Obliczenia Naukowe - L4

Piotr Maciejończyk

1 grudnia 2023

1 Ilorazy różnicowe

Zadanie polegało na zaimplementowaniu funkcji obliczającej i zwracającej wektor ilorazów różnicowych dla węzłów $x_0, \ldots, x_n$ oraz wartości interpolowanej funkcji w tych węzłach $f(x_0), \ldots, f(x_n)$. Z wykładu wiadomo, iż wielomian $p \in \Pi_n$, gdzie $p(x_i) = f(x_i)$ $(0 \leq i \leq n)$ można przedstawić jako:

$$p(x) = \sum_{j=0}^{n} c_j q_j(x), \text{ gdzie:}$$

$$\begin{gathered} q_0(x) = 1 \\ q_1(x) = (x - x_0) \\ q_2(x) = (x - x_0)(x - x_1) \\ . \\ . \\ . \\ q_n(x) = (x - x_0)(x - x_1) \cdot \ldots \cdot (x - x_{n-1}) \end{gathered}$$

Można wyznaczyć $c_0, c_1, \ldots, c_n$ i zauważyć, że c_i zależy od f w punktach $x_0, x_1, \ldots, x_i$. **Ilorazem różnicowym** opartym na węzłach $x_0, x_1, \ldots, x_n$ nazywamy wielkość $c_n = f[x_0, x_1, \ldots, x_n]$, która jest współczynnikiem przy q_n. Zatem:

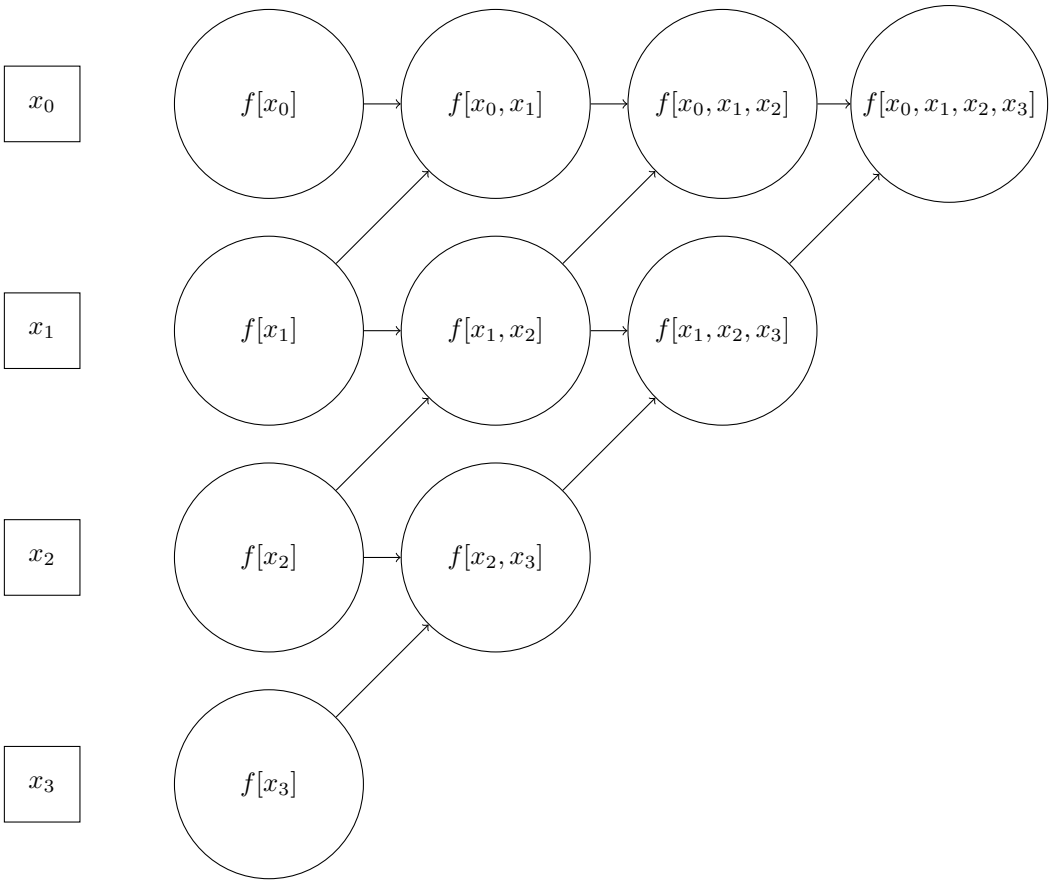
$$\begin{gathered} c_0 = f[x_0] \\ c_1 = f[x_0, x_1] \\ c_2 = f[x_0, x_1, x_2] \\ . \\ . \\ . \\ c_n = f[x_0, x_1, \ldots, x_n] \end{gathered}$$

Zgodnie z twierdzeniem z wykładu o ilorazach różnicowych wyższego rzędu, spełniają one zależności:

$$f[x_i] = f(x_i)$$

$$f[x_0, x_1, \ldots, x_n] = \frac{f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}]}{x_n - x_0}$$

Kolejne wyrazy można łatwo obliczyć stosując wizualizację z Rysunku 1., który znajduje się poniżej:



Rysunek 1: Przykład obliczania kolejnych ilorazów różnicowych rzędów 0, 1, 2 i 3

Jak można zauważyć powyżej (oraz posługując się przykładem grafu), ilorazy różnicowe wyższych rzędów można uzyskać poprzez obliczanie wartości wierzchołków kolejnych kolumn. Końcowym wynikiem jest wektor wartości wierzchołków w pierwszym rzędzie.

Posługując się tym sposobem myślenia zaimplementowałem funkcję obliczającą ilorazy różnicowe. W pierwszej pętli do wektora ilorazów różnicowych wstawiam odpowiednio wartości z kolumny pierwszej (o postaci $f[x_i]$). Następnie iteracyjnie obliczam kolejne wartości dla kolejnych kolumn. Poniżej umieściłrm pseudokod mojej funkcji:

Algorithm 1 Funkcja obliczająca ilorazy różnicowe

```
1: input: x - wektor węzłów x_0, ..., x_n, f - wektor wartości funkcji f(x_0), ..., f(x_n)
2: output: quotients - wektor zawierający obliczone ilorazy różnicowe
3: function IlorazyRoznicowe(x, f)
4:     if length(x) ≠ length(f) or length(x) == 0 then
5:         throw error
6:     end if
7:     n ← length(x)
8:     quotients ← initialize array of size n
9:     for i ← 1 to n do
10:        quotients[i] ← f[i]
11:    end for
12:    for j ← 2 to n do
13:        for k ← n down to j do
14:            quotients[k] ← (quotients[k]−quotients[k − 1])/(x[k] − x[k − j + 1])
15:        end for
16:    end for
17:    return quotients
18: end function
```

Sam algorytm wykonywany jest w czasie $O(n^2)$, ponieważ:

$$\sum_{j=2}^{n}\sum_{k=j}^{n} 1 = \sum_{j=2}^{n}(n-j+1) = \sum_{j=2}^{n} n - \sum_{j=2}^{n} j + \sum_{j=2}^{n} 1 = n(n-1) - \frac{(n-1)(n+2)}{2} + (n-1) = \frac{n(n-1)}{2} = O(n^2)$$

2 Wartość wielomianu interpolacyjnego w postaci Newtona

Zadanie dotyczyło implementacji funkcji obliczającej wartość wielomianu interpolacyjnego stopnia n w postaci Newtona $N_n(x)$ w punkcie $x = t$ za pomocą uogólnionego algorytmu Hornera, w czasie $O(n)$.

Uogólniony algorytm Hornera to modyfikacja klasycznego algorytmu Hornera, który służy do efektywnego obliczania wartości wielomianu. Algorytm Hornera jest szczególnie przydatny, gdy chcemy unikać wielokrotnego obliczania potęg danej wartości, co pozwala zaoszczędzić czas obliczeniowy.

Uogólniony algorytm Hornera rozszerza klasyczny algorytm na wielomiany interpolacyjne, gdzie wartości nie są jedynie współczynnikami wielomianu, ale ilorazami różnicowymi Newtona.

Dla wielomianu interpolacyjnego stopnia n o postaci Newtona, uogólniony algorytm Hornera można opisać w następujący sposób:

1. Inicjalizuj zmienną **result** wartością współczynnika najwyższej potęgi wielomianu (ilorazu różnicowego o najwyższym stopniu).
2. Iteruj od najwyższego stopnia wielomianu w dół do zera.
3. Mnóż współczynnik **result** przez $(x - x_i)$ (gdzie x_i to węzeł interpolacyjny) i dodaj iloraz różnicowy o stopniu odpowiadającym obecnej iteracji.
4. Powtarzaj kroki 2-3 dla wszystkich stopni wielomianu.

Uogólniony algorytm Hornera pozwala efektywnie obliczać wartość wielomianu interpolacyjnego w danym punkcie, minimalizując liczbę potrzebnych operacji mnożenia i dodawania. Jest to szczególnie ważne w przypadku wielomianów o wysokim stopniu.

Teraz pokażę, dlaczego zmodyfikowany algorytm Hornera działa. Klasyczny algorytm Hornera dla wielomianu stopnia n o postaci:

$$p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n,$$

można przedstawić jako:

$$p(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + a_nx)\ldots)).$$

Uogólniony algorytm Hornera dla wielomianu interpolacyjnego w postaci Newtona rozszerza ten pomysł. Wielomian interpolacyjny stopnia n w postaci Newtona można zapisać jako:

$$p(x) = f[x_0] + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + \ldots + (x - x_0)\ldots(x - x_{n-1})f[x_0, \ldots, x_n],$$

a następnie można te równanie przekształcić do:

$$p(x) = f[x_0] + (x - x_0)(f[x_0, x_1] + (x - x_1)(f[x_0, x_1, x_2] + \ldots + (x - x_{n-1})f[x_0, \ldots, x_n])\ldots))$$

W tym kontekście każdy składnik $(x-x_k)$ jest pomnażany przez odpowiedni iloraz różnicowy $f[x_0, \ldots, x_k]$ i dodawany do dotychczasowego wyniku. Skorzystałem z tej zależności i w dokładnie ten sposób zaimplementowałem moją funkcję. Poniżej umieściłem jej pseudokod:

Algorithm 2 Funkcja obliczająca wartość wielomianu interpolacyjnego w postaci Newtona

```
1: input:
2: x - wektor węzłów x_0 ..., x_n,
3: fx - wektor ilorazów różnicowych f[x_0], f[x_0, x_1], ..., f[x_0, ..., x_n]
4: t - punkt, w którym należy obliczyć wartość wielomianu
5: output: nt - wartość wielomianu w punkcie t
6: function WartośćWielomianu(x, fx, t)
7:     if length(x) ≠ length(fx) or length(x) == 0 then
8:         throw error
9:     end if
10:    n ← length(x)
11:    nt ← fx[n]
12:    for i ← n − 1 down to 1 do
13:        nt = nt · (t − x[i]) + fx[i]
14:    end for
15:    return nt
16: end function
```

Nietrudno jest zauważyć, że powyższy algorytm ma liniową złożoność obliczeniową $O(n)$, gdyż wszystkie operacje wykonują się w stałym czasie $O(1)$ oraz istnieje tylko jedna pętla, która wykonuje się n razy.

3 Współczynniki postaci naturalnej

Celem tego zadania było zaimplementowanie funkcji, która dla otrzymanego wielomianu w postaci Newtona zwraca współczynniki jego postaci naturalnej w casie $O(n^2)$.

Zgodnie z poprzednimi oznaczeniami przyjmijmy, że wielomian p jest w postaci Newtona, czyli:

$$p(x) = \sum_{k=0}^{n} c_k q_k(x) = \sum_{k=0}^{n} [f_0, \ldots, f_k] \prod_{j=0}^{k-1} (x - x_j)$$

Można zauważyć, że współczynnikiem przy x^n jest $c_n = [f_0, \ldots, f_n]$. Jeszcze raz spójrzmy na postać wielomianu p, którą wykorzystaliśmy w algorytmie Hornera, czyli:

$$f[x_0]+(x-x_0)(f[x_0,x_1]+(x-x_1)(f[x_0,x_1,x_2]+\ldots+(x-x_{n-2})(f[x_0,\ldots,x_{n-1}]+(x-x_{n-1})f[x_0,\ldots,x_n])\ldots))$$

Na podstawie tej postaci można utworzyć ciąg wielomianów w_i, gdzie:

$$w_n(x) = f[x_0, \ldots, x_n],$$
$$w_i(x) = f[x_0, \ldots, x_i] + (x - x_i)w_{i+1} \text{ dla } 0 \leq i < n$$

Stosując te wzory, możemy zapisać wielomian p, jako:

$$p(x) = w_0(x) = f[x_0] + (x - x_0)w_1 = f[x_0] + (x - x_0)(f[x_0, x_1] + (x - x_1)w_2) = \ldots$$

Dla wielomianu p w postaci Newtona stopnia n, każdy wielomian w_i jest stopnia $deg(w_i) = n - i$, zatem $deg(w_i) = deg(w_{i+1}) + 1$. Wymnóżmy wyrażenie $w_i(x)$:

$$w_i(x) = c_i - x_i \cdot w_{i+1} + x \cdot w_{i+1}$$

Dla lepszej wizualizacji i zrozumienia problemu, postanowiłem prześledzić kilka konkretnych wielomianów w:

$$w_n = f[x_0, \ldots, x_n] = c_n,$$

$$(1)\ w_{n-1} = \underbrace{c_{n-1} - x_{n-1} \cdot w_n}_{a_{n-1}} + x \cdot \underbrace{w_n}_{a_n}$$

$$(2)\ w_{n-2} = c_{n-2} - x_{n-2} \cdot w_{n-1} + x \cdot w_{n-1} =$$
$$c_{n-2} - x_{n-2} \cdot (c_{n-1} - x_{n-1} \cdot w_n + x \cdot w_n) + x \cdot (c_{n-1} - x_{n-1} \cdot w_n + x \cdot w_n) =$$
$$\underbrace{c_{n-2} - x_{n-2} \cdot \underbrace{(c_{n-1} - x_{n-1} \cdot w_n)}_{\text{not updated } a_{n-1}} + x \cdot \underbrace{(-x_{n-2} \cdot w_n + c_{n-1} - x_{n-1} \cdot w_n)}_{\text{updated } a_{n-1}} + x^2 \cdot \underbrace{w_n}_{a_n}}_{a_{n-2}}$$

$$(3)\ w_{n-3} = c_{n-3} - x_{n-3} \cdot w_{n-2} + x \cdot w_{n-2} =$$
$$c_{n-3} - x_{n-3} \cdot (c_{n-2} - x_{n-2} \cdot w_{n-1} + x \cdot w_{n-1}) + x \cdot (c_{n-2} - x_{n-2} \cdot w_{n-1} + x \cdot w_{n-1}) =$$
$$c_{n-3} - x_{n-3} \cdot (c_{n-2} - x_{n-2} \cdot (c_{n-1} - x_{n-1} \cdot w_n + x \cdot w_n) + x \cdot (c_{n-1} - x_{n-1} \cdot w_n + x \cdot w_n))$$
$$+x \cdot (c_{n-2} - x_{n-2} \cdot (c_{n-1} - x_{n-1} \cdot w_n + x \cdot w_n) + x \cdot (c_{n-1} - x_{n-1} \cdot w_n + x \cdot w_n)) =$$
$$\underbrace{c_{n-3} - x_{n-3} \cdot \underbrace{(c_{n-2} - x_{n-2} \cdot (c_{n-1} - x_{n-1} \cdot w_n))}_{\text{not updated } a_{n-2}} + x \cdot \underbrace{((-x_{n-3} \cdot w_n) + \underbrace{(-x_{n-2} \cdot w_n + c_{n-1} - x_{n-1} \cdot w_n)}_{\text{previous updated } a_{n-1}}}_{\text{updated } a_{n-2}}}_{a_{n-3}}$$
$$+x \cdot w_n) + x \cdot (c_{n-2} - x_{n-2} \cdot (c_{n-1} - x_{n-1} \cdot w_n + x \cdot w_n) + x \cdot (c_{n-1} - x_{n-1} \cdot w_n)) + x^3 \cdot \underbrace{w_n}_{a_n}$$

Jak można zauważyć powyżej, współczynnik przy x^n przyjmuje formę $a_n = c_n$. Kolejnym ważnym spostrzeżeniem jest fakt, że w równaniu (1) współczynnik a_{n-1} przyjmuje postać $a_{n-1} = c_{n-1} - x_{n-1} \cdot a_n$. **Oznaczmy ten wzór jako (*)**.

Przejdźmy teraz do równania (2), w którym to sprowadziłem postać wielomianu w_{n-2} do postaci naturalnej. Wielomian ten jest stopnia o jeden wyższego niż w_{n-1}, lecz współczynnik przy najwyższej potędze nie ulega zmianie. Do obliczenia współczynnika przy a_{n-2} możemy posłużyć się wzorem (*), co sprawdza się w tym przypadku, pod warunkiem, że nie zaktualizowaliśmy współczynnika a_{n-1}.

No właśnie, musimy zaktualizować współczynnik a_{n-1}. Różni się od poprzedniej wartości o dokładnie $-x_{n-2} \cdot a_n$. **Oznaczmy ten wzór jako (!)**. Posłuży on do aktualizowania współczynników wielomianu.

W ostatnim kroku przejdźmy jeszcze do równania (3). W tym przypadku częściowo wyprowadziłem wielomian w_3 do postaci naturalnej. Można zauważyć, że wzory (*) oraz (!) zwracają poprawne wyniki, zatem wykorzystałem je w moim algorytmie, którego pseudokod przedstawiam poniżej:

Algorithm 3 Funkcja obliczająca dla wielomianu w postaci Newtona współczynniki jego postaci naturalnej

```
1:  input:
2:  x - wektor węzłów x_0 ..., x_n,
3:  fx - wektor ilorazów różnicowych f[x_0], f[x_0, x_1], ..., f[x_0, ..., x_n]
4:  output: a - wektor zawierający obliczone współczynniki postaci naturalnej
5:  function wspNaturalne(x, fx)
6:      if length(x) ≠ length(fx) or length(x) == 0 then
7:          throw error
8:      end if
9:      n ← length(x)
10:     a ← initialize array of size n
11:     a[n] ← fx[n]
12:     for i ← n − 1 down to 1 do
13:         a[i] = −(a[i + 1] · x[i]) + fx[i]    // zastosowanie wzoru (*)
14:         for j ← i + 1 to n − 1 do
15:             a[j] = a[j] − (x[i] · a[j + 1])    // zastosowanie wzoru (!)
16:         end for
17:     end for
18:     return a
19: end function
```

Złożoność obliczeniowa tego algorytmu wynosi $O(n^2)$, ponieważ wszystkie operacje wykonywane są w stałym czasie $O(1)$, a ilość operacji w podwójnej pętli wynosi:

$$\sum_{i=1}^{n-1} 1 \sum_{j=i+1}^{n-1} 1 = \sum_{i=1}^{n-1} (n-i-1) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} 1 = n(n-1) - \tfrac{n(n-1)}{2} - (n-1) = O(n^2)$$

4 Funkcja do wizualizacji

Należało napisać funkcję, która zinterpoluje zadaną funkcję $f(x)$ w przedziale $[a, b]$ za pomocą wielomianu interpolacyjnego stopnia n w postaci Newtona, a następnie narysuje wielomian interpolacyjny i interpolowaną funkcję.

Zacznę od przedstawienia mojego pseudokodu:

Algorithm 4 Funkcja do wizualizacji

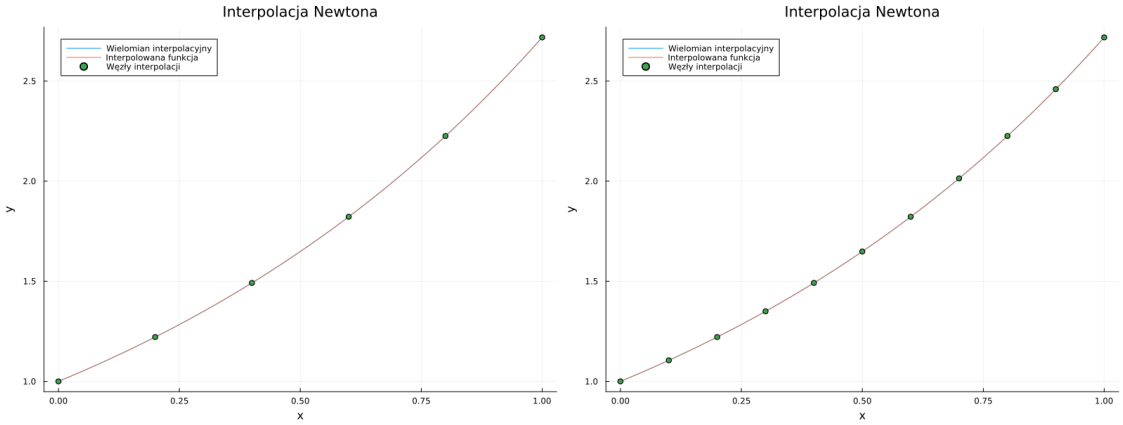
input:
f - funkcja $f(x)$ zadana jako anonimowa funkcja,
a - początek przedziału interpolacji,
b - koniec przedziału interpolacji,
n - stopień wielomianu interpolacyjnego
output: p - wykres wielomianu interpolacyjnego i interpolowanej funkcji na przedziale [a, b]
function RYSUJNNFX(f, a, b, n)
    $x \leftarrow$ **initialize array of size** $n + 1$
    $y \leftarrow$ **initialize array of size** $n + 1$
    $h \leftarrow (b - a)/n$
    // Generowanie węzłów równoodległych
    for $i \leftarrow 1$ **to** $n + 1$ **do**
        $x[i] \leftarrow a = (i - 1) \cdot h$
        $y[i] \leftarrow f(x[i])$
    end for
    $ilorazyRoznicowe \leftarrow IlorazyRoznicowe(x, y)$
    $xAxis \leftarrow$ **assign proper x values for the plot**
    $s \leftarrow length(xAxis)$
    $yPolynomial \leftarrow$ **initialize array of size** s
    $yFunction \leftarrow$ **initialize array of size** s
    for xs **in** $xAxis$ **do**
        $yPolynomial \leftarrow warNewton(x, ilorazyRoznicowe, xs)$
        $yFunction \leftarrow f(xs)$
    end for
    $p \leftarrow xAxis, yPolynomial, yFunction$
    return p
end function

Algorytm generuje węzły równoodległe, a następnie oblicza ilorazy różnicowe za pomocą funkcji z Zadania 1. Później, program oblicza wartości dla anonimowej funkcji dla argumentów osi X oraz wartości dla wielomianu interpolacyjnego, które są liczone za pomocą funkcji z Zadania 2. Na koniec program zwraca gotowy wykres.

5 Testy dla 'porządnych' funkcji

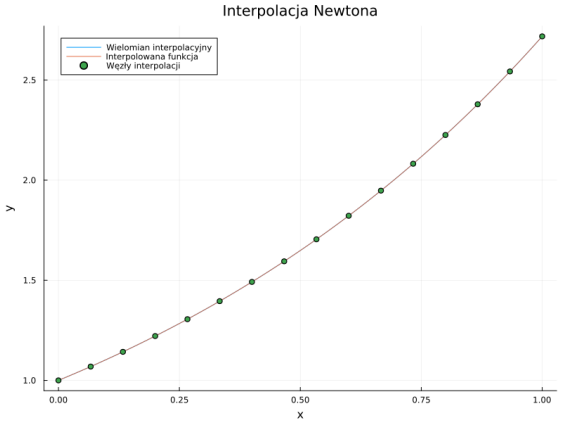
Celem zadania było przetestowanie funkcji *rysujNnfx()* do generowania wykresów opisanej powyżej dla podanych przykładów:

1. $f_1(x) = e^x$, $[a, b] = [0, 1]$, $n \in \{5, 10, 15\}$



(a) n = 5

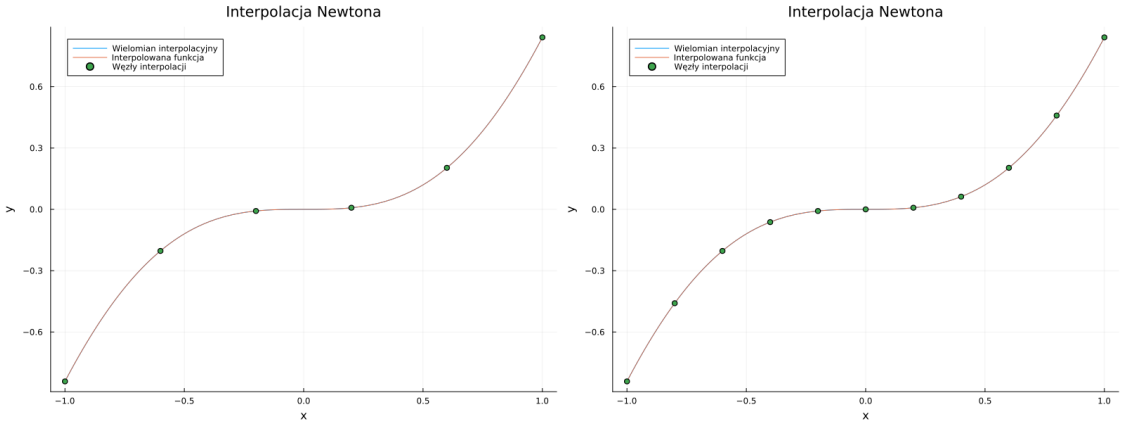
(b) n = 10



(c) n = 15

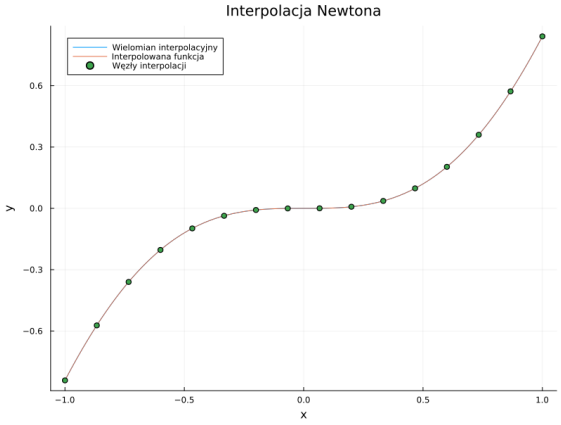
Rysunek 2: Interpolacja funkcji $f_1(x) = e^x$

2. $f_2(x) = x^2 sin(x)$, $[a, b] = [-1, 1]$, $n \in \{5, 10, 15\}$



(a) n = 5

(b) n = 10



(c) n = 15

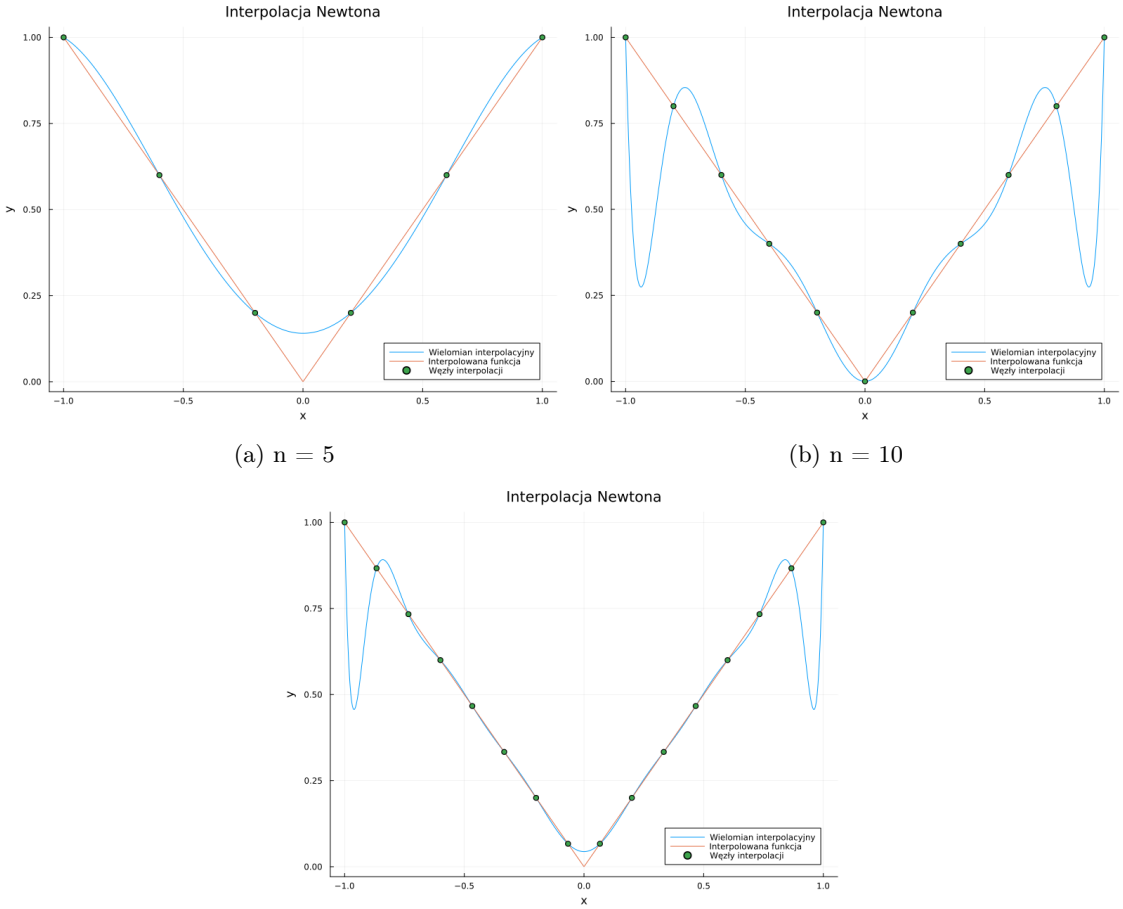
Rysunek 3: Interpolacja funkcji $f_2(x) = x^2 sin(x)$

Wykresy zwrócone przez *rysujNnfx()* pokazują, że obie funkcje f_1 oraz f_2 udało się bardzo dobrze zinterpolować. Wartości wielomianu interpolacyjnego bardzo dokładnie odwzorowują wartości funkcji na zadanym przedziale.

6 Testy dla 'nieporządnych' funkcji

Celem zadania było przetestowanie funkcji *rysujNnfx()* do generowania wykresów opisanej powyżej dla podanych przykładów:

1. $f_1(x) = |x|$, $[a, b] = [-1, 1]$, $n \in \{5, 10, 15\}$



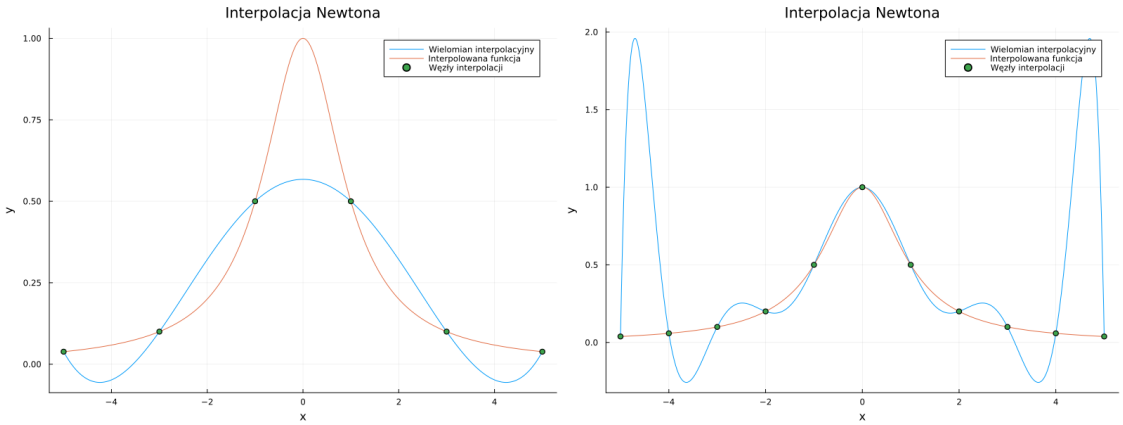
(a) n = 5

(b) n = 10

(c) n = 15

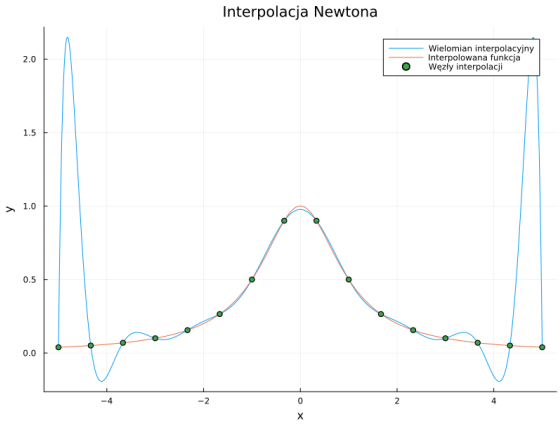
Rysunek 4: Interpolacja funkcji $f_1(x) = |x|$

2. $f_2(x) = \frac{1}{1+x^2}$, $[a, b] = [-5, 5]$, $n \in \{5, 10, 15\}$



(a) n = 5

(b) n = 10



(c) n = 15

Rysunek 5: Interpolacja funkcji $f_2(x) = \frac{1}{1+x^2}$

W porównaniu do wyników z Zadania 5. zastosowanie interpolacji na funkcjach z tego zadania okazało się nie aż tak skuteczne. Dla obu funkcji można zauważyć, że zachodzi zjawisko Runge'go, które dotyczy błędów interpolacji wielomianowej przy równoodległym rozmieszczeniu węzłów interpolacji. W skrócie, może prowadzić do pogorszenia jakości interpolacji, gdy liczba węzłów wzrasta.

W wielomianowej interpolacji równoodległej, im więcej węzłów interpolacji dodamy, tym wyższy stopień wielomianu będzie potrzebny do dokładnego przechodzenia przez te punkty. W przypadku interpolacji za pomocą wielomianów interpolacyjnych, takich jak interpolacja Newtona lub interpolacja Lagrange'a, wysoki stopień wielomianu może prowadzić do zjawiska oscylacji w okolicach końców przedziału, co dzieje się na otrzymanych wykresach.

W praktyce oznacza to, że choć dodanie kolejnych węzłów interpolacji może poprawić dokładność na obszarze, gdzie węzły są zagęszczone, to na końcach przedziału, gdzie węzły są rzadkie, mogą wystąpić duże oscylacje, co prowadzi do błędów interpolacyjnych.

Aby ograniczyć zjawisko Rungego w interpolacji równoodległej, można zastosować kilka technik:

1. **Dobór przedziału:** Zamiast interpolować na całym przedziale, można ograniczyć się do interpolacji w pewnym podprzedziale, gdzie funkcja jest dobrze zachowana. Wybierając przedział, w którym funkcja jest mniej skomplikowana lub ma mniejsze oscylacje, można zminimalizować wpływ zjawiska Runge'go.

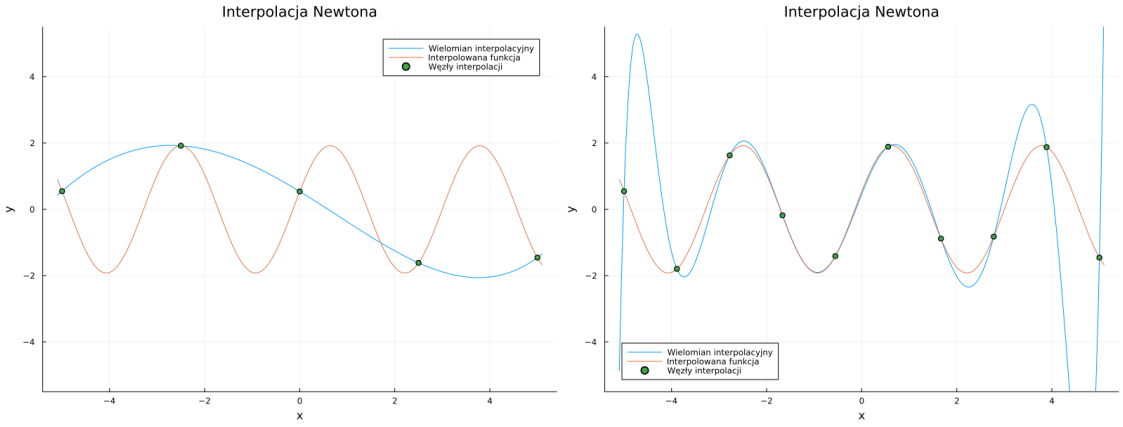
2. **Zwiększenie liczby węzłów na krańcach:** W miejscach, gdzie zjawisko Rungego jest bardziej widoczne (na końcach przedziału), można użyć większej liczby węzłów. To pozwoli na lepszą lokalną reprezentację funkcji w tych obszarach.

3. **Użycie interpolacji wielomianowej niższego stopnia:** Czasem lepszym rozwiązaniem jest użycie wielomianów niższego stopnia, które mniej podlegają zjawisku oscylacji.

7 Testy dla funkcji *naturalna()* z Zadania 3.

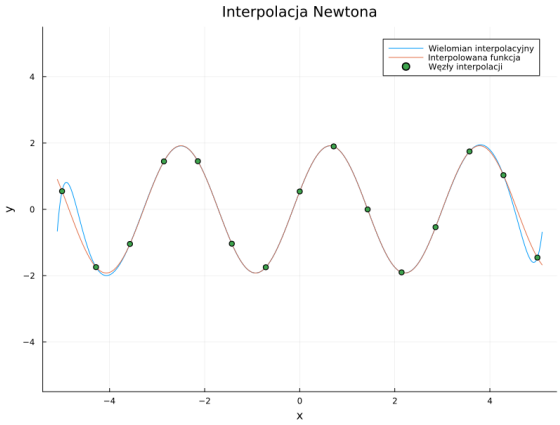
Postanowiłem wykonać parę graficznych testów dla funkcji *naturalna()* i umieściłem swój kod źródłowy tej części w pliku **tests_nat.jl**. W moim programie obliczałem współczynniki wielomianu w postaci naturalnej wielomianu interpolacyjnego stopnia n w postaci Newtona, a następnie porównywałem wykres otrzymanego wielomianu z interpolowaną funkcją. Poniżej znajdują się wykresy dla trzech przykładowych funkcji dla różnych wartości n:

1. $f_1(x) = sin(2x) + cos(2x - 1)$, $n \in \{5, 10, 15\}$



(a) n = 5

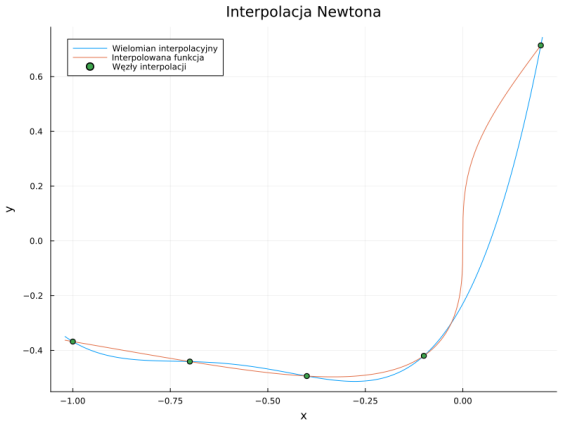
(b) n = 10



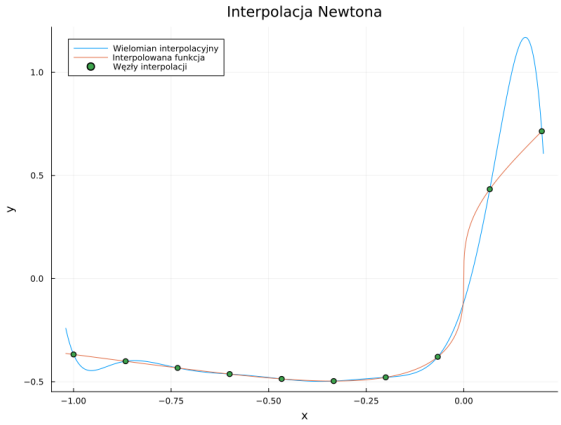
(c) n = 15

Rysunek 6: Wyniki dla $f_1(x) = sin(2x) + cos(2x - 1)$

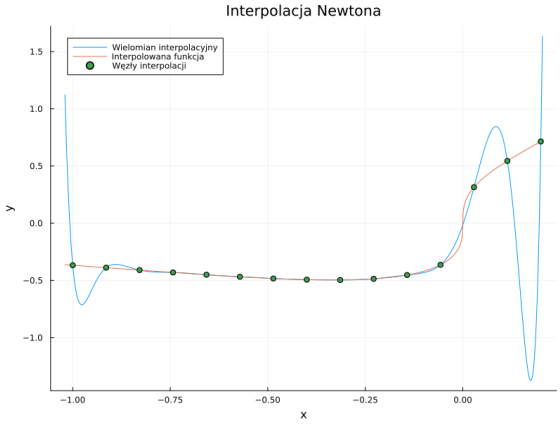
2. $f_2(x) = \sqrt[3]{x} \cdot e^x$, $n \in \{5, 10, 15\}$



(a) n = 5



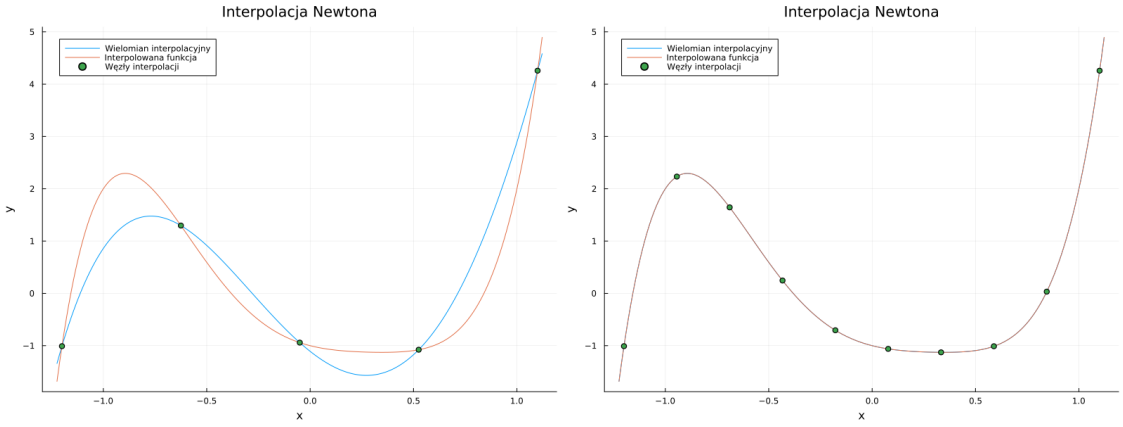
(b) n = 10



(c) n = 15

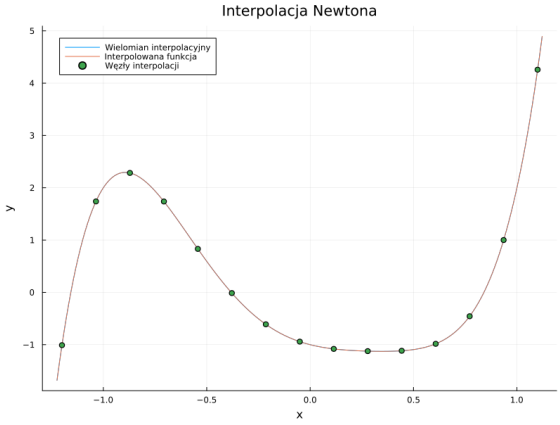
Rysunek 7: Wyniki dla $f_2(x) = \sqrt[3]{x} \cdot e^x$

3. $f_3(x) = 5x^5 - 4x^3 + 3x^2 - 2x + x - 1$, $n \in \{5, 10, 15\}$



(a) n = 5

(b) n = 10



(c) n = 15

Rysunek 8: Wyniki dla $f_3(x) = 5x^5 - 4x^3 + 3x^2 - 2x + x - 1$

Patrząc na otrzymane wyniki można stwierdzić, że wykresy dla wielomianów powstałych z współczynników zwróconych przez funkcję *naturalna()* bardzo dobrze aproksymują rzeczywiste wartości interpolowanych funkcji. Można zatem wnioskować, że funkcja ta jest zaimplementowana poprawnie.