Metody numeryczne

Mateusz Kwolek

10. Skonstruuj naturalny splajn kubiczny dla funkcji i węzłów z zadania 8. Sporządź jego wykres.

8. Znajdź wartości funkcji

$$f(x) = \frac{1}{1 + 5x^2} \tag{6}$$

w punktach $-7/8, -5/8, -3/8, -1/8, 1/8, 3/8, 5/8, 7/8$ a następnie skonstruuj wielomian interpolacyjny Lagrange'a oparty na tych węzłach i wartościach funkcji (6) w tych węzłach. Narysuj wykres wielomianu interpolacyjnego. w przedziale $[-1.25, 1.25]$, zaznaczając na nim węzły i wartości w węzłach. Punkt dodatkowy: znajdź *współczynniki* wielomianu interpolacyjnego.

1) Algorytm

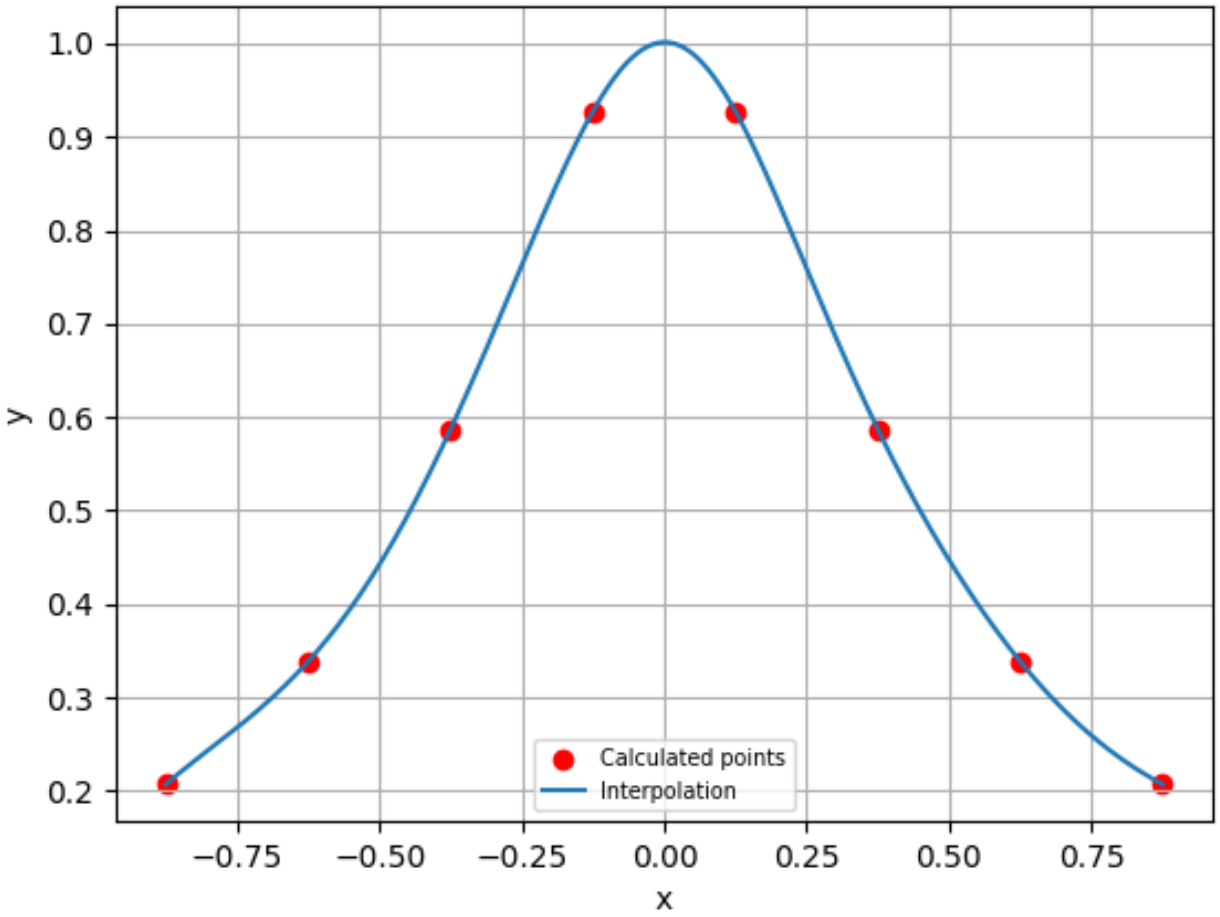
Algorytmem, który zaimplementowałem na potrzeby wykonania zadania jest interpolacja Hermite'a. Obejmuje ona funkcje, która nie tylko przechodzi przez dane punkty, ale także uwzględnia wartości pochodnych w tych punktach. Dzięki temu zabiegowi otrzymane obliczenia są jeszcze dokładniejsze.

2) Kod źródłowy

Do stworzenia programu użyłem języka Python ze względu na jego prostotę, bogactwo bibliotek oraz częste zastosowanie do wykonywania obliczeń matematycznych.
Do wykonania wykresu tempa zbieżności metod użyłem biblioteki matplotlib.
Po uruchomieniu programu zostanie wyświetlone okno z wykresem gdzie poruszając się możemy dokładnie przyjrzeć się wynikom. Te zostaną również zapisane w pliku png.

```python
import numpy as np
import matplotlib.pyplot as plt

# Zadane punkty funkcji
x_array = np.array([-7 / 8, -5 / 8, -3 / 8, -1 / 8, 1 / 8, 3 / 8, 5 / 8, 7 / 8])

# Formowanie funkcji
def function(x):
    return 1 / (1 + 5 * x * x)

# Obliczenie wartości funkcji w punktach x
def calculate_y_array(x_array):
    y_array = []
    for x in x_array:
        y_array.append(function(x))

    return y_array

# Obliczenie wag dla interpolacji Hermite'a
# uwzględniającą wartości funkcji w punktach x
# oraz wartości pochodnych funkcji w tych punktach
def calculate_weights(size, d=3):
    weights = []
    factorials = {}

    def get_factorial(n):
        if n == 0 or n == 1:
            return 1

        if n in factorials:
            return factorials[n]

        else:
            fac = n * get_factorial(n - 1)
            factorials[n] = fac
            return fac

    fac_d = get_factorial(d)

    for k in range(size + 1):
        sign = 1

        if (k - d) % 2 == 1:
            sign = -1

        summation = 0
        for i in range(max(0, k - d), min(size - d + 1, k + 1)):
            summation += fac_d / (get_factorial(k - i) * get_factorial(d - k + i))

        weight = sign * summation
        weights.append(weight)

    return weights

# Interpolacja Hermite'a
def fh_interpolation_function(x, weights, x_array, y_array):
    if x in x_array:
        return function(x)

    denominator = 0
    numerator = 0
    iterations = len(x_array)
    for k in range(iterations):
        fraction = weights[k] / (x - x_array[k])
        denominator += fraction
        numerator += fraction * y_array[k]

    r = numerator / denominator

    return r

y_array = calculate_y_array(x_array)
weights = calculate_weights(len(x_array))

# Konstrukcja wykresu
plt.scatter(x_array, y_array, color="RED", label="Calculated points")
x_plt = np.linspace(min(x_array), max(x_array), 1000)
y_plt = [fh_interpolation_function(x, weights, x_array, y_array) for x in x_plt]

plt.plot(x_plt, y_plt, label=f"Interpolation")
plt.xlabel("x")
plt.ylabel("y")
plt.grid()
plt.legend(loc='lower center', fontsize='x-small')

plt.savefig('wykres.png')
plt.show()
```

3) Wykres



4) Wykres funkcji z zadania 8