Algorytmy macierzowe

Laboratorium 2

Sprawozdanie

Łukasz Stępień, Szymon Urbański

1. Temat zadania

Laboratorium polegało na zaimplementowaniu i przetestowaniu rekurencyjnego algorytmu odwracania, faktoryzacji LU oraz obliczania wyznacznika macierzy. Należało również narysować wykres zależności czasu i ilości obliczeń od rozmiarów macierzy oraz określić złożoność obliczeniową tych algorytmów.

2. Rozwiązanie

2.1. Rekurencyjne odwracanie macierzy

Zaimplementowano rekurencyjne odwracanie macierzy.

- **Pseudokod algorytmu:**

```
inverse(A):
        if A jest w wymiarach 2x2:
                return odwrócone A (według definicji)

        Podziel macierz A na macierze A11, A12, A21, A22 (2)

        A11_inv = inverse(A11) (3)
        S22 = A22 - A21 * A11_inv * A12
        S22_inv = inverse(S22)

        C1 = A11_inv * A12
        C2 = S22_inv * A21 * A11_inv

        B11 = A11_inv + C1 * C2
        B12 = - C1 * S22_inv
        B21 = - C2
        B22 = S22_inv

        Połącz B11, B12, B21, B22 (4)
        Return B
```

- **Złożoność obliczeniowa:**

Niech T(n) oznacza czas procedury *inverse* dla dwóch macierzy *nxn*.

(1) W przypadku bazowym (*n=2*), wykonujemy elementarną operacje odwracania, więc:

$$T(n) = \Theta(1)$$

(2) Zakładamy, że podział wykonuje się w stałym czasie $\Theta(1)$.

(3) Wykonujemy łącznie 2 wywołania rekurencyjne procedury *inverse*. Ponieważ w każdym tym wywołaniu argumentem są macierze $\frac{n}{2}$ x $\frac{n}{2}$, co wnosi $T\left(\frac{n}{2}\right)$ do łącznego

czasu działania, czas tych dwóch wywołań to $2T\left(\frac{n}{2}\right)$. Uwzględniamy 7 mnożeń macierzy zawierających $\frac{n^2}{4}$ elementów. Każde z siedmiu mnożeń wymaga $\Theta(n^{lg7})$, a ich liczba jest stała, łączny czas ich wykonywania to $\Theta(n^{lg7})$. Następnie bierzemy pod uwagę 4 działania addytywne. Każde z nich wymaga $\Theta(n^2)$, a ich liczba jest stała, więc łączny czas ich wykonywania to $\Theta(n^2)$.

(4) Zakładamy, że łączenie wykonuje się w stałym czasie $\Theta(1)$.

Rekurencja opisująca czas działania algorytmu odwracania macierzy przedstawia się równaniem:

$$T(n) = \begin{cases} \Theta(1) & dla\ n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n^{lg7}) & dla\ n > 1 \end{cases}$$

Rozwiązaniem tej rekurencji jest $T(n) = \Theta(n^{lg7})$.

- **Najważniejsze fragmenty kodu:**

```
def inverse(A):  # inversion of matrix with recursion
    n = len(A)
    if n == 2:
        return gauss_inverse(A)

    A11 = [row[:n // 2] for row in A[:n // 2]]
    A12 = [row[n // 2:] for row in A[:n // 2]]
    A21 = [row[:n // 2] for row in A[n // 2:]]
    A22 = [row[n // 2:] for row in A[n // 2:]]

    A11_inv = inverse(A11)
    S22 = sub_matrix(A22, mul_matrix(mul_matrix(A21, A11_inv), A12))
    S22_inv = inverse(S22)

    C1 = mul_matrix(A11_inv, A12)
    C2 = mul_matrix(mul_matrix(S22_inv, A21), A11_inv)

    B = [[0 for _ in range(2)] for _ in range(2)]

    B[0][0] = add_matrix(A11_inv, mul_matrix(C1, C2))
    B[0][1] = neg_matrix(mul_matrix(C1, S22_inv))
    B[1][0] = neg_matrix(C2)
    B[1][1] = S22_inv

    res = []
    for i in range(2):
        for j in range(len(B[i][0])):
            res.append(B[i][0][j] + B[i][1][j])
    return res
```

2.2. Faktoryzacja LU

- **Pseudokod algorytmu:**

```
LU(A):
        if A jest w wymiarach 2x2:
                return sfaktoryzowane A (według definicji)

        Podziel macierz A na macierze A11, A12, A21, A22 (2)

        L11, U11 = LU(A11) (3)
        U11_inv = inverse(U11)
        L21 = A21 * U11_inv
        L11_inv = inverse(L11)
        U12 = L11_inv * A12

        S = A22 - A21 * U11_inv * L11_inv * A12
        L22, U22 = LU(S)

        Połącz L11, L21, L22 (4)
        Połącz U11, U12, U22

        Return L, U
```

- **Złożoność obliczeniowa:**

Niech T(n) oznacza czas procedury *LU* dla dwóch macierzy *nxn*.

(1) W przypadku bazowym (*n=2*), wykonujemy operacje elementarną, więc:
$$T(n) = \Theta(1)$$
(2) Zakładamy, że podział wykonuje się w stałym czasie $\Theta(1)$.

(3) Wykonujemy łącznie 2 wywołania rekurencyjnej procedury *LU(A)*. Ponieważ w każdym tym wywołaniu argumentem są macierze $\frac{n}{2}$ x $\frac{n}{2}$, co wnosi $T\left(\frac{n}{2}\right)$ do łącznego czasu działania, czas tych dwóch wywołań to $2T\left(\frac{n}{2}\right)$. Uwzględniamy pięć mnożeń oraz 2 odwracania macierzy. zawierających $\frac{n^2}{4}$ elementów. Łączny czas ich wykonywania to $\Theta(n^{lg7})$.

(4) Zakładamy, że łączenie wykonuje się w stałym czasie $\Theta(1)$.

Rekurencja opisująca czas działania algorytmu faktoryzacji LU przedstawia się równaniem:

$$T(n) = \begin{cases} \Theta(1) & dla\, n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n^{lg7}) & dla\, n > 1 \end{cases}$$

Rozwiązaniem tej rekurencji jest $T(n) = \Theta(n^{lg7})$.

- **Najważniejsze fragmenty kodu:**

```python
def LU(A):
    n = len(A)
    if n == 2:
        return doolittle_LU(A)

    A11 = [row[:n // 2] for row in A[:n // 2]]
    A12 = [row[n // 2:] for row in A[:n // 2]]
    A21 = [row[:n // 2] for row in A[n // 2:]]
    A22 = [row[n // 2:] for row in A[n // 2:]]

    L11, U11 = LU(A11)
    U11_inv = inverse(U11)
    L21 = mul_matrix(A21, U11_inv)
    L11_inv = inverse(L11)
    U12 = mul_matrix(L11_inv, A12)
    S = sub_matrix(A22, mul_matrix(mul_matrix(
        mul_matrix(A21, U11_inv), L11_inv), A12))
    L22, U22 = LU(S)

    U = [[U11, U12], [
        [[0 for _ in range(n - len(U22))] for _ in range(n - len(U22))], U22]]
    L = [[L11, [[0 for _ in range(n - len(L11))]
                for _ in range(n - len(L11))]], [L21, L22]]

    U_res = []
    for i in range(2):
        for j in range(len(U[i][0])):
            U_res.append(U[i][0][j] + U[i][1][j])

    L_res = []
    for i in range(2):
        for j in range(len(L[i][0])):
            L_res.append(L[i][0][j] + L[i][1][j])

    return [L_res, U_res]
```

2.3. Obliczanie wyznacznika macierzy

Zaimplementowano obliczanie wyznacznika macierzy.

- **Pseudokod algorytmu:**

  determinant(A):

  L, U = LU(A) (1)

  return suma iloczynów liczb występujących na diagonalach macierzy (2)

- **Złożoność obliczeniowa:**

  (1) Koszt faktoryzacji LU to $\Theta(n^{lg7})$.

  (2) Kosz sumowania iloczynów to $\Theta(n)$.

  Zatem złożoność obliczeniowa obliczania wyznacznika macierzy to $\Theta(n^{lg7})$.

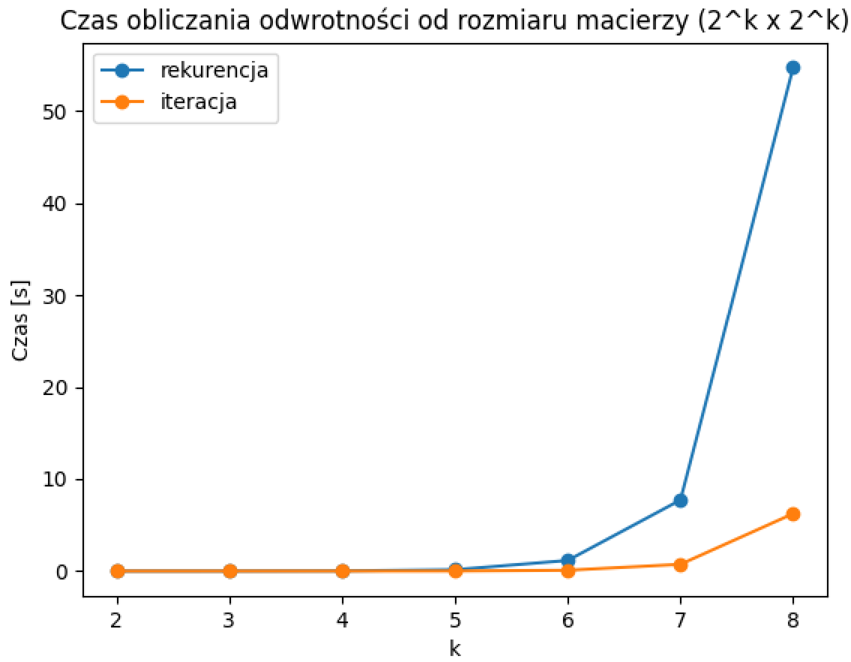
- **Najważniejsze fragmenty kodu:**

```python
def determinant(A):
    if len(A) != len(A[0]):
        print("Error")
        return 0
    global cnt_m
    L, U = LU(A)
    res = 1
    for i in range(len(A)):
        res *= L[i][i] * U[i][i]
        cnt_m += 2
    return res
```

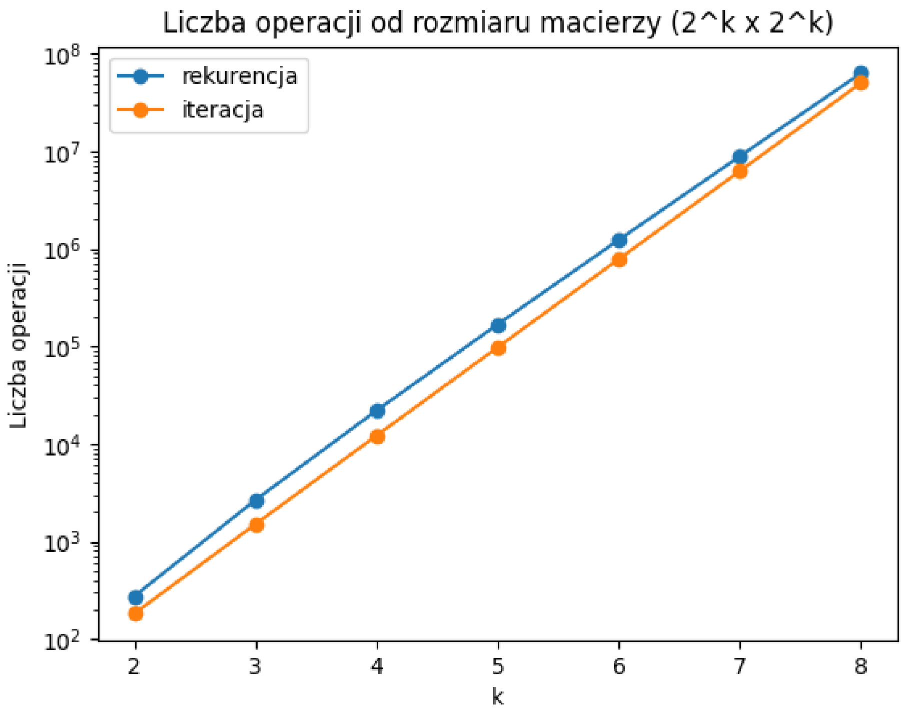
3. Wyniki

3.1. Rekurencyjne odwracanie macierzy

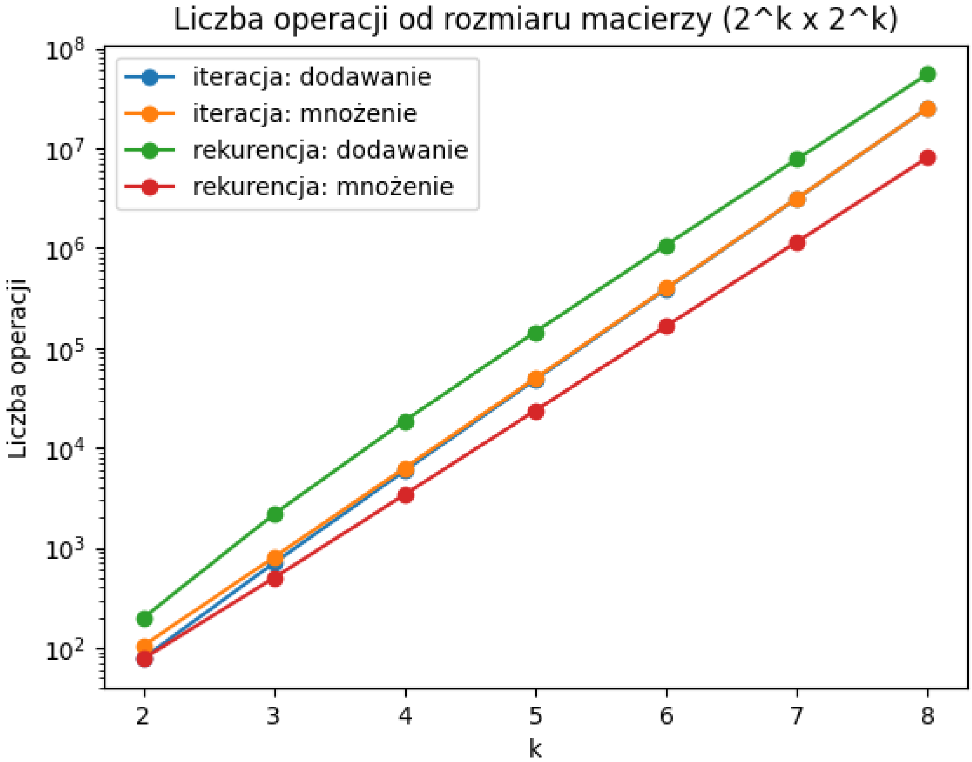
Na poniższym wykresie przedstawiono porównanie czasu działania rekurencyjnego oraz iteracyjnego algorytmu odwracania macierzy w zależności od ilości elementów.



Wygenerowano wykres przedstawiający liczbę wykonanych operacji arytmetycznych w zależności od wielkości macierzy dla obu wariantów algorytmu.

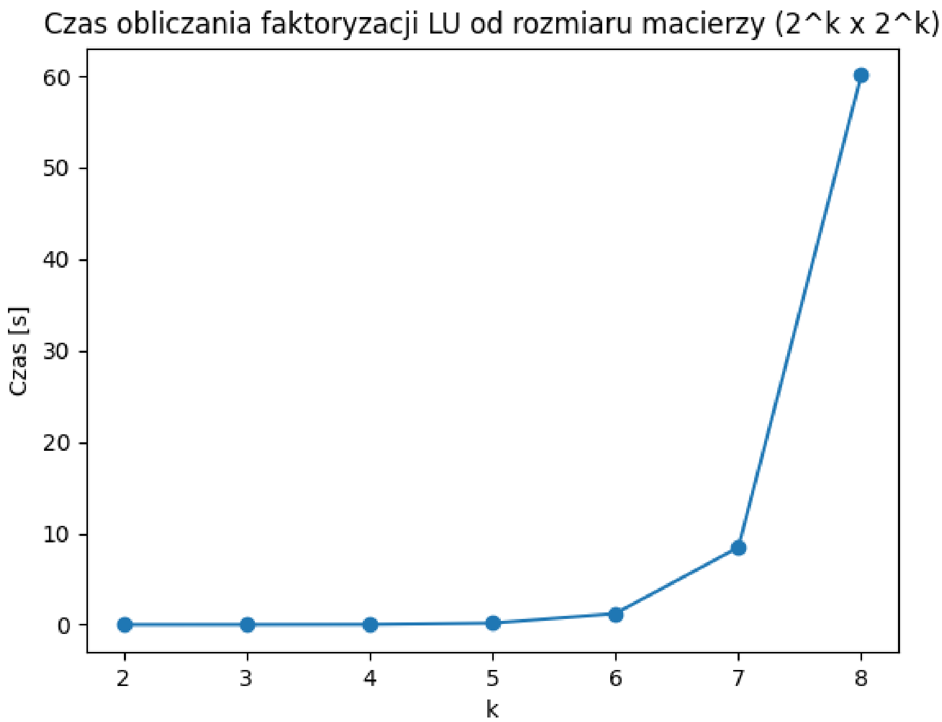


Wygenerowano wykres przedstawiający liczbę wykonanych operacji arytmetycznych z podziałem na dodawanie i odejmowanie w zależności od wielkości macierzy.

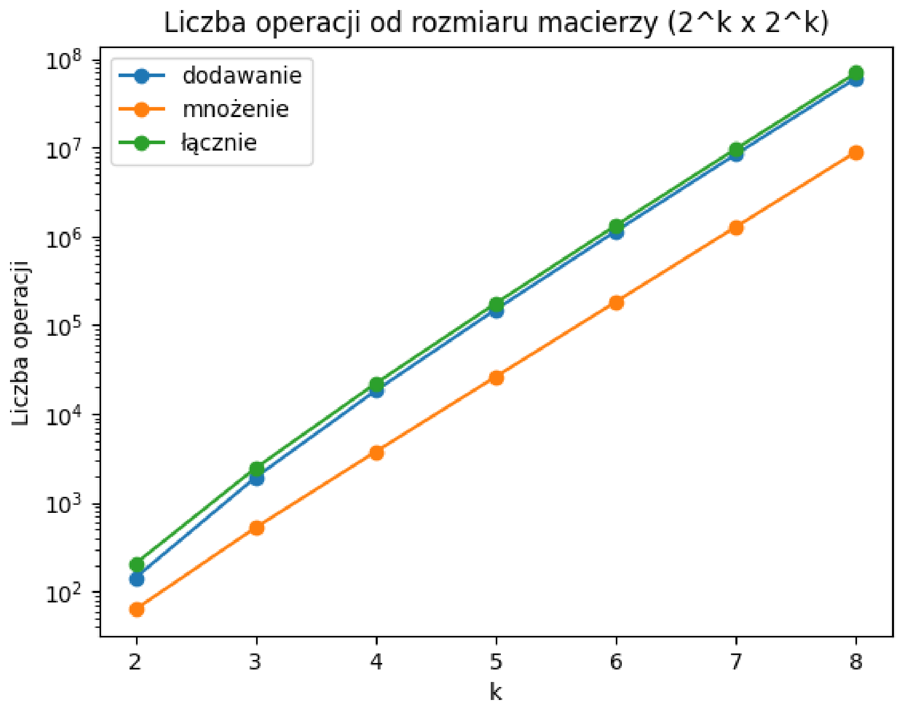


3.2. Faktoryzacja LU

Na poniższym wykresie przedstawiono czas działania algorytmu faktoryzacji LU macierzy w zależności od ilości elementów.

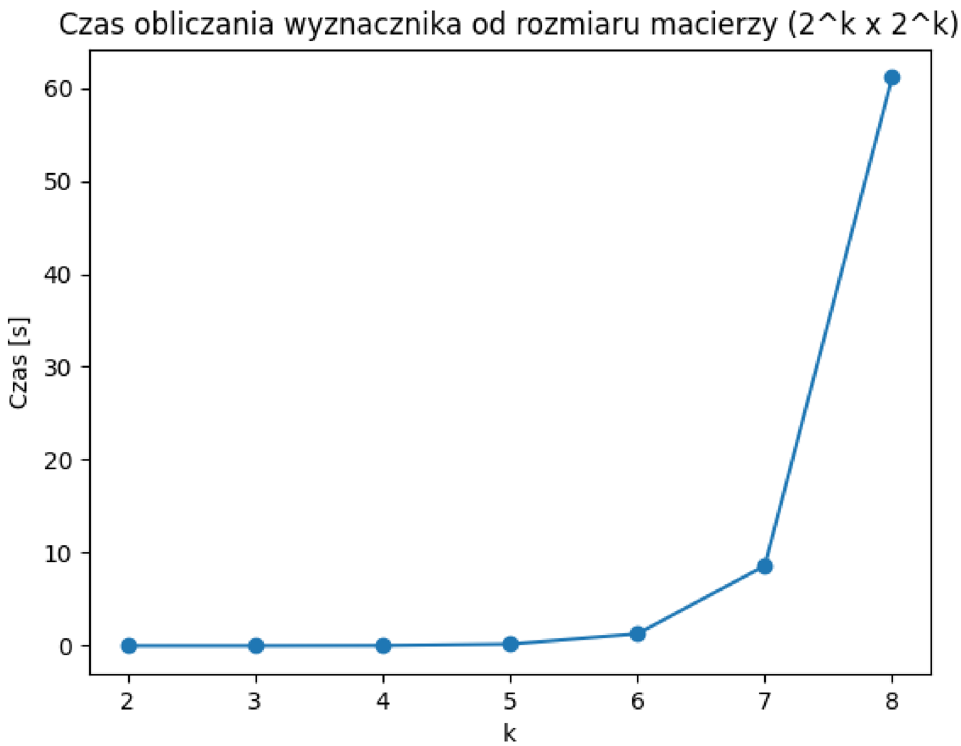


Wygenerowano wykres przedstawiający liczbę wykonanych operacji arytmetycznych w zależności od wielkości macierzy.

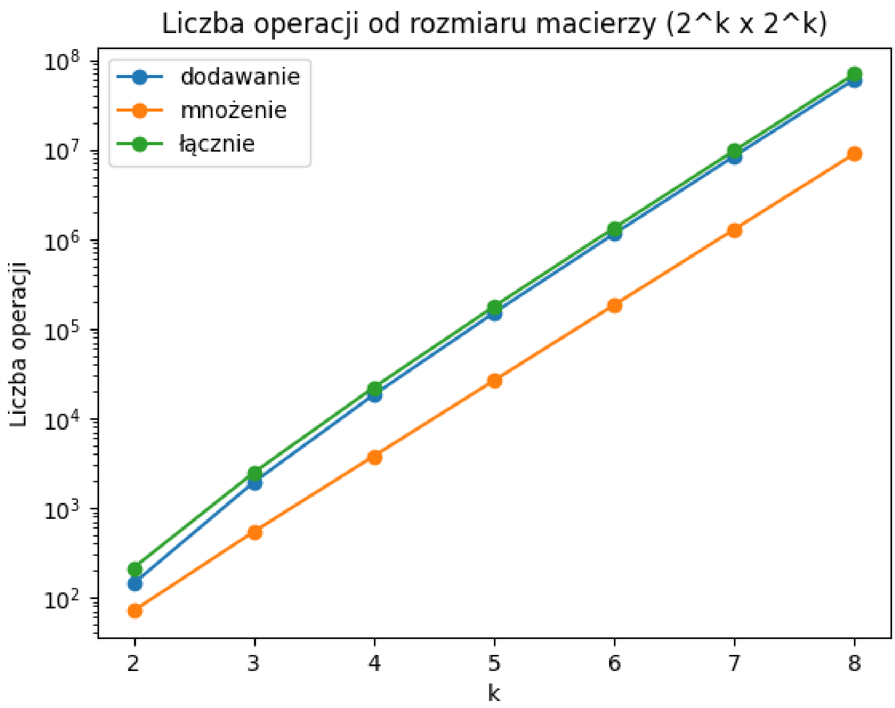


3.3. Obliczanie wyznacznika macierzy

Na poniższym wykresie przedstawiono czas działania algorytmu obliczającego wyznacznik macierzy w zależności od wielkości macierzy.



Wygenerowano wykres przedstawiający liczbę wykonanych operacji arytmetycznych w zależności od wielkości macierzy.



3.4. Testy w MATLAB

Przetestowano działanie algorytmów z tymi zaimplementowanymi w programie MATLAB.

Dla wygenerowanej macierzy 4x4:

$$A = \begin{bmatrix} 0.3996362682016685 & 0.3775324097440138 & 0.2867355288648659 & 0.9335904617261223 \\ 0.8442372135947669 & 0.6695498023698692 & 0.4219842325585101 & 0.4341797318639589 \\ 0.026568629132153628 & 0.6848164865354961 & 0.006902246086230951 & 0.49542134706786245 \\ 0.5269780713781076 & 0.8534918403914511 & 0.5867139799665525 & 0.08457429827957923 \end{bmatrix}$$

zaimplementowane algorytmy zwróciły następujące wyniki:

- odwracanie macierzy:

$$A^{-1}_{rekurencja} = \begin{bmatrix} -0.8765338450388764 & 2.428286254740784 & -0.2518254400791182 & -1.3151671533372182 \\ -0.8687881777468327 & 0.20343962875281996 & 1.414221858831219 & 0.2616314251197229 \\ 1.874993051255954 & -2.4225519429693705 & -1.8459044518565995 & 2.5521677277589014 \\ 1.2218026336803671 & -0.3776865897158288 & 0.10283993464741556 & -0.32667751529510675 \end{bmatrix}$$

$$A^{-1}_{iteracja} = \begin{bmatrix} -0.8765338450389437 & 2.4282862547407595 & -0.251825440079116 & -1.3151671533372231 \\ -0.8687881777467514 & 0.2034396287528648 & 1.4142218588312185 & 0.2616314251197207 \\ 1.8749930512559558 & -2.4225519429693727 & -1.8459044518565995 & 2.5521677277589006 \\ 1.22180263368036 & -0.37768658971583224 & 0.10283993464741538 & -0.32667751529510636 \end{bmatrix}$$

$$A^{-1}_{MATLAB} = \begin{bmatrix} -0.876533845038943 & 2.428286254740761 & -0.251825440079116 & -1.315167153337223 \\ -0.868788177746753 & 0.203439628752865 & 1.414221858831219 & 0.261631425119721 \\ 1.874993051255957 & -2.422551942969374 & -1.845904451856599 & 2.552167727758900 \\ 1.221802633680360 & -0.377686589715832 & 0.102839934647415 & -0.326677515295107 \end{bmatrix}$$

- faktoryzacja LU

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2.112514005282272 & 1 & 0 & 0 \\ 0.066482026898385 & -5.154335859014112 & 1 & 0 \\ 1.3186442605659066 & -2.7787592047393996 & 0.3148056717479142 & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} 0.3996362682016685 & 0.3775324097440138 & 0.2867355288648661 & 0.9335904617261223 \\ 0 & -0.12799270066232526 & -0.18374858798053834 & 0.9335904617261223 \\ 0 & 0 & -0.9592624491277839 & -7.494236825277966 \\ 0 & 0 & 0 & -3.06112282964024 \end{bmatrix}$$

$$L_{MATLAB} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.031470573322662 & 1 & 0 & 0 \\ 0.624206162547884 & 0.656207556469453 & 1 & 0 \\ 0.473369642757176 & 0.091281780169313 & 0.267373392633044 & 1 \end{bmatrix}$$

$$U_{MATLAB} = \begin{bmatrix} 0.844237213594767 & 0.669549802369869 & 0.421984232558510 & 0.434179731863959 \\ 0 & 0.663745370386842 & -0.006377839645509 & 0.481757461981024 \\ 0 & 0 & 0.327494008074824 & -0.502576252940785 \\ 0 & 0 & 0 & 0.818462796227376 \end{bmatrix}$$

- obliczanie wyznacznika

$$\det(A) = -0.15019943469607316$$

$$\det(A)_{MATLAB} = -0.150199434696073$$

4. Wnioski

Wykres przedstawiający porównanie czasu działania rekurencyjnego oraz iteracyjnego algorytmu odwracania macierzy w zależności od ilości elementów pozwala stwierdzić, że o wiele lepiej radzi sobie metoda iteracyjna, pomimo gorszej złożoności obliczeniowej. NA kolejnych wykresach można zaobserwować również większą liczbę działań arytmetycznych w algorytmie rekurencyjnym. Jednakże w algorytmie rekurencyjnym widzimy przeważającą ilość dodawań i odejmowań niż mnożeń i dzieleń, co usprawnia ten algorytm pod względem numerycznym.

Następne algorytmy bazują na rekurencyjnym odwracaniu macierzy, dlatego reszta wykresów jest podobna do tych pierwszych.

Analiza wyników działania zaimplementowanych algorytmów oraz tych zaimplementowanych w MATLAB pozwala wnioskować o poprawności tych pierwszych. Wyniki są praktycznie identyczne z dokładnością do szumu numerycznego.

5. Bibliografia

- wykład z przedmiotu „Algorytmy macierzowe” przygotowany przez prof. dr hab. Macieja Paszyńskiego
- https://en.wikipedia.org/wiki/LU_decomposition
- Thomas H. Cormen - „Wprowadzenie do algorytmów”