Algorytmy Macierzowe

Sprawozdanie II

Grupa wtorek 13:00b

Michał Kuszewski i Michał Nożkiewicz

4 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.

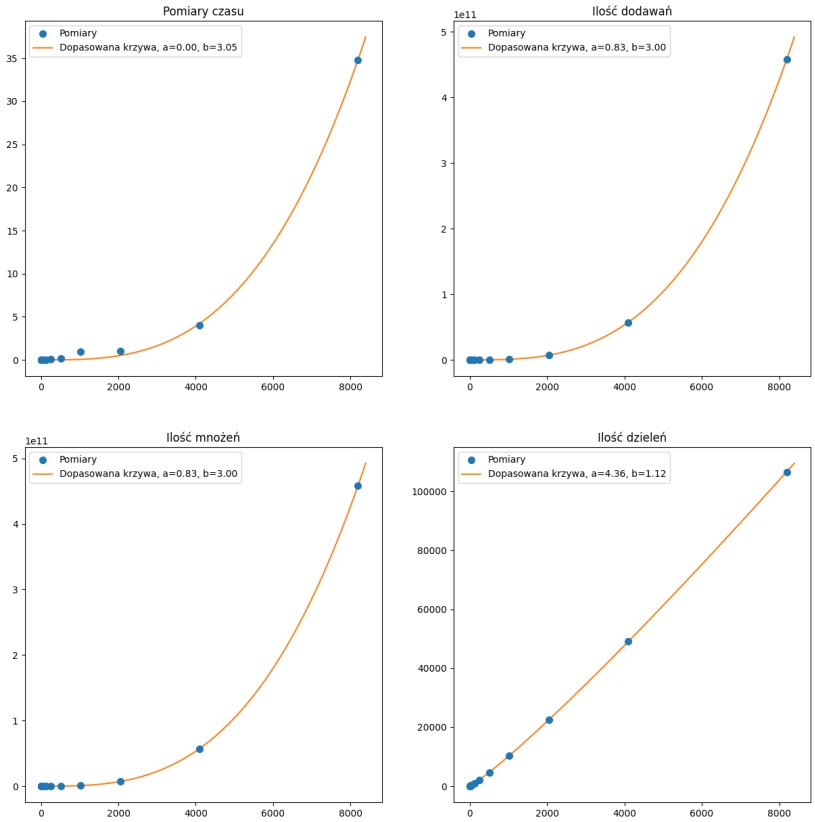
3 Analiza algorytmów

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. Funkcja curve_fit do wyznaczenia optymalnych parametrów stosuje metodę najmniejszych kwadratów.

3.1 Algorytm odwracania macierzy

Wymiar macierzy	Pomiary czasu[s]	Ilość dodawań	Ilość mnożeń	Ilość dzieleń
1	0.000100	0	0	1
2	0.000200	2	6	2
4	0.000200	36	60	4
8	0.000600	392	504	8
16	0.001300	3600	4080	16
32	0.002100	30752	32736	32
64	0.003200	254016	262080	64
128	0.013100	2064512	2097024	128
256	0.011400	16646400	16776960	256
512	0.060700	133693952	134217216	512
1024	0.253100	1071645696	1073740800	1024
2048	0.770300	8581548032	8589932544	2048
4096	3.047200	68685926400	68719472640	4096
8192	20.790900	549621604352	549755805696	8192

Tabela 1: Wyniki pomiarów dla algorytmu Bineta

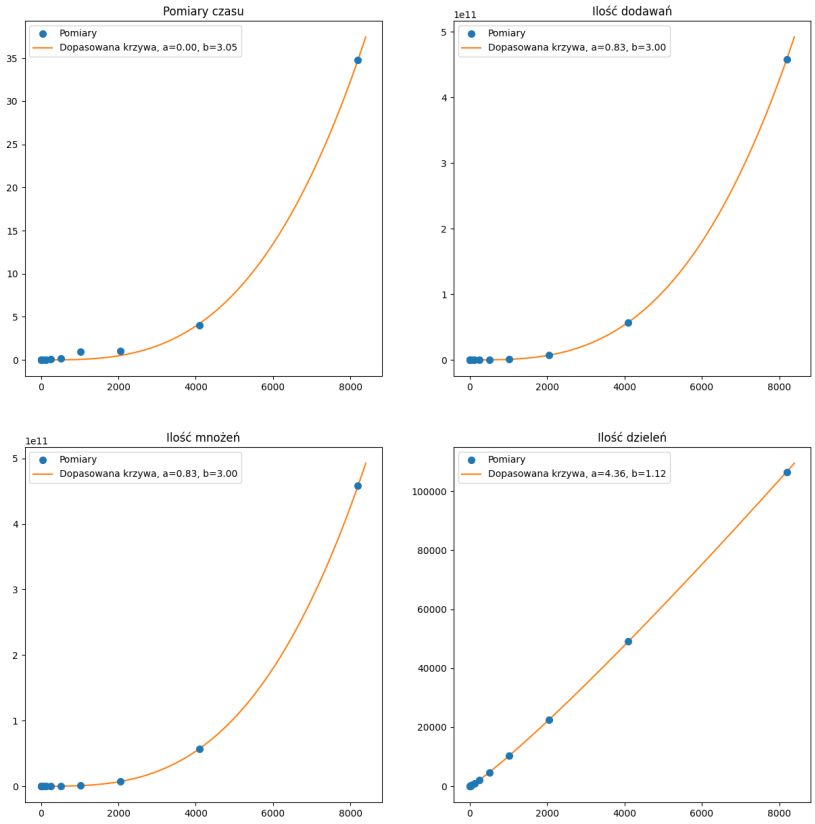


Rysunek 1: Wykres z dopasowanymi krzywymi

3.2 Faktoryzacja LU

Wymiar macierzy	Pomiary czasu[s]	Ilość dodawań	Ilość mnożeń	Ilość dzieleń
1	0.000100	0	0	0
2	0.000100	1	3	2
4	0.000400	22	42	8
8	0.001500	276	396	24
16	0.003200	2744	3336	64
32	0.013100	24464	27120	160
64	0.013800	206688	218016	384
128	0.041400	1699648	1746624	896
256	0.087200	13787008	13978752	2048
512	0.294100	111067392	111843072	4608
1024	0.378400	891651584	894773760	10240
2048	1.103700	7145722880	7158254592	22528
4096	4.104600	57215956992	57266178048	49152
8192	43.683900	457928642560	458129731584	106496

Tabela 2: Wyniki pomiarów dla algorytmu Strassena

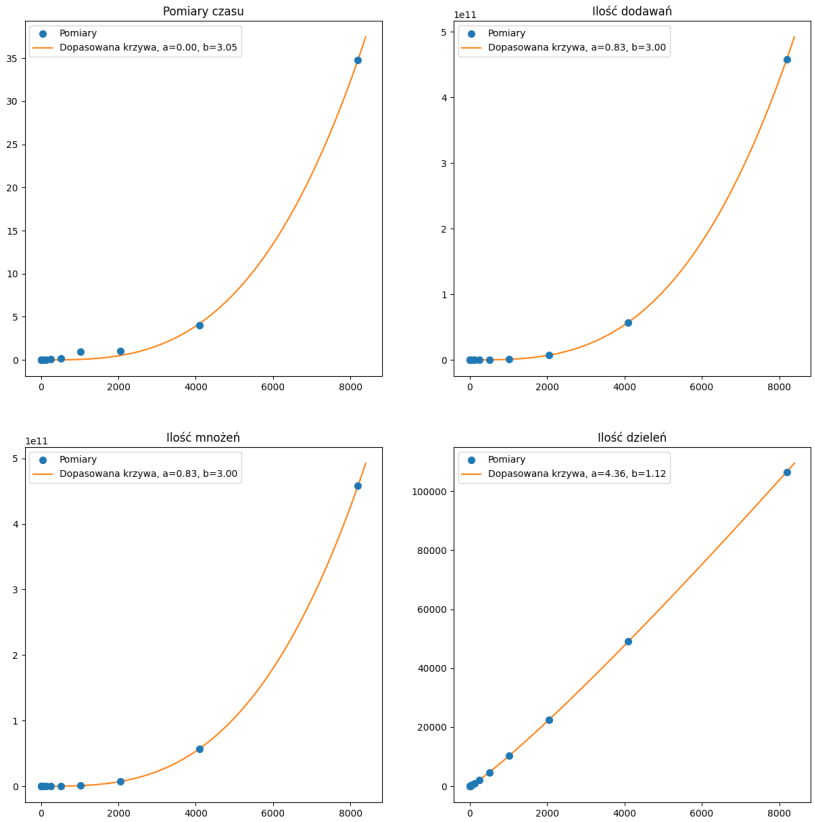


Rysunek 2: Wykres z dopasowanymi krzywymi

3.3 Wyznacznik macierzy

Wymiar macierzy	Pomiary czasu[s]	Ilość dodawań	Ilość mnożeń	Ilość dzieleń
1	0.000000	0	1	0
2	0.000100	1	5	2
4	0.000400	22	46	8
8	0.001600	276	404	24
16	0.002600	2744	3352	64
32	0.006400	24464	27152	160
64	0.014400	206688	218080	384
128	0.042700	1699648	1746752	896
256	0.087200	13787008	13979008	2048
512	0.137200	111067392	111843584	4608
1024	0.980900	891651584	894774784	10240
2048	1.023500	7145722880	7158256640	22528
4096	4.050200	57215956992	57266182144	49152
8192	34.819700	457928642560	458129739776	106496

Tabela 3: Wyniki pomiarów dla algorytmu AlphaTensor



Rysunek 3: Wykres z dopasowanymi krzywymi

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

TODO