Algorytmy Macierzowe

Sprawozdanie II

Grupa wtorek 13:00b

Michał Kuszewski i Michał Nożkiewicz

6 listopada 2023

1 Opis zadania i użyte narzędzia

Naszym zadaniem było zaimplementowanie i dokonanie analizy trzech algorytmów:

1. Rekurencyjne odwracanie macierzy
2. Rekurencyjna LU faktoryzacja
3. Rekurencyjne obliczanie wyznacznika

Do realizacji zadania użyliśmy języka Python. Korzystaliśmy z bibliotek numpy, matplotlib, pandas i scipy.

2 Pseudokody Algorytmów

2.1 Rekurencyjne odwracanie macierzy

W tym algorytmie macierz dzielimy na 4 równych rozmiarów podmacierze.

Algorytm 1: Matrix Inverse

Data: Matrix A
Result: Matrix $B = A^{-1}$

n = A.size
if $n = 1$ **then**
    return $1/A[0,0]$; `// inverse the only element`
else
    $A_{1,1}^{-1} := inverse(A_{1,1})$
    $T_1 := A_{2,1} * A_{1,1}^{-1}$
    $S_{2,2} := A_{2,2} - (T_1 * A_{1,2})$
    $S_{2,2}^{-1} := inverse(S_{2,2})$
    $T_2 := A_{1,1}^{-1} * A_{1,2} * S_{2,2}^{-1}$
    $B_{1,1} := A_{1,1}^{-1} + (T_2 * T_1)$
    $B_{1,2} := -T_2$
    $B_{2,1} := -S_{2,2}^{-1} * T_1$
    $B_{2,1} := S_{2,2}^{-1}$
    return B

2.2 LU faktoryzacja

Podobnie jak w algorytmie odwracania macierzy, dzielimy macierz na cztery części. W trakcie algorytmu używana jest także funkcja odwracania macierzy zdefiniowana powyżej.

Algorytm 2: LU

Data: Matrix A
Result: Matrices L and U, $L * U = A$

n = A.size
if $n = 1$ **then**
  return $[1], A$
else
  $L_{1,1}, U_{1,1} := lu(A_{1,1})$
  $U_{1,1}^{-1} := inverse(U_{1,1})$
  $T_1 := A_{2,1} * U_{1,1}^{-1}$
  $L_{2,1} := T_1$
  $L_{1,1}^{-1} := inverse(L_{1,1})$
  $T_2 := L_{1,1}^{-1} * A_{1,2}$
  $U_{1,2} := T_2$
  $L_{2,2}, U_{2,2} := lu(A_{2,2} - T_1 * T_2)$
  $L_{1,1} = \mathbf{0}$
  $U_{2,1} = \mathbf{0}$
  return L, U

2.3 Obliczanie wyznacznika

Algorytm korzysta z dekompozycji LU. Aby obliczyć wyznacznik wystarczy

Algorytm 3: LU

Data: Matrix A
Result: Determinant of matrix A

$L, U := lu(A)$
$det := 1$
for $i = 0;\ i < A.size;\ i = i + 1$ **do**
  $det := det \cdot U[i, i]$
return det

2.4 Istotne fragmenty implementacji

Jako, że sam kod w pythonie nie różnił się praktycznie niczym od pseudokodu uznaliśmy, że nie ma sensu zamieszczać fragmentów kodu.

3 Analiza algorytmów

Do mnożenia macierzy użyliśmy rekurencyjnych funkcji z pierwszego laboratorium, a dokładnie algorytmu Bineta i Strassena. Dla każdego z algorytmów dokonaliśmy pomiarów czasu wykonania, a także zliczyliśmy ilość operacji zmiennoprzecinkowych (dodawania, mnożenia oraz dzielenia). Do danych przedstawionych na wykresie krzywą dopasowaliśmy z użyciem funkcji curve_fit z pakietu scipy.optimize. Założyliśmy, że wyniki pomiarów, są zależne od rozmiarów macierzy zależnością $y = ax^b + \epsilon$, gdzie a i b to szukane parametry, a ϵ to błąd pomiarów. W analizie złożności interesuje na

tak naprawdę tylko parametr b. Funkcja curve_fit do wyznaczenia optymalnych parametrów stosuje metodę najmniejszych kwadratów. Znalezione parametry zamieściliśmy na wykresach z dokładnością do dwóch miejsc po przecinku.

3.1 Algorytm odwracania macierzy

Jeśli $T(n)$ oznaczymy jako złożoność algorytmu odwracania macierzy to funkcję tą można opisać rekurencyjnie w następujący sposób.

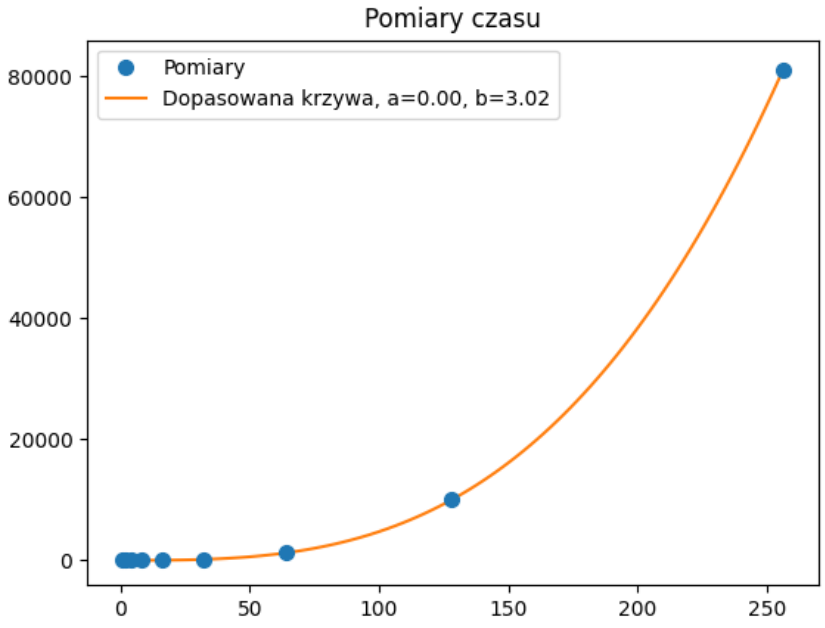
$$T(n) = 2T\left(\frac{n}{2}\right) + O(n^{\omega}) \tag{1}$$

Z twierdzenia o rekurencji uniwersalnej wynika, że $T(n) = O(n^{\omega})$
$O(n^{\omega})$ oznacza złożoność mnożenia macierzy. Możliwe wartości wykładnika to 3 dla mnożenia Bineta oraz $\log_2 7$ dla Strasena.

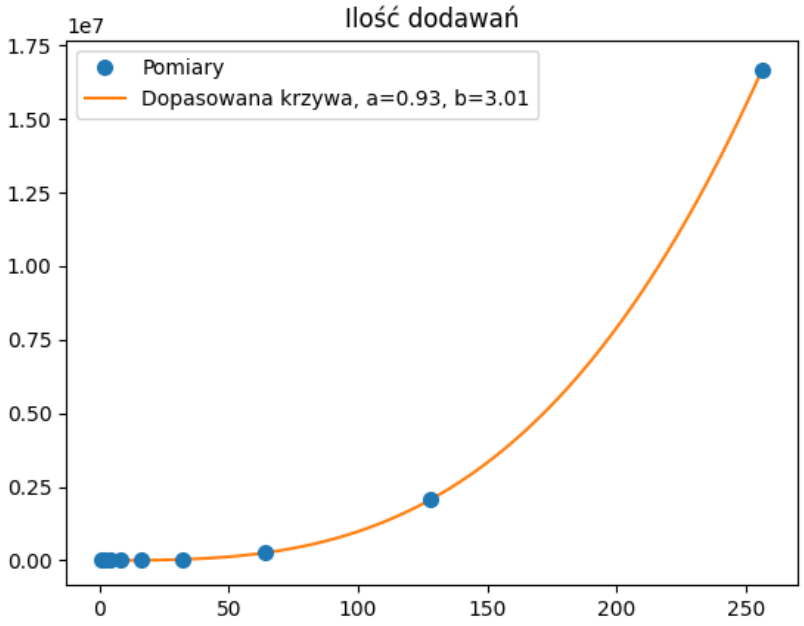
3.1.1 Mnożenie Bineta

Wymiar macierzy	Pomiary czasu[ms]	Ilość dodawań	Ilość mnożeń	Ilość dzieleń
1	0.0000	0	0	1
2	0.0001	2	6	2
4	0.0003	36	60	4
8	0.0030	392	504	8
16	0.0223	3600	4080	16
32	0.2032	30752	32736	32
64	1.2447	254016	262080	64
128	9.9929	2064512	2097024	128
256	80.9650	16646400	16776960	256

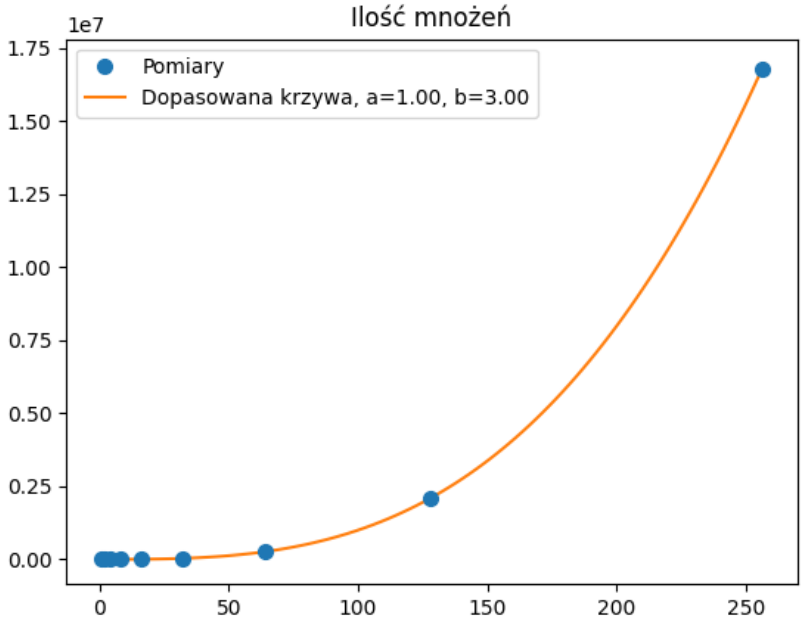
Tabela 1: Wyniki pomiarów dla odwracania macierzy z mnożeniem Bineta



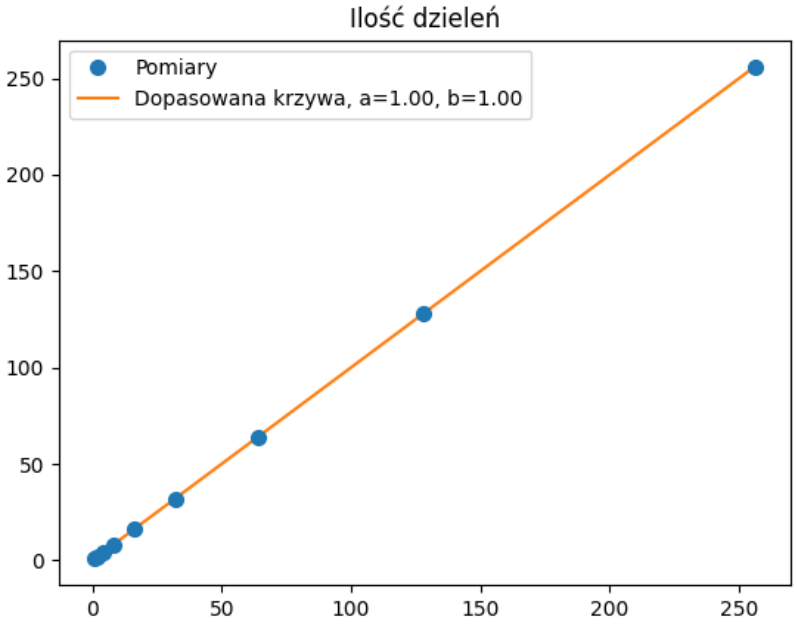
Rysunek 1: Czas obliczeń



Rysunek 2: Ilość dodawań



Rysunek 3: Ilość mnożeń

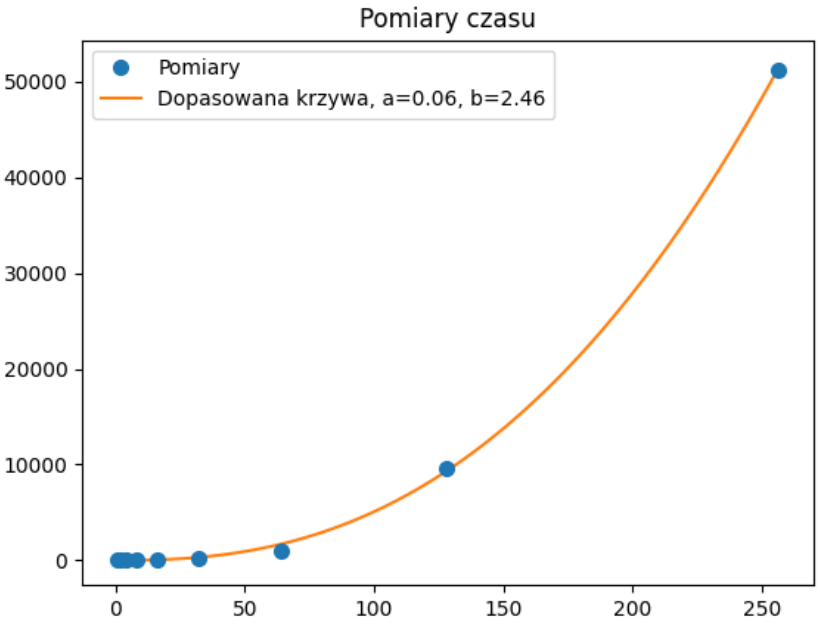


Rysunek 4: Ilość dzieleń

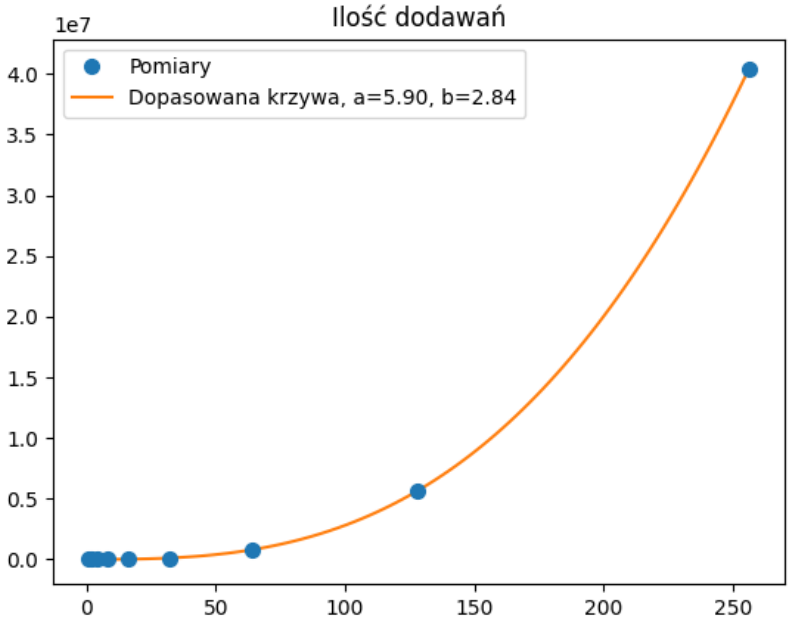
3.1.2 Mnożenia Strassena

Wymiar macierzy	Pomiary czasu[ms]	Ilość dodawań	Ilość mnożeń	Ilość dzieleń
1	0.0000	0	0	1
2	0.0000	2	6	2
4	0.0004	120	54	4
8	0.0044	1460	402	8
16	0.0241	13092	2862	16
32	0.1660	103916	20130	32
64	1.0004	778068	141102	64
128	9.5860	5652236	988098	128
256	51.2216	40394964	6917454	256

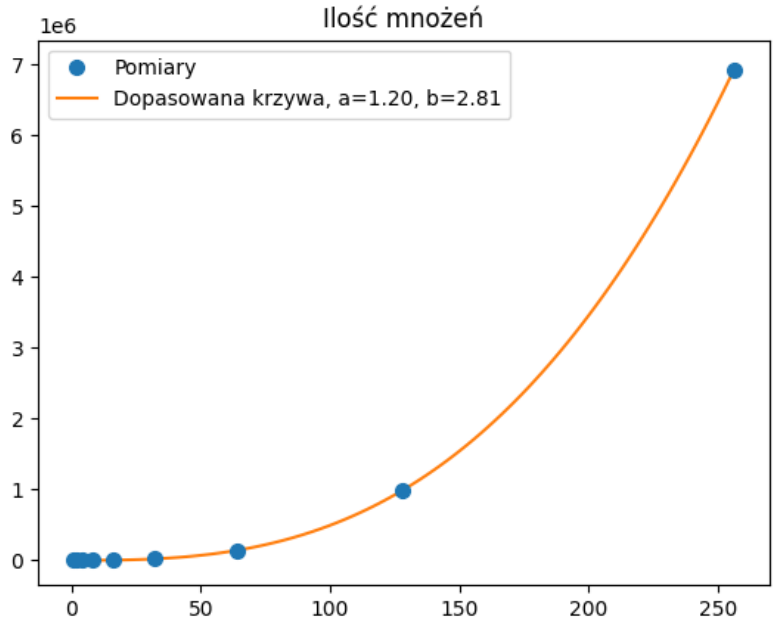
Tabela 2: Wyniki pomiarów dla odwracania macierzy z mnożeniem Strassena



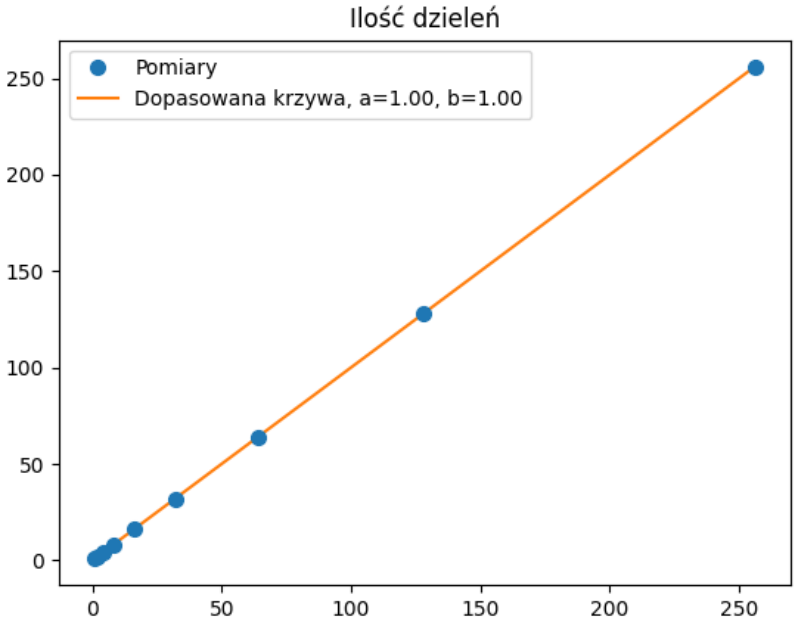
Rysunek 5: Czas obliczeń



Rysunek 6: Ilość dodawań



Rysunek 7: Ilość mnożeń



Rysunek 8: Ilość dzieleń

3.2 Faktoryzacja LU

Podobną analizę można wykonać dla faktoryzacji LU.
Oznaczmy $C(n)$ jako złożoność faktoryzacji LU. Mamy wtedy następującą zależność.

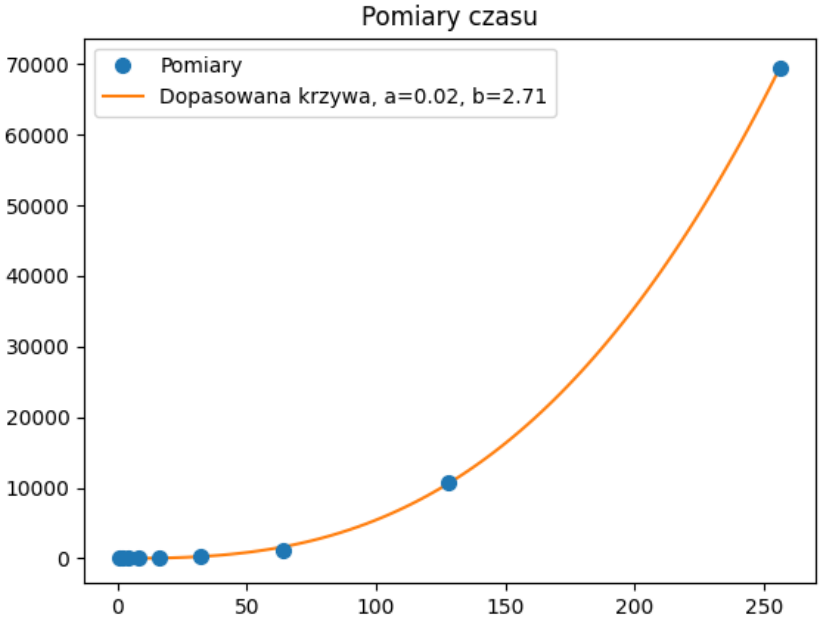
$$C(n) = 2C\left(\frac{n}{2}\right) + 2T\left(\frac{n}{2}\right) + O(n^{\omega}) \tag{2}$$

Podstawiając za $T(n)$ wcześniej obliczone $O(n^\omega)$ uzyskujemy analogiczne równania rekurencyjne jak dla T, a więc i ten sam wynik $C(n) = O(n^\omega)$.

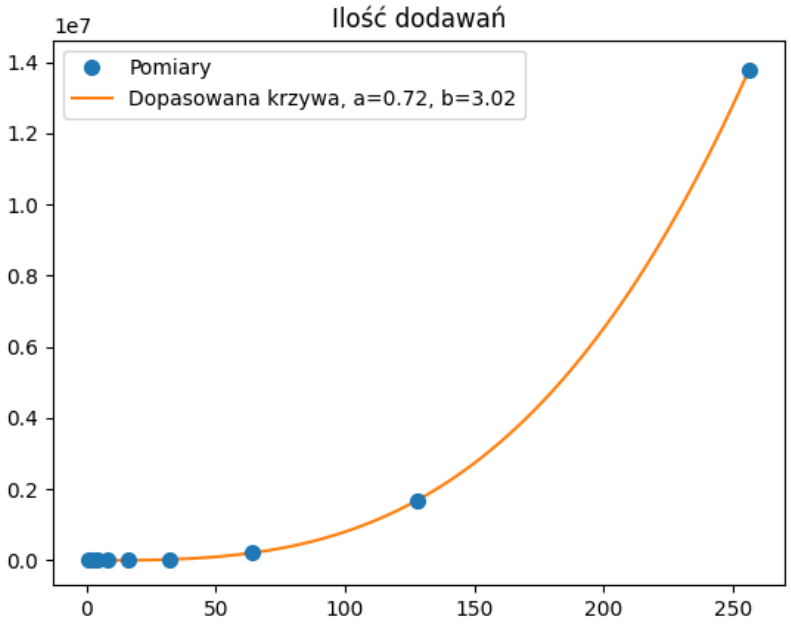
3.2.1 Mnożenia Bineta

Wymiar macierzy	Pomiary czasu[ms]	Ilość dodawań	Ilość mnożeń	Ilość dzieleń
1	0.0000	0	0	0
2	0.0000	0	3	2
4	0.0003	17	42	8
8	0.0024	254	396	24
16	0.0201	2652	3336	64
32	0.1931	24088	27120	160
64	1.0388	205168	218016	384
128	10.7971	1693536	1746624	896
256	69.4522	13762496	13978752	2048

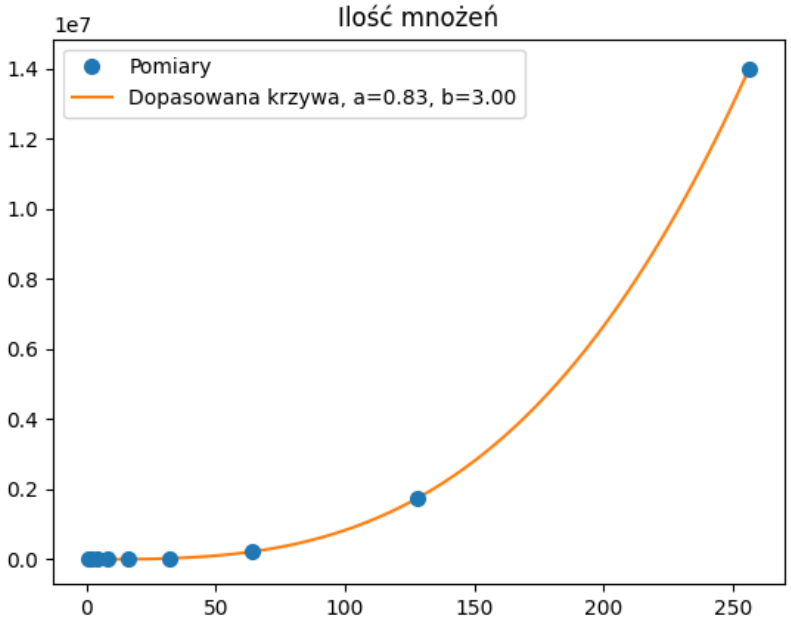
Tabela 3: Wyniki pomiarów dla faktoryzacji LU z mnożeniem Bineta



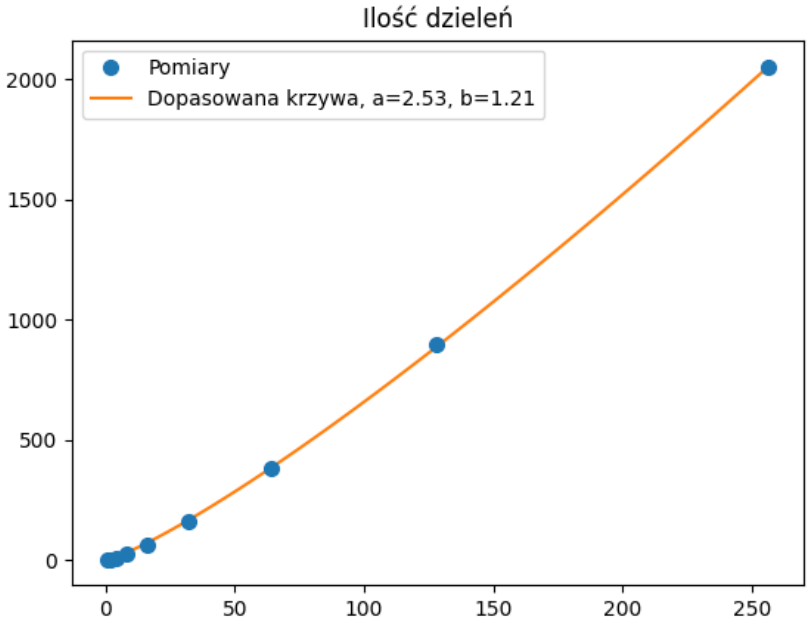
Rysunek 9: Czas obliczeń



Rysunek 10: Ilość dodawań



Rysunek 11: Ilość mnożeń

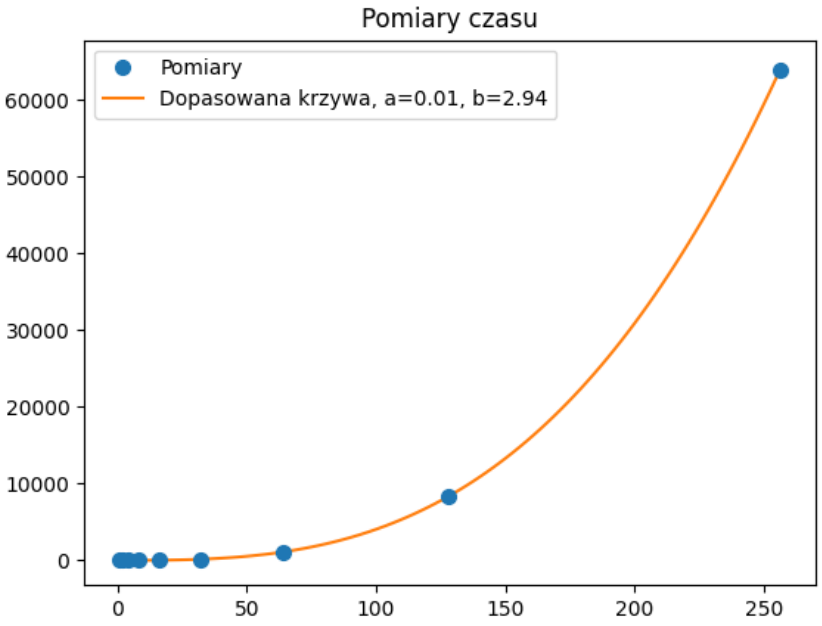


Rysunek 12: Ilość dzieleń

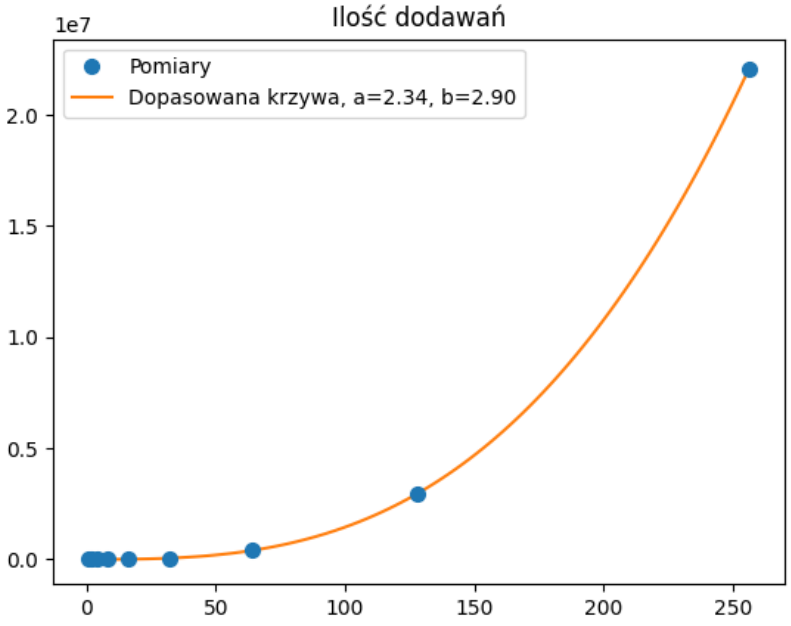
3.2.2 Mnożenia Strassena

Wymiar macierzy	Pomiary czasu[ms]	Ilość dodawań	Ilość mnożeń	Ilość dzieleń
1	0.0000	0	0	0
2	0.0000	0	3	2
4	0.0003	59	39	8
8	0.0024	704	351	24
16	0.0274	6330	2829	64
32	0.1228	51178	22035	160
64	0.9818	394030	170133	384
128	8.3764	2963346	1313139	896
256	63.8708	22049054	10157925	2048

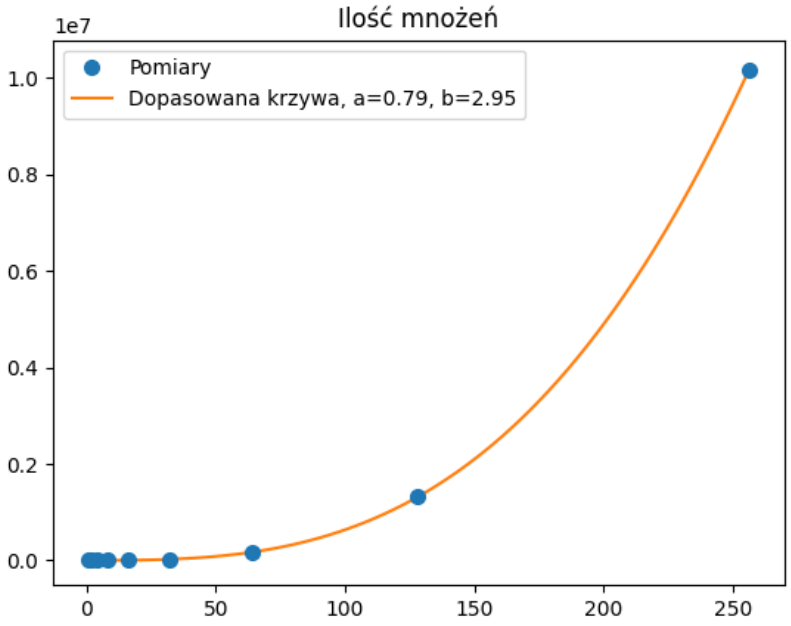
Tabela 4: Wyniki pomiarów dla faktoryzacji LU z mnożeniem Strassena



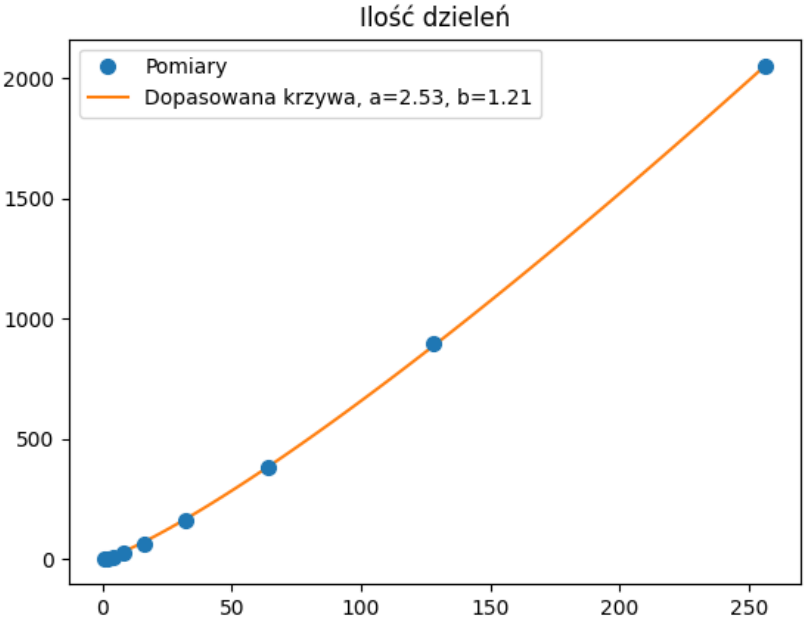
Rysunek 13: Czas obliczeń



Rysunek 14: Ilość dodawań



Rysunek 15: Ilość mnożeń



Rysunek 16: Ilość dzieleń

3.3 Wyznacznik macierzy

Obliczenie wyznacznika polegało na wyznaczeniu dekompozycji LU, a następnie przemnożeniu elementów z diagonali macierzy U. Wyniky są analogiczne jak dla faktoryzacji LU.

4 Sprawdzenie poprawności

Poprawność implementacji naszych algorytmów sprawdzaliśmy z użyciem funkcji biblioteki numpy realizujących analogiczne algorytmy. Macierze porównywaliśmy obliczając normę Frobeniusa z ich różnicy. Dla każdego algorytmu wynik normy macierzowej był w zakresie dopuszczalnego błędu.

5 Wnioski

Po wynikach widać, że użycie lepszych algorytmów mnożenia macierzy może przyśpieszyć algorytmy odwracania macierzy, faktoryzacji LU, a także obliczania wyznacznika. Widać to najlepiej na przykładzie algorytmu odwracania macierzy. Dla Strassena otrzymaliśmy bardzo niską złożoność $O(n^{2.46})$, lecz wynika to z błędu przy pomiarze czasu. Więcej mówi nam w tym przypadku ilość operacji zmiennoprzecinkowych, gdyż one nie są obarczone błędem pomiarowym, a jedynie niedokładnością dopasowywania krzywej.