Lista 5

Wojciech Rymer

Styczeń 2023

1 Abstrakt

Ta praca przedstawia zbiór algorytmów rozwiązujących klasę układów równań liniowych których macierz współczynników ma postać klatkową trój-diagonalną. Proponowane algorytmy opierają się na eliminacji Gaussa i rozkładzie LU w wariancie zwykłym i z częściowym wyborem. Celem pracy było dostosowanie tradycyjnych metod rozwiązywania układów równań liniowych do specyficznej postaci macierzy by zwiększyć ich wydajność.

2 Opis Problemu

Jednostka badawcza dużej firmy działającej w branży chemicznej prowadzi intensywne badania. Wynikiem tych badań są modele pewnych zjawisk chemii kwantowej. Rozwiązanie tych modeli, w pewnym szczególnym przypadku, sprowadza się do rozwiązania układu równań liniowych

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

dla macierzy $\mathbf{A} \in \mathbb{R}^{nxn}$ i wektora prawych stron $\mathbf{b} \in \mathbb{R}^n$, $n \geqslant 4$. Macierz $\mathbf{A}$ jest rzadką macierzą klatkową trój-diagonalną o strukturze:

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

$v = n/l$, zakładając, że $n/l \in \mathbb{Z}$, gdzie $l \geqslant 2$ jest rozmiarem wszystkich bloków wewnętrznych: $\mathbf{A}_k$, $\mathbf{B}_k$ i $\mathbf{C}_k$. Macierz $\mathbf{A}_k \in \mathbb{R}^{l \times l}$, $k = 1, ..., v$. $\mathbf{B}_k \in \mathbb{R}^{l \times l}$, $k = 2, ..., v$ ma postać:

$$\mathbf{B_k} = \begin{pmatrix} b^k_{1,1} & \dots & b^k_{1,l-2} & b^k_{1,l-1} & b^k_{1,l} \\ 0 & \dots & 0 & 0 & b^k_{2,l} \\ 0 & \dots & 0 & 0 & b^k_{3,l} \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & b^k_{l,l} \end{pmatrix}$$

$\mathbf{C_k} \in \mathbb{R}^{l \times l}$ dla $k = 1, ..., v-1$. jest macierzą diagonalną:

$$\mathbf{C_k} = \begin{pmatrix} c^k_1 & 0 & 0 & \dots & 0 \\ 0 & c^k_2 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & c^k_{l-1} & 0 \\ 0 & \dots & 0 & 0 & c^k_l \end{pmatrix}$$

Rozwiązanie $\mathbf{Ax} = \mathbf{b}$ klasycznymi algorytmami dla dużych macierzy rzędu np. $n = 500000$ jest mało efektywne obliczeniowo Potrzebny jest algorytm dedykowany do problemu. Użyta musi zostać też specjalna struktura danych która pamięta tylko elementy niezerowe.

3 Struktura danych

Aby efektywnie przechowywać dane, wykorzystana została struktura SparseMatrix z biblioteki SparseArrays. Format CSC oferuje liniową złożoność pamięciową względem ilości elementów. Implementacja SparseMatrix w języku julia pamięta w wektorze elementy niezerowe, które są posortowane względem kolumn. Na potrzeby analizy złożoności zakłada się że dostęp do elementów realizowany jest w czasie stałym.

By przyśpieszyć działanie algorytmów, w momencie wczytywania macierzy $\mathbf{A}$ alokowane jest miejsce na elementy, które zostaną dodane do macierzy na wskutek obliczeń. Pozwala to ograniczyć ilość operacji wstawiania do wektora, które są kosztowna.

Format CSC zapewnia krótszy czas dostępu do kolumn w porównaniu do wierszy. Proponowane algorytmy częściej potrzebują dostępu do wierszy niż do kolumn. By wykorzystać potencjał formatu CSC, miejscami macierz jest transponowana.

4 Opis Algorymtów

Do rozwiązania problemu zaimplementowane zostały cztery algorytmy rozwiązujące układ równań. Opierają się one na znanych sposobach rozwiązywania układów równań, które zostały dostosowane do aktualnego problemu.

4.1 Eliminacja Gaussa

Zamysł Standardowym podejściem do rozwiązywania układu równań liniowych jest użycie eliminacji Gaussa. Pierwszym krokiem jest sprowadzenie macierzy współczynników do postaci trójkątnej, tzn. wyzerowanie wszystkich elementów pod główną przekątną. Elementy zerujemy kolejno kolumnami, zaczynąc od lewej strony. Aby wyzerować element $a_{j,i}$, należy odjąc od wiersza j wiersz i przemnożony przez $q = \frac{a_{i,j}}{a_{i,i}}$. Przykład zerowania pierwszej kolumny

$$\begin{bmatrix} 1 & 2 & 0 \\ 2 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & 0 \\ 0 & -4 & 1 \\ 1 & 1 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & 0 \\ 0 & -4 & 1 \\ 0 & -2 & 1 \end{bmatrix}$$

Zerując macierz, należy też wykonywać te same operacje dla wektora prawych stron $\mathbf{b}$, tzn, $\mathbf{b}[j] = \mathbf{b}[j] - \mathbf{b}[i] * q$. Kolejnym krokiem w eliminacji Gaussa jest wyznaczenie wartości wektora $\mathbf{x}$. Mając macierz $\mathbf{A}$ w postaci trójkątnej górnej, wyznaczenie $\mathbf{x}$ zaczynamy od $x_n = b_n/a_{n,n}$. Wyznaczając x_i znane są x_k dla $k = i-1, ..., n$ więc w każdym równaniu będziemy mieli tylko jedną niewiadomą i rozwiązując to równanie otrzymamy jej wartość. Kierujemy się wzorem:

$$x_i = \frac{b_i - \sum_{k=n}^{i-1} x_k a_{i,k}}{a_{i,i}}$$

Dostosowanie do problemu Tę metodę można przerobić do specyfiki problemu i zmodyfikować w obydwu krokach. Zauważmy, że przy zerowaniu kolumn ilość niezerowych elementów poniżej przekątnej jest równa co najwyżej l. Tak samo ilość elementów w wierszach przy odejmowaniu jest równa co najwyżej $2l + 1$. Przy wyliczaniu $\mathbf{x}$ ilość elementów niezerowych na prawo od przekątnej również jest ograniczona przez l. Wszystkie te warunki pozwalają znacząco ograniczyć liczbę wykonywanych operacji.

Analiza Złożoności W pierwszym kroku, dla każdej z n kolumn, zerujemy co najwyżej l elementów i dla każdego z nich odejmujemy wektory o długości co najwyżej $2l+1$. Daje nam to złożoność obliczeniową $O((2l^2 + l)n)$, Przy rozwiązywaniu układu równań z macierzą trójkątną dla n elementów, wykonywane jest co najwyżej $l + 1$ operacji, daje to $O((l + 1)n)$. W sumie dostajemy złożoność obliczeniową $O((2l^2 + 2l + 1)n)$ co przy założeniu, że l jest stałe daje $O(n)$. Wszystkie działania odbywają się na macierzy $\mathbf{A}$ oraz wektorach $\mathbf{x}$ i $\mathbf{b}$. Złożoność pamięciowa zależy od ilości elementów w tych strukturach. Dla $\mathbf{x}$ i $\mathbf{b}$ mamy po n elementów. Dla macierzy $\mathbf{A}$ przechowywanej w strukturze SparseMatrix, ilość zajmowanej pamięci jest zależna wyłącznie od ilości niezerowych elementów. Jednakże, aby przyśpieszyć działanie algorytmów, przy wczytywaniu macierzy alokowana jest dodatkowa pamięć. Dla każdego wiersza, przyjmuje się, że może on zawierać $3 * l$ elementów niezerowych. Daje to $3ln$ zaalokowanych elementów. Złożoność pamięciowa algorytmu to $O((3l + 2)n)$ co przy założeniiu że l jest stałe daje $O(n)$.

Algorithm 1 Sprowadzenie macierzy do postaci trójkątnej

input : A, **b**, n, l
output: A, **b**

$\mathbf{A} \leftarrow transpone(\mathbf{A})$
for $i \leftarrow 1$ **to** $n-1$ **do**
  $firstRow \leftarrow$min(i+1,n)
  lastRow$\leftarrow$min
  for $k \leftarrow firstRow$ **to** $lastRow$ **do**
    $q \leftarrow -A[i,k]/A[i,i]$;
    $A[i:lastRow,k] = A[i:lastRow,k] + A[i:lastRow,i] * q$
  end
  $A[i,k] = 0$
  $b[k] = b[k] + b[i] * q$
end
$\mathbf{A} \leftarrow transpone(\mathbf{A})$

Algorithm 2 Rozwiązanie układu równań z macierzą trójkątną

input : A, **b**, n, l
output: x

$\mathbf{x} \leftarrow zeroes(n)$
$\mathbf{x}[n] = \mathbf{b}[n]/\mathbf{A}[n,n]$ **for** $i \leftarrow n-1$ **to** 1 **do**
  $rowEnd \leftarrow$min(i+l,n)
  for $k \leftarrow rowEnd$ **to** $i+1$ **do**
    $\mathbf{b}[i] = \mathbf{b}[i] - \mathbf{A}[i,k] * \mathbf{x}[k]$
  end
  $\mathbf{x}[i] = \mathbf{b}[i]/\mathbf{A}[i,i]$
end

4.2 Eliminacja Gaussa z częściowym wyborem

Zamysł Wykonując eliminację gaussa możliwa jest sytuacja gdzie wyliczając $q = \frac{a_{i,j}}{a_{i,i}}$, $a_{i,i}$ będzie na tyle mniejsze od $a_{i,j}$, że skutkować to będzie znaczącym błędem obliczeń. W skrajnych przypadkach możliwe jest $a_{i,i} = 0$ co skutkowałoby błędem. By temu zapobiec i zminimalizować błąd, dla każdej i-tej kolumny można wybrać taki wiersz j, że $|a_{ji}| = max_{n>k>i}|a_{k,i}|$ i zamienić wiersz j z i. W **b** również zamieniamy i z j.

Optymalizacja Faktyczna zamiana ze sobą całych wierszów byłaby wyjątkowo kosztowna biorąc pod uwagę, że tę operację nalezy wykonać dla każdej kolumny. By temu zapobiec, w implementacji użyty został wektor permutacji $P \in \mathbb{R}^n$, taki że przy inicjalizacji $P[k] = k$ dla $k = 1, ..., n$. Przy zamianie wierszy wykonywany jest $swap(P[i], P[j])$. Przy dostępie do elementów wektora zamiast $a_{i,j}$ odwołanie następuje do $a_{P[i],j}$.

Złożoność Użycie wektora permutacji nie wpływa na stały czas dostępu do elementów. Różnicą w złożoności leży w wyszukaniu elementu maksymalnego i innej długości wierszy przy odejmowaniu. Dla każdej z n kolumn elementu maksymalnego szukamy spośród l kandydatów. Ograniczenie na długość wiersza wzrasta do $3l$ Nowa złożoność obliczeniowa to $O((2l^2 + 2l + 1)n) + O(nl) = O((2l^2 + 3l + 1)n)$ lub $O(n)$ dla stałego l. Złożoność pamięciowa zwiększa się o $|P| = n$ i wynosi $O(3l + 3)n)$.

Algorithm 3 Sprowadzenie macierzy do postaci trójkątnej z częściowym wyborem

input : **A**, **b**, n, l
output: **A**, **b**, **P**

$\mathbf{A} \leftarrow transpone(\mathbf{A})$
$\mathbf{P} \leftarrow [1..n]$ **for** $i \leftarrow 1$ **to** $n-1$ **do**
  $firstRow \leftarrow$min(i+1,n)
  lastRow$\leftarrow$min
  t $\leftarrow i$
  $max \leftarrow |\mathbf{A}[i][\mathbf{P}[i]]|$
  for $k \leftarrow firstRow$ **to** $lastRow$ **do**
    $c = |\mathbf{A}[i, P[k]]|$
    if $c > max$ **then**
      $max \leftarrow c$
      $t \leftarrow k$
    end
  end
  $swap(P[i], P[t])$
  for $k \leftarrow firstRow$ **to** $lastRow$ **do**
    $q \leftarrow -A[i, \mathbf{P}[k]/A[i, \mathbf{P}[i]];$
    $A[i : lastRow, \mathbf{P}[k]] = A[i : lastRow, \mathbf{P}[k]] + A[i : lastRow, \mathbf{P}[i]] * q$
  end
  $A[i, \mathbf{P}[k]] = 0$
  $b[k] = b[k] + b[i] * q$
end
$\mathbf{A} \leftarrow transpone(\mathbf{A})$

4.3 Rozkład LU

Zamysł Zauważmy, że wykonując eliminacje gaussa, operacje dodania do jednego wiersza, drugiego wymnożonego przez skalar można zapisać jako macierz. Przykład:
Niech $\mathbf{A_1} = \mathbf{A}$, Wtedy

$$\begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 0 \\ 2 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & -4 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$
$$\mathbf{L_1 A_1 = A_2}$$

Tak samo $\mathbf{L_1 b_1 = b_2}$. Oznaczmy finalny produkt eliminacji gaussa jako $\mathbf{A_n}$ i $\mathbf{b_n}$. Wtedy

$$\mathbf{L_{n-1}, ..., L_1 A_1 = A_n}$$
$$\mathbf{L_{n-1}, ..., L_1 b_1 = b_n}$$

Niech $\mathbf{U} = \mathbf{A_n}$. Wtedy:

$$\mathbf{U = L_{n-1}, ..., L_1 A}$$
$$\mathbf{(L_{n-1}, ..., L_1)^{-1} U = A}$$
$$\mathbf{L_1^{-1}, ..., L_{n-1}^{-1} U = A}$$
$$\mathbf{LU = A}$$

L ma postać:

$$L = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ q_{1,1} & 1 & 0 & \dots & 0 \\ q_{2,1} & q_{2,2} & 1 & \dots & 0 \\ q_{3,1} & q_{3,2} & q_{3,3} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_{n-1,1} & q_{n-1,2} & q_{n-1,3} & \dots & 1 \end{pmatrix}$$

Gdzie $q_{w,k} = \frac{a_{w,k}}{a_{k,k}}$. Przyjrzyjmy się teraz układowi równań, który mamy rozwiązać:

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

Zadanie znalezienia **x** sprowadza się teraz do rozwiązania dwóch układów równań:

$$\mathbf{Ly} = \mathbf{b} \text{ i } \mathbf{Ux} = \mathbf{y}$$

Jako że zarówno **L** jak i **U** są macierzami trójkątnymi to obydwa te układy rozwiązuje się w zasadniczo ten sam sposób co w drugim kroku eliminacji Gaussa. Umożliwia to zamianę wektora prawych stron **b** niezależnie od macierzy **A**.

Optymalizacja Od strony implementacji, wykonywana jest tradycyjna eliminacja Gaussa, z tym dodatkiem, że przechowywane są mnożniki q. W **L**, $q_{w,k}$ znajdują się tam gdzie w **A** $a_{w,k}$. Co więcej, $q_{w,k}$ wyliczane są w momencie zerowania $a_{w,k}$. Żeby nie zużywać pamięci i czasu na tworzenie nowej macierzy, współczynniki $q_{w,k}$ przechowywane są w miejscach po zerowanych $a_{w,k}$ w **A**. Przy rozwiązywaniu równań, brana pod uwagę jest ograniczona długość wierszy w **U** i **L**, w obydwu przypadkach wynosi nie więcej niż $l + 1$.

Złożoność Porównując rozkład LU do eliminacji gaussa, w tej metodzie potrzeba rozwiązać dodatkowy układ równań z macierzą trójkątną $\mathbf{Ly} = \mathbf{b}$. Dla każdej z n kolumn wykonujemy nie więcej niż $l+1$ operacji. Daje nam to złożoność kroku $O((l + 1)n)$ i sumaryczną złożoność obliczeniową $O((2l^2 + 3l + 2)n)$ czyli $O(n)$ dla stałego l. Złożoność pamięciowa zwiększa się jedynie o rozmiar wektora **y** czyli $O(n)$. Elementy **L** są przechowywane w **A** więc nie są brane pod uwagę. Daje to złożoność pamięciową $O((3l + 3)n)$

Implementacja Przechowując macierz L i U w jednej strukturze, nie pamiętamy głównej przekątnej macierzy L, gdyż składa się ona z samych 1.

Algorithm 4 Obliczenie macierzy LU

input : A, n, l
output: LU

A $\leftarrow transpone(\mathbf{A})$
for $i \leftarrow 1$ **to** $n - 1$ **do**
  $firstRow \leftarrow$min(i+1,n)
  lastRow$\leftarrow$min
  for $k \leftarrow firstRow$ **to** $lastRow$ **do**
    $q \leftarrow -A[i, k]/A[i, i]$;
    $A[i : lastRow, k] = A[i : lastRow, k] + A[i : lastRow, i] * q$
  end
  $A[i, k] = -q$
end
LU $\leftarrow transpone(\mathbf{A})$

4.4 Rozkład LU z częściowym wyborem

Zamysł Obliczając rozkład LU, tak samo jak w tradycyjnej eliminacji Gaussa, może wystąpić przypadek $a_{k_k} = \epsilon$ gdzie ϵ jest wyjątkowo małe. Problem ten rozwiązujemy tak samo jak w Gaussie. Dla każdej kolumny szukamy $|a_{ji}| = max_{n>k>i}|a_{k,i}|$ a następnie zamieniamy wiersz i z j. Zamiana wierszy następuje zarówno w **A** jak i w **L**. Gdy rozwiązujemy układ równań, wektor b również musi ulec permutacji.

Optymalizacja Dostosowanie tego algorytmu do problemu wykorzystuje te same właściwości macierzy co algorytmy wyżej. Wiersze macierzy i elementy wektora nie są faktycznie zamieniane. Wykorzystywany jest wektor permutacji **P** przez który realizowany jest dostęp do wierszy **A**, **L** i elementów **b**. Nałożone jest też ograniczenie na ilość elementów w wierszu, które wynosi $3l$ dla **A** oraz l dla **L**.

Złożoność Częściowy wybór wymaga znalezienia największego względem modułu elementu zerowanej kolumny znajdującego się pod przekątną. Dla każdej z n kolumn, jest co najwyżej l niezerowych elementów pod przekątną, co daje złożoność $O(nl)$. Zarządzanie wektorem permutacji pochłania $O(n)$ operacji. Zwiększa to złożoność obliczeniową do $O((2l^2 + 3l + 2)n)$ czyli $O(n)$ dla stałego l. W trakcie wykonywania algorytmu pamiętana jest macierz permutacji **P**. Dodaje to do złożoności pamięciowej $O(n)$ czyli wynosi ona $O((3l + 4)n)$.

Algorithm 5 Rozwiązanie układu równań z użyciem rozkładu LU

input : LU, **b**, n, l
output: x

$\mathbf{x} \leftarrow zeroes(n)$
$\mathbf{y} \leftarrow zeroes(n)$
$\mathbf{y}[1] = \mathbf{b}[1]$
for $i \leftarrow 2$ **to** n **do**
  $rowStart \leftarrow$max(i-l,1)
  for $k \leftarrow rowStart$ **to** $i - 1$ **do**
    $\mathbf{b}[i] = \mathbf{b}[i] - \mathbf{LU}[i, k] * \mathbf{y}[k]$
  end
  $\mathbf{y}[i] = \mathbf{b}[i]$
end
$\mathbf{x}[n] = \mathbf{b}[n]/\mathbf{LU}[n, n]$ **for** $i \leftarrow n - 1$ **to** 1 **do**
  $rowEnd \leftarrow$min(i+l,n)
  for $k \leftarrow rowEnd$ **to** $i + 1$ **do**
    $\mathbf{y}[i] = \mathbf{y}[i] - \mathbf{LU}[i] * \mathbf{x}[k]$
  end
  $\mathbf{x}[i] = \mathbf{y}[i]/\mathbf{LU}[i, i]$
end

Algorithm 6 Obliczenie macierzy LU z częściowym wyborem

input : A, n, l
output: LUP

$\mathbf{A} \leftarrow transpone(\mathbf{A})$
$\mathbf{P} \leftarrow [1..n]$ **for** $i \leftarrow 1$ **to** $n - 1$ **do**
  $firstRow \leftarrow$min(i+1,n)
  lastRow$\leftarrow$min
  t $\leftarrow i$
  $max \leftarrow |\mathbf{A}[i][\mathbf{P}[i]]|$
  for $k \leftarrow firstRow$ **to** $lastRow$ **do**
    $c = |\mathbf{A}[i, P[k]]|$
    if $c > max$ **then**
      $max \leftarrow c$
      $t \leftarrow k$
    end
  end
  $swap(P[i], P[t])$
  for $k \leftarrow firstRow$ **to** $lastRow$ **do**
    $q \leftarrow -A[i, \mathbf{P}[k]/A[i, \mathbf{P}[i]]$;
    $A[i : lastRow, \mathbf{P}[k]] = A[i : lastRow, \mathbf{P}[k]] + A[i : lastRow, \mathbf{P}[i]] * q$
  end
  $A[i, \mathbf{P}[k]] = -q$
end
$\mathbf{LU} \leftarrow transpone(\mathbf{A})$

Algorithm 7 Rozwiązanie układu równań z użyciem rozkładu LU z częściowym wyborem

input : LU, **b**, n, l
output: x

$\mathbf{x} \leftarrow zeroes(n)$
$\mathbf{y} \leftarrow zeroes(n)$
$\mathbf{y}[1] = \mathbf{b}[\mathbf{P}[1]]$
for $i \leftarrow 2$ **to** n **do**
  $rowStart \leftarrow$max(i-l,1)
  for $k \leftarrow rowStart$ **to** $i - 1$ **do**
    $\mathbf{b}[\mathbf{P}[i]] = \mathbf{b}[\mathbf{P}[i]] - \mathbf{LU}[\mathbf{P}[i], k] * \mathbf{y}[k]$
  end
  $\mathbf{y}[i] = \mathbf{b}[\mathbf{P}[i]]$
end
$\mathbf{x}[n] = \mathbf{b}[n]/\mathbf{LU}[\mathbf{P}[n], n]$ **for** $i \leftarrow n - 1$ **to** 1 **do**
  $rowEnd \leftarrow$min(i+l,n)
  for $k \leftarrow rowEnd$ **to** $i + 1$ **do**
    $\mathbf{y}[i] = \mathbf{y}[i] - \mathbf{LU}[i] * \mathbf{x}[k]$
  end
  $\mathbf{x}[i] = \mathbf{y}[i]/\mathbf{LU}[\mathbf{P}[i], i]$
end

5 Porównanie algorytmów

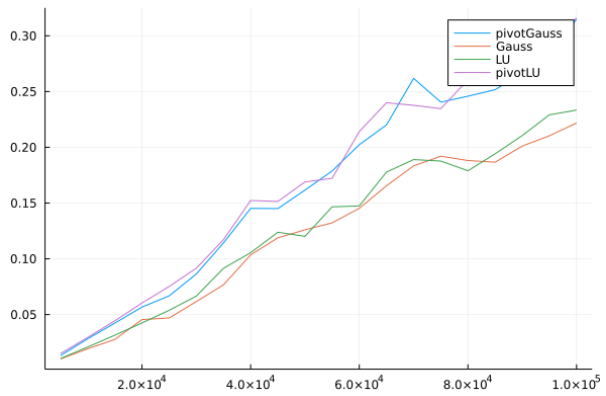
By zbadać algorytmy przeprowadzone zostały testy ich błędu, złożoności obliczeniowej i złożoności pamięciowej.

5.1 Testy

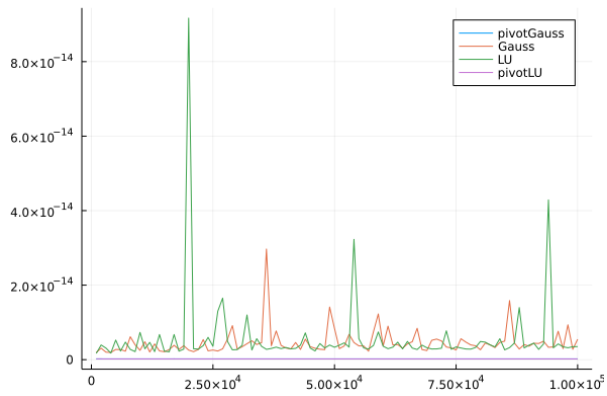
Testy zostały przeprowadzone na danych wygenerowanych przy pomocy modułu *matrixgen* dla $n = 5000, 10000, ..., 100000$, $l = 4$ i wskaźniku uwarunkowania $ck = 1.0$. Dla każdego n wykonano 10 prób i obliczono z nich średnią.

rozmiar macierzy	**16**	**10000**	**50000**	**100000**	**300000**	**500000**
Gauss	7.008282e-16	3.594413e-15	4.498712e-15	1.468484e-14	3.705667e-15	3.657342e-15
Gauss z wyborem	3.053113e-16	3.813283e-16	4.111377e-16	3.980127e-16	3.477466e-16	3.431419e-16
LU	7.008282e-16	3.594413e-15	4.498712e-15	1.468484e-14	3.705667e-15	3.657342e-15
LU z wyborem	3.053113e-16	3.813283e-16	4.111377e-16	3.980127e-16	3.477466e-16	3.431419e-16

Tabela 1: Porównanie błędu względnego obliczeń dla różnych n

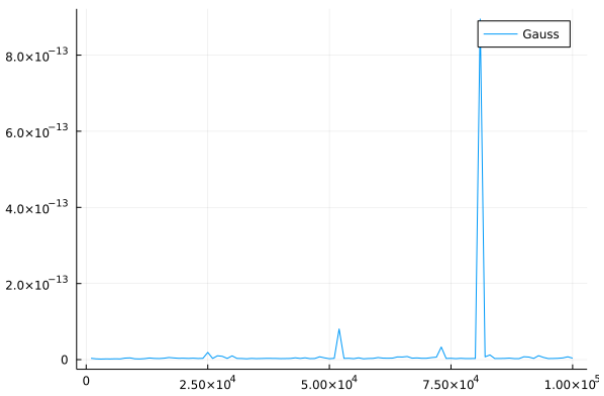


(a) Złożoność obliczeniowa

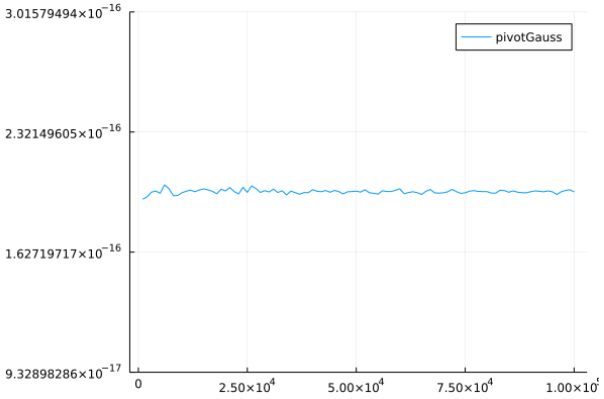


(b) Błąd obliczeń

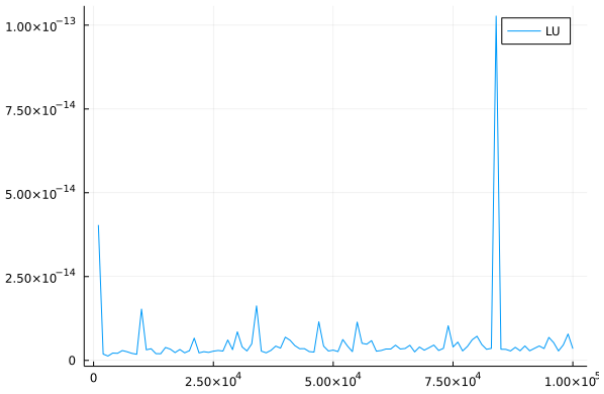
Rysunek 1: Porównanie algorytmów



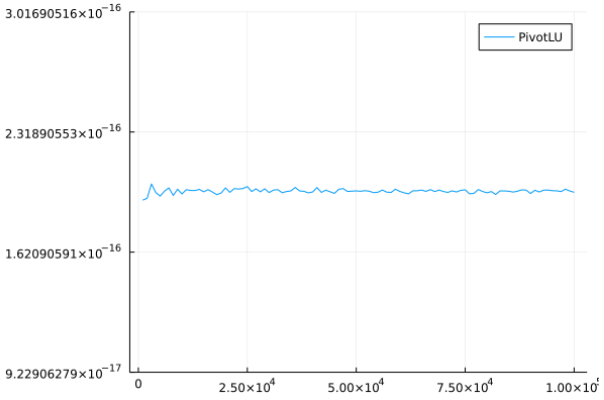
(a) Gauss



(b) Gauss z wyborem

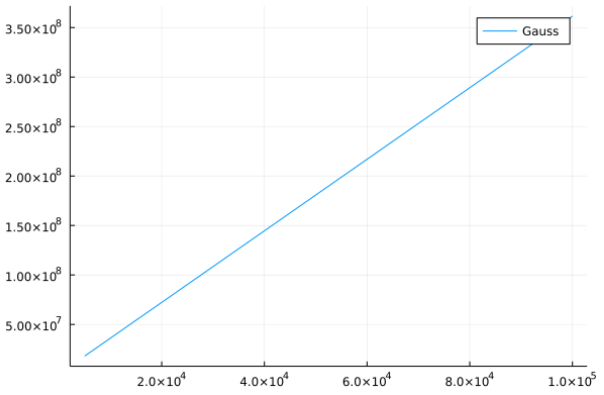


(c) LU

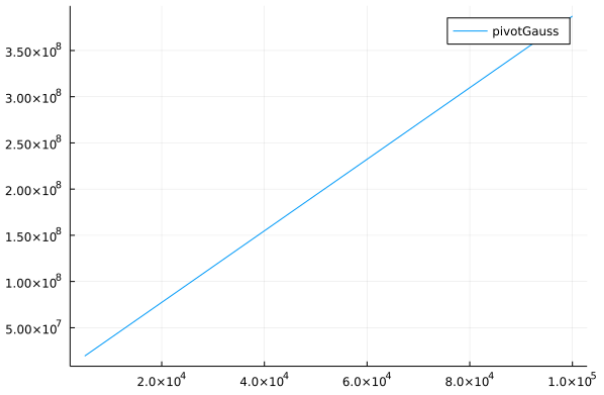


(d) LU z wyborem

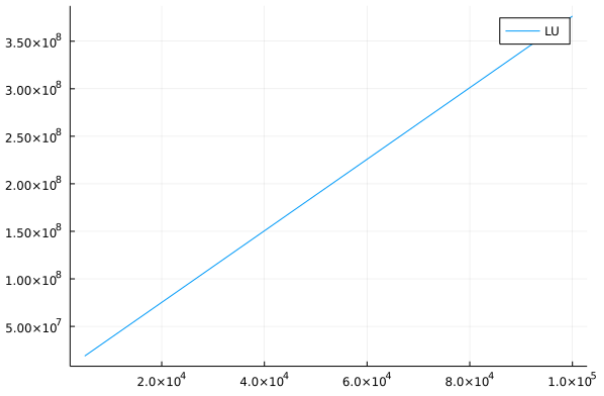
Rysunek 2: Analiza błędu



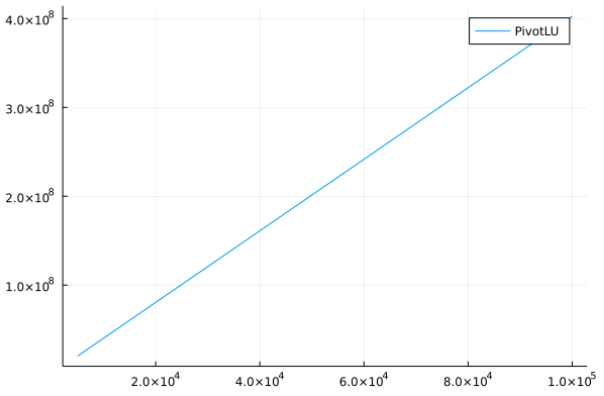
(a) Gauss



(b) Gauss z wyborem

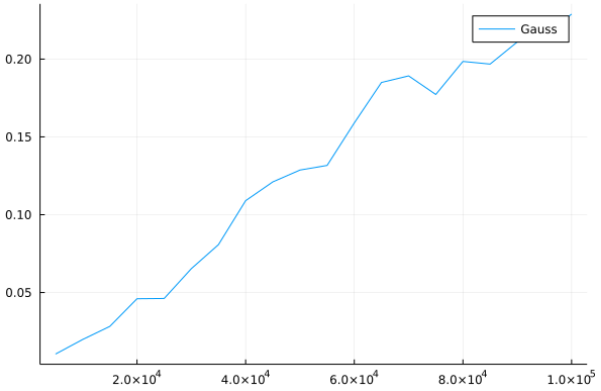


(c) LU

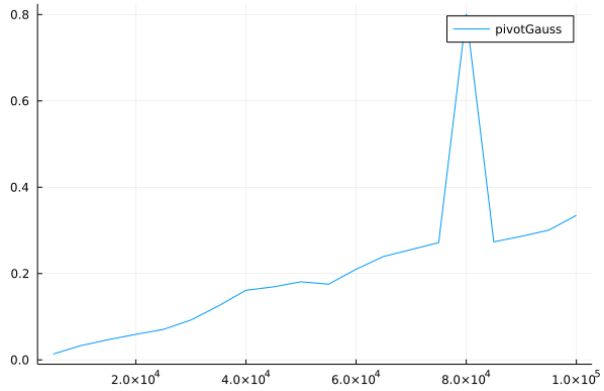


(d) LU z wyborem

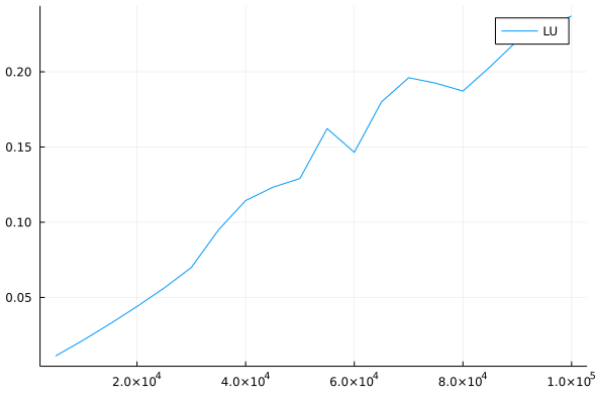
Rysunek 3: Analiza zużycia pamięci



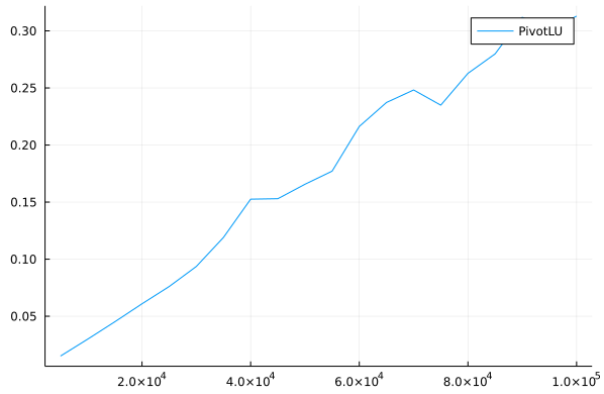
(a) Gauss



(b) Gauss z wyborem

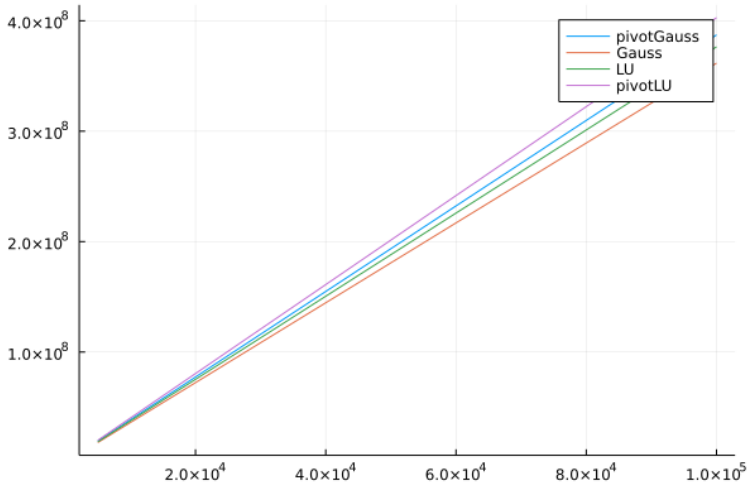


(c) LU



(d) LU z wyborem

Rysunek 4: Analiza czasu działania



Rysunek 5: Porównanie zużycia pamięci

5.2 Interpretacja

Zebrane dane potwierdzają przeprowadzoną wcześniej analizę złożoności. Algorytmy wykonują się w złożoności $O(n)$ dla $l = const$. Wersje bez częściowego wyboru działały szybciej, ale dla niektórych danych zwracały znaczący błąd. Dzieje się tak z powodu małych danych na przekątnej macierzy. Dla tej samej wersji wyboru, rozkład LU i eliminacja Gaussa, działają w podobnym czasie i mają zbliżone wyniki. Błąd obliczeń nie zależy od wielkości macierzy.

6 Wnioski

Mierząc się z problemem można wykorzystać jego specyfikę, aby znacząco przyśpieszyć działanie programów. Znaną metodykę da się zmodyfikować aby lepiej pasowała do zadania. W czasie optymalizacji należy dokładnie wiedzieć na jakich strukturach danych się pracuje. Użycie SpraseMatrix bez analizy jego działania skutkowałoby znacznie zwiększoną złożonością obliczeniową, dopiero dostosowanie tej struktury pod problem pozwoliło uzyskać zadowalające wyniki. Rozkład LU przy zbliżonym czasie i złożoności pamięciowej daje możliwość zamiany wektora prawych stron co jest pożądane. Problem małych współczynników macierzy na przekątnej ma znaczący wpływ na poprawność danych i należy używać wersji algorytmów z częściowym wyborem.