Sprawozdanie z Listy 5
Obliczenia Naukowe

Tomasz Hałas

29 grudnia 2021

Spis treści

1 Zadanie 1.

1.1 Opis problemu

Zadanie polegało na rozwiązaniu układu równań liniowych:

$$\mathbf{Ax} = \mathbf{b},$$

gdzie macierz $\mathbf{A}$ jest rzadką macierzą kwadratową oraz dany jest wektor prawych stron b. Macierz $\mathbf{A}$ wygląda nastepująco:

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

:

Struktura owej macierzy składa się z kwadratowych macierzy wewnętrznych (bloków): $\mathbf{A_k}$, $\mathbf{B_k}$, $\mathbf{C_n}$, $\mathbf{0}$, które:

1. $\mathbf{A_k}$ jest macierzą gęstą.
2. $\mathbf{B_k}$ ma tylko dwie ostatnie kolumny niezerowe.
3. $\mathbf{C_n}$ jest macierzą diagonalną.
4. $\mathbf{0}$ jest kwadratową macierzą zerową.

Naszym zadaniem jest znaleść optymalny sposób na rozwiązanie $\mathbf{Ax} = \mathbf{b}$, zarówno algorytmicznie jak i pamięciowo.

1.2 Problem przechowywania i złożoności czasowej

Nasza macierz $\mathbf{A}$ zawiera bardzo dużo 0 i jest macierzą rzadką. Przechowywanie jej w pamięci zajmowałoby $\theta(n^2)$ miejsca. Aby ograniczyć zużycie pamięci zapamiętuje tylko te indeksy $\mathbf{A}$, które nie są 0. Język Julia umożliwia mi to za pomocą bibloteki *SparseArray*. Za jej pomocą przechowuje a_{ij}, gdzie i-te wiersze i j-te kolumne, nie zawierają 0. Umożliwia mi to łatwą i szybką iteracje po macierzy $\mathbf{A}$.

Innym problem jest złożoność czasowa. Mianowicie algorytm eliminacji Gaussa dla macierzy rzadkiej ma złożoność $\theta(n^3)$. Naszym zadaniem jest zredukowanie go do $\theta(n)$ uwzględniając, że **A** zawiera bardzo dużo 0.

1.3 Eliminacja Gaussa

Algorytm rozwiązuje układy równań liniowych oraz wyznacza rozkład LU, wykorzystując operacje elementarne(dodawanie, odejmowanie wielokrotności wierszów i kolumn), aby sprowadzić macierz do macierzy schodkowej górnej.
W tym celu iteracyjnie będziemy zerować wszystkie elementy znadującę się pod przekątną. Posłużymy się tutaj tak zwanym „mnożnikiem" $= \frac{a_{ik}}{a_{kk}}$. Będziemy go stosować do weliminowania elementu a_{ik}. W tym celu należy od i-tego elementu odjać k-ty wiersz pomnożony przez mnożnik. Niestety algorytm nie zadziała w sytuacji, gdy napotkamy zero na diagonali macierzy. Podczas wykonywania eliminacji Gaussa bedzięmy równiez jednocześnie zmieniać wektor prawych stron. Algorytm ma złożoność $\theta(n^3)$.

```
input n, (a_ij)
for k = 1 to n − 1 do
    for i = k + 1 to n do
        z ← a_ik / a_kk
        a_ik ← 0
        for j = k + 1 to n do
            a_ij ← a_ij − z a_kj
        end do
    end do
end do
output (a_ij)
```

:

Rysunek 1: Pseudokod algorytmu.

w celu rozwiązania układu równań skorzytsamy z algorytmu podstawienia wstecz.

```
input n, (a_ij), (b_i)
for i = n to 1 step −1 do
    x_i ← (b_i − Σ_{j=i+1}^{n} a_ij x_j) / a_ii
end do
output (x_i)
```

:

Rysunek 2: Pseudokod algorytmu.

1.3.1 Eliminacja Gaussa z częściowym wyborem elementu głównego

Aby zapobiec sytuacji, gdy napotkamy zero na diagonali macierzy, zastosujemy eliminacje Gaussa z częściowym wyborem elementu głównego. Polega ona na znalezieniu największego bezwględnie elementu w kolumnie i odpowiednim „przestawieniu" wierszy macierzy, tak aby znaleziony element był w odpowiednim miejscu.

1.4 Rozkład LU

Otrzymujemy go wykonując metodę eliminacji Gaussa. W jej wyniku otrzymujemy macierz dolno trójkątna (**L**) oraz górno trójkątną (**U**)

:

$$\mathbf{L} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}.$$

,gdzie rozwiązanie $Ax = b$ dzieli się na dwa etapy:

1. $Lz = b$ względem z
2. $Ux = z$ względem x

Widzimy, że dużą zaletą tej motedy jest fakt, że przy zmianie wektora prawych stron nie musimy dokonywać ponowych obliczeń dla jednego równania. Złożoność algortymiczna wynosi $\theta(n^3)$, natomiast złożoność algorytmiczna układu powyżych równań wynosi $\theta(n^2)$. **U** uzyskujemy w klasycznym alogrtymie eliminacji Gaussa. Natomiast macierz **L** uzyskujemy zapisując w niej kolejne mnożniki.
Dla rozkładu LU również istnieje metoda z częściowym wyborem elementu głównego i działa ona analogicznie jak przypadku eliminacji Gaussa.

1.5 Optymalizacja

Zważywszy na fakt, że nasza macierz jest rzadka i wstępuje w niej wiele zer, jesteśmy wstanie skrócić ilość iteracji jaką wykonuje nasz algorytm, z racji tego, że nie musimy zerować wielu elementów w kolumnach.
Iterując po każdej kolumnie nie musimy rozpatrywac wszystkich rzędów, z tego względu

że dalsza część macierzy zawiera 0. Wiemy też, że maksymalne wychylenie pod diagonalnę wynosi l (rozmiar jednej macierzy bloczkowej). Oprócz tego maskylanie musimy „wyeliminować" mniej więcej l kolejnych rzędów. A więc złożoność naszej procedury wynosi $\theta(nl^2)$ co jest równe $\theta(n)$.

Identyczne optymalizacje wykonywane są dla częściowego wyboru elementu głównego, lecz w jego przypadku musimy wybrać za każdym razem pivota, co spowoduje nie więcej niż $\theta(n)$, ale z większa dokładnością do stałej.

1.5.1 Rozkład LU

Sytuacja jest idetyczna jest w przykładzie z eliminacją Gaussa. Jedynie co jest zwiększone to liczba operacji, którą musimy wykonać, aby zrobić **LU** co spowoduje, że otrzymamy nie więcej niż $\theta(n)$, ale z większa dokładnością do stałej. Analogicznie dla rozkładu **LU** z częściowym wyborem elementu głównego.

1.6 Testy

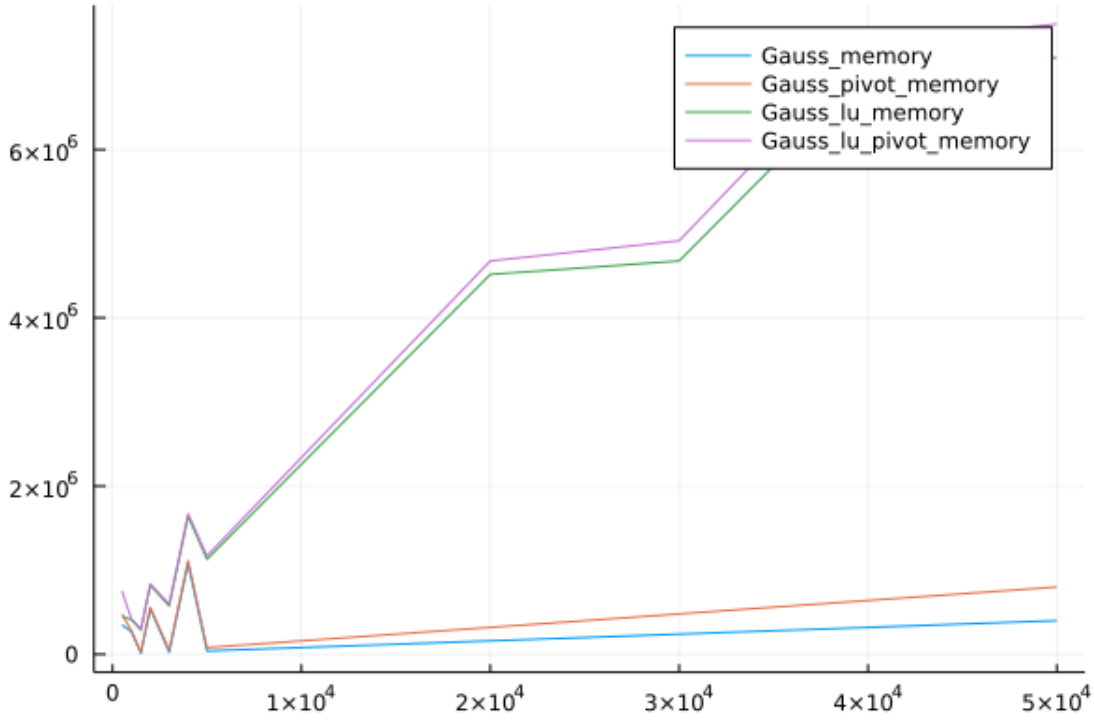
Zaimplemenotwane zostały testy (moduł Test.jl) w celu sprawdzenia poprawności moich funkcji. Wykonuje je dla danych testowych w zadaniu. Wszystkie testy zostały poprawnie napisane.

1.7 Rozwiązanie

W celu otrzymania wyników wywołuje 4 funkcje rozwiązujące równania liniowe za pomocą eliminacji Gaussa lub rozkładu **LU** otrzymanego z eliminacji Gaussa. Macierz **A** tworzę za pomocą funkcji blockmat dla poszczególnych n z cond $= 1$, a nastepnię dla której wyliczam wektor prawych stron **b**. Wykonuje ten proces 20 razy dla pewnego n i licze średnią z czasu wykonywania oraz zużytej pamięci. Na końcu tworzę wykres.

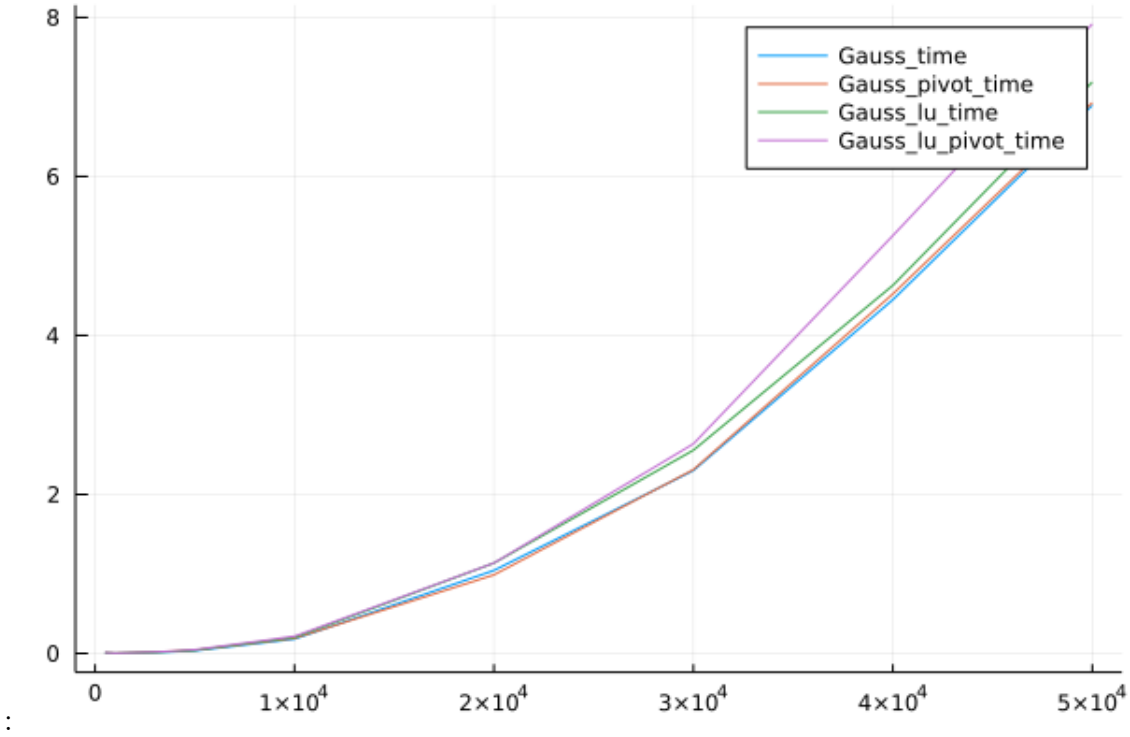
1.8 Wnioski

W swojej implementacji starałem się osiągnąć złożoność $\theta(n)$. Pod spodem znadują się wyniki mojego rozwiązania.



:

Rysunek 3: Wykres zużycia pamięci.



:

Rysunek 4: Wykres czasu.

Widzimy, że na wykresie pamięci nasza złożonosc jest liniowa. Powodem dla których rozkład LU ma bardzo duże zużycie pamięci jest fakt, że przechowuje dwie tablice U oraz L, a więc dla bardzo dużych danych mamy sporo więcej żużytej pamięci, plus dodatkowo wykorzystuje L i U do rozwiązania układy równań $LUx = b$.

Natomiast wykres czasu zupełności zaprzecza faktu, że mamy złożoność liniową. Zaczynając od interpretacji wyników widzimy, że rozkład LU zajmuje najwięcej czasu, gdyż jego „solver" jak i zwiększona ilość operacji potrzebują go wiecęj nic w zwykłej eliminacji Gaussa. Wyniki przedstawione na wykresie w zupełności spełniaja „oczekwiną kolejność" poszczególnych funkcji. Dodatkowo warto nademnić tutaj, że nie wykorzystujemy przewagi jaką daje nam fakt, że jedno równanie mamy policzone w rozkładzie **LU**. Niestety za każdym razem na nowo liczymy cały układ równań.

1.9 Powód

Powodem najprawdopobniej braku złożoności liniowej na wykresie czasu, jest fakt, że zakładamy *SparseArray* iż ma dostęp rzędu $\theta(1)$ do poszczególnych elementów, co nie zawsze jest zgodne z rzeczywistością oraz nasza złożoność liniowa zawiera stałe (tutaj

przykładowo l^2), co wypływa na wykres czasu.

Na wykresie pamięci widoczne sa również „skoki", które oznaczają, że dla coraz większych danych algorytmy są coraz efektywniej rzadzą sobie pod względem alokacji pamięci w wykonaniu zdania.

1.10 Podsumowanie

Widzimy, jak istotne jest dostosowywanie odpowiednio metod i odpowiednich zależności w celu rozwiązania danego problemu. W tym przypadku dało nam to możliwość zaoszczedzić czas i zasoby potrzebne do wykonania zadania.