Algorytmy Macierzowe laboratorium 2.

Odwracanie macierzy

LU faktoryzacja

Eliminacja Gaussa

Obliczanie wyznacznika

Antoni Kucharski, Joachim Grys

5.11.2024

1 Wprowadzenie

1.1 Podział macierzy na bloki

Blokowe algorytmy macierzowe zaczynają się od podziału macierzy wejściowej A na cztery podmacierze zgodnie z zasadą:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Jeżeli A jest macierzą $n \times n$, gdzie n jest nieparzyste to należy przyjąć

- $k = \frac{n-1}{2}$
- $m = \frac{n+1}{2}$

Wówczas rozmiary poszczególnych macierzy blokowych są następujące:

- A_{11}: $k \times k$
- A_{12}: $k \times m$
- A_{21}: $m \times k$
- A_{22}: $m \times m$

2 Rekurencyjne odwracanie macierzy

2.1 Idea algorytmu

Algorytm rekurencyjnego (lub blokowego) odwracania macierzy rozpoczyna się od podziału macierzy wejściowej A na cztery podmacierze:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Następnie w celu wyznaczenia macierzy odwrotnej

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = A^{-1}$$

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$$

będziemy musieli przekształcić macierz skrajnie z lewej strony równania na macierz identycznościową. Każde przekształcenie również musi się odbyć po prawej stronie, gdzie początkowo się znajdowała macierz identycznościowa. Po wykonaniu wszystkich przekształceń uzyskamy po prawej stronie macierz A^{-1}.

Wykonując konieczne przekształcenia otrzymamy

$$\begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}^{-1}(I + A_{12}S_{22}^{-1}A_{21}A_{11}^{-1}) & -A_{11}^{-1}A_{12}S_{22}^{-1} \\ -S_{22}^{-1}A_{21}A_{11}^{-1} & S_{22}^{-1} \end{bmatrix}$$

gdzie $S_{22} = A_{22} - A_{21}A_{11}^{-1}A_{12}$ oraz wszystkie odwrotności podmacierzy (czyli de facto tylko A_{11}^{-1} i S_{22}^{-1}) powstały wskutek rekurencyjnego wywołania algorytmu.

2.2 Pseudokod

Pseudokod algorytmu wygląda mniej więcej tak:

$\texttt{invert}([a]) \rightarrow \left[\frac{1}{a}\right]$

$\texttt{invert}(A)\texttt{:}$

$A_{11}, A_{12}, A_{21}, A_{22} = \texttt{partition}(A)$

$A_{11}^{-1} = \texttt{invert}(A_{11})$

$S_{22} = A_{22} - \texttt{mult}(\texttt{mult}(A_{21}, A_{11}^{-1}), A_{12})$

$S_{22}^{-1} = \texttt{invert}(S_{22})$

$T = \texttt{mult}(\texttt{mult}(\texttt{mult}(A_{12}, S_{22}^{-1}), A_{21}), A_{11}^{-1})$

$B_{11} = \texttt{mult}(A_{11}^{-1}, \texttt{eye\_like}(A_{11}) + T)$

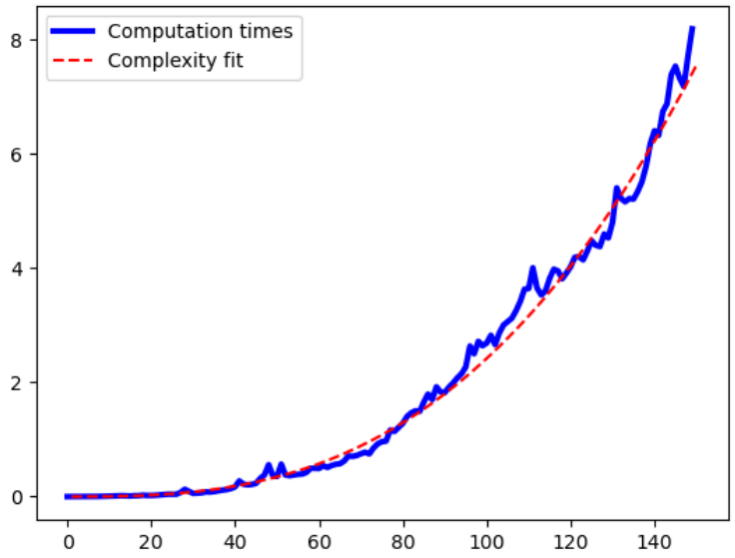
$B_{12} = -\texttt{mult}(\texttt{mult}(A_{11}^{-1}, A_{12}), S_{22}^{-1})$

$B_{21} = -\texttt{mult}(\texttt{mult}(S_{22}^{-1}, A_{21}), A_{11}^{-1})$

$B_{22} = S_{22}^{-1}$

$\texttt{return} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

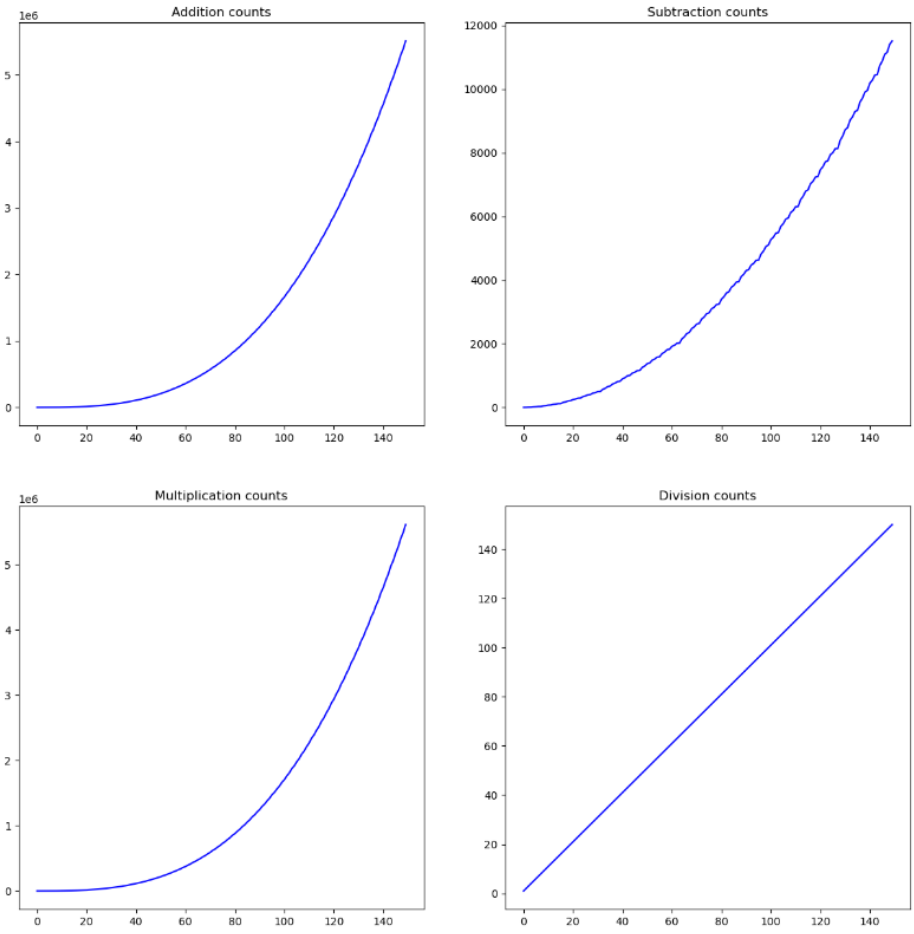
2.3 Analiza algorytmu



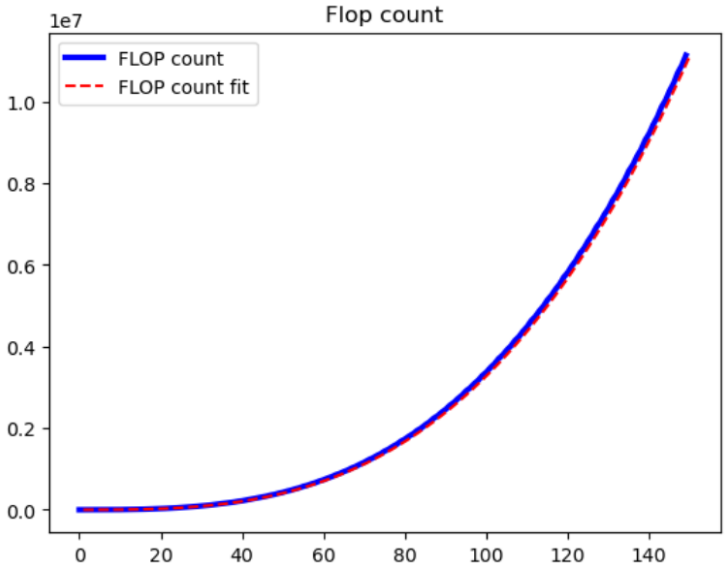
Rysunek 1: Wykres zależności czasu trwania (s) od wielkości macierzy

Na powyższym wykresie w kolorze niebieskim została pokazana analiza czasu trwania odwracania macierzy. Dodatkowo linią czerwoną przerywaną została dobrana krzywa do uzyskanych wyników za pomocą funkcji `curve_fit` z modułu `scipy.optimize`. Funkcja ta dobrała parametry a, n do funkcji $T(x) = ax^n$, które najlepiej odzwierciedlają uzyskane wyniki.

Bardziej interesujący będzie wykładnik n w tej funkcji, ponieważ zadecyduje on o empirycznej złożoności obliczeniowej algorytmu. `curve_fit` wskazała wartość $n \approx 2.81$, co nam daje złożoność czasową $O(n^{2.81})$. Obliczenie empiryczne jest niestety niepoprawne, ponieważ w algorytmie używaliśmy *metody Binet'a* mnożenia macierzy, która ma złożoność $O(n^3)$, zatem nie ma możliwości uzyskania niższej niż ta.



Rysunek 2: Liczba poszczególnych operacji w zależności od wielkości macierzy



Rysunek 3: Liczba wszystkich operacji w zależności od wielkości macierzy

W analizie wszystkich operacji zmiennoprzecinkowych również użyliśmy funkcji `curve_fit` do wyznaczenia wartości parametrów a, n dla funkcji $T(x) = ax^n$. W tym wypadku otrzymaliśmy wynik $n \approx 3.01$, zatem liczba operacji zmiennoprzecinkowych w zależności od ilości wierszy macierzy jest rzędu $O(n^{3.01})$.

3 LU faktoryzacja

3.1 Idea algorytmu

LU faktoryzacja macierzy A polega na znalezieniu takich dwóch macierzy L i U, gdzie L jest macierzą trójkątną dolną, a U trójkątną górną, przy czym $LU = A$.

Zaczynamy klasycznie od podziału A na bloki:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Następnie znajdujemy rekurencyjnie LU faktoryzację macierzy A_{11}:

$$L_{11}U_{11} = A_{11}$$

Przyjmując $S = A_{22} - A_{21}U_{11}^{-1}L_{11}^{-1}A_{12}$ i obliczając rekurencyjnie $L_S U_S = S$ jesteśmy w stanie wyrazić macierze L i U następująco:

$$L = \begin{bmatrix} L_{11} & 0 \\ A_{21}U_{11}^{-1} & L_S \end{bmatrix}$$

$$U = \begin{bmatrix} U_{11} & L_{11}^{-1}A_{12} \\ 0 & U_S \end{bmatrix}$$

3.2 Pseudokod

lu($[a]$) $\rightarrow [1], [a]$

lu(A):

$A_{11}, A_{12}, A_{21}, A_{22}$ = partition(A)

L_{11}, U_{11} = lu(A_{11})

U_{11}^{-1} = inverse(U_{11})

L_{21} = mult(A_{21}, U_{11}^{-1})

L_{11}^{-1} = inverse(L_{11})

U_{12} = mult(L_{11}^{-1}, A_{12})

$S = A_{22}$ - mult(mult(mult(A_{21}, U_{11}^{-1}), L_{11}^{-1}), A_{12})

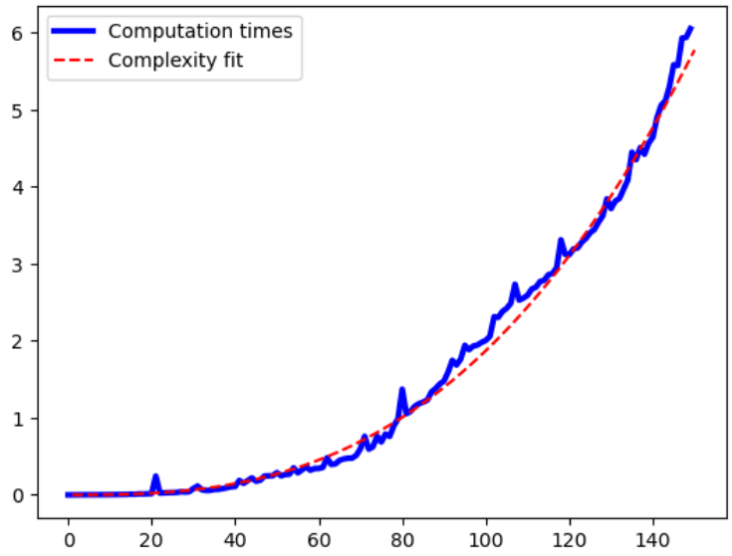
L_S, U_S = lu(S)

U_{22} = U_S

L_{22} = L_S

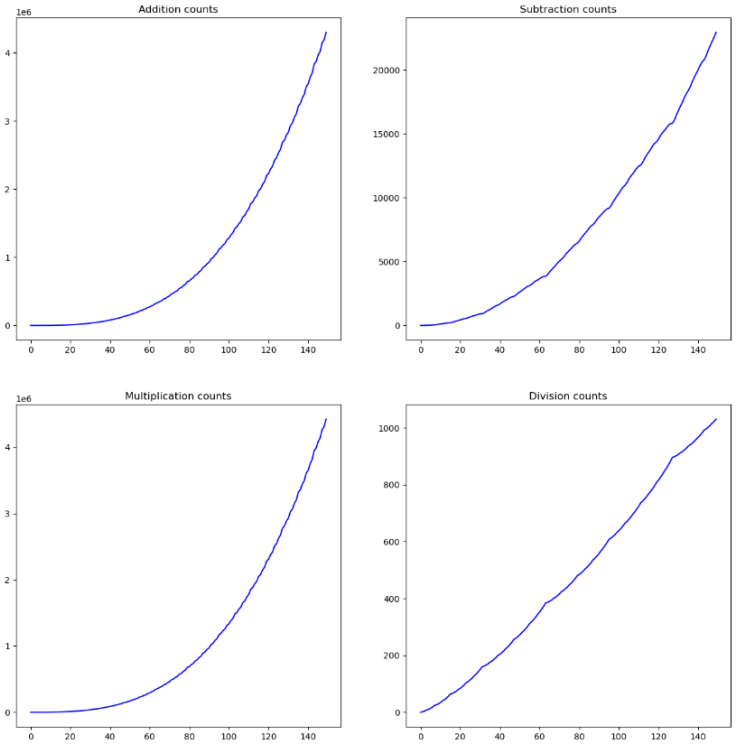
return $\begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}$, $\begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$

3.3 Analiza algorytmu

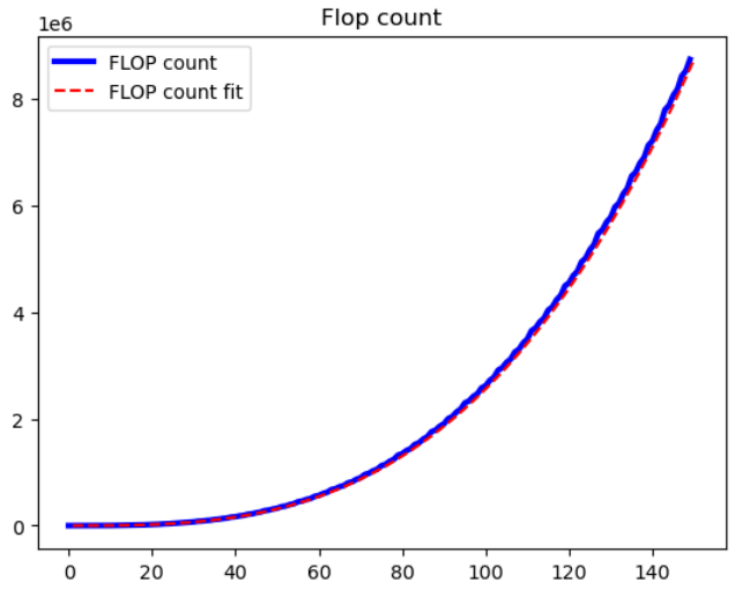


Rysunek 4: Wykres zależności czasu trwania faktoryzacji LU od wielkości macierzy

Podobnie jak w przypadku odwracania macierzy użyliśmy funkcji `curve_fit` do analizy empirycznej złożoności czasowej. W tym przypadku wyszła $O(n^{2.78})$, która niestety również nie jest najbardziej zadowalająca, skoro wiemy, że wykonujemy mnożenie macierzy o koszcie $O(n^3)$.



Rysunek 5: Liczba poszczególnych operacji zmiennoprzecinkowych w faktoryzacji LU



Rysunek 6: Zależność łącznej liczby operacji zmiennoprzecinkowych od wielkości macierzy

Funkcja `curve_fit` wskazała, że liczba operacji zmiennoprzecinkowych jest rzędu $O(n^{3.03})$.

4 Rekurencyjna eliminacja Gaussa

Rekurencyjna eliminacja Gaussa to metoda rozwiązywania układów równań liniowych, która wykorzystuje podejście rekurencyjne do eliminacji zmiennych w macierzy, przekształcając układ do postaci trójkątnej. Zamiast przekształcać całą macierz jednocześnie, metoda ta dzieli macierz na mniejsze bloki, które są przetwarzane osobno, co pozwala na bardziej efektywne wykorzystanie zasobów obliczeniowych. Proces zaczyna się od wyznaczenia dekompozycji LU dla mniejszych bloków macierzy, co prowadzi do wyznaczenia macierzy dolnej LL i górnej UU. Następnie obliczane są odpowiednie macierze odwrotne, które są wykorzystywane do wyznaczenia tzw. dopełnienia Schura – macierzy SS, która reprezentuje zredukowany problem dla pozostałych zmiennych. Dla macierzy SS również wykonuje się dekompozycję LU, aż cały układ zostanie przekształcony do formy umożliwiającej szybkie wyznaczenie rozwiązania. Rekurencyjna eliminacja Gaussa jest szczególnie użyteczna w przypadku dużych macierzy o strukturze blokowej, ponieważ umożliwia rozwiązywanie problemu krok po kroku, bez konieczności jednoczesnego operowania na całej macierzy.

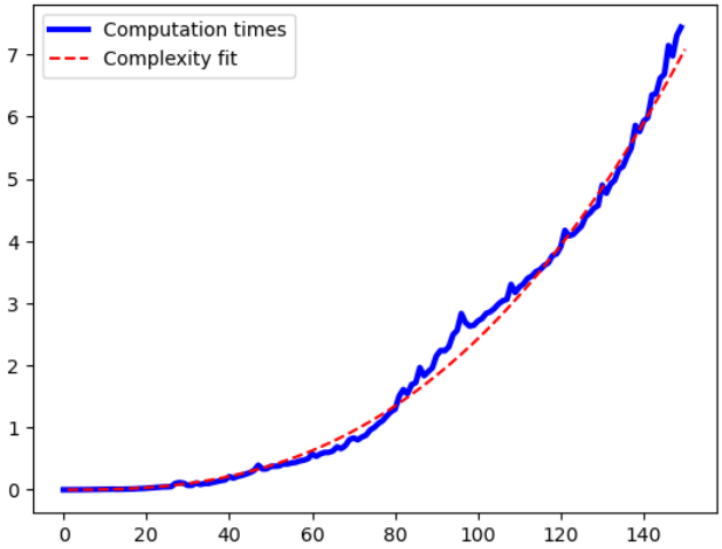
Algorithm 1 Rekurencyjna eliminacja Gaussa

function GaussElimination(A, b, mult)
  $(A_{11}, A_{12}, A_{21}, A_{22}) \leftarrow \text{matrix\_partitions}(A)$
  $n \leftarrow \frac{\text{shape}(A)[0]}{2}$
  $b_1 \leftarrow b[: n]$
  $b_2 \leftarrow b[n :]$
  $(L_{11}, U_{11}) \leftarrow \text{lu\_factorization}(A_{11}, \text{mult})$
  $L_{11}^{-1} \leftarrow \text{invert\_matrix}(L_{11}, \text{mult})$
  $U_{11}^{-1} \leftarrow \text{invert\_matrix}(U_{11}, \text{mult})$
  $S \leftarrow A_{22} - \text{mult}(\text{mult}(\text{mult}(A_{21}, U_{11}^{-1}), L_{11}^{-1}), A_{12})$
  $(L_S, U_S) \leftarrow \text{lu\_factorization}(S, \text{mult})$
  $C_{11} \leftarrow U_{11}$
  $C_{12} \leftarrow \text{mult}(L_{11}^{-1}, A_{12})$
  $C_{22} \leftarrow U_S$
  $\text{RHS}_1 \leftarrow \text{mult}(L_{11}^{-1}, b_1)$
  $\text{RHS}_2 \leftarrow \text{mult}(\text{invert\_matrix}(L_S, \text{mult}), b_2 - \text{mult}(\text{mult}(\text{mult}(A_{21}, U_{11}^{-1}), L_{11}^{-1}), b_1))$
  $x_2 \leftarrow \text{mult}(\text{invert\_matrix}(U_S, \text{mult}), \text{RHS}_2)$
  $x_1 \leftarrow \text{mult}(\text{invert\_matrix}(U_{11}, \text{mult}), \text{RHS}_1 - \text{mult}(C_{12}, x_2))$
  $x \leftarrow \text{concatenate}(x_1, x_2)$
  return x
end function

5 Rekurencyjne liczenie wyznacznika

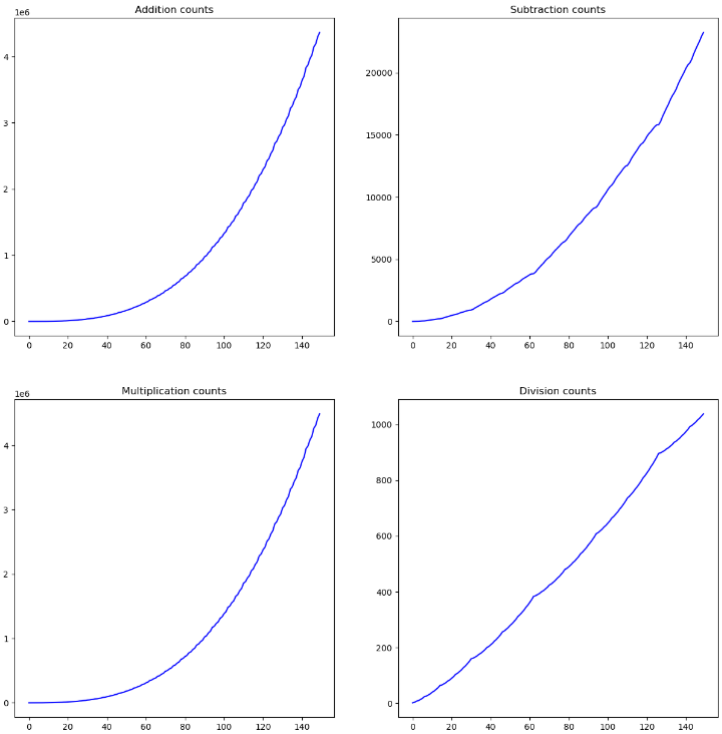
Wyznaczanie wyznacznika macierzy za pomocą rozkładu LU jest znacznie bardziej efektywną metodą w porównaniu do tradycyjnego rozwinięcia Laplace'a, szczególnie w przypadku większych macierzy. Rozkład LU polega na rozłożeniu macierzy AA na iloczyn dwóch macierzy: macierzy dolnotrójkątnej LL oraz macierzy górnotrójkątnej UU, gdzie A=LUA=LU. Po przeprowadzeniu rozkładu LU, wyznacznik macierzy AA można wyznaczyć w prosty sposób jako iloczyn wyznaczników macierzy LL i UU. Ponieważ macierz LL jest dolnotrójkątna z jedynkami na diagonali, jej wyznacznik wynosi zawsze 1, a wyznacznik macierzy UU to po prostu iloczyn elementów znajdujących się na jej przekątnej. W rezultacie, wyznacznik macierzy AA jest równy iloczynowi elementów na przekątnej macierzy UU. Metoda ta jest znacznie bardziej wydajna obliczeniowo, ponieważ rozkład LU można przeprowadzić w czasie O(n3)O(n3), a samo wyznaczanie wyznacznika po rozkładzie wymaga jedynie wykonania prostych operacji mnożenia. Dlatego też metoda z wykorzystaniem rozkładu LU jest powszechnie stosowana do obliczania wyznacznika dużych macierzy w praktyce inżynieryjnej i naukowej.

	Rozmiar	Operacje addytywne	Operacje multiplikatywne	Wszystkie operacje zmiennoprzecinkowe	Czas wykonania
0	4	62	72	134	0.002701
1	8	1018	594	1612	0.015897
2	16	10576	4356	14932	0.101679
3	32	90962	30738	121700	1.445360
4	64	712536	214596	927132	5.064898
5	128	5310786	1495842	6806628	35.039661
6	256	38514632	10435572	48950204	255.147330
7	512	275067634	72883890	347951524	1771.125482

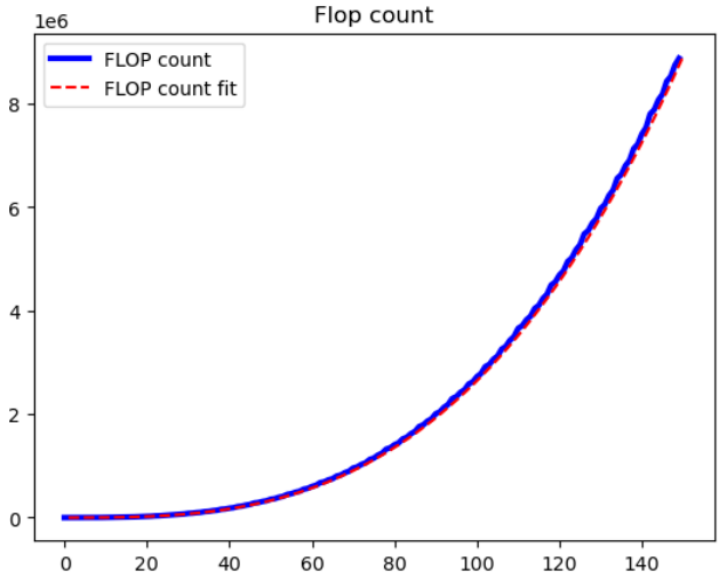


Rysunek 7: Czas trwania obliczania wyznacznika w zależności od wielkości macierzy

"Wyznaczona"teoretyczna złożoność obliczeniowa względem ilości operacji zmiennoprzecinkowych to $\approx O(n^{2.7978\cdots})$. Znów jest niższa, niż minimalna, która wynosi $O(n^3)$.



Rysunek 8: Wykresy zależności liczby poszczególnych operacji od wielkości macierzy



Rysunek 9: Wykres zależności łącznej liczby operacji zmiennoprzecinkowych od wielkości macierzy

"Wyznaczona"teoretyczna złożoność obliczeniowa względem ilości operacji zmiennoprzecinkowych to $\approx O(n^{2.8301\cdots})$

6 Wnioski

- operacje odrawcania oraz podziału LU macierzy są bardzo przydatne - można je wykorzystywać np. w odwracaniu macierzy lub liczeniu układów równań.
- empiryczne obliczenia złożoności obliczeniowej powyższych algorytmów nie spisało się najlepiej, ponieważ złożoność czasowa wyniosła poniżej $O(n^3)$. Mnożenie macierzy, które wykonujemy w powyższych programach wyznacza minimalną złożoność, która powinna wyjść $O(n^3)$.
- Złożoność liczby FLOPSów od wielkości macierzy całkiem dobrze została wyznaczona, ponieważ wychodziło mniej więcej rzędu $O(n^3)$.