Metody Numeryczne - Zadanie XI

Mateusz Kamiński

December 2025

1 Wprowadzenie

Celem zadania jest konstrukcja naturalnego splajnu kubicznego interpolującego funkcję

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

w przedziale

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

z wykorzystaniem nierównoodległych węzłów Czebyszewa.

2 Postać rozwiązywanego układu równań

Dla naturalnego splajnu kubicznego wyznaczane są wartości drugich pochodnych

$$\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_{n-2})^T,$$

przy warunkach brzegowych

$$\xi_0 = 0, \quad \xi_{n-1} = 0.$$

W przypadku nierównoodległych węzłów interpolacji $x_0, x_1, \ldots, x_{n-1}$, gdzie odległość między węzłami się zmienia

$$h_j = x_{j+1} - x_j,$$

otrzymujemy niestały trójdiagonalny układ równań liniowych

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

o współczynnikach

$$A_{j,j-1} = h_{j-1}, \quad A_{j,j} = 2(h_{j-1} + h_j), \quad A_{j,j+1} = h_j,$$

oraz

$$b_j = 6\left(\frac{f_{j+1}-f_j}{h_j} - \frac{f_j - f_{j-1}}{h_{j-1}}\right), \quad j = 1, \ldots, n-2.$$

Układ równań dla niewiadomych ξ_j w postaci poglądowej można zapisać jako

$$\begin{pmatrix} 2(h_0+h_1) & h_1 & 0 & \cdots & 0 \\ h_1 & 2(h_1+h_2) & h_2 & \ddots & \vdots \\ 0 & h_2 & 2(h_2+h_3) & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & h_{n-2} \\ 0 & \cdots & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \vdots \\ \xi_{n-1} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix},$$

Układ ten rozwiązywany jest za pomocą algorytmu Thomasa.

3 Wyznaczanie współczynników splajnu

Po wyznaczeniu wartości drugich pochodnych $\boldsymbol{\xi}$, wartości splajnu w dowolnym punkcie $x \in [x_j, x_{j+1}]$ obliczane są ze wzoru

$$P(x) = A_j f_j + B_j f_{j+1} + C_j \xi_j + D_j \xi_{j+1},$$

gdzie

$$A_j = \frac{x_{j+1}-x}{h_j}, \quad B_j = \frac{x-x_j}{h_j}, \quad h_j = x_{j+1} - x_j,$$

$$C_j = \frac{h_j^2}{6}(A_j^3 - A_j), \quad D_j = \frac{h_j^2}{6}(B_j^3 - B_j).$$

4 Węzły Czebyszewa

Zastosowanie węzłów Czebyszewa z czynnikiem (-1) (odwrócenie kolejności węzłów) nie zmienia własności interpolacyjnych splajnu, a jedynie porządkuje węzły rosnąco na przedziale $[a, b]$, co jest wymagane przy konstrukcji splajnu oraz wyznaczaniu odcinków $[x_j, x_{j+1}]$.

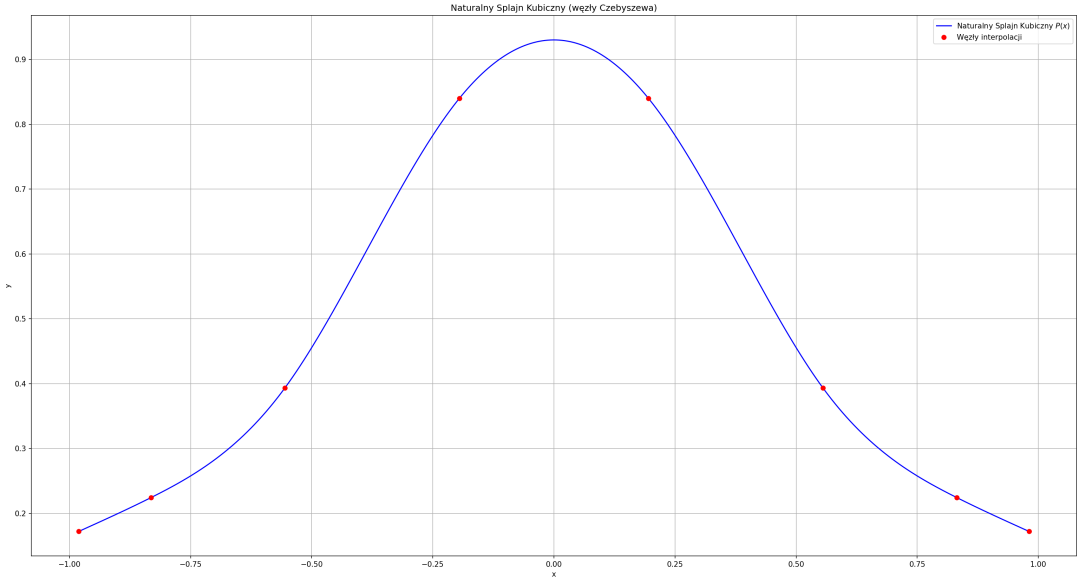
5 Implementacja

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

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

def thomas_factor(subdiagonal: vector, diagonal: vector) -> tuple[vector,
    vector]:
    n = len(diagonal)
    L = np.zeros(n - 1)

    for i in range(1, n):
        L[i - 1] = subdiagonal[i - 1] / diagonal[i - 1]
        diagonal[i] -= L[i - 1] * subdiagonal[i - 1]

    return L, diagonal

def thomas_solve(L: vector, U: vector, b: vector, superdiagonal: vector) ->
     vector:
    n = len(U)

    # forward
    for i in range(1, n):
        b[i] -= L[i - 1] * b[i - 1]

    # back
    x = np.zeros(n)
    x[-1] = b[-1] / U[-1]
    for i in range(n - 2, -1, -1):
        x[i] = (b[i] - superdiagonal[i] * x[i + 1]) / U[i]

    return x

def get_spline_coefficients(x_val: num, x_j: num, x_j1: num) -> tuple[num,
    num, num, num]:
    h = x_j1 - x_j

    A = (x_j1 - x_val) / h
    B = (x_val - x_j) / h

    h_squared = h * h

    # x_{j+1} - x{j} = h
    C = (1.0 / 6.0) * (A ** 3 - A) * h_squared
    D = (1.0 / 6.0) * (B ** 3 - B) * h_squared

    return A, B, C, D
```

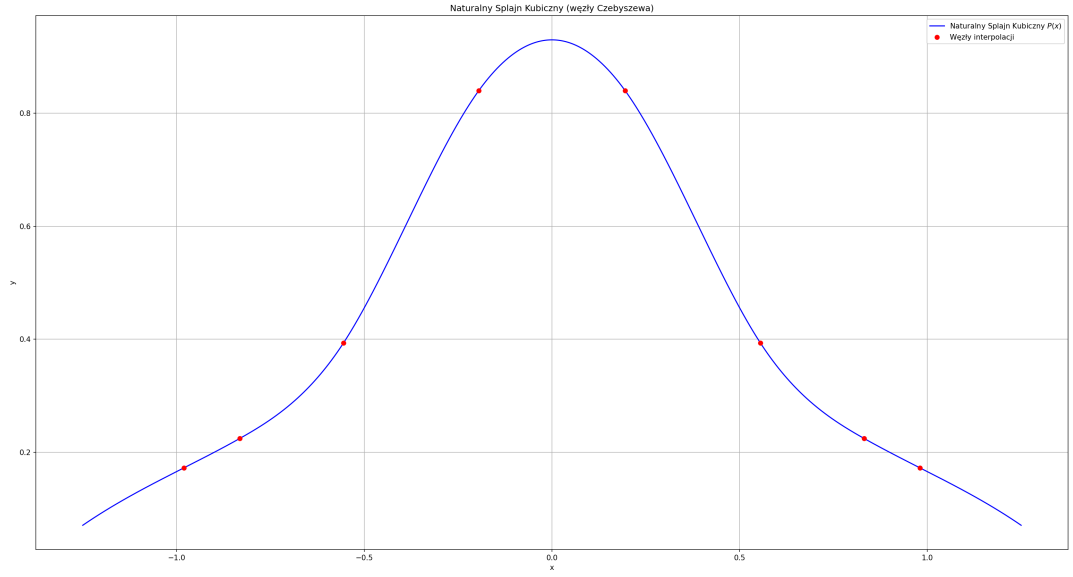
```
def spline_function(x_val: num, x: vector, f: vector, ksi: vector) -> num:
    j = np.searchsorted(x, x_val) - 1
    j = max(0, min(j, len(x) - 2))

    A, B, C, D = get_spline_coefficients(x_val, x[j], x[j + 1])

    P_x = A * f[j] + B * f[j + 1] + C * ksi[j] + D * ksi[j + 1]
    return P_x


def plot_results(x: vector, f: vector, ksi: vector):
    # Generowanie punktów do rysowania splajnu
    x_range = np.linspace(-1.25, 1.25, 1000)

    # Obliczanie wartości splajnu w punktach x_range
    s_spline = np.zeros_like(x_range)
    for k, val in enumerate(x_range):
        s_spline[k] = spline_function(val, x, f, ksi)

    plt.figure(figsize=(10, 6))
    plt.plot(x_range, s_spline, label='Naturalny Splajn Kubiczny $P(x)$',
        color='blue')
    plt.plot(x, f, 'o', label='Węzły interpolacji', color='red')
    plt.title('Naturalny Splajn Kubiczny (węzły Czebyszewa)')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid(True)
    plt.legend()
    plt.show()

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])

    # długość każdego segmentu h[i] = x[i+1] - x[i]
    h = np.diff(x)
```

```
    diagonal: vector = np.zeros(n - 2, dtype=num)
    subdiagonal: vector = np.zeros(n - 3, dtype=num)
    superdiagonal: vector = np.zeros(n - 3, dtype=num)

    b: vector = np.zeros(n - 2, dtype=num)

    for i in range(n - 3):
        diagonal[i] = 2 * (h[i] + h[i + 1])
        subdiagonal[i] = h[i + 1]
        superdiagonal[i] = h[i + 1]
        b[i] = 6 * ((f[i + 2] - f[i + 1]) / h[i + 1] - (f[i + 1] - f[i]) /
            h[i])

    diagonal[-1] = 2 * (h[-2] + h[-1])
    b[-1] = 6 * ((f[-1] - f[-2]) / h[-1] - (f[-2] - f[-3]) / h[-2])

    L, U = thomas_factor(subdiagonal, diagonal)
    ksi_without_zero = thomas_solve(L, U, b, superdiagonal)

    ksi = np.zeros(n, dtype=num)
    ksi[1:n-1] = ksi_without_zero

    plot_results(x, f, ksi)

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

6 Wyniki



Rysunek 1: Splajn kubiczny w granicach od pierwszego punktu do ostatniego



Rysunek 2: Splajn kubiczny w granicach [-1.25, 1.25]