Obliczenia Naukowe
lista 5

Mateusz Tofil

9 stycznia 2022

1 Opis problemu

Zadanie posiada swoją historyjkę, natomist rozwiązanie zadania sprowadza się do rozwiązania układu równań liniowych

$$Ax = b$$

dla macierzy $A \in R^{n \times n}$ i wektora prawych stron $b \in R^n$, $n \geq 4$. Macierz A jest rzadką, tj. mająca dużą ilośc elementów zerowych, i blokową o strukturze:

$$A = \begin{pmatrix} A_1 & C_1 & 0 & 0 & 0 & \dots & 0 \\ B_2 & A_2 & C_2 & 0 & 0 & \dots & 0 \\ 0 & B_3 & A_3 & C_3 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & B_{v-2} & A_{v-2} & C_{v-2} & 0 \\ 0 & \dots & 0 & 0 & B_{v-1} & A_{v-1} & C_{v-1} \\ 0 & \dots & 0 & 0 & 0 & B_v & A_v \end{pmatrix}$$

gdzie $v = n/l$, zakładając, że n jest podzielne przez l, gdzie $l \geq 2$ jest rozmiarem wszystkich kwadratowych macierzy wewnetrznych (bloków): A_k, B_k, C_k, gdzie $A_k \in R^{l \times l}$, $k = 1, ..., v$ są macierzami gęstymi, $B_k \in R^{l \times l}$, $k = 2, ..., v$ są postaci:

$$B_k = \begin{pmatrix} 0 & \dots & 0 & b^k_{1l-1} & b^k_{1l} \\ 0 & \dots & 0 & b^k_{2l-1} & b^k_{2l} \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & b^k_{ll-1} & b^k_{ll} \end{pmatrix}$$

a, $C_k \in R^{l \times l}$, $k = 1, ..., v-1$ są macierzami diagonalnymi:

$$C_k = \begin{pmatrix} c^k_1 & 0 & 0 & \dots & 0 \\ 0 & c^k_2 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & c^k_{l-1} & 0 \\ 0 & \dots & 0 & 0 & c^k_l \end{pmatrix}$$

2 Cel zadania

2.1 Metoda eliminacji Gaussa

Zaimplementować metodę eliminacji Gaussa, w dwóch wariantach:

1. bez wyboru elementu głównego
2. z częściowymym wyborem elementu głównego

Ponadto napisać funkcję rozwiązujące układ równań $Ax = b$ przy pomocy-odpowiedniego wariantu.

2.2 Rozkład LU

Zaimplementować funkcję wyznaczającą rozkład LU macierzy A metodą eliminacji Gaussa. podobnie jak poprzednio w dwóch wariantach oraz funkcję rozwiązującą układ równań $Ax = b$ na podstaiwe rozkładu LU.

2.3 Sparse Arrays

Aby efektywnie przechowywać macierz rzadką skorzystamy z biblioteki podstawowej w języku `Julia` Sparce Arrays. Zaletą ten biblioteki jest to, że pamięta ona tylko niezerowe elementy naszej macierzy. Zakładamy wtedy, że dostęp do elementu macierzy mamy wtedy w czasie stałym, chociaż doskonale wiemy, że tak nie jest.

3 Opis algorytmu eliminacji Gaussa

3.1 Bez wyboru elementu głównego

Metoda eliminacji Gaussa to metoda sprowadzenie układu równania do równoważnego układu z macierzą trójkątną górną, z którek łatwo wyznaczyć rozwiązanie metodą podstawienia wstecz. Przbieg algorytmu polega na zerowaniu elementów leżących pod przekątną.

Przykład. Wyznaczenie elementu a_{i1} to od i-tego wiersza odejmujemy pierwszy wiersz pomnożony przez *mnoznik* w postaci a_{i1}/a_{11}. W konsekwencji, w pierwszym kroku wyzerowane zostaną wszystkie elementy poniżej pierwszego wiersza w pierwszej kolumnie. Powtarzamy algorytm dla kolejnych kolumnach. Ważne jest, żeby pamiętać aby podczas modyfikowania i-tego wiersza macierzy równolegle modyfikować też i-ty element wektora prawych stron, aby układy pozostały tożsame.

Problem tej metody pojawia się, gdy na przekątenj znajduje się zerowy element, ponieważ obliczając nasz mnożnik dzielilibysmy przez zero, bądź element bliski zeru. Zatem rozwiązaniem tego problemu jest rozszerzenie algorytmu o częściowy wybór elementu głównego.

3.2 Z częściowym wyborem elementu głównego

Ten wariant metody eliminacji Gaussa polega na odpowiednim spermutowaniu kolejności wierszy macierzy tak, aby na przekątnej znajdowały się niezerowe elementy oraz najlepiej stosunkowo duże. Wybór odpowiedniego wiersza polega na znalezieniu największejh wartości w kolumnie spośród wierszy poniżej przekątnej.

Permutacje zapamiętujemy w `wektorze permutacji`, tak aby nie modyfikować macierzy, co mogłoby być bardziej kosztowe dla macierzy o duzych rozmairach. Implemnetacja w kodzie, to zmiana polegająca na odnoszeniu się do odpowiedniej pozycji w wektorze permutacji zamiast bezpośrednio do wiersza.

3.3 Optymalizacja

Powyższe algorytmy można zoptywalizować. Zauważnmy, że dzięki strukturze naszej macierzy (macierz trójkątna) możemy w każdej następnej interacji zawęzić przedziały iteracji pętli. Nie ma potrzeby iterować całą kolumne, bo zera nic nie zmieniają. Zatem, wystarczy że w drugiej pętli bedziemy iterować od $k+1$ do $min(n, k+l+1)$, a w trzeciej od $k+1$ do $min(n, k+l)$.

1: **procedure** GAUSS(A, n, l, b)
2: **for** $k = 1$ to $n-1$ **do**
3: **for** $i = k+1$ to $min(n, k+l+1)$ **do**
4: $multiplier \leftarrow a_{ik}/a_{kk}$
5: $a_{ik} \leftarrow 0$
6: **for** $j = k+1$ to $min(n, k+l)$ **do**
7: $a_{ij} \leftarrow a_{i_j} - multiplier * a_{kj}$
8: **end for**
9: $b_i \leftarrow b_i - multiplier * b_i$
10: **end for**
11: **end for**
12: **for** $i = n$ downto 1 **do**
13: $x_i \leftarrow \frac{b_i - \sum_{j=i+1}^{min(n,i+l)} a_{ij}x_j}{a_{ii}}$
14: **end for**
15: **return** $\bar{x}$
16: **end procedure**

Pętla główna wykonuje się $n-1$ razu, natomiast druga i trzecia najwyżej po l razy, więc jeśli l jest stała, to wyznaczenie macierzy wykonuje suę w czasie liniowym. Wyliczenie wektora wyników to również pętla, która wykonuje suę n razy, wraz z wewnętrzną pętlą, która wykonuje sue l razy. Stąd złożoność algorytmu to $\mathcal{O}(n)$.

1: **procedure** GAUSSWITHCHOICE(A, n, l, b)
2: $p \leftarrow [1, ..., n]$
3: **for** $k = 1$ to $n-1$ **do**
4: j : max(a_{p_jk}) poniżej przekątnej w kolumnie k
5: swap(p_k, p_j)
6: **for** $i = k+1$ to $min(n, k+l+1)$ **do**
7: $multiplier \leftarrow a_{p_jk}/a_{p_kk}$
8: $a_{p_ik} \leftarrow 0$
9: **for do**$j = k+1$ to $min(n, k+2*l)$
10: $a_{p_ij} \leftarrow multiplier * a_{p_kj}$
11: **end for**
12: $b_{p_i} \leftarrow b_{p_i} - multiplier * b_{p_i}$
13: **end for**
14: **return** A
15: **end for**
16: **end procedure**

W Wariancie z częściwoymym wyborem elementu głównego pętla wykonuje

się $n-1$ razy, szukanie maksymlanego elementu to pętla wykonująca l iteracji, kolejna pętla również wykonuje suę l razy, a ostatnia $2l$ razy. Wyznaczanie rozwiązania jest analogiczne do wariatu bez wyboru elementu głównego, więc złożoność całego algorytmu $\mathcal{O}(n)$.

4 Opis algorytmu - rozkład LU

4.1 Bez wyboru elementu głównego

Rozkładem LU macierzy A nazywamy przedstawienie macierzy A w postaci iloczynu $A = LU$, gdzie L jest macierzą trójkątną dolną, U jest macierzą trójkątną górną. Macierz U jest macierzą A przekształconą do postaci trójkątnej górnej (przy pomocy metody eliminacji Gaussa), a macierz L jest otrzymywana poprzez zapamiętywanie mnożników użytych do przekształceń (mnożnik służący do wyzerowania elementu a_{ij} zostanie zapisany w i-tym wierszu j-tej kolumnie macierzy L). Dodatkowym wymogiem jest to, żeby macierz L na swojej przekątnej zawierała jedynki.

Zastosowanie optymalizacyjne są analogiczne jak w przypadku metody eliminacji Gaussa. Złożoność obliczeniowa wyznaczania rozkładu LU również jest asymptoczynie taka sama jak metoy eliminacji Gausaa tj. $\mathcal{O}(n)$.

```
1:  procedure LU(A, n, l)
2:      L_ij ← 0
3:      U_ij ← A_ij
4:      for k = 1 to n − 1 do
5:          L_kk ← 1
6:          for i = k + 1 to min(n, k + l + 1) do
7:              multiplier ← a_ik/a_kk
8:              L_ik ← multiplier
9:              U_ik ← 0
10:             for j = k + 1 to min(n, k + l) do
11:                 U_ij ← U_i_j − multiplier ∗ U_kj
12:             end for
13:         end for
14:     end for
15:     L_nn ← 1
16:     return L, U
17: end procedure
```

4.2 Z częściowym wyborem elementu głównego

Tak samo jak w klasycznej metodzie eleminacji Gaussa, również przy wyznaczaniu rozkładu LU, możemy się wspomóc częściowym wyborem elementu głównego, aby uniknąc problemu z ewentualnymi zerami na przekątnej. Dzieje się to analogicznie jak w metodzie eliminacji Gaussa z częściowym wyborem, tj. dokonujemy takich permutacji, aby na przekątnej znajdował się największy element

z kolumny (ponizej przekątnej).

4.3 Rozwiązywanie układu równań przy wykorzystaniu LU

Gdy znamy już rozkład LU macierzy A to rozwiązanie układu $Ax = b$ sprowadza się do rozwiązania dwóch prostych układów równań.

$$Lz = b, Ux = z$$

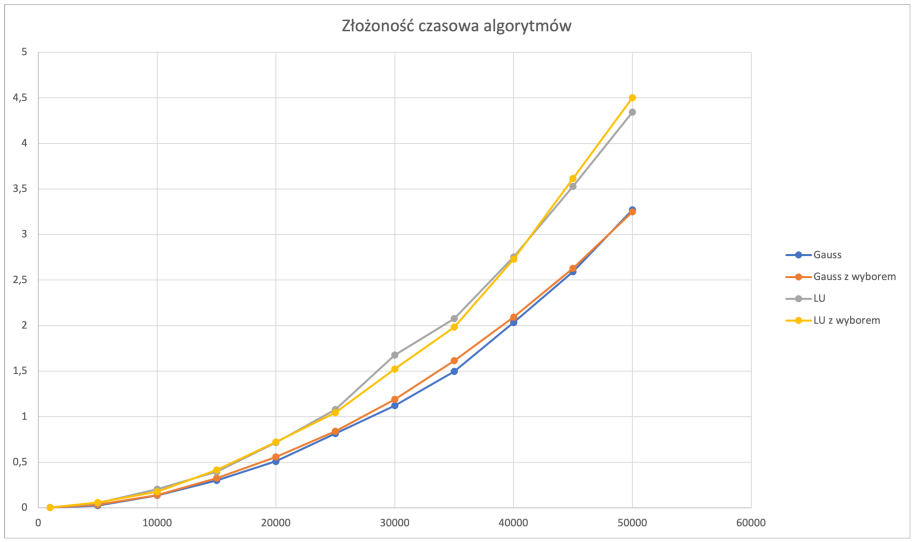
, ktore rozwiązujemy stosując podstawienia w przód dla L i podstawienie wstecz dla U. Algorytm z wariantem z częściowym wyborem elementu głównego wygląda podobnie. Różni się tylko dodaniem wektora permutacji i odnoszenia się do odpowiednich wierszy wskazywanych przez ten wektor.

Obie pętle zewnętrzne wykonują się maksymalnie n razy, a wewnętrzne najwyżej l-razy, stąd złożoność obliczeniowa tego algorytmu to $\mathcal{O}(n)$.

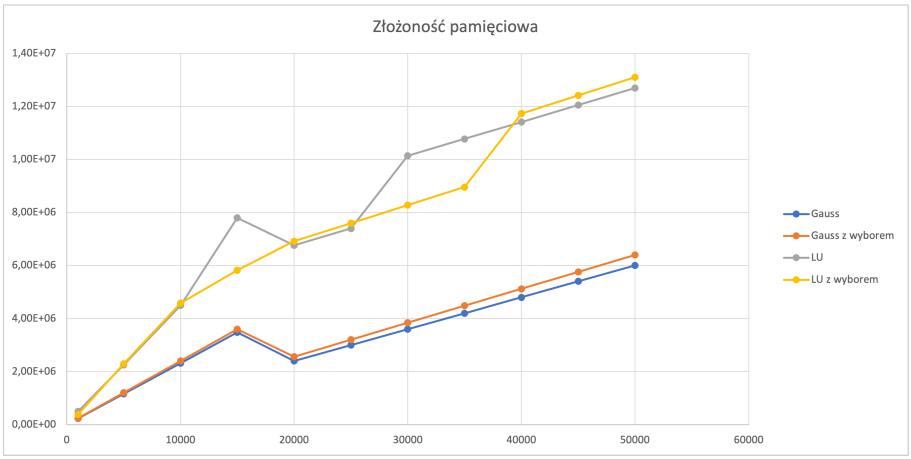
1: **procedure** SOLVELU(L, U, n, l, b)
2: **for** $i = 1$ to $n - 1$ **do**
3: $b_i \leftarrow b_i - \sum_{j=i+1}^{min(n,i+l+1)} L_{ij} * b_j$
4: **end for**
5: **for** $i = n$ downto 1 **do**
6: $x_i \leftarrow b_i - \sum_{j=i+1}^{min(n,i+l)} U_{ij} * x_j$
7: **end for**
8: **return** x
9: **end procedure**

5 Testy

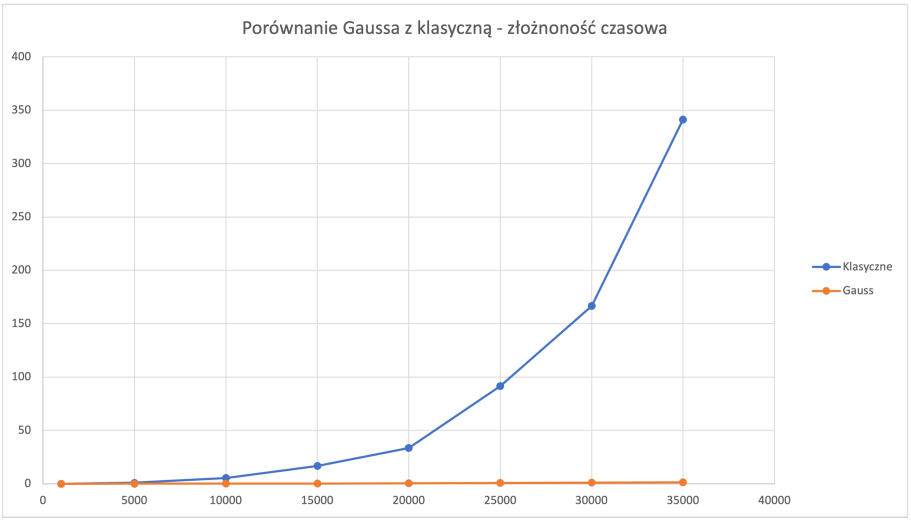
Sprawdzenie poprawności powyższych algorytmów przeprowadziłem w następujący sposób. Wygenerowałem macierz A i wyznaczając na jej podstawie wektor prawych stron b, tak aby zachodziło równanie $Ax = b$, gdzie x to wektor jednostkowy. Wtedy rozwiązując układ równań dla A i b wynikiem powinien być wektor jednostkowy.



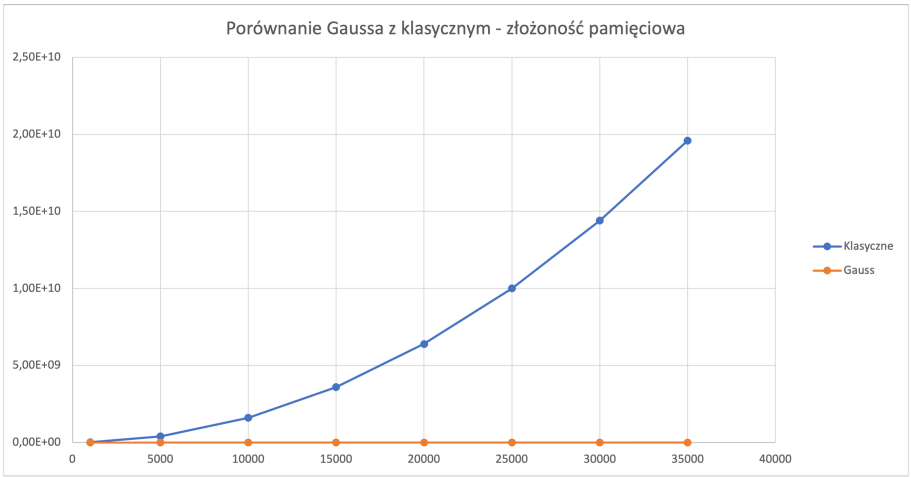
Rysunek 1: Wykres funkcji porównujący czas wykonywania zaimplementowanych algorytmów



Rysunek 2: Wykres funkcji porównujący pamięc wykonywania zaimplementowanych algorytmów



Rysunek 3: Wykres funkcji porównujący czas zaimplementowanej funkcji z funkcją z pakietu LinearAlgebra



Rysunek 4: Wykres funkcji porównujący pamięc zaimplementowanej funkcji z funkcją z pakietu LinearAlgebra

Widać, że standardowa funkcja nie jest tak bardzo efektywna jak zaimplementowane przez mnie algorytmy. Modyfikacje pozwoliły znacząco zmniejszyć zapotrzebowanie na pamieć i czas wykonywania algorytmu.

Algorytm z wyborem zabiera torchę więcej czasu i pamięci od klasycznej wersji, lecz umożliwa rozwiązywanie układów równań z zerami na przekątnej. Obliczanie rozkładu LU również wymaga więcej zasobów, ale wygenerowane

macierze mogą potem zostać użyte ponownie do rozwiązywania układów równań z innym wektorem prawych stron, bez konieczności obliczenia ich od nowa.

6 Wnioski

Zoptymalizowanie znanych algorytmów pod konkretny przykład z zadania, pozwoliło uzyskać złożoność $\mathcal{O}(n)$. Ponadto nie przechowujemy zbędnych informacji, jak zera w macierzy, oszczędzając w ten sposób pamięć.