Obliczenia naukowe - lab 5

Jakub Musiał 268442

Styczeń 2024

Oznaczenie: $n \in \mathbb{N} \implies [n] = \{1, ..., n\}$

Opis problemu

Zadaniem jest przygotowanie biblioteki (modułu w języku `Julia`) o nazwie `blocksys`, która umożliwi efektywne rozwiązywanie układów równań liniowych reprezentowanych macierzowo:

$$Ax = b$$

Gdzie $A \in \mathbb{R}^{n \times n}$ - macierz współczynników, $b \in \mathbb{R}^n$ - wektor prawych stron oraz $x \in \mathbb{R}^n$ - szukane rozwiązanie.

Dodatkowo wiemy, że A jest macierzą blokową o następującej strukturze:

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

Gdzie:

- $0 \in \mathbb{R}^{l \times l}$ - kwadratowa macierz zerowa
- $v = \frac{n}{l}$ - wymiar macierzy blokowej A dla bloków wymiaru $l \times l$ (zakładając, że $l|n$)
- $k \in [v] \implies A_k \in \mathbb{R}^{l \times l}$ - kwadratowa macierz gęsta
- $k \in \{2, ..., v\} \implies B_k \in \mathbb{R}^{l \times l} \wedge B_k = \begin{bmatrix} 0 & \cdots & 0 & b_1^k \\ 0 & \cdots & 0 & b_2^l \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & b_l^k \end{bmatrix}$
- $k \in [v-1] \implies C_k \in \mathbb{R}^{l \times l} \wedge C_k = \begin{bmatrix} c_1^k & 0 & 0 & \cdots & 0 \\ 0 & c_2^k & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c_{l-1}^k & 0 \\ 0 & \cdots & 0 & 0 & c_l^k \end{bmatrix}$

Znając strukturę macierzy A można zdefiniować efektywne pamięciowo struktury przechowujące taką macierz. Taką strukturę można zdefiniować na wiele sposobów, jednak w swoich eksperymentach korzystałem z dwóch implementacji:

Implementacja ogólna:

```
struct DictMatrix
    n::Int64
    l::Int64
    M::Dict{Tuple{Int64, Int64, Float64}}
end
```

Elementy niezerowe macierzy są przechowywane w słowniku, gdzie kluczem jest para indeksów (i, j) takich, że i - indeks wiersza oraz j - indeks kolumny.
Pobranie elementu z indeksu (i, j) zwraca wartość z tego indeksu, jeśli jest on kluczem w słowniku albo 0 w przeciwnym przypadku.
Przypisanie wartości elementowi na indeksie (i, j) przypisuje zadaną wartość do klucza (i, j) w słowniku, jeśli wartość ta nie jest zerem. W przeciwnym przypadku klucz (i, j) jest usuwany ze słownika.

Tak zdefiniowana struktura przechowuje wyłącznie niezerowe elementy, zatem na podstawie struktury macierzy A możemy zauważyć (wiedząc, że l jest stałe), że złożoność pamięciowa takiej struktury jest liniowa:

$$M_A : O(v \cdot l^2) = O(\frac{n}{l} \cdot l^2) = O(n \cdot l) = O(n)$$

$$M_B = M_C : O((v-1) \cdot l) = O((\frac{n}{l} - 1) \cdot l) = O(n - l) = O(n)$$

Dodatkowo możemy zauważyć, ze koszt zamortyzowany operacji pobrania oraz ustawienia elementu ze słownika jest stały ($O(1)$), jednak przy bardzo dużych macierzach istnieje ryzyko występowania kolizji funkcji hashującej, co może pogorszyć złożoność czasową tych operacji. Niestety jest to nieuniknione dla tak zdefiniowanej struktury ogólnej, która nie zakłada nic o postaci przechowywanej macierzy.

Implementacja uwzględniająca postać macierzy A:

```
struct Matrix
    n::Int64
    l::Int64
    A::Vector{Matrix{fl}}
    B::Vector{Vector{fl}}
    C::Vector{Matrix{fl}}
end

function QCMatrix(n::int, l::int)
    v = div(n, l)
    A = [zeros(fl, l, l) for _ in 1:v]
    B = [zeros(fl, l) for _ in 2:v]
    C = [zeros(fl, l, l) for _ in 1:v-1]
    new(n, l, A, B, C)
end
```

Znając specyficzną strukturę macierzy A możemy zaimplementować bardziej dokładną strukturę opisującą taką macierz. Musimy zatem przechowtywać wyłącznie bloki $A_k : k \in [v]$, $B_k : k \in \{2, ..., v\}$ oraz $C_k : k \in [v-1]$. Przeprowadziwszy rozumowanie analogiczne do tego dla struktury ogólnej, możemy

zauważyć, że złożoność pamięciowa jest liniowa dla stałego l (przechowanie bloków C_k jako macierze jest konieczne ze względu na operacje wykonywane na macierzy w zadanych algorytmach).

$$M_A : O(v \cdot l^2) = O(\frac{n}{l} \cdot l^2) = O(n \cdot l) = O(n)$$

$$M_B : O((v-1) \cdot l) = O((\frac{n}{l} - 1) \cdot l) = O(n - l) = O(n)$$

$$M_C : O((v-1) \cdot l^2) = O((\frac{n}{l} - 1) \cdot l^2) = O(nl - l^2) = O(n)$$

Dla tak zdefiniowanej struktury operacja pobrania wartości z i-tej kolumny i j-tego rzędu polega na sprawdzeniu, do którego bloku należy podana para indeksów i zwrócenie wartości z odpowiedniego bloku lub zera, jeśli (i, j) nie znajduje się w żadnym z bloków. Analogicznie przypisanie wartości zmienia element w macierzy wyłącznie jeśli (i, j) jest w jednym z przechowywanych bloków.
Przynależność pary indeksów (i, j) do poszczególnych bloków możemy zdefiniować następująco:

Niech: $r_b = \lfloor \frac{i-1}{l} \rfloor + 1 \wedge c_b = \lfloor \frac{j-1}{l} \rfloor + 1$
Wtedy:

$$c_b = r_b \implies \text{"}(i, j) \in A_{r_b}\text{"}$$

$$c_b = r_b - 1 \implies \text{"}(i, j) \in B_{r_b - 1}\text{"}$$

$$c_b = r_b + 1 \implies \text{"}(i, j) \in C_{r_b}\text{"}$$

Możemy zauważyć, że takie operacje na macierzy wykonują się zawsze w czasie stałym, zatem możemy się spodziewać, że dla dużych macierzy wejściowych, algorytmy implementowane algorytmy będą się wykonywały szybciej niż dla struktóry ogólnej - `DictMatrix`.

Zadanie 1

Problem

Zaimplementować funkcję rozwiązującą układ równań liniowych zadanych przez wyrażenie macierzowe

$$Ax = b$$

za pomocą eliminacji Gaussa oraz eliminacji Gaussa z częściowym wyborem elementu głównego, uwzględniając specyficzną strukturę macierzy A.

Rozwiązanie

Metoda eliminacji Gaussa polega na zerowaniu elementów pod przekątną macierzy, iterując po jej kolumnach tak, że w rezultacie otrzymujemy macierz górno trójkątną.

Algorithm 1 Eliminacja Gaussa

Require: A, b, n, l

1: **for** $k \leftarrow 1$ to $n-1$ **do**
2: **for** $i \leftarrow k+1$ to n **do**
3: $l_{ik} \leftarrow \frac{A_{ik}}{A_{kk}}$
4: **for** $j \leftarrow k+1$ to n **do**
5: $A_{ij} \leftarrow A_{ij} - l_{ik} \cdot A_{kj}$
6: $b_i \leftarrow b_i - l_{ik} \cdot b_k$
7: **return** (A, b)

Widzimy jednak, że ten algorytm ma złożoność obliczeniową $O(n^3)$, co wynika z trzech zagnieżdżonych pętli, których liczba iteracji jest zależna od rozmiaru macierzy A. Wiemy jednak, że pod przekątną zadanej macierzy jest wiele zer, co pozwoli nam przyspieszyć opisany powyżej algorytm. Aby to zrobić będziemy chcieli w wewnętrznych pętlach iterować od przekątnej do ostatniego niezerowego elementu.

Rozważmy zatem pętlę z linii 2, w której iterujemy po rzędach od $(k+1)$-ego do ostatniego. Wiemy jednak, że pod przekątną w k-tej kolumnie jest dokładnie $\min\{l - (k \mod l), n-k\}$ (zakładając, że indeksujemy od 1). Wynika to z faktu, że przechodząc po przekątnej zawsze jesteśmy w bloku A_i, zatem musimy wyzerować elementy pod przekątną w podmacierzy A_i oraz wszystkie elementy z podmacierzy B_{i+1} - dokładnie l elementów w kolumnie $k = i \cdot l$, dla której $l - (i \cdot l \mod l) = l - 0 = l$. Musimy jednak uwzględnić także ostatnie bloki w macierzy, dla których $l - (k \mod l) > n$. Stąd możemy zamienić ogranicznie $i \in \{k+1, ..., n\}$ w pętli z linii 2 na $i \in \{k+1, min\{k+l-(k \mod l), n\}\}$.

Analogiczne rozumowanie możemy przeprowadzić dla pętli z linii 4. Tutaj wiemy, że dla każdego k będziemy modyfikowali wartości wyłącznie pomiędzy przekątnymi podmacierzy A_i oraz przekątnymi podmacierzy C_i. Zauważmy, że te przekątne są do siebie "równoległe" oraz w odległości l. Zatem wystarczy, że będziemy rozpatrywać tylko l elementów na prawo od przekątnej, jednak musimy ponownie uwzględnić ostatnie bloki macierzy. Stąd możemy zamienić ogranicznie pętli z linii 4 z obecnego $j \in \{k+1, ..., n\}$ na $j \in \{min\{k+l, n\}\}$.

Taka modyfikacja algorytmy eliminacji Gaussa gwarantuje nam złożoność liniową: najbardziej zewnętrzna pętla wykonuje się n razy, a pętle wewnętrzne $O(l)$ razy. Zatem, wiedząc, że l jest stałą, całkowita złożoność obliczeniowa zmodyfikowanego algorytmu to $O(n * l^2) = O(n)$.

Algorytm eliminacji Gaussa z wyborem elementu głównego różni się od standardowego algorytmu tym, że dla każdej kolumny przed wykonaniem kroku eliminacji szukamy elementu o maksymalnej wartości bezwzględnej oraz zamieniami miejscami rząd obecny z tym, w którym znajduje się znaleziony element (o ile element maksymalny nie znajduje się w aktualnym rzędzie).
Zabieg ten ma na celu zminimalizowanie błędów oblczeniowych wynikających z występowania wartości $\varepsilon \approx 0$ na przekątnej badanej macierzy. Dla takiej sytuacji w k-tym kroku eliminacji gaussa otrzymujemy:

$$l_{ik} = \frac{A_{ik}}{A_{kk}} = \frac{A_{ik}}{\varepsilon} = A_{ik} \cdot \varepsilon^{-1}$$

Zauważmy, że dla małych wartości ε ($< \epsilon_{mach}$) powyższe wyrażenie oraz późniejsze obliczenia mogą być numerycznie niepoprawne ze względu na ograniczoną precyzję arytmetyki.

Zatem wybór elementu głównego powinien zapobiec błędom obliczeniowym wynikającym z występowania wartości bliskich zeru na przekątnej macierzy.
Dodatkowo możemy zauważyć, że struktura macierzy A pozwala nam na zawężenie obszaru poszukiwań takiego elementu - ostatnim rzędem zawierającym element niezerowy w k-tej kolumnie będzie rząd $r_{max} = min\{k+l-(k \mod l), n\}$ (jak w poprzednim algorytmie). W związku z zamianą rzędów macierzy musimy również rozważyć maksymalny zakres kolumn w najbardziej wewnętrznej pętli - tutaj nie mamy stałej odległości między przekątną, a elementem z bloku C_k, zatem musimy uwzględnić wszystkie kolumny z bloku C_k. Stąd otrzymujemy zmodyfikowany algorytm eliminacji Gaussa z częściowym wyborem elementu głównego:

Algorithm 2 Eliminacja Gaussa z częściowym wyborem elementu głównego

Require: A, b, n, l
1: **for** $k \leftarrow 1$ to $n-1$ **do**
2: $\quad r_{max} \leftarrow min\{k+l-(k \mod l), n\}$
3: $\quad r_{v_{max}} \leftarrow k + argmax_{i \in \{k,\ldots,r_{max}\}}(|A_{i,k}|) - 1$
4: $\quad$ **if** $r_{v_{max}} \neq k$ **then**
5: $\quad\quad swap\_rows(A, k, r_{v_{max}})$
6: $\quad c_{max} \leftarrow min\{k+2l-(k \mod l), n\}$
7: $\quad$ **for** $i \leftarrow k+1$ to r_{max} **do**
8: $\quad\quad l_{ik} \leftarrow \frac{A_{ik}}{A_{kk}}$
9: $\quad\quad$ **for** $j \leftarrow k+1$ to n **do**
10: $\quad\quad\quad A_{ij} \leftarrow A_{ij} - l_{ik} \cdot A_{kj}$
11: $\quad\quad b_i \leftarrow b_i - l_{ik} \cdot b_k$
12: **return** (A, b)

Podobnie jak w klasycznej wersji algorytmu eliminacji Gaussa wiemy, ze najbardziej zewnętrzna pętla wykonuje się $O(n)$ razy, natomiast pętle wewnętrzne $O(l)$ razy, zatem dla stałego l nasz algorytm ma liniową złozoność czasową.

Mając algorytmy wyznaczające macierz górnotrójkątną za pomocą elimincaji Gaussa (bez wyboru oraz z wyborem elementu głównego) możemy napisać funkcję rozwiązującą układ równań liniowych $Ax = b$.

Algorithm 3 Rozwiązanie układu równań liniowych

Require: A, b, n, l, a_{pp}

1: $(A', b') \leftarrow \begin{cases} gaussian\_elimination\_pp(A, b, n, l) & : a_{pp} = true \\ gaussian\_elimination(A, b, n, l) & : else \end{cases}$

2: $x = zeros(n)$

3: **for** $k \leftarrow n$ down to 1 **do**

4: $\quad \Sigma \leftarrow 0$

5: $\quad c_{max} \leftarrow \begin{cases} min\{k + 2l - (k \mod l), n\} & : a_{pp} = true \\ min\{k + l, n\} & : else \end{cases}$

6: $\quad x_k \leftarrow (b'_k - \sum_{k+1}^{c_{max}} (A'_{ki} * x_i))/A'_{kk}$

7: **return** x

W algorytmie przechodząc od ostatniego (n-tego) rzędu macierzy górnotrójkątnej (w którym tylko jedna zmienna ma niezerowy współczynnik) do pierwszego rzędu rozwiązujemy równania liniowe, podstawiając wartości x_k uzyskane w poprzednich iteracjach. W ten sposób w każdej iteracji otrzymujemy równanie z jedną niewiadomą x:

$$ax + b = y$$

Gdzie:

- $a = A'_{kk}$
- $x = x_k$
- $b = \sum_{k+1}^{c_{max}} (A'_{ki} * x_i)$: podstawienie wartości $x_{>k}$
- $y = b_i$: i-ty element przekształconego zadanego wektora prawych stron

Rozwiąznie takiego równania możemy uzyskać po przekształceniu powyższego równiania:

$$x = \frac{y - b}{a}$$

Skąd uzyskujemy wyrażenie w 6 linii pseudokodu opisanego algorytmu.

Ostatecznie otrzymujemy algorytm o liniowej złożoności czasowej:

- Wyznaczenie macierzy górnotrójkątnej - $O(n)$
- Rozwiązanie układu równań na podstawie uzyskanej macierzy i wektora prawych stron - $O(n)$ (musimy rozwiązać n równań, dla każdego licząc sumę $O(l)$ wcześniej wyznaczonych elementów).

Zadanie 2

Problem

Zaimplementować funkcję wyznaczającą rozkład trójkątny ($A = LU$) macierzy A za pomocą metody eliminacji Gaussa oraz eliminacji Gaussa z częściowym wyborem elementu głównego, uwzględniając specyficzną strukturę zadanej macierzy.

Rozwiązanie

Rozkład trójkątny macierzy A możemy przedstawić jako iloczyn macierzy dolno i górnotrójkątnej $A = LU$, gdzie:

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & 1 \end{bmatrix} \wedge U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1,n-1} & u_{1n} \\ 0 & u_{22} & \cdots & u_{2,n-1} & u_{2n} \\ 0 & 0 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & u_{n-1,n-1} & u_{n-1,n} \\ 0 & 0 & \cdots & 0 & u_{nn} \end{bmatrix}$$

Możemy zatem rozkład LU pamiętać w jednej macierzy w realizacji naszego algorytmu:

$$LU = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1,n-1} & u_{1n} \\ l_{21} & u_{22} & \cdots & u_{2,n-1} & u_{2n} \\ l_{31} & l_{32} & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & u_{n-1,n-1} & u_{n-1,n} \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & u_{nn} \end{bmatrix}$$

Dodatkowo możemy zauważyć, że ze względu na strukturę badanych macierzy nie będziemy musieli przechowywać elementów poza blokami A_k, B_k oraz C_k, co znaczy, że obie wcześniej zaimplementowane struktury reprezentujące macierz A wystarczą by poprawnie reprezentować uzyskany rozkład trójkątny.

Rozważmy zatem algorytm wyznaczania oczekiwanego rozkładu. Wiemy, że rozkład LU jest równoznaczny eliminacji Gaussa, ponieważ macierz U możemy zdefiniować jako A^n (macierz A w n-tym kroku metody GE). Wiemy jednak, że $A^1 = A$ oraz w $k+1$ kroku GE wyznaczamy $A^{k+1} = L^k \cdot A^k$, gdzie:

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

Zatem sprowadzenie macierzy A do szukanej w algorytmie GE macierzy U możemy zpisać jako:

$$U = A^n = L^{n-1} \cdot L^{n-1} \cdot ... \cdot L^1 \cdot A$$

To pozwala nam wyznaczyć szukaną macierz dolnotrójkątną jako:

$$L = L^1_- \cdot L^2_- \cdot ... \cdot L^{n-1}_-$$

Gdzie L^k_- to macierz L^k "bez minusów" przy wartościach $l_{>k,k}$.

Wiemy zatem, że macierz U wyznaczamy jako rezultat algorytmu GE, a także, że w trakcie przebiegu algorytmu wyznaczamy wszystkie szukane wartości $l_{>k,k}$. Stąd aby uzyskać rozkład LU musimy zmodyfikować oryginalny algorytm eliminacji Gaussa tak, aby zapamiętywać wyznaczane wartości $l_{>k,k}$.
Musimy także uwzględnić strukturę zadanej macierzy (nie rozpatrujemy niezerowych elementów pod przekątną), zatem będziemy pamiętać wyłącznie niezerowe elementy macierzy L - $l_{ik} : i \in \{k+1, min\{k+l-(k \mod l), n\}\}$.
Dodatkowo przy wyznaczaniu rozkładu LU macierzy nie będziemy przyjmować wektora prawych stron b, jako że jest on zbędny dla zadanego problemu.

Algorithm 4 Rozkład LU

Require: A, n, l
1: **for** $k \leftarrow 1$ to $n-1$ **do**
2: $\quad r_{max} \leftarrow min\{k+l-(k \mod l), n\}$
3: $\quad$ **for** $i \leftarrow k+1$ to r_{max} **do**
4: $\quad\quad l_{ik} \leftarrow \frac{A_{ik}}{A_{kk}}$
5: $\quad\quad A_{ik} \leftarrow l_i k$
6: $\quad\quad c_{max} \leftarrow min\{k+l, n\}$
7: $\quad\quad$ **for** $j \leftarrow k+1$ to c_{max} **do**
8: $\quad\quad\quad A_{ij} \leftarrow A_{ij} - l_{ik} \cdot A_{kj}$
9: **return** A

Podobną modyfikację możemy wykonać dla algorytmu wyznaczania rozkładu trójkątnego z częściowym wyborem elementu głównego. Analogicznie jak w poprzednim zadaniu musimy jednak uwzględnić większe zakresy indeksów rzędów i kolumn, po których będziemy iterować wynikające z zamian rzędów po znalezieniu elementu maksymalnego.

Uwaga: poniższy pseudokod zawiera także modyfikacje konieczne do użycia algorytmu do rozwiązania układu równań $LUx = b$ z **zadania 3**, które zostaną opisane w dalszej części sprawozdania.

Algorithm 5 Rozkład LU z częściowym wyborem elementu głównego

Require: A, b, n, l
1: $\pi_b \leftarrow [n]$
2: **for** $k \leftarrow 1$ to $n-1$ **do**
3: $\quad r_{max} \leftarrow min\{k+l-(k \mod l), n\}$
4: $\quad r_{v_{max}} \leftarrow k + argmax_{i \in \{k,\ldots,r_{max}\}}(|A_{i,k}|) - 1$
5: $\quad$ **if** $r_{v_{max}} \neq k$ **then**
6: $\quad\quad swap\_rows(A, k, r_{v_{max}})$
7: $\quad\quad swap(\pi_b, k, r_{v_{max}})$
8: $\quad c_{max} \leftarrow min\{k+2l-(k \mod l), n\}$
9: $\quad$ **for** $i \leftarrow k+1$ to r_{max} **do**
10: $\quad\quad l_{ik} \leftarrow \frac{A_{ik}}{A_{kk}}$
11: $\quad\quad A_{ik} \leftarrow l_i k$
12: $\quad\quad$ **for** $j \leftarrow k+1$ to n **do**
13: $\quad\quad\quad A_{ij} \leftarrow A_{ij} - l_{ik} \cdot A_{kj}$
14: **return** (A, π_b)

W związku z tym, że wprowadzone do wcześniejszych algorytmów poprawki mają stałą złożoność czasową, oba algorytmy wyznaczania rozkładu trójkątnego mają całkowitą złożoność $O(n)$.

Zadanie 3

Problem

Na podstawie funckji z **zadania 2** zaimplementować funkcję rozwiązującą układ równań liniowych zadanych jako wyrażenie macierzowe $Ax = b$, jeśli wcześniej został wyznaczony rozkład trójkątny macierzy wejściowej - $A = LU$.

Rozwiązanie

Znając rozkład LU macierzy zadanej macirzy możemy wyznaczyć efektywny algorytm rozwiązywania układu $Ax = b$ poprzez podstawianie $y = Ux$, co sprowadza problem do rozwiązania dwóch trójkątnych układów równań:

$$Ly = b$$
$$Ux = y$$

Możemy napisać uogólniony algorytm dla rozkładu LU wyznaczanym bez wyboru oraz z wyborem elementu głównego. Aby to zrobić algorytm musi przyjmować permujację wektora prawych stron b. Dla podstawowej wersji algorytmu wyznaczania LU ta permutacja to zwyczajnie $[n]$, jako że w trakcie przebiegu GE w żaden sposób nie permutujemy wierszy macierzy wejściowej. Dla wersji z częściowym wyborem elementu głównego permutacja wyznaczana jest równoznacznie do permutacji rzędów macierzy A (wektor π_b w psedukodzie algorytmu 5).

Rozwiąznanie układu pierwszego z macierzą dolnotrójkątną sprowadza się zatem do rozwiazania kolejnych równań liniowych postaci $x + b = y \implies x = y - b$ (brak współczynnika przy x wynika ze struktury macierzy L, której przekątna składa się wyłącznie z jedynek), przechodząc od pierwszego równania do n-tego oraz podstawiając w kolejnych iteracjach $x = y_k$, $b = \sum_{c_{min}}^{k-1}(L_{ki} \cdot y_i)$ i $y = b_{\pi_b(k)}$.
Rozwiązanie układu drugiego z macierzą górnotrójkątną przeprowadzamy analogicznie - rozwiązujemy kolejne równia liniowe $ax + b = y \implies x = \frac{y-b}{a}$ od pierwszego do n-tego, podstawiając w kolejnych iteracjach $a = U_{kk}$, $x = x_k$, $b = \sum_{k+1}^{c_{max}}(U_{ki} * x_i)$ i $y = x_k$

Algorithm 6 Rozwiązanie układu równań liniowych z rozkładem LU

Require: $LU, b, n, l, a_{pp}, \pi_b$

1:
2: $y = zeros(n)$
3: **for** $k \leftarrow 1$ to n **do**
4: $\quad \Sigma \leftarrow 0$
5: $\quad c_{min} \leftarrow max\{1, (\lfloor \frac{k-1}{l} \rfloor + 1) * l\}$
6: $\quad y_k \leftarrow b_{\pi_b(k)} - \sum_{k+1}^{c_{max}}(L_{ki} * x_i)$
7:
8:
9: $x = zeros(n)$
10: **for** $k \leftarrow n$ down to 1 **do**
11: $\quad \Sigma \leftarrow 0$
12: $\quad c_{max} \leftarrow \begin{cases} min\{k + 2l - (k \mod l), n\} & : a_{pp} = true \\ min\{k + l, n\} & : else \end{cases}$
13: $\quad x_k \leftarrow (y_k - \sum_{k+1}^{c_{max}}(U_{ki} * x_i))/U_{kk}$
14: **return** x

▷ $Ly = b$

▷ $Ux = y$

Opisany algorytm ma liniową złożoność czasową, co wynika z liniowej liczby równań w układach $Ly = b$ i $Ux = y$, a także ze stałej (zależnej od l) liczby niezerowych elementów w każdym rzędzie obu układów.

Wyniki i obserwacje

Rozważmy zaimplementowane algorytmy rozwiązujące układy równań liniowych zadanych jako wyrażenie macierzowe $Ax = b$:

1. **[zadanie 1]** Sprowadzenie macierzy wejściowej do postaci górno-trójkątnej za pomocą metody eliminacji Gaussa oraz rozwiązanie układu z uzyskaną macierzą oraz przekształconym wektorem prawych stron

2. **[zadanie 1]** Sprowadzenie macierzy wejściowej do postaci górno-trójkątnej za pomocą metody eliminacji Gaussa z częściowym wyborem elementu głównego oraz rozwiązanie układu z uzyskaną macierzą oraz przekształconym wektorem prawych stron

3. **[zadanie 3]** Wyznaczenie rozkładu trójkątnego $A = LU$ zadanej macierzy za pomocą metody eliminacji Gaussa oraz rozwiązanie dwóch układów równań z wyznaczonymi macierzami trójkątnymi

4. **[zadanie 3]** Wyznaczenie rozkładu trójkątnego $A = LU$ zadanej macierzy za pomocą metody eliminacji Gaussa z częściowym wyborem elementu głównego oraz rozwiązanie dwóch układów równań z wyznaczonymi macierzami trójkątnymi

Wiemy, że metoda eliminacji Gaussa z częściowym wyborem elementu głównego ma na celu zminimalizowanie błędów numerycznych wynikających z występowania elementów bliskich zeru na przekątnej badanej macierzy. Możemy się zatem spodziewać, że wersje algorytmów z zadań 1 i 3 z cześciowym wyborem będą zwracały bardziej dokładne wyniki.

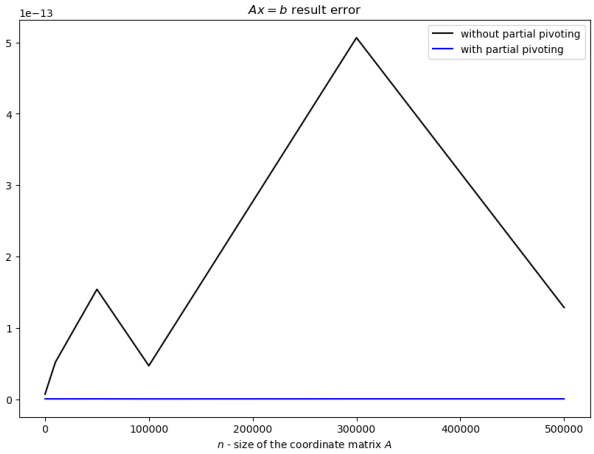
Poniższe tabele oraz wykresy przedstawiają otrzymane dla danych testowych błędy względne uzyskanego rozwiązania względem oczekiwanego: $x = (1, ..., 1)^T$. Na podstawie uzyskanych wyników możemy potwierdzić wcześniejsze założenie. Widzimy, że wersje algorytmów z częściowym wyborem elementu głównego znacząco poprawiają dokładność uzyskanych wyników.

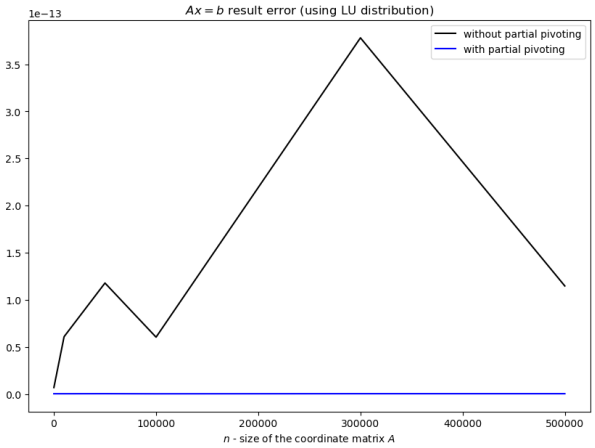
n	δ	δ_{pp}
16	$6.906622391776103e-15$	$3.744431865468923e-16$
10000	$5.1830342510443897e-14$	$3.506881556722316e-16$
50000	$1.5393951981921677e-13$	$4.057095823578191e-16$
100000	$4.681482376769146e-14$	$2.986658830105658e-16$
300000	$5.066539517131064e-13$	$4.006581448861896e-16$
500000	$1.2844098793303182e-13$	$3.983755283200343e-16$

Table 1: Błedy względne algorytmów z zadania 1 - wersja bez wyboru oraz z wyborem elementu głównego

n	δ	δ_{pp}
16	$6.9057300056503065e-15$	$3.488821756485609e-16$
10000	$6.088918182764565e-14$	$3.4879383986397025e-16$
50000	$1.1781027273694962e-13$	$4.011083832342482e-16$
100000	$6.037144640301416e-14$	$2.9483811410155386e-16$
300000	$3.77994966263058e-13$	$3.959134799391461e-16$
500000	$1.1468784565401847e-13$	$3.944640686227618e-16$

Table 2: Błedy względne algorytmów z zadania 3 (LU) - wersja bez wyboru oraz z wyborem elementu głównego





Rozważmy także czas działania badanych algorytmów. Wiemy, że wyznaczenie rozkładu trójkątnego $A = LU$ macierzy wejściowej nie jest zależne od wektora prawych stron b. To znaczy, że dla ciągu układów równań z jedną macierzą współczynników A oraz z różnymi wektorami prawych stron b możemy wyznaczyć taki rozkład raz dla wszystkich wektorów b. W związku z tym rozwiązanie wielu układów równań z tą samą macierzą współczynników powinno być efektywniejsze dla algorytmu wykorzystującego rozkład LU.

Poniższy wykres przedstawia średni czas rozwiązania jednego układu równań w ciągu układów z jedną macierzą współczynników. Na podstawie otrzymanych czasów średnich możemy potwierdzić nasze założenie.
Dodatkowo możemy zauważyć, że dla każdego z badanych algorytmów wyspecjalizowana implementacja struktury reprezentującej macierz A o opisanej na początku strukturze okazała się mieć lepszy czas wykonywania operacji pobierania i ustawiania wartości macierzy, czego można było się spodziewać, biorąc pod uwagę fakt, że ma ona stałą złożoność czasową operacji w scenariuszu "worst case", kiedy implementacja słownikowa ma zamortyzowaną złożoność operacji $O(1)$, natomiast w scenariuszu "worst case" jest to już $O(n)$, co efektywnie pogarsza całkowitą złożoność badanych algorytmów.