assignment_1

October 27, 2024

1 Assignment 1 - Matrix-matrix multiplication

Note: Delayed Assessment Permit (DAP) was used for this coursework, hence why submission was "late". Due date, according to DAP is 28/10/2024 at 5pm.

18008928

[19]:
```
import numpy as np
import timeit as timeit
from numpy.testing import assert_almost_equal
import matplotlib.pylab as plt
import numba
```

1.0.1 Part 1: a better function

[21]:
```
# Slow matrix multiplication function

def slow_matrix_product(mat1, mat2):
    """Multiply two matrices."""
    assert mat1.shape[1] == mat2.shape[0]
    result = []
    for c in range(mat2.shape[1]):
        column = []
        for r in range(mat1.shape[0]):
            value = 0
            for i in range(mat1.shape[1]):
                value += mat1[r, i] * mat2[i, c]
            column.append(value)
        result.append(column)
    return np.array(result).transpose()
```

[22]:
```
# Faster matrix multiplication function

def fast_matrix_product(mat1, mat2):
    """Multiply two matrices."""
    assert mat1.shape[1] == mat2.shape[0]
    result = []
    for r in range(mat1.shape[0]):
```

```
        column = []
        for c in range(mat2.shape[1]):
            value = np.sum(mat1[r,:]*mat2[:, c])
            column.append(value)
        result.append(column)
    return np.array(result)
```

[23]:
```
# Assertion Checker

def test_matrix_function(func):

    test_dimensions = [2,3,4,5]

    for n in test_dimensions:
        matrix1 = np.random.rand(n,n)
        matrix2 = np.random.rand(n,n)

        assert_almost_equal(func(matrix1,matrix2), matrix1@matrix2)

        return "Your matrix multiplier works as intended"

test_matrix_function(fast_matrix_product)
```

[23]:
```
'Your matrix multiplier works as intended'
```

Reasons why my function is better than slow_matrix_product:

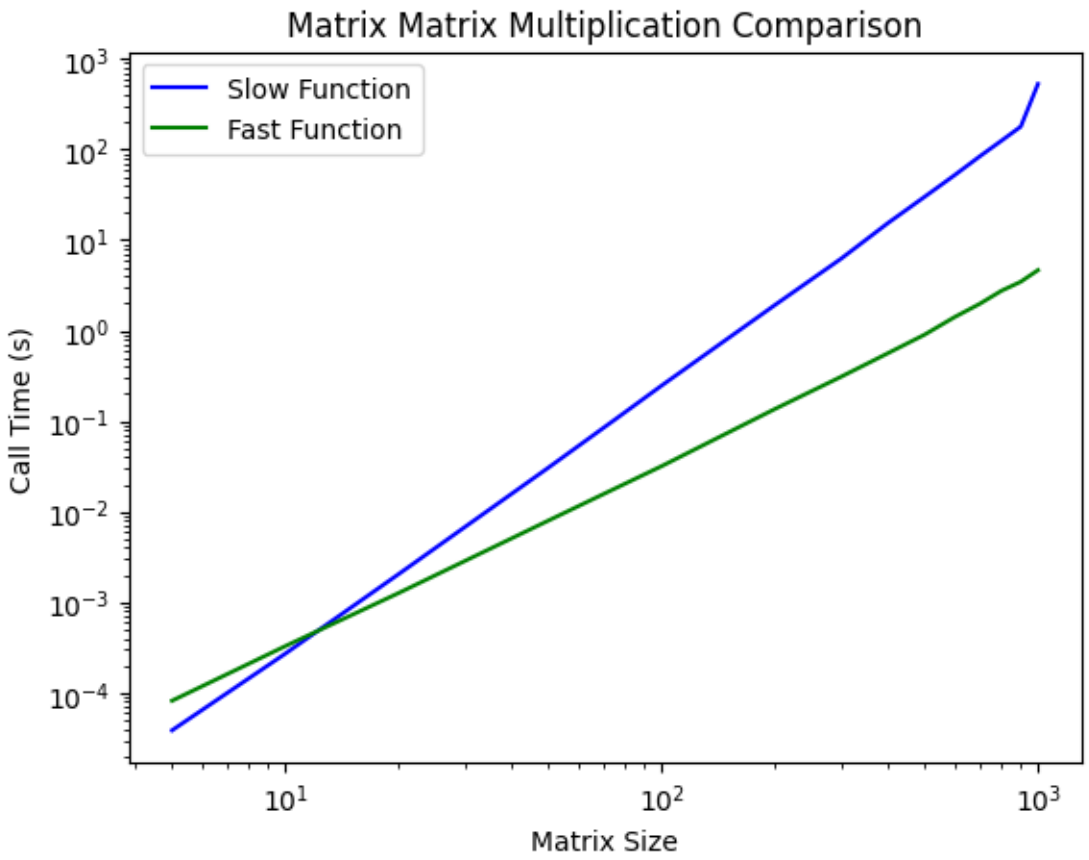
1. Python is an interpreted language, and looks at code line by line. It therefore executes each line for each iteration, meaning for loops are quite slow; having nested for loops makes this process even slower. Thus, truncating the 3 for loops in slow_matrix_product into 2 makes the process faster.
2. Removed transpose() by swapping the for loops of mat1 and mat2, so that less unnecessary calculations are done.

[7]:
```
def fast_comparison_plots(slow_func,fast_func):

    # Ensuring function works correctly
    test_matrix_function(fast_func)

    # Setting up execution timers

    slow_times = []
    fast_times = []

    matrix_size = [5,10,20,50,75]+list(range(100,1001,100))

    for n in matrix_size:
        matrix1 = np.random.rand(n,n)
```

```
        matrix2 = np.random.rand(n,n)

        t_slow = %timeit -o (slow_func(matrix1,matrix2))
        t_fast = %timeit -o (fast_func(matrix1,matrix2))

        slow_times.append(t_slow.average)
        fast_times.append(t_fast.average)

        print(f"[Slow] N={n} Average Run time: {t_slow.average}")
        print(f"[Fast] N={n} Average Run Time: {t_fast.average}")

    # Setting up plots

    plt.plot(matrix_size, slow_times, "b", label="Slow Function")
    plt.plot(matrix_size, fast_times, "g", label="Fast Function")

    plt.xlabel("Matrix Size")
    plt.xscale("log")
    plt.yscale("log")
    plt.ylabel("Call Time (s)")
    plt.title("Matrix Matrix Multiplication Comparison")
    plt.legend()

fast_comparison_plots(slow_matrix_product,fast_matrix_product)
```

```
38.7  s ± 51.7 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
82.1  s ± 120 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
[Slow] N=5 Average Run time: 3.867387285988246e-05
[Fast] N=5 Average Run Time: 8.209301571893905e-05
272  s ± 849 ns per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
329  s ± 4.17  s per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
[Slow] N=10 Average Run time: 0.00027164214284026197
[Fast] N=10 Average Run Time: 0.00032889530002804733
2.05 ms ± 12.4  s per loop (mean ± std. dev. of 7 runs, 100 loops each)
1.27 ms ± 22.7  s per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
[Slow] N=20 Average Run time: 0.0020504004288730876
[Fast] N=20 Average Run Time: 0.0012677341857592443
30.7 ms ± 236  s per loop (mean ± std. dev. of 7 runs, 10 loops each)
7.99 ms ± 95.2  s per loop (mean ± std. dev. of 7 runs, 100 loops each)
[Slow] N=50 Average Run time: 0.030676552856207958
[Fast] N=50 Average Run Time: 0.007994026714337191
104 ms ± 1.06 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
18 ms ± 219  s per loop (mean ± std. dev. of 7 runs, 100 loops each)
[Slow] N=75 Average Run time: 0.10360126857579287
[Fast] N=75 Average Run Time: 0.017977090857124754
247 ms ± 1.84 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
31.9 ms ± 89.2  s per loop (mean ± std. dev. of 7 runs, 10 loops each)
[Slow] N=100 Average Run time: 0.24736087140627205
```

```
[Fast] N=100 Average Run Time: 0.031905584283439176
1.92 s ± 14.8 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
136 ms ± 735  s per loop (mean ± std. dev. of 7 runs, 10 loops each)
[Slow] N=200 Average Run time: 1.917286314336317
[Fast] N=200 Average Run Time: 0.13642832428616072
6.23 s ± 24.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
310 ms ± 2.44 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[Slow] N=300 Average Run time: 6.233110928575376
[Fast] N=300 Average Run Time: 0.3098908285610378
15.5 s ± 102 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
565 ms ± 3.67 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[Slow] N=400 Average Run time: 15.465616128573727
[Fast] N=400 Average Run Time: 0.5651047571362662
30 s ± 98.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
903 ms ± 7.79 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[Slow] N=500 Average Run time: 30.036983014250705
[Fast] N=500 Average Run Time: 0.9034976428707263
51.9 s ± 172 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.41 s ± 12.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[Slow] N=600 Average Run time: 51.887093785717816
[Fast] N=600 Average Run Time: 1.4076988856374686
1min 23s ± 157 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.97 s ± 12.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[Slow] N=700 Average Run time: 83.72877304289224
[Fast] N=700 Average Run Time: 1.973480528519888
2min 4s ± 225 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
2.75 s ± 18.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[Slow] N=800 Average Run time: 124.91598564279931
[Fast] N=800 Average Run Time: 2.7541521714135473
2min 58s ± 332 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
3.46 s ± 51.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[Slow] N=900 Average Run time: 178.5298914286042
[Fast] N=900 Average Run Time: 3.4630153285605565
The slowest run took 8.80 times longer than the fastest. This could mean that an
intermediate result is being cached.
8min 48s ± 11min 20s per loop (mean ± std. dev. of 7 runs, 1 loop each)
4.65 s ± 59.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[Slow] N=1000 Average Run time: 528.091690157141
[Fast] N=1000 Average Run Time: 4.648584114254585
```



1.0.2 Part 2

[34]:
```
#  Numba matrix multiplication Function

@numba.njit(fastmath=True)
def numba_matrix_product(mat1, mat2):
    """Multiply two matrices."""
    assert mat1.shape[1] == mat2.shape[0]
    result = np.empty((mat1.shape[0],mat2.shape[1]), dtype=mat1.dtype)
    dt = mat1.dtype

    for r in range(mat1.shape[0]):
        for c in range(mat2.shape[1]):
            value = dt.type(0)
            for i in range(mat1.shape[1]):
                value += mat1[r, i] * mat2[i, c]
            result[r,c] = value
    return result
```

```
matrix1 = np.random.rand(10, 10)
matrix2 = np.random.rand(10, 10)

test_matrix_function(numba_matrix_product)
```

[34]:
```
'Your matrix multiplier works as intended'
```

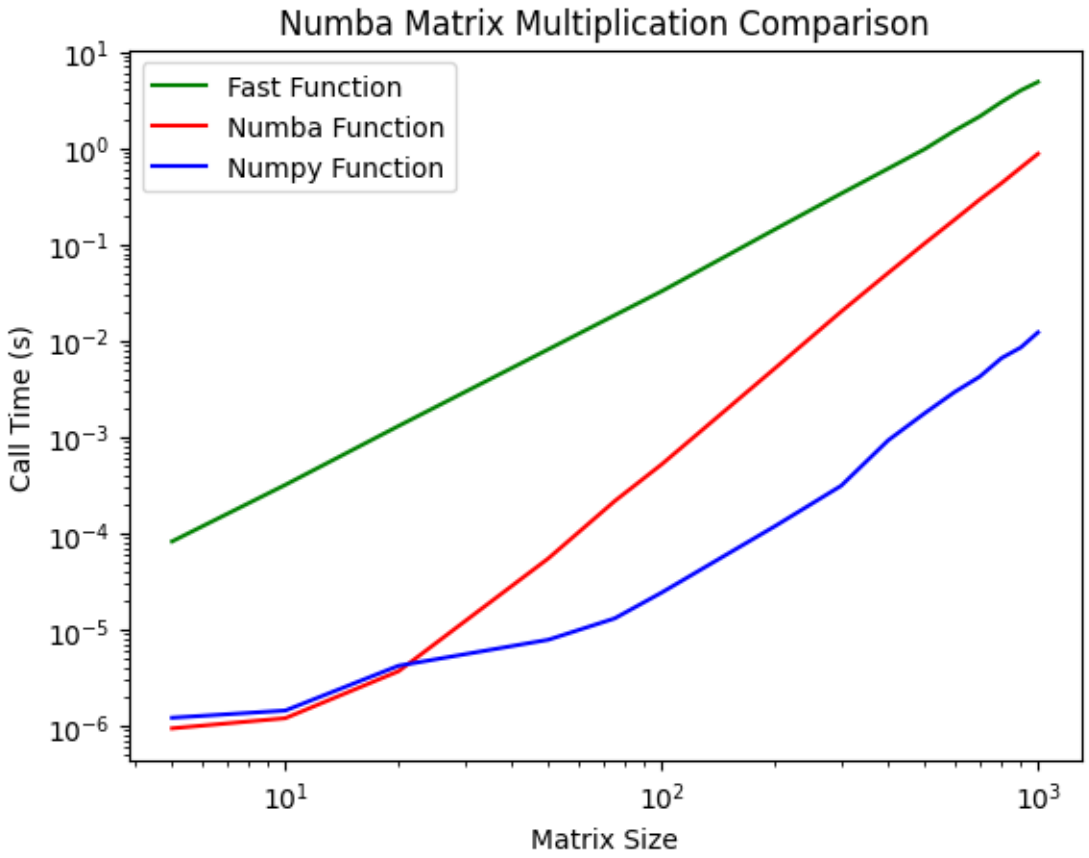
[36]:
```
def numba_comparison_plots(fast_func,numba_func):

    # Copy and pasting time_and_plot function, and changing to account for␣
 ↪numba.

    # Ensuring function works correctly
    test_matrix_function(fast_func)
    test_matrix_function(numba_func)

    # Setting up execution timers

    fast_times = []
    numba_times= []
    np_times = []

    matrix_size = [5,10,20,50,75]+list(range(100,1001,100))

    for n in matrix_size:
        matrix1 = np.random.rand(n,n)
        matrix2 = np.random.rand(n,n)

        t_fast = %timeit -o (fast_func(matrix1,matrix2))
        t_numba= %timeit -o (numba_func(matrix1,matrix2))
        t_np = %timeit -o (matrix1@matrix2)

        fast_times.append(t_fast.average)
        numba_times.append(t_numba.average)
        np_times.append(t_np.average)

        print(f"[Fast] N={n} Average Run Time: {t_fast.average}")
        print(f"[Numba] N={n} Average Run Time: {t_numba.average}")
        print(f"[Numpy] N={n} Average Run time: {t_np.average}")

    # Setting up plots

    plt.plot(matrix_size, fast_times, "g", label="Fast Function")
    plt.plot(matrix_size, numba_times, "r", label="Numba Function")
    plt.plot(matrix_size, np_times, "b", label="Numpy Function")

    plt.xlabel("Matrix Size")
```

```
    plt.xscale("log")
    plt.yscale("log")
    plt.ylabel("Call Time (s)")
    plt.title("Numba Matrix Multiplication Comparison")
    plt.legend()

numba_comparison_plots(fast_matrix_product,numba_matrix_product)
```

```
82.1  s ± 856 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
942 ns ± 7.88 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.21  s ± 10.9 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Fast] N=5 Average Run Time: 8.214125428348779e-05
[Numba] N=5 Average Run Time: 9.424478714354336e-07
[Numpy] N=5 Average Run time: 1.2138842428435704e-06
318  s ± 3.03  s per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
1.2  s ± 9.67 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.44  s ± 21.4 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Fast] N=10 Average Run Time: 0.0003178699285989361
[Numba] N=10 Average Run Time: 1.2011764857452365e-06
[Numpy] N=10 Average Run time: 1.4422060714901558e-06
1.3 ms ± 19.3  s per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
3.69  s ± 59.3 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
4.21  s ± 52.4 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
[Fast] N=20 Average Run Time: 0.0012992129428312184
[Numba] N=20 Average Run Time: 3.691176285834185e-06
[Numpy] N=20 Average Run time: 4.211141571535595e-06
8.12 ms ± 35.1  s per loop (mean ± std. dev. of 7 runs, 100 loops each)
54.8  s ± 745 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
7.83  s ± 110 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
[Fast] N=50 Average Run Time: 0.008120342999962823
[Numba] N=50 Average Run Time: 5.478265999949404e-05
[Numpy] N=50 Average Run time: 7.834984285623899e-06
18.3 ms ± 347  s per loop (mean ± std. dev. of 7 runs, 10 loops each)
215  s ± 3.16  s per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
13  s ± 116 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
[Fast] N=75 Average Run Time: 0.018311147139008555
[Numba] N=75 Average Run Time: 0.00021525839998918985
[Numpy] N=75 Average Run time: 1.3045093714525658e-05
32.5 ms ± 334  s per loop (mean ± std. dev. of 7 runs, 10 loops each)
518  s ± 6.93  s per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
24.2  s ± 533 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
[Fast] N=100 Average Run Time: 0.03252240999801351
[Numba] N=100 Average Run Time: 0.0005184733571285116
[Numpy] N=100 Average Run time: 2.4193337140604856e-05
143 ms ± 5.5 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
5.16 ms ± 14.5  s per loop (mean ± std. dev. of 7 runs, 100 loops each)
118  s ± 1.41  s per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
[Fast] N=200 Average Run Time: 0.14282738428695924
```

```
[Numba] N=200 Average Run Time: 0.005157236713649971
[Numpy] N=200 Average Run time: 0.00011829674285836517
336 ms ± 11.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
20 ms ± 200  s per loop (mean ± std. dev. of 7 runs, 100 loops each)
311  s ± 3.86  s per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
[Fast] N=300 Average Run Time: 0.3361030285845378
[Numba] N=300 Average Run Time: 0.020010562857413396
[Numpy] N=300 Average Run time: 0.0003110981285904668
611 ms ± 17.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
50.4 ms ± 401  s per loop (mean ± std. dev. of 7 runs, 10 loops each)
922  s ± 10.8  s per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
[Fast] N=400 Average Run Time: 0.6109964428802154
[Numba] N=400 Average Run Time: 0.05038833428573396
[Numpy] N=400 Average Run time: 0.0009222633000130632
973 ms ± 16 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
102 ms ± 1.47 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
1.76 ms ± 25  s per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
[Fast] N=500 Average Run Time: 0.9732830285626862
[Numba] N=500 Average Run Time: 0.1021284799989579
[Numpy] N=500 Average Run time: 0.001760904042848519
1.51 s ± 49.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
180 ms ± 1.95 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
2.92 ms ± 340  s per loop (mean ± std. dev. of 7 runs, 100 loops each)
[Fast] N=600 Average Run Time: 1.5098242285395307
[Numba] N=600 Average Run Time: 0.17956828714148806
[Numpy] N=600 Average Run time: 0.0029176618567933995
2.12 s ± 38.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
291 ms ± 4.95 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
4.22 ms ± 35.2  s per loop (mean ± std. dev. of 7 runs, 100 loops each)
[Fast] N=700 Average Run Time: 2.1216443856912
[Numba] N=700 Average Run Time: 0.2907759000081569
[Numpy] N=700 Average Run time: 0.0042188125715724055
3 s ± 35.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
431 ms ± 4.02 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
6.58 ms ± 204  s per loop (mean ± std. dev. of 7 runs, 100 loops each)
[Fast] N=800 Average Run Time: 2.999714071496523
[Numba] N=800 Average Run Time: 0.4310624856568341
[Numpy] N=800 Average Run time: 0.006575607714309756
3.96 s ± 61.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
625 ms ± 8.99 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
8.47 ms ± 102  s per loop (mean ± std. dev. of 7 runs, 100 loops each)
[Fast] N=900 Average Run Time: 3.9554865285754204
[Numba] N=900 Average Run Time: 0.6246559428649822
[Numpy] N=900 Average Run time: 0.008472736428957432
4.84 s ± 87.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
869 ms ± 12.8 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
12.2 ms ± 341  s per loop (mean ± std. dev. of 7 runs, 100 loops each)
[Fast] N=1000 Average Run Time: 4.841583871448945
```

```
[Numba] N=1000 Average Run Time: 0.869054099944021
[Numpy] N=1000 Average Run time: 0.01216575771403898
```



[38]:
```
def diff_memory_plots(numba_func):

    # Copy and pasting time_and_plot function, and changing to account for␣
 ↪nmemory layouts of matrices for C and Fortran.

    # Ensuring function works correctly
    test_matrix_function(numba_func)

    # Setting up execution timers

    f_and_f_times= []
    c_and_c_times= []
    c_and_f_times= []
    f_and_c_times= []

    matrix_size = [5,10,20,50,75]+list(range(100,1001,100))
```

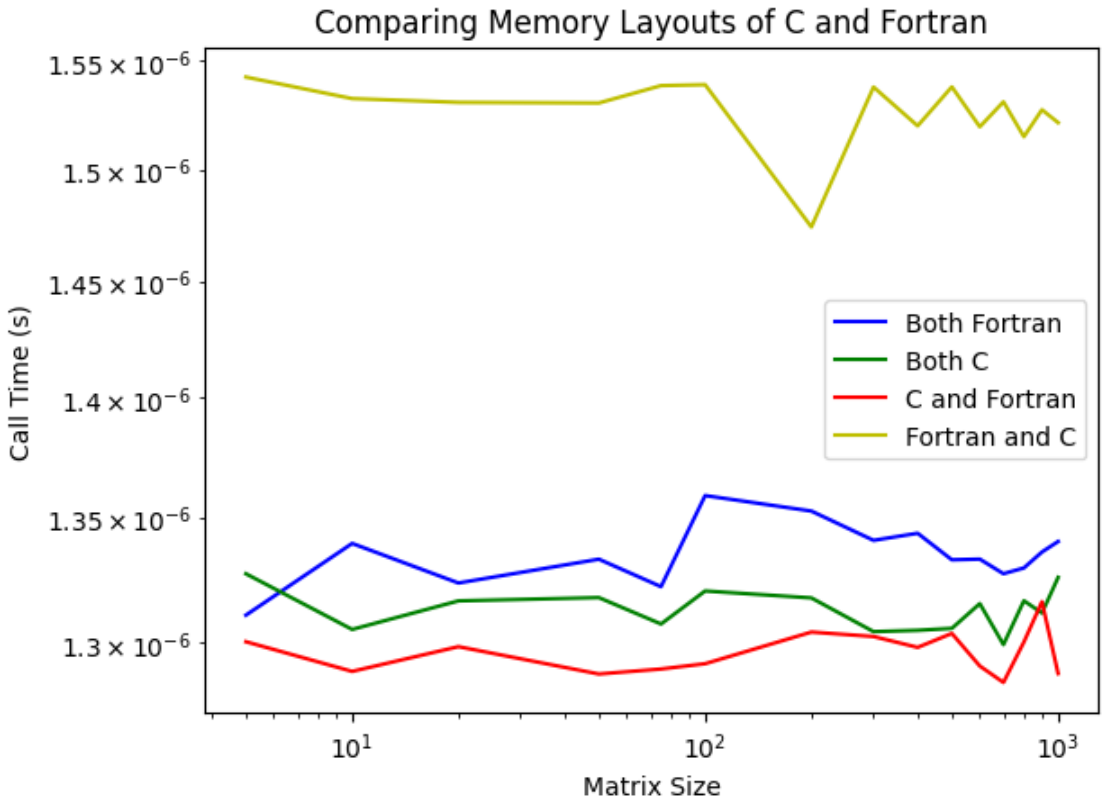
```
    for n in matrix_size:
        fortran_matrix = np.asfortranarray(np.random.rand(10, 10))
        c_matrix = np.ascontiguousarray(np.random.rand(10, 10))


        t_f_and_f= %timeit -o (numba_func(fortran_matrix,fortran_matrix))
        t_c_and_c= %timeit -o (numba_func(c_matrix,c_matrix))
        t_c_and_f= %timeit -o (numba_func(c_matrix,fortran_matrix))
        t_f_and_c= %timeit -o (numba_func(fortran_matrix,c_matrix))

        f_and_f_times.append(t_f_and_f.average)
        c_and_c_times.append(t_c_and_c.average)
        c_and_f_times.append(t_c_and_f.average)
        f_and_c_times.append(t_f_and_c.average)

        print(f"[Both Fortran] N={n} Average Run time: {t_f_and_f.average}")
        print(f"[Both C] N={n} Average Run Time: {t_c_and_c.average}")
        print(f"[C and Fortran] N={n} Average Run Time: {t_c_and_f.average}")
        print(f"[Fortran and C] N={n} Average Run Time: {t_f_and_c.average}")

    # Setting up plots

    plt.plot(matrix_size, f_and_f_times, "b", label="Both Fortran")
    plt.plot(matrix_size, c_and_c_times, "g", label="Both C")
    plt.plot(matrix_size, c_and_f_times, "r", label="C and Fortran")
    plt.plot(matrix_size, f_and_c_times, "y", label="Fortran and C")

    plt.xlabel("Matrix Size")
    plt.xscale("log")
    plt.yscale("log")
    plt.ylabel("Call Time (s)")
    plt.title("Comparing Memory Layouts of C and Fortran")
    plt.legend()


diff_memory_plots(numba_matrix_product)
```

```
1.31  s ± 29.9 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.33  s ± 13.3 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.3  s ± 16.7 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.54  s ± 12.1 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=5 Average Run time: 1.3107623428971108e-06
[Both C] N=5 Average Run Time: 1.3272859285519059e-06
[C and Fortran] N=5 Average Run Time: 1.3002785714010573e-06
[Fortran and C] N=5 Average Run Time: 1.5424398000551654e-06
1.34  s ± 16.6 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.31  s ± 18 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.29  s ± 21.7 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

```
1.53  s ± 16.6 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=10 Average Run time: 1.3395134142733045e-06
[Both C] N=10 Average Run Time: 1.3051141428295524e-06
[C and Fortran] N=10 Average Run Time: 1.288651700065072e-06
[Fortran and C] N=10 Average Run Time: 1.5324351428342717e-06
1.32  s ± 13 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.32  s ± 14.3 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.3  s ± 17.7 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.53  s ± 17.1 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=20 Average Run time: 1.3235677999577354e-06
[Both C] N=20 Average Run Time: 1.3164363857691309e-06
[C and Fortran] N=20 Average Run Time: 1.2983347428962589e-06
[Fortran and C] N=20 Average Run Time: 1.530675328602748e-06
1.33  s ± 9.52 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.32  s ± 11.3 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.29  s ± 18.8 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.53  s ± 14.3 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=50 Average Run time: 1.3331380428613295e-06
[Both C] N=50 Average Run Time: 1.3177450857391316e-06
[C and Fortran] N=50 Average Run Time: 1.287639885675162e-06
[Fortran and C] N=50 Average Run Time: 1.530387685700719e-06
1.32  s ± 7.78 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.31  s ± 16.7 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.29  s ± 11.9 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.54  s ± 18.5 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=75 Average Run time: 1.322112857164549e-06
[Both C] N=75 Average Run Time: 1.30726005712391e-06
[C and Fortran] N=75 Average Run Time: 1.2895864999653507e-06
[Fortran and C] N=75 Average Run Time: 1.5384535428602248e-06
1.36  s ± 27.6 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.32  s ± 11.3 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.29  s ± 20.1 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.54  s ± 21.1 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=100 Average Run time: 1.35904108563305e-06
[Both C] N=100 Average Run Time: 1.32038898573124e-06
[C and Fortran] N=100 Average Run Time: 1.2916536857041396e-06
[Fortran and C] N=100 Average Run Time: 1.5388775428624026e-06
1.35  s ± 10.9 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.32  s ± 22.2 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.3  s ± 12.7 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.47  s ± 8.36 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=200 Average Run time: 1.3526877713988403e-06
[Both C] N=200 Average Run Time: 1.3176196428747582e-06
[C and Fortran] N=200 Average Run Time: 1.3040812142592456e-06
[Fortran and C] N=200 Average Run Time: 1.4740745713934303e-06
1.34  s ± 13.7 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.3  s ± 10.1 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.3  s ± 23.1 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

```
1.54  s ± 16.1 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=300 Average Run time: 1.3407565142759785e-06
[Both C] N=300 Average Run Time: 1.3042414714249648e-06
[C and Fortran] N=300 Average Run Time: 1.3023215857267912e-06
[Fortran and C] N=300 Average Run Time: 1.5377901285953287e-06
1.34  s ± 22.1 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.3  s ± 17.8 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.3  s ± 11.3 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.52  s ± 19.8 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=400 Average Run time: 1.3436459142581693e-06
[Both C] N=400 Average Run Time: 1.3047669143415987e-06
[C and Fortran] N=400 Average Run Time: 1.298046871454322e-06
[Fortran and C] N=400 Average Run Time: 1.519831228568884e-06
1.33  s ± 12.6 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.31  s ± 19.4 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.3  s ± 15.4 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.54  s ± 26.5 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=500 Average Run time: 1.3328913142572022e-06
[Both C] N=500 Average Run Time: 1.3055336713857414e-06
[C and Fortran] N=500 Average Run Time: 1.3035378857582277e-06
[Fortran and C] N=500 Average Run Time: 1.5378822428839548e-06
1.33  s ± 19.7 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.32  s ± 18.4 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.29  s ± 12.4 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.52  s ± 12.5 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=600 Average Run time: 1.3331690142929023e-06
[Both C] N=600 Average Run Time: 1.3153538999280759e-06
[C and Fortran] N=600 Average Run Time: 1.290758985726695e-06
[Fortran and C] N=600 Average Run Time: 1.5193842143552112e-06
1.33  s ± 16.3 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.3  s ± 11.5 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.28  s ± 28.2 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.53  s ± 20.9 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=700 Average Run time: 1.3272999285254627e-06
[Both C] N=700 Average Run Time: 1.2991442713953022e-06
[C and Fortran] N=700 Average Run Time: 1.2843569571351898e-06
[Fortran and C] N=700 Average Run Time: 1.530939657041537e-06
1.33  s ± 26.9 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.32  s ± 19.3 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.3  s ± 19.1 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.51  s ± 13.4 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=800 Average Run time: 1.3296114571525584e-06
[Both C] N=800 Average Run Time: 1.3164957143100244e-06
[C and Fortran] N=800 Average Run Time: 1.3001674428648714e-06
[Fortran and C] N=800 Average Run Time: 1.5148568428454122e-06
1.34  s ± 20.3 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.31  s ± 15.2 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.32  s ± 15.2 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

```
1.53  s ± 17.6 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=900 Average Run time: 1.3360412999921079e-06
[Both C] N=900 Average Run Time: 1.3115925000248744e-06
[C and Fortran] N=900 Average Run Time: 1.3161094856914133e-06
[Fortran and C] N=900 Average Run Time: 1.5272670571160103e-06
1.34  s ± 19.5 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.33  s ± 19.3 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.29  s ± 11.9 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
1.52  s ± 23.3 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
[Both Fortran] N=1000 Average Run time: 1.3402966857289096e-06
[Both C] N=1000 Average Run Time: 1.3258023999431835e-06
[C and Fortran] N=1000 Average Run Time: 1.2878284856477488e-06
[Fortran and C] N=1000 Average Run Time: 1.5213262713846885e-06
```



The combination of C and Fortran array might be the fastest because more efficient memory access means it has the easiest time working through the loop, since values stored adjacent to one another are easier to access.