Krzysztof Dobrucki
Numer albumu 268507
Informatyka Algorytmiczna
Semestr Zimowy 2023/2024
Politechnika Wrocławska
08.01.2024r.

Lista 5

Obliczenia Naukowe

---

1. Ogólny opis problemu

Potrzebujemy rozwiązać układ równań zadany jako $Ax = b$, gdzie $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, $n > 3$. Macierz A jest szczególnej postaci:

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

gdzie $v = \frac{n}{l}$, $A_k \in \mathbb{R}^{l \times l}$ są macierzami gęstymi, $0 \in \mathbb{R}^{l \times l}$ są macierzami zerowymi, $B_k \in \mathbb{R}^{l \times l}$ mają postać:

$$B_k = \begin{bmatrix} 0 & 0 & \dots & 0 & b_1^{(k)} \\ 0 & 0 & \dots & 0 & b_2^{(k)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & b_{l-1}^{(k)} \\ 0 & 0 & \dots & 0 & b_l^{(k)} \end{bmatrix}$$

natomiast $C_k \in \mathbb{R}^{l \times l}$ mają postać:

$$C_k = \begin{bmatrix} c_1^{(k)} & 0 & 0 & \dots & 0 \\ 0 & c_2^{(k)} & 0 & \dots & 0 \\ 0 & 0 & c_3^{(k)} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & c_l^{(k)} \end{bmatrix}$$

Symbole n i l używane dalej w opisach algorytmów oznaczać będą, jak powyżej, kolejno rozmiar macierzy i rozmiar pojedynczego bloku. Celem listy jest wykorzystanie szczególnej struktury macierzy A do optymalizacji standardowych algorytmów rozwiązywania układów równań liniowych pod kątem złożoności obliczeniowej i pamięciowej.

---

2. Uwagi Techniczne

Załączone pliki:

- blocksys.jl - główny moduł programu
- utils.jl - funkcje odpowiedzialne za obsługę plików
- sparsematrix.jl - struktura optymalnie przechowująca macierz rzadką
- analysis.jl - analiza wydajności algorytmów z pliku *blocksys.jl*
- tests_auto.jl - testy automatyczne
- tests_manual.jl - prosty program do testów manualnych
- matrixgen.jl - moduł do generowania macierzy autorstwa prof. dr hab. Pawła Zielińskiego

Wykorzystany moduł:

W języku Julia **SparseArrays** to moduł (module) zawierający funkcje i struktury danych związane z obsługą macierzy rzadkich. Macierze rzadkie są szczególnym rodzajem macierzy, w którym większość elementów to zera. W przypadku tradycyjnych macierzy, przechowywanie dużej liczby zer może być marnotrawstwem pamięci, dlatego macierze rzadkie pozwalają zaoszczędzić miejsce, przechowując jedynie elementy różne od zera wraz z ich indeksami.

Moduł SparseArrays w języku Julia umożliwia manipulację i operacje na tego rodzaju macierzach, co jest szczególnie ważne w kontekście obliczeń naukowych i innych obszarów, gdzie efektywne zarządzanie pamięcią jest istotne. Używam go w mojej klasie **SparseMatrix**, aby poprawić złożoności moich implementacji algorytmów:

```
mutable struct SparseMatrix
    matrix::SparseMatrixCSC{Float64, Int}
    size::Int
    block_size::Int
    blocks_number::Int
    operation_count::Int
end
```

gdzie: matrix - reprezentacja rzadkiej macierzy, size - wielkość macierzy (n), block_size - wielkośc bloku, blocks_number - liczba bloków, operation_count - zmienna do zlicznia liczby operacji na macierzy.

---

3. Metoda eliminacji Gaussa

Klasyczna metoda eliminacji Gaussa

Opis

Najbardziej podstawowy spośród implementowanych algorytmów składa się z dwóch głównych etapów. Pierwszy z nich, zwany fazą rzeczywistej eliminacji, bazuje na obserwacji, że układ równań uzyskany w wyniku podstawowych operacji na wierszach macierzy jest równoważny pierwotnemu układowi. Metoda ta polega na odejmowaniu odpowiednich wielokrotności wierszy od tych, które się znajdują poniżej, aby wyzerować wszystkie wartości pod przekątną. Dokładniej, dla macierzy rozważanej jako $[A|b]$ (uwzględniając wektor prawych stron), w i-tym kroku wykonujemy:

$$[\, a_{j,1} \quad a_{j,2} \quad \dots \quad a_{j,l} \quad b_j \,] = [\, a_{j,1} \quad a_{j,2} \quad \dots \quad a_{j,l} \quad b_j \,] - \\ + \frac{a_{j,i}}{a_{i,i}} [\, a_{i,1} \quad a_{i,2} \quad \dots \quad a_{i,l} \quad b_i \,]$$

dla każdego $j > i$. Warto zaznaczyć, że ta metoda może zawodzić w przypadku, gdy wartość na przekątnej jest bliska zeru, ponieważ występuje ona w mianowniku, przez który mnożony jest odejmowany wiersz, co może prowadzić do błędów numerycznych. Algorytm z częściowym wyborem elementu głównego, opisany później, został wprowadzony w celu rozwiązania tego problemu.

Warto podkreślić, że po i-tym kroku wszystkie wartości pod przekątną w i-tej kolumnie stają się zerami. Po wykonaniu n - 1 kroków otrzymujemy macierz górnotrójkątną, co znacznie ułatwia proces rozwiązania.

Drugi etap algorytmu obejmuje rozwiązanie przekształconego układu. Możemy to zrobić, przechodząc od n do 1 i korzystając z poniższych wzorów:

$$x_n = \frac{b_n}{a_{nn}}$$

oraz

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

Złożoność obliczeniowa

Klasyczna wersja procedury eliminacji ma złożoność $O(n^3)$, ponieważ w każdym z k kroków (dla $k = 1, 2, \ldots, n-1$) musimy odjąć k-ty wiersz od (n - k) wierszy poniżej niego. Każde takie odejmowanie obejmuje aktualizację każdej z $n+1$ wartości w danym wierszu. Warto jednak zauważyć, że w każdej kolumnie poniżej diagonali jest maksymalnie l niezerowych wartości, które nie wymagają dalszej eliminacji, ponieważ ich wyzerowanie zostało już osiągnięte.

Przy zastosowaniu specjalnej struktury macierzy, gdzie w każdym kroku możemy odejmować jedynie od l wierszy, złożoność procesu eliminacji zostaje zredukowana do $O(l^2 \cdot (n-1)) = O(n)$. Warto zauważyć, że faza rozwiązywania układu z macierzą trójką również zostaje zoptymalizowana do $O(n)$, ponieważ w każdym wierszu mamy maksymalnie l niezerowych wartości, co pozwala na ograniczenie sumowania do najwyżej $l-1$ składników.

W związku z tym cały proces eliminacji, obejmujący obie fazy, osiąga zoptymalizowaną złożoność $O(n)$, co stanowi istotną poprawę w porównaniu do standardowej wersji, gdzie złożoność wynosi $O(n^3)$. Optymalizacja ta została osiągnięta dzięki wykorzystaniu szczególnej struktury macierzy.

Algorytm

Metoda eliminacji Gaussa

Input:

A – macierz współczynników

b – wektor prawych stron

n – rozmiar macierzy

l – rozmiar bloku

Output:

x – wektor rozwiązań

Code:

```
1  | for k from 1 to n − 1 do
2  |   for i from k + 1 to A.{last\_row}(k) do
3  |     m ← a_ik / a_kk
4  |     a_ik ← 0
5  |     for j from k + 1 to A.{last\_column}(k) do
6  |       a_ij ← a_ij − m · a_kj
7  |       b_i ← b_i − m · b_k
8  | x ← b
9  | x_n ← b_n / a_nn
10 | for i from n − 1 down to 1 do
```

```
11 |     for j from i + 1 to A.{last\_column}(i) do
12 |         x_i ← x_i − a_ij · x_j
13 |     x_i ← x_i / a_ii
14 |   return x
```

Metoda eliminacji Gaussa z częściowym wyborem elementu głównego

Opis

Dodanie częściowego wyboru elementu głównego do metody eliminacji Gaussa stanowi korzystne rozszerzenie, szczególnie w przypadku, gdy wartości na przekątnej macierzy są bardzo małe, co może prowadzić do błędów numerycznych w obliczeniach. Element główny w kontekście tej metody to wartość, która jest używana do zerowania pozostałych elementów w tej samej kolumnie. W standardowej wersji algorytmu jest to zawsze wartość a_{kk}.

Częściowy wybór polega na znalezieniu wiersza p, dla którego $|a_{pk}| = \max_{k \neq i \leq n} |a_{ik}|$, a następnie zamianie miejscami w macierzy $[A|b]$ wierszy p i k. Procedura ta pozwala na wybranie "lidera" dla każdej kolumny. W praktyce wiersze nie są fizycznie zamieniane, ale stosuje się wektor permutacji P, gdzie $P[k]$ to wiersz, w którym znajduje się element główny dla kolumny k. Jeśli procedura częściowego wyboru w k-tym kroku wybiera p-ty wiersz, to zamienia się miejscami $P[k]$ i $P[p]$.

Efekt fazy eliminacji jest identyczny jak w podstawowej wersji - uzyskujemy układ równań z macierzą górnotrójkątną, który jest następnie rozwiązywany w podobny sposób.

Złożoność obliczeniowa

Należy zauważyć, że procedura wyznaczania elementu głównego, wykonywana raz w każdym z $(n-1)$ kroków eliminacji, nadal wymaga czasu $O(n)$, co oznacza, że złożoność niezoptymalizowanego algorytmu utrzymuje się na poziomie $O(n^3)$. W fazie wyznaczania rozwiązania nie zachodzą znaczące zmiany, a złożoność również pozostaje niezmieniona.

Stosowane techniki optymalizacyjne są analogiczne do opisanych wcześniej. Poniżej diagonali macierzy w każdym etapie mamy co najwyżej l niezerowych wartości, co pozwala na ograniczenie obszaru poszukiwań elementu głównego do stałej liczby iteracji. Dodatkowo, ponieważ wiersz zawierający element główny w k-tym kroku "staje się" k-tym wierszem, może on zawierać maksymalnie l niezerowych wartości. Oznacza to, że będziemy odejmować jedynie od tych wartości. Ograniczenie kolumn, od których odejmujemy, musi jednak zachodzić łagodniej niż w standardowej wersji algorytmu. Wynika to z faktu, że w k-tym kroku element główny może znajdować się maksymalnie w $(k+l)$-tym wierszu, co wymaga uwzględnienia wartości niezerowych w danym wierszu aż do $(k+2l)$-tej kolumny. Niemniej jednak liczba iteracji pozostaje stała. Powyższe rozważania pozwalają ograniczyć złożoność fazy eliminacji do $O(n)$.

Podczas fazy wyznaczania rozwiązania również trzeba stosować "łagodniejsze" ograniczenie do iteracji, które obejmuje przedział od $(k+1)$ do $(k+2l)$, co wprowadza złożoność $O(n)$.

W wyniku tych optymalizacji udało się zredukować złożoność tej zoptymalizowanej wersji algorytmu rozwiązywania układu równań do $O(n)$.

Algorytm

Metoda eliminacji Gaussa z częściowym wyborem elementu głównego

Input:

A – macierz współczynników

b – wektor prawych stron

n – rozmiar macierzy

l – rozmiar bloku

Output:

x – wektor rozwiązań

Code:

```
1  | P ← [1, 2, ..., n]
2  | for k from 1 to n − 1 do
3  |     j ← i, gdzie |a_{P[i],k}| ← max_{k≠i} |a_{P[i],k}|
4  |     P[k] ↔ P[j]
5  | for i from k + 1 to A.{last\_row}(k) do
6  |     m ← a_{P[i],k} / a_{P[k],k}
7  |     a_{P[i],k} ← 0
8  |     for j from k + 1 to A.{last\_column}(k + l) do
9  |         a_{P[i],j} ← a_{P[i],j} − m · a_{P[k],j}
10 |     b_{P[i]} ← b_{P[i]} − m · b_{P[k]}
11 | x ← b
12 | x_n ← b_{P[n]} / a_{P[n],n}
13 | for i from n − 1 down to 1 do
14 |     for j from i + 1 to A.{last\_column}(i + l) do
15 |         x_{P[i]} ← x_{P[i]} − a_{P[i],j} · x_j
16 |     x_{P[i]} ← x_{P[i]} / a_{P[i],i}
17 | return x
```

---

4. Rozkład LU

Rozkładem LU nazywamy taki podział $A = LU$, gdzie L jest macierzą dolnotrójkątną, a U – górnotrójkątną. Układ równań $Ax = LUx = b$ możemy wówczas rozwiązać jako dwa proste układy z macierzami trójkątnymi:

$$Lz = b$$

$$Ux = z$$

Ta dwustopniowość pozwala nam również na wielokrotne rozwiązywanie układów dla różnych wektorów prawych stron b, wyznaczając sam rozkład LU tylko raz.

Klasyczna metoda rozkładu LU

Opis

Algorytm wykorzystuje obserwację, że metoda eliminacji Gaussa efektywnie generuje macierz U, a po pewnych zmianach także macierz L. Oznaczając przez $L^{(k)}$ macierz przekształcenia A w kroku k, mamy:

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

Więc w k-tym kroku:

$$L^{(k)} \cdot \begin{bmatrix} w_1^{(k)} \\ w_2^{(k)} \\ \vdots \\ w_n^{(k)} \end{bmatrix} = \begin{bmatrix} w_1^{(k)} \\ w_2^{(k)} \\ \vdots \\ w_k^{(k)} \\ w_{k+1}^{(k)} - m_k^{(k+1)} w_k^{(k)} \\ \vdots \\ w_n^{(k)} - m_k^{(n)} w_k^{(k)} \end{bmatrix}$$

gdzie $w_i^{(k)}$ jest i-tym wierszem macierzy A w k-tym kroku eliminacji, a $m_k^{(i)} = \frac{a_k^{(i)}}{a_k^{(k)}}$.

Wtedy:

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

$$\downarrow\downarrow\downarrow$$

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

Złożenie wszystkich powyższych macierzy to:

$$L^{(n-1)^{-1}} \dots L^{(2)^{-1}} L^{(1)^{-1}} = L = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ m_(2,1) & 1 & 0 & \dots & 0 \\ m_(3,1) & m_(3,2) & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ m_(n,1) & m_(n,2) & \dots & m_(n,n-1) & 1 \end{bmatrix}$$

i w praktyce to macierzą dolnotrójkątną.

Opierając się na tym spostrzeżeniu, wprowadzimy niewielkie modyfikacje w porównaniu do standardowego algorytmu eliminacji Gaussa:

- Zamiast wyzerowywać wartości pod diagonalą, zapisywać będziemy w tych miejscach mnożniki m, które posłużyły do ich eliminacji. Dzięki temu zachowamy pełny rozkład LU w jednej macierzy, gdyż L posiada jedynki na diagonali, których nie jesteśmy zobowiązani przechowywać fizycznie.
- Rozważać będziemy jedynie macierz A zamiast wierszy złożonej macierzy $[A|b]$, gdyż wektor prawych stron b jest nam potrzebny wyłącznie w etapie rozwiązania układu, gdzie również ulega modyfikacji.

Faza wyznaczania wektora x składa się z dwóch pętli, rozwiązujących kolejno układy równań $Lz = b$ oraz $Ux = z$. Obydwie są analogiczne do opisanej w przypadku podstawowej eliminacji Gaussa.

Złożoność obliczeniowa

Ze względu na minimalne różnice między wyznaczaniem rozkładu LU a fazą eliminacji w standardowym algorytmie (takie same struktury i zakresy pętli), w niezoptymalizowanej wersji oba te etapy mają identyczną złożoność $O(n^3)$. Dzięki zastosowaniu tych samych optymalizacji możemy jednak zredukować ją do $O(n)$.

Zauważając, że macierz L ma na diagonali jedynki, możemy stwierdzić, że każdy element z_i zawiera składnik b_i. Aby zaoszczędzić pamięć, możemy nadpisywać wektor b z rozwiązaniami pierwszego układu. Ponadto macierz przechowująca rozkład LU ma tę samą strukturę co macierz A (wynika to z tych samych obserwacji, co optymalizacja pierwszego etapu), co umożliwia uproszczenie rozwiązania obu układów, ograniczając ich złożoność z $O(n^2)$ do $O(n)$.

Algorytm

Wyznaczanie rozkładu LU

Input:

A – macierz współczynników

n – rozmiar macierzy

l – rozmiar bloku

Output:

LU – macierz zawierająca rozkład

Code:

```
1 | for k from 1 to n − 1 do
2 |   for i from k + 1 to A.{last\_row}(k) do
3 |     m ← a_ik / a_kk
4 |     a_ik ← m
5 |     for j from k + 1 to A.{last\_column}(k) do
6 |       a_ij ← a_ij − m · a_kj
7 | return A
```

Wyznaczanie rozwiązań z rozkładem LU

Input:

LU – macierz zawierająca rozkład

b – wektor prawych stron

n – rozmiar macierzy

l – rozmiar bloku

Output:

x – wektor rozwiązań

Code:

```
1 | for k from 1 to n − 1 do
2 |   for i from k + 1 to LU.{last\_row}(k) do
3 |     b_i ← b_i − LU_{i,k} · b_k
4 | for i from n − 1 down to 1 do
5 |   for j from i + 1 to LU.{last\_column}(i) do
6 |     x_i ← x_i − LU_{i,j} · x_j
7 |   x_i ← x_i / LU_{i,i}
8 | return x
```

Rozkład LU z częściowym wyborem elementu głównego

Opis

Analogicznie do standardowej eliminacji Gaussa, wariant z częściowym wyborem elementu głównego stanowi odpowiedź na problemy numeryczne związane z bardzo małymi wartościami na przekątnej macierzy A. Korzystając z poprzednich podpunktów, wprowadzamy subtelne zmiany w algorytmie eliminacji z wyborem, zapisując zera przy użyciu czynników m i pomijając operacje na wektorze b. Faza wyznaczania rozkładu musi również zwracać wektor permutacji P, który jest niezbędny do obliczeń rozwiązań.

Złożoność obliczeniowa

Podobnie jak w wersji bez wyboru elementu głównego, standardowy algorytm eliminacji z częściowym wyborem nie wprowadza żadnych zmian w zakresach iteracji. Mimo że niezoptymalizowana wersja ma złożoność $O(n^3)$, zastosowane optymalizacje skutkują zredukowaniem jej do $O(n)$. Proces wyznaczania rozwiązań zachowuje koncepcję znaną z standardowego rozkładu LU, co przekłada się na niezoptymalizowaną złożoność rzędu $O(n^2)$. Jednakże, w przypadku optymalizacji, ze względu na obecność wektora permutacji, konieczne jest zastosowanie "łagodnych" ograniczeń na liczbę iteracji, więc osiągnięto złożoność rzędu $O(n)$.

Algorytm

Wyznaczanie rozkładu LU z częściowym wyborem

Input:

A – macierz współczynników

n – rozmiar macierzy

l – rozmiar bloku

Output:

LU – macierz zawierająca rozkład

P – wektor permutacji

Code:

$P \leftarrow [1, 2, \ldots, n]$
for k **from** 1 **to** $n - 1$ **do**
  $j \leftarrow i$, że $|a_{P[i],k}| \leftarrow \max_{k \not< i \leq A.\{\text{last}\_\text{row}\}(k)} |a_{P[i],k}|$
  $P[k] \leftrightarrow P[j]$
for i **from** $k + 1$ **to** $A.\{\text{last}\_\text{row}\}(k)$ **do**
  $m \leftarrow \frac{a_{P[i],k}}{a_{P[k],k}}$

```
7  |      a_{P[i],k} ← m
8  |    for j from k + 1 to A.{last\_column}(k + l) do
9  |      a_{P[i],j} ← a_{P[i],j} − m · a_{P[k],j}
10 |    return A, P
```

Wyznaczanie rozwiązań z rozkładem LU z częściowym wyborem

Input:

LU – macierz zawierająca rozkład

P – wektor permutacji

b – wektor prawych stron

n – rozmiar macierzy

l – rozmiar bloku

Output:

x – wektor rozwiązań

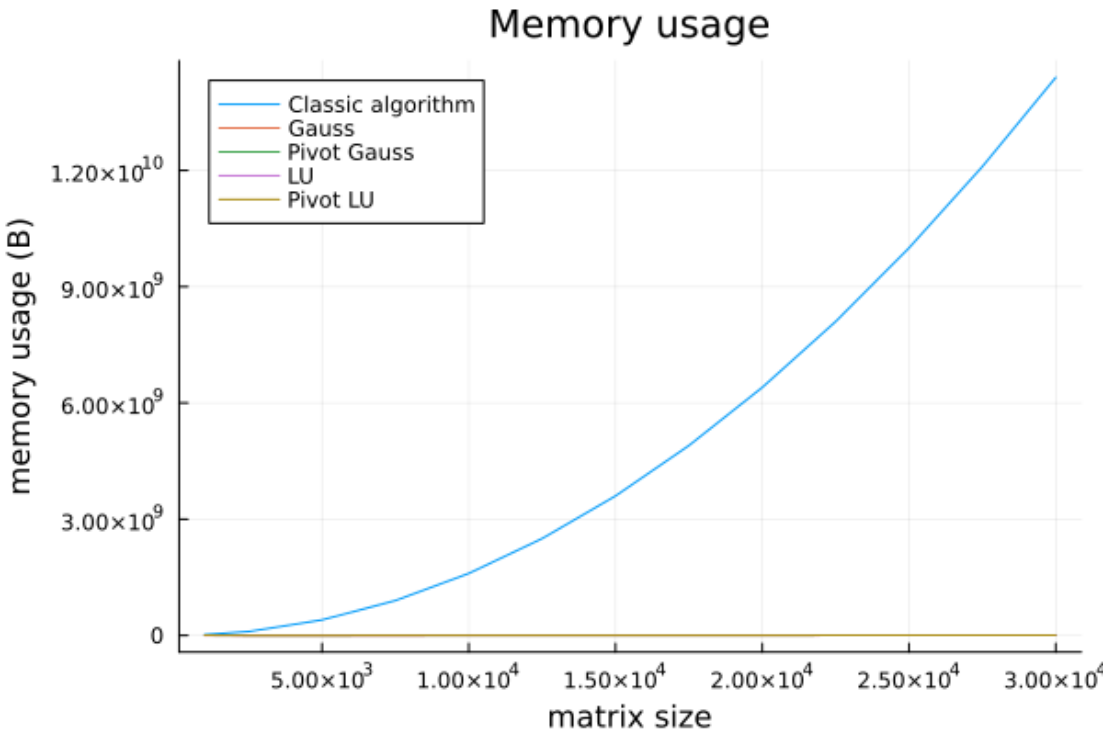
Code:

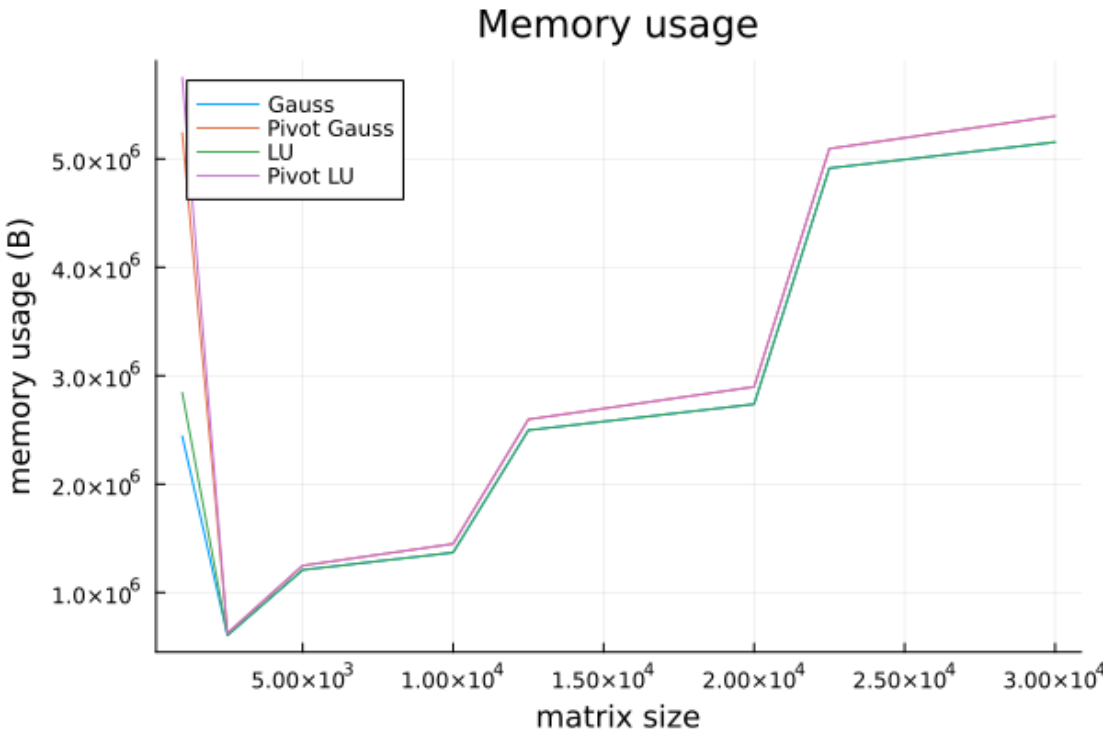
```
1  |  for i from 2 to n do
2  |    for j from LU.{first\_column}(P[i]) to i − 1 do
3  |      bP[i] ← bP[i] − LUP[i, k] · bP[k]
4  |  for i from n − 1 down to 1 do
5  |    x_i ← bP[i]
6  |    for j from i + 1 to LU.{last\_column}(i + l) do
7  |      x_i ← x_i − LUP[i, j] · x_j
8  |    x_i ← x_i / LUP[i,i]
9  |  return x
```

---

5. Eksperymentalne sprawdzanie złożoności

Wykorzystując informację, że macierze opisane na liście posiadają znaczną ilość zerowych elementów, kompleksowość pamięciowa zastosowanych algorytmów została skutecznie ograniczona dzięki użyciu macierzy rzadkich z modułu SparseArrays. To staje się istotne zwłaszcza dla dużych wartości n. Algorytmy, które zostały wdrożone, poddane zostaną porównaniu z tzw. "metodą tradycyjną" oznaczoną jako $A\backslash b$, gdzie macierz A jest przechowywana w postaci gęstej. Wykresy porównawcze złożoności pamięciowej zostały przedstawione na rysunku 1.

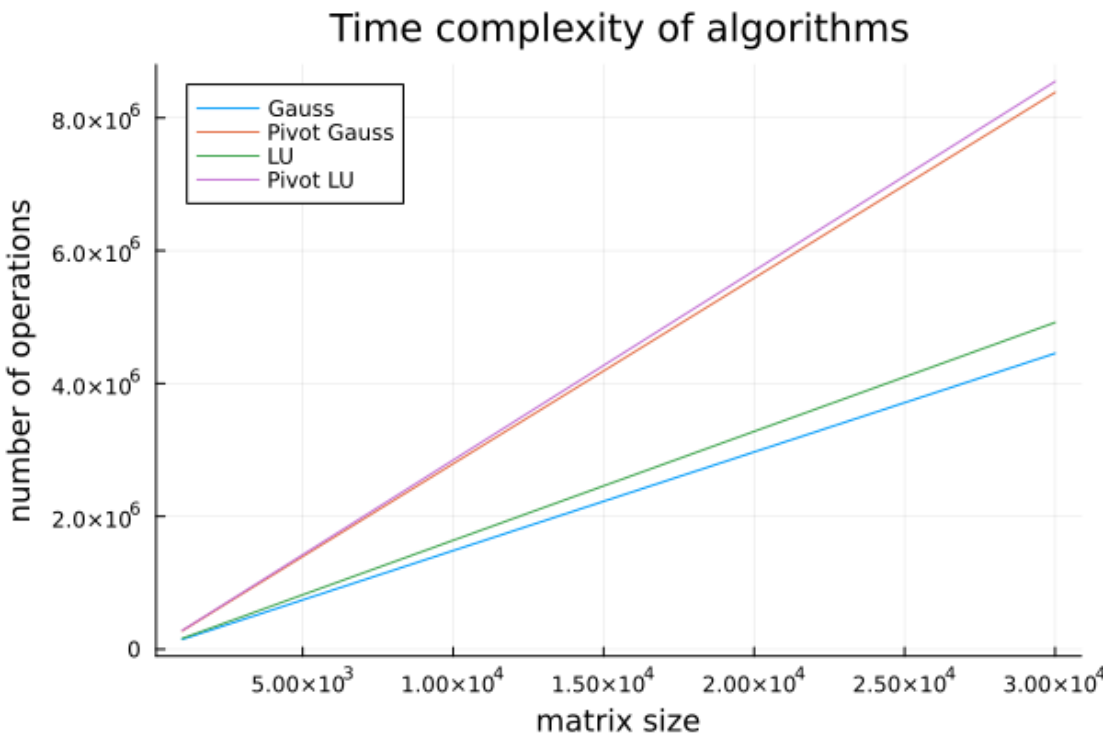




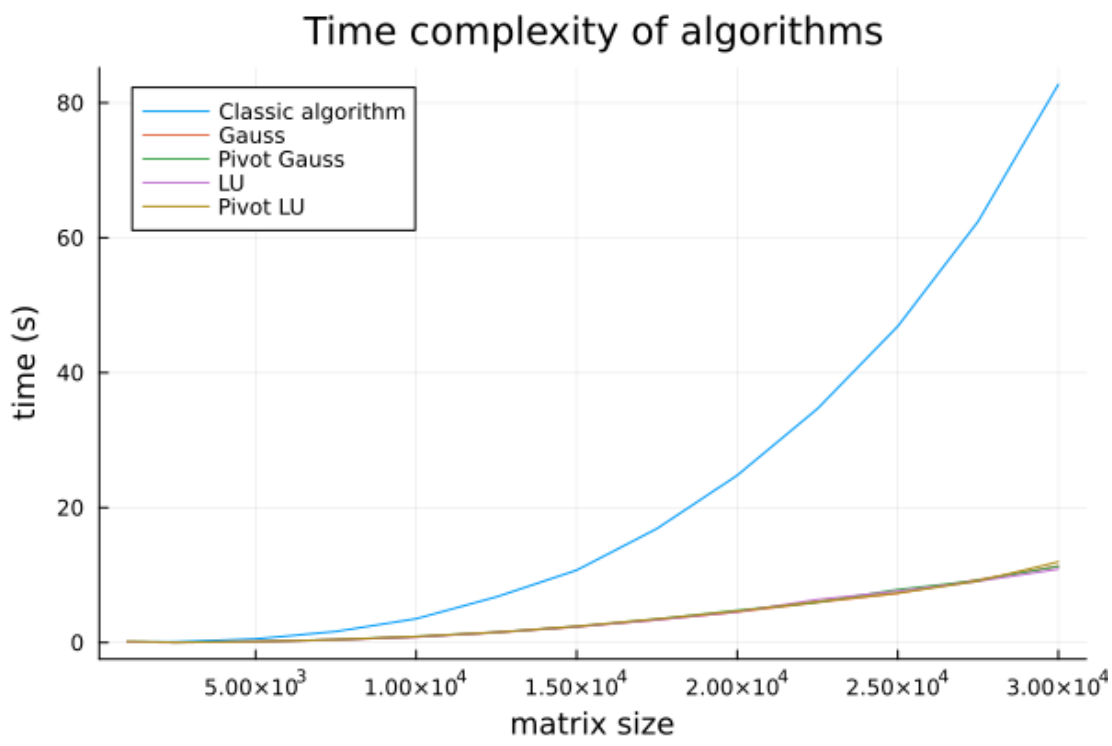
Rysunek 1: Zmierzone wartości zużycia pamięci w bajtach, zależne od rozmiaru macierzy dla algorytmów, uzyskane zostały przy użyciu makra `@timed`. Jak można zauważyć, korzystanie z macierzy rzadkich istotnie ogranicza zapotrzebowanie na pamięć. Różnice między wariantami z wyborem i bez wynikają z konieczności przechowywania wektora P. Dodatkowo widać pewne zawirowania na wykresie po prawej, prawdopodobnie mają one podłoże w sposobie implementacji - teoretyczny wykres powinien być bardziej liniowy.

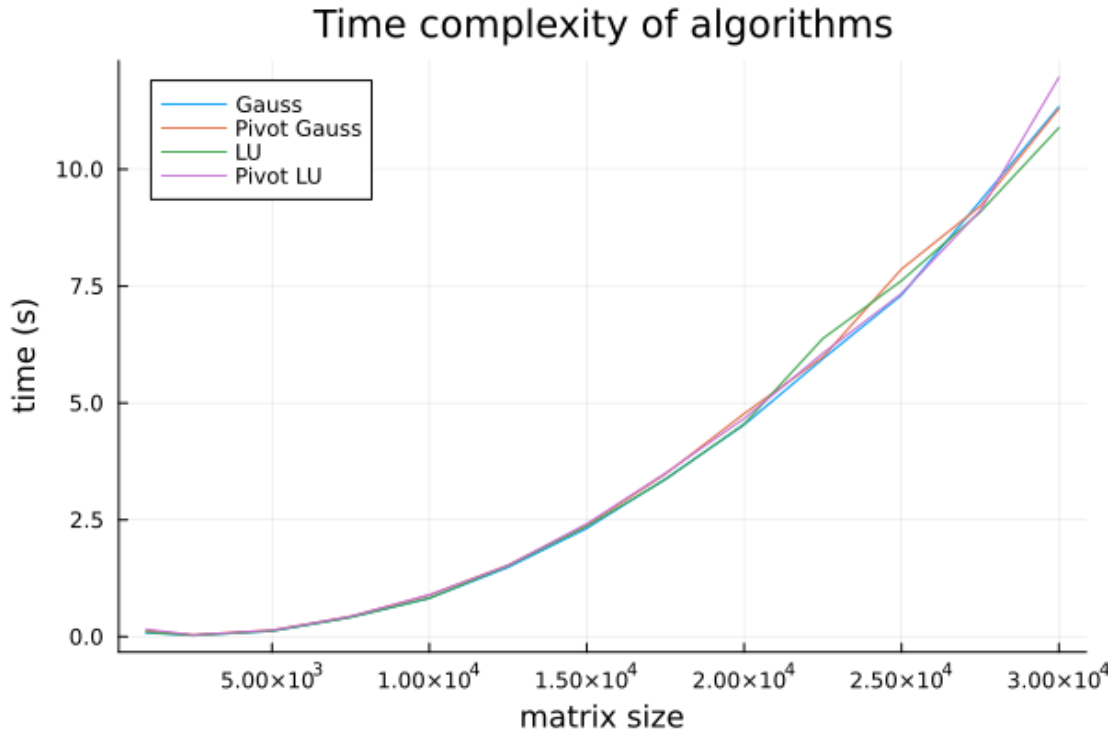
Złożoność obliczeniowa algorytmów została oceniona dwoma sposobami: z wykorzystaniem makra `@timed` oraz poprzez liczenie odwołań do elementów macierzy A. W analizach przyjęto założenie, że dostęp do elementów macierzy odbywa się w stałym czasie. Chociaż w rzeczywistości tak nie jest, co widać na wykresach z rysunku 3,

drugi sposób z rysunku 2 pozwala zignorować ten koszt, co w zasadzie potwierdza trafność naszych analiz.



Rysunek 2: Przedstawia pomiary liczby odwołań do elementów macierzy w zależności od jej rozmiaru dla zaimplementowanych algorytmów. Zgodnie z oczekiwaniami, wszystkie osiągają złożoność $O(n)$. Warianty z wyborem mają większą złożoność niż te bez, co jest efektem mniejszych ograniczeń na zakresy iteracji. Niemniej jednak pozwalają one efektywnie pracować z macierzami, gdzie na przekątnej występują niewielkie wartości. Warianty LU mają większą złożoność niż ich odpowiedniki, ponieważ w drugiej fazie rozwiązują dwa układy równań. Z drugiej strony, ich pierwsza faza może być przeprowadzona tylko raz i wielokrotnie używana dla różnych wektorów b.





Rysunek 3: Przedstawia pomiary czasu pracy w sekundach w zależności od rozmiaru macierzy dla zaimplementowanych algorytmów, uzyskane za pomocą makra `@timed`. Ograniczenie zakresów iteracji do niezerowych kolumn i wierszy przynosi znaczne oszczędności czasowe w porównaniu do naiwnego podejścia. Zaimplementowane algorytmy osiągają bardzo podobne wyniki do siebie.

---

6. Testowanie napisanego kodu

- Testy automatyczne napisane przy pomocy makr `@testset` i `@test` znajdują się w pliku *tests_auto.jl*
- Testy manualne można wykonać za pomocą pliku *tests_manual.jl* (instrukcja znajduje się poniżej)

Na podstawie przeprowadzonych testów można stwierdzić, że funkcje działają prawidłowo ze złożonością $O(n)$ (przy założeniu czytania danych w czasie stałym), a błąd numeryczny jest średnio mniejszy niż 10^{-14}.

---

7. Wnioski

Z analizy powyższych obserwacji można wysunąć, że modyfikacja standardowych algorytmów w celu dostosowania ich do specyficznej struktury macierzy zadania jest znacząco opłacalna. Algorytmy w swej podstawowej wersji posiadają złożoność sześcienną $O(n^3)$, jednak po wprowadzeniu odpowiednich modyfikacji uzyskano warianty o złożoności nawet liniowej $O(n)$ (przy zakładanym stałym dostępie do $SparseArray$), co stanowi istotne ograniczenie zużywanych zasobów.

Dodatkowo, zauważalna jest rozbieżność między wykresami liczby operacji a wykresami czasu, co sugeruje błędność założenia o stałym dostępie do elementów tablicy $SparseArray$. Mimo że wykresy liczby operacji wskazują na liniową złożoność, wykresy czasu sugerują złożoność kwadratową. W związku z tym można wnioskować, że założenie to nie jest prawidłowe.