Obliczenia Naukowe Laboratoria - Lista

Kajetan Plewa

5 stycznia 2026

1 Opis algorytmów

1.1 Użyta struktura

Pierwszym i najważniejszym krokiem było przygotowanie odpowiedniej struktury, pozwalającej na optymalne przechowywanie danych, bez utraty znaczących informacji.
Poniżej znajdują się dwie wykorzystywane w zadaniu struktury:

Listing 1: Struktury

```
mutable struct Block
        fields::Matrix{Float64}
end

mutable struct BlockMatrix
        n::Int
        l::Int
        v::Int
        AList::Vector{Block}
        BList::Vector{Block}
        CList::Vector{Block}
end
```

Podstawowa struktura *block* składa się z macierzy lxl - początkowo zarówno dla B jak i C miał być używany vektor - jednak dwa poniższe czynniki wymuszają użycie macierzy w obu przypadkach:

- implementacja pivota, która wiąże się z zamianą wierszów
- odejmowanie w back_substitution - dla C powoduje odejmowanie pod przekatną bloku C

Główna struktura przetrzymująca wszystkie dane - *BlockMatrix* - składa się z trzech parametrów wprowadzanych do zadania - n, v, l oraz trzech wektorów przechowujących bloki budujące macierz.

1.2 Eliminacja Gaussa

Algorytm ten został zaimplementowany na podstawie algorytmu pokazanego na wykładzie - został jednak on dostosowany do charakterestycznej struktury podanej w zadaniu.
Korzystając z oczywistych faktów iż:

$$\frac{0}{pivot} = 0 \tag{1}$$

Oraz

$$0 - 0 * a_{kj} = 0 \tag{2}$$

Można ograniczyć się do działania w obrębie bloków, gdyż zera po za nimi i tak pozostaną zerami. W funkcji *gaussianElimination* korzystam z tej konkluzji i przekazuje do dalszego działania tylko trzy potrzebne bloki, czyli:

- A_k
- B_{k+1}
- C_k

Funkcja eliminate! Tutaj odbywa się faktyczny proces eliminacji Gaussa - iterujemy po $1 : l-1$ - czyli po wartościach na przekątnej bloku A i dla każdego a_{kk} wybieramy pivot. W zależności od wartości parametru *pivotOn* jest on równy:

- $|a_{kk}| = \max_{k \leq i \leq n} |a_{ik}|$
- $|a_{kk}|$

Następnie rozpatrzone zostały dwa przypadki

- Zerowanie współczynnikow pod przekątna w bloku A. Podczas tego procesu należy pamiętać o tym, że współczynniki w bloku C po prawej stronie również zostaną zmodyfikowane.
- Zerowanie współczynnik pod przekątna znajdujących się w bloku B. Z charakterystycznej konstrukcji możemy zrobić to tylko raz gdyż współczynniki znajdują się tylko w pierwszym wierszu i ostatniej kolumnie

Ostatnim krokiem jest dostosowanie współczynników w kolumnie i wierszu l. W klasycznej eliminacji Gaussa, nie należy przejmować się ostatnim współczynnikiem lecz w tym wypadku trzeba pamiętać o ostatniej kolumnie B. Ich modyfikacja wpływa też na blok A_{k+1}

1.3 Funkcja back_substitution

Pozwala ona na wyliczenie macierzy po eliminacji gaussa oraz jest potrzebna do rozwiązania $Ux = y$ w solverze LU.
Jej działanie jest proste - iteruje ona od końca po przekątnej i oblicza wartość $x[i]$ korzystając z wzoru przedstawionego na wykładzie.

$$x_k = \frac{b_k - \sum_{k+1 \leq j \leq n} u_{kj} x_j}{u_{kk}} \tag{3}$$

1.4 Funkcja forward_substitution

Działa ona analogicznie do funkcji back_substitution, lecz zamiast iterować od n do 1, iteruje od 1 do n.
Warto podkreślić tutaj główną różnice między tymi dwoma funkcjami - obie operują na *macierzy trójkątnej*, lecz podczas gdy pierwsza z nich wirtualnie zakłada, że zera są pod przekątna, druga działa odwrotnie - zera nad przekątną.

1.5 Rozkład LU

Jest on oparty całkowicie na eliminacji Gaussa. Najbardziej istotną różnicą jest to, że w funkcji /textiteliminateLU! w przeciwieństwie do funkcji *eliminate!* nie ustawiamy współczynnika a_{ik} na 0 tylko na obliczony iloraz $\frac{a_{ik}}{pivot}$. W ten sposób pod przekątną znajduje się macierz L.
Reszta modyfikacji przeprowadzona jest tak jak w Gaussie - w ten sposób nad przekątną otrzymujemy macierz U. Jest to optymalny sposób na przechowywanie macierzy **LU** (zera w pozostałej części oryginalnej macierzy dalej pozostają zerami).

1.6 Funkcja getPivot

W rozwiązaniu zaimplementowany został częściowy wybór pivota. Jego działanie jest proste i zgodne z logiką z wykładu, z pojedynczą różnicą.

- wybierane jest interesujące nas a_{kk}

- znalezienie najwiekszego elementu w tej kolumnie (poniżej współczynnika a_{kk}), z zastrzeżeniem, że szukamy go lokalnie - czyli w kolumnie, w której się znajdujemy. Jest to niestety konieczne założenie gdyż w przeciwnym razie wiersz z B_{k+1}, który rozciąga się na $A_{k+1}iC_{k+1}$ zaburzył by strukturę macierzy. Założenie to ma jednakże niewielki wpływ na jakość wyników - rezygnujemy tylko z jednego pola na $O(l)$.
- Ostatni krok to zamiana wiersza k z wierszem o największym współczynniku a_{pk}

1.7 Funkcja solverLU

Pierwszym krokiem jest stworzenie vectora przechowującego rzeczywistą lokalizację każdego wiersza - początkowo będzie ona wyglądać w ten sposób: $p[i] = i$ lecz wraz z wybieraniem pivota w *eliminateLU!* wiersze będą swapowane. Vektor ten pozwala zachować informacje o strukturze macierzy jednocześnie pozwalając na wielokrotne użycie tego samego rozkładu LU dla różnych b. Gdyby swapowanie vektora b odbywało się w funkcji eliminateLU, rozkład musiałby być wykonywany wielokrotnie dla tej samej macierzy.

2 Złożoność czasowa i pamięciowa

2.1 Złożonośc pamięciowa

Struktura *BlockMatrix* składa się z 3 vektorów o rozmiarze $O(v)$ gdzie $v = \frac{n}{l}$. Każdy z tych wektorów ma rozmiar $l * l = l^2$ więc ostatecznie zajmowane miejsce jest rzędu:

$$3 * v * l^2 = 3 * \frac{n}{l} * l^2 = 3 * nl \approx O(nl) \tag{4}$$

Porównując do oryginlnego rozmiaru macierzy - $O(n^2)$ - widać zdecydowaną poprawę.

2.2 Złożoność czasowa

Podczas wykonywania zarówno rozdziału LU jak i eliminacji Gaussa przechoidzimy przez każdy punkt na przekątnej - puntków tych jest n. Następnie dla każdego punktu wykonujemy $O(l)$ operacji odejmowania współczynników oraz potencjalnie wybieramy pivot - za pomocą funkcji getPivot o złożoności $O(l)$. Spodziewana złożoność czasowa wynosi więc:

$$O(n) * (O(l) + O(l)) = O(n) \tag{5}$$

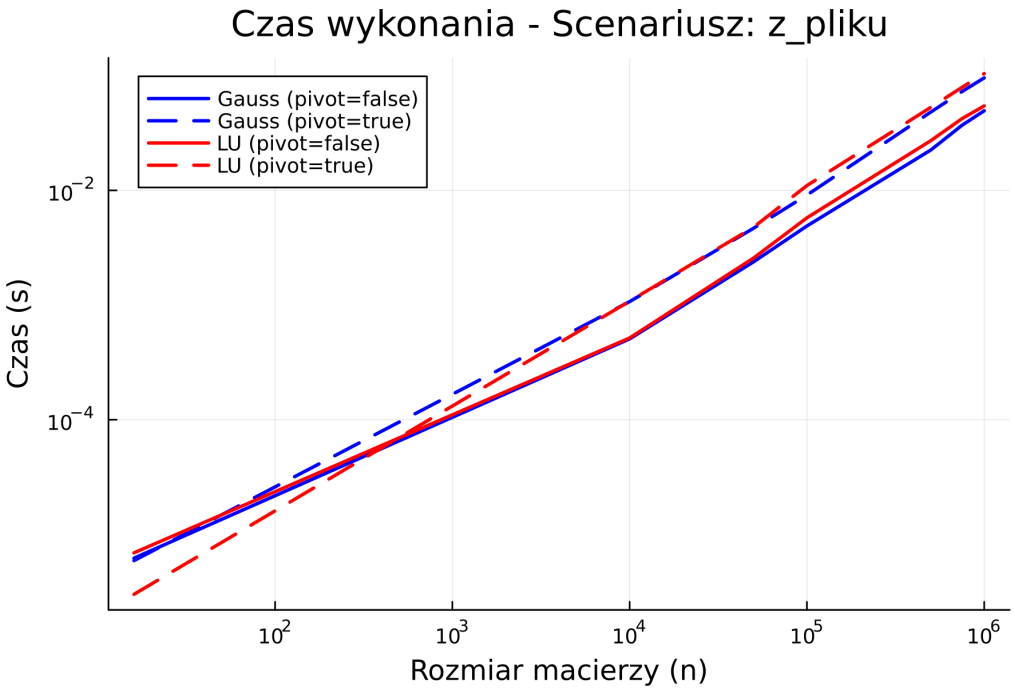
Ponieważ l jest stałą. Analiza rozwiązywania za pomocą back i forward substitution jest tożsama. Osiągnieta złożoność również wynosi $O(n)$.

3 Wyniki

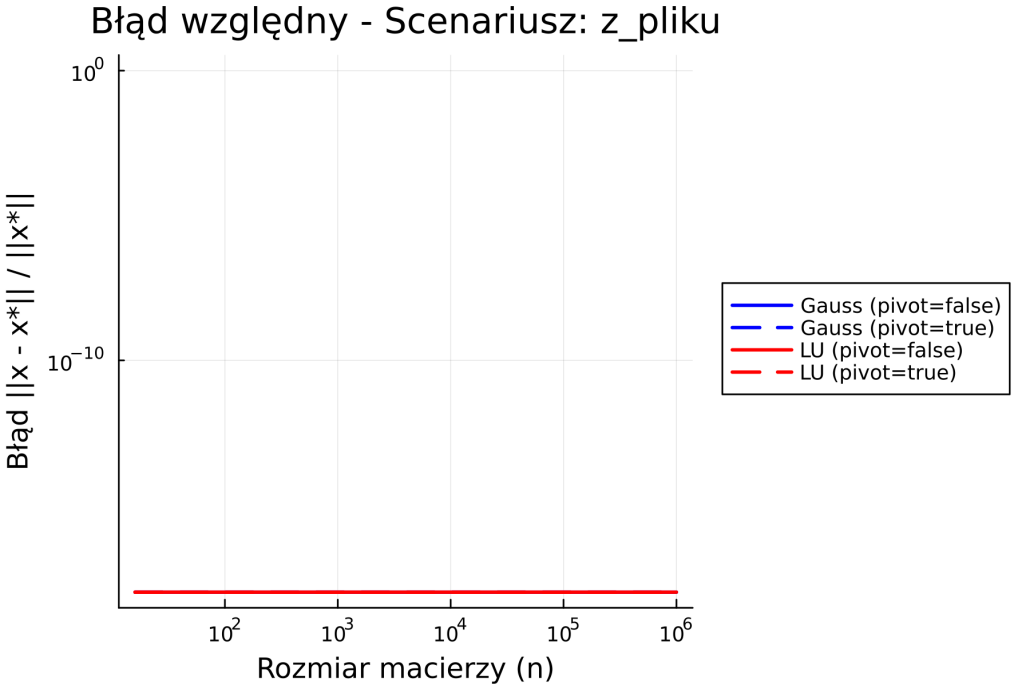
Poniżej zostaną zaprezentowane wyniki. Będą to wykresy czasu i błędu pomiaru, a n.

Interpretacja

- Wyniki pierwszego wykresu zgadzają się z tymi spodziewanymi. Dla LU i Gaussa czas rośnie liniowo. Co więcej włączenie częściowego wyboru pivota zwiększa minimalnie czas wykonywania programu. Na wykresie widać, iż wzrost ten jest mniej więcej stały co potwierdza złożoność funkcji *getPivot*.
- Drugi wykres przedstawia jak zmieniał się błąd względny wraz z wzrostem wartości n. Widać, że jest on stały - niezależnie od n. Sugeruje to poprawność zaimplementowanego rozwiązania.



Rysunek 1: Czas działania algorytmu vs n



Rysunek 2: Błąd względny vs n.

4 Wnioski

Wykonanie tego zadania pozwala dostrzec ogromny potencjał w teoretycznej analizie otrzymywanych danych. Zebranie informacji o spodziewanych danych otrzymywanych przez program pozwala przygotować infrastrukturę pod charakterystyke danych, przyspieszając działanie programu i zmiejszając zużycie potrzebnych zasobów. W czasach tak szybko rosnącej ceny RAMu umiejętność ta wydaje się kluczowa z perspektywy programisty.