

# Zadanie Numeryczne 10

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 równoodległych węzłach  $-\frac{7}{8}, -\frac{5}{8}, -\frac{3}{8}, -\frac{1}{8}, \frac{1}{8}, \frac{3}{8}, \frac{5}{8}, \frac{7}{8}$ .

## 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

$x_i$	$x_1$	$x_2$	$\dots$	$x_n$
$f_i = f(x_i)$	$f_1$	$f_2$	$\dots$	$f_n$
$\xi_i$	$\xi_1$	$\xi_2$	$\dots$	$\xi_n$

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

### 2.2. Interpolacja

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

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

gdzie

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

Jednak w rzeczywistości nie znamy wartości  $\xi_i$ . Korzystając z wymogu ciągłości pierwszej pochodnej  $y_j(x)$  w węzłach oraz z faktu, że węzły są równoodległe otrzymujemy trójdzielny symetryczny układ równań:

$$A\xi = b$$

gdzie

$$A_{j,j-1} = 1, \quad A_{j,j} = 4, \quad A_{j,j+1} = 1, \quad A \in \mathbb{R}^{(n-2) \times (n-2)}$$

$$\xi = (\xi_2, \dots, \xi_{n-1})^T, \quad \xi \in \mathbb{R}^{n-2}$$

$$b_j = \frac{6}{h^2}(f_{j-1} - 2f_j + f_{j+1}), \quad b \in \mathbb{R}^{n-2}$$

$h$  to stała odległość między węzłami

Dzięki wykorzystaniu faktoryzacji Cholesky'ego układ można rozwiązać w czasie  $\mathcal{O}(n)$ .

Następnie 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*.

### 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(values: array, h: float) -> vector:
    N = len(values)

    diag = np.full(N - 2, 4, dtype=np.float64)
    subdiag = np.full(N - 3, 1, dtype=np.float64)
    b = np.zeros(N - 2, dtype=np.float64)
    for i in range(N - 2):
        b[i] = (6 / h**2) * (values[i] - 2 * values[i + 1] + values[i + 2])

    C_diag, C_subdiag = cholesky_tridiagonal(diag, subdiag)
    ksi = np.zeros(N, dtype=np.float64)
    ksi[1 : N - 1] = solve_cholesky_tridiagonal(C_diag, C_subdiag, b)
```

```

    return ksi

def cholesky_tridiagonal(diag: array, subdiag: array) -> tuple[array, array]:
    """
    Zwraca (diagonala C, poddiagonala C)
    """

    N = len(diag)
    C_diag = np.zeros(N, dtype=np.float64)
    C_subdiag = np.zeros(N - 1, dtype=np.float64)

    C_diag[0] = np.sqrt(diag[0])
    for i in range(1, N):
        C_subdiag[i - 1] = subdiag[i - 1] / C_diag[i - 1]
        C_diag[i] = np.sqrt(diag[i] - C_subdiag[i - 1] ** 2)

    return C_diag, C_subdiag

def back_substitution_tridiagonal(diag: array, subdiag: array, b: vector) -> vector:
    N = len(diag)
    x = np.zeros(N, dtype=np.float64)

    x[-1] = b[-1] / diag[-1]
    for i in range(N - 2, -1, -1):
        x[i] = (b[i] - subdiag[i] * x[i + 1]) / diag[i]

    return x

def forward_substitution_tridiagonal(diag: array, subdiag: array, b: vector) -> vector:
    N = len(diag)
    x = np.zeros(N, dtype=np.float64)

    x[0] = b[0] / diag[0]
    for i in range(1, N):
        x[i] = (b[i] - subdiag[i - 1] * x[i - 1]) / diag[i]

    return x

def solve_cholesky_tridiagonal(C_diag: array, C_subdiag: array, b: vector) -> vector:
    #  $Ax=b \Leftrightarrow C(C^T x)=b$ ,  $C^T x=y$ ,  $Cy=b$ 
    y = forward_substitution_tridiagonal(C_diag, C_subdiag, b)

```

```

x = back_substitution_tridiagonal(C_diag, C_subdiag, y)
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([-7 / 8, -5 / 8, -3 / 8, -1 / 8, 1 / 8, 3 / 8, 5 / 8, 7 / 8])
    values = np.array([f(x) for x in nodes])
    ksi = calc_ksi(values, 2 / 8)

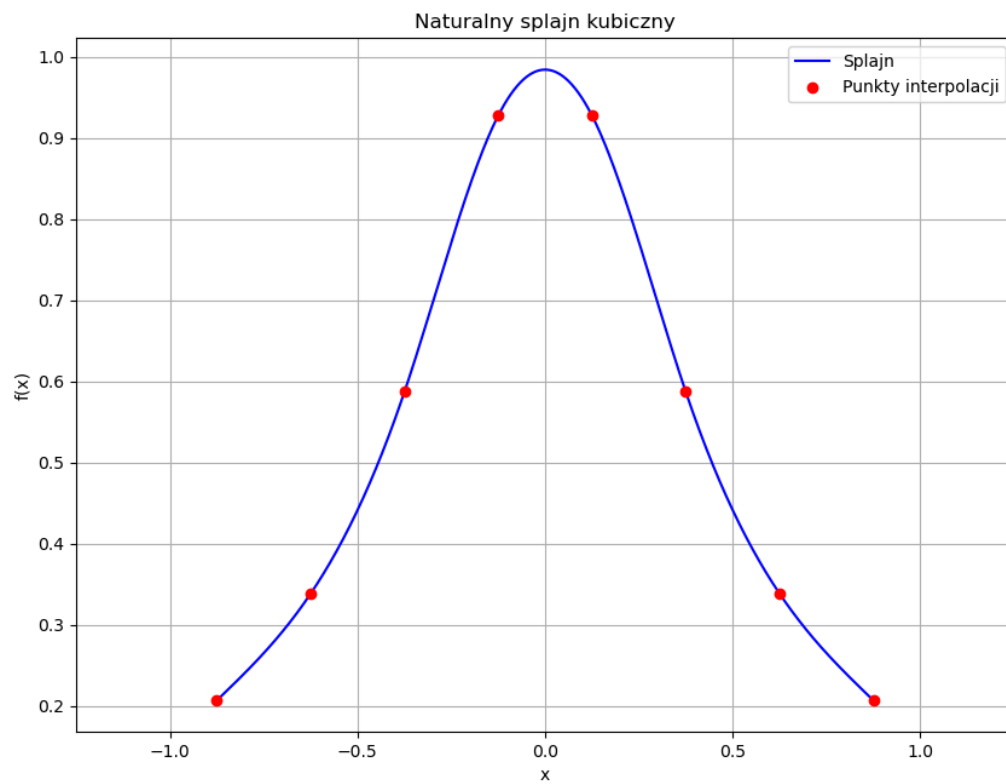
    x_plot = np.linspace(nodes[0], nodes[-1], 1000)
    y_plot = np.array([sample(x, nodes, values, ksi) for x in x_plot])
    plt.plot(x_plot, y_plot, label="Splajn", color="blue")
    plt.scatter(nodes, values, color="red", label="Punkty interpolacji", zorder=5)
    plt.xlabel("x")
    plt.ylabel("f(x)")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