

Akademia Górniczo-Hutnicza

WYDZIAŁ INFORMATYKI



ALGORYTMY MACIERZOWE

Ćwiczenie 2

Rekurencyjne odwracanie macierzy, faktoryzacja LU, eliminacja Gaussa i obliczanie wyznacznika

Jakub Szewczyk i Albert Arnautov

Kraków, 4 listopada 2025

Spis treści

1 Wstęp	3
2 Pseudokod	3
2.1 Rekurencyjne odwracanie macierzy	3
2.2 Rekurencyjna eliminacja Gaussa	4
2.3 Rekurencyjna LU faktoryzacja oraz liczenie wyznacznika	5
3 Ważne fragmenty kodu	6
3.1 Generowanie losowej macierzy rozmiaru n	6
3.2 Implementacja rekurencyjnego odwracania macierzy	6
3.3 Implementacja rekurencyjnej eliminacji Gauusa	7
3.4 Implementacja rekurencyjnej faktoryzacji LU	7
3.5 Implementacja rekurencyjnego obliczania wyznacznika	8
4 Wykresy i porównanie metod	9
4.1 Rekurencyjne odwracanie macierzy	9
4.2 Rekurencyjna eliminacja Gaussa	10
4.3 Rekurencyjna LU faktoryzacja	11

1 Wstęp

Celem ćwiczenia było zaimplementowanie rekurencyjnych algorytmów odwracania macierzy, dekompozycji LU, eliminacji Gaussa i obliczania wyznacznika

2 Pseudokod

Poniżej zostały przedstawione pseudokody poszczególnych funkcji rekurencyjnych

2.1 Rekurencyjne odwracanie macierzy

Algorytm 1 Rekurencyjne odwracanie macierzy

```
1: procedure RECURSIVEINVERSE( $A$ )
2:    $n \leftarrow$  number of rows in  $A$ 

3:   if  $n = 1$  then
4:      $\text{inv} \leftarrow$  create  $1 \times 1$  zero matrix
5:      $\text{inv}[0][0] \leftarrow 1.0/A[0][0]$ 
6:     return  $\text{inv}$ 
7:   end if

8:    $k \leftarrow \lfloor n/2 \rfloor$ 

9:    $A_{11} \leftarrow A[0 : k, 0 : k]$ 
10:   $A_{12} \leftarrow A[0 : k, k : n]$ 
11:   $A_{21} \leftarrow A[k : n, 0 : k]$ 
12:   $A_{22} \leftarrow A[k : n, k : n]$ 

13:   $A_{11}^{-1} \leftarrow \text{RecursiveInverse}(A_{11})$ 
14:   $S_{22} \leftarrow A_{22} - A_{21} \cdot A_{11}^{-1} \cdot A_{12}$ 
15:   $S_{22}^{-1} \leftarrow \text{RecursiveInverse}(S_{22})$ 

16:   $B_{11} \leftarrow A_{11}^{-1} + A_{11}^{-1} \cdot A_{12} \cdot S_{22}^{-1} \cdot A_{21} \cdot A_{11}^{-1}$ 
17:   $B_{12} \leftarrow -A_{11}^{-1} \cdot A_{12} \cdot S_{22}^{-1}$ 
18:   $B_{21} \leftarrow -S_{22}^{-1} \cdot A_{21} \cdot A_{11}^{-1}$ 
19:   $B_{22} \leftarrow S_{22}^{-1}$ 

20:  result  $\leftarrow$  block matrix  $\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$ 
21:  return result
22: end procedure
```

2.2 Rekurencyjna eliminacja Gaussa

Algorytm 2 Rekurencyjna eliminacja Gaussa

```

1: procedure RECURSIVEGAUSSIANELIMINATION( $A, b$ )
2:    $n \leftarrow$  number of rows in  $A$ 

3:   if  $n = 1$  then
4:     return  $b/A[0, 0]$ 
5:   end if

6:    $k \leftarrow \lfloor n/2 \rfloor$ 

7:   Partition  $A$  and  $b$  into blocks:
8:      $A_{11} = A[1 : k, 1 : k]$ 
9:      $A_{12} = A[1 : k, k + 1 : n]$ 
10:     $A_{21} = A[k + 1 : n, 1 : k]$ 
11:     $A_{22} = A[k + 1 : n, k + 1 : n]$ 
12:     $b_1 = b[1 : k]$ 
13:     $b_2 = b[k + 1 : n]$ 

14:    $[L_{11}, U_{11}] \leftarrow$  LUFactorization( $A_{11}$ )
15:    $L_{11}^{-1} \leftarrow$  RecursiveInverse( $L_{11}$ )
16:    $U_{11}^{-1} \leftarrow$  RecursiveInverse( $U_{11}$ )

17:    $S \leftarrow A_{22} - A_{21} \cdot U_{11}^{-1} \cdot L_{11}^{-1} \cdot A_{12}$ 
18:    $[L_s, U_s] \leftarrow$  LUFactorization( $S$ )
19:    $L_s^{-1} \leftarrow$  RecursiveInverse( $L_s$ )

20:    $RHS_1 \leftarrow L_{11}^{-1} \cdot b_1$ 
21:    $T \leftarrow L_s^{-1} \cdot A_{21} \cdot U_{11}^{-1} \cdot L_{11}^{-1} \cdot b_1$ 
22:    $RHS_2 \leftarrow L_s^{-1} \cdot b_2 - T$ 

23:    $RHS \leftarrow \begin{bmatrix} RHS_1 \\ RHS_2 \end{bmatrix}$ 

24:    $U_{12} \leftarrow L_{11}^{-1} \cdot A_{12}$ 
25:    $U_{22} \leftarrow U_s$ 

26:   Construct upper triangular matrix:
27:    $U \leftarrow \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$ 

28:    $U^{-1} \leftarrow$  RecursiveInverse( $U$ )
29:    $x \leftarrow U^{-1} \cdot RHS$ 

30:   return  $x$ 
31: end procedure

```

2.3 Rekurencyjna LU faktoryzacja oraz liczenie wyznacznika

Algorytm 3 Rekurencyjna faktoryzacja LU

```

1: procedure RECURSIVELUFACTORIZATION( $A$ )
2:    $n \leftarrow$  number of rows in  $A$ 

3:   if  $n = 1$  then
4:     return ( $I_1, A$ )
5:   end if

6:    $k \leftarrow \lfloor n/2 \rfloor$ 

7:   Partition  $A$  into blocks:
8:    $A_{11} = A[1 : k, 1 : k]$ 
9:    $A_{12} = A[1 : k, k + 1 : n]$ 
10:   $A_{21} = A[k + 1 : n, 1 : k]$ 
11:   $A_{22} = A[k + 1 : n, k + 1 : n]$ 

12:   $[L_{11}, U_{11}] \leftarrow$  RecursiveLUFactorization( $A_{11}$ )
13:   $U_{11}^{-1} \leftarrow$  RecursiveInverse( $U_{11}$ )
14:   $L_{21} \leftarrow A_{21} \cdot U_{11}^{-1}$ 
15:   $L_{11}^{-1} \leftarrow$  RecursiveInverse( $L_{11}$ )
16:   $U_{12} \leftarrow L_{11}^{-1} \cdot A_{12}$ 

17:   $S \leftarrow A_{22} - A_{21} \cdot U_{11}^{-1} \cdot L_{11}^{-1} \cdot A_{12}$ 

18:   $[L_s, U_s] \leftarrow$  RecursiveLUFactorization( $S$ )
19:   $L_{22} \leftarrow L_s$ 
20:   $U_{22} \leftarrow U_s$ 

21:  Construct lower triangular matrix:
22:   $L \leftarrow \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}$ 

23:  Construct upper triangular matrix:
24:   $U \leftarrow \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$ 

25:  return ( $L, U$ )
26: end procedure

```

Algorytm 4 Rekurencyjne obliczanie wyznacznika

```

1: procedure DETERMINANT( $A$ )
2:    $[L, U] \leftarrow$  LUFactorization( $A$ )
3:    $det_U \leftarrow 1$ 
4:    $n \leftarrow$  number of rows in  $U$ 

5:   for  $i \leftarrow 0$  to  $n - 1$  do
6:      $det_U \leftarrow det_U \times U[i, i] \times L[i, i]$ 
7:   end for

8:   return  $det_U$ 
9: end procedure

```

3 Ważne fragmenty kodu

3.1 Generowanie losowej macierzy rozmiaru n

```
1 from random import uniform
2
3 def randomize_matrix(n):
4     return [[uniform(1e-8, 1) for _ in range(n)] for _ in range(n)]
```

3.2 Implementacja rekurencyjnego odwracania macierzy

```
1 def recursive_inverse(A, mat_mul):
2     n = A.shape[0]
3
4     if n == 1:
5         common.counter_mul += 1
6         inv = np.zeros((1, 1))
7         inv[0][0] = 1.0 / A[0][0]
8         return inv
9
10    k = n // 2
11
12    A11 = A[:k, :k]
13    A12 = A[:k, k:]
14    A21 = A[k:, :k]
15    A22 = A[k:, k:]
16
17    A11_inv = recursive_inverse(A11, mat_mul)
18    S22 = A22 - mat_mul(mat_mul(A21, A11_inv), A12)
19    S22_inv = recursive_inverse(S22, mat_mul)
20
21    B11 = A11_inv + mat_mul(mat_mul(mat_mul(A11_inv, A12), S22_inv), A21), A11_inv)
22    B12 = -mat_mul(mat_mul(A11_inv, A12), S22_inv)
23    B21 = -mat_mul(mat_mul(S22_inv, A21), A11_inv)
24    B22 = S22_inv
25
26    result = np.block([
27        [B11, B12],
28        [B21, B22]
29    ])
30
31    return result
```

3.3 Implementacja rekurencyjnej eliminacji Gauusa

```
1 def gaussian_elimination(A, b, mat_mul):
2     n = A.shape[0]
3
4     if n == 1:
5         return b / A[0, 0]
6
7     k = n // 2
8
9     A11 = A[:k, :k]
10    A12 = A[:k, k:]
11    A21 = A[k:, :k]
12    A22 = A[k:, k:]
13
14    b1 = b[:k]
15    b2 = b[k:]
16
17    L11, U11 = LU_factorization(A11, mat_mul)
18    L11_inv = recursive_inverse(L11, mat_mul)
19    U11_inv = recursive_inverse(U11, mat_mul)
20    S = A22 - mat_mul(mat_mul(A21, U11_inv), L11_inv), A12)
21    Ls, Us = LU_factorization(S, mat_mul)
22    Ls_inv = recursive_inverse(Ls, mat_mul)
23
24    RHS1 = mat_mul(L11_inv, b1)
25    RHS2 = mat_mul(Ls_inv, b2) - mat_mul(mat_mul(mat_mul(Ls_inv, A21), U11_inv),
26        ↪ L11_inv), b1)
27    common.counter_sub += 1
28    RHS = np.vstack([RHS1, RHS2])
29
30    U12 = mat_mul(L11_inv, A12)
31    U22 = Us
32
33    bottom_left_zero = np.zeros((n - k, k), dtype=A.dtype)
34
35    U = np.block([
36        [U11, U12],
37        [bottom_left_zero, U22]
38    ])
39
40    U_inv = recursive_inverse(U, mat_mul)
41
42    x = mat_mul(U_inv, RHS)
43
44    return x
```

3.4 Implementacja rekurencyjnej faktoryzacji LU

```
1 def LU_factorization(A, mat_mul):
2     n = A.shape[0]
3
4     if n == 1:
5         return np.eye(1), A.copy()
6
7     k = n // 2
8
9     A11 = A[:k, :k]
10    A12 = A[:k, k:]
11    A21 = A[k:, :k]
```

```

12     A22 = A[k:, k:]
13
14     L11, U11 = LU_factorization(A11, mat_mul)
15     U11_inv = recursive_inverse(U11, mat_mul)
16     L21 = mat_mul(A21, U11_inv)
17     L11_inv = recursive_inverse(L11, mat_mul)
18     U12 = mat_mul(L11_inv, A12)
19     L22 = S = A22 - mat_mul(mat_mul(mat_mul(A21, U11_inv), L11_inv), A12)
20     common.counter_sub += 1
21     Ls, Us = LU_factorization(S, mat_mul)
22     U22 = Us
23     L22 = Ls
24
25     top_right_zero = np.zeros((k, n - k), dtype=A.dtype)
26     bottom_left_zero = np.zeros((n - k, k), dtype=A.dtype)
27
28     L = np.block([
29         [L11, top_right_zero],
30         [L21, L22]
31     ])
32
33     U = np.block([
34         [U11, U12],
35         [bottom_left_zero, U22]
36     ])
37
38     return L, U

```

3.5 Implementacja rekurencyjnego obliczania wyznacznika

```

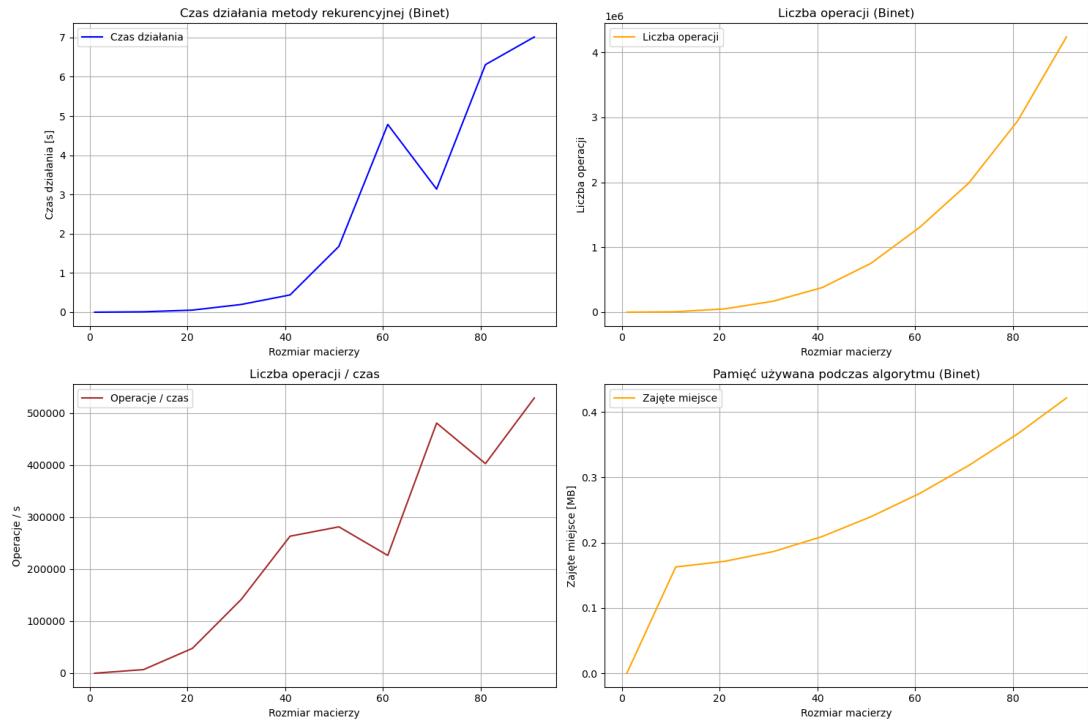
1 def determinant(A, mat_mul):
2     L, U = LU_factorization(A, mat_mul)
3     det_U = 1
4     for i in range(len(U)):
5         det_U *= U[i][i] * L[i][i]
6
7     return det_U

```

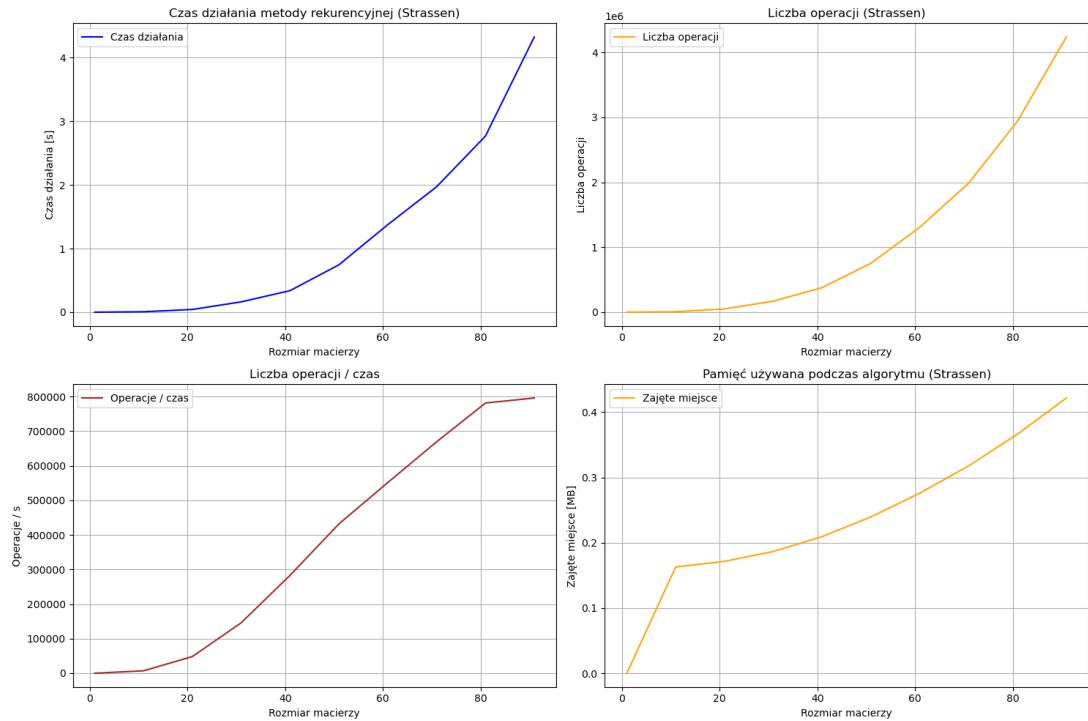
4 Wykresy i porównanie metod

W tym rozdziale przedstawiono wykresy czasów działania zaimplementowanych metod rekurencyjnych, liczbę operacji zmiennoprzecinkowych, liczbę flopsów podzieloną przez czas działania w sekundach oraz użycie pamięci w zależności od rozmiaru macierzy.

4.1 Rekurencyjne odwracanie macierzy

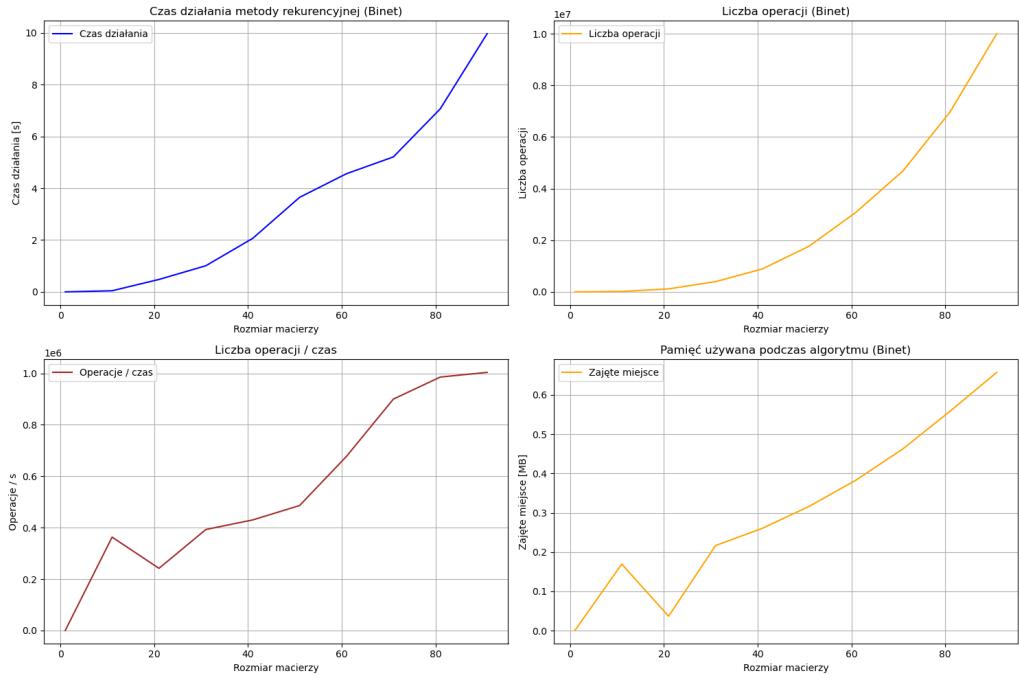


Rysunek 1: Wykresy działania algorytmu w zależności od rozmiaru macierzy - mnożenie macierzy metodą Bineta

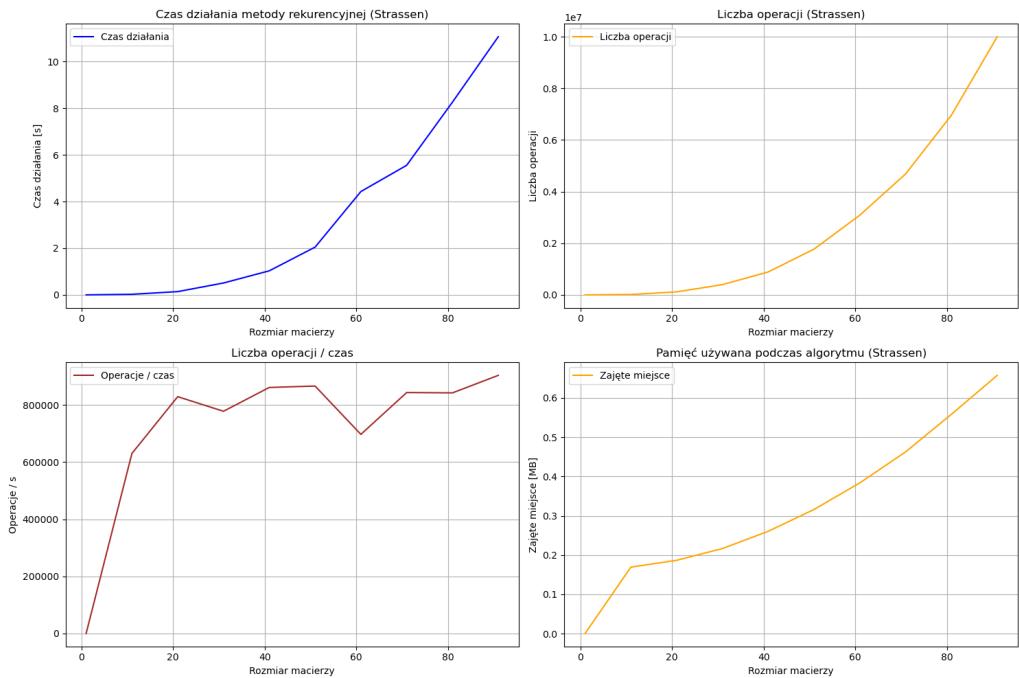


Rysunek 2: Wykresy działania algorytmu w zależności od rozmiaru macierzy - mnożenie macierzy metodą Strassena

4.2 Rekurencyjna eliminacja Gaussa

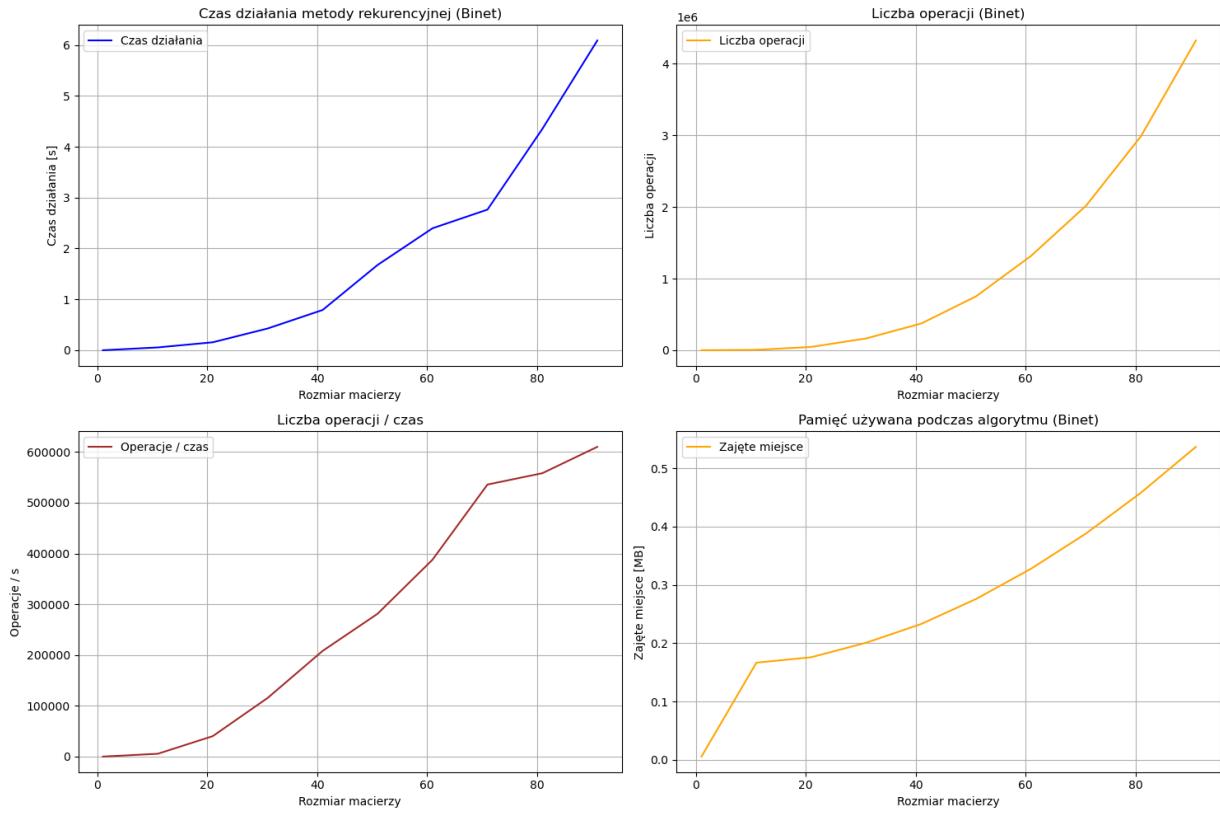


Rysunek 3: Wykresy działania algorytmu w zależności od rozmiaru macierzy - mnożenie macierzy metodą Bineta

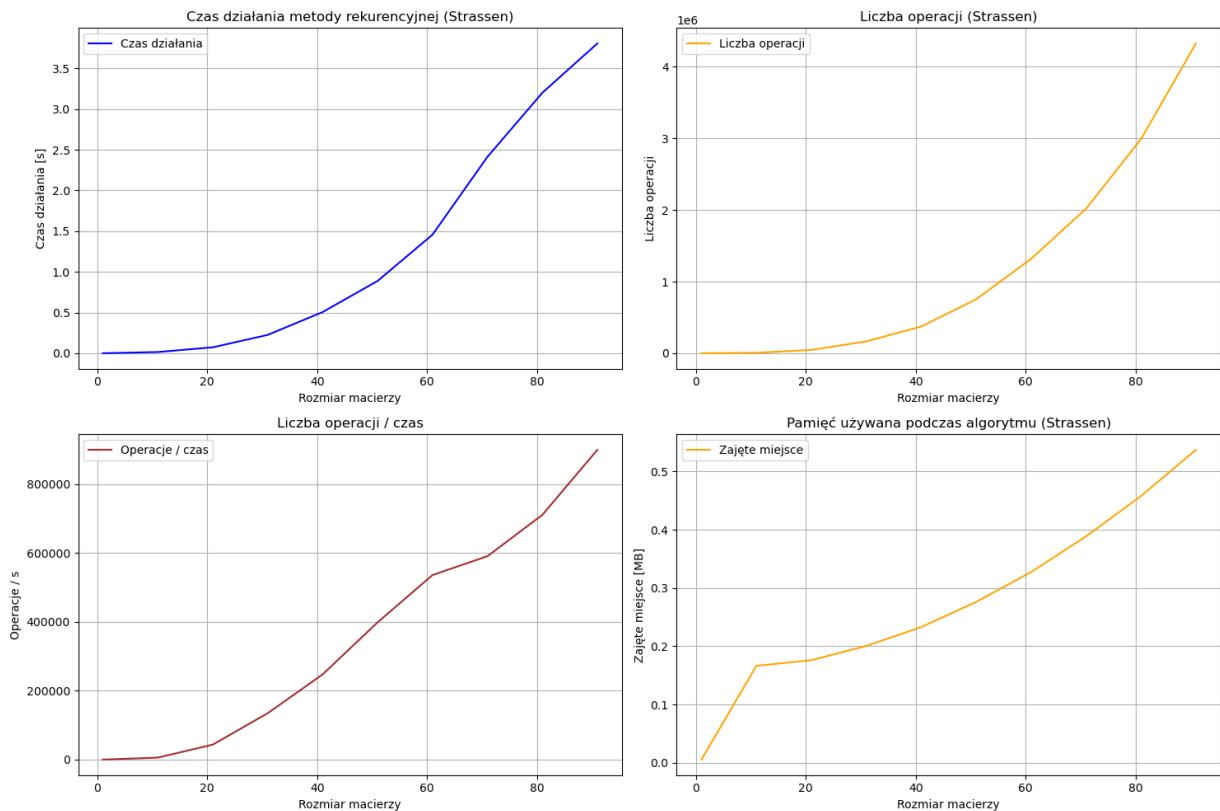


Rysunek 4: Wykresy działania algorytmu w zależności od rozmiaru macierzy - mnożenie macierzy metodą Strassena

4.3 Rekurencyjna LU faktoryzacja



Rysunek 5: Wykresy działania algorytmu w zależności od rozmiaru macierzy - mnożenie macierzy metodą Bineta



Rysunek 6: Wykresy działania algorytmu w zależności od rozmiaru macierzy - mnożenie macierzy metodą Strassena