Metody Numeryczne - Zadanie XV

Mateusz Kamiński

December 2025

1 Wstęp

Celem zadania jest, stosując metodę Laguerre'a ze strategią obniżania stopnia wielomianu i wygładzania, znaleźć wszystkie rozwiązania równania

2 Opis

2.1 Równanie

Równanie ma postać

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

2.2 Schemat Hornera

Do obliczenia pierwszej i drugiej pochodnej używamy schematu Hornera.

Wielomian zespolony stopnia n:

$$P(z) = a_0 + a_1 z + a_2 z^2 + ... + a_n z^n$$

można przekształcić w postać zagnieżdżoną (Hornera):

$$P(z) = a_0 + z(a_1 + z(a_2 + ... + z(a_{n-1} + za_n)...))$$

co pozwala obliczyć jego wartość w punkcie z w jednej pętli.

Dodatkowo, pierwsza i druga pochodna mogą być obliczone w tym samym przebiegu. Z rachunku różniczkowego wiemy, że $(z \cdot b_{i-1})' = b_{i-1} + zb'_{i-1}$

$$b_i = a_i + zb_{i-1} \quad \rightarrow \quad P(z_0) = b_n$$

to pochodna spełnia zależność rekurencyjną:

$$c_i = b_{i-1} + zc_{i-1} \quad \rightarrow \quad P'(z_0) = c_n$$

a druga pochodna:

$$d_i = c_{i-1} + zd_{i-1} \quad \rightarrow \quad P''(z_0) = 2d_n$$

Dzięki temu w jednej iteracji możemy policzyć P(z), P'(z) i P''(z), bez osobnego liczenia potęg ani pochodnych.

3 Implementacja

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

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

def horner(coeffs: vector, z: num) -> tuple[num, num, num]:
    # stopień wielomianu
    n = len(coeffs) - 1
    b = coeffs[0]  # P(z)
    c = 0          # P'(z)
    d = 0          # P''(z)

    for i in range(1, n+1):
        d = c + z * d
        c = b + z * c
        b = coeffs[i] + z * b

    return b, c, 2*d

def laguerre(coeffs: vector, z: num, tolerance=1e-9, iterations=1000) ->
tuple[num, int]:
    n = len(coeffs) - 1

    for i in range(iterations):
        P, P_prime, P_double_prime = horner(coeffs, z)

        # aby nie liczyć dwa razy
        # numerator = n * P
        nP = n * P

        square = (n - 1) * (((n - 1) * P_prime ** 2) - nP *
P_double_prime)
```

```
        denominator1 = P_prime + square
        denominator2 = P_prime - square
        denominator = denominator1 if abs(denominator1) >
abs(denominator2) else denominator2

        z_new = z - nP / denominator

        if abs(z_new - z) < tolerance:
            return z_new, i + 1

        z = z_new

    return z, iterations

def forward_substitution(coeffs: vector, z0: num) -> vector:
    n = len(coeffs) - 1
    b = np.zeros(n, dtype=num)

    b[0] = coeffs[0]
    for i in range(1, n):
        b[i] = coeffs[i] + z0 * b[i-1]

    return b

def main():
    # z^4 + i z^3 - z^2 - i z + 1
    coeffs = np.array([1, 1j, -1, -1j, 1], dtype=num)
    roots = []
    iterations = []

    while len(coeffs) > 2:
        z0 = 0 + 0j
        root, iter = laguerre(coeffs, z0)
        roots.append(root)
        iterations.append(iter)

        coeffs = forward_substitution(coeffs, root)

    # równanie kwadratowe, lecimy deltą!
    quad_roots = np.roots(coeffs)
    roots.extend(quad_roots)

    print("Pierwiastki wielomianu: ")
    for root in roots:
        print(root)

    print(iterations)

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

4 Wyniki

Zastosowanie metody Laguerre'a dla początkowego przybliżenia $z = 0 + 0i$ i podanego wielomianu daje następujące pierwiastki:

$$z_0 = (0.5877852472400774 - 0.8090169947213557i)$$

$$z_1 = (0.9510565188312418 + 0.3090169919309872i)$$

$$z_2 = (-0.5877852510371671 - 0.8090169917970373i)$$

$$z_3 = (-0.9510565150341519 + 0.30901699458740584i)$$

Liczba iteracji potrzebna do znalezienia kolejnych pierwiastków: [133, 71]

Pozostałe pierwiastki zostały obliczone w czasie stałym za pomocą wzoru na równanie kwadratowe