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

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

procedure RecursiveInverse(A)
    $n \leftarrow$ number of rows in A

    if $n = 1$ **then**
        inv $\leftarrow$ create 1×1 zero matrix
        inv[0][0] $\leftarrow 1.0/A[0][0]$
        return inv
    end if

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

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

    $A_{11}^{-1} \leftarrow$ RecursiveInverse(A_{11})
    $S_{22} \leftarrow A_{22} - A_{21} \cdot A_{11}^{-1} \cdot A_{12}$
    $S_{22}^{-1} \leftarrow$ RecursiveInverse(S_{22})

    $B_{11} \leftarrow A_{11}^{-1} + A_{11}^{-1} \cdot A_{12} \cdot S_{22}^{-1} \cdot A_{21} \cdot A_{11}^{-1}$
    $B_{12} \leftarrow -A_{11}^{-1} \cdot A_{12} \cdot S_{22}^{-1}$
    $B_{21} \leftarrow -S_{22}^{-1} \cdot A_{21}) \cdot A_{11}^{-1})$
    $B_{22} \leftarrow S_{22}^{-1}$

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

2.2 Rekurencyjna eliminacja Gaussa

Algorytm 2 Rekurencyjna eliminacja Gaussa

procedure RecursiveGaussianElimination(A, b)
  $n \leftarrow$ number of rows in A

  if $n = 1$ **then**
    return $b/A[0,0]$
  end if

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

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

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

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

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

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

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

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

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

  return x
end procedure

2.3 Rekurencyjna LU faktoryzacja oraz liczenie wyznacznika

Algorytm 3 Rekurencyjna faktoryzacja LU

procedure RecursiveLUFactorization(A)
  $n \leftarrow$ number of rows in A

  if $n = 1$ **then**
    return (I_1, A)
  end if

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

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

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

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

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

  Construct lower triangular matrix:
    $L \leftarrow \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}$
  Construct upper triangular matrix:
    $U \leftarrow \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$
  return (L, U)
end procedure

Algorytm 4 Rekurencyjne obliczanie wyznacznika

procedure Determinant(A)
  $[L, U] \leftarrow$ LUFactorization(A)
  $det_U \leftarrow 1$
  $n \leftarrow$ number of rows in U

  for $i \leftarrow 0$ to $n - 1$ **do**
    $det_U \leftarrow det_U \times U[i, i] \times L[i, i]$
  end for

  return det_U
end procedure

3 Ważne fragmenty kodu

3.1 Generowanie losowej macierzy rozmiaru n

```python
from random import uniform

def randomize_matrix(n):
    return [[uniform(1e-8, 1) for _ in range(n)] for _ in range(n)]
```

3.2 Implementacja rekurencyjnego odwracania macierzy

```python
def recursive_inverse(A, mat_mul):
    n = A.shape[0]

    if n == 1:
        common.counter_mul += 1
        inv = np.zeros((1, 1))
        inv[0][0] = 1.0 / A[0][0]
        return inv

    k = n // 2

    A11 = A[:k, :k]
    A12 = A[:k, k:]
    A21 = A[k:, :k]
    A22 = A[k:, k:]

    A11_inv = recursive_inverse(A11, mat_mul)
    S22 = A22 - mat_mul(mat_mul(A21, A11_inv), A12)
    S22_inv = recursive_inverse(S22, mat_mul)

    B11 = A11_inv + mat_mul(mat_mul(mat_mul(mat_mul(A11_inv, A12), S22_inv), A21), A11_inv)
    B12 = -mat_mul(mat_mul(A11_inv, A12), S22_inv)
    B21 = -mat_mul(mat_mul(S22_inv, A21), A11_inv)
    B22 = S22_inv

    result = np.block([
        [B11, B12],
        [B21, B22]
    ])

    return result
```

3.3 Implementacja rekurencyjnej eliminacji Gauusa

```python
def gaussian_elimination(A, b, mat_mul):
    n = A.shape[0]

    if n == 1:
        return b / A[0, 0]

    k = n // 2

    A11 = A[:k, :k]
    A12 = A[:k, k:]
    A21 = A[k:, :k]
    A22 = A[k:, k:]

    b1 = b[:k]
    b2 = b[k:]

    L11, U11 = LU_factorization(A11, mat_mul)
    L11_inv = recursive_inverse(L11, mat_mul)
    U11_inv = recursive_inverse(U11, mat_mul)
    S = A22 - mat_mul(mat_mul(mat_mul(A21, U11_inv), L11_inv), A12)
    Ls, Us = LU_factorization(S, mat_mul)
    Ls_inv = recursive_inverse(Ls, mat_mul)

    RHS1 = mat_mul(L11_inv, b1)
    RHS2 = mat_mul(Ls_inv, b2) - mat_mul(mat_mul(mat_mul(mat_mul(Ls_inv, A21), U11_inv),
    ↪  L11_inv), b1)
    common.counter_sub += 1
    RHS = np.vstack([RHS1, RHS2])

    U12 = mat_mul(L11_inv, A12)
    U22 = Us

    bottom_left_zero = np.zeros((n - k, k), dtype=A.dtype)

    U = np.block([
        [U11, U12],
        [bottom_left_zero, U22]
    ])

    U_inv = recursive_inverse(U, mat_mul)

    x = mat_mul(U_inv, RHS)

    return x
```

3.4 Implementacja rekurencyjnej faktoryzacji LU

```python
def LU_factorization(A, mat_mul):
    n = A.shape[0]

    if n == 1:
        return np.eye(1), A.copy()

    k = n // 2

    A11 = A[:k, :k]
    A12 = A[:k, k:]
    A21 = A[k:, :k]
    A22 = A[k:, k:]

    L11, U11 = LU_factorization(A11, mat_mul)
    U11_inv = recursive_inverse(U11, mat_mul)
    L21 = mat_mul(A21, U11_inv)
    L11_inv = recursive_inverse(L11, mat_mul)
    U12 = mat_mul(L11_inv, A12)
    L22 = S = A22 - mat_mul(mat_mul(mat_mul(A21, U11_inv), L11_inv), A12)
    common.counter_sub += 1
    Ls, Us = LU_factorization(S, mat_mul)
    U22 = Us
    L22 = Ls

    top_right_zero = np.zeros((k, n - k), dtype=A.dtype)
    bottom_left_zero = np.zeros((n - k, k), dtype=A.dtype)

    L = np.block([
        [L11, top_right_zero],
        [L21, L22]
    ])

    U = np.block([
        [U11, U12],
        [bottom_left_zero, U22]
    ])

    return L, U
```

3.5 Implementacja rekurencyjnego obliczania wyznacznika

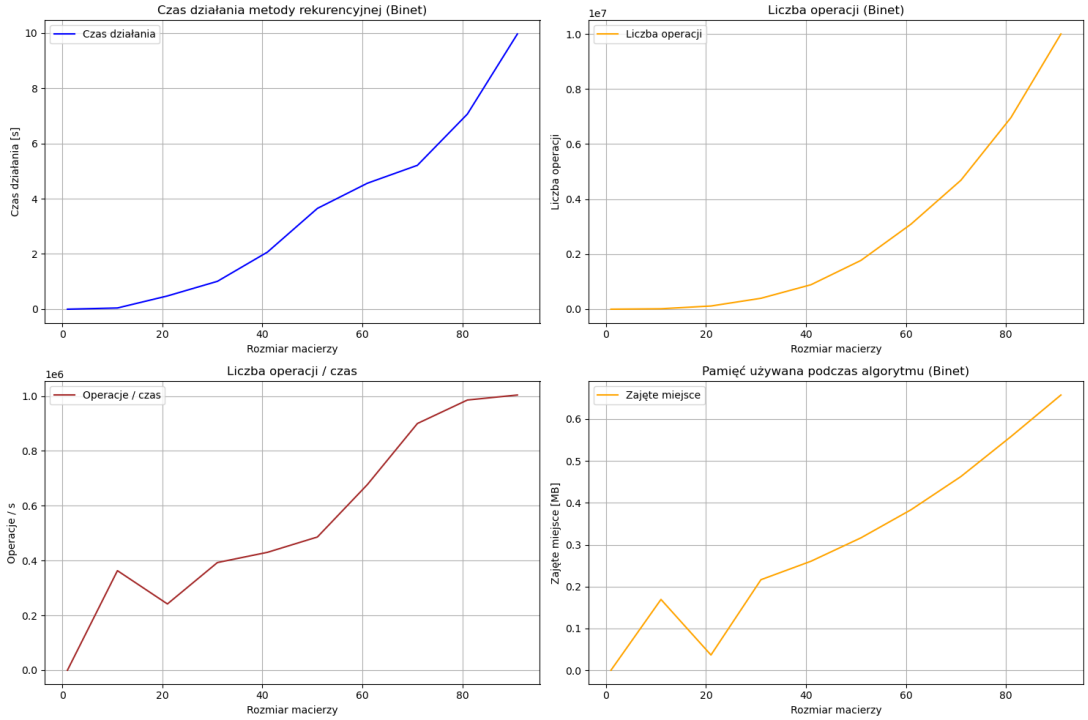
```python
def determinant(A, mat_mul):
    L, U = LU_factorization(A, mat_mul)
    det_U = 1
    for i in range(len(U)):
        det_U *= U[i][i]  * L[i][i]

    return det_U
```

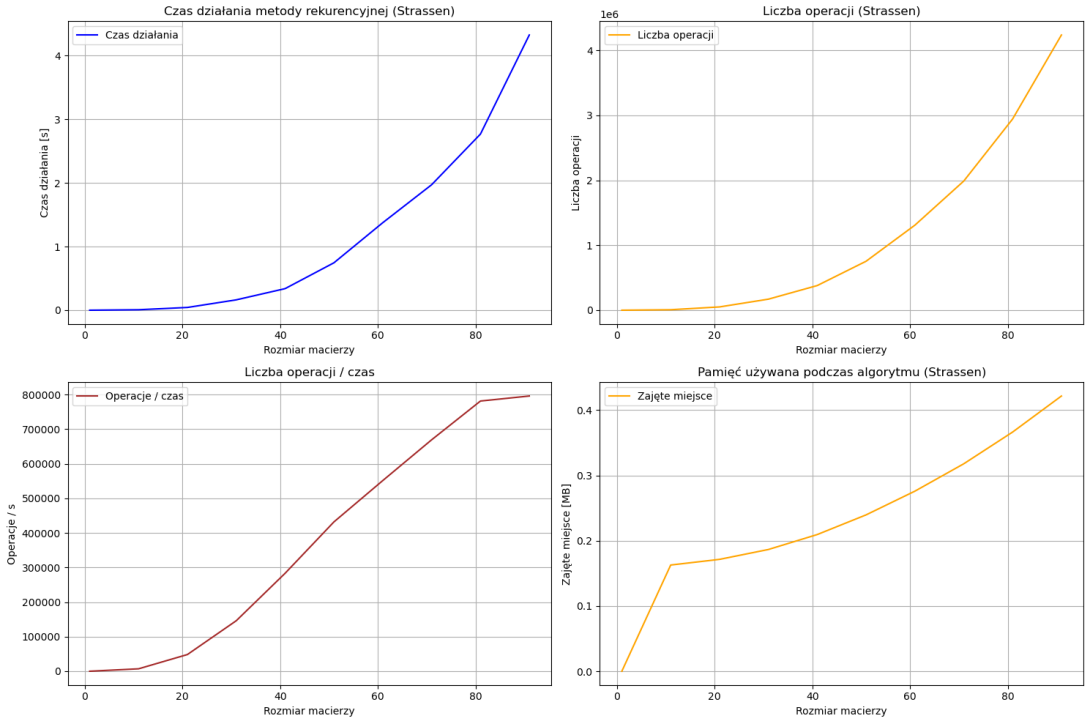
4 Wykresy i porównanie metod

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

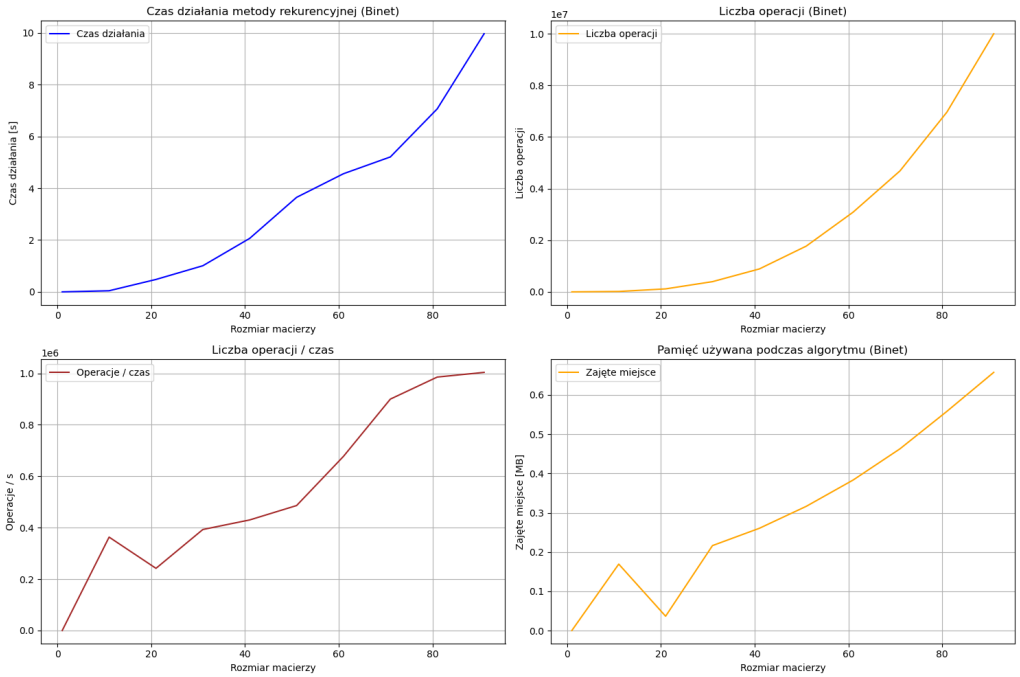


Rysunek 1: Wykresy działania algorytmu w zależności od rozmiaru macierzy - mnożenie macierzy metodą Bineta

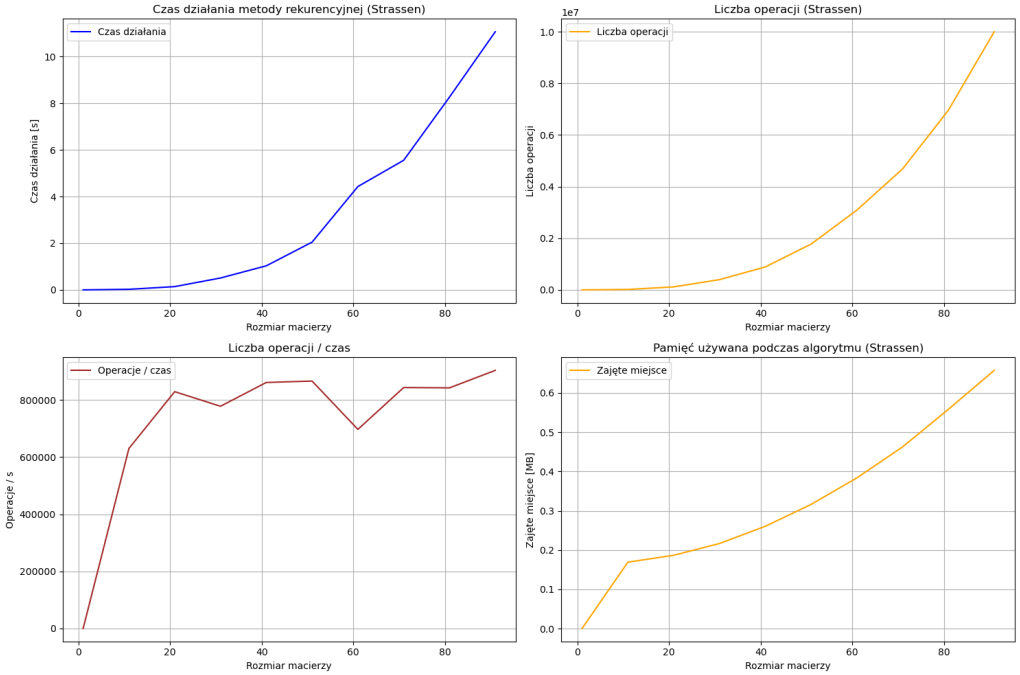


Rysunek 2: Wykresy działania algorytmu w zależności od rozmiaru macierzy - mnożenie macierzy metodą Strassena

4.2 Rekurencyjna eliminacja Gaussa

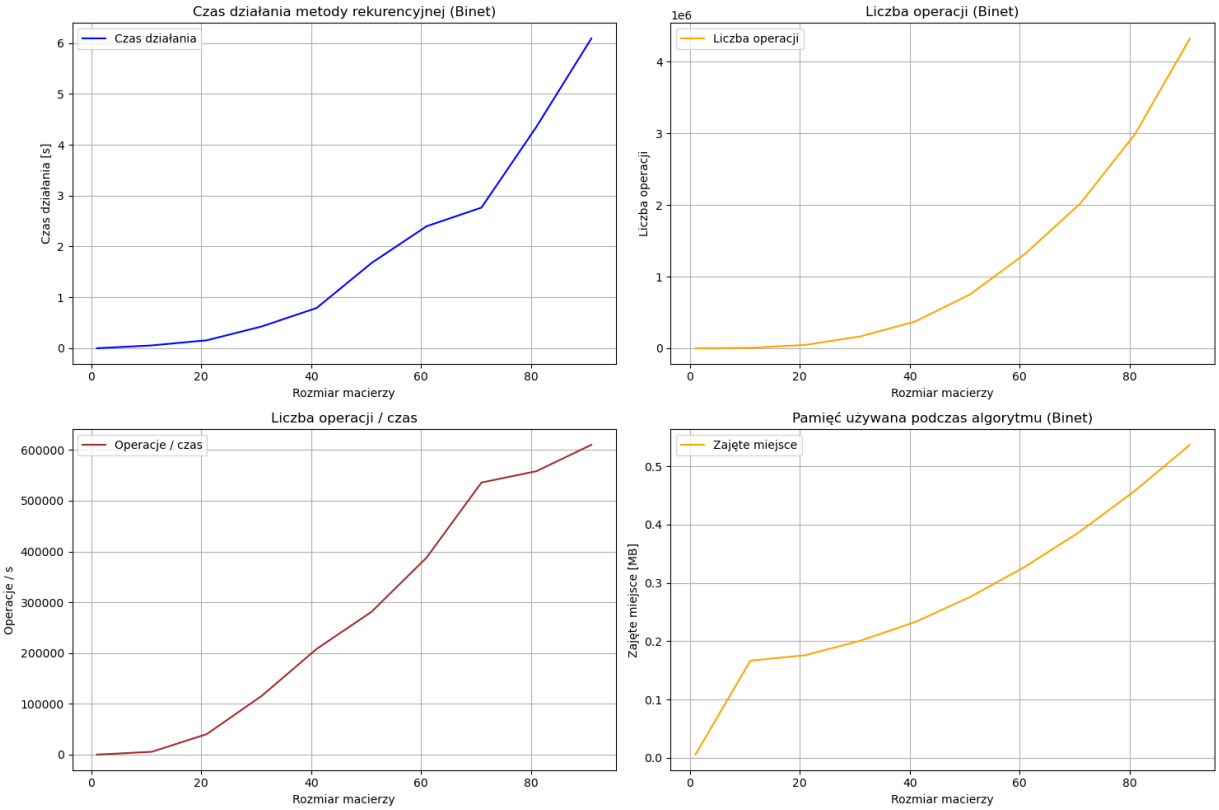


Rysunek 3: Wykresy działania algorytmu w zależności od rozmiaru macierzy - mnożenie macierzy metodą Bineta

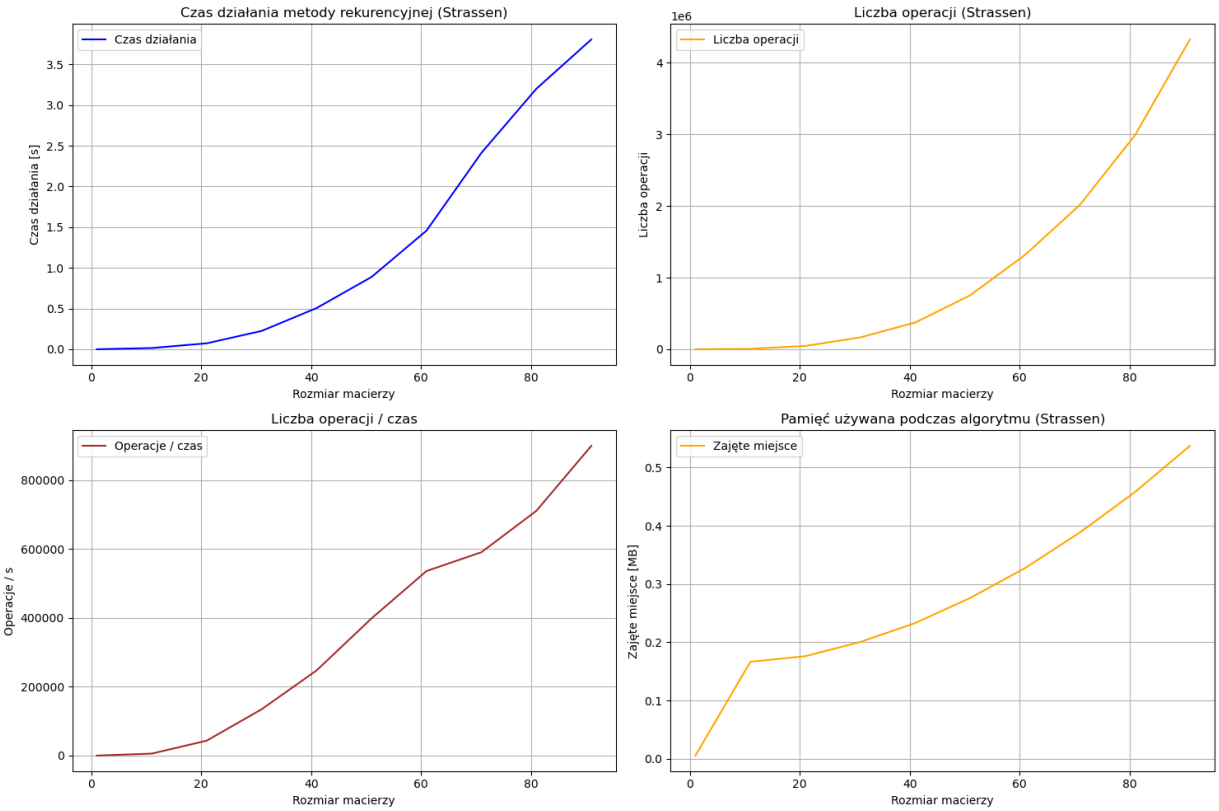


Rysunek 4: Wykresy działania algorytmu w zależności od rozmiaru macierzy - mnożenie macierzy metodą Strassena

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