

Iteracja jest zadana przez:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{z}$$

gdzie $\mathbf{z}$ wyznaczamy przez rozwiązanie układu równań:

$$\mathbf{J}(\mathbf{x}_k)\mathbf{z} = \mathbf{g}(\mathbf{x}_k)$$

Proces ten zatrzymujemy gdy $\|\mathbf{x}_{k+1} - \mathbf{x}\| \leq \varepsilon$.

3. Kod

Program napisano w Pythonie z użyciem pakietu *NumPy*.

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

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


def g(x: vector) -> vector:
    g_1 = 2 * x[0] ** 2 + x[1] ** 2 - 2
    g_2 = (x[0] - 0.5) ** 2 + (x[1] - 1) ** 2 - 0.25
    return np.array([g_1, g_2])


def jacobian(x: vector) -> matrix:
    j_11 = 4 * x[0]
    j_12 = 2 * x[1]
    j_21 = 2 * x[0] - 1
    j_22 = 2 * x[1] - 2
    return np.array([[j_11, j_12], [j_21, j_22]])


def newton_method(x_0: vector, eps: float = 1e-8, limit: int = 1000):
    x = x_0.copy()

    for i in range(limit):
```

```
        z = np.linalg.solve(jacobian(x), g(x))
        x_new = x - z

        if np.linalg.norm(x_new - x) <= eps:
            return (x_new, i + 1)

        x = x_new

    return (x, limit)


def main():
    # Dobrano punkty startowe tak, aby trafić w miejsca zerowe
    guesses = np.array([[0.6, 1], [0.4, 1]])

    for x_0 in guesses:
        x, steps = newton_method(x_0)
        print(f"Znaleziono pierwiastek x = {x} w {steps} iteracjach")


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

4. Wyniki ($\varepsilon = 10^{-8}$)

Program został uruchomiony dla dwóch punktów startowych dobranych w pobliżu spodziewanych przecięć krzywych. Znaleziono dwa rozwiązania układu równań, oba po 8 iteracjach:

- dla $\mathbf{x}_0 = [0.6, 1]$ wynik $\mathbf{x} \approx [0.87912070, 0.67401305]^T$,
- dla $\mathbf{x}_0 = [0.4, 1]$ wynik $\mathbf{x} \approx [0.18639893, 1.38942826]^T$