Sprawozdanie Lista 5

Paweł Krzyszczak

styczeń 2025

Wstęp: Przedstawienie problemu

Problemem jest rozwiązanie układu równań liniowych

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

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

Macierz $\mathbf{A}$ jest macierzą rzadką (tzn. zawierającą dużą liczbę elementów zerowych) o strukturze blokowej, której układ przedstawia się następująco:

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

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

Macierze występujące w tym układzie mają następującą postać:

- $\mathbf{A}_k \in \mathbb{R}^{\ell\times\ell}$, dla $k = 1, \ldots, v$ – macierze gęste,
- $\mathbf{0}$ – macierze zerowe o wymiarze $\ell \times \ell$,
- $\mathbf{B}_k \in \mathbb{R}^{\ell\times\ell}$, dla $k = 2, \ldots, v$ – macierze, które mają tylko dwie ostatnie kolumny niezerowe:

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

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

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

W przypadku dużych rozmiarów macierzy, klasyczne przechowywanie macierzy $\mathbf{A}$ jako tablicy o wymiarach $n \times n$ oraz stosowanie standardowych algorytmów dla macierzy gęstych nie są efektywne. W związku z tym, konieczne jest zastosowanie specjalnej struktury danych, która przechowuje jedynie elementy niezerowe. Do tego celu została użyta biblioteka `SparseArrays`. Zakłada się, że dostęp do elementu macierzy następuje w czasie stałym – $O(1)$, jednak w praktyce może to nie zawsze mieć miejsce.

Algorytmy zostały zmodyfikowane w taki sposób, aby uwzględniały specyficzną strukturę macierzy $\mathbf{A}$, czyli jej rzadkość i regularność rozkładu elementów zerowych i niezerowych. Wszystkie algorytmy zostały zaimplementowane w języku Julia.

Aby przyspieszyć działanie algorytmów, opracowano funkcje pomocnicze, które są kluczowe w pracy z macierzą o strukturze blokowej, gdzie każdy blok ma wymiary $\ell \times \ell$. Funkcje te mają na celu efektywne wyznaczanie indeksów granicznych wierszy i kolumn należących do poszczególnych bloków. Działanie tych funkcji zostało przedstawione poniżej.

Funkcja `last_row(k, l, n)`

Funkcja ta wyznacza ostatni wiersz dla bloku zawierającego wiersz k. Jest to istotne do określenia zakresu wierszy w danym 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.

Funkcja ta pozwala na przesunięcie indeksu k w obrębie jego bloku, z uwzględnieniem 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 ta zapewnia, że indeks ostatniej kolumny nie przekroczy rozmiaru 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 ta oblicza początek bloku, w którym znajduje się kolumna i, i zapewnia, że indeks nie będzie mniejszy niż 1.

1 Metoda eliminacji Gaussa

Metoda eliminacji Gaussa jest jedną z technik 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 ta polega na przekształceniu macierzy A do postaci trójkątnej górnej U:

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

gdzie $\mathbf{b}'$ jest przekształconym wektorem prawych stron. Operację tę wykonuje się za pomocą operacji elementarnych na wierszach, eliminując elementy poniżej głównej przekątnej, co umożliwia rozwiązanie układu za pomocą podstawienia wstecznego.

1.1 Wariant bez wyboru elementu głównego

1.1.1 Idea matematyczna

Proces eliminacji 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:

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

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:

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

Po wykonaniu $i-tych$ operacji wszystkie wartości poniżej przekątnej w $i-tej$ kolumnie zostaną zerowe. Po $n-1$ krokach układ sprowadza się do układu z macierzą trójkątną górną, który można rozwiązać przez podstawienie wstecz.

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

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

Warto wspomnieć o problemach związanych z tym algorytmem. Gdy wartości na diagonali są bliskie zeru, może dojść do dzielenia przez zero i błędów numerycznych. Rozwiązaniem tego problemu jest wariant algorytmu z częściowym wyborem elementu głównego, który będzie opisany poniżej.

1.1.2 Pseudokod

Algorithm 1 Eliminacja Gaussa bez wyboru elementu głównego

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

$n \leftarrow$ rozmiar macierzy **A**
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 **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 x

1.1.3 Analiza złożoności

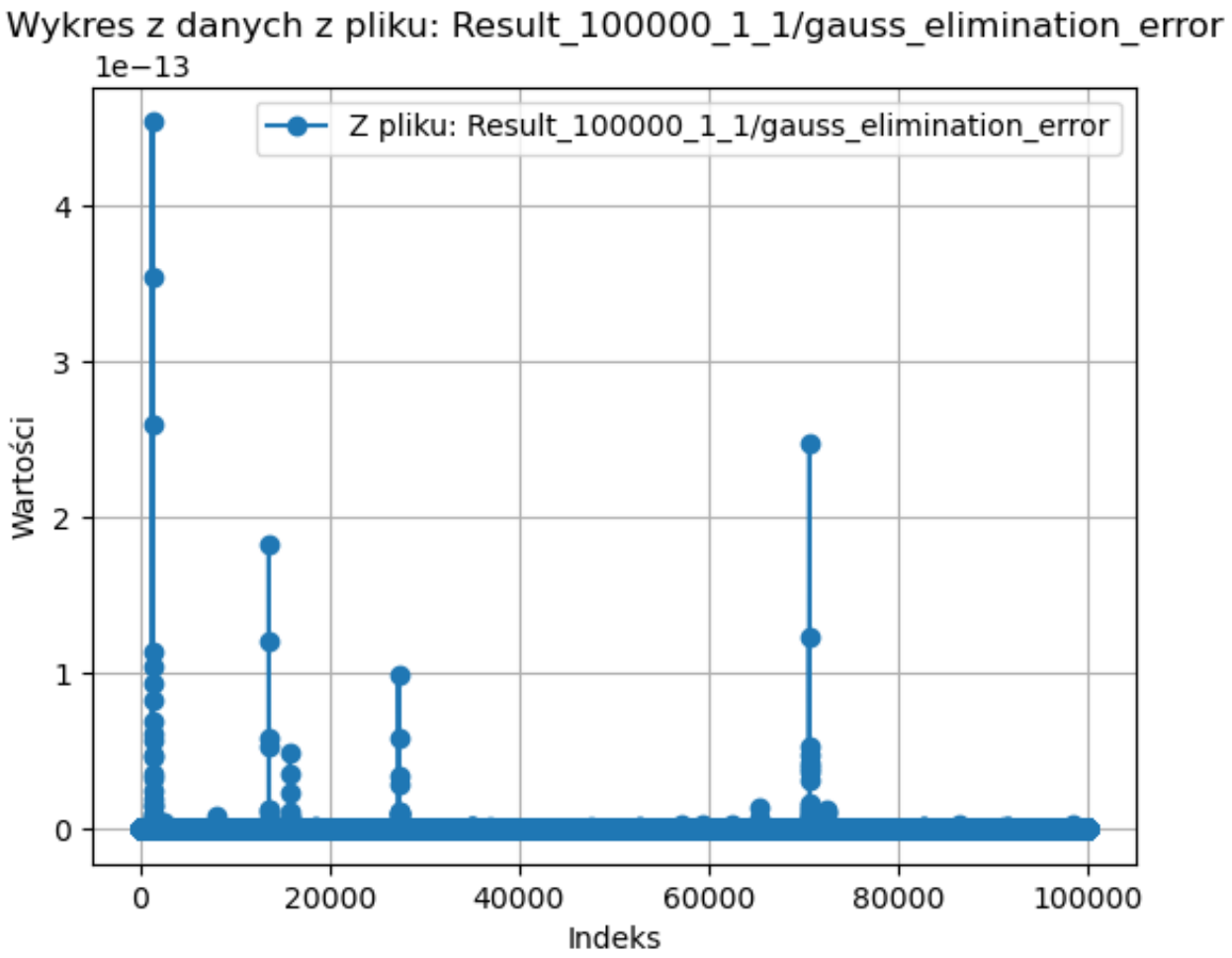
- Złożoność czasowa:

  Funkcja wykonuje eliminację Gaussa, która wymaga zagnieżdżonych pętli iterujących po wierszach i kolumnach macierzy, prowadząc do złożoności $O(n^3)$.

- Złożoność pamięciowa:

  Wymagania pamięciowe wynoszą $O(n)$. ponieważ macierz **A** jest przechowywana za pomocą `Sparse Arrays`, co zapewnia dostęp w czasie stałym do elementów macierzy. Wektor **b** również zajmuje pamięć w rozmiarze $O(n)$.

1.1.4 Wyniki eksperymentu



Rysunek 1: Wykres z wartościa błędem względnym dla danych n=500000

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 elementów w k-tej kolumnie. Jest to element, przez który dzielimy, chcąc uzyskać mnożnik ($A[k,k]$).

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

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

Następnie musimy zamienić wiersze, aby ten element stał się elementem głównym. Zwykła zamiana jest dosyć kosztowna, dlatego zamiast tego stosujemy wektor permutacji wierszy p. Przechowujemy w nim informacje o aktualnej pozycji danego wiersza. W związku z tym, zamiast odwoływać się do danego wiersza bezpośrednio, będziemy odwoływać się do jego pozycji zapisanej w wektorze permutacji. Zmieniając wiersze w macierzy **A**, musimy również zaktualizować współrzędne w wektorze prawych stron **b**.

Reszta algorytmu, czyli eliminacja Gaussa i podstawienie wsteczne, jest analogiczna do 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 $\mathbf{Ax} = \mathbf{b}$

```
1  n ← rozmiar macierzy A
2  p ← [1, 2, ..., n] // Tablica permutacji wierszy
3  x ← wektor zer o rozmiarze n
4  for k ← 1 to n − 1 do
5  |  lr ← last_row(k, ℓ, n)
6  |  lc ← last_column(k, 2ℓ, n)
7  |  max_index ← k
8  |  max_element ← 0.0
9  |  for i ← k to lr do
10 |  |  current ← |A[p[i], k]|
11 |  |  if current > max_element then
12 |  |  |  max_element ← current
13 |  |  |  max_index ← i
14 |  |  end
15 |  end
16 |  Zamień p[k] z p[max_index] // Aktualizacja permutacji wierszy
17 |  for i ← k + 1 to lr do
18 |  |  if |A[p[k], k]| < 1 × 10^−12 then
19 |  |  |  error: Element na przekątnej A jest zbyt bliski zeru
20 |  |  end
21 |  |  m ← A[p[i],k] / A[p[k],k]
22 |  |  A[p[i], k] ← 0
23 |  |  for j ← k + 1 to lc do
24 |  |  |  A[p[i], j] ← A[p[i], j] − m · A[p[k], j]
25 |  |  end
26 |  |  b[p[i]] ← b[p[i]] − m · b[p[k]]
27 |  end
28 end
29 for i ← n down to 1 do
30 |  lc ← last_column(p[i], 2ℓ, n)
31 |  sum ← 0
32 |  for j ← i + 1 to lc do
33 |  |  sum ← sum + A[p[i], j] · x[j]
34 |  end
35 |  x[i] ← (b[p[i]] − sum) / A[p[i],i]
36 end
37 return x
```

1.2.3 Analiza złożoności

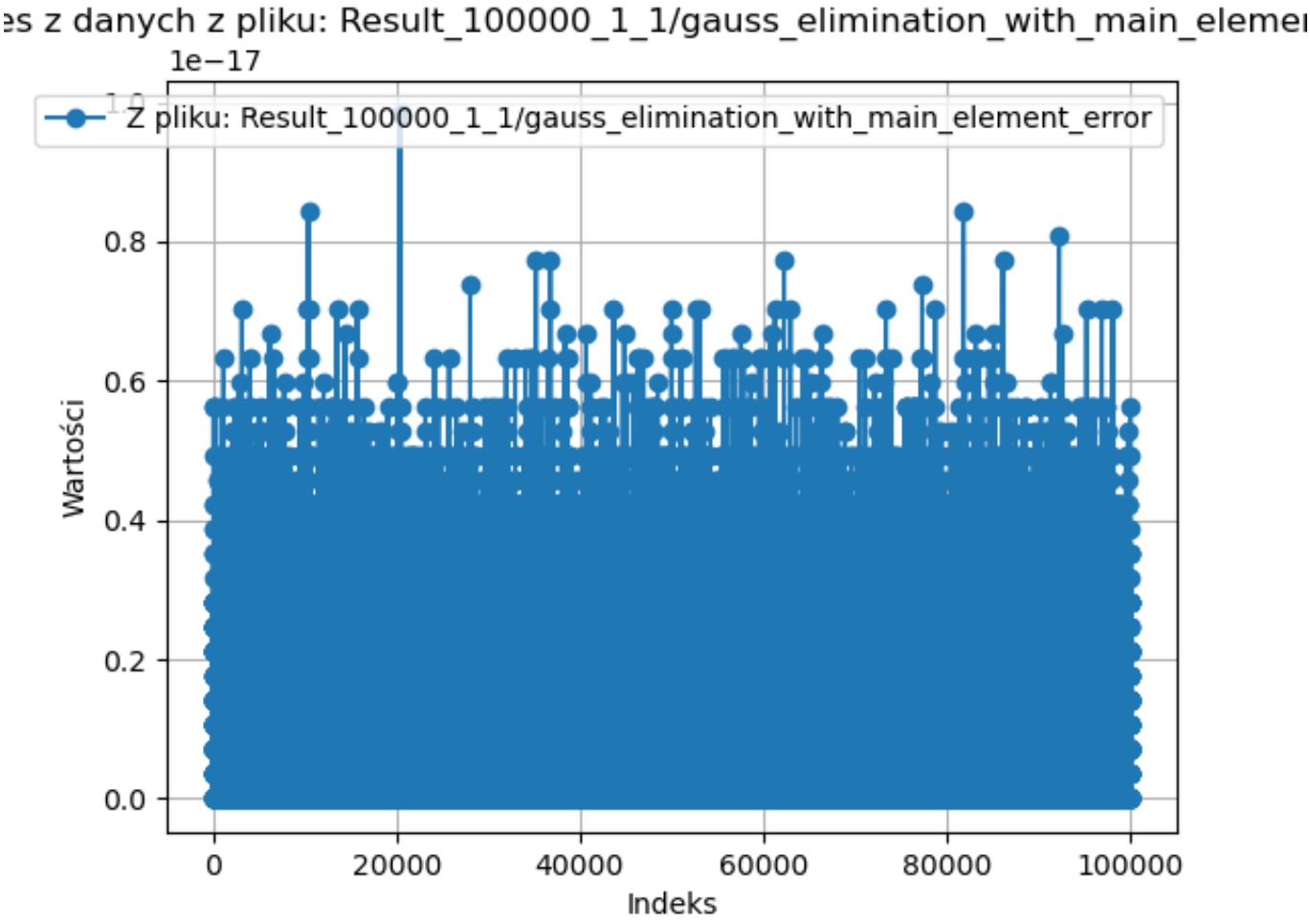
- Złożoność czasowa

  Dodanie wyboru elementu głównego (maksymalny element w kolumnie) zwiększa czas wykonywania, ale nadal jest to złożoność $O(n^3)$.

- Złożoność pamięciowa

  Wymagania pamięciowe wynoszą $O(n)$, ponieważ macierz **A** jest przechowywana z użyciem `Sparse Arrays`, co zapewnia dostęp w czasie stałym do elementów macierzy, a wektor **b**, podobnie jak tablica permutacji **p**, zajmuje także $O(n)$ pamięci.

1.2.4 Wyniki eksperymentu



Rysunek 2: Wykres z wartościa błędem względnym dla danych n=500000

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, tworzy 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 czasowa

  Wyznaczanie rozkładu LU (bez wyboru elementu głównego) również wymaga zagnieżdżonych pętli, co daje złożoność $O(n^3)$.

- Złożoność pamięciowa

  Wymagania pamięciowe wynoszą $O(n)$. ponieważ macierz **A** jest przechowywana za pomocą `Sparse Arrays`, co zapewnia dostęp w czasie stałym do elementów macierzy. Wektor **b** również zajmuje pamięć w rozmiarze $O(n)$.

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:

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

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**

$n \leftarrow$ rozmiar pierwszego wymiaru **A**
$p \leftarrow [1, 2, \ldots, n]$ `// Tablica permutacji wierszy`
for $k \leftarrow 1$ ***to*** $n - 1$ **do**
  $lr \leftarrow \text{last\_row}(k, \ell, n)$
  $lc \leftarrow \text{last\_column}(k, \ell, n)$
  $max\_index \leftarrow k$
  $max\_element \leftarrow 0.0$
  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
    $m \leftarrow \frac{A[p[i],k]}{A[p[k],k]}$
    $A[p[i], k] \leftarrow m$
    for $j \leftarrow k + 1$ ***to*** lc **do**
      $A[p[i], j] \leftarrow A[p[i], j] - m \cdot A[p[k], j]$
    end
  end
end
return A, p

2.2.3 Analiza złożoności

- Złożoność czasowa

  Rozkład LU z wyborem elementu głównego (w przypadku macierzy rzadkiej) ma podobną złożoność do klasycznego rozkładu LU, czyli $O(n^3)$.

- Złożoność pamięciowa

  Wymagania pamięciowe wynoszą $O(n)$. ponieważ macierz **A** jest przechowywana za pomocą `Sparse Arrays`, co zapewnia dostęp w czasie stałym do elementów macierzy. Wektor **b** również zajmuje pamięć w rozmiarze $O(n)$.

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 $\mathbf{U}$ z macierzy $\mathbf{A}$):

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

- $\mathbf{Ux} = \mathbf{y}$, 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 $\mathbf{Ax} = \mathbf{b}$ z wykorzystaniem rozkładu LU 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 ← rozmiar macierzy A
y ← wektor zer o rozmiarze n
x ← wektor zer o rozmiarze n
for i ← 1 to n do
    sum ← 0.0
    fc ← first_column(i, ℓ)
    for j ← fc to i − 1 do
        sum ← sum + A[i, j] · y[j]
    end
    y[i] ← b[i] − sum
end
for i ← n down to 1 do
    sum ← 0.0
    lc ← last_column(i, ℓ, n)
    for j ← i + 1 to lc do
        sum ← sum + A[i, j] · x[j]
    end
    x[i] ← (y[i] − sum) / A[i, i]
end
return x
```

Algorithm 6 Rozwiązanie układu równań $\mathbf{Ax} = \mathbf{b}$ z wykorzystaniem rozkładu LU z częściowym wyborem elementu głównego

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

$n \leftarrow$ rozmiar macierzy $\mathbf{A}$
$x \leftarrow$ wektor zer o rozmiarze n
$y \leftarrow$ wektor zer o rozmiarze n
for $i \leftarrow 1$ ***to*** n **do**
    $sum \leftarrow 0.0$
    $fc \leftarrow \text{first\_column}(p[i], \ell)$
    for $j \leftarrow fc$ ***to*** $i - 1$ **do**
        $sum \leftarrow sum + A[p[i], j] \cdot y[j]$
    end
    $y[i] \leftarrow b[p[i]] - sum$
end
for $i \leftarrow n$ ***down to*** 1 **do**
    $sum \leftarrow 0.0$
    $lc \leftarrow \text{last\_column}(p[i], \ell, n)$
    for $j \leftarrow i + 1$ ***to*** lc **do**
        $sum \leftarrow sum + A[p[i], j] \cdot x[j]$
    end
    $x[i] \leftarrow \frac{y[i] - sum}{A[p[i], i]}$
end
return x

3.3 Analiza złożoności

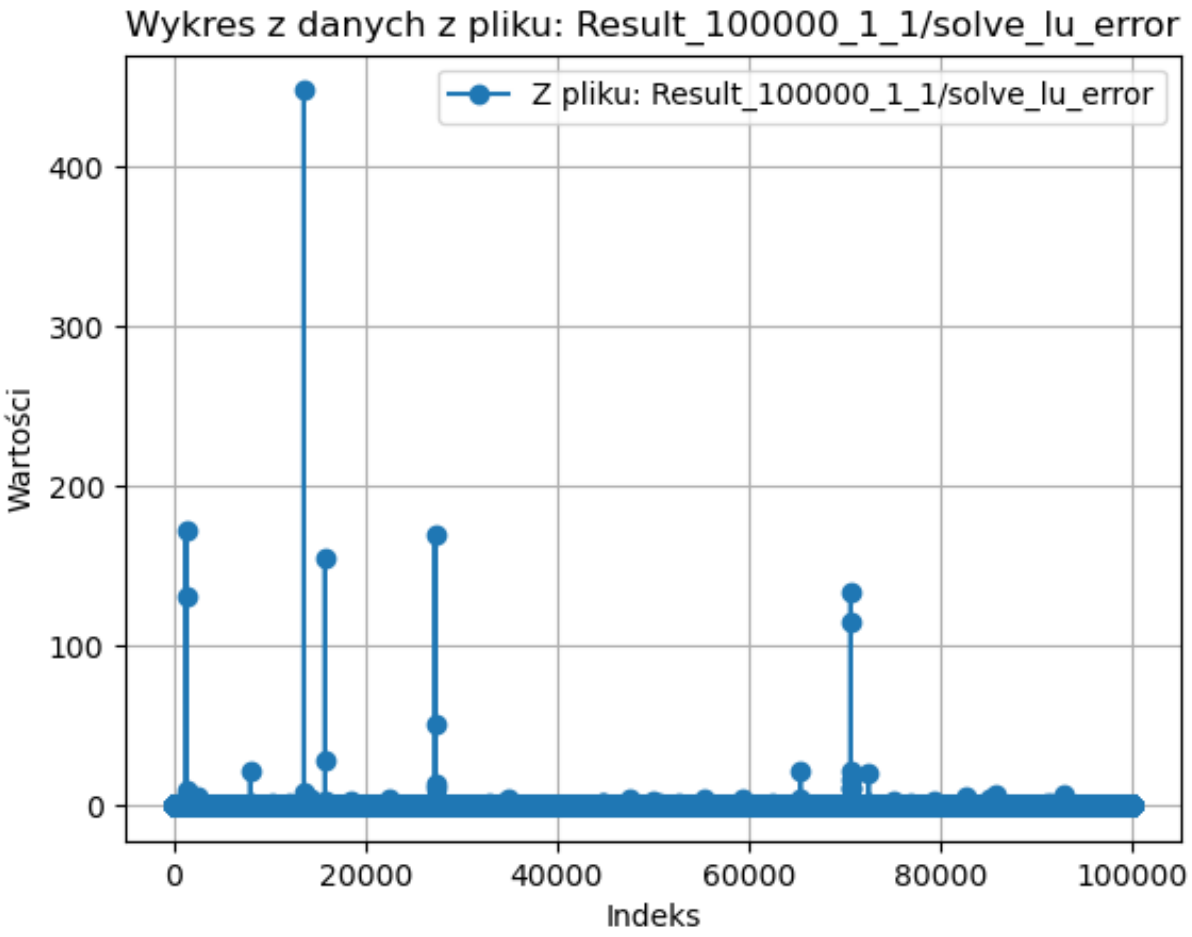
- Złożoność czasowa

  Rozwiązywanie układu równań z rozkładem LU zarówno z wyborem elementu głównego i nie ma złożoność $O(n^2)$.

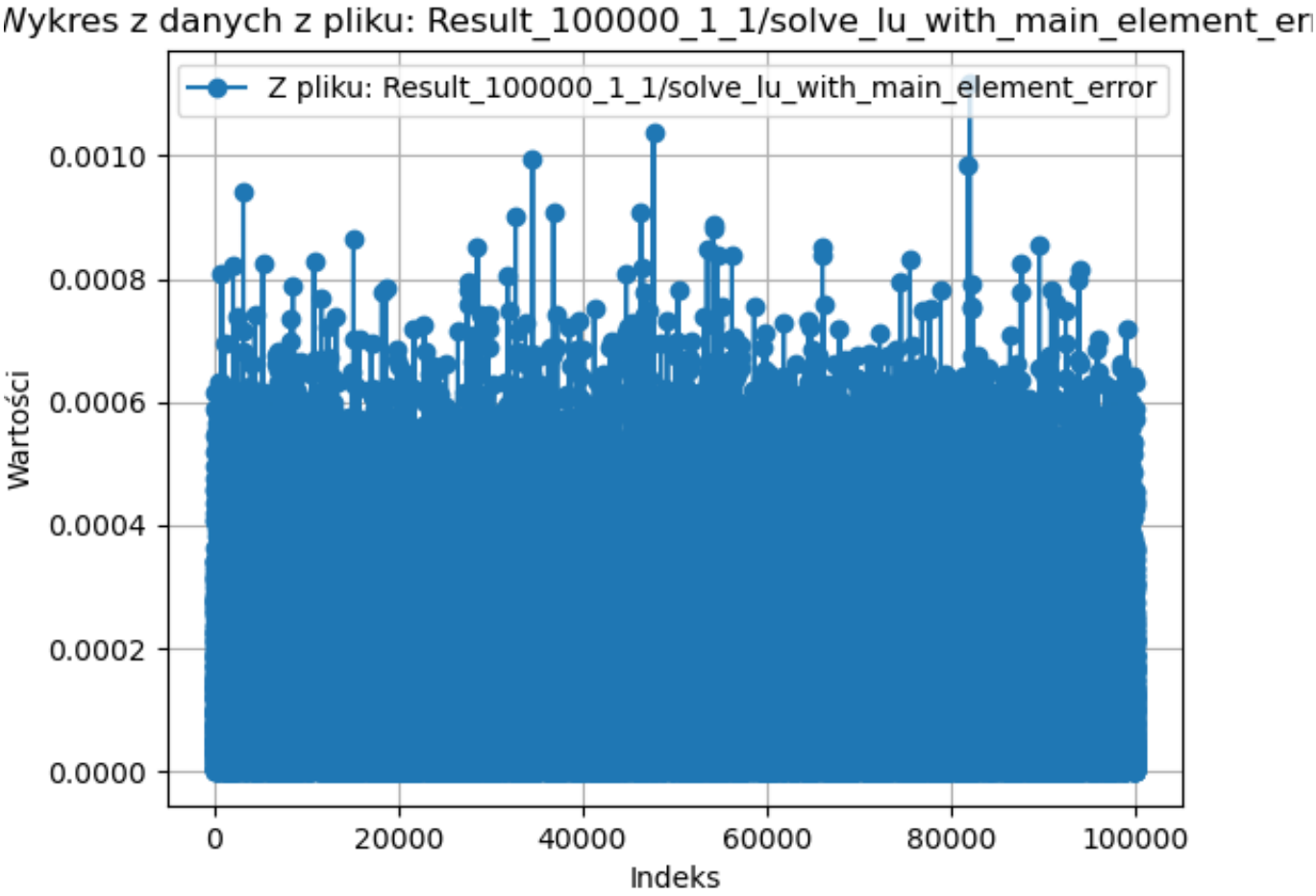
- Złożoność pamięciowa

  Wymagania pamięciowe wynoszą $O(n)$. ponieważ macierz $\mathbf{A}$ jest przechowywana za pomocą `Sparse Arrays`, co zapewnia dostęp w czasie stałym do elementów macierzy. Wektor $\mathbf{b}$ również zajmuje pamięć w rozmiarze $O(n)$.

3.4 Wyniki eksperymentu



Rysunek 3: Wykres z wartościa błędem względnym Ax=b rozkładu LU dla danych n=500000



Rysunek 4: Wykres z wartościa błędem względnym Ax=b rozkładu LU z wyborem elementu głównym dla danych n=500000

4 Wnioski i obserwacje

W przypadku rozwiązywania układu równań z macierzą blokową o określonej strukturze, korzystniejsze będzie zastosowanie jednej z metod wykorzystujących częściowy wybór elementu głównego. Choć te metody są nieco wolniejsze, osiągają znacznie mniejsze błędy względne, co ma dla nas istotne znaczenie.

Metoda Gaussa z częściowym wyborem elementu głównego zazwyczaj działa szybciej, dlatego może być odpowiednia, gdy zależy nam na czasie, nawet jeśli różnica w czasie obliczeń nie jest znaczna.

Jeśli natomiast musimy wielokrotnie rozwiązywać układ równań dla tej samej macierzy A, ale różnych wektorów b, bardziej opłacalnym rozwiązaniem będzie metoda z rozkładem LU z częściowym wyborem elementu głównego, ponieważ w tej sytuacji jest bardziej efektywna.