Assignment 9

矩阵相乘算法：

Input： $\boldsymbol{A}=[a_{ik}]_{M\times K},\boldsymbol{B}=[b_{kj}]_{K\times N}$

Output： $\boldsymbol{C}=[c_{ij}]_{M\times N}$

Let $\boldsymbol{C}$ be a new matrix

For i $=1\ldots M$

  For $j=1\ldots N$

    Let sum $=0$

    For $k=1\ldots K$

      Set sum $=$ sum $+\,a_{ik}b_{kj}$

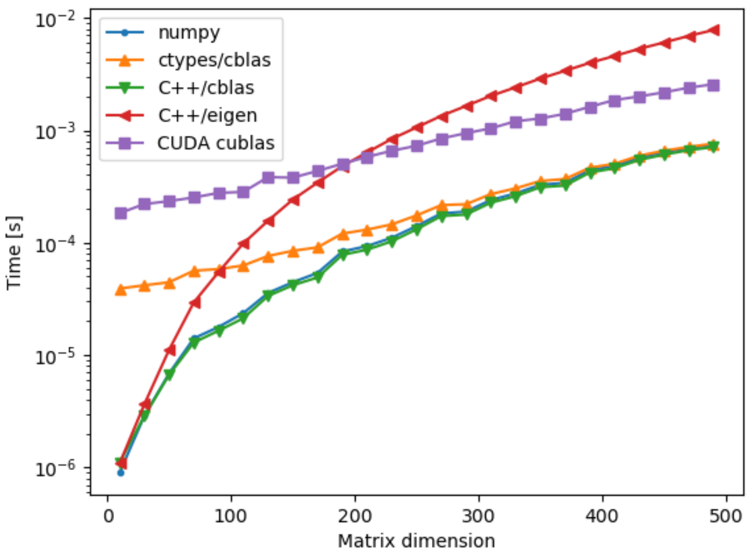
    $c_{ij}=$ sum

Return $\boldsymbol{C}$

本次作业，我们将基于上文矩阵相乘算法(Matrix Mulitplication)，* 使用原始的Python语言实现一个 `mat_mult(A,B)` 函数

- 使用Numpy.dot对自己实现的函数进行验证
- 随机生成10x10-5000x5000的矩阵，对比 `mat_mult(A,B)` 和 `np.dot(A,B)` 的计算时间，并使用 `matplotlib` 画图，x轴为矩阵大小，y轴为计算时间

案例结果：



In [ ]:

```python
import numpy as np
import timeit as ti
```

In [ ]:

```python
def mat_multi(A,B):
    M,K = A.shape
    K,N = B.shape
    C=np.zeros((M,N))
    for i in range(M):
        for j in range(N):
            sum_1 = 0
            for k in range(K):
                sum_1 += A[i,k]* B[k,j]
                C[i,j] = sum_1
    return C
```

In [ ]:

```python
#Test example
A=np.array([[1,2,3],
            [4,5,6]])
B=np.array([[1,2],
             [6,7],
             [11,12]])

C_ref = np.dot(A,B)
print('Numpy answer=',C_ref)

C_myfunc = mat_multi(A,B)
print('MyAnswer=',C_myfunc)
```

```
Numpy answer= [[ 46  52]
 [100 115]]
MyAnswer= [[ 46.  52.]
 [100. 115.]]
```

In [ ]:

```python
#Benchmark example

rslt = []
rslt2 = []
for n_size in range(10, 100, 5):
    print(f'Matrix size of {n_size}x{n_size}...')

    A = np.random.rand(n_size,n_size)
    B = np.random.rand(n_size,n_size)

    c = [n_size, ]

    #Run np.dot(B,A) 10 times and use the shortest run time as its performance metric
    t = ti.Timer(lambda: np.dot(B, A))
    c.append(np.min(t.repeat(10, 1)))
    print(f"    Numpy CPU time = {c[-1]:.2e}s")

    d = [n_size, ]

    #Run mat_multi(B,A) 10 times and use the shortest run time as its performance metr
    #Complete this function
    t = ti.Timer(lambda: mat_multi(B, A))
    d.append(np.min(t.repeat(10, 1)))
    print(f"    My implementation CPU time = {d[-1]:.2e}s")


    rslt.append(c)
    rslt2.append(d)
```

```
#Convert list into numpy array
rslt = np.array(rslt)
rslt2 = np.array(rslt2)

print(rslt)
print(rslt2)
```

```
Matrix size of 10x10...
   Numpy CPU time = 1.90e-06s
   My implementation CPU time = 3.52e-04s
Matrix size of 15x15...
   Numpy CPU time = 3.30e-06s
   My implementation CPU time = 1.13e-03s
Matrix size of 20x20...
   Numpy CPU time = 4.60e-06s
   My implementation CPU time = 2.67e-03s
Matrix size of 25x25...
   Numpy CPU time = 4.80e-06s
   My implementation CPU time = 5.03e-03s
Matrix size of 30x30...
   Numpy CPU time = 6.60e-06s
   My implementation CPU time = 8.76e-03s
Matrix size of 35x35...
   Numpy CPU time = 6.20e-06s
   My implementation CPU time = 1.47e-02s
Matrix size of 40x40...
   Numpy CPU time = 6.60e-06s
   My implementation CPU time = 2.08e-02s
Matrix size of 45x45...
   Numpy CPU time = 7.20e-06s
   My implementation CPU time = 2.89e-02s
Matrix size of 50x50...
   Numpy CPU time = 1.00e-05s
   My implementation CPU time = 3.94e-02s
Matrix size of 55x55...
   Numpy CPU time = 1.17e-05s
   My implementation CPU time = 5.39e-02s
Matrix size of 60x60...
   Numpy CPU time = 9.10e-06s
   My implementation CPU time = 6.69e-02s
Matrix size of 65x65...
   Numpy CPU time = 1.22e-05s
   My implementation CPU time = 8.52e-02s
Matrix size of 70x70...
   Numpy CPU time = 1.54e-05s
   My implementation CPU time = 1.07e-01s
Matrix size of 75x75...
   Numpy CPU time = 1.22e-05s
   My implementation CPU time = 1.36e-01s
Matrix size of 80x80...
   Numpy CPU time = 1.87e-05s
   My implementation CPU time = 1.58e-01s
Matrix size of 85x85...
   Numpy CPU time = 2.37e-05s
   My implementation CPU time = 1.91e-01s
Matrix size of 90x90...
   Numpy CPU time = 2.06e-05s
   My implementation CPU time = 2.32e-01s
Matrix size of 95x95...
   Numpy CPU time = 2.47e-05s
   My implementation CPU time = 2.70e-01s
[[1.00000000e+01 1.90000003e-06]
 [1.50000000e+01 3.30000000e-06]
 [2.00000000e+01 4.60000001e-06]
 [2.50000000e+01 4.79999994e-06]
 [3.00000000e+01 6.60000001e-06]
 [3.50000000e+01 6.20000003e-06]
 [4.00000000e+01 6.60000001e-06]
 [4.50000000e+01 7.20000003e-06]
 [5.00000000e+01 9.99999997e-06]
 [5.50000000e+01 1.17000001e-05]
 [6.00000000e+01 9.09999994e-06]
 [6.50000000e+01 1.21999999e-05]
 [7.00000000e+01 1.53999999e-05]
 [7.50000000e+01 1.21999999e-05]
 [8.00000000e+01 1.86999999e-05]
 [8.50000000e+01 2.37000000e-05]
 [9.00000000e+01 2.06000000e-05]
 [9.50000000e+01 2.46999999e-05]]
[[1.000000e+01 3.522000e-04]
 [1.500000e+01 1.127100e-03]
 [2.000000e+01 2.665700e-03]
 [2.500000e+01 5.027000e-03]
 [3.000000e+01 8.757500e-03]
 [3.500000e+01 1.473310e-02]
 [4.000000e+01 2.082200e-02]
 [4.500000e+01 2.888000e-02]
 [5.000000e+01 3.937830e-02]
 [5.500000e+01 5.385000e-02]
 [6.000000e+01 6.688670e-02]
 [6.500000e+01 8.516230e-02]
 [7.000000e+01 1.072242e-01]
 [7.500000e+01 1.358514e-01]
 [8.000000e+01 1.581315e-01]
 [8.500000e+01 1.907103e-01]
 [9.000000e+01 2.315961e-01]
 [9.500000e+01 2.702485e-01]]
```

In [ ]:

```
#Plot performance plot using Matplotlib
import matplotlib.pyplot as plt
x_1 = rslt[:, 0]
y_1 = rslt[:, 1]
x_2 = rslt2[:, 0]
y_2 = rslt2[:, 1]
```

In [ ]:

```
x = np.arange(10, 100)
plt.plot(x_1, y_1, 's-', color = 'r', label='Numpy' )
plt.plot(x_2, y_2, 'o-', color = 'g', label='Python' )
plt.xlabel("Matrix dimesion")
plt.ylabel("Time [s]")
plt.title("Speed Comparison")
plt.legend()
plt.show()
```