Metody Numeryczne - Zadanie IX

Mateusz Kamiński

December 2025

1 Wprowadzenie

Celem ćwiczenia było zaimplementowanie interpolacji Lagrange'a w celu wyznaczenia wielomianu interpolacyjnego opartego o węzły Czebyszewa i porównanie wyników z zadania 8.

2 Opis

2.1 Węzły Czebyszewa

W zadaniu wyznaczamy wielomian interpolacyjny dla tej samej funkcji

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

w przedziale

$$x \in [-1.25, 1.25]$$

ale w węzłach typu Czebyszewa:

$$x_j = \cos \frac{2j - 1}{16}\pi, \quad j = 1, 2, \ldots, 8.$$

Interpolacja odbywa się analogicznie jak w zadaniu 8.

3 Implementacja

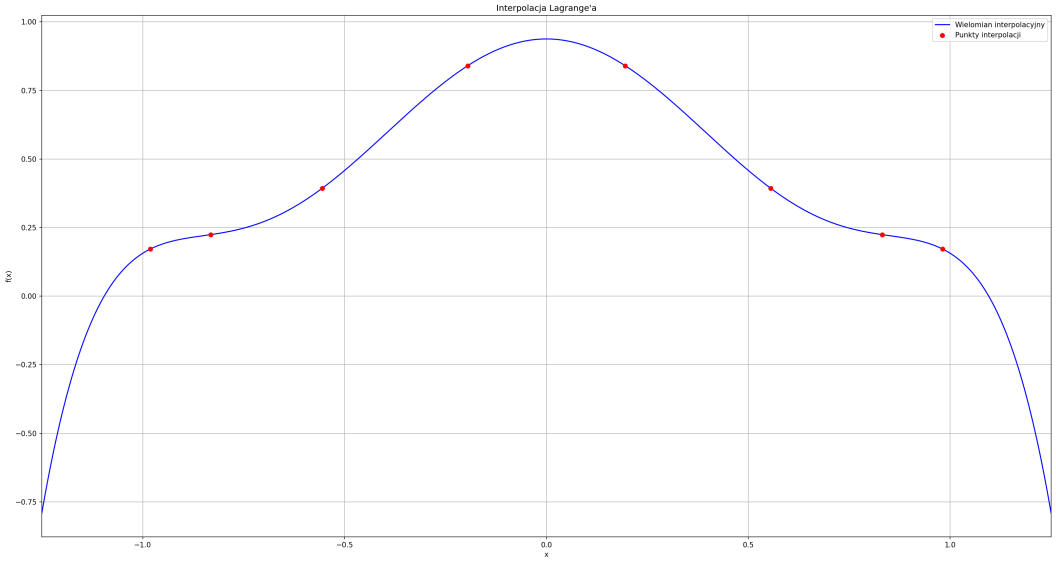
```
import numpy as np
from numpy.typing import NDArray
import matplotlib.pyplot as plt

vector = NDArray[np.float64]
```

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

def l_j(x_val: num, x: vector, j: int) -> num:
    n = len(x)
    numerator: num = np.float64(1.0)
    denominator: num = np.float64(1.0)

    # 0...j-1
    for k in range(0, j):
        numerator *= x_val - x[k]
        denominator *= x[j] - x[k]

    # j+1...n-1
    for k in range(j + 1, n):
        numerator *= x_val - x[k]
        denominator *= x[j] - x[k]

    return numerator / denominator

def coefficients(x: vector, f: vector) -> matrix:
    n = len(x)

    lj = np.zeros(n, dtype=num)
    for j in range(n):
        lj[j] = l_j(0, x, j)

    # współczynniki
    a = np.zeros(n, dtype=num)
    f_k = f.copy()

    for k in range(n):
        a[k] = np.sum(lj * f_k)

        for j in range(n):
            f_k[j] = (f_k[j] - a[k]) / x[j]

    return a

def P(x: num, a: vector) -> num:
    # Wyliczenie wielomianu schematem hornera
    val = 0.0
    for coeff in reversed(a):
        val = val * x + coeff

    return val

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

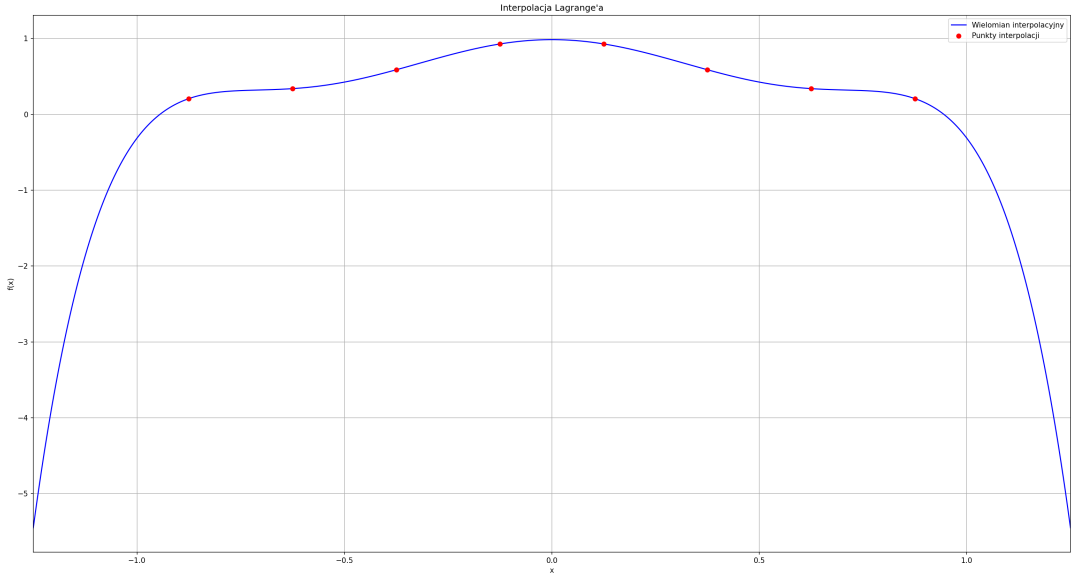
```
def x_j(j: int) -> num:
    return np.cos(((2*j - 1) / 16)* np.pi)

def main():
    n = 8
    x: vector = np.zeros(n, dtype=num)
    for i in range(n):
        x[i] = x_j(i + 1)

    f: vector = np.zeros(n, dtype=num)

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

    x_uniform: vector = np.array([-7/8, -5/8, -3/8, -1/8, 1/8, 3/8, 5/8,
        7/8], dtype=num)
    f_uniform: vector = np.zeros(n, dtype=num)

    for i in range(n):
        f_uniform[i] = f_x(x_uniform[i])

    coeff = coefficients(x, f)
    coeff_uniform = coefficients(x_uniform, f_uniform)

    x_plot = np.linspace(-1.25, 1.25, 1000)
    y_plot = np.array([P(x, coeff) for x in x_plot])
    y_plot_uniform = np.array([P(x, coeff_uniform) for x in x_plot])

    plt.plot(x_plot, y_plot, label="Czebyszew", color="blue")
    plt.plot(x_plot, y_plot_uniform, label="Równomierne", color="green")

    plt.scatter(x, f, color="red", label="Punkty interpolacji Czebyszewa",
        zorder=5)
    plt.scatter(x_uniform, f_uniform, color="#006400", label="Punkty
        interpolacji równomiernej", zorder=5)

    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.title("Interpolacja Lagrange'a")
    plt.grid(True)
    plt.legend()
    plt.xlim(-1.25, 1.25)
    plt.show()

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

4 Wyniki

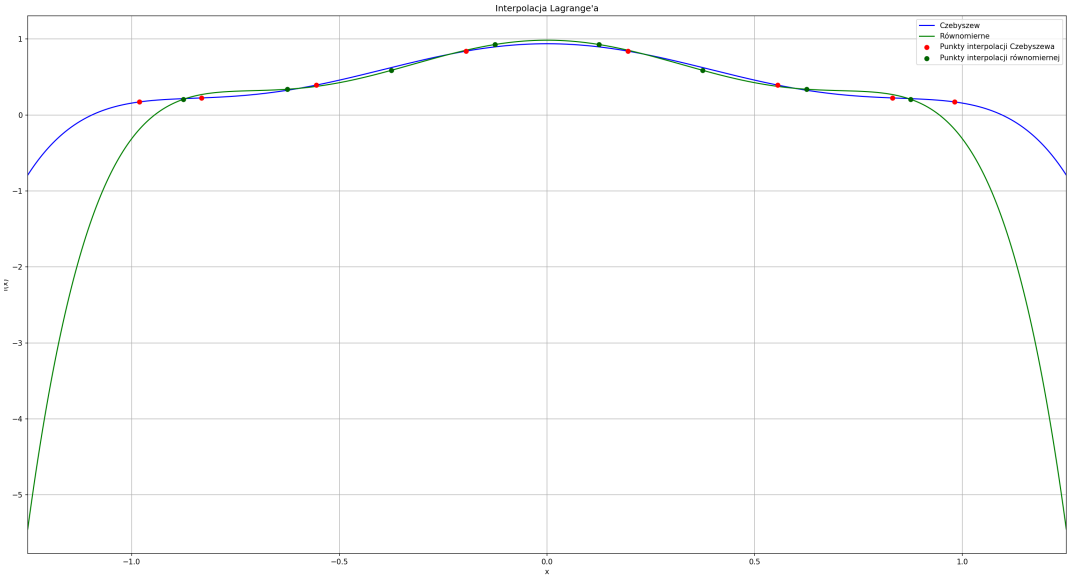
Rozkład punktów poprzez węzły Czebyszewa okazuje się lepszym, gładszym przybliżeniem funkcji. Interpolacja w tych węzłach ma również zmniejszone odchylenia na krańcach przedziału w porównaniu do węzłów równomiernie rozmieszczonych.



Rysunek 1: Interpolacji oparta o węzły Czebyszewa



Rysunek 2: Interpolacja równomierna z zadania 8



Rysunek 3: Nałożone funkcje na siebie