Obliczenia Naukowe,
Laboratorium, Lista 4

Jerzy Wroczyński

2020-12-06

1 Zadanie 1.

Napisać funkcję obliczającą ilorazy różnicowe dla podanych węzłów oraz wartości danej funkcji w tych węzłach.

1.1 Dane wejściowe

- `x` — wektor długości $n + 1$ zawierający węzły $x_0, \ldots, x_n$
- `f` — wektor długości $n + 1$ zawierający wartości interpolowanej funkcji w węzłach $f(x_0), \ldots, f(x_n)$

1.2 Dane wyjściowe

- `fx` — wektor długości $n + 1$ zawierający obliczone ilorazy różnicowe

1.3 Rozwiązanie

Kod źródłowy znajduje się w pliku `code/interpolation.jl` jako funkcja `ilorazyRoznicowe`.

1.4 Opis użytego algorytmu

Do obliczania ilorazów różnicowych dla podanych węzłów użyłem zamiast tablicy dwuwymiarowej jednej tablicy długości $n + 1$. Użycie tutaj tablicy dwuwymiarowej jest tutaj absolutnie zbędne, bo jest nieoptymalne pod względem zajmowanej pamięci — nie potrzebujemy pamiętać każdego ilorazu różnicowego.
Musimy obliczyć $f[x_0], f[x_0, x_1], f[x_0, x_1, x_2], \ldots f[x_0, \ldots, x_n]$.

Idea algorytmu jest taka: w danym momencie „pamiętamy” tylko jedną kolumnę macierzy trójkątnej reprezentującej wszystkie ilorazy różnicowe:

$$\begin{bmatrix}
f[x_0] & f[x_0, x_1] & \ldots & f[x_0, \ldots, x_{m-1}] & f[x_0, \ldots, x_m] & \ldots & f[x_0, \ldots, x_{n-1}] & f[x_0, \ldots, x_n] \\
f[x_1] & f[x_1, x_2] & \ldots & f[x_1, \ldots, x_m] & f[x_1, \ldots, x_{m+1}] & \ldots & f[x_1, \ldots, x_n] & \\
\vdots & \vdots & & \vdots & \vdots & \iddots & & \\
f[x_k] & f[x_k, x_{k+1}] & \ldots & f[x_k, \ldots, x_{k+m-1}] & f[x_k, \ldots, x_{k+m}] & & & \\
f[x_{k+1}] & f[x_{k+1}, x_{k+2}] & \ldots & f[x_{k+1}, \ldots, x_{k+m}] & & & & \\
\vdots & & \iddots & & & & & \\
f[x_{n-1}] & f[x_{n-1}, x_n] & & & & & & \\
f[x_n] & & & & & & &
\end{bmatrix}$$

Za każdym razem musimy wygenerować całą kolumnę, bo potem będziemy z tych wartości korzystać (ostatni iloraz $f[x_0, \ldots, x_n]$ potrzebuje ich wszystkich), ale w danym momencie zapisujemy do wektora wyjściowego tylko wartości

$f[x_0,\dots,x_m]$ dla $m \in [0,n] \cap \mathbb{N}$. Przy czym w programie `ilorazyRoznicowe` $m =$ `len - p` gdzie `p` to iterator, a `len` $= n+1$.
Poniżej został przedstawiony algorytm — używamy tutaj oznaczeń takich samych jak w specyfikacji.

```
1: len ← |x|
2: tmp ← f
3: output ← [tmp[1]]
4: for p ← (len − 1) ... 1 do
5:     m = len − p
6:     for k ← 1 ... p do
7:         tmp[k] ← (tmp[k+1] − tmp[k]) / (x[k+m] − x[k])
8:     end for
9:     output[m] ← tmp[1]
10: end for
11: return output
```

W linijce 1 ustalamy długość wektora, czyli dowiadujemy się ile wynosi $n+1$. W 2 kopiujemy wektor wartości w podanych węzłach do tablicy roboczej (określającej stan w jednej kolumnie wcześniej wspomnianej macierzy). W 3 definiujemy wektor wyjściowy, który ma na początku jeden element $f[x_0]$ — załatwia nam to pierwszą iterację.

Zaczynamy iterować (4) po liczbie elementów w kolumnie macierzy. Pierwszą iterację dla `len` $= n+1$ mamy już wykonaną, więc zaczynamy od `len` $- 1 = n$. Generujemy poszczególne ilorazy różnicowe w danej kolumnie (7) oraz zachowujemy tylko iloraz różnicowy postaci $f[x_0,\dots,x_m]$.

Na końcu zwracamy wektor wyjściowy zawierający wszystkie ilorazy, które nas interesują (ilorazy postaci $f[x_0,\dots,x_m]$ dla $m \in [0;n]$).

1.5 Prosty test

Funkcja ta zostanie jeszcze przetestowana w dalszych zadaniach, jednakże dla potwierdzenia działania w pliku `code/ex-1.jl` znajduje się prosty test implementujący przykład z wykładu:

x	3	1	5	6
f(x)	1	-3	2	4

Po uruchomieniu otrzymujemy poprawne wyniki z jednym zastrzeżeniem — ostatni iloraz ma lekki błąd $2 \cdot 10^{-17}$. Jest to błąd, którego się nie uniknie, bo wynika z błędu pojawiającego się przy wykonywaniu jakichkolwiek działań.

$f[x_0]$	$f[x_0,x_1]$	$f[x_0,x_1,x_2]$	$f[x_0,x_1,x_2,x_3]$
1	2	$-0.375 = \frac{-3}{8}$	$0.17500\dots002 \approx \frac{7}{40}$

Dla porównania rachunki ręczne na tych samych danych:

x_k	$f[x_k]$	$f[x_k,x_{k+1}]$	$f[x_k,x_{k+1},x_{k+2}]$	$f[x_0,x_1,x_2,x_3]$
3	**1**	**2**	$\mathbf{\frac{-3}{8}}$	$\mathbf{\frac{7}{40}}$
1	-3	$\frac{5}{4}$	$\frac{3}{20}$	
5	2	2		
6	4			

2 Zadanie 2.

Napisać funkcję obliczającą wartość wielomianu interpolacyjnego stopnia n w postaci Newtona $N_n(x)$ w punkcie $x = t$ za pomocą algorytmu uogólnionego Hornera w czasie $O(n)$.

2.1 Dane wejściowe

- `x` — wektor długości $n+1$ zawierający węzły $x_0, \dots, x_n$
- `fx` — wektor długości $n+1$ zawierający ilorazy różnicowe $f[x_0], \dots, f[x_0, \dots, x_n]$
- `t` — punkt, w którym należy obliczyć wartość wielomianu

2.2 Dane wyjściowe

- `nt` — wartość wielomianu w punkcie `t`

2.3 Rozwiązanie

Kod źródłowy znajduje się w pliku `code/interpolation.jl` jako funkcja `warNewton`.

2.4 Opis algorytmu

Wzór interpolacyjny Newtona ma postać

$$N_n(x) = \sum_{k=0}^{n} f[x_0, \dots, x_k] \cdot \prod_{j=0}^{k-1} (x - x_j),$$

przy czym $\prod_{j=0}^{k-1}(x - x_j)$ dla $k = 0$ wynosi 1.

Oczywiście, możemy zapisać ten wielomian w równoważnej postaci:

$$N_n(x) = f[x_0] + f[x_0, x_1] \cdot (x - x_0) + \dots + f[x_0, x_1, \dots, x_n] \cdot (x - x_0)(x - x_1) \cdots (x - x_{n-1})$$

wówczas

$$N_n(x) = f[x_0] + (x - x_0)\big(f[x_0, x_1] + f[x_0, x_1, x_2](x - x_1) + \dots + f[x_0, x_1, \dots, x_n](x - x_1) \cdots (x - x_{n-1})\big).$$

Wyciągnęliśmy przed nawias $(x - x_0)$ — teraz możemy tak zrobić jeszcze $(n-2)$ razy:

$$N_n(x) = f[x_0] + (x - x_0)\Big(f[x_0, x_1] + (x - x_1)\big(\dots (x - x_{n-2})(f[x_0, x_1, \dots, x_n](x - x_{n-1}) + f[x_0, \dots, x_{n-1}])\big) \dots\Big)$$

Łatwo zauważyć, że mamy do czynienia z rekurencją postaci

$$\begin{cases} w_n(x) = f[x_0, x_1, \dots, x_n] \\ w_k(x) = f[x_0, x_1, \dots, x_k] + (x - x_k) \cdot w_{k+1}(x) \qquad (k \in [n-1; 0] \cap \mathbb{N}) \end{cases}$$

i na przykład właśnie mamy w środku $f[x_0, \dots, x_n](x - x_{n-1}) + f[x_0, \dots, x_{n-1}]$ co jest równe $w_{n-1}(x)$. Za to po zastosowaniu tej rekurencji, po „zwinięciu" wzoru N_n otrzymamy $f[x_0] + (x - x_0)w_1(x)$ co jest równe $w_0(x)$.

Zatem $N_n(x) = w_0(x)$.

W takim razie, żeby obliczyć wartość $N_n(\texttt{t})$, należy obliczyć wartość $w_0(\texttt{t})$ co wiąże się z policzeniem wartości wszystkich funkcji $w_k(\texttt{t})$ dla $k = 0, \dots, n$.

2.5 Złożoność obliczeniowa

Mamy do obliczenia n równań postaci $w_k(x) = f[x_0, x_1, \dots, x_k] + (x - x_k) \cdot w_{k+1}(x)$, bo w_n jest już wiadome, więc zaczynamy od w_{n-1} idąc w dół do w_0. Za każdym razem mamy do policzenia to samo równanie, które zajmuje $O(1)$ czasu, więc złożoność obliczeniowa całego algorytmu wynosi $O(n)$.

3 Zadanie 3.

Napisać funkcję obliczającą współczynniki postaci naturalnej wielomianu interpolacyjnego stopnia n w postaci Newtona $N_n(x)$.

3.1 Dane wejściowe

- `x` — wektor długości $n+1$ zawierający węzły $x_0, \dots, x_n$
- `fx` — wektor długości $n+1$ zawierający ilorazy różnicowe $f[x_0], \dots, f[x_0, \dots, x_n]$

3.2 Dane wyjściowe

- `a` — wektor długości $n+1$ zawierający obliczone współczynniki postaci naturalnej $(a_n x^n + a_{n-1}x^{n-1} + \cdots a_1 x + a_0)$

3.3 Rozwiązanie

Kod źródłowy znajduje się w pliku `code/interpolation.jl` jako funkcja `naturalna`.

3.4 Opis algorytmu

Z poprzedniego zadania (2.4) wiemy, że wielomian interpolacyjny Newtona $N_n(x)$ możemy zapisać przy pomocy rekurencji:

$$\begin{cases} w_n(x) = f[x_0, x_1, \dots, x_n] \\ w_k(x) = f[x_0, x_1, \dots, x_k] + (x - x_k) \cdot w_{k+1}(x) \qquad (k \in [n-1; 0] \cap \mathbb{N}) \end{cases}$$

gdzie $N_n(x) = w_0(x)$. Przyjrzyjmy się wzorowi, który „tworzy” rekurencję:

$$w_k(x) = f[x_0, x_1, \dots, x_k] + (x - x_k) \cdot w_{k+1}(x) \qquad (k \in [n-1; 0] \cap \mathbb{N})$$

Łatwo zauważyć, że wielomian w_k będzie miał stopień o jeden wyższy niż w_{k+1}. Zapiszmy powyższy wzór tak, aby bardziej przypominał postać naturalną:

$$w_k(x) = x \cdot w_{k+1}(x) - w_{k+1} \cdot x_k + f[x_0, \dots, x_k],$$

a następnie wprowadźmy ciąg $b_m^{(k)}$ określający współczynnik przy x^m dla wielomianu w_k. Wówczas mamy:

$$w_k(x) = x \cdot \left(b_m^{(k+1)} x^m + b_{m-1}^{(k+1)} x^{m-1} + \cdots + b_1^{(k+1)} x\right) + \left(b_m^{(k+1)} x^m + b_{m-1}^{(k+1)} x^{m-1} + \cdots + b_1^{(k+1)} x\right) \cdot x_k + f[x_0, \dots, x_k]$$

gdzie m określa stopień wielomianu w_{k+1}. Jako, że w_n ma stopień zero ($w_n(x) = f[x_0, \dots, x_n]$) a każdy następny wielomian w_k ma stopień o jeden wyższy niż poprzedni w_{k+1} można stwierdzić, że $m = (n-1) - k$.

Teraz możemy przekształcić wyżej wyliczony wzór do postaci zbliżonej do żądanej:

$$\begin{aligned} w_k(x) = b_m^{(k+1)} x^{m+1} &+ \left(b_{m-1}^{(k+1)} - b_m^{(k+1)} x_k\right) x^m + \left(b_{m-2}^{(k+1)} - b_{m-1}^{(k+1)} x_k\right) x^{m-1} + \\ &+ \cdots + \left(b_0^{(k+1)} - b_1^{(k+1)} x_k\right) x - b_0^{(k+1)} x_k + f[x_0, \dots, x_k] \end{aligned} \tag{1}$$

gdzie x_k to są węzły interpolacyjne.

Stosując nowy wzór na rekurencję (1) nasz wielomian $w_0(x)$ będzie wielomianem zapisanym w postaci naturalnej:

$$w_0(x) = b_n^{(1)} x^n + \left(b_{n-1}^{(1)} - b_n^{(1)} x_0\right) x^{n-1} + \cdots + \left(b_0^{(1)} - b_1^{(1)} x_0\right) x - b_0^{(1)} x_0 + f[x_0],$$

a jako, że $N_n(x) = w_0(x)$ otrzymamy tym samym postać naturalną wielomianu interpolacyjnego Newtona.

Idea jest taka: zaczynając od góry (od w_n) liczymy kolejno współczynniki dla $w_{n-1}, w_{n-2}, \dots, w_1, w_0$ w sposób iteracyjny. Jako że w_n jest znane stosujemy wzór (1) n razy, żeby uzyskać w_0.

3.5 Złożoność obliczeniowa

Mamy n równań (nie wliczamy już wiadomego w_n) gdzie za każdym razem mamy $m+1$ obliczeń współczynników. Liczba m rośnie w zależności od k, bo $m = (n-1) - k$. Wówczas można całkowitą liczbę obliczanych współczynników (ile współczynników musieliśmy obliczyć) utożsamić z sumą ciągu arytmetycznego $c_n = n$.
Zatem złożoność obliczeniowa wynosi $O\left(\frac{n(n+1)}{2}\right) = O\left(\frac{n^2}{2}\right) = O(n^2)$.

3.6 Prosty test

Dla potwierdzenia działania tejże funkcji w pliku `code/ex-3.jl` znajduje się prosty test implementujący przykład z wykładu. Najpierw obliczamy ilorazy różnicowe na podstawie podanych węzłów i wartości interpolowanej funkcji w tych węzłach:

x	3	1	5	6
f(x)	1	-3	2	4

$f[x_0]$	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$
1	2	$-0.375 = \frac{-3}{8}$	$0.17500\ldots002 \approx \frac{7}{40}$

Teraz możemy obliczyć współczynniki postaci naturalnej wielomianu interpolacyjnego Newtona, używając programu `naturalna`:

a_3	a_2	a_1	a_0
$0.175 = \frac{7}{40}$	$-1.95 = \frac{-39}{20}$	$7.525 = \frac{301}{40}$	$-8.75 = \frac{-35}{4}$

Powyższe współczynniki zgadzają się z wynikiem podanym przez zaufane źródło (WolframAlpha), które podaje taki sam wynik: $\frac{7x^3}{40} - \frac{39x^2}{20} + \frac{301x}{40} - \frac{35}{4}$ (Link do wyrażenia w WolframAlpha, sekcja *«Expanded form»*).

4 Zadanie 4.

Napisać funkcję, która zinterpoluje zadaną funkcję f w przedziale $[a; b]$ za pomocą wielomianu interpolacyjnego stopnia n postaci Newtona. Następnie narysuje wielomian interpolacyjny i interpolowaną funkcję. Należy użyć węzłów równoodległych czyli $x_k = a + kh$ dla $h = \frac{b-a}{n}$, $k = 0, 1, \ldots, n$.

4.1 Dane wejściowe

- `f` — funkcja f zadana jako anonimowa funkcja
- `a,b` — przedział interpolacji
- `n` — stopień wielomianu interpolacyjnego

4.2 Dane wyjściowe

`Nothing` — funkcja niczego nie zwraca, za to generuje odpowiednie wykresy.

4.3 Rozwiązanie

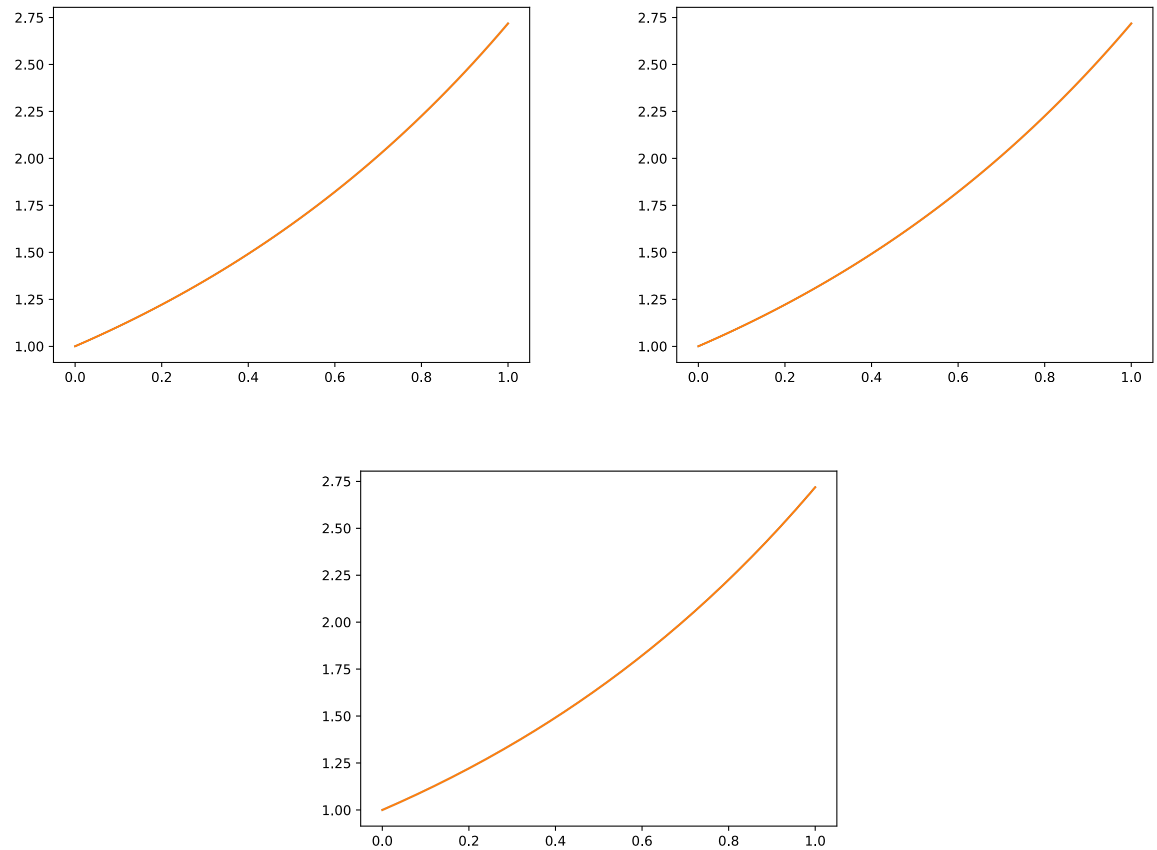
Kod źródłowy znajduje się w pliku `code/interpolation.jl` jako funkcja `rysujNnfx`.

5 Zadanie 5.

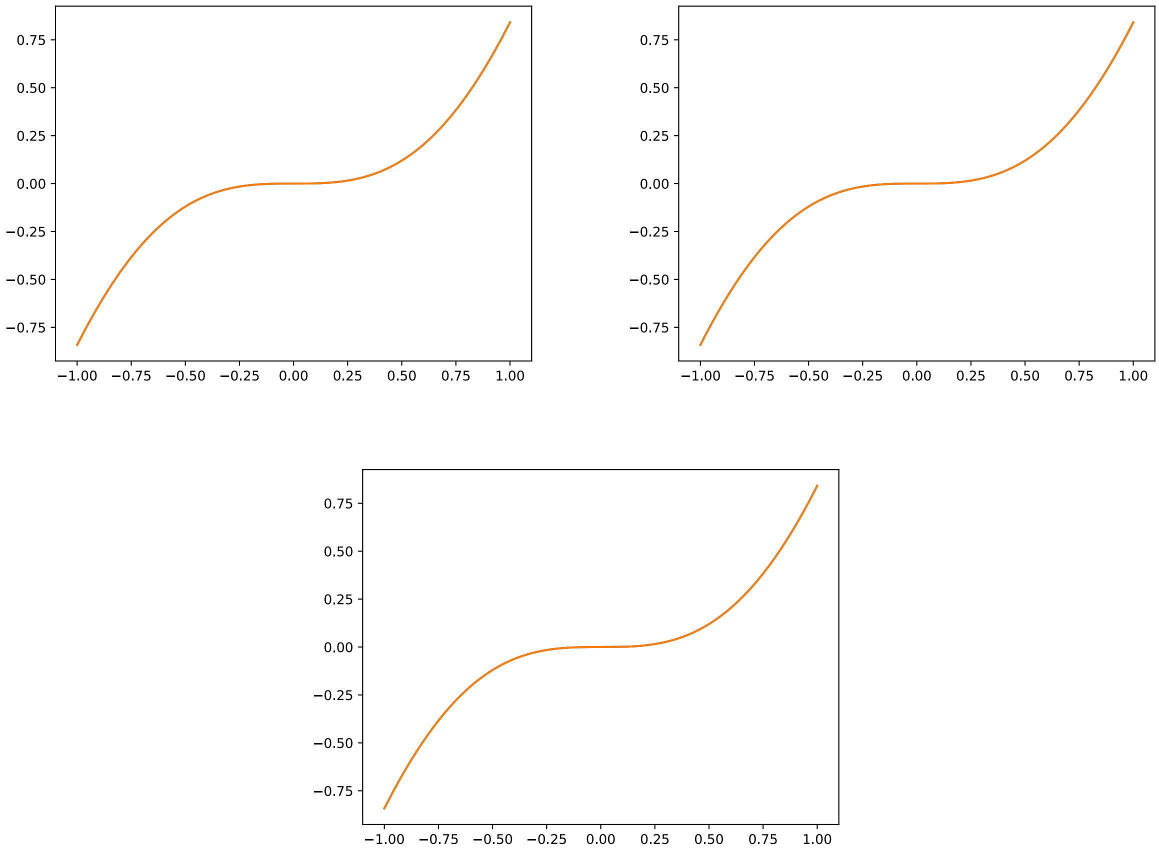
Przetestować metodę `rysujNnfx` na kilku zadanych poniżej funkcjach.

(Kod źródłowy w pliku `code/ex-5.jl`.)

5.1 Wyniki



Rysunek 1: $f(x) = e^x$ w przedziale $[0; 1]$ dla $n = 5, 10, 15$



Rysunek 2: $f(x) = x^2 \cdot sinx$ w przedziale $[-1; 1]$ dla $n = 5, 10, 15$

5.2 Obserwacje i wnioski

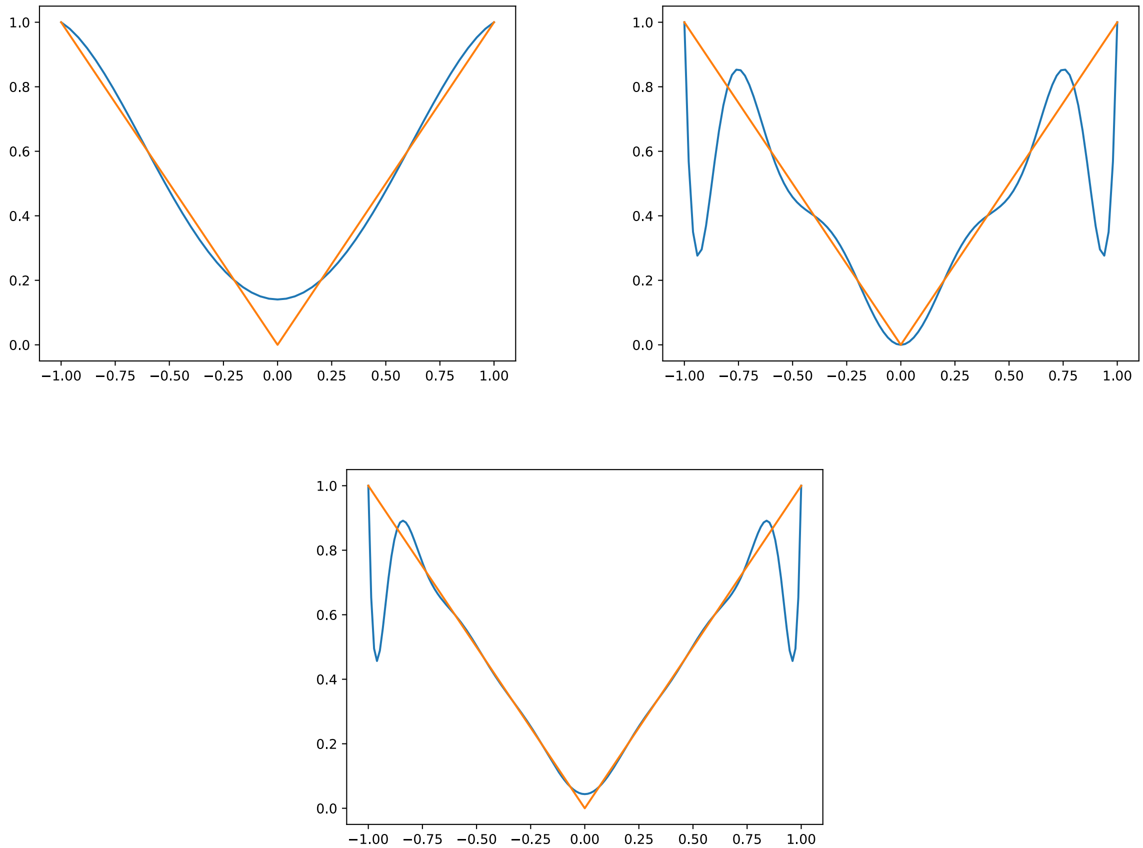
Wynikowe wykresy nie są zaskakujące — oryginalna funkcja i jej interpolacja nakładają się na siebie, czyli interpolacja praktycznie idealnie odwzorowuje funkcję wejściową. Dzieje się tak, ponieważ powyższe funkcje są funkcjami ciągłymi i gładkimi (wszystkie ich pochodne też są ciągłe) co oznacza, że takie „przybliżenie” funkcji przy pomocy wielomianu interpolacyjnego jest jak najbardziej efektywne.

6 Zadanie 6.

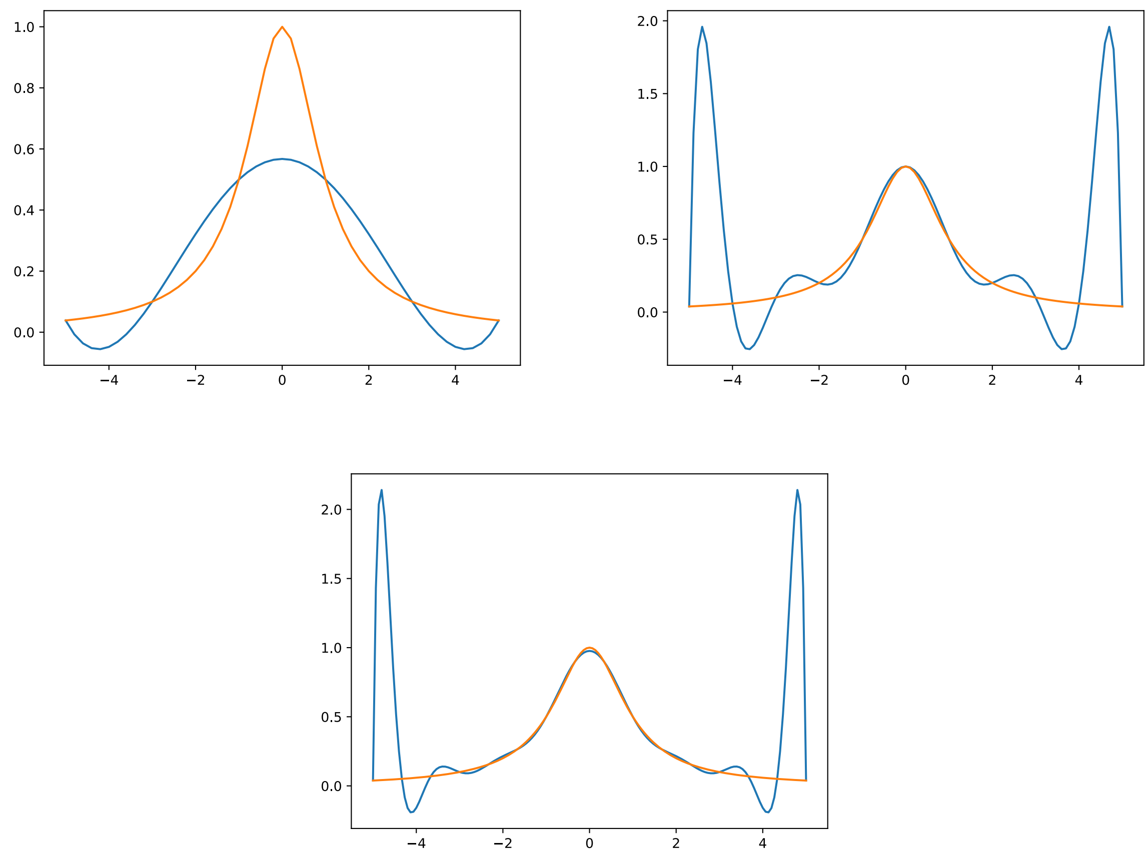
Przetestować metodę `rysujNnfx` na kilku zadanych poniżej funkcjach.

(Kod źródłowy w pliku `code/ex-6.jl`.)

6.1 Wyniki



Rysunek 3: $f(x) = |x|$ w przedziale $[-1; 1]$ dla $n = 5, 10, 15$



Rysunek 4: $f(x) = \frac{1}{1+x^2}$ w przedziale $[-5; 5]$ dla $n = 5, 10, 15$

6.2 Obserwacje i wnioski

Tym razem mamy bardzo duże odchylenia wielomianu interpolacyjnego od funkcji wejściowej. Mamy tutaj do czynienia ze zjawiskiem Runge'ego. Zjawisko to jest zwłaszcza spotykane właśnie w przypadkach funkcji nieciągłych (np. $f(x) = |x|$) lub po prostu w przypadkach interpolacji funkcji dla dużego stopnia wielomianu interpolacyjnego i dla węzłów równoodległych (np. $f(x) = \frac{1}{1+x^2}$).

Warto jednak zauważyć, że duże odchylenia mają miejsce głównie na brzegach zadanego przedziału. Wówczas w celu zwiększenia precyzji należałoby zwiększyć liczbę węzłów w tych miejscach, gdzie mamy takie problemy. Jednym z możliwych rozwiązań jest zastosowanie węzłów Czebyszewa (otrzymywanych z miejsc zerowych wielomianów Czebyszewa), które mają więcej węzłów przy końcach przedziału.