12.270

EE25BTECH11049-Sai Krishna Bakki

October 4, 2025

Question

If

$$\mathbf{A} = \begin{pmatrix} 2 & 4 \\ 1 & 3 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 4 & 6 \\ 5 & 9 \end{pmatrix},$$

$(\mathbf{AB})^T$ is equal to

Theoretical Solution

Given

$$\mathbf{A} = \begin{pmatrix} 2 & 4 \\ 1 & 3 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 4 & 6 \\ 5 & 9 \end{pmatrix}, \mathbf{A}^T = \begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix}, \mathbf{B}^T = \begin{pmatrix} 4 & 5 \\ 6 & 9 \end{pmatrix} \tag{1}$$

$(\mathbf{AB})^T$ can also be written as $\mathbf{B}^T\mathbf{A}^T$

$$(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T \tag{2}$$

$$\implies \begin{pmatrix} 4 & 5 \\ 6 & 9 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix} \tag{3}$$

$$\implies \begin{pmatrix} 8+20 & 4+15 \\ 12+36 & 6+27 \end{pmatrix} \tag{4}$$

$$\implies \begin{pmatrix} 28 & 19 \\ 48 & 33 \end{pmatrix} \tag{5}$$

$\therefore (\mathbf{AB})^T$ is equal to $\begin{pmatrix} 28 & 19 \\ 48 & 33 \end{pmatrix}$.

C Code

```c
#include <stdio.h>

// This function multiplies two 2x2 matrices (A and B) and stores the
// transpose of the result in the `result` matrix.
// Matrices are passed as pointers to 1D arrays of size 4 (row-major order).
void multiply_and_transpose(double* A, double* B, double* result) {
    double product[4];

    // Perform matrix multiplication: C = A * B
    // C[0,0] = A[0,0]*B[0,0] + A[0,1]*B[1,0]
    product[0] = A[0] * B[0] + A[1] * B[2];
    // C[0,1] = A[0,0]*B[0,1] + A[0,1]*B[1,1]
    product[1] = A[0] * B[1] + A[1] * B[3];
    // C[1,0] = A[1,0]*B[0,0] + A[1,1]*B[1,0]
```

C Code

```
    product[2] = A[2] * B[0] + A[3] * B[2];
    // C[1,1] = A[1,0]*B[0,1] + A[1,1]*B[1,1]
    product[3] = A[2] * B[1] + A[3] * B[3];

    // Transpose the product matrix and store it in the result
    // result[0,0] = product[0,0]
    result[0] = product[0];
    // result[0,1] = product[1,0]
    result[1] = product[2];
    // result[1,0] = product[0,1]
    result[2] = product[1];
    // result[1,1] = product[1,1]
    result[3] = product[3];
}
```

Python Code Through Shared Output

```
import ctypes
import numpy as np
import os

# Define the name of the shared library based on the operating
    system
if os.name == 'nt': # Windows
    lib_name = 'matrix_ops.dll'
else: # Linux, macOS, etc.
    lib_name = 'matrix.so'

# Check if the library file exists before trying to load it
if not os.path.exists(lib_name):
    print(fError: Shared library '{lib_name}' not found.)
    print(Please compile 'matrix_ops.c' first. See README.md for
        instructions.)
    exit()
```

Python Code Through Shared Output

```
# Load the shared library
c_lib = ctypes.CDLL(os.path.abspath(lib_name))

# Define the argument types and return type for the C function.
# This ensures Python sends the data in the correct format.
# The function expects three arguments: pointers to C doubles.
c_lib.multiply_and_transpose.argtypes = [
    ctypes.POINTER(ctypes.c_double),
    ctypes.POINTER(ctypes.c_double),
    ctypes.POINTER(ctypes.c_double)
]
# The C function doesn't return a value; it modifies the 'result'
    array in place.
c_lib.multiply_and_transpose.restype = None

# Define the input matrices using numpy.
# It's crucial to specify the dtype as np.double to match '
    c_double' in ctypes.
A = np.array([[2, 4], [1, 3]], dtype=np.double)
```

Python Code Through Shared Output

```
# Convert the numpy arrays into a format that ctypes can use.
# This gets a C-compatible pointer to the underlying data buffer
    of the array.
A_ptr = A.ctypes.data_as(ctypes.POINTER(ctypes.c_double))
B_ptr = B.ctypes.data_as(ctypes.POINTER(ctypes.c_double))
result_ptr = result_from_c.ctypes.data_as(ctypes.POINTER(ctypes.
    c_double))

# Call the C function with the pointers to the data
c_lib.multiply_and_transpose(A_ptr, B_ptr, result_ptr)

# Print the original matrices and the final result
print(Matrix A:\n, A)
print(\nMatrix B:\n, B)
print(\nResult from C function (AB)T:\n, result_from_c)
```

Python Code

```
import numpy as np
# Define the matrices A and B from the problem
A = np.array([[2, 4],
              [1, 3]])

B = np.array([[4, 6],
              [5, 9]])
# Step 1: Calculate the product of A and B (A multiplied by B)
# The '@' operator is used for matrix multiplication in numpy
product_AB = A @ B
# Step 2: Calculate the transpose of the resulting matrix
# The .T attribute returns the transpose of a numpy array
transpose_of_product = product_AB.T
# Print the original matrices and the final result for clarity
print(Matrix A:\n, A)
print(\nMatrix B:\n, B)
#print(\nProduct of A and B (AB):\n, product_AB)
print(\nTranspose of the product (AB)T:\n, transpose_of_product)
```