Obliczenia Naukowe - Lista 5

Paweł Dychus (244941)

Styczeń 2020

Opis problemu

Rozwiązanie układu równań liniowych

$$\boldsymbol{A}x = \boldsymbol{b}$$

dla macierzy $A \in \mathbb{R}^{n\times n}$ i wektora prawych stron $b \in \mathbb{R}^n$, $n > 4$. Macierz A jest postaci:

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

Gdzie dla $v = n/l$, czyli n jest podzielne przez l, oraz $l \geq 2$, mamy następujące macierze kwadratowe:

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

ii) $\boldsymbol{0} \in \mathbb{R}^{l\times l}$ – macierz zerowa,

iii) $\boldsymbol{B}_i \in \mathbb{R}^{l\times l}$, $i = 2, \ldots, v$ – macierze z niezerowymi dwoma ostatnimi kolumnami:

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

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

$$\boldsymbol{C}_i = \begin{pmatrix} c^i_1 & 0 & 0 & \cdots & 0 \\ 0 & c^i_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c^i_{l-1} & 0 \\ 0 & \cdots & 0 & 0 & c^i_l \end{pmatrix}. \tag{3}$$

Do rozwiązania układu równań $\boldsymbol{A}x = \boldsymbol{b}$ wykorzystać dwie metody:

a. Metodą eliminacji Gaussa bez i z częściowym wyborem elementu głównego.

b. Rozkład $\boldsymbol{LU}$ w wersji bez i z częściowym wyborem elementu głównego i rozwiązać $\boldsymbol{LU}x = \boldsymbol{b}$.

Przechowywanie macierzy

Jak łatwo zauważyć, macierz $\boldsymbol{A}$ jest macierzą rzadką, czyli ma wiele elementów zerowych, dokładnie:

$$v \cdot l^2 + (v-1) \cdot 2l + (v-1) \cdot l = n(l+3) - 3l$$

Przechowywanie takiej macierzy w naiwnej postaci zabiera znaczną ilość pamięci(macierz $n \times n$). Aby to zoptymalizować wykorzystamy `SparseMatrixCSC` z języka `Julia`. Ta struktura danych przechowuje tylko wartości niezerowe w skompresowanym porządku kolumnowym, efektywnie zmniejszając złożoność pamięciową z $O(n^2)$ do $O(n)$.

Opis algorytmów

Metoda eliminacji Gaussa do rozwiązywania układu równań

Podstawowa metoda eliminacji Gaussa polega na zerowaniu kolejnych $k-1$ elementów w k wierszu, poprzez odejmowanie stronami równań, w celu doprowadzenia do postaci macierzy trójkątnej górnej, której rozwiązanie jest oczywiste.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n-1} & a_{2,n} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n-1} & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \cdots \\ b_{n-1} \\ b_n \end{pmatrix} \tag{4}$$

$$\vdots$$

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\ 0 & d_{2,2} & d_{2,3} & \cdots & d_{2,n-1} & d_{2,n} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & d_{n-1,n-1} & d_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & d_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ h_2 \\ \cdots \\ h_{n-1} \\ h_n \end{pmatrix} \tag{5}$$

Gdzie $b_{i,i}$ i h_i to produkt odejmowania kolejnych równań. Na przykład: $d_{2,i} = a_{2,i} - (a_{2,1}/a_{1,1}) \cdot a_{1,i}$, $d_{2,1} = 0$. W szczególności $d_{1,i} = a_{1,i}$, podobnie h.

Aby możliwe było wykonanie powyższej metody każdy z elementów diagonalnych w macierzy musi być różny od zera. W momencie kiedy tak nie jest wymagana jest modyfikacja algorytmu. Szukamy takiego elementu który w aktualnie przeszukiwanej kolumnie nie posiada zera i zamieniamy z aktualnym wierszem. Dokładniej w i-tym kroku algorytm szuka w i-tej kolumnie element(zwany *elementem głównym*) o największej wartości bezwzględnej i wiersz z takim elementem zamienia z miejscem wiersza na pozycji i. Zamiana jest możliwa, ponieważ w przeciwnym wypadku macierz jest osobliwa.

Po otrzymaniu macierzy trójkątnej górnej wystarczy rozwiązać układ równań prostym algorytmem *podstawiania wstecz*. Jest on zdefiniowany takim wzorem:

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

dla wierszy i od n do 1.

Metoda eliminacji Gaussa ma złożoność $O(n^3)$, a algorytm *podstawiania wstecz* $O(n^2)$. Zatem, aby rozwiązać układ równań, trzeba wykonać łącznie $O(n^3)$ operacji.

Optymalizacja względem zadanej macierzy $\boldsymbol{A}$

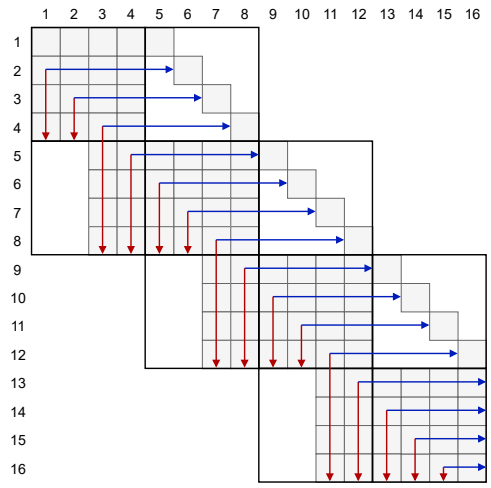
Macierz $\boldsymbol{A}$ ma postać trójdiagonalno-blokową, pozwoli to nam znacznie zmniejszyć złożoność obliczeniową. Wystarczy zaobserwować, że dla danej stałej l można zmniejszyć ilość iteracji względem kolumn i samych wierszy. Rozumowanie przedstawia poniższy rysunek.

Na przykładzie pokazanym na rysunku zastosowano $l = 4$ i $n = 16$ (w celu wizualiacji). Strzałka czerwona (wertykalna), przedstawia zakres zamienianych wierszy, a niebieska (horyzontalna), przedstawia elementy w wierszach, które będą podlegać zmianom, w kroku danym indeksem wiersza.
Strzałka niebieska zmienia się iteracyjnie dla danego r w taki sposób: $r + l$, z ograniczeniem górnym równym n. Widać to po macierzy przekątnej $\boldsymbol{C}_i$, gdzie indeks przekątnej w i-tym wierszu, to przesunięcie o długość bloku $\boldsymbol{A}_i$. Matematycznie można to przedstawić tak:

$$x\_max(r) = min\left\{r + l, n\right\}$$

Wybór wierszy jest równie oczywisty, wystarczy zwiększyć zakres o długość l gdy $k \equiv (l-1) \mod l$, czyli w momencie, kiedy pojawiają się niezerowe wartości, z 2 ostatnich kolumn, następującej macierzy blokowej $\boldsymbol{B}_i$. Ogólny wzór wygląda tak:

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



Rysunek 1 – Zakres iteracji w Gaussie dla macierzy $\boldsymbol{A}$

Jak można zauważyć, dla l określonego stałą, eliminacja Gaussa osiągnie złożoność obliczeniową $O(n)$. *Podstawianie wstecz*, które również można uściślić korzystając z faktu, że wartości niezerowe będą od przekątnej macierzy do przekątnej macierzy blokowej $\boldsymbol{C}_i$, również osiąga złożoność $O(n)$.

Poniżej przedstawiono pseudokod implementacji algorytmu eliminacji gaussa, dla macierzy A. Przypadek bez wyboru częściowego elementu głównego. W komentarzach podano złożoność algorytmiczną poszczególnych linii kodu.

Funkcja `eliminacja_gaussa` :
Dane:

$\boldsymbol{A}$	–	macierz dana określonym wzorem z zadania
`b`	–	wektor prawych stron
`n`	–	rozmiar macierzy $\boldsymbol{A}$
`l`	–	rozmiar bloku macierzy $\boldsymbol{A}$

Wyniki:

x	–	wektor rozwiązania układu $\boldsymbol{A}x = \boldsymbol{b}$

Algorithm 1: Eliminacja Gaussa

```
function eliminacja_gaussa(A, b, n, l)
    for k ← 1 to n − 1 ;                                              // O(n)
     do
        y_max ← min(l + l · ⌊(k+1)/l⌋, n);
        x_max ← min(k + l, n);
        for i ← k + 1 to y_max ;                                      // O(l)
         do
            if A[k][k] = 0 then
                error zero na przekątnej!
            c ← A[i][k]/A[k][k];
            A[i][k] ← 0 ;
            for j ← k + 1 to x_max ;                                  // O(l)
             do
                A[i][j] ← A[i][j] − c · A[k][j];
            b[i] ← b[i] − c · b[k];
    for i ← n downto 1 ;                          // Podstawianie wstecz, O(n)
     do
        x_max ← min(i + l, n);
        for j ← k + 1 to x_max ;                                      // O(l)
         do
            suma ← suma + x[i] · A[i][j];
        x[i] ← (b[i] − suma)/A[i][i];
    return x ;
```

Częściowy wybór elementu głównego

Metoda eliminacji Gaussa z częściowym wyborem elementu głównego polega na zmianie wierszy w danym i-tym kroku w celu:

1. Uniknięcia zerowych wartości na przekątnej, które terminują poprzednią metodę.
2. Redukcji błędu, przez szukanie maksymalnej wartości współczynnika.

Po zamianie wierszy w każdym i-tym kroku, dalszy algorytm pozostaje niezmieniony.
W praktyce zamieniane wierszy jest procesem dosyć czasochłonnym, dlatego wykorzystamy tablicę permutacji wierszy (p), która będzie pamiętać pod danym indeksem i, do którego wiersza w danym kroku się odnieść.

Wymagana jest zmiana wartości x_max, ponieważ poprzednie ograniczenie nie wystarcza. Zamiana wierszy, może już w pierwszej iteracji wymagać odwołania do komórek o indeksie $2 * l$. Na przykład, zamiana wiersza 4 z 1 , wymaga iteracji do 8-ej kolumny włącznie ($l = 4$). Schemat wyboru x_max pozostaje jednak intuicyjny. Wystarczy zauważyć, że tak jak w poprzednim przypadku, zakres iteracji w wierszach rośnie dopiero, gdy niezerowe wartości macierzy blokowej $\boldsymbol{B}_i$ wchodzą w zakres y_max, który z oczywistych powodów nie zmieni się względem poprzedniego algorytmu. W praktyce znaczy to, tyle, że wystarczy zwiększyć zakres iteracji w wierszach, dwukrotnie względem wzoru na y_max. Daje to nam:

$$x\_max(r) = min\left\{2l + l \cdot \left\lfloor \frac{k+1}{l} \right\rfloor, n\right\}$$

Poniżej pseudokod zmodyfikowanego algorytmu eliminacji Gaussa, z dodanym częściowym wyborem elementu głównego, na macierzy $\boldsymbol{A}$ z zadania.

Funkcja `eliminacja_gaussa_B` :
Dane:

$\boldsymbol{A}$	–	macierz dana określonym wzorem z zadania
`b`	–	wektor prawych stron
`n`	–	rozmiar macierzy $\boldsymbol{A}$
`l`	–	rozmiar bloku macierzy $\boldsymbol{A}$

Wyniki:

x	–	wektor rozwiązania układu $\boldsymbol{A}x = \boldsymbol{b}$

Algorithm 2: Eliminacja Gaussa z częściowym wyborem elementu głównego

function `eliminacja_gaussa_B`($\boldsymbol{A}$, $\boldsymbol{b}$, n, l)
  p $\leftarrow [1,\cdots,n]$
  for k $\leftarrow 1$ **to** $n-1$; // $O(n)$
  do
    $y\_max \leftarrow \min\left(l + l \cdot \lfloor \frac{k+1}{l} \rfloor, n\right)$;
    $x\_max \leftarrow \min\left(2l + l \cdot \lfloor \frac{k+1}{l} \rfloor, n\right)$;
    $[maximum, p_{max}] = max(abs(A[k : y\_max, k]))$; `// el. główny, wybór częściowy`
    if $maximum = 0$ **then**
      error Macierz osobliwa
    swap (p[k], p[p_{max}]);
    for i $\leftarrow$ k $+ 1$ **to** y_max ; // $O(l)$
    do
      c $\leftarrow \boldsymbol{A}$[p[i]][k]/$\boldsymbol{A}$[p[k]][k];
      $\boldsymbol{A}$[p[i]][k] $\leftarrow 0$;
      for j $\leftarrow$ k $+ 1$ **to** x_max ; // $O(l)$
      do
        $\boldsymbol{A}$[p[i]][j] $\leftarrow \boldsymbol{A}$[p[i]][j] $-$ c $\cdot \boldsymbol{A}$[p[k]][j];
      $\boldsymbol{b}$[p[i]] $\leftarrow \boldsymbol{b}$[p[i]] $-$ c $\cdot \boldsymbol{b}$[p[k]];
  for i $\leftarrow n$ **downto** 1 ; `// Podstawianie wstecz,` $O(n)$
  do
    $x\_max \leftarrow \min\left(2l + l \cdot \lfloor \frac{k+1}{l} \rfloor, n\right)$;
    for j $\leftarrow$ k $+ 1$ **to** x_max ; // $O(l)$
    do
      suma $\leftarrow$ suma $+ \boldsymbol{x}$[i] $\cdot \boldsymbol{A}$[p[i]][j];
    $\boldsymbol{x}$[i] $\leftarrow (\boldsymbol{b}$[p[i]] $-$ suma$)/\boldsymbol{A}$[p[i]][i];
  return $\boldsymbol{x}$;

Jeżeli zaś chodzi o złożoność obliczeniową, to chociaż sama długość iteracji (horyzontalnie) się wydłuży, to nie wpłynie to na ogół, przy założeniu, że l jest stałą. Algorytm podstawiania wstecz również się nie zmieni, poza faktem, że należy się odwołać do tablicy permutacji, które jest wykonywane w czasie stałym. Daje to nam ponownie złożoność $O(n)$.

Rozkład LU

Idea rozkładu $\boldsymbol{LU}$ polega na przedstawieniu macierzy $\boldsymbol{A}$ iloczynem macierzy trójkątnych(górnej i dolnej):

$$\boldsymbol{A} = \boldsymbol{LU}.$$

W naszym przykładzie będziemy stosować rozkład *Doolittle'a*, mianowicie $\boldsymbol{L}$ będzie macierzą trójkątną jedynkową dolną, a $\boldsymbol{U}$ macierzą trójkątną górną. Taka postać jak się okazuje, jest prosta do liczenia.

Rozkład możemy uzyskać z pomocą znanej już nam metody eliminacji Gaussa. Macierz $\boldsymbol{A}$ zostaje przekształcona do postaci macierzy trójkątnej górnej $\boldsymbol{U}$, natomiast macierz dolną $\boldsymbol{L}$, wyznaczymy pamiętając mnożniki, użyte do zerowana kolejnych elementów, w kolumnach pod diagonalą (tzn. zamiast zer w $a_{i,j}$, wstawimy mnożniki). Całość możemy wykonać na jednej strukturze danych, czyli wejściowej $\boldsymbol{A}$. Jedynie trzeba pamiętać, że wartości na diagonali będą przedstawiać wartości $\boldsymbol{U}$, a nie $\boldsymbol{L}$, co w praktyce nie jest szczególnym problemem.

Złożoność obliczeniowa wyznaczenia rozkładu $\boldsymbol{LU}$ wynosi $O(n^3)$, pozwala on jednak skutecznie liczyć układy równań w których macierz jest stała, a zmienia się jedynie wektor prawych stron. Wtedy rozwiązywanie układu równań sprowadza się do prostego podstawienia postaci:

$$\begin{cases} \boldsymbol{L}z = \boldsymbol{b} \\ \boldsymbol{U}x = z \end{cases} \tag{6}$$

Ze względu na postacie macierzy $\boldsymbol{LU}$ rozwiązanie takiego równania można wykonać w czasie $O(n^2)$

Optymalizacja względem zadanej macierzy $\boldsymbol{A}$

Rozkład $\boldsymbol{LU}$ wyznaczymy zmieniając minimalnie wyżej przedstawione algorytmy eliminacji Gaussa (w przypadku z wyborem elementu głównego jak i bez). Mianowice wystarczy, że zamiast zerować współczynnik równania, wpiszemy w niego wartość mnożnika. A linijkę ze zmianą prawego wektora wyrzucimy (rozpatrujemy rozkład samej macierzy, a nie układ równań).

Mając już rozkład $\boldsymbol{LU}$ możemy przejść do jego rozwiązania: równanie 6. Algorytm rozwiązywania macierzy górnej jest już nam znany, jest to *podstawianie wstecz* i jego użyjemy do rozwiązania $\boldsymbol{U}x = z$. W przypadku $\boldsymbol{L}z = \boldsymbol{b}$, musimy użyć algorytmu *podstawiania w przód*, ma on podobną ideą co algorytm wstecz, zmieni się jedynie kolejność indeksowania jak i ogranicznik zakresu, dla danej macierzy $\boldsymbol{A}$ z zadania.

Każdy krok naszego zoptymalizowanego algorytmu *podstawiania w przód*, będzie kończył sumowanie na diagonali, a zaczynał na pozycjach jak w oryginalnej macierzy $\boldsymbol{A}$. Mając to na uwadze, możemy łatwo zauważyć, że dla pierwszych l elementów zaczniemy od indeksu równego 1, w kolejnym kroku od $l-1$, a w jeszcze kolejnych od $ml-1, m \in \mathbb{Z}$. Można to przedstawić takim wzorem:

$$x\_min(r) = max\left\{ l \cdot \left\lfloor \frac{r-1}{l} \right\rfloor - 1, 1 \right\}$$

Poniżej przedstawiono pseudokod implementacji algorytmu rozwiązywania układu $\boldsymbol{LU}$, dla macierzy A. W komentarzach podano złożoność algorytmiczną poszczególnych linii kodu.

Funkcja `rozwiązanie_LU` :
Dane:

$\boldsymbol{A}$	–	macierz rozkładu $\boldsymbol{LU}$
`b`	–	wektor prawych stron
`n`	–	rozmiar macierzy $\boldsymbol{A}$
`l`	–	rozmiar bloku macierzy $\boldsymbol{A}$

Wyniki:

`x`	–	wektor rozwiązania układu $\boldsymbol{LU}x = \boldsymbol{b}$

Algorithm 3: Rozwiązywanie układu równiań z użyciem rozkładu $\boldsymbol{LU}$

function `rozwiazanie_lu`($\boldsymbol{A}$, $\boldsymbol{b}$, n, l)
  for i $\leftarrow 1$ **to** n ; // `Podstawianie wprzód,` $O(n)$
  do
    suma $\leftarrow 0$;
    $x\_min \leftarrow \min\left(l \cdot \lfloor \frac{i+1}{l} \rfloor, n\right)$;
    for j $\leftarrow x\_min$ **to** $i-1$; // $O(l)$
    do
      suma $\leftarrow$ suma $+ z[j] \cdot \boldsymbol{A}[i][j]$;
    $z[i] \leftarrow \boldsymbol{b}$[i] $-$ suma; // `diagonala w` $\boldsymbol{L}$ `ma wartość 1`
for i $\leftarrow n$ **downto** 1 ; // `Podstawianie wstecz,` $O(n)$
do
  suma $\leftarrow 0$;
  $x\_max \leftarrow \min(i+l, n)$;
  for j $\leftarrow i+1$ **to** x_max ; // $O(l)$
  do
    suma $\leftarrow$ suma $+ x[j] \cdot \boldsymbol{A}[i][j]$;
  $x[i] \leftarrow (z$[i] $-$ suma$)/\boldsymbol{A}[i][i]$;
return $\boldsymbol{x}$;

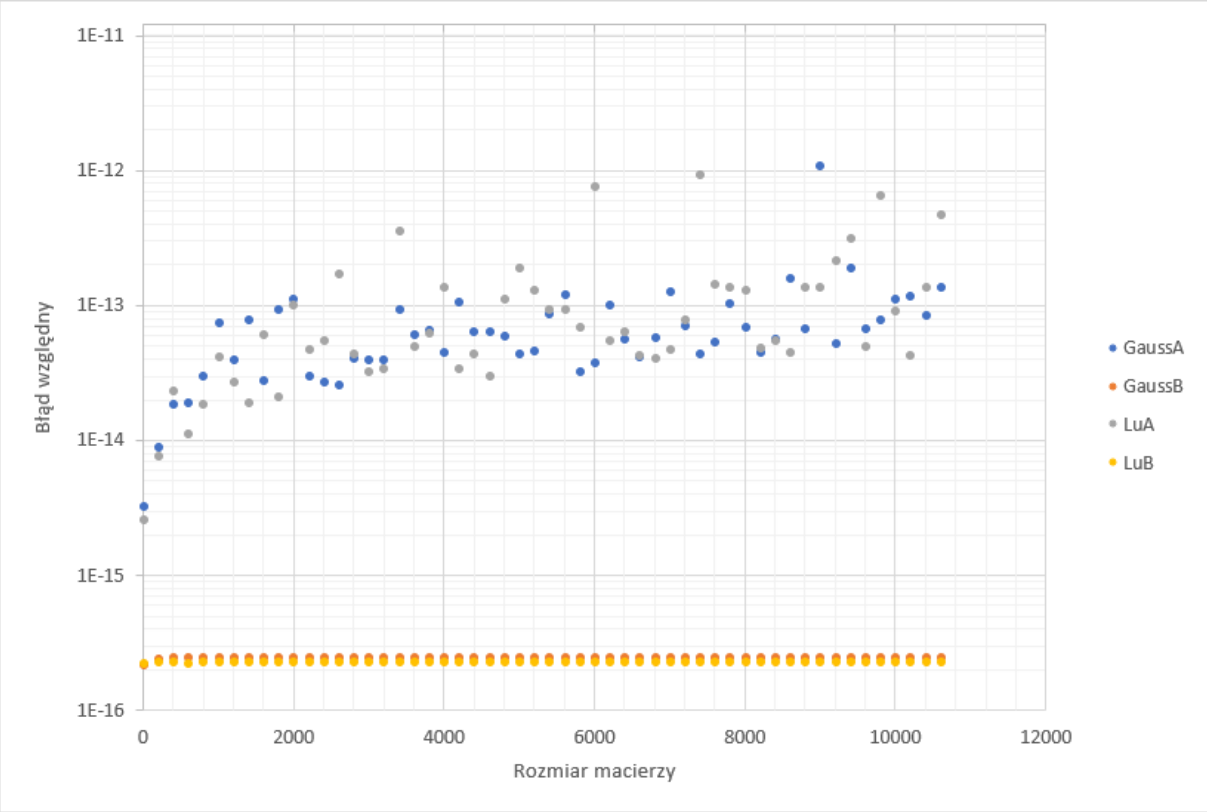
Złożoność obliczeniowa samego rozkładu $\boldsymbol{LU}$ wynosi $O(n)$, ponieważ jest ono nieznaczącą pod względem złożoności, modyfikacją algorytmu eliminacji Gaussa. Co zaś się tyczy algorytmu rozwiązywania układu równań z użyciem $\boldsymbol{LU}$, również otrzymujemy $O(n)$. Jest to spowodowane tym, że oba algorytmy, *podstawianie wstecz* jak i *podstawianie wprzód* mają złożoność liniową, dzięki wartości stałej l, ograniczającej zakres iteracji podczas sumowania.

Wyniki

Poniżej przedstawiono czasy oraz użytą pamięć dla zaimplementowanych algorytmów jak i porównanie z funkcją biblioteczną. Dane były zbierane na macierzach $\boldsymbol{A}$, określonych z zadania, generowanych losowo z użyciem dołączonego modułu. Przyjęto za wartość l stałą równą 4. N zmieniało się od 16 do 10616, ze skokami o wartość 200. Każdy krok był powtarzany 32-krotnie, żeby zminimalizować rozbieżność.

Błąd względny

Poniższy wykres przedstawia błędy względne wszystkich 4 metod. Ze względu na różnicę wielkości danych, zastosowano skalę logarytmiczną.



Rysunek 2 – Błąd względny

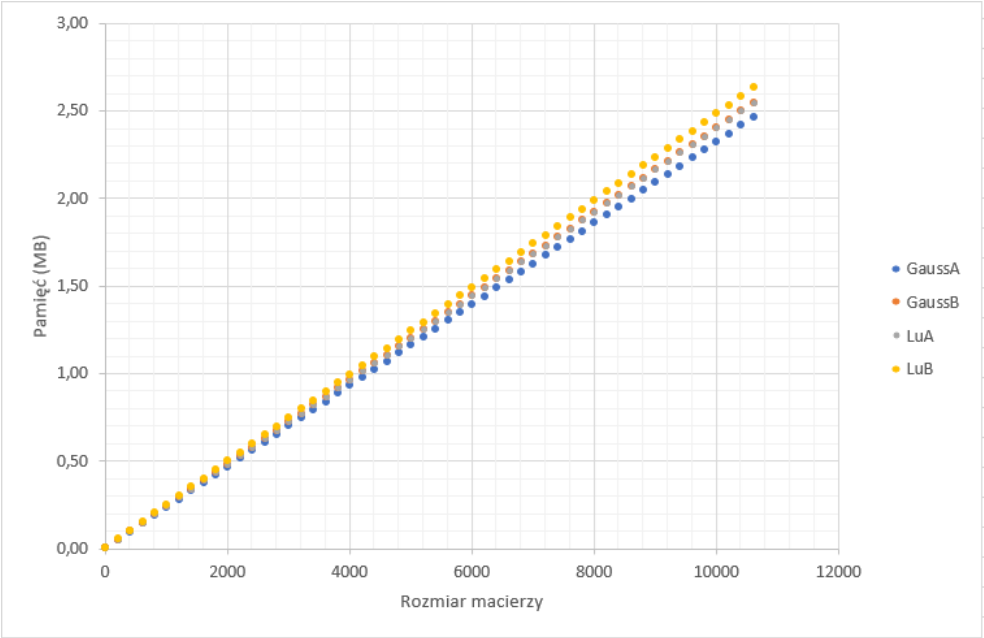
Jak widać, w obu algorytmach, częściowy wybór elementu głównego, pozwolił zminimalizować błąd obliczeniowy. W algorytmie GaussB i LuB, błąd wacha się między $2,21 \cdot 10^{-16}$, a $2,29 \cdot 10^{-16}$. W przypadku A, błąd wynosi od $1,05 \cdot 10^{-12}$, do $3,2 \cdot 10^{-15}$. Widać to dokładniej na załączonej tabeli: 4.

Tablica 4 – Minimum i maximum błędu względnego

Algorytm	minimum	maximum
GaussA	$3,211 \cdot 10^{-15}$	$1,05883 \cdot 10^{-12}$
GaussB	$2,15057 \cdot 10^{-16}$	$2,46098 \cdot 10^{-16}$
LuA	$2,53481 \cdot 10^{-15}$	$9,16634 \cdot 10^{-13}$
LuB	$2,21139 \cdot 10^{-16}$	$2,28694 \cdot 10^{-16}$

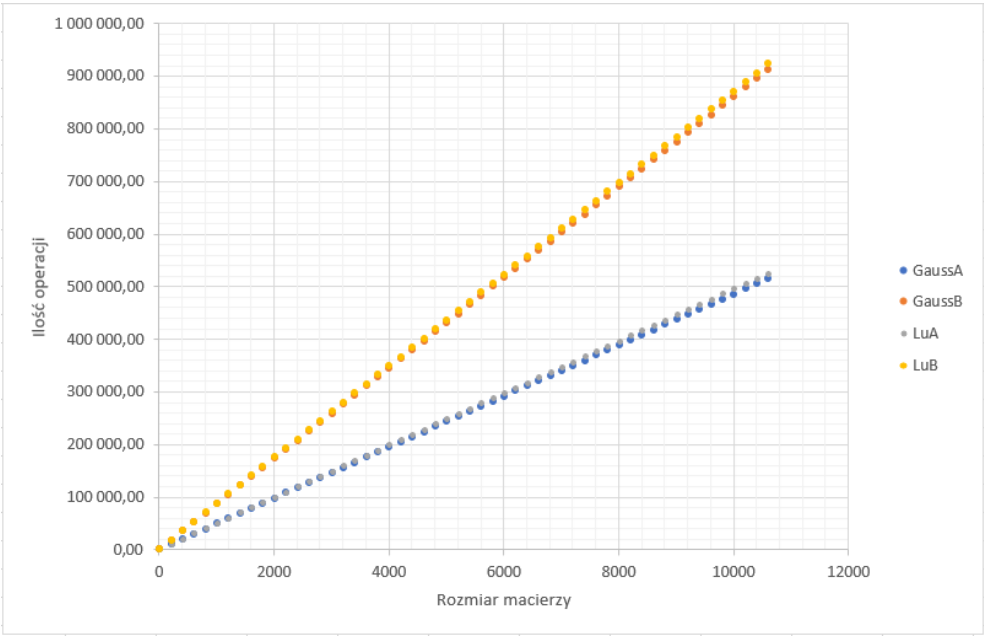
Złożoność

Poniższe wykresy przedstawiają złożoność pamięciową oraz licznik operacji arytmetycznych. W pierwszym przypadku dane podano w megabajtach(MB)



Rysunek 3 – Użyta pamięć

Zgodnie z przewidywaniami, użyta pamięć rośnie liniowo w każdym przypadku. Widać również, że metody z wyborem elementu głównego, zabierają nieco więcej pamięci. W tym przypadku LuA, miało praktycznie identyczną ilość rezerwowanej pamięci co GaussB.



Rysunek 4 – Liczba operacji

Jak widać, metody bez wyboru elementu głównego mają zmniejszoną ilość operacji. Licza operacji LuA jest minimalnie większa od GaussA i tak samo liczba operacji w LuB jest minimalnie większa od GaussB.

Wnioski

Najbardziej oczywisty staje się fakt, że wybieranie elementu głównego jest dobrym sposobem, żeby zminimalizować błąd względny i zachować dobry wskaźnik uwarunkowania wykonywanych obliczeń. Metody bez wybierania elementu głównego, skutecznie doprowadziły do stworzenia źle uwarunkowanego zadania. Nie jest to jednak rozwiązanie idealne, algorytmy z 'pivotem', zużywają więcej pamięci, oraz wykonują większą ilość operacji.

Warto również zaznaczyć, że metoda ***LU*** jest zdecydowanie lepsza od algorytmu Gaussa, jeżeli mamy do rozwiązania wiele układów z jedną ustaloną macierzą. Wykonywanie algorytmu GauusA lub GaussB, mija się w takim przypadku z celem, podzczas gdy, rozkład ***LU*** pozwoli wykonać jeden 'stosunkowo wolny' rozkład i wiele 'ekstremalnie szybkich' rozwiązań układu, dla różnych prawych wektorów.

Można również stwierdzić, że algorytmy dostosowane pod zadaną macierz przewyższają ogólny algorytm, czy to ***LU***, czy eliminacji Gaussa, które dla wyżej przeprowadzonych eksperymentów, nie są nawet w stanie wykonać się dla większych n (tudzież wykonują się ekstremalnie długo), ze względu na ich złożoność obliczeniową jak i pamięciową.

PS. Co zaś się tyczy algorytmów bibliotecznych Julii (\,LU), korzystają one z polyalgorytmów, czyli silnika, który dobiera odpowiedni algorytm dla danej struktury danych i jej postaci. Taki polialgorytm dla `SparseMatrixCSC` potrafił wykonać się szybciej od zaimplementowaych algorytmów, nawet jeśli zżerał więcej pamięci. Testu dla zwykłej, nieefektywnej macierzy z zerami nawet nie wykonywałem, ze względu na oczywistość wyników. Co zaś się tyczy testów czasowych, były one niestabilne(nagłe skoki wartości), prawdopodobnie spowodowane architekturą systemu(cache itp..).