

Task 1.

We wrote MATLAB scripts for calculating OLS and OLA. The algorithms were tested against MATLAB's linear convolution function (`conv()`). Both scripts are included in the return files.

Overlap-add:

```
OLA convolved data:
Columns 1 through 21
-2    7   -15    5   31   -82   119  -104    29   44   -57   -3   90  -131   107  -77   72  -62    6   62  -78

Columns 22 through 42
22    43   -76   92  -116   126  -106   76   -32   -37   82   -48   -1    5    9   -6    1    0    0    0    0
```

Figure 1: Convolved data

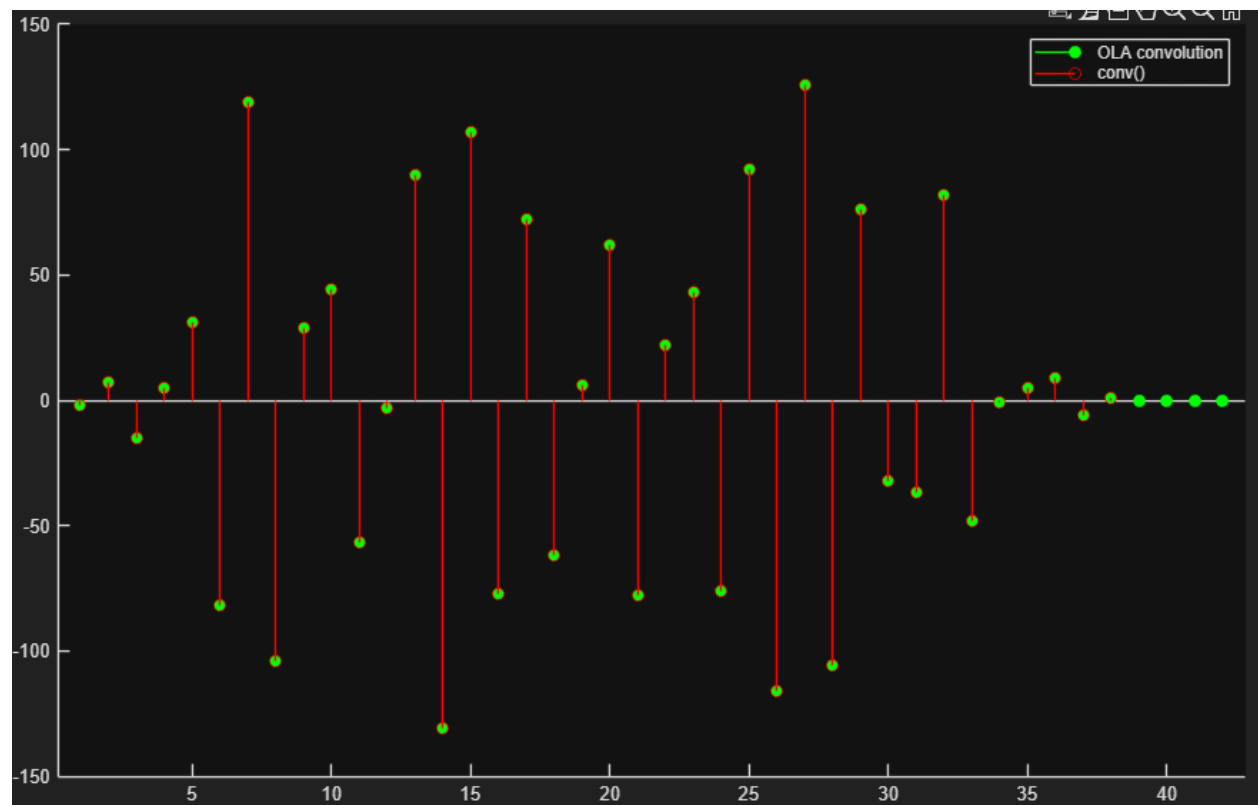


Figure 2: OLA vs `conv()`

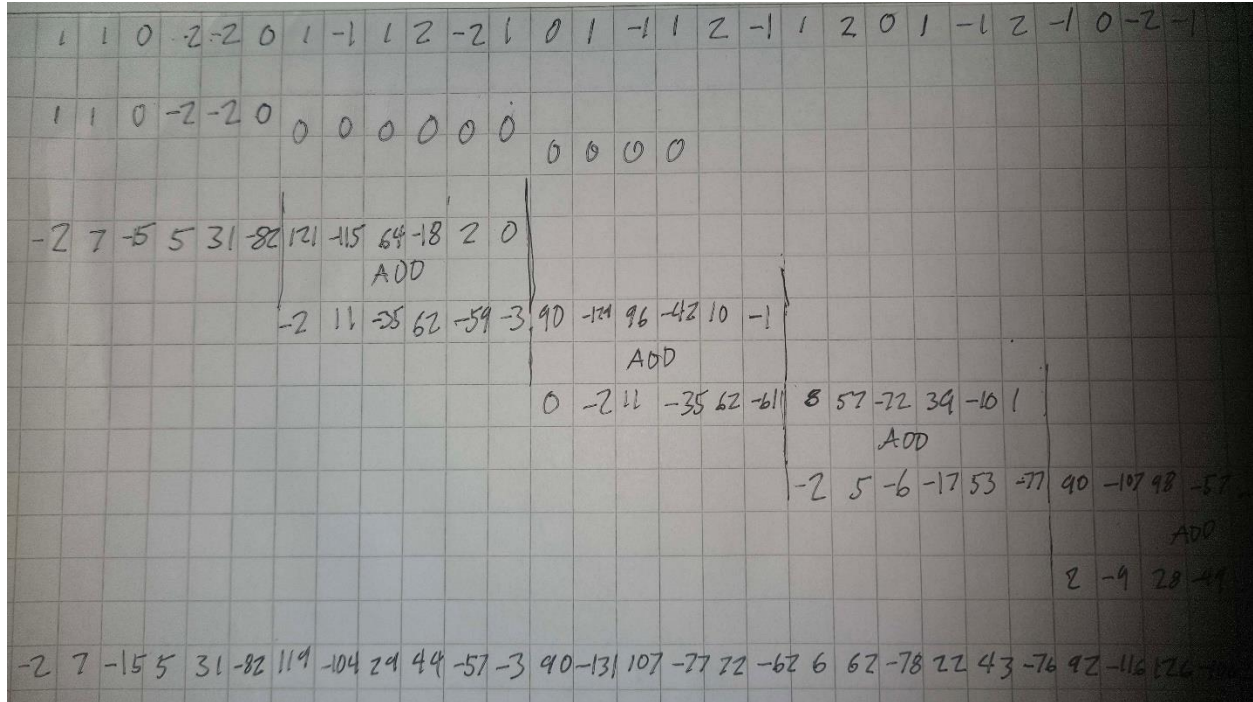


Figure 3: Sketch of OLA convolution

Overlap-save:

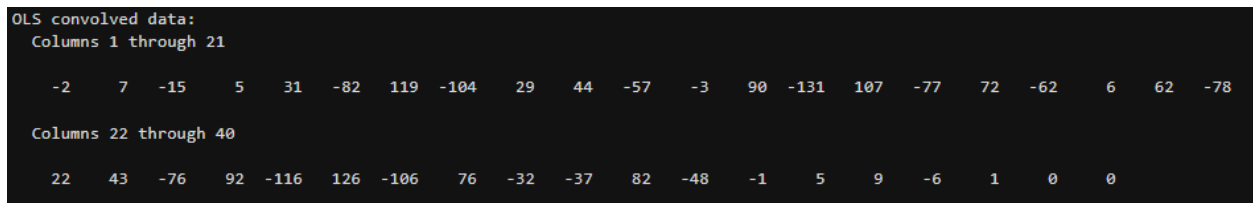


Figure 4: OLS convolved data

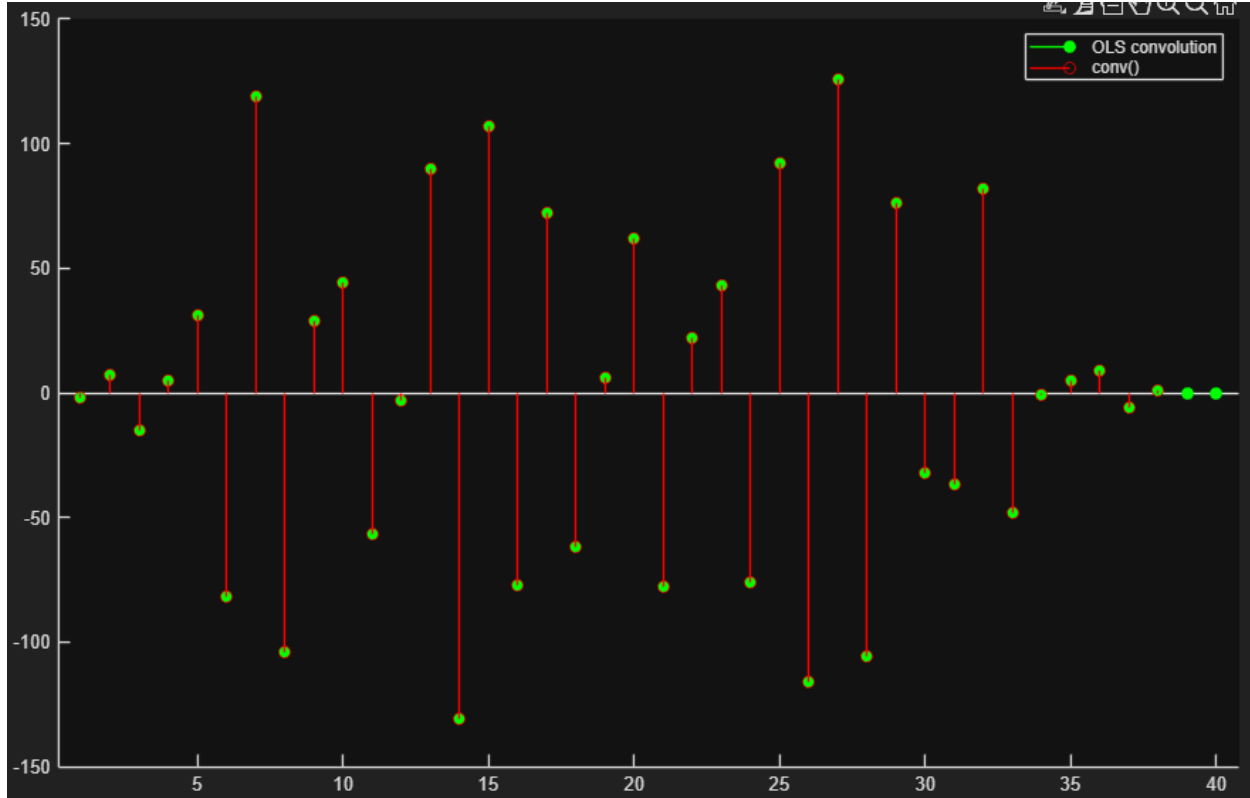


Figure 5: OLS vs conv()

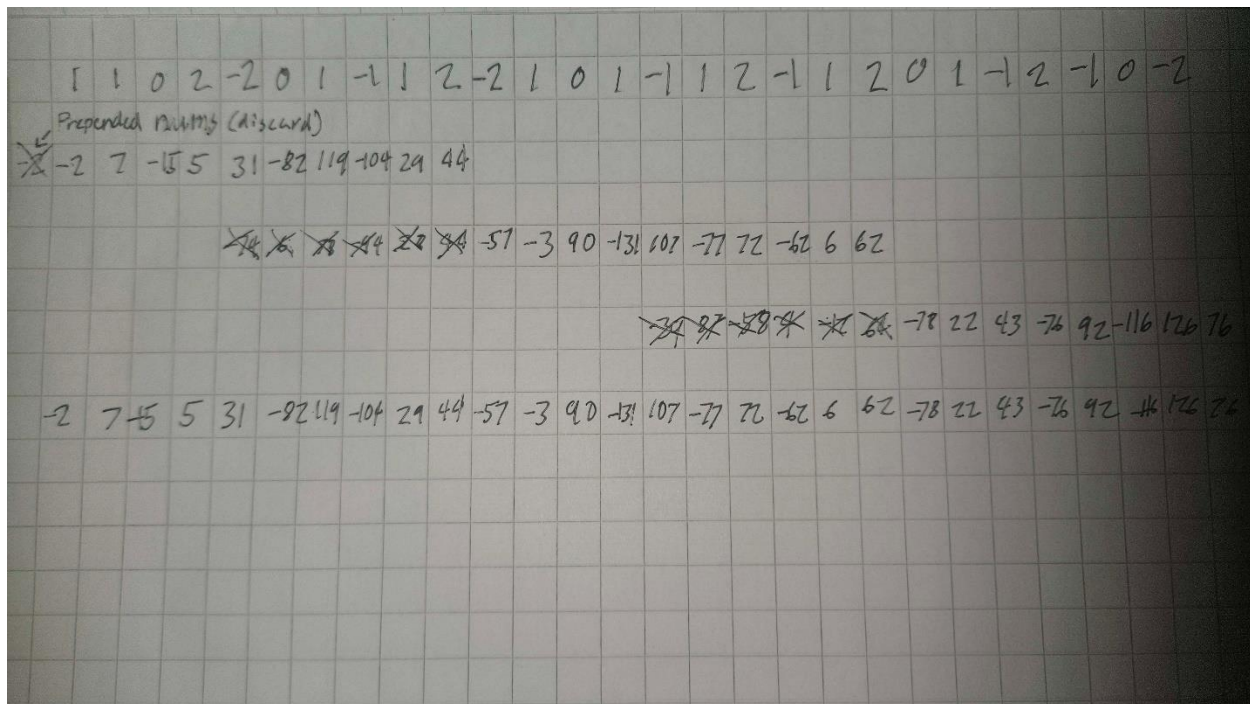


Figure 6: Sketch of OLS convolution

Task 2.

N	f _{in} [kHz]
281	190

a)

There are 10 FIR filters of length N, which all operate at full sample rate. There are 8 complex-valued filters and 2 real-valued ones. The input is complex. We can't exploit symmetry because the input and (some) coefficients are complex. Although the filter bank is critically downsampled, we won't use noble identities yet.

From opcounts.pdf:

Filter with N coefficients:

- *Both coefficients and input real: N MPLYs and N – 1 ADDs*
- *Coefficients complex and input real (or vice versa): 2N MPLYs and 2N – 2ADDs are needed as real and imaginary parts are processed separately*
- *Both coefficients and input complex: 4N MPLYs and 4N – 2 ADDs are needed, if the computation is based on the Eq. 4*

So we have:

$$\text{ADDs:} \quad 8(4N - 2) + 2(2N - 2) = 8976 + 1120 = 10096$$

$$\text{MPLYs:} \quad 8(4N) + 2(2N) = 8992 + 1124 = 10116$$

Per second:

$$\text{ADDs:} \quad 10096 \cdot f_{in} = 1918240 \text{ kADDs / s}$$

$$\text{MPLYs:} \quad 10116 \cdot f_{in} = 1922040 \text{ kMPLYs / s}$$

b)

In this case we can use a noble identity to move the downsamplers upstream from the FIRs. In practice, this means that each FIR has to process 1/10th of the elements in the signal. Otherwise, the problem remains the same.

$$\text{ADDs:} \quad 8(4N - 2) + 2(2N - 2) = 8976 + 1120 = 10096$$

$$\text{MPLYs:} \quad 8(4N) + 2(2N) = 8992 + 1124 = 10116$$

$$\text{Decimated sample rate: } f_{out} = f_s/10 = 19 \text{ kHz}$$

Per second:

$$\text{ADDs:} \quad 10096 \cdot f_{\text{out}} = 191824 \text{ kADDs / s}$$

$$\text{MPLYs:} \quad 10116 \cdot f_{\text{out}} = 192204 \text{ kMPLYs / s}$$

This implementation gives a 90 % reduction in real operations per second compared to the original direct implementation.

c)

After decomposing the prototype filter, the 281 coefficients are distributed among the 10 subfilters. This means that the total number of coefficients goes down to 281.

There are 2 real-valued and 8 complex-valued filters. We approximate that the real FIRs have $\frac{2N}{10}$ of the taps and the complex FIRs have $\frac{8N}{10}$ of the taps. The IDFT takes 20 MPLYs and 84 ADDs.

$$\text{ADDs:} \quad 2 \cdot \left(\frac{2N}{10} - 2 \right) + 8 \cdot \left(\frac{4N}{10} - 2 \right) + 84 = 991.6 + 84 = 1075.6 \text{ ADDs}$$

$$\text{MPLYs:} \quad 2 \cdot \frac{2N}{10} + 8 \cdot \frac{4N}{10} + 20 = 1011.6 + 20 = 1031.6 \text{ MPLYs}$$

We decimate before filtering.

$$\text{Decimated sample rate: } f_{\text{out}} = f_s/10 = 19 \text{ kHz}$$

Per second:

$$\text{ADDs:} \quad 1075.6 \cdot f_{\text{out}} = 20436.4 \text{ kADDs / s}$$

$$\text{MPLYs:} \quad 1031.6 \cdot f_{\text{out}} = 19600.4 \text{ kMPLYs / s}$$

Operation count from each step:

	a	b	c
ADDs / s	1918240	191824	20436.4
MPLYs / s	1922040	192204	19600.4

There is a 90 % reduction in real operations from a to b, and a ~90 % reduction in real operations from b to c. Overall, the computation load is decreased by 99 % from a to c. That is a massive optimization.