Obliczenia naukowe
Lista 4

Dominik Kaczmarek, nr albumu 261757

11 grudnia 2022

Spis treści

1 Cel

Głównym celem listy 4 jest implementacja algorytmu, który przy użyciu **interpolacji wielomianowej** będzie przybliżał zadaną funkcję `f`.
Mając $n + 1$ punktów (x_i, y_i), gdzie $i = 0, 1, \ldots, n$ oraz $f(x_i) = y_i$ postaramy się wyznaczyć wielomian $p \in \prod_n$, którego wartości dla argumentów (*węzłów*) x_i będą pokrywać się z wartościami y_i tj.

$$p(x_i) = f(x_i) = y_i, \text{ dla } i = 0, 1, \ldots, n$$

Na mocy `Twierdzenia 1. (Wykład 6)` wiemy, że istnieje dokładnie jeden taki wielomian $p \in \prod_n$ spełaniający powyższe założenia.
Aby algorytm działał poprawnie, kluczową kwestią będzie odpowiednie przedstawienie wielomianu p. Chodzi o to, żeby wielomianu p **nie** reprezentować jako kombinacji liniowej wielomianów $1, x, x^2, \ldots, x^n$, tj.

$$p(x) = \mathbf{a_0} + \mathbf{a_1}x + \mathbf{a_2}x^2 + \cdots + \mathbf{a_n}x^n$$

Musielibyśmy wtedy obliczyć $n + 1$ współczynników $a_0, a_1, \ldots, a_n$, co prowadzi do układu równań z macierzą *Vandermonde'a*, która jest źle uwarunkowana. (W zadaniu 3. obliczymy $a_0, a_1, \ldots, a_n$, ale przy użyciu innej metody)
Zamiast tego skorzystamy z *postaci Newtona wzoru interpolacyjnego*

$$p(x) = \sum_{k=0}^{n} c_k q_k(x) = \sum_{k=0}^{n} f[x_0, x_1, \ldots, x_k] \prod_{j=0}^{k-1} (x - x_j),$$

gdzie $c_k = f[x_0, x_1, \ldots, x_k] (k = 0, 1, \ldots, n)$ są ilorazami różnicowymi, a q_k to iloczyny postaci

$$\begin{aligned}
q_0(x) &= 1 \\
q_1(x) &= (x - x_0) \\
q_2(x) &= (x - x_0)(x - x_1) \\
&\vdots \\
q_n(x) &= (x - x_0)(x - x_1) \ldots (x - x_{n-1})
\end{aligned}$$

W zadaniu 1. zajmiemy się wyznaczeniem ilorazów różnicowych, a w zadaniu 2. przy ich użyciu skonstruujemy algorytm wyliczający wartość wielomianu $p(x)$ posatci Newtona.

2 Zadanie 1

2.1 Opis problemu

Napisać funkcję obliczającą ilorazy różnicowe bez użycia tablicy dwuwymiarowej (macierzy).

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

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 $f(x_0), \ldots, f(x_n)$

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

2.2 Rozwiązanie

Chcąc skorzystać ze wzoru *Newtona*

$$p(x) = \sum_{k=0}^{n} c_k q_k(x) = \sum_{k=0}^{n} f[x_0, x_1, \ldots, x_k] \prod_{j=0}^{k-1} (x - x_j),$$

musimy najpierw wyznaczyć ilorazy różnicowe $c_k = f[x_0, x_1, \ldots, x_k]$. Wiemy, że skoro p spełnia warunki interpolacji to

$$p(x_i) = \sum_{k=0}^{n} c_k q_k(x) = f(x_i)$$

Na tej podstawie moglibyśmy wyznaczyć $c_0, c_1, \ldots, c_k$ rozwiązując układ równań

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ 1 & q_1(x_1) & 0 & 0 & \ldots & 0 \\ 1 & q_1(x_2) & q_2(x_2) & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & q_1(x_n) & q_2(x_n) & q_3(x_n) & \ldots & q_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}$$

My jednak chcemy zrobić to sprytniej korzystając z tablicy jednowymiarowej zamiast tablicy dwuwymiarowej (macierzy). Możemy zauważyć, że c_0 zależy od $f(x_0)$, c_1 zależy od $f(x_0)$ oraz $f(x_1)$, ... , c_n zależy od $f(x_0), f(x_1), f(x_2) \ldots, f(x_n)$. Stąd c_k oznaczyliśmy jako $f[x_0, x_1, \ldots, x_k]$. Widzimy również, że $f[x_0] = f(x_0)$. Reguła ta zachodzi dla wszystkich $f[x_k]$ $(0 \leqslant k \leqslant n)$, stąd otrzymujemy

$$f[x_k] = f(x_k), \text{ dla } 0 \leqslant k \leqslant n$$

Wykorzystamy ten fakt do tworzenia ilorazów różnicowych wyższych rzędów. Chcąc obliczyć c_k dla $k = 1, \ldots, n$ skorzystamy z `Twierdzenia 3. (Wykład 6)`, które mówi nam, że ilorazy różnicowe spełniają równość:

$$f[x_0, x_1, \ldots, x_n] = \frac{f[x_1, x_2, \ldots, x_n] - f[x_0, x_1, \ldots, x_{n-1}]}{x_n - x_0}$$

Mając wektory `x` oraz `f` długości $n+1$, które reprezentują nam punkty $(x_k, f(x_k))$ możemy wyliczyć c_k konstruując tablicę trójkątną (poniżej przykład dla $n = 3$).

$$\begin{array}{c|llll|} x_0 & f[x_0] \to & f[x_0, x_1] \to & f[x_0, x_1, x_2] \to & f[x_0, x_1, x_2, x_3] \\ x_1 & f[x_1] \overset{\nearrow}{\to} & f[x_1, x_2] \overset{\nearrow}{\to} & f[x_1, x_2, x_3] \nearrow & \\ x_2 & f[x_2] \overset{\nearrow}{\to} & f[x_2, x_3] \nearrow & & \\ x_3 & f[x_3] \nearrow & & & \end{array}$$

Algorytm na wyliczenie ilorazów różnicowych prezentuje się następująco:

Algorithm 1: `ilorazyRoznicowe (x,f)`

Data: $\tilde{x}$ - wektor węzłów $x_0, \ldots, x_n$; $\tilde{f}$ - wektor $f(x_0), \ldots, f(x_n)$
Result: $\tilde{c}$ - wektor ilorazów różnicowych
$n \leftarrow length(x)$;
$\tilde{c} \leftarrow \tilde{f}$;
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow n$ **to** i **do**
    $c_j = \frac{c_j - c_{j-1}}{x_j - x_{j-i}}$;
  end
end
return $\tilde{c}$;

Dla wizualizacji działania algorytmu prześledźmy iteracje dla `x` i `f` długości $n = 4$.

i	j	c_0	c_1	c_2	c_3
-	-	$f[x_0]$	$f[x_1]$	$f[x_2]$	$f[x_3]$
1	3	$f[x_0]$	$f[x_1]$	$f[x_2]$	$f[x_2,x_3]$
1	2	$f[x_0]$	$f[x_1]$	$f[x_1,x_2]$	$f[x_2,x_3]$
1	1	$f[x_0]$	$f[x_0,x_1]$	$f[x_1,x_2]$	$f[x_2,x_3]$
2	3	$f[x_0]$	$f[x_0,x_1]$	$f[x_1,x_2]$	$f[x_1,x_2,x_3]$
2	2	$f[x_0]$	$f[x_0,x_1]$	$f[x_0,x_1,x_2]$	$f[x_1,x_2,x_3]$
3	3	$f[x_0]$	$f[x_0,x_1]$	$f[x_0,x_1,x_2]$	$f[x_0,x_1,x_2,x_3]$

Tabela 1: Wartości tablicy c (`fx`) w kolejnych iteracjach

Widzimy, że możemy skorzystać z tablicy jednowymiarowej, ponieważ na wyjściu interesują nas tylko wartości $f[x_0,\ldots,x_k] = c_k$ i nie potrzebujemy zapisywać w pamięci wszystkich wartości postaci $f[x_i,\ldots,x_k]$, gdzie $0 < i \leqslant k$.

3 Zadanie 2

3.1 Opis problemu

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)
```

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

Wyniki:
`nt` – wartość wielomianu w punkcie `t`.

3.2 Rozwiązanie

Mając wyliczone ilorazy różnicowe $c_0,c_1,\ldots,c_n$ możemy w prosty sposób wyznaczyć wartość wielomianu $p(x)$ dla zadanego x w czasie $\mathbf{O(n)}$. Skorzystamy tutaj ze *wzoru interpolacyjnego postaci Newtona*, jak zapowiedziałem na wstępie tego sprawozdania.

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

Na początku zamieńmy kolejność sumowania w ten sposób, że zaczniemy od $k = n$, a skończymy na $k = 0$ (możemy to zrobić, ponieważ dodawanie jest przemienne). Rozpiszmy ten wzór co pozwoli nam lepiej dostrzec pewne własności

$$\sum_{k=n}^{0} c_k \prod_{j=0}^{k-1}(x - x_j) =$$

$$= c_n(x-x_{n-1})(x-x_{n-2})\ldots(x-x_0) + c_{n-1}(x-x_{n-2})(x-x_{n-3})\ldots(x-x_0)+$$

$$\vdots$$

$$+c_4(x-x_3)(x-x_2)(x-x_1)(x-x_0) + c_3(x-x_2)(x-x_1)(x-x_0)+$$

$$+c_2(x-x_1)(x-x_0)+c_1(x-x_0)+c_0$$

Możemy zauważyć, że $q_{k+1} = q_k(x - x_k)$. Dokładając do tego ilorazy różnicowe c_k otrzymamy wzór rekurencyjny na $p(x)$ (uogólniony algorytm Hornera):

$$\begin{aligned} w_n(x) &= c_n \\ w_k(x) &= c_k + (x-x_k)w_{k+1}(x) \ (k = n-1, \ldots, 1, 0) \\ p_n(x) &= w_0(x) \end{aligned}$$

Algorithm 2: `warNewton (x, fx, t)`

Data: $\tilde{x}$ - wektor węzłów $x_0, \ldots, x_n$; $\tilde{fx}$ - wektor ilorazów różnicowych $c(x_0), \ldots, c(x_n)$; t - punkt dla którego liczymy $p(t)$
Result: nt - wartosc wielomianu w punkcie t
$n \leftarrow length(x)$;
$nt \leftarrow fx[n-1]$;
for $i \leftarrow n-2$ **to** 0 **do**
| $nt \leftarrow fx[i] + (t - x[i]) * nt$
end
return nt;

Przykład dla $n = 4$:

$$nt = fx[3] \tag{1}$$
$$nt = fx[2] + fx[3](t-x_2) \tag{2}$$
$$nt = fx[1] + fx[2](t-x_1) + fx[3](t-x_2)(t-x_1) \tag{3}$$
$$nt = fx[0] + fx[1](t-x_0) + fx[2](t-x_1)(t-x_0) + fx[3](t-x_2)(t-x_1)(t-x_0) \tag{4}$$

$$nt = \sum_{i=0}^{3} fx[i] \prod_{j=0}^{i-1}(t-x_j) \equiv p(x) = \sum_{k=0}^{n} c_k q_k(x)$$

4 Zadanie 3

4.1 Opis problemu

Znając współczynniki wielomianu interpolacyjnego w postaci Newtona

$$c_0 = f[x_0], c_1 = f[x_0, x_1], c_2 = f[x_0, x_1, x_2], \ldots, c_n = f[x_0, \ldots, x_n] \text{ (ilorazy różnicowe)}$$

oraz węzły $x_0, x_2, \ldots, x_n$ napisać funkcję obliczającą, w czasie $O(n^2)$, współczynniki jego postaci naturalnej $a_0, \ldots, a_n$ tzn. $a_n x_n + a_{n-1} x_{n-1} + \cdots + a_1 x + a_0$.
`function naturalna(x::VectorFloat64, fx::VectorFloat64)`

Dane:
`x` – wektor długości $n+1$ zawierający węzły $x_0, \ldots, x_n$
`x[1]` $= x_0$, ..., `x[n+1]` $= x_n$
`fx` – wektor długości $n+1$ zawierający ilorazy różnicowe
`fx[1]`$= f[x_0]$,
`fx[2]` $= f[x_0, x_1]$,
$\vdots$
`fx[n]` $= f[x_0, \ldots, x_{n-1}]$,
`fx[n+1]` $= f[x_0, \ldots, x_n]$

Wyniki:
`a` – wektor długości n + 1 zawierający obliczone współczynniki postaci naturalnej
`a[1]` $= a_0$,

`a[2]` $= a_1$,
$\vdots$
`a[n]` $= a_{n-1}$,
`a[n+1]` $= a_n$

4.2 Rozwiązanie

Tym razem naszym zadaniem jest wyliczenie współczynników wielomianu $p(x)$ zapisanego w postaci naturalnej:

$$p(x) = \sum_{i=0}^{n} a_i x^i$$

Współczynnik a_n możemy wyznaczyć od razu $a_n = c_n$. Z kolejnymi współczynnikami niestety będzie więcej zachodu. Skorzytsamy ze wskazówki zawartej w poleceniu i rozpiszemy uogólniony algorytm Hornera

$$\begin{aligned}
w_n &= c_n \\
w_{n-1} &= c_{n-1} + (x - x_{n-1})w_n = c_{n-1} + xc_n - c_n x_{n-1} \\
w_{n-2} &= c_{n-2} + (x - x_{n-2})w_{n-1} = \\
&= c_{n-2} + \mathbf{x}c_{n-1} + \mathbf{x^2}c_n - \mathbf{x}x_{n-1}c_n - x_{n-2}c_{n-1} - \mathbf{x}x_{n-2}c_n + x_{n-1}x_{n-2}c_n \\
&= \mathbf{x^2}c_n + \mathbf{x}(c_{n-1} - c_n(x_{n-1} + x_{n-2})) + c_{n-2} - x_{n-2}(c_{n-1} - c_n x_{n-1})
\end{aligned}$$

Przyjrzyjmy się jeszcze raz wielomianowi w_{n-2}, a konkretnie jego części $(x - x_{n-2})w_{n-1}$.

$$\begin{aligned}
x \cdot w_{n-1} &= \mathbf{x^2} \underbrace{c_n}_{a_n} + \mathbf{x} \underbrace{(c_{n-1} - x_{n-1}c_n)}_{\text{stare } a_{n-1}} \\
-x_{n-2} \cdot w_{n-1} &= \mathbf{x} \cdot \underbrace{-x_{n-2}c_n}_{\text{nowa część do } a_{n-1}} - \underbrace{x_{n-2}(c_{n-1} - x_{n-1}c_n)}_{\text{wyjściowa część } a_{n-2} \text{ (bez } +c_{n-2})}
\end{aligned}$$

Obliczmy jesczcze składowe $x \cdot w_{n-2}$ oraz $x_{n-3} \cdot w_{n-2}$ $w_{n-3} = c_{n-3} + (x - x_{n-3})w_{n-2}$.

$$\begin{aligned}
x \cdot w_{n-2} &= \mathbf{x^3} \underbrace{c_n}_{a_n} + \mathbf{x^2} \underbrace{(c_{n-1} - c_n(x_{n-1} + x_{n-2}))}_{\text{połączone części } a_{n-1} \text{ z } w_{n-1}} + \mathbf{x} \underbrace{(c_{n-2} - x_{n-2}(c_{n-1} - c_n x_{n-1}))}_{\text{stare } a_{n-2}} \\
-x_{n-3} \cdot w_{n-2} &= \mathbf{x^2} \underbrace{c_n(-x_{n-3})}_{\text{nowa część do } a_{n-1}} + \mathbf{x} \underbrace{(-x_{n-3})(c_{n-1} - c_n(x_{n-1} + x_{n-2}))}_{\text{nowa część do } a_{n-2}} + \underbrace{(-x_{n-3})(c_{n-2} - x_{n-2}(c_{n-1} - c_n x_{n-1}))}_{\text{wyjściowa część } a_{n-3} \text{ (bez } +c_{n-3})}
\end{aligned}$$

Możemy zauważyć, że w każdej iteracji najpierw wyzaczamy "bazowe" częsci współczynników stojących przy potęgach $x^i (1 \leqslant i \leqslant k+1)$, poprzez rozwiązanie równania $x \cdot w_{n-k}$. Mają one postać:

$$a_i = c_i - x_i + a_{i+1}$$

Następnie będac w i-tej iteracji dla każdego j takiego, że $i < j \leqslant n$ dodajemy do a_j składnik postaci $-x_{n-i}a_{j+1}$, aktualizując go. Wzór na a_j wyglądałby następująco:

$$a_j = a_j - x_i a_{i+1}$$

.

Algorithm 3: `naturalna (x,c)`

Data: $\tilde{x}$ - wektor węzłów $x_0, \ldots, x_n$; $\tilde{c}$ - wektor ilorazów różnicowych $c_0, \ldots, c_n$
Result: $\tilde{a}$ - wektor współczynników wielomianu
$n \leftarrow length(x)$;
$a_n \leftarrow c_n$;
for $i \leftarrow n-1$ **to** 0 **do**
  $a_i \leftarrow c_i - x_i a_{i+1}$;
  for $j \leftarrow i+1$ **to** $n-1$ **do**
    $a_j \leftarrow a_j - x_i a_{j+1}$;
  end
end
return $\tilde{a}$;

5 Zadanie 4

5.1 Opis problemu

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

- W interpolacji użyć węzłów równoodległych, tj.
  $x_k = a + kh, h = (b-a)/n, k = 0, 1, \ldots, n.$
- **Nie wyznaczać** wielomianu interpolacyjnego w jawnej postaci, tylko skorzystać z funkcji `ilorazyRoznicowe` i `warNewton`.

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

Dane:
`f` - funkcja $f(x)$ zadana jako anonimowa funkcja,
`a,b` - przedział interpolacji,
`n` - stopień wielomianu interpolacyjnego.

Wyniki:
- funkcja rysuje wielomian interpolacyjny i interpolowaną funkcję w przedziale $[a, b]$.

5.2 Rozwiązanie

W tym zadaniu chcemy połączyć wcześniej napisane algorytmy `ilorazyRoznicowe` i `warNewton` i napisać funkcję, która narysuje wykres interpolowanej funkcji $f(x)$ oraz wielomian interpolacyjny.
Na wejściu przyjmujemy funkcję `f`, przedział `[a,b]` oraz stopień wielomianu interpolacyjnego `n`. Na samym początku chcemy wyznaczyć $n+1$ węzłów na przedziale $[a, b]$. Zgodnie z zaleceniem w poleceniu węzły x_k będziemy konstruować w następujący sposób:

$$x_k = a + kh,$$

gdzie h - odległość między sąsiednymi punktami ($h = \frac{b-a}{n}$), $k = 0, 1, \ldots, n$. Potrzebujemy także utworzyć wektor $f_k = f(x_k)$, aby móc skorzystać z funkcji `c = ilorazyRoznicowe (x,f)`.
W zadaniu nie mieliśmy sprecyzowane z ilu punktów mamy uwtorzyć nasze wykresy. Postanowiłem, że liczbę punktów m uzależnimy od długości przedziału $[a, b]$:

$$m = \lceil b - a \rceil \cdot 200$$

W ten sposób dostaniemy m punktów $a_i (0 \leqslant i < m)$, które podobnie jak wcześniej wyznaczymy z równania

$$a_i = a + \frac{i}{200} \qquad (0 \leqslant i < m)$$

Dla nich wyznaczymy wartości $p_i =$ `warNewton (x, c,` a_i`)`, oraz $f(a_i)$ na podstawie których narysujemy wykresy.

6 Zadanie 5

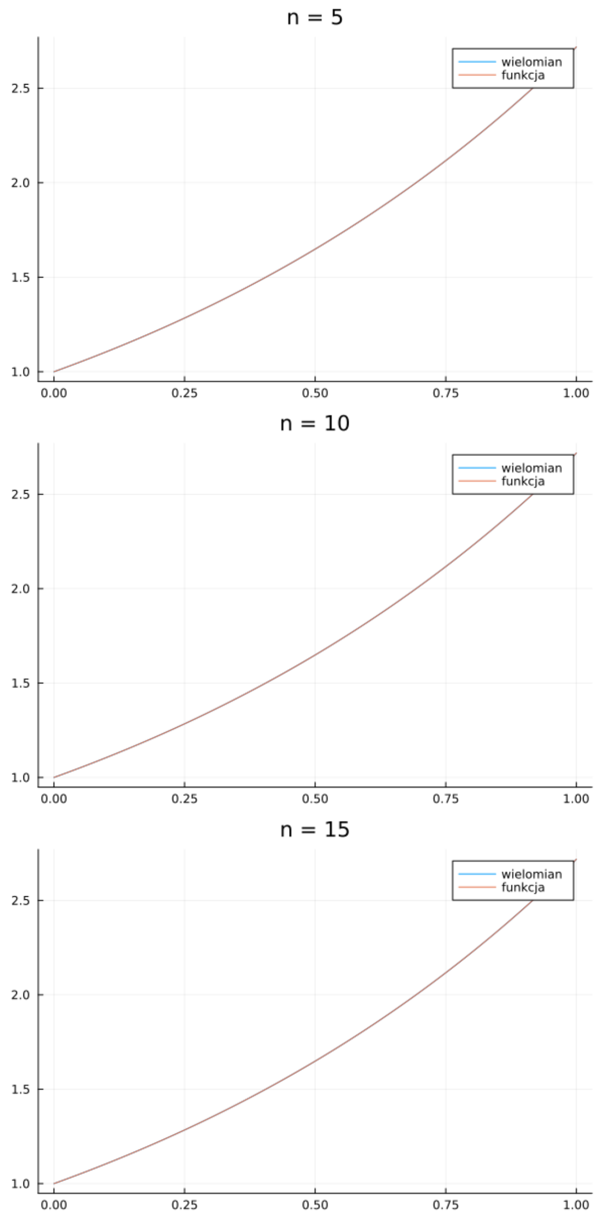
6.1 Opis problemu

Przetestować funkcję `rysujNnfx(f,a,b,n)` na następujących przykładach:

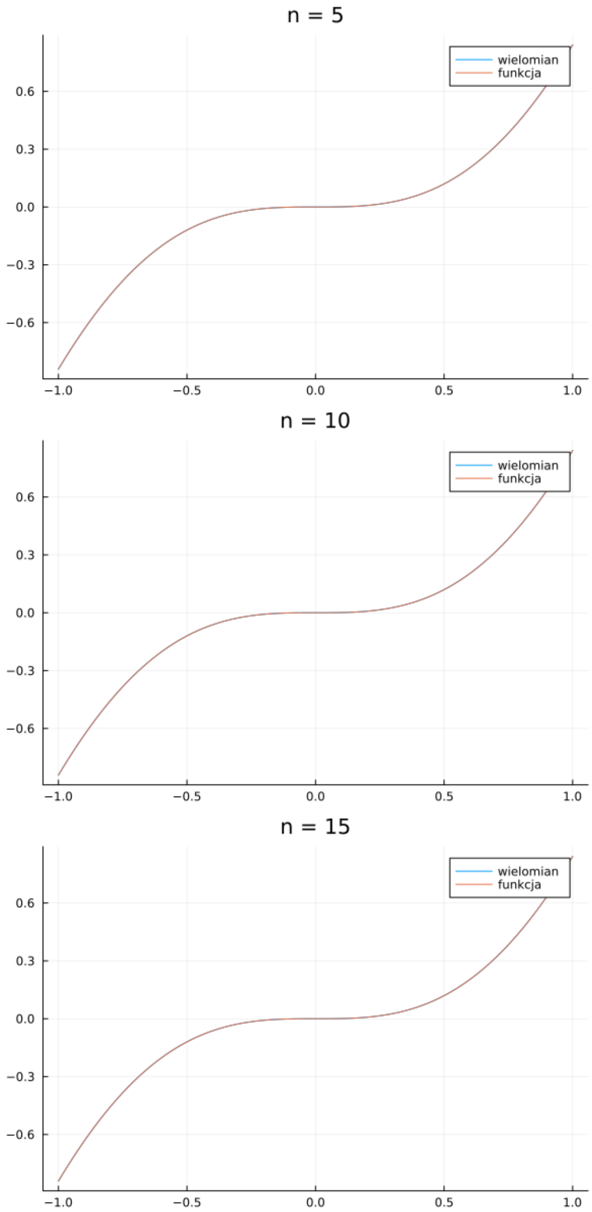
1. $e^x, [0, 1], n = 5, 10, 15,$
2. $x^2 sin(x), [-1, 1], n = 5, 10, 15$

6.2 Wyniki i interpretacja

Widzimy, że obie testowane funkcje dają się bardzo dokładnie interpolować. Dla każdej wartości $n = 5, 10, 15$ wielomiany interpolacyjne i funckje interpolowane nakładają się na siebie, zatem metoda sprawdza się idealnie dla tych funkcji, a algorytmy działają poprawnie.



Rysunek 1: $f(x) = e^x, [0, 1]$



Rysunek 2: $f(x) = x^2 sin(x), [-1, 1]$

7 Zadanie 6

7.1 Opis problemu

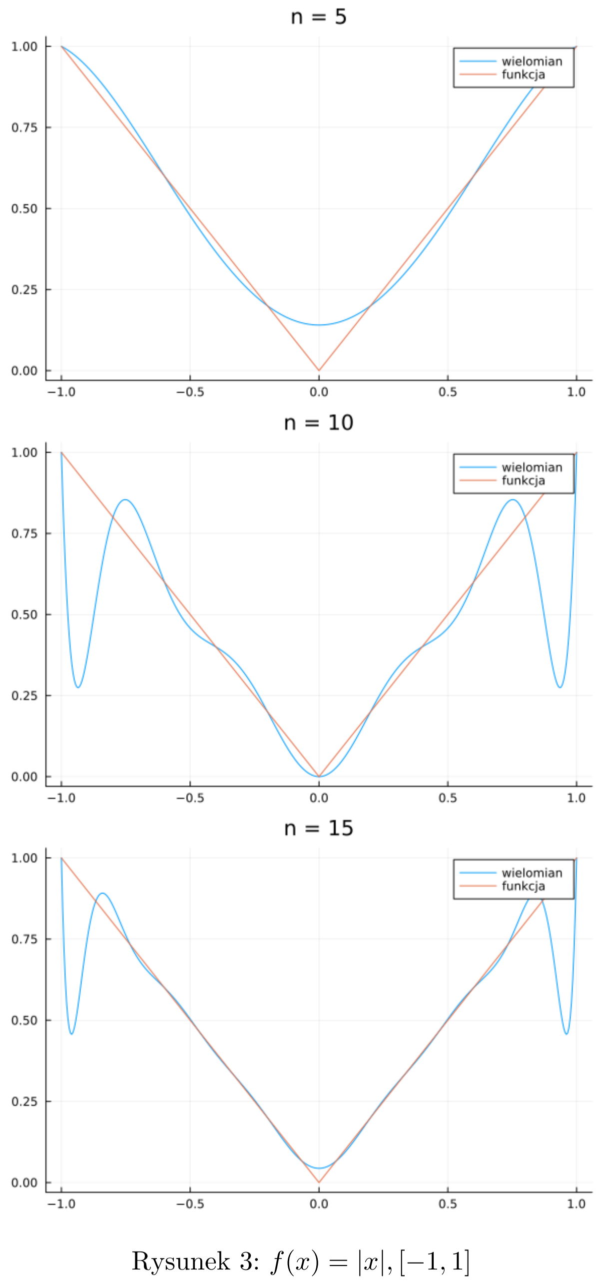
Przetestować funkcję `rysujNnfx(f,a,b,n)` na następujących przykładach (zjawisko rozbieżności):

1. $|x|, [-1, 1], n = 5, 10, 15$,
2. $\frac{1}{1+x^2}, [-5, 5], n = 5, 10, 15$ (zjawisko Runge'go).

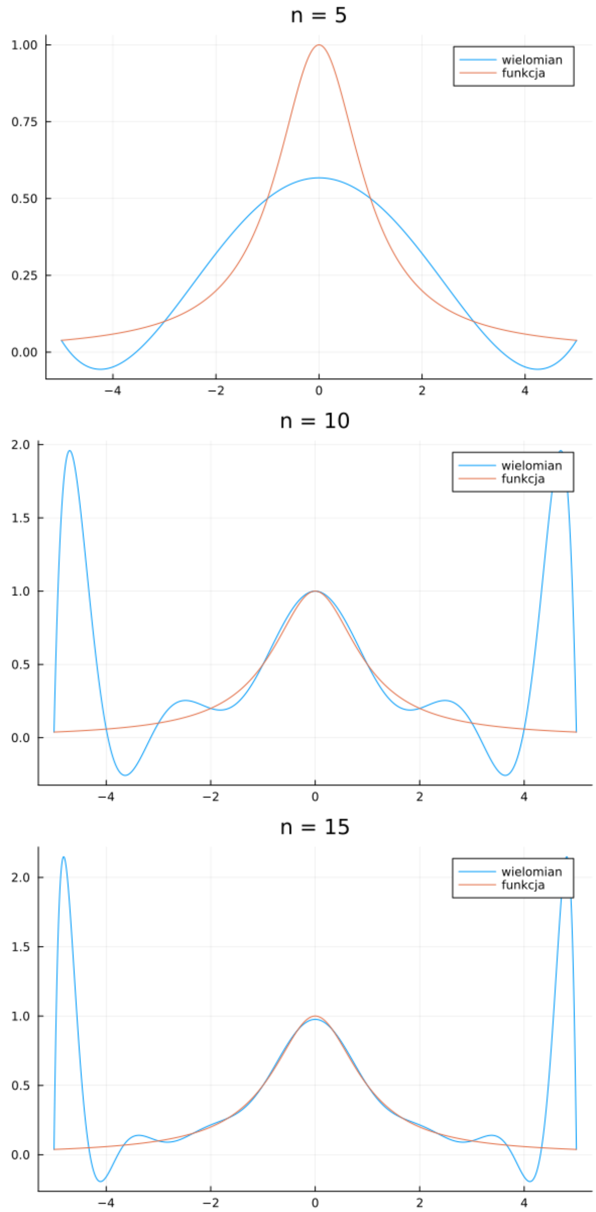
7.2 Wyniki i interpretacja

Widzimy, że w tym przypadku otrzymane wykresy dla obu funkcji nie nakładają się na siebie tak precyzyjnie jak w poprzednim zadaniu. Zwiększenie liczby węzłów pwowduje zwiększenie precyzji w okolicach środka przedziału, jednak dla x-ów znajdujących się blisko krawędzi przedziału, dokładność się nie polepsza, a wykres wielomianu interpolacyjnego zaczyna wariować,.
W przypadku funkcji $|x|$, na przedziale $[-1, 1]$ przykład, gdzie $n = 5$ różni się od pozostałych. Wartości wielomianu na krańcach przedziału są najbardziej zbliżone do wartości funkcji interpolowanej, jednak

problemem staje się środek przedziału $x = 0$, gdzie powstała dosyć duża rozbieżność między uzyskanymi wynikami. Stało się tak, ponieważ węzły dla $x = 5$ są następujące $x_0 = -1.0, x_1 = -0.6, x_2 = -0.2, x_3 = -0.2, x_4 = -0.6, x_5 = 1.0$. Żaden z węzłów nie wypadł w punkcie $x = 0$, stąd taka niedokładność. Wydawać by się mogło że przy zwiększeniu liczby węzłów dostaniemy większą dokładność, jednak wtedy powstają błędy przy granicach przedziału (**efekt Rungego**). Możemy z tego wywnioskować, że funkcja $f(x) = |x|$ nie nadaje się do użycia interpolacji wielomianowej.
Podobnie sytuacja wygląda dla funkcji $f(x) = \frac{1}{1+x^2}$ na przedziale $[-5, 5]$. W tym przypadku dla $n = 5$ nasz wielomian interpolacyjny zwraca wyniki bliskie oryginałom tylko dla węzłów. Niestety, dołożenie większej liczby węzłów na niewiele się zdaje, ponieważ mimo lepszej aproksymacji w środku przedziału, funkcja znowu zaczyna znacząco wariować na krańcach, gdzie wyniki stają się zupełnie rozbieżne (**efekt Rungego**).



Rysunek 3: $f(x) = |x|, [-1, 1]$



Rysunek 4: $f(x) = \frac{1}{1+x^2}, [-5, 5]$

8 Wnioski

Interpolacja wielomianowa okazała się skuteczną metodą przybliżania funkcji, gdy znamy jej wartości tylko w niektórych punktach. Ma jednak kilka ograniczeń, które skutecznie zaprezentowały nam funkcje $f(x) = |x|$ oraz $g(x) = \frac{1}{1+x^2}$ z zadania 6. Interpolacja wielomianowa powinna dobrze sobie radzić z funkcjami gładkimi, których wykres nie posiada "ostrych" fragmentów. Potwierdziły nam to funkcje $h(x) = e^x, t(x) = x^2 sin(x)$ z zadania 5, których wykresy idealnie nakładały się z wyliczonym wielomianem. Nie jest to jednak reguła, o czym przekonała nas z kolei funkcja $g(x) = \frac{1}{1+x^2}$, gdzie wykres $t(x)$ był gładki, a mimo to mogliśmy zaobserwować **Efekt Rungego**.
Efekt Rungego to zjawisko pogorszenia jakości interpolacji wielomianowej, mimo zwiększenia liczby węzłów. Początkowo ze wzrostem liczby węzłów n przybliżenie poprawia się, jednak po dalszym wzroście n, zaczyna się pogarszać, co jest szczególnie widoczne na końcach przedziałów.(patrz `zadanie 6.`)
Występuje on dla interpolacji za pomocą wielomianów wysokich stopni, gdy węzły są w równych odległościach od siebie albo funkcja znacząco odbiega od gładkiej $f(x) = |x|$.
Chcąc pozbyć się efektu Rungego dla funkcji $f(x) = |x|$, $g(x) = \frac{1}{1+x^2}$ w zadaniu 6., moglibyśmy inaczej wyznaczać rozkład węzłów do interpolacji wielomianowej. Powinny być one gęściej rozłożone na krańcach przedziału.