Sprawozdanie z listy nr 4 Obliczenia Naukowe

Marek Świergoń (261750)

11 grudnia 2022, PWr, WIiT INA

0 Wstęp: przedstawienie problemu interpolacji

Na początku istotne jest sformułowanie problemu, który będziemy próbowali rozwiązać z wykorzystaniem zaimplementowanych metod. Dla pewnych par $(x_i, y_i = f(x_i))$ (zwanych węzłami interpolacji) takich, że x_i są parami różne i $i = 0, ..., n$, będziemy chcieli wyznaczyć wielomian $p_n(x)$ stopnia co najwyżej n, który będzie interpolować funkcję f w tych parach. Oznacza to, że dla dowolnego $i \in \{0, 1, ..., n\}$, $p(x_i) = f(x_i) = y_i$. Z wykładu wiemy, że istnieje dokładnie jeden wielomian takiego stopnia.

Gdybyśmy chcieli bezpośrednio uzyskać wielomian interpolacyjny w postaci naturalnej, tj.

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n,$$

to musielibyśmy wyznaczyć współczynniki $a_0, a_1, ..., a_n$; zadanie to prowadzi do układu równań z macierzą Vandermonde'a. Macierz ta jest bardzo źle uwarunkowana, co przekreśla nam taki schemat działania z numerycznego punktu widzenia (bardzo duże błędy przy niewielkich zaburzeniach, patrz Sprawozdanie z listy nr 2). Przedstawiamy zatem wielomian p_n w innej bazie. Niech

$$q_0(x) = 1,$$

$$q_1(x) = (x - x_0),$$

$$q_2(x) = (x - x_0)(x - x_1),$$

$$\vdots$$

$$q_n(x) = (x - x_0)(x - x_1)...(x - x_{n-1}).$$

Otrzymane wielomiany q_i są bazą dla wielomianów stopnia n, w związku z tym istnieje c_i, takie że

$$p_n(x) = \sum_{i=0}^{n} c_i \cdot q_i(x).$$

Taką postać wielomianu p_n nazywamy postacią Newtona.

Korzystamy z faktu, że p spełnia warunki interpolacji i wyznaczamy $c_0, c_1, ..., c_n$; są to tzw. ilorazy różnicowe, a ich wyliczaniem zajmujemy się w zadaniu 1.

Mając wyliczone ilorazy różnicowe możemy wyliczyć wartość wielomianu interpolacyjnego w danym punkcie: efektywną realizacją tego zajmiemy się w zadaniu 2.

W zadaniu 3. przekształcamy wielomian w postaci Newtona do postaci naturalnej, dzięki czemu możemy łatwo zobaczyć stopień wielomianu.

Ostatecznie w zadaniu 4. przedstawiamy na wykresie uzyskany wielomian interpolacyjny oraz funkcję, którą ten wielomian interpoluje.

Implementacje algorytmów z zadań 1-4 zostały umieszczone w module `Interpolacja`, w pliku `interpolacja.jl`. Testy do metod z modułu znajdują się w pliku `tests.jl`.

1 Zadanie 1.

1.1 Idea algorytmu

Z pomocą tego algorytmu, mając pary (węzły) $(x_i, y_i = f(x_i))$, możemy wyliczyć ilorazy różnicowe $c_i = f[x_0, x_1, ..., x_i]$. Zauważmy, że dla x_0 zachodzi $f[x_0] = f(x_0)$ (z ostatniego wzoru we wprowadzeniu). Ponadto, na wykładzie udowodnione zostało twierdzenie o ilorazach różnicowych wyższych rzędów, wyglądające następująco:

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

Przy pomocy tej wiedzy, łatwo możemy stworzyć algorytm pracujący na tablicy dwuwymiarowej, który będzie korzystał z powyższej zależności rekurencyjnej i warunku brzegowego. Takie podejście można usprawnić, pracując na tablicy jednowymiarowej.

Istotnym spostrzeżeniem jest to, że po wyliczeniu wszystkich ilorazów różnicowych zależnych od k węzłów, wszystkie te ilorazy oprócz $f[x_0, x_1, ..., x_k]$ stają się dla nas niepotrzebne, można je zatem umiejętnie nadpisać ilorazami różnicowymi opartymi na większej liczbie węzłów.

W związku z tym na początku do tablicy rozmiaru $n+1$ zapisujemy wartości funkcji w węzłach interpolacji (ponieważ $f(x_i) = f[x_i]$ dla $i \in \{0, ..., n\}$). Następnie zaczynając od n i kończąc na 1 (chcemy zachować $f[x_0]$), wyliczamy $f[x_{i-1}, x_i]$ i wstawiamy w tablicy w miejsce $f[x_i]$. Dla ilorazów różnicowych powyższych rzędów postępujemy analogicznie, przy czym kończymy na $i = 2, 3, ..., n-1$ (zachowujemy kolejne ważne ilorazy róznicowe). Korzystamy tutaj z faktu, że liczba różnych od siebie ilorazów różnicowych opartych na $k+1$ węzłach jest o jeden mniejsza od liczby ilorazów różnicowych opartych na k węzłach.

Zawartość zwracanej pod koniec tablicy w danych iteracjach przedstawiona jest poniżej. Każda kolumna to tablica w danej iteracji, kolejność iteracji na schemacie jest od lewej do prawej, a pogrubione są ważne dla nas ilorazy różnicowe. Złożoność obliczeniowa algorytmu to $O(n^2)$, zaś pamięciowa to $O(n)$.

$\mathbf{f[x_0]}$	$\mathbf{f[x_0]}$	$\mathbf{f[x_0]}$	...	$\mathbf{f[x_0]}$	$\mathbf{f[x_0]}$
$f[x_1]$	$\mathbf{f[x_0, x_1]}$	$\mathbf{f[x_0, x_1]}$	...	$\mathbf{f[x_0, x_1]}$	$\mathbf{f[x_0, x_1]}$
$f[x_2]$	$f[x_1, x_2]$	$\mathbf{f[x_0, x_1, x_2]}$	...	$\mathbf{f[x_0, x_1, x_2]}$	$\mathbf{f[x_0, x_1, x_2]}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$f[x_{n-1}]$	$f[x_{n-2}, x_{n-1}]$	$f[x_{n-3}, x_{n-2}, x_{n-1}]$	...	$\mathbf{f[x_0, x_1, ..., x_{n-1}]}$	$\mathbf{f[x_0, x_1, ..., x_{n-1}]}$
$f[x_n]$	$f[x_{n-1}, x_n]$	$f[x_{n-2}, x_{n-1}, x_n]$	...	$f[x_1, x_2, ..., x_n]$	$\mathbf{f[x_0, x_1, ..., x_{n-1}, x_n]}$

1.2 Pseudokod

Algorithm 1: `ilorazyRoznicowe`

Data:
- x – wektor długości $n+1$ zawierający węzły x_0, ..., x_n,
- y – wektor długości $n+1$ zawierający wartości interpolowanej funkcji w węzłach: $y_0 = f(x_0)$, ..., $y_n = f(x_n)$.

Result: c – wektor długości $n+1$ zawierający obliczone ilorazy różnicowe: $c_0 = f[x_0]$, ..., $c_{n-1} = f[x_0, ..., x_{n-1}]$, $c_n = f[x_0, ..., x_n]$.

begin
- $c = y$
- **for** i ***from*** *1* ***to*** n **do**
  - **for** k ***from*** n ***down to*** i **do**
    - $c_k = \frac{c_k - c_{k-1}}{x_k - x_{k-i}}$
- **return** c

2 Zadanie 2.

2.1 Idea algorytmu

Zadaniem algorytmu jest wyliczenie wartości wielomianu interpolacyjnego w punkcie t. Wielomian jest w postaci Newtona, dane są potrzebne ilorazy różnicowe (wyliczane np. w zadaniu 1) oraz węzły interpolacji $x_0, ..., x_n$.

Korzystając z poniższego wzoru, łatwo jest zaimplementować algorytm działający w czasie $O(n^2)$:

$$p_n(x) = \sum_{i=0}^{n} c_i \cdot q_i(x) = \sum_{i=0}^{n} c_i \prod_{j=0}^{i-1}(x - x_j).$$

Wykonamy jednak takie przekształcenia, które sprowadzą czas działania algorytmu do liniowego. Możemy bowiem wyciągać wspólne czynniki przed nawias:

$$\sum_{i=0}^{n} c_i \prod_{j=0}^{i-1}(x - x_j) = f[x_0] + (x - x_0)(f[x_0, x_1] + \sum_{i=2}^{n} f[x_0, x_1, ..., x_i] \prod_{j=0}^{i-1}(x - x_j)) = .$$

$$= f[x_0] + (x - x_0)(f[x_0, x_1] + (x - x_1)(...(f[x_0, ..., x_{n-1}] + (x - x_{n-1})f[x_0, ..., x_n])...)).$$

Stosując kolejność obliczeń zgodną ze stopniem zagnieżdżenia nawiasów, otrzymujemy następującą zależność rekurencyjną, określaną uogólnieniem schematu Hornera na bazę wielomianów stopnia n postaci Newtona:

$$w_n(x) = f[x_0, ..., x_n]$$

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

$$p_n(x) = w_0(x)$$

Dzięki takim przekształceniom algorytm ma złożoność obliczeniową $O(n)$; złożoność pamięciowa to $O(1)$ (nie potrzebuje dodatkowych tablic poza danymi).

2.2 Pseudokod

Algorithm 2: `warNewton`

Data:
- x – wektor długości $n+1$ zawierający węzły x_0, ..., x_n,
- c – wektor długości $n+1$ zawierający ilorazy różnicowe: $c_0 = f[x_0]$, ..., $c_{n-1} = f[x_0, ..., x_{n-1}]$, $c_n = f[x_0, ..., x_n]$.
- t – punkt, w którym należy obliczyć wartość wielomianu interpolującego

Result: r – wartość wielomianu interpolującego w punkcie t.

begin
  $t = c_n$
  for i ***from*** $n-1$ ***down to*** 0 **do**
    $r = c_i + (t - x_i) \cdot r$
  return r

3 Zadanie 3.

3.1 Idea algorytmu

Algorytm ma za zadanie sprowadzić wielomian w postaci Newtona do wielomianu w postaci naturalnej, czyli zrównać

$$\sum_{i=0}^{n} c_i \prod_{j=0}^{i-1}(x - x_j) = p_n(x) = \sum_{i=0}^{n} a_i x^i.$$

Zastosowane podejście będzie inspirowane schematem Hornera wyprowadzonym w zadaniu 2., jednak konieczne będą pewne modyfikacje.

Na wstępie zauważmy, że $a_n = c_n = f[x_0, ..., x_n]$ i jest to pierwsze przypisanie wykonywane przez algorytm. Niestety obliczenie kolejnych współczynników $a_{n-1}, ..., a_1, a_0$ nie jest możliwe w czasie liniowym. Aby to zobrazować, pokażę początkowe iteracje „odwijania" schematu Hornera.

$$w_{n-1} = c_{n-1} + (x - x_{n-1})w_n = c_{n-1} + (x - x_{n-1})c_n,$$

stąd dostajemy informację, że przy x^{n-1} mamy $c_{n-1} - x_{n-1}c_n$. Ale gdy rozwiniemy wzór dalej:

$$w_{n-2} = c_{n-2} + (x - x_{n-2})w_{n-1} =$$

$$= c_{n-2} + x^2 c_n + x(c_{n-1} - x_{n-1}c_n - \mathbf{x_{n-2}c_n}) - x_{n-2}(c_{n-1} - x_{n-1}c_n),$$

skąd dowiadujemy się, że do współczynnika przy x^{n-1} musimy jeszcze dołożyć $-x_{n-2}c_n$. Analogiczna sytuacja występuje dla wszystkich współczyników a_k, $k < n$.

W związku z powyższym, w algorytmie przy każdym cofnięciu się w schemacie Hornera, poza wyznaczeniem „startowej" wartości współczynnika dla aktualnej potęgi rozwinięcia, trzeba do współczynników a_k o wyższych potęgach dodać odkryty w tym kroku składnik. Wynosi on $-x_i \cdot a_{j+1}$, gdzie i to wartość wykładnika potęgi x, przy której odwinięciu obecnie jesteśmy, a j jest indeksem aktualizowanego obecnie współczynnika a_j. Aktualizacja obejmuje współczynniki od $i+1$ do $n-1$ dla danego i, gdzie i z każdym krokiem odwinięcia schematu Hornera zmniejsza się, tj. $i = n-1, ..., 0$.

Algorytm działa w czasie $O(n^2)$ i wykorzystuje $O(n)$ pamięci.

3.2 Pseudokod

Algorithm 3: `naturalna`

Data:
- x – wektor długości $n+1$ zawierający węzły x_0, ..., x_n,
- c – wektor długości $n+1$ zawierający ilorazy różnicowe: $c_0 = f[x_0]$, ..., $c_{n-1} = f[x_0, ..., x_{n-1}]$, $c_n = f[x_0, ..., x_n]$.

Result: a – wektor długości $n+1$ zawierający obliczone współczynniki postaci naturalnej: a_0 przy x^0, a_1 przy x^1, ..., a_n przy x^n.

```
begin
    a_n = c_n
    for i from n − 1 down to 0 do
        a_i = c_i − x_i · a_{i+1}
        for j from i + 1 to n − 1 do
            a_j = a_j − x_i · a_{j+1}
    return a
```

4 Zadanie 4.

4.1 Idea algorytmu

W tym zadaniu wykorzystamy funkcje zaimplementowane w dwóch pierwszych zadaniach. Dla zadanej funkcji f, przedziału interpolacji $[a, b]$ i maksymalnego stopnia wielomianu interpolacyjnego n, chcemy zestawić graficznie wielomian interpolujący z funkcją f.

Zaczynamy od wyznaczenia $n+1$ równoodległych węzłów, czyli $x_0 = a < x_1 < ... < x_{n-1} < x_n = b$, gdzie $x_i = a + i \cdot h$, $h = \frac{b-a}{n}$ oraz $i = 0, 1, ..., n$. Dla takich węzłów x_i wyliczamy wartości $y_i = f(x_i)$. Teraz z pomocą metody `ilorazyRoznicowe(x,y)` wyznaczamy ilorazy różnicowe $c_i = f[x_0, x_1, ..., x_i]$, $i = 0, 1, ..., n$. Mając ilorazy różnicowe potrafimy już wyliczać wartości wielomianu interpolacyjnego w postaci Newtona w zadanym punkcie z przedziału interpolacji.

Aby móc uzyskać sensowny wykres, należy podzielić przedział interpolacji na znacznie mniejsze przedziały niż te pomiędzy węzłami. Najlepiej by było, żeby wśród punktów krańcowych nowopowstałych przedziałów znalazły się również same węzły interpolacji. W związku z tym liczbę punktów oznaczam jako $R \cdot n + 1$;

dostajemy wtedy R punktów dla każdego z przedziałów między węzłami ($[x_i, x_{i+1}]$), a dodanie jedynki umożliwia uwzględnienie prawego krańca b.

Dla określonych w ten sposób równoodległych punktów t_k wyliczamy wartości f i `warNewton(x, c, `t_k`)`, czyli wartości wielomianu interpolującego. Na końcu zestawiamy uzyskane dane na wykresie.

4.2 Pseudokod

Algorithm 4: `rysujNnfx`

Data:
- f – funkcja, którą chcemy interpolować
- $[a, b]$ – przedział interpolacji
- n – maksymalny stopień wielomianu interpolacyjnego.

Result: wykres $f(x)$ i wielomianu interpolacyjnego na $[a, b]$.

begin

$R = 100$ //rodzielczość rysowanego wykresu

$h = \frac{b-a}{n}$

for i ***from*** 0 ***to*** n **do**

$x_i = a + i \cdot h$ //węzły równoodległe

$y_i = f(x_i)$

$c =$ `ilorazyRoznicowe(`x, y`)`

liczbaPunktow $= R \cdot n + 1$ //+1 bo uwzględniamy kraniec b

odstep $= \frac{b-a}{\text{liczbaPunktow}-1}$

for i ***from*** 0 ***to*** *liczbaPunktow* **do**

$x_{wykres_i} = a + i \cdot$ odstep //argumenty na OX wykresu

$w_{wykres_i} =$`warNewton(`x, c, y`)` //wartości wielomianu do wykresu

$y_{wykres_i} = f(x_i)$ //wartości funkcji do wykresu

wykres = (argumenty: x_{wykres}, wartości: y_{wykres}, w_{wykres})

return *wykres*

5 Zadanie 5.

5.1 Cel zadania

Przetestować funkcję `rysujNnfx(f,a,b,n)` dla danych:

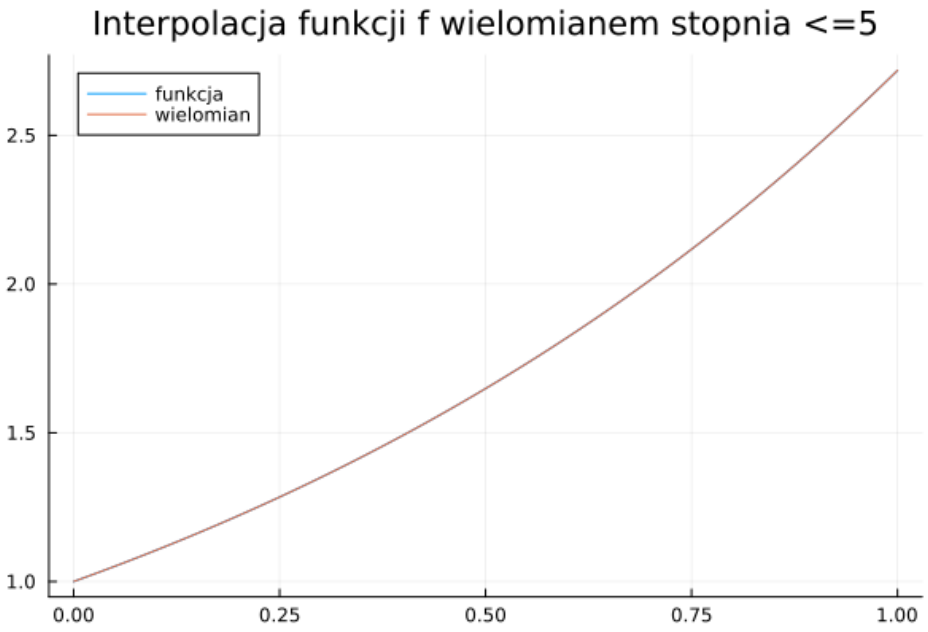
- $f(x) = e^x$, $[a, b] = [0, 1]$, $n \in \{5, 10, 15\}$,
- $f(x) = x^2 sin(x)$, $[a, b] = [-1, 1]$, $n \in \{5, 10, 15\}$.

5.2 Rozwiązanie

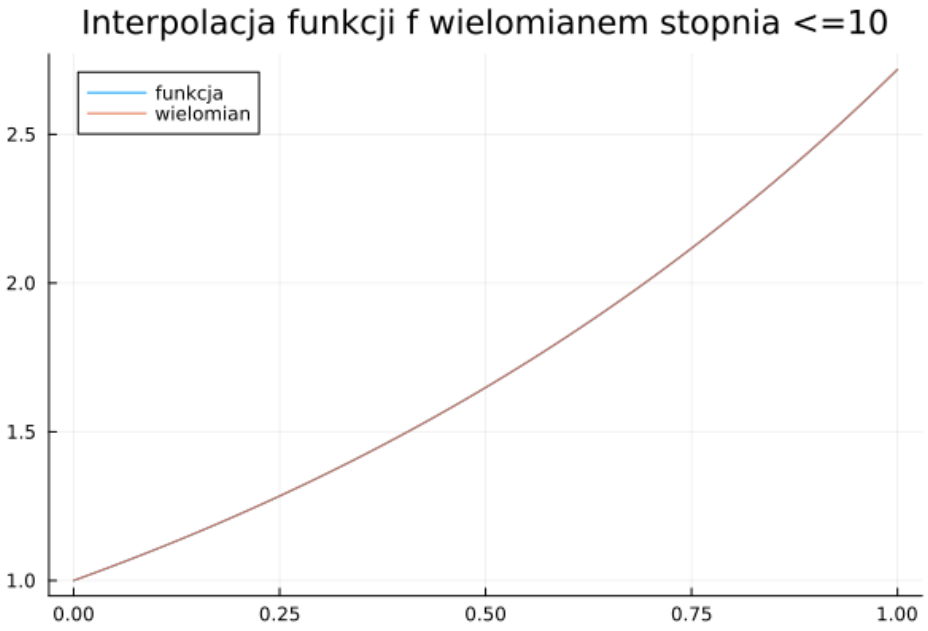
Zgodnie z celem zadania, wykorzystana została zaimplementowana w zadaniu 4. metoda `rysujNnfx(f,a,b,n)`. Znajduje się ona w module `Interpolacja` w pliku `interpolacja.jl`. W pętli po wartościach n metoda `rysujNnfx()` została wywołana najpierw dla pierwszej, a potem dla drugiej funkcji. Zwrócone przez metodę wykresy zostały zapisane do odpowiednich plików i przedstawione w następnym podrozdziale.

5.3 Interpretacja wyników i wnioski

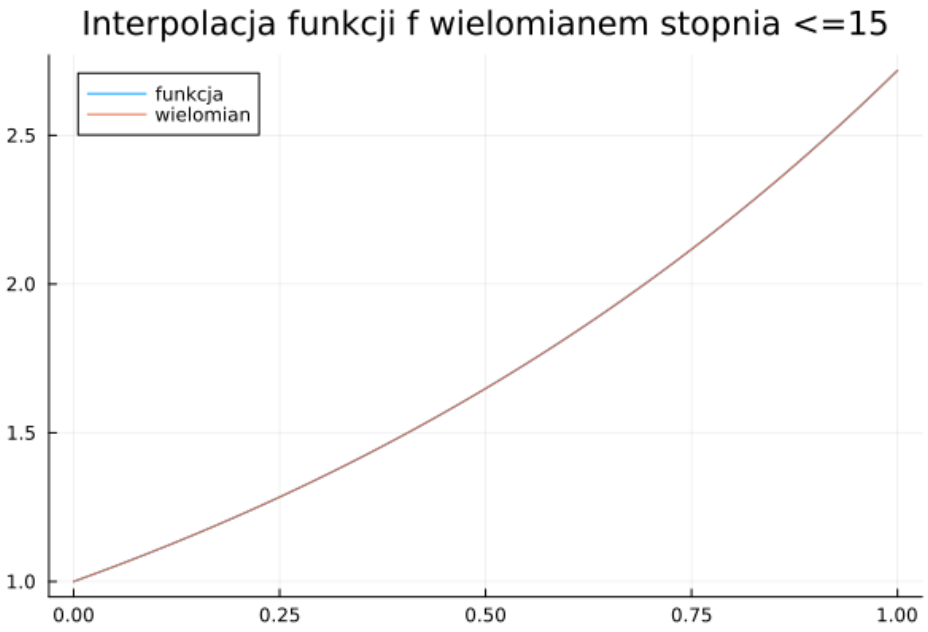
Poniższe wykresy są wynikiem działania metody `rysujNnfx(f,a,b,n)` dla podanych w celu zadania funkcji i reszty parametrów.



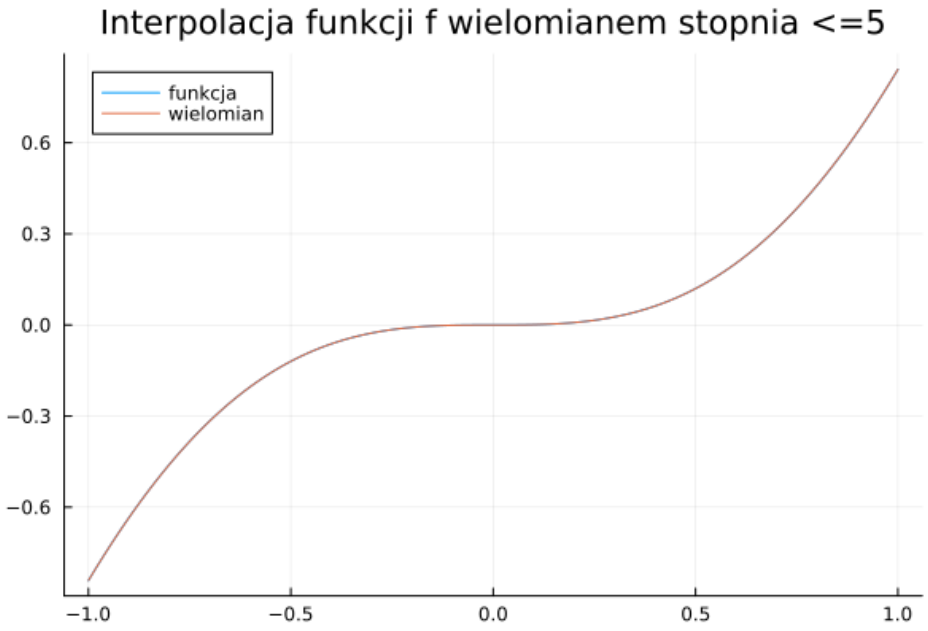
Rysunek 1: Wykres wielomianu interpolacyjnego stopnia $\leqslant 5$ i funkcji $f(x) = e^x$ na przedziale interpolacji $[0, 1]$.



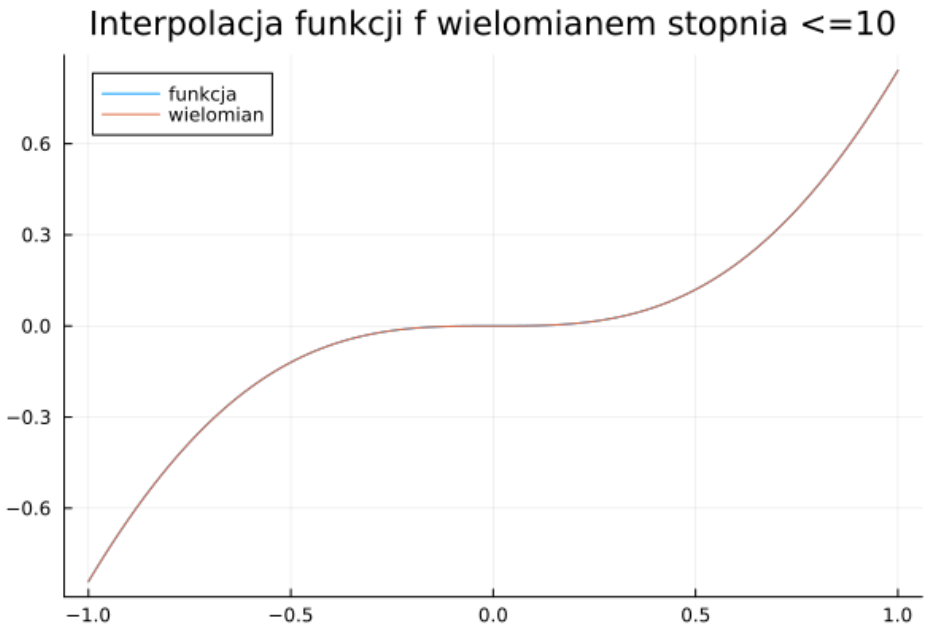
Rysunek 2: Wykres wielomianu interpolacyjnego stopnia $\leqslant 10$ i funkcji $f(x) = e^x$ na przedziale interpolacji $[0, 1]$.



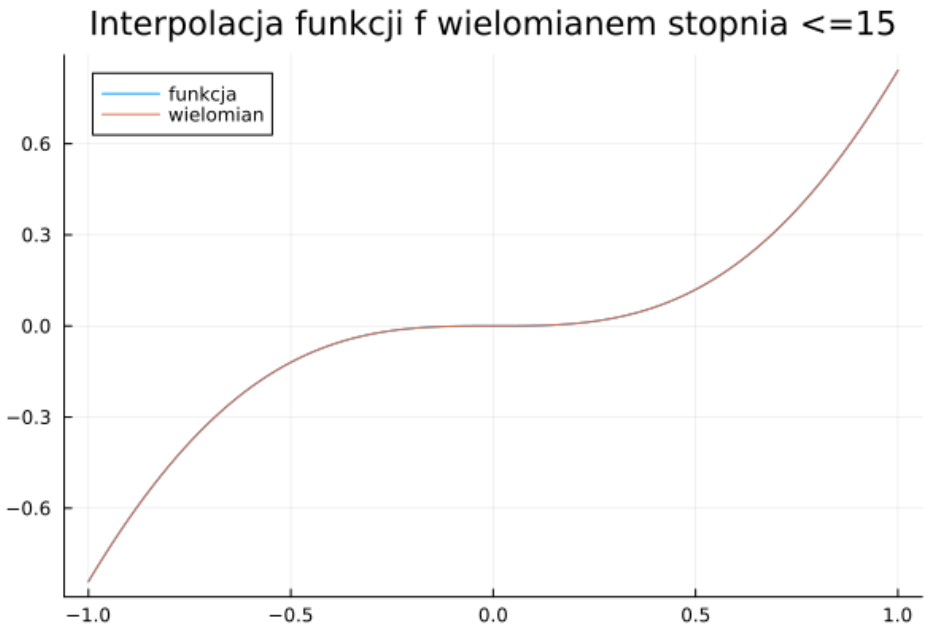
Rysunek 3: Wykres wielomianu interpolacyjnego stopnia $\leqslant 15$ i funkcji $f(x) = e^x$ na przedziale interpolacji $[0, 1]$.



Rysunek 4: Wykres wielomianu interpolacyjnego stopnia $\leqslant 5$ i funkcji $f(x) = x^2 sin(x)$ na przedziale interpolacji $[-1, 1]$.



Rysunek 5: Wykres wielomianu interpolacyjnego stopnia $\leqslant 10$ i funkcji $f(x) = x^2 sin(x)$ na przedziale interpolacji $[-1, 1]$.



Rysunek 6: Wykres wielomianu interpolacyjnego stopnia $\leqslant 15$ i funkcji $f(x) = x^2 sin(x)$ na przedziale interpolacji $[-1, 1]$.

W przypadku obu funkcji już dla wielomianu interpolacyjnego stopnia 5 nie widzimy na wykresie różnic między tym wielomianem a interpolowaną funkcją. Oznacza to, że te funkcje bardzo łatwo dają się interpolować wielomianami, a zaimplementowana metoda działa poprawnie.

6 Zadanie 6.

6.1 Cel zadania

Przetestować funkcję `rysujNnfx(f,a,b,n)` dla danych:

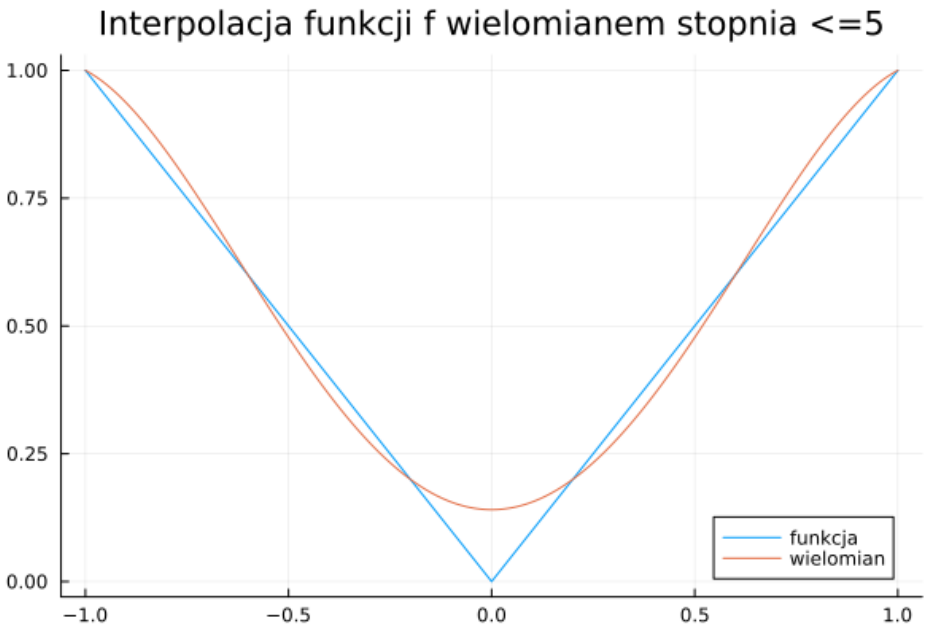
- $f_1(x) = |x|$, $[a, b] = [-1, 1]$, $n \in \{5, 10, 15\}$,
- $f_2(x) = \frac{1}{1+x^2}$, $[a, b] = [-5, 5]$, $n \in \{5, 10, 15\}$.

6.2 Rozwiązanie

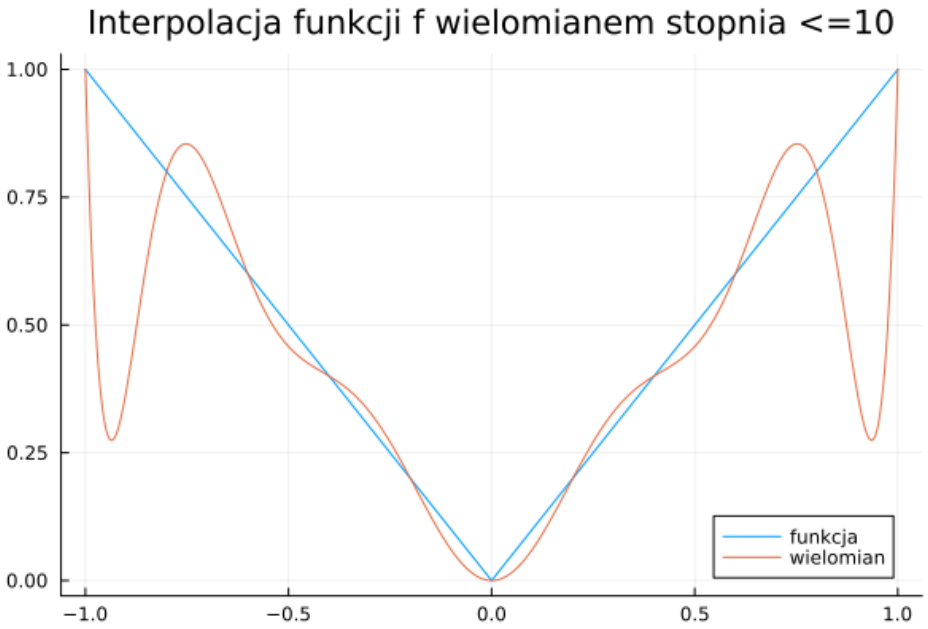
Tak jak w zadaniu 5., wykorzystana została zaimplementowana w zadaniu 4. metoda `rysujNnfx(f,a,b,n)`. Znajduje się ona w module `Interpolacja` w pliku `interpolacja.jl`. W pętli po wartościach n została ona wywołana najpierw dla pierwszej, a potem dla drugiej funkcji. Zwrócone przez metodę wykresy zostały zapisane do odpowiednich plików i przedstawione w następnym podrozdziale.

6.3 Interpretacja wyników i wnioski

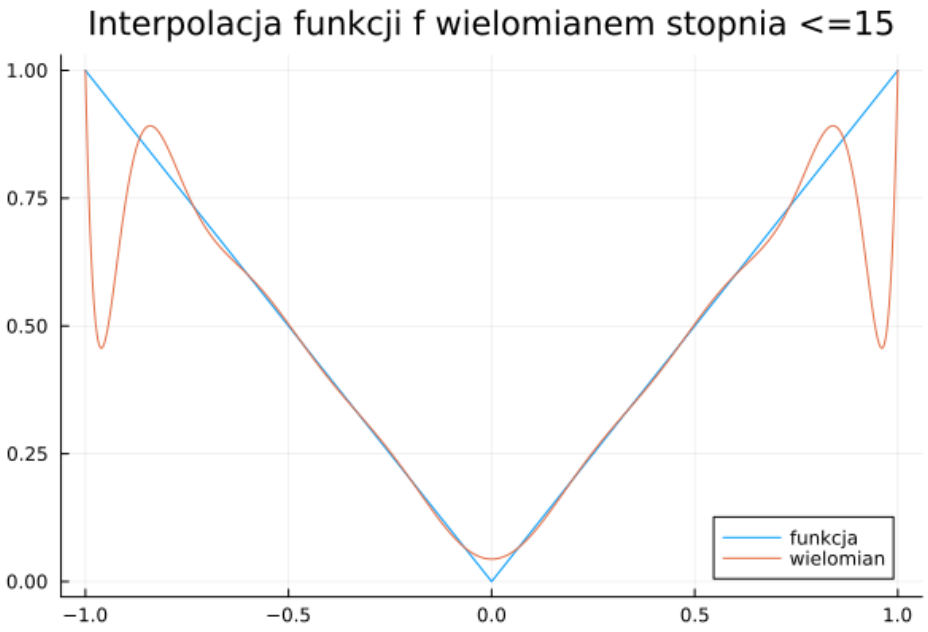
Poniższe wykresy są wynikiem działania metody `rysujNnfx(f,a,b,n)` dla podanych w celu zadania funkcji i reszty parametrów.



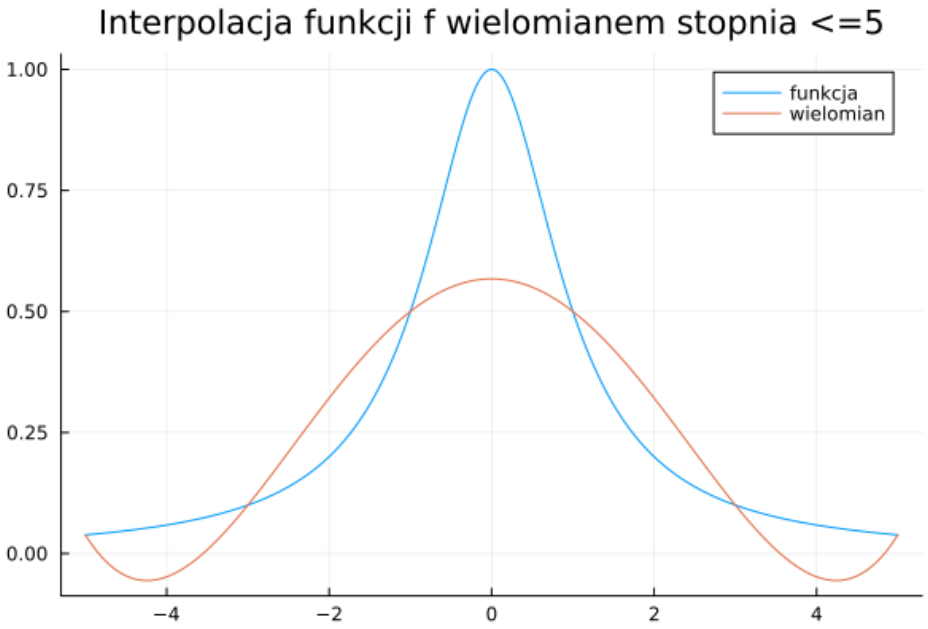
Rysunek 7: Wykres wielomianu interpolacyjnego stopnia $\leqslant 5$ i funkcji $f_1(x) = |x|$ na przedziale interpolacji $[-1, 1]$.



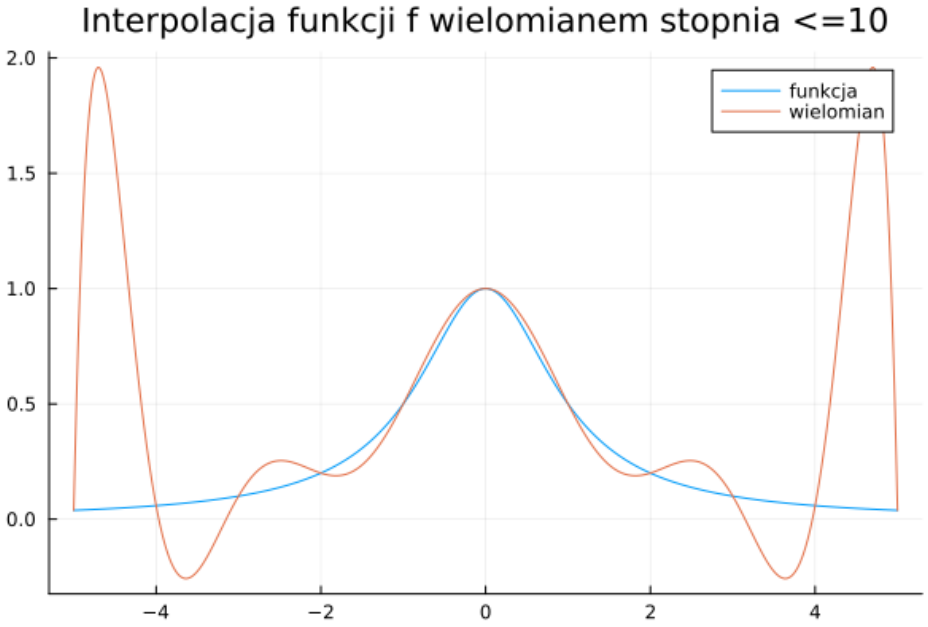
Rysunek 8: Wykres wielomianu interpolacyjnego stopnia $\leqslant 10$ i funkcji $f_1(x) = |x|$ na przedziale interpolacji $[-1, 1]$.



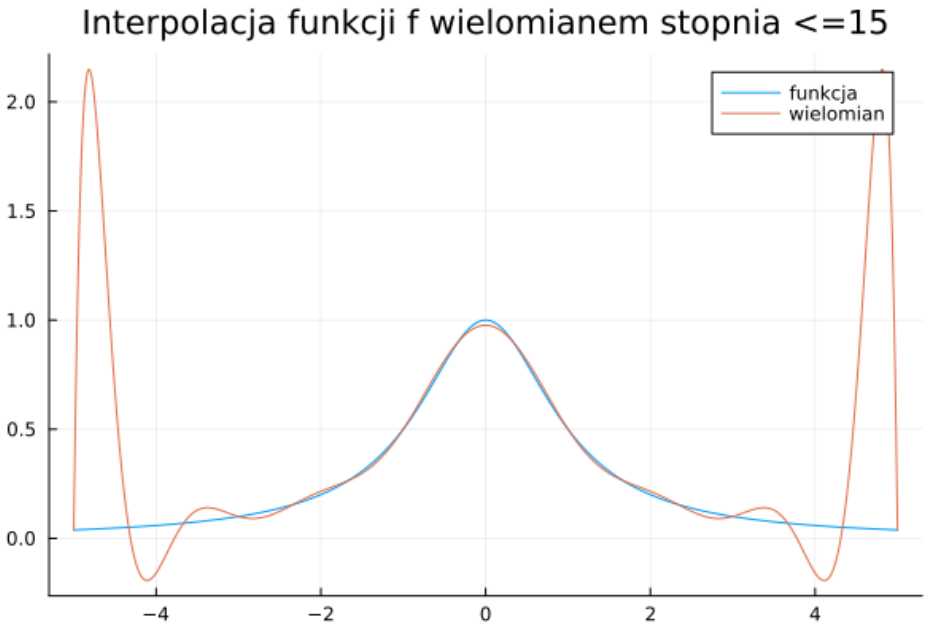
Rysunek 9: Wykres wielomianu interpolacyjnego stopnia $\leqslant 15$ i funkcji $f_1(x) = |x|$ na przedziale interpolacji $[-1, 1]$.



Rysunek 10: Wykres wielomianu interpolacyjnego stopnia $\leqslant 5$ i funkcji $f_2(x) = \frac{1}{1+x^2}$ na przedziale interpolacji $[-5, 5]$.



Rysunek 11: Wykres wielomianu interpolacyjnego stopnia $\leqslant 10$ i funkcji $f_2(x) = \frac{1}{1+x^2}$ na przedziale interpolacji $[-5, 5]$.



Rysunek 12: Wykres wielomianu interpolacyjnego stopnia $\leqslant 15$ i funkcji $f_2(x) = \frac{1}{1+x^2}$ na przedziale interpolacji $[-5, 5]$.

W przypadku tych funkcji widzimy istotne różnice w wartościach między funkcją a interpolującym ją wielomianem. Dodatkowo, zwiększenie stopnia wielomianu nie przynosi poprawy błędu interpolacji na całym przedziale.

Przy funkcji $f_1(x) = |x|$ istotnym problemem jest brak ciągłej pochodnej na całym przedziale interpolacji. Na wykresie widzimy „czubek" dla $x = 0$; przedstawienie go dokładnie z pomocą (gładkich z zasady) wielomianów jest bardzo trudne i prowadzi do silnych zniekształceń na krańcach przedziału.

Dla funkcji $f_2(x) = \frac{1}{1+x^2}$ sprawa komplikuje się jeszcze bardziej. Na wykresach zauważamy bardzo duży wzrost błędu interpolacji na krańcach przedziału wraz ze zwiększaniem stopnia wielomianu interpolującego. Zjawisko to nazywa się Efektem Rungego (od nazwiska Carla Rungego, niemieckiego matematyka).

W omówionym na wykładzie twierdzeniu o błędzie interpolacji określone zostało następujące równanie:

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\zeta_x)}{(n+1)!} \prod_{i=0}^{n} (x - x_i),$$

gdzie $\zeta \in (a, b)$, $f \in C^{n+1}[a, b]$ i p_n jest wielomianem interpolacyjnym stopnia n na węzłach $x_0, x_1, \ldots, x_n \in [a, b]$.

Zauważmy, że dla f_2 wartość pochodnej w punkcie szybko rośnie rzędowo wraz ze wzrostem n. Wzrost ten jest tak szybki, że przewyższa znajdujące się w mianowniku $(n+1)!$ oraz zmniejszające się odległości $\prod_{i=0}^{n}(x-x_i)$ dla węzłów równoodległych. Można teoretycznie pokazać, że górne ograniczenie błędu interpolacji

$$\lim_{n\to\infty} (\max_{a<x<b} |f(x) - p_n(x)|) = +\infty,$$

co w praktyce ujrzeliśmy na wykresach.

Najprostszym rozwiązaniem problemu rosnącego błędu interpolacji f_2 byłaby zmiana węzłów interpolacji z równoodległych na np. wyznaczane z pomocą wielomianów Czebyszewa; idea tej strategii została przedstawiona na wykładzie.