Metody Numeryczne - Zadanie XII

Mateusz Kamiński

December 2025

1 Wprowadzenie

Celem zadania jest skonstruowanie interpolacji funkcjami wymiernymi według algorytmu Floatera i Hormanna z parametrem $d = 3$ dla funkcji

$$f(x) = \frac{1}{1 + 5x^2}$$

oraz węzłów z zadania 8:

$$x = \left[-\frac{7}{8}, -\frac{5}{8}, -\frac{3}{8}, -\frac{1}{8}, \frac{1}{8}, \frac{3}{8}, \frac{5}{8}, \frac{7}{8}\right].$$

2 Algorytm Floatera–Hormanna

Niech $p_i(x)$ będzie wielomianem interpolującym rozpiętym na punktach $x_i, x_{i+1}, \ldots, x_{i+d}$. Wybieramy parametr d, który określa liczbę sąsiednich węzłów branych pod uwagę w każdym lokalnym wielomianie — większe d zwiększa dokładność, ale może prowadzić do oscylacji, mniejsze d daje interpolację bardziej lokalną i stabilną numerycznie. Wówczas interpolacja $r(x)$ w postaci barycentrycznej jest dana wzorem:

$$r(x) = \frac{\sum_{k=0}^{n} \frac{w_k f_k}{x - x_k}}{\sum_{k=0}^{n} \frac{w_k}{x - x_k}},$$

gdzie wagi w_k obliczamy raz na początku algorytmu. Dla ogólnych węzłów wagi są dane wzorem:

$$w_k = (-1)^i \prod_{\substack{i \in J_k \\ j=i,\, j \neq k}}^{i+d} \frac{1}{x_k - x_j}, \quad J_k = \{i \in [0, n-d] : k - d \leqslant i \leqslant k\}.$$

W naszym przypadku węzły są równooddalone, co pozwala uprościć wzór do postaci użytej w implementacji:

$$w_k = (-1)^{k-d} \sum_{i \in J_k} \binom{d}{k-i},$$

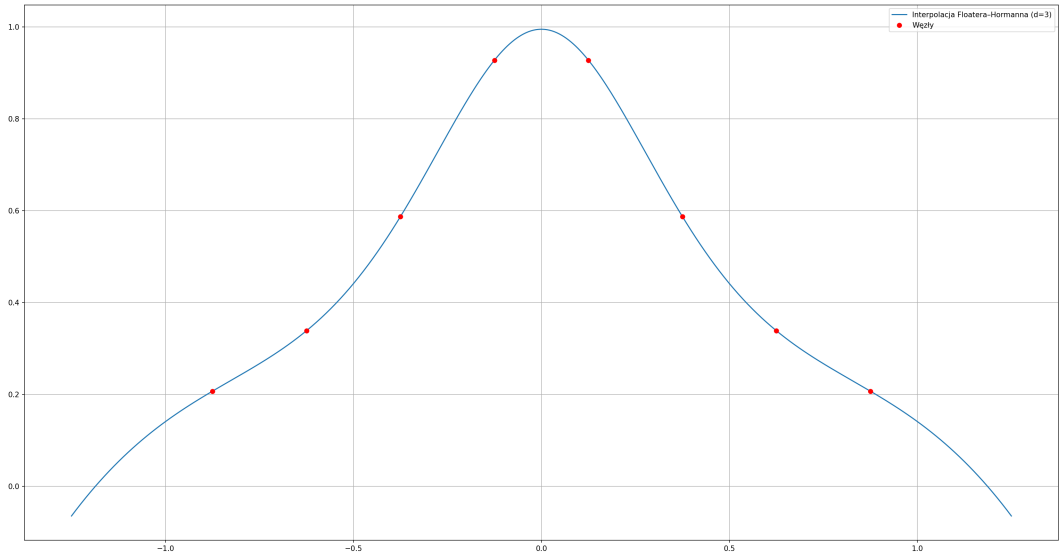
Dodatkowo, jeżeli x znajduje się bardzo blisko któregoś węzła x_l, wynik interpolacji przyjmujemy jako f_l, aby uniknąć dzielenia przez zero i zapewnić dokładne odwzorowanie wartości węzłów.

3 Implementacja

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.typing import NDArray
from scipy.special import comb

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

def f_x(x: num) -> num:
    return 1.0 / (1 + 5 * x * x)

def floater_hormann_weights(x: vector, d: int) -> vector:
    n = len(x)
    w = np.zeros(n, dtype=num)

    for k in range(n):
        start = max(0, k - d)
        end = min(k, n - d - 1)

        s = 0.0
        for i in range(start, end + 1):
            s += comb(d, k - i, exact=True)

        sign = 1 if (k - d) % 2 == 0 else -1
        w[k] = sign * s

    return w

def floater_hormann_function(x_nodes: vector, f:vector, w: vector, x_eval:
    vector) -> vector:
    r = np.zeros_like(x_eval)
    for idx, x in enumerate(x_eval):
        diff = x - x_nodes

        mask = np.isclose(diff, 0.0, atol=1e-12)
        if np.any(mask):
            r[idx] = f[mask][0]
```

```
        else:
            w_diff = w / diff
            numerator = np.sum(w_diff * f)
            denominator = np.sum(w_diff)
            r[idx] = numerator / denominator

    return r


def main():
    x: vector = np.array([-7/8, -5/8, -3/8, -1/8, 1/8, 3/8, 5/8, 7/8],
        dtype=num)
    n = len(x)
    f: vector = np.zeros(n, dtype=num)

    for i in range(n):
        f[i] = f_x(x[i])

    d = 3
    w = floater_hormann_weights(x, d)
    print(w)

    x_eval = np.linspace(-1.25, 1.25, 400)
    y_eval = floater_hormann_function(x, f, w, x_eval)

    plt.plot(x_eval, y_eval, label="Interpolacja Floatera Hormanna (d=3)"
        )
    plt.plot(x, f, 'ro', label="Węzły")
    plt.legend()
    plt.grid(True)
    plt.show()

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

4 Wyniki



Rysunek 1: Interpolacja Floatera-Hormanna w przedziale [-1.25, 1.25]