Sprawozdanie Obliczenia Naukowe

Piotr Zapała

November 2023

Spis treści

1 Zadanie 1

1.1 Opis problemu

W zadaniu pierwszym jesteśmy proszeni o zaimplementowanie funkcji która oblicza ilorazy różnicowe.

`function ilorazyRoznicowe(x::VectorFloat64, f::VectorFloat64)`

Nasza funkcja powinna przyjmować jako argumenty następujące parametry:

Dane:
x - wektor długości $n + 1$ zawierający węzły $x_0, \ldots, x_n$, $x[1] = x_0, \ldots, x[n + 1] = x_n$
f - wektor długości $n + 1$ zawierający wartości interpolowanej funkcji w węzłach $f(x_0), \ldots, f(x_n)$.

Zaimlementowana funkcja `"ilorazyRoznicowe"` powinna zwracać następujące wyniki:

Wyniki:
fx - wektor długości $n + 1$ zawierający obliczone ilorazy różnicowe
$fx[1] = f[x_0], fx[2] = f[x_0, x_1], \ldots, fx[n] = f[x_0, \ldots, x_{n-1}], fx[n + 1] = f[x_0, \ldots, x_n]$.

Autor powyższego zadania wymaga aby nasza funkcja została zaimplementowana bez użycia tablicy dwuwymiarowej (macierzy).

1.2 Rozwiązanie

W celu obliczania ilorazów różnicowych wyższych rzędów, należy się posłużyć poniższym rekurencyjnym wzorem.

$$f[x_i, x_{i+1}, \ldots, x_{i+j}] = \frac{f[x_i, x_{i+2}, \ldots, x_{i+j}] - f[x_i, x_{i+1}, \ldots, x_{i+j-1}]}{x_{i+j} - x_i}$$

Jeśli znamy węzły x_i oraz wartości funkcji $f(x_i)$, czyli ilorazy zerowego rzędu, jesteśmy w stanie za pomocą powyższego wzoru utworzyć tablicę ilorazów różnicowych wyższych rzędów.

Przykładowa tablica ilorazów w przypadku, gdy znamy cztery węzły może wyglądać następująco:

x_0 $f[x_0]$ | $f[x_0, x_1]$ $f[x_0, x_1, x_2]$ $f[x_0, x_1, x_2, x_3]$
x_1 $f[x_1]$ | $f[x_1, x_2]$ $f[x_1, x_2, x_3]$
x_2 $f[x_2]$ | $f[x_2, x_3]$
x_3 $f[x_3]$ |

Pionowa kreska oddziela wielkości dane od obliczonych.

Pierwszy wierz tablicy zawiera iloraz, które wystepują we wzorze interpolacyjnym Newtona.

Jeżeli przyjmiemy, że $c_{ij} = f[x_i, x_{i+1}, \ldots, x_{i+j}]$, nasza tablica ilorazów różnicowych ma następującą postać:

$$\begin{array}{ll|lllll}
x_0 & c_{00} & c_{01} & c_{02} & \ldots & c_{0,n-1} & c_{0n} \\
x_1 & c_{10} & c_{11} & c_{12} & \ldots & c_{1,n-1} & \\
\ldots & \ldots & \ldots & \ldots & & & \\
x_{n-1} & c_{n-1,0} & c_{n-1,1} & & & & \\
x_n & c_{n0} & & & & &
\end{array}$$

Nasz wzór rekurencyjny przyjmuje następującą postać:

$$c_{ij} = \frac{c_{i+1,j-1} - c_{i,j-1}}{x_{i+j} - x_i}$$

Zgodnie z wymaganiami autora, algorytm obliczania ilorazów różnicowych powinien wykonywać obliczenia bez użycia tablicy dwuwymiarowej. Aby osiągnąć wymagany efekt należy użyć tablicy d, początkową wartością zmiennej d_i jest $c_0 = f(x_i)$ z drugiej kolumny trójkątnej tablicy ilorazów, następnymi wartościami - wielkości $c_{i-1,1}, \ldots, c_{1,i-1}, c_{0i}$. Aby tablica d zawierała w każdej chwili ilorazy, które będą potrzebne później, należy tworzyć naszą tablicę kolumnami, a w każdej kolumnie - z dołu do góry.

Algorithm 1 ilorazyRoznicowe

```
function ILORAZYROZNICOWE(x, f)
    n ← length(f)
    for i ← 0, n do
        d[i] ← f(x_i)
    end for
    for j ← 1, n do
        for i ← n, j do
            d_i ← (d_i − d_{i−1})/(x_i − x_{i−j})
        end for
    end for
return d
end function
```

2 Zadanie 2

2.1 Opis problemu

W zadaniu drugim należy napisać funkcję obliczającą wartość wielomianu interpolacyjnego stopnia n w postaci Newtona $N_n(x)$ w punkcie $x = t$ za pomocą uogólnionego algorytmu Hornera, w czasie $O(n)$.

`function warNewton(x::VectorFloat64, fx::VectorFloat64, t::Float64)`

Nasza funkcja powinna przyjmować jako argumenty następujące parametry:

Dane:

x - wektor długości $n+1$ zawierający węzły $x_0, \ldots, x_n$, $x[1] = x_0, \ldots, x[n+1] = x_n$
fx - wektor długości $n+1$ zawierający obliczone ilorazy różnicowe
$fx[1] = f[x_0], fx[2] = f[x_0, x_1], \ldots, fx[n] = f[x_0, \ldots, x_{n-1}], fx[n+1] = f[x_0, \ldots, x_n]$.
t - punkt, w którym należy obliczyc wartość wielomianu

Zaimlementowana funkcja "`warNewton`" powinna zwracać następujący wynik:

Wyniki:

nt - wartość wielomianu w punkcie t.

2.2 Rozwiązanie

Rozwiązanie zadania polega na zaimplementowaniu uogólnionego algorytmu Hornera, w czasie $O(n)$. Schemat Hornera jest algorytmem służącym do bardzo szybkiego obliczania wartości wielomianu. Mamy wielomian w postaci naturalnej $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$, gdzie współczynniki $a_0, a_1, \ldots, a_n$ mogą być liczbami rzeczywistymi lub zespolonymi, natomiast x jest zmienną niezależną. Naszym zadaniem jest znaleźć wartość $p(c) =?$, w dowolnym punkcie rzeczywistym $x = c$.

Ilość operacji jakie należy wykonać aby wyliczyć wartośc $p(c)$ to:

- n dodawań
- n mnożeń $(a_0 * x^n, a_1 * x^{n-1}, \ldots, a_{n-1} * x)$
- $n-1$ mnożeń $(x^2, x^3, \ldots, x^n)$

W sumie do obliczenia wartości wielomian $p(x)$ w punkcie $x = c$ potrzeba $3n - 1$ operacji.

Z pomocą algorytmu Hornera jesteśmy w stanie zmniejszyć ilość wykonywanych operacji.
Po podzieleniu wielomianu $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$ przez dwumian $x - c$ otrzymujemy:
$p(x) = (x - c) * q(x) + b_n$, gdzie $q(x) = b_0x^{n-1} + b_1x^{n-2} + \cdots + b_{n-1}$,
oraz b_n jest resztą z dzielenia $p(x)$ przez $x - c$, $b_n = p(c)$.

Z powyższych wzorów wynika, że

$$a_0x^n + a_1x^{n-1} + \cdots + a_n = (x - c)(b_0x^{n-1} + b_1x^{n-2} + \cdots + b_{n-1}) + b_n =$$
$$= b_0x^n + (-b_0 * c + b_1)x^{n-1} + (-b_1 * c + b_2)x^{n-2} + \cdots + (-b_{n-1} * c + b_n)$$

Jeśli powrównamy współczynniki przy $x^i, i = n, \ldots, 0$, otrzymamy następujący algorytm:

$a_0 = b_0$
$a_1 = -b_0 * c + b_1$
$a_2 = -b_1 * c + b_2$
$\vdots$
$a_n = -b_{n-1} * c + b_n$

Stąd otrzymujemy:

$b_0 = a_0$
$b_i = b_{i-1} * c + a_1, i = 1, 2, \ldots, n$
$b_n = p(c)$

Koszt algorytmu to $2n$ działań, mamy n mnożeń i n dodawań.

Powyższy algorytm można bez jakichkolwiek problemów stosować do obliczenia wartości wielomianu w zadanym punkcie, natomiast warto zwrócić uwagę, iż wielomian musi być w postaci naturalnej. W naszym zadaniu musimy obliczyć wartość wielomianu interpolacyjnego Newtona, który jest w następującej postaci:

$$p(x) = a_0 + \sum_{i=1}^{n} a_i \prod_{j=0}^{i-1} (x - x_j) = a_0 + a_1(x - x_0) + a_2(x - x_1)(x - x_0) + \cdots + a_n(x - x_{n-1}) \cdots (x - x_1)(x - x_0)$$

Gdzie $a_0, a_1, \ldots, a_n$, to ilorazy różnicowe, a $x_0, x_1, \ldots, x_{n-1}$ węzły interpolacyjne.
Zatem powyższą zależność można przedstawić używając ogółnionych wzorów Hornera.

$$w_n(x) = f[x_0, \ldots, x_n]$$
$$w_k = f[x_0, \ldots, x_k] + (x - x_k) * w_{k+1}(x),\ k \in [n-1, 0]$$

Powyższe zależnośći możemy już w łatwy sposób przełożyć na kod, który prezentuje się następująco:

Algorithm 2 warNewton

```
function WARNEWTON(x, fx, t)
    n ← length(fx)
    b ← fx[n]
    for i ← n − 1, 1 do
        b ← fx[i] + (t − x[i]) ∗ b
    end for
return b
end function
```

3 Zadanie 3

3.1 Opis problemu

Zadanie trzecie polega na zaimplementowaniu funkcji obliczającej współczynniki wielomianu w postaci naturalnej $a_0, \ldots, a_n$ tzn. $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$. W celu obliczenia współczynników w postaci naturalnej należy posłużyć się współczynnikami wielomianu interpolacyjnego w postaci Newtona $c_0 = f[x_0], c_1 = f[x_0, x_1], c_2 = f[x_0, x_1, x_2], \ldots, c_n = f[x_0, \ldots, x_n]$ oraz węzłami $x_0, x_1, \ldots, x_n$.

Autor zadania wymaga aby nasza funkcja działała w czasie $O(n^2)$.

`function naturalna(x::VectorFloat64, fx::VectorFloat64)`

Nasza funkcja powinna przyjmować jako argumenty następujące parametry:

Dane:

x - wektor długości $n + 1$ zawierający węzły $x_0, \ldots, x_n$, $x[1] = x_0, \ldots, x[n + 1] = x_n$

fx - wektor długości $n + 1$ zawierający obliczone ilorazy różnicowe
$fx[1] = f[x_0], fx[2] = f[x_0, x_1], \ldots, fx[n] = f[x_0, \ldots, x_{n-1}], fx[n + 1] = f[x_0, \ldots, x_n]$.

Zaimplementowana funkcja `"naturalna"` powinna zwracać następujący wynik:

Wyniki:

a - wektor długości $n + 1$ zawierający obliczone współczynniki postaci naturalnej
$a[1] = a_0, a[2] = a_1, \ldots, a[n] = a_{n-1}, a[n + 1] = a_n$.

3.2 Rozwiązanie

Jak się okazuję, jeżeli zaczniemy się cofać jak algorytmie Hornera, jesteśmy w stanie wyliczyć wartość przy obecnej potędze. Przykładowo dla x^i mamy $c_i - x_i a_{i+1}$, następnie należy zaktualizować współczynniki przy wyższych potęgach naszego wielomianu. Można zauważyć, iż dla każdego a_j w i-tej iteracji, dodajemy $-x_{n-1}a_{j+1}$. Powyższe rozważania prowadzą nas do zaprojektowania algorytmu wyliczającego współczynniki w postaci naturalnej.

Algorithm 3 naturalna

```
function NATURALNA(x, f)
    n ← length(x)
    for i ← 0, n do
        a[i] ← 0
    end for
    a[n] ← fx[n]
    for i ← (n − 1), 1 do
        a[i] ← fx[i] − x[i] ∗ a[i + 1]
        for j ← (i + 1), (n − 1) do
            a[j] ← a[j] − x[i] + a[j + 1]
        end for
    end for
return a
end function
```

4 Zadanie 4

4.1 Opis problemu

W zadaniu czwartym jesteśmy proszeni aby zaimplementować funkcję, która zinterpoluje zadaną funkcję $f(x)$ za pomocą wielomianu interpolacyjnego stopnia n w postaci Newtona.

Następnie narysuje wielomian interpolacyjny i interpolowaną funkcję. W celu narysowania wykresu naszej funkcji można się posłużyć następującymi pakietami `Plots, PyPlot lub Gadfly`.

W interpolacji należy użyć węzłów równoległych, tj. $x_k = a + kh, h = \frac{(b-a)}{n}, k = 0, 1, \ldots, n$.

Autor zadania wymaga, abyśmy nie wyznaczali wielomianu interpolacyjnego w postaci jawnej.

Powinniśmy skorzystać z funkcji `ilorazyRoznicowe` i `warNewton`.

`function rysujNnfx(f, a::Float64, b::Float64, n::Int)`

Nasza funkcja powinna przyjmować jako argumenty następujące parametry:

Dane:

f - funkcja $f(x)$ zadana jako anonimowa funkcja

a, b - przedział interpolacji

n - stopień wielomianu interpolacyjnego

Zaimlementowana funkcja `"rysujNnfx"` powinna zwracać następujący wynik:

Wyniki:

- zaimplementowana funkcja rysuje wielomian i interpolowaną funkcję w przedziale $[a, b]$

4.2 Rozwiązanie

Celem zadania czwartego jest połączenie zaimplementowanych metod w program pozwalający na rysowanie wykresów funkcji podanych do interpolacji, jak również rysowanie zinterpolowanej funkcji. Autor zadania zarzyczył sobie, aby nie wyznaczać wielomianu interpolacyjnego w jawnej postaci, stąd potrzeba użycia zaimplementowanych metod. Funkcja powinna wykonać interpolacje na podanym przedziale $[a, b]$, dodatkowo należy użyć węzłów równoległych, stąd $x_k = a + kh, h = \frac{(b-a)}{n}, k = 0, 1, \ldots, n$.

Dzięki możliwości rysowania wykresów, osoba korzystająca z programu, jest w stanie porównać reprezentację graficzną danego problemu.

Poniższy pseudokod obrazuję sposób wykonania zadania.

Algorithm 4 rysujNnfx

```
function RYSUJNNFX(f, a, b, n)
    h ← (b−a)/n
    for j ← 0, n do
        x[j] ← 0
        y[j] ← 0
    end for
    for k ← 0, n do
        x[k + 1] ← a + k ∗ h
        y[k + 1] ← f(x[k + 1])
    end for
    x ← ilorazyRoznicowe(x, y)
    points ← 50 ∗ (n + 1)
    dx ← (b−a)/(points−1)
    for z ← 0, points do
        func[z] ← 0
        xs[z] ← 0
        polynomial[z] ← 0
    end for
    xs[1] ← a
    polynomial[1] ← func[1] ← y[1]
    for i ← 2, points do
        xs[i] ← xs[i − 1] + dx
        polynomial[i] ← warNewton(x, c, xs[i])
        func[i] ← f(xs[i])
    end for
    p ← plot(xs, [polynomial, func])
    dispaly(p) return p
end function
```

5 Zadanie 5

5.1 Opis problemu

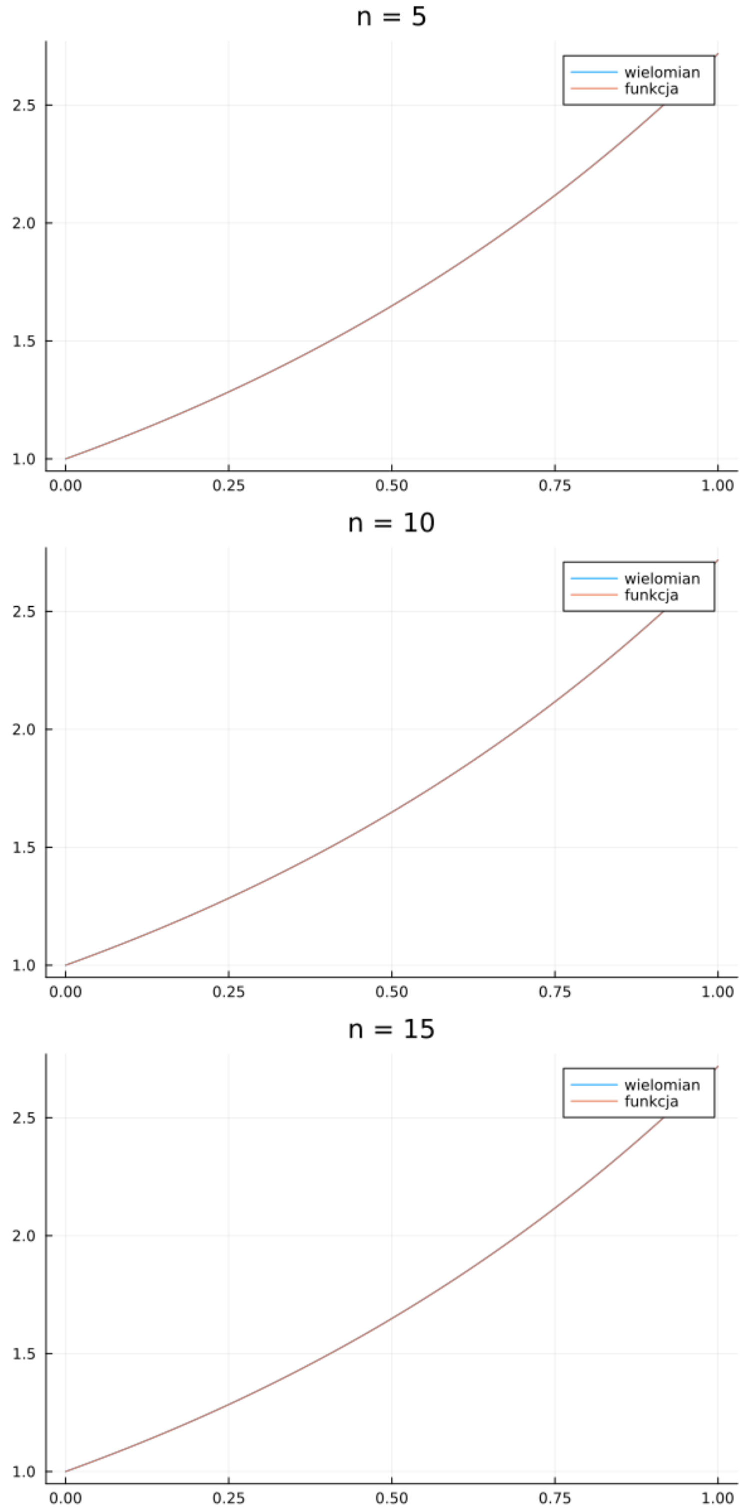
Zadanie piąte polega na przetestowaniu funkcji `rysujNnfx(f,a,b,n)` na następujących przykładach:

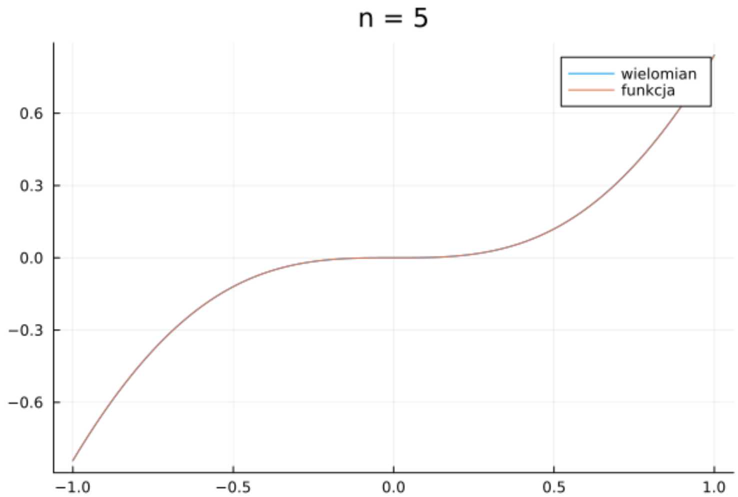
(a) $e^x, [0, 1], n = 5, 10, 15$
(b) $x^2 \sin(x), [-1, 1], n = 5, 10, 15$

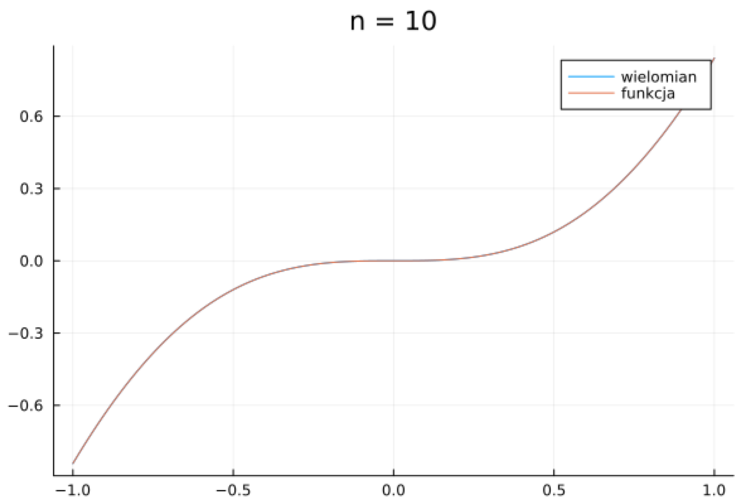
5.2 Rozwiązanie

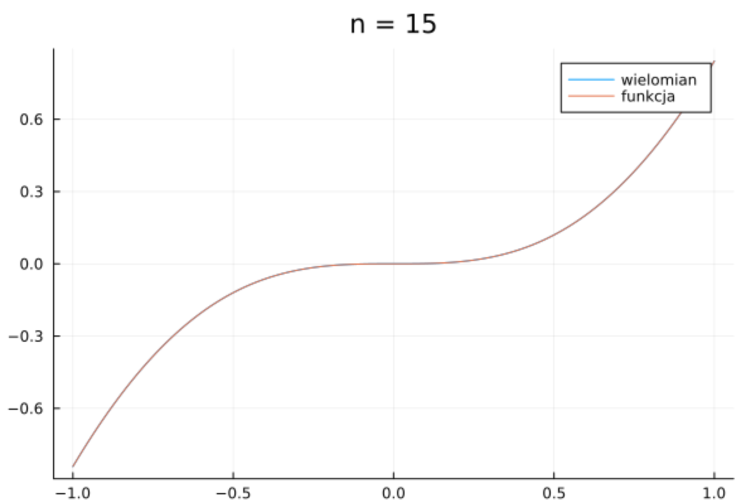
Rozwiązanie powyższego zadania polega, na zaimplementowaniu opowiedniego programu testującego, który wykorzystuje funkcję z zadania 4, oraz w którym parametrami zadanymi do funkcji są dane podane przez autora zadania.

5.3 Wyniki


n = 5
wielomian
funkcja
2.5
2.0
1.5
1.0
0.00
0.25
0.50
0.75
1.00
n = 10
wielomian
funkcja
2.5
2.0
1.5
1.0
0.00
0.25
0.50
0.75
1.00
n = 15
wielomian
funkcja
2.5
2.0
1.5
1.0
0.00
0.25
0.50
0.75
1.00







5.4 Wnioski

Przyglądając się powyższym wykresom, możemy z całą pewnością stwierdzić, iż udało się nam bardzo zinterpolować zadane funkcję. Wykresy funkcji oraz wielomianu interpolacyjnego niemal się pokrywają, co ciekawe udało się nam to dokonać dla względnie małych stopni tego wielomianu.

6 Zadanie 6

6.1 Opis problemu

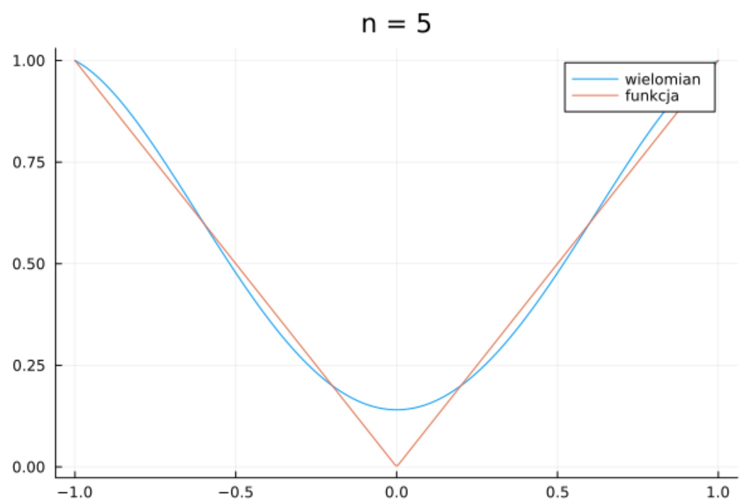
Zadanie szóste polega na przetestowaniu funkcji `rysujNnfx(f,a,b,n)` na następujących przykładach:

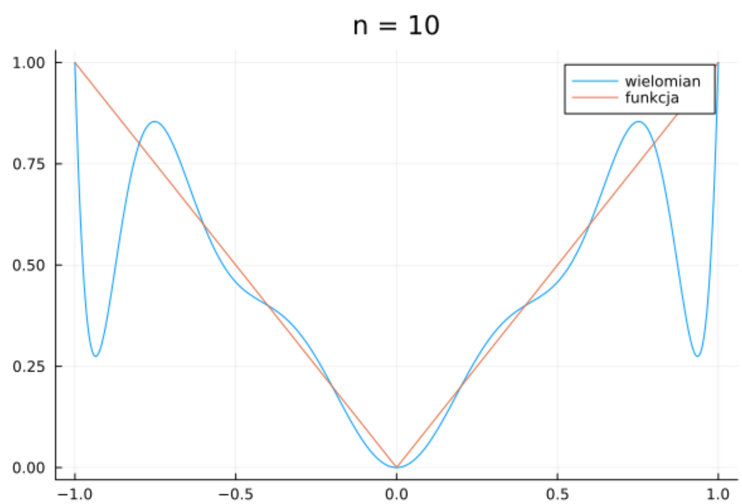
(a) $|x|, [-1, 1], n = 5, 10, 15$
(b) $\frac{1}{1+x^2}, [-5, 5], n = 5, 10, 15$

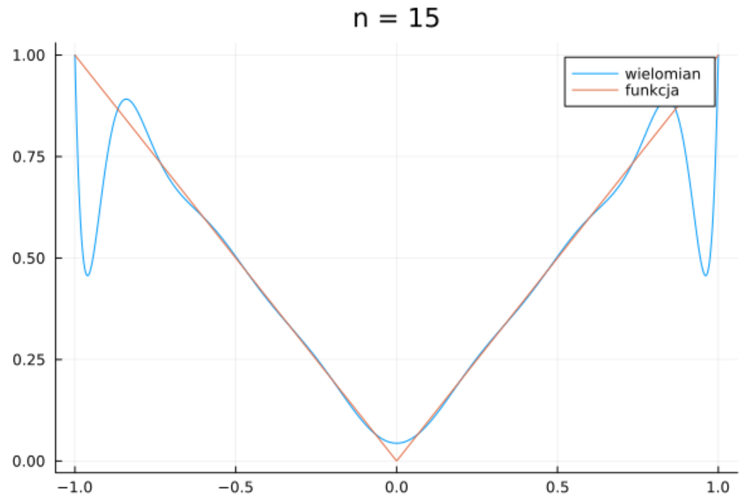
6.2 Rozwiązanie

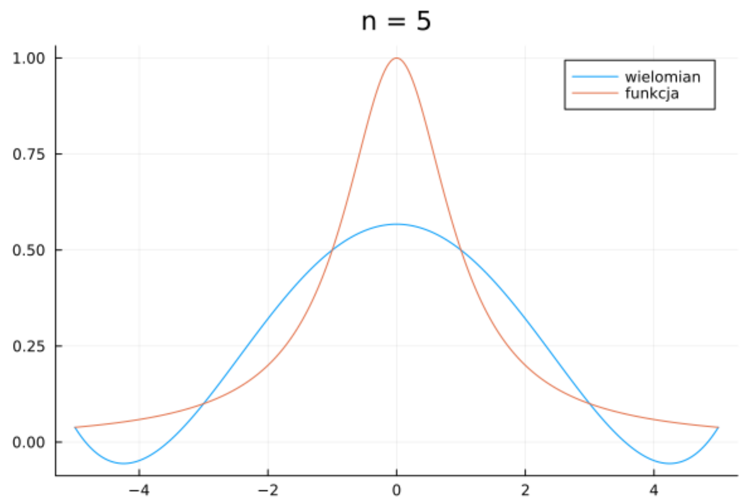
Rozwiązanie zadania szóstego polega, na zaimplementowaniu opowiedniego programu testującego, który wykorzystuje funkcję z zadania 4, oraz w którym parametrami zadanymi do funkcji są dane podane przez autora zadania.

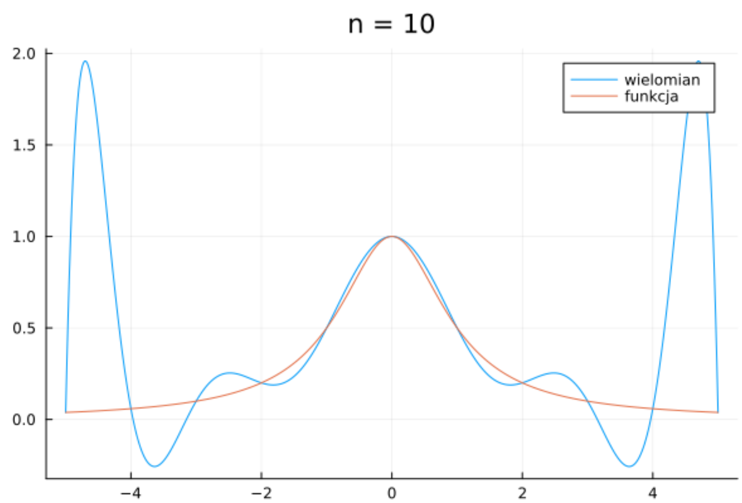
6.3 Wyniki

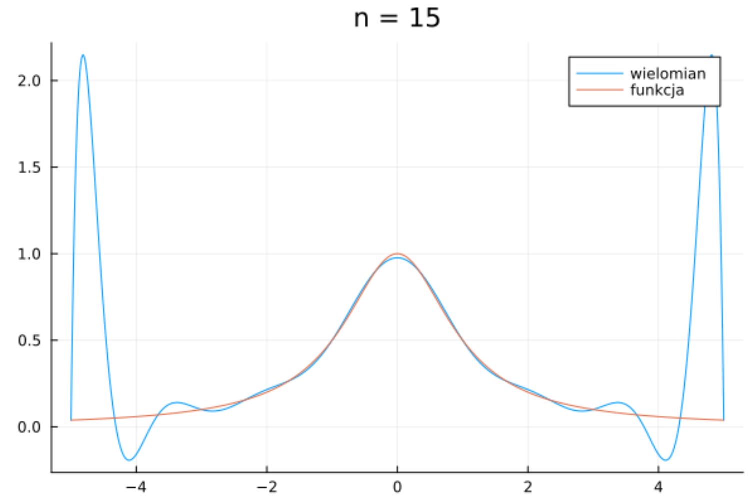





n = 15
wielomian
funkcja
1.00
0.75
0.50
0.25
0.00
−1.0
−0.5
0.0
0.5
1.0


n = 5
wielomian
funkcja
1.00
0.75
0.50
0.25
0.00
−4
−2
0
2
4


n = 10
wielomian
funkcja
2.0
1.5
1.0
0.5
0.0
−4
−2
0
2
4



6.4 Wnioski

Z całą pewnością możemy stwierdzić, że interpolacja wielomianowa jest dobrą metodą przybliżania funkcji, niemniej jednak należy pomiętać o jej ograniczeniach, gdyż nie w każdym przypadku możemy otrzymać oczekiwany wynik. W przypadku funkcji z zadania piątego, nasz metoda poradziła sobie wzorowo, lecz w przypadku funkcji takich jak w zadaniu szóstym. Zwyczajnie funkcje z zadania szóstego posiadają bardzo ostre kształy, które z pewnością sprawiają bardzo dużo problemów w sytuacji interpolowania. Zatem należy się zastanowić, czy ta metoda aby na pewno jest odpowiednia do tego typu funkcji.

Literatura

[1] D. Kincaid, W. Cheney, Analiza numeryczna, Wydawnictwa Naukowo-Techniczne, Warszawa 2006. ISBN 83-204-3078-X