Sprawozdanie
Obliczenia naukowe - lista 5

Kamil Król

244949

Opis problemu oraz struktury danych

Celem listy jest rozwiązanie układu równań liniowych: $\boldsymbol{Ax} = \boldsymbol{b}$, dla danej macierzy $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ i wektora prawych stron $\boldsymbol{b} \in \mathbb{R}^n$, gdzie $n \geqslant 4$.

Zgodnie z treścią listy macierz $\boldsymbol{A}$ jest rzadką macierzą (tj. mającą dużo elementów zerowych) blokową o następującej strukturze:

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

$v = n/\ell$, zakładając że n jest podzielne przez ℓ, gdzie $\ell \geqslant 2$ jest rozmiarem wszystkich kwadratowych macierzy wewnętrznych (bloków) $\boldsymbol{A}_i$, $\boldsymbol{B}_i$, $\boldsymbol{C}_i$, $\boldsymbol{0}$.

Macierze oznaczone przez $\boldsymbol{0}$ są macierzami zerowymi tzn. $\boldsymbol{0} \in \mathbb{R}^{\ell \times \ell}$. Macierze $\boldsymbol{A}_i$, $\boldsymbol{B}_i$, $\boldsymbol{C}_i$, są następującej postaci:

(i) $\boldsymbol{A}_i \in \mathbb{R}^{\ell \times \ell}$, $i = 1, \ldots, v$ – macierze gęste,

(ii) $\boldsymbol{B}_i \in \mathbb{R}^{\ell \times \ell}$, $i = 2, \ldots, v$ – macierze mające tylko dwie ostatnie kolumny niezerowe:

$$\boldsymbol{B}_i = \begin{pmatrix} 0 & \cdots & 0 & b^i_{1\,\ell-1} & b^i_{1\,\ell} \\ 0 & \cdots & 0 & b^i_{2\,\ell-1} & b^i_{2\,\ell} \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & b^i_{\ell\,\ell-1} & b^i_{\ell\,\ell} \end{pmatrix},$$

(iii) $\boldsymbol{C}_i \in \mathbb{R}^{\ell \times \ell}$, $i = 1, \ldots, v-1$ – macierze diagonalne:

$$\boldsymbol{C}_i = \begin{pmatrix} c^i_1 & 0 & 0 & \cdots & 0 \\ 0 & c^i_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c^i_{\ell-1} & 0 \\ 0 & \cdots & 0 & 0 & c^i_\ell \end{pmatrix}.$$

Dodatkową informacją o danych jest ich duży rozmiar. W konsekwencji wyklucza to pamiętanie macierzy $\boldsymbol{A}$ jako tablicę o wymiarach $n \times n$ oraz użycie standardowych (bibliotecznych) algorytmów dla macierzy gęstych tj. takich, gdzie nie zakłada się dużej liczby elementów zerowych. Zatem dodatkowym wymaganiem co do zadania jest specjalna struktura pamiętająca efektywnie macierz $\boldsymbol{A}$. Taka, która pamięta tylko elementy niezerowe macierzy $\boldsymbol{A}$. W tym celu w moich programach użyłem biblioteki *SparseArrays*, która pamięta tylko elementy niezerowe. W naszych rozważaniach założymy, że możemy uzyskać dostęp do elementów macierzy w czasie stałym – $O(1)$. W rzeczywistości tak nie jest.
Samo użycie tej biblioteki jednak nie gwarantuje nam szybkiego algorytmu. Pojawia się bowiem problem w momencie nadpisywania elementów macierzy, które są zerami. W momencie takiego przypisania konieczna jest alokacja dodatkowej pamięci aby przechować nowy niezerowy element. Jedną z kluczowych

optymalizacji, którą zastosujemy będzie sztuczne przypisanie zer do *pól* macierzy $\boldsymbol{A}$ w obszarach gdzie zera faktycznie występują, ale zostaną one w przyszłości nadpisane. Takie obszary to obszary macierzy $\boldsymbol{C}$ poza elementami na diagonali. Zastosowanie tej optymalizacji pozwoliło aby wykresy przedstawiające czas wykonywania algorytmów od wielkości macierzy, które są przedstawione na końcu sprawozdania, faktycznie przypominały liniowe.
Ponadto konieczna jest modyfikacja standardowych algorytmów tak, aby uwzględniały one specyficzną postać macierzy $\boldsymbol{A}$, tj. jej rzadkość, regularność występowania elementów zerowych i niezerowych.

Zaimplementowane przeze mnie algorytmy (napisane w języku Julia i opisane poniżej) działają na macierzy transponowanej, ze względu na optymalizację tzn. dostęp do elementów po kolumnach jest szybszy niż po wierszach. W dalszych rozważaniach pominiemy ten fakt gdyż transpozycja nie zmienia wewnętrznej struktury macierzy $\boldsymbol{A}$ oraz nie zmienia ogólnej idei algorytmu. Odnośnik do dokumentacji: julialang - SparseArrays.

Opis metody eliminacji Gaussa bez wyboru elementu głównego

Ideą eliminajci Gaussa jest to aby początkowy układ równań sprowadzić do układu prostszego. Dokładniej chodzi o sprowadzenie macierzy A do postaci macierzy trójkątnej górnej. W celu doprowadzenia macierzy do takiej postaci, algorytm posługuje się podstawowymi operacjami na macierzy tzn. dodawanie, odejmowanie wierszy oraz mnożenie wierszy przez pewnien niezerowy współczynnik.
Dokładniej, proces polega na eliminowaniu (zerowaniu) elementów znajdujących się w pod przekątną macierzy. Zatem dla kolumny pierwszej będziemy chcieli wyzerować wszystkie elementy znajdujące się pod pierwszym elementem w tej kolumnie, dla kolumny drugiej – wszystkie elementy pod drugim (patrząc od góry) elementem w danej kolumnie. Bardziej formalnie, naszym celem będzie wyeliminowanie niewiadomej x_1 ze wszystkich $n-1$ pozostałych równań (czyli wszystkich poza pierwszym). Dla $i = 2, \ldots, n$ będziemy odejmować odpowiednią wielokrotność równania pierwszego od i-tego równania zerując przy tym współczynnik przy x_1 w i-tym równaniu.
Ogólnie dla macierzy A i jej elementów a_{ij}, gdzie $i, j = 1, \ldots, n$ i dla jej k-tej kolumny, będziemy zerować wszystkie elementy poniżej elementu na przekątnej w tej kolumnie, czyli a_{kk}. Dla $i = k+1, \ldots, n$ będziemy odejmować odpowiednią wielokrotność równania k-tego od i-tego równania zerując przy tym współczynnik przy x_k w i-tym równaniu. Inaczej aby wyzerować to x_k będziemy musieli pomnożyć wiersz k przez pewien współczynnik ψ_{ik}, króty dalej będziemy nazywać mnożnikiem, a następnie odjąć go od i-tego wiersza. Łatwo zauważyć, że mnożnik ten możemy obliczyć, ze wzoru $\psi_{ik} = \frac{a_{ik}}{a_{kk}}$. Widać też, że w przypadku kiedy na diagonali pojawi się element równy zero nasz algorytm nie zadziała, ponieważ pojawi się konieczność dzielenia przez zero.
Zanim zobaczymy jak można rozwiązać ten problem musimy przytoczyć jedną z własności układu równań Ax=b. Mianowicie, wiersze macierzy A i odpowiadające im wartości w wektorze b możemy dowolnie ze sobą zamieniać. Nie zmianiamy w ten sposób układu równań. Zwróćmy również uwagę na fakt, że aby istniało rozwiązanie tego równania macierz A musi być nieosobliwa. Teraz możemy przedstawić sposób na uniknięcie dzielenia przez 0.

Opis metody eliminacji Gaussa z częściowym wyborem elementu głównego

Przedstawiona poniżej metoda nosi nazwę metody eliminacjii Gaussa z częściowym wyborem elementu głównego. Czym jest element główny? Jest to element macierzy znajdujący się na diagonali. Dla k-tej kolumny, elementem głównym będzie element a_{kk} macierzy A. Jest to również ten element przez który dzielimy chcąc obliczyć mnożnik ψ_{ik} (k oraz i są opisane tak samo jak w poprzednim paragrafie). Częściowy wybór polega na wybraniu największego elementu co do wartości bezwzględnej z danej kolumny k oraz zamianie wierszy tak aby ten element stał się elementem głównym – elementem na diagonali. Możemy to zrobić, ponieważ macierz A jest nieosobliwa.
Zamiana wierszy macierzy jest kosztownym procesem, który może być zastąpiony przez wektor permutacji wierszy p. Zapamiętujemy w nim, na którym miejscu aktualnie znajduje się dany wiersz. Zatem jedyna różnica w samym algorytmie będzie taka, że zamiast odwoływać się do danego wiersza bezpośrednio, będziemy się odwoływać do jego pozycji zapisanej w wektorze permutacji. Zwróćmy uwagę na to, że zmieniając wiersze w macierzy A musimy zmieniać też poszczególne współrzędne w wekotrze prawych stron – b. Złożoność obu przedstawionych powyżej algorytmów wynosi $O(n^3)$.

Teraz kiedy znamy procedurę, która pozwala nam sprowadzić macierz A do macierzy trójkątnej górnej, pojawia się pytanie jak z tą wiedzą rozwiązać układ równań $Ax = b$. W tym celu zastosujemy algorytm podstawiania wstecz. Opiera się on o wzór poniżej.

$$x_i = \frac{b_i - \sum_{j=i+1}^{n}(a_{ij} \cdot x_j)}{a_{ii}}$$

Widać zatem, że idąc *od końca* ($i = n, \ldots, 1$) jesteśmy w stanie obliczyć wszystkie wartości x. Złożoność tego algorytmu wynosi $O(n^2)$, a cały proces prowadzący do rozwiązania równania ma złożoność $O(n^3)$ (bo eliminacja Gaussa ma złożoność $O(n^3)$).
Widać tu, że najkosztowniejszym fragmentem opisanego wyżej algorytmu rozwiązywania układu równań jest eliminacja Gaussa. Przedstawiony poniżej rozkład LU będzie umożliwiał rozwiązywanie układu równań dla różnych wektorów prawych stron przy tylko jednym wywołaniu metody eliminacji Gaussa zajmującej $O(n^3)$. Rozwiązywanie kolejnych układów równań z tą samą macierzą A będzie robione w czasie $O(n^2)$.

Zadanie 1

Celem tego zadania było napisanie funkcji rozwiązującej układ $\boldsymbol{Ax} = \boldsymbol{B}$ metodą eliminacji Gaussa uwzględniającej specyficzną postać macierzy $\boldsymbol{A}$ dla dwóch wariantów:

a) bez wyboru elementu głównego,

b) z częściowym wyborem elementu głównego.

Znając wyżej opisane algorytmy należało zmodyfikować je pod opisaną w treści listy zadań strukturę. Wiemy, że rozwiązanie naszego równania z użyciem niezmodyfikowanych algorytmów ma złożoność $O(n^3)$. Celem jest zrobienie tego samego w czasie $O(n)$. Zakładamy tu, że rozmiar bloku ℓ jest stałą. Inne założenie opisane wyżej to założenie, że dostęp do elementu macierzy (Sparse Arrays) jest w czasie stałym.

Zastosowane modyfikacje

Macierz $\boldsymbol{A}$ jest macierzą rzadką o specyficznej blokowo-trójdiagonalnej strukturze. Oznacza to, że większość elementów w danej kolumnie nie będzie wymagało zerowania. Znana i uporządkowana struktura pozwala nam na ograniczenie ilości wykonywanych operacji, gdyż są one wymagane tylko dla pewnego ograniczonego obszaru macierzy $\boldsymbol{A}$. Wyznaczymy teraz te ograniczenia. W celu przedstawienia metody rozważmy macierz $\boldsymbol{A}$ dla $\ell = 4$ i jakiegoś n.

$$\boldsymbol{A} = \begin{pmatrix}
a_{11}^1 & a_{12}^1 & a_{13}^1 & a_{14}^1 & c_{11}^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
a_{21}^1 & a_{22}^1 & a_{23}^1 & a_{24}^1 & 0 & c_{22}^1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
a_{31}^1 & a_{32}^1 & a_{33}^1 & a_{34}^1 & 0 & 0 & c_{33}^1 & 0 & 0 & 0 & 0 & 0 & \cdots \\
a_{41}^1 & a_{42}^1 & a_{43}^1 & a_{44}^1 & 0 & 0 & 0 & c_{44}^1 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & b_{13}^2 & b_{14}^2 & a_{11}^2 & a_{12}^2 & a_{13}^2 & a_{14}^2 & c_{11}^2 & 0 & 0 & 0 & \cdots \\
0 & 0 & b_{23}^2 & b_{24}^2 & a_{21}^2 & a_{22}^2 & a_{23}^2 & a_{24}^2 & 0 & c_{22}^2 & 0 & 0 & \cdots \\
0 & 0 & b_{33}^2 & b_{34}^2 & a_{31}^2 & a_{32}^2 & a_{33}^2 & a_{34}^2 & 0 & 0 & c_{33}^2 & 0 & \cdots \\
0 & 0 & b_{43}^2 & b_{44}^2 & a_{41}^2 & a_{42}^2 & a_{43}^2 & a_{44}^2 & 0 & 0 & 0 & c_{44}^2 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & b_{13}^3 & b_{14}^3 & a_{11}^3 & a_{12}^3 & a_{13}^3 & a_{14}^3 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & b_{23}^3 & b_{24}^3 & a_{21}^3 & a_{22}^3 & a_{23}^3 & a_{24}^3 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & b_{33}^3 & b_{34}^3 & a_{31}^3 & a_{32}^3 & a_{33}^3 & a_{34}^3 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & b_{43}^3 & b_{44}^3 & a_{41}^3 & a_{42}^3 & a_{43}^3 & a_{44}^3 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$

Kiedy przyjrzymy się czterem (ℓ) pierwszym kolumnom, widzimy, że musimy w nich wyzerować kolejno 3, 2, 1+4, 0+4 elementów. Zatem pętla która *idzie* po wierszach danej kolumny będzie mogła zakończyć pracę wcześniej. W naszym przypadku na rysunku, w przypadku kolumny pierwszej, pętla będzie mogła zakończyć pracę już po 3 iteracjach, ponieważ wszystkie elementy poniżej a_{41}^1 są już zerami. Ogólnie jeśli przyjrzymy się $\ell - 2$ pierwszym kolumnom to możemy zauważyć, że jedyne elementy niezerowe występują w bloku $\boldsymbol{A}_1$, a więc tylko w ℓ pierwszych wierszach. Rozpatrując kolejne ℓ kolumn widać, że niezerowe elementy będą się znajdować w bloku $\boldsymbol{B}_2$ lub $\boldsymbol{C}_1$, lub w bloku $\boldsymbol{A}_2$, a więc poniżej tych bloków, czyli poniżej wiersza o numerze 2ℓ, nie pojawią się żadne elementy niezerowe. Biorąc kolejne ℓ kolumn sytuacja jest podobna – elementy niezerowe wystąpią **najniżej** w bloku $\boldsymbol{B}_3$ lub $\boldsymbol{A}_3$, a więc nie niżej niż wiersz o numerze 3ℓ licząc od góry (3ℓ pierwszych wierszy). Patrząc na przedstawioną wyżej zależność możemy wywnioskować wzór na ostatni element niezerowy w danej kolumnie k.

$$ostatni_{wiersz}(k) = min\{\ell + \ell \times \lfloor \frac{k+1}{\ell} \rfloor, n\}, k - \text{indeks kolumny}$$

Kolejna obserwacja to fakt, że podczas odejmowania wiersza z elementem głównym od innego wiersza nie musimy obliczać tych elementów dla których wiersz główny ma zera. Niech wiersz k będzie naszym wierszem głównym (czyli tym zawierającym element główny), a obecnie eliminowany element będzie w wierszu i-tym. Oznacza to, że wykonujemy działanie na wierszach $R_i \leftarrow R_i - \psi_{ik} \cdot R_k$, gdzie R_i i R_k to odpowiednio i-ty i k-ty wiersz, a ψ_{ik} to mnożnik (przedstawiony w opisie podstawowej metody eliminacji Gaussa). Teraz widać, że dla j-tego elementu wiersza R_k będącego zerem, nazwijmy go $r_j^k = 0$, wykonanie powyższego działania nie zmieni odpowiedniej wartości j-tego elementu wiersza R_i. Wynika z tego, że pętla która wykonuje to działanie na wierszach może skończyć wcześniej pracę. Teraz wyprowadzimy wzór na ostatnią kolumnę, która może zawierać niezerowy element.

Zauważmy, że ostatnie elementy w wierszach to te leżące na diagonali bloków $\boldsymbol{C}$. Są one odległe od diagonali macierzy $\boldsymbol{A}$ o dokładnie ℓ. Zatem ostatni element niezerowy w wierszu w będzie się znajdował w kolumnie o indeksie $min\{\ell + w, n\}$. Podsumowując:

$$ostatnia_{kolumna}(w) = min\{\ell + w, n\}, w - \text{indeks wiersza}$$

Udało się zatem określić ograniczenia pętli w metodzie eliminacji Gaussa, które znacznie zredukują liczbę wykonywanych operacji. Kolejny etap rozwiązywania układu równań to algorytm podstawiania wstecz, który również może zostać zmodyfikowany w celu redukcji wykonywanych operacji.
W algorytmnie tym sumujemy elementy w danym wierszu pomnożone przez pewne liczby. My jednak wiemy, że znaczna część z tych elementów jest zerami, a więc wyniki iloczynów które otrzymamy też będą zerami nie zmianiając tym wartości sumy. Ponadto wiemy na jakiej pozycji występuje ostatni element niezerowy. Jest to zależność wyprowadzona powyżej tzn. wzór na $ostatnia_{kolumna}$. Możemy zatem zakończyć pętlę wcześniej redukując tym liczbę operacji.
Poniżej znajduje się pseudokod omówionego algorytmu w celu podsumowania naszych rozważań.

Algorytm 1: Zmodyfikowana metoda eliminacjii Gaussa

Dane wejściowe:

$\boldsymbol{A}$ – macierz z zadania o opisanej w zadaniu strukturze,
$\boldsymbol{b}$ – wektor prawych stron,
n – rozmiar macierzy $\boldsymbol{A}$,
ℓ – rozmiar bloku macierzy $\boldsymbol{A}$.

Dane wyjściowe:

$\boldsymbol{x}$ – wektor zawierający rozwiązania układu $\boldsymbol{Ax} = \boldsymbol{b}$.

function `eliminacja_gaussa(`$\boldsymbol{A}$, $\boldsymbol{b}$, n, ℓ`)`

- **for** $\mathsf{k} \leftarrow 1$ **to** $n-1$ **do**
  - $ostatni_{wiersz} \leftarrow \min\{\ell + \ell \cdot \lfloor \frac{\mathsf{k}+1}{\ell} \rfloor, n\}$
  - $ostatnia_{kolumna} \leftarrow \min\{\mathsf{k}+\ell, n\}$
  - **for** $\mathsf{i} \leftarrow \mathsf{k}+1$ **to** $ostatni_{wiersz}$ **do**
    - **if** $\boldsymbol{A}[\mathsf{k},\mathsf{k}] = 0$ **then**
      - **error** współczynnik na przekątnej równy zero
    - $\psi \leftarrow \frac{\boldsymbol{A}[\mathsf{i},\mathsf{k}]}{\boldsymbol{A}[\mathsf{k},\mathsf{k}]}$
    - $\boldsymbol{A}[\mathsf{i},\mathsf{k}] \leftarrow 0$
    - **for** $\mathsf{j} \leftarrow \mathsf{k}+1$ **to** $ostatnia_{kolumna}$ **do**
      - $\boldsymbol{A}[\mathsf{i},\mathsf{j}] \leftarrow \boldsymbol{A}[\mathsf{i},\mathsf{j}] - \psi \cdot \boldsymbol{A}[\mathsf{k},\mathsf{j}]$
    - $\boldsymbol{b}[\mathsf{i}] \leftarrow \boldsymbol{b}[\mathsf{i}] - \psi \cdot \boldsymbol{b}[\mathsf{k}]$
- **for** $\mathsf{i} \leftarrow n$ **downto** 1 **do**
  - $ostatnia_{kolumna} \leftarrow \min\{\mathsf{i}+\ell, n\}$
  - **for** $\mathsf{j} \leftarrow \mathsf{k}+1$ **to** $ostatnia_{kolumna}$ **do**
    - $\mathsf{suma} \leftarrow \mathsf{suma} + \boldsymbol{x}[\mathsf{i}] \cdot \boldsymbol{A}[\mathsf{i},\mathsf{j}]$
  - $\boldsymbol{x}[\mathsf{i}] \leftarrow \frac{\boldsymbol{b}[\mathsf{i}]-\mathsf{suma}}{\boldsymbol{A}[\mathsf{i},\mathsf{i}]}$
- **return** $\boldsymbol{x}$

Pozostało jeszcze określenie złożoności powyższego algorytmu. Widać, że pierwsza pętla wykona się $n-1$ ($O(n)$) razy, a pierwsza wewnątrz niej maksymalnie 2ℓ razy. Kolejna pętla wykonująca odejmowanie wierszy wykona się maksymalnie ℓ razy. ℓ jest stałą, zatem sama eliminacja Gausa wykonuje się w czasie $O(n)$. Teraz przyjrzyjmy się algorytmowi podstawiana wstecz. Zewnętrzna pętla wykona się $O(n)$ razy, a wewnętrzna co najwyżej ℓ, co daje łącznie $O(n)$. Zatem złożoność całego przedstawionego wyżej algorytmu rozwiązującego układ równań $\boldsymbol{Ax} = \boldsymbol{b}$ wynosi $O(n)$.

Przedstawiony wyżej algorytm jest modyfikacją metody eliminacji Gaussa bez wyboru elementu głównego. Teraz przedstawimy modyfikację tej metody z częściowym wyborem elementu głównego. Zasadę działania tej metody omówiliśmy wyżej w sekcji *Metoda eliminacji Gaussa z częsciowym wyborem elementu głównego*. Znajduje się tam opis jak efektywnie zamieniać wiersze i jak przechowywać informację o permutacji wierszy. Algorytm będzie zatem analogiczny. Wprowadzone zostanie wybieranie elementu głównego oraz zamiana wierszy za pomocą wektora permutacji.
Modyfikacje zastosowane w Algorytmie 1 wymagają jednak zweryfikowania. Podczas pracy naszego nowego algorytmu dochodzi do zamiany wierszy, więc w momencie odejmowania wierszy istnieje możliwość pojawienia się nowych elementów niezerowych będących dalej niż $ostatnia_{kolumna}$. Widać zatem, że nasze ograniczenie $ostatnia_{kolumna}$ wymaga zdefiniowania na nowo. Ograniczenie to dotyczy również modyfikacji algorytmu podstawiania wstecz. Przyjrzyjmy się teraz postaci macierzy $\boldsymbol{A}$ i zobaczmy jaki jest najdalszy możliwy indeks kolumny, dla którego możemy wygenerować niezerowy element w zadanym wierszu. Zauważmy, że w momencie eliminowania współczynników z $\ell - 2$ pierwszych kolumn, najdalszy niezerowy współczynnik może zostać wygenerowany na 2ℓ-tej pozycji. (kolumnie o indeksie 2ℓ). Współczynnik ten zostaje wygenerowany poprzez odjęcie wielokrotności wiersza głównego od wiersza ℓ-tego. Podczas eliminowania elementów z kolejnych ℓ kolumn, niezerowy element może zostać wygenerowany

najdalej w kolumnie o indeksie 3ℓ, poprzez odejmowanie wiersza głównego od wiersza 2ℓ-tego. Zatem widzimy, że *zasięg*, w którym mogą się pojawić elementy niezerowe zwiększył się o ℓ. Zatem mamy nowy wzór:

$$ostatnia_{kolumna}(w) = min\{2\ell + w, n\}, w - \text{indeks wiersza}$$

Oszacowanie na $ostatni_{wiersz}$ pozostaje takie same. Pozostaje oszacowanie złożoności tego algorytmu. Pojawiła się nowa pętla wyliczająca element główny, ale jej złożoność zależy tylko od ℓ, a zatem $O(1)$. Jedna z pętli w algorytmie podstawiania wstecz zwiększyła swoją maksymalną liczbę iteracji o ℓ, ale nie zmienia to złożoności całego algorytmu. Zatem łączna złożoność przedstawionego algorytmu 2 wynosi $O(n)$.

Algorytm 2: Zmodyfikowana metoda eliminacjii Gaussa z częściowym wyborem elementu głównego

Dane wejściowe:

$\boldsymbol{A}$ – macierz z zadania o opisanej w zadaniu strukturze,
$\boldsymbol{b}$ – wektor prawych stron,
n – rozmiar macierzy $\boldsymbol{A}$,
ℓ – rozmiar bloku macierzy $\boldsymbol{A}$.

Dane wyjściowe:

$\boldsymbol{x}$ — wektor zawierający rozwiązania układu $\boldsymbol{Ax} = \boldsymbol{b}$.

function `eliminacja_gaussa_częściowy_wybór(`$\boldsymbol{A}$, $\boldsymbol{b}$, n, ℓ`)`

$\pi \leftarrow \{\mathsf{i} : \mathsf{i} \in \{1, \ldots, n\}\}$
for $\mathsf{k} \leftarrow 1$ **to** $n - 1$ **do**
  $ostatni_{wiersz} \leftarrow \min\{\ell + \ell \cdot \lfloor \frac{\mathsf{k}+1}{\ell} \rfloor, n\}$
  $ostatnia_{kolumna} \leftarrow \min(2\ell + k, n)$
  for $\mathsf{i} \leftarrow \mathsf{k} + 1$ **to** $ostatni_{wiersz}$ **do**
    $r_{\max} \leftarrow \mathsf{m}$ takie, że: $\boldsymbol{A}[\pi[\mathsf{m}], \mathsf{k}] = \max(|\boldsymbol{A}[\pi[\mathsf{q}], \mathsf{k}]| : \mathsf{q} \in \{\mathsf{i}, \ldots, ostatni_{wiersz}\})$
    if $\pi[r_{\max}] = 0$ **then**
      error podana macierz jest osobliwa
    `swap` $(\pi[\mathsf{k}], \pi[r_{\max}])$
    $\psi \leftarrow \frac{\boldsymbol{A}[\pi[\mathsf{i}],\mathsf{k}]}{\boldsymbol{A}[\pi[\mathsf{k}],\mathsf{k}]}$
    $\boldsymbol{A}[\pi[\mathsf{i}], \mathsf{k}] \leftarrow 0$
    for $\mathsf{j} \leftarrow \mathsf{k} + 1$ **to** $ostatnia_{kolumna}$ **do**
      $\boldsymbol{A}[\pi[\mathsf{i}], \mathsf{j}] \leftarrow \boldsymbol{A}[\pi[\mathsf{i}], \mathsf{j}] - \psi \cdot \boldsymbol{A}[\pi[\mathsf{k}], \mathsf{j}]$
    $\boldsymbol{b}[\pi[\mathsf{i}]] \leftarrow \boldsymbol{b}[\pi[\mathsf{i}]] - \psi \cdot \boldsymbol{b}[\pi[\mathsf{k}]]$
for $\mathsf{i} \leftarrow n$ **downto** 1 **do**
  $ostatnia_{kolumna} \leftarrow \min\{2\ell + \pi[\mathsf{i}], n\}$
  for $\mathsf{j} \leftarrow \mathsf{k} + 1$ **to** $ostatnia_{kolumna}$ **do**
    $\mathsf{suma} \leftarrow \mathsf{suma} + \boldsymbol{x}[\mathsf{j}] \cdot \boldsymbol{A}[\pi[\mathsf{i}], \mathsf{j}]$
  $\boldsymbol{x}[\mathsf{i}] \leftarrow (\boldsymbol{b}[\pi[\mathsf{i}]] - \mathsf{suma})/\boldsymbol{A}[\pi[\mathsf{i}], \mathsf{i}]$
return $\boldsymbol{x}$

Opis rozkładu LU

Ideą eliminacji Gaussa było sprowadzenie macierzy $\boldsymbol{A}$ (z równania $\boldsymbol{Ax} = \boldsymbol{b}$) do macierzy trójkątnej górnej, ponieważ taki układ jesteśmy w stanie rozwiązać stosunkowo prosto. Idea rozkładu LU jest taka sama i jest ściśle powiązana z metodą eliminacji Gaussa. Jest to metoda służąca do rozwiązywania równania $\boldsymbol{Ax} = \boldsymbol{b}$. Metoda polega na przedstawieniu macierzy $\boldsymbol{A}$ w postaci iloczynu dwóch macierzy trójkątnych: macierzy $\boldsymbol{L}$ (lower) oraz macierzy $\boldsymbol{U}$ (upper). Macierz $\boldsymbol{L}$ jest macierzą trójkątną dolną, a macierz $\boldsymbol{U}$ macierzą trójkątną górną. Mając taki rozkład jesteśmy w stanie zapisać nasz układ równań w postaci $\boldsymbol{LUx} = \boldsymbol{b}$. Rozwiązanie tego układu sprowadza się do rozwiązania dwóch równań z macierzami trójkątnymi (górną i dolną):

$$\begin{cases} \boldsymbol{Lz} = \boldsymbol{b} \\ \boldsymbol{Ux} = \boldsymbol{z} \end{cases}$$

Ważnym założeniem jest to, że jedna z macierzy (L lub U) ma na diagonali same jedynki. Zapewnia to jednoznaczność rozkładu. Rozkład LU jest szczególnie użyteczny w przypadku rozwiązywania wielu układów równań z tą samą macierzą $\boldsymbol{A}$ i ze zmieniającym się wektorem prawych stron. Mając raz zrobiony rozkład LU (koszt $O(n^3)$) możemy kolejne równania rozwiązywać z kosztem $O(n^2)$. Inną zaletą rozkładu LU jest to, że całość procesu można wykonać tylko na macierzy $\boldsymbol{A}$ – nie potrzebna jest dodatkowa pamięć, gdyż zarówno macierz L jak i U są zapamiętywane na jednej macierzy.
W celu otrzymania macierzy $\boldsymbol{L}$ i $\boldsymbol{U}$ wystarczy wykonać eliminację Gaussa na macierzy $\boldsymbol{A}$. Macierz trójkątna powstała w wyniku eliminacji Gaussa staje się macierzą $\boldsymbol{U}$. Macierz $\boldsymbol{L}$ powstaje w wyniku zapamiętania mnożników użytych w algorytmie eliminacji Gaussa. Stąd widać, że złożoność wykonania rozkładu LU to $O(n^3)$. Daje to jednak możliwość rozwiązywania kolejnych układów równań (takich w których macierz $\boldsymbol{A}$ pozostaje taka sama) z kosztem $O(n^2)$.

Zadanie 2

Celem tego zadania było napisanie funkcji wyznaczającej rozkład LU macierzy $\boldsymbol{A}$ metodą eliminacji Gaussa uwzględniającą specyficzną postać macierzy $\boldsymbol{A}$ dla dwóch wariantów:

a) bez wyboru elementu głównego,

b) z częściowym wyborem elementu głównego.

Zastosowane modyfikacje
Funkcja wyznaczająca rozkład LU będzie drobną modyfikacją **algorytmu 1** (dla wariantu a) oraz **algorytmu 2** (dla wariantu b). Żadne ograniczenia nie zmieniają się. Pierwsza różnica polega na tym, że zamiast przypisywać zera w obszar, w którym znajdowały się wyeliminowane współczynniki, algorytm będzie przypisywał mnożnik. Zatem linia algorytmu 1 zamiast „$\boldsymbol{A}[i,k] \leftarrow 0$” będzie miała postać: „$\boldsymbol{A}[i,k] \leftarrow \psi$”. Linia „$\boldsymbol{A}[i,k] \leftarrow 0$” w algorytmie 2 zamieni się na: „$\boldsymbol{A}[\pi[i],k] \leftarrow \psi$”. Druga różnica to usunięcie z obu algorytmów linii, która dokonuje modyfikacji wektora prawych stron – $\boldsymbol{b}$, ponieważ tej modyfkacji będzie dokonywał algorytm rozwiązujący równanie na podstawie rozkładu LU. Wyliczanie wartości x również zostanie zrobione w oddzielnym algorytmie.
Pozostało określić złożoność przedstawionych wyżej algorytmów. Algorytm 1 oraz algorytm 2 miały złożoność $O(n)$ (przy założeniu, że ℓ jest stałą). Dwa wyżej opisane algorytmy będą miały taką samą złożoność, ponieważ są to w istocie takie same algorytmy różniące się jedynie kilkoma linijkami, modyfikacje zastosowane do zmienionych linii kodu nie zmieniają złożoności całości algorytmu.

Zadanie 3

Celem zadania było napisanie funkcji rozwiązującej układ równań $\boldsymbol{Ax} = \boldsymbol{b}$ (uwzględniającą specyficzną postać macierzy $\boldsymbol{A}$) jeśli wcześniej został już wyznaczony rozkład LU przez funkcje z poprzedniego zadania.
Algorytm będzie się składał z rozwiązania dwóch układów równań.

$$\begin{cases} \boldsymbol{Lz} = \boldsymbol{b} \\ \boldsymbol{Ux} = \boldsymbol{z} \end{cases}$$

Rozwiązanie pierwszego z nich tzn. $\boldsymbol{Lz} = \boldsymbol{b}$ wykonamy algorytmem analogicznym do algorytmu podstawiania wstecz – nazwijmy go algorytmem podstawiania wprzód. Algorytm ten zaczyna się od pierwszego wiersza i w każdej iteracji oblicza kolejne wartości współrzędnych wektora $\boldsymbol{z}$ korzystając z coraz dłuższych wierszy. Intuicyjnie jest to przejście przez macierz trójkątną dolną zaczynające się od pierwszego elementu wiersza, a kończące się na ostatnim elemencie ostatnigo wiersza. Ilość iteracji możemy tu ograniczyć wyznaczając ograniczenia w podobny sposób jak zrobiliśmy to dla algorytmu podstawiania wstecz. Zauważmy, że elemety zerowe w macierzy L występują na tych samych pozycjach co elementy zerowe w początkowej macierzy $\boldsymbol{A}$. Zatem sumowanie wszystkich kolumn poniżej diagonali w danym wierszu w nie jest konieczne, ponieważ część z nich jest zerami. Możemy zacząć od kolumny, która może zawierać jakiś element niezerowy. Wyznaczymy teraz ograniczenie na najmniejszy indeks kolumny, który może zawierać elementy niezerowe w zadanym wierszu w. Przeprowadzając rozumowanie analogiczne do tego przy wyznaczaniu $ostatni_{wiersz}$ dla algorytmu podstawiania wstecz możemy wyprowadzić wzór na $poczatkowa_{kolumna}$:

$$poczatkowa_{kolumna}(w) = max\{\ell \times \lfloor \frac{w-1}{\ell} \rfloor, 1\}, w - \text{indeks wiersza}$$

Rozwiązanie drugiego równania wykonamy znanym nam już algorytmem podstawiania wstecz.
Algorytm rozwiązujący układ równań z podanego rozkładu LU wykonanego metodą eliminacji Gaussa bez wyboru elementu głównego znajduje się poniżej.

Algorytm 3: Algorytm rozwiązujący układ równań przy użyciu rozkładu LU (bez wyboru elementu głównego)

Dane wejściowe:

$\boldsymbol{A}$	–	macierz z zadania o opisanej w zadaniu strukturze, po przekształceniu do postaci, w której pod diagonalą znajdują się elementy macierzy L, a nad diagonalą elementy macierzy U,
$\boldsymbol{b}$	–	wektor prawych stron,
n	–	rozmiar macierzy $\boldsymbol{A}$,
ℓ	–	rozmiar bloku macierzy $\boldsymbol{A}$.

Dane wyjściowe:

$\boldsymbol{x}$	—	wektor zawierający rozwiązania układu $\boldsymbol{Ax} = \boldsymbol{b}$.

function `rozwiąż_z_LU(`$\boldsymbol{A}$, $\boldsymbol{b}$, n, ℓ`)`

- **for** i $\leftarrow 1$ **to** n **do**
  - suma $\leftarrow 0$
  - $poczatkowa_{kolumna} \leftarrow \max\{\ell \cdot \lfloor \frac{\text{i}-1}{\ell} \rfloor, 1\}$
  - **for** j $\leftarrow poczatkowa_{kolumna}$ **to** i $- 1$ **do**
    - suma $\leftarrow$ suma $+$ z[j] $\cdot \boldsymbol{A}$[i, j]
  - z[i] $= \boldsymbol{b}$[i] $-$ suma
- **for** i $\leftarrow n$ **downto** 1 **do**
  - suma $\leftarrow 0$
  - $ostatnia_{kolumna} \leftarrow \min\{\text{i} + \ell, n\}$
  - **for** j $\leftarrow$ i $+ 1$ **to** $ostatnia_{kolumna}$ **do**
    - suma $\leftarrow$ suma $+ \boldsymbol{x}$[j] $\cdot \boldsymbol{A}$[i, j]
  - $\boldsymbol{x}$[i] $\leftarrow$ (z[i] $-$ suma)$/\boldsymbol{A}$[i, i]
- **return** $\boldsymbol{x}$

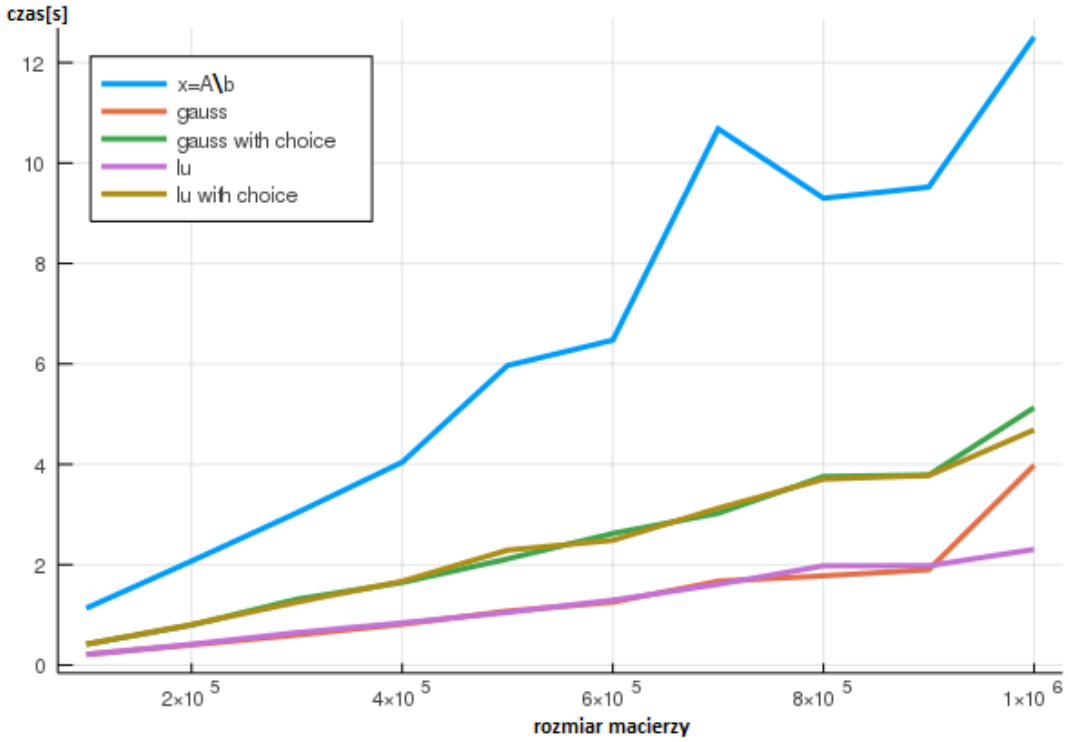
Algorytm ten wykona się w czasie $O(n)$, ponieważ pierwsza pętla (algorytm podstawiania wprzód) wykona się co najwyżej n razy, a pętla wewnątrz niej w czasie $O(1)$, oczywiście jeśli zakładamy, że ℓ jest stałą! Druga pętla to znany nam zmodyfikowany na potrzeby zadania algorytm podstawiania wstecz, dla którego udowodniliśmy już złożoność liniową.

Pozostała jeszcze kwestia analogicznego algorytmu dla rozkładu LU wykonanego metodą eliminacji Gaussa z częściowym wyborem elementu głównego. Będzie to algorytm identyczny z tą różnicą, że zamiast odwoływać się bezpośrednio do indeksów wierszy macierzy L i U oraz indeksów wektora $\boldsymbol{b}$, będziemy odwoływać się do ich indeksów zapamiętanych w wektorze permutacji. Złożoność tego algorytmu będzie oczywiście taka sama – $O(n)$.

Wyniki i wnioski

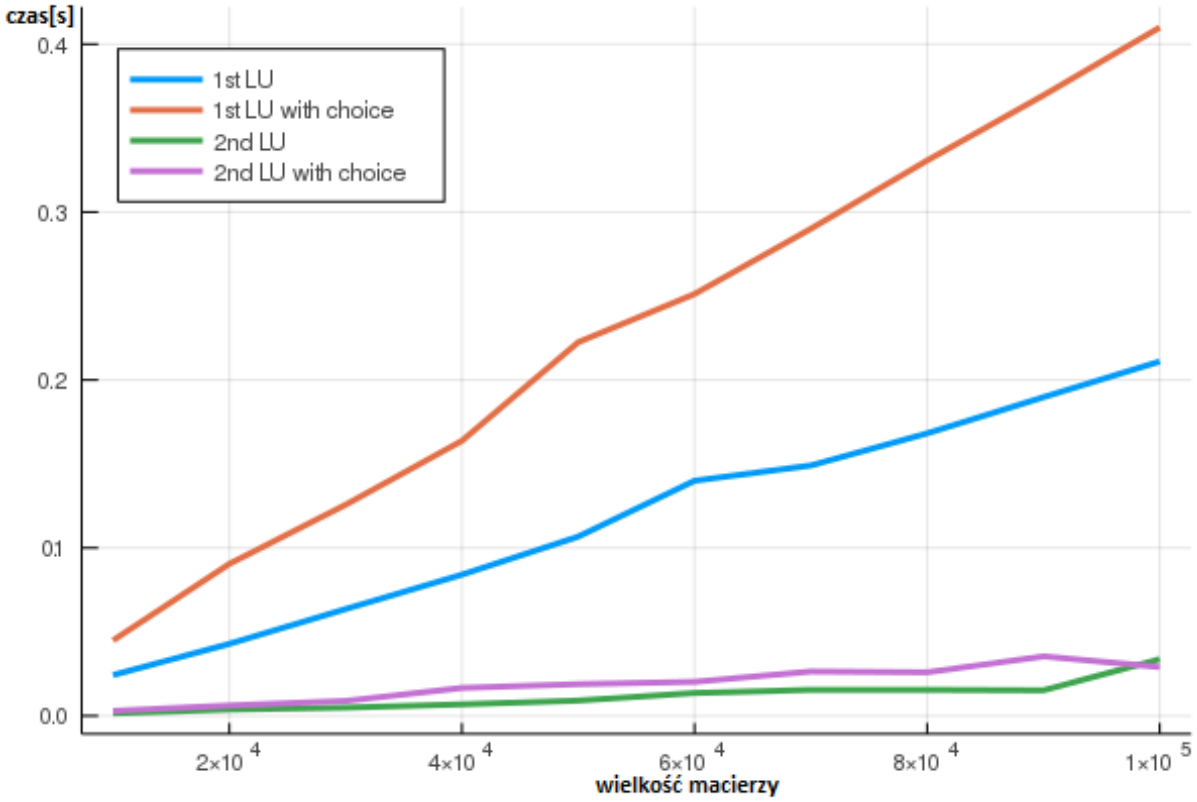
Programy testujące jakie napisałem znajdowały poprawne rozwiązanie dla przykładowych danych (dane od prowadzącego kurs) dla wszystkich algorytmów rozwiązywania układu równań $\boldsymbol{Ax} = \boldsymbol{b}$. Wektory $\boldsymbol{x}$ dla tych danych były wektorami jednostkowymi, stąd łatwo było sprawdzić poprawność algorytmów.

Wykres zależności czasu działania poszczególnych algorytmów dla losowych macierzy blokowo - trójprzekątniowych (wskaźnik uwarunkowania wynosił 10, $\ell = 10$) od wielkości tych macierzy znajduje się poniżej. Zauważmy, że dzięki efektywnemu przechowywaniu macierzy byliśmy w stanie przechować macierz rzędu 10^6 (komputer, na którym przeprowadzono eksperyment posiada 8GB pamięci RAM).



Wykres zależności czasu działania poszczególnych algorytmów od wielkości macierzy

Widać, że standardowa implementacja metody eliminacji Gaussa w języku Julia (niebieski wykres) wypada najsłabiej. Inna obserwacja to fakt, że algorytmy bez wyboru elementu głównego (pomarańczowy i różowy) okazały się znacznie szybsze niż ich odpowiedniki, które wykonywały częściowy wybór elementu głównego (zielony i smutny żółty). Widać też, ze rozkład Gaussa (dla obu wariantów) działał w bardzo podobnym czasie co rozkład LU. Można z tego wyciągnąć wniosek, że rozkład Gaussa powinniśmy robić tylko w sytuacji jeśli jesteśmy całkowicie pewni, że będziemy musieli rozwiązać tylko jeden układ równań z daną macierzą. W innym przypadku warto zrobić rozkład LU, gdyż kosztuje nas to tylko niewiele więcej, a w przypadku konieczności rozwiązania innego układu równań gdzie zmieni się tylko wektor prawych stron, jesteśmy w stanie zrobić to już znacznie mniejszym nakładem pracy. Wykres pokazujący czasy rozwiązywania układu równań za pomocą rozkładu LU w zależności od wielkości macierzy znajduje się poniżej.



Wykres zależności czasu rozwiązywanie układu równań za pomocą rozkładu LU od wielkości macierzy dla przypadku gdzie zrobiono to pierwszy raz oraz dla przypadku gdzie zrobiono to znając już rozkład LU danej macierzy.

Widać, że drugie (w praktyce oczywiście też kolejne) wywołanie funkcji rozwiązującej układ równań liniowych ze znanym już rozkładem LU zajęło znacznie mniej czasu niż rozwiązanie pierwsze wymagające jeszcze dodatkowo obliczenia rozkładu LU. Patrząc na wykresy można wyciągnąć wniosek, że to właśnie obliczanie rozkładu LU jest najbardziej kosztownym etapem algorytmu. Na tym wykresie widać również fakt zaobserwowany na poprzednim wykresie – algorytmy, które nie wykonują wyboru elementu głównego są szybsze.

Dla wszystkich algorytmów zostały również obliczone błędy względne w przypadku kiedy wektor prawych stron $\boldsymbol{b}$ był wyliczany. Macierze dla jakich wykonano eksperymenty miały wskaźnik uwarunkowania równy 10.0, a $\ell = 4$. Wyniki znajdują się w poniższej tabeli.

rozmiar macierzy	gauss	gauss w wyborem
3000	$4.298823120065619 \cdot 10^{-15}$	$5.574749567941993 \cdot 10^{-16}$
4000	$3.941408230930561 \cdot 10^{-15}$	$5.558576358655353 \cdot 10^{-16}$
7000	$6.760775100733844 \cdot 10^{-15}$	$5.392236603514981 \cdot 10^{-16}$
10000	$6.265367149198456 \cdot 10^{-15}$	$5.370519558488765 \cdot 10^{-16}$

rozmiar macierzy	lu	lu z wyborem
3000	$2.292966452680386 \cdot 10^{-14}$	$4.63979368199035 \cdot 10^{-16}$
4000	$1.9944709360505984 \cdot 10^{-14}$	$4.576590595628056 \cdot 10^{-16}$
7000	$1.5751008487442212 \cdot 10^{-14}$	$4.667824409854897 \cdot 10^{-16}$
10000	$1.6235415592478852 \cdot 10^{-14}$	$4.582962454682111 \cdot 10^{-16}$

Widać, że algorytmy wykonujące częściowy wybór zwróciły dokładniejsze wyniki. Błąd względny w ich przypadku był rzędu 10^{-16}. Dla porównania algorytmy, które nie wykonywały wyboru elementu głównego miały błędy rzędu 10^{-15} dla metody eliminacji Gaussa oraz 10^{-14} dla rozkładu LU.

Pierwszym wnioskiem z całej listy jest to, że zaimplementowane algorytmy faktycznie miały złożoność liniową (potwierdzają to również wykresy). Kolejny wniosek to fakt, że algorytmy dokonujące częściowego wyboru elementu głównego okazały się mieć większą złożoność, ale też zwracały dokładniejsze wyniki w odróżnieniu od algorytmów nie wykonujących wyboru. Zwróćmy też uwagę na fakt, że metody bez wyboru elementu głównego nie zadziałają w momencie kiedy na diagonali pojawi się element zerowy, zatem nie zawsze będzie możliwe ich użycie. Kolejny wniosek dotyczy rozkładu LU. Okazał się on bowiem wydajniejszy w przypadku kiedy rozwiązujemy wiele układów równań w których zmianie ulega tylko wektor prawych stron. Kolejny wniosek to fakt, że dostosowanie reprezentacji danych pod konkretną ich strukturę może dać dużo większe możliwości. W tym przypadku była to możliwość przechowania macierzy rzędu nawet 10^6 bez żadnego problemu. Standardowa reprezentacja macierzy nie dałaby takiej możliwości. Ponadto dzięki reprezentacji dobranej pod strukturę macierzy, ilość użytej pamięci była znacznie mniejsza. Ostatni wniosek to fakt, że dopasowanie algorytmów pod konkretną strukturę danych może znacznie przyspieszyć algorytm. W tym przypadku (również dzięki odpowiedniej reprezentacji macierzy w pamięci komputera) udało się *zejść* ze złożoności $O(n^3)$ do złożoności liniowej, co jest znaczną poprawą.