Obliczenia naukowe

Lista 5

Aleksandra Czarniecka (272385)

styczeń 2025

Wstęp: Przedstawienie problemu

Problemem jest rozwiązanie układu równań liniowych

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

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

Macierz $\mathbf{A}$ jest macierzą rzadką (tj. mającą dużo elementów zerowych) blokową o następującej strukturze:

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

$v = n/\ell$, zakładając że n jest podzielne przez ℓ, gdzie $\ell \geq 2$ jest rozmiarem wszystkich kwadratowych macierzy wewnętrznych (bloków): $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$.

Macierze są następującej postaci:

- $\mathbf{A}_k \in \mathbb{R}^{\ell\times\ell}$, $k = 1, \ldots, v$ – macierze gęste,
- $\mathbf{0}$ – kwadratowe macierze zerowe stopnia ℓ,
- $\mathbf{B}_k \in \mathbb{R}^{\ell\times\ell}$, $k = 2, \ldots, v$ – macierze mające tylko dwie ostatnie kolumny niezerowe

$$\mathbf{B}_k = \begin{pmatrix} 0 & \cdots & 0 & b^k_{1\,\ell-1} & b^k_{1\,\ell} \\ 0 & \cdots & 0 & b^k_{2\,\ell-1} & b^k_{2\,\ell} \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & b^k_{\ell\,\ell-1} & b^k_{\ell\,\ell} \end{pmatrix},$$

- $\mathbf{C}_k \in \mathbb{R}^{\ell\times\ell}$, $k = 1, \ldots, v-1$ – macierze diagonalne

$$\mathbf{C}_k = \begin{pmatrix} c^k_1 & 0 & 0 & \cdots & 0 \\ 0 & c^k_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c^k_{\ell-1} & 0 \\ 0 & \cdots & 0 & 0 & c^k_\ell \end{pmatrix}.$$

Testowane macierze mają duży rozmiar, zatem wyklucza to pamiętanie macierzy $\mathbf{A}$ jako tablicę o wymiarach $n \times n$ oraz użycie standardowych (bibliotecznych) algorytmów dla macierzy gęstych (tj. takich, gdzie nie zakłada się dużej liczby elementów zerowych). Zatem potrzebna jest specjalna struktura pamiętająca efektywnie macierz $\mathbf{A}$, która pamięta tylko elementy niezerowe. W tym celu została użyta biblioteka `SparseArrays`. Zakładamy, że dostęp do elementu macierzy jest w czasie stałym – $O(1)$. Jednak w rzeczywistości tak nie jest.

Standardowe algorytmy zostały zaadaptowane tak, aby uwzględniały one specyficzną postać macierzy $\mathbf{A}$, tj. jej rzadkość, regularność występowania elementów zerowych i niezerowych. Wszystkie algorytmy zostały napisane w języku Julia.

W celu przyspieszenia pracy algorytmów zostały stworzone funkcje pomocnicze, które są kluczowe do pracy z macierzą o strukturze blokowej, gdzie każdy blok ma wymiary $\ell \times \ell$. Ich celem jest efektywne wyznaczanie indeksów granicznych wierszy i kolumn należących do poszczególnych bloków. Wyjaśnienie działania poszczególnych funkcji przedstawiono poniżej.

Funkcja `last_row(k, l, n)`

Funkcja ta wyznacza ostatni wiersz dla bloku zawierającego wiersz k. Jest to niezbędne do określenia zakresu wierszy dla danego bloku. Zwraca wartość:

$$\text{last\_row}(k, \ell, n) = \min\left(\ell + \ell \cdot \left\lfloor \frac{k+1}{\ell} \right\rfloor, n\right),$$

gdzie:

- ℓ – rozmiar bloku (liczba wierszy w każdym bloku),
- n – całkowita liczba wierszy w macierzy.

Działanie funkcji polega na przesunięciu indeksu k w obrębie jego bloku, a następnie uwzględnieniu ograniczenia wynikającego z całkowitego rozmiaru macierzy n.

Funkcja `last_column(k, l, n)`

Funkcja ta wyznacza ostatnią kolumnę dla bloku zawierającego kolumnę k. Zwraca wartość:

$$\text{last\_column}(k, \ell, n) = \min(k + \ell, n),$$

gdzie:

- k – indeks kolumny,
- ℓ – rozmiar bloku (liczba kolumn w każdym bloku),
- n – całkowita liczba kolumn w macierzy.

Funkcja zapewnia, że indeks ostatniej kolumny nie wyjdzie poza całkowity rozmiar macierzy.

Funkcja `first_column(i, l)`

Funkcja ta wyznacza pierwszą kolumnę bloku, do którego należy kolumna i. Zwraca wartość:

$$\text{first\_column}(i, \ell) = \max\left(\ell \cdot \left\lfloor \frac{i-1}{\ell} \right\rfloor - 1, 1\right),$$

gdzie:

- i – indeks kolumny,
- ℓ – rozmiar bloku.

Funkcja oblicza początek bloku, w którym znajduje się kolumna i, i gwarantuje, że indeks nie będzie mniejszy niż 1.

1 Metoda eliminacji Gaussa

Metoda eliminacji Gaussa służy do rozwiązywania układów równań liniowych w postaci:

$$A\mathbf{x} = \mathbf{b},$$

gdzie $A \in \mathbb{R}^{n\times n}$ jest macierzą współczynników, $\mathbf{x} \in \mathbb{R}^n$ jest wektorem niewiadomych, a $\mathbf{b} \in \mathbb{R}^n$ jest wektorem prawych stron.

Metoda opiera się na przekształceniu macierzy A do postaci trójkątnej górnej U:

$$U\mathbf{x} = \mathbf{b}',$$

gdzie $\mathbf{b}'$ to przekształcony wektor prawych stron. Przekształcenie to osiąga się za pomocą operacji elementarnych na wierszach, eliminując elementy poniżej głównej przekątnej za pomocą operacji elementarnych na wierszach, co pozwala rozwiązać układ za pomocą podstawienia wstecznego.

1.1 Wariant bez wyboru elementu głównego

1.1.1 Idea matematyczna

Proces eliminacji w macierzy polega na zerowaniu elementów poniżej przekątnej. Dla każdej kolumny k, zerujemy elementy poniżej a_{kk} poprzez odjęcie odpowiednich wielokrotności wiersza k od kolejnych wierszy. Mnożnik m_{ik} obliczamy jako:

$$m_{ik} = \frac{A[i,k]}{A[k,k]}.$$

I w kroku k, dla $i > k$, elementy macierzy są modyfikowane:

$$A[i,j] \leftarrow A[i,j] - m_{ik}A[k,j],\ \forall j \geq k,$$

a wektor prawych stron:

$$b[i] \leftarrow b[i] - m_{ik}b[k].$$

Skutkiem tych działań, po $i-tym$ kroku wszystkie wartości pod przekątną w $i-tej$ kolumnie przyjmą wartość 0. Zatem po $n-1$ krokach układ sprowadza się do łatwiejszego w rozwiązaniu układu z macierzą górnotrójkątną.

Po zakończeniu eliminacji macierz A jest przekształcona w macierz trójkątną górną U, którą rozwiązujemy przez podstawienie wstecz (zaczynając od ostatniego wiersza):

$$x_n = \frac{b_n}{u_{nn}}, \quad x_i = \frac{b'_i - \sum_{j=i+1}^{n} u_{ij}x_j}{u_{ii}},\ i = n-1, \ldots, 1.$$

Należy wspomnieć o problemach jakie mogą się pojawić. Gdy na diagonali pojawią się wartości bliskie zeru, prowadzi to do dzielenia przez zero i błędów numerycznych algorytmu. Rozwiązaniem tego problemu jest wariant algorytmu z częściowym wyborem elementu głównego, opisany następnie.

1.1.2 Pseudokod

Algorithm 1 Eliminacja Gaussa bez wyboru elementu głównego

Input: macierz rzadka $\mathbf{A}$ w formacie SparseMatrixCSC, wektor $\mathbf{b}$, rozmiar bloku ℓ
Output: wektor $\mathbf{x}$, rozwiązanie układu $\mathbf{Ax} = \mathbf{b}$

$n \leftarrow$ rozmiar macierzy $\mathbf{A}$
$\mathbf{x} \leftarrow$ wektor zer o rozmiarze n
for $k \leftarrow 1$ ***to*** $n-1$ **do**
  $lr \leftarrow \text{last\_row}(k, \ell, n)$
  $lc \leftarrow \text{last\_column}(k, \ell, n)$
  for $i \leftarrow k+1$ ***to*** lr **do**
    if $|A[k,k]| < 1 \times 10^{-12}$ **then**
      error: Element na przekątnej $\mathbf{A}$ jest zbyt bliski zeru
    end
    $m \leftarrow \frac{A[i,k]}{A[k,k]}$ $A[i,k] \leftarrow 0$
    for $j \leftarrow k+1$ ***to*** lc **do**
      $A[i,j] \leftarrow A[i,j] - m \cdot A[k,j]$
    end
    $b[i] \leftarrow b[i] - m \cdot b[k]$
  end
end
for $i \leftarrow n$ ***down to*** 1 **do**
  $lc \leftarrow \text{last\_column}(i, \ell, n)$ $sum \leftarrow 0$
  for $j \leftarrow i+1$ ***to*** lc **do**
    $sum \leftarrow sum + A[i,j] \cdot x[j]$
  end
  $x[i] \leftarrow \frac{b[i]-sum}{A[i,i]}$
end
return $\mathbf{x}$

1.1.3 Analiza złożoności

- Złożoność obliczeniowa

  Standardowa eliminacja Gaussa, stosowana do rozwiązywania układu równań liniowych, ma złożoność obliczeniową rzędu $O(n^3)$.

  W pierwszym etapie algorytmu, eliminacji Gaussa, celem jest przekształcenie macierzy do formy górnotrójkątnej. W tym celu musimy wyzerować około $\frac{1}{2}n^2$ elementów macierzy. Proces ten polega na wykonaniu operacji takich jak mnożenie wierszy przez współczynniki i ich odejmowanie, co jest procesem liniowym w odniesieniu do liczby elementów w danym wierszu.

  Dla każdego wiersza, proces wyzerowania elementów poniżej przekątnej wymaga wykonania operacji na $n-k-1$ wierszach, gdzie k oznacza numer aktualnie przetwarzanego wiersza. Powtarzamy ten proces dla każdego z $n-1$ wierszy, co skutkuje złożonością $O(n^3)$. Dla każdego wiersza, wykonujemy operacje wyzerowania dla elementów poniżej przekątnej, a każda operacja ma złożoność liniową.

  W drugim etapie, po uzyskaniu macierzy górnotrójkątnej, rozwiązujemy układ równań przez podstawianie. Dla każdego wiersza i musimy wyznaczyć zmienną x_i, co wiąże się z $O(n)$ operacjami, ponieważ w przypadku macierzy o wymiarach $n \times n$ każda zmienna zależy od pozostałych n zmiennych. Zatem, rozwiązanie układu równań za pomocą eliminacji Gaussa ma złożoność $O(n^2)$ w tym etapie.

  Zatem, całkowita złożoność standardowej eliminacji Gaussa wynosi:

  $$O(n^3) \text{ (pierwszy etap)} + O(n^2) \text{ (drugi etap)} = O(n^3).$$

  W przypadku macierzy rzadkich, gdzie liczba niezerowych elementów w każdej kolumnie wynosi co najwyżej

ℓ, czyli ℓ jest stałą, złożoność obliczeniowa zmienia się na korzyść algorytmu. W takich macierzach, liczba elementów poniżej przekątnej, które musimy wyzerować, jest proporcjonalna do $\ell \cdot (n-1)$, co oznacza, że liczba operacji jest znacznie mniejsza niż w przypadku macierzy pełnych. Zatem, proces eliminacji Gaussa w przypadku macierzy rzadkich ma złożoność $O(\ell \cdot n)$, ponieważ odejmowanie wierszy odbywa się tylko dla ℓ elementów w każdym wierszu.

Dodatkowo, każda operacja odejmowania ma złożoność $O(\ell^2)$, ponieważ wymaga obliczeń dla ℓ niezerowych elementów w wierszu.

Po przekształceniu macierzy do formy górnotrójkątnej, proces podstawiania i rozwiązywania układu równań nadal wymaga $O(n)$ operacji, ponieważ ℓ jest stałą. Zatem rozwiązanie układu równań po eliminacji w przypadku macierzy rzadkich ma złożoność $O(n)$.

Zatem, całkowita złożoność obliczeniowa dla macierzy rzadkich wynosi:

$$O(\ell \cdot n) \text{ (pierwszy etap)} + O(n) \text{ (drugi etap)} = O(n) \quad \text{(zakładając, że } \ell \text{ jest stałą).}$$

- Złożoność pamięciowa:

  Wymagania pamięciowe wynoszą **O(n)**, gdyż macierz **A** jest przechowywana z użyciem `Sparse Arrays`, co z założeń daje dostęp w czasie stałym do elementu macierzy, a wektor b zajmuje także $O(n)$ pamięci.

1.2 Wariant z częściowym wyborem elementu głównego

1.2.1 Idea matematyczna

Wariant ten rozwiązuje problem wartości bliskich zeru na przekątnej macierzy A, przez częściowy wybór elementu głównego, czyli elementu macierzy znajdującego się na diagonali, którego w $k-tym$ kroku używamy do wyzerowania pozostałych w $k-tej$ kolumnie. Jest to element przez, który dzielimy chcąc uzyskać mnożnik ($A[k,k]$).

Zatem w każdym kroku wybierany jest element o największej wartości bezwzględnej w bieżącej kolumnie jako element główny:

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

Trzeba także zamienić wiersze tak aby ten element stał się elementem głównym, jednak zwykła zamiana jest dosyć kosztowna. Zatem może to być zastąpione przez wektor permutacji wierszy p. Zapamiętujemy w nim, na którym miejscu aktualnie znajduje się dany wiersz. Zatem jedyna różnica w samym algorytmie będzie taka, że zamiast odwoływać się do danego wiersza bezpośrednio, będziemy się odwoływać do jego pozycji zapisanej w wektorze permutacji. Zwróćmy uwagę na to, że zmieniając wiersze w macierzy A musimy zmieniać też poszczególne współrzędne w wektorze prawych stron – b.

Reszta algorytmu, czyli eliminacja Gaussa i podstawienie wsteczne, jest analogiczna jak w wersji bez wyboru elementu głównego.

1.2.2 Pseudokod

Algorithm 2 Eliminacja Gaussa z częściowym wyborem elementu głównego

Input: macierz rzadka **A** w formacie SparseMatrixCSC, wektor **b**, rozmiar bloku ℓ
Output: wektor **x**, rozwiązanie układu $\mathbf{Ax} = \mathbf{b}$

$n \leftarrow$ rozmiar macierzy **A**
$p \leftarrow [1, 2, \ldots, n]$ `// Tablica permutacji wierszy`
x $\leftarrow$ wektor zer o rozmiarze n
for $k \leftarrow 1$ ***to*** $n - 1$ **do**
  $lr \leftarrow \text{last\_row}(k, \ell, n)$
  $lc \leftarrow \text{last\_column}(k, 2\ell, n)$
  $max\_index \leftarrow k$
  $max\_element \leftarrow 0.0$
  for $i \leftarrow k$ ***to*** lr **do**
    $current \leftarrow |A[p[i], k]|$
    if $current > max\_element$ **then**
      $max\_element \leftarrow current$
      $max\_index \leftarrow i$
    end
  end
  Zamień $p[k]$ z $p[max\_index]$ `// Aktualizacja permutacji wierszy`
  for $i \leftarrow k + 1$ ***to*** lr **do**
    if $|A[p[k], k]| < 1 \times 10^{-12}$ **then**
      error: Element na przekątnej **A** jest zbyt bliski zeru
    end
    $m \leftarrow \frac{A[p[i],k]}{A[p[k],k]}$
    $A[p[i], k] \leftarrow 0$
    for $j \leftarrow k + 1$ ***to*** lc **do**
      $A[p[i], j] \leftarrow A[p[i], j] - m \cdot A[p[k], j]$
    end
    $b[p[i]] \leftarrow b[p[i]] - m \cdot b[p[k]]$
  end
end
for $i \leftarrow n$ ***down to*** 1 **do**
  $lc \leftarrow \text{last\_column}(p[i], 2\ell, n)$
  $sum \leftarrow 0$
  for $j \leftarrow i + 1$ ***to*** lc **do**
    $sum \leftarrow sum + A[p[i], j] \cdot x[j]$
  end
  $x[i] \leftarrow \frac{b[p[i]] - sum}{A[p[i],i]}$
end
return x

1.2.3 Analiza złożoności

- Złożoność obliczeniowa Sytuacja wygląda tutaj analogicznie do poprzedniego wariantu. Tablica permutacji nie zwiększa złożoności obliczeniowej, ponieważ dostęp do indeksów jest w czasie stałym.

  Zatem tutaj również, całkowita złożoność obliczeniowa dla macierzy rzadkich wynosi:

  $$O(\ell \cdot n) \text{ (pierwszy etap)} + O(n) \text{ (drugi etap)} = O(n) \quad \text{(zakładając, że } \ell \text{ jest stałą).}$$

- Złożoność pamięciowa

  Wymagania pamięciowe wynoszą także **O(n)**, gdyż dodatkowa tablica permutacji **P**, zajmuje $O(n)$ pamięci.

2 Rozkład LU

Rozkład LU polega na rozkładzie macierzy $A \in \mathbb{R}^{n \times n}$ na iloczyn dwóch macierzy:

$$\mathbf{A} = \mathbf{LU},$$

gdzie **L** (lower) jest macierzą dolnotrójkątną, w której zapisywane są mnożniki, a **U** (upper) jest macierzą górnotrójkątną, jak w metodzie eliminacji Gaussa. Jedna z macierzy, w tym przypadku macierz **L** ma na diagonali same jedynki, co zapewnia jednoznaczność rozkładu. Następnie układ równań $A\mathbf{x} = \mathbf{b}$ można rozwiązać w dwóch krokach:

1. Rozwiązanie $L\mathbf{y} = \mathbf{b}$ za pomocą podstawienia w przód.
2. Rozwiązanie $U\mathbf{x} = \mathbf{y}$ za pomocą podstawienia wstecz.

Mając raz wyznaczony rozkład **LU** macierzy **A** można łatwo i mniejszym kosztem rozwiązywać układ równań dla różnych wektorów prawych stron.

2.1 Wariant bez wyboru elementu głównego

2.1.1 Idea matematyczna

Na podstawie wcześniejszych rozważań, można zauważyć, że metoda eliminacji Gaussa tak właściwie tworzy macierz **U**, a po dodaniu zapisu mnożników, zamiast zerowania, to również macierz **L**.

Macierze **L** i **U** są przechowywane w jednym obiekcie **A**, co oszczędza pamięć. Po zakończeniu algorytmu macierz **A** przyjmuje postać:

$$A = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ l_{21} & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & u_{nn} \end{bmatrix},$$

gdzie elementy poniżej przekątnej reprezentują L, a powyżej U.

2.1.2 Pseudokod

Algorithm 3 Rozkład LU macierzy A bez wyboru elementu głównego

Input: macierz rzadka **A** w formacie SparseMatrixCSC, rozmiar bloku ℓ
Output: macierz **A** po dekompozycji LU

```
n ← rozmiar pierwszego wymiaru A
for k ← 1 to n − 1 do
    lr ← last_row(k, ℓ, n)
    lc ← last_column(k, ℓ, n)
    for i ← k + 1 to lr do
        if |A[k, k]| < 1 × 10^−12 then
            error: Element na przekątnej A jest zbyt bliski zeru
        end
        m ← A[i,k] / A[k,k]
        A[i, k] ← m
        for j ← k + 1 to lc do
            A[i, j] ← A[i, j] − m · A[k, j]
        end
    end
end
return A
```

2.1.3 Analiza złożoności

- Złożoność obliczeniowa

Rozkład **LU** to praktycznie pierwszy etap algorytmu Gaussa, tylko zamiast zerować elementy pod przekątną, zapamiętywane są tam mnożniki. Niemniej jednak, nie zmienia to całkowitej złożoności.

Przy takich samych optymalizacjach otrzymujemy:

$$O(\ell \cdot n) \text{ (pierwszy etap)} + O(n) \text{ (drugi etap)} = O(n) \quad \text{(zakładając, że } \ell \text{ jest stałą).}$$

- Złożoność pamięciowa

  W tej metodzie wszystko dzieje się na macierzy **A**, którą utrzymujemy za pomocą `Sparse Arrays`, zatem z początkowych założeń mamy złożoność **O**(**n**).

2.2 Wariant z częściowym wyborem elementu głównego

2.2.1 Idea matematyczna

Wariant ten jest analogiczny do odpowiadającego mu wariantu z metody eliminacji Gaussa. On także rozwiązuje problem wartości bliskich zeru na przekątnej macierzy A, przez częściowy wybór elementu głównego.

Zatem jako element główny wybieramy:

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

Do zamiany wierszy w tym przypadku również używamy tablicy permutacji.

Reszta algorytmu jest analogiczna jak w wersji bez wyboru elementu głównego.

2.2.2 Pseudokod

Algorithm 4 Rozkład LU macierzy A z częściowym wyborem elementu głównego

Input: macierz rzadka **A** w formacie SparseMatrixCSC, rozmiar bloku ℓ
Output: macierz **A** po dekompozycji LU oraz tablica permutacji **p**

```
1  n ← rozmiar pierwszego wymiaru A
2  p ← [1, 2, ..., n] // Tablica permutacji wierszy
3  for k ← 1 to n − 1 do
4  |  lr ← last_row(k, ℓ, n)
5  |  lc ← last_column(k, ℓ, n)
6  |  max_index ← k
7  |  max_element ← 0.0
8  |  for i ← k to lr do
9  |  |  current ← |A[p[i], k]|
10 |  |  if current > max_element then
11 |  |  |  max_element ← current
12 |  |  |  max_index ← i
13 |  |  end
14 |  end
15 |  Zamień p[k] z p[max_index] // Aktualizacja permutacji wierszy
16 |  for i ← k + 1 to lr do
17 |  |  if |A[p[k], k]| < 1 × 10^−12 then
18 |  |  |  error: Element na przekątnej A jest zbyt bliski zeru
19 |  |  end
20 |  |  m ← A[p[i],k] / A[p[k],k]
21 |  |  A[p[i], k] ← m
22 |  |  for j ← k + 1 to lc do
23 |  |  |  A[p[i], j] ← A[p[i], j] − m · A[p[k], j]
24 |  |  end
25 |  end
26 end
27 return A, p
```

2.2.3 Analiza złożoności

- Złożoność czasowa

  Analogicznie złożoność wynosi $\mathbf{O(n)}$.

- Złożoność pamięciowa

  Analogicznie złożoność wynosi $\mathbf{O(n)}$.

3 Rozwiązanie układu równań $A\mathbf{x} = \mathbf{b}$ z wykorzystaniem rozkładu LU

3.1 Idea matematyczna

Mając rozkład **LU** macierzy **A** można rozbić układ równań $\mathbf{Ax} = \mathbf{b}$ na dwa podukłady z macierzami trójkątnymi:

- $\mathbf{Ly} = \mathbf{b}$, gdzie rozwiązanie to podstawienie w przód (analogicznie jak przekształcenia wykonywane przy tworzeniu macierzy **U** z macierzy **A**):

$$y_i = b_i - \sum_{j=1}^{i-1} l_{ij} y_j, \quad i = 1, \ldots, n.$$

- $\mathbf{Ux} = \mathbf{y}$, gdzie rozwiązanie to podstawienie wstecz (jak w eliminacji Gaussa):

$$x_i = \frac{y_i - \sum_{j=i+1}^{n} u_{ij} x_j}{u_{ii}}, \quad i = n, \ldots, 1.$$

3.2 Pseudokod

Algorithm 5 Rozwiązanie układu równań Ax = b z wykorzystaniem rozkładu LU bez wyboru elementu głównego

Input: macierz rzadka **A** w formacie SparseMatrixCSC, wektor **b**, rozmiar bloku ℓ
Output: Wektor **x**, rozwiązanie układu $\mathbf{Ax} = \mathbf{b}$

```
1  n ← rozmiar macierzy A
2  y ← wektor zer o rozmiarze n
3  x ← wektor zer o rozmiarze n
4  for i ← 1 to n do
5  |  sum ← 0.0
6  |  fc ← first_column(i, ℓ)
7  |  for j ← fc to i − 1 do
8  |  |  sum ← sum + A[i, j] · y[j]
9  |  end
10 |  y[i] ← b[i] − sum
11 end
12 for i ← n down to 1 do
13 |  sum ← 0.0
14 |  lc ← last_column(i, ℓ, n)
15 |  for j ← i + 1 to lc do
16 |  |  sum ← sum + A[i, j] · x[j]
17 |  end
18 |  x[i] ← (y[i] − sum) / A[i, i]
19 end
20 return x
```

Algorithm 6 Rozwiązanie układu równań Ax = b z wykorzystaniem rozkładu LU z częściowym wyborem elementu głównego

Input: macierz rzadka **A** w formacie SparseMatrixCSC, wektor **b**, tablica permutacji **p**, rozmiar bloku ℓ
Output: Wektor **x**, rozwiązanie układu $\mathbf{Ax} = \mathbf{b}$

```
1  n ← rozmiar macierzy A
2  x ← wektor zer o rozmiarze n
3  y ← wektor zer o rozmiarze n
4  for i ← 1 to n do
5  |  sum ← 0.0
6  |  fc ← first_column(p[i], ℓ)
7  |  for j ← fc to i − 1 do
8  |  |  sum ← sum + A[p[i], j] · y[j]
9  |  end
10 |  y[i] ← b[p[i]] − sum
11 end
12 for i ← n down to 1 do
13 |  sum ← 0.0
14 |  lc ← last_column(p[i], ℓ, n)
15 |  for j ← i + 1 to lc do
16 |  |  sum ← sum + A[p[i], j] · x[j]
17 |  end
18 |  x[i] ← (y[i] − sum) / A[p[i], i]
19 end
20 return x
```

3.3 Analiza złożoności

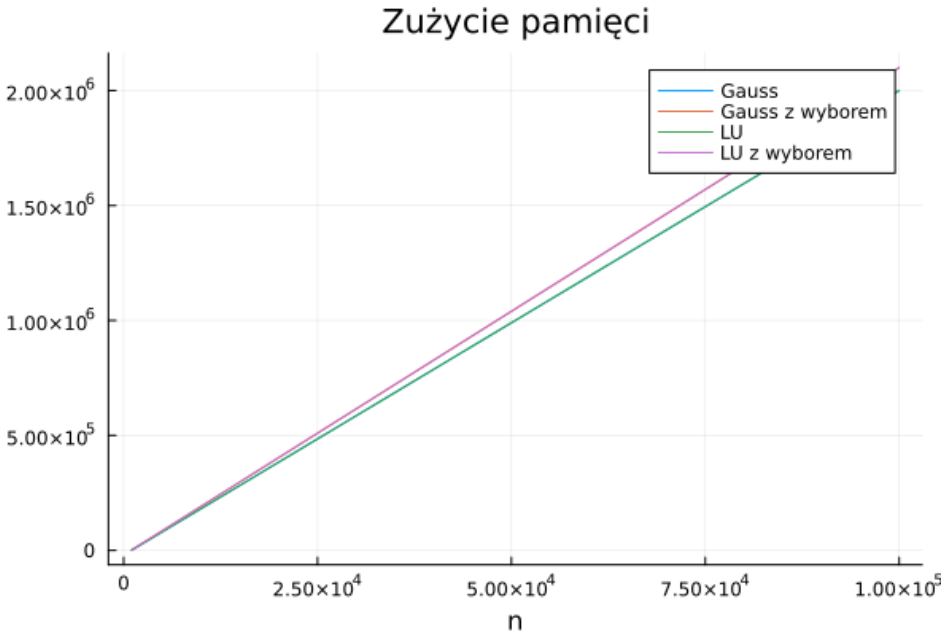
- Złożoność obliczeniowa

  Problem ten jest już niezależny od operacji na macierzy i sprowadza się do dwóch prostych równań, które przy opisanych wcześniej optymalizacjach będą miały złożoność $\mathbf{O(n)}$.

- Złożoność pamięciowa Wektor prawych stron oraz macierz zajmują $\mathbf{O(n)}$.

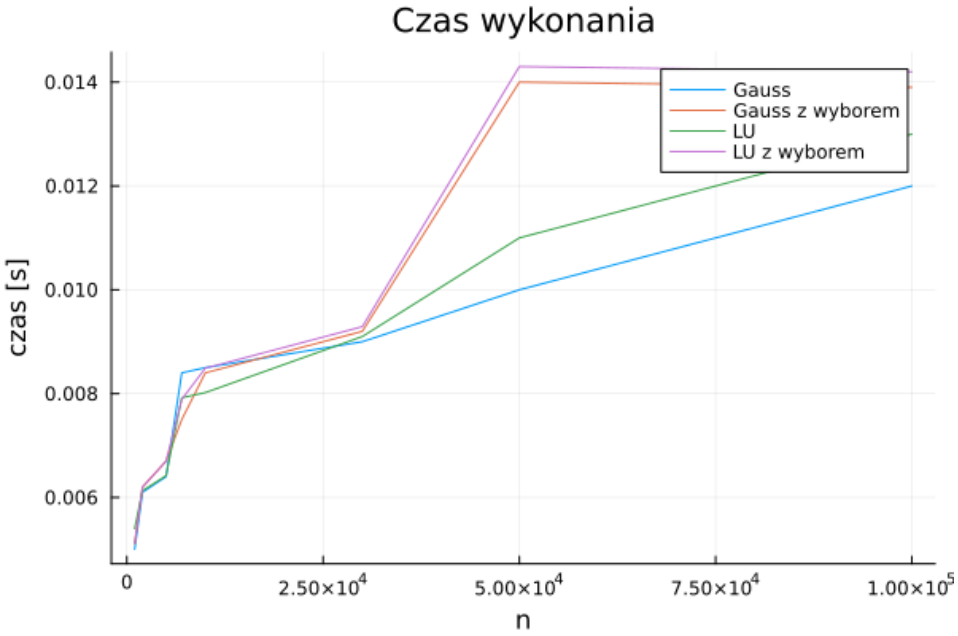
4 Wyniki i ich interpretacja

Wyniki zostały przedstawione dla macierzy, której wskaźnik uwarunkowania wynosił 10, a $\ell = 4$.



Rysunek 1: Wykres przedstawiający złożoność pamięciową implementowanych algorytmów.

Jak można było się spodziewać, warianty z częściowym wyborem elementu głównego mają większe zużycie, ze względu na przechowywanie wektora permutacji.



Rysunek 2: Wykres przedstawiający złożoność czasową implementowanych algorytmów.

Z wykresu widać, że algorytmy z częściowym wyborem elementu głównego są wolniejsze niż ich odpowiedniki bez wyboru. Obie metody – Gaussa i LU – wypadły podobnie. Zatem jeśli rozwiązywać więcej układów z jedną macierzą, warto zrobić rozkład LU, gdyż czas wykonania kolejnych wywołań będzie mniejszy.

Wszystkie algorytmy zostały również przetestowane pod kątem błędów względnych, gdy wektor prawych stron był obliczany. Eksperymenty przeprowadzono na macierzach o wskaźniku uwarunkowania równym 10.0 oraz z parametrem $\ell = 4$.

Rozmiar A	Metoda eliminacji Gaussa	Metoda eliminacji Gaussa z wyborem
3000	$5.294944023095919 \cdot 10^{-15}$	$4.583789514971993 \cdot 10^{-16}$
4000	$3.941097250330561 \cdot 10^{-15}$	$4.558576358655353 \cdot 10^{-16}$
7000	$5.572015100704354 \cdot 10^{-15}$	$4.298286603514981 \cdot 10^{-16}$
10000	$5.296361445179956 \cdot 10^{-15}$	$4.485705759438765 \cdot 10^{-16}$

Rozmiar A	Metoda LU	Metoda LU z wyborem
3000	$2.292369454680386 \cdot 10^{-14}$	$3.63977268190935 \cdot 10^{-16}$
4000	$1.9944709360505984 \cdot 10^{-14}$	$3.506990595628056 \cdot 10^{-16}$
7000	$2.1756038487442212 \cdot 10^{-14}$	$3.667424709454897 \cdot 10^{-16}$
10000	$1.7735465592478852 \cdot 10^{-14}$	$3.582567451682111 \cdot 10^{-16}$

Widać, że algorytmy wykorzystujące częściowy wybór elementu głównego osiągnęły dokładniejsze wyniki, z błędem względnym rzędu 10^{-16}. Dla porównania, algorytmy, które nie przeprowadzały wyboru elementu głównego, uzyskały błędy rzędu 10^{-15} w przypadku metody eliminacji Gaussa oraz 10^{-14} dla rozkładu LU.

Dodatkowo dla każdej z metod zostały przeprowadzone testy, dla przykładowych danych (umieszczonych przy liście zadań), gdzie sprawdzana jest poprawność algorytmów, dla różnych rozmiarów macierzy (od 16 do 500000). Każdy test weryfikuje, czy rozwiązanie układu równań $Ax = b$ daje wektor x składający się z jedynek.

5 Wnioski

Dostosowanie algorytmów do struktury danych pozwoliło obniżyć złożoność z $O(n^3)$ do liniowej, co stanowi znaczącą poprawę. Odpowiednia reprezentacja macierzy także znacznie zmniejszyła zużycie pamięci. Dlatego dokładna analiza problemu jest kluczowa, aby ominąć naiwne podejście. Dostosowanie reprezentacji danych do struktury macierzy

umożliwiło przechowywanie macierzy rzędu 10^6, co byłoby niemożliwe w standardowej reprezentacji.

Warte zauważenia jest, że metody bez wyboru elementu głównego są ograniczone – nie sprawdzają się, gdy na diagonali występuje element zerowy. Natomiast algorytmy z częściowym wyborem elementu głównego charakteryzowały się większą złożonością, lecz dawały dokładniejsze i pewniejsze wyniki. Za to rozkład LU jest najbardziej wydajny w przypadku układów, gdzie zmiany dotyczą wyłącznie wektora prawej strony.