

Algorytmy Macierzowe

Sprawozdanie I

Grupa wtorek 13:00b

Michał Kuszewski i Michał Nożkiewicz

22 października 2023

1 Opis zadania i użyte narzędzia

Naszym zadaniem było zaimplementowanie i dokonanie analizy trzech algorytmów do mnożenia macierzy:

1. Algorytm Bineta
2. Algorytm Strassena
3. Algorytm znaleziony przez AlphaTensor (sztuczną inteligencję firmy DeepMind)

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

2 Pseudokody Algorytmów

We wszystkich algorytmach będziemy zapisać macierze blokowo, gdzie $A_{i,j}$ oznacza odpowiedni blok macierzy A .

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$$

2.1 Algorytm Bineta

W algorytmie Bineta obie macierze są dzielone na 4 równe podmacierze.

Algorytm 1: Binet Matrix Multiplication

Data: Matrices A and B

Result: Matrix $C = A \cdot B$

```
1  $n = A.size$ 
2 if  $n = 1$  then
3   return  $A \odot B$  ; // element-wise multiplication
4 else
5    $C_{1,1} := Binet(A_{1,1}, B_{1,1}) + Binet(A_{1,2}, B_{2,1})$ 
6    $C_{1,2} := Binet(A_{1,1}, B_{1,2}) + Binet(A_{1,2}, B_{2,2})$ 
7    $C_{2,1} := Binet(A_{2,1}, B_{1,1}) + Binet(A_{2,2}, B_{2,1})$ 
8    $C_{2,2} := Binet(A_{2,1}, B_{1,2}) + Binet(A_{2,2}, B_{2,2})$ 
9   return  $C$ 
```

2.2 Algorytm Strassena

W algorytmie Strassena dzielimy wejściowe macierze identycznie jak w algorytmie Bineta.

Algorytm 2: Strassen Matrix Multiplication

Data: Matrices A and B
Result: Matrix $C = A \cdot B$

```
1  $n = A.size$ 
2 if  $n = 1$  then
3   return  $A \odot B$  ; // element-wise multiplication
4 else
5    $P_1 := Strassen(A_{1,1} + A_{2,2}, B_{1,1} + B_{2,2})$ 
6    $P_2 := Strassen(A_{2,1} + A_{2,2}, B_{1,1})$ 
7    $P_3 := Strassen(A_{1,1}, B_{1,2} - B_{2,2})$ 
8    $P_4 := Strassen(A_{2,2}, B_{2,1} - B_{1,1})$ 
9    $P_5 := Strassen(A_{1,1} + A_{1,2}, B_{2,2})$ 
10   $P_6 := Strassen(A_{2,1} - A_{1,1}, B_{1,1} + B_{1,2})$ 
11   $P_7 := Strassen(A_{1,2} - A_{2,2}, B_{2,1} + B_{2,2})$ 
12   $C_{1,1} := P_1 + P_4 - P_5 + P_7$ 
13   $C_{1,2} := P_3 + P_5$ 
14   $C_{2,1} := P_2 + P_4$ 
15   $C_{2,2} := P_1 - P_2 + P_3 + P_6$ 
16   $C_{2,2} := P_1 - P_2 + P_3 + P_6$ 
17  return  $C$ 
```

2.3 Algorytm sztucznej inteligencji

W tym algorytmie pierwszą macierz dzielimy na 20 bloków(4 wiersze i 5 kolumn), a drugą macierz dzielimy na 25 bloków(5 wierszy i 5 kolumn)

Jako, że cały algorytm zajmuje ponad 100 linijek zapisaliśmy tylko fragment.

Algorytm 3: AlphaTensor Matrix Multiplication

Data: Matrices A and B
Result: Matrix $C = A \cdot B$

```
1  $n = A.size[0]$ 
2 if  $n = 1$  then
3   return  $A \odot B$  ; // element-wise multiplication
4 else
5    $H_1 := Ai(A_{3,2}, -B_{2,1} - B_{2,5} - B_{3,1})$ 
6    $H_2 := Ai(A_{2,2} + A_{2,5} - A_{3,5}, -B_{2,5} - B_{5,1})$ 
7   ...
8    $H_{76} := Ai(A_{1,3} + A_{3,3}, -B_{1,1} + B_{1,4} - B_{1,5} + B_{2,4} + B_{3,4} - B_{3,5})$ 
9    $C_{1,1} := -H_{10} + H_{12} + H_{14} - H_{15} - H_{16} + H_{53} + H_5 - H_{66} - H_7$ 
10   $C_{2,1} := H_{10} + H_{11} - H_{12} + H_{13} + H_{15} + H_{16} - H_{17} - H_{44} + H_{51}$ 
11  ...
12   $C_{4,5} := -H_{12} - H_{29} + H_{30} - H_{34} + H_{35} + H_{39} + H_3 - H_{45} + H_{57} + H_{59}$ 
13  return  $C$ 
```

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.

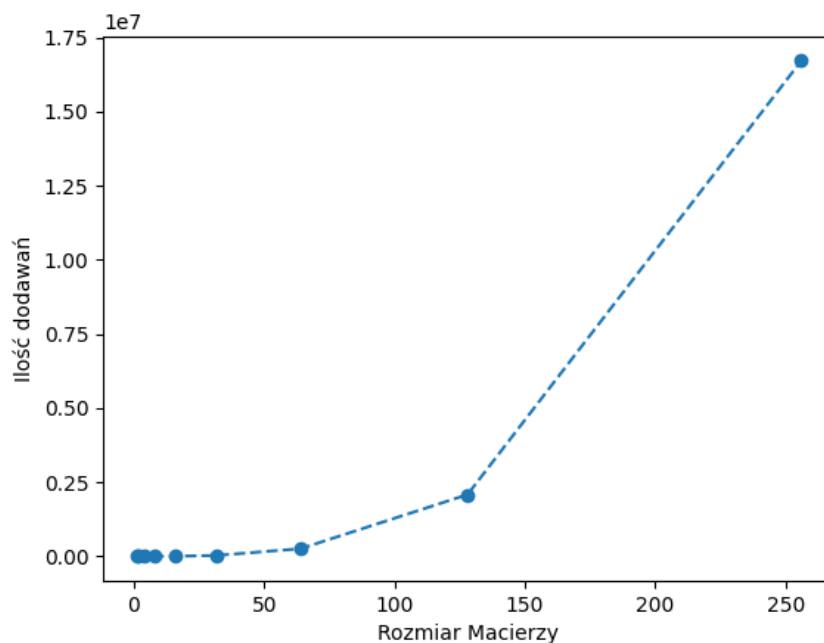
3 Analiza algorytmów

Dla każdego z algorytmów dokonaliśmy pomiarów czasu wykonania, a także zliczyliśmy ilość operacji zmiennoprzecinkowych (dodawania i mnożenia). 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. Funkcja `curve_fit` do wyznaczenia optymalnych parametrów stosuje metodę najmniejszych kwadratów, jest to równoważne temu, że błąd ma rozkład normalny z wartością oczekiwaną równą 0.

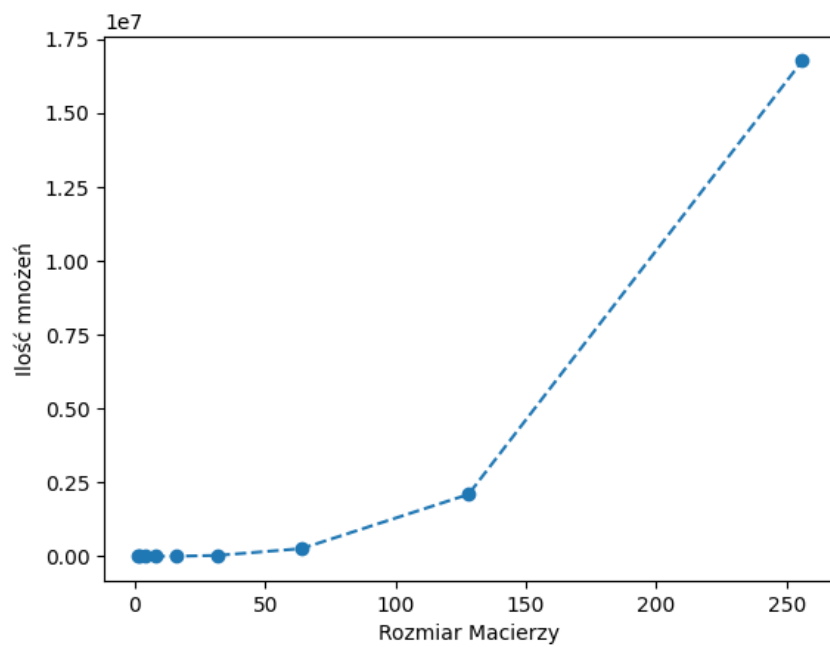
3.1 Algorytm Bineta

Rozmiar macierzy[wymiar 1]	Ilość dodawań	Ilość mnożeń	Czas[s]
1	0	1	0.000000
2	4	8	0.000100
4	48	64	0.000600
8	448	512	0.006100
16	3840	4096	0.042700
32	31744	32768	0.188100
64	258048	262144	1.976100
128	2080768	2097152	10.197400
256	16711680	16777216	80.757700

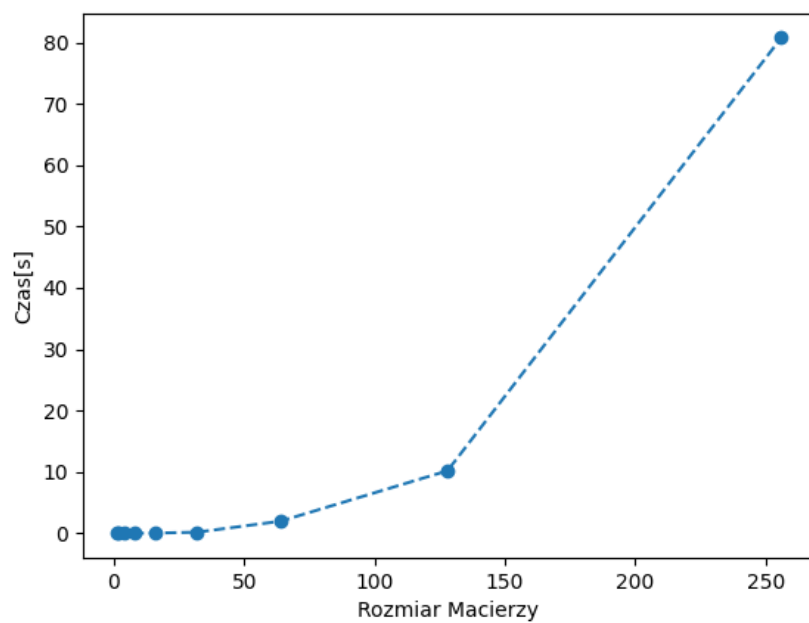
Tabela 1: Wyniki pomiarów dla algorytmu Bineta



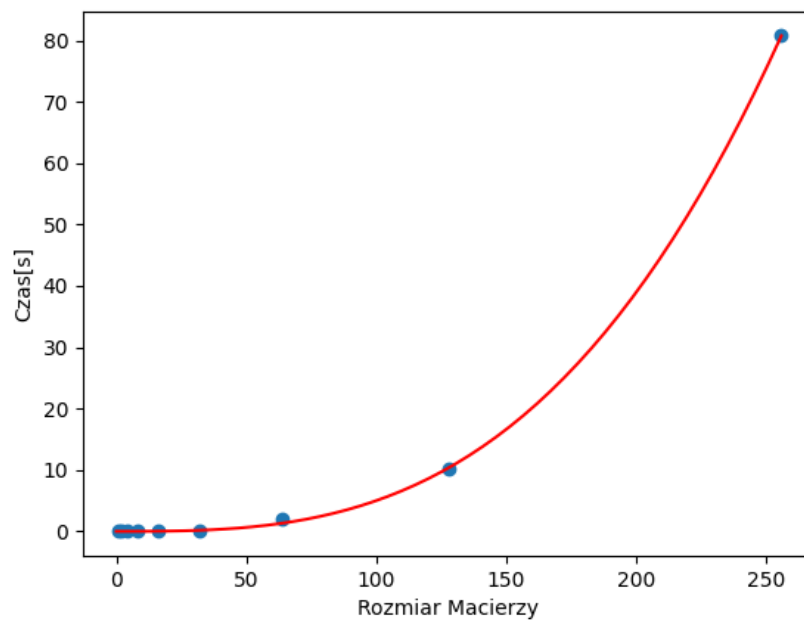
Rysunek 1: Pomiary ilości operacji dodawania



Rysunek 2: Pomiary ilości operacji mnożenia



Rysunek 3: Pomiary czasu



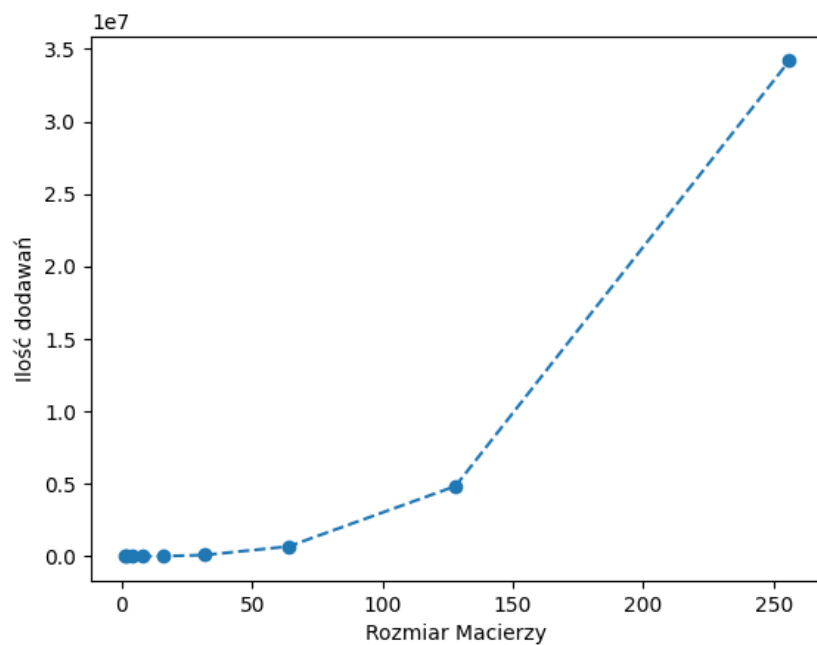
Rysunek 4: Krzywa dopasowana do pomiaru czasu

Krzywa, którą udało się dopasować do danych jest postaci $y = ax^b$, gdzie $a = 5.948$, a $b = 2.961$. Pokrywa się to z teoretyczną złożonością algorytmu, która wynosi $O(n^3)$.

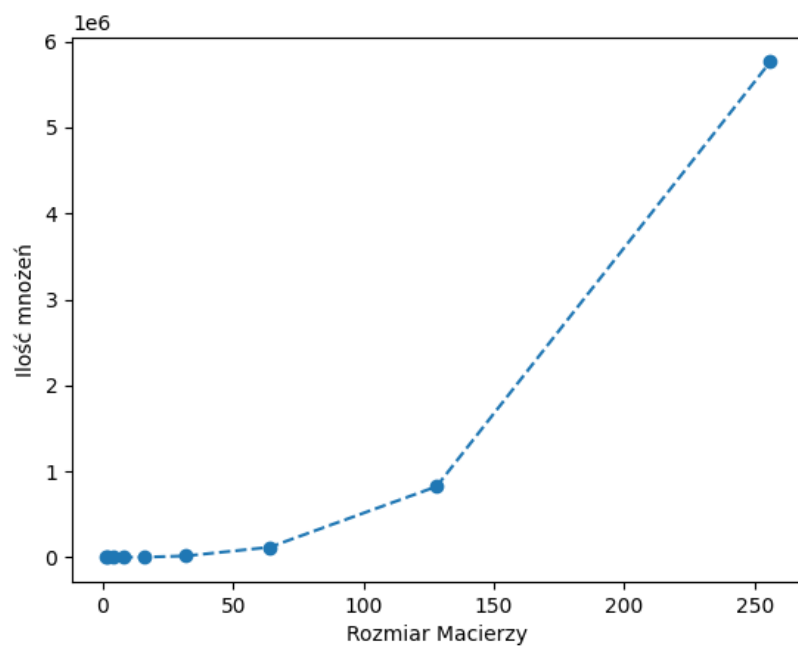
3.2 Algorytm Strassena

Rozmiar macierzy[wymiar 1]	Ilość dodawań	Ilość mnożeń	Czas[s]
1	0	1	0.000000
2	18	7	0.000100
4	198	49	0.000700
8	1674	343	0.006200
16	12870	2401	0.027300
32	94698	16807	0.187400
64	681318	117649	0.955400
128	4842954	823543	6.281100
256	34195590	5764801	46.003400

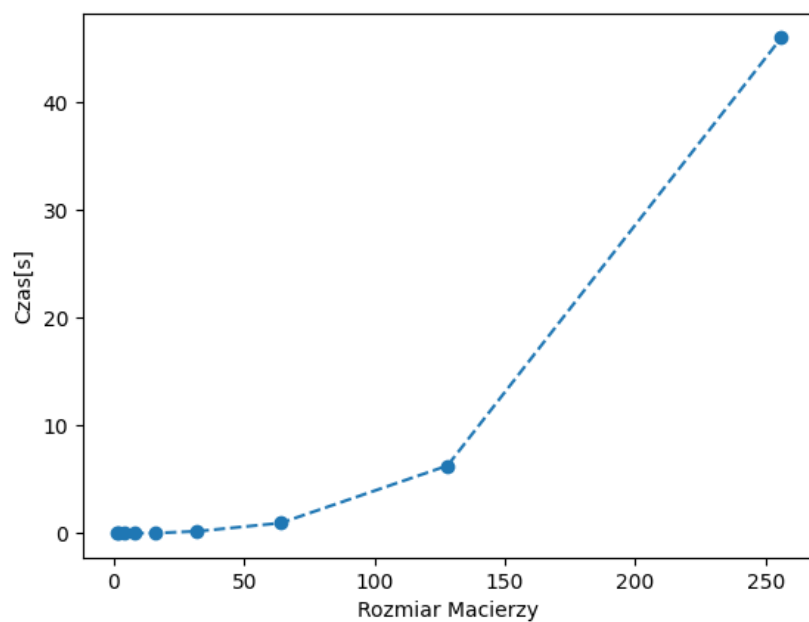
Tabela 2: Wyniki pomiarów dla algorytmu Strassena



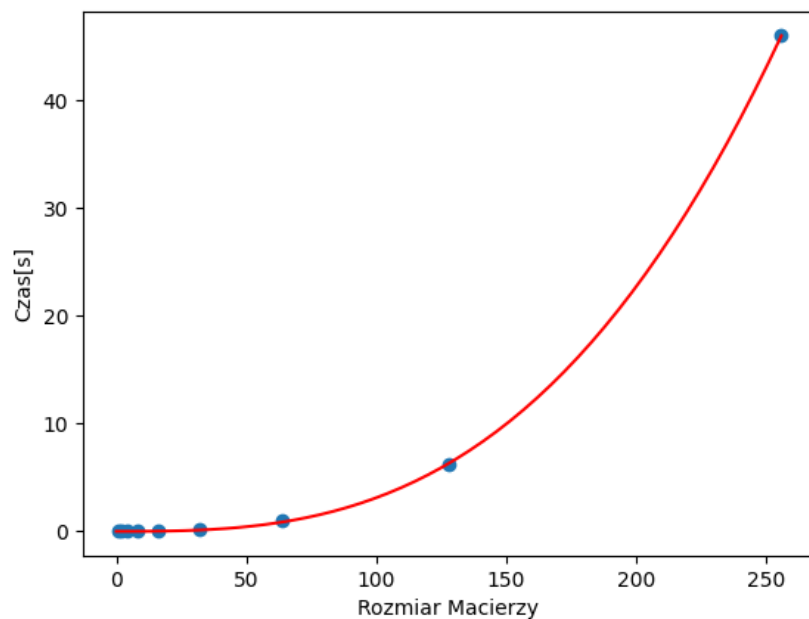
Rysunek 5: Pomiary ilości operacji dodawania



Rysunek 6: Pomiary ilości operacji mnożenia



Rysunek 7: Pomiary czasu



Rysunek 8: Krzywa dopasowana do pomiaru czasu

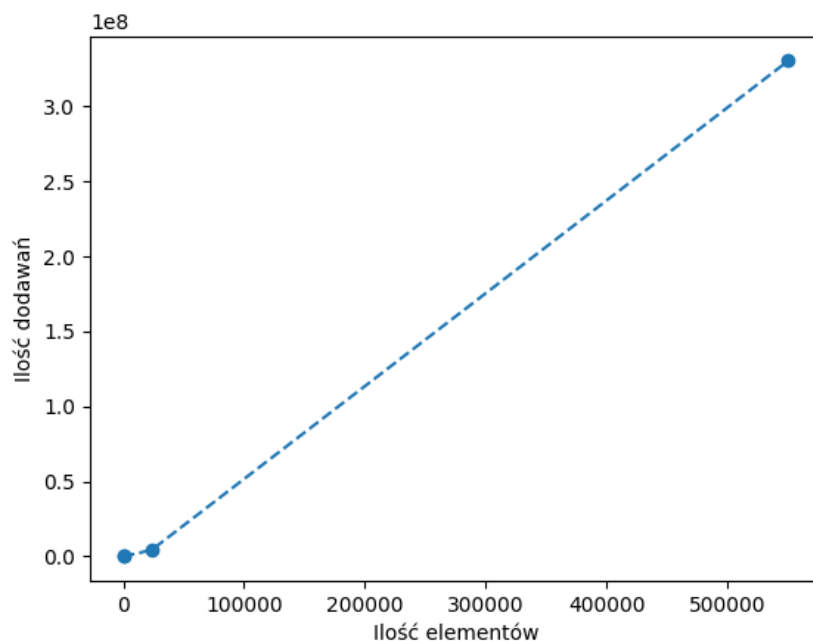
W algorytmie Strassena widać spadek w ilości wykonanych mnożeń, przy jednoczesnym wzroście wykonanych dodawań. Tym razem parametry miały wartości $a = 5.763$, $b = 2.866$. Przy czym teoretyczna złożoność algorytmu Strassena wynosi około $O(n^{2.807})$.

3.3 Algorytm Sztucznej inteligencji

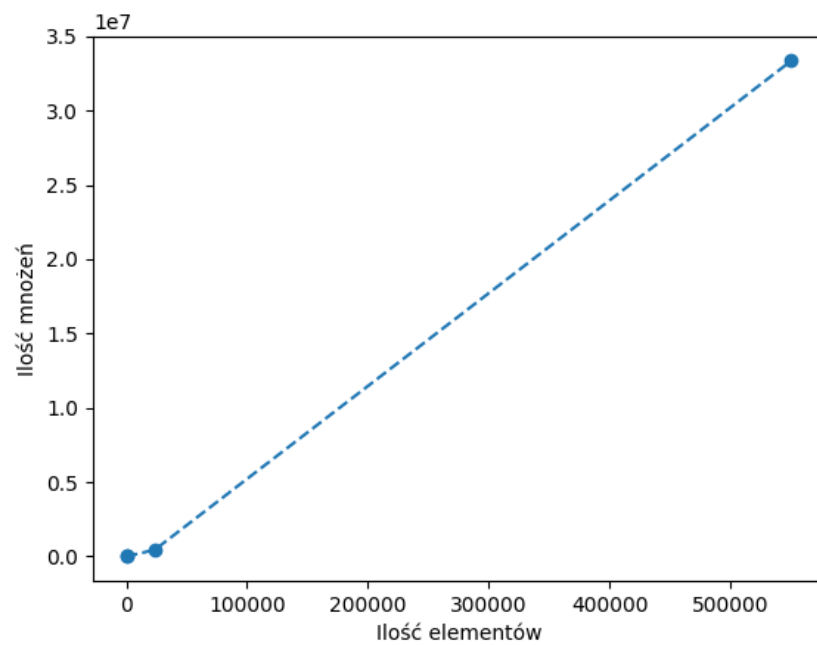
W poprzednich algorytmach testowane macierze były wymiarów $2^k \times 2^k$, gdzie k była liczbą naturalną. Algorytm znaleziony przez AlphaTensor stosuje się jednak do mnożenia macierzy z których jedna ma wymiary $4^k \times 5^k$, a druga $5^k \times 5^k$. Algorytm da się uogólnić do mnożenia macierzy postaci $4^a \times 5^b$ i $5^b \times 5^b$, gdzie $a \neq b$. Lecz jako, że jest to algorytm rekurencyjny, w pewnym momencie rozmiary macierzy będą nieodpowiednie i mnożenie będzie trzeba zakończyć stosując inny algorytm, a to utrudniałoby analizę złożoności takiego algorytmu. Ponadto jako, że dwie macierze mają różne rozmiary, poniżej staramy się znaleźć zależność czasu działania od łącznej liczby elementów w obu macierzach.

Łączna liczba elementów macierzy	Ilość dodawań	Ilość mnożeń	Czas[s]
45	540	76	0.001800
1025	52800	5776	0.084700
23625	4272000	438976	5.039200
550625	330456000	33362176	382.081000

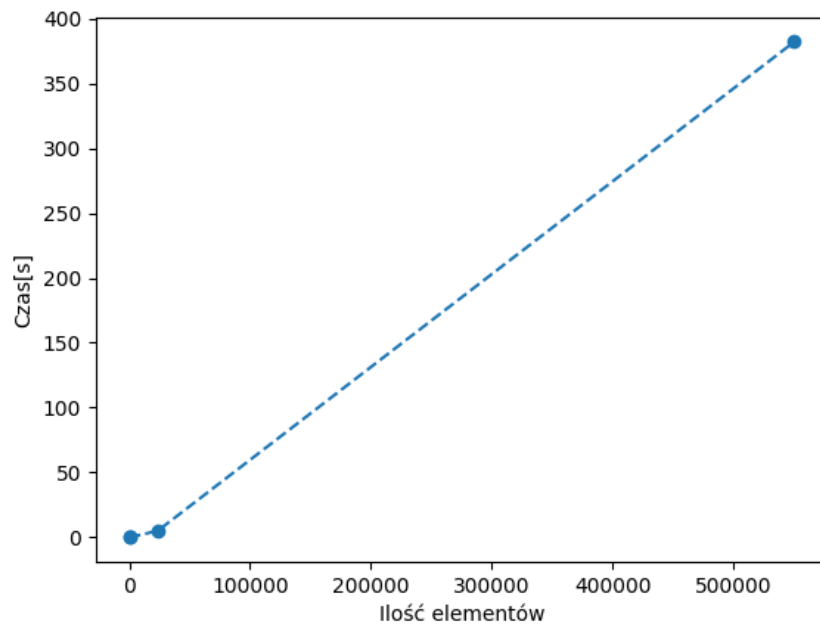
Tabela 3: Wyniki pomiarów dla algorytmu AlphaTensor



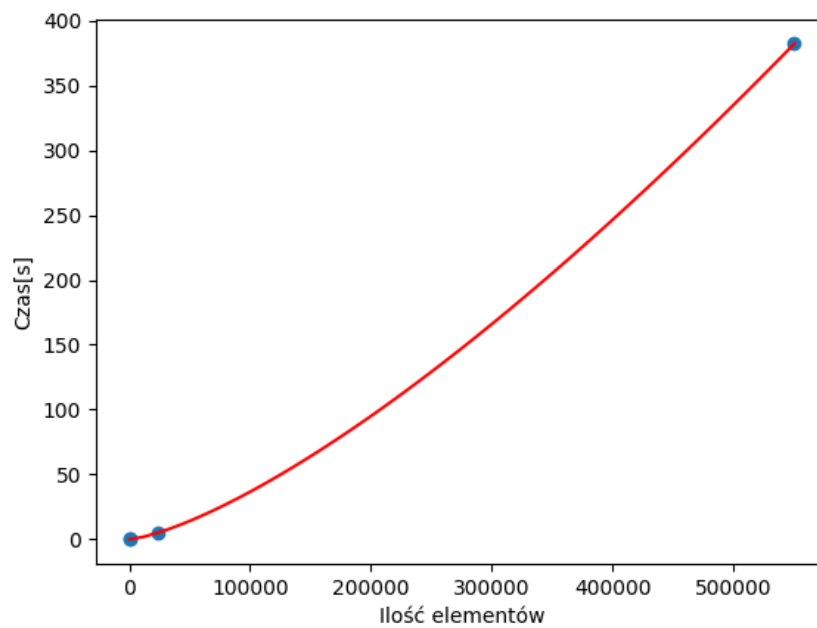
Rysunek 9: Pomiary ilości operacji dodawania



Rysunek 10: Pomiary ilości operacji mnożenia



Rysunek 11: Pomiary czasu



Rysunek 12: Krzywa dopasowana do pomiaru czasu

Krzywa na ostatnim rysunku opisana jest równaniem $y = 4.906 \cdot x^{1.374}$

4 Bibliografia

<https://www.deepmind.com/blog/discovering-novel-algorithms-with-alphatensor>