Algorytmy Mnożenia Macierzy

Bartłomiej Tempka, Radosław Kawa

Algorytm Binét'a

```python
def Binet_multiply(A, B):
    global binet_operation_counter
    n = len(A)
    C = np.zeros((n, n))

    if n <= 1:
        # Jeśli macierze sa 1x1 to wykonaj mnożenie bez rekurencji
        binet_operation_counter += 1
        return np.dot(A, B)

    # Dziel macierze na bloki
    half_size = n // 2
    A11 = A[:half_size, :half_size]
    A12 = A[:half_size, half_size:]
    A21 = A[half_size:, :half_size]
    A22 = A[half_size:, half_size:]

    B11 = B[:half_size, :half_size]
    B12 = B[:half_size, half_size:]
    B21 = B[half_size:, :half_size]
    B22 = B[half_size:, half_size:]

    # Wykonaj rekurencyjne mnożenie macierzy
    C11 = Binet_multiply(A11, B11) + Binet_multiply(A12, B21)
    binet_operation_counter += n
    C12 = Binet_multiply(A11, B12) + Binet_multiply(A12, B22)
    binet_operation_counter += n
    C21 = Binet_multiply(A21, B11) + Binet_multiply(A22, B21)
    binet_operation_counter += n
    C22 = Binet_multiply(A21, B12) + Binet_multiply(A22, B22)
    binet_operation_counter += n

    # Sklej wynik z powrotem w jedną macierz
    C[:half_size, :half_size] = C11
    C[:half_size, half_size:] = C12
    C[half_size:, :half_size] = C21
    C[half_size:, half_size:] = C22

    return C
```

Podział macierzy na ćwiartki

$$\begin{bmatrix} (A_{11}B_{11} + A_{12}B_{21}) & (A_{11}B_{21} + A_{12}B_{22}) \\ (A_{21}B_{11} + A_{22}B_{21}) & (A_{21}B_{12} + A_{22}B_{22}) \end{bmatrix}$$

Algorytm Strassena

```python
def Strassen_multiply(A,B):
    global strassen_operation_counter
    n = len(A)
    C = np.zeros((n, n))
    if n <= 1:
        # Jeśli macierze sa 1x1 to wykonaj mnożenie bez rekurencji
        strassen_operation_counter += 1
        return np.dot(A, B)

    # Dziel macierze na bloki
    half_size = n // 2
    A11 = A[:half_size, :half_size]
    A12 = A[:half_size, half_size:]
    A21 = A[half_size:, :half_size]
    A22 = A[half_size:, half_size:]

    B11 = B[:half_size, :half_size]
    B12 = B[:half_size, half_size:]
    B21 = B[half_size:, :half_size]
    B22 = B[half_size:, half_size:]

    P1 = Strassen_multiply(A11+ A22, B11 + B22)
    strassen_operation_counter += n*2
    P2 = Strassen_multiply(A21 + A22, B11)
    strassen_operation_counter += n
    P3 = Strassen_multiply(A11, B12 - B22)
    strassen_operation_counter += n
    P4 = Strassen_multiply(A22, B21 - B11)
    strassen_operation_counter += n
    P5 = Strassen_multiply(A11 + A12, B22)
    strassen_operation_counter += n
    P6 = Strassen_multiply(A21 - A11, B11 + B12)
    strassen_operation_counter += n*2
    P7 = Strassen_multiply(A12 - A22, B21 + B22)
    strassen_operation_counter += n*2

    C11 = P1 + P4 - P5 + P7
    strassen_operation_counter += n*3
    C12 = P3 + P5
    strassen_operation_counter += n
    C21 = P2 + P4
    strassen_operation_counter += n
    C22 = P1 - P2 + P3 + P6
    strassen_operation_counter += n*3

    # Sklej wynik z powrotem w jedną macierz
    C[:half_size, :half_size] = C11
    C[:half_size, half_size:] = C12
    C[half_size:, :half_size] = C21
    C[half_size:, half_size:] = C22
```

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \quad P_2 = (A_{21} + A_{22})B_{11}$$
$$P_3 = A_{11}(B_{12} - B_{22})P_4 = A_{22}(B_{21} - B_{11}) \quad P_5 = (A_{11} + A_{12})B_{22}$$
$$P_6 = (A_{21} - A_{11})(B_{11} + B_{12}) \quad P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} (P_1 + P_4 - P_5 + P_7) & (P_3 + P_5) \\ (P_2 + P_4) & (P_1 - P_2 + P_3 + P_6) \end{bmatrix}$$

Algorytm Alphatensor

```
def Alpha(A,B):
    global alphatensor_operation_counter
    n = len(A)
    half_size = n // 2
    C = np.zeros((n, n))
    if n == 1:
        alphatensor_operation_counter += 1
        return np.dot(A,B)
    A1 = A[:half_size, :half_size]
    A2 = A[:half_size, half_size:]
    A3 = A[half_size:, :half_size]
    A4 = A[half_size:, half_size:]

    B1 = B[:half_size, :half_size]
    B2 = B[:half_size, half_size:]
    B3 = B[half_size:, :half_size]
    B4 = B[half_size:, half_size:]

    P1 = Alpha(A3-A4,B2)
    alphatensor_operation_counter += half_size**2
    P2 = Alpha(A1+A3-A4,B2+B3+B4)
    alphatensor_operation_counter +=(half_size**2)*4
    P3 = Alpha(A1-A2+A3-A4,B3+B4)
    alphatensor_operation_counter += (half_size**2)*4
    P4 = Alpha(A2,B3)
    P5 = Alpha(A1+A3,B1+B2+B3+B4)
    alphatensor_operation_counter += (half_size**2)*4
    P6 = Alpha(A1,B1)
    P7 = Alpha(A4,B2+B4)
    alphatensor_operation_counter += (half_size**2)*2

    C1 = P4 + P6
    alphatensor_operation_counter += (half_size**2)
    C2 = -P2 +P5 - P6-P7
    alphatensor_operation_counter += (half_size**2)*3
    C3 = -P1 +P2-P3-P4
    alphatensor_operation_counter += (half_size**2)*3
    C4 = P1+P7
    alphatensor_operation_counter += (half_size**2)

    C[:half_size, :half_size] = C1
    C[:half_size, half_size:] = C2
    C[half_size:, :half_size] = C3
    C[half_size:, half_size:] = C4
    return C
```

Użycie tensora do policzenia mnożenia

U

	0	1	2	3	4	5	6
0	0	1	1	0	1	1	0
1	0	0	-1	1	0	0	0
2	1	1	1	0	1	0	0
3	-1	-1	-1	0	0	0	1

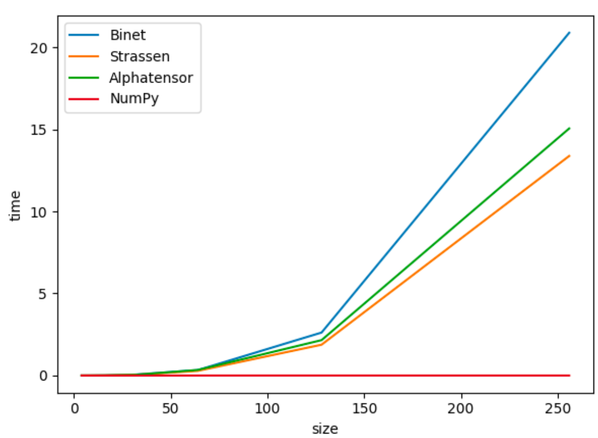
V

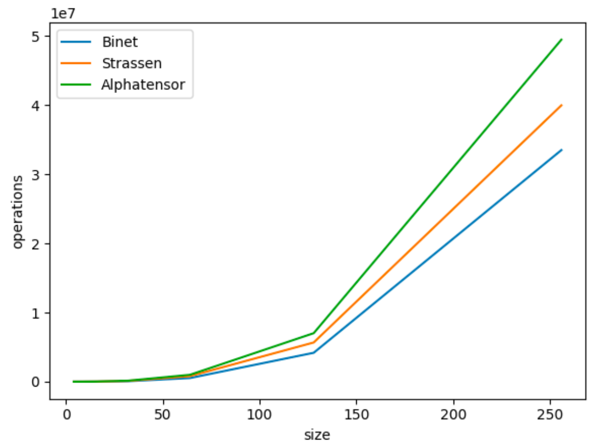
	0	1	2	3	4	5	6
0	0	0	0	0	1	1	0
1	1	1	0	0	1	0	1
2	0	1	1	1	1	0	0
3	0	1	1	0	1	0	1

W

	0	1	2	3	4	5	6
0	0	0	0	1	0	1	0
1	0	-1	0	0	1	-1	-1
2	-1	1	-1	-1	0	0	0
3	1	0	0	0	0	0	1

Porównanie prędkości algorytmów





Złożoność obliczeniowa

Złożoność obliczeniową algorytmów możemy oszacować używając równań rekurencyjnych
Metod Binéc'a potrzebuje 8 mnożeń i 4 dodawania na każdy krok rekurencji więc otrzymujemy równanie:

$T(n) = 8T(n/2) + 4(n/2)^2, T(1) = 1$

Metod Strassena potrzebuje 7 mnożeń i 18 dodawań:

$T(n) = 7T(n/2) + 18(n/2)^2, T(1) = 1$

Algorytm Alphatensor potrzebuje 7 mnożeń i 22 dodawań (używamy tylko metody przeznaczonej do mnożenia macierzy wielkości 2x2 i stosujemy ją rekurencyjnie)

$T(n) = 7T(n/2) + 22(n/2)^2, T(1) = 1$

Po rozwiązaniu tych równań otrzymujemy kolejno

Binét: $T(n) = n^2(n-1) = O(n^3)$

Strassen: $T(n) = (2n)^{\frac{log(7)}{log(2)}} - 6n^2 \approx (2n)^{2.807} - 6n^2 = O(n^{2.807})$

Alphatensor: $T(n) = \frac{25}{3}(2n)^{\frac{log(7)}{log(2)}}$ - $\frac{22}{3}(n^2) \approx O(n^{2.807})$