Sprawozdanie Obliczenia Naukowe

Piotr Zapała

January 2024

1 Wprowadzenie do problemu

Został przed nami postawiony problem pewnej jednostki badawczej z branży chemicznej.
Firma proawdzi intensywne badania, których wynkiem są pewne modele zjawisk z dziedziny chemii kwantowej.
Rozwiązanie tych modeli, w pewnym szczególnym przypadku, sprowadza się do rozwiązania układu równań liniowych

$$\boldsymbol{Ax=b.}$$

Gdzie $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ jest macierzą współczynników, a $\boldsymbol{b} \in \mathbb{R}^n$ jest wektorem prawych stron, dla $n \geqslant 4$.
$\boldsymbol{A}$ jest macierzą rzadką, tj. mającą dużą elementów zerowych, i blokową o następującej strukturze:

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

Liczba wewnętrznych bloków $\boldsymbol{A}_k$ macierzy $\boldsymbol{A}$ wynosi $v = n/l$, zakładając, że n jest podzielne przez l.
Gdzie n jest rozmiarem macierzy $\boldsymbol{A}$, a l jest rozmiarem wewnętrznych bloków: $\boldsymbol{A}_k$, $\boldsymbol{B}_k$, $\boldsymbol{C}_k$.
Dokładniej mamy, że $\boldsymbol{A}_k \in \mathbb{R}^{l\times l}$, $k = 1, \dots, v$ jest macierzą gęstą, a $\boldsymbol{0}$ jest kwadratową macierzą zerową stopnia l.
Macierz $\boldsymbol{B}_k \in \mathbb{R}^{l\times l}$, $k = 2, \dots, v$ jest następujacej postaci:

$$\boldsymbol{B}_k = \begin{pmatrix} 0 & \dots & 0 & b_1^k \\ 0 & \dots & 0 & b_2^k \\ \vdots & & \vdots & \vdots \\ 0 & \dots & 0 & b_l^k \end{pmatrix}$$

$\boldsymbol{B}_k$ ma elementy niezerowe jedynie w ostatniej kolumnie.
Natomiast $\boldsymbol{C}_k \in \mathbb{R}^{l\times l}$, $k = 1, \dots, v-1$ jest macierzą diagonalną, czyli jedynie na przekątnej posiada elementy niezerowe.

$$\boldsymbol{C}_k = \begin{pmatrix} c_1^k & 0 & 0 & \dots & 0 \\ 0 & c_2^k & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & c_{l-1}^k & 0 \\ 0 & \dots & 0 & 0 & c_l^k \end{pmatrix}$$

Dla macierzy $\boldsymbol{A} \in \mathbb{R}^{16\times 16}$ zgodnie z $v = \frac{n}{l}$, otrzymujemy $v = 4$, co prowadzi do następujących macierzy wewnętrznych: $\boldsymbol{A}_1$, $\boldsymbol{A}_2$, $\boldsymbol{A}_3$, $\boldsymbol{A}_4$, $\boldsymbol{B}_1$, $\boldsymbol{B}_2$, $\boldsymbol{B}_3$, $\boldsymbol{C}_2$, $\boldsymbol{C}_3$, $\boldsymbol{C}_4$. Gdzie $\boldsymbol{A}_{k=1,\dots,4}, \boldsymbol{B}_{k=1,\dots,3}, \boldsymbol{C}_{k=2,\dots,4} \in \mathbb{R}^{4\times 4}$, a przykład takiej macierzy wygląda następująco:

$$
\boldsymbol{A} = \left(\begin{array}{cccc|cccc|cccc|cccc}
a^1_{1,1} & a^1_{1,2} & a^1_{1,3} & a^1_{1,4} & c^1_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a^1_{2,1} & a^1_{2,2} & a^1_{2,3} & a^1_{2,4} & 0 & c^1_2 & 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
a^1_{3,1} & a^1_{3,2} & a^1_{3,3} & a^1_{3,4} & 0 & 0 & c^1_3 & 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
a^1_{4,1} & a^1_{4,2} & a^1_{4,3} & a^1_{4,4} & 0 & 0 & 0 & c^1_4 & 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\hline
0 & 0 & 0 & b^2_1 & a^2_{1,1} & a^2_{1,2} & a^2_{1,3} & a^2_{1,4} & c^2_1 & 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & b^2_2 & a^2_{2,1} & a^2_{2,2} & a^2_{2,3} & a^2_{2,4} & 0 & c^2_2 & 0 & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & b^2_3 & a^2_{3,1} & a^2_{3,2} & a^2_{3,3} & a^2_{3,4} & 0 & 0 & c^2_3 & 0 & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & b^2_4 & a^2_{4,1} & a^2_{4,2} & a^2_{4,3} & a^2_{4,4} & 0 & 0 & 0 & c^2_4 & 0 & \vdots & \vdots & \vdots \\
\hline
\vdots & \vdots & \vdots & 0 & 0 & 0 & 0 & b^3_1 & a^3_{1,1} & a^3_{1,2} & a^3_{1,3} & a^3_{1,4} & c^3_1 & 0 & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & b^3_2 & a^3_{2,1} & a^3_{2,2} & a^3_{2,3} & a^3_{2,4} & 0 & c^3_2 & 0 & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & b^3_3 & a^3_{3,1} & a^3_{3,2} & a^3_{3,3} & a^3_{3,4} & 0 & 0 & c^3_3 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & b^3_4 & a^3_{4,1} & a^3_{4,2} & a^3_{4,3} & a^3_{4,4} & 0 & 0 & 0 & c^4_4 \\
\hline
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 0 & 0 & 0 & 0 & b^4_1 & a^4_{1,1} & a^4_{1,2} & a^4_{1,3} & a^4_{1,4} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & b^4_2 & a^4_{2,1} & a^4_{2,2} & a^4_{2,3} & a^4_{2,4} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & b^4_3 & a^4_{3,1} & a^4_{3,2} & a^4_{3,3} & a^4_{3,4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b^4_4 & a^4_{4,1} & a^4_{4,2} & a^4_{4,3} & a^4_{4,4}
\end{array}\right)
$$

Firma napotkała problem z efektywnym rozwiązywaniem takich układów równań.
W głównej mierze ze względu na wymagania czasowe oraz pamięciowe, ponieważ n jest bardzo duże.
Stąd wykluczamy pamiętanie macierzy jako tablicy $n \times n$ oraz użycie standardowych algorytmów dla macierzy gęstych.
Zatem jedynym możliwym podejściem w naszej sytuacji jest, przygotować specjalną strukturę, która będzie pamiętać jedynie elementy niezerowe naszej macierzy. Dodatkowo należy przystosować algorytmy rozwiązujące nasze układy równań tak, aby uwzględniały one specyficzną strukturę macierzy. Jak się okazuję problem rozwiązania takiego układu równań, przy założeniu, że l jest stałą można zredukować z $O(n^3)$ do $O(n)$.

2 Rozwiązanie problemu

W moim przypadku zastosowaną strukturą danych jest wbudowany w języku `Julia` słownik `SortedDict`. Jest zbudowany na podstawie struktury tablicy asocjacyjnej z haszowaniem. Jest to dynamiczna struktura danych, która pozwala na efektywne przypisywanie i dostęp do wartości za pomocą kluczy.
Ogólny zamysł polega na tym, aby używać zagnieżdżonych słowników, główny słownik zawiera klucze reprezentujące numery wierszy, natomiast wartościami są słowniki reprezentujące elementy w danym wierszu.
Dzięki zastosowaniu takiego podejścia, pamiętamy tylko elementy niezerowe, oraz znamy ich dokładne położenie w macierzy.
Przykład słownika reprezentującego powyższą macierz znajduje się poniżej.

SortedDict{Int64, SortedDict{Int64, Float64}}
(1 =>SortedDict(1 => $\mathbf{a^1_{1,1}}$, 2 => $\mathbf{a^1_{1,2}}$, 3 => $\mathbf{a^1_{1,3}}$, 4 => $\mathbf{a^1_{1,4}}$, 5 => $\mathbf{c^1_1}$),
2 =>SortedDict(1 => $\mathbf{a^1_{2,1}}$, 2 => $\mathbf{a^1_{2,2}}$, 3 => $\mathbf{a^1_{2,3}}$, 4 => $\mathbf{a^1_{2,4}}$, 6 => $\mathbf{c^1_2}$),
3 =>SortedDict(1 => $\mathbf{a^1_{3,1}}$, 2 => $\mathbf{a^1_{3,2}}$, 3 => $\mathbf{a^1_{3,3}}$, 4 => $\mathbf{a^1_{3,4}}$, 7 => $\mathbf{c^1_3}$),
4 =>SortedDict(1 => $\mathbf{a^1_{4,1}}$, 2 => $\mathbf{a^1_{4,2}}$, 3 => $\mathbf{a^1_{4,3}}$, 4 => $\mathbf{a^1_{4,4}}$, 8 => $\mathbf{c^1_4}$),
5 =>SortedDict(4 => $\mathbf{b^2_1}$, 5 => $\mathbf{a^2_{1,1}}$, 6 => $\mathbf{a^2_{1,2}}$, 7 => $\mathbf{a^2_{1,3}}$, 8 => $\mathbf{a^2_{1,4}}$, 9 => $\mathbf{c^2_1}$),
6 =>SortedDict(4 => $\mathbf{b^2_2}$, 5 => $\mathbf{a^2_{2,1}}$, 6 => $\mathbf{a^2_{2,2}}$, 7 => $\mathbf{a^2_{2,3}}$, 8 => $\mathbf{a^2_{2,4}}$, 10 => $\mathbf{c^2_2}$),
7 =>SortedDict(4 => $\mathbf{b^2_3}$, 5 => $\mathbf{a^2_{3,1}}$, 6 => $\mathbf{a^2_{3,2}}$, 7 => $\mathbf{a^2_{3,3}}$, 8 => $\mathbf{a^2_{3,4}}$, 11 => $\mathbf{c^2_3}$),
8 =>SortedDict(4 => $\mathbf{b^2_4}$, 5 => $\mathbf{a^2_{4,1}}$, 6 => $\mathbf{a^2_{4,2}}$, 7 => $\mathbf{a^2_{4,3}}$, 8 => $\mathbf{a^2_{4,4}}$, 12 => $\mathbf{c^2_4}$),
9 =>SortedDict(8 => $\mathbf{b^3_1}$, 9 => $\mathbf{a^3_{1,1}}$, 10 => $\mathbf{a^3_{1,2}}$, 11 => $\mathbf{a^3_{1,3}}$, 12 => $\mathbf{a^3_{1,4}}$, 13 => $\mathbf{c^3_1}$),
10 =>SortedDict(8 => $\mathbf{b^3_2}$, 9 => $\mathbf{a^3_{2,1}}$, 10 => $\mathbf{a^3_{2,2}}$, 11 => $\mathbf{a^3_{2,3}}$, 12 => $\mathbf{a^3_{2,4}}$, 14 => $\mathbf{c^3_2}$),
11 =>SortedDict(8 => $\mathbf{b^3_3}$, 9 => $\mathbf{a^3_{3,1}}$, 10 => $\mathbf{a^3_{3,2}}$, 11 => $\mathbf{a^3_{3,3}}$, 12 => $\mathbf{a^3_{3,4}}$, 15 => $\mathbf{c^3_3}$),
12 =>SortedDict(8 => $\mathbf{b^3_4}$, 9 => $\mathbf{a^3_{4,1}}$, 10 => $\mathbf{a^3_{4,2}}$, 11 => $\mathbf{a^3_{4,3}}$, 12 => $\mathbf{a^3_{4,4}}$, 16 => $\mathbf{c^3_4}$),
13 =>SortedDict(12 => $\mathbf{b^4_1}$, 13 => $\mathbf{a^4_{1,1}}$, 14 => $\mathbf{a^4_{1,2}}$, 15 => $\mathbf{a^4_{1,3}}$, 16 => $\mathbf{a^4_{1,4}}$),
14 =>SortedDict(12 => $\mathbf{b^4_2}$, 13 => $\mathbf{a^4_{2,1}}$, 14 => $\mathbf{a^4_{2,2}}$, 15 => $\mathbf{a^4_{2,3}}$, 16 => $\mathbf{a^4_{2,4}}$),
15 =>SortedDict(12 => $\mathbf{b^4_3}$, 13 => $\mathbf{a^4_{3,1}}$, 14 => $\mathbf{a^4_{3,2}}$, 15 => $\mathbf{a^4_{3,3}}$, 16 => $\mathbf{a^4_{3,4}}$),
16 =>SortedDict(12 => $\mathbf{b^4_4}$, 13 => $\mathbf{a^4_{4,1}}$, 14 => $\mathbf{a^4_{4,2}}$, 15 => $\mathbf{a^4_{4,3}}$, 16 => $\mathbf{a^4_{4,4}}$))

2.1 Opis algorytmów

Algorytm eliminacji Gaussa jest metodą stosowaną do rozwiązywania układów równań liniowych. Polega na przekształcaniu układu równań do postaci schodkowej (lub schodkowej zredukowanej), co ułatwia znalezienie rozwiązań. Eliminacja Gaussa może być również używana do obliczania rangi macierzy, wyznacznika oraz do dekompozycji macierzy. Wersja bez częściowego wyboru elementu głównego polega na wyborze elementów diagonalnych jako pivotów (o ile nie są zerowe) i eliminowaniu elementów poniżej pivotu w danej kolumnie. Szczegółowy przykład tej metody prezentuję poniżej.

Rozpoczynamy od pierwszego wiersza macierzy opisującej nasz układ równań Zakładając, że element a_{11} jest niezerowy (jeśli jest, to niezbędny będzie wybór elementu głównego, o którym niżej) dzielimy wszystkie elementy pierwszego wiersza wraz z elementem z wektora prawych stron przez a_{11}. W efekcie uzyskujemy:

$$\left[\begin{array}{cccccc|c} a_{11} & a_{12} & a_{13} & a_{14} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & a_{24} & \dots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & a_{34} & \dots & a_{3n} & b_3 \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots & a_{4n} & b_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \dots & a_{nn} & b_n \end{array}\right]$$

Na przekątnej części kwadratowej (zamiast a_{11}) pojawia się jedynka a pozostałe liczby z pierwszego wiersza, oznaczone tu jako w_{1j} nie zmieniają wartość. Są to już elementy docelowej macierzy, zaznaczono je więc na czerwono. Zmieniła się też wartość w kolumnie reprezentującej wektor prawych stron, oznaczona teraz jako z_1. Wszystkie elementy pierwszego wiersza (te czerwone) nie zmienią już swojej wartości.

$$\left[\begin{array}{cccccc|c} 1 & w_{12} & w_{13} & w_{14} & \dots & w_{1n} & z_1 \\ a_{21} & a_{22} & a_{23} & a_{24} & \dots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & a_{34} & \dots & a_{3n} & b_3 \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots & a_{4n} & b_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \dots & a_{nn} & b_n \end{array}\right]$$

Teraz, wykorzystując ten nowo otrzymany pierwszy wiersz będziemy "produkowali" zera w całej kolumnie poniżej jedynki. Aby uzyskać zero bezpośrednio pod jedynką (w drugim wierszu, pierwszej kolumnie) odejmujemy od całego drugiego wiersza wiersz pierwszy, pomnożony przez a_{21}. W efekcie pod jedynką pojawi się zero a pozostałe liczby w drugim wierszu (y_{xx} i y_x) zmienią swoje wartości. Nie będą to jednak wartości docelowe tylko wyniki pośrednie. Dopiero "produkowanie" jedynki zamiast elementu y_{22} przekształci wiersz drugi do postaci docelowej.

$$\left[\begin{array}{cccccc|c} 1 & w_{12} & w_{13} & w_{14} & \dots & w_{1n} & z_1 \\ 0 & y_{22} & y_{23} & y_{24} & \dots & y_{2n} & y_2 \\ a_{31} & a_{32} & a_{33} & a_{34} & \dots & a_{3n} & b_3 \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots & a_{4n} & b_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \dots & a_{nn} & b_n \end{array}\right]$$

Podobnie postępujemy z wierszem trzecim. Odejmujemy od niego pierwszy wiersz pomnożony przez a_{31}. Pojawia się kolejne zero w kolumnie pierwszej i kolejne tymczasowe współczynniki y_{ij} na pozostałych miejscach w wierszu trzecim.

$$\left[\begin{array}{cccccc|c} 1 & w_{12} & w_{13} & w_{14} & \dots & w_{1n} & z_1 \\ 0 & y_{22} & y_{23} & y_{24} & \dots & y_{2n} & y_2 \\ 0 & y_{32} & y_{33} & y_{34} & \dots & y_{3n} & y_3 \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots & a_{4n} & b_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \dots & a_{nn} & b_n \end{array}\right]$$

Dalej postępujemy tak samo z wierszem czwartym, piątym itd. aż do wiersza n-tego. Kolumna pierwsza składa się z jedynki na przekątnej (w miejscu elementu a_{11}) i z samych zer w pozostałych wierszach. W ten sposób pierwszy wiersz i pierwsza kolumna otrzymały już postać docelową i w dalszych rachunkach nie będą już zmieniane. Zakończył się też pierwszy "krok" metody eliminacji Gaussa.

$$\left[\begin{array}{cccccc|c} 1 & w_{12} & w_{13} & w_{14} & \dots & w_{1n} & z_1 \\ 0 & y_{22} & y_{23} & y_{24} & \dots & y_{2n} & y_2 \\ 0 & y_{32} & y_{33} & y_{34} & \dots & y_{3n} & y_3 \\ 0 & y_{42} & y_{43} & y_{44} & \dots & y_{4n} & y_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & y_{n2} & y_{n3} & y_{n4} & \dots & y_{nn} & y_n \end{array}\right]$$

W następnym "kroku" przesuwamy się w dół i w prawo wzdłuż przekątnej, na miejsce elementu a_{22}, który teraz ma wartość y_{22}. Dzielimy cały wiersz przez y_{22} i "produkujemy" kolejną jedynkę na przekątnej oraz docelowe wartości współczynników wij w prawo od niej. Oczywiście optymalnie napisany program nie będzie dzielił elementów zerowych (w lewo od przekątnej), gdyż nie zmienia to ich wartości i jest zwykłym marnowaniem czasu

$$\left[\begin{array}{cccccc|c} 1 & w_{12} & w_{13} & w_{14} & \dots & w_{1n} & z_1 \\ 0 & 1 & w_{23} & w_{24} & \dots & w_{2n} & z_2 \\ 0 & y_{32} & y_{33} & y_{34} & \dots & y_{3n} & y_3 \\ 0 & y_{42} & y_{43} & y_{44} & \dots & y_{4n} & y_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & y_{n2} & y_{n3} & y_{n4} & \dots & y_{nn} & y_n \end{array}\right]$$

Dalej, jak w poprzednik "kroku", "produkujemy" zero w kolumnie drugiej wierszu trzecim, odejmując od wiersza trzeciego wiersz drugi, cały pomnożony przez y_{32}.

$$\left[\begin{array}{cccccc|c} 1 & w_{12} & w_{13} & w_{14} & \dots & w_{1n} & z_1 \\ 0 & 1 & w_{23} & w_{24} & \dots & w_{2n} & z_2 \\ 0 & 0 & r_{33} & r_{34} & \dots & r_{3n} & r_3 \\ 0 & y_{42} & y_{43} & y_{44} & \dots & y_{4n} & y_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & y_{n2} & y_{n3} & y_{n4} & \dots & y_{nn} & y_n \end{array}\right]$$

Postępujemy tak dalej z wierszem czwartym, piątym itd., otrzymując zera w całej kolumnie drugiej pod przekątną.

$$\left[\begin{array}{cccccc|c} 1 & w_{12} & w_{13} & w_{14} & \dots & w_{1n} & z_1 \\ 0 & 1 & w_{23} & w_{24} & \dots & w_{2n} & z_2 \\ 0 & 0 & r_{33} & r_{34} & \dots & r_{3n} & r_3 \\ 0 & 0 & r_{43} & r_{44} & \dots & r_{4n} & r_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & r_{n3} & r_{n4} & \dots & r_{nn} & r_n \end{array}\right]$$

Kończymy w ten sposób drugi "krok" metody eliminacji Gaussa a w kolejnych "krokach", postępując tak samo, "produkujemy" jedynki na przekątnej części kwadratowej i zera poniżej. W końcu otrzymujemy docelową postać macierzy. Opisany przez tę macierz układ daje się rozwiązać wprost "od dołu". Z ostatniego równania mamy $x_n = z_n$, potem wyliczamy x_{n-1} z wiersza poprzedniego itd.

$$\left[\begin{array}{cccccc|c} 1 & w_{12} & w_{13} & w_{14} & \dots & w_{1n} & z_1 \\ 0 & 1 & w_{23} & w_{24} & \dots & w_{2n} & z_2 \\ 0 & 0 & 1 & w_{34} & \dots & w_{3n} & z_3 \\ 0 & 0 & 0 & 1 & \dots & w_{4n} & z_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & z_n \end{array}\right]$$

1. Funkcja divideRow

Funkcja `divideRow` skaluje dany wiersz row macierzy **matrix_dict** i odpowiadający mu element wektora vector przez wartość **pivot**. Jest to krok normalizacji w algorytmie eliminacji Gaussa, gdzie wiersz jest dzielony przez element diagonalny, tak aby ten element miał wartość 1.

Szczegóły działania:

-Iteruje po każdej kolumnie **col** aktualnego wiersza **row** w **matrix_dict**.
-Dzieli każdy element w wierszu przez wartość **pivot**.
-Dzieli również odpowiadający element w **vector** przez **pivot**.

Dla każdego wiersza, ta funkcja dzieli $l + 1$ elementów przez pivot.
Operacje te mają złożoność $O(l)$, ponieważ wykonujemy $l + 1$ operacji dzielenia, gdzie koszt każdego wynosi $O(1)$.
Zatem złożoność czasowa funckji `divideRow` wynosi $O(l)$

```
Function divideRow(matrix_dict, vector, row, pivot)
    For each column col and value in matrix_dict[row]:
        matrix_dict[row][col] /= pivot
    End For
    vector[row] /= pivot
End Function
```

2. Funkcja subtractMultipleOfRow

Funkcja `subtractMultipleOfRow` wykonuje operację wiersz odejmowany od wiersza macierzy **matrix_dict**, co jest kluczowym krokiem w algorytmie eliminacji Gaussa. Odejmuje wielokrotność wiersza **source_row** od **target_row**, skalowaną przez **factor**.

Szczegóły działania:

-Iteruje po każdej kolumnie **col** w wierszu **source_row**.
-Odejmuje od wiersza **target_row** wartość **factor** pomnożoną przez odpowiadający element z **source_row**.
-Usuwa kolumnę z **target_row** w **matrix_dict**, jeśli wynikowy element jest zerowy.
-Aktualizuje również element wektora **vector** dla **target_row**, odejmując od niego pomnożony przez **factor** element z **source_row**.

Ta funkcja wykonuje co najwyżej $l + 1$ operacji odejmowania
i potencjalnego usuwania dla każdego elementu wiersza źródłowego.
Odejmowanie ma złożoność $O(l)$ ale usunięcie elementu ze słownika (w najgorszym przypadku) może mieć złożoność $O(l)$,
zatem ogólna złożoność czasowa to $O(l^2)$ dla każdego wywołania funkcji.

```
Function subtractMultipleOfRow(matrix_dict, vector, source_row, target_row, factor)
    For each column col and value in matrix_dict[source_row]:
        If col exists in matrix_dict[target_row]:
            matrix_dict[target_row][col] -= value * factor
            If matrix_dict[target_row][col] == 0:
                Delete matrix_dict[target_row][col]
            End If
        Else:
            matrix_dict[target_row][col] = -value * factor
        End If
    End For
    vector[target_row] -= vector[source_row] * factor
End Function
```

3. Funkcja gaussEliminationFirstLRowsWithoutPartialPivoting

Funkcja ta przeprowadza eliminację Gaussa na pierwszych **l** wierszach macierzy **matrix_dict**, nie stosując częściowego wyboru elementu głównego.

Szczegóły działania:

-Iteruje po pierwszych **l** wierszach macierzy.
-Dla każdego wiersza **row**, znajduje **pivot** (element na przekątnej) i, jeśli nie jest on zerowy, normalizuje wiersz za pomocą `divideRow`.
-Następnie, dla każdego wiersza poniżej **row**, używa `subtractMultipleOfRow` do wyzerowania elementów poniżej **pivota**.

Dla każdego z pierwszych l wierszy wykonujemy `divideRow` raz i `subtractMultipleOfRow` l razy. Daje to $O(l^2 + l^3)$ operacji, zatem ogólna złożoność czasowa to $O(l^3)$.

```
Function gaussEliminationFirstLRowsWithoutPartialPivoting(matrix_dict, vector, n, l)
    For each row from 1 to l:
        pivot = matrix_dict[row][row]
        If pivot != 0:
            divideRow(matrix_dict, vector, row, pivot)
            For each target_row from (row+1) to l:
                If col exists in matrix_dict[target_row]:
                    factor = matrix_dict[target_row][row]
                    subtractMultipleOfRow(matrix_dict, vector, row, target_row, factor)
                End If
            End For
        End If
    End For
End Function
```

3. Funkcja gaussEliminationLastLRowsWithoutPartialPivoting

Funkcja ta przeprowadza eliminację Gaussa na ostatnich **l** wierszach macierzy **matrix_dict**, nie stosując częściowego wyboru elementu głównego.

Szczegóły działania:

-Iteruje po ostatnich **l** wierszach macierzy.
-Dla każdego wiersza **row**, znajduje **pivot** (element na przekątnej) i, jeśli nie jest on zerowy, normalizuje wiersz za pomocą `divideRow`.
-Następnie, dla każdego wiersza poniżej **row**, używa `subtractMultipleOfRow` do wyzerowania elementów poniżej **pivota**.

Dla każdego z ostatnich l wierszy wykonujemy `divideRow` raz i `subtractMultipleOfRow` l razy. Daje to $O(l^2 + l^3)$ operacji, zatem ogólna złożoność czasowa to $O(l^3)$.

```
Function gaussEliminationLastLRowsWithoutPartialPivoting(matrix_dict, vector, n, l)
    For each row from (n-l+1) to n:
        pivot = matrix_dict[row][row]
        If pivot != 0:
            divideRow(matrix_dict, vector, row, pivot)
            For each target_row from (row+1) to n:
                If col exists in matrix_dict[target_row]:
                    factor = matrix_dict[target_row][row]
                    subtractMultipleOfRow(matrix_dict, vector, row, target_row, factor)
                End If
            End For
        End If
    End For
End Function
```

3. Funkcja gaussEliminationWithoutPartialPivoting

Funkcja ta wykonuje eliminację Gaussa bez częściowego wyboru elementu głównego na całej macierzy.

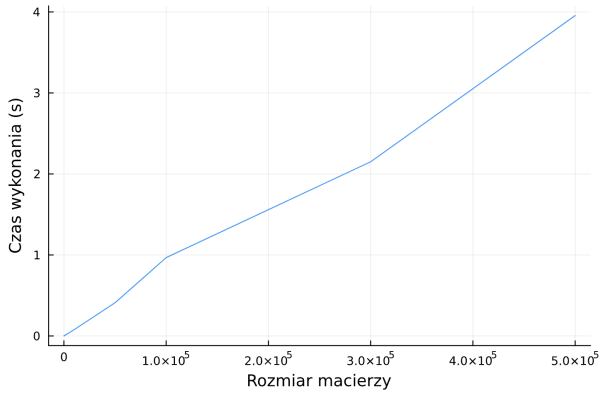
Szczegóły działania:

-Najpierw wywołuje `gaussEliminationFirstLRowsWithoutPartialPivoting` dla pierwszych **l** wierszy.
-Następnie, iteruje przez kolejne wiersze macierzy od **l** do **n-l** i przeprowadza normalizację i eliminację dla każdego wiersza, używając `divideRow` i `subtractMultipleOfRow`.
-Po przeprowadzeniu operacji na środkowej części macierzy, wywoływana jest `gaussEliminationLastLRowsWithoutPartialPivoting` która wykonuje operacje na ostatnich **l** wierszach.
-Zwraca zmodyfikowaną macierz **matrix_dict**.

Wywołujemy `gaussEliminationFirstLRowsWithoutPartialPivoting`, która ma $O(l^3)$, a następnie dla każdego z $O(n-2l)$ wierszy wykonujemy `divideRow` i `subtractMultipleOfRow`. Dla `divideRow` mamy $O((n-2l)l)$ a dla `subtractMultipleOfRow` mamy $O((n-2l)l^2)$, co daje $O(nl^2)$ dla tej części. Następnie wywołujemy `gaussEliminationLastLRowsWithoutPartialPivoting`, która ma $O(l^3)$. W rezultacie mamy $O(l^3 + nl^2)$ co dla dużych n daje nam $O(nl^2)$.

```
Function gaussEliminationWithoutPartialPivoting(matrix_dict, vector, n, l)
    Call gaussEliminationFirstLRowsWithoutPartialPivoting(matrix_dict, vector, n, l)
    For each row from l to (n-l):
        pivot = matrix_dict[row][row]
        If pivot != 0:
            divideRow(matrix_dict, vector, row, pivot)
            For each target_row from (row+1) to min(n, row+l):
                If col exists in matrix_dict[target_row]:
                    factor = matrix_dict[target_row][row]
                    subtractMultipleOfRow(matrix_dict, vector, row, target_row, factor)
                End If
            End For
        End If
    End For
    Call gaussEliminationLastLRowsWithoutPartialPivoting(matrix_dict, vector, n, l)
    Return matrix_dict
End Function
```

W przypadku gdy l jest stałą, złożoność czasowa $O(nl^2)$ redukuję się do $O(n)$, stąd otrzymujemy algorytm działający w czasie liniowym.
Jeśli chodzi o złożoność pamięciową to wynosi ona $O(nl) = O(n)$, gdyż mamy po $l+1$ lub $l+2$ wartości w n wierszach, wszystkie operacje wykonuję w miejscu na jednej macierzy, a elementy zerowe po eliminacji kolejnych wartości w kolumnach są z niej usuwane.
Poniżej prezentuję empiryczny test złożoności czasowej mojego algorytmu.



Rysunek 1: Wykres czasu wykonywania eliminacjii gaussa w zależności od rozmiaru macierzy

Eliminacja Gaussa z częściowym wyborem elementu głównego to ulepszona wersja klasycznego algorytmu eliminacji Gaussa, który służy do rozwiązywania układów równań liniowych.

Częściowy wybór elementu głównego (ang. partial pivoting) polega na wyborze największego (pod względem wartości bezwzględnej) elementu w kolumnie jako pivotu, aby zminimalizować błędy zaokrągleń w obliczeniach numerycznych. Stąd jedyną modyfikacją względem wersji bez wyboru głównego jest taka, że zanim wybierzemy pivota, wywołujemy funkcję `partialPivot`, która zamienia nam wiersze w naszej macierzy oraz elementy w wektorze prawych stron.

Jeśli chodzi o złożoność czasową i pamięciową funkcji `gaussEliminationWithPartialPivoting` to pozostają one niezmienne względem wersji bez wyboru elementu głównego.

Pseudokod omawianego programu prezentuję poniżej.

```
Function partialPivot(matrix_dict, vector, n, row)
    max_value := |matrix_dict[row][row]|
    max_row := row
    For i from row + 1 to n:
        If row exists in matrix_dict[i] and |matrix_dict[i][row]| > max_value:
            max_value := |matrix_dict[i][row]|
            max_row := i
        end If
    end For
    If max_row != row:
        swap matrix_dict[row] with matrix_dict[max_row]
        swap vector[row] with vector[max_row]
    end If
End Function
```

Algorytm rozkładu LU bez wyboru elementu głównego wykorzystuje metodę eliminacji Gaussa do przekształcenia macierzy kwadratowej A w dwie macierze trójkątne: dolną L i górną U, gdzie L jest macierzą dolnotrójkątną z jedynkami na głównej przekątnej, a U jest macierzą górnotrójkątną. Poniżej przedstawiam pseudokod mojego algorytmu i jego szczegółowy opis.

```
Function LUdecompositionWithoutPivoting(matrix_dict, n)
    L := Dict{Int, Dict{Int, Float64}}()
    For i from 1 to n:
        L[i] := Dict{Int, Float64}(i => 1.0)
    end For
    Call fillMatrixesWithoutPivoting(L, matrix_dict, 1, l, 0)
    Call fillMatrixesWithoutPivoting(L, matrix_dict, l, n-l, l)
    Call fillMatrixesWithoutPivoting(L, matrix_dict, n-l+1, n, 0)
    return L, matrix_dict
end


Function fillMatrixesWithoutPivoting1(L, U, start, stop)
    For row from start to stop:
        pivot := U[row][row]
        For target_row from (row+1) to stop:
            If row exists in U[target_row]:
                L[target_row][row] := U[target_row][row] / pivot
                For each column col and value in U[row]:
                    If col >= row:
                        If col exists in U[target_row]:
                            U[target_row][col] -= L[target_row][row] * value
                            If U[target_row][col] == 0.0:
                                Delete U[target_row][col]
                            end If
                        else
                            U[target_row][col] := -L[target_row][row] * value
                            If U[target_row][col] == 0.0
                                Delete U[target_row][col]
                            end If
                        end If
                    end For
            end If
        end For
    end For
end


Function fillMatrixesWithoutPivoting2(L, U, start, stop, l)
    For row from start to stop:
        pivot := U[row][row], i := 0
        For target_row from (row+1) to row+l-i:
            If row exists in U[target_row]:
                L[target_row][row] := U[target_row][row] / pivot
                For each column col and value in U[row]:
                    If col >= row:
                        If col exists in U[target_row]:
                            U[target_row][col] -= L[target_row][row] * value
                            If U[target_row][col] == 0.0:  Delete U[target_row][col] end If
                        else
                            U[target_row][col] := -L[target_row][row] * value
                            If U[target_row][col] == 0.0 Delete U[target_row][col] end If
                        end If
                    end For
            end If
        end For
        i = i + 1
    end For
end
```

Inicjalizacja:

Algorytm rozpoczyna od inicjalizacja słownika L. Ten słowniki będzie przechowywać elementy macierzy dolnotrójkątnej L. Dla każdego wiersza i od 1 do n (gdzie n to rozmiar macierzy): Macierz L jest inicjalizowana tak, że na głównej przekątnej ma jedynki, a pozostałe elementy są zerami.

Proces Tworzenia Macierzy U i L:

Funkcja `fillMatrixesWithoutPivoting1`, gdzie **start=1**, **stop=l** i **k=0**
1.Dla każdego wiersza **row** od **start=1** do **stop=l**:
2.Wartość **pivotu** jest określana jako element na przekątnej **U[row][row]**.
3.Dla każdego wiersza **target_row** poniżej **row** (od **row+1** do **stop**):
4.Jeśli istnieje element w **U[target_row]** w bieżącej kolumnie row:
5.Element w macierzy L dla **target_row** i **row** jest ustawiany jako **U[target_row][row] / pivot**.
6.Wiersz **target_row** w macierzy U jest aktualizowany:
7.Dla każdej kolumny **col** począwszy od **row** do końca wiersza:
8.Jeśli element istnieje w **U[target_row]**, odejmuje się od niego mnożnik (wartość w **L[target_row][row]**) pomnożony przez odpowiednią wartość z wiersza row macierzy U.
9.Jeśli wynikowy element w **U[target_row][col]** jest równy 0, usuwa się go, aby utrzymać strukturę rzadką macierzy.
Następnie ponownie wywołujemy funkcję `fillMatrixesWithoutPivoting1` jeszcze raz dla **start=n-l+1**, **stop=n**.
Oraz wywołujemy `fillMatrixesWithoutPivoting2` raz dla **start=l**, **stop=n-l**.

Zwracanie Wyniku

Po zakończeniu procesu, algorytm zwraca dwie macierze: L i U, które są dolnotrójkątną i górnotrójkątną częścią rozkładu LU macierzy wejściowej.

Złożoność obliczeniowa funkcji fillMatrixesWithoutPivoting1 oraz fillMatrixesWithoutPivoting2

Funkcja `fillMatrixesWithoutPivoting1` jest wywoływana dla różnych zakresów wierszy, zdefiniowanych przez **start** i **stop**.
Dla każdego wiersza **row** od **start** do **stop**:
W każdym wierszu **target_row** od **row+1** do **stop**, aktualizuje elementy w macierzy L i U.
Każda taka aktualizacja wymaga przejścia przez średnio l elementów w wierszu **U[row]** i potencjalnego wykonania operacji odejmowania i usuwania, co daje złożoność $O(l^2)$ dla każdej iteracji wewnętrznej pętli.
Zatem całkowita złożoność dla `fillMatrixesWithoutPivoting1` dla jednego zakresu to $O((stop - start))l^2$.
Funkcja **fillMatrixesWithoutPivoting2** działa podobnie, różnica jest taka, że w pętli **For target_row from (row+1) to stop:** zamiast **stop** mamy **row+l-1**.
Jest to pewna optymalizacja ze względu na specyficzną postać macierzy, gdyż dla danego wiersza chcę produkować elementy zerowe w co najwyżej **l** kolumnach poniżej **pivota**.

Złożoność obliczeniowa funkcji LUdecompositionWithoutPivoting

Inicjalizacja L ma złożoność $O(n)$.

`fillMatrixesWithoutPivoting1(L, matrix_dict, 1, l)` ma złożoność $O(l^3)$, ponieważ **stop - start + k** to l.
`fillMatrixesWithoutPivoting2(L, matrix_dict, l, n-l, l)` ma złożoność $O((n-2l)l^2) = O(nl^2)$.
`fillMatrixesWithoutPivoting1(L, matrix_dict, n-l+1, n)` ma ponownie złożoność $O(l^3)$.

Zatem całkowita złożoność czasowa funkcji wyznaczającej rozkład
LU to $O(nl) + O(l^3) + O((n-2l)l^2) + O(l^3) = O(nl^2)$, co dla dużych n i stałego l daję $O(n)$.
Natomiast złożoność pamięciowa wynosi $O(nl) = O(n)$

Jeżeli chodzi o funkcję `LUdecompositionWithPivoting`, to jedyną zmianą jest użycie funkcji `partialPivot` na **matrix_dict** przed wyborem **pivota**, złożoność czasowa i pamięciowa pozostaję bez zmian.

Do rozwiązania macierzy powstającej z eliminacjii gaussa, używam funkcji `backwardSubstitution`, która rozwiązuje układ równań liniowych z macierzą górnotrójkątną. Jest ona wykorzystywana w przypadku, gdy macierz współczynników układu równań została już przekształcona do postaci górnotrójkątnej.

Tworzony jest wektor **x** o długości **n**, wypełniony zerami. Ten wektor będzie przechowywał rozwiązanie układu równań.
Funkcja iteruje przez wiersze macierzy **U** od ostatniego wiersza do pierwszego (indeks **i** zmienia się od **n** do **1**).
Dla każdego wiersza **i**:
Rozpoczyna się od przypisania **x[i]** wartości odpowiadającego elementu wektora **y** (prawa strona układu równań).
Następnie, od **x[i]** odejmowana jest suma iloczynów każdego elementu w wierszu **U[i]** i odpowiadających im wartości w wektorze **x** (dla kolumn **j** większych niż **i**).
Wartość **x[i]** jest następnie dzielona przez element diagonalny macierzy **U** w tym wierszu **(U[i][i])**, co daje wartość zmiennej w tym wierszu układu równań.

W celu rozwiązania **LUx=b**, używam funkcji `solveLU`, która najpierw wykorzystuje `forwardSubstitution` na macierzy **L** i wektorze **b**, by znaleźć wektor **y**.
Następnie używa `backwardSubstitution` na macierzy **U** i wektorze **y**, by znaleźć rozwiązanie **x**.
Zwraca rozwiązanie układu równań, wektor **x**.
Funkcja `forwardSubstitution` rozwiązuje układ równań liniowych z macierzą dolnotrójkątną.

Inicjalizuje wektor **y** zerami o długości **n**. Iteruje przez wszystkie wiersze macierzy **L** od pierwszego do ostatniego.
Dla każdego wiersza **i**:
Oblicza sumę iloczynów wartości z macierzy **L[i]** i odpowiadających im wartości z wektora **y**.
Wartość **y[i]** jest równa **b[i]** minus obliczona suma.

Pseudokody podanych funkcji prezentuję poniżej, ich złożoność czasowa ze względu na rzadkość macierzy wynosi $O(nl) = O(n)$, gdyż l jest stałą.

```
Function forwardSubstitution(L, b, n)
    Initialize y as a zero vector of length n
    For i from 1 to n:
        Initialize sum as 0.0
        For each j in keys of L[i]:
            sum += L[i][j] * y[j]
        End For
        y[i] := b[i] - sum
    End For
    Return y
End Function


Function backwardSubstitution(U, y, n)
    Initialize x as a zero vector of length n
    For i from n down to 1:
        Initialize sum as 0.0
        For each j in keys of U[i]:kl
            sum += U[i][j] * x[j]
        End For
        x[i] := (y[i] - sum) / U[i][i]
    End For
    Return x
End Function


Function solveLU(L, U, b, n)
    y := forwardSubstitution(L, b, n)
    x := backwardSubstitution(U, y, n)
Return x
End Function
```

3 Wyniki

Metoda	Błąd 1	Błąd 2	Błąd 3	Błąd 4	Błąd 5	Błąd 6
Gauss Without Pivoting	4.54e-15	5.74e-14	1.07e-13	5.10e-14	3.15e-13	1.88e-13
Gauss With Pivoting	2.90e-16	3.42e-16	4.04e-16	2.93e-16	3.98e-16	3.97e-16

Tabela 1: Błędy dla różnych metod eliminacji Gaussa

Metoda	Błąd 1	Błąd 2	Błąd 3	Błąd 4	Błąd 5	Błąd 6
LU Without Pivoting	7.47e-15	5.62e-14	1.63e-13	5.31e-14	3.50e-13	1.09e-13
LU With Pivoting	0.05868	0.43561	0.35546	0.45519	0.35447	0.35342

Tabela 2: Błędy dla różnych metod rozkładu LU

W testach użyłem normy euklidesowej, w celu wyznaczania błędów względnych poszczególnych metod rozwiązywania układu równań.

4 Wnioski

Wyniki dla różnych metod eliminacji Gaussa i rozkładu LU, zarówno z wyborem elementu głównego (pivoting), jak i bez niego (without pivoting), prowadzą do następujących wniosków:

1. **Niskie Błędy w Eliminacji Gaussa z Wyborem Elementu Głównego:** Metoda eliminacji Gaussa z wyborem elementu głównego wykazuje znacznie niższe błędy w porównaniu do metody bez wyboru elementu głównego. Jest to wskaźnik większej stabilności numerycznej i dokładności w obliczeniach przy zastosowaniu wyboru elementu głównego.

2. **Porównanie Metod Rozkładu LU:** Metoda LU bez wyboru elementu głównego prezentuje wyższe błędy niż metoda eliminacji Gaussa z wyborem elementu głównego. Rozkład LU z wyborem elementu głównego, pokazując znacznie wyższe błędy, może wskazywać na mniejszą efektywność w danym przypadku, prawdopodobnie związane z charakterystyką użytej macierzy lub implementacją metody.

3. **Znaczenie Stabilności Numerycznej:** Wyniki potwierdzają, jak istotna jest stabilność numeryczna w obliczeniach matematycznych. Metody z wyborem elementu głównego zazwyczaj zapewniają większą stabilność, co przekłada się na dokładniejsze obliczenia.

4. **Wybór Odpowiedniej Metody:** Wybierając metodę rozwiązania układu równań, należy uwzględnić zarówno wymagania dokładności, jak i specyfikę macierzy. W sytuacjach, gdzie dokładność jest priorytetem, metody z wyborem elementu głównego są preferowane.

5. **Potrzeba Dalszej Analizy:** Wyniki dla rozkładu LU z wyborem elementu głównego sugerują potrzebę dalszej analizy tej metody, aby zrozumieć przyczyny relatywnie dużych błędów, które mogą wynikać z konkretnych właściwości macierzy lub detalów implementacji.

Literatura

[1] D. Kincaid, W. Cheney, Analiza numeryczna, Wydawnictwa Naukowo-Techniczne, Warszawa 2006. ISBN 83-204-3078-X