ESE 531: Digital Signal Processing

Lec 22: April 18, 2017

Fast Fourier Transform (con't)

Previously

- Circular Convolution
  - Linear convolution with circular convolution
- Discrete Fourier Transform
  - Linear convolution through circular
  - Linear convolutions through DFT
- Fast Fourier Transform

- Today
  - Circular convolution as linear convolution with aliasing
  - DTFT, DFT, FFT practice

Circular Convolution

- Circular Convolution:

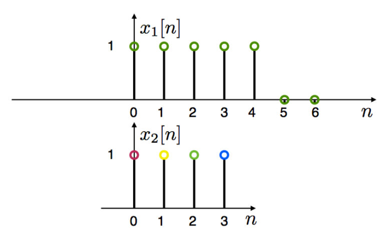
$$x_1[n] \circledN x_2[n] \triangleq \sum_{m=0}^{N-1} x_1[m] x_2[((n-m))_N]$$

For two signals of length N

Note: Circular convolution is commutative

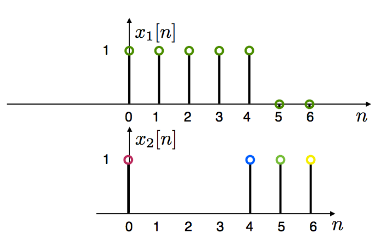
$$x_2[n] \circledN x_1[n] = x_1[n] \circledN x_2[n]$$

Compute Circular Convolution Sum



$$x_1[n] \circledN x_2[n] \triangleq \sum_{m=0}^{N-1} x_1[m] x_2[((n-m))_N]$$

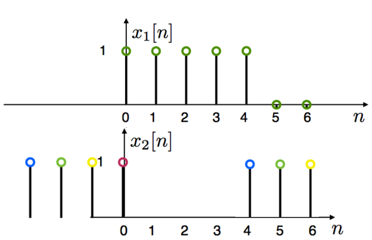
Compute Circular Convolution Sum



$$x_1[n] \circledN x_2[n] \triangleq \sum_{m=0}^{N-1} x_1[m] x_2[((n-m))_N]$$

Compute Circular Convolution Sum

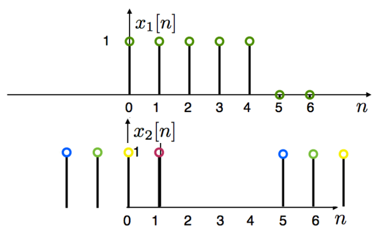
y[0]=2



$$x_1[n] \circledN x_2[n] \triangleq \sum_{m=0}^{N-1} x_1[m] x_2[((n-m))_N]$$

Compute Circular Convolution Sum

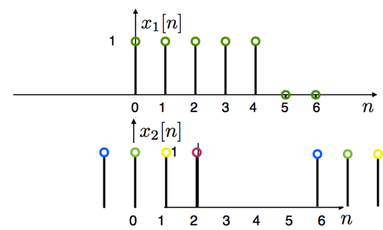
y[0]=2
y[1]=2



$$x_1[n] \circledN x_2[n] \triangleq \sum_{m=0}^{N-1} x_1[m]x_2[((n-m))_N]$$

Compute Circular Convolution Sum

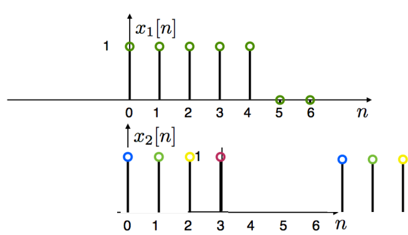
y[0]=2
y[1]=2
y[2]=3



$$x_1[n] \circledN x_2[n] \triangleq \sum_{m=0}^{N-1} x_1[m]x_2[((n-m))_N]$$

Compute Circular Convolution Sum

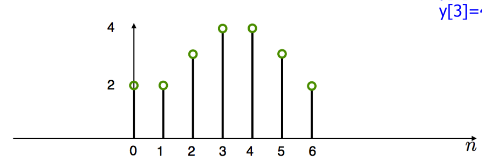
y[0]=2
y[1]=2
y[2]=3
y[3]=4



$$x_1[n] \circledN x_2[n] \triangleq \sum_{m=0}^{N-1} x_1[m]x_2[((n-m))_N]$$

Result

y[0]=2
y[1]=2
y[2]=3
y[3]=4



$$x_1[n] \circledN x_2[n] \triangleq \sum_{m=0}^{N-1} x_1[m]x_2[((n-m))_N]$$

Linear Convolution

- We start with two non-periodic sequences:

$$x[n] \quad 0 \le n \le L-1$$
$$h[n] \quad 0 \le n \le P-1$$

  - E.g. x[n] is a signal and h[n] a filter's impulse response
- We want to compute the linear convolution:

$$y[n] = x[n] * h[n] = \sum_{m=0}^{L-1} x[m]h[n-m]$$

  - y[n] is nonzero for $0 \le n \le L+P-2$ with length M=L+P-1

Requires LP multiplications

Linear Convolution via Circular Convolution

- Zero-pad x[n] by P-1 zeros

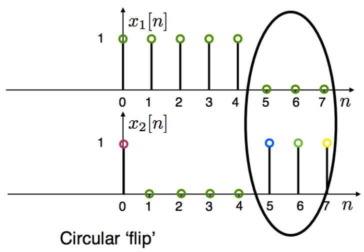
$$x_{zp}[n] = \begin{cases} x[n] & 0 \le n \le L-1 \\ 0 & L \le n \le L+P-2 \end{cases}$$

- Zero-pad h[n] by L-1 zeros

$$h_{zp}[n] = \begin{cases} h[n] & 0 \le n \le P-1 \\ 0 & P \le n \le L+P-2 \end{cases}$$

- Now, both sequences are length M=L+P-1

Example



M = L + P - 1 = 8

$$y[n] = x_1[n] \circledast_8 x_2[n] = x_1[n] * x_2[n]$$

Circular Conv. as Linear Conv. w/ Aliasing

- If the DTFT $X(e^{j\omega})$ of a sequence x[n] is sampled at N frequencies $\omega_k=2\pi k/N$, then the resulting sequence X[k] corresponds to the periodic sequence

$$\tilde{x}[n] = \sum_{r=-\infty}^{\infty} x[n - rN].$$

- And $X[k] = \begin{cases} X(e^{j(2\pi k/N)}), & 0 \le k \le N-1, \\ 0, & \text{otherwise}, \end{cases}$ is the DFT of one period given as

$$x_p[n] = \begin{cases} \tilde{x}[n], & 0 \le n \le N-1, \\ 0, & \text{otherwise}. \end{cases}$$

Circular Conv. as Linear Conv. w/ Aliasing

$$x_p[n] = \begin{cases} \tilde{x}[n], & 0 \le n \le N-1, \\ 0, & \text{otherwise}. \end{cases}$$

- If x[n] has length less than or equal to N, then $x_p[n]=x[n]$
- However if the length of x[n] is greater than N, this might not be true and we get aliasing in time
  - N-point convolution results in N-point sequence

Circular Conv. as Linear Conv. w/ Aliasing

- Given two N-point sequences ($x_1[n]$ and $x_2[n]$) and their N-point DFTs ($X_1[k]$ and $X_2[k]$)
- The N-point DFT of $x_3[n]=x_1[n]*x_2[n]$ is defined as

$$X_3[k] = X_3(e^{j(2\pi k/N)})$$

- And therefore $X_3[k]=X_1[k]X_2[k]$, where the inverse DFT of $X_3[k]$ is

$$x_{3p}[n] = \begin{cases} \sum_{r=-\infty}^{\infty} x_3[n - rN], & 0 \le n \le N-1, \\ 0, & \text{otherwise}, \end{cases}$$

Circular Conv. as Linear Conv. w/ Aliasing

$$x_{3p}[n] = \begin{cases} \sum_{r=-\infty}^{\infty} x_3[n - rN], & 0 \le n \le N-1, \\ 0, & \text{otherwise}, \end{cases}$$

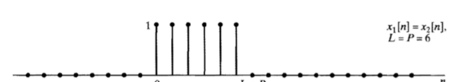
- Therefore

$$x_{3p}[n] = x_1[n] \circledast x_2[n]$$

- The N-point circular convolution is the sum of linear convolutions shifted in time by N

Example 1:

- Let

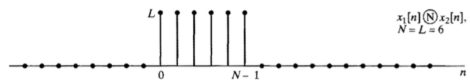


- The N=L=6-point circular convolution results in

Example 1:

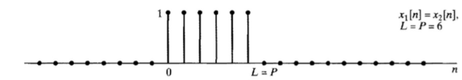
- Let

- The N=L=6-point circular convolution results in



Example 1:

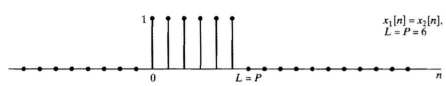
- Let



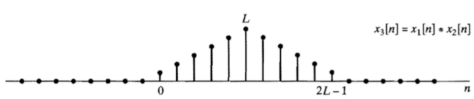
- The linear convolution results in

Example 1:

- Let

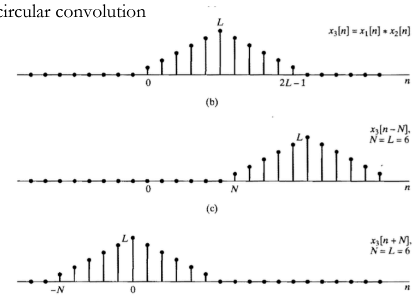


- The linear convolution results in



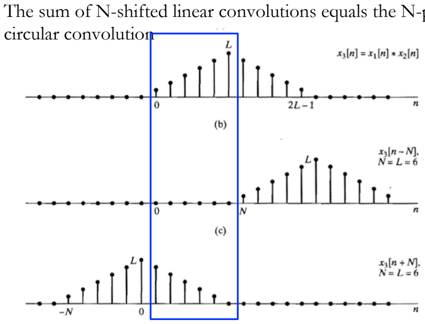
Example 1:

- The sum of N-shifted linear convolutions equals the N-point circular convolution



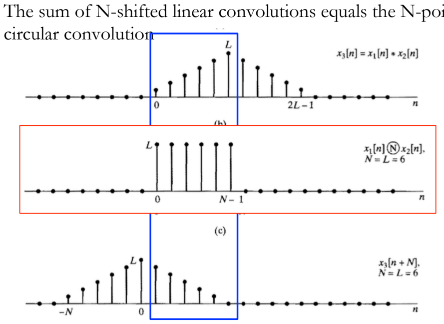
Example 1:

- The sum of N-shifted linear convolutions equals the N-point circular convolution



Example 1:

- The sum of N-shifted linear convolutions equals the N-point circular convolution

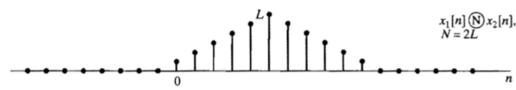


Example 1:

- If I want the circular convolution and linear convolution to be the same, what do I have to do?

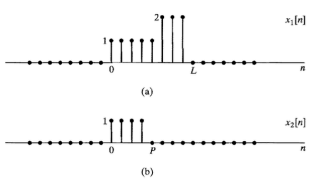
Example 1:

- If I want the circular convolution and linear convolution to be the same, what do I have to do?
  - Take the N=2L-point circular convolution



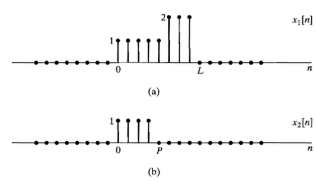
Example 2:

- Let

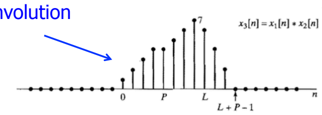


Example 2:

- Let



Linear convolution

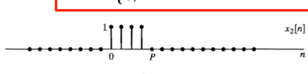


- What does the L-point circular convolution look like?

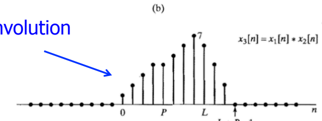
Example 2:

- Let

$$x_{3p}[n] = \begin{cases} x_1[n] \text{ Ⓛ } x_2[n] = \sum_{r=-\infty}^{\infty} x_3[n-rL], & 0 \le n \le L-1, \\ 0, & \text{otherwise.} \end{cases}$$



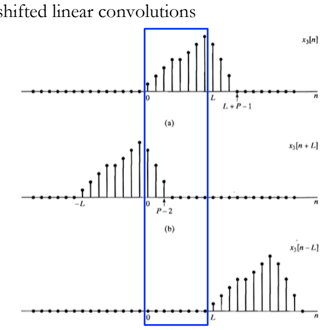
Linear convolution



- What does the L-point circular convolution look like?

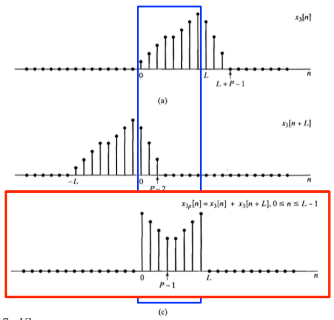
Example 2:

- The L-shifted linear convolutions



Example 2:

- The L-shifted linear convolutions



Discrete Fourier Transform

- The DFT

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn} \quad \text{Inverse DFT, synthesis}$$

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

- It is understood that,

$$x[n] = 0 \quad \text{outside } 0 \le n \le N-1$$
$$X[k] = 0 \quad \text{outside } 0 \le k \le N-1$$

DFT vs. DTFT

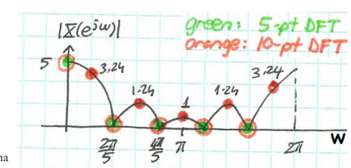
- The DFT are samples of the DTFT at N equally spaced frequencies

$$X[k] = X(e^{j\omega})|_{\omega=k\frac{2\pi}{N}} \qquad 0 \le k \le N-1$$

DFT vs DTFT

- Back to example

$$X[k] = \sum_{n=0}^{4} W_{10}^{nk}$$
$$= e^{-j\frac{4\pi}{10}k}\frac{\sin(\frac{\pi}{2}k)}{\sin(\frac{\pi}{10}k)}$$



Fast Fourier Transform Algorithms

- We are interested in efficient computing methods for the DFT and inverse DFT:

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

$$W_N = e^{-j(\frac{2\pi}{N})}.$$

Eigenfunction Properties

- Most FFT algorithms exploit the following properties of W_N^{kn}:
  - Conjugate Symmetry

$$W_N^{k(N-n)} = W_N^{-kn} = (W_N^{kn})^*$$

  - Periodicity in n and k

$$W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n}$$

  - Power $\quad W_N^2 = W_{N/2}$

FFT Algorithms via Decimation

- Most FFT algorithms decompose the computation of a DFT into successively smaller DFT computations.
  - Decimation-in-time algorithms decompose x[n] into successively smaller subsequences.
  - Decimation-in-frequency algorithms decompose X[k] into successively smaller subsequences.
- We mostly discuss decimation-in-time algorithms today.
- Note: Assume length of x[n] is power of 2 ($N = 2^\nu$). If not, zero-pad to closest power.

Decimation-in-Time FFT

- We start with the DFT

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

- Separate the sum into even and odd terms:

$$X[k] = \sum_{n \text{ even}} x[n] W_N^{kn} + \sum_{n \text{ odd}} x[n] W_N^{kn}$$

  - These are two DFTs, each with half the number of samples (N/2)

Decimation-in-Time FFT

Let $n = 2r$ (n even) and $n = 2r + 1$ (n odd):

$$\begin{aligned} X[k] &= \sum_{r=0}^{(N/2)-1} x[2r] W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x[2r+1] W_N^{(2r+1)k} \\ &= \sum_{r=0}^{(N/2)-1} x[2r] W_N^{2rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1] W_N^{2rk} \end{aligned}$$

$$X[k] = \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1] W_{N/2}^{rk}$$

Decimation-in-Time FFT

Hence:

$$\begin{aligned} X[k] &= \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1] W_{N/2}^{rk} \\ &\triangleq G[k] + W_N^k H[k], \quad k = 0, \ldots, N-1 \end{aligned}$$

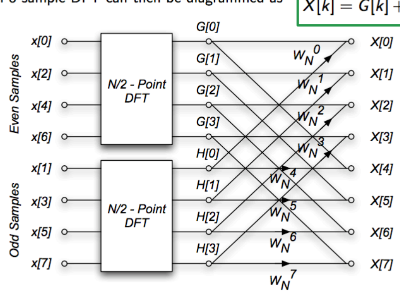
where we have defined:

$$G[k] \triangleq \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} \quad \Rightarrow \text{DFT of even samples}$$

$$H[k] \triangleq \sum_{r=0}^{(N/2)-1} x[2r+1] W_{N/2}^{rk} \quad \Rightarrow \text{DFT of odd samples}$$

Decimation-in-Time FFT

An 8 sample DFT can then be diagrammed as $X[k] = G[k] + W_N^k H[k]$



Decimation-in-Time FFT

- So,

$$\begin{aligned} G[k + (N/2)] &= G[k] \\ H[k + (N/2)] &= H[k] \end{aligned}$$

- The periodicity of G[k] and H[k] allows us to further simplify. For the first N/2 points we calculate G[k] and $W_N^k H[k]$, and then compute the sum

$$X[k] = G[k] + W_N^k H[k] \qquad \forall \{k : 0 \le k < \frac{N}{2}\}.$$

How does periodicity help for $\frac{N}{2} \le k < N$?

Decimation-in-Time FFT

$$X[k] = G[k] + W_N^k H[k] \qquad \forall\{k : 0 \le k < \frac{N}{2}\}.$$

for $\frac{N}{2} \le k < N$:

$$W_N^{k+(N/2)} = ?$$

$$X[k + (N/2)] = ?$$

Decimation-in-Time FFT

$$X[k] = G[k] + W_N^k H[k] \qquad \forall\{k : 0 \le k < \frac{N}{2}\}.$$

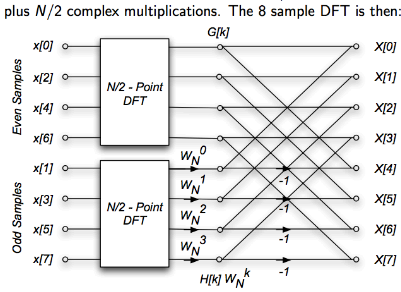
for $\frac{N}{2} \le k < N$:

$$W_N^{k+(N/2)} = \text{-}1$$

$$X[k + (N/2)] = G[k] - W_N^k H[k]$$

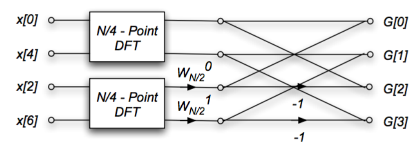
Decimation-in-Time FFT

The N-point DFT has been reduced two $N/2$-point DFTs, plus $N/2$ complex multiplications. The 8 sample DFT is then:



Decimation-in-Time FFT

- We can use the same approach for each of the N/2 point DFT's. For the N = 8 case, the N/2 DFTs look like



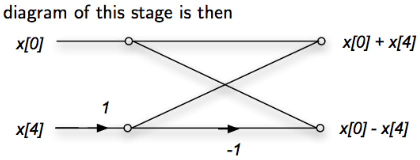
*Note that the inputs have been reordered again.

Decimation-in-Time FFT

- At this point for the 8 sample DFT, we can replace the N/4 = 2 sample DFT's with a single butterfly. The coefficient is

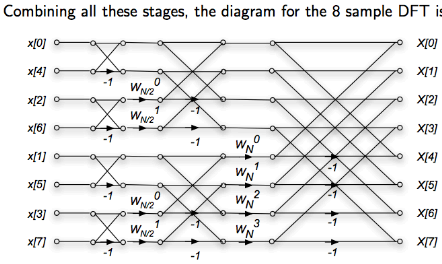
$$W_{N/4} = W_{8/4} = W_2 = e^{-j\pi} = -1$$

The diagram of this stage is then



Decimation-in-Time FFT

Combining all these stages, the diagram for the 8 sample DFT is:



- $3=\log_2(N)=\log_2(8)$ stages
- $4=N/2=8/2$ multiplications in each stage
  - 1st stage has trivial multiplication

Decimation-in-Time FFT

- In general, there are $\log_2 N$ stages of decimation-in-time.
- Each stage requires N/2 complex multiplications, some of which are trivial.
- The total number of complex multiplications is (N/2) $\log_2 N$, or O(N $\log_2 N$)

Decimation-in-Time FFT

- In general, there are $\log_2 N$ stages of decimation-in-time.
- Each stage requires N/2 complex multiplications, some of which are trivial.
- The total number of complex multiplications is (N/2) $\log_2 N$, or O(N $\log_2 N$)
- The order of the input to the decimation-in-time FFT algorithm must be permuted.
  - First stage: split into odd and even. Zero low-order bit (LSB) first
  - Next stage repeats with next zero-lower bit first.
  - Net effect is reversing the bit order of indexes

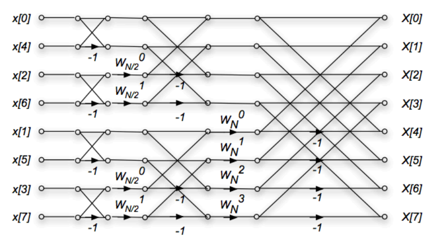
Decimation-in-Time FFT

This is illustrated in the following table for $N = 8$.

Decimal	Binary	Bit-Reversed Binary	Bit-Reversed Decimal
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Decimation-in-Time FFT

Combining all these stages, the diagram for the 8 sample DFT is:



- $3=\log_2(N)=\log_2(8)$ stages
- $4=N/2=8/2$ multiplications in each stage
  - 1st stage has trivial multiplication

Decimation-in-Frequency FFT

The DFT is

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}$$

If we only look at the even samples of $X[k]$, we can write $k = 2r$,

$$X[2r] = \sum_{n=0}^{N-1} x[n] W_N^{n(2r)}$$

We split this into two sums, one over the first $N/2$ samples, and the second of the last $N/2$ samples.

$$X[2r] = \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n+N/2] W_N^{2r(n+N/2)}$$

Decimation-in-Frequency FFT

But $W_N^{2r(n+N/2)} = W_N^{2rn} W_N^{rN} = W_N^{2rn} = W_{N/2}^{rn}$.
We can then write

$$\begin{aligned} X[2r] &= \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n+N/2] W_N^{2r(n+N/2)} \\ &= \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n+N/2] W_N^{2rn} \\ &= \sum_{n=0}^{(N/2)-1} (x[n] + x[n+N/2])\, W_{N/2}^{rn} \end{aligned}$$

This is the $N/2$-length DFT of first and second half of $x[n]$ summed.

Decimation-in-Frequency FFT

$$X[2r] = \text{DFT}_{\frac{N}{2}}\{(x[n] + x[n+N/2])\}$$
$$X[2r+1] = \text{DFT}_{\frac{N}{2}}\{(x[n] - x[n+N/2])\,W_N^n\}$$

(By a similar argument that gives the odd samples)

Decimation-in-Frequency FFT

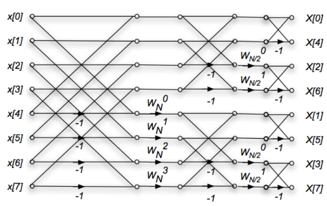
$$X[2r] = \text{DFT}_{\frac{N}{2}}\{(x[n] + x[n+N/2])\}$$
$$X[2r+1] = \text{DFT}_{\frac{N}{2}}\{(x[n] - x[n+N/2])\,W_N^n\}$$

(By a similar argument that gives the odd samples)

Continue the same approach is applied for the $N/2$ DFTs, and the $N/4$ DFT's until we reach simple butterflies.

Decimation-in-Frequency FFT

The diagram for and 8-point decimation-in-frequency DFT is as follows



This is just the decimation-in-time algorithm reversed!
The inputs are in normal order, and the outputs are bit reversed.

Example 1:

A long *periodic* sequence x of period $N = 2^r$ (r is an integer) is to be convolved with a finite-length sequence h of length K.

(a) *Show* that the output y of this convolution (filtering) is *periodic*; what is its period?

Example 1:

A long *periodic* sequence x of period $N = 2^r$ (r is an integer) is to be convolved with a finite-length sequence h of length K.

(a) *Show* that the output y of this convolution (filtering) is *periodic*; what is its period?

(b) Let $K = mN$ where m is an integer; N is large. How would you implement this convolution *efficiently*? Explain your analysis clearly.
Compare the *total number of multiplications* required in your scheme to that in the direct implementation of FIR filtering. (Consider the case $r = 10$, $m = 10$).

Example 2:

A sequence $x = \{x[n], n = 0,1,\ldots,N-1\}$ is given; let $X(e^{j\omega})$ be its DTFT.

(a) Suppose $N = 10$. You want to evaluate both $X(e^{j2\pi 7/12})$ and $X(e^{j2\pi 3/8})$. The only computation you can perform is one DFT, on any one input sequence of your choice. Can you find the desired DTFT values? *(Show your analysis and explain clearly.)*

Example 2:

A sequence $x = \{x[n], n = 0,1,\ldots,N-1\}$ is given; let $X(e^{j\omega})$ be its DTFT.

(a) Suppose $N = 10$. You want to evaluate both $X(e^{j2\pi 7/12})$ and $X(e^{j2\pi 3/8})$. The only computation you can perform is one DFT, on any one input sequence of your choice. Can you find the desired DTFT values? *(Show your analysis and explain clearly.)*

(b) Suppose N is large. You want to obtain $X(e^{j\omega})$ at the following $2M$ frequencies:

$$\omega = \frac{2\pi}{M}m, \quad m = 0,1,\ldots,M-1 \quad \text{and} \quad \omega = \frac{2\pi}{M}m + \frac{2\pi}{N}, \quad m = 0,1,\ldots,M-1.$$

Here $M = 2^{\mu} \ll N = 2^{\nu}$

A standard radix-2 FFT algorithm is available. You may execute the FFT algorithm *once* or *more than once*, and *multiplications* and *additions* outside of the FFT are *allowed*, if necessary.

You want to get the $2M$ DTFT values with as few *total multiplications* as possible (*including those in the FFT*). Give explicitly the best method you can find for this, with an estimate of the *total number of multiplications* needed in terms of M and N.

Does your result change if extra multiplications outside of FFTs are *not* allowed?

Big Ideas

- Discrete Fourier Transform (DFT)
  - For finite signals assumed to be zero outside of defined length
  - N-point DFT is sampled DTFT at N points
  - Useful properties allow easier linear convolution
- Fast Convolution Methods
  - Use circular convolution (i.e DFT) to perform fast linear convolution
    - Overlap-Add, Overlap-Save
  - Circular convolution is linear convolution with aliasing
- Fast Fourier Transform
  - Enable computation of an N-point DFT (or DFT^{-1}) with the order of just $N \cdot \log_2 N$ complex multiplications.
- Design DSP systems to minimize computations!

Admin

- Project
  - Due 4/25