Alogorytm eliminacji Gaussa dla specyficznych macierzy rzadkich

Lista 5 na laboratoria do kursu Obliczenia naukowe

Patryk Rygiel, 250080

Grudzień 2020

1 Wstęp

Celem zadania było opracowanie algorytmu rozwiązującego układ $Ax = b$, gdzie A - macierz, b - wektor prawych stron, przy pomocy eliminacji Gaussa z uwzględnieniem specyficznej postaci macierzy A (zostanie ona opisana w kolejnej sekcji). Rozwiązanie zadania zawiera:

- Algorytm eliminacji Gaussa bez wyboru elementu głównego oraz z cześciowym wyborem
- Algorytm wyznaczania rokładu LU metodą eliminacji Gaussa bez wyboru elementu głównego oraz z częściowym wyborem
- Algorytm rozwiązujący równanie $Ax = b$ przy użyciu wyznaczonego rozkładu LU

W ramach zadania zaproponoway został własny sposób pamiętania tylko elementów niezerowych macierzy - uwzględniający ich specyficzną charakterystykę. Macierze nie są pamiętane przy użyciu struktury **Sparse-Array** oferowanej przez język **Julia**.

2 Specyfikacja macierzy A

Macierz $A \in \mathbb{R}^{n \times n}$ jest macierzą rzadką i blokową następującej postaci:

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

Rysunek 1: Struktura macierzy A

$v = n/l$, zakładamy, że n jest podzielne przez l, gdzie l jest rozmiarem wszystkich kwadratowych macierzy wewnętrznych: A_k, B_k i C_k. Mianowicie, $A_k \in \mathbb{R}^{l\times l}$, $k = 1, ..., v$ jest macierzą gęstą, **0** jest kwadratową macierzą zerową stopnia l, macierz $B_k \in \mathbb{R}^{l\times l}$, $k = 2, ..., v$ jest następującej postaci:

$$\boldsymbol{B_k} = \begin{pmatrix} 0 & \cdots & 0 & b_1^k \\ 0 & \cdots & 0 & b_2^k \\ \vdots & & \vdots & \vdots \\ 0 & \cdots & 0 & b_\ell^k \end{pmatrix},$$

Rysunek 2: Struktura podmacierzy B

B_k ma tylko jedną, ostatnią, kolumnę niezerową. Natomiast $C_k \in \mathbb{R}^{l\times l}$, $k = 1, ..., v-1$ jest macierzą diagonalną:

$$\boldsymbol{C_k} = \begin{pmatrix} c_1^k & 0 & 0 & \cdots & 0 \\ 0 & c_2^k & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c_{\ell-1}^k & 0 \\ 0 & \cdots & 0 & 0 & c_\ell^k \end{pmatrix}.$$

Rysunek 3: Struktura podmacierzy C

3 Metoda pamiętania macierzy w pamięci

Ze względu na możliwą dużą wielkość macierzy A, musi być ona efektywnie trzymana w pamięci. W tym celu została opracowana specyficzna metoda, która pozwala efektywnie przechowywać macierz oraz jej budowa współgra z operacjami wykonywanymi w eliminacji Gaussa. Macierz A jest trzymana w pamięci jako czwórka:

- **n** - wielkość macierzy kwadratowej A,
- **l** - wielkość macierzy blokowych: A_k, B_k i C_k,
- **data** - lista długości n zawierająca wiersze macierzy A,
- **shift** - wektor długości n zawierający dla każdego wiersza macierzy jego przesunięcie względem pierwszej kolumny

Lista **data** nie pamięta całych wierszy macierzy A ze względu na niską wydajność takiego rozwiązania, o którym mowa była już wcześniej. Analizując pierwsze l wierszy macierzy A można zauważyć, że są one postaci: $(A_1, C_1, 0, ..., 0)$. Na podstawie tej obserwacji tworzymy pierwsze l wierszy w strukturze w następujący sposób:

- Do listy **data** dodajemy l wektorów długości $2l$. Wektor k-ty składa się z k-tego wiersza macierzy A_1 oraz k-tego wiersza macierzy C_1.
- Do wektora **shift** dodajemy na pierwszych l miejscach liczbę 0, gdyż przesunięcie odpowiadających wierszy w liście **data** względem pierwszej kolumny macierzy A jest równe 0.

Po wykonaniu tego kroku lista **data** oraz wektor **shift** wyglądają następująco (przykład dla $n = 16$ i $l = 4$):

- **data** = $\begin{pmatrix} a_1^1 & a_2^1 & a_3^1 & a_4^1 & c_1^1 & 0 & 0 & 0 \\ a_5^1 & a_6^1 & a_7^1 & a_8^1 & 0 & c_2^1 & 0 & 0 \\ a_9^1 & a_{10}^1 & a_{11}^1 & a_{12}^1 & 0 & 0 & c_3^1 & 0 \\ a_{13}^1 & a_{14}^1 & a_{15}^1 & a_{16}^1 & 0 & 0 & 0 & c_4^1 \end{pmatrix}$
- **shift** = $(0, 0, 0, 0)$

Analizując następne wiersze macierzy A można zauważyć, że są one postaci: $(0, ..., 0, B_k, A_k, C_k, 0, ..., 0)$. Wiemy, że macierze B mają wypełnioną tylko ostatnią kolumnę zatem nie ma potrzeby zapamiętywania całej tej macierzy, a jedynie jej ostatnią kolumnę. Na podstawie tej obserwacji i operacji, którą przeprowadziliśmy dla pierwszych l wierszy, analogicznie wypełniamy kolejne $n - 2l$ wierszy (ostatnie l wierszy wypełnione zostanie w inny sposób):

- Do listy **data** dodajemy $v - 2$ bloki l wektorów długości $2l + 1$. Blok i-ty odpowiada blokowi wierszy w macierzy A, który zawiera podmacierze A_i, B_i oraz C_i. Zatem w każdym bloku mamy l wierszy, które dodajemy do listy **data**. Są one długości $2l + 1$, gdyż są postaci: $(b_k^i, a_{(k-1)\cdot l+1}^k, a_{(k-1)\cdot l+2}^k, ..., c_k^i, ...)$ - k-ty wiersz macierzy A_i, C_i oraz element b_k^i.
- Do wektora **shift** dodajemy $n-2l$ wartości. Dla każdego wiersza z i-tego bloku wierszy macierzy A, do wektora **shift** dodawana jest wartość odpowiadająca przesunięciu tego wiersza względem pierwszej kolumny. Ze względu na to, że podmacierz B_k ma wypełnioną tylko ostatnią kolumnę, dla wszystkich wierszy będących w k-tym bloku wartość przesunięcia będzie wynosić $((k-1)\cdot l)-1$, gdyż na lewo od macierzy B_k znajduje się $k-2$ zerowych macierzy **0**, które są wielkości $l \times l$. Dodatkowo nie ma potrzeby pamiętać pierwszych $l-1$ wierszy macierzy B_k, dlatego dodatkowo przesunięcie powiększamy o $l-1$

Po wykonaniu tego kroku lista **data** oraz wektor **shift** wyglądają następująco (przykład dla $n = 16$ i $l = 4$):

- **data** = $\begin{pmatrix} a_1^1 & a_2^1 & a_3^1 & a_4^1 & c_1^1 & 0 & 0 & 0 & \\ a_5^1 & a_6^1 & a_7^1 & a_8^1 & 0 & c_2^1 & 0 & 0 & \\ a_9^1 & a_{10}^1 & a_{11}^1 & a_{12}^1 & 0 & 0 & c_3^1 & 0 & \\ a_{13}^1 & a_{14}^1 & a_{15}^1 & a_{16}^1 & 0 & 0 & 0 & c_4^1 & \\ b_1^2 & a_1^2 & a_2^2 & a_3^2 & a_4^2 & c_1^2 & 0 & 0 & 0 \\ b_2^2 & a_5^2 & a_6^2 & a_7^2 & a_8^2 & 0 & c_2^2 & 0 & 0 \\ b_3^2 & a_9^2 & a_{10}^2 & a_{11}^2 & a_{12}^2 & 0 & 0 & c_3^2 & 0 \\ b_4^2 & a_{13}^2 & a_{14}^2 & a_{15}^2 & a_{16}^2 & 0 & 0 & 0 & c_4^2 \\ b_1^3 & a_1^3 & a_2^3 & a_3^3 & a_4^3 & c_1^3 & 0 & 0 & 0 \\ b_2^3 & a_5^3 & a_6^3 & a_7^3 & a_8^3 & 0 & c_2^3 & 0 & 0 \\ b_3^3 & a_9^3 & a_{10}^3 & a_{11}^3 & a_{12}^3 & 0 & 0 & c_3^3 & 0 \\ b_4^3 & a_{13}^3 & a_{14}^3 & a_{15}^3 & a_{16}^3 & 0 & 0 & 0 & c_4^3 \end{pmatrix}$
- **shift** = $(0, 0, 0, 0, 3, 3, 3, 3, 7, 7, 7, 7)$

Ostatnim krokiem jest dodanie ostatnich l wierszy macierzy A. W tym wypadku ów wiersze są postaci: $(0, ..., 0, B_v, A_v)$. Zatem mając na uwadze wcześniejsze spostrzeżenia, ostatnie l wierszy listy **data** wypełnimy wektorami długości $l+1$, gdyż macierz A_v ma wielkość $l \times l$, a dla macierzy B_v wystarczy pamiętać tylko element w ostatniej kolumnie. Wektor **shift** wypełniamy analogicznie jak we wcześniejszym kroku.

Zatem ostatecznie po wszystkich trzech krokach struktura trzymająca macierz A wygląda następująco (przykład dla $n = 16$ i $l = 4$):

- **n** = 16,
- **l** = 4,
- **data** =
$$\begin{pmatrix}
a_1^1 & a_2^1 & a_3^1 & a_4^1 & c_1^1 & 0 & 0 & 0 & \\
a_5^1 & a_6^1 & a_7^1 & a_8^1 & 0 & c_2^1 & 0 & 0 & \\
a_9^1 & a_{10}^1 & a_{11}^1 & a_{12}^1 & 0 & 0 & c_3^1 & 0 & \\
a_{13}^1 & a_{14}^1 & a_{15}^1 & a_{16}^1 & 0 & 0 & 0 & c_4^1 & \\
b_1^2 & a_1^2 & a_2^2 & a_3^2 & a_4^2 & c_1^2 & 0 & 0 & 0 \\
b_2^2 & a_5^2 & a_6^2 & a_7^2 & a_8^2 & 0 & c_2^2 & 0 & 0 \\
b_3^2 & a_9^2 & a_{10}^2 & a_{11}^2 & a_{12}^2 & 0 & 0 & c_3^2 & 0 \\
b_4^2 & a_{13}^2 & a_{14}^2 & a_{15}^2 & a_{16}^2 & 0 & 0 & 0 & c_4^2 \\
b_1^3 & a_1^3 & a_2^3 & a_3^3 & a_4^3 & c_1^3 & 0 & 0 & 0 \\
b_2^3 & a_5^3 & a_6^3 & a_7^3 & a_8^3 & 0 & c_2^3 & 0 & 0 \\
b_3^3 & a_9^3 & a_{10}^3 & a_{11}^3 & a_{12}^3 & 0 & 0 & c_3^3 & 0 \\
b_4^3 & a_{13}^3 & a_{14}^3 & a_{15}^3 & a_{16}^3 & 0 & 0 & 0 & c_4^3 \\
b_1^4 & a_1^4 & a_2^4 & a_3^4 & a_4^4 & & & & \\
b_2^4 & a_5^4 & a_6^4 & a_7^4 & a_8^4 & & & & \\
b_3^4 & a_9^4 & a_{10}^4 & a_{11}^4 & a_{12}^4 & & & & \\
b_4^4 & a_{13}^4 & a_{14}^4 & a_{15}^4 & a_{16}^4 & & & &
\end{pmatrix}$$
- **shift** = $(0, 0, 0, 0, 3, 3, 3, 3, 7, 7, 7, 7, 11, 11, 11, 11)$

Zużycie pamięciowe takiej struktury jest równe $O(n \cdot l) + O(n)$. Jeżeli traktujemy l jako stałą, zużycie pamięciowe redukuje się do $O(n)$ co jest dużą poprawą nad pamiętaniem macierzy w postaci gęstej, które jest $O(n^2)$.

Na wybór takiej struktury do trzymania macierzy A w pamięci, miał także wpływ charakter operacji wykonywanych w algoyrmtie eliminacji Gaussa. Permutacje wierszy wykonywane przy częściowym wyborze elemntu głównego w zaproponowanej strukturze wykonywane sa w czasie stałym $O(1)$ ze względu na proste zamienienie indeksów w liście **data** oraz wektorze **shift**. Więcej szczgeółów na temat adaptacji algorytmu eliminacji Gaussa do specyficznej stkruktury macierzy A, jest zawarte w następnych sekcjach.

4 Algorytm eliminacji Gaussa

4.1 Konstrukcja algorytmu uwzględniającego specyfikę macierzy A

Klasyczny algorytm eliminacji Gaussa rozwiązuje równanie $Ax = b$ ze złożonością $O(n^3)$. Wymaga on przejścia po każdym wierszu macierzy A i wyzerowaniu kolumny pod obecnym elementem przekątniowym w celu uzyskania macierzy górno trójkątnej:

$$A^{(1)}x = b^{(1)} \begin{pmatrix} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + ... + a_{1n}^{(1)}x_n = b_1^{(1)} \\ a_{21}^{(1)}x_1 + a_{22}^{(1)}x_2 + ... + a_{2n}^{(1)}x_n = b_2^{(1)} \\ . \\ . \\ . \\ a_{n1}^{(1)}x_1 + a_{n2}^{(1)}x_2 + ... + a_{nn}^{(1)}x_n = b_n^{(1)} \end{pmatrix}$$

Eliminacja zmiennej x_1 z równań 2-go do n-tego wymaga przemnożenia równania pierwszego przez:

$l_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}$ dla $i = 2, ...n$

Oraz odjęcie go od pozostałych wierszy poniżej. Identyczną operację należy wykonać dla każdego wiersza macierzy A.

Zauważmy, że odejmowany wiersz jest mnożony przez czynnik mający w liczniku wartość pod obecnym elementem przekątniowym. Analizując strukturę macierzy A zauważamy, że dla każdego elementu przekątniowego są pod nim maksymalnie l elementy nie zerowe (gdy pod elementem przekątniowym jest ostatnia kolumna macierzy B_k jest dokładnie l, w inny wypadku tylko elementy macierzy A_k czyli $< l$), gdyż podmacierz A_k ma wielkość $l \times l$ oraz macierz B_k ma zerowe wszystkie elementy poza ostatnią kolumną. W takim razie nie ma potrzeby wykonywania odejmowania wiersza k od każdego wiersza poniżej, a jedynie od maksymalnie l kolejnych. Dzięki takiej optymalizacji, uznając l za stałą sam proces eliminacji Gaussa zmniejsza się do $O(n^2)$.

Kolejną obserwacją jaką można poczynić jest zauważenie, że w wierszach także występują elementy zerowe zatem po przemnożeniu przez niezerowy czynnik l_{ik} są one dalej zerowe. W związku z tym od elementów w wierszach poniżej, w tych samych kolumnach co elementy zerowe w obecnym wierszu, odejmujemy zero. Odejmowanie zera nie zmienia odjemnej zatem możemy tę operację pominąć. O pomijanie tej operacji dba zaproponowana struktura trzymania macierzy, która w liście **data** przechowuje dla k-tego wiersza tylko elementy z podmacierzy B_j, A_j oraz C_j. Jak już zostało wyjaśnione powyżej wiersze w liście **data** są maksymalnie długości $2l + 1$, zatem potrzebne jest wykonanie dokładnie tylu odejmowań dla każdego wiersza poniżej. W związku z tym uznając l za stała, oraz biorąc pod uwagę wcześniejszą optymalizację, złożoność eliminacji Gaussa zmniejszamy do $O(n)$.

Po wykonaniu tyhc kroków otrzymujemy macierz górną trójkątnej na podstawie, której konieczna jest ekstrakcja wektora wynikowego. W klasycznej implementacji taka ekstrakcja ma złożoność $O(n^2)$. Przebiega ona następująco:

$$\begin{pmatrix} u_{11}x_1 + u_{12}x_2 + ... + u_{1n}x_n = b_1 \\ u_{22}x_2 + ... + u_{2n}x_n = b_2 \\ . \\ . \\ . \\ u_{nn}x_n = b_n \end{pmatrix}$$

Mając powyższy układ równań, wynikający bezpośrednio z postaci macierzy górno trójkątnej, rozwiązujemy rózwnanie $Ux = b$. Z ostatniego równania wyznaczamy x_n

$x_n = \frac{b_n}{u_{nn}}$

Następnie wyznaczamy x_k dla $k = n - 1, ..., 1$:

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

Zauważmy, że obliczając x_k obliczamy sumę, w której każdy czynnik ma postać element wiersza k razy już wyliczony element wektora wyjściowego x. Ponownie korzystając ze struktury macierzy A, wiemy, że k-ty wiersz ma maksymalnie $2l + 1$ elementów nie zerowych. Zatem sumowanie wystarczy, że się odbędzie od elementu $k + 1$ do ostatniego elementu w k-tym wierszu z listy **data**. w taki sposób dla każdego wyliczenia x_k wykonane zostanie maksymalnie $2l + 1$ dodawań. Zatem traktując l jako stałą, złożoność ekstrakcji rozwiązania równania $Ux = b$, gdzie U jest macierzą górno trójkątną, obniża się do $O(n)$.

Podsumowując oba kroki potrzebne do rozwiązania równania $Ax = b$, traktując l jako stałą, otrzymujemy złożoność czasową równą $O(n) + O(n)$ czyli $O(n)$. Zatem zaproponowany algorytm działa w czas liniowym od wielkości macierzy A przy stałym parametrze l.

Dostosowanie klasycznej eliminacji Gaussa do zaproponowanej struktury macierzy, jest łatwym zadaniem, ze względu na proste wyznaczenie elementu a_{ij}, do którego możemy się odwołać stosując wektor **shift**:

a_{ij} = `A.data[i][j - shift[i]]`

4.2 Częściowy wybór elementy głównego

Częściowy wybór elementu głównego polega na zamianie miejscami obecnego wiersza z innym wierszem znajdującym się poniżej, jeżeli element przekątniowy iwersza obecnego jest mniejszy bezwzględenie od elementu, który został by elementme przekątniowym po zamianie. Taka operacja sprawia, że na przekątnej występują największe elementy co zapewnia dokładniejsze działania, ze względu na dzielenie przez element przekątniowy - gdy jet on bliski zeru, wyniki tracą na dokładności. Częściowy wybór elementu głównego dla eliminacji Gaussa ma złożoność $O(n^2)$, gdyż dla każdego wiersza nalezy sprawdzić każdy wiersz pod nim aby wyznaczyć, w którym wierszu zawiera się maksimum kolumny w celu zamiany.

Tak jak zauważyliśmy już poprzednio, dla danego wiersza pod elementem przekątniowym macierzy A znajduje się maksymalnie l elementów nie zerowych. Zatem wystarczy dokonać tylko porównania pomiędzy tymi wierszami w celu wyznaczenia potencjalnej zamiany wierszy. Ponownie uznając l za stałą, złożoność częściowego wyboru elementu głównego maleje do $O(n)$.

Zamiana wierszy w zaproponowanej strukturze jest prosta do wykonania. Wystarczy zamienić miejscami wiersze listy **data** oraz wektora **shift** - odbywa się to w czasi stałym $O(1)$. Ważnym aspektem implementacyjnym jest rozpatrzenie przypadku, gdy k-ty wiersz zostanie zamieniony z wierszem o innej wartości przesunięciu - inna wartość w wektorze **shift**, zamiana wierszy pomiędzy blokami. W takim wypadku potrzebne jest rozszerzenie wiersza k o dodatkowe l elementów zerowych na końcu, gdyż będą w tych miejscach wykonywane operacje podczas eliminacji Gaussa na kolejnym bloku. Taka operacja wymaga dynamicznej alokacji elementów, jednak jest dosyć wydajnym rozwiązaniem i może się pojawić maksymalnie $v - 1$ razy.

4.3 Podsumowanie

Poprzez użycie specyficznej stuktury do przechowywania macierzy A oraz dostosowanie algorytmu eliminacji Gaussa, możliwe jest zmniejszenie jego złożoności do $O(n)$, oczywiście przy założeniu, że l jest stałe. Jest to możliwe zarówno bez wyboru elementu głównego jak i z wyborem częściowym. Dodatkowo zaproponowana stuktura trzymania macierzy A w pamięci, pozwala w naturlany sposób przenieść klasyczny algorytm eliminacji Gaussa na wersję zoptymalizowaną do specyfiki zadania.

5 Algorytm wyznaczania macierzy LU

5.1 Konstrukcja algorytmu uwzględniającego specyfikę macierzy A

Algorytm wyznaczenia rozkładu **LU** mozna wykonać przy użyciu klasycznego algorytmu eliminacji Gaussa o złożoności $O(n^3)$. Rozkład **LU** mozna trzymać efektywnie jako jedną macierz, gdyż macierz **L** jest dolno trójkątna o przekątnej złożonej z samych jedyenk, natomiast **U** jest macierzą górno trójkątną - zatem w miejscu przekątnej trzymana jest przekątna macierzy **U**, a w pozostałych miejscach nie kolidujące ze sobą elementy obu macierzy.

Ponownie należy przejść po każdym wierszu macierzy i podzielić kolumnę pod elementem przekątniowym przez ów element. A dla pozostałych elementów wyliczyć uzupełnienie Schura:

- $a_{ik} = a_{ik}/a_{kk}$, dla $i > k$,
- $a_{ij} = a_{ij} - \sum_{j=k+1}^{n} a_{ik}a_{kj}$, dla $i > k$,

Bazując na analizie wykonanej dla klasycznej eliminacji Gaussa, możemy zauważyć, że ilość wierszy, dla których należy wykonać powyższe operację dla wiersz k-tego można ograniczyć przez l, a ilość sumowanych czynników w uzupełnieniu Schura można ograniczyć prze $2l + 1$. Zatem analogicznie jak przy optymalizacji eliminacji Gauss, dekompozycja do rozkładu **LU** może zostać wykonana w czasie $O(n)$, gdy l jest stałą.

Ekstrakcja wektora wynikowego x przy użyciu rozkładu **LU** wymaga policzenia dwóch równań: $Ly = b$ oraz $Ux = y$ przy jednym równaniu $Ax = b$ w klasycznej eliminacji Gaussa. Jednak już przy eliminacji Gauss wskazana została optymalizacja wyliczania wektora wynikowego x w czasie liniowym $O(n)$. Zatem używając tego samego algorytmu dwukrotnie do wyliczenia obu równań otrzymujemy wektor wynikowy x w czasie $O(2n) = O(n)$.

5.2 Częściowy wybór elementy głównego

Częsciowy wybór elementu głównego przy stosowaniu rozkładu **LU** nie różni się niczym od tego stosowanego w eliminacji Gaussa, zatem stosując tą samą optymalizację jesteśmy w stanie wykonać go w czasie $O(n)$. Jedyną różnicą jest to, że dla rozkładu **LU** konieczne jest zwrócenie wektora permutacji aby wiadome było jak nalezy spermutować wektor b. Jednak nie powoduje to żadnych dodatkowych obliczeń, gdyż w eliminacji Gaussa permutowany był wektor b natomiast przy rozkładzie **LU** te same operacje wykonujemy na wektorze permutacji P.

5.3 Podsumowanie

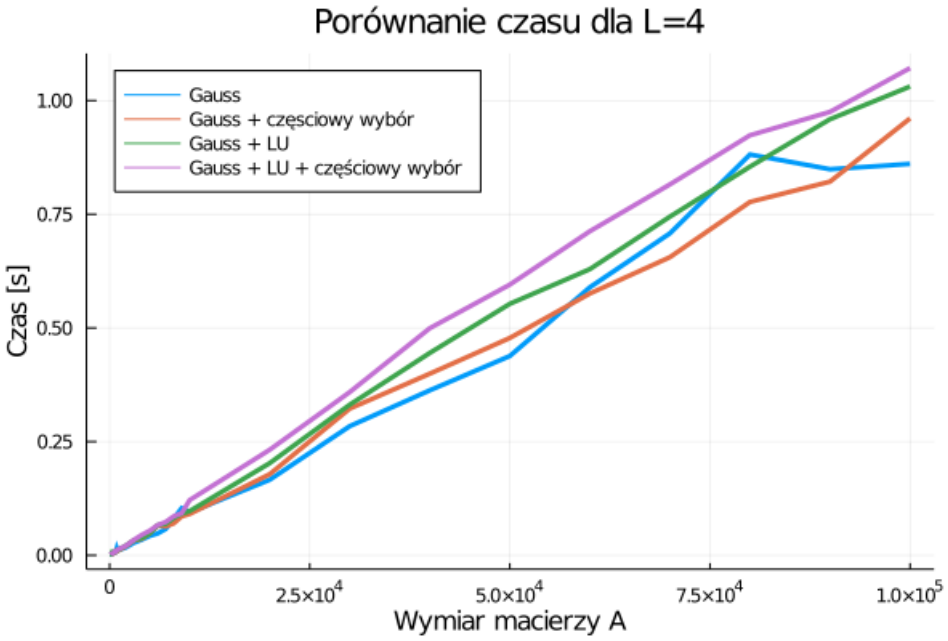
Stosując te same optymalizacje i strukturę przechowywania macierzy A w pamięci, możliwe jest zmniejszenie złożoności algorytmu dokonującego dekompozycji do macierzy **LU** oraz wyliczającego równanie $Ax = b$, zarówno z cześciowym wyborem elementu głównego jak i bez, do $O(n)$.

6 Wyniki

W ramach porównywania metod zostały wykonane eksperymenty na macierzach o wielkościach:
$100, 200, ..., 900, 1000, 2000, ..., 9000, 10000, 20000, ..., 90000, 100000$, generowanych przy użyciu modułu **matrixgen.jl** napisanego przed Dr. Zielińskiego. Wszytskie macierze zostały wygenerowane z tymi samymi pozostałymi parametrami: $l = 4$ oraz $ck = 10$. Dla każdej macierzy wykonano 20 powtórzeń dla każdego algorytmu. W każdej iteracji liczony był czas działania algorytmu oraz jego błąd względny (przeprowadzone eksperymenty zamieszczone są w **Jupyter Notebook**, Benchmark.ipynb).

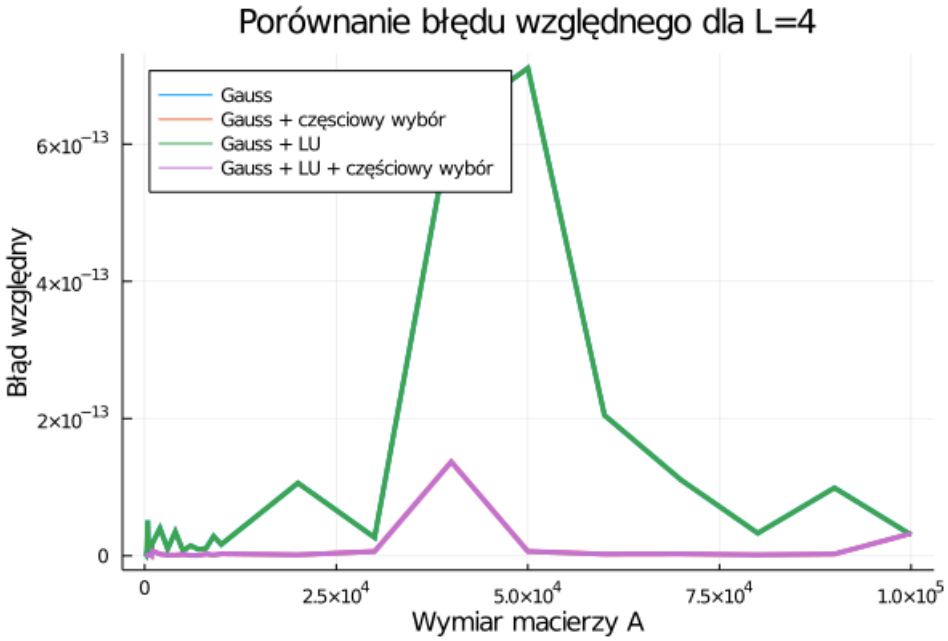
Analizując figurę (4) przedstawiającą złożoność czasową wszystkich algorytmów przy stałej wartości l można zauważyć, że mają one złożoność liniową $O(n)$ - w porównaniu do złożoności $O(n^3)$ klasycznego algorytmu eliminacji Gaussa. Dla porównania został wykonany także wbudowany w języku **Julia** algorytm eliminacji Gaussa dla $n = 50000$. Ze względu na nie obsługiwanie przez ten algorytm specyficznej stuktura pamiętania macierzy, do trzymania jej w pamieci uzyto modułu **SparseArray**. Czas wykonywania algorytmu wyniósł 90 sekund - zatem zastosowane algorytmy i stuktura trzymania macierzy A w pamięci, ewidentnie poprawiły wydajność algorytmu.

Porównując 4 metody do obliczania równania $Ax = b$, można zauważyć, że metody nie korzystające z rozkładu **LU** osiągają lepsze rezultaty czasowe. Jest spodziewane zachowanie, gdyż w obu algorytmach cześć odpowiadająca za eliminację Gaussa wykoywana jest jednakową ilość razy, natomiast ekstrakcja wektora x przy korzystaniu z rokładu **LU** wymaga rozwiązania dwóch równań $Ly = b$ oraz $Ux = y$ przy jednym równaniu $Ax = b$ w klasycznej eliminacji Gaussa. Zaletą rozkładu **LU** jest to, że pozwala on przyśpieszyć wykonywanie eliminacji Gaussa, gdy jedyną zmienna jest wektor b, gdyż wymagane jest wtedy tylko jednorazowe wyznaczenie rozkładu. W przypadku klasycznej eliminacji Gaussa niezależnie czy zmieni się wektor b czy macierz A, konieczne jest wykonanie całego algorytmu od początku.



Rysunek 4: Czas działania algorytmu wraz ze wzrostem wielkości macierzy A

Metody korzystające z częściowego wyboru elementu głównego z reguły wykonywały się wolniej niż ich odpowiednicy bez wyboru elementu (wokół $n = 7.5 \times 10^4$ widoczny jest wzrost czasu dla algorytmu eliminacji Gaussa bez częściowego wyboru elementu, jednak jest to raczej zaburzenie nie mające nic wspólnego z samym algorytmem). Takie zachowanie jest także spodziewane, gdyż wybór elementu głównego wymaga dodatkowego sprawdzenia wierszy poniżej i potencjalnej ich permutacji. Jest to przydatna operacja, gdy zależy nam na minimalizacji błędu względnego jednak wymaga większej ilości operacji za czym idzie jej większy koszt czasowy.



Rysunek 5: Wartość błędu względnego wraz ze wzrostem wielkości macierzy A

Analizując figurę (5) przedstawiającą błędy względne wykonane przez wszystkie algorytmy, należy zaznaczyć, że wykres odpowiadający metodom bez wyboru elementu głównego pokrywa się zarówno dla klasycznej eliminacji Gaussa jak i dla tej używającej rozkładu **LU**, to samo dzieje się dla sytuacji bez wyboru elementu głównego. Powodem takiej sytuacji jest liczenie dokładnie tych samych wartości tylko w inny sposób bez użycia elementu głównego, oraz wykonaniem dokładnie tych samych permutacji przy metodach z wyborem elementu głównego.

Metody uzywające częściowego wyboru ewidentnie generują dużo niższy błąd oraz są stabilniejsze jeśli chodzi o wartość błędu - mniejsze skoki na wykresie. Metody bez wyboru elementu głównego natomiast, nie minimalizuja tak dobrze błędu względnego oraz zachowują się niestabilniej - są dużo bardziej zależne od danych. Ponownie jest to spodziewane zachowanie, gdyz powodem stosowania częściowego wyboru elementu głównego jest właśnie minimalizacja błędu względnego poprzez zapobieganie dzieleniu przez wartości bliskie zeru - na przekątnej umieszcza się jak najwyższe wartości.

7 Podsumowanie

Dla pewnych danych tradycyjnie stosowane algorytmy mogą nie działać wystarczająco optymalnie zarówno w aspekcie czasowym jak i pamięciowym. Znając specyfikację problemu możliwe jest zaadoptowanie i zoptymalizowanie istniejącego algorytmu i struktur danych w celu rozwiązywania problemu dużo efektywniej. Tak jak widoczne było w tym zadaniu, tradycyjny algorytm elminacji Gaussa ma złożoność $O(n^3)$ co dla dużych n z jakimi przyszło pracować w tym zadaniu, staje się bardzo nieefektywny biorąc pod uwagę rzadkość i blokowość macierzy A. W takim wypadku poprzez analizę zagadnienia i specyfikacji, możliwe jest obniżenie złożoności do $O(n)$, oczywiście przy pewnych poczynionych założeniach l - stała, oraz zoptymalizowanie samego sposobu przechowywania macierzy w pamięci w celu obsługi danych, których przy tradycyjnym podejściu nie dałoby się przetworzyć.