Obliczenia Naukowe - Laboratorium 5

Michał Waluś 279695

Styczeń 2026

1 Problem

Dany jest problem równoważny rozwiązaniu dużego układu równań, którego współczynniki można przedstawić jako rzadką macierz kwadratową o specyficznej strukturze.
Układ składa się z $n \geqslant 4$ równań, przy czym znamy również parametr $l \in \mathbb{N}_{\geqslant 2}$, taki że $v = \frac{n}{l} \in \mathbb{N}$. Nasz problem sprowadza się do następującej postaci

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

gdzie $\mathbf{A} \in \mathbb{R}^{n \times n}$ oraz $\mathbf{b} \in \mathbb{R}^n$, a macierz $\mathbf{A}$ ma następującą strukturę:

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{C}_1 & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_2 & \mathbf{A}_2 & \mathbf{C}_2 & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_3 & \mathbf{A}_3 & \mathbf{C}_3 & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_2 & \mathbf{A}_2 & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{A}_{v-3} & \mathbf{C}_{v-3} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{B}_{v-2} & \mathbf{A}_{v-2} & \mathbf{C}_{v-2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{B}_{v-1} & \mathbf{A}_{v-1} & \mathbf{C}_{v-1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{B}_v & \mathbf{A}_v \end{pmatrix} \tag{2}$$

Macierze $\mathbf{0}$, $\mathbf{A}_k$, $\mathbf{B}_k$ oraz $\mathbf{C}_l$ są wszystkie kwadratowymi macierzami o wymiarach $l \times l$, gdzie $\mathbf{0}$ to macierz zerowa, dla $k = 1, \ldots, v$, $\mathbf{A}_k$ jest macierzą gęstą, dla $k = 2, \ldots, v$, $\mathbf{B}_k$ ma niezerowy tylko pierwszy wierz i ostatnią kolumnę, tj. jest w następującej postaci:

$$\mathbf{B}_k = \begin{pmatrix} b^k_{1,1} & b^k_{1,2} & \cdots & b^k_{1,l-1} & b^k_{1,l} \\ 0 & 0 & \cdots & 0 & b^k_{2,l} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & b^k_{l-1,l} \\ 0 & 0 & \cdots & 0 & b^k_{l,l} \end{pmatrix} \tag{3}$$

Natomiast dla $k = 1, \ldots, v-1$, $\mathbf{C}_k$ jest macierzą diagonalną:

$$\mathbf{C}_k = \begin{pmatrix} c^k_{1,1} & 0 & \cdots & 0 & 0 \\ 0 & c^k_{2,2} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & c^k_{l-1,l-1} & 0 \\ 0 & 0 & \cdots & 0 & c^k_{l,l} \end{pmatrix} \tag{4}$$

Celemami są:

1. Wczytywanie macierzy **A** oraz wektora **b** z pliku.
2. Generowanie wektora **b** na bazie macierzy **A**, przyjmując $\mathbf{x} = (1, 1, \ldots, 1)$ oraz użycie go do sprawdzenia algorytmu, wyliczając błąd względny.
3. Rozwiązanie problemu (1) używając eliminacji Gaussa:
   (a) bez wyboru elementu głównego.
   (b) z częściowym wyborem elementu głównego.
4. Wyznaczenie rozkładu LU macierzy **A** używając eliminacji Gaussa:
   (a) bez wyboru elementu głównego.
   (b) z częściowym wyborem elementu głównego.
5. Rozwiązanie układu równań $\mathbf{Ax} = \mathbf{b}$ używając wcześniej wyznaczonego rozkładu LU.

2 Metoda Eliminacji Gaussa

Metoda eliminacji Gaussa pozwala na rozwiązanie układu równań liniowych, poprzez przekształcenie go do postaci "trójkąta".

Mamy początkowy układ n równań z n niewiadomymi:

$$\begin{aligned}
a_{1,1}^{(1)}x_1 + a_{1,2}^{(1)}x_2 + \cdots + a_{1,n-1}^{(1)}x_{n-1} + a_{1,n}^{(1)}x_n &= b_1^{(1)} \\
a_{2,1}^{(1)}x_1 + a_{2,2}^{(1)}x_2 + \cdots + a_{2,n-1}^{(1)}x_{n-1} + a_{2,n}^{(1)}x_n &= b_2^{(1)} \\
\vdots \qquad \vdots \qquad\qquad\qquad \vdots \qquad\quad \vdots \quad \vdots & \\
a_{n,1}^{(1)}x_1 + a_{n,2}^{(1)}x_2 + \cdots + a_{n,n-1}^{(1)}x_{n-1} + a_{n,n}^{(1)}x_n &= b_n^{(1)}
\end{aligned}$$

Może on również zostać zapisany jako $A^{(1)}x = b^{(1)}$, gdzie A to macierz współczynników, x wektor niewiadomych, a b wektor prawych stron. Przekształcamy go, odejmując stronami pierwsze równanie, pomnożone przez odpowiedni czynnik, od wszystkich pozostałych równań. Dla j-ego równania (gdzie $j = 2, \ldots, n$) ten czynnik to

$$l_{1,j} = \frac{a_{j,1}^{(1)}}{a_{1,1}^{(1)}}$$

Powstałe w ten sposób współczynniki tworzą macierz $A^{(2)} = (a_{i,j}^{(2)})_{i,j=1}^{n}$ oraz wektor prawych stron $b^{(2)} = (b_i^{(2)})_{i=1}^{n}$. Istotną własnością jest że dla $j > 1$, $a_{j,1}^{(2)} = 0$.

Analogicznie postępujemy dalej, gdzie w k-tym kroku, od równań znajdujących się w rzędach $j = k+1, k+2, \ldots, n$ odejmujemy k-te równanie przemnożone przez

$$l_{k,j} = \frac{a_{j,k}^{(k)}}{a_{k,k}^{(k)}}$$

co zeruje współczynniki w k-tej kolumnie, dla rzędów o indeksach większych od k.

Finalnie otrzymujemy postać:

$$\begin{aligned} a_{1,1}^{(n)}x_1 + a_{1,2}^{(n)}x_2 + \cdots + a_{1,n-1}^{(n)}x_{n-1} + a_{1,n}^{(n)}x_n &= b_1^{(n)} \\ a_{2,2}^{(n)}x_2 + \cdots + a_{2,n-1}^{(n)}x_{n-1} + a_{2,n}^{(n)}x_n &= b_2^{(n)} \\ \ddots \qquad\qquad \vdots \quad \vdots \\ a_{n,n}^{(n)}x_n &= b_n^{(n)} \end{aligned}$$

Jeśli powyższa postać jest nieosobliwa, prosto możemy wyznaczyć rozwiązanie - zaczynając od ostatniej linii otrzymujemy

$$x_n = \frac{b_n^{(n)}}{a_{n,n}^{(n)}}$$

a następnie, idąc od dołu, dla $k = n-1, n-2, \ldots, 1$ otrzymujemy:

$$x_k = \frac{b_k^{(n)} - \sum_{j=k+1}^{n} a_{k,j}^{(n)} x_j}{a_{k,k}^{(n)}} \tag{5}$$

Zauważmy, że przyjmując że suma od $n+1$ do n jest pusta i jej wartość wynosi 0, to ten wzór jest również poprawny dla $k = n$.

2.1 Częściowy wybór

W eliminiacji Gaussa z cześciowym wyborem w k-tym kroku znajdujemy taki element $a_{p,k}^{(k)}$, że

$$|a_{p,k}^{(k)}| = \max_{k \leqslant i \leqslant n} |a_{i,k}^{(k)}| \tag{6}$$

Czyli element o największym module w k-tej kolumnie (licząc tylko rzędy, których pierwszych $k-1$ elementów zostały zredukowane do zero), a następnie zamienienie p-tego rzędu z k-tym rzędem oraz p-tej wartości w wektorze b z k-tą.

3 Postać w pamięci

Z powodu dużego rozmiaru naszej macierzy $\mathbf{A}$, zapamiętywanie jej całej w pamięci komputera byłoby marnowaniem zasobów, wystarczy zapamiętać tylko macierze $\mathbf{A}_k$, $\mathbf{B}_k$ oraz $\mathbf{C}_k$. Ponieważ macierze $\mathbf{A}_k$ są gęste, to trzeba je zapamiętać w całości, zatem zapamiętujemy je jako wektor (tablicę) macierzy $l \times l$. Tablica ta zawiera $v \cdot l \cdot l = n \cdot l = O(nl)$ liczb.

Macierze $\mathbf{B}_k$ zawierają za to znacznie mniej elementów, więc nie musimy pamiętać ich w całości. Każdą z nich pamiętamy jako wektor (tablicę) długości $2l-1$, gdzie pierwszy rząd macierzy to elementy o indeksach od 1 do l, a ostatnia kolumna o indeksach od l do $2l-1$. Zauważym, że indeks l, to element $b_{1,l}$, który jest wspólny. Zapamiętujemy je wszystkie jako tablicę tablic długości $v-1$, gdzie na k-tym indeksie znajduje się reprezentacja macierzy $\mathbf{B}_{k+1}$. Tablica ta zawiera łącznie $(v-1)(2l-1) = 2n - 2l - v + 1 = O(n)$ liczb.

Macierze $\mathbf{C}_k$ moglibyśmy zapamiętać tylko jako tablicę zawierającą ich przekątne, ale ponieważ w algorytmie eliminiacji Gaussa odejmujemy całe rzędy, to musielibyśmy tworzyć dla nich macierze w obrębie algorytmu (które musiałby być wszystkie i tak pamiętane do końca, by można było wyznaczyć rozwiązania), zatem zapamiętujemy je w całości, tak samo jak macierze $\mathbf{A}_k$, jako tablicę macierzy, zawierającą $(v-1) \cdot l \cdot l = n \cdot l - l^2 = O(nl)$ liczb.

Nasza reprezentacji macierzy $\mathbf{A}$, łącznie zajmie $O(nl)$ miejsca w pamięci, co dla stałych wartości l jest liniowe względem wielkości macierzy.

3.1 Wczytywanie z pliku

Macierz $\mathbf{A}$ jest zapisana w pliku, w następującym formacie:

- Pierwszy wierz zawiera 2 liczby naturalne oddzielone spacją, w formacie `n l`, gdzie n to rozmiar macierzy, a l to wymiar macierzy $\mathbf{A}_k$.
- Pozostałe wiersze zawierają 2 liczby naturalne i liczbę wymierną ze skończonym rozwinięciem dziesiętnym, w formacie `r c v`, gdzie v to wartość macierzy w rzędzie r i kolumnie c, tj. $a_{r,c}$.

Definiujemy $b_r = \lfloor \frac{r-1}{l} \rfloor + 1$ i $b_c = \lfloor \frac{c-1}{l} \rfloor + 1$ (od blok), które pozwalają ustalić w której macierzy się znajduje dany element, np. $b_c = 3$ znaczy że jesteśmy w 3 "bloku $l \times l$"od lewej, a także $r_l = r - (b_r - 1) \cdot l$ oraz $c_l = c - (b_c - 1) \cdot l$ (od "lokalne"), wyznaczające indeks w obrębie danego bloku.

Dla danych r i c mamy następujące przypadki, gdzie umieszczamy odczytaną wartość.

- Jeśli $b_r = b_c$, to znaczy, że wartość znajduje się w którejś z macierzy $\mathbf{A}_k$, dokładniej tej o indeksie b_r. Dodatkowo, element tej jest w jej r_l rzędzie oraz c_l kolumnie.
- Jeśli $c + l = r$, to wtedy element jest przesunięty o l w prawo od głównej przekątnej, czyli znajduje się on na przekątnej macierzy $\mathbf{C}_{b_r}$, na pozycji (r_l, c_l) (które są sobie równe).
- Jeśli $b_r = b_c + 1$ i $r_l = 1$, to element znajduje się w pierwszym rzędzie bloku poniżej przekątnej, więc należy do $\mathbf{B}_{b_r}$, czyli zapisujemy go w b_c-tej tablicy na pozycji c_l.
- Jeśli $b_r = b_c + 1$ i $c_l = l$ to znajduje się w tym samym bloku co wyżej, ale w ostatniej kolumnie, zatem zapisujemy go na $(l - 1) + r_l$ pozycji tej samej tablicy. Składnik $l - 1$ wynika ze struktury tablicy zawierającej elementy $\mathbf{B}_k$.
- Jeśli element nie spełna żadnego z powyższych warunków, to nie jest żadnym z elementów macierzy $\mathbf{A}$, który może być niezerowy i plik nie definiuje poprawnie macierzy.

Wektor $\mathbf{b}$ jest zapisany w pliku, gdzie pierwszy rząd zawiera długość wektora, a pozostałe rzędy po 1 liczbie, która jest kolejną wartością należącą do wektora i wczytanie go jest trywialne.

4 Algorytmy

4.1 Elimininacja Gaussa bez wyboru

Z powodu specyficznej postaci macierzy $\mathbf{A}$ podczas wykonywania algorytmu eliminiacji Gaussa na niej, w danym momencie nie musimy rozpatrzać większości rzędów.

Rozważane wartości wyglądają następująco:

- Dla pierwszych $l - 1$ kolumn (czyli też $l - 1$ kroków metody), wartości występują tylko w $\mathbf{A}_1$ oraz w pierwszym rzędzie $\mathbf{B}_2$. Zatem dla pierwszych kroków możemy rozważać tylko $\mathbf{A}_1$, $\mathbf{C}_1$ oraz pierwsze rzędy $\mathbf{B}_2$ i $\mathbf{A}_2$. Nie musimy rozważać macierzy $\mathbf{C}_2$, ponieważ wszystkie wartości nad nią to 0.
- Dla l-tej kolumny niezerowe wartości są również w ostatniej kolumnie $\mathbf{B}_2$, zatem rozważamy również całą macierz $\mathbf{A}_2$. Po wykonaniu tego kroku macierz $\mathbf{B}_2$ powinna zawierać same zera.

```
Function EliminacjaGaussa(An[v], Bn[v − 1], Cn[v − 1], b[n]):
    Define factor, cur, index;
    for k ← 1 to v do
        for i ← 1 to l do
            cur ← An[k][i, i] ;          // element w i-tym rzędzie i kolumnie A_k
            for j ← i + 1 to l do
                /* Wszystkie rzędy w An[k] poniżej bieżącego                */
                factor ← An[k][j, i]/cur;
                /* W implementacji możemy pominąć pierwsze i − 1 elementów z
                   obu rzędów                                               */
                An[k][j] ← An[k][j] − factor · An[k][i];
                b[(k − 1)l + j] ← b[(k − 1)l + j] − factor · b[(k − 1)l + i];
                if k < v then
                    /* Rozważamy macierz Cn[k], której nie ma w ostatnich
                       rzędach. W implementacji wystarczy rozważyć pierwsze i
                       kolumn.                                              */
                    Cn[k][j] ← Cn[k][j] − factor · Cn[k][i];
                end
            end
            if k < v then
                /* Poniżej An[v] nie ma więcej rzędów                       */
                factor ← Bn[k][i]/cur;
                for j ← i to l do
                    /* kl − 1 to koniec rzędu, poza elementem z Bnv         */
                    Bn[k][j] = Bn[k][j] − factor · An[k][i, j];
                end
                An[k + 1][1] ← An[k + 1][1] − factor · Cn[k][i];
                b[k · l + 1] ← b[k · l + 1] − factor · b[(k − 1) · l + i];
                if i = l then
                    /* Rozważając ostatnią kolumnę A_k, musimy pamiętać o
                       kolunie B_{k+1}                                      */
                    for j ← 2 to l do
                        factor ← Bn[k][l − 1 + j]/cur;
                        Bn[k][l − 1 + j] ← Bn[k][l − 1 + j] − factor · cur ;   /* lub
                         ustawiamy na 0 */
                        An[k + 1][j] ← An[k + 1][j] − factor · Cn[k][l];
                        b[k · l + j] ← b[k · l + j] − factor · b[k · l];
                    end
                end
            end
        end
    end
end
```

Algorithm 1: Eliminacja Gaussa bez wyboru

- Dalej postępujemy analogicznie, w krokach $kl+1$ do $(k+1)l-1$ (dla $k=2,\ldots,v-1$) rozważamy macierze $\mathbf{A}_k$, $\mathbf{C}_k$ oraz pierwsze rzędy $\mathbf{A}_{k+1}$ i $\mathbf{B}_{k+1}$, a w $(k+1)l$-tym kroku również resztę macierzy $\mathbf{A}_{k+1}$ i ostatnią kolumnę $\mathbf{B}_{k+1}$.
- W ostatnich l krokach/rzędach nie ma już macierzy $\mathbf{C}_v$, macierz $\mathbf{B}_v$ jest zerami, a także nie ma żadnej macierzy poniżej, więc rozważamy tylko $\mathbf{A}_v$.

Bazując na powyższych rozważaniach możemy prosto napisać algorytm eliminacji Gaussa dla macierzy $\mathbf{A}$.

Pseudo-kod algorytmu został przedstawiony jako "Algorithm 1".

Tablice z macierzami $\mathbf{A}_k$ nazywamy An, analogicznie Cn dla $\mathbf{C}_k$ oraz Bn dla tablicy z reprezentacjami $\mathbf{B}_k$. Dla danej macierzy M, M[1,2] oznacza element w 1 rzędzie i 2 kolumnie, a M[1] pierwszy rząd macierzy.

Po wykonaniu eliminacji Gaussa jesteśmy w stanie prosto wyliczyć rozwiązania korzystając ze wzoru (5). Dodatkowo, ponieważ macierze $\mathbf{B}_k$ leżą poniżej przekątnej (i powinny być macierzami zerowymi) możemy je pominąć. Pseudokod algorytmu został przedstawiony jako "Algorithm 2".

Function `Rozwiazania`($An[v]$, $Cn[v-1]$, $b[n]$)`:`
  Define $x[n]$, s, b_k, k_l;
  for $k \leftarrow n$ **down to** 1 **do**
    $b \leftarrow \lfloor \frac{k-1}{l} \rfloor + 1$; `// Blok`
    $k_l \leftarrow k-(b-1)\cdot l$;
    $s \leftarrow b[k] - \sum_{j=k_l+1}^{l} An[b][k_l,j]\cdot x[(b-1)l+j]$;
    if $b < v$ **then**
      `/* Na pozycjach dalszych niż` k_l `powinny być 0 */`
      $s \leftarrow s - \sum_{j=1}^{k_l} Cn[b][k_l,j]\cdot x[b\cdot l+j]$;
    end
    $x[k] \leftarrow s/An[b][k_l,k_l]$;
  end
  Return x;
end

Algorithm 2: Wyznaczanie rozwiązań układu równań.

4.1.1 Złożoność

Podczas analizy złożoności czasowej i pamięciowej będziemy traktować te 2 algorytmy jako jedna całość.

Ponieważ algorytm używa tylko kilka nowych zmiennych oraz jedną tablicę długości n, to jego złożoność pamięciowa jest liniowa - $O(n)$.

Zaczynając od góry, pierwsza pętla wykona się v razy, a wewnętrzna l razy i ponieważ nic nie wykonuje się pomiędzy nimi to możemy je traktować jako jedną pętlę która wykonuje się $v\cdot l = n$ razy.
Operacja przypisania elementu wykonuje się w czasie stałym, a odejmowanie rzędów macierzy długości l ma złożoność liniową względem l - $O(l)$. Pierwsza pętle indeksowana przez j wykonuje się maksymalnie l razy, zatem jej złożoność czasowo to $O(l^2)$.

Poniżej (w instrukcji 'If') mamy 2 pętle, które wykonują się $O(l)$ razy, wewnątrz których instrukcję mają złożoność liniową względem l, zatem wszystkie operacje wewnątrz instrukcji 'If' są łącznie złożności $O(l^2)$.
Podsumowując to wszystkie, funkcja EliminacjaGaussa ma złożoność czasową $n \cdot O(l^2) = O(nl^2)$, co dla stałych wartości l jest złożonością liniową względem n.

Wewnątrz funkcji "Rozwiazania"mamy jedną pętle wykonującą się n razy, wewnątrz której najdroższymi operacjami jest wyliczanie wartości zmiennej s, używając sum maksymalnie l elementów, zatem złożoność czasowa całej funkcji to $O(nl)$.

Otrzymujemy zatem, że złożoność czasowa rozwiązania naszego układu równań to $O(nl^2)$, co dla stałych wartości l jest liniowe względem rozmiaru macierzy - $O(n)$.

4.2 Eliminacja Gaussa z częściowym wyborem

Dodanie częściowego wyboru jest nie za trudną modyfikacją origynalnego algorytmu, jednak z powodu zamieniania rzędów, macierze $\mathbf{C}_k$ mogą mieć wartości w dowolnych komórkach nie tylko pod przekątną. Co więcej możemy zamieniać z pierwszym rzędem $\mathbf{B}_{k+1}$, co powoduje, że pierwsza wartość z $\mathbf{C}_{k+1}$ może się znaleźć w dowolnym rzędzie w pierwszej kolumnie na prawo od $\mathbf{C}_k$. Dodatkowo, dokonując częściowego wyboru dla ostatniej kolumny którejś z $\mathbf{A}_k$, musimy rozważać całą ostatnią kolumnę $\mathbf{B}_{k+1}$, więc wartość może się znaleźć w dowolnej kolumnie w pierwszym rzędzie nad $\mathbf{C}_{k+1}$ (l komórek na prawo od ostatniego rzędu $\mathbf{C}_k$).

Z powodów opisanych powyżej musimy zdefiniować nową strukturę, która będzie przechowywać te wartości. Ponieważ interesuje nas tylko pierwsza kolumna i ostatni rząd tego "bloku", to możemy zrobić to w podobny sposób do tablicy zawierającej $\mathbf{B}_k$, tylko odbite względem przekątnej, tj. komórki o indeksach od 1 do l to pierwsza kolumna, a od l do $2l-1$ to ostatni rząd. Ponieważ w ostatnich $2l$ rzędach (2 bloki) macierz $\mathbf{A}$ się kończy zaraz po macierzy $\mathbf{C}_{v-1}$ lub $\mathbf{A}_v$, to będziemy potrzebować tylko $v-2$ takich tablic. Będziemy nazywać te bloki macierzami $\mathbf{D}_k$, indeksując tak, że macierz $\mathbf{D}_k$ znajduje się na prawo od macierzy $\mathbf{C}_k$. Przechowując one łącznie $(v-2)(2l-1) = 2n - 4l - v + 1 = O(n)$ liczb, zatem nie zmieniają złożoności pamięciowej algorytmu.

Poniżej poprzez wybranie danego rzędu rozumiemy, że zawiera on element o maksymalnym module w danej kolumnie, tj. spełnia równość (6).

Podczas dokonywania wyboru dla j-tej kolumny k-tego bloku rozważamy następujące rzędy:

- Zawsze rozważamy rzędy macierzy $\mathbf{A}_k$ o indeksach większych i równych j. Oznaczmy indeks wybranego rzędu przez $p \geqslant j$. Jeśli $p = j$, nic nie zmieniamy, ale jeśli $p > j$ zamieniamy j-ty rząd macierzy $\mathbf{A}_k$ z p-tym, jeżeli $k < v$, to również zamieniamy te rzędy w macierzy $\mathbf{C}_k$, a jeśli $k < v-1$, to zamieniamy je też w macierzy $\mathbf{D}_k$ (ponieważ poza pierwszą kolumną muszą być one zerami, to w implementacji rozpatrzamy tylko ją).

- Jeżli $k < v$, to poniżej macierzy $\mathbf{A}_k$, znajduje się macierz $\mathbf{B}_{k-1}$, więc musimy rozpatrzyć jej pierwszy rząd. Jeśli zostanie on wybrany, to zamieniamy go z j-tym rzędem macierzy $\mathbf{A}_k$, a także pierwszy rząd $\mathbf{A}_{k+1}$ z j-tym rzędem $\mathbf{C}_k$. Dodatkowo, jeśli $k < v-1$, to zamieniamy pierwszy rząd $\mathbf{C}_{k+1}$ z j-tym rzędem $\mathbf{D}_k$, ale ponieważ w żadnych poprzednich krokach nie modyfikowaliśmy wartości w dalszych kolumnach tych macierzy, to rozpatrzamy tylko pierwszą kolumnę ($\mathbf{C}_{k+1}$ przed wykonaniem eliminacji Gaussa jest macierzą diagonalną, więc w pierwszym rzędzie ma niezerową wartość tylko w pierwszej kolumnie).

```
Function ZamianaRzedow(An[v], Bn[v − 1], Cn[v − 1], b[n], k, j):
    Define Dn[v − 2], cur, p;
    Dn ← same zera;
    cur ← An[k][p, j] = max_{j⩽i⩽l} An[k][i, j];
    if k < v then
        if |Bn[k][j]| > |cur| then
            cur ← Bn[k][j];
            p ← l + 1;
        end
        if i = l then
            for i ← 2 to l do
                if |Bn[k][l − 1 + i]| > |cur| then
                    cur ← Bn[k][l − 1 + i];
                    p ← l + i;
                end
            end
        end
    end
    b[(k − 1)l + j] ↔ b[(k − 1)l + p]];
    if j + 1 ⩽ p ⩽ l then
        /* Wybór z pierwszego przypadku                                  */
        An[k][j] ↔ An[k][p] ;                                  // Zamiana
        if k < v then
            Cn[k][j] ↔ Cn[k][p];
        end
        if k < v − 1 then
            Dn[k][j] ↔ Dn[k][p];
        end
    end
    if p = l + 1 then
        /* Wybór z drugiego przypadku                                    */
        An[k][j] ↔ Bn[k][1 : l] ;                  // Elementy od 1 do l
        Cn[k][j] ↔ An[k + 1][1];
        if k < v − 1 then
            Dn[k][j] ↔ Cn[k + 1][1, 1];
        end
    end
    if p > l + 1 then
        /* Wybór z trzeciego przypadku                                   */
        An[k][l, l] ↔ Bn[k][p − 1];
        Cn[k][j] ↔ An[k + 1][p − l];
        if k < v − 1 then
            Dn[k][l : 2l − 1] ↔ Cn[k + 1][p − l];
        end
    end
end
```

Algorithm 3: Częściowy Wybór

- Jeżli oprócz $k < v$, mamy jeszcze $j = l$, czyli ropatrzamy ostatnią kolumnę $\mathbf{A}_k$, to rozważamy też całą ostatnią kolumnę $\mathbf{B}_{k+1}$, jeśli wybierzemy p-ty rząd z tej macierzy, to dokonujemy podobnych przekształceń jak w drugim punkcie (tj. zamieniamy j-ty rząd $\mathbf{A}_k$ z p-tym $\mathbf{B}_{k+1}$ i analogicznie dla $\mathbf{C}_k$ z $\mathbf{A}_{k+1}$ i $\mathbf{D}_k$ z $\mathbf{C}_{k+1}$) z tą różnicą, że musimy teraz rozpatrać cały rząd $\mathbf{C}_{k+1}$, a nie tylko pierwszą kolumnę. Warto również zauważyć, że ponieważ ostatni rząd $\mathbf{A}_k$ ma niezerową wartość tylko w ostatniej kolumnie, to nie musimy rozważać pozostałych kolumn $\mathbf{A}_k$ ani $\mathbf{B}_{k+1}$.

Dodatkowo we wszystkich przypadkach zamieniamy wartości $\mathbf{b}$.

Pseudokod algorytmu został przedstawiony jako "Algorithm 3".

Pozostała część algorytmu eliminacji Gaussa wygląda podobnie do tej bez wyboru, jedyne różnice polegają na rozważaniu całych rzędów macierzy $\mathbf{C}_k$ zamiast części do przekątnej oraz dodatkowo należy pamiętać o odemowaniu rzędów $\mathbf{D}_k$, od $\mathbf{C}_{k+1}$ podczas odemowania od $\mathbf{B}_{k+1}$ oraz, od pozostałych rzędów $\mathbf{D}_k$, w wewnętrznej pętli (pierwsza pętla indeksowana j w oryginalnym algorytmie).

Algorytm wyznaczania rozwiązań ma podobne zmiany i jego pseudokod wygląda następująco:

Function `Rozwiazania`($An[v]$, $Cn[v-1]$, $Dn[v-2]$, $b[n]$)**:**
  Define $x[n]$, s, b_k, k_l;
  for $k \leftarrow n$ **down to** 1 **do**
    $b \leftarrow \lfloor \frac{k-1}{l} \rfloor + 1$; `// Blok`
    $k_l \leftarrow k - (b-1) \cdot l$;
    $s \leftarrow b[k] - \sum_{j=k_l+1}^{l} An[b][k_l, j] \cdot x[(b-1)l + j]$;
    if $b < v$ **then**
      $s \leftarrow s - \sum_{j=1}^{l} Cn[b][k_l, j] \cdot x[b \cdot l + j]$;
    end
    if $b < v - 1$ **then**
      $s \leftarrow s - Dn[b][k_l] \cdot x[(b+1) \cdot l + 1]$;
      if $k_l = l$ **then**
        $s \leftarrow s - \sum_{j=2}^{l} Dn[b][l - 1 + j] \cdot x[(b+1) \cdot l + j]$;
      end
    end
    $x[k] \leftarrow s / An[b][k_l, k_l]$;
  end
  Return x;
end

Jedyne zmiany w powyższym algorytmie to rozważanie całych rzędów $\mathbf{C}_k$ oraz rozważanie $\mathbf{D}_k$.

Algorytm zamiany rzędów podczas wyboru rozważa tylko jedną kolumnę w maksymalnie $l+1$ rzędach, a podczas zamiany zamienia ze sobą dwa rzędy długości maksymalnie $3l+1$, więc ma złożoność czasową $O(l)$ i wykonujemy go łącznie n razy. Pozostałe zmiany wprowadzone do algorytmu eliminacji Gaussa nie zmieniają jego złożoności obliczeniowej (dodawanie rzędów macierzy $\mathbf{D}_k$ oraz pozostałych $\mathbf{C}_k$ ma złożoność niewiększą niż dodawanie rzędów $\mathbf{A}_k$), zatem złożoność obliczeniowa eliminacji Gaussa z częściowym wyborem elementu głównego ma złożoność $O(nl + nl^2) = O(nl^2)$, czyli taką samą jak prostsza wersja - dla stałych wartości l liniową

względem rozmiaru macierzy. Dodana została tylko tablica Dn długości zawierająca $O(n)$ liczb oraz 1 zmienna, zatem złożoność pamięciowa również pozostaje liniowa.

4.3 Rozkład LU

Rozkład LU to rozkład macierzy ma macierz dolno- i górno-trójkątną, gdzie L to macierz dolno trójkątna zawierająca 1 na przekątnej, a U to macierz górno trójkątna. Jesteśmy w stanie ją zapisać za pomocą oryginalnej macierzy pomijając przekątną z L. Algorytm eliminacji Gaussa prosto wyznacza ten rozkład, gdzie U to wynikowa macierz górnotrójkątna, a L poniżej przekątnej zawiera współczynniki, które były użyte podczas odejmowania rzędów ($a_{i,j}$ zawiera współczynnik który został dobrany by wyzerować ten element tj. $a_{i,j}/a_{i,i}$).

4.3.1 Bez wyboru elementu głównego

Jesteśmy w stanie prosto wyznaczyć rozkład LU przekształcając oryginalny algorytm:

- Nie modyfikujemy **b**, więc usuwamy wszystkie modyfikacje tego wektora oraz go z listy argumentów.
- Za każdym razy gdy wyznaczamy $factor$ zapisujemy go w elemencie z którego został wyznaczony (tj. element który dzielimy przez cur) i pomijamy go potem przy odejmowaniu.

Algorytm ten oczywiście dalej ma taką samą złożoność czasową i pamięciową.

Jeśli chcemy rozwiązać układ równań z macierzą w postaci LU, najpierw wywołujemy oryginalną wersję algorytmu eliminacji Gaussa bez wyboru pomijając wczystkie modyfikacje macierzy (zostawiając modyfikacje **b**), a $factor$ zamiast obliczać, odczytujemy z macierzy **A** - jesli oryginalnie został wyznaczony poprzez $An[k][j,i]/cur$, to jest zapisany jako $An[k][j,i]$. Następnie postępujemy tak jak w oryginalnym algorytmie (Algorithm 2). Jeśli nie chcemy modyfikować **b**, możemy skopiować jego wartość i ją użyć. Ponieważ pomijamy tu drogie modyfikacje rzędów, algorytm ten ma złożoność $O(nl)$.

4.3.2 Z częściowym wyborem elementu głównego

Algorytm rozkładu LU z częściowym wyborem elementu głównego wygląda bardzo podobnie do tego bez wyboru, z tą różnicą, że modyfikujemy algorytm eliminacji Gaussa z częściowym wyborem. Znowu pomijamy tu wszystkie modyfikacje wektora **b**, jednak ponieważ nasza oryginalna struktura nie zawiera macierzy $\mathbf{D}_k$, to musimy zwrócić ich wartości (lub zmodyfikować strukturę tak by wstępnie zawierała tam zera), a także jeżeli chcemy rozwiązać ten układ równań, musimy zapisać i zwrócić dokonane permutacje (wybory). Permutacje zapisujemy w tablicy długości n, co nie wpływa na złożoność obliczeniową, dla ułatwienia implementacji wybrałem by zapisywać kolejne wartości zmiennej p.

Proces rozwiązywania wygląda bardzo podobnie, znowu używamy naszego oryginalnego algorytmu, z jedynie modyfikacjami wektora **b**, z tą różnicą, że teraz również dokonujemy zamian jego wartości, bazując na wartości p odczytanej z tablicy, co jest prostą modyfikacja algorytmu 3, pozostawiając jedynie modyfikacje wektora prawych stron. Samo wyznaczenie rozwiązań wygląda tak jak w oryginalnym algorytmie eliminacji z częściowym wyborem (musimy mieć dostęp do tablicy Dn zwróconej przez funkcję dokonującą rozkładu).

Jeśli chcemy używać tej samej funkcji do obu wariantów rozkładów LU, to wystarczy do algorytmu dla częściowego wyboru przekazać tablicę tablic 0 jako Dn oraz tablicę zer jako tablicę wartości p (przy naszej implementacji wartości mniejsze od $j+1$ są pomijane przy zamianach).

Ponieważ są to proste modyfikacje oryginalnego algorytmu, nie zmieniają one wynikowej złożoności obliczeniowej, ani pamięciowej.

5 Wyznaczanie b

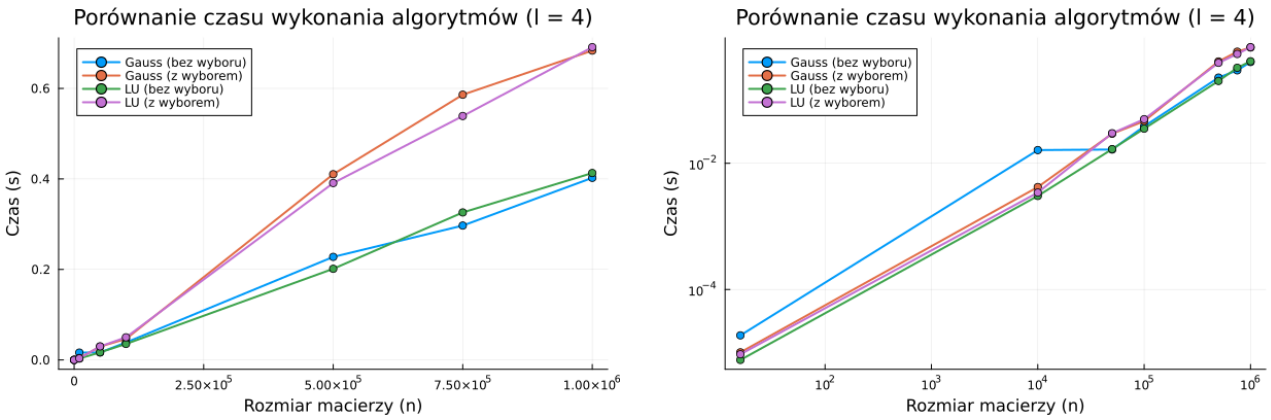
Mając daną macierz $\mathbf{A}$, wyznacznie wektora $\mathbf{b}$, tak żeby $\mathbf{x}$ zawierał same 1 jest proste. Dla k-tej wartości $\mathbf{b}$ dodajemy wszystkie liczby z k-tego rzędu macierzy $\mathbf{A}$, czyli z macierzy $\mathbf{B}_j$, $\mathbf{A}_j$ i $\mathbf{C}_j$ pamiętając o strukturze macierzy. Czas złożoność czasowa takiego algortymu jest liniowa dla stałych wartości l i ma też liniową złożoność czasową.

6 Porównanie czasów działania

Czas działania dla danych testowych ze strony prowadzącego zostały przedstawione w Tabeli 1 oraz na Rysunku 1. Dodatkowo, używając funkcji blockmat z pliku matrixgen.jl ze strony prowadzącego zostały wygenerowane macierze o zmiennych wartościach l (czasy działania rzedstawione w Tabeli 2 i na Rysunku 2 (a)) oraz zmiennych wartościach n (czasy działania przedstawione w Tabeli 3 i na Rysunku 2 (b)).

n	Gauss	Gauss (wybór)	LU	LU (wybór)
16	1.85×10^{-5}	1.01×10^{-5}	7.85×10^{-6}	1.11×10^{-5}
10000	4.50×10^{-3}	4.98×10^{-3}	2.76×10^{-3}	4.96×10^{-3}
50000	2.10×10^{-2}	2.88×10^{-2}	2.17×10^{-2}	2.26×10^{-2}
100000	3.88×10^{-2}	5.45×10^{-2}	3.84×10^{-2}	5.10×10^{-2}
500000	2.02×10^{-1}	4.15×10^{-1}	2.12×10^{-1}	4.15×10^{-1}
750000	3.03×10^{-1}	5.28×10^{-1}	3.10×10^{-1}	5.64×10^{-1}
1000000	4.281×10^{-1}	6.90×10^{-1}	4.11×10^{-1}	7.02×10^{-1}

Tabela 1: Czasy działania dla danych ze strony ($l = 4$)

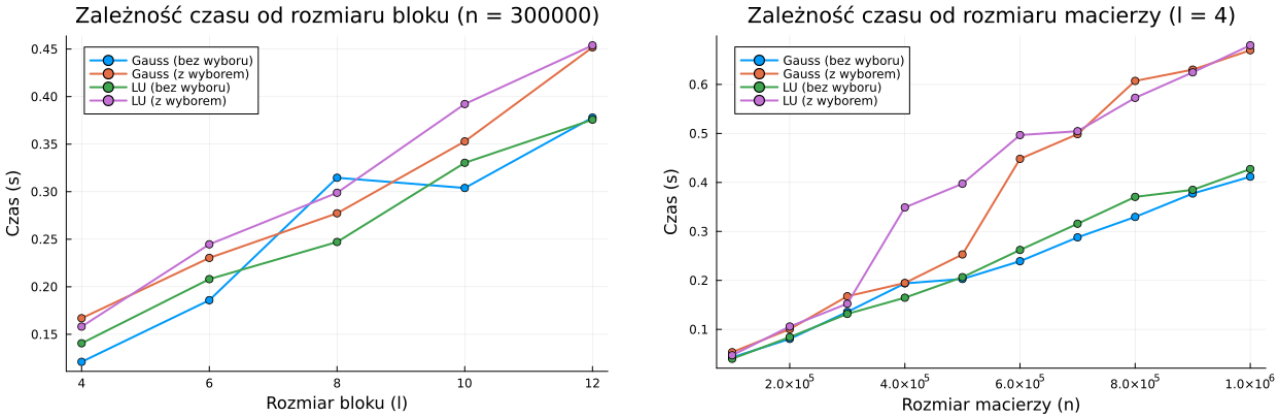


(a) skala liniowa

(b) skala logarytmiczna

Rysunek 1: Czasy działania dla danych ze strony ($l = 4$)

Jak oczekiwane, dla dużych wartości n, eliminacja bez wyboru jest szybsza od eliminiacji z wyborem, chociaż jest to kosztem minimalnie gorszych wyników. Na wykresie 2(b) widać przeskok dla algorytmów z wyborem w okolicach n wielkości 4000000, może być to spowodowane funkcją generującą tworzącą macierz dla której zachodzi relatywnie mało zamian rzędów dla mniejszych danych. Pomimo dziwnego zachowania, po przeskoku wykres dalej sugeruje czas liniowy, co jest zgodne z oczekiwaniami. Tak jak oczekiwane, wykonywanie rozkładu LU jest



(a) Zmienne l ($n = 300000$)

(b) Zmienne n ($l = 4$)

Rysunek 2: Czasy działania dla danych wygenerowanych poprzez matrixgen.jl

l	Gauss	Gauss (wybór)	LU	LU (wybór)
4	1.19×10^{-1}	1.55×10^{-1}	1.24×10^{-1}	1.63×10^{-1}
6	1.83×10^{-1}	2.18×10^{-1}	2.62×10^{-1}	2.23×10^{-1}
8	2.64×10^{-1}	2.82×10^{-1}	2.57×10^{-1}	3.05×10^{-1}
10	3.36×10^{-1}	3.66×10^{-1}	3.12×10^{-1}	3.71×10^{-1}
12	3.87×10^{-1}	4.17×10^{-1}	3.84×10^{-1}	4.74×10^{-1}

Tabela 2: Czasy działania dla zmiennych wartości l ($n = 300000$)

wolniejsze niż samej eliminacji, aczkolwiek w bardzo niewielkim stopniu.

Na wykresie 2(a) widzimy zależność czasu działania od l i wykres sugeruje liniową zależność, mimo analizy złożoności sugerującej kwadratową (względem l), może to znaczyć że kwadratowy składnik ma niewielki współczynnik i niewielki wpływ na obliczenia. Widzimy również, że jak oczekiwane, algorytmy z wyborem są wolniejsze oraz rozkład LU jest wolniejszy od zwykłej eliminacji. Dla $l = 8$ widzimy skok czasu eliminacji Gaussa, prawdopodobnie spowodowany procesem w tle.

7 Wnioski

Niektóre problemy, mimo ich pozornej trudności da się szybko (pod względem złożoności obliczeniowej) rozwiązać relatywnie prostym algorytmem. Mimo tego, że typowy algorytm rozwią-

n	Gauss	Gauss (wybór)	LU	LU (wybór)
100000	4.28×10^{-2}	5.34×10^{-2}	4.00×10^{-2}	4.71×10^{-2}
200000	8.06×10^{-2}	1.01×10^{-1}	8.42×10^{-2}	1.06×10^{-1}
300000	1.36×10^{-1}	1.68×10^{-1}	1.32×10^{-1}	1.52×10^{-1}
400000	1.94×10^{-1}	1.95×10^{-1}	1.65×10^{-1}	3.49×10^{-1}
500000	2.03×10^{-1}	2.53×10^{-1}	2.07×10^{-1}	3.97×10^{-1}
600000	2.39×10^{-1}	4.48×10^{-1}	2.62×10^{-1}	4.97×10^{-1}
700000	2.88×10^{-1}	4.99×10^{-1}	3.16×10^{-1}	5.05×10^{-1}
800000	3.30×10^{-1}	6.07×10^{-1}	3.70×10^{-1}	5.73×10^{-1}
900000	3.77×10^{-1}	6.30×10^{-1}	3.85×10^{-1}	6.25×10^{-1}
1000000	4.12×10^{-1}	6.70×10^{-1}	4.27×10^{-1}	6.80×10^{-1}

Tabela 3: Czasy działania dla zmiennych wartości n ($l = 4$)

zywania takiego układu równań jest tu niemożliwy do zaaplikowania, z powodu bardzo dużego rozmiaru macierzy byłby on bardzo nieefektywny, ponieważ ma sześcienną złożoność czasową $O(n^3)$. Tymczasem wykorzystując specjalną strukturę macierzy, jesteśmy w stanie rozwiązać ten układ w czasie liniowym, co powoduje że samo wczytanie macierzy zajmuje dłużej niż rozwiązanie.

Problem ten pokazuje, że pomimo tego, że domyślne funkcje biblioteczne są bardzo dobrym rozwiązaniem do większości problemów (i często posiadają lepsze optymalizacje niż to co jesteśmy w stanie szybko napisać), niekiedy warto jest samemu napisać rozwiązanie, które korzysta ze specyficznych własności naszego problemu, co pozwoli na znacznie lepszą złożność czasową i pamięciową. Często nawet najlepsze generyczne funkcje nie mogą osiągnąć tak dużej skuteczności jak dość prosta funkcja, która korzysta z wiedzy o naszym problemie.