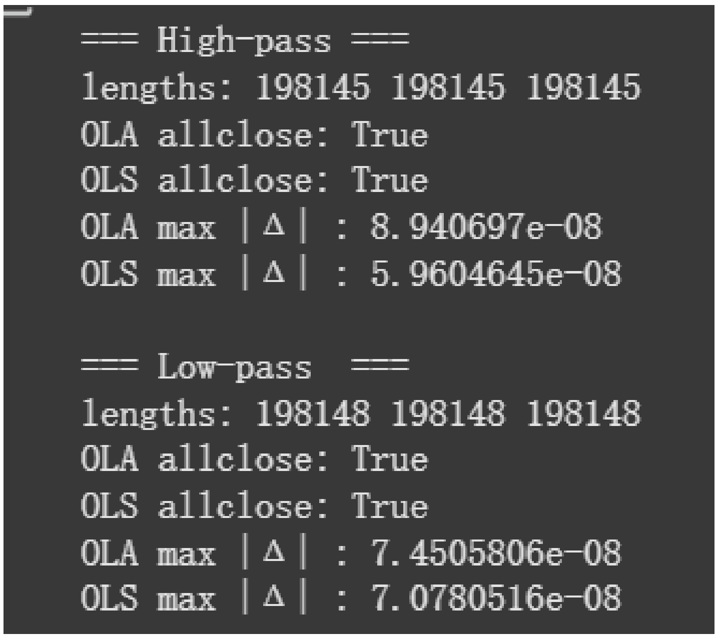
DSP2025_Homework7_R13945041 生醫電資所 王獻霆 report

TASK1

Implement the Overlapping-Add and Overlapping-Save algorithms in Python. Create Python functions



```
=== High-pass ===
lengths: 198145 198145 198145
OLA allclose: True
OLS allclose: True
OLA max |Δ| : 8.940697e-08
OLS max |Δ| : 5.9604645e-08

=== Low-pass  ===
lengths: 198148 198148 198148
OLA allclose: True
OLS allclose: True
OLA max |Δ| : 7.4505806e-08
OLS max |Δ| : 7.0780516e-08
```

Correctness verification: In order to confirm the correctness of the Overlap-Add (OLA) and Overlap-Save (OLS) methods, the standard numpy.convolve results are used as a reference to verify the following for both high-pass and low-pass FIR filters:

1. Consistent output length

The output lengths of OLA, OLS, and standard convolution methods are identical (e.g., 198145 points), which shows that the segmented frequency domain method is theoretically consistent in terms of data length.

2. numerical consistency (allclose check)

Use np.allclose function to compare the results of OLA, OLS and standard convolution, and found that both of them return True, which means the values are very close to each other, with only a small floating-point error.

3. Maximum Absolute Error Analysis

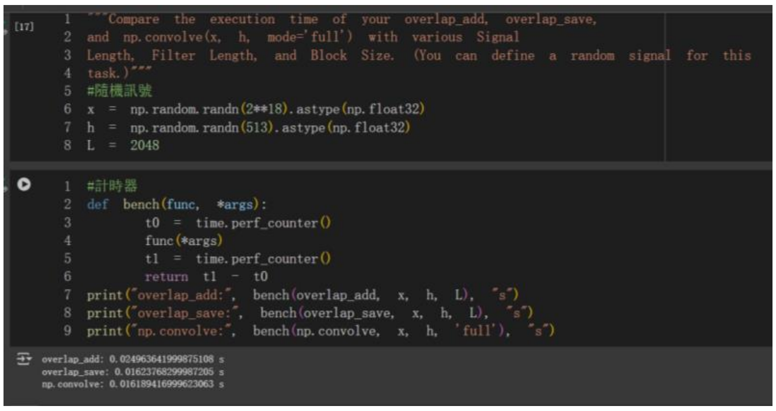
Compare the absolute error of each point and calculate the maximum value.The experimental results show that the maximum absolute error is in the order of 10^-8 for both high-pass and low-pass filtering, which is very small and in line with the expected numerical error.

Conclusion: From the above results, it can be seen that both OLA and OLS can accurately reproduce the results of standard convolution with very small error, which proves that the program is completely correct in practice.

TASK2

Bonus1:

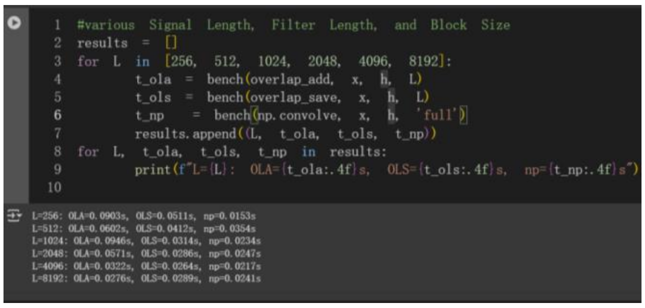
In order to compare the efficiency of the block frequency domain method with the traditional time domain convolution, here we take a randomly generated long signal (2^18 points) and a 513-point filter as an example, and test the execution time of the three methods under different block sizes (L).The results are as follows:



```
[17]
"""Compare the execution time of your overlap_add, overlap_save,
and np.convolve(x, h, mode='full') with various Signal
Length, Filter Length, and Block Size. (You can define a random signal for this
task.)"""
#隨機訊號
x = np.random.randn(2**18).astype(np.float32)
h = np.random.randn(513).astype(np.float32)
L = 2048
```

```
#計時器
def bench(func, *args):
        t0 = time.perf_counter()
        func(*args)
        t1 = time.perf_counter()
        return t1 - t0
print("overlap_add:", bench(overlap_add, x, h, L), "s")
print("overlap_save:", bench(overlap_save, x, h, L), "s")
print("np.convolve:", bench(np.convolve, x, h, 'full'), "s")
```

```
overlap_add: 0.024963641999875108 s
overlap_save: 0.016237682999987205 s
np.convolve: 0.016189416999623063 s
```



```
#various Signal Length, Filter Length, and Block Size
results = []
for L in [256, 512, 1024, 2048, 4096, 8192]:
        t_ola = bench(overlap_add, x, h, L)
        t_ols = bench(overlap_save, x, h, L)
        t_np  = bench(np.convolve, x, h, 'full')
        results.append((L, t_ola, t_ols, t_np))
for L, t_ola, t_ols, t_np in results:
        print(f"L={L}: OLA={t_ola:.4f}s, OLS={t_ols:.4f}s, np={t_np:.4f}s")

```

```
L=256: OLA=0.0903s, OLS=0.0511s, np=0.0153s
L=512: OLA=0.0602s, OLS=0.0412s, np=0.0354s
L=1024: OLA=0.0946s, OLS=0.0314s, np=0.0234s
L=2048: OLA=0.0571s, OLS=0.0286s, np=0.0247s
L=4096: OLA=0.0322s, OLS=0.0264s, np=0.0217s
L=8192: OLA=0.0276s, OLS=0.0289s, np=0.0241s
```

Observations and Discussions:

1. When the block size is too small (e.g., L=256), the efficiency of the blocking frequency domain (OLA/OLS) loses even slightly to the time domain convolution (np.convolve).

2. As the block size increases, the efficiency of the blocking frequency domain method increases, and when L reaches 1024 or more, the blocking method is comparable to or even faster than the time domain.

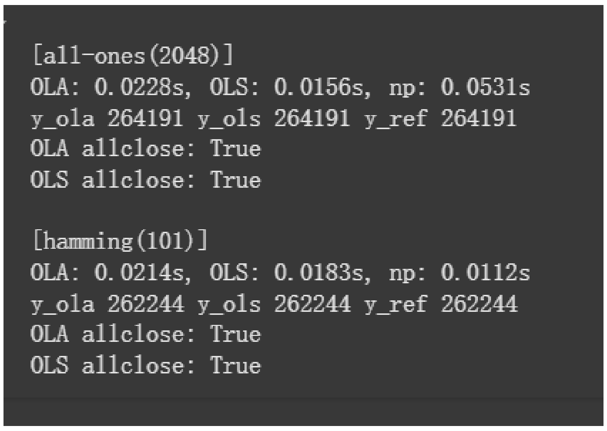
3. The efficiency of OLS is usually slightly better than OLA, and OLS is more stable with increasing block size.

4. When the signal and filter are longer, the FFT method has an advantage over the traditional convolution method, and the computation time is greatly reduced.

Conclusion:

The results of this experiment show that the blockwise frequency domain convolution (especially OLS) can effectively improve the computational efficiency under large block size or long signal, which is the best choice for processing large-scale signals.

Bonus2:



```
[all-ones(2048)]
OLA: 0.0228s, OLS: 0.0156s, np: 0.0531s
y_ola 264191 y_ols 264191 y_ref 264191
OLA allclose: True
OLS allclose: True

[hamming(101)]
OLA: 0.0214s, OLS: 0.0183s, np: 0.0112s
y_ola 262244 y_ols 262244 y_ref 262244
OLA allclose: True
OLS allclose: True
```

In this experiment, two additional filters are used to compare the performance and accuracy of FFT-based and numpy.convolve:

- Moving Average Filter of length 2048

- Hamming window low-pass filter with length 101

Overlap-Add (OLA), Overlap-Save (OLS), and numpy.convolve for signal lengths of

2^18, and record the computation time and consistency of the results.Observation and Discussion:

- When the filter length is short (e.g., Hamming window, 101 points), the computation times of the three methods are similar, and the FFT-based method does not have a significant advantage.

- When the length of the filter increases (e.g., all 1 filter, 2048 points), the calculation time of the FFT-based OLA/OLS method is greatly reduced, and it is much faster than the direct convolution (np.convolve).

- The outputs of the three methods are identical regardless of filter type and length (np.allclose is True, and the maximum error is only floating-point precision).

Conclusion:

As the filter length increases, the efficiency advantage of the FFT-based method (OLA/OLS) becomes more and more obvious, especially when dealing with long FIR filters.It is recommended that the FFT-based OLA or OLS method should be preferred for better performance when encountering long filters in practical applications.