Zadanie Numeryczne 11

Mateusz Wojtyna

1. Wstęp

Należało skonstruować naturalny splajn kubiczny dla funkcji

$$f(x) = \frac{1}{1+5x^2}, \quad x \in [-1.25, 1.25]$$

w węzłach Czebyszewa

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

2. Opis

2.1. Naturalny splajn kubiczny

Splajn kubiczny to wielomian interpolacyjny trzeciego stopnia, który oprócz wymogu przejścia przez podane węzły, musi również spełniać zadane wartości drugich pochodnych w węzłach. Zatem mamy tabelkę postaci

x_i	x_1	x_2	...	x_n
$f_i = f(x_i)$	f_1	f_2	...	f_n
ξ_i	ξ_1	ξ_2	...	ξ_n

gdzie ξ_i to pochodne drugiego stopnia wielomianu interpolacyjnego. **Naturalny** splajn kubiczny to splajn kubiczny z $\xi_1 = \xi_n = 0$.

2.2. Interpolacja

W każdym przedziale $\big[x_j, x_{j+1}\big]$, $j = 1, ..., n-1$ konstruujemy wielomian

$$y_j(x) = Af_j + Bf_{j+1} + C\xi_j + D\xi_{j+1}$$

gdzie

$$A = \frac{x_{j+1} - x}{x_{j+1} - x_j}, \quad B = \frac{x - x_j}{x_{j+1} - x_j}$$
$$C = \frac{1}{6}(A^3 - A)\left(x_{j+1} - x_j\right)^2, \quad D = \frac{1}{6}(B^3 - B)\left(x_{j+1} - x_j\right)^2$$

Jednak w rzeczywistości nie znamy wartości ξ_i. Korzystając z wymogu ciągłości pierwszej pochodnej $y_j(x)$ w węzłach otrzymujemy trójdiagonalny układ równań

$$\boldsymbol{A\xi} = \boldsymbol{b}$$

gdzie

$$\boldsymbol{A}_{j,j-1} = \frac{h_j}{6}, \quad \boldsymbol{A}_{j,j} = \frac{h_j + h_{j+1}}{3}, \quad \boldsymbol{A}_{j,j+1} = \frac{h_{j+1}}{6}, \quad \boldsymbol{A} \in \mathbb{R}^{(n-2)\times(n-2)}$$
$$\boldsymbol{\xi} = (\boldsymbol{\xi}_2, ..., \boldsymbol{\xi}_{n-1})^T, \quad \boldsymbol{\xi} \in \mathbb{R}^{n-2}$$
$$\boldsymbol{b}_j = \frac{f_{j+1} - f_j}{h_{j+1}} - \frac{f_j - f_{j-1}}{h_j}, \quad \boldsymbol{b} \in \mathbb{R}^{n-2}$$
$$h_j = x_j - x_{j-1}$$

Dzięki wykorzystaniu algorytmu Thomasa układ można rozwiązać w czasie $\mathcal{O}(n)$.

Teraz chcąc uzyskać wartość splajnu dla dowolnego x, musimy ustalić do którego przedziału x należy. Najszybciej można to zrobić w $\mathcal{O}(\log n)$ używając *binary search*. Wymaga to uporządkowania węzłów rosnąco.

3. Kod

Program napisano w Pythonie z użyciem pakietów *NumPy* oraz *Matplotlib*.

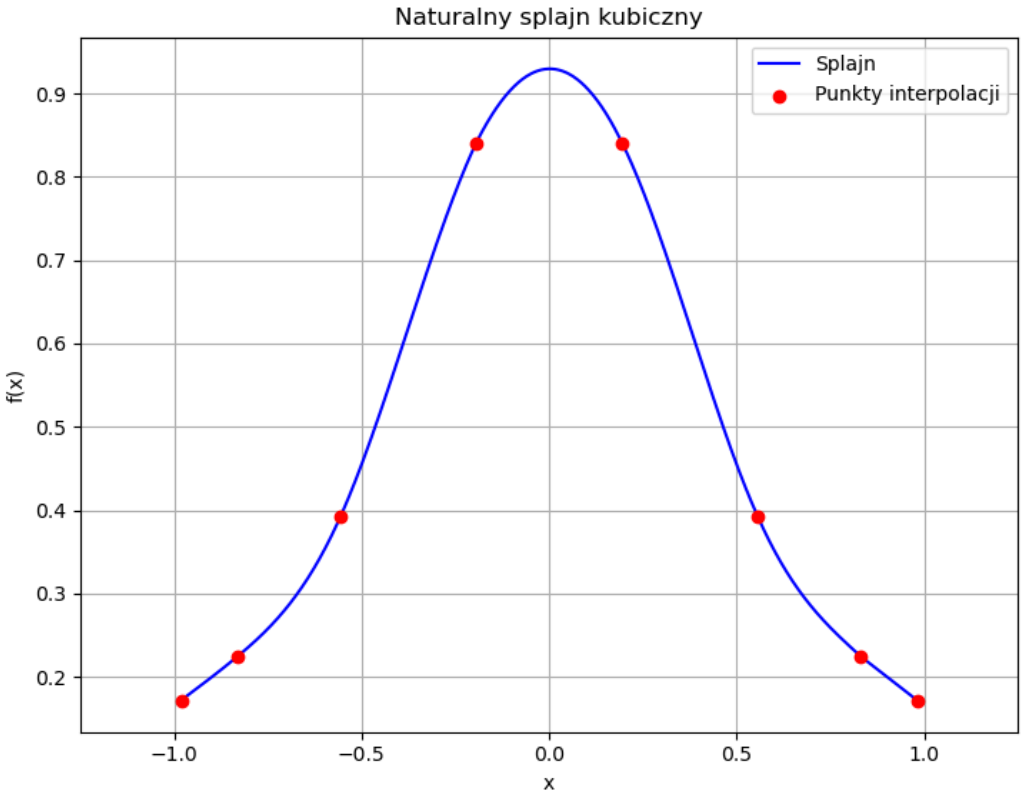
```
from bisect import bisect_left
import numpy as np
from numpy.typing import NDArray
import matplotlib.pyplot as plt

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



def calc_ksi(nodes: array, values: array) -> vector:
    n = len(nodes)

    # układ dla ksi_2, ..., ksi_(n-1)
    underdiag = np.zeros(n - 3)
```

```
    diag = np.zeros(n - 2)
    overdiag = np.zeros(n - 3)
    b = np.zeros(n - 2)

    h = np.diff(nodes)

    # i = 0
    diag[0] = (h[0] + h[1]) / 3
    overdiag[0] = h[1] / 6
    b[0] = (values[2] - values[1]) / h[1] - (values[1] - values[0]) / h[0]

    # i = 1, ..., n-4
    for i in range(1, n - 4 + 1):
        diag[i] = (h[i] + h[i + 1]) / 3
        underdiag[i - 1] = h[i] / 6
        overdiag[i] = h[i + 1] / 6

        b[i] = (values[i + 2] - values[i + 1]) / h[i + 1] - (
            values[i + 1] - values[i]
        ) / h[i]

    # i = n-3
    diag[n - 3] = (h[n - 3] + h[n - 2]) / 3
    underdiag[n - 4] = h[n - 3] / 6
    b[n - 3] = (values[n - 1] - values[n - 2]) / h[n - 2] - (
        values[n - 2] - values[n - 3]
    ) / h[n - 3]

    xi = np.zeros(n)
    xi[1 : n - 1] = thomas_solve(diag, underdiag, overdiag, b)

    return xi


def thomas_solve(diag: array, underdiag: array, overdiag: array, b: vector) -> vector:
    N = len(diag)
    x = np.zeros(N, dtype=np.float64)
    y = np.zeros(N, dtype=np.float64)
    u = np.zeros(N - 1, dtype=np.float64)

    l = diag[0]
    y[0] = b[0] / l
    for i in range(N - 1):
        u[i] = overdiag[i] / l
        l = diag[i + 1] - u[i] * underdiag[i]
        y[i + 1] = (b[i + 1] - underdiag[i] * y[i]) / l
```

```
    # backsubstitution
    x[-1] = y[-1]
    for i in range(N - 2, -1, -1):
        x[i] = y[i] - u[i] * x[i + 1]

    return x


def y_j(j: int, x: float, nodes: array, values: array, ksi: vector) -> float:
    a = (nodes[j + 1] - x) / (nodes[j + 1] - nodes[j])
    b = (x - nodes[j]) / (nodes[j + 1] - nodes[j])
    c = 1 / 6 * (a**3 - a) * (nodes[j + 1] - nodes[j]) ** 2
    d = 1 / 6 * (b**3 - b) * (nodes[j + 1] - nodes[j]) ** 2

    return a * values[j] + b * values[j + 1] + c * ksi[j] + d * ksi[j + 1]


def f(x: float):
    return 1 / (1 + 5 * x**2)


def sample(x: float, nodes: array, values: array, ksi: vector):
    N = len(nodes)
    j = bisect_left(nodes, x) - 1

    if j < 0:
        j = 0
    elif j > N - 2:
        j = N - 2

    return y_j(j, x, nodes, values, ksi)


def main():
    np.set_printoptions(suppress=True)

    nodes = np.array([np.cos(((2 * j - 1) / 16) * np.pi) for j in range(8, 0, -1)])
    values = np.array([f(x) for x in nodes])
    ksi = np.zeros(len(nodes))
    ksi = calc_ksi(nodes, values)

    x_plot = np.linspace(nodes[0], nodes[-1], 1000)
    y_plot = np.array([sample(x, nodes, values, ksi) for x in x_plot])
    plt.plot(x_plot, y_plot, label="Splajn", color="blue")
    plt.scatter(nodes, values, color="red", label="Punkty interpolacji", zorder=5)
```

```
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.title("Naturalny splajn kubiczny")
    plt.grid(True)
    plt.legend()
    plt.xlim(-1.25, 1.25)
    plt.show()


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

4. Wyniki



Rysunek 1: Naturalny splajn kubiczny funkcji f.
Węzły Czebyszewa nie są równoodległe.