7.3

February 21, 2019

```
In [1]: # Content from Proakis
        # Code © 2019, Alexander Kain
        import numpy as np
        from numpy.fft import fft, ifft, rfft, irfft

        from matplotlib import pyplot as plt
        %matplotlib inline
        plt.rcParams['figure.figsize'] = (10.0, 8.0)

        import sympy as sym
        sym.init_printing(use_unicode=True)
```

0.1 7.3 Linear Filtering Methods Based on the DFT

We can use the DFT as a computational tool for linear filtering of systems with finite impulse responses. Due to the existence of the FFT this approach is often more efficient than time-domain convolution.

0.1.1 7.3.1. Use of the DFT in Linear Filtering

We know that the product of two DFTs is equivalent to circular convolution. We seek a methodology that is equivalent to linear convolution.

Suppose we have $x[n]$ of length L which excites an FIR filter with impulse response $h[n]$ of length M:

$$x[n] = 0, \quad n < 0 \text{ and } n \geq L$$

$$h[n] = 0, \quad n < 0 \text{ and } n \geq M$$

In time domain, the output sequence $y[n]$ can be computed by convolving $x[n]$ and $h[n]$

$$y[n] = \sum_{k=0}^{M-1} h[k]x[n-k]$$

The length of $y[n]$ will be exactly $L + M - 1$.

The frequency domain equivalent is

$$Y(\omega) = X(\omega)H(\omega)$$

If the sequence $y[n]$ is to be represented uniquely in the frequency domain by samples of its spectrum $Y(\omega)$, the number of distinct samples must equal or exceed $L+M-1$. Therefore, a DFT of size $N \geq L+M-1$ is required and

$$Y[k] = X[k]H[k]$$

where $X[k]$ and $H[k]$ are the N-point DFTs of the sequences $x[n]$ and $h[n]$.

This implies that the $N \geq L+M-1$-point circular convolution of $x[n]$ with $h[n]$ is *equivalent* to linear convolution.

```
In [2]: # Example 7.3.1

        x = np.r_[1, 2, 2, 1]
        h = np.r_[1, 2, 3]
        L = len(x)
        M = len(h)
        N = L + M - 1

        # linear convolution
        y = np.zeros(N)
        for n in range(N):
            for k in range(M):
                ix = n-k
                if 0 <= ix < L:
                    y[n] += h[k] * x[ix]

        assert np.allclose(y, np.convolve(x, h))  # TD library implementation
        y

Out[2]: array([ 1.,  4.,  9., 11.,  8.,  3.])

In [3]: # aside: list comprehension version
        y = np.array([
                np.sum([h[k] * x[n-k] for k in range(M) if 0 <= (n-k) < L])
                for n in range(N)])

        assert np.allclose(y, np.convolve(x, h))

In [4]: X = rfft(x, N)  # equivalent to zero-padding
        H = rfft(h, N)
        Y = X * H
        y = irfft(Y)

        from scipy.signal import fftconvolve
        assert np.allclose(y, fftconvolve(x,h))  # FD library implementation
        y

Out[4]: array([ 1.,  4.,  9., 11.,  8.,  3.])
```

```
In [5]: fftn = 1 << (N-1).bit_length()  # FFT performs best with powers of two
        print(fftn)

        X = rfft(x, fftn)
        H = rfft(h, fftn)
        Y = X * H
        y = irfft(Y)[:N]  # keep only what is needed

        assert np.allclose(y, fftconvolve(x,h))  # FD library implementation
        y
```

```
8
```

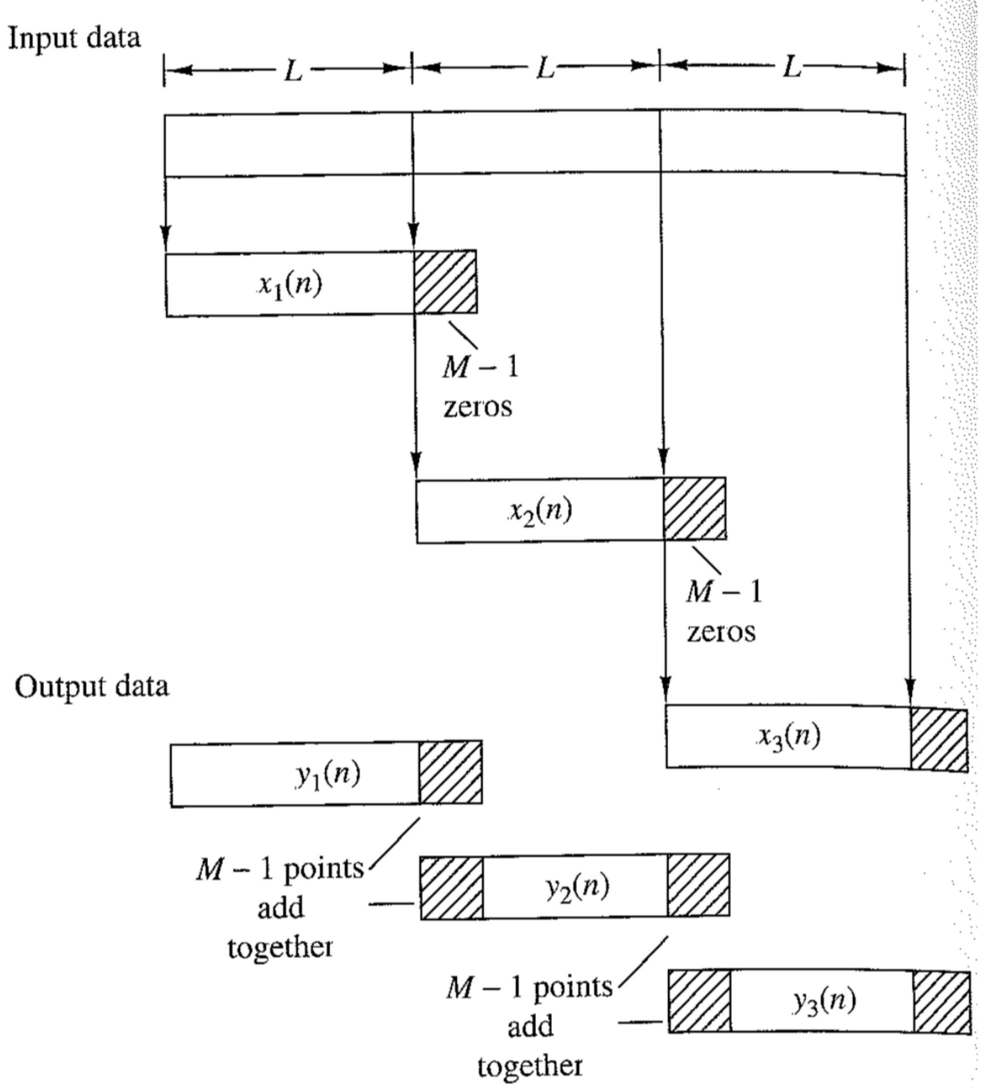
```
Out[5]: array([ 1.,  4.,  9., 11.,  8.,  3.])
```

0.1.2 7.3.2. Filtering of long Data Sequences

A long signal must be segmented to fixed-size blocks prior to processing, which will then be processed one *segment/block/frame* at a time, and then fit back together.

Overlap-add method In this method, the size of the block is L (it is assumed that $L >> M$), and the size of the DFTs is $N \geq L + M - 1$.

The *last* $M - 1$ points from each block must be overlapped and added to the *first* $M - 1$ points of the succeeding block.



Input data
L
L
L
$x_1(n)$
$M - 1$ zeros
$x_2(n)$
$M - 1$ zeros
Output data
$x_3(n)$
$y_1(n)$
$M - 1$ points add together
$y_2(n)$
$M - 1$ points add together
$y_3(n)$