Obliczenia naukowe

Laboratorium

Lista 5

Kinga Majcher
272354

Styczeń 2025

1 Przedstawienie problemu

Dana jest specyficzna macierz blokowa $A \in \mathbb{R}^{n \times n}$ postaci:

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

gdzie $A_k \in \mathbb{R}^{l \times l}$, $k = 1, \ldots, v$ jest macierzą gęstą, 0 jest kwadratową macierzą zerową stopnia l, macierz $B_k \in \mathbb{R}^{l \times l}$, $k = 2, \ldots, v$ jest następującej postaci:

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

macierz $C_k \in \mathbb{R}^{l \times l}$, $k = 1, \ldots, v-1$ jest następującej postaci:

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

a $v = n/l$, zakładając, że n jest podzielne przez l.

Naszym zadaniem jest rozwiązywanie układu równań $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$, gdzie $\mathbf{b} \in \mathbb{R}^n$ metodą eliminacji Gaussa. Z powodu dużego rozmiaru macierzy i dużej liczby elementów zerowych znajdujących się w niej wykluczone jest pamiętanie jej jako standardowa tablica $n \times n$ oraz rozwiązywanie układu równań standardową metodą. Należy więc zaadaptować algorytm eliminacji Gaussa do specyfiki tej konkretnej macierzy

2 Metoda eliminacji Gaussa

Idea metody

Metoda eliminacji Gaussa jest jedną z metod rozwiązywania układów równań liniowych. Można ją podzielić na dwa etapy: przekształcenie macierzy do macierzy górnotrójkątnej oraz rozwiązanie przekształconego układu równań.

Przekształcenie macierzy do macierzy górnotrójkątnej

W celu przekształcenia macierzy do macierzy górnotrójkątnej będziemy przemnażać wiersze przez odpowiednie współczynniki, a następnie odejmować je od siebie.

Zapiszmy układ równań $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ w innej postaci:

$$\begin{aligned} a_{1,1}^{(1)}x_1 + a_{1,2}^{(1)}x_2 + \cdots + a_{1,n}^{(1)}x_n &= b_1^{(1)}, \\ a_{2,1}^{(1)}x_1 + a_{2,2}^{(1)}x_2 + \cdots + a_{2,n}^{(1)}x_n &= b_2^{(1)}, \\ \vdots \qquad\qquad & \vdots \qquad\quad \vdots \\ a_{n,1}^{(1)}x_1 + a_{n,2}^{(1)}x_2 + \cdots + a_{n,n}^{(1)}x_n &= b_n^{(1)}. \end{aligned}$$

W celu wyeliminowania x_1 z równań od 2-go do n-tego przemnażamy 1-sze równanie przez:

$$I_{i1} = \frac{a_{i,1}^{(1)}}{a_{1,1}^{(1)}},\ i = 2, \ldots, n$$

i odejmujemy od pozostałych. Wówczas otrzymujemy:

$$\begin{aligned} a_{1,1}^{(1)}x_1 + a_{1,2}^{(1)}x_2 + \cdots + a_{1,n}^{(1)}x_n &= b_1^{(1)}, \\ a_{2,2}^{(2)}x_2 + \cdots + a_{2,n}^{(2)}x_n &= b_2^{(2)}, \\ \vdots \qquad\quad & \vdots \qquad\quad \vdots \\ a_{n,2}^{(2)}x_2 + \cdots + a_{n,n}^{(2)}x_n &= b_n^{(2)}. \end{aligned}$$

W celu wyeliminowania każdej kolejnej zmiennej x_k z równań od k+1-szego do n-tego przemnażamy k-te równanie przez:

$$I_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}},\ i = k+1, \ldots, n.$$

Ostatecznie po $n-1$ krokach otrzymujemy:

$$\begin{aligned} a_{1,1}^{(1)}x_1 + a_{1,2}^{(1)}x_2 + \cdots + a_{1,n}^{(1)}x_n &= b_1^{(1)}, \\ a_{2,2}^{(2)}x_2 + \cdots + a_{2,n}^{(2)}x_n &= b_2^{(2)}, \\ \ddots \quad \vdots \qquad\quad & \vdots \\ a_{n,n}^{(n)}x_n &= b_n^{(n)}, \end{aligned}$$

czyli układ z macierzą górnotrójkątną równoważny z układem pierwotnym.

Ważną obserwacją jest to, że aby wariant ten działał poprawnie elementy znajdujące się na przekątnej macierzy nie mogą być zerowe, ponieważ w celu przekształcenia macierzy dzielimy przez te wartości. Najlepiej, gdyby elementy te były możliwie dalekie od zera, gdyż posługując się arytmetyką o ograniczonej precyzji dzielenie przez małe wartości również może dawać nieoczekiwane wyniki.

Rozwiązanie przekształconego układu równań

Rozwiązanie układu równań z macierzą górnotrójkątną zaczynamy od ostatniego równania. Wyznaczamy x_n z wzoru:

$$x_n = \frac{b_n}{a_{n,n}}.$$

Mając tę wartość możemy obliczyć każdą kolejną z wzoru

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} x_j}{a_{i,i}}.$$

W ten sposób otrzymujemy cały wektor rozwiązań x.

Analiza złożoności eliminacji Gaussa

Złożoność obliczeniowa

Standardowa wersja eliminacji Gaussa ma złożoność obliczeniową $O(n^3)$. Musimy wyzerować w najgorszym wypadku $O(\frac{1}{2}n^2)$ elementów macierzy, każdy poprzez przemnożenie przez wyznaczony współczynnik danego wiersza macierzy, a następnie odjęcie go od dalszych wierszy. Odejmowanie dwóch wierszy jest procesem liniowym, w każdym etapie musimy go dokonać dla $n-k-1$ wierszy, a cały ten proces powtórzyć dla wszystkich $n-k-1$ wierszy, stąd złożoność tego etapu standardowej eliminacji Gaussa wynosi $O(n^3)$. Etap drugi jest mniej złożony i można bezpośrednio zauważyć, że jego złożoność wynosi $O(n^2)$.

Złożoność $O(n^3)$ jest złożonością bardzo dużą, zwłaszcza dla macierzy o dużym n. Wykorzystanie więc standardowej eliminacji Gaussa dla naszej konkretnej macierzy $\mathbf{A}$ nie ma więc sensu, zwłaszcza, że nasza macierz jest macierzą rzadką, a więc wykonywalibyśmy bardzo dużo obliczeń w celu wyzerowania elementów, które już są zerowe. Ulepszymy więc tę metodę korzystając z wiedzy na temat struktury macierzy.

Możemy zauważyć, że w pierwszym etapie w każdej kolumnie mamy do wyzerowania co najwyżej tyle niezerowych elementów pod przekątną co rozmiar bloku l. Z elementami już będącymi zerami nie trzeba już nic robić, więc liczba elementów do wyzerowania została ograniczona do $l \cdot (n-1) = O(n)$, gdy l jest stałą.

Również w etapie odejmowania wierszy nie musimy odejmować od siebie elementów zerowych. Oznacza to dla nas, że złożoność tego etapu nie jest zależna od n tylko od l, ma więc on złożoność $O(l^2)$.

Cały pierwszy etap ma więc złożoność obliczeniową $O(n) \cdot O(l^2)$, czyli $O(n)$, gdy l jest stałą.

Po przekształceniu macierzy do macierzy górnotrójkątnej w jednym wierszu mamy co najwyżej $O(l)$ wartości niezerowych. Można więc przekształcić wzór do wyznaczania x_i do postaci:

$$x_i = \frac{b_i - \sum_{j=i+1}^{i+l} a_{i,j} \cdot x_j}{a_{i,i}}.$$

Wówczas przy założeniu, że l jest stałą wyznaczenie całego wektora rozwiązań ma złożoność $O(n)$.

Jako, że etapy te wykonują się po sobie i oba mają złożoność $O(n)$ to cała metoda eliminacji Gaussa dla macierzy o takiej konkretnej strukturze ma złożoność obliczeniową $O(n)$.

Złożoność pamięciowa

Do przechowywania macierzy $\mathbf{A}$ wykorzystywana jest specjalna struktura, która nie przechowuje wartości zerowych w macierzy. Wektor prawych stron ma natomiast złożoność $O(n)$.

2.1 Wariant podstawowy

W implementacji wykorzystujemy dodatkowo dwie funkcje:

- `getLastColumn(A, row)` - funkcja, która dla podanej macierzy o strukturze zgodnej z zadaniem i danego jej wiersza zwraca indeks kolumny, w której występuje ostatnia niezerowa wartość w danym rzędzie
- `getLastRow(A, column)` - funkcja, która dla podanej macierzy o strukturze zgodnej z zadaniem i danej jej kolumny zwraca indeks wiersza, w którym występuje ostatnia niezerowa wartość w danej kolumnie

Obie funkcje działają w czasie $O(1)$, więc ich użycie nie wpływa na złożoność algorytmów.

2.1.1 Pseudokod

Algorithm 1: Funkcja rozwiązująca układ równań metodą eliminacji Gaussa

Input: A – macierz współczynników (BlockMatrix), b - wektor prawych stron
Output: x – wektor rozwiązania

$n \leftarrow size(b)$;
$\overline{x} \leftarrow \overline{0}$;
for k ***from*** 1 ***to*** *n-1* **do**
  $Inv \leftarrow \frac{1}{\mathbf{A[k,k]}}$;
  for i ***from*** $k+1$ ***to*** `getLastRow(A, k)` **do**
    if $\mathbf{A[i,k]} \neq 0$ **then**
      $l \leftarrow \mathbf{A[i,k]} \cdot Inv$;
      for j ***from*** $k+1$ ***to*** `getLastColumn(A, k)` **do**
        $\mathbf{A[i,j]} \leftarrow \mathbf{A[i,j]} - l \cdot \mathbf{A[k,j]}$;
      end
      $b[i] \leftarrow b[i] - l \cdot b[k]$
    end
  end
end
$x[n] \leftarrow \frac{b[n]}{A[n,n]}$;
for i ***from*** $n-1$ ***downto*** *1* **do**
  $x[i] \leftarrow b[i]$;
  for j ***from*** $i+_1$ ***downto*** `getLastColumn(A, i)` **do**
    $x[i] \leftarrow x[i] - A[i,j] \cdot x[j]$;
  end
  $x[i] \leftarrow \frac{x[i]}{\mathbf{A[i,i]}}$
end
return $\overline{x}$

2.2 Wariant z częściowym wyborem elementu głównego

Jak wcześniej wspomniano, ważnym założeniem w metodzie eliminacji Gaussa jest to, by elementy na przekątnej nie były zerowe, a najlepiej jakby były możliwie dalekie od zera.

Z tego też powodu, możemy poprzestawiać wiersze macierzy tak, aby na przekątnej znajdywały się jak największe elementy. Wybieramy więc taki wiersz m, że:

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

Element z tego wiersza zostanie zastosowany jako element główny, czyli zamieniony z wierszem k i zastosowany do zerowania elementów poniżej przekątnej po zamianie wierszy.

Zamiast faktycznie zamieniać miejscami wiersze, co byłoby dość kosztowne, zapisujemy po prostu wykonane zamiany w wektorze permutacji P.

Poza tą modyfikacją, metoda działa jak wariant podstawowy.

2.2.1 Pseudokod

Algorithm 2: Eliminacja Gaussa z częściowym wyborem elementu głównego (dla macierzy blokowej)

Input: A – macierz współczynników (BlockMatrix), b – wektor prawych stron
Output: x – wektor rozwiązania

$n \leftarrow size(b)$;
$x \leftarrow \overline{0}$;
$p \leftarrow [1, 2, ..., n]$;
for k ***from*** 1 ***to*** $n-1$ **do**
  $m \leftarrow k$;
  for i ***from*** $k+1$ ***to*** `getLastRow`(A, k) **do**
    if $|A[p[i], k]| > |A[p[m], k]|$ **then**
      $m \leftarrow i$
    end
  end
  $p[k], p[m] \leftarrow p[m], p[k]$;
  $Inv \leftarrow \frac{1}{\mathbf{A[k,k]}}$;
  for i ***from*** $k+1$ ***to*** `getLastRow`(A, k) **do**
    $l \leftarrow A[p[i], k] \cdot Inv$;
    $A[p[i], k] \leftarrow 0.0$;
    for j ***from*** $k+1$ ***to*** `getLastColumn`$(A, k + A.l)$ **do**
      $A[p[i], j] \leftarrow A[p[i], j] - l \cdot A[p[k], j]$;
    end
    $b[p[i]] \leftarrow b[p[i]] - l \cdot b[p[k]]$;
  end
end
$x[n] \leftarrow \frac{b[p[n]]}{A[p[n],n]}$;
for i ***from*** $n-1$ ***downto*** 1 **do**
  $x[i] \leftarrow b[p[i]]$;
  for j ***from*** $i+1$ ***to*** `getLastColumn`$(A, i + A.l)$ **do**
    $x[i] \leftarrow x[i] - A[p[i], j] \cdot x[j]$;
  end
  $x[i] \leftarrow \frac{x[i]}{A[p[i],i]}$;
end
return $\overline{x}$

3 Rozkład LU

Rozkładem LU nazywamy taki konkretny podział macierzy $\mathbf{A} = \mathbf{LU}$, gdzie macierz $\mathbf{L}$ jest macierzą dolnotrójkątną (L - *lower*), a $\mathbf{U}$ jest macierzą górnotrójkątną (U - *upper*).

Układ równań $\mathbf{A}x = b$ możemy wówczas zapisać jako $\mathbf{LU}x = b$, a więc sprowadzić do rozwiązania dwóch układów z macierzami trójkątnymi:

$$\mathbf{U}x = y$$

$$\mathbf{L}y = b$$

Jest to przydatne w sytuacji, gdy chcemy rozwiązać układ równań $\mathbf{A}x = b$ dla jednej macierzy $\mathbf{A}$ i różnych wektorów b. Wówczas zamiast jednego układu dostajemy dwa, aczkolwiek proste obliczeniowo, bowiem są to układy z macierzami trójkątnymi.

Idea metody

Możemy zauważyć, że w pierwszym etapie metody eliminacji Gaussa tworzymy macierz górnotrójkątną. Macierz ta jest naszą macierzą $\mathbf{U}$. Wówczas naszym rozkładem $\mathbf{LU}$ jest macierz górnotrójkątna $\mathbf{U} = \mathbf{A^{(n)}}$ i taka macierz $\mathbf{L}$, że $\mathbf{A} = \mathbf{LU}$.

Każde przejście od macierzy $\mathbf{A^{(k)}}$ do macierzy $\mathbf{A^{(k+1)}}$ możemy zapisać jako:

$$\mathbf{A^{(k+1)}} = \mathbf{L^{(k)}}\mathbf{A^{(k)}},$$

gdzie

$$L^{(k)} = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & -l_{k+1,k} & 1 & & & \\ & & -l_{k+2,k} & & \ddots & & \\ & & \vdots & & & 1 & \\ & & -l_{n,k} & & & & 1 \end{bmatrix}$$

Proces sprowadzania macierzy $\mathbf{A^{(1)}}$ do macierzy górnotrójkątnej możemy więc zapisać jako:

$$\mathbf{A^{(n)}} = \mathbf{L^{(n-1)}} \dots \mathbf{L^{(2)}}\mathbf{L^{(1)}}\mathbf{A^{(1)}}.$$

Wówczas po wykonaniu przekształceń:

$$\mathbf{U} = \mathbf{L^{(n-1)}} \dots \mathbf{L^{(2)}}\mathbf{L^{(1)}}\mathbf{A}$$
$$\mathbf{A} = (\mathbf{L^{(n-1)}} \dots \mathbf{L^{(2)}}\mathbf{L^{(1)}})^{-1}\mathbf{U}$$
$$\mathbf{A} = \mathbf{L^{(1)^{-1}}}\mathbf{L^{(2)^{-1}}} \dots \mathbf{L^{(n-1)^{-1}}}\mathbf{U}$$

$$L^{(k)^{-1}} = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & l_{k+1,k} & 1 & & & \\ & & l_{k+2,k} & & \ddots & & \\ & & \vdots & & & 1 & \\ & & l_{n,k} & & & & 1 \end{bmatrix},$$

więc $\mathbf{L} = \mathbf{L^{(1)^{-1}}}\mathbf{L^{(2)^{-1}}} \dots \mathbf{L^{(n-1)^{-1}}}$:

$$L = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & 1 \end{bmatrix}$$

Można więc zauważyć, że w macierzy $\mathbf{L}$ występują te same współczynniki co te wyliczane w metodzie eliminacji Gaussa do zerowania elementów pod przekątną

Uwagi do implementacji

Zamiast zerować wartości macierzy pod przekątną, w ich miejscu zapisywane będą mnożniki l_{ij}, czyli te, które wcześniej służyły do zerowania tych elementów macierzy. Program ten więc zmienia postać podanej do niego macierzy **A**.

Analiza złożoności

Algorytm rozkładu **LU** jest bardzo zbliżony do pierwszego etapu algorytmu standardowej eliminacji Gaussa. Jedyną różnicą, jest fakt zapamiętywania mnożników służących do zerowania elementów pod przekątną - nie zmienia to jednak całkowitej złożoności.

Wersja niezoptymalizowana ma więc złożoność $O(n^3)$.

Po zastosowaniu takich samych optymalizacji jak w przypadku eliminacji Gaussa dla macierzy o takiej konkretnej strukturze otrzymujemy złożoność $O(n)$.

3.1 Wariant podstawowy

3.1.1 Pseudokod

Algorithm 3: Funkcja generująca macierze L i U metodą LU dla macierzy blokowej

Input: A – macierz współczynników (BlockMatrix)
Output: A – zmodyfikowana macierz, w której górny trójkąt zawiera U, a dolny trójkąt bez przekątnej zawiera L

```
1  n ← A.n;
2  for k from 1 to n − 1 do
3  |  Inv ← 1/A[k,k];
4  |  for i from k + 1 to getLastRow(A, k) do
5  |  |  if A[k, k] ≠ 0 then
6  |  |  |  l ← A[i, k] · Inv;
7  |  |  |  A[i, k] ← l;
8  |  |  |  for j from k + 1 to getLastColumn(A, k) do
9  |  |  |  |  A[i, j] ← A[i, j] − l · A[k, j];
10 |  |  |  end
11 |  |  end
12 |  end
13 end
```

3.2 Wariant z częściowym wyborem elementu głównego

3.2.1 Pseudokod

Algorithm 4: Funkcja generująca macierze L i U metodą LU dla macierzy blokowej z częściowym wyborem elementu głównego

Input: A – macierz współczynników (BlockMatrix)
Output: A – macierz, w której górny trójkąt zawiera U, a dolny trójkąt bez przekątnej zawiera L, p – wektor permutacji wierszy

```
1  n ← A.n;
2  p ← [1, 2, ..., n];
3  for k from 1 to n − 1 do
4  |  m ← k;
5  |  for i from k + 1 to getLastRow(A, k) do
6  |  |  if |A[p[i], k]| > |A[p[m], k]| then
7  |  |  |  m ← i;
8  |  |  end
9  |  end
10 |  p[k], p[m] ← p[m], p[k];
11 |  inv ← 1 / A[p[k], k];
12 |  for i from k + 1 to getLastRow(A, k) do
13 |  |  l ← A[p[i], k] · inv;
14 |  |  A[p[i], k] ← l;
15 |  |  for j from k + 1 to getLastColumn(A, k + A.l) do
16 |  |  |  A[p[i], j] ← A[p[i], j] − l · A[p[k], j];
17 |  |  end
18 |  end
19 end
20 return p;
```

4 Rozwiązywanie układu wykorzystując rozkład LU

Idea metody

Rozkład **LU** może zostać wykorzystany do rozwiązania układu równań $\mathbf{A}x = b$.

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

Zamiast rozwiązywać ten układ otrzymujemy dwa układy z macierzami trójkątnymi:

$$\mathbf{U}x = y$$

$$\mathbf{L}y = b$$

W celu rozwiązania tych układów najpierw dokonujemy przekształceń na wektorze b, który musimy poddać takim przekształceniom, jak w standardowej eliminacji Gaussa poddajemy macierz $\mathbf{A}$ w każdym z $n - 1$ kroków.

W następnym kroku rozwiązujemy układy równań $\mathbf{U}x = y$ i $\mathbf{L}y = b$ analogicznie jak w drugim kroku standardowej eliminacji Gaussa.

Analiza złożoności

Poddanie wektora b przekształceniom ma teoretyczną złożoność $O(n^2)$ w wersji niezoptymalizowanej. Rozwiązywanie układów równań w wersji niezoptymalizowanej również ma złożoność $O(n^2)$. Cała metoda ma więc złożoność $O(n^2)$.

Możemy jednak, podobnie jak w poprzednich metodach, zoptymalizować tę złożoność. Ograniczyć można ilość iteracji w wewnętrznej pętli przy modyfikowaniu wektora b, co zmniejsza złożoność do $O(n)$, a w dalszej części możemy dokonać przekształceń jak w drugim etapie zwykłej eliminacji Gaussa, co również redukuje złożoność tego etapu do $O(n)$.

Cała metoda rozwiązywania układów równań z wykorzystaniem rozkładu **LU** ma więc złożoność $O(n)$.

4.1 Wariant podstawowy

4.1.1 Pseudokod

Algorithm 5: Rozwiązywanie układu równań $Ax = b$ za pomocą rozkładu LU

Input: **LU** – macierz, zawierająca rozkład **LU**, b – wektor prawych stron
Output: x – rozwiązanie układu równań $\mathbf{Ax = b}$

```
1  n ← LU.n;
2  for k from 1 to n − 1 do
3  |  for i from k + 1 to getLastRow(LU, k) do
4  |  |  b[i] ← b[i] − LU[i, k] · b[k];
5  |  end
6  end
7  x ← 0̄;
8  x[n] ← b[n] / LU[n,n];
9  for i from n − 1 downto 1 do
10 |  x[i] ← b[i];
11 |  for j from i + 1 to getLastColumn(LU, i) do
12 |  |  x[i] ← x[i] − LU[i, j] · x[j];
13 |  end
14 |  x[i] ← x[i] / LU[i,i];
15 end
16 return x;
```

4.2 Wariant z częściowym wyborem elementu głównego

4.2.1 Pseudokod

Algorithm 6: Rozwiązywanie układu równań $Ax = b$ za pomocą rozkładu LU z częściowym wyborem elementu głównego

Input: **LU** – macierz, zawierająca rozkład **LU**, P – wektor permutacji, b – wektor prawych stron
Output: x – rozwiązanie układu równań $\mathbf{Ax} = \mathbf{b}$

```
1  n ← LU.n;
2  for k from 2 to n do
3  |  for i from getFirstColumn(LU, P[k]) to k − 1 do
4  |  |  b[P[k]] ← b[P[k]] − LU[P[k], i] · b[P[i]];
5  |  end
6  end
7  x ← 0̄;
8  x[n] ← b[P[n]] / LU[P[n],n];
9  for i from n − 1 downto 1 do
10 |  x[i] ← b[P[i]];
11 |  for j from i + 1 to getLastColumn(LU, i + LU.l) do
12 |  |  x[i] ← x[i] − LU[P[i], j] · x[j];
13 |  end
14 |  x[i] ← x[i] / LU[P[i],i];
15 end
16 return x;
```

5 Opis zastosowanej struktury danych

Do przechowywania macierzy o specyficznej strukturze jak w zadaniu została wykorzystana specjalna struktura, dzięki której jest to efektywne.

5.1 Pola struktury

- `n::Int` - Rozmiar macierzy (liczba wierszy i kolumn)
- `l::Int` - Rozmiar pojedynczego bloku
- `rows::Vector{Tuple{Int, Vector{Float64}}}`:
  - `offset::Int` - Indeks pierwszej kolumny w wierszu, od którego zaczynają się wartości różne od zera
  - `values::Vector{Float64}` - Wektor wartości znajdujących się pomiędzy pierwszą i ostatnią niezerową wartością włącznie w wierszu

5.2 Funkcje

- Konstruktor:
  - Tworzy pustą macierz o wymiarze n i rozmiarze bloku l
  - Ustawia wiersze na początkowo puste (każdy wiersz to (0, Float64[ ])).
- Getter ($M[i, j]$):
  - Zwraca wartość z pozycji (i, j)
  - Jeśli wartość na (i, j) nie istnieje, zwraca 0.0 (bo nie przechowujemy zer)
- Setter ($M[i, j] = value$):

  - Ustawia wartość na pozycji (i, j)
  - Obsługuje przypadki, gdy wiersz jest pusty, wartość ma być dodana przed istniejącymi wartościami, po istniejących wartościach lub w zakresie istniejących wartości
  - Automatycznie dodaje zera w miejscach pomiędzy istniejącymi wartościami, jeśli to konieczne

- Ładowanie macierzy z pliku (`loadMatrixFromFile`):
  - Wczytuje macierz z pliku tekstowego o formacie:
    * Pierwsza linia określa rozmiar macierz$n$ i rozmiar bloku l
    * Kolejne linie określają pozycje i, j i wartości do zapisania w macierzy

- Znajdowanie ostatniej kolumny z wartościami (`getLastColumn`):
  - Zwraca indeks ostatniej kolumny w wierszu, która może zawierać wartości niezerowe w danym wierszu

- Znajdowanie ostatniego wiersza z wartościami (`getLastRow`):
  - Zwraca indeks ostatniego wiersza, który może zawierać wartości niezerowe w danej kolumnie

- Wyświetlanie macierzy (`printMatrix`):
  - Wyświetla zawartość macierzy w formacie: wiersz, offset i wartości

5.3 Analiza złożoności

5.3.1 Złożoność obliczeniowa

- Złożoność wstawiania wartości:
  Jest zależna od miejsca, w którym wartość wstawiamy. W najgorszym wypadku jest to $O(k)$. gdzie k to liczba wstawionych zer między najbliższą niezerową wartością, a wstawianą wartością. W przypadku macierzy o konkretnej strukturze z zadania nie będzie to złożoność gorsza niż $O(l)$.

- Złożoność odczytywania wartości: $O(1)$

5.3.2 Złożoność pamięciowa

Przechowujemy n wierszy macierzy, gdzie w każdym znajduje się $O(l)$ elementów istotnych, więc złożoność wynosi $O(n \cdot l)$. Dla stałego l złożoność ta wynosi więc $O(n)$.

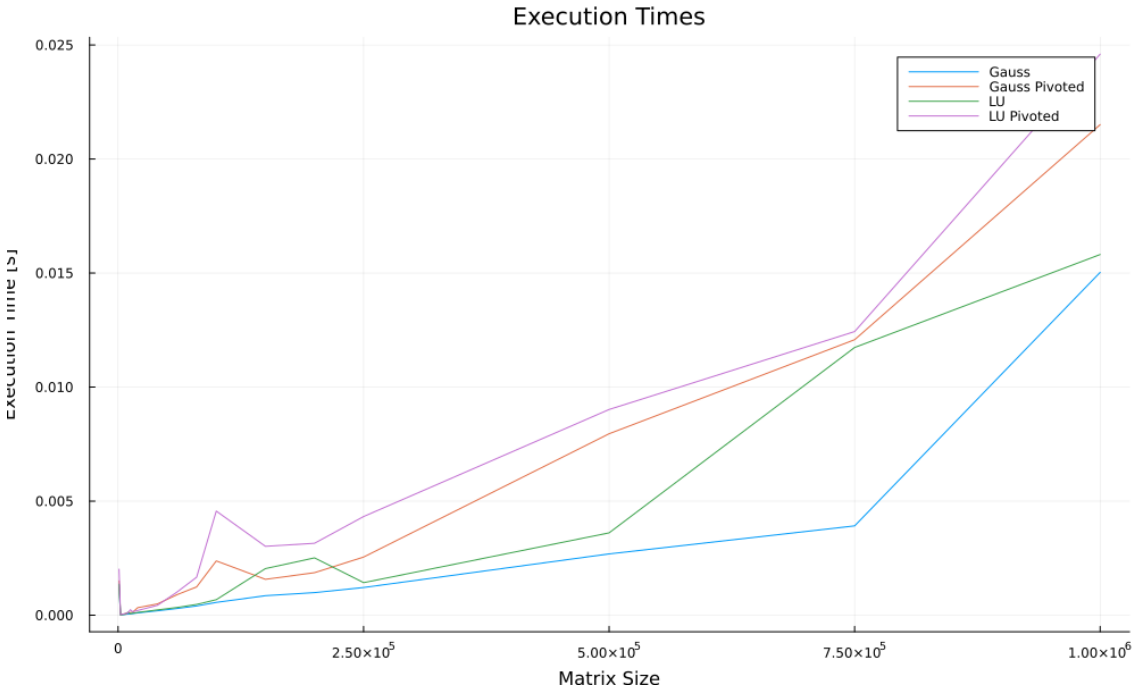
6 Wyniki

6.1 Informacje o testach

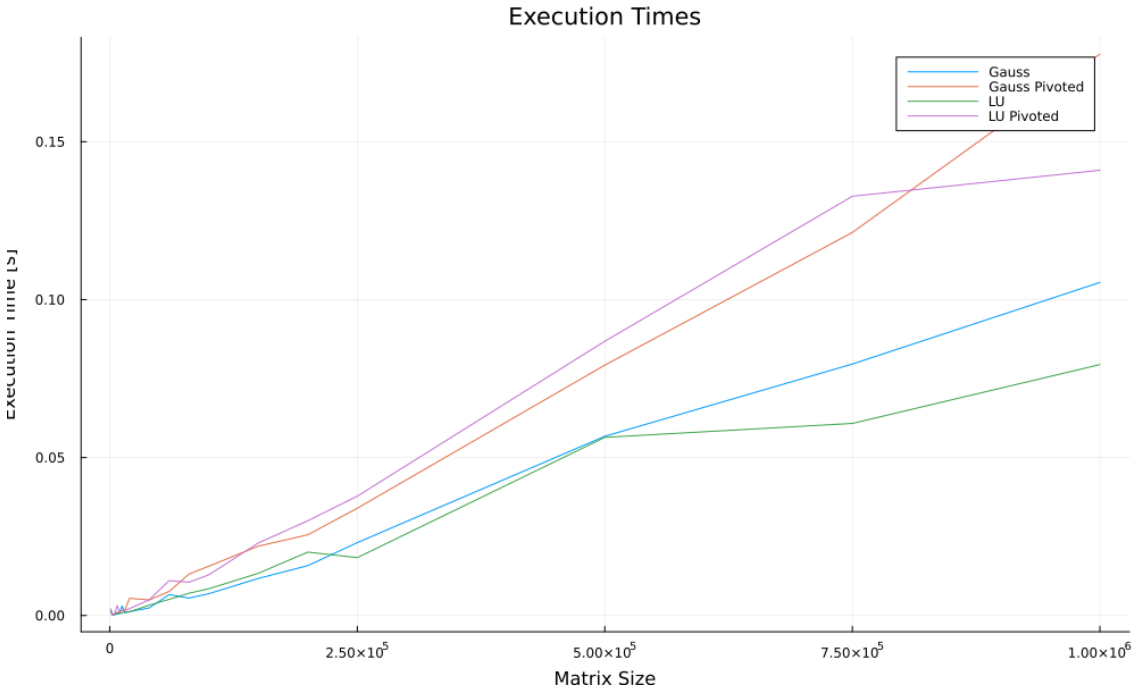
Testy zostały przeprowadzone dla macierzy o rozmiarach:
{1000, 2500, 5000, 7500, 10000, 12500, 15000, 20000, 40000, 60000, 80000, 100000, 150000, 200000, 250000, 500000, 750000, 1000000} i rozmiarze poszczególnego bloku 5 i 20. Każdy test został przeprowadzony 50 razy, a wynik został uśredniony. Czas był mierzony za pomocą `@timed`, a wektory prawych stron były generowane poprzez przemnożenie macierzy przez wektor jedynek.

Wyniki zostały przedstawione na wykresach i w tabelach.

6.2 Czasy wykonywania



Rysunek 1: Czasy wykonywania dla poszczególnych metod i rozmiaru bloku 5



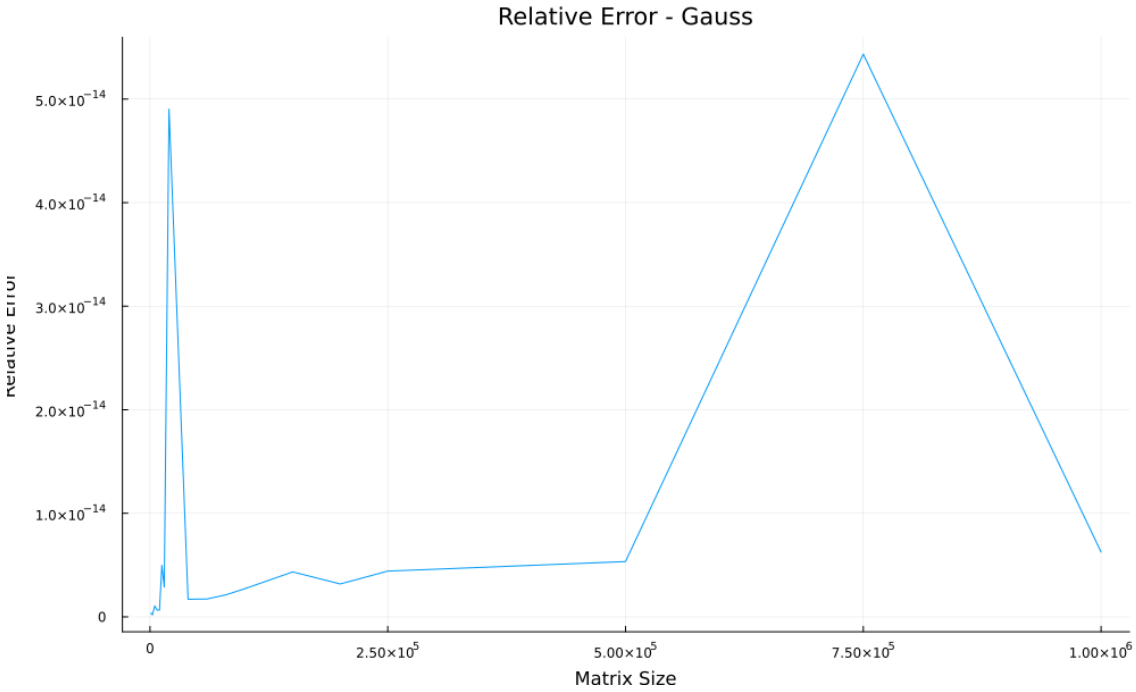
Rysunek 2: Czasy wykonywania dla poszczególnych metod i rozmiaru bloku 20

n	l	Gauss	Gauss z wyborem	LU	LU z wyborem
1000	5	1.329×10^{-3}	1.504×10^{-3}	1.363×10^{-3}	2.016×10^{-3}
1000	20	1.365×10^{-3}	1.593×10^{-3}	1.609×10^{-3}	2.157×10^{-3}
2500	5	1.135×10^{-3}	2.206×10^{-5}	1.293×10^{-5}	2.275×10^{-5}
2500	20	2.393×10^{-4}	2.284×10^{-4}	1.765×10^{-4}	2.297×10^{-4}
5000	5	2.378×10^{-5}	4.595×10^{-5}	2.668×10^{-5}	4.650×10^{-5}
5000	20	3.087×10^{-4}	5.329×10^{-4}	3.031×10^{-4}	6.017×10^{-4}
7500	5	3.646×10^{-5}	6.678×10^{-5}	1.317×10^{-4}	7.470×10^{-5}
7500	20	4.707×10^{-4}	9.458×10^{-4}	8.754×10^{-4}	3.096×10^{-3}
10000	5	4.846×10^{-5}	9.011×10^{-5}	6.720×10^{-5}	1.021×10^{-4}
10000	20	7.075×10^{-4}	1.041×10^{-3}	6.561×10^{-4}	1.051×10^{-3}
12500	5	5.747×10^{-5}	1.133×10^{-4}	8.585×10^{-5}	2.391×10^{-4}
12500	20	3.041×10^{-3}	1.576×10^{-3}	7.495×10^{-4}	1.708×10^{-3}
15000	5	7.075×10^{-5}	1.354×10^{-4}	1.204×10^{-4}	1.503×10^{-4}
15000	20	8.771×10^{-4}	1.542×10^{-3}	1.084×10^{-3}	1.583×10^{-3}
20000	5	1.001×10^{-4}	3.296×10^{-4}	2.313×10^{-4}	2.064×10^{-4}
20000	20	2.411×10^{-3}	5.433×10^{-3}	3.283×10^{-3}	5.011×10^{-3}
40000	5	1.920×10^{-4}	4.930×10^{-4}	3.373×10^{-4}	1.014×10^{-3}
40000	20	6.616×10^{-3}	7.635×10^{-3}	5.124×10^{-3}	1.019×10^{-2}
60000	5	2.917×10^{-4}	8.986×10^{-4}	4.719×10^{-4}	1.666×10^{-3}
60000	20	5.502×10^{-3}	1.310×10^{-2}	7.048×10^{-3}	1.053×10^{-2}
80000	5	4.044×10^{-4}	1.240×10^{-3}	6.800×10^{-4}	4.564×10^{-3}
80000	20	6.907×10^{-3}	1.561×10^{-2}	8.464×10^{-3}	1.286×10^{-2}
100000	5	5.606×10^{-4}	2.379×10^{-3}	2.042×10^{-3}	3.020×10^{-3}
100000	20	5.787×10^{-3}	1.561×10^{-2}	1.336×10^{-2}	2.296×10^{-2}
150000	5	8.548×10^{-4}	1.573×10^{-3}	2.513×10^{-3}	3.152×10^{-3}
150000	20	2.307×10^{-2}	2.195×10^{-2}	1.831×10^{-2}	3.000×10^{-2}
200000	5	9.896×10^{-4}	1.864×10^{-3}	1.427×10^{-3}	3.610×10^{-3}
200000	20	5.675×10^{-2}	2.554×10^{-2}	5.640×10^{-2}	3.778×10^{-2}
250000	5	1.214×10^{-3}	2.549×10^{-3}	3.610×10^{-3}	9.019×10^{-3}
250000	20	7.961×10^{-2}	3.396×10^{-2}	6.081×10^{-2}	8.683×10^{-2}
500000	5	2.687×10^{-3}	7.955×10^{-3}	1.174×10^{-2}	1.243×10^{-2}
500000	20	1.055×10^{-1}	7.927×10^{-2}	7.943×10^{-2}	1.328×10^{-1}
750000	5	3.915×10^{-3}	1.208×10^{-2}	1.581×10^{-2}	2.460×10^{-2}
750000	20	1.777×10^{-1}	1.213×10^{-1}	5.640×10^{-2}	1.410×10^{-1}
1000000	5	1.503×10^{-2}	2.151×10^{-2}	1.581×10^{-2}	2.460×10^{-2}
1000000	20	5.675×10^{-2}	1.055×10^{-1}	7.943×10^{-2}	1.410×10^{-1}

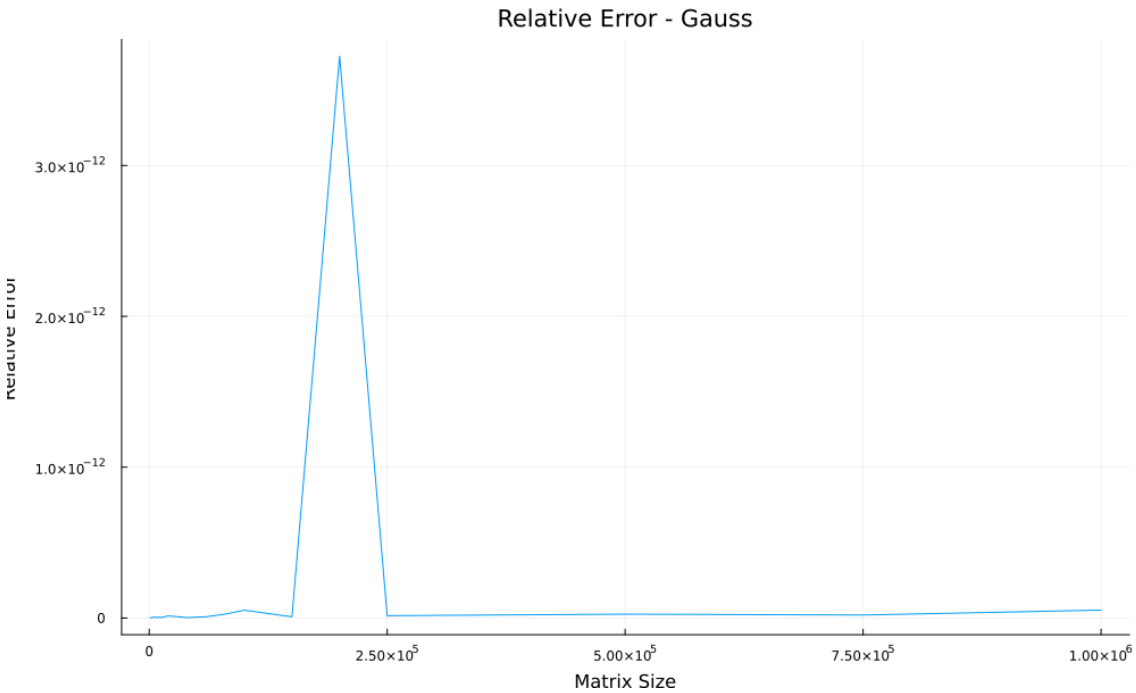
Tabela 1: Czasy dla różnych metod, rozmiarów problemów oraz wartości l, uporządkowane względem n.

6.3 Błędy względne

6.3.1 Metoda Gaussa

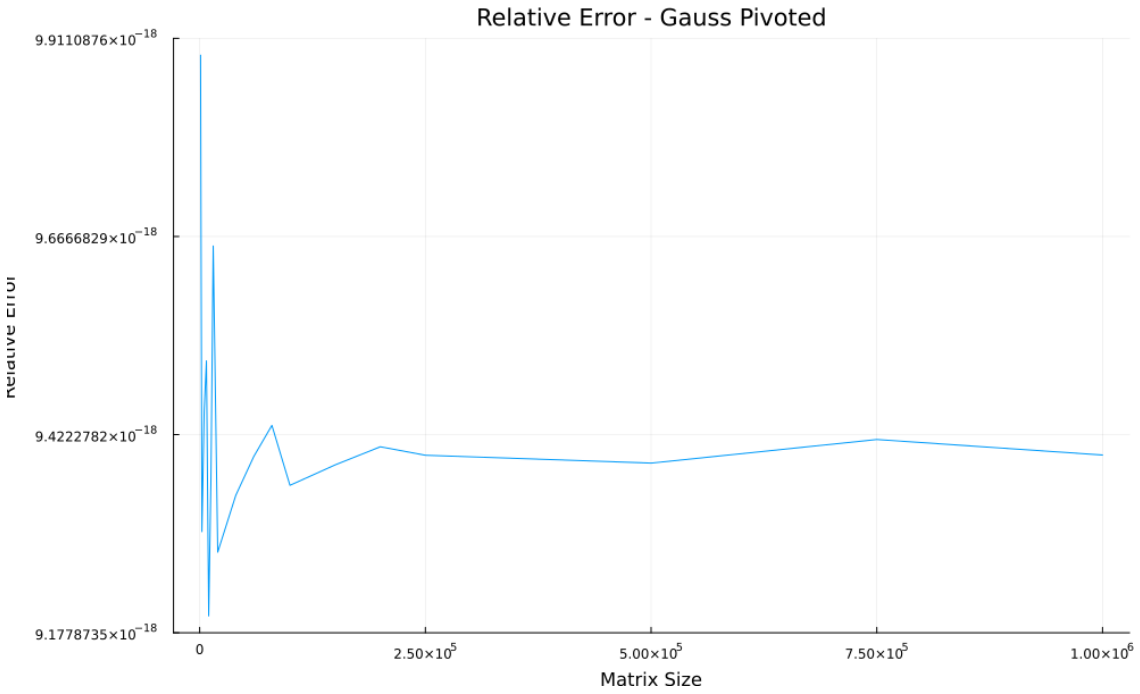


Rysunek 3: Błędy względne dla metody Gaussa i rozmiaru bloku 5

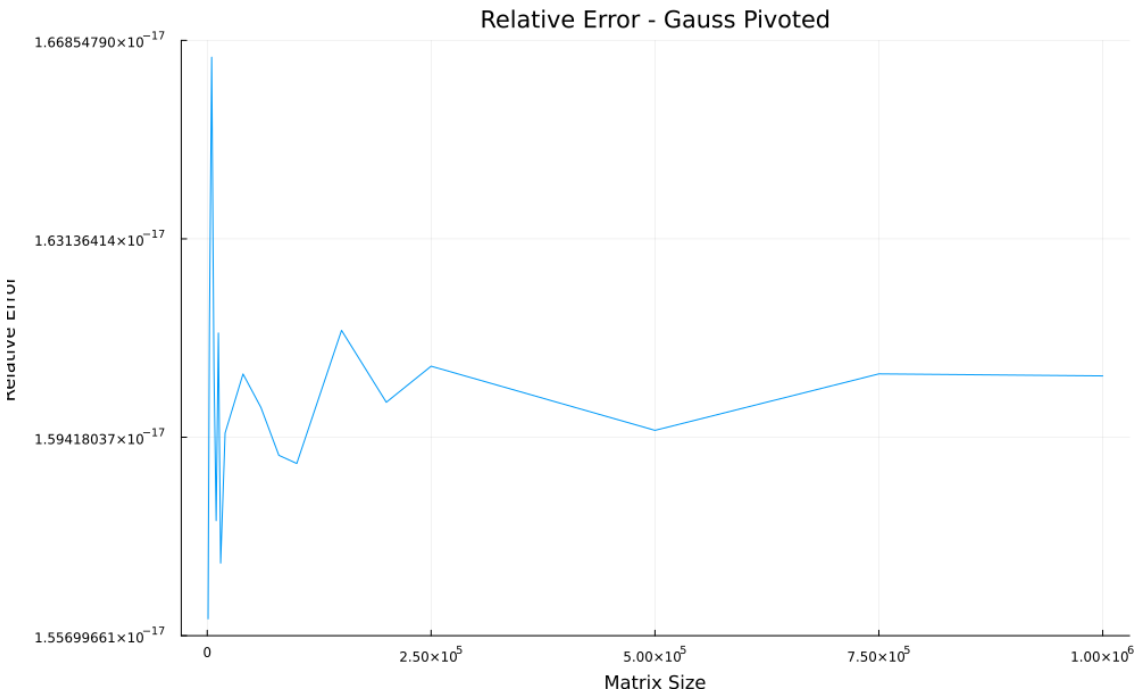


Rysunek 4: Błędy względne dla metody Gaussa i rozmiaru bloku 20

6.3.2 Metoda Gaussa z częściowym wyborem elementu głównego

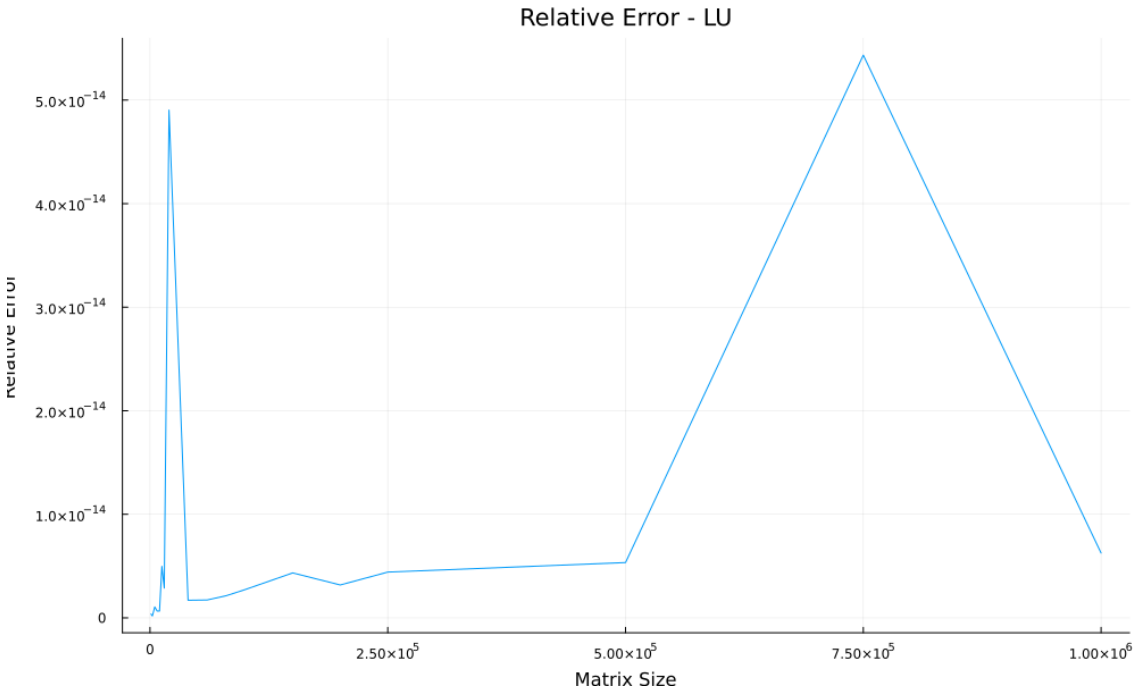


Rysunek 5: Błędy względne dla metody Gaussa z częściowym wyborem elementu głównego i rozmiaru bloku 5

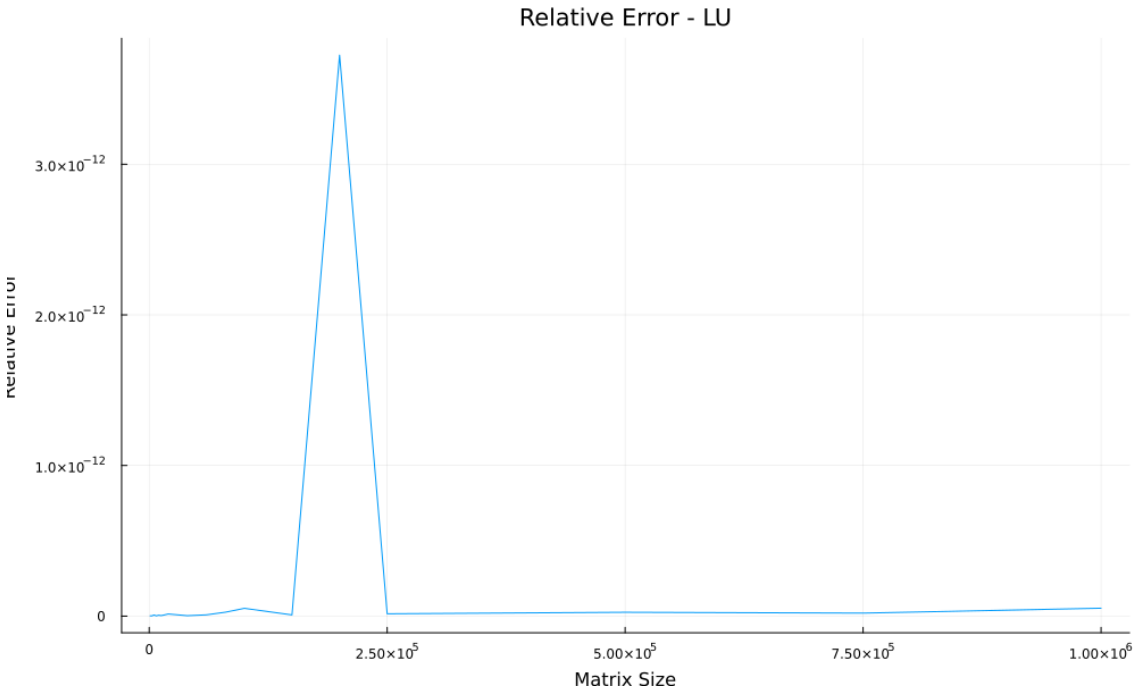


Rysunek 6: Błędy względne dla metody Gaussa z częściowym wyborem elementu głównego i rozmiaru bloku 20

6.3.3 Metoda z wykorzystaniem rozkładu LU

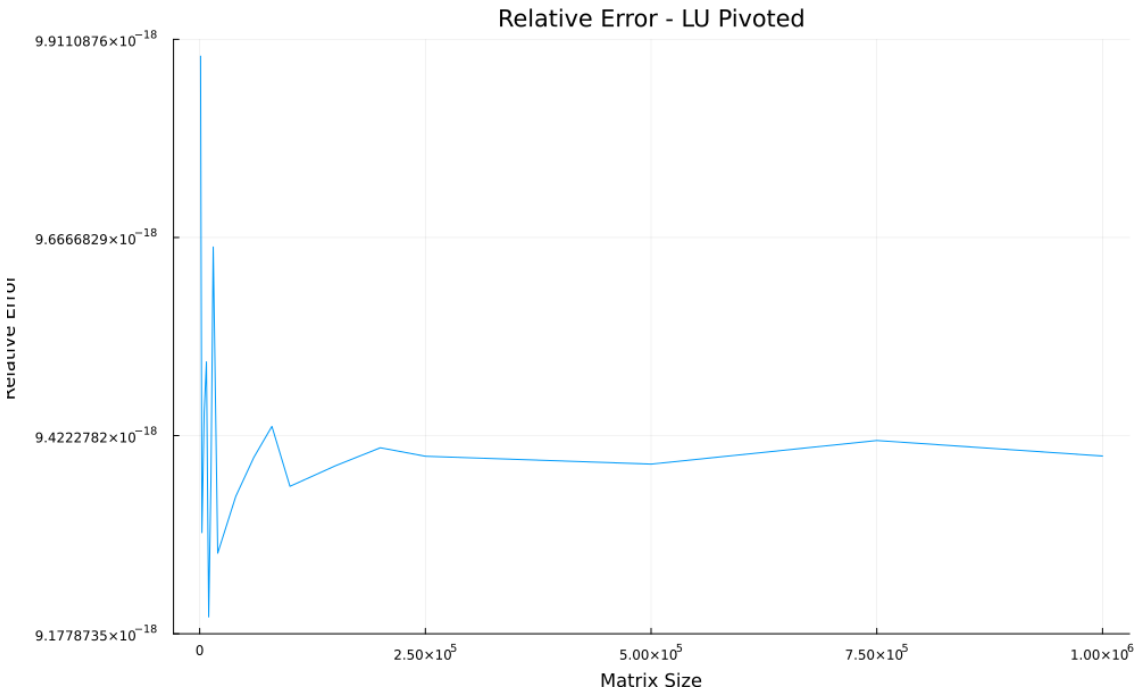


Rysunek 7: Błędy względne dla metody z wykorzystaniem rozkładu LU i rozmiaru bloku 5

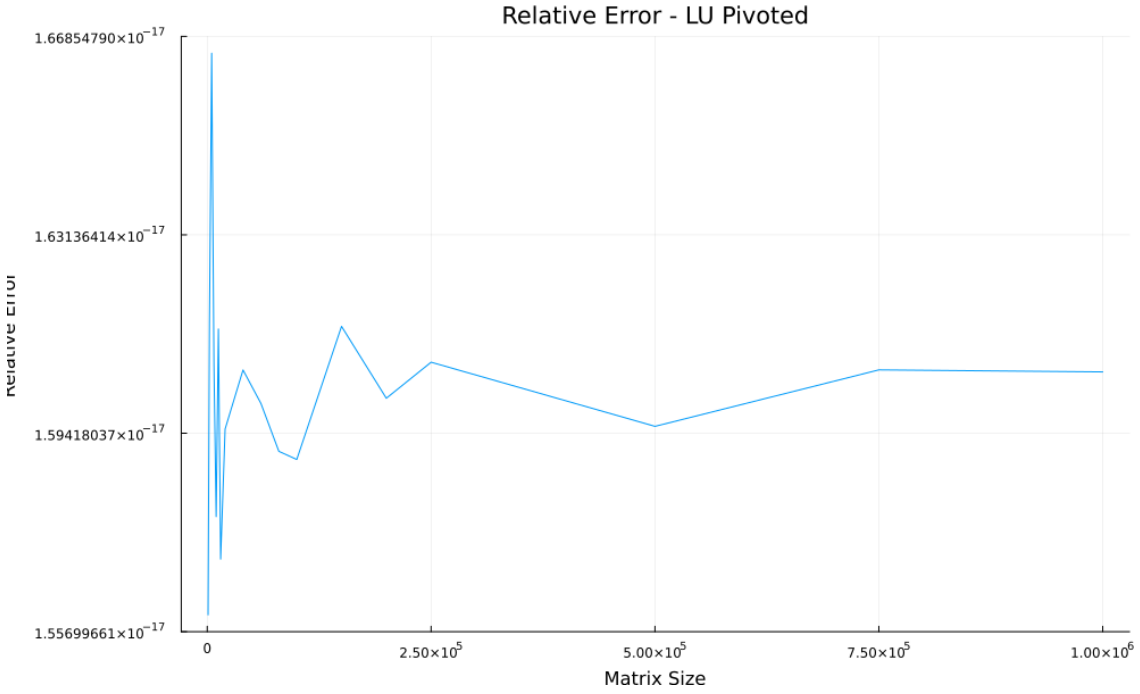


Rysunek 8: Błędy względne dla metody z wykorzystaniem rozkładu LU i rozmiaru bloku 20

6.3.4 Metoda z wykorzystaniem rozkładu LU z częściowym wyborem elementu głównego



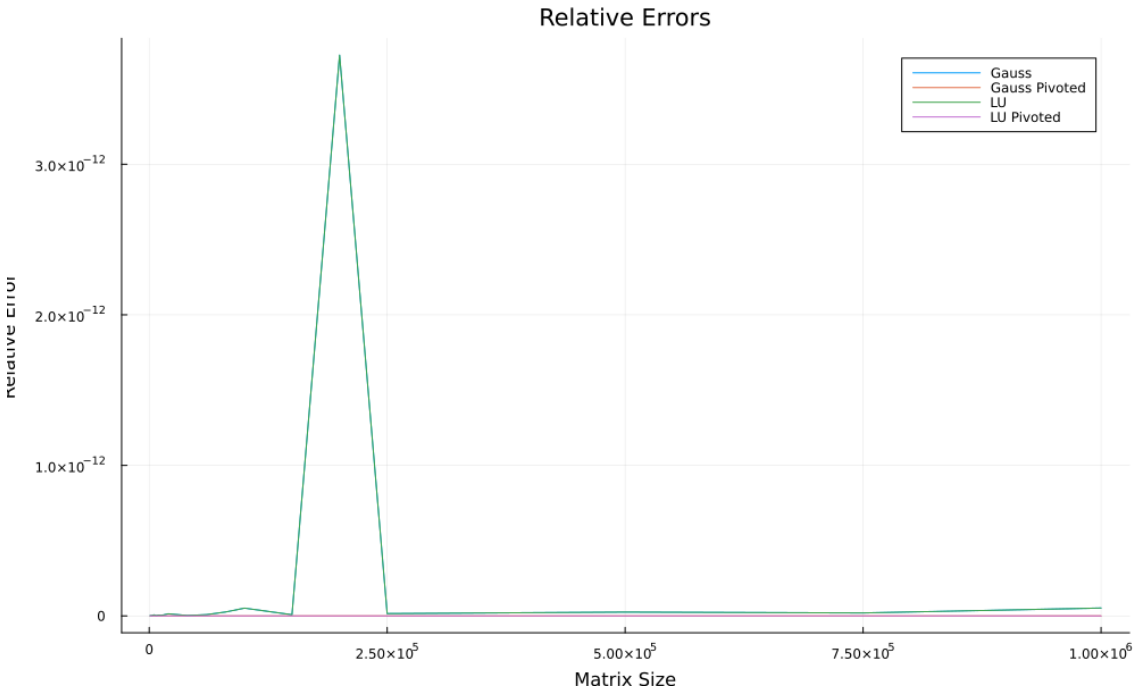
Rysunek 9: Błędy względne dla metody z wykorzystaniem rozkładu LU z częściowym wyborem elementu głównego i rozmiaru bloku 5



Rysunek 10: Błędy względne dla metody z wykorzystaniem rozkładu LU z częściowym wyborem elementu głównego i rozmiaru bloku 20

6.3.5 Porównanie błędów względnych otrzymywanych w każdej z metod

Rysunek 11: Błędy względne dla różnych metod i rozmiaru bloku 5



Rysunek 12: Błędy względne dla różnych metod i rozmiaru bloku 20

n	l	Gauss	Gauss z wyborem	LU	LU z wyborem
1000	5	3.68281×10^{-16}	9.89034×10^{-18}	3.68281×10^{-16}	9.89034×10^{-18}
1000	20	2.12016×10^{-15}	1.56015×10^{-17}	2.12016×10^{-15}	1.56015×10^{-17}
2500	5	1.85862×10^{-16}	9.30258×10^{-18}	1.85862×10^{-16}	9.30258×10^{-18}
2500	20	1.23763×10^{-15}	1.61497×10^{-17}	1.23763×10^{-15}	1.61497×10^{-17}
5000	5	1.05415×10^{-15}	9.45227×10^{-18}	1.05415×10^{-15}	9.45227×10^{-18}
5000	20	7.06058×10^{-15}	1.66539×10^{-17}	7.06058×10^{-15}	1.66539×10^{-17}
7500	5	6.56352×10^{-16}	9.51329×10^{-18}	6.56352×10^{-16}	9.51329×10^{-18}
7500	20	1.96807×10^{-15}	1.60722×10^{-17}	1.96807×10^{-15}	1.60722×10^{-17}
10000	5	6.56981×10^{-16}	9.19862×10^{-18}	6.56981×10^{-16}	9.19862×10^{-18}
10000	20	5.95812×10^{-15}	1.57855×10^{-17}	5.95812×10^{-15}	1.57855×10^{-17}
12500	5	4.97275×10^{-15}	9.36801×10^{-18}	4.97275×10^{-15}	9.36801×10^{-18}
12500	20	3.39845×10^{-15}	1.61371×10^{-17}	3.39845×10^{-15}	1.61371×10^{-17}
15000	5	2.87165×10^{-15}	9.65495×10^{-18}	2.87165×10^{-15}	9.65495×10^{-18}
15000	20	6.51904×10^{-15}	1.57060×10^{-17}	6.51904×10^{-15}	1.57060×10^{-17}
20000	5	4.90313×10^{-14}	9.27725×10^{-18}	4.90313×10^{-14}	9.27725×10^{-18}
20000	20	1.39489×10^{-14}	1.59497×10^{-17}	1.39489×10^{-14}	1.59497×10^{-17}
40000	5	1.69979×10^{-15}	9.34734×10^{-18}	1.69979×10^{-15}	9.34734×10^{-18}
40000	20	2.84690×10^{-15}	1.60604×10^{-17}	2.84690×10^{-15}	1.60604×10^{-17}
60000	5	1.72512×10^{-15}	9.39542×10^{-18}	1.72512×10^{-15}	9.39542×10^{-18}
60000	20	8.37742×10^{-15}	1.59976×10^{-17}	8.37742×10^{-15}	1.59976×10^{-17}
80000	5	2.13472×10^{-15}	9.43366×10^{-18}	2.13472×10^{-15}	9.43366×10^{-18}
80000	20	2.58259×10^{-14}	1.59079×10^{-17}	2.58259×10^{-14}	1.59079×10^{-17}
100000	5	2.72921×10^{-15}	9.35973×10^{-18}	2.72921×10^{-15}	9.35973×10^{-18}
100000	20	5.16917×10^{-14}	1.58926×10^{-17}	5.16917×10^{-14}	1.58926×10^{-17}
150000	5	4.33956×10^{-15}	9.38490×10^{-18}	4.33956×10^{-15}	9.38490×10^{-18}
150000	20	8.01283×10^{-15}	1.61420×10^{-17}	8.01283×10^{-15}	1.61420×10^{-17}
200000	5	3.17469×10^{-15}	9.40726×10^{-18}	3.17469×10^{-15}	9.40726×10^{-18}
200000	20	3.72442×10^{-12}	1.60074×10^{-17}	3.72442×10^{-12}	1.60074×10^{-17}
250000	5	4.41946×10^{-15}	9.39681×10^{-18}	4.41946×10^{-15}	9.39681×10^{-18}
250000	20	1.54266×10^{-14}	1.60749×10^{-17}	1.54266×10^{-14}	1.60749×10^{-17}
500000	5	5.33832×10^{-15}	9.38713×10^{-18}	5.33832×10^{-15}	9.38713×10^{-18}
500000	20	2.54535×10^{-14}	1.59547×10^{-17}	2.54535×10^{-14}	1.59547×10^{-17}
750000	5	5.43382×10^{-14}	9.41623×10^{-18}	5.43382×10^{-14}	9.41623×10^{-18}
750000	20	1.97665×10^{-14}	1.60605×10^{-17}	1.97665×10^{-14}	1.60605×10^{-17}
1000000	5	6.26107×10^{-15}	9.39708×10^{-18}	6.26107×10^{-15}	9.39708×10^{-18}
1000000	20	5.25014×10^{-14}	1.60567×10^{-17}	5.25014×10^{-14}	1.60567×10^{-17}

Tabela 2: Błędy względne dla różnych metod, rozmiarów problemów oraz wartości l, uporządkowane względem n.

6.4 Wnioski

6.4.1 Czasy wykonywania

Dość spodziewaną obserwacją jest to, że wraz z wzrostem n rośnie czas wykonywania dla każdej z metod. Wartości te rosną raczej liniowo, co zgadzało by się z naszymi przewidywaniami co do złożoności algorytmów.

Czasy wykonywania rosną także wraz z wzrostem wielkości pojedynczego bloku, co jest obserwacją może mniej oczywistą, ale jednak nadal spodziewaną. Wzrost ten również jest liniowy.

Metody, które wykorzystują częściowy wybór elementu głównego wypadają wolniej niż te, które tego nie robią, co także jest spodziewane, jako że wykonują one więcej operacji.

6.4.2 Błędy względne

Metody nie wykorzystujące częściowego wyboru elementu głównego mają zdecydowanie większe błędy względne, co również jest spodziewane, gdyż element główny częściowo wybieramy właśnie po to by uzyskiwane wyniki były bardziej zbliżone do rzeczywistych wartości.

Obie metody nie wykorzystujące częściowego wyboru elementu głównego osiągają podobne wartości błędów względnych - są to wartości rzędu około 10^{-15} dla $l = 5$ i około 10^{-14} dla $l = 20$.

Obie metody wykorzystujące częściowy wybór elementu głównego osiągają podobne wartości błędów względnych - są to wartości rzędu około 10^{-18} dla $l = 5$ i około 10^{-17} dla $l = 20$.

Można więc stwierdzić, że wszystkie metody zwracają bardziej dokładne wyniki dla macierzy o mniejszej wielkości pojedynczego bloku.

6.4.3 Wnioski ogólne

W przypadku rozwiązywania układu równań z macierzą blokową o takiej konkretnej strukturze lepszym rozwiązaniem jest użycie którejś z metod wykorzystującej częściowy wybór elementu głównego. Metody te są nieco wolniejsze, jednak błędy względne przez nie osiągane są zdecydowanie mniejsze, a to jest dla nas ważne.

Metoda Gaussa z częściowym wyborem elementu głównego jest w tym przypadku na ogół nieco szybsza, więc jej wybór może być dobry gdy zależy nam na czasie, nawet jeśli różnica ta jednorazowo nie jest aż tak duża.

W sytuacji gdy potrzebujemy rozwiązywać układ równań wiele razy dla tej samej macierzy $\mathbf{A}$, ale dla różnych wektorów b lepszym wyborem będzie metoda z rozkładem $\mathbf{LU}$ z częściowym wyborem elementu głównego, gdyż w takiej sytuacji jest ona bardziej opłacalna.

Dzięki optymalizacji problemu rozwiązywania układu równań z macierzą o znanej strukturze byliśmy w stanie zmniejszyć złożoność tego problemu z $O(n^3)$ do $O(n)$ przy zachowaniu pełnej funkcjonalności i z problemu, na którego rozwiązanie można by czekać godzinami sprowadziliśmy go do rozwiązywalnego w czasie praktycznie pomijalnym. Jest to ważna obserwacja w kwestii implementacji jakichkolwiek algorytmów gdzie pracujemy na dużych danych o określonej strukturze.