Obliczenia Naukowe

Lista 5

Jakub Gogola
236412

15 grudnia 2018

1 Zadanie I

1.1 Problem

W zadaniu 1., należało rozważyć problem rozwiązania układu równań liniowych następującej postaci:

$$\mathbf{Ax} = \mathbf{b}$$

dla danej macierzy współczynników $\mathbf{A} \in \mathbb{R}^{n \times n}$ i wektora prawych stron $\mathbf{b} \in \mathbb{R}^n$. Macierz $\mathbf{A}$ jest macierzą rzadką i blokową o następującej strukturze:

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

gdzie $v = \frac{n}{l}$, zakładając, że n jest podzielne przez l, gdzie l jest rozmiarem wszystkich kwadratowych macierzy wewnętrznych (bloków): $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$. Mianowicie, $\mathbf{A}_k \in \mathbb{R}^{l \times l}$, $k = 1, \dots, v$ jest macierzą gęstą, $\mathbf{0}$ jest kwadratową macierzą zerową stopnia l, macierz $\mathbf{B}_k \in \mathbb{R}^{l \times l}$, $k = 2, \dots, v$ jest następującej postaci:

$$\mathbf{B}_k = \begin{pmatrix} 0 & \dots & 0 & b_1^k \\ 0 & \dots & 0 & b_2^k \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & b_l^k \end{pmatrix},$$

$\mathbf{B}_k$ ma tylko jedną, ostatnią, kolumnę niezerową. Natomiast macierz $\mathbf{C}_k \in \mathbb{R}^{l \times l}$, $k = 1, \dots, v-1$ jest macierzą diagonalną:

$$\mathbf{C}_k = \begin{pmatrix} c_1^k & 0 & 0 & \dots & 0 \\ 0 & c_2^k & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & c_{l-1}^k & 0 \\ 0 & \dots & 0 & 0 & c_l^k \end{pmatrix}.$$

1.2 Klasyczne algorytmy

Do rozwiązywania układu równań liniowych logicznym wydaje się zastosowanie eliminacji Gaussa oraz rozkładu LU zdanej macierzy. Wyspecyfikowany powyżej układ równań daje się jak najbardziej rozwiązać stosując wspomniane metody, ale wówczas, ze względu na to, że macierz **A** jest macierzą rzadką, to złożoność takiej operacji jest rzędu $\mathcal{O}(n^3)$. Postawionym na liście zadaniem było dostosowanie klasycznych algorytmów pozwalających sobie radzić z takimi problemami do specyficznej struktury zadanej macierzy, aby zredukować złożoność takiego algorytmu do rzędu $\mathcal{O}(n)$.

1.2.1 Eliminacja Gaussa

Algorytm eliminacji Gaussa polega na sprowadzeniu wejściowego układu równań do macierzy schodkowej górnej wykorzystując do tego celu jedynie operacje elementarne (dodawanie, odejmowanie i mnożenie przez czynnik $l \neq 0$) na wierszach i kolumnach. Ponadto można dowolnie przestawiać wiersze i kolumny macierzy. W efekcie otrzymujemy macierz, przy pomocy której, mając dany wektor prawych stron, możemy znaleźć wektor **x** zawierający rozwiązania układu równań.

W celu otrzymania wynikowej macierzy schodkowej dokonuje się w omawianym algorytmie eliminacji niezerowych elementów pod diagonalą macierzy. Stosuje się w tym celu tak zwane *mnożniki* (będziemy je oznaczać za pomocą symbolu l_{ij}). W pierwszym kroku eliminowana jest niewiadoma x_1 z $n-1$ równań odejmując dla $i = 2, \ldots, n$ odpowiednią krotności pierwszego równania od $i-$tego równania, aby wyzerować w nim współczynnik x_1. Zauważmy, że jest to równoważne wyznaczeniu x_1 z pierwszego równania i podstawienia do pozostałych równań w układzie. Zauważmy, że w celu wyzerowania elementu a_{ik} należy od $i-$tego wiersza odjąć $k-$ty wiersz pomnożony przez współczynnik (*mnożnik*) $l_{ik} \leftarrow \frac{a_{ik}}{a_{kk}}$. Należy w tym momencie zauważyć, że ta wersja algorytmu nie zadziała w przypadku, gdy na diagonali macierzy, w $k-$tym kroku, wystąpi wartość 0. W tej wersji algorytmu jedynym sposobem na poradzenie sobie z taką sytuacją jest zamiana ze sobą miejscami kolumn lub wierszy. Zauważmy, że przy dokonywaniu eliminacji należy również dokonać zmian w wektorze prawych stron **b**. W celu obliczenia wektora **x**, po zakończeniu procesu eliminacji Gaussa, należy wykorzystać algorytm podstawienia wstecz.

Złożoność obliczeniowa przedstawionego powyżej algorytmu wynosi $\mathcal{O}(n^3)$.

1.2.2 Eliminacja Gaussa z częściowym wyborem elementu głównego

Jest to pewna modyfikacja algorytmu Gaussa, dzięki której można poradzić sobie z ewentualnymi 0 występującymi na diagonali macierzy. Można zauważyć, że warunek $a_{kk} \neq 0$ na ogół nie zapewnia numerycznej stabilności algorytmu. W celu zapobiegania takim sytuacjom stosuje się klasyczny algorytm Gaussa rozszerzony o *wybór elementu głównego w kolumnie*. Polega on na znalezieniu w kolumnie największego (z dokładnością do wartości bezwzględnej) elementu i odpowiednim przestawieniu wierszy macierzy tak, aby wybrany element znalazł się w określonym miejscu na diagonali. Możemy wyrazić to następującym wzorem:

$$|a_{kk}| = |a_{s(k),k}| = max\{|a_{ik}| : i = k, \ldots, n\}$$

gdzie $s(k)$ jest wektorem permutacji, w którym pamiętana jest kolejność przestawień.

1.2.3 Rozkład LU

Jest to kolejna metoda rozwiązywania układów równań liniowych ściśle powiązana z metodą eliminacji Gaussa. W rozkładzie **LU** wyznacza się dwie macierze trójkątne (schodkowe): dolnotrójkątną (**L**) i górnotrójkątną (**U**).

Dla zadanego układu równań $\mathbf{Ax} = \mathbf{b}$ można zapisać macierz **A** jako iloczyn pewnych dwóch macierzy trójkątnych: dolnej **L** (ang. *lower*) i górnej **U** (ang. *upper*), gdzie:

$$\mathbf{L} = \begin{bmatrix} l_{11} & 0 & \ldots & 0 \\ l_{21} & l_{22} & \ldots 0 & \\ \vdots & \vdots & \ddots 0 & \\ l_{n1} & l_{n2} & \ldots & l_{nn} \end{bmatrix}$$

$$\mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix}$$

Wówczas początkowy układ równań przyjmuje następującą postać: $\mathbf{LUx} = \mathbf{b}$. Jego rozwiązanie w istocie sprowadza się do rozwiązania następujących dwóch układów równań z każdą z utworzonych dwóch macierzy trójkątnych:

$$\mathbf{Ux} = \mathbf{b}^{'}$$
$$\mathbf{Lb}^{'} = \mathbf{b}$$

Złożoność obliczeniowa tego algorytmu jest nie większa niż $\mathcal{O}(n^2)$.

W celu zapewnienia jednoznaczności rozkładu, na diagonali jednej z powstałych w wyniku rozkładu macierzy, znajdują się wartości 1.

1.3 Dostosowanie algorytmów do postaci macierzy rzadkiej A

Zgodnie ze specyfikacją problemu, przedstawioną na początku sprawozdania, wiadomo, że macierz $\mathbf{A}$ jest rozmiaru $n \times n$ i składa się z podmacierzy o rozmiarach $l \times l$, gdzie spełniony jest warunek, że $l \mid n$. Każda z podmacierzy, co opisano wcześniej, jest w dosyć specyficznej postaci. Oto przedstawienie fragmentu macierzy $\mathbf{A}$, gdzie zakładamy, że n jest dowolne, ale l jest równe 4 (zatem spełniona jest zależności, że $n = 4k$, $k \in \mathbb{N}$). Poniżej przedstawiono kilka początkowych wierszy i kolumn macierzy $\mathbf{A}$:

$$\begin{pmatrix} a_{11}^1 & a_{12}^1 & a_{13}^1 & a_{14}^1 & c_{11}^1 & 0 & 0 & 0 & \dots \\ a_{21}^1 & a_{22}^1 & a_{23}^1 & a_{24}^1 & 0 & c_{22}^1 & 0 & 0 & \dots \\ a_{31}^1 & a_{32}^1 & a_{33}^1 & a_{34}^1 & 0 & 0 & c_{33}^1 & 0 & \dots \\ a_{41}^1 & a_{42}^1 & a_{43}^1 & a_{44}^1 & 0 & 0 & 0 & c_{44}^1 & \dots \\ 0 & 0 & 0 & b_1^2 & a_{11}^2 & a_{12}^2 & a_{13}^2 & a_{14}^2 & \dots \\ 0 & 0 & 0 & b_2^2 & a_{21}^2 & a_{22}^2 & a_{23}^2 & a_{24}^2 & \dots \\ 0 & 0 & 0 & b_3^2 & a_{31}^2 & a_{32}^2 & a_{33}^2 & a_{34}^2 & \dots \\ 0 & 0 & 0 & b_4^2 & a_{41}^2 & a_{42}^2 & a_{43}^2 & a_{34}^2 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

1.3.1 Algorytm eliminacji Gaussa

Możemy zauważyć, że przy takiej konstrukcji macierzy $\mathbf{A}$ w kolejnych kolumnach, w celu otrzymania macierzy górnotrójkątnej, musimy wyeliminować kolejno następującą liczbę składników: $3, 2, 1, 4, 3, 2, 1, 4, \dots$. Widzimy, że dla każdej podmacierzy $\mathbf{A}_k$ eliminujemy odpowiednio $l-1, l-2, l-3$ współczynników a następnie, całą ostatnią kolumnę podmacierzy $\mathbf{B}_k$, czyli l współczynników. Otrzymujemy zatem podany wcześniej ciąg. Zauważamy zależność, że każdy wyraz ciągu możemy zapisać za pomocą wzoru $l - k \mod l$, gdzie k jest numerem kolejnej kolumny w macierzy $\mathbf{A}$. Możemy zatem przedstawić pseudokod algorytmu Gaussa dostosowanego do struktury macierzy $\mathbf{A}$ z zadania.

Opis parametrów:
`A` - macierz rzadka $\mathbf{A}$,
`b` - wektor prawych stron
`n` - rozmiar macierzy $\mathbf{A}$
`l` - rozmiar podmacierzy

Dane wynikowe:
Wektor rozwiązań układu $\mathbf{x}$.

Możemy zauważyć, że pętla (1) wykona się n razy, a pętle (2) i (3) co najwyżej l razy, zatem złożoność czasowa przedstawionego algorytmu wynosi $\mathcal{O}(n \cdot l^2)$. Jest to zatem złożoność liniowa.

1.3.2 Algorytm eliminacji Gaussa z częściowym wyborem elementu głównego

Algorytm eliminacji Gaussa z częściowym wyborem elementu głównego jest bardzo podobny do przedstawionego powyżej algorytmu z tą małą różnicą, że dochodzi tutaj jeszcze wektor permutacji, w którym pamiętane są przestawianie elementów w kolumnach.

```
1: function GAUSS(A, b, n, l)
2:     for k ← 1 to n do                                          ▷ Pętla (1)
3:         for i ← k + 1 to k + l − (k mod l) do                  ▷ Pętla (2)
4:             multiplier ← A[i,k] / A[k,k]
5:             A[i, k] ← 0
6:             for j ← k + 1 to MIN(k + l, n) do                  ▷ Pętla (3)
7:                 A[i, j] ← A[i, j] − A[k, j]
8:             end for
9:             b[i] ← b[i] − multiplier · b[k]
10:        end for
11:    end for
12:    x[1 : n] ← {0, . . . , 0}
13:    for i ← n downto 1 do
14:        ∑ ← 0
15:        for j ← i + 1 to MIN(n, i + l) do
16:            sum ← sum + A[i, j] ∗ x[j]
17:        end for
18:        x[i] ← (b[i] − sum) / A[i,i]
19:    end for
20:    return x
21: end function
```

```
1: function GAUSS-WITH-CHOOSE(A, b, n, l)
2:     perm[1 : n] ← {1, . . . , n}
3:     for k ← 1 to n do                                          ▷ Pętla (1)
4:         maxrow ← k
5:         maxelement ← |A[k, k]|
6:         for i ← k + 1 to k + l − (k mod l) do                  ▷ Pętla(2)
7:             if |A| > maxelement then
8:                 maxelement ← |A[k, perm[i]]|
9:                 maxrow = i
10:            end if
11:        end for
12:        SWAP(perm[k], perm[maxrow])
13:        for i ← k + 1 to k + l − (k mod l) do                  ▷ Pętla (3)
14:            multiplier ← A[perm[i],k] / A[perm[k],k]
15:            A[perm[i], k] ← 0
16:            for j ← k + 1 to MIN(k + 2 ∗ l, n) do              ▷ Pętla (4)
17:                A[perm[i], j] ← A[perm[i], j] − A[perm[k], j]
18:            end for
19:            b[perm[i]] ← b[perm[i]] − multiplier · b[perm[k]]
20:        end for
21:    end for
22:    x[1 : n] ← {0, . . . , 0}
23:    for i ← n downto 1 do
24:        ∑ ← 0
25:        for j ← i + 1 to MIN(n, i + l) do
26:            sum ← sum + A[perm[i], j] ∗ x[j]
27:        end for
28:        x[i] ← (b[perm[i]] − sum) / A[perm[i],i]
29:    end for
30:    return x
31: end function
```

Opis parametrów:
`A` - macierz rzadka **A**,
`b` - wektor prawych stron
`n` - rozmiar macierzy **A**
`l` - rozmiar podmacierzy

Dane wynikowe:
Wektor rozwiązań układu **x**.

W przypadku eliminacji Gaussa z częściowym wyborem elementu głównego złożoność algorytmu będzie podobna jak wcześniej, ale z racji na obecność dodatkowej pętli (pętla (2)), w której wybierany jest element główny, oraz faktu, że pętla (4) wykona się $2l$ razy (ze względu na to, że ostatni, niezerowy element, można stworzyć w kolumnie o indeksie $2l$, gdy eliminujemy czynniki niezerowe z pierwszych $l-1$ kolumn), że wyniesie ona nie więcej niż $\mathcal{O}(n)$, aczkolwiek będzie nieco większa (z dokładnością do stałej), ponieważ wykonujemy nieco wiecej operacji.

1.3.3 Rozkład LU

Do rozkładu LU zostały zaadaptowane przedstawione wcześniej algorytmy z tą różnicą, że w miejsce eliminowanego elementu nie zostaje podstawiona wartość 0, ale mnożnik użyty do wyeliminowania tego elementu (dla funkcji `Gauss` linia 5, dla funkcji `Gauss-with-choose` linia 15). Nie są też wyliczane wektory **x** oraz aktualizowany wektor **b**, ponieważ te czynności wykonuje się dopiero przy rozwiązywaniu układu równań przy użyciu rozkładu **LU**. Złożoność tych algorytmów jest taka sama, jak obu przedstawionych wariantów algorytmów eliminacji Gaussa.

Algorytm rozwiązywania takiego układu wygląda następująco (przedstawione zostały dwa wariant - `solve-lu`, gdzie do wygenerowania rozkładu został użyty zwykły algorytm eliminacji Gaussa, oraz `solve-lu-choose`, gdzie do generowania rozkładu został użyty algorytm Gaussa z częściowym wyborem).

```
1:  function SOLVE-LU(A, b, n, l)
2:      b'[1 : n] ← {0, ..., 0}
3:      for i ← i to n do
4:          sum ← 0
5:          for j ← MAX(1, l · (i−1)/l) to i − 1 do
6:              sum ← sum + A[i, j]b'[j]
7:          end for
8:          b'[i] ← b[i] − sum
9:      end for
10:     x[1 : n] ← {0, ..., 0}
11:     for i ← n downto 1 do
12:         ∑ ← 0
13:         for j ← i + 1 to MIN(n, i + l) do
14:             sum ← sum + A[i, j] ∗ x[j]
15:         end for
16:         x[i] ← (b'[i] − sum)/A[i, i]
17:     end for
18:     return x
19: end function
```

Parametry wejściowe i wyjściowe są identyczne jak przy algorytmach rozwiązujących układ metodą eliminacji Gaussa, ale z tą różnicą, że do funkcji `solve-lu-choose` został dodatkowo podany wektor permutacji *perm*, aby można było odtworzyć kolejność wierszy po wykonaniu rozkładu **LU** metodą eliminacji Gaussa z wyborem elementu głównego. Złożoność obu algorytmów wynosi $\mathcal{O}(n)$.

1.4 Sposób pamiętania macierzy rzadkich w języku `Julia`

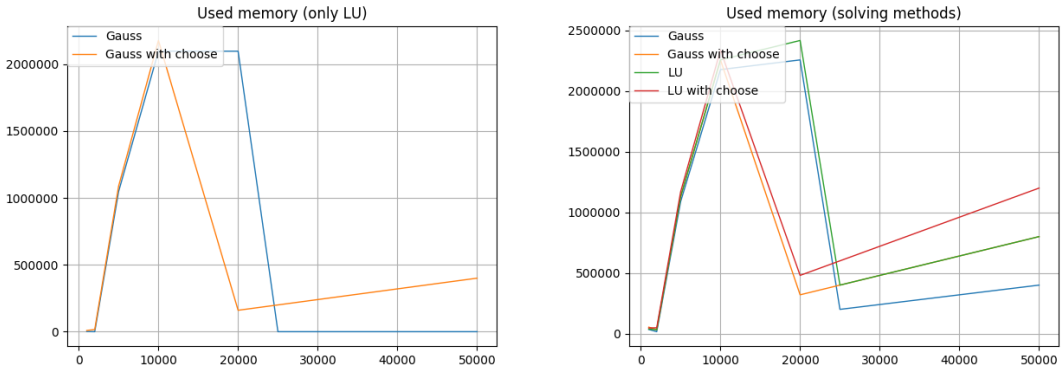
Język Julia udostępnia specjalną strukturę do pamiętania macierzy rzadkich nazwaną `SparseArrays`. Pamięta ona jedynie niezerowe elementy zadanej macierzy rzadkiej, przez co oszczędza pamięć. Sposób implementacji tej struktury pozwala na szybszy dostęp do elementów macierzy poprzez odwoływanie się do nich przez kolumny niż przez wiersze. Dla celów badania złożoności czasowej naszych algorytmów przyjęliśmy, że dostęp do elementu takiej macierzy, reprezentowanej za pomo-

```
1: function SOLVE-LU(A, perm, b, n, l)
2:     b'[1 : n] ← {0, . . . , 0}
3:     for i ← i to n do
4:         sum ← 0
5:         for j ← MAX(1, l · (i−1)/l) to i − 1 do
6:             sum ← sum + A[perm[i], j]b'[j]
7:         end for
8:         b'[i] ← b[perm[i]] − sum
9:     end for
10:    x[1 : n] ← {0, . . . , 0}
11:    for i ← n downto 1 do
12:        ∑ ← 0
13:        for j ← i + 1 to MIN(n, i + l) do
14:            sum ← sum + A[perm[i], j] ∗ x[j]
15:        end for
16:        x[i] ← (b'[i] − sum) / A[perm[i], i]
17:    end for
18:    return x
19: end function
```

cą `SparseArrays` jest stały ($\mathcal{O}(1)$), chociaż w rzeczywistości tak nie jest i realny czas wykonania funkcji w języku `Julia` jest kwadratowy co zostanie przedstawione na odpowiednim wykresie.

1.5 Wyniki

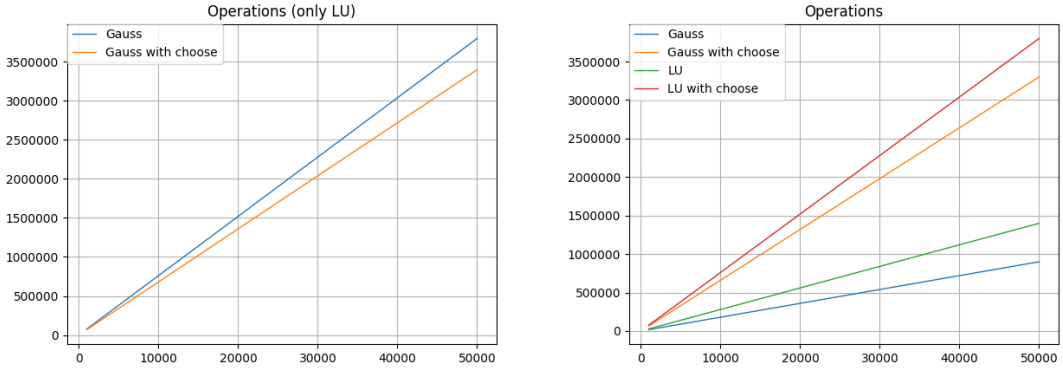
Zostały przeprowadzone testy dla zaimplementowanych algorytmów, dla których zbadano złożoność pamięciową oraz złożoność czasową (tutaj sprawdzono dwa warianty - zakładając, że czas dostępu do pojedynczego elementu macierzy jest stały (w tym przypadku za złożoność czasową przyjmuje się liczbę wykonywanych iteracji, ponieważ pozostałe operacje wykonywane są w czasie $\mathcal{O}(1)$) oraz mierząc rzeczywisty czas wykonania danej funkcji za pomocą makra `@timed` zwracającego między innymi czas działania programu oraz zużytą przez niego pamięć). Wyniki zostały zaprezentowane na poniższych wykresach.



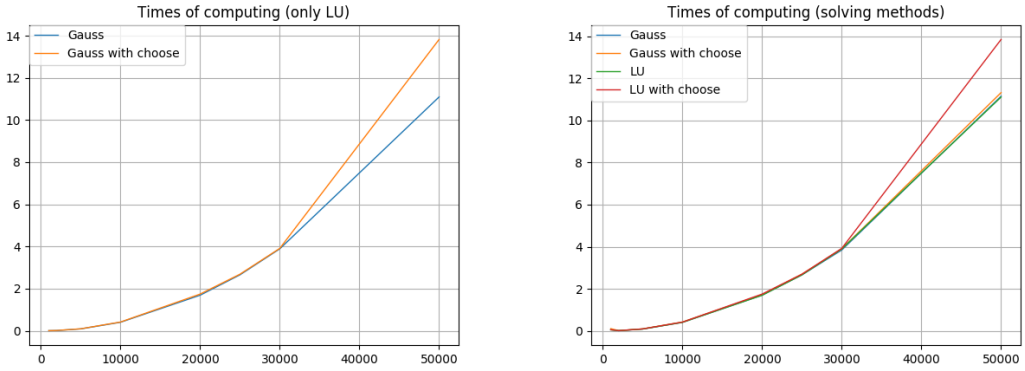
Rysunek 1: Wykresy zużycia pamięci: (a) tylko dla rozkładów LU, (b) dla wszystkich metod rozwiązujących układy równań

1.6 Obserwacje

Można zauważyć, że wraz ze wzrostem rozmiaru macierzy algorytmy są bardziej efektywne (zużywają mniej zasobów) pod względem alokacji pamięci potrzebnej do wykonania operacji (rysunek 1.). Widać również, że liczba iteracji dla wszystkich metod jest zależnością liniową od rozmiaru macierzy **A** (rysunek 2.). Rzeczywisty czas wykorzystany przez programy jest funkcją kwadratową (rysunek 3.).



Rysunek 2: Wykresy liczby operacji: (a) tylko dla rozkładów LU, (b) dla wszystkich metod rozwiązujących układy równań



Rysunek 3: Wykresy rzeczywistego czasu: (a) tylko dla rozkładów LU, (b) dla wszystkich metod rozwiązujących układy równań

1.7 Wnioski

Widzimy, że na rzeczywisty czas wykonania wpłynęło odwoływanie się do elementów umieszczonych w strukturze `SparseArrays`, przez co na wykresie widać kwadratowy czas wykonania algorytmów. Biorąc jednak pod uwagę założenie, że dostęp do elementów macierzy jest stały otrzymujemy, że rzeczywiście wprowadzone modyfikacje sprawiły, że klasyczne algorytmy rozwiązywania równań liniowych dobrze sprawdzają się dla zadanej macierzy rzadkiej **A** i, że liczba operacji z każdego z modyfikowanych algorytmów wynosi $\mathcal{O}(n)$ (n może być przemnożone przez jakieś stałe, ale złożoność nadal pozostaje liniowa). Widać również, że rozwiązywanie równań za pomocą rozkładu **LU** jest efektywniejsze niż wykonywanie tego za pomocą jedynie eliminacji Gaussa. Widać również, że algorytmy nie korzystające z wyboru elementu głównego mają o wiele mniejszą złożoność obliczeniową.

Wnioskiem płynącym z całego projektu powinien być fakt, że czasami niewielka modyfikacja klasycznego algorytmu (w tym wypadku ograniczenie "przeszukiwania" macierzy tylko do określonych indeksów) potrafi prowadzić do rozwiązania skomplikowanego problemu w dosyć łatwy sposób i małym nakładem pracy.