

Metody Numeryczne - Zadanie X

Mateusz Kamiński

December 2025

1 Wprowadzenie

Celem zadania jest konstrukcja naturalnego splajnu kubicznego interpolującego zadaną funkcję oraz węzłami z zadania 8.

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

w przedziale

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

2 Postać rozwiązywanego układu równań

Dla naturalnego splajnu kubicznego wyznaczane są wartości drugich pochodnych

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

przy warunkach brzegowych

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

Prowadzi to do rozwiązywania układu liniowego

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

gdzie macierz $A \in \mathbb{R}^{(n-2) \times (n-2)}$ ma postać trójdiamondalną:

$$A = \begin{pmatrix} 4 & 1 & 0 & \cdots & 0 \\ 1 & 4 & 1 & \ddots & \vdots \\ 0 & 1 & 4 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & 1 & 4 \end{pmatrix}.$$

Wektor prawej strony dany jest wzorem

$$b_i = \frac{6}{h^2} (f_{i+2} - 2f_{i+1} + f_i), \quad i = 0, 1, \dots, n-3,$$

gdzie h oznacza stały krok, ponieważ punkty interpolacji są równomiernie rozłożone.

Do rozwiązyania układu równań $A \xi = b$, używamy faktoryzacji Choleskiego wraz z forward oraz back substitution.

3 Wyznaczanie współczynników splajnu

Po wyznaczeniu wartości drugich pochodnych ξ 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

$$\begin{aligned} A_j &= \frac{x_{j+1} - x}{h}, & B_j &= \frac{x - x_j}{h}, & h &= x_{j+1} - x_j \\ C_j &= \frac{h^2}{6}(A_j^3 - A_j), & D_j &= \frac{h^2}{6}(B_j^3 - B_j), \end{aligned}$$

4 Implementacja

```
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 cholesky_tridiagonal(diagonal: matrix, subdiagonal: matrix) -> tuple[
    matrix, matrix]:
    n = diagonal.shape[0]
    C_diag = np.zeros(n, dtype=np.float64)
    C_subdiag = np.zeros(n - 1, dtype=np.float64)

    C_diag[0] = np.sqrt(diagonal[0])
    for i in range(1, n):
        C_subdiag[i - 1] = subdiagonal[i - 1] / C_diag[i - 1]
        C_diag[i] = np.sqrt(diagonal[i] - C_subdiag[i - 1]**2)

    return C_diag, C_subdiag
```

```

def forward_substitution_tridiagonal(diagonal: matrix, subdiagonal: matrix,
    b: matrix) -> vector:
    n = b.shape[0]
    y = np.zeros(n, dtype=np.float64)

    y[0] = b[0] / diagonal[0]
    for i in range(1, n):
        y[i] = (b[i] - subdiagonal[i - 1] * y[i - 1]) / diagonal[i]

    return y

def backward_substitution_tridiagonal(diagonal: matrix, subdiagonal: matrix
    , y: matrix) -> vector:
    n = y.shape[0]
    z = np.zeros(n, dtype=np.float64)

    z[-1] = y[-1] / diagonal[-1]
    for i in range(n - 2, -1, -1):
        z[i] = (y[i] - subdiagonal[i] * z[i + 1]) / diagonal[i]

    return z

def get_spline_coefficients(x_val: num, x_j: num, h: num) -> tuple[num, num
    , num, num]:
    #  $x_{j+1} - x_j = h$ 
    #  $x_{j+1} = h + x_j$ 

    A = (x_j + h - 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, h: num)
-> num:
    n = len(x)
    j = np.searchsorted(x, x_val) - 1
    if j < 0:
        j = 0
    if j >= n - 1:
        j = n - 2

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

```

```

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, h: num):
    # Generowanie punktów do rysowania splajnu
    x_range = np.linspace(x[0], x[-1], 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, h)

    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')
    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 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])

    diagonal: vector = np.full(n - 2, 4.0)
    subdiagonal: vector = np.full(n - 3, 1.0)

    # stałe h
    h = x[1] - x[0]
    b: vector = np.zeros(n - 2, dtype=num)
    for i in range(n - 2):
        b[i] = (6.0 / (h * h) * (f[i + 2] - 2*f[i + 1] + f[i]))

    C_diag, C_subdiag = cholesky_tridiagonal(diagonal, subdiagonal)
    y = forward_substitution_tridiagonal(C_diag, C_subdiag, b)
    ksi_without_zero = backward_substitution_tridiagonal(C_diag, C_subdiag,
                                                          y)

```

```

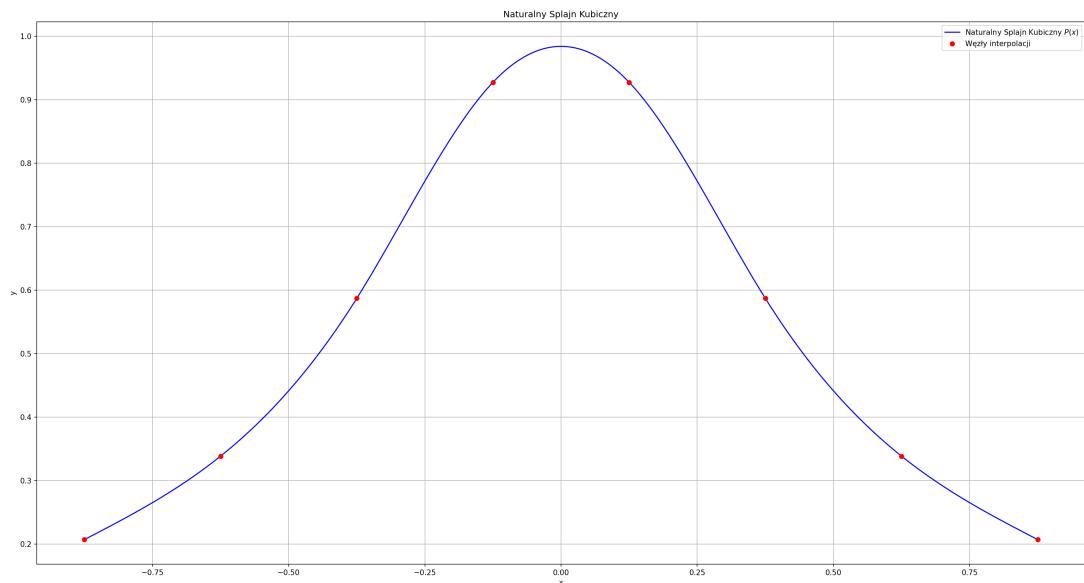
ksi = np.zeros(n, dtype=num)
ksi[1:n-1] = ksi_without_zero

plot_results(x, f, ksi, h)

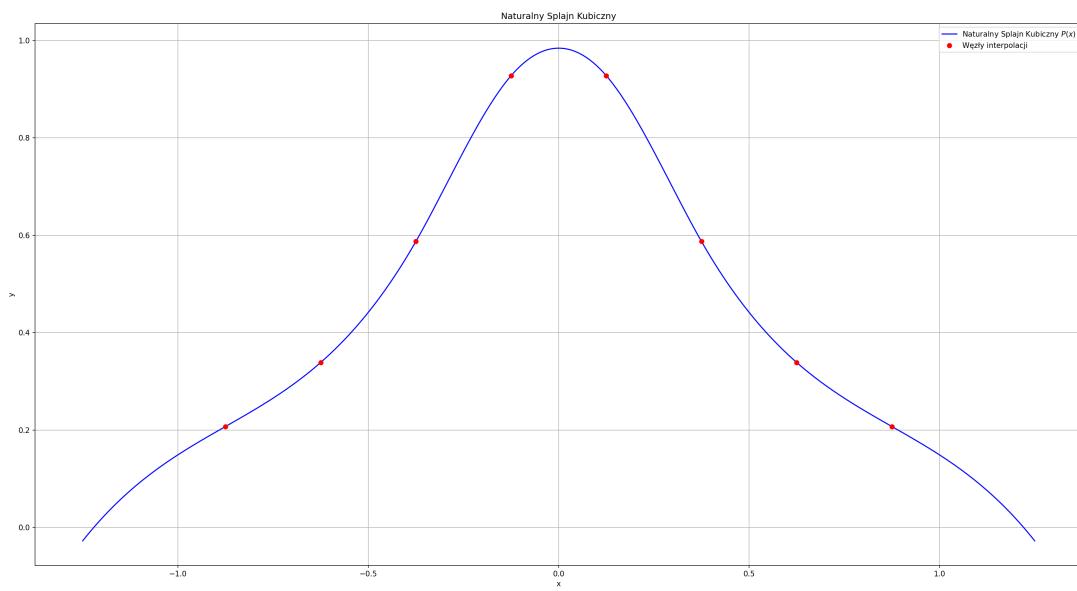
if __name__ == "__main__":
    main()

```

5 Wyniki



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



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