MUSI6202 Assignment 1
Punahamoa Walker and Yi Wu

## Convolution

The purpose of this assignment is to investigate two different methods of convolution and determining which is more efficient. These two methods are:

- Implementing convolution in the time domain by reversing a signal and calculating the product with changing amounts of delay.
- Converting the signals into the frequency domain, multiplying the responses together and converting the output back into the time domain.

Due to the properties of the Fourier transform, both of these methods are equivalent though one will be more efficient than the other.

## Time Domain Convolution

Convolution in the time domain is defined as:

$$x * h = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$

Where *x* and *h* are two signals and $\tau$ is the delay between the two signals. In MATLAB, this was implemented using a for loop to implement the change in $\tau$ and vector multiplication to multiply the overlapping portion of the signals. Since this integral ranges from ∞ to -∞, if signals *x* and *h* are of finite length, many of the values of the convolution will be zero. The amount of non-zero entries is equal to the sum of the length of the two signals minus 1. The equation is:

$$length(y) = \text{length}(x) + \text{length}(h) - 1 = 200 + 100 - 1 = 299$$

Therefore, *x* is 200 entries long and *h* is 100 entries long, the convolution will have a length of 299 entries.
The figure below shows the output of the myTimeConv() function written in MATLAB which utilises the aforementioned algorithm. In this case, *x* is a 200 entry long DC signal of amplitude 1 and *h* is a triangular signal of length 51 and max value of 1.

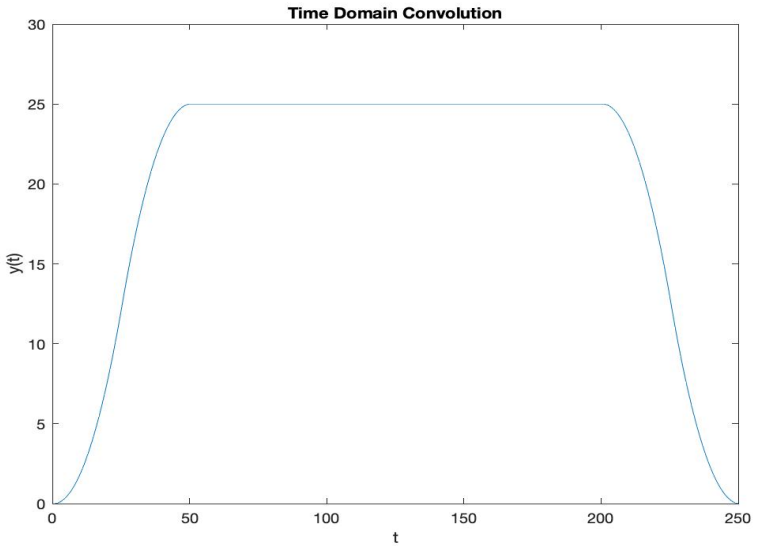


*Figure 1: Output of myTimeConv()*

## Frequency Domain Convolution

In order to implement convolution in the frequency domain, the two signals first need to be converted using the MATLAB function fft(). This performs a fast Fourier transform on the signals and converts them to the frequency domain. Once both signals are in the frequency domain, they can be multiplied together. This resulting signal is then converted back into the time domain using the ifft() function which implements an inverse fast Fourier transform. The result of this function is shown below in figure 2.

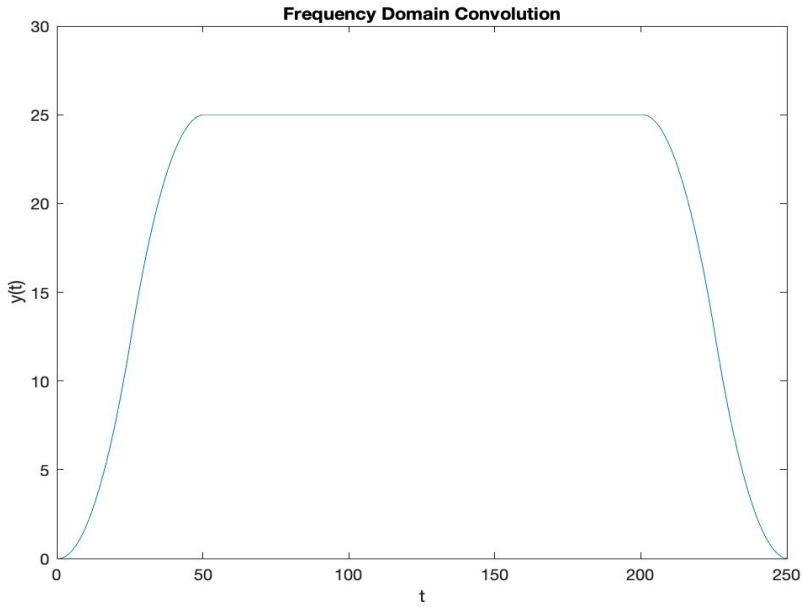


*Figure 2: Frequency Domain Convolution*

In this instance, the input signals are the same as in the time domain convolution with *x* being a DC signal of amplitude 1 and *h* being a triangular signal with a max amplitude of 1 and a length of 51. In order to make this function work identically to the predefined conv() function in MATLAB, zero-padding must be implemented in order to avoid mixing convolution results due to circular convolution. Since the length of the output when conv() is used is longer than two input vectors zero-padding is required for the outputs to be identical.

Comparison with MATLAB Implementation

Both of these produce a similar output to that of the MATLAB function conv(). However, there are differences in the performances of these 3 functions. In order to test this, a function compareConv() was written in order to compare the 3 methods. These functions were tested with two sets of inputs, the DC waveform and triangular waveform from the previous 2 sections, and also 'piano.wav.' and 'impulse-response.wav'. The results for these are outlined in the tables below.

**'x': DC signal of length 200**
**'h': symmetric triangular signal of length 51**

| Method | Time(s) |
|---|---|
| conv() | 0.0010 |
| myTimeConv() | 0.0006 |
| myFreqConv() | 0.0001 |

**Table. 1 Time Comparison**

| Comparison | y_time | y_freq |
|---|---|---|
| Mean Difference with conv() | $1.0e-15 * -0.0763$ | $1.0e-15 * -0.3137$ |
| Mean Absolute difference with conv() | $1.0e-15 * 0.0763$ | $1.0e-15 * 0.3137$ |
| Standard Deviation of the Difference with conv() | 0 | 0 |

**Table. 2 Output Difference with conv() Statistics**

**'x': 'piano.wav'**
**'h': 'impulse-resopnse.wav'**

| Method | Time(s) |
|---|---|
| conv() | 0.0302 |
| myTimeConv() | 2.2373 |
| myFreqConv() | 0.0209 |

**Table. 3 Time Comparison**

| Comparison | y_time | y_freq |
|---|---|---|
| Mean Difference with conv() | 0 | $1.0e-18 * -0.1730$ |
| Mean Absolute difference with conv() | 0 | $1.0e-18 * 0.1730$ |
| Standard Deviation of the Difference with conv() | 0 | 0 |

**Table. 4 Output Difference with conv() Statistics**

These three functions all produce virtually identical outputs. This is shown in tables 2 and 4 with the mean error and mean standard deviation being 0 or very close to 0. There a difference in the efficiency of each method and this is demonstrated in tables 1 and 3. From tables 1 we can see that all three functions perform at a similar speed with the DC signal and triangle wave with myFreqConv() being the fastest, myTimeConv() being the next fastest and conv() being the slowest. This is because the input signals are relatively short. When we input 'piano.wav' and 'impulse-response.wav' the differences in efficiency become more apparent due to the length of the signals. Under these circumstances, myTimeConv() takes approximately 100 times longer to complete than myFreqConv(). Our frequency convolution method is the most time efficient method in this case and Matlab's built in conv() function come very close to it. This is because the frequency domain convolution relies on vector multiplication while the time domain convolution requires a large loop which requires more calculations and therefore is more inefficient.