

# 6.003: Signal Processing

## Filtering with DFT

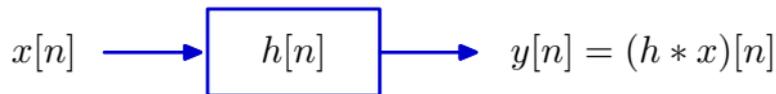
*22 October 2020*

## Filtering

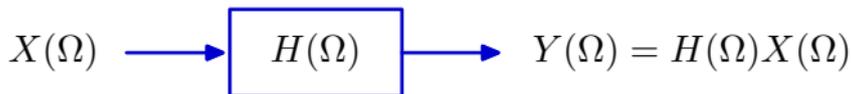
---

We can view filtering in both the time and frequency domains:

Time Domain:



Frequency Domain:



Each frequency component  $X(\Omega)$  is scaled by a factor  $H(\Omega)$ , which can be possibly complex.

The system is completely described by the set of scale factors  $H(\cdot)$ , which we refer to as the **frequency response** of the system.

## Filtering

---

**In lecture 8B:** constructed a time-domain filter (described by its unit sample response  $h[\cdot]$ ) by reasoning about its frequency response  $H(\cdot)$ .

Our focus was on creating a filter that boosted low frequency content in an input signal. We started by first creating a *low-pass* filter that removes high frequencies.

To this end, we started with a filter defined by  $H(\Omega) = \frac{1}{2} + \frac{1}{2} \cos(\Omega)$ , which has the nice property that it goes to 0 around  $\Omega = \pm\pi$  and it goes to 1 at  $\Omega = 0$  (i.e., the highest frequencies are the most attenuated).

Let's explore this filter a little bit, and see what happens when we apply it to a piece of music.

## Check Yourself!

---

If  $H(\Omega) = \frac{1}{2} + \frac{1}{2} \cos(\Omega)$ , what is the shape of  $h[n]$ ?

For what values of  $n$  is  $h[n]$  nonzero?

## Check Yourself!

---

If  $H_2(\Omega) = (H(\Omega))^2 = \left(\frac{1}{2} + \frac{1}{2} \cos(\Omega)\right)^2$ , what is the shape of  $h_2[n]$ ?

For what values of  $n$  is  $h_2[n]$  nonzero?

## Check Yourself!

---

How about  $H_L(\Omega) = (H(\Omega))^{1000} = \left(\frac{1}{2} + \frac{1}{2} \cos(\Omega)\right)^{1000}$ ?

What does  $h_L[n]$  look like? How many nonzero values do we expect?

## Check Yourself!

---

$h_L[\cdot]$  was a decent low-pass filter, but our original goal was bass boost.

Let  $x[\cdot]$  be our original input, and  $(x * h_L)[\cdot]$  represent a low-passed version of  $x[\cdot]$ .

We want to use a single convolution to produce a new signal with the high frequencies still present, but the low frequencies amplified:

$$(x * h_B)[n] = x[n] + c(x * h_L)[n]$$

Find an expression for  $h_B[n]$ .

## Downsides of This Approach

---

The approach we took last time was to determine a desired frequency response  $H(\cdot)$ , compute the unit sample response  $h[\cdot]$  of that filter, and then convolve an input signal with  $h[\cdot]$ .

This approach worked reasonably well, but it has a few downsides:

- Required a frequency response for which  $h[\cdot]$  is reasonably straightforward to calculate.
- Long unit sample responses  $h[\cdot]$  can mean that the convolution itself can take a long time.

There are many ways to overcome these issues (some of which are beyond the scope of 6.003). We'll take a look at one such approach today.

## Filtering in the Frequency Domain

---

Could we have avoided these issues by working exclusively in the frequency domain?

Yes, kind of. We certainly could have multiplied  $X(\cdot)$  by some desired frequency response  $H(\cdot)$  and then converted back:

$$y[n] = (x * h)[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\Omega)H(\Omega)e^{j\Omega n} d\Omega$$

But this approach is not super great, either. Why?

## An Alternative Approach: Use the DFT

---

If we know the frequency response we're interested in, we can take a different approach:

- compute the DFT of  $x[\cdot]$
- scale each frequency component by the appropriate  $H(\Omega)$  value
- compute the inverse DFT

For our low-pass filter, this involves scaling certain frequency components by some positive value, and leaving the others unchanged.

## Check Yourself!

---

What are all of the issues with the following code for applying such a filter?

```
from lib6003.fft import fft, ifft
from lib6003.audio import wav_read, wav_write

x, fs = wav_read('super.wav')
N = len(x)

fc = 400 # cutoff frequency in Hz; boost all frequency content below fc

X = fft(x)
Y = X[:] # copy values from X
Y[:fc] = [i*5 for i in X[:fc]]

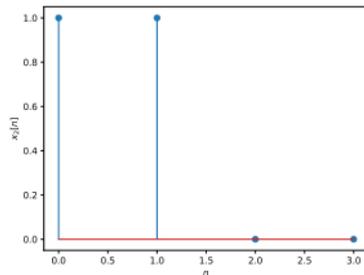
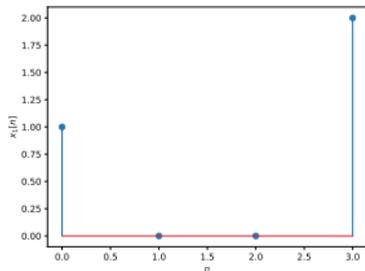
y = ifft(Y)
assert all(abs(i.imag) < 1e-12 for i in y)
wav_write([i.real for i in y], fs, 'super_bb.wav')
```

## Filtering with DFT

---

What if we want to apply a filter for which we don't know the frequency response  $H(\Omega)$ , but we *do* know the unit sample response  $h[n]$ ?

Consider two simpler signals, shown below (and assumed to be zero outside the indicated range):



What does  $(x_1 * x_2)[n]$  look like?

What does the following signal look like?

```
y = ifft([i*j for i,j in zip(fft(x1), fft(x2))])
```

## Filtering with DFT

---

```
from lib6003.fft import *  
import matplotlib.pyplot as plt
```

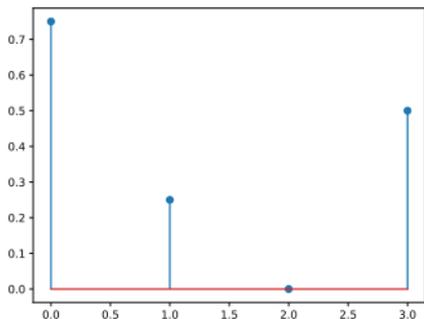
```
x1 = [1, 0, 0, 2]
```

```
x2 = [1, 1, 0, 0]
```

```
y = ifft([i*j for i,j in zip(fft(x1), fft(x2))])
```

```
plt.stem([i.real for i in y])
```

```
plt.show()
```



:O

:(

## Filtering with DFT

---

Let's look more closely to see why we get this surprising result!

$$x_1[n] = \delta[n] + 2\delta[n - 3]$$

$$x_2[n] = \delta[n] + \delta[n - 1]$$

Let  $y = (x_1 * x_2)[n]$ , then  $Y(\Omega) = X_1(\Omega)X_2(\Omega)$

$$X_1(\Omega) = 1 + 2e^{-j3\Omega}$$

$$X_2(\Omega) = 1 + e^{-j\Omega}$$

$$Y(\Omega) = 1 + e^{-j\Omega} + 2e^{-j3\Omega} + 2e^{-j4\Omega}$$

Double checking, we find that

$$y[n] = \delta[n] + \delta[n - 1] + 2\delta[n - 3] + 2\delta[n - 4] \quad \checkmark$$

## Check Yourself!

---

Let's now think about  $Y'[k] = \frac{1}{N}Y\left(\frac{2\pi k}{N}\right)$ . With  $N = 4$ , we have:

$$\begin{aligned} Y'[k] &= \frac{1}{4}Y\left(\frac{\pi}{2}k\right) \\ &= \frac{1}{4}\left(1 + e^{-j\frac{\pi}{2}k} + 2e^{-j\frac{3\pi}{2}k} + 2e^{-j\frac{4\pi}{2}k}\right) \end{aligned}$$

If we compute the inverse DFT to find  $y[n]$ , what do we find?

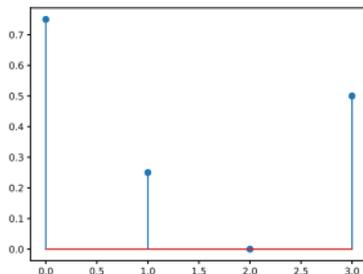
## Extra Scaling Factor

---

What about the extra scaling factor? We have

$$y[n] = y[n + 4] = 3\delta[n] + \delta[n - 1] + 2\delta[n - 3]$$

But that's not quite what we saw in our graph from earlier:



It has the right shape, but differs in scale.

## Extra Scaling Factor

---

Remember that  $Y'[k] = \frac{1}{4} \left( 1 + e^{-j\frac{\pi}{2}k} + 2e^{-j\frac{3\pi}{2}k} + 2e^{-j\frac{4\pi}{2}k} \right)$

We computed the values in that plot by multiplying the DFT of  $x_1[\cdot]$  by the DFT of  $x_2[\cdot]$ . Let's see where the math takes us if we do out that analysis:

## Generalizing

---

Comparing multiplication of DFTs to multiplication of DTFTs, we found that multiplying DFTs:

- led to aliasing of some samples in the convolution formula (if  $(x * h)[n]$  was nonzero for  $n \geq N$  or  $n < 0$ , those values will alias back to values in the range  $0 \leq n < N$ ).
- led to an additional scaling factor of  $\frac{1}{N}$

We'll formalize these two ideas on the following slide

## Circular Convolution

---

Multiplication of DFTs corresponds to **circular** convolution (and scaling by  $1/N$ ) in time. Assume that  $F[k]$  is the product of the DFTs of  $f_1[n]$  and  $f_2[n]$ .

$$\begin{aligned} f[n] &= \sum_{k=0}^{N-1} F[k] e^{j\frac{2\pi k}{N}n} = \sum_{k=0}^{N-1} F_1[k] F_2[k] e^{j\frac{2\pi k}{N}n} \\ &= \sum_{k=0}^{N-1} F_1[k] \left( \frac{1}{N} \sum_{m=0}^{N-1} f_2[m] e^{-j\frac{2\pi k}{N}m} \right) e^{j\frac{2\pi k}{N}n} \\ &= \frac{1}{N} \sum_{m=0}^{N-1} f_2[m] \sum_{k=0}^{N-1} F_1[k] e^{j\frac{2\pi k}{N}(n-m)} \\ &= \frac{1}{N} \sum_{m=0}^{N-1} f_2[m] f_{1p}[n-m] \end{aligned}$$

where  $f_{1p}[n] = f_1[n \bmod N]$  is a periodically extended version of  $f_1[n]$ .

We refer to this as **circular** or **periodic** convolution:

$$\frac{1}{N} (f_1 \circledast f_2)[n] \stackrel{\text{DFT}}{\iff} F_1[k] \times F_2[k]$$

## Circular Convolution

---

$$X_1[k] \times X_2[k] \stackrel{\text{DFT}}{\iff} \frac{1}{N} (x_1 \circledast x_2)[n]$$

where  $\circledast$  denotes the **circular convolution** operator.

The result of circular convolution is equivalent to:

- a periodically-extended version (periodic in  $N$ ) of the result of convolving the two signals:

$$(x \circledast h)[n] = \sum_{m=-\infty}^{\infty} (x * h)[n - mN]$$

- a convolution of one of the signals with a periodically-extended version of the other:

$$(x \circledast h)[n] = (x * h_p)[n], \text{ where } h_p[n] = \sum_{m=-\infty}^{\infty} h[n - mN]$$