Ćwiczenie 2b

Ewelina Badeja

1. Treść zadania (ćwiczenie 2b skupia się na drugim punkcie)

1. Dla jednej z poniższych funkcji (*podanej w zadaniu indywidualnym*) wyznacz dla zagadnienia Lagrange'a wielomian interpolujący w postaci Lagrange'a i Newtona.

   Interpolację przeprowadź dla różnej liczby węzłów (np. n = 3, 4, 5, 7, 10, 15, 20). Dla każdego przypadku interpolacji porównaj wyniki otrzymane dla różnego rozmieszczenia węzłów: równoodległe oraz Czebyszewa*.

   Oceń dokładność, z jaką wielomian przybliża zadaną funkcję.

   Poszukaj wielomianu, który najlepiej przybliża zadaną funkcję.

   Wyszukaj stopień wielomianu, dla którego można zauważyć efekt Runge'go (dla równomiernego rozmieszczenia węzłów). Porównaj z wyznaczonym wielomianem dla węzłów Czebyszewa.

2. Podobną analizę przeprowadź dla zagadnienia Hermite'a.

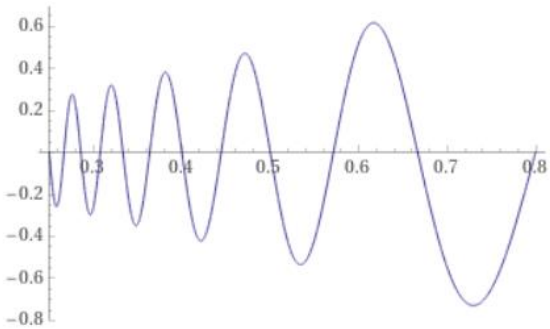
Funkcja, dla której zostały wykonane obliczenia:

1. $f(x) = x \cdot \sin\left(\frac{k\pi}{x}\right)$

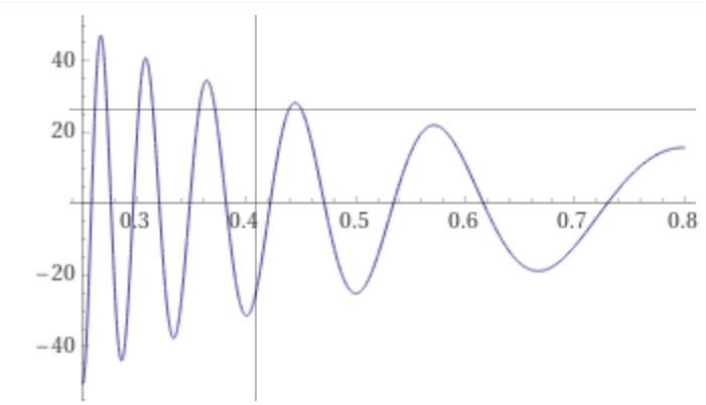
Parametry funkcji:

k=4, [1/4, 0.8]

(W nawiasie kwadratowym podano przedział x). Funkcja i jej pochodna narysowane z pomocą programu wolfram alfa prezentują się tak:



Wykres 1, dana funkcja



Wykres 2, pochodna funkcji

2. **Zmiany w kodzie źródłowym, w stosunku do zadania 2a**

W stosunku do zadania 2a zmieniona została funkcja do obliczania współczynników wzorem Newtona *(Fragment kodu 1)*, żeby móc wykorzystywać informacje o pochodnych. Założyłam, że mamy informacje o wartości funkcji i pochodnej w każdym węźle.

```
def wspolczynniki_newtona(xn, yn):
    #tworzenie tabeli do przechowywania różnic
    tab = [[0 for _ in range(len(xn) * 2 + 1)] for pom in range(len(xn) * 2)]
    x = copy(xn)
    y = copy(yn)

    #wypełnianie tabeli wartościami x, f(x) i f'(x)
    for i in range(0, len(xn) * 2, 2):
        tab[i][0] = x[int(i / 2)]
        tab[i + 1][0] = x[int(i / 2)]
        tab[i][1] = y[int(i / 2)]
        tab[i + 1][1] = y[int(i / 2)]
        tab[i + 1][2] = pochodna(x[int(i / 2)])

    #obliczanie różnic
    for i in range(2, len(xn) * 2 + 1):
        for j in range(i - 1, len(xn) * 2):
            if i != 2 or j % 2 == 0:
                tab[j][i] = (tab[j][i - 1] - tab[j - 1][i - 1]) / (tab[j][0] - tab[abs(i - 1 - j)][0])

    #pobieranie współczynników wielomianu
    wspolczynniki = []
    for i in range(len(xn) * 2):
        wspolczynniki.append(tab[i][i + 1])

    return wspolczynniki
```

Fragment kodu 1

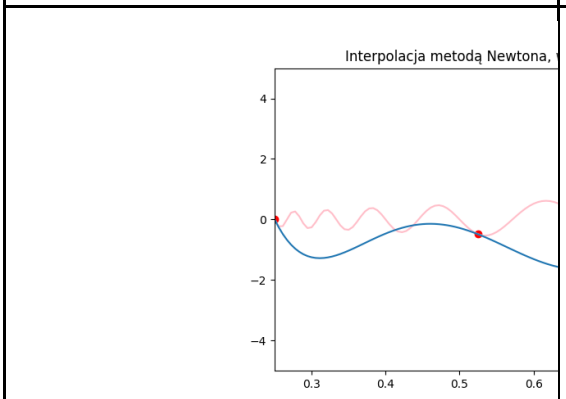
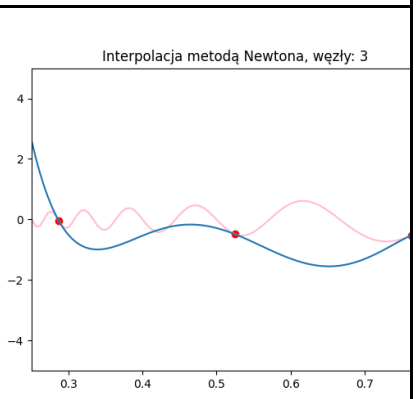
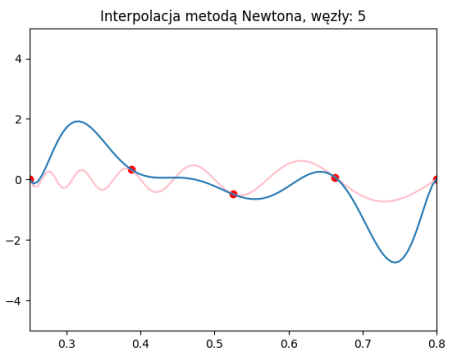
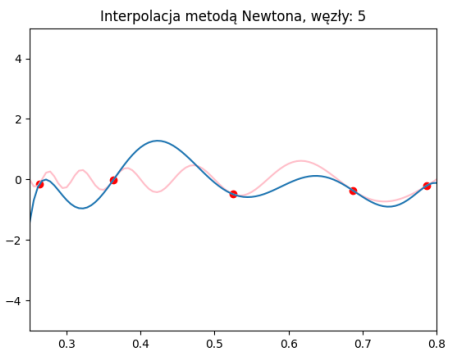
Dodana została implementacja uprzednio obliczonej pochodnej *(Fragment kodu 2)*.

```
def pochodna(x):
    return sin((pi * 4) / x) - (4 * pi * cos((pi * 4) / x)) / x
```

Fragment kodu 2

3. Porównanie rozmieszczenia równo odległego i Czebyszewa

Poniżej zestawienie wykresów dla obu rozstawień dla interpolacji Hermite'a. Dla każdego węzła jest liczona również jego pochodna, więc dla n węzłów mamy 2n informacji, zatem stopień wielomianu jest większy niż liczba węzłów. Na wykresach czerwone kropki to węzły, na różowo zaznaczono wykres danej funkcji, a na niebiesko wielomian, liczony w 100 punktach.

Równoodległe	Czebyszewa
	
	

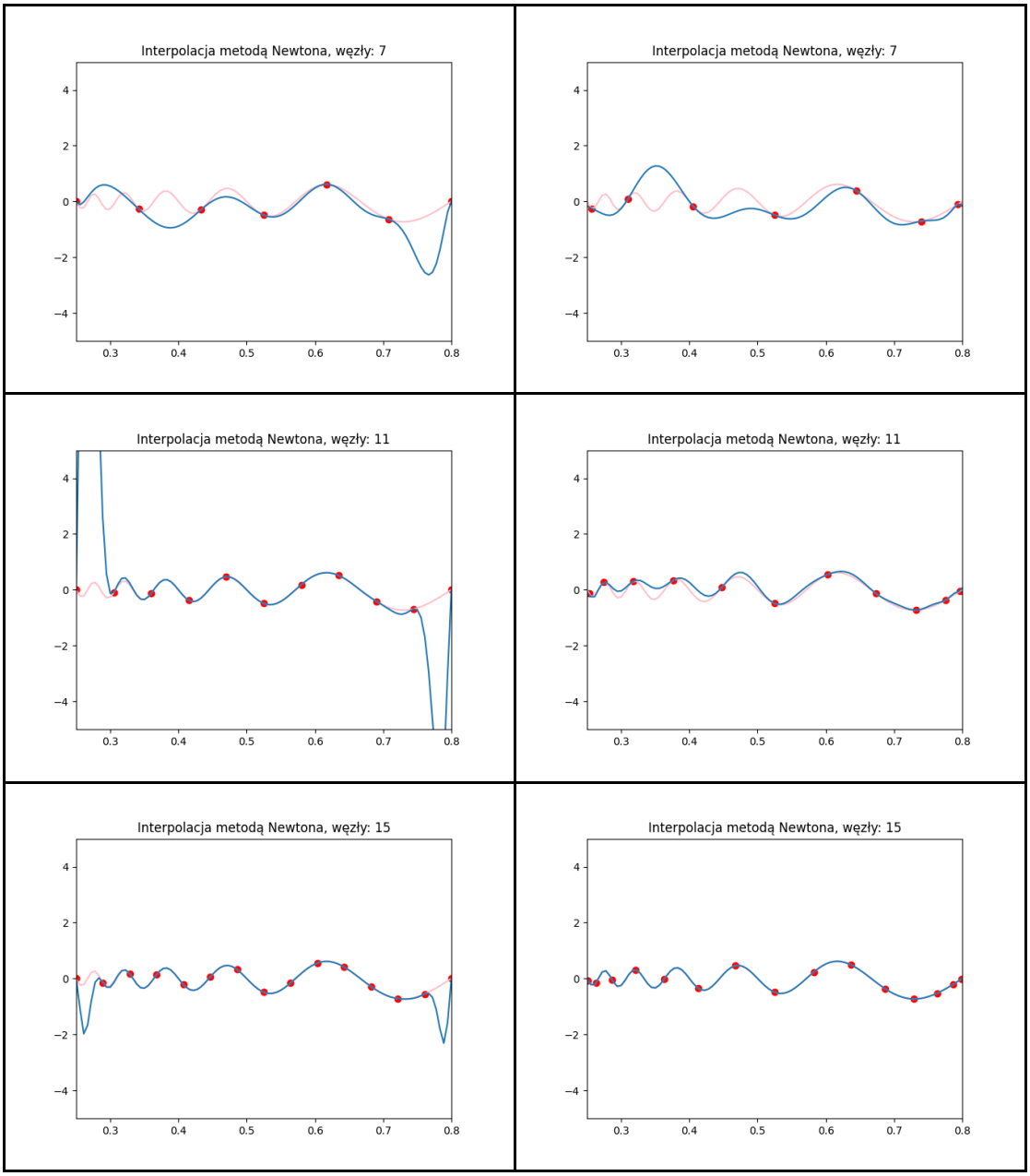


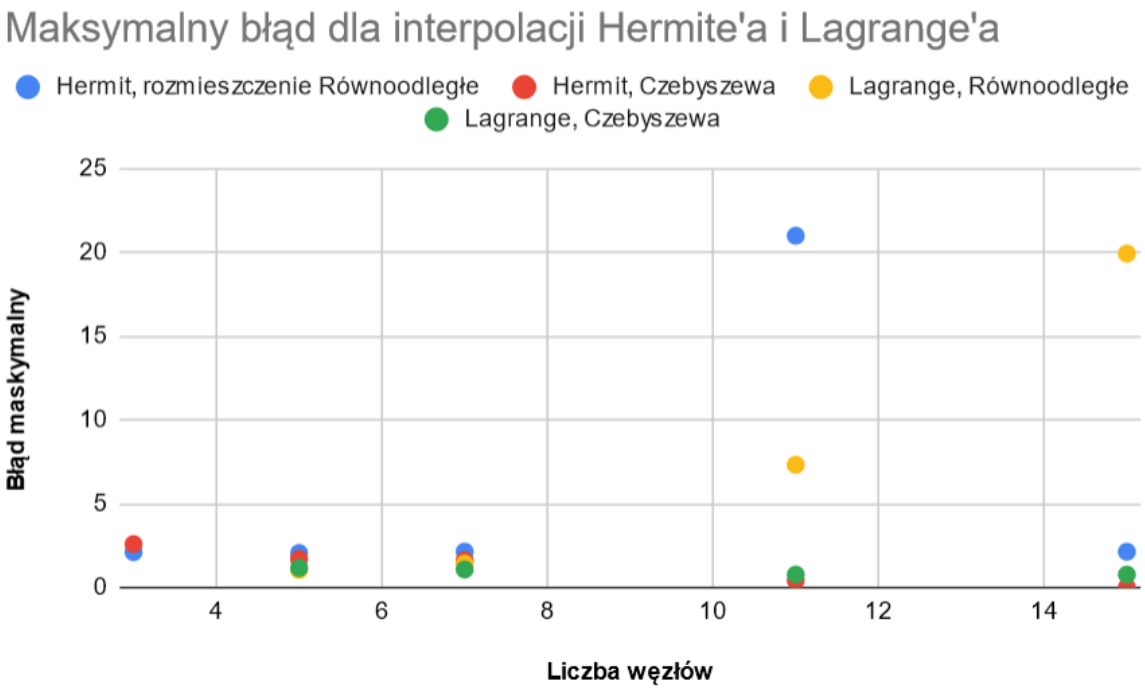
Tabela 1

Interpolacja Hermite'a		
Węzły	**Równo odległe**	**Czebyszewa**
3	2.1135568040618464	2.593397113460046
5	2.0600910495619673	1.7007215160511207
7	2.1440337972291834	1.616696760680811
11	20.99623179473233	0.421639389242507
15	2.1361400467973555	0.01697720776412609

Tabela 2, błąd maksymalny dla interpolacji Hermite'a

Interpolacja Lagrange’a				
	Równo odległe		Czebyszewa	
Węzły	**Lagrange**	**Newton**	**Lagrange**	**Newton**
5	1.050518778526849	1.050518778526849	1.166451895431842	1.166451895431842
7	8.24214205215318	1.4266029830381017	1.0775864211317023	1.077586421131703 7
11	7.325153196794687	7.325153196794691	0.7714149756231838	0.771414975623190 2
15	19.93581532663308	19.935815326633655	0.7730370224734595	0.773037022473372 9

Tabela 3, błąd maksymalny dla interpolacji Lagrange’a



Wykres 3, porównanie błędów dla obu metod (obie metody korzystają ze wzoru Newtona)

Obserwacje:

- Najlepsze wyniki dawała metoda Hermite’a z rozmieszczeniem Czebyszewa
- Pomimo 2 razy większej ilości danych, dla mniejszej liczby węzłów (5 lub 7) interpolacja Hermite’a dawała gorsze wyniki niż interpolacja Lagrange’a (*Tabela 2* i *Tabela 3*)

- Interpolacja Hermite’a jest dużo bardziej wrażliwa na rozmieszczenie (z rozmieszczeniem Czebyszewa daje rekordowo dokładne wyniki, a z rozmieszczeniem równoodległym rekordowo duże błędy, tj. 20.99623179473233 dla 11 węzłów)

4. Najlepiej przybliżony wielomian

Wielomian z najmniejszym błędem maksymalnym dla liczby węzłów z zakresu <3,15> miał błąd maksymalny 0.01697720776412609 (rozmieszczenie Czebyszewa, 15 węzłów. Wykres znajduje się w *Tabeli 1).*

5. Efekt Runge’go dla metody Hermite’a

Efekt ten udało się zaobserwować licząc dla 7 i 11 węzłów z rozmieszczeniem równoodległym (*Tabela 1*). Dla 7 węzłów błąd był ponad dziesięciokrotnie mniejszy niż dla 11 węzłów. Co ciekawe, dla interpolacji Lagrange’a błąd dla 7 węzłów błąd był pięciokrotnie mniejszy niż dla 11 węzłów.

Efektu tego nie zaobserwowałam licząc z rozmieszczeniem Czebyszewa. Wraz ze wzrostem liczby węzłów błąd maleje.

6. Końcowe wnioski

Dla większej liczby węzłów metoda Hermite’a daje lepsze wyniki i komputer nie potrzebuje wcale dużo więcej czasu na obliczanie wyników tą metodą, więc jeśli tylko mamy informacje o pochodnych, warto je wykorzystać.

Sporą zaletą metody Hermite’a jest to, że dla danej liczby węzłów wykres interpolowanego wielomianu optycznie dużo bardziej przypomina wykres danej funkcji. Pomaga to (nawet dla mniejszej liczby węzłów) łatwiej wyobrazić sobie, jak oryginalna funkcja może się zachowywać.

7. Dane techniczne

- Język programowania: Python 3.10
- Środowisko: PyCharm Professional
- System: Windows Server 2019
- Procesor: Intel Core i7-7700HQ 2.8 GHz