Ćwiczenie 2

Ewelina Badeja

1. Treść zadania

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.

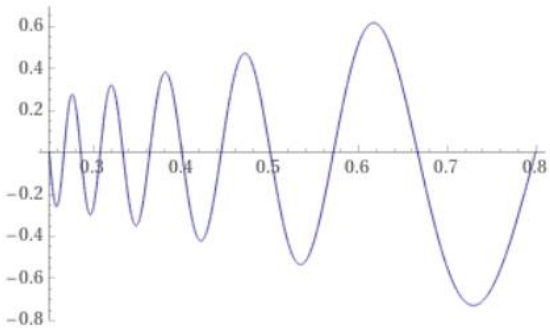
Funkcja, dla której zostały wykonane obliczenia:

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 narysowana z pomocą programu wolfram alfa prezentuje się tak:



Wykres 1

2. Kod źródłowy

Tworząc program najpierw napisałam funkcje do interpolowania metodami Lagrange'a i Newtona. Z obliczaniem współczynników w metodzie Newtona miałam problemy (błąd w kodzie, którego nie byłam w stanie znaleźć), więc 2 linijki wzięłam z internetu (cała reszta jest mojego autorstwa).

```python
def Lk(x, k, xn):
    iloczyn = 0
    for i in range(len(xn)):
        if i != k:
            if iloczyn == 0:
                iloczyn = (x - xn[i]) / (xn[k] - xn[i])
            else:
                iloczyn *= (x - xn[i]) / (xn[k] - xn[i])
    return iloczyn


def interpolacja_lagranga(x, xn, yn):
    Pn = 0
    for k in range(len(xn)):
        Pn += yn[k] * Lk(x, k, xn)
    return Pn
```

Fragment kodu 1

```python
def wspolczynniki_newtona(xn, yn):

    m = len(xn)
    x = copy(xn)
    a = copy(yn)
    for k in range(1, m):
        a[k:m] = (a[k:m] - a[k - 1]) / (xn[k:m] - xn[k - 1])
    return a


def interpolacja_newtona(x, xn, yn):
    tab = wspolczynniki_newtona(xn, yn)
    n = len(xn) - 1
    wynik = tab[n]

    # Schemat Hornera
    for k in range(1, n + 1):
        wynik = tab[n - k] + (x - xn[n - k]) * wynik
    return wynik
```

Fragment kodu 2

Następnie wprowadziłam możliwość doboru liczby węzłów oraz metody ich rozmieszczania (równoodległe albo Czebyszewa).

```
wezly = int(input("Podaj ilosc wezlow:\n"))
rozmieszczenie = input("Podaj sposób rozmieszczenie:(C - czebyszewa, cokolwiek innego - liniowy)\n")

xn = linspace(0.25, 0.8, wezly)
if rozmieszczenie == "C":
    for m in range(wezly):
        xn[m] = 0.5 * ((0.8 - 0.25) * cos(pi * (2 * m + 1) / (2 * (wezly - 1) + 2)) + (0.8 + 0.25))

yn = [funkcja(xn[i]) for i in range(wezly)]
x = linspace(0.25, 0.8, 100)
y = [0 for _ in range(100)]
y_f = [funkcja(x[i]) for i in range(100)]
```

Fragment kodu 3

Węzły Czebyszewa były obliczane według wzoru:

$$x_m = \frac{1}{2}\left[(b-a)\cos\frac{2m+1}{2n+2}\pi)+(b+a)\right],\ m=0,1,2,\ldots,n$$

Kolejnym krokiem była wizualizacja wyników. Na wykresach czerwone kropki oznaczają węzły, różowy wykres to dana funkcja, a niebieski to otrzymana funkcja wielomianowa. Funkcja była rysowana w 100 punktach.

```
# rysowanie wykresu dla interpolacji metodą Newtona
for i in range(100):
    y[i] = interpolacja_newtona(x[i], xn, yn)
plt.plot(x, y)
plt.xlim([0.25, 0.8])
plt.ylim([-5, 5])
plt.plot(x, y_f, color="pink")
plt.scatter(xn, yn, color="red")
plt.title("Interpolacja metodą Newtona, " + str(wezly) + " węzłów")
plt.show()
print("N:", max_blad(x, y))
```

Fragment kodu 4

(Rysowanie wykresów dla metody Lagrange'a wygląda tożsamo). Ostatnim krokiem było wyliczanie maksymalnego błędu (nie zdążyłam zaimplementować obliczania odchylenia standardowego).

```
def max_blad(x, y):
    blad = 0
    for i in range(len(x)):
        diff = y[i] - funkcja(x[i])
        if blad < abs(diff):
            blad = abs(diff)
    return blad
```

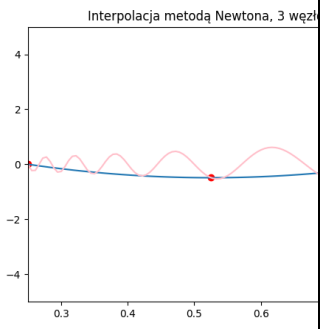
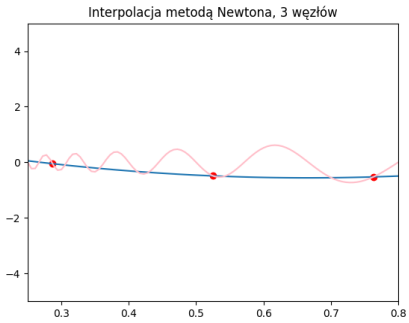
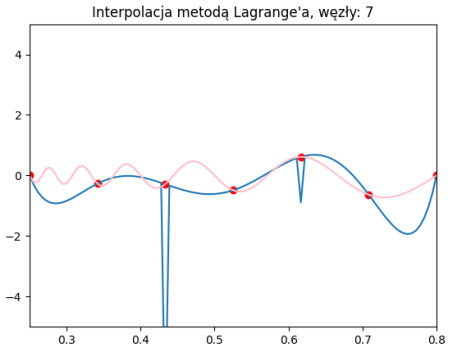
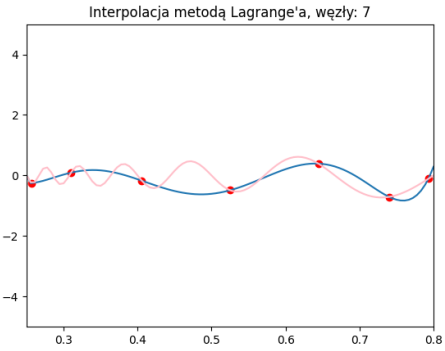
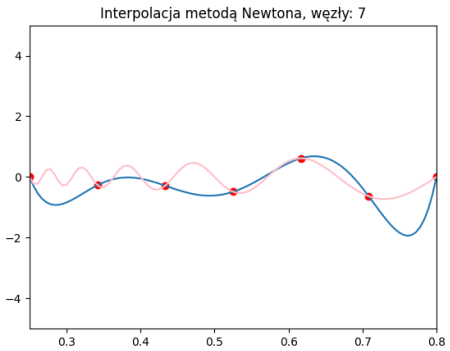
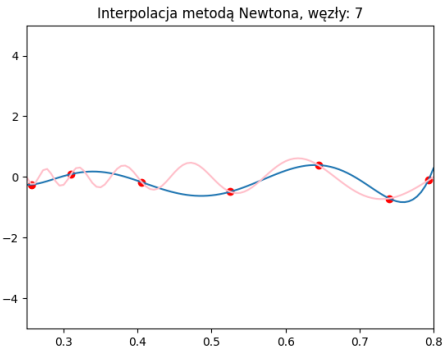
Fragment kodu 5

```
def funkcja(x):
    return x * sin((pi * 4) / x)
```

Fragment kodu 6, implementacja dane funkcji

3. Porównanie rozmieszczenia równo odległego i Czebyszewa

Poniżej zestawienie wykresów dla obu rozstawień (wykresy dla metody Lagrange’a lub Newtona. Oba były wstawiane tylko wtedy, kiedy były między nimi istotne różnice)

Równoodległe	Czebyszewa
	
	
	

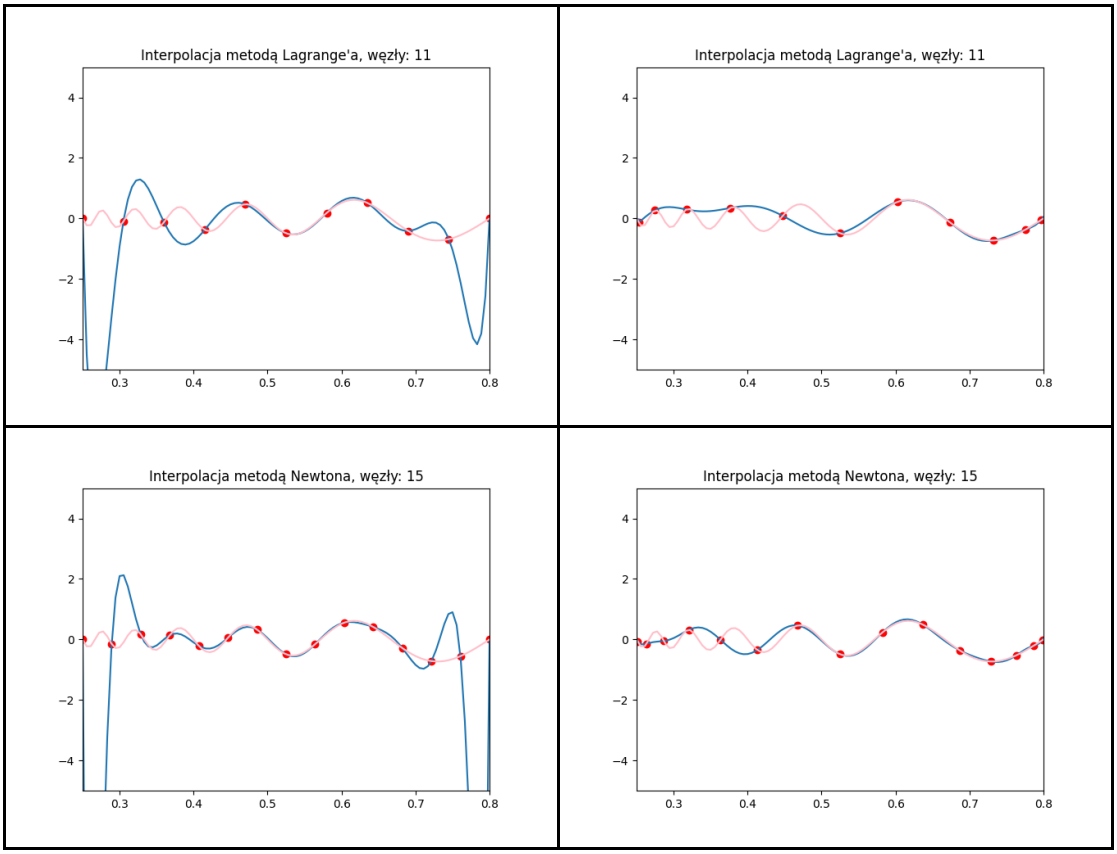


Tabela 1

	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

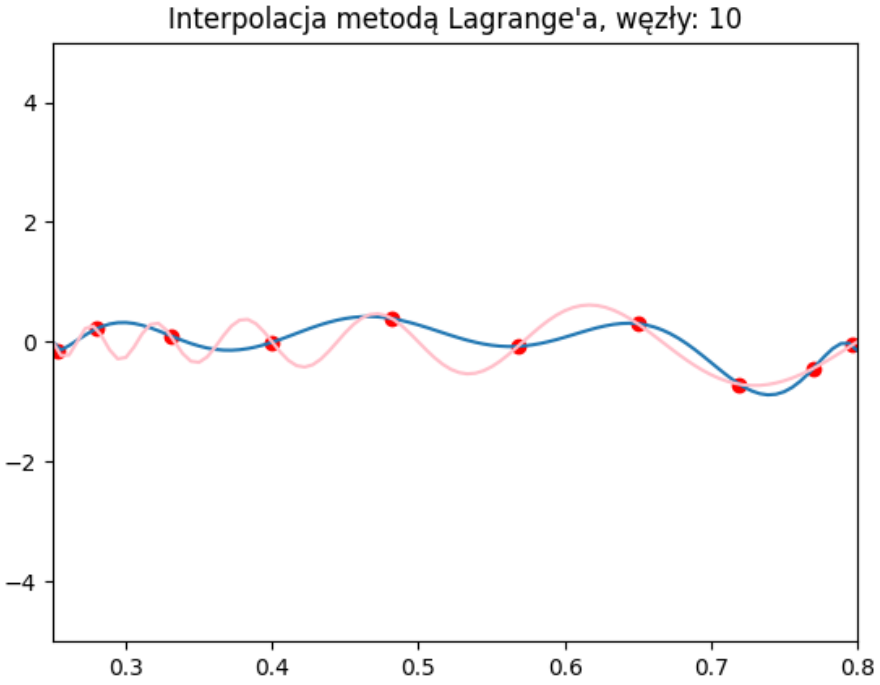
Tabela 2, błąd maksymalny

Obserwacje:

- Rozmieszczenie Czebyszewa daje lepsze wyniki dla większej liczby węzłów
- Metody Lagrange’a i Newtona dają zazwyczaj bardzo podobne wyniki, jednak metoda Lagrange’a ma większą tendencję do zwracania pojedynczych wartości, które są bardzo niezgodne z funkcją (w tym przypadku dla 7 węzłów)

4. Najlepiej przybliżony wielomian

Wielomian z najmniejszym błędem maksymalnym został wyliczony korzystając z metody Lagrange'a dla 10 węzłów z rozmieszczeniem Czebyszewa. Błąd ten wynosił 0.6240174825445816.



Wykres 2

5. **Efekt Runge'go**

Efekt ten udało się zaobserwować licząc dla 7 i 11 węzłów z rozmieszczeniem równoodległym (wykresy są w Tabeli 1). Dla 7 węzłów błąd był ponad pięciokrotnie mniejszy niż dla 11 węzłów.

Efektu tego nie zaobserwowałam licząc z rozmieszczeniem Czebyszewa. Dla 15 węzłów błąd jest minimalnie większy, jednak nie jest to drastyczna zmiana.

6. **Końcowe wnioski**

Mam podejrzenia, że pojawił się błąd w implementacji w metodzie Lagrange'a (czasem w okolicach węzłów można zaobserwować pojedyncze znaczne zmiany wartości). Nie przeszkodziło to jednak w przeprowadzeniu obserwacji i zauważeniu efektu Runge'go oraz różnicy między rozstawieniami węzłów.

Rozstawienie Czebyszewa okazało się znacząco lepsze, jednak na przyszłość zamiast liczyć je wzorem, policzę je funkcją numpy.polynomial.chebyshev.Chebyshev.interpolate.

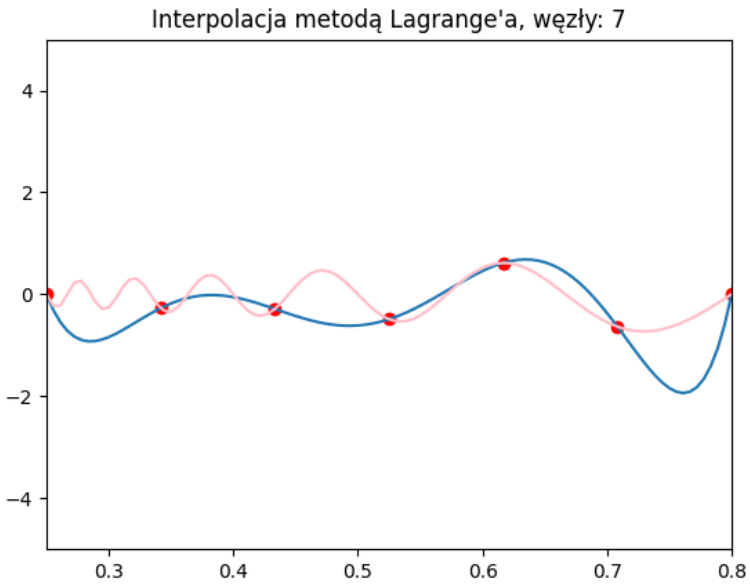
7. **Dane techniczne**
   - Język programowania: Python 3.10
   - Środowisko: PyCharm Professional
   - System: Windows Server 2019
   - Procesor: Intel Core i7-7700HQ 2.8 GHz
8. **Poprawki**

Poprawiłam implementację wzoru, według którego liczę metodą Lagrange'a. Wymusiłam, żeby wielomian w punktach, które mają wartość równą węzłom, przyjmował wartość funkcji w węźle.

```python
def interpolacja_lagranga(x, xn, yn):
    if x in xn:
        for i in range(len(xn)):
            if xn[i] == x:
                return yn[i]

    Pn = 0
    for k in range(len(xn)):
        Pn += yn[k] * Lk(x, k, xn)
    return Pn
```

Fragment kodu 7, poprawiona jedna z funkcji z Fragmentu kodu 1



Wykres 3, Interpolacja metodą Lagrange'a z 7 węzłami po naniesieniu poprawek(wykres przed poprawkami można zobaczyć w tabeli 1)