Sprawozdanie z listy 5

Michał Gancarczyk 244923

Streszczenie

Celem zamieszczonych w liście zadań jest zaimplementowanie zestawu funkcji pozwalających na efektywne rozwiązanie układów równań liniowych, a następnie porównanie ich wydajności przy użyciu danych testowych.

Pierwsza sekcja sprawozdania poświęcona jest przedstawieniu specyfiki problemu. Kolejne dwie sekcje służą do opisania wykorzystanych do rozwiązania układów metod, oraz przedstawiają zastosowane podczas ich implementacji optymalizacje.

Ostatnie dwie sekcje zawierają wyniki działania zaimplementowanych algorytmów, oraz wnioski.

Do rozwiązania zadań użyty został język Julia, wersja 1.3.0.

1 Opis problemu

Celem zadania jest opracowanie funkcji rozwiązujących, za pośrednictwem metody eliminacji Gaussa, oraz rozkładu LU, układ równań liniowych

$$Ax = b$$

dla danej macierzy współczynników $A \in \mathbb{R}^{n\times n}$ i wektora prawych stron $b \in \mathbb{R}^n$, $n \geq 4$.

Macierz A jest macierzą rzadką o następującej strukturze:

$$A = \begin{pmatrix} A_1 & C_1 & 0 & 0 & 0 & \cdots & 0 \\ B_1 & A_2 & C_2 & 0 & 0 & \cdots & 0 \\ 0 & B_3 & A_3 & C_3 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & B_{v-2} & A_{v-2} & C_{v-2} & 0 \\ 0 & \cdots & 0 & 0 & B_{v-1} & A_{v-1} & C_{v-1} \\ 0 & \cdots & 0 & 0 & 0 & B_v & A_v \end{pmatrix},$$

gdzie $v = n/l$, $l \geq 2$ - wielkość bloku, a $A_k, B_k, C_K \in \mathbb{R}^{l\times l}$ są macierzami wewnętrznymi. A_k jest macierzą gęstą, natomiast postaci macierzy B_k i C_k zaprezentowane zostały poniżej:

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

$$C_k = \begin{pmatrix} C^k_1 & 0 & 0 & \cdots & 0 \\ 0 & C^k_2 & 0 & \cdots & 0 \\ 0 & 0 & C^k_3 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & C^k_{l-1} & 0 \\ 0 & \cdots & 0 & 0 & C^k_l \end{pmatrix}$$

1.1 Przechowywanie macierzy

Celem ograniczenia pamięciowego nakładu programu implementacje algorytmów, oraz funkcji pomocniczych w języku Julia operują na macierzy A przetrzymywanej jako `SparseMatrixCSC{Float64, Int64}`.

Dodatkowo, dla usprawnienia czasowego dostępu do elementów w algorytmach użyto macierzy transponowanej.

Oba usprawnienia nie mają wpływu na teoretyczny zarys działania zaimplementowanych metod, dlatego ich użycie zostanie pominięte w dalszej części sprawozdania.

2 Eliminacja Gaussa

2.1 Opis metody

Metoda eliminacji Gaussa pozwala na rozwiązanie układów równań pierwszego stopnia. Jej działanie podzielić można na dwa etapy.

Pierwszy etap działania metody polega na przekształceniu struktury macierzy wejściowej do postaci macierzy trójkątnej, uzyskując w ten sposób zera pod jej przekątną. Przekształcenie realizowane jest przy pomocy iteracji po przekątnej macierzy. Wybierany jest element z przekątnej $a_{i,i}$, a następnie zerowana jest kolumna poniżej tego elementu. Dla każdego wiersza do którego należy wyzerowana kolumna wybierany jest współczynnik, którego przemnożenie przez orginalną wartość w zerowanej kolumnie i wartość z wiersza i pozwala, po odjęciu od orginalnej wartości z zerowanej kolumny, osiągnąć 0. Ten sam współczynnik użyty jest następnie w podobny sposób dla każdej kolejnej wartości we wierszu, oraz w odpowiadającym miejscu wektora prawych stron.

Po zakończeniu powyższego etapu kolumny pod przekątną macierzy są wyzerowane, zmieniając tym samym jej postać do oczekiwanej macierzy trójkątnej.

Istotną własnością stworzonej w ten sposób macierzy jest istnienie co najwyżej jednego niezerowego elementu w jej ostatnim wierszu. Ponieważ macierz wraz z wektorem prawych stron jest w dalszym ciągu równoważnym orginalnemu układem równań bezproblemowe jest określenie wartości ostatniej niewiadomej w rozwiązaniu. Korzystając z otrzymanej wartości i podobnego rozumowania, możliwe jest określenie wartości przedostatniej niewiadomej na podstawie przedostatniego wiersza macierzy.

Powyższy zabieg stanowi istotę drugiego etapu działania metody. Po wykonaniu opisanych operacji dla każdego rzędu macierzy możliwe jest stopniowe ustalenie pełnego rozwiązania układu równań, kończąc tym samym metodę.

Z opisaną powyżej metodą powiązane są dwa istotne problemy. Pierwszy problem zaszyty jest w działaniu etapu początkowego - jeśli na przekątnej macierzy napotkana zostanie wartość 0, wyznaczenie współczynnika pozwalającego wyzerować poniższą kolumnę stanie się niemożliwe, co w efekcie uniemożliwi wyznaczenie rozwiązania.

W celu rozwiązania problemu możliwa jest zamiana wiersza z zerem na przekątnej na inny wiersz należący do macierzy. Zaimplementowany algorytm wybiera wiersz, w którym niezerowa wartość jest ponadto największa co do modułu. Opisana zamiana jest zawsze możliwa, przy założeniu, że macierz wejściowa nie jest osobliwa.

Kolejnym problemem opisanej metody jest jej złożoność obliczeniowa - ze względu na wielokrotnie zagnieżdżone pętle pierwszy etap ma złożoność $O(n^3)$, a etap końcowy $O(n^2)$, co łącznie nadaje eliminacjii Gaussa złożoność $O(n^3)$. Ze względu na specyficznią postać danych wejściowych istnieje jednak możliwość zoptymalizowania ostatecznej implementacji algorytmu.

2.2 Użyty algorytm

W celu odpowiedniego dobrania optymalizacji algorytmu przydatna jest analiza przykładowych danych wejściowych.

$$\begin{pmatrix}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} & C_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A_{1,5} & A_{1,6} & A_{1,7} & A_{1,8} & 0 & C_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A_{1,9} & A_{1,10} & A_{1,11} & A_{1,12} & 0 & 0 & C_{1,3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A_{1,13} & A_{1,14} & A_{1,15} & A_{1,16} & 0 & 0 & 0 & C_{1,5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & B_{1,1} & B_{1,2} & A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} & C_{2,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & B_{2,3} & B_{2,4} & A_{2,5} & A_{2,6} & A_{2,7} & A_{2,8} & 0 & C_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & B_{2,5} & B_{2,6} & A_{2,9} & A_{2,10} & A_{2,11} & A_{2,12} & 0 & 0 & C_{2,3} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & B_{2,7} & B_{2,8} & A_{2,13} & A_{2,14} & A_{2,15} & A_{2,16} & 0 & 0 & 0 & C_{2,5} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & B_{2,1} & B_{2,2} & A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4} & C_{3,1} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & B_{3,3} & B_{3,4} & A_{3,5} & A_{3,6} & A_{3,7} & A_{3,8} & 0 & C_{3,2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & B_{3,5} & B_{3,6} & A_{3,9} & A_{3,10} & A_{3,11} & A_{3,12} & 0 & 0 & C_{3,3} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & B_{3,7} & B_{3,8} & A_{3,13} & A_{3,14} & A_{3,15} & A_{3,16} & 0 & 0 & 0 & C_{3,5} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{3,1} & B_{3,2} & A_{4,1} & A_{4,2} & A_{4,3} & A_{4,4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{4,3} & B_{4,4} & A_{4,5} & A_{4,6} & A_{4,7} & A_{4,8} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{4,5} & B_{4,6} & A_{4,9} & A_{4,10} & A_{4,11} & A_{4,12} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{4,7} & B_{4,8} & A_{4,13} & A_{4,14} & A_{4,15} & A_{4,16}
\end{pmatrix}$$

Jak można zauważyć na powyższej macierzy liczby na całej długości przekątnej mogą być różne od zera. Oznacza to, że iteracja po przekątnej w eliminacji musi pozostać niezmieniona. Poszczególne kroki iteracyjne mogą jednak ulec uproszczeniu - ponieważ po wybraniu elementu z przekątnej następuje "zerowanie" kolumny pod nim, możliwe jest ograniczenie wewnętrznej pętli korzystając z faktu, że znaczna część kolumny już od początku złożona jest z zer.

Dla elementu $A_{1,1}$ konieczne jest wyzerowanie jedynie kolumn z liczbami $A_{1,5}$, $A_{1,9}$ i $A_{1,13}$, co redukuje maksymalny zakres pętli z całej wysokości macierzy do zadanej wielkości bloku. Istotne jest jednak rozpatrzenie sytuacji w przypadku elementów $A_{1,6}$, oraz $A_{1,11}$, zerowane kolumny są bowiem w ich przypadku połączeniem fragmentu bloku A_1, oraz bloku B_1.

Po wprowadzeniu omówionej korekty wyprowadzić można wzór na dolne ograniczenie zerowanej kolumny:

$$\texttt{min}\left(\left\lfloor\frac{y+1}{l}+1\right\rfloor\cdot l, n\right),$$

gdzie y jest współrzędną elementu z przekątnej, l - wielkością bloku, a n - rozmiarem macierzy.

W podobny do poprzedniego sposób możliwe jest ograniczenie zakresu najbardziej zagnieżdżonej w wykresie pętli. W większości sytuacji ostatnim elementem w danym wierszu jest oddalony o l miejsc element z bloku C. Wyjątek stanowi ostatni blok A_4, którego prawa strona jest równocześnie prawą stroną macierzy A.

Na podstawie powyższych operacji możliwe jest wyprowadzenie wzoru

$$\texttt{min}\left(y+l, n\right),$$

który pozwala ograniczyć wiersz macierzy z prawej strony.

Zakładając, że l jest stałą złożność dwóch wewnętrzych pętli w algorytmie ($O(2l)$, oraz $O(l)$) jest pomijalna. W związku z tym ostateczną złożonością algorytmu jest $O(n)$, ze względu na konieczność iteracji po całej przekątnej macierzy

Poniżej przedstawiony został pseudokod obrazujący metodę eliminacji Gaussa z wprowadzonymi usprawnieniami.

Algorytm 1: Rozwiązanie układu metodą Gaussa

```
1  function gaussel (A, n, l, b)
   Dane : Macierz rzadka A, o rozmiarze n i wielkości bloku l, oraz wektor prawych stron b
   Wynik: Rozwiązanie układu równań r
2  for y ← 1 to n − 1 do
3  |  if A_{y,y} = 0 then
4  |  |  error "zero"
5  |  rt ← min(y + l, n)
6  |  dn ← min(⌊(y+1)/l + 1⌋ · l, n)
7  |  for yy ← y + 1 to dn do
8  |  |  z ← A_{y,yy} / A_{y,y}
9  |  |  A_{y,yy} ← 0
10 |  |  for x ← y + 1 to rt do
11 |  |  |  A_{x,yy} ← A_{x,yy} − z * A_{x,y}
12 |  |  end
13 |  |  b_{yy} ← b_{yy} − z · b_y
14 |  end
15 end
16 for y = n downto 1 do
17 |  rt ← min(y + l, n)
18 |  sum ← 0
19 |  for x ← y + 1 to rt do
20 |  |  sum ← sum + A_{x,y} · r_x
21 |  end
22 |  r_y ← (b_y − sum) / A_{y,y}
23 end
24 return r
```

Jak można zauważyć powyższy algorytm nie jest zdolny rozwiązać układu równań, gdzie macierzy wejściowa A posiada liczbę 0 na przekątnej. Analogicznie do teoretycznej wersji algorytmu eliminacji możliwe jest zmodyfikowanie wersji zoptymalizowanej tak, aby wybierała ona inny wiersz z macierzy i kontynuowała obliczenia po dokonaniu zamiany.

W celu uniknięcia kosztownych obliczeniowo operacji wiersze macierzy nie zostają zamieniane miejscami. Zamiast tego wprowadzona została dodatkowa tablica *ord* o początkowych wartościach $1, 2, ..., n$, która pośredniczy we wszystkich odwołaniach do wiersza macierzy. W momencie gdy konieczne jest przestawienie wierszy macierzy dokonywana jest jedynie zamiana wartości dwóch odpowiadających im wartości z tablicy pośredniczącej.

Ze względu na możliwość przestawiania wierszy macierzy dotychczasowe założnie o prawym ograniczeniu wiersza oddalonym o maksymalnie l miejsc od elementu z przekątnej jest nieprawdziwe. Do zapewnienia poprawnego działania algorytmu jest więc konieczne zmodyfikowanie wzoru pozwalającego na określenie prawego ograniczenia:

$$\texttt{min}\left(\left\lfloor \frac{y+1}{l} + 2 \right\rfloor \cdot l, n\right).$$

Poniższy pseudokod prezentuje algorytm po zaimplementowaniu opisanych powyżej zmian.

Algorytm 2: Rozwiązanie układu metodą Gaussa z częściowym wyborem elementu głównego

function `pgaussel` (A, n, l, b)
Dane : Macierz rzadka A, o rozmiarze n i wielkości bloku l, oraz wektor prawych stron b
Wynik: Rozwiązanie układu równań r
$ord \leftarrow [1, 2, ..., n]$
for $y \leftarrow 1$ **to** $n - 1$ **do**
  $dn \leftarrow \texttt{min}\left(\left\lfloor \frac{y+1}{l} + 1 \right\rfloor \cdot l, n\right)$
  $rt \leftarrow \texttt{min}\left(\left\lfloor \frac{y+1}{l} + 2 \right\rfloor \cdot l, n\right)$
  for $yy \leftarrow y + 1$ **to** dn **do**
    $max \leftarrow |A_{y,ord[y]}|$
    $maxy \leftarrow y$
    for $i \leftarrow yy$ **to** dn **do**
      if $|A_{y,ord[i]}| > max$ **then**
        $maxy \leftarrow i$
        $max \leftarrow |A_{y,ord[i]}|$
    end
    if $|max| < \mathit{eps}(\mathit{Float64})$ **then**
      error "singular"
    $ord[y] \longleftrightarrow ord[maxy]$
    $z \leftarrow \frac{A_{y,ord[yy]}}{A_{y,ord[y]}}$
    $A_{y,ord[yy]} \leftarrow 0$
    for $x \leftarrow y + 1$ **to** rt **do**
      $A_{x,ord[yy]} \leftarrow A_{x,ord[yy]} - z \cdot A_{x,ord[y]}$
    end
    $b_{ord[yy]} \leftarrow b_{ord[yy]} - z \cdot b_{ord[y]}$
  end
end
for $y \leftarrow n$ **downto** 1 **do**
  $rt \leftarrow \texttt{min}\left(\left\lfloor \frac{ord[y]+1}{l} + 2 \right\rfloor \cdot l, n\right)$
  $sum \leftarrow 0$
  for $x \leftarrow y + 1$ **to** rt **do**
    $sum \leftarrow sum + A_{x,ord[y]} \cdot r_x$
  end
  $r_y \leftarrow \frac{b_{ord[y]} - sum}{A_{y,ord[y]}}$
end
return r

3 Rozkład LU

3.1 Opis metody

Do rozwiązania układu równań można wykorzystać rozkład LU macierzy.

Idea rozkładu LU macierzy A polega na przedstawieniu macierzy w formie iloczynu

$$A = LU,$$

gdzie L jest macierzą trójkątną dolną, o wszystkich wartościach na przekątnej równych 1, a U to macierz trójkątna górna.

Rozwiązanie zadanego układu równań liniowych polega na rozwiązaniu układu

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

Zaletą tego rozwiązania jest zmniejszenie kosztów obliczeniowych w przypadku, gdy dla tej samej macierzy A używane będą różne wektory prawych stron b.

W celu wyprowadzenia raozkładu LU macierzy A można posłużyć się opisaną wcześniej metodą eliminacji Gaussa. W trakcie działania metody macierz A zostanie stopniowo przekształcona w macierz trójkątną górną U.

Macierz trójkątną dolną L obliczyć można podczas przeprowadzania eliminacji Gaussa. W trakcie działania algorytmu mnożnik użyty do zerowania elementu $a_{i,j}$ powinien zostać zapisany we wierszu i, w kolumnie j w macierzy wynikowej L. Wykonanie tego kroku dla wszystkich mnożników pozwala na całkowite obliczenie macierzy trójkątnej dolnej.

Złożoność obliceniowa rozkładu to $O(n^3)$, a wyliczenie rozwiązania układu równań jest górnie ograniczone przez $O(n^2)$ operacji.

3.2 Użyty algorytm

Do opisanego powyżej algorytmu można wprowadzić szereg usprawnień poprawiających zarówno jego złożoność obliczeniową, jak i pamięciową.

Podczas wykonywania rozkładu macierz A zostaje stopniowo przekształcana na macierz trójkątną górną U. Oznacza to, że wszelkie wartości w macierzy pod jej przekątną powinny być równe 0. Możliwe jest więc wykorzystanie miejsc pod przekątną A zapisując w nich współczynniki z macierzy L. W ten sposób całość rozkładu może odbyć się na macierzy wejściowej niwelując potrzebę wykonania dodatkowych alokacji.

Warto zauważyć, że ten sposób przechowywania obu macierzy nie pozwala na równoczesne zapisanie ich przekątnych. Nie stanowi to jednak problemu, gdyż z założenia przekątna L składa się z samych jedynek, przez co jej zapisanie nie jest konieczne pod warunkiem wprowadzenia odpowiednich korekt w algorytmie.

Do usprawnienia złożoności obliczeniowej algorytmu można wykorzystać analogiczne jak w przypadku metody eliminacji Gaussa rozumowanie, oparte na specyficznej strukturze macierzy wejściowej. Podobnie jak w poprzedniej metodzie wynikiem usprawnienia jest osiągnięcie liniowej złożoności obliczeniowej $O(n)$ zarówno dla rozkładu, jak i rozwiązywaniu macierzy.

Algorytm 3: Obliczanie rozkładu LU macierzy

```
1  function ludecomp (A, n, l)
   Dane : Macierz rzadka A, o rozmiarze n i wielkości bloku l
   Wynik: Rozkład LU macierzy A
2  for y ← 1 to n − 1 do
3  |  if A_{y,y} = 0 then
4  |  |  error "zero"
5  |  rt ← min (y + l, n)
6  |  dn ← min (⌊(y+1)/l + 1⌋ · l, n)
7  |  for yy ← y + 1 to dn do
8  |  |  A_{y,yy} ← A_{y,yy} / A_{y,y}
9  |  |  for x ← y + 1 to rt do
10 |  |  |  A_{x,yy} ← A_{x,yy} − A_{y,yy} · A_{x,y}
11 |  |  end
12 |  end
13 end
14 return A
```

Algorytm 4: Rozwiązywanie układu równań liniowych z rozkładu LU

function `lusolve` (A, n, l, b)
Dane : Macierz rzadka A w rozkładzie LU, o rozmiarze n i wielkości bloku l, oraz wektor prawych stron b
Wynik: Rozwiązanie układu równań r
for $y \leftarrow 1$ **to** n **do**
  $sum \leftarrow 0$
  $lt \leftarrow \texttt{max}\left(l \cdot \left\lfloor \frac{y-1}{l} \right\rfloor - 1, 1\right)$
  for $x \leftarrow lt$ **to** $y - 1$ **do**
    $sum \leftarrow sum + A_{x,y} \cdot e_x$
  end
  $e_y \leftarrow b_y - sum$
end
for $y \leftarrow n$ **downto** 1 **do**
  $rt \leftarrow \texttt{min}\,(y + l, n)$
  $sum \leftarrow 0$
  for $x \leftarrow y + 1$ **to** rt **do**
    $sum \leftarrow sum + A_{x,y} \cdot r_x$
  end
  $r_y \leftarrow \frac{e_y - sum}{A_{y,y}}$
end
return r

Zaprezentowane powyżej algorytmy obciążone są inherentnym, związanym z działaniem "naiwnej" implementacji eliminacji Gaussa problemem, który uniemożliwia poprawne działanie dla macierzy wejściowej zawiarającej zero na przekątnej. Problem ten można jednak rozwiązać w sposób analogiczny do metody eliminacji Gaussa, stosując tablicę pośredniczącą w adresacji wiersza.

Warto zauważyć, że pomimo konieczności przestawiania wierszy macierzy jedynie podczas rozkładu, implementacje obu funkcji muszą zostać zmodyfikowane tak, aby poprawnie używały tablicy pośredniczącej *ord*.

4 Wyniki

Wszystkie zamieszczone poniżej wyniki wyliczone zostały przy rozmiarze bloku $l = 4$.

Poniższa tabela prezentuje złożoności czasowe, oraz obliczeniowe dla obu wymionionych wcześniej sposobów realizacji eliminacji Gaussa. Dla porównania wyniki zestawione są z funkcją \ ze standardowej biblioteki języka *Julia*.

Czasowa i pamięciowa złożoność metody eliminacji Gaussa						
n	$x = A\backslash b$		bez wyboru		z wyborem	
16	1.305955264	2.560 KB	$2.8141 \cdot 10^{-5}$	3.472 KB	0.001105779	3.680 KB
100	1.337779713	81.504 KB	0.000218657	42.832 KB	0.00045497	82.352 KB
500	1.309763193	2.006 MB	0.00115739	194.960 KB	0.002111832	400.944 KB
1000	1.371330423	8.012 MB	0.002992632	398.928 KB	0.005141568	812.880 KB
5000	3.465335424	200.060 MB	0.045766066	2.031 MB	0.046768635	4.109 MB
10000	-	-	0.173971812	4.071 MB	0.204630223	8.229 MB
50000	-	-	6.057894036	20.391 MB	5.459760897	41.189 MB

Zaimplementowane metody sprawdzono również pod względem dokładności. W poniższej tabeli zestawione są błędy względne wyników, jakie otrzymano dla posczególnych implementacji, wraz z błędami generowanymi przez standardową funkcję.

Błąd względny metody eliminacji Gaussa			
n	$x = A\backslash b$	bez wyboru	z wyborem
16	$2.254873622441467 \cdot 10^{-16}$	$5.509324452557031 \cdot 10^{-16}$	$1.6184142622847344 \cdot 10^{-16}$
100	$2.5510982866352576 \cdot 10^{-16}$	$4.145727976299207 \cdot 10^{-15}$	$2.615512026824473 \cdot 10^{-16}$
500	$2.139016888732267 \cdot 10^{-16}$	$7.056165653029498 \cdot 10^{-15}$	$2.315553705957619 \cdot 10^{-16}$
1000	$2.2720274092844913 \cdot 10^{-16}$	$3.506102944338744 \cdot 10^{-15}$	$2.342807347416358 \cdot 10^{-16}$
5000	$2.3018862032594885 \cdot 10^{-16}$	$4.0823329545176814 \cdot 10^{-14}$	$2.3093706409668854 \cdot 10^{-16}$
10000	-	$6.9651196613135928 \cdot 10^{-15}$	$2.3365908972749264 \cdot 10^{-16}$
50000	-	$8.2610108930327355 \cdot 10^{-15}$	$2.3545890915380506 \cdot 10^{-16}$

Powyższe dane pozwalają zauważyć, że wynik działania "naiwnej", nie przestawiającej wierszy macierzy, implementacji metody eliminacji Gaussa jest obarczony większym błędem niż wynik eliminacji z wyborem elementu głównego.

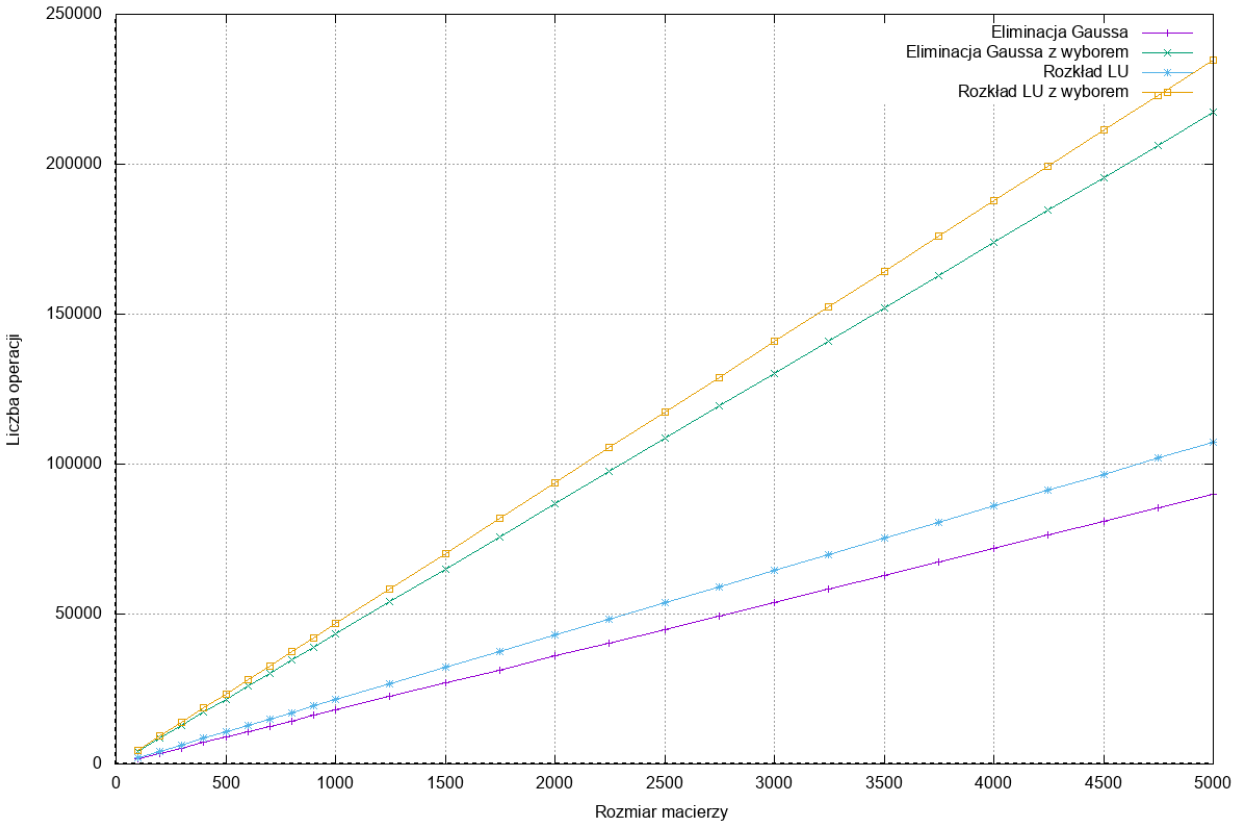
Ponieważ ta prawidłowość zachodzi również w przypadku rozkładu LU, to zestawienie błędów zostało w jego przypadku pominięte.

Złożności czasowe i pamięciowe dla posczególnych etapów, oraz pełnego rozwiązania równania metodą rozkładu LU zostały przedstawione poniżej. Ze względu na ilość danych wyniki podzielone zostały na dwie tabele.

Czasowa i pamięciowa złożoność metody rozkładu LU bez wyboru								
n	$x = A\backslash b$		rozkład		rozwiązanie		razem	
16	1.33726496	2.560 KB	$2.3862 \cdot 10^{-5}$	3.264 KB	$7.879 \cdot 10^{-6}$	416B	0.00519626	3.680 KB
100	1.301305135	81.504 KB	0.000159609	35.696 KB	$4.3778 \cdot 10^{-5}$	5.328 KB	0.006205516	41.040 KB
500	1.31011346	2.006 MB	0.001021301	158.960 KB	0.000230276	59.856 KB	0.007553597	218.832 KB
1000	1.35047106	8.012 MB	0.002152966	326.960 KB	0.000305311	127.792 KB	0.007335098	454.784 KB
5000	3.529011924	200.060 MB	0.036166835	1.671 MB	0.001808476	671.696 KB	0.044450964	2.343 MB
10000	-	-	0.170446973	3.351 MB	0.003296519	1.352 MB	0.185011678	4.703 MB
50000	-	-	5.674600029	16.791 MB	0.015908801	6.792 MB	5.890401637	23.583 MB

Czasowa i pamięciowa złożoność metody rozkładu LU z wyborem								
n	$x = A\backslash b$		rozkład		rozwiązanie		razem	
16	1.314837244	2.560 KB	$4.6705 \cdot 10^{-5}$	3.504 KB	$1.0879 \cdot 10^{-5}$	416B	0.005488974	3.936 KB
100	1.305518518	81.504 KB	0.00037436	70.000 KB	$7.1287 \cdot 10^{-5}$	10.576 KB	0.007081155	80.592 KB
500	1.319163012	2.006 MB	0.001448348	337.328 KB	0.000276705	87.504 KB	0.007205985	424.848 KB
1000	1.365444179	8.012 MB	0.003566094	685.296 KB	0.000542546	183.440 KB	0.009503735	868.768 KB
5000	3.558162046	200.060 MB	0.043573427	3.469 MB	0.002856909	951.344 KB	0.052583412	4.421 MB
10000	-	-	0.192189908	6.949 MB	0.004405843	1.911 MB	0.240070846	8.861 MB
50000	-	-	6.439434712	34.789 MB	0.031810784	9.591 MB	6.270859826	44.381 MB

Poniższy wykres prezentuje dla każdego z algorytmów liczbę operacji potrzebnych do rozwiązania układu o zadanej wielkości. Operacje liczone zostały jako przebiegi iteracyjne najbardziej zagnieżdżonych w algorytmie pętli. Dla opisanego w pierwszej sekcji algorytmu 1 licznik iteracji został zwiększany za każdym wykonaniem linii 11 i 20 w pseudokodzie funkcji.



Rys. 1. Algorytmy przed i po modyfikacji

5 Wnioski

Analiza powyższego wykresu potwierdza liniową złożoność obliczeniową zaimplementowanych algorytmów.

Porównania danych w powyższych tabelach pozwalają stwierdzić, że zarówno dla rozkładu LU, jak i eliminacji Gaussa wersja metody z wyborem elementu głównego pozwala uzyskać dokładniejsze wyniki wyniki kosztem zwiększonej złożoności obliczeniowej i pamięciowej.

Zestawiając ze sobą statystyki prowadzone dla eliminacji Gaussa, oraz rozkładu LU zauważyć można, że obliczanie równania przy użyciu rozkładu niesie ze sobą większy nakład czasowy i pamięciowy. Istotne jest jednak, że obserwację tę poczyniono analizując wyniki dla rozwiązania jednego układu równań przez każdą z metod. W przypadku, gdy wymagane jest rozwiązanie identycznego układu, o zmienionym wektorze prawych stron wynik byłby zgoła odwrotny, gdyż sam rozkład wymagałby policzenia tylko raz. Metoda eliminacji Gaussa musiałaby zostać zastosowana w całości dla każdego z układów, pomimo ich nieznacznych różnic.

Porównanie wyników zaimplementowanych w pakiecie funkcji z czasem i złożonością pamięciową algorytmu rozwiązania układu zaprojektowanego dla generalnego zastosowania pozwala zauważyć, że optymalizacja dla konkretnie zadanej postaci danych jest opłacalna. We wszystkich sprawdzonych przypadkach fukcja ze standardowej biblioteki języka wymagała, dla wystarczająco dużych danych, więcej czasu i pamięci tymczasowej do wyprowadzenia wyników. Dla dużych macierzy użycie standardowej funkcji nie pozwoliło na rozwiązanie układu, ze względu na próbę alokacji pamięci przekraczającą dostępne na użytym systemie 16 GiB.