Sprawozdanie
Obliczenia naukowe - lista 5

Kamil Król

244949

Opis problemu oraz struktury danych

Celem listy jest rozwiązanie układu równań liniowych: $\boldsymbol{Ax} = \boldsymbol{b}$, dla danej macierzy $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ i wektora prawych stron $\boldsymbol{b} \in \mathbb{R}^n$, gdzie $n \geqslant 4$.

Zgodnie z treścią listy macierz $\boldsymbol{A}$ jest rzadką macierzą (tj. mającą dużo elementów zerowych) blokową o następującej strukturze:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_1 & \boldsymbol{C}_1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{B}_2 & \boldsymbol{A}_2 & \boldsymbol{C}_2 & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}_3 & \boldsymbol{A}_3 & \boldsymbol{C}_3 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{B}_{v-2} & \boldsymbol{A}_{v-2} & \boldsymbol{C}_{v-2} & \boldsymbol{0} \\ \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{B}_{v-1} & \boldsymbol{A}_{v-1} & \boldsymbol{C}_{v-1} \\ \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{B}_v & \boldsymbol{A}_v \end{pmatrix}, \tag{1}$$

$v = n/\ell$, zakładając że n jest podzielne przez ℓ, gdzie $\ell \geqslant 2$ jest rozmiarem wszystkich kwadratowych macierzy wewnętrznych (bloków) $\boldsymbol{A}_i$, $\boldsymbol{B}_i$, $\boldsymbol{C}_i$, $\boldsymbol{0}$.

Macierze oznaczone przez $\boldsymbol{0}$ są macierzami zerowymi tzn. $\boldsymbol{0} \in \mathbb{R}^{\ell\times\ell}$. Macierze $\boldsymbol{A}_i$, $\boldsymbol{B}_i$, $\boldsymbol{C}_i$, są następującej postaci:

(i) $\boldsymbol{A}_i \in \mathbb{R}^{\ell\times\ell}$, $i = 1, \ldots, v$ – macierze gęste,

(ii) $\boldsymbol{B}_i \in \mathbb{R}^{\ell\times\ell}$, $i = 2, \ldots, v$ – macierze mające tylko dwie ostatnie kolumny niezerowe:

$$\boldsymbol{B}_i = \begin{pmatrix} 0 & \cdots & 0 & b^i_{1\,\ell-1} & b^i_{1\,\ell} \\ 0 & \cdots & 0 & b^i_{2\,\ell-1} & b^i_{2\,\ell} \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & b^i_{\ell\,\ell-1} & b^i_{\ell\,\ell} \end{pmatrix}, \tag{2}$$

(iii) $\boldsymbol{C}_i \in \mathbb{R}^{\ell\times\ell}$, $i = 1, \ldots, v-1$ – macierze diagonalne:

$$\boldsymbol{C}_i = \begin{pmatrix} c^i_1 & 0 & 0 & \cdots & 0 \\ 0 & c^i_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c^i_{\ell-1} & 0 \\ 0 & \cdots & 0 & 0 & c^i_\ell \end{pmatrix}. \tag{3}$$

Dodatkową informacją o danych jest ich duży rozmiar. W konsekwencji wyklucza to pamiętanie macierzy $\boldsymbol{A}$ jako tablicę o wymiarach $n\times n$ oraz użycie standardowych (bibliotecznych) algorytmów dla macierzy gęstych tj. takich, gdzie nie zakłada się dużej liczby elementów zerowych. Zatem dodatkowym wymaganiem co do zadania jest specjalna struktura pamiętająca efektywnie macierz $\boldsymbol{A}$. Taka która pamięta tylko elementy niezerowe macierzy $\boldsymbol{A}$. W tym celu w moich programach użyłem biblioteki *SparseArrays*, która pamięta tylko elementy niezerowe. W naszych rozważaniach założymy, że możemy uzyskać dostęp do elementów macierzy w czasie stałym – $O(1)$. W rzeczywistości tak nie jest.
Ponadto konieczna jest modyfikacja standardowych algorytmów tak, aby uwzględniały one specyficzną postać macierzy $\boldsymbol{A}$, tj. jej rzadkość, regularność występowania elementów zerowych i niezerowych.

Zaimplementowane przeze mnie algorytmy (napisane w języku Julia i opisane poniżej) działają na macierzy transponowanej, ze względu na optymalizację tzn. dostęp do elementów po kolumnach jest szybszy niż po wierszach. W dalszych rozważaniach pominiemy ten fakt gdyż transpozycja nie zmienia wewnętrznej struktury macierzy $\boldsymbol{A}$ oraz nie zmienia ogólnej idei algorytmu. Odnośnik do dokumentacji: julialang - SparseArrays.

Opis metody eliminacji Gaussa bez wyboru elementu głównego

Ideą eliminajci Gaussa jest to aby początkowy układ równań sprowadzić do układu prostszego. Dokładniej chodzi o sprowadzenie macierzy A do postaci macierzy trójkątnej górnej. W celu doprowadzenia macierzy do takiej postaci algorytm posługuje się podstawowymi operacjami na macierzy tzn. dodawanie, odejmowanie wierszy oraz mnożenie wierszy przez pewnien niezerowy współczynnik.
Dokładniej, proces polega na eliminowaniu (zerowaniu) elementów znajdujących się w pod przekątną macierzy. Zatem dla kolumny pierwszej będziemy chcieli wyzerować wszystkie elementy znajdujące się pod pierwszym elementem w tej kolumnie, dla kolumny drugiej – wszystkie elementy pod drugim (patrząc od góry) elementem w danej kolumnie. Bardziej formalnie, naszym celem będzie wyeliminowanie niewiadomej x_1 ze wszystkich $n-1$ pozostałych równań (czyli wszystkich poza pierwszym). Dla $i = 2, \ldots, n$ będziemy odejmować odpowiednią wielokrotność równania pierwszego od i-tego równania zerując przy tym współczynnik przy x_1 w i-tym równaniu.
Ogólnie dla macierzy A i jej elementów a_{ij}, gdzie $i, j = 1, \ldots, n$ i dla jej k-tej kolumny, będziemy zerować wszystkie elementy poniżej elementu na przekątnej w tej kolumnie, czyli a_{kk}. Dla $i = k+1, \ldots, n$ będziemy odejmować odpowiednią wielokrotność równania k-tego od i-tego równania zerując przy tym współczynnik przy x_k w i-tym równaniu. Inaczej aby wyzerować to x_k będziemy musieli pomnożyć wiersz k przez pewien współczynnik ψ_{ik}, króty dalej będziemy nazywać mnożnikiem, a następnie odjąć go od i-tego wiersza. Łatwo zauważyć, że mnożnik ten możemy obliczyć, ze wzoru $\psi_{ik} = \frac{a_{ik}}{a_{kk}}$. Widać też, że w przypadku kiedy na diagonali pojawi się element równy zero nasz algorytm nie zadziała, ponieważ pojawi się konieczność dzielenia przez zero.
Zanim zobaczymy jak można rozwiązać ten problem musimy przytoczyć jedną z własności układu równań Ax=b. Mianowicie, wiersze macierzy A i odpowiadające im wartości w wektorze b możemy dowolnie ze sobą zamieniać. Nie zmianiamy w ten sposób układu równań. Zwróćmy również uwagę na fakt, że aby istniało rozwiązanie tego równania macierz A musi być nieosobliwa. Teraz możemy przedstawić sposób na uniknięcie dzielenia przez 0.

Opis metody eliminacji Gaussa częściowym wyborem elementu głównego

Przedstawiona poniżej metoda nosi nazwę metody eliminacjii Gaussa z częściowym wyborem elementu głównego. Czym jest element główny? Jest to element macierzy znajdujący się na diagonali. Dla k-tej kolumny, elementem głównym będzie element a_{kk} macierzy A. Jest to również ten element przez który dzielimy chcąc obliczyć mnożnik ψ_{ik} (k oraz i są opisane tak samo jak w poprzednim paragrafie). Częściowy wybór polega na wybraniu największego elementu co do wartości bezwzględnej z danej kolumny k oraz zamianie wierszy tak aby ten element stał się elementem głównym – elementem na diagonali. Możemy to zrobić, ponieważ macierz A jest nieosobliwa.
Zamiana wierszy macierzy jest kosztownym prcesem, który może być zastąpiony przez wektor permutacji wierszy p. Zapamiętujemy w nim, na którym miejscu aktualnie znajduje się dany wiersz. Zatem jedyna różnica w samym algorytmie będzie taka, że zamiast odwoływać się do danego wiersza bezpośrednio, będziemy się odwoływać do jego pozycji zapisanej w wektorze permutacji. Zwróćmy uwagę na to, że zmieniając wiersze w macierzy A musimy zmieniać też poszczególne współrzędne w wekotrze prawych stron – b. Złożoność obu przedstawionych powyżej algorytmów wynosi $O(n^3)$.

Teraz kiedy znamy procedurę, która pozwala nam sprowadzić macierz A do macierzy trójkątnej górnej, pojawia się pytanie jak z tą wiedzą rozwiązać układ równań $Ax = b$. W tym celu zastosujemy algorytm podstawiania wstecz. Opiera się on o wzór poniżej.

$$x_i = \frac{b_i - \sum_{j=i+1}^{n}(a_{ij} \cdot x_j)}{a_{ii}}$$

Widać zatem, że idąc *od końca* ($i = n, \ldots, 1$) jesteśmy w stanie obliczyć wszystkie wartości x. Złożoność tego algorytmu wynosi $O(n^2)$, a cały proces prowadzący do rozwiązania równania ma złożoność $O(n^3)$ (bo eliminacja Gaussa ma złożoność $O(n^3)$).
Widać tu, że najkosztowniejszym fragmentem opisanego wyżej algorytmu rozwiązywania układu równań jest eliminacja Gaussa. Przedstawiony poniżej rozkład LU będzie umożliwiał rozwiązywanie układu równań dla różnych wektorów prawych stron przy tylko jednym wywołaniu metody eliminacji Gaussa zajmującej $O(n^3)$. Rozwiązywanie kolejnych układów równań z tą samą macierzą A będzie robione w czasie $O(n^2)$.

Zadanie 1

Celem tego zadania było napisanie funkcji rozwiązującej układ $\boldsymbol{Ax} = \boldsymbol{B}$ metodą eliminacji Gaussa uwzględniającej specyficzną postać macierzy $\boldsymbol{A}$ dla dwóch wariantów:

a) bez wyboru elementu głównego,

b) z częściowym wyborem elementu głównego.

Znając wyżej opisane algorytmy należało zmodyfikować je pod opisaną w treści listy zadań strukturę. Wiemy, że rozwiązanie naszego równania z użyciem niezmodyfikowanych algorytmów ma złożoność $O(n^3)$. Celem jest zrobienie tego samego w czasie $O(n)$. Zakładamy tu, że rozmiar bloku ℓ jest stałą. Inne założenie opisane wyżej to założenie, że dostęp do elementu macierzy (Sparse Arrays) jest w czasie stałym.
Zastosowane modyfikacje
Macierz $\boldsymbol{A}$ jest macierzą rzadką o specyficznej blokowo-trójdiagonalnej strukturze. Oznacza to, że większość elementów w danej kolumnie nie będzie wymagało zerowania. Znana i uporządkowana struktura pozwala nam na ograniczenie ilości wykonywanych operacji, gdyż są one wymagane tylko dla pewnego ograniczonego obszaru macierzy $\boldsymbol{A}$. Wyznaczymy teraz te ograniczenia. W celu przedstawienia metody rozważmy macierz $\boldsymbol{A}$ dla $\ell = 4$ i jakiegoś n.

$$\boldsymbol{A} = \begin{pmatrix}
a^1_{11} & a^1_{12} & a^1_{13} & a^1_{14} & c^1_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
a^1_{21} & a^1_{22} & a^1_{23} & a^1_{24} & 0 & c^1_{22} & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
a^1_{31} & a^1_{32} & a^1_{33} & a^1_{34} & 0 & 0 & c^1_{33} & 0 & 0 & 0 & 0 & 0 & \cdots \\
a^1_{41} & a^1_{42} & a^1_{43} & a^1_{44} & 0 & 0 & 0 & c^1_{44} & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & b^2_{13} & b^2_{14} & a^2_{11} & a^2_{12} & a^2_{13} & a^2_{14} & c^2_{11} & 0 & 0 & 0 & \cdots \\
0 & 0 & b^2_{23} & b^2_{24} & a^2_{21} & a^2_{22} & a^2_{23} & a^2_{24} & 0 & c^2_{22} & 0 & 0 & \cdots \\
0 & 0 & b^2_{33} & b^2_{34} & a^2_{31} & a^2_{32} & a^2_{33} & a^2_{34} & 0 & 0 & c^2_{33} & 0 & \cdots \\
0 & 0 & b^2_{43} & b^2_{44} & a^2_{41} & a^2_{42} & a^2_{43} & a^2_{44} & 0 & 0 & 0 & c^2_{44} & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & b^3_{13} & b^3_{14} & a^3_{11} & a^3_{12} & a^3_{13} & a^3_{14} & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & b^3_{23} & b^3_{24} & a^3_{21} & a^3_{22} & a^3_{23} & a^3_{24} & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & b^3_{33} & b^3_{34} & a^3_{31} & a^3_{32} & a^3_{33} & a^3_{34} & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & b^3_{43} & b^3_{44} & a^3_{41} & a^3_{42} & a^3_{43} & a^3_{44} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$

Kiedy przyjrzymy się czterem (ℓ) pierwszym kolumnom, widzimy, że musimy w nich wyzerować kolejno 3, 2, 1+4, 0+4 elementów. Zatem pętla która *idzie* po wierszach danej kolumny będzie mogła zakończyć pracę wcześniej. W naszym przypadku na rysunku, w przypadku kolumny pierwszej, pętla będzie mogła zakończyć pracę już po 3 iteracjach, ponieważ wszystkie elementy poniżej a^1_{41} są już zerami. Ogólnie jeśli przyjrzymy się $\ell - 2$ pierwszym kolumnom to możemy zauważyć, że jedyne elementy niezerowe występują w bloku $\boldsymbol{A}_1$, a więc tylko w ℓ pierwszych wierszach. Rozpatrując kolejne ℓ kolumn widać, że niezerowe elementy będą się znajdować w bloku $\boldsymbol{B}_2$ lub $\boldsymbol{C}_1$, lub w bloku $\boldsymbol{A}_2$, a więc poniżej tych bloków, czyli poniżej wiersza o numerze 2ℓ, nie pojawią się żadne elementy niezerowe. Biorąc kolejne ℓ kolumn sytuacja jest podobna – elementy niezerowe wystąpią **najniżej** w bloku $\boldsymbol{B}_3$ lub $\boldsymbol{A}_3$, a więc nie niżej niż wiersz o numerze 3ℓ licząc od góry (3ℓ pierwszych wierszy). Patrząc na przedstawioną wyżej zależność możemy wywnioskować wzór na ostatni element niezerowy w danej kolumnie k.

$$ostatni_{wiersz}(k) = min\{\ell + \ell \times \lfloor \frac{k+1}{\ell} \rfloor, n\}, k - \text{indeks kolumny}$$

Kolejna obserwacja to fakt, że podczas odejmowania wiersza z elementem głównym od innego wiersza nie musimy obliczać tych elementów dla których wiersz główny ma zera. Niech wiersz k będzie naszym wierszem głównym (czyli tym zawierającym element główny), a obecnie eliminowany element będzie w

wierszu i-tym. Oznacza to, że wykonujemy działanie na wierszach $R_i \leftarrow R_i - \psi_{ik} \cdot R_k$, gdzie R_i i R_k to odpowiednio i-ty i k-ty wiersz, a ψ_{ik} to mnożnik (przedstawiony w opisie podstawowej metody eliminacji Gaussa). Teraz widać, że dla j-tego elementu wiersza R_k będącego zerem, nazwijmy go $r_j^k = 0$, wykonanie powyższego działania nie zmieni odpowiedniej wartości j-tego elementu wiersza R_i. Wynika z tego, że pętla która wykonuje to działanie na wierszach może skończyć wcześniej pracę. Teraz wyprowadzimy wzór na ostatnią kolumnę, która może zawierać niezerowy element.
Zauważmy, że ostatnie elementy w wierszach to te leżące na diagonali bloków $\boldsymbol{C}$. Są one odległe od diagonali macierzy $\boldsymbol{A}$ o dokładnie ℓ. Zatem ostatni element niezerowy w wierszu w będzie się znajdował w kolumnie o indeksie $min\{\ell + w, n\}$. Podsumowując:

$$ostatnia_{kolumna}(w) = min\{\ell + w, n\}, w - \text{indeks wiersza}$$

Udało się zatem określić ograniczenia pętli w metodzie eliminacji Gaussa, które znacznie zredukują liczbę wykonywanych operacji. Kolejny etap rozwiązywania układu równań to algorytm podstawiania wstecz, który również może zostać zmodyfikowany w celu redukcji wykonywanych operacji.
W algorytmnie tym sumujemy elementy w danym wierszu pomnożone przez pewne liczby. My jednak wiemy, że znaczna część z tych elementów jest zerami, a więc wyniki iloczynów które otrzymamy też będą zerami nie zmianiając tym wartości sumy. Ponadto wiemy na jakiej pozycji występuje ostatni element niezerowy. Jest to zależność wyprowadzona powyżej tzn. wzór na $ostatnia_{kolumna}$. Możemy zatem zakończyć pętlę wcześniej redukując tym liczbę operacji.
Poniżej znajduje się pseudokod omówionego algorytmu w celu podsumowania naszych rozważań.

Algorytm 1: Zmodyfikowana metoda eliminacjii Gaussa

Dane wejściowe:

- $\boldsymbol{A}$ – macierz z zadania o opisanej w zadaniu strukturze,
- $\boldsymbol{b}$ – wektor prawych stron,
- n – rozmiar macierzy $\boldsymbol{A}$,
- ℓ – rozmiar bloku macierzy $\boldsymbol{A}$.

Dane wyjściowe:

- $\boldsymbol{x}$ – wektor zawierający rozwiązania układu $\boldsymbol{Ax} = \boldsymbol{b}$.

```
function eliminacja_gaussa(A, b, n, ℓ)
    for k ← 1 to n − 1 do
        ostatni_wiersz ← min{ℓ + ℓ · ⌊(k+1)/ℓ⌋, n}
        ostatnia_kolumna ← min(k + ℓ, n)
        for i ← k + 1 to ostatni_wiersz do
            if A[k, k] = 0 then
                error współczynnik na przekątnej równy zeru
            ψ ← A[i,k] / A[k,k]
            A[i, k] ← 0
            for j ← k + 1 to ostatnia_kolumna do
                A[i, j] ← A[i, j] − ψ · A[k, j]
            b[i] ← b[i] − ψ · b[k]
    for i ← n downto 1 do
        ostatnia_kolumna ← min(i + ℓ, n)
        for j ← k + 1 to ostatnia_kolumna do
            suma ← suma + x[i] · A[i, j]
        x[i] ← (b[i] − suma) / A[i,i]
    return x
```

Pozostało jeszcze określenie złożoności powyższego algorytmu. Widać, że pierwsza pętla wykona się $n-1$ ($O(n)$) razy, a pierwsza wewnątrz niej maksymalnie 2ℓ razy. Kolejna pętla wykonująca odejmowanie wierszy wykona się maksymalnie ℓ razy. ℓ jest stałą, zatem sama eliminacja Gausa wykonuje się w czasie $O(n)$. Teraz przyjrzyjmy się algorytmowi podstawiana wstecz. Zewnętrzna pętla wykona się $O(n)$ razy, a wewnętrzna co najwyżej ℓ, co daje łącznie $O(n)$. Zatem złożoność całego przedstawionego wyżej algorytmu rozwiązującego układ równań $\boldsymbol{Ax} = \boldsymbol{b}$ wynosi $O(n)$.

Zadanie 3

Celem tego zadania było napisanie funkcji obliczającej współczynniki $a_0, \ldots, a_n$ postaci naturalnej wielomianu interpolacyjnego dla zadanych współczynników $d_0 = f[x_0], d_1 = f[x_0, x_1], \ldots d_n = f[x_0, \ldots, x_n]$

tego wielomianu w postaci Newtona oraz węzłów $x_0, \ldots, x_n$. Ponadto funkcja miała działać w czasie $O(n^2)$.

Dane:

`x` – wektor długości $n+1$ zawierający węzły $x_0, \ldots, x_n$,

`fx` – wektor długości $n+1$ zawierający ilorazy różnicowe.

Oczekiwany wynik:

`a` – wektor długości $n+1$ zawierający obliczone współczynniki postaci naturalnej.

Opis:

Przypomnijmy, że wartości $d_0, d_1, \ldots, d_n$ są współczynnikami wielomianu interpolacyjnego w postaci Newtona. Punktem wyjściowym to wyprowadzenia algorytmu będzie uogólniony algorytm Hornera. Najpierw jednak zapiszmy wielomian interpolacyjny w postaci Newtona:

$$p(x) = \underbrace{d_0 + (x - x_0)\underbrace{(d_1 + (x - x_1)(d_2 + \ldots + (x - x_{n-2})\overbrace{(d_{n-1} + (x - x_{n-1})\underbrace{d_n}_{W_n})}^{W_{n-1}})\ldots)}_{W_1}}_{W_0}$$

Idea jest bardzo podobna do wyprowadzania uogólnionego algorytmu Hornera w poprzednim zadaniu. Teraz przyjrzyjmy się zaznaczonym wielomianom W_k. Zauważmy też, że ich stopnie rosną (patrząc od góry do dołu).

$$\begin{aligned}
W_n(x) &= d_n \\
W_{n-1}(x) &= d_{n-1} + (x - x_{n-1})W_n \\
\ldots &= \ldots \\
W_k(x) &= d_k + (x - x_k)W_{k+1} \text{ dla } 0 \leqslant k < n \\
\ldots &= \ldots \\
W_0(x) &= p_0(x)
\end{aligned}$$

Dodatkowo mamy też, że $deg(W_k) = deg(W_{k+1}) + 1$. Obie te obserwacje są kluczowe dla wyprowadzenia algorytmu. Przypomnijmy, że wartości $d_0, d_1, \ldots, d_n$ oraz $x_0, \ldots, x_n$ są danymi, a więc są znane. Zastanówmy się jak obliczyć współczynniki wielomianu W_{n-1} w postaci naturalnej. Mamy $deg(W_{n-1}) = deg(W_n) + 1 = 0 + 1 = 1$, a zatem W_{n-1} możemy ogólnie zapisać jako $W_{n-1}(x) = a_0x^0 + a_1x^1$. Z drugiej strony patrząc na tabelę wyżej możemy go zapisać jako

$$W_{n-1}(x) = d_{n-1}+(x-x_{n-1})W_n = d_{n-1}+(x-x_{n-1})d_n = d_{n-1}+xd_n-x_{n-1}d_n = (d_{n-1}-d_nx_{n-1})x^0+(d_n)x^1$$

Otrzymaliśmy współczynniki naturalne wielomianu W_{n-1}. Konkretniej mamy, że $a_0 = d_{n-1} - d_nx_{n-1}$ i $a_1 = d_n$. Zróbmy to samo dla W_{n-2}.

$$W_{n-2}(x) = d_{n-2} + (x - x_{n-2})W_{n-1} = d_{n-2} + (x - x_{n-2})(a_0x^0 + a_1x^1) =$$

$$(d_{n-2} - x_{n-2}a_0)x^0 + (a_0 - a_1x_{n-2})x^1 + a_1x^2$$

Obliczyliśmy współczynniki W_{n-2} w postaci naturalnej. Jeśli ten wielomian zapiszemy jako $W_{n-2} = b_0x^0 + b_1x^1 + b_2x^2$ to współczynniki będą następujące: $b_0 = d_{n-2} - x_{n-2}a_0$, $b_1 = a_0 - a_1x_{n-2}$, $b_2 = a_1$. Widzimy zatem, że współczynniki przy najwyższej potędze wielomianów W_{n-1} i W_{n-2} są sobie równe. Ogólnie mamy, że współczynniki przy najwyższej potędze dla wielomianów W_k i W_{k-1} są sobie równe. Inna kluczowa obserwacja to fakt, że licząc współczynniki naturalne wielomianu W_{n-2} korzystaliśmy tylko z danych i ze współczynników naturalnych wielomianu W_{n-1}. Podobnie obliczając współczynniki wielomianu W_{n-1} korzystaliśmy tylko z danych i współczynników wielomianu W_n. Widzimy zatem, że współczynniki naturalne wielomianu W_k jesteśmy w stanie obliczyć znając współczynniki wielomianu W_{k+1}. Oznacza to, że możemy to zrobić używając jednej tablicy. Ponadto jesteśmy w stanie zrobić to w czasie $O(n)$. W szczególności możemy obliczyć współczynniki wielomianu $W_0 = p_0$ w postaci naturalnej licząc kolejno współczynniki wielomianów W_n, W_{n-1}, ..., W_1, W_0 każdy w czasie $O(n)$ co daje łączny czas $O(n^2)$. Teraz dla podsumowania pseudokod.

Pseudokod algorytmu

Algorytm 2: Obliczanie współczynników naturalnych wielomianu interpolacyjnego.

```
function naturalna(x, fx)
    n ← length(fx)-1
    a[n] ← fx[n]
    for i ← n − 1 downto 0 do
        a[i] ← fx[i] − a[i + 1] ∗ x[i]
        for j ← i + 1 to n − 1 do
            a[j] ← a[j] − a[j + 1] ∗ x[i]
    return a
```

Zadanie 4

Celem zadania było napisanie funkcji interpolującej zadaną funkcję $f(x)$ w przedziale $[a, b]$ za pomocą wielomianu interpolacyjnego stopnia n w postaci Newtona, a także rysującej wykresy funkcji f oraz otrzymanego wielomianu interpolacyjnego. W interpolacji funkcji należało użyć węzłów równoodległych.
Dane:

`f` – zadana funkcja,

`a, b` – przedział interpolacji,

`n` – stopień wielomianu interpolacyjnego.

Oczekiwany wynik:

– wykres funkcji f oraz wielomianu interpolacyjnego w przedziale $[a, b]$.

Opis:
Na początku wyznaczyłem węzły interpolacyjne $x_1, \ldots, x_{n+1}$ w taki sposób aby odległość między nimi wynosiła $\frac{b-a}{n}$. Następnie obliczyłem wartości funkcji w tych punktach tj. $f(x_1), \ldots, f(x_{n+1})$. W celu obliczenia ilorazów różnicowych posłużyłem się funkcją z zadania pierwszego – `ilorazyRoznicowe`. Następnie użyłem funkcji z zadania drugiego tj. `warNewton` do obliczenia wartości wielomianu interpolacyjnego w potrzebnych punktach. W celu uzyskania dokładniejszego wykresu, punkty dla których rysowałem wykres musiałem zagęścić. Zrobiłem to mnożąc dane n razy obrany przeze mnie parametr gęstości równy 40. Dzięki temu uzyskałem dokładniejsze wykresy.

Zadanie 5

Celem zadania było przetestowanie funkcji `rysujNnfx(f,a,b,n)` (z zadania 4) na następujących przykładach:

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

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

Poniżej narysowane wykresy dla obu funkcji. Na przykładach tych funkcji widać, że wybranie równoodległych węzłów dało bardzo dokładne przybliżenia funkcji. Dla żadnego z wykresów nie zaobserwowano rozbieżności. Kolejna rzecz warta zaobserwowania to fakt, że dla wszystkich wartości n funkcje były bardzo dobrze przybliżone.

Wykres funkcji e^x i jej wielomianu interpolacyjnego dla danego stopnia n

Wykres funkcji $x^2 \sin x$ i jej wielomianu interpolacyjnego dla danego stopnia n

Zadanie 6

Celem zadania było przetestowanie funkcji `rysujNnfx(f,a,b,n)` (z zadania 4) na następujących przykładach:

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

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

Wykresy otrzymane za pomocą metody `rysujNnfx(f,a,b,n)` prezentują poniższe wykresy.

Wykres funkcji $|x|$ i jej wielomianu interpolacyjnego dla danego stopnia n

Wykres funkcji $\frac{1}{1+x^2}$ i jej wielomianu interpolacyjnego dla danego stopnia n

Widać, że dla funkcji $|x|$ pojawiają się większe odchylenia niż dla funkcji z zadania poprzedniego. Widać też, że wraz ze zwiększaniem stopnia wielomianu odchylenia/błędy, w szczególności na końcach przedziału, znacznie wzrastają. Dla $n = 20$ te odchylenia są już na tyle duże, że wykres traci na czytelności. Dla drugiej funkcji sytuacja wygląda tak samo – wraz ze zwiększaniem stopnia wielomianu błąd rośnie, a błędy najbardziej rosną na końcach przedziału. Jest to sprzeczne z intuicją, ponieważ zwiększajac stopień wielomianu interpolacyjnego oczekujemy lepszego przybliżenia. Zaobserwowane zjawisko nosi nazwę zjawiska Rungego. (Sama funkcja $\frac{1}{1+x^2}$ nazywana jest funkcją Rungego). Polega ono na tym, że dla pewnych funkcji błąd wielomianu interpolacyjnego wyliczonego za pomocą równoodległych węzłów dąży do nieskończoności wraz ze wzrostem stopnia tego wielomianu interpolacyjnego.
Pojawia się pytanie czy można poprawić dokładność wielomianów interpolacyjnych dla takich funkcji. Z pomocą przychodzą nam węzły Czebyszewa będące pierwiastkami wielomianów Czebyszewa pierwszego rodzaju. Ich użycie zwiększa liczbę węzłów w miejscach, które są trudniejsze do przybliżenia, czyli między innymi na końcach przedziału. Skutkuje to otrzymaniem dokładniejszego przybliżenia. Poniżej znajdują się wykresy z wielomianami interpolacyjnymi stworzonymi na podstawie węzłów Czebyszewa.

Wykres funkcji $|x|$ i jej wielomianu interpolacyjnego skonstruowanego przy pomocy węzłów Czebyszewa

Wykres funkcji $\frac{1}{1+x^2}$ i jej wielomianu interpolacyjnego stworzonego przy pomocy węzłów Czebyszewa