Zadanie Trzecie

1 Polecenie

Celem zadania drugiego jest zaimplementowanie metody interpolacji Newtona na węzłach Czebyszewa

Program ma:

- [X] umożliwiać wybór jednej z kilku funkcji:
  - [X] liniowa,
  - [X] $|x|$,
  - [X] wielomian,
  - [X] trygonometryczna
  - [X] ich złożenia.
- [X] Wartości wielomianów **interpolowanych** należy obliczać używając schematu Hornera.
- [X] Wartości wielomianów **interpolacyjnych** należy obliczać bezpośrednio, skorzystanie ze schematu Hornera nie jest bowiem możliwe bez uprzedniego przekształcenia wielomianu interpolacyjnego do postaci kanonicznej.
- [X] Użytkownik wybiera: funkcję, liczbę węzłów interpolacyjnych, przedział interpolacji.
- [X] Położenie węzłów wyliczane jest z odpowiednich wzorów,
- [X] Wartości w węzłach interpolacyjnych wyliczane są przy użyciu funkcji wybranej przez użytkownika.
- [X] Program ma rysować wykres funkcji oryginalnej i wielomianu interpolującego oraz zaznaczać węzły interpolacji
- [X] W sprawozdaniu należy zamieścić przykładowe wykresy.
- [X] Zbadać w jaki sposób zmiana liczby węzłów wpływa na dokładność interpolacji.
- [X] Ile węzłów potrzeba do interpolacji wielomianu N-tego stopnia?

2 Teoria

- Metoda interpolacji Newtona na węzłach Czebyszewa to technika interpolacji wielomianowej, która wykorzystuje węzły interpolacji równomiernierozmieszczone na przedziale interpolacji, zwanych węzłami Czebyszewa.

- Węzły te są zdefiniowane jako:

  $x_k = \cos$ [U+2061] $(\pi\frac{(2k-1)}{2n})$, k=1,2,...,n , gdzie n jest stopniem interpolacji.

- Wielomian interpolacyjny Newtona dla danych $n + 1$ węzłów interpolacji $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ jest dany przez wzór:

  $P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_n(x - x_0)(x - x_1)\ldots(x - x_{n-1})$

  gdzie współczynniki $a_0, a_1, \ldots, a_n$ są wyliczane rekurencyjnie przy użyciu różnic dzielonych:

  $a_k = f(x_0, x_1, \ldots, x_k)$, gdzie różnice dzielone są zdefiniowane jako:

  $f(x_i, x_j, \ldots, x_k) = f(x_i, x_j, \ldots, x_{k-1}) - f(x_j, x_{j+1}, \ldots, x_k)x_i - x_k$

  a dla k=0:

  $f(x_i) = y_i$

3 Program

W naszym programie wykorzystujemy dwie wolnościowe biblioteki: `numpy` oraz `matplotlib`.

[1]:
```
# IMPORTS & CONSTANTS
import csv
import numpy as np; from numpy import sin, cos
import numpy.testing as npt # WARN: nie działa do 'raise'
import matplotlib.pyplot as plt

DEBUG = False
INTERACTIVE = False
```

Obliczanie wartości funkcji schematem Hornera zaimplementowaliśmy doskonale już w pierwszym zadaniu i jego implementacja nadal się nie zmieniła.

[2]:
```
def horner(coefs, x): # lista współczynników
    result = 0
    for coef in coefs: result = result*x + coef
    return result
```

Wybrać można spośród funkcji: 1. $x + 1$ – liniowa 2. $|x|$ – moduł 3. $x^3 - x^2 + x - 1$ – wielomian 4. $\cos(x)$ – trygonometryczna 5. $(x + 1) \cdot \sin(x)$ – złożenie: liniowa + trygonometryczna 6. $|x| \cdot \cos(x)$ – złożenie: moduł + trygonometryczna

[3]:
```
def fun(index, x): # wybór funkcji
    match index:
        case 1: return x+1
        case 2: return abs(x)
        case 3: return horner((1,-1,1,-1), x)
        case 4: return cos(x)
        case 5: return (x+1) * sin(x)
        case 6: return abs(x) * cos(x)
```

[4]:
```
def chebyshev_nodes(n): return [cos((2*k - 1) * np.pi / (2*n)) for k in range(1,␣
↪n+1)]
def divided_diff(x, y): # iloraz różnicowy, przyjmuje listy jako argumenty
    n = len(y)
    coef = np.zeros([n, n])
    coef[:,0] = y # the first column is y

    for j in range(1,n):
        for i in range(n-j):
            coef[i][j] = (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j]-x[i])
    return coef[0, :]
def newton_poly(coef, x_data, x): # evaluate the newton polynomial at x
    n = len(x_data) - 1
    p = coef[n]
    for k in range(1, n+1): p = coef[n-k] + (x - x_data[n-k])*p
    return p
```

Najważniejsze obliczenia w naszym programie znaldują się w bloku kodu poniżej i są to funkcja `interpolateWithPlot`, która jest połączeniem wcześniejszych funkcji `interpolate` i `myPlot`.

[5]:
```
def interpolateWithPlot(func_i, node_c=5, l_edg=-1, r_edg=1, step=0.1):
    # 1. Wylicz węzły czybyszewa
    nodes = chebyshev_nodes(node_c)
    # 2. Wylicz wartości funkcji w węzłach
    values = [fun(func_i, node) for node in nodes]
    if DEBUG: print("Węzły czebyszewa to...\n x:", nodes, "\ny:", values)
    # 3. Wylicz ilorazy różnicowe
    coefs = divided_diff(nodes, values)
    if DEBUG: print("coefs =\n", coefs)
    # 4. Interpolacja wielomianowa
    x_vals = np.arange(l_edg, r_edg+step, step)
    y_true = [ fun(func_i, x)                for x in x_vals ]
    y_intrp = [ newton_poly(coefs, nodes, x) for x in x_vals ]; del coefs # no␣
↪longer needed
    if DEBUG:
        print("Długości...")
        print("\t x_vals:", len(x_vals))
        print("\t y_true:", len(y_true))
        print("\t y_intrp:", len(y_intrp))
        print("\t nodes & values:", len(nodes), len(values))

    #return x_vals, y_true, y_intrp, nodes, values, func_i

    plt.scatter(nodes, values, color="red", label='Węzły interpolacji')
    plt.plot(x_vals, y_true, color="green", label="Funkcja oryginalna", alpha=0.
↪5)
    plt.plot(x_vals, y_intrp, color="blue", label="Wielomian interpolacyjny")
```

```
    plt.title(f"Interpolacja Newtona na węzłach Czebyszewa, funkcja {func_i}")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.grid(True)
    plt.show()
```

4 Testy jednostkowe

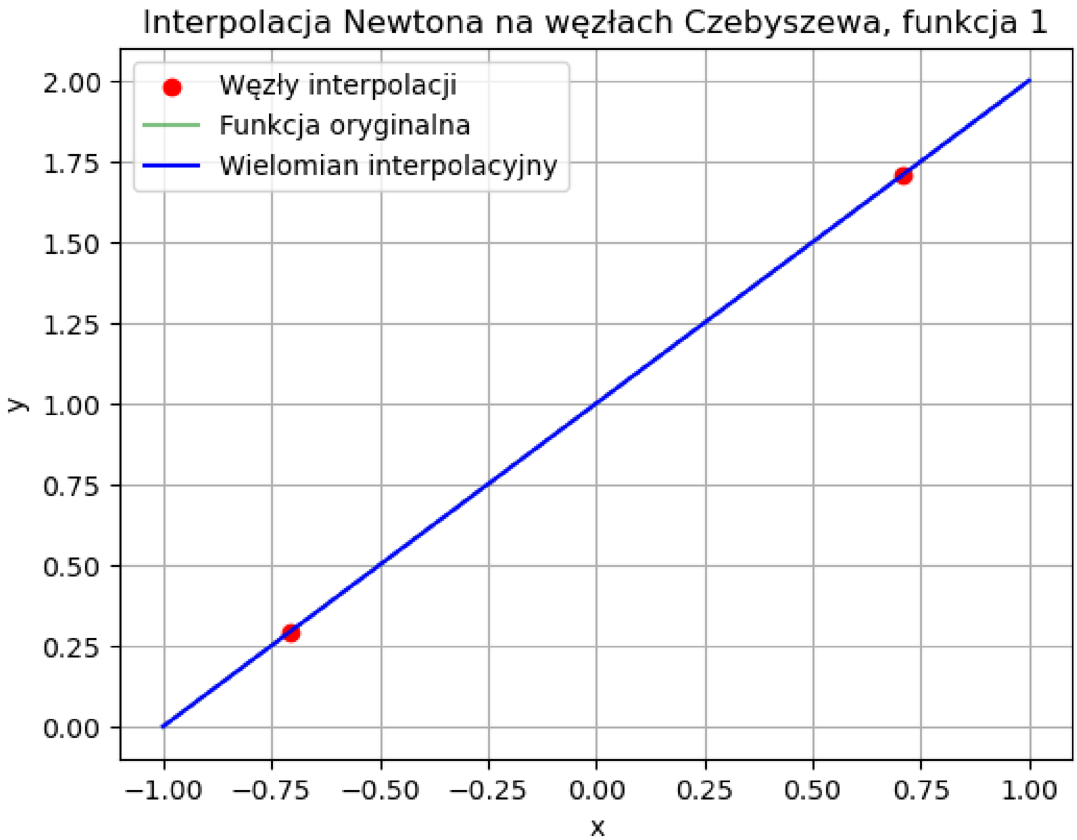
Jako profesjonaliści piszemy testy jednostkowe. # Testy funkcji do wybierania funkcji

[6]:
```
npt.assert_equal(fun(1,1),      2,      err_msg="Nie działa wybranie funkcji o␣
 ↪indeksie 1")
npt.assert_equal(fun(2,-1),     1,      err_msg="Nie działa wybranie funkcji o␣
 ↪indeksie 2")
npt.assert_equal(fun(3,1),      0,      err_msg="Nie działa wybranie funkcji o␣
 ↪indeksie 3")
npt.assert_equal(fun(4,0),      1,      err_msg="Nie działa wybranie funkcji o␣
 ↪indeksie 4 (łatwe)")
npt.assert_equal(fun(4,np.pi), -1,      err_msg="Nie działa wybranie funkcji o␣
 ↪indeksie 4")
npt.assert_equal(fun(5,0),      0,      err_msg="Nie działa wybranie funkcji o␣
 ↪indeksie 5 (łatwe)")
npt.assert_equal(fun(6,np.pi), -np.pi, err_msg="Nie działa wybranie funkcji o␣
 ↪indeksie 6")
```

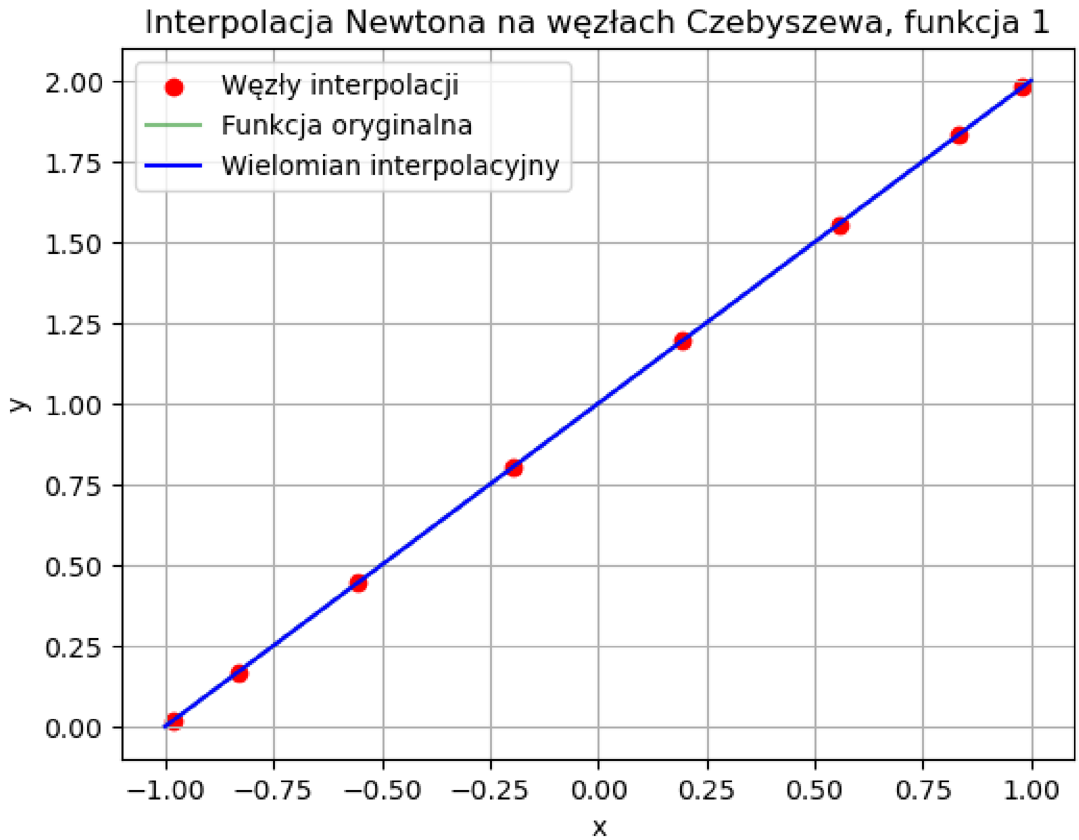
5 Wykresy dla poszczególnych funkcji

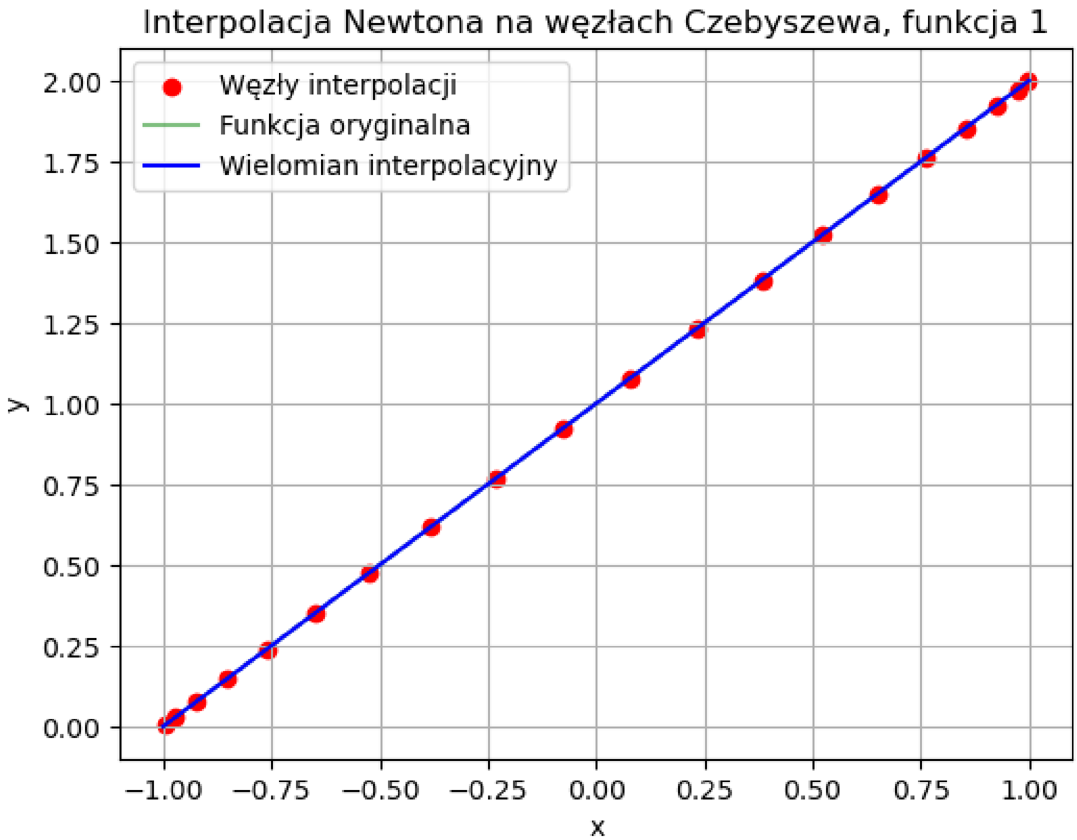
Dla każdego z przykładów przeprowadziliśmy serię testów aby określić ile węzłów czybyszewa potrzeba do subiektywnie dostatecznie dobrego odwzorowania funkcji na wykresie. Typowo dla węzłów czybyszewa przyjeliśmy przedział $[-1, 1]$, w którym wybraliśmy punkty co 0.1. ## Funkcja liniowa Zapewnie zdziwieniem dla nikogo nie będzie, że wystarczą dwa węzły. Zwiększenie ich liczby przynosi pomijalne skutki.

[7]:
```
for nodes in 2, 8, 20: interpolateWithPlot(1, node_c=nodes, step=0.1) # func_i,␣
 ↪node_c
```



Interpolacja Newtona na węzłach Czebyszewa, funkcja 1
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


Interpolacja Newtona na węzłach Czebyszewa, funkcja 1
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
y
x
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

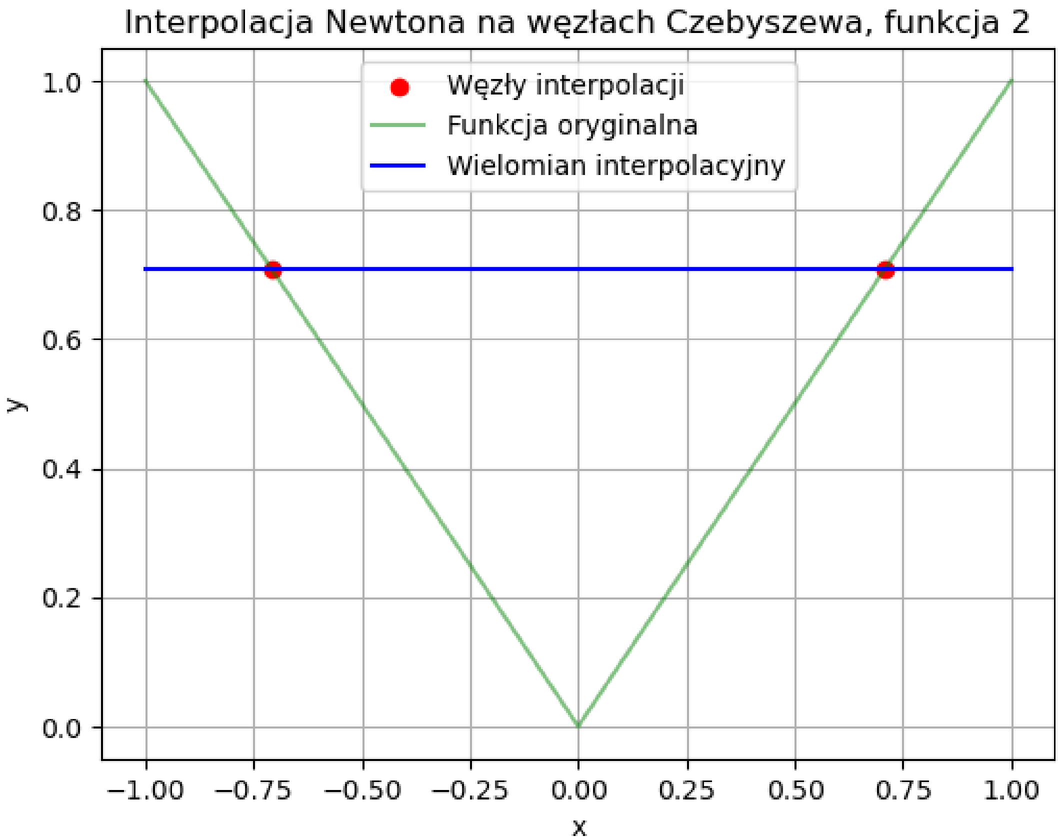


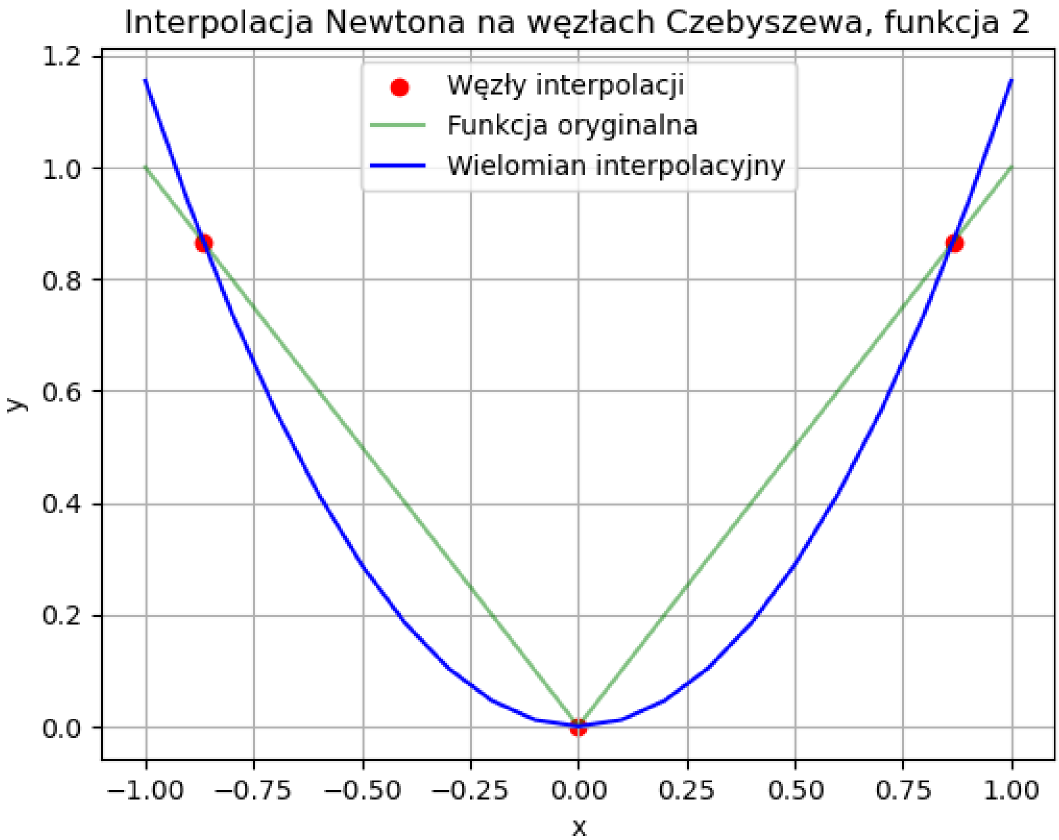
5.1 Moduł

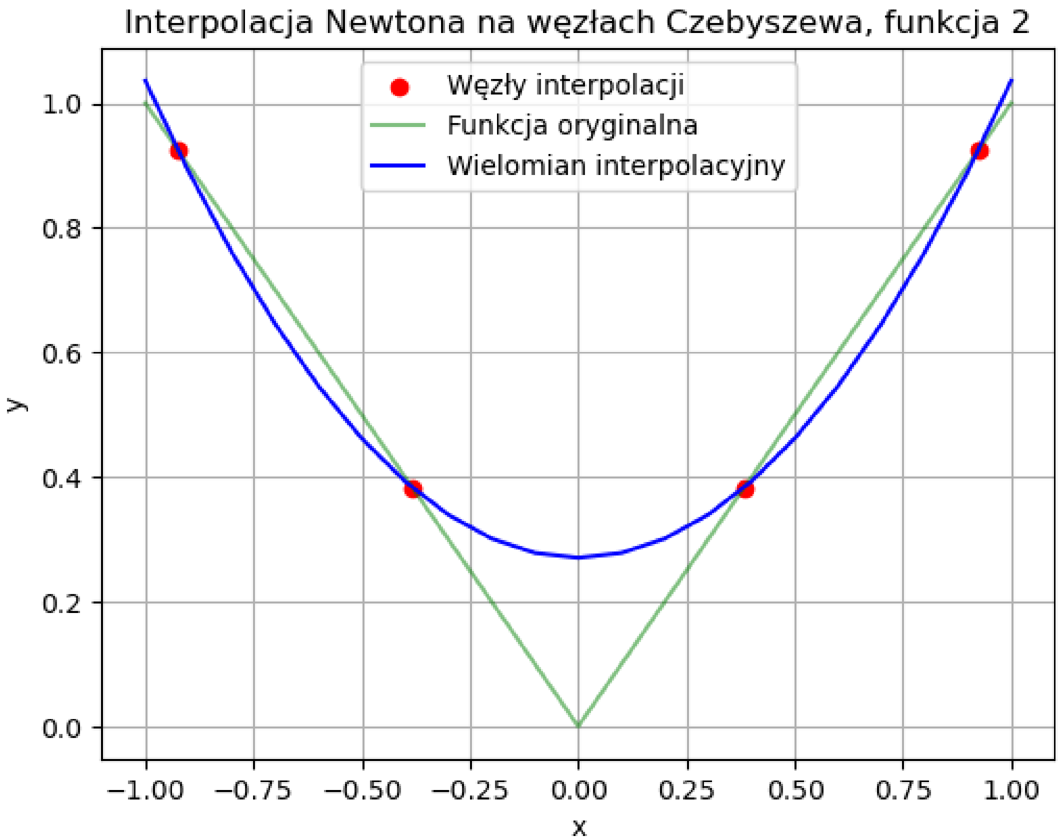
Dla modułu sprawa wygląda ciekawie, ponieważ moduł to najprostsza funkcja **liniowa parzysta**, a my stosujemy interpolację **wielomianową**.

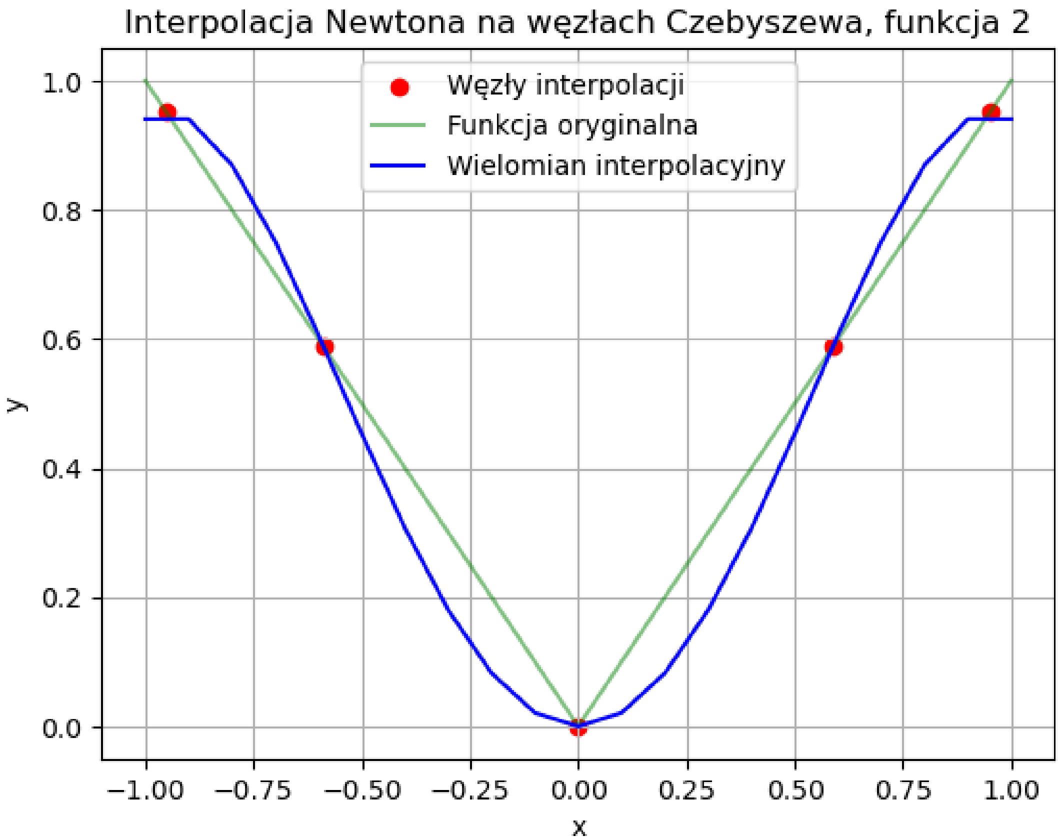
Można zauważyć, że w przypadku parzystej i większej od dwóch liczby węzłów aproksymacji na większości funkcji jest dokładniejsza w przypadku nieparzystej ich liczby. Dzieje się tak, ponieważ przy nieparzystej liczbie węzłów jeden z nich przechodzi mniej więcej przez środek przedziału, jednak z drugiej strony powoduje to lepszą dokładność we wskazaniu minimum takiej funkcji.

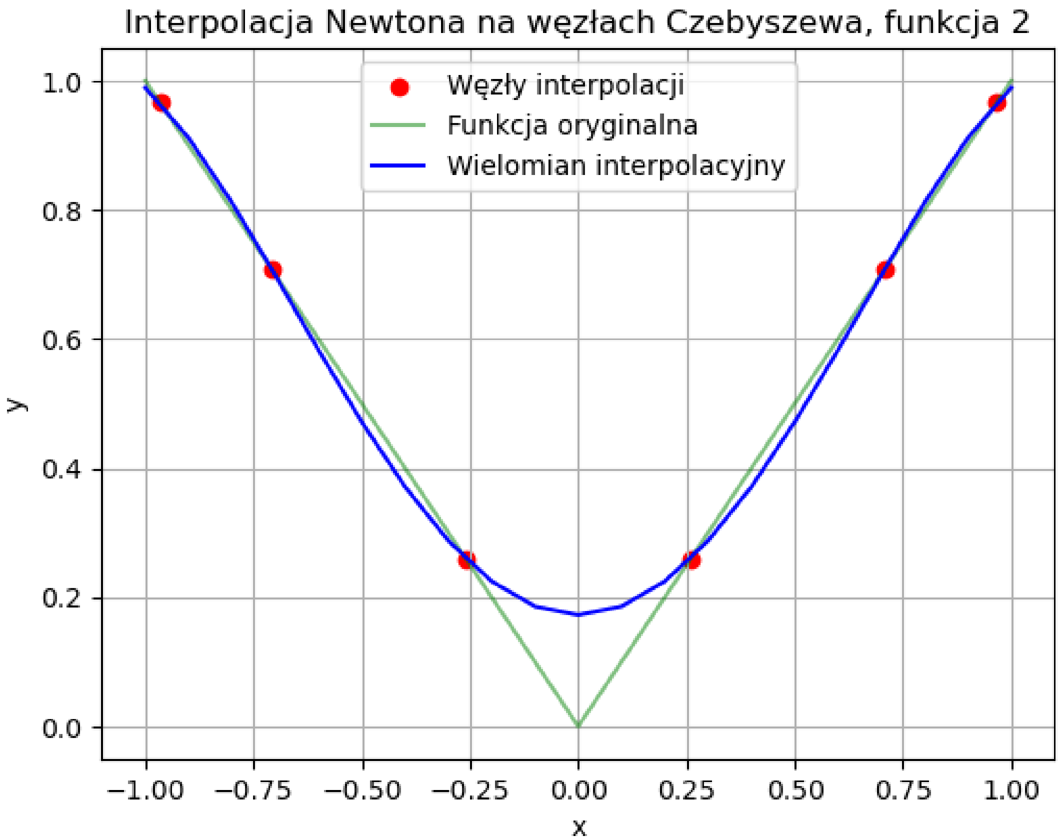
[8]:
```
for nodes in range(2,9): interpolateWithPlot(2, node_c=nodes, step=0.1)
```

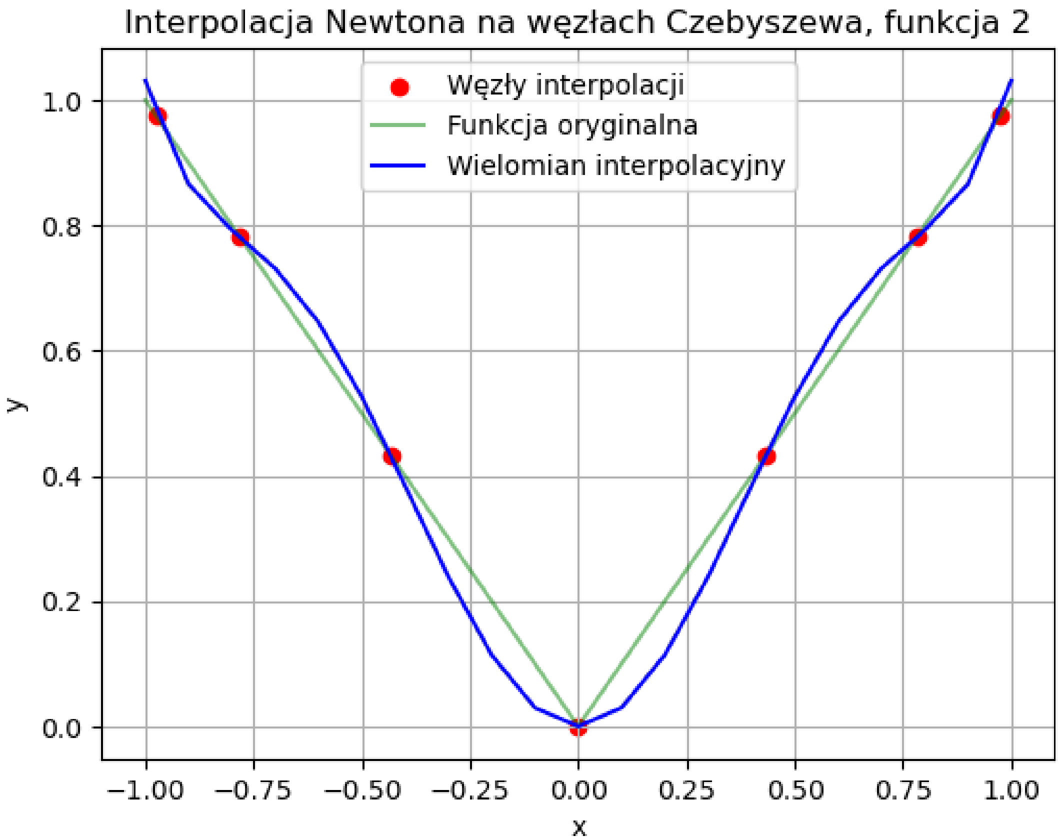

Interpolacja Newtona na węzłach Czebyszewa, funkcja 2
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
1.0
0.8
0.6
0.4
0.2
0.0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


Interpolacja Newtona na węzłach Czebyszewa, funkcja 2
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
y
x
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

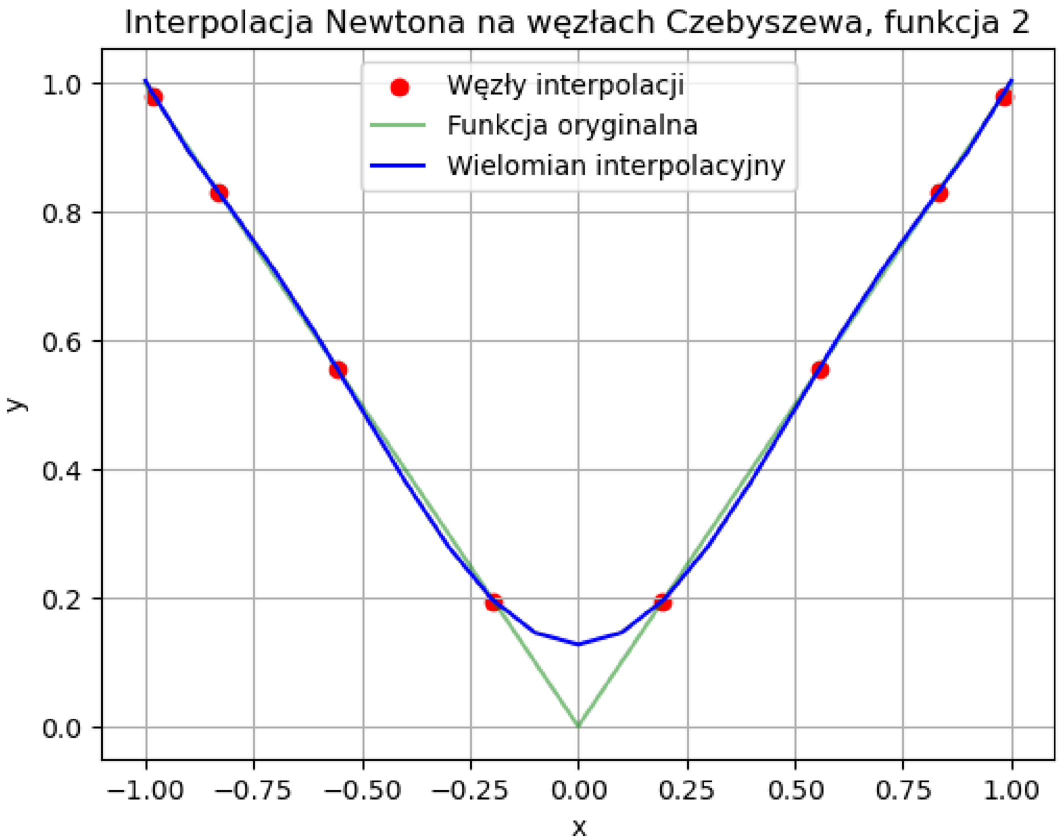

Interpolacja Newtona na węzłach Czebyszewa, funkcja 2
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
1.0
0.8
0.6
0.4
0.2
0.0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


Interpolacja Newtona na węzłach Czebyszewa, funkcja 2
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
1.0
0.8
0.6
0.4
0.2
0.0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


Interpolacja Newtona na węzłach Czebyszewa, funkcja 2
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



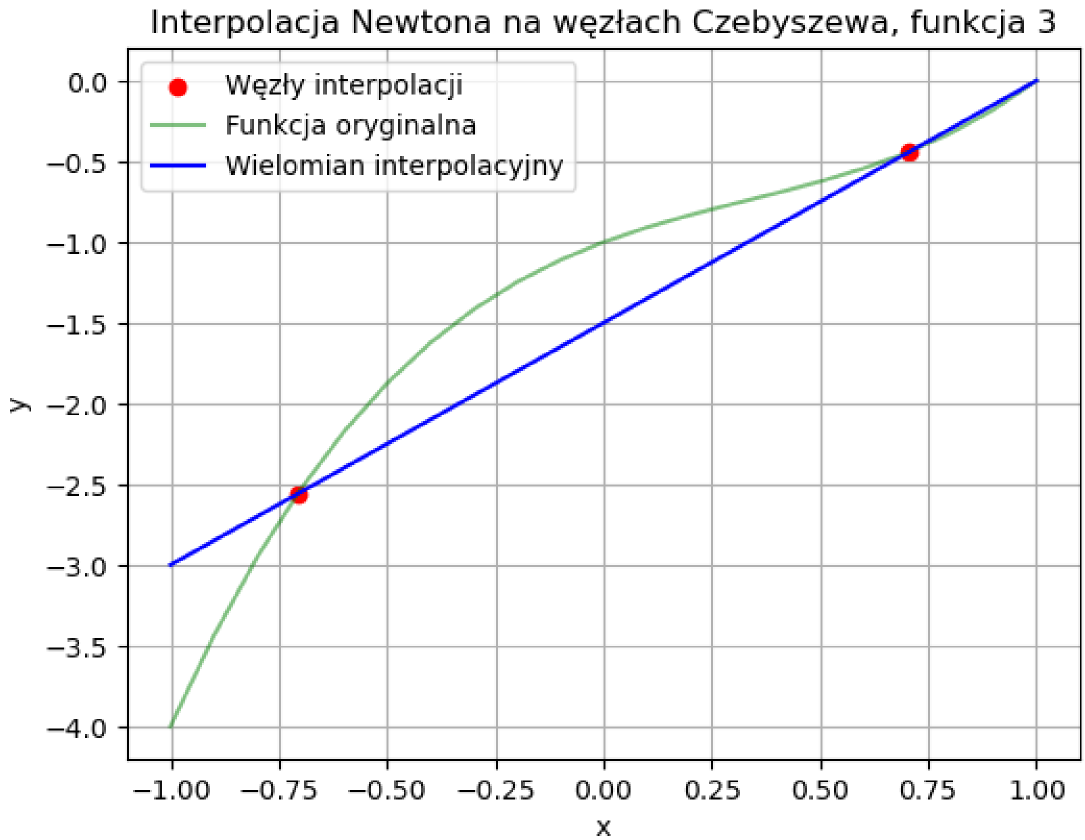
Interpolacja Newtona na węzłach Czebyszewa, funkcja 2
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

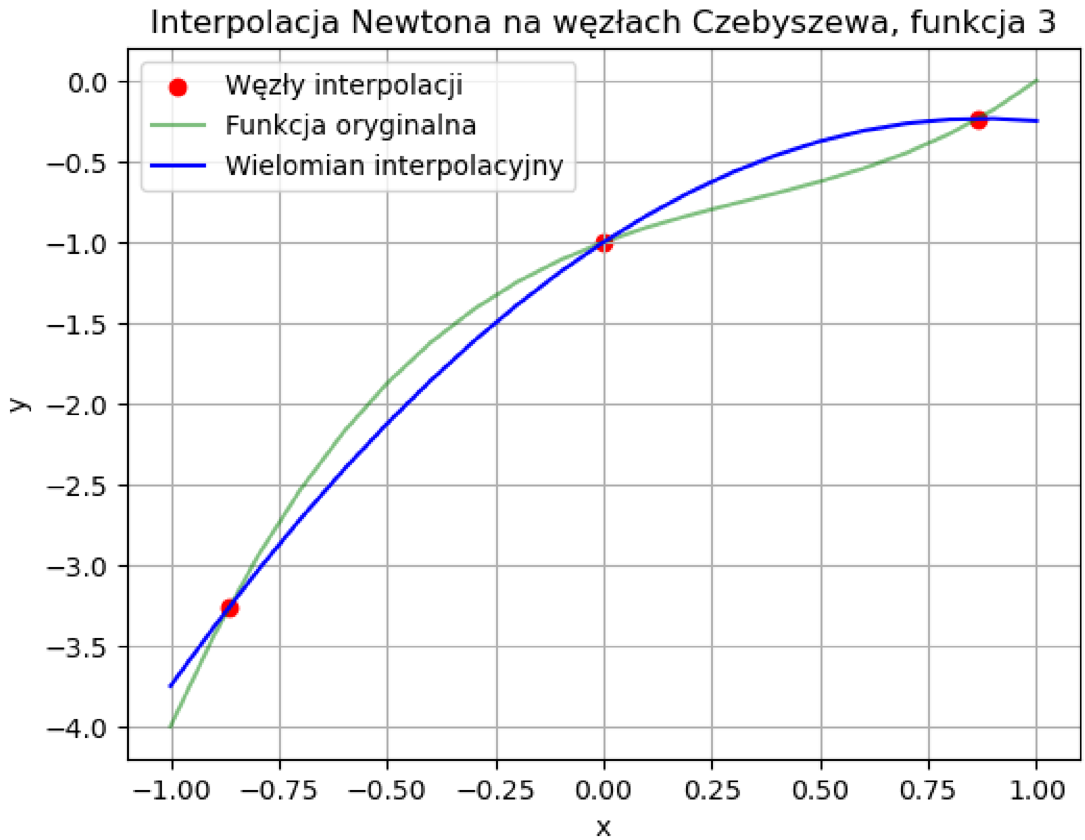


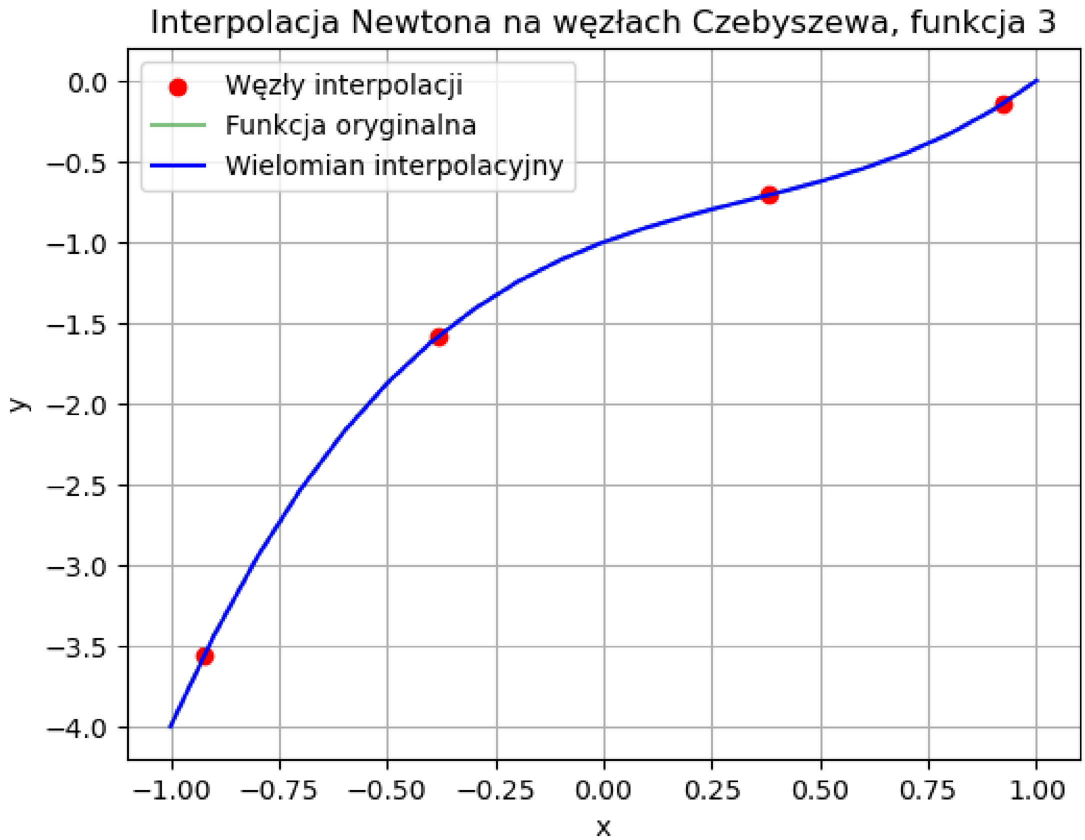
5.2 Wielomian

Jak widać aby dokładnie odwzorować wielomian n-tego stopnia potrzebne jest przynajmniej n węzłów

[9]:
```
for nodes in range(2,5): interpolateWithPlot(3, node_c=nodes, step=0.1)
```


Interpolacja Newtona na węzłach Czebyszewa, funkcja 3
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

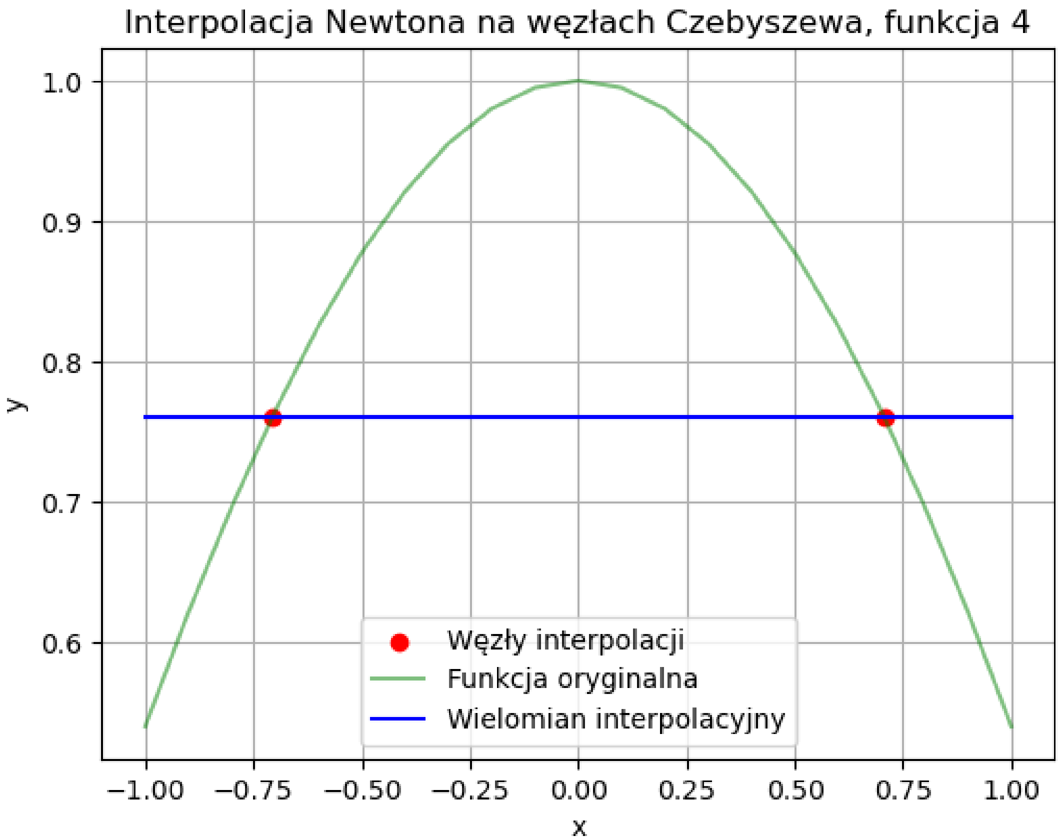

Interpolacja Newtona na węzłach Czebyszewa, funkcja 3
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
y
x
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



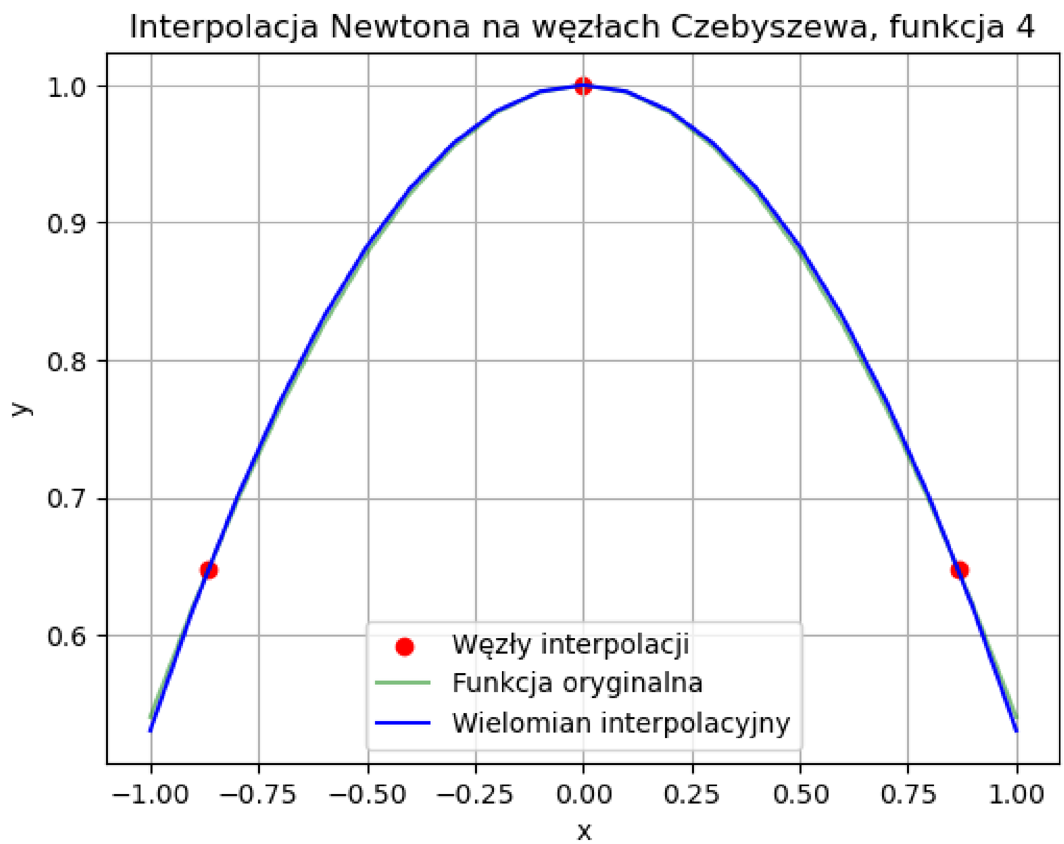
5.3 Funkcja trygonometryczna

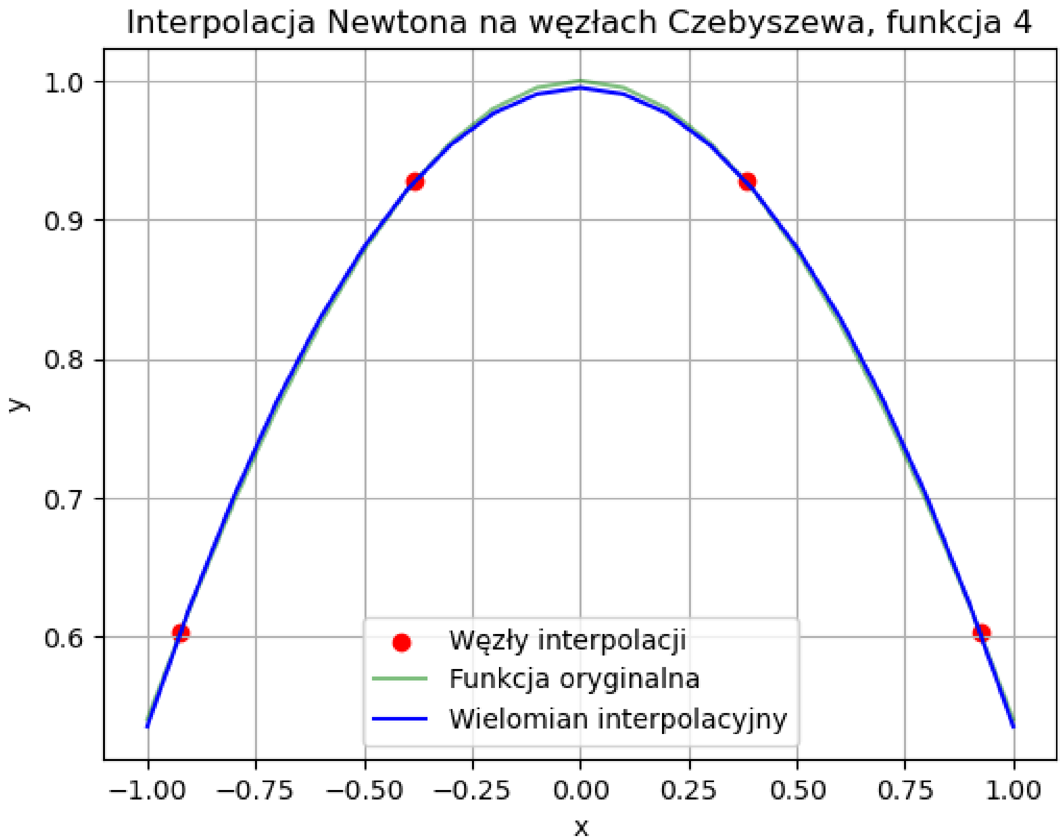
Jak widać cosinus jest świetnie interpolowany już przy trzech węzłach:

[10]:
```
for nodes in range(2,5): interpolateWithPlot(4, node_c=nodes, step=0.1)
```



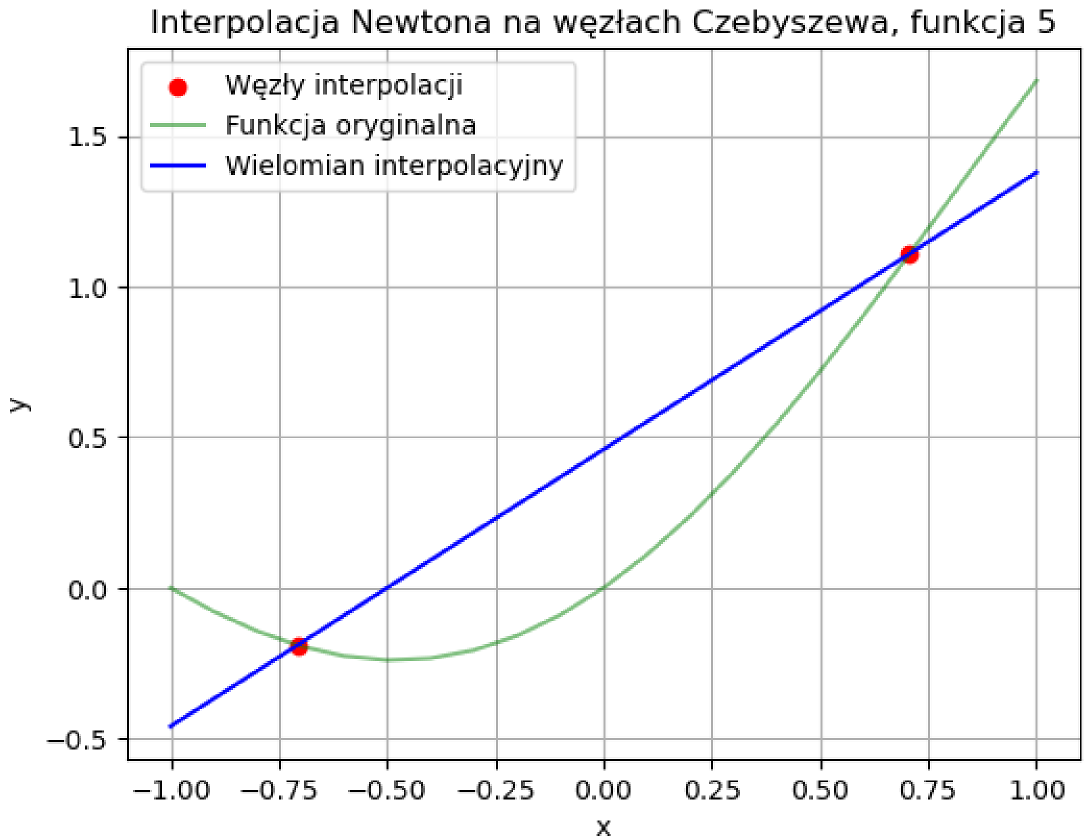
Interpolacja Newtona na węzłach Czebyszewa, funkcja 4
1.0
0.9
0.8
0.7
0.6
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny

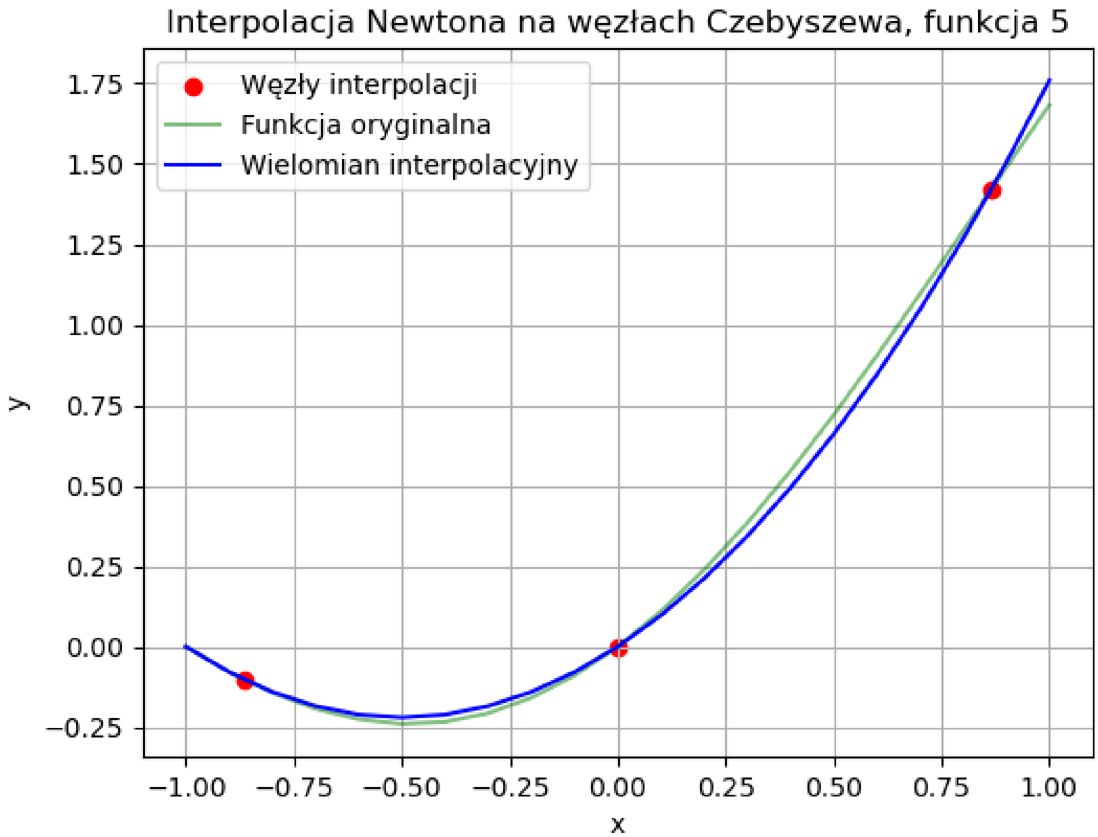

Interpolacja Newtona na węzłach Czebyszewa, funkcja 4
y
x
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny

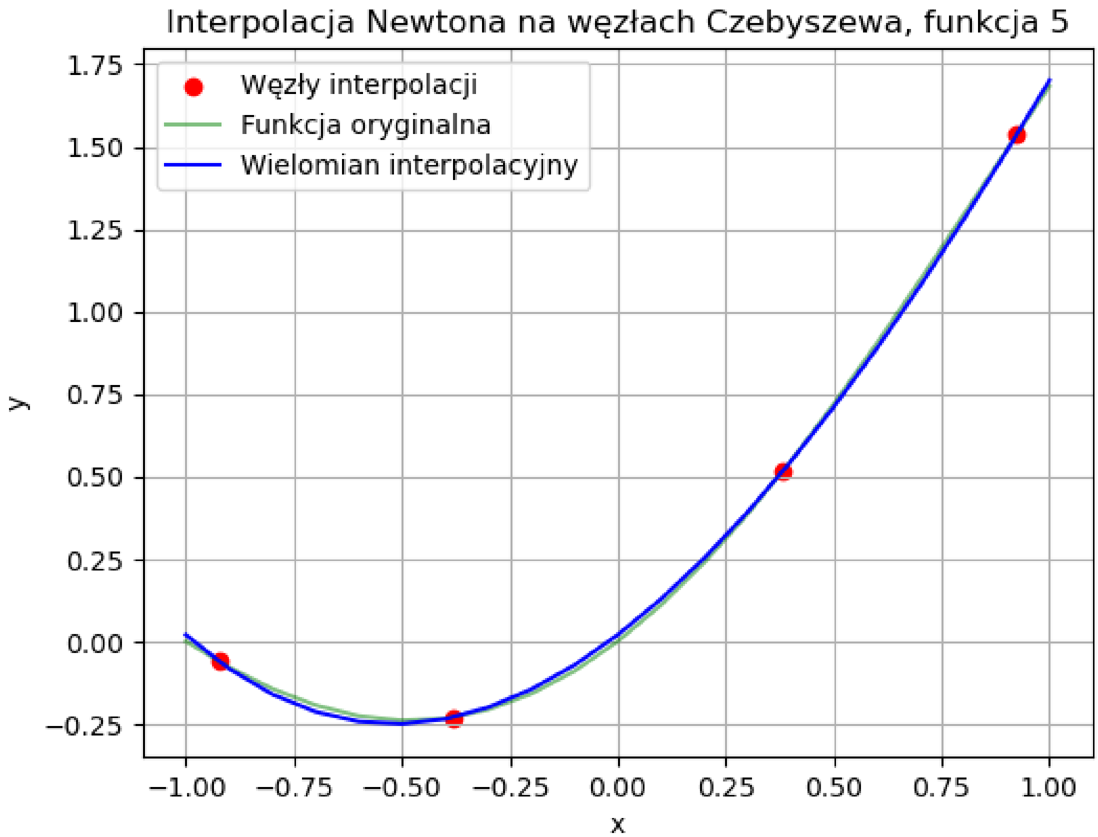


5.4 Liniowa + trygonometryczna

[11]:
```
for nodes in range(2,5): interpolateWithPlot(5, node_c=nodes, step=0.1)
```

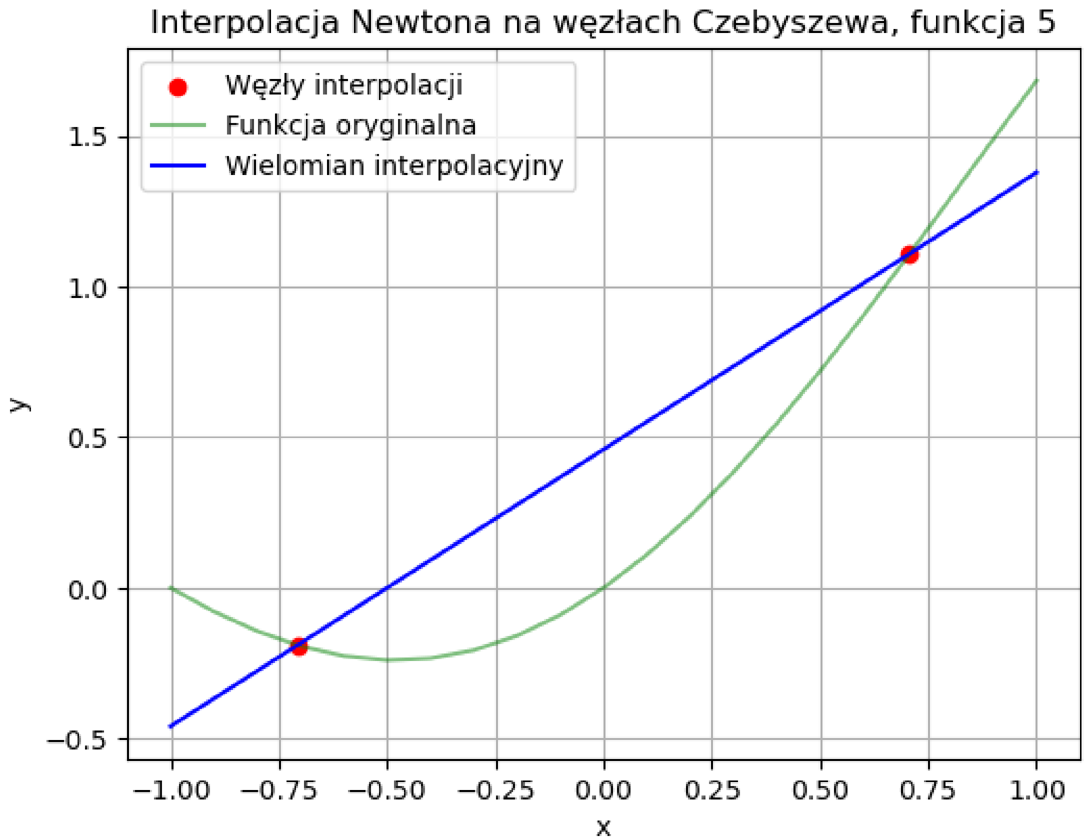

Interpolacja Newtona na węzłach Czebyszewa, funkcja 5
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
y
x
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
−0.5
0.0
0.5
1.0
1.5


Interpolacja Newtona na węzłach Czebyszewa, funkcja 5
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

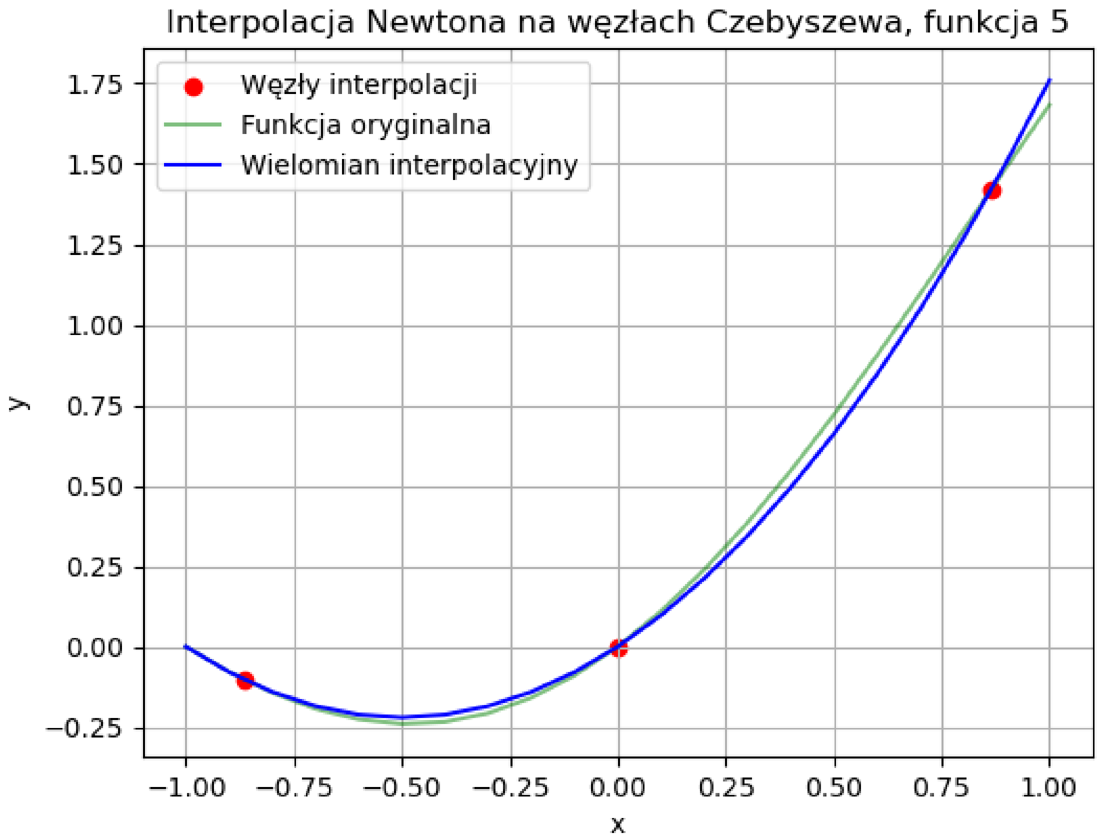


5.5 Moduł + trygonometryczna

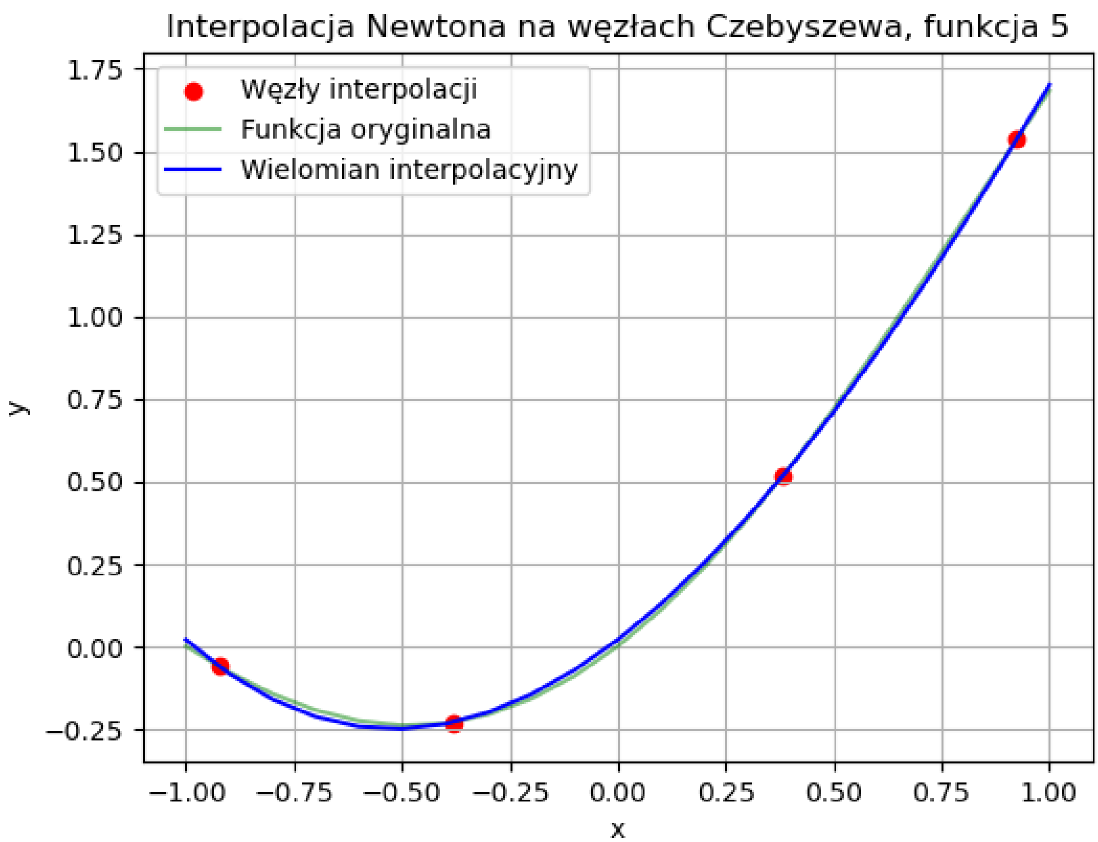
[12]:
```
for nodes in range(2,5): interpolateWithPlot(5, node_c=nodes, step=0.1)
```



Interpolacja Newtona na węzłach Czebyszewa, funkcja 5
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
y
x
1.5
1.0
0.5
0.0
−0.5
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



Interpolacja Newtona na węzłach Czebyszewa, funkcja 5
Węzły interpolacji
Funkcja oryginalna
Wielomian interpolacyjny
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x



6 Część interaktywna

[13]:
```
def printFunctions(): # Wybór funkcji do interpolacji
    print("Wbudowane funkcje:")
    print("1. Funkcja liniowa (x + 1)")
    print("2. Funkcja |x|")
    print("3. Funkcja wielomianowa (x^3 - x^2 + x - 1)")
    print("4. Funkcja trygonometryczna (cos(x))")
    print("5. Złożenie funkcji (x + 1) * sin(x)")
    print("6. Złożenie funkcji |x| * cos(x)")

def inputParameters(): # Wprowadzenie parametrów
    l_edg = -1; r_edg = 1
    func_i = int(input("Wybierz indeks funkcji: "))
    node_c = int(input("Podaj liczbę węzłów interpolacyjnych: "))
    #l_edg = float(input("Podaj początek przedziału interpolacji: "))
    #r_edg = float(input("Podaj koniec przedziału interpolacji: "))
    return func_i, node_c#, l_edg, r_Edg
```

```
if INTERACTIVE: # switch to true
    printFunctions()
    func_i, node_c = inputParameters()
    interpolateWithPlot(func_i, node_c=node_c, step=0.1)
```

7 Wnioski

1. Metoda ta jest uniwersalna i pozwala na interpolację każdej funkcji ciągłej.
2. Węzły interpolacji pokrywają się z punktami przecięcia oryginalnych wykresów z wykresami funkcji aproksymującej, co pozwala stwierdzić prawdiłową implementację metody numerycznej.
3. Dzięki zastosowaniu węzłów Czebyszewa interpolacja unika błędu Rungego, który występuje przy interpolacji na węzłach równoodległych.
4. W jaki sposób zmiana liczby węzłów wpływa na dokładność interpolacji:

- dla funkcji liniowej - dla dokładnej interpolacji potrzeba 2 węzłów, każdy dodatkowy powyżej tej liczby nie wpływa zbytnio na dokładność,
- dla funkcji moduł z x - każda interpolacja składająca się z parzystej liczby węzłów (zaczynając od 3) jest w miarę dokładna, ale fałszywie podwyższa wartość minimum funkcji, z kolei nieparzysta liczba węzłów powoduje mniejszą dokładność aproksymacji, lecz za to minimum funkcji jest w niej rysowana w tym miejscu, w którym znajduje się oryginalne,
- dla funkcji wielomianowej N-tego stopnia - potrzebuje takiej ilości węzłów jaka odpowiada stopniowi wielomianu,
- dla funkcji trygonometrycznej - dla cosinusa odpowiednia ilość to nieparzysta liczba węzłów (począwszy od 3), przy parzystej (począwszy od 4) jest trochę mniej dokładna, lecz jest to wynik w miarę dostateczny, dla funkcji sinus jest odwrotnie tzn. trochę lepsza jest parzysta liczba węzłów,
- dla funkcji złożonej sinus + liniowa - liczba węzłów wpływa podobnie na dokładność jak przy samej funkcji sinus (fucnkja liniowa za dużo nie zmienia),
- dla funkcji złożonej cosinus * moduł - liczba węzłów wpływa podobnie na dokładność jak przy samej funkcji cosinus (fucnkja liniowa za dużo nie zmienia).