Sprawozdanie z listy nr 5 Obliczenia Naukowe

Marek Świergoń (261750)

7 stycznia 2022, PWr, WIiT INA

0 Wstęp: przedstawienie problemu

Na wstępie warto sformułować problem, który będziemy próbowali rozwiązać z wykorzystaniem zaimplementowanych metod. W ramach tej listy potrzebujemy sprawnie i dokładnie rozwiązać układ równań postaci $Ax = b$, gdzie $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, $n \geqslant 4$. W rozwiązywaniu pomoże nam specyficzna postać macierzy A; macierz A jest macierzą blokową, którą można zapisać następująco:

$$A = \begin{bmatrix} A_1 & C_1 & 0 & 0 & 0 & \dots & 0 \\ B_2 & A_2 & C_2 & 0 & 0 & \dots & 0 \\ 0 & B_3 & A_3 & C_3 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & B_{v-2} & A_{v-2} & C_{v-2} & 0 \\ 0 & \dots & 0 & 0 & B_{v-1} & A_{v-1} & C_{v-1} \\ 0 & \dots & 0 & 0 & 0 & B_v & A_v \end{bmatrix},$$

gdzie $v = \frac{n}{l}$, $A_k \in \mathbb{R}^{l\times l}$, $k = 1, \dots, v$ jest macierzą gęstą (wypełnioną niezerowymi elementami), $0 \in \mathbb{R}^{l\times l}$ to macierz zerowa stopnia l, $B_k \in \mathbb{R}^{l\times l}$, $k = 2, \dots, v$ jest postaci:

$$B_k = \begin{bmatrix} b^k_{1,1} & \dots & b^k_{1,l-2} & b^k_{1,l-1} & b^k_{1,l} \\ 0 & \dots & 0 & 0 & b^k_{2,l} \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & b^k_{l,l} \end{bmatrix},$$

czyli ma tylko pierwszy wiersz niezerowy i ostatnią kolumnę niezerową. Z kolei $C_k \in \mathbb{R}^{l\times l}$, $k = 1, \dots, v-1$ jest macierzą mającą niezerową tylko przekątną:

$$C_k = \begin{bmatrix} c^k_1 & 0 & 0 & \dots & 0 \\ 0 & c^k_2 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & c^k_{l-1} & 0 \\ 0 & \dots & 0 & 0 & c^k_l \end{bmatrix}.$$

Zauważmy, że n oznacza rozmiar całej macierzy A, zaś l to rozmiar jednego bloku wewnątrz macierzy blokowej A.

Z własności struktury macierzy A będziemy korzystali, dostosowując do niej algorytmy rozwiązywania układów równań, dzięki czemu znacząco zmniejszymy ich złożoności.

Aby zrozumieć usprawnienia wynikające ze struktury macierzy, najpierw należy zapoznać się z teorią stojącą za zastosowanymi metodami, tj. metodą eliminacji Gaussa oraz rozkładem LU; w następnym rozdziale zostało zapisane krótkie wprowadzenie teoretyczne wraz z opisem optymalizacji.

1 Metoda eliminacji Gaussa

1.1 Idea wariantu podstawowego metody

Metoda eliminacji Gaussa jest jedną z metod rozwiązywania układów równań lniowych postaci $Ax = b$. Podzielić ją można na dwa etapy: w pierwszym doprowadzamy macierz A do górnotrójkątnej, a w drugim rozwiązujemy układ równań z otrzymaną w etapie pierwszym macierzą A.

1.1.1 Etap pierwszy: przekształcanie macierzy

W tym etapie będziemy odejmowali kolejne wiersze przemnożone przez odpowiedni współczynnik od wierszy znajdujących się pod nimi, doprowadzając do wyzerowania wszystkich elementów pod przekątną macierzy A. Zauważmy, że układ $Ax = b$ możemy przedstawić następująco:

$$\begin{array}{ccccccccc} a_{1,1}^{(1)}x_1 & + & a_{1,2}^{(1)}x_2 & + & \dots & + & a_{1,n}^{(1)}x_n & = & b_1^{(1)} \\ a_{2,1}^{(1)}x_1 & + & a_{2,2}^{(1)}x_2 & + & \dots & + & a_{2,n}^{(1)}x_n & = & b_2^{(1)} \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n,1}^{(1)}x_1 & + & a_{n,2}^{(1)}x_2 & + & \dots & + & a_{n,n}^{(1)}x_n & = & b_n^{(1)} \end{array}$$

gdzie indeks górny oznacza numer kroku etapu pierwszego metody eliminacji Gaussa. Wyeliminujmy zmienną x_1 z równań od drugiego do n-tego. Robimy to, mnożąc pierwsze równanie przez $l_{i,1} = \frac{a_{i,1}^{(1)}}{a_{1,1}^{(1)}}$, dla $i = 2, \dots, n$ i odejmując przemnożone równanie od pozostałych.

Po tym kroku otrzymujemy następujący układ:

$$\begin{array}{ccccccccc} a_{1,1}^{(1)}x_1 & + & a_{1,2}^{(1)}x_2 & + & \dots & + & a_{1,n}^{(1)}x_n & = & b_1^{(1)} \\ 0 & + & a_{2,2}^{(2)}x_2 & + & \dots & + & a_{2,n}^{(2)}x_n & = & b_2^{(2)} \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ 0 & + & a_{n,2}^{(2)}x_2 & + & \dots & + & a_{n,n}^{(2)}x_n & = & b_n^{(2)} \end{array}$$

W kolejnych kolumnach postępujemy analogicznie, stąd po $k-1$ krokach mamy:

$$\begin{array}{ccccccccccc} a_{1,1}^{(1)}x_1 & + & a_{1,2}^{(1)}x_2 & + & \dots & & + & & a_{1,n}^{(1)}x_n & = & b_1^{(1)} \\ & & a_{2,2}^{(2)}x_2 & + & \dots & & + & & a_{2,n}^{(2)}x_n & = & b_2^{(2)} \\ & & & \ddots & & & & & \vdots & & \vdots \\ & & & & a_{k,k}^{(k)}x_k & + & \dots & + & a_{k,n}^{(k)}x_n & = & b_k^{(k)} \\ & & & & a_{k,k}^{(k)}x_k & + & \dots & + & a_{k,n}^{(k)}x_n & = & b_k^{(k)} \end{array}$$

Usuwamy x_k z równań od k+1-ego do n-tego, mnożąc k-te równanie przez $l_{i,k} = \frac{a_{i,k}^{(k)}}{a_{k,k}^{(1)}}$, dla $i = k+1, \dots, n$ i odejmując przemnożone równanie od równań pod nim.

Ostatecznie po $n-1$ krokach otrzymujemy układ równoważny pierwotnemu z macierzą górnotrójkątną. Zauważmy, że aby ten wariant metody eliminacji Gaussa działał poprawnie, elementy znajdujące się na przekątnej muszą być dalekie od zera. Wynika to ze wzoru na wyznaczane współczynniki, w którym te elementy (elementy główne) znajdują się w mianowniku. Dzielenie przez elementy bliskie zeru sprawia kłopoty z numerycznego punktu widzenia, stąd potrzeba bardziej zaawansowanej wersji metody z częściowym wyborem elementu głównego.

1.1.2 Etap drugi: rozwiązanie przekształconego układu

Rozwiązywanie układu równań z macierzą górnotrójkątną rozpoczynamy „od dołu". Wyliczamy x_n z ostatniego równania następująco:

$$x_n = \frac{b_n}{a_{n,n}}.$$

Mając x_n, możemy wyliczyć x_{n-1}, i tak dalej, korzystając ze wzoru:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} x_j}{a_{i,i}}$$

W ten sposób otrzymujemy rozwiązanie układu równań $Ax = b$.

1.2 Implementacja wariantu podstawowego i optymalizacje wynikające ze struktury macierzy; teoretyczna analiza złożoności

1.2.1 Złożoność obliczeniowa metody

Znając ideę standardowej wersji procesu eliminacji, możemy zauważyć, że złożoność wynosi $O(n^3)$. Wynika to z faktu, że w etapie pierwszym musimy w k-tej kolumnie wyzerować $(n - k - 1)$ elementów poprzez wyznaczenie współczynnika, przemnożenie go przez k-ty wiersz i odjęcie przemnożonego wiersza od tego, w którym chcemy wyzerować element. Odejmowanie od siebie wierszy jest procesem liniowym, a elementów do wyzerowanie mamy łącznie w najgorszym przypadku $O(\frac{1}{2}n^2)$, stąd złożoność $O(n^3)$. Etap drugi jest mniej złożony, patrząc wprost na wzory wyznaczania rozwiązania możemy określić złożoność tego etapu na $O(n^2)$.

Złożoność tę można znacząco zmniejszyć, korzystając z własności struktury macierzy A danej w zadaniu.

Po pierwsze, znacznemu zmniejszeniu ulega liczba elementów, którą trzeba wyzerować w etapie pierwszym; w każdej kolumnie mamy maksymalnie tyle niezerowych wartości pod przekątną, co rozmiar bloku l. Oczywiście nie ma sensu zerować elementów będących zerami, zatem znacząco ograniczyliśmy liczbę elementów do wyzerowania, do $l * (n - 1) = O(n)$, gdy l jest stałą.

Po drugie, przy odejmowaniu wierszy, nie potrzebujemy odejmować elementów zerowych. Wiersz elementu przekątnej, którym w danym momencie chcemy zerować elementy pod nim, ma już w danym kroku po lewej od tego elementu same zera, a po prawej od tego elementu ma $(l - 1)$ wartości mogących nie być zerami (ostatni element to przekątna macierzy blokowej C_k). Oznacza to, że odejmowanie od siebie wierszy jest tu procesem zależnym od l zamiast od n, zatem ma złożoność stałą, gdy l jest stałą.

Widać zatem, iż złożoność obliczeniowa etapu pierwszego to $O(l^2 * (n - 1)) = O(n)$, gdy $l = O(1)$.

W etapie drugim również możemy zmniejszyć złożoność, z $O(n^2)$ do liniowej. W każdym wierszu macierzy trójkątnej uzyskanej po etapie pierwszym mamy co najwyżej l niezerowych wartości, wzór z wersji niezoptymalizowanej można przekształcić tak:

$$x_i = \frac{b_i - \sum_{j=i+1}^{i+l} x_j}{a_{i,i}}$$

W związku z tym, wyznaczenie całego wektora rozwiązań ma złożoność $O(n)$, przy założeniu, że l jest stałą.

Etap pierwszy i drugi wykonywane są po kolei, dla każdego z nich wykazaliśmy złożoność liniową. Stąd dzięki wykorzystaniu struktury macierzy z zadania zredukowaliśmy złożoność z $O(n^3)$ do $O(n)$.

1.2.2 Złożoność pamięciowa metody

Metoda otrzymuje jako parametry:

- obiekt struktury `BlockMatrix`; w rozdziale 3 i 5 zobaczymy, że ma on rozmiar liniowy względem wielkości problemu,
- wektor prawych stron b, zajmujący $O(n)$ pamięci.

Dodatkowo metoda tworzy wektor rozwiązania x, także zajmujący $O(n)$ pamięci oraz iteratory pętli i zmienną pomocniczą l, zajmujące stałą ilość pamięci.

Widzimy zatem, że przy zoptymalizowanym przechowywaniu macierzy A złożoność pamięciowa to $O(n)$; gdybyśmy macierz przechowywali wprost w tablicy dwuwymiarowej, to złożoność pamięciowa wynosiłaby $O(n^2)$.

1.2.3 Uwagi do implementacji i pseudokod

W strukturze wykorzystywanej do przechowania macierzy i opisanej dokładnie w dalszej części sprawozdania zaimplementowano dwie metody ważne z punktu widzenia eliminacji Gaussa. Metoda `lastRow` zwraca ostatni od lewej wiersz mający niezerowy element w zadanej kolumnie. Analogicznie, metoda `lastColumn` zwraca dla danego wiersza indeks ostatniej od góry kolumny mającej w tym wierszu wartość niezerową. Metody te korzystają ze struktury macierzy - nie muszą przechodzić po całym wierszu/kolumnie by uzyskać indeks.

Produktem przedstawionej wcześniej idei wraz z optymalizacjami jest przedstawiony poniżej pseudokod metody `gaussElimination(A, b)` z pliku `blocksys.jl`.

Algorithm 1: `gaussElimination`

Data:
- A – macierz blokowa o strukturze zgodnej z zadaniem
- b – wektor prawych stron
- n – rozmiar macierzy
- l – rozmiar bloku

Result: x – wektor rozwiązania, dla którego $Ax = b$

begin
- //etap pierwszy
- **for** k ***from*** 1 ***to*** $n-1$ **do**
  - **for** i ***from*** $k+1$ ***to*** *A.lastRow(k)* **do**
    - $l = a_{i,k}/a_{k,k}$
    - $a_{i,k} = 0$ //zerowanie nie jest konieczne
    - **for** j ***from*** $k+1$ ***to*** *A.lastColumn(k)* **do**
      - $a_{i,j} = a_{i,j} - l \cdot a_{k,j}$
    - $b_i = b_i - l \cdot b_k$
- //etap drugi
- $x = b$
- $x_n = b_n/a_{n,n}$
- **for** i ***from*** $n-1$ ***down to*** *1* **do**
  - **for** j ***from*** $i+1$ ***to*** *A.lastColumn(i)* **do**
    - $x_i = x_i - a_{i,j} \cdot x_j$
  - $x_i = x_i/a_{i,i}$
- **return** x

1.3 Idea wariantu metody z częściowym wyborem elementu głównego

Przy wariancie podstawowym metody eliminacji Gaussa zauważyliśmy, że w etapie pierwszym (zerowanie elementów pod przekątną) elementy główne, tj. elementy znajdujące się na przekątnej macierzy A nie mogą być bliskie zeru. W przeciwnym razie rozwiązanie będzie obarczone dużym błędem numerycznym lub wręcz będzie niemożliwe do uzyskania ze względu na błąd dzielenia przez zero.

Aby pozbyć się małych wartości z przekątnej, możemy przestawiać wiersze macierzy A tak, by na przekątną trafiły jak największe elementy. Tę ideę realizuje częściowy wybór elementu głównego. Polega on na wybraniu takiego wiersza m, dla którego

$$|a_{m,k}| = \max_{k \leqslant i \leqslant n} |a_{i,k}|.$$

Element z tego wiersza będzie zastosowany jako element główny, tj. zostanie zamieniony z wierszem k i zastosowany do zerowania elementów znajdujących się pod przekątną po zamianie wierszy.

Poza tą modyfikacją metoda działa analogicznie jak ta w wariancie podstawowym.

1.4 Implementacja wariantu z częściowym wyborem z optymalizacjami; teoretyczna analiza złożoności

Przy implementacji eliminacji Gaussa z częściowym wyborem nie warto wprost przestawiać wierszy macierzy. Efektywniejsze jest zapamiętanie przestawień w tablicy permutacji P. W tej tablicy w k-tym elemencie znajduje się indeks początkowy wiersza, który po zamianach wierszy obecnie znajduje się na miejscu wiersza, który na początku był k-ty. Wtedy, gdy dla k-tego elementu głównego wybrany zostaje element z m-tego wiersza, to dochodzi do zamiany elementów $P[k]$ i $P[m]$.

1.4.1 Złożoność obliczeniowa metody

Zauważmy, że dostęp do indeksów wierszy po permutacjach wynikających z częściowego wyboru jest stały, nie ma zatem wpływu na asymptotyczną złożoność obliczeniową metody. Znajdowanie elementu głównego wykonywane jest w każdym z $(n-1)$ kroków etapu pierwszego metody. W przypadku braku optymalizacji, złożoność częściowego wyboru jest liniowa. Ponadto częściowy wybór nie ma wpływu na asymptotyczną złożoność etapu drugiego, stąd złożoność całej metody pozostaje dalej $O(n^3)$, zgodnie z rozważaniami z wariantu podstawowego.

Poza zastosowaniem tych samych sztuczek optymalizacyjnych co w przypadku podstawowej wersji metody, należy zająć się przyspieszeniem samego procesu wyboru elementu głównego. Zauważmy, że pod przekątną mamy maksymalnie l niezerowych wartości. W związku z tym szukanie elementu głównego można ograniczyć tylko do tych l elementów. Względem wersji podstawowej, przez permutacje wierszy zmianie może ulec liczba elementów niezerowych znajdujących się na prawo od elementu głównego. Ze względu na fakt, że wiersz z wybranym elementem głównym do zamiany może być co najwyżej l miejsc poniżej wiersza, z którym zostanie zamieniony, to również na prawo od nowowybranego elementu głównego może być maksymalnie tylko o l więcej elementów niezerowych niż w przypadku wariantu podstawowego. Oznacza to, iż najdalszym na prawo od k-tego elementu głównego elementem niezerowym jest ten znajdujący się w kolumnie $(k+l+l)$-tej. Wobec tego liczba elementów odejmowanych od wierszy poniżej dalej jest stała, gdy l jest stałe. Czyli dalej złożoność obliczeniowa etapu pierwszego po optymalizacjach wynosi $O(n)$.

W etapie drugim też musimy uwzględnić fakt, że elementy niezerowe mogą być o co najwyżej l miejsc bardziej na prawo niż w wariancie podstawowym. Ponieważ l zakładamy jako stałe, to tu również otrzymujemy złożoność $O(n)$, czyli nie zwiększyliśmy asymptotycznie złożonośći metody eliminacji Gaussa poprzez dodanie częściowego wyboru elementu głównego.

1.4.2 Złożoność pamięciowa metody

Teoretyczna analiza złożoności pamięciowej wariantu metody z częściowym wyborem elementu głównego silnie pokrywa się z tą przeprowadzoną dla wariantu podstawowego, patrz rozdział 1.2.2. Jedyną różnicą jest tworzona w tym wariancie tablica przestawień P, mająca rozmiar $O(n)$.

Zatem tutaj również złożoność pamięciowa to $O(n)$ przy optymalnej strukturze macierzy; dla macierzy zapisanej nieoptymalnie, z użyciem tablicy dwuwymiarowej, złożoność pamięciowa metody wynosiłaby $O(n^2)$.

1.4.3 Uwagi do implementacji i pseudokod

Metody `lastColumn` i `lastRow` zostały objaśnione przy podstawowym wariancie metody eliminacji Gaussa. Produktem przedstawionej wcześniej idei wraz z optymalizacjami jest przedstawiony poniżej pseudokod metody `gaussEliminationPartialPivoting(A, b)` z pliku `blocksys.jl`.

Algorithm 2: `gaussEliminationPartialPivoting`

Data:
- A – macierz blokowa o strukturze zgodnej z zadaniem
- b – wektor prawych stron
- n – rozmiar macierzy
- l – rozmiar bloku

Result: x – wektor rozwiązania, dla którego $Ax = b$

begin

$P = [1, 2, \ldots, n]$
//etap pierwszy
for k ***from*** 1 ***to*** $n-1$ **do**
$m = i$ takie, że $|a_{P[i],k}| = \max_{k \leqslant j \leqslant \texttt{A.lastRow(k)}} |a_{P[i],k}|$
swap($P[k]$, $P[m]$)
for i ***from*** $k+1$ ***to*** *A.lastRow(k)* **do**
$l = a_{P[i],k}/a_{P[k],k}$
$a_{P[i],k} = 0$ //zerowanie nie jest konieczne
for j ***from*** $k+1$ ***to*** *A.lastColumn(k + l)* **do**
$a_{P[i],j} = a_{P[i],j} - l \cdot a_{P[k],j}$
$b_{P[i]} = b_{P[i]} - l \cdot b_{P[k]}$
//etap drugi
$x = b$
$x_n = b_{P[n]}/a_{P[n],n}$
for i ***from*** $n-1$ ***down to*** *1* **do**
for j ***from*** $i+1$ ***to*** *A.lastColumn(i+l)* **do**
$x_{P[i]} = x_{P[i]} - a_{P[i],j} \cdot x_j$
$x_{P[i]} = x_{P[i]}/a_{P[i],i}$
return x

2 Rozkład LU

2.1 Idea wariantu podstawowego

Zauważmy, że pierwszy etap metody eliminacji Gaussa tworzy macierz górnotrójkątną. Macierz tą będziemy nazywać $U = A^{(n)}$. Wtedy rozkładem LU jest podział macierzy A na dolnotrójkątną L i górnotrójkątną U tak, że $A = LU$.

Macierz L jest macierzą przekształceń macierzy A w etapie pierwszym metody Gaussa. Przejście od macierzy $A^{(k)}$ do $A^{(k+1)}$ i wektora $b^{(k)}$ do $b^{(k+1)}$ możemy zapisać następująco:

$$A^{(k+1)} = L^{(k)} A^{(k)}, b^{(k+1)} = L^{(k)} b^{(k)},$$

gdzie

$$L^{(k)} = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -l_{k+1,k} & 1 & & \\ & & -l_{k+2,k} & & 1 & \\ & & \vdots & & & \ddots \\ & & -l_{n,k} & & & 1 \end{bmatrix},$$

pozostałe elementy $L^{(k)}$ są zerami.

Zatem proces sprowadzania macierzy $A^{(1)}$ do macierzy górno trójkątnej $U = A^{(n)}$ możemy zobrazować tak:

$$U = L^{(n-1)} \dots L^{(2)} L^{(1)} A^{(1)}$$
$$A = (L^{(n-1)} \dots L^{(2)} L^{(1)})^{-1} U$$
$$A = L^{(1)^{-1}} L^{(2)^{-1}} \dots L^{(n-1)^{-1}} U.$$

Wtedy macierz

$$L^{(k)^{-1}} = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & l_{k+1,k} & 1 & & \\ & & l_{k+2,k} & & 1 & \\ & & \vdots & & & \ddots \\ & & l_{n,k} & & & 1 \end{bmatrix}$$

oraz

$$L = \begin{bmatrix} 1 & & & & \\ l_{2,1} & 1 & & & \\ l_{3,1} & l_{3,2} & 1 & & \\ \vdots & & & \ddots & \\ l_{n,1} & l_{n,2} & \dots & l_{n,n-1} & 1 \end{bmatrix},$$

gdzie $L = L^{(1)^{-1}} L^{(2)^{-1}} \dots L^{(n-1)^{-1}}$ i pozostałe elementy $L^{(k)^{-1}}$ i L są zerami.

Warto nadmienić, że $l_{i,j}$ to są te same współczynniki, które były wyliczane w metodzie Gaussa do zerowania elementów pod przekątną, co przyda się w implementacji.

Wyznaczanie rozkładu LU jest przydatne, kiedy chcemy rozwiązać układ równań $Ax = b$ dla jednej zadanej macierzy A i wielu wektorów prawych stron b. Wynika to z tego, iż rozwiązanie układu równań $Ax = b$ możemy rozbić na rozwiązanie dwóch prostych obliczeniowo układów równań z macierzami trójkątnymi, mianowicie pierwszy układ to $Ly = b$, który odpowiada nadrobieniu z wektorem b przekształceń wykonywanych przy tworzeniu macierzu U z A, czyli

$$y = L^{-1} b = L^{(n-1)} \dots L^{(2)} L^{(1)} b = b^{(n)}.$$

Drugi układ równań to de facto wykonanie drugiego etapu eliminacji Gaussa, czyli $Ux = y$.

2.2 Implementacja wariantu podstawowego metod z optymalizacjami; teoretyczna analiza złożoności

2.2.1 Uwagi do implementacji

Implementacja metody generującej rozkład LU silnie bazuje na etapie pierwszym metody eliminacji Gaussa. Różnice polegają na tym, że pracujemy tylko na macierzy A oraz zamiast zerować elementy pod przekątną macierzy A, to wstawiamy tam współczynniki $l_{i,j}$, które do wyzerowania tych elementów posłużyły. Oznacza to, że w komputerowej realizacji macierze L i U przechowujemy w jednej macierzy:

$$LU = \begin{bmatrix} u_{1,1} & u_{1,2} & \dots & u_{1,n-1} & u_{1,n} \\ l_{2,1} & u_{2,2} & \dots & u_{2,n-1} & u_{2,n} \\ \vdots & \vdots & & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \dots & l_{n,n-1} & u_{n,n} \end{bmatrix}.$$

Z macierzy L tracimy tylko elementy z przekątnej, jednak nie stanowi to problemu, gdyż znajdowały się tam tylko jedynki.

Implementacja metody rozwiązującej układ $Ax = b$ dla danej macierzy LU i wektora b polega najpierw na modyfikacjach wektora b zgodnych z ideą, a potem na wykonaniu etapu drugiego wariantu podstawowego metody eliminacji Gaussa z wykorzystaniem macierzy LU.

Podobnie jak przy eliminacji Gaussa zastosowane zostały metody `lastRow()` i `lastColumn()` wskazujące na ostatnie elementy niezerowe; ich działanie zostało opisane w rozdziale 1.2.2.

2.2.2 Złożoność obliczeniowa metod

Metoda wyznaczająca rozkład LU jest tak naprawdę pierwszym etapem eliminacji Gaussa z drobnymi zmianami opisanymi w poprzedniej sekcji. Zmiany te nie mają wpływu na asymptotyczną złożność metody, stąd wersja niezoptymalizowana pod strukturę ma złożoność $O(n^3)$. Przy optymalizacji stosujemy te same techniki, co w metodzie Gaussa, polegające na ograniczeniu liczby iteracji poszczególnych pętli wewnętrznych do liczby zależnej od l, którą traktujemy jako stałą. W związku z tym, teoretyczna złożoność obliczeniowa wersji zoptymalizowanej pod strukturę macierzy blokowej z zadania wynosi $O(n)$.

W przypadku metody rozwiązującej układ równań dla zadanego rozkładu LU i wektora b, najpierw dokonujemy modyfikacji na wektorze b. W wersji niezoptymalizowanej wywoływane zagnieżdżone w sobie dwie pętle, liczba iteracji obu z nich jest zależna od rozmiaru macierzy n. Stąd w wersji niezoptymalizowanej ta część metody ma złożoność $O(n^2)$. Druga i ostatnia część metody polega na rozwiązaniu układu równań z macierzą górnotrójkątną (ignorujemy tu elementy macierzy L). Jest ona analogiczna do etapu drugiego metody Gaussa, stąd złożoność tej części to również $O(n^2)$. Zatem ostatecznie wersja niezoptymalizowana tej metody ma złożoność $O(n^2)$. Przy optymalizowaniu tej metody, możemy zauważyć, że liczbę iteracji pętli wewnętrznej przy wyliczaniu modyfikacji do wektora b można zmniejszyć do stałej, zależnej od l wartości. Stąd „nadrobienie" zmian wektora b w wersji zoptymalizowanej ma złożoność $O(n)$. Przy drugiej części metody stosujemy identyczną metodę optymalizacji co przy etapie drugim metody eliminacji Gaussa. Ostatecznie cała metoda, po optymalizacji pod strukturę macierzy, ma złożoność $O(n)$.

Choć asymptotycznie metoda rozwiązująca układ równań z wykorzystaniem rozkładu LU ma taką samą złożoność co metoda eliminacji Gaussa, to jest szybsza, ponieważ nie musimy wykonywać już żadnej operacji na macierzy LU, w przeciwieństwie do konieczności zmodyfikowania macierzy A w metodzie Gaussa. Oznacza to, że dalej stosowanie rozkładu LU jest opłacalne, gdy chcemy wyliczyć rozwiązania układu równań $Ax = b$ dla jednej macierzy A i wielu wektorów prawych stron b.

2.2.3 Złożoność pamięciowa metod

Metoda generująca rozkład LU otrzymuje jako argument obiekt struktury `BlockMatrix`; w rozdziale 3 i 5 zobaczymy, że ma on rozmiar liniowy względem wielkości problemu. Poza tym metoda tworzy iteratory pętli i zmienną pomocniczą l, zajmujące stałą ilość pamięci. Macierz A otrzymana na wejściu metody jest przekształcana w macierz LU, czyli nie ma potrzeby tworzyć dodatkowego obiektu `BlockMatrix`. Widzimy zatem, że przy zoptymalizowanym przechowywaniu macierzy A złożoność pamięciowa to $O(n)$; gdybyśmy macierz przechowywali wprost w tablicy dwuwymiarowej, to złożoność pamięciowa wynosiłaby $O(n^2)$.

Metoda rozwiązująca układ równań na wejściu otrzymuje:

- obiekt struktury `BlockMatrix` ($O(n)$ pamięci),
- wektor prawych stron b, zajmujący $O(n)$ pamięci.

Dodatkowo metoda tworzy wektor rozwiązania x, także zajmujący $O(n)$ pamięci oraz iteratory pętli, zajmujące stałą ilość pamięci. Wobec tego metoda rozwiązująca układ równań ma złożoność pamięciową $O(n)$ dla optymalnej struktury przechowującej macierz; bez tych optymalizacji złożoność pamięciowa to $O(n^2)$.

2.2.4 Pseudokody

Produktem przedstawionej wcześniej idei wraz z optymalizacjami są przedstawione poniżej pseudokody metod `generateLU!(A)` i `solveGivenLU!(LU, b)` z pliku `blocksys.jl`.

Algorithm 3: `generateLU`

Data:
- A – macierz blokowa o strukturze zgodnej z zadaniem
- n – rozmiar macierzy
- l – rozmiar bloku

Result: LU – macierz z rozkładem LU

begin

```
for k from 1 to n − 1 do
    for i from k + 1 to A.lastRow(k) do
        l = a_{i,k} / a_{k,k}
        a_{i,k} = l //zamiast zerowania jak w Gaussie
        for j from k + 1 to A.lastColumn(k) do
            a_{i,j} = a_{i,j} − l · a_{k,j}
return A
```

Algorithm 4: `solveGivenLU`

Data:
- LU – macierz z rozkładem LU
- b – wektor prawych stron
- n – rozmiar macierzy
- l – rozmiar bloku

Result: x – wektor rozwiązania, dla którego $Ax = b$

begin

```
//nadrobienie modyfikacji wektora, Ly=b
for k from 1 to n − 1 do
    for i from k + 1 to LU.lastRow(k) do
        b_i = b_i − LU_{i,k} · b_k
//etap drugi jak w Gaussie, Ux=y
x = b
x_n = b_n / LU_{n,n}
for i from n − 1 down to 1 do
    for j from i + 1 to LU.lastColumn(i) do
        x_i = x_i − LU_{i,j} · x_j
    x_i = x_i / LU_{i,i}
return x
```

2.3 Idea wariantu z częściowym wyborem elementu głównego

Przy generowaniu rozkładu LU musimy uważać, żeby na przekątnej nie znalazły się elementy bliskie zeru, ze względów analogicznych co przy metodzie eliminacji Gaussa. Pomoże w tym strategia częściowego wyboru elementu głównego, w której chcemy wybrać taki wiersz m, dla którego

$$|a_{m,k}| = \max_{k \leqslant i \leqslant n} |a_{i,k}|.$$

Element z tego wiersza zastosujemy jako element główny, analogicznie jak w metodzie Gaussa. Dzięki temu jest większa szansa na uniknięcie błędów numerycznych oraz błędu dzielenia przez zero.

Poza zamianami wierszy przy wyborze elementu głównego, metoda generowania rozkładu LU działa tak samo jak ta w wariancie podstawowym.

Metoda rozwiązująca układ równań z zadanym rozkładem LU działa analogicznie jak ta w wariancie podstawowym; trzeba jedynie zwracać uwagę na fakt, że przy wykonywaniu modyfikacji wektora b też trzeba będzie przestawiać jego składowe (w taki sam sposób, w jaki były przestawiane wiersze macierzy). Trzeba też pamiętać o tym, że składowe rozwiązania będą w złej kolejności.

2.4 Implementacja wariantu z częściowym wyborem z optymalizacjami; teoretyczna analiza złożoności

2.4.1 Uwagi do implementacji

Tak jak w przypadku implementacji częściowego wyboru elementu głównego w metodzie eliminacji Gaussa, nie będziemy przestawiać wierszy macierzy, lecz zapamiętamy permutacje wierszy w tablicy pomocniczej P. Oszczędzi to wielu operacji na pamięci; ponadto czas dostępu do konkretnego elementu tablicy P jest stały. Działanie tablicy P zostało wyjaśnione w rozdziale 1.4. Należy pamiętać o tym, że metoda generująca rozkład LU musi zwrócić poza macierzą LU również tablicę P, ponieważ będzie ona potrzebna w metodzie rozwiązującej układ równań.

2.4.2 Analiza złożoności obliczeniowej

Podobnie jak w przypadku wariantu bez wyboru elementu głównego, złożoność metody generującej rozkład LU jest asymptotycznie taka sama, jak pierwszego etapu metody eliminacji Gaussa z częściowym wyborem elementu głównego. Wynika to z tego, że wykonywane operacje są identyczne z jednym wyjątkiem, mianowicie przy generowaniu rozkładu LU nie pracujemy na wektorze b. Zatem niezoptymalizowana wersja ma złożoność $O(n^3)$, zaś po zastosowaniu analogicznych optymalizacji, złożoność obliczeniowa ulega zmniejszeniu do $O(n)$.

Z kolei wobec metody rozwiązującej układ równań zastosowano analogiczne optymalizacje, co w etapie drugim metody eliminacji Gaussa z częściowym wyborem elementu głównego (patrz rozdział 1.4.1). Oznacza to, że złożoność rozwiązania układu równań ze zmodyfikowanym już wektorem b wynosi bez optymalizacji $O(n^2)$, a z optymalizacjami $O(n)$. Dodatkowymi operacjami rozważanej metody (względem etapu drugiego metody eliminacji Gaussa) są te związane z nadrobieniem modyfikacji wektora b wraz z permutacjami, tj. $Lz = Pb$. Tutaj również udało się ograniczyć liczbę iteracji, korzystając z wiedzy na temat pierwszego elementu niezerowego w danym wierszu (z uwzględnieniem przestawień wierszy P) macierzy LU. Dzięki temu liczba iteracji wewnętrznej pętli jest stała, zaś zewnętrzna pętla wykonuje się $O(n)$ razy. Mamy zatem wiedzę na temat złożoności obliczeniowej każdego z fragmentów metody i możemy wywnioskować, że po optymalizacjach pod strukturę macierzy złożoność została zredukowana do $O(n)$.

2.4.3 Analiza złożoności pamięciowej

Analiza złożoności pamięciowej tych metod pokrywa się z tą przeprowadzoną dla wariantu bez wyboru elementu głównego, patrz rozdział 2.2.3. Jedyną różnicą jest utworzona w metodzie generującej rozkład LU i przekazywana do metody rozwiązującej układ równań tablica przestawień wierszy P. Rozmiar tej tablicy jest liniowy względem rozmiaru problemu.

Widzimy zatem, że wprowadzenie strategii częściowego wyboru elementu głównego nie zmieniło asymptotycznej złożoności pamięciowej. Wynosi ona dalej $O(n)$ przy optymalnej strukturze przechowującej macierze i $O(n^2)$ przy braku zoptymalizowania struktur (np. tablica dwuwymiarowa).

2.4.4 Pseudokody

Produktem przedstawionej wcześniej idei wraz z optymalizacjami są przedstawione poniżej pseudokody metod `generateLUPartialPivoting!(A)` i solveGivenPartialPivotedLU!(LU, P, b) z pliku blocksys.jl.

Algorithm 5: `generateLUPartialPivoting`

Data:
- A – macierz blokowa o strukturze zgodnej z zadaniem
- n – rozmiar macierzy
- l – rozmiar bloku

Result: LU – macierz z rozkładem LU, P – tablica przestawień wierszy

begin

$P = [1, 2, \ldots, n]$

for k ***from*** 1 ***to*** $n - 1$ **do**

$m = i$ takie, że $|a_{P[i],k}| = \max_{k \leqslant j \leqslant \texttt{A.lastRow(k)}} |a_{P[i],k}|$

swap($P[k]$, $P[m]$)

for i ***from*** $k + 1$ ***to*** *A.lastRow(k)* **do**

$l = a_{P[i],k} / a_{P[k],k}$

$a_{P[i],k} = l$ //zamiast zerowania jak w Gaussie

for j ***from*** $k + 1$ ***to*** *A.lastColumn(k + l)* **do**

$a_{P[i],j} = a_{P[i],j} - l \cdot a_{P[k],j}$

return A, P

Algorithm 6: `solveGivenPartialPivotedLU`

Data:
- LU – macierz z rozkładem LU
- P – tablica przestawień wierszy
- b – wektor prawych stron
- n – rozmiar macierzy
- l – rozmiar bloku

Result: x – wektor rozwiązania, dla którego $Ax = b$

begin

//nadrobienie modyfikacji wektora, Ly=Pb

for k ***from*** 2 ***to*** n **do**

for i ***from*** *LU.firstColumn(P[k])* ***to*** $k - 1$ **do**

$b_{P[k]} = b_{P[k]} - LU_{P[k],i} \cdot b_{P[i]}$

//etap drugi jak w Gaussie, Ux=y

$x_n = b_{P[n]} / LU_{P[n],n}$

for i ***from*** $n - 1$ ***down to*** *1* **do**

$x_i = b_{P[i]}$

for j ***from*** $i + 1$ ***to*** *LU.lastColumn(i + l)* **do**

$x_i = x_i - LU_{P[i],j} \cdot x_j$

$x_i = x_i / LU_{P[i],i}$

return x

3 Opis zastosowanej struktury danych

3.1 Sposób przechowywania macierzy i optymalizacje dostępu

Macierze są przechowywane w specjalnie do tego utworzonej strukturze `BlockMatrix`. Jej „sercem" jest obiekt klasy `SparseMatrixCSC{Float64, Int}`, który idealnie sprawdza się do przechowywania ogromnych, rzadkich macierzy. Obiekty te charakteryzują się tym, że nie zajmują pamięci proporcjonalnie do liczby komórek, do których można się odwołać, lecz proporcjonalnie do liczby komórek **zainicjalizowanych wartością**. Wobec tego, macierze rzadkie, które nie mają wielu wartości niezerowych, przechowywane są w tym obiekcie z wykorzystaniem znacznie mniejszej ilości pamięci, niż w przypadku dwuwymiarowej tablicy.

Podczas testów działania tej struktury można zauważyć, że czas dostępu do komórki zainicjalizowanej uprzednio wartością jest w przybliżeniu stały. Inaczej jest gdy chcemy się odwołać do komórki niezainicjalizowanej wartością; dostęp wtedy nie jest stały i zależy od rozmiaru macierzy. Różnica ta ma kolosalny wpływ na czas działania metod, co zostanie zobrazowane w rozdziale 5. W związku z tym, przy importowaniu macierzy z pliku (wszystkie metody importu/eksportu znajdują się w pliku `iofuncs.jl`) i zapisywaniu do obiektu `SparseMatrixCSC` można zastosować dwie strategie:

- inicjalizacja w obiekcie tylko pól z wartościami niezerowymi z macierzy (metoda nieoptymalna),
- inicjalizacja w obiekcie zarówno pól z wartościami niezerowymi, jak i zer, co do których wiadomo, że będą wykorzystywane (metoda optymalna).

Warto tutaj nadmienić, że ze względu na optymalizacje metod wymienione w poprzednich rozdziałach, liczba zer faktycznie wykorzystywanych przez metody jest niewielka, więc rozmiar obiektu utworzonego z użyciem optymalnej strategii jest mniej niż kilkukrotnie większy od obiektu utworzonego z pierwszą strategią. Mamy więc dalej strukturę rozmiaru liniowego względem wielkości problemu.

Zera, do których metody mogą się odwoływać (skorzystano tu z opisu struktury macierzy A ze wstępu), to:

- zera zawarte w macierzach B_k,
- zera znajdujące się pod przekątną macierzy C_k,
- $l-1$ zer znajdujących się na prawo od elementów przekątnej macierzy C_k (te zera są konieczne tylko przy metodach z częściowym wyborem elementu głównego)

Widzimy zatem, że wymienione powyżej sztucznie dodane zera nie zajmą bardzo dużo pamięci.

3.2 Dodatkowe pola struktury danych

W strukturze `BlockMatrix` poza obiektem `SparseMatrixCSC` znajdują się następujące pola:

- rozmiar macierzy n,
- rozmiar bloku l,
- liczba bloków macierzy,
- licznik zliczający liczbę dostępów do komórek macierzy,
- licznik zliczający liczbę modyfikacji komórek macierzy.

Dodatkowo zaimplementowano omawiane wcześniej metody zwracające indeksy pierwszych i ostatnich niezerowych elementów w danym wierszu lub kolumnie, tj. `firstRow`, `firstColumn`, `lastRow`, `lastColumn`.

3.3 Sposób przechowywania wektorów

Do przechowywania wektorów wykorzystano strukturę `VectorFloat64`, bez żadnych zmian w strukturze.

4 Testowanie implementacji

4.1 Testy jednostkowe

W pliku `tests.jl` znajdują się testy metod rozwiązywania układów równań. Każdy wariant metody został przetestowany na dwa sposoby:

- rozwiązywanie układu równań $Ax = b$, dla którego wiemy, że rozwiązaniem x jest wektor jedynek,
- rozwiązywanie układu równań $Ax = b$ dla losowego wektora b i macierzy A i następnie sprawdzenie wprost równości $Ax = b$.

W obu przypadkach macierz A jest rozmiaru $n = 5000$, $l = 5$ (oczywiście można te wartości zmienić) i jest generowana z użyciem metody generującej `blockmat()` zaimplementowanej przez prowadzącego kurs profesora Pawła Zielińskiego.

4.2 Program testowy

Na potrzeby zadania napisany został interaktywny program testowy, z użyciem którego można przetestować zaimplementowane metody zgodnie z wymogami listy. Po uruchomieniu programu, wprowadzić można następujące komendy:

- `readMatrix filename` – wczytywanie macierzy z pliku tekstowego i generowanie wektora prawych stron na jej podstawie (wtedy rozwiązanie to wektor jedynek),
- `readVector filename` – wczytywanie wektora prawych stron z pliku tekstowego,
- `readBoth filename_matrix filename_vector` – wczytywanie macierzy i wektora z plików tekstowych,
- `solveGauss` – rozwiązanie układu równań Ax=b metodą eliminacji Gaussa bez wyboru,
- `solveGaussPivot` – rozwiązanie układu równań Ax=b metodą eliminacji Gaussa z częściowym wyborem,
- `generateLU` – generowanie rozkładu LU macierzy,
- `generateLUPivot` – generowanie rozkładu LU macierzy z częściowym wyborem,
- `solveWithLU` – rozwiązanie układu równań Ax=b z użyciem wygenerowanego wcześniej rozkładu LU komendą generateLU,
- `solveWithLUPivot` – rozwiązanie układu równań Ax=b z użyciem wygenerowanego wcześniej rozkładu LU komendą generateLUPivot,
- `write filename` – zapisanie rozwiązania do pliku tekstowego (jeżeli zastosowano komendę `readMatrix` przy wczytywaniu to w pierwszym wierszu zapisuje również błąd względny rozwiązania),
- `help` – lista komend,
- `about` – informacje o programie,
- `exit` – wyjście z programu.

5 Empiryczne porównanie zaimplementowanych algorytmów

5.1 Badanie złożoności czasowej

W tym eksperymencie sprawdzona zostanie złożoność czasowa zaimplementowanych metod rozwiązywania układów równań. Wyniki badań porównane zostaną z teoretyczną złożonością obliczeniową algorytmów.

5.1.1 Metodyka badań

Program, w którym zostały wykonane badania, zapisany został w pliku `complexitytests.jl`. Poza zaimplementowanymi metodami, sprawdzono średni czas rozwiązywania układu równań metodą nieoptymalną, tj. wprost wyliczając x ze wzoru $Ax = b$ (dalej jako metoda bruteforce).

Dla każdego z rozmiarów wygenerowano 50 losowych macierzy o rozmiarze bloku $l = 5$ z użyciem metody `blockmat()` oraz po jednym wektorze prawych stron na macierz. Czas generowania tych obiektów nie był brany pod uwagę przy pomiarach czasu. Rozmiary generowanych macierzy były następujące:

- dla metody bruteforce – $n \in \{1000, 2500, 5000, 7500, 10000, 12500, 15000\}$,
- dla metod zoptymalizowanych wywoływanych na nieoptymalnie utworzonych obiektach `SparseMatrixCSC` – $n \in \{1000, 2500, 5000, 7500, 10000, 12500, 15000, 20000, 40000, 60000, 80000, 100000\}$,

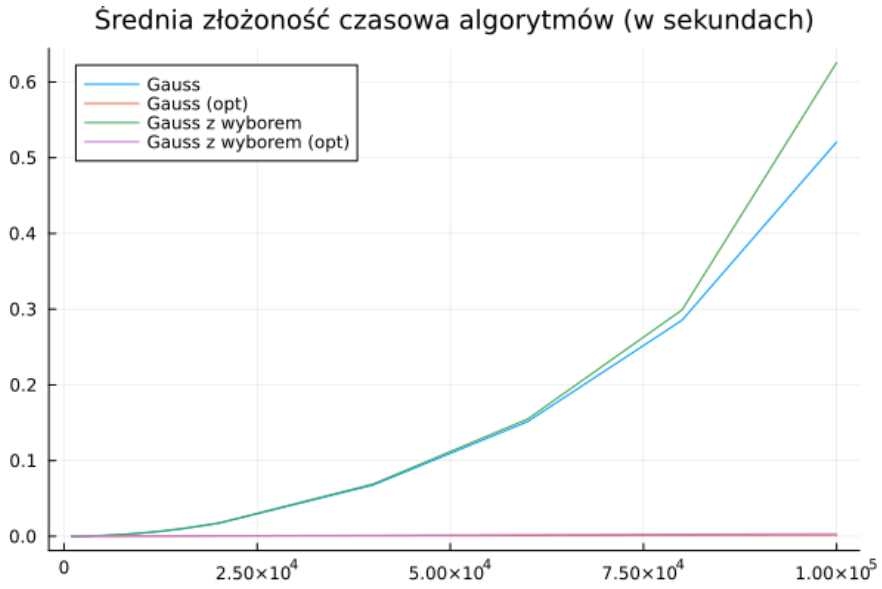
- dla metod zoptymalizowanych wywoływanych na optymalnie utworzonych obiektach `SparseMatrixCSC` – $n \in \{1000, 2500, 5000, 7500, 10000, 12500, 15000, 20000, 40000, 60000, 80000, 100000, 150000, 200000, 250000, 500000, 750000, 1000000\}$.

Czas wykonywania się poszczególnych metod został zmierzony z użyciem makra `@timed` i uśredniony. Metody były wywoływane do tych samych macierzy i wektorów prawych stron. Wyniki przedstawiono na wykresach.

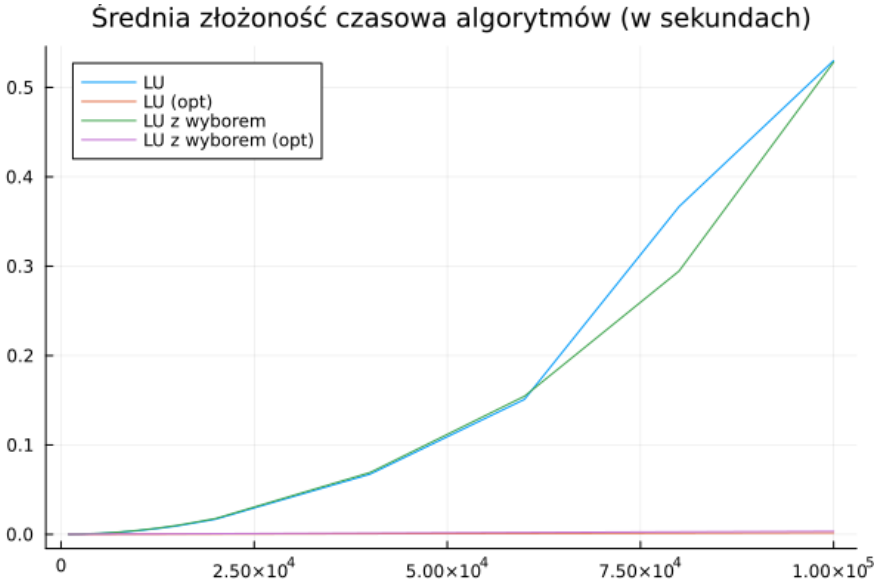
Dodatkowo, dla tych samych macierzy, zmierzono średnią liczbę odczytów elementów macierzy przy wywoływaniu metod. Wykorzystano do tego pole `elementAccessCount` znajdujące się w strukturze `BlockMatrix`.

5.1.2 Wyniki badań

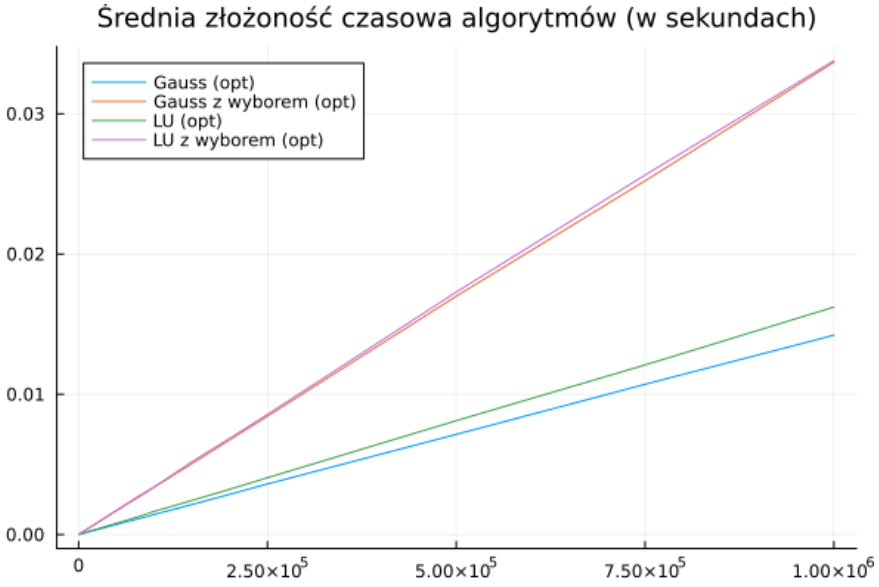
Rysunek 1: Wykres średniego czasu wykonywania się tradycyjnej, nieoptymalnej metody i zoptymalizowanych metod wywołanych na optymalnie utworzonych strukturach macierzy w zależności od rozmiaru problemu.



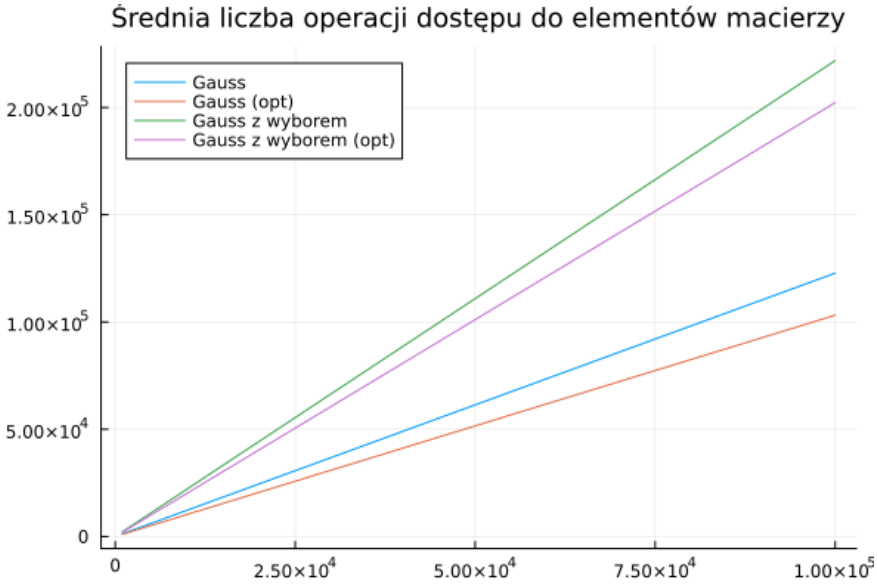
Rysunek 2: Wykres średniego czasu wykonywania się metod bazujących na eliminacji Gaussa wywołanych na optymalnie (opt.) i nieoptymalnie utworzonych strukturach macierzy w zależności od rozmiaru problemu.



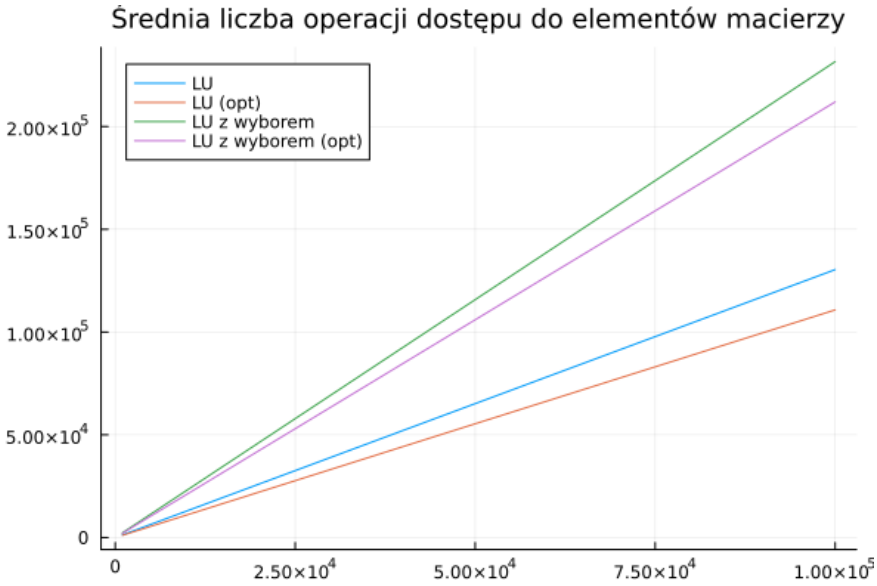
Rysunek 3: Wykres średniego czasu wykonywania się metod bazujących na rozkładzie LU wywołanych na optymalnie (opt.) i nieoptymalnie utworzonych strukturach macierzy w zależności od rozmiaru problemu.



Rysunek 4: Wykres średniego czasu wykonywania się metod bazujących na eliminacji Gaussa i rozkładzie LU wywołanych na optymalnie utworzonych strukturach macierzy w zależności od rozmiaru problemu.



Rysunek 5: Wykres średniej liczby odczytów elementów macierzy dla metod bazujących na eliminacji Gaussa wywołanych na optymalnie (opt.) i nieoptymalnie utworzonych strukturach macierzy w zależności od rozmiaru problemu.



Rysunek 6: Wykres średniej liczby odczytów elementów macierzy dla metod bazujących na rozkładzie LU wywołanych na optymalnie (opt.) i nieoptymalnie utworzonych strukturach macierzy w zależności od rozmiaru problemu.

5.1.3 Wnioski

Na Wykresie 1 widzimy jak bardzo nieoptymalna jest metoda bruteforce. Rozwiązywanie nią większych układów równań jest w praktyce niemożliwe, dlatego zaimplementowane zostały metody zoptymalizowane pod strukturę macierzy.

Istotnym czynnikiem wpływającym na czas działania metod okazał się czas dostępu do elementów macierzy. Okazało się, że w strukturze `SparseMatrixCSC` czas dostępu do elementów zainicjalizowanych wartością jest stały, zaś próba dostępu do elmentu niezainicjalizowanego jest czasochłonna i zależna od rozmiaru macierzy. Stąd pojawiła się strategia optymalnego tworzenia macierzy, polegająca na dodawaniu zer, do których będziemy się odwoływać; omówiona ona została w rozdziale 3.1. Wykresy 2 i 3 pokazują jak bardzo istotny jest to czynnik. Metody operujące na nieoptymalnie utworzonych macierzach wykonywały się w czasie ponadliniowym, niezgodnym z przypuszczeniami wynikającymi z teoretycznej analizy złożoności. Widzimy

jednak, że średnia liczba operacji odczytu komórek macierzy rosła liniowo względem rozmiaru problemu, zgodnie z naszymi oczkiwaniami (patrz wykresy 5 i 6).

Z kolei na wykresie 4 widzimy, że teoretyczne rozważania pokrywają się z rzeczywistością wtedy, gdy operujemy tylko na elementach zainicjalizowanych wartością w strukturze `SparseMatrixCSC`. Czas wykonywania się metod rozwiązywania układu równań liniowych jest liniowy, przy czym czynnik znajdujący się przy n jest większy dla metod ze strategią częściowego wyboru. Strategia ta wiąże się z dodatkowymi operacjami potrzebnymi do znalezienia elmentu głównego.

Wyniki tego eksperymentu dobitnie pokzały jak bardzo istotne jest optymalizowanie metod pod struktury, na których będą wywoływane. Tylko dzięki temu byliśmy w stanie rozwiązywać układy równań z setkami tysięcy, a nawet milionami niewiadomych.

5.2 Badanie złożoności pamięciowej

W tym eksperymencie sprawdzone zostaną przypuszczenia dotyczące złożoności pamięciowej zaimplementowanych metod (w teorii powinna być liniowa) oraz zależność rozmiaru struktury względem wielkości problemu. Ponadto zobrazowany zostanie wpływ optymalizacji polegającej na sztucznym dodaniu zer na rozmiar struktury macierzy.

5.2.1 Metodyka badań

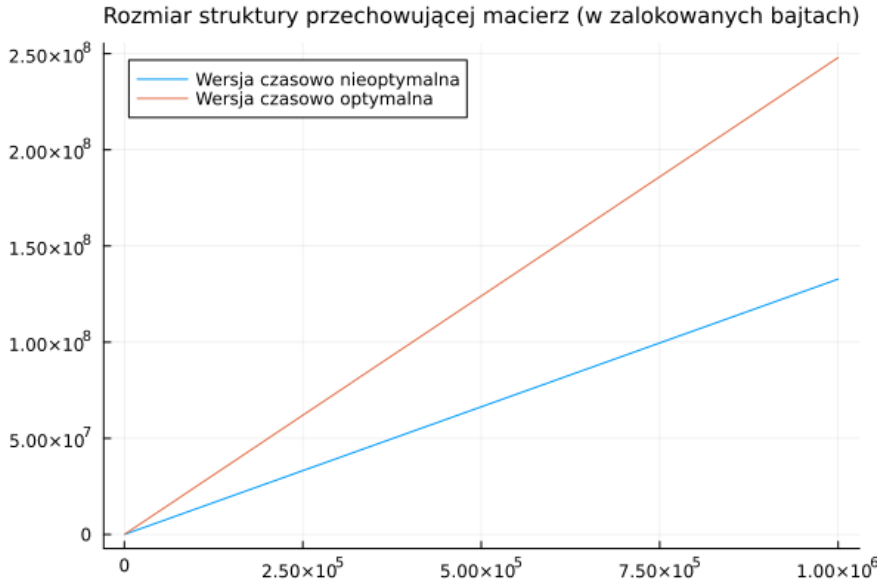
Analogicznie do poprzedniego eksperymentu, program, w którym zostały wykonane badania, zapisany został w pliku `complexitytests.jl`.

Dla każdego z rozmiarów wygenerowano 50 losowych macierzy o rozmiarze bloku $l = 5$ z użyciem metody `blockmat()` oraz po jednym wektorze prawych stron na macierz. Rozmiary generowanych macierzy były następujące:

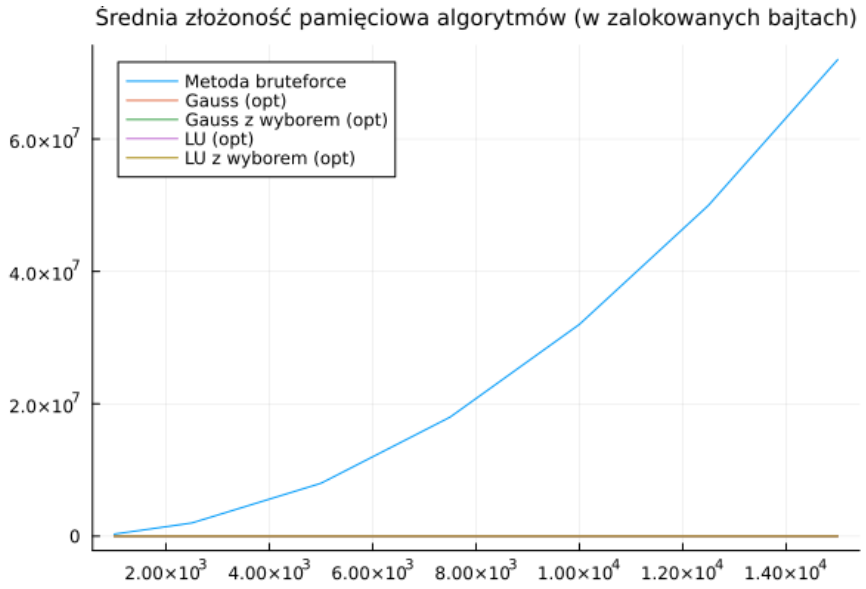
- dla metody bruteforce – $n \in \{1000, 2500, 5000, 7500, 10000, 12500, 15000\}$,
- dla metod zoptymalizowanych wywoływanych na nieoptymalnie utworzonych obiektach `SparseMatrixCSC` – $n \in \{1000, 2500, 5000, 7500, 10000, 12500, 15000, 20000, 40000, 60000, 80000, 100000\}$,
- dla metod zoptymalizowanych wywoływanych na optymalnie utworzonych obiektach `SparseMatrixCSC` – $n \in \{1000, 2500, 5000, 7500, 10000, 12500, 15000, 20000, 40000, 60000, 80000, 100000, 150000, 200000, 250000, 500000, 750000, 1000000\}$.

Ilość pamięci wykorzystanej przez poszczególne metody została zmierzona z użyciem makra `@timed`. Metody były wywoływane do tych samych macierzy i wektorów prawych stron. Ponadto, z użyciem metody `Base.summarysize()` zmierzony został rozmiar utworzonych struktur przechowujących macierze. Wyniki uśredniono i przedstawiono na wykresach.

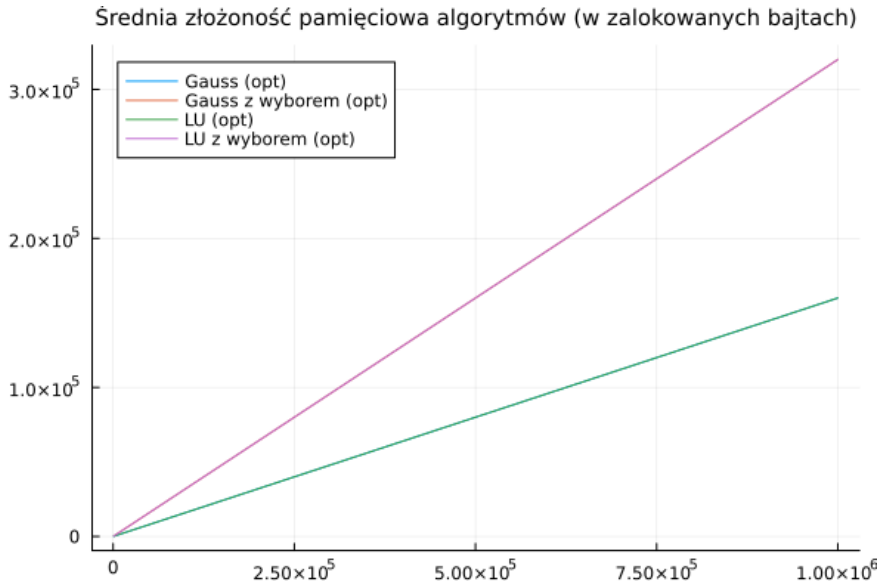
5.2.2 Wyniki badań



Rysunek 7: Wykres ilości pamięci zajmowanej przez strukturę przechowującą macierz w zależności od rozmiaru problemu.



Rysunek 8: Wykres złożoności pamięciowej metody tradycyjnej (nieoptymalnej) i metod zoptymalizowanych wywołanych na optymalnie utworzonych strukturach macierzy w zależności od rozmiaru problemu.



Rysunek 9: Wykres złożoności pamięciowej metod zoptymalizowanych wywołanych na optymalnie utworzonych strukturach macierzy w zależności od rozmiaru problemu.

5.2.3 Interpretacja wyników i wnioski

Na wykresie 7 widzimy, że obiekty przechowujące macierz blokową A zajmują liniową w stosunku do rozmiaru problemu ilość pamięci, niezależnie od zastosowanej strategii. Choć wersja z dodanymi sztucznie zerami zajmuje prawie dwa razy więcej pamięci, to uzyskana dzięki tej optymalizacji redukcja złożoności obliczeniowej w pełni to dodatkowe zużycie pamięci rekompensuje.

Zastosowanie metody bez jakichkolwiek optymalizacji powoduje zapisywanie do obiektu `SparseMatrixCSC` elementów kompletnie zbędnych; zapisujemy dosłownie wszystkie elementy, w tym mnóstwo zer. Stąd zmierzona złożoność pamięciowa tej metody jest tak duża, że na wykresie 8 liczba bajtów zaalokowanych przez metody zoptymalizowane jest praktycznie niewidoczna. Widać zatem jak bardzo oszczędne dla pamięci jest wykorzystanie własności struktury macierzy pod optymalizacje algorytmów.

Z kolei wykres 9 upewnia nas w przekonaniu, iż optymalnie napisane metody rozwiązujące układ równań mają liniową złożoność pamięciową. Warto tutaj zwrócić uwagę na to, że metody wykorzystujące strategię częściowego wyboru korzystają z większej ilości pamięci, co wynika z konieczności tworzenia (i przekazywania w przypadku rozkładu LU) tablicy przestawień wierszy P.

5.3 Badanie jakości otrzymywanych rozwiązań

Celem tego eksperymentu jest sprawdzenie, jak bardzo dokładne wyniki dają poszczególne metody rozwiązywania układów równań.

5.3.1 Metodyka badań

Program, w którym zostały wykonane badania, zapisany jest w pliku `qualitytests.jl`. Wygenerowanych zostało po 100 losowych macierzy rozmiarów $n \in \{50, 100, 1000, 10000, 100000, 1000000\}$ oraz wielkości bloków $l \in \{2, 5, 10\}$. Do wygenerowania macierzy wykorzystano metodę `blockmat()` zaimplementowaną przez prowadzącego kurs profesora Pawła Zielińskiego. Każda z tych macierzy A została wczytana z wygenerowaniem wektora prawych stron b takiego, aby rozwiązaniem był wektor jedynek. Następnie rozwiązano układ równań $Ax = b$ czterema zaimplementowanymi metodami: z wykorzystaniem eliminacji Gaussa z częściowym wyborem elementu głównego i bez wyboru oraz z wykorzystaniem rozkładu LU z częściowym wyborem elementu głównego i bez wyboru. Wyniki przyrównano do właściwego rezultatu, podając błąd względny:

$$\delta = \frac{||x - \tilde{x}||}{||x||},$$

gdzie x jest wektorem jedynek, a $\tilde{x}$ jest rozwiązaniem uzyskanym z wykorzystaniem danej metody. Do wyliczenia normy wykorzystano funkcję `norm()` z pakietu `LinearAlgebra`. W tabeli podano błąd względny średni dla każdej z metod i każdego z rozmiarów n i l.

5.3.2 Wyniki badań

n	l	$\overline{\delta}_{gauss}$	$\overline{\delta}_{gaussPivot}$	$\overline{\delta}_{LU}$	$\overline{\delta}_{LUPivot}$
50	2	$8.5635060333 * 10^{-16}$	$4.0211935214 * 10^{-16}$	$8.5635060333 * 10^{-16}$	$4.0211935214 * 10^{-16}$
	5	$6.6392036865 * 10^{-15}$	$4.1146456087 * 10^{-16}$	$6.6392036865 * 10^{-15}$	$4.1146456087 * 10^{-16}$
	10	$2.3178383497 * 10^{-14}$	$4.9383173502 * 10^{-16}$	$2.3178383497 * 10^{-14}$	$4.9383173502 * 10^{-16}$
10^2	2	$6.9031643252 * 10^{-16}$	$4.0608488135 * 10^{-16}$	$6.9031643252 * 10^{-16}$	$4.0608488135 * 10^{-16}$
	5	$6.9613435924 * 10^{-14}$	$4.2984047381 * 10^{-16}$	$6.9613435924 * 10^{-14}$	$4.2984047381 * 10^{-16}$
	10	$2.6344251230 * 10^{-14}$	$5.2117477002 * 10^{-16}$	$2.6344251230 * 10^{-14}$	$5.2117477002 * 10^{-16}$
10^3	2	$1.0146974615 * 10^{-15}$	$4.1666451905 * 10^{-16}$	$1.0146974615 * 10^{-15}$	$4.1666451905 * 10^{-16}$
	5	$2.7517733805 * 10^{-14}$	$4.4538794396 * 10^{-16}$	$2.7517733805 * 10^{-14}$	$4.4538794396 * 10^{-16}$
	10	$8.7630661855 * 10^{-14}$	$5.4871984849 * 10^{-16}$	$8.7630661855 * 10^{-14}$	$5.4871984849 * 10^{-16}$
10^4	2	$3.8489209390 * 10^{-15}$	$4.1818473300 * 10^{-16}$	$3.8489209390 * 10^{-15}$	$4.1818473300 * 10^{-16}$
	5	$1.1825514695 * 10^{-13}$	$4.4311669885 * 10^{-16}$	$1.1825514695 * 10^{-13}$	$4.4311669885 * 10^{-16}$
	10	$3.1444555552 * 10^{-13}$	$5.5196016883 * 10^{-16}$	$3.1444555552 * 10^{-13}$	$5.5196016883 * 10^{-16}$
10^5	2	$1.5052180591 * 10^{-14}$	$4.1825399276 * 10^{-16}$	$1.5052180591 * 10^{-14}$	$4.1825399276 * 10^{-16}$
	5	$4.5715888421 * 10^{-13}$	$4.4370083723 * 10^{-16}$	$4.5715888421 * 10^{-13}$	$4.4370083723 * 10^{-16}$
	10	$7.3166905125 * 10^{-13}$	$5.5139172168 * 10^{-16}$	$7.3166905125 * 10^{-13}$	$5.5139172168 * 10^{-16}$
10^6	2	$1.2866527329 * 10^{-13}$	$4.1822433281 * 10^{-16}$	$1.2866527329 * 10^{-13}$	$4.1822433281 * 10^{-16}$
	5	$1.2921470152 * 10^{-12}$	$4.4364930369 * 10^{-16}$	$1.2921470152 * 10^{-12}$	$4.4364930369 * 10^{-16}$
	10	$4.8707239961 * 10^{-12}$	$5.5143746832 * 10^{-16}$	$4.8707239961 * 10^{-12}$	$5.5143746832 * 10^{-16}$

Tabela 1: Średni błąd względny metod rozwiązywania układów równań $Ax = b$ dla macierzy losowych A rozmiaru n i wielkości bloku l.

5.3.3 Interpretacja wyników i wnioski

Zgodnie z oczekiwaniami, zastosowanie strategii częściowego wyboru zmniejszyło błąd względny metod rozwiązujących układ równań. Różnica jest w szczególności widoczna w przypadku większych macierzy z większymi rozmiarami bloków. Może wynikać to z tego, że gdy bloki są większe, to mamy więcej elementów do wyzerowania i jest większa szansa, że metody bez strategii wyboru dadzą duży błąd numeryczny. Z kolei metody ze strategią częściowego wyboru mają więcej elementów, spośród których możemy wybrać element główny; jest zatem większa szansa na uzyskanie dużego elementu głównego, nie dającego błędów numerycznych przy dzieleniu przez niego. Widzimy, że metody wykorzystujące strategie częściowego wyboru dają podobny średni błąd dla każdego rozmiaru macierzy i bloku.

Możemy zatem wywnioskować, że gdy zależy nam na dokładniejszych wynikach, to warto zastosować strategię częściowego wyboru elementu głównego.