DFT-Based FIR Filtering

See Proakis & Manolakis 7.3
Motivation: DTFT View of Filtering

There are two views of filtering:

* Time Domain
* Frequency Domain

The FD viewpoint is indispensable for analysis and design of filters:

* Passband, Stopband, etc. of \(|H^f(\theta)|\)
* Linearity of Phase \(\angle H^f(\theta)\), etc.

Q: What about using DTFT for implementation?

* Compute DTFT of input signal and filter
* Multiply the two and take inverse DTFT

A: NO!! Can’t compute DTFT – must compute at infinite many frequency values
Desired Intention: But Does It Work?

But wait…

• If input signal is finite length, the DFT computes “samples of the DTFT”
• Likewise, if filter impulse response is finite length

Q: So… can we use this?

\[
x(0), x(1), \ldots, x(N-1)
\]

\[
N\text{-pt DFT} \quad X[f[k] \quad H[f[k]X[f[k] \quad N\text{-pt IDFT}
\]

\[
y(n) = h[n]x[n]?
\]

\[
h(0), h(1), \ldots, h(N-1)
\]
DFT Theory and Cyclic Convolution

A: Not Necessarily!!!!

DFT Theory (Sect. 7.2.2 in Proakis & Manolakis) tells us:

$$\text{IDFT}\{H^f[k]X^f[k]\} = h[n] \oplus x[n]$$

Thus… this block diagram gives something called cyclic convolution, not the “ordinary” convolution we want!!!

Q: When does it work???
A: Only when we “trick” the DFT Theory into making circular = linear convolution!!!!!!

Q: So… when does Cyclic = Linear Convolution???

Easiest to see from an example!!!!!
Linear Convolution for the Example

What does **linear convolution** give for 2 finite duration signals:

Original Signals:

![Graph of x[n] and h[n]](attachment:image)

Length $N_1 = 9$

Length $N_2 = 5$

First Non-Zero Output is at $n=0$:

![Graph of x[n] and h[-n]](attachment:image)

(flip, no shift – since $n=0$, multiply and add up)
Linear Convolution for the Example (cont.)

Last Non-Zero Output is at \( n = N_1 + N_2 - 2 = 12 \):

The non-zero outputs are for \( n = 0, 1, \ldots, 12 \) \Rightarrow 13 of them

In General: Length of Output of Linear Convolution = \( N_1 + N_2 - 1 \)
Cyclic Convolution for the Example

Now… What does *cyclic convolution* give for these 2 signals:

“Original” Signals:
1. Zero-Pad Shorter Signal to Length of Longer One
2. Then Periodize Each

“First” Output Sample:
1. Flip periodized version around this point
2. No shift needed to get $n = 0$ Output Value
3. Sum over one cycle

Same as in Linear Conv!!!!

Not Present in Linear Conv!!!!
Linear Convolution for the Example (cont.)

“Last” Output Sample (i.e., \( n = 8 \)):

1. Flip periodized version around this point
2. Shift by 8 to get \( n = 8 \) Output Value
3. Sum over one cycle

Note: If I try to compute the output for \( n = 9 \) \( \Rightarrow \) Exactly the same case as for \( n = 0 \)!!
Thus, the output is cyclic (i.e., periodic) with unique values for \( n = 0, 1, \ldots, 8 \)

In General: Length of Output of Cyclic Convolution = \( \max \{ N_1, N_2 \} \)
Making Cyclic = Linear Convolution

So…. Some of the output values of cyclic conv are different from linear conv!!!
   Some of the output values of cyclic conv are same as linear conv
And….   
The length of cyclic conv differs from the length of linear conv!!!

From the example above we can verify:
   If we choose \( K \geq N_1 + N_2 - 1 \)
   And Zero-Pad Each Signal to Length \( K \)
   Then Cyclic = Linear Convolution

\[
N_1 = \text{Length of } x[n] \\
N_2 = \text{Length of } h[n] \\
\]

Exercise: Verify this for the above example!!!!
**Simple Frequency-Domain Implementation of FIR Filtering**

- **Zero-Pad Both to Length** $K \geq N_1 + N_2 - 1$

- **Why Do This?**
  The FFT’s Efficiency Makes This Faster Than Time-Domain Implementation (In Many Cases)

- **The DFT of** $h(n)$ **is usually Pre-Computed**

- If $K$ is Strictly $> N_1 + N_2 - 1$
  Then there will be extra zeros here that can be ignored
Problems with the *Simple* FD Implementation

Q: What if $N_1 >> N_2$?
A: Then, need **Really Big FFT** ➔ Not Good!!!
   (Input signal much longer than filter length)
   Also… can’t get any output samples until after whole signal is available and FFT processing is done. **Long Delay**.

   **Example**: Filter 0.2 sec of a radar signal sampled at $F_s = 50$ MHz
   
   $N_1 = (0.2 \text{ sec}) \times (50 \times 10^6 \text{ samples/sec}) = 10^7 \text{ samples}$
   
   FFT Size $> 10^7$ ➔ **Really Big FFT!!!!

Q: What if $N_1$ is unknown in advance?
   **Example**: Filtering a stream of audio
A: FFT size can’t be set ahead of time – difficult programming

*Simple* FD Implementation Has Serious Limitations!!!
Better FD-Based FIR Filter Implementations

Two Very Similar Methods Exist
- Overlap – Add (OLA)
- Overlap – Save (OLS)

Both methods exploit linearity of filter:
- Break input signal into a sum of blocks
- Output = sum of response to each block

The difference between OLA & OLS lies in how the \( x_i[n] \) blocks are formed

\[
x[n] = \sum_{i} x_i[n]
\]

\[
z[n] = (x \ast h)[n] = \sum_{i} (x_i \ast h)[n] = \sum_{i} z_i[n]
\]
OLA Method for FD-Based FIR

For OLA: Choose $x_i[n]$ to be non-overlapped blocks of length $N_B$
(block are contiguous)

$$x_i[n] = \begin{cases} 
  x[n], & iN_B \leq n < (i+1)N_B \\
  0, & \text{otherwise}
\end{cases}$$

$N_B$ is a Design Choice

<See Fig. 5.8 (a) on next slide>

Q: Now what happens when each of these length-$N_B$ blocks gets convolved with the length-$N_2$ filter?

A: The output block has length $N_2 + N_B - 1 > N_B$

* Output Blocks are Bigger than Input Blocks
* But are separated by $N_B$ points
* Thus… Output Blocks Overlap
* Total Output = “Sum of Overlapped Blocks”

<See Fig. 5.8 (b) – (d) on next slide>
Figure 5.8 Illustration of overlap-add convolution: (a) the input signal $x[n]$; (b), (c), (d) the three output segments $z_{i-1}[n]$ through $z_{i+1}[n]$; (e) a segment of the output signal $z[n]$ corresponding to $x_i[n]$ and $x_{i+1}[n]$. 
OLA Method Steps

Assume: Filter $h[n]$ length $N_2$ is specified

Choose: Block Size $N_B$ & FFT Size $N_{FFT} = 2^p \geq N_B + N_2 - 1$

Choose $N_B$ such that:

$$N_B + N_2 - 1 = 2^p$$

It gives minimal complexity for method (see below)

Run:

Zero-Pad $h[n]$ & Compute $N_{FFT}$-pt FFT (can be pre-computed)

For each $i$ value (“For Each Block”)

- Compute $z_i[n]$ using Simple FD-Based Method
  - Zero-Pad $x_i[n]$ & Compute $N_{FFT}$-pt FFT
  - Multiply by FFT of $h[n]$  
  - Compute IFFT to get $z_i[n]$

- Overlap the $z_i[n]$ with previously computed output blocks
- Add it to the output buffer

<See Fig. 5.8 (e) on previous slide>
OLA Method Complexity

- The FFT of filter \( h[n] \) can be pre-computed \( \Rightarrow \) Don’t Count it!
- We’ll measure complexity using \# Multiplies/Input Sample
- Use \( 2N_{FFT}\log_2 N_{FFT} \) Real Multiplies as measure for FFT
- Assume input samples are Real Valued
  
  Can do 2 real-signal FFT’s for price of \( \approx 1 \) Complex FFT (Classic FFT Result!)

- For Each Pair of Input Blocks
  - One FFT: \( 2N_{FFT}\log_2 N_{FFT} \) Real Multiplies
  - Multiply DFT \( \times \) DFT: \( 4N_{FFT} \) Real Multiplies
  - One IFFT: \( 2N_{FFT}\log_2 N_{FFT} \) Real Multiplies
  - Total: \( 4N_{FFT} [1 + \log_2 N_{FFT}] \) Real Multiplies
    
    \[ = 4(N_B + N_2 - 1) [1 + \log_2(N_B + N_2 - 1)] \]

- The Number of Input Samples = 2 Blocks = \( 2N_B \)

- \# Multiplies/Input Sample = \( 2(1 + (N_2 - 1)/ N_B) [1 + \log_2(N_B + N_2 - 1)] \)
Comparison to TD Method Complexity

Complexity of TD Method
• Filter \( h[n] \) has length of \( N_2 \)
• To get each output sample:
  ▶ Multiply each filter coefficient by a signal sample: \( N_2 \) Multiplies
• \# Multiplies/Input Sample = \( N_2 \) Multiplies

Condition Needed For OLA to Be More Efficient:

\[
2 \left( 1 + \frac{N_2 - 1}{N_B} \right) \left[ 1 + \log_2 \left( N_2 + N_B - 1 \right) \right] < N_2
\]

Thus… For a given \( N_2 \), Choose \( N_B \) to minimize the left-hand side
FD Complexity vs TD Complexity

Plot of: \[
\frac{\text{Left-Hand Side}}{\text{Right-Hand Side}} \text{ of (5.38)}
\]

Contours = \((\text{OLA Multiplies})/(\text{TD Multiplies})\)

Optimal Block Size For Filter Length of \(\approx 125\) (Compare to Table 5.2 in Porat)

OLA More Efficient Only For Filters Longer than 19