Множення Матриць

Виконав Пристайчук Дмитро, ММШІ-1

Крок 1: Базовий випадок рекурсії

Якщо розміри матриць `A` (n x m) та `B` (m x p) менші або рівні пороговому значенню `L` (тобто `n <= L`, `m <= L`, `p <= L`), використовується стандартне множення матриць NumPy. Також стандартне множення використовується, якщо будь-який з розмірів дорівнює 1.

```
# Base case: if matrices are small enough, use standard multiplication
if n <= L and m <= L and p <= L:
    return A @ B

# Special case: if any dimension is 1, use standard multiplication
if n == 1 or m == 1 or p == 1:
    return A @ B
```

Крок 2: Розділення матриць на блоки

Якщо матриці більші за порогове значення, вони розділяються на чотири підблоки приблизно однакового розміру.

```
# Divide matrices into blocks
n_half = n // 2
m_half = m // 2
p_half = p // 2

# Split A
A11 = A[:n_half, :m_half]
A12 = A[:n_half, m_half:]
A21 = A[n_half:, :m_half]
A22 = A[n_half:, m_half:]

# Split B
B11 = B[:m_half, :p_half]
B12 = B[:m_half, p_half:]
B21 = B[m_half:, :p_half]
B22 = B[m_half:, p_half:]
```

Крок 3: Рекурсивне множення блоків

Виконується рекурсивне множення підблоків згідно з формулами блочного множення матриць:

```
# Recursive multiplication of blocks
C11 = matrix_multiply(A11, B11, L) + matrix_multiply(A12, B21, L)
C12 = matrix_multiply(A11, B12, L) + matrix_multiply(A12, B22, L)
C21 = matrix_multiply(A21, B11, L) + matrix_multiply(A22, B21, L)
C22 = matrix_multiply(A21, B12, L) + matrix_multiply(A22, B22, L)
```

Крок 4: Об'єднання блоків результату

Отримані блоки `C11`, `C12`, `C21`, `C22` об'єднуються для формування результуючої матриці `C`.

```
# Combine blocks to form C
top = np.hstack((C11, C12))
bottom = np.hstack((C21, C22))
C = np.vstack((top, bottom))

return C
```

Повна реалізація функції в Python

```
import numpy as np

def matrix_multiply(A, B, L):
    """
    Multiply matrices A and B using divide-and-conquer approach.
    Only perform direct multiplication if dimensions are <= L.

    Parameters:
    -----------
    A, B : numpy.ndarray
        Input matrices to multiply
    L : int
        Threshold for direct multiplication

    Returns:
    --------
    C : numpy.ndarray
        Result matrix C = A×B
    """
    # Get dimensions
    n, m = A.shape
    m2, p = B.shape

    # Check if matrices can be multiplied
    if m != m2:
        raise ValueError("Matrix dimensions incompatible for
multiplication")

    # Base case: if matrices are small enough, use standard multiplication
```

```
    if n <= L and m <= L and p <= L:
        return A @ B

    # Special case: if any dimension is 1, use standard multiplication
    if n == 1 or m == 1 or p == 1:
        return A @ B

    # Divide matrices into blocks
    n_half = n // 2
    m_half = m // 2
    p_half = p // 2

    # Split A
    A11 = A[:n_half, :m_half]
    A12 = A[:n_half, m_half:]
    A21 = A[n_half:, :m_half]
    A22 = A[n_half:, m_half:]

    # Split B
    B11 = B[:m_half, :p_half]
    B12 = B[:m_half, p_half:]
    B21 = B[m_half:, :p_half]
    B22 = B[m_half:, p_half:]

    # Recursive multiplication of blocks
    C11 = matrix_multiply(A11, B11, L) + matrix_multiply(A12, B21, L)
    C12 = matrix_multiply(A11, B12, L) + matrix_multiply(A12, B22, L)
    C21 = matrix_multiply(A21, B11, L) + matrix_multiply(A22, B21, L)
    C22 = matrix_multiply(A21, B12, L) + matrix_multiply(A22, B22, L)

    # Combine blocks to form C
    top = np.hstack((C11, C12))
    bottom = np.hstack((C21, C22))
    C = np.vstack((top, bottom))

    return C
```

Результати виконання

```
Multiplying 256x256 matrices with threshold L=64
Divide-and-conquer time: 0.0032 seconds
NumPy time: 0.0005 seconds
Maximum difference: 1.4210854715202004e-13
```