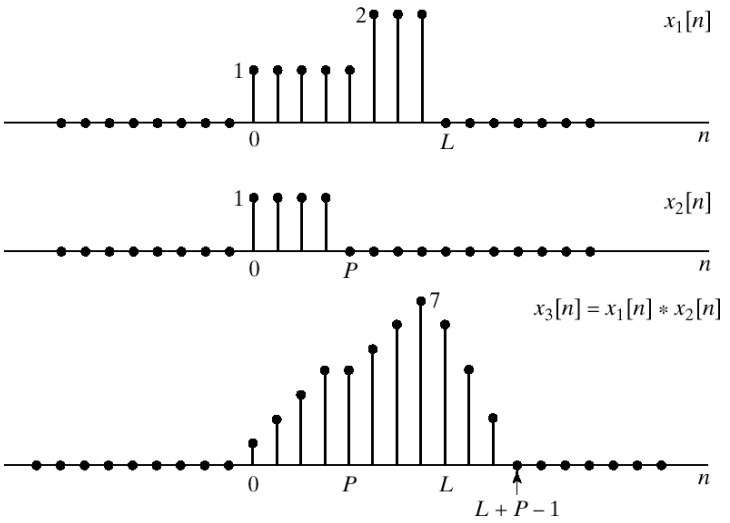
ECE 4270
Fundamentals of Digital Signal Processing

Lecture 21: Block Convolution

School of Electrical and Computer Engineering
Georgia Institute of Technology
Summer 2004

In a Circular (Aliased) Convolution, Some Parts of the Output Equal the Linear Convolution, and Some Don't …

Linear Convolution Example Again



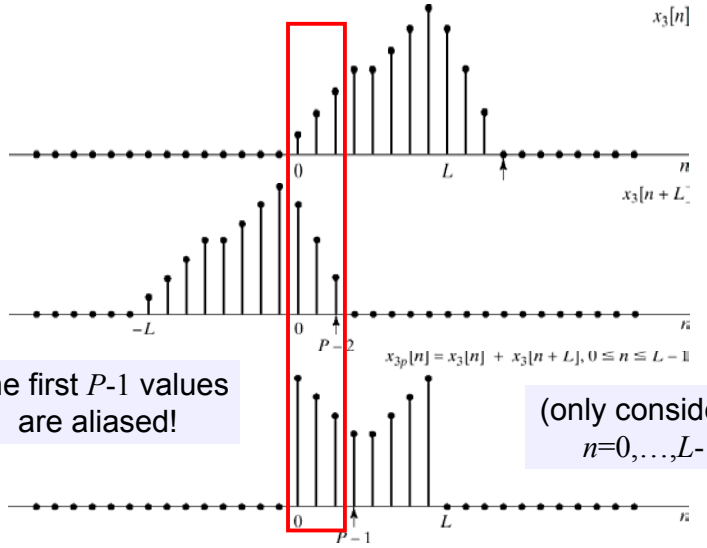
Aliased Convolution - 1

- If we tried to implement the convolution on the previous slide by multiplying the L-point DFTs of the two sequences, the output would be the linear convolution, but repeated (aliased) every L samples:

$$\tilde{x}_3[n] = \sum_{r=-\infty}^{\infty} x_3[n - rN] = x_3[n \text{ modulo } N] = x_3[((n))_N]$$

- If we examine the values in the principal interval of n=0,…,L-1, what do we find?

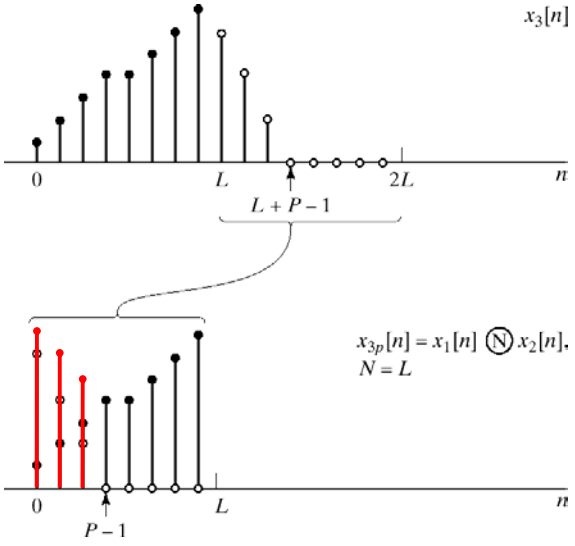
Aliased Convolution - 2



The first P-1 values are aliased!

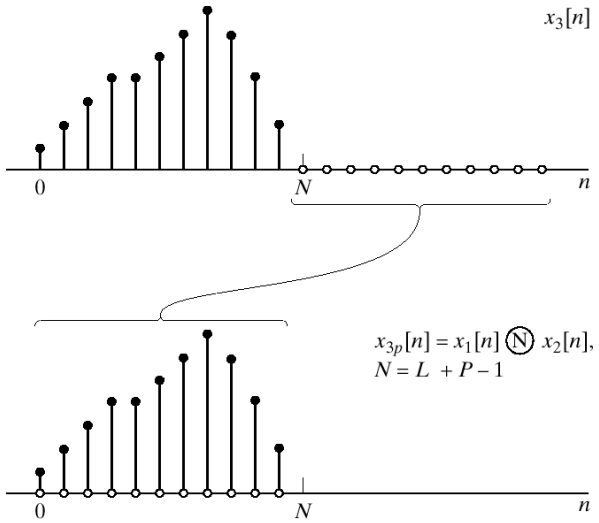
(only considering n=0,…,L-1)

Aliased Convolution - 3



Another way to view it: the second block of L samples is aliased back onto the first block of L samples

Aliased Convolution - 4



If the DFT is big enough, namely $N \ge L+P-1$, then the "aliasing" simply overlaps zeroes onto the principal interval

Terminology: Fast Convolution

- We can accomplish the convolution of two finite-length sequences by computing the convolution sum directly:

$$y[n] = \sum_{n=0}^{L+P-2} x[m]\, h[n-m]$$

- Or by computing the DFT of each sequence, multiplying DFTs, and computing the inverse DFT of the product:

$$y[n] = DFT_N^{-1}\{DFT_N(x[n]) \bullet DFT_N(h[n])\} \quad N \ge L+P-1$$

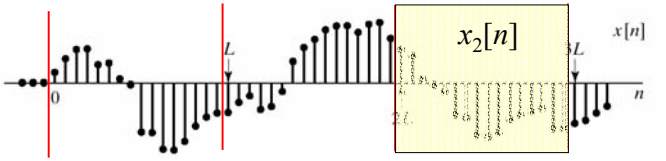
- The latter approach is called "fast convolution" because (assuming FFTs are used), it often requires fewer multiplications

Filtering Long Sequences

- Sometimes we want to filter a sequence that is very long
  - could save up all the samples, then either
    - » do a really long time-domain convolution, or
    - » use really big DFTs to do it in the frequency domain
  - but big DFTs may become impractical; besides
  - we get long *latency*: we have to wait a long time to get any output
- Sometimes we want to filter a sequence of indefinite length
  - and then even the methods above don't work

Using Linearity to Filter a Long Signal - 1

- Any long sequence can be broken up into shorter blocks:



$$x_r[n] = \begin{cases} x[n+rL], & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$$

$$x[n] = \sum_{r=0}^{\infty} x_r[n-rL]$$

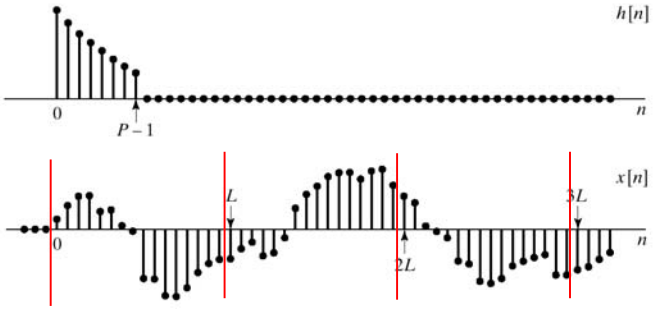
Using Linearity to Filter a Long Signal - 2

- We can then perform a linear operation on the long signal by
  - performing it on each of the shorter segments, and
  - combining them, using linear shift-invariance, to form the complete output signal
- Specifically, for linear filtering:

$$x[n] = \sum_{r=0}^{\infty} x_r[n-rL] \rightarrow y[n] = x[n] * h[n] = \sum_{r=0}^{\infty} y_r[n-rL]$$

$$y_r[n] = x_r[n] * h[n]$$

Block Convolution - Overlap Add Method

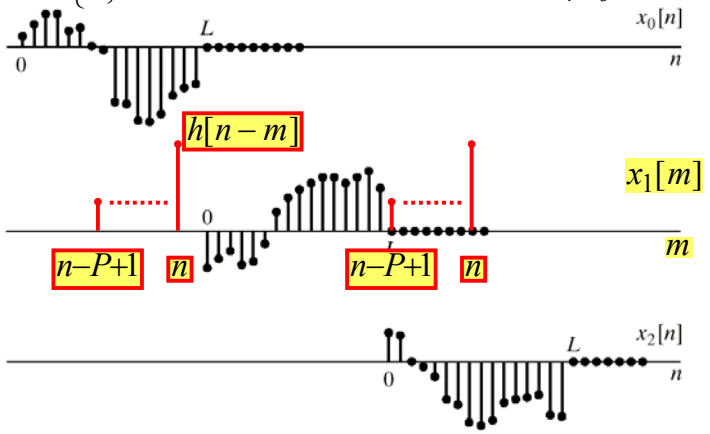


$$x[n] = \sum_{r=0}^{\infty} x_r[n-rL] \rightarrow y[n] = x[n] * h[n] = \sum_{r=0}^{\infty} y_r[n-rL]$$

$$x_r[n] = \begin{cases} x[n+rL], & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases} \rightarrow y_r[n] = x_r[n] * h[n]$$

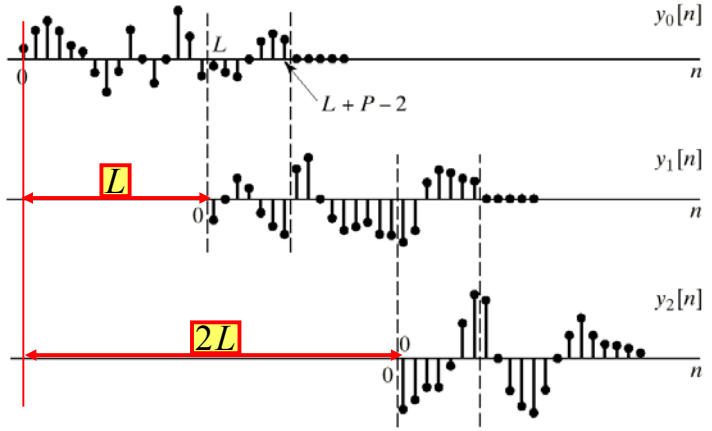
Segmenting the Input in OLA

$$x_r[n] = \begin{cases} x[n+rL], & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases} \qquad x[n] = \sum_{r=0}^{\infty} x_r[n-rL]$$



Putting the Output Pieces Together

$$y[n] = \sum_{r=0}^{\infty} y_r[n-rL] \qquad y_r[n] = x_r[n] * h[n]$$



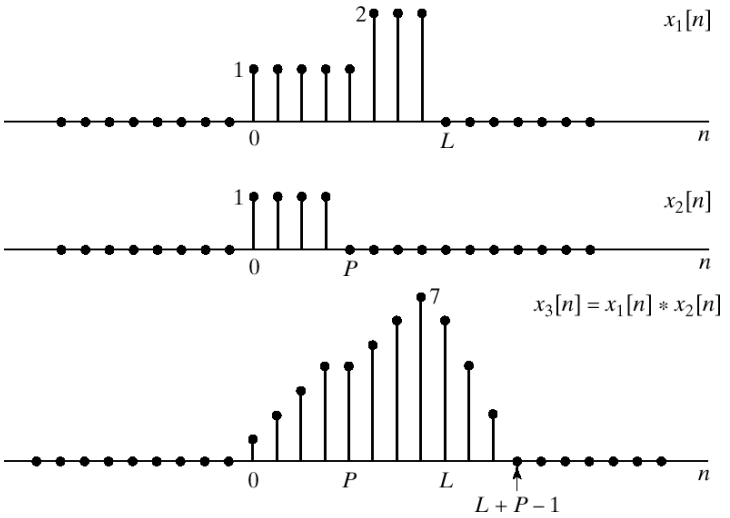
Filtering the Segments

- The Overlap-Add (OLA) method calls for filtering each segment separately, then adding the results
- The filtering of the individual segments can be done by any legitimate means
  - time-domain convolution
  - frequency domain "fast convolution" using DFTs (implemented with the FFT algorithm)
    - » DFT size N should be at least $L+P-1$ so that you get a linear convolution result for each individual segment

Block Filtering with Circular Convolution

- Alternatively, we can use a smaller DFT and allow the convolution of the segments to be circular instead of linear
  - $N = \max\{L,P\}$
  - fewer multiplications per DFT this way
- We saw earlier that in this case, only some of the output values of the circular convolution are equal to samples of the linear convolution
- The Overlap-Save (OLS) method of block convolution uses circular convolutions and retains only the "good" samples to build up the output

Linear Convolution Example Again



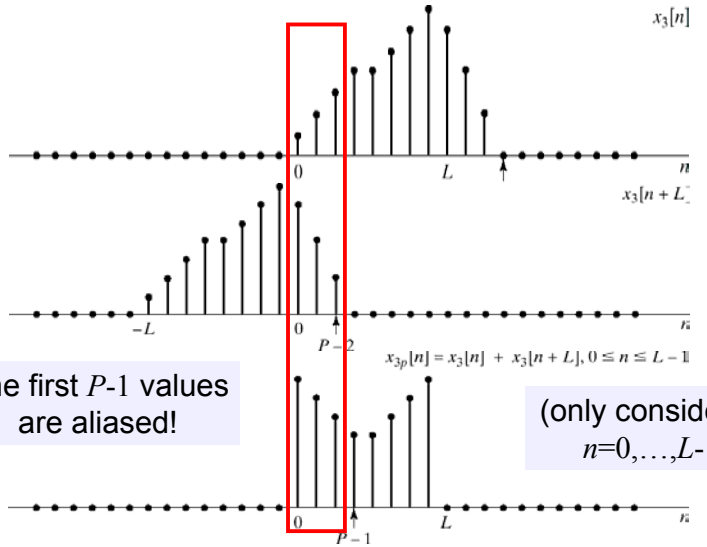
Aliased Convolution - 1

- If we tried to implement the convolution on the previous slide by multiplying the L-point DFTs of the two sequences, the output would be the linear convolution, but repeated (aliased) every L samples:

$$\tilde{x}_3[n] = \sum_{r=-\infty}^{\infty} x_3[n - rN] = x_3[n \text{ modulo } N] = x_3[((n))_N]$$

- If we examine the values in the principal interval of n=0,…,L-1, what do we find?

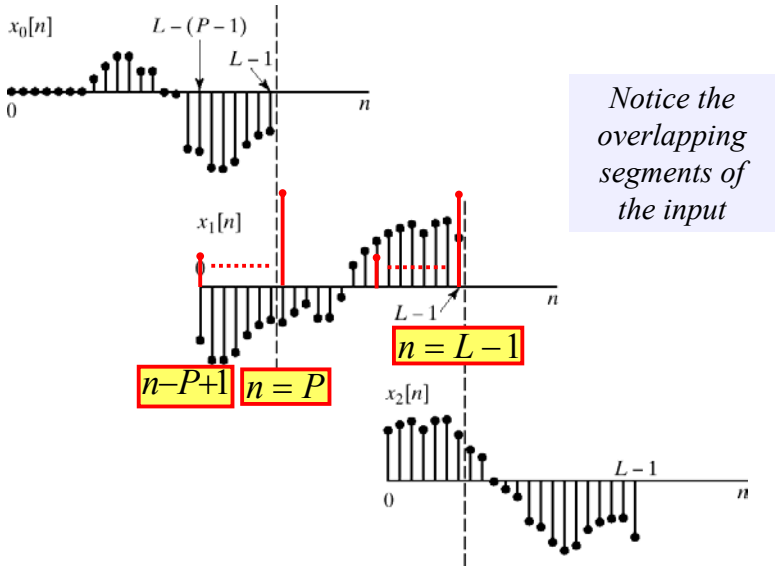
Aliased Convolution - 2



The first P-1 values are aliased!

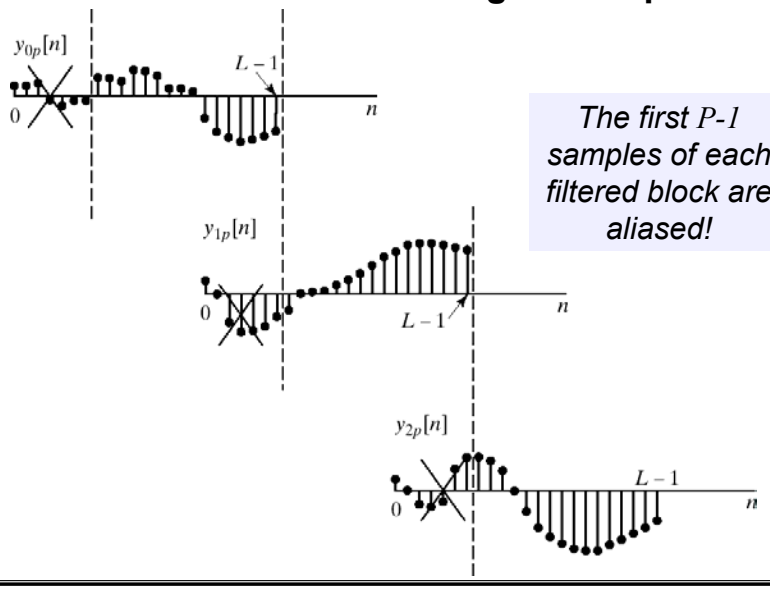
(only considering n=0,…,L-1)

OLS Method - Segmenting the Input



Notice the overlapping segments of the input

OLS Method - Extracting the Output



The first P-1 samples of each filtered block are aliased!

Efficient Computation of the DFT: The Fast Fourier Transform, or FFT

Computation Required

- We'll count complex multiplies; the number of complex adds is about the same
- Let $\mu(N)$ be the number of multiplies needed to compute an N-point DFT using an FFT algorithm, and let $\beta(N)$ be the number of multiplies needed to compute an N-point DFT in brute-force fashion
- Thus we start with

$$\beta(N) = N^2$$

The Goal of "the" FFT Algorithm ...

- … is to compute the DFT of size N with significantly fewer than N^2 complex multiplications and additions
- To accomplish this, researchers have come up with entire families of fast algorithms; these are *collectively* referred to as "the" fast Fourier transform (FFT) algorithm
- The first (or at least most timely) FFT algorithm was published by Jim Cooley and John Tukey in 1965 and is now referred to as the radix-2 Cooley-Tukey FFT algorithm
  - this is the main version we will consider …

Computation of the DFT

- In order for the DFT to be useful for linear filtering, nonlinear filtering, spectrum analysis, *etc*., we need efficient computation algorithms for

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad k = 0, 1, \cdots, N-1$$

- Using the above directly requires N complex multiplications and N-1 complex additions for each of the N DFT values $\Rightarrow$ $\beta(N) = N^2$ complex multiplications. For example,

$$N = 1024 \quad \Rightarrow \quad \beta(1024) \approx 10^6$$