Zadanie Numeryczne 15

Mateusz Wojtyna

1. Wstęp

Należało znaleźć wszystkie rozwiązania równania

$$z^4 + iz^3 - z^2 - iz + 1 = 0$$

stosując metodę Laguerre'a razem ze strategią obniżania stopnia wielomianu i wygładzania.

2. Opis

Metoda Laguerre'a dla danego przybliżenia z_k oblicza kolejne przybliżenie z_{k+1} wzorem

$$z_{k+1} = z_k - \frac{nP(z_k)}{P'(z_k) \pm \sqrt{(n-1)\left[(n-1)(P'(z_k))^2 - nP(z_k)P''(z_k)\right]}}$$

gdzie n to stopień wielomianu a znak w mianowniku wybierany jest tak, aby zmaksymalizować jego moduł.

Algorytm programu działa następująco:

1. Wartości $P(z)$, $P'(z)$ oraz $P''(z)$ obliczane są przy użyciu schematu Hornera.
2. Po znalezieniu miejsca zerowego z_0 z zadaną dokładnością, wygładzamy go, potem robimy deflację wielomianu. Wielomian dzielony jest przez $(z - z_0)$ przy użyciu *forward substitution*, co obniża jego stopień o 1.
3. Proces powtarzany jest do momentu, gdy stopień wielomianu spadnie do 2. Pozostałe pierwiastki obliczane są analitycznie ze wzoru na pierwiastki równania kwadratowego.

3. Kod

Program napisano w Pythonie z użyciem pakietu *NumPy*.

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

vector = NDArray[np.complex128]
array = NDArray[np.complex128]
num = np.complex128


def horner(z: num, coeffs: array):
    """
    Zwraca (f(z), f'(z), f''(z))
    """
    N = len(coeffs) - 1
    p = coeffs[0]  # f(z)
    p_prime = 0  # f'(z)
    p_prime2 = 0  # f''(z)

    for k in range(1, N + 1):
        p_prime2 = p_prime + z * p_prime2
        p_prime = p + z * p_prime
        p = p * z + coeffs[k]

    return p, p_prime, 2 * p_prime2


def laguerre(z: num, coeffs: array, eps=1e-8, limit=1000):
    n = len(coeffs) - 1

    for i in range(limit):
        P, P_prime, P_prime2 = horner(z, coeffs)

        sqrt = np.sqrt((n - 1) * ((n - 1) * P_prime**2 - n * P * P_prime2))
        denom1 = P_prime + sqrt
        denom2 = P_prime - sqrt
        denom = denom1 if abs(denom1) > abs(denom2) else denom2

        z_new = z - n * P / denom

        if np.abs(z_new - z) <= eps:
            return z_new, i + 1

        z = z_new
```

```
    return z, limit


def forward_substitution(coeffs: vector, z_0: num) -> vector:
    """
    Dzieli wielomian P(z) przez (z - z_0).
    Zwraca wspolczynniki nowego wielomianu o stopniu n-1.
    """
    n = len(coeffs) - 1
    new_coeffs = np.zeros(n, dtype=num)

    new_coeffs[0] = coeffs[0]
    for i in range(1, n):
        new_coeffs[i] = coeffs[i] + z_0 * new_coeffs[i - 1]

    return new_coeffs


def calc_roots(coeffs: array):
    roots = []
    steps_list = []
    original_coeffs = coeffs.copy()

    while len(coeffs) > 3:
        # 1. Policzenie Laguerrem
        root, steps = laguerre(0, coeffs)
        steps_list.append(steps)

        # 2. Wygładzenie
        wygladzony_root, s = laguerre(root, original_coeffs)
        roots.append(wygladzony_root)

        # 3. Deflacja
        coeffs = forward_substitution(coeffs, wygladzony_root)

    # Ostatnie 2 pierwiastki liczymy analitycznie
    roots.extend(np.roots(coeffs))

    return roots, steps_list


def main():
    coeffs = np.array([1, 1j, -1, -1j, 1], dtype=num)  # a_n, ... ,a_0
    roots, steps = calc_roots(coeffs)
    print(roots, steps)
```

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

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

$$
\begin{aligned}
z_0 &= 0.58778525 - 0.80901699i && \text{(po 6 iteracjach)} \\
z_1 &= -0.95105652 + 0.30901699i && \text{(po 5 iteracjach)} \\
z_2 &= 0.95105652 + 0.30901699i && \text{(analitycznie)} \\
z_3 &= -0.58778525 - 0.80901699i && \text{(analitycznie)}
\end{aligned}
$$