Chapter 7

The Discrete Fourier Transform

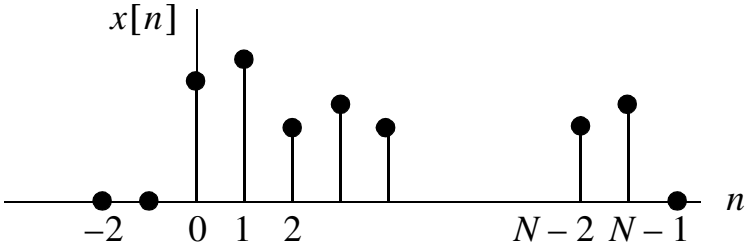
Contents

Up to this point we have analyzed LTI systems using the Fourier transform and the z-transform. A third, and computationally useful transform is the *discrete Fourier transform* (DFT). The DFT is a sequence which we will see corresponds to equally spaced samples of the Fourier transform of a finite duration signal. The most significant feature of the DFT is that it can be calculated via efficient algorithms.

The DFT may be introduced in terms of the discrete Fourier series (DFS), as the text chooses to, or in a more direct manner as chosen here.

7.1 Sampling the Fourier Transform

Consider a finite duration signal $x[n]$ assumed to be zero everywhere except possibly for $n = 0, 1, \ldots, N-1$.



- The Fourier transform of $x[n]$ is the periodic function

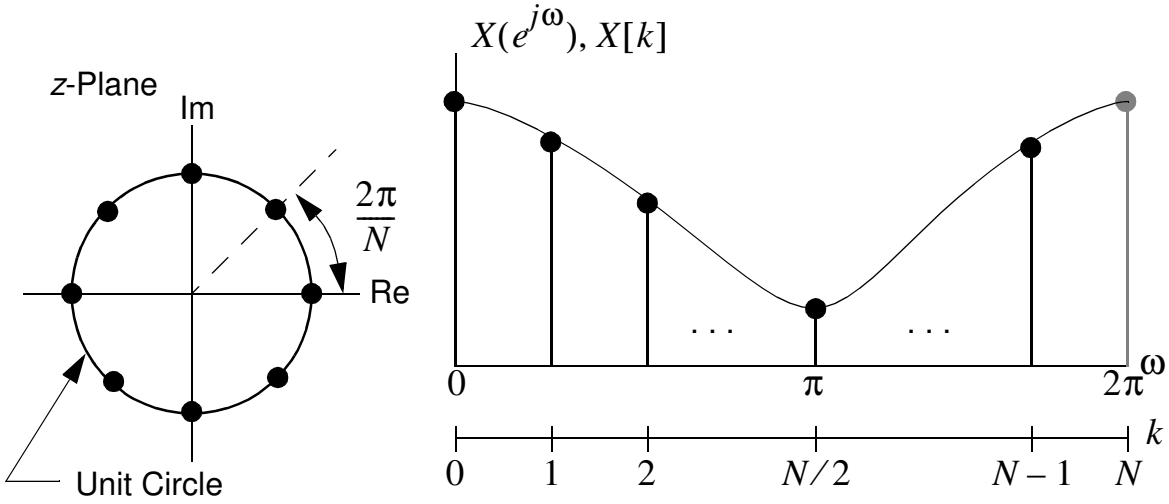
$$X(e^{j\omega}) = \sum_{n=0}^{N-1} x[n]e^{-j\omega n}$$

- Consider $X(e^{j\omega})$ at discrete frequencies $\omega_k = \frac{2\pi k}{N}, k = 0, 1, \ldots, N-1$

$$X(e^{j\omega_k}) = X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn}$$

where

$$W_N \equiv e^{-j2\pi/N}$$



(a) Sampling $X(z)$ (b) Sampling $X(e^{j\omega})$

The DFT as sampling the DTFT

7.2 The DFT - IDFT Pair

- **Discrete Fourier Transform (DFT)** (analysis)

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

- **Inverse Discrete Fourier Transform (IDFT)** (synthesis)

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

proof:

To show that the IDFT does indeed return $x[n]$ from $X[k]$ write

$$\begin{aligned}\frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn} &= \frac{1}{N}\sum_{k=0}^{N-1}\left[\sum_{m=0}^{N-1} x[m]W_N^{km}\right]W_N^{-kn} \\ &= \sum_{m=0}^{N-1} x[m]\underbrace{\left[\frac{1}{N}\sum_{k=0}^{N-1} W_N^{k(m-n)}\right]}_{\delta[m-n]} \\ &= x[n]\end{aligned}$$

The last two lines follow from

$$\begin{aligned}\sum_{k=0}^{N-1} W_N^{k(m-n)} &= \frac{1-e^{-j2\pi(m-n)}}{1-e^{-j2\pi(m-n)/N}} \\ &= \begin{cases} N, & m=n \\ 0, & m\neq n \end{cases} \\ &= N\delta[m-n]\end{aligned}$$

- A shorthand notation for indicating the relationship between $x[n]$ and $X[k]$ is

$$x[n] \overset{\mathcal{DFT}}{\longleftrightarrow} X[k]$$

- In the synthesis of $x[n]$ from the $X[k]$'s we are actually representing $x[n]$ in terms of a Fourier series, or more specifically a discrete Fourier series (DFS) with coefficients $X[k]$

- When a finite length signal is expanded in a Fourier series we know that periodic extensions on either side of the signal are

implied, thus the IDFT of $X[k]$ actually results in a periodic signal

$$\begin{aligned}\tilde{x}[n] &= \sum_{r=-\infty}^{\infty} x[n+rN] \\ &= x[(n \bmod N)] = x[((n))_N]\end{aligned}$$

with N the sequence period

- In this chapter $\tilde{x}[n]$ is used to represent the finite sequence $x[n]$ along with its periodic extensions

- Similarly $\tilde{X}[k]$ is used to represent $X[k]$ along with its periodic extensions

  - Note: Since $X[k]$ was initially defined in terms of $X(e^{j\omega})$, one can argue that for k outside the interval $[0, N-1]$ the periodic extensions of $X[k]$ automatically exist

- Formally, to maintain a duality type relationship

$$\begin{aligned}\tilde{X}[k] &= \sum_{r=-\infty}^{\infty} X[k+rN] \\ &= X[(k \bmod N)] = X[((k))_N]\end{aligned}$$

with N the sequence period

- We can also view the DFT as a linear transformation on the sequence $\{x[n]\}$

Let

$$\mathbf{x}_N = \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} \qquad \mathbf{X}_N = \begin{bmatrix} X[0] \\ X[1] \\ \vdots \\ X[N-1] \end{bmatrix}$$

and

$$\mathbf{W}_N = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix}$$

Now we can write an N-point DFT in matrix form as

$$\mathbf{X}_N = \mathbf{W}_N \mathbf{x}_N$$

The IDFT is just

$$\mathbf{x}_N = \mathbf{W}_N^{-1}\mathbf{X}_N \overset{\text{also}}{=} \underbrace{\frac{1}{N}\mathbf{W}_N^*\mathbf{X}_N}_{\text{directly from IDFT}}$$

- Comparing the two expressions for $\mathbf{x}_N$ we conclude that

$$\mathbf{W}_N^{-1} = \frac{1}{N}\mathbf{W}_N^* \text{ or } \mathbf{W}_N\mathbf{W}_N^* = N\mathbf{I}_N$$

where $\mathbf{I}_N$ is an $N \times N$ identity matrix, thus $\mathbf{W}_N$ is an orthogonal matrix

Example 7.1: A Pulse Sequence

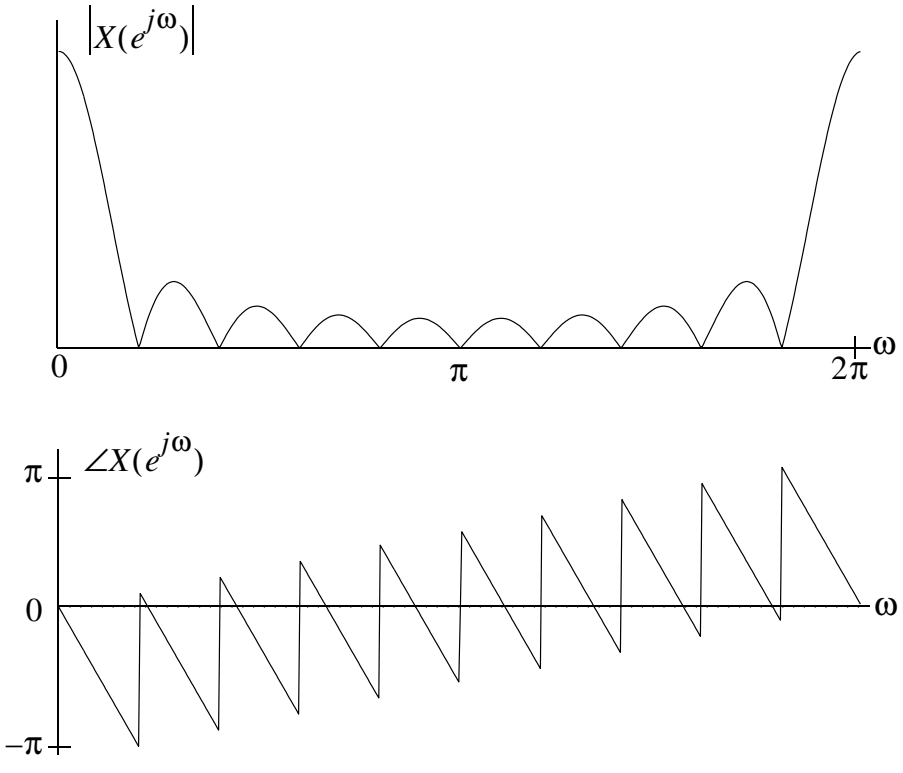
Consider the N-point DFT of the signal

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

for $N \ge L$.

- To begin with find the Fourier transform of $x[n]$

$$X(e^{j\omega}) = \sum_{n=0}^{L-1} x[n]e^{-j\omega n} = \sum_{n=0}^{L-1} e^{-j\omega n}$$
$$= \frac{1-e^{-j\omega L}}{1-e^{-j\omega}} = \frac{\sin(\omega L/2)}{\sin(\omega/2)} e^{-j\omega(L-1)/2}$$



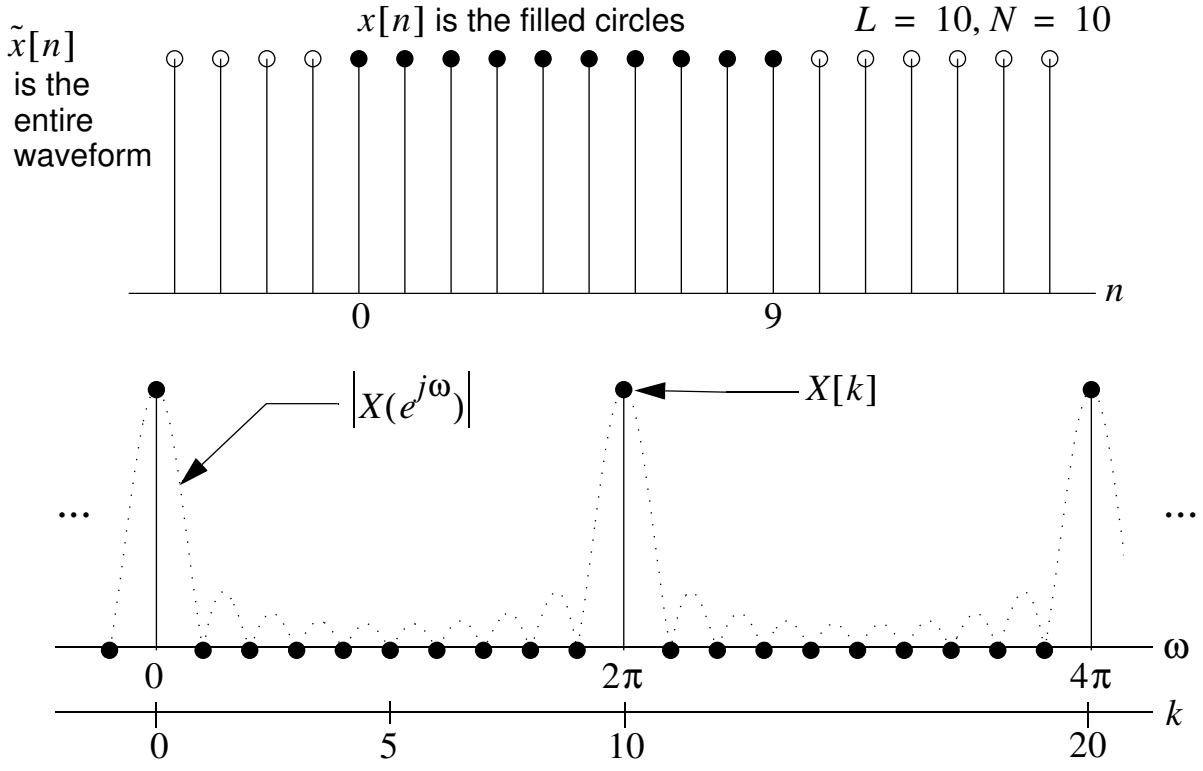
Magnitude and phase of $X(e^{j\omega})$ for $L = 10$

- The DFT is simply $X(e^{j\omega})$ evaluated at $\omega_k = 2k\pi/N$, $k = 0, 1, \ldots, N-1$, thus

$$X[k] = \frac{\sin(\pi k L/N)}{\sin(\pi k/N)} e^{-j\pi k(L-1)/N}, \quad k = 0, 1, \ldots, N-1$$

- If $N = L$, then

$$X[k] = \begin{cases} L, & k = 0 \\ 0, & k = 1, 2, \ldots, L-1 \end{cases}$$

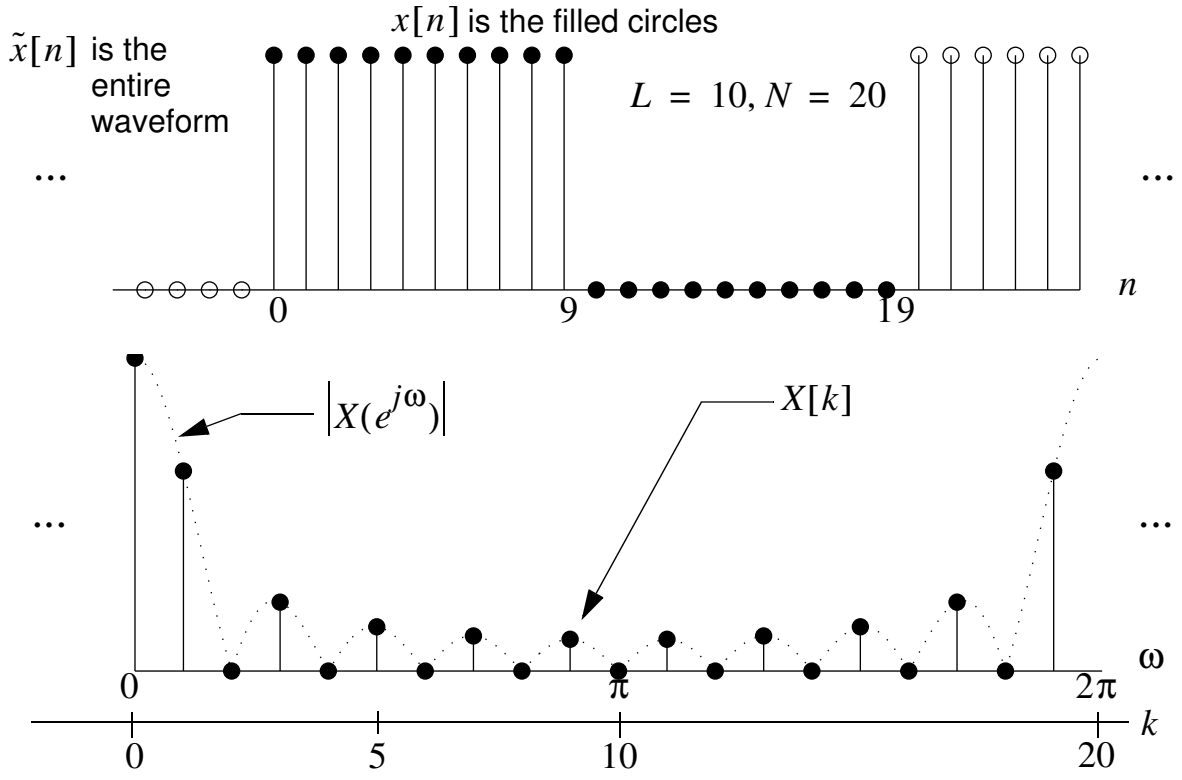


$|X(e^{j\omega})|$ and $\tilde{X}[k]$ for $N = L = 10$

- Clearly frequency sampling the Fourier transform with too few points results in loss of spectral detail

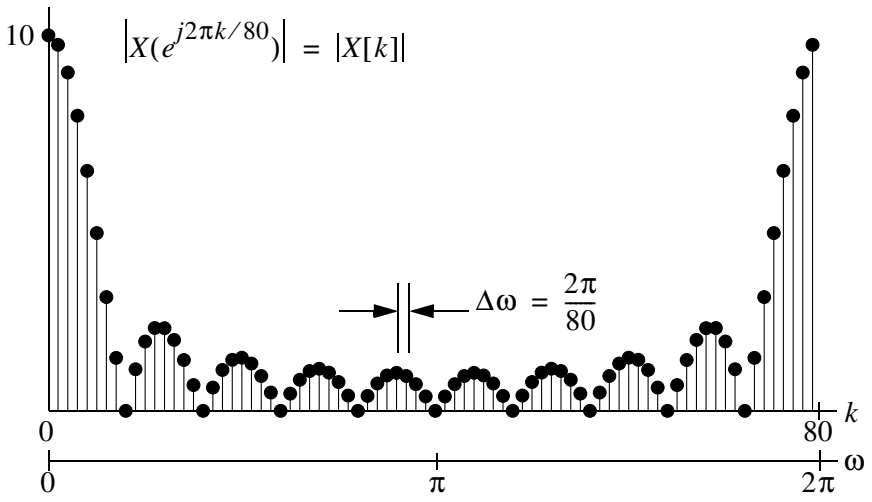
- In the case of $N = L$ the DFT sees $x[n]$ as a constant rather than a rectangular pulse, as depicted by $\tilde{x}[n]$ which is $x[n]$ including its periodic extensions

- By appending the sequence with $N - L$ zeros closer spaced frequency samples can be obtained
- Consider zero-padding with 10 samples so the total DFT length is $N = 20$, but the non-zero sequence length is only $L = 10$



$|X(e^{j\omega})|$ and $\tilde{X}[k]$ for $L = 10$, $N = 20$

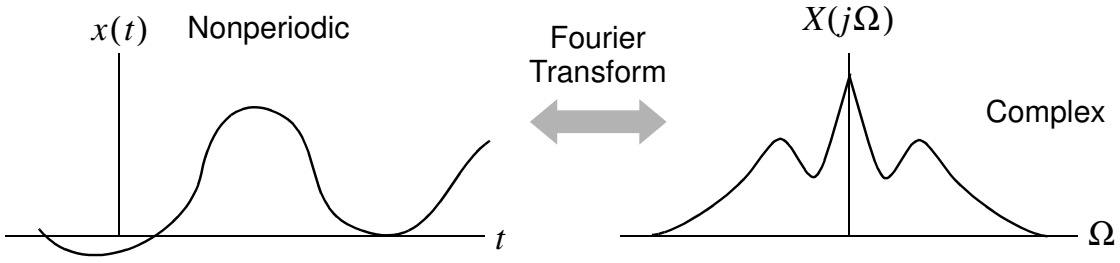
- The $\tilde{x}[n]$ waveform shows that we now have a periodic waveform of 10 ones followed by 10 zeros, etc., so we have twice as many samples on the $(0, 2\pi]$ interval of the same underlying Fourier transform
- Zero-padding further, say increase the DFT length to 80 but keep the number of ones in $x[n]$ at 10
- The zero padding has allowed the DFT to more closely approximate the true Fourier transform



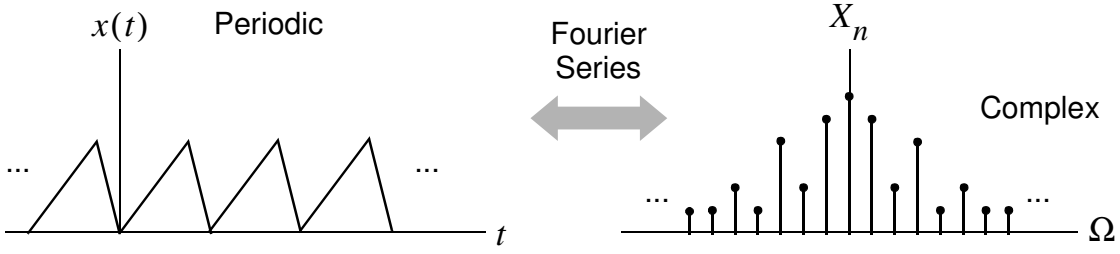
Magnitude and phase of $X[k]$ for $L = 10$ and $N = 80$

7.3 Summary of Fourier Representations

Continuous-Time Fourier Representations

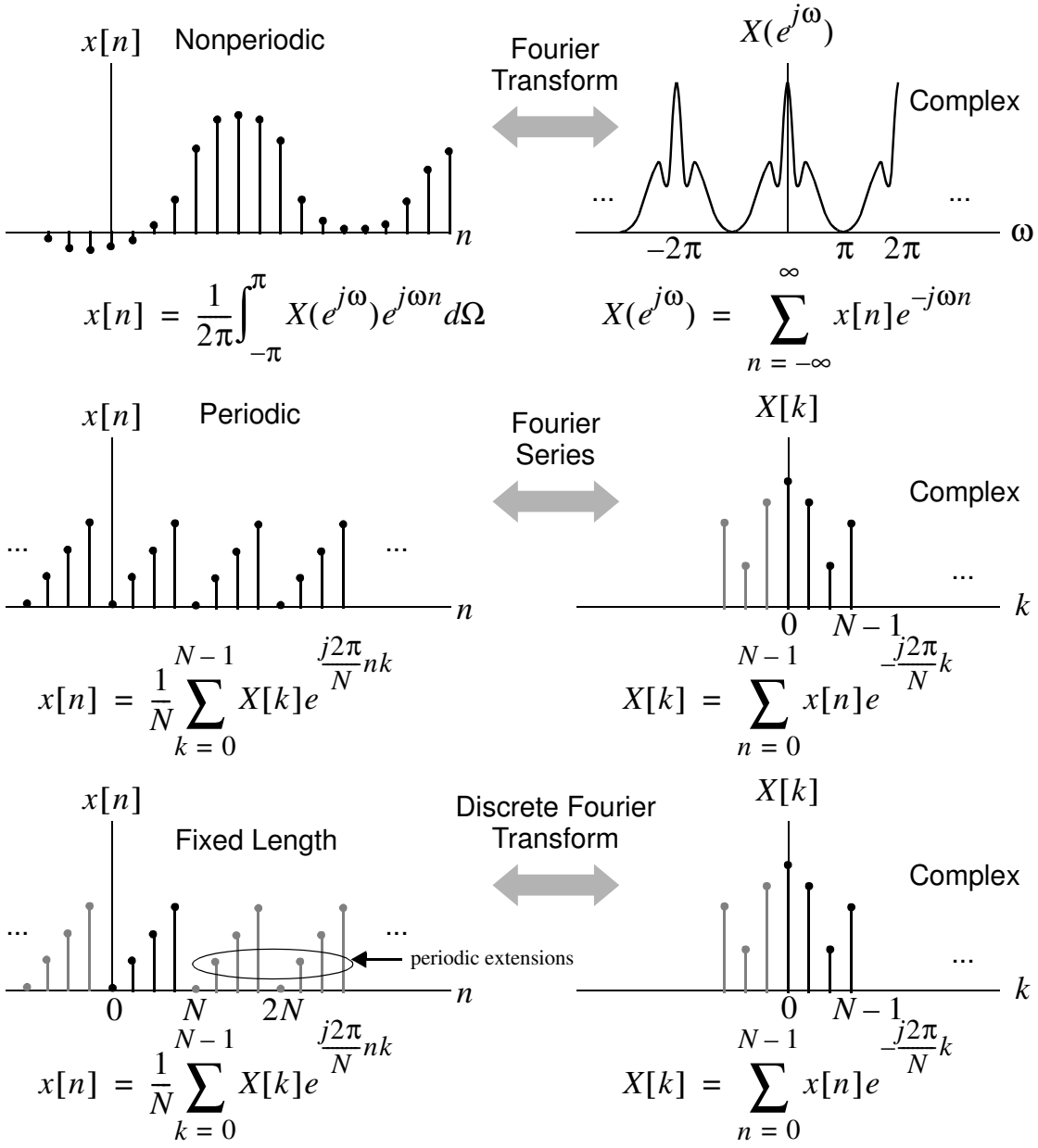


$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(j\Omega)e^{j\Omega t}d\Omega \qquad X(j\Omega) = \int_{-\infty}^{\infty} x(t)e^{-j\Omega t}dt$$



$$x(t) = \sum_{n=-\infty}^{\infty} X_n e^{jn\Omega_0 t},\ \Omega_0 = \frac{2\pi}{T} \qquad X_n = \frac{1}{T}\int_{-\infty}^{\infty} x(t)e^{jn\Omega_0 t}dt$$

Discrete-Time Fourier Representations



7.4 DFT Properties

The DFT has properties similar to those of the Fourier transform and z-transform. A major difference to consider is the implicit periodicity in both the time and frequency domains.

7.4.1 Linearity

Consider finite length sequences $x_1[n]$ and $x_2[n]$ of length N_1 and N_2 respectively. To compute the DFT of

$$x_3[n] = ax_1[n] + bx_2[n]$$

the length of $x_3[n]$ must be $N_3 \geq \max[N_1, N_2]$ which then requires that either/both $x_1[n]$ or $x_2[n]$ be augmented with an appropriate number of zeros. Following zero padding we can write

$$ax_1[n] + bx_2[n] \overset{\mathcal{DFT}}{\longleftrightarrow} aX_1[k] + bX_2[k]$$

with $N \geq \max[N_1, N_2]$ implied.

7.4.2 Circular Sequence Shift

From the study of Fourier transform properties we know that if

$$x[n] \overset{\mathcal{F}}{\longleftrightarrow} X(e^{j\omega})$$

then

$$x[n-m] \overset{\mathcal{F}}{\longleftrightarrow} e^{-j\omega m} X(e^{j\omega})$$

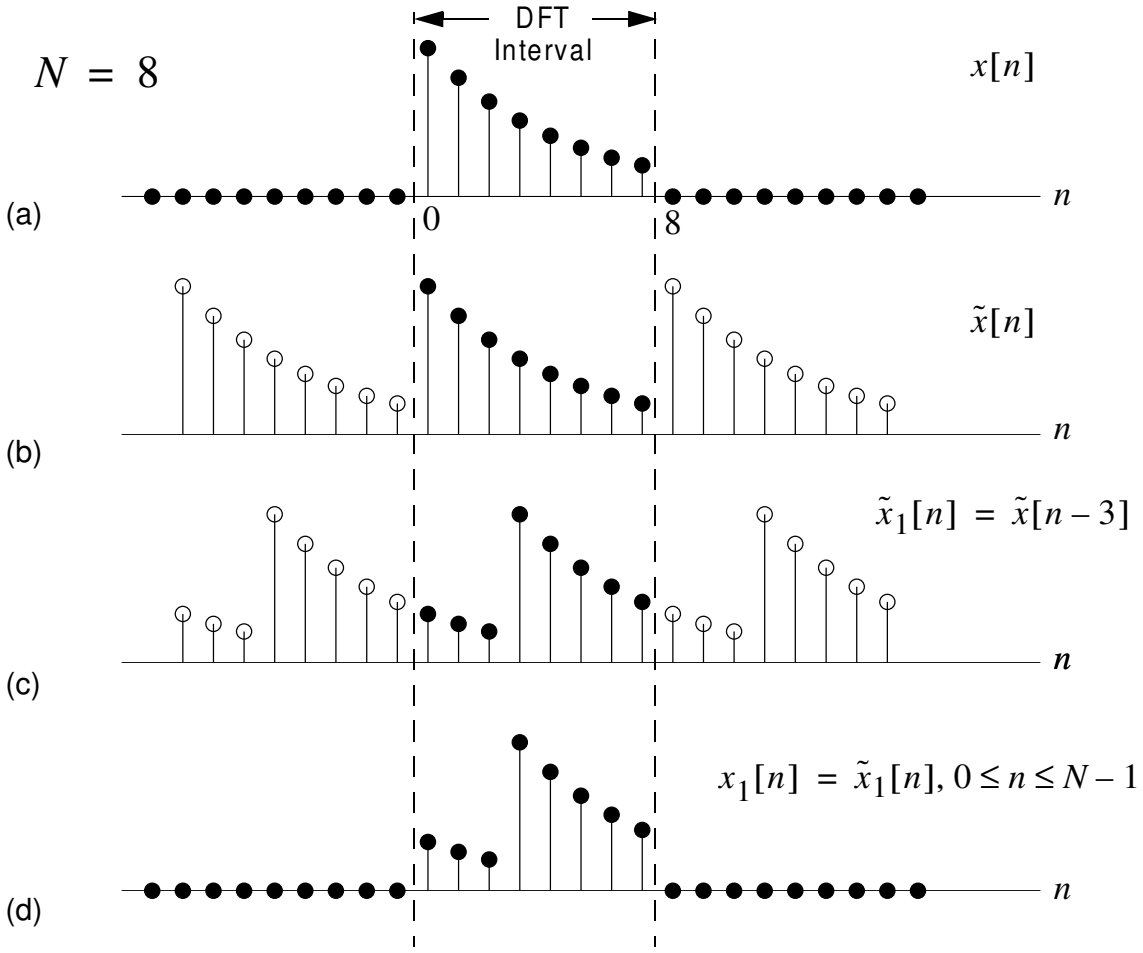
Consider now the relationship between $x[n]$ and $x_1[n]$ given the following DFT relationships:

$$x[n] \overset{\mathcal{DFT}}{\longleftrightarrow} X[k]$$
$$x_1[n] \overset{\mathcal{DFT}}{\longleftrightarrow} W_N^{mk} X[k]$$

- By definition

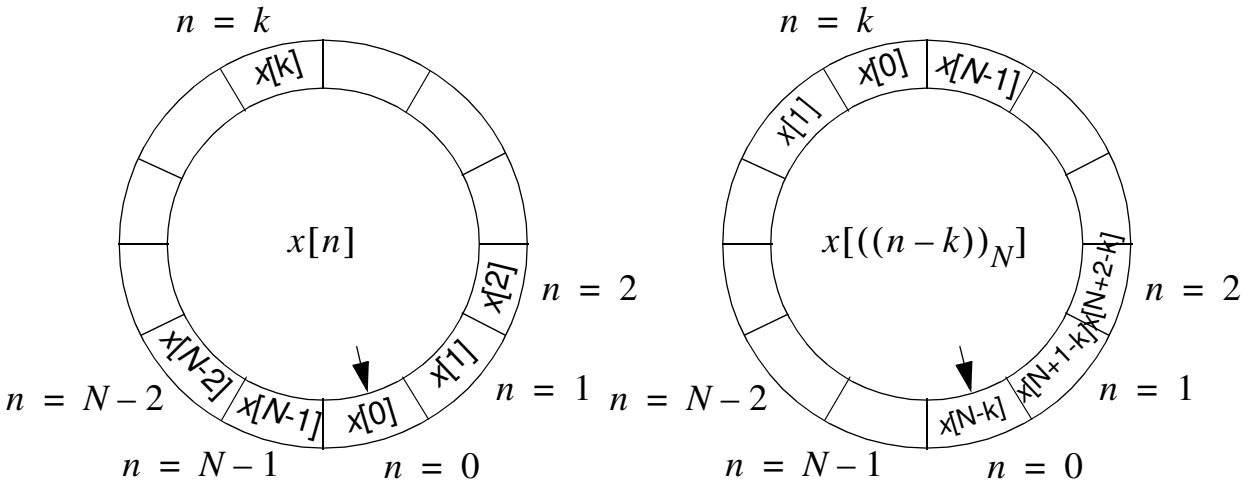
$$\begin{aligned} x_1[n] &= \frac{1}{N}\sum_{k=0}^{N-1} W_N^{mk} X[k] W_N^{-nk} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} X[k] W_N^{-(n-m)k} \\ &= x[((n-m))_N], \quad 0 \le n \le N-1 \end{aligned}$$

- This result is made clearer by considering the periodic sequences $\tilde{x}[n]$ and $\tilde{x}_1[n]$. In particular consider the following example with $m = 3$



Circular shift of a finite-length sequence by 3

- Another interpretation of the circular shift property is to view $x[n]$ as points on the circumference of a cylinder



Circular shift viewed on a cylinder

- In summary

$$x[((n-m))_N] \overset{\mathcal{DFT}}{\longleftrightarrow} W_N^{km} X[k]$$

- A quick Python example

```
1 n = arange(8)
2 x = [1,1,1,1,1,0,0,0]
3 # Shift to left by 1
4 y = fft.ifft(exp(1j*2*pi*1*n/8)*fft.fft(x)).real
5 y
6 # Line above returns:
7 array([1., 1., 1., 1., 0., 0., 0., 1.])
```

7.4.3 Duality

Consider the DFT of $X[n]$ assuming

$$x[n] \overset{\mathcal{DFT}}{\longleftrightarrow} X[k]$$

$$\begin{aligned}\text{DFT}\{X[n]\} &= \sum_{n=0}^{N-1} X[n] W_N^{kn} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} N X[n] W_N^{-(-k)n} \\ &= N x[((-k))_N],\ 0 \le k \le N-1\end{aligned}$$

where $x[((-k))_N]$ is $x[k]$ index reversed modulo N.

- In summary we can write

$$X[n] \overset{\mathcal{DFT}}{\longleftrightarrow} N x[((-k))_N] = N x[N-k],\ 0 \le k \le N-1$$

7.4.4 Symmetry Properties

Two fundamental relationships useful in developing various symmetry properties are:

1. $x^*[n] \overset{\mathcal{DFT}}{\longleftrightarrow} X^*[((-k))_N],\ 0 \le n \le N-1$
2. $x^*[((-n))_N] \overset{\mathcal{DFT}}{\longleftrightarrow} X^*[k],\ 0 \le n \le N-1$

proof of 1:

$$\begin{aligned}\text{DFT}\{x^*[n]\} &= \sum_{n=0}^{N-1} x^*[n] W_N^{kn} \\ &= \left[\underbrace{\sum_{n=0}^{N-1} x[n] W_N^{(-k)n}}_{X[((-k))_N]} \right]^* \\ &= X^*[N-k], \;\; 0 \le k \le N-1\end{aligned}$$

---

proof of 2:

$$\begin{aligned}\text{IDFT}\{X^*[k]\} &= \frac{1}{N} \sum_{k=0}^{N-1} X^*[k] W_N^{-kn} \\ &= \left[\underbrace{\frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-k(-n)}}_{x[((-n))_N]} \right]^* \\ &= x^*[N-n], \;\; 0 \le n \le N-1\end{aligned}$$

- The conjugate-symmetric portion of a finite-duration sequence $x[n]$ is

$$x_{ep}[n] = \frac{1}{2}\{x[n] + x^*[N-n]\}, \;\; 0 \le n \le N-1$$

with p used to denote the fact that the underlying sequences are actually periodic

  - Note: $x_{ep}[0] = \text{Re}\{x[0]\}$

- The conjugate-antisymmetric portion of $x[n]$ is

$$x_{op}[n] = \frac{1}{2}\{x[n] - x^*[N-n]\}, \;\; 0 \le n \le N-1$$

Note that $x_{op}[0] = \text{Im}\{x[0]\}$

- Using the two fundamental relationships established above, it follows that for $0 \le n, k \le N-1$

$$\text{Re}\{x[n]\} \overset{\mathcal{DFT}}{\longleftrightarrow} X_{ep}[k] = \frac{1}{2}\{X[k] + X^*[N-k]\}$$

$$j\,\text{Im}\{x[n]\} \overset{\mathcal{DFT}}{\longleftrightarrow} X_{op}[k] = \frac{1}{2}\{X[k] - X^*[N-k]\}$$

and

$$x_{ep}[n] = \frac{1}{2}\{x[n] + x^*[N-n]\} \overset{\mathcal{DFT}}{\longleftrightarrow} \text{Re}\{X[k]\}$$

$$x_{op}[n] = \frac{1}{2}\{x[n] - x^*[N-n]\} \overset{\mathcal{DFT}}{\longleftrightarrow} j\,\text{Im}\{X[k]\}$$

Special Case: Real $x[n]$

- Consider

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] W_N^{kn} \\ &= \left[\sum_{n=0}^{N-1} x[n] W_N^{(-k)n} \right]^* \\ &= X^*[((-k))_N] = X^*[N-k], \; 0 \le k \le N-1 \end{aligned}$$

thus it follows that

$$|X[k]| = |X[N-k]|,\ 0 \le k \le N-1 \ \text{(mag. even)}$$
$$\angle X[k] = -\angle X[N-k],\ 0 \le k \le N-1 \ \text{(phase odd)}$$

- Thus for real sequences we see that the DFT is unique over just the first $N/2$ points

7.4.5 Circular Convolution

Suppose that N-point sequences $x_1[n]$ and $x_2[n]$ have DFTs $X_1[k]$ and $X_2[k]$ respectively. Consider the product

$$X_3[k] = X_1[k]X_2[k]$$

We expect $x_3[n]$, the inverse DFT of $X_3[k]$, to be a "convolution" of $x_1[n]$ with $x_2[n]$. Since $x_1[n]$ and $x_2[n]$ actually have inherent periodic extensions (i.e. $\tilde{x}_1[n]$ and $\tilde{x}_2[n]$), the convolution will actually be a periodic convolution. That is,

$$x_3[n] = \sum_{m=0}^{N-1} \tilde{x}_1[m]\tilde{x}_2[n-m],\ 0 \le n \le N-1$$

- Investigate $x_3[n]$ in terms of the IDFT of $X_3[k]$

$$\begin{aligned} x_3[n] &= \frac{1}{N}\sum_{k=0}^{N-1} X_1[k]X_2[k]W_N^{-kn} \\ &= \frac{1}{N}\sum_{k=0}^{N-1}\left[\sum_{m=0}^{N-1} x_1[m]W_N^{km}\right] X_2[k]W_N^{-kn} \\ &= \sum_{m=0}^{N-1} x_1[m]\left[\frac{1}{N}\sum_{k=0}^{N-1} X_2[k]W_N^{-k(n-m)}\right] \\ &= \sum_{m=0}^{N-1} x_1[m]x_2[((n-m))_N],\ 0 \le n \le N-1 \end{aligned}$$

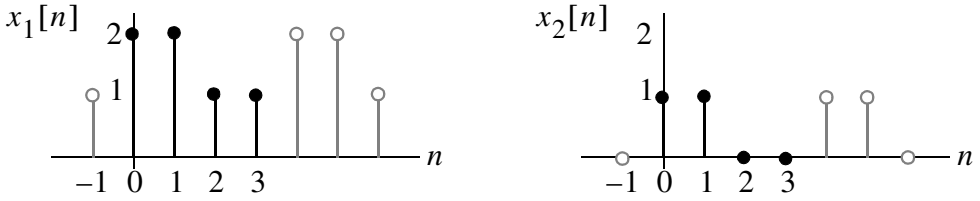
- The above expression for $x_3[n]$ is referred to as an *N-point circular convolution*

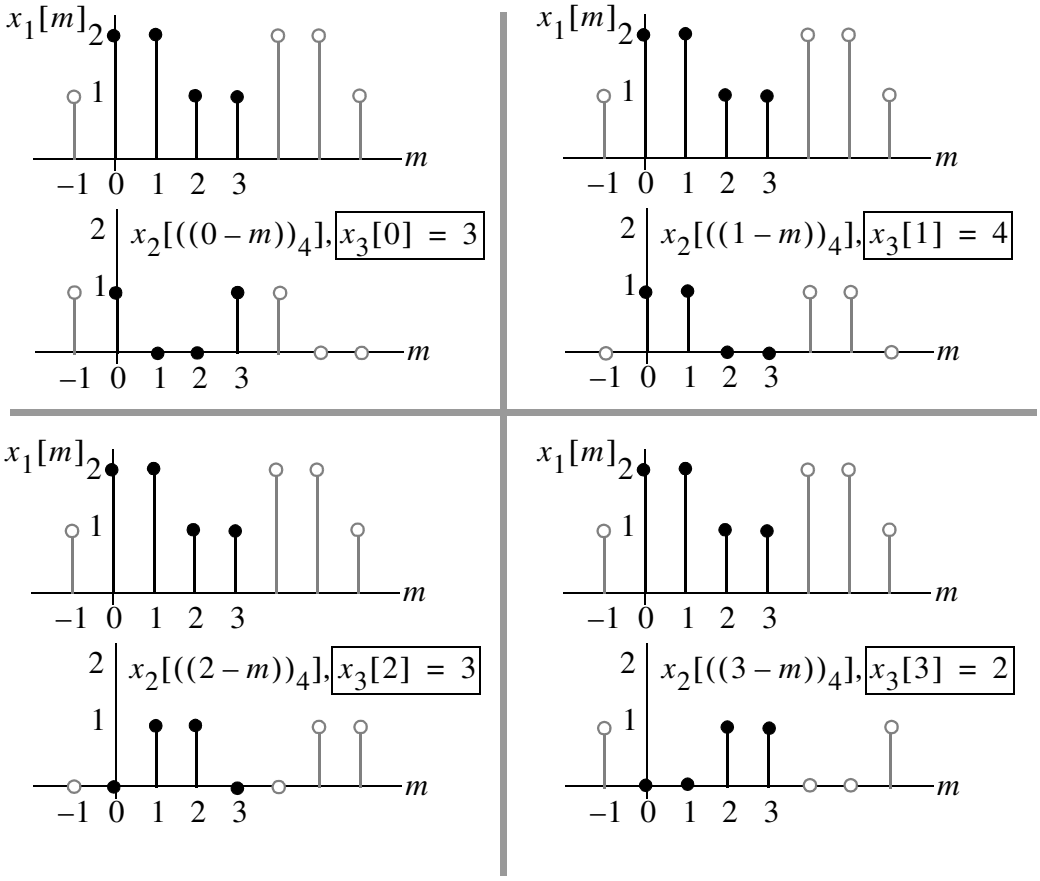
- In the notes an N-point circular convolution will be denoted with the symbol $\circledast_N$. Like linear convolution, circular convolution is commutative i.e.,

$$\begin{aligned} x_3[n] &= x_1[n] \circledast_N x_2[n] \\ &= x_2[n] \circledast_N x_1[n] \end{aligned}$$

Example 7.2: 4-Point Circular Convolution

Consider the 4-point circular convolution of the following two sequences.





- Find $x_3[k] = \text{IDFT}\{X_1[k]X_2[k]\}$
- First find $X_1[k]$ and $X_2[k]$

$$X_1[k] = 2 + 2e^{-j\pi k/2} + e^{-j\pi k} + e^{-j3\pi k/2}$$

$$\begin{aligned} X_1[0] &= 2 + 2 + 1 + 1 = 6 \\ X_1[1] &= 2 - 2j - 1 + j = 1 - j \\ X_1[2] &= 2 - 2 + 1 - 1 = 0 \\ X_1[3] &= 2 + 2j - 1 - j = 1 + j \end{aligned}$$

- Similarly

$$X_2[k] = 1 + e^{-j\pi k/2}$$

$$\begin{aligned} X_2[0] &= 1 + 1 = 2 \quad X_2[1] = 1 - j \\ X_2[2] &= 1 - 1 = 0 \quad X_2[3] = 1 + j \end{aligned}$$

- To obtain $X_3[k]$ form the product $X_1[k]X_2[k]$

$$\begin{aligned} X_3[0] &= 12 \quad X_3[1] = -2j \\ X_3[2] &= 0 \quad X_3[3] = 2j \end{aligned}$$

- Inverse transforming we obtain

$$x_3[n] = \frac{1}{4}\left[12 - 2je^{j\pi n/2} + 2je^{j3\pi n/2}\right]$$

thus

$$\begin{aligned} x_3[0] &= (12 - 2j + 2j)/4 = 3 \\ x_3[1] &= (12 + 2 + 2)/4 = 4 \\ x_3[2] &= (12 + 2j - 2j)/4 = 3 \\ x_3[3] &= (12 - 2 - 2)/4 = 2 \end{aligned}$$

which is the expected result

- A quick Python check yields:

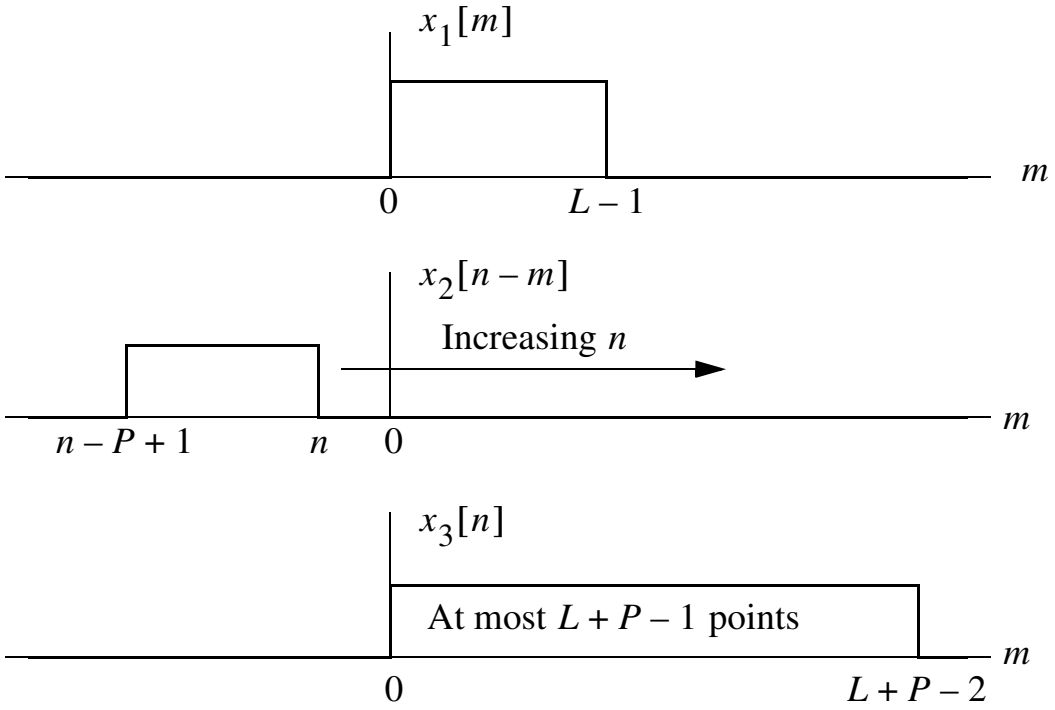
```
x1 = [2,2,1,1]
x2 = [1,1,0,0]
# For the product of the transforms
X3 = fft.fft(x1)*fft.fft(x2)
# Inverse transform
x3 = fft.ifft(X3)
# Numerical errors will result in small imaginary parts
x3.real
# Line above returns:
array([3., 4., 3., 2.])
```

7.4.6 Time Aliasing in Circular Convolution

Consider two finite length sequences $x_1[n],\ n = 0, 1, \ldots, L-1$ and $x_2[n],\ n = 0, 1, \ldots, P-1$. The linear convolution of $x_1[n]$ with $x_2[n]$ is

$$x_3[n] = \sum_{m=-\infty}^{\infty} x_1[m]x_2[n-m]$$

- Since the sequences are finite there are at most $L + P - 1$ nonzero points in $x_3[n]$



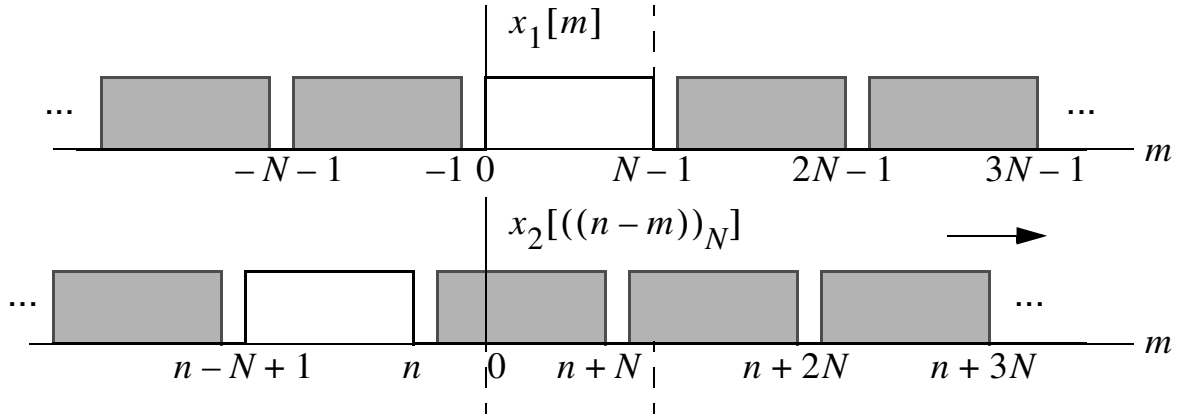
Linear convolution of $x_1[n]$ with $x_2[n]$

- The circular convolution of $x_1[n]$ and $x_2[n]$ is

$$x_{3p}[n] = \sum_{m=0}^{N-1} x_1[m]x_2[((n-m))_N]$$

where N is arbitrary, but one may initially consider $N = \max[L, P]$

- The subscript $3p$ is used to denote the fact that the circular convolution will result in a periodic sequence, while the linear convolution is aperiodic
- It is instructive to consider the periodic extensions of $x_1[m]$ and $x_2[((n-m))_N]$ when graphically viewing the convolution sum.

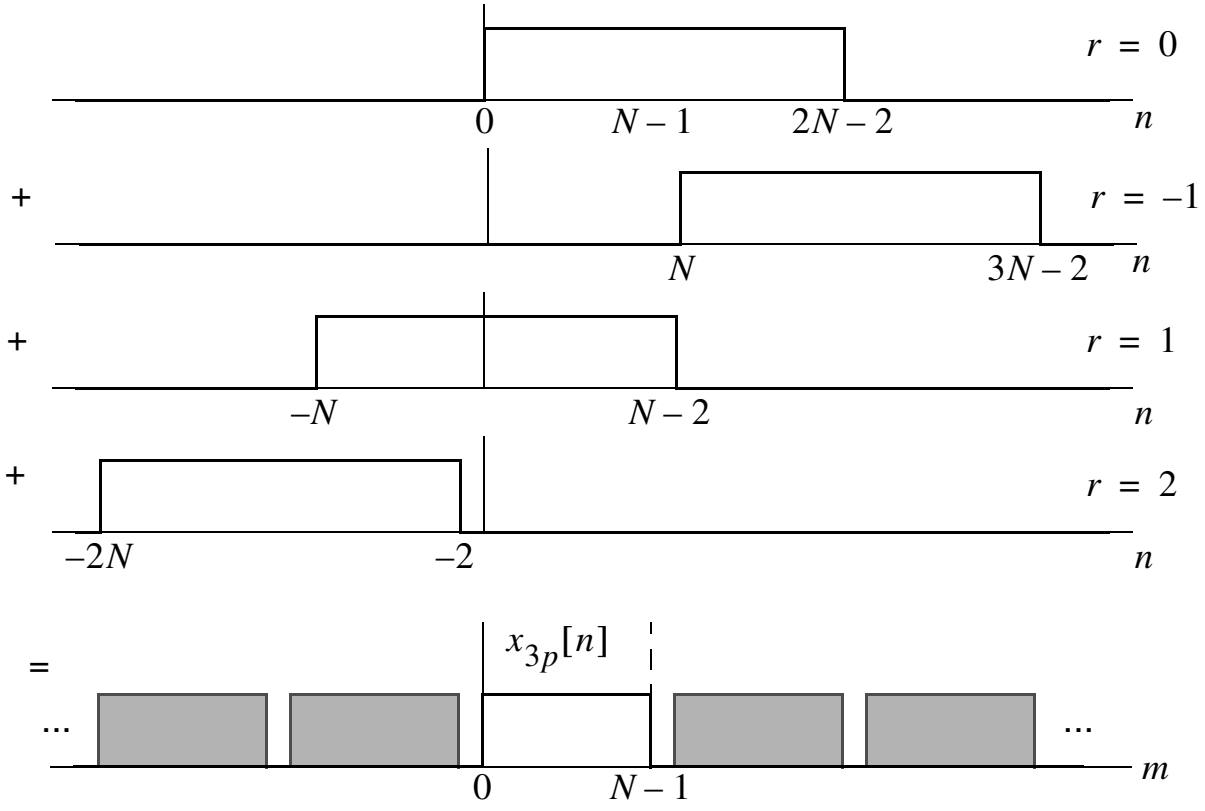


Sequences of the circular convolution sum resulting in $x_{3p}[n]$

- Clearly $x_{3p}[n]$ has *time aliasing* with respect to $x_3[n]$
- We can also view the circular convolution as a *periodic convolution*
- From the above figure showing $x_1[m]$ and $x_2[((n-m))_N]$ we see that $x_{3p}[n]$ can be written as an infinite sum of linear convolutions i.e.

$$x_{3p}[n] = \sum_{r=-\infty}^{\infty} \left[\sum_{m=-\infty}^{\infty} x_1[m] x_2[n-m+rN] \right]$$
$$= \sum_{r=-\infty}^{\infty} x_3[n+rN]$$

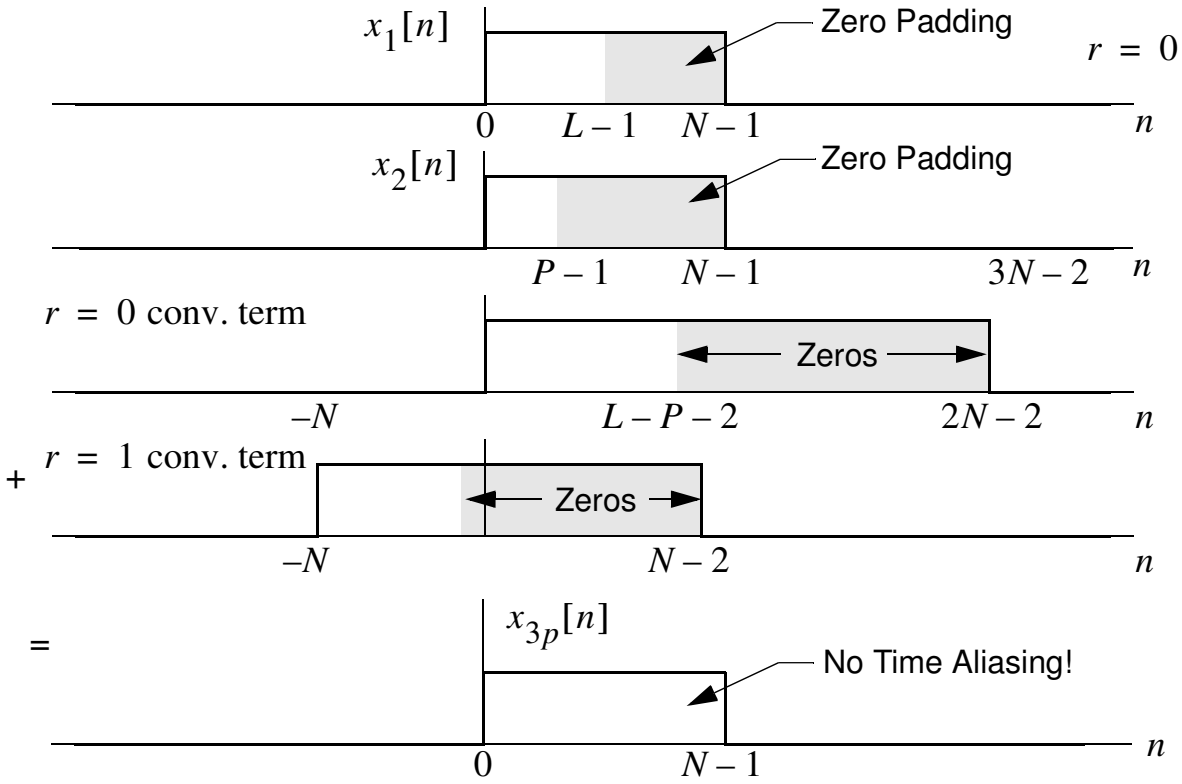
- The following figure graphically shows how $x_{3p}[n]$ is formed as a superposition of time shifted linear convolutions
- The interval of most interest is $0 \leq n \leq N-1$



The formation of $x_{3p}[n]$ as a sum of translates of $x_3[n]$

- In general we conclude that the linear and circular convolution are not equivalent
- Since $x_{3p}[k]$ can be efficiently computed using a *fast Fourier transform* (FFT), we would like to know under what conditions circular and linear convolution give equivalent results
- On the interval $0 \leq n \leq N-1$ the only translate of $x_3[n]$ causing time aliasing is $r = 1$ (i.e. $x_3[n+N]$)

- If we choose $N \geq L + P - 1$, then the contributions from $x_3[n + N]$ will be zero on $0 \leq n \leq N - 2$



Circular convolution performing linear convolution

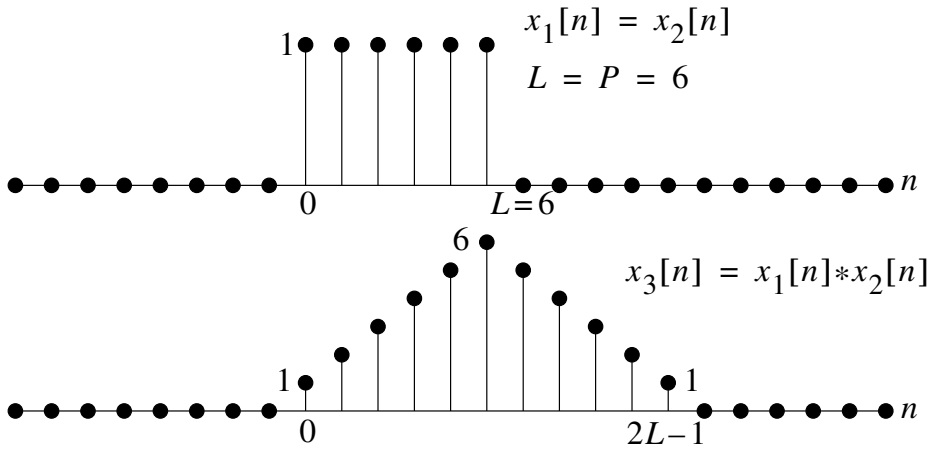
Example 7.3: Two Six-Point Sequences

Compare the linear and N-point circular convolution of

$$x_1[n] = x_2[n] = \begin{cases} 1, & 0 \leq n \leq 5 \\ 0, & \text{otherwise} \end{cases}$$

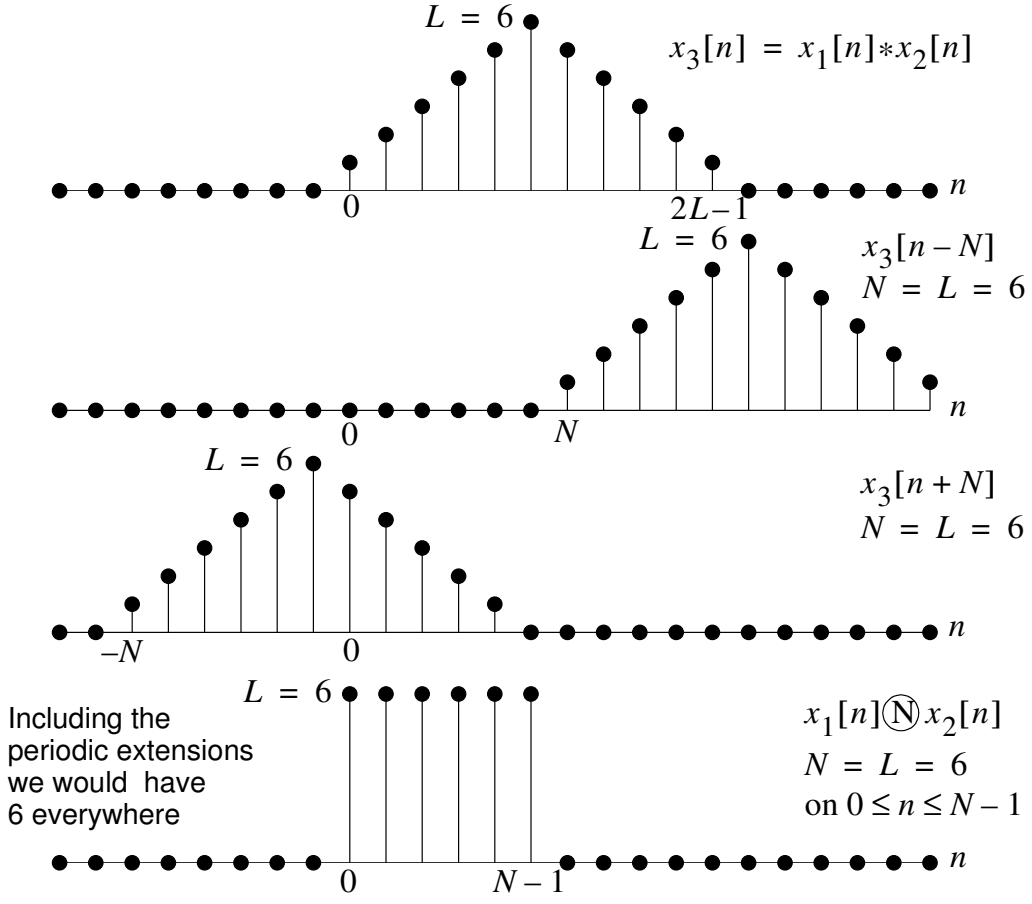
Note that $L = P = 6$.

- The linear convolution of two rectangles is the triangle sequence $x_3[n]$ shown below



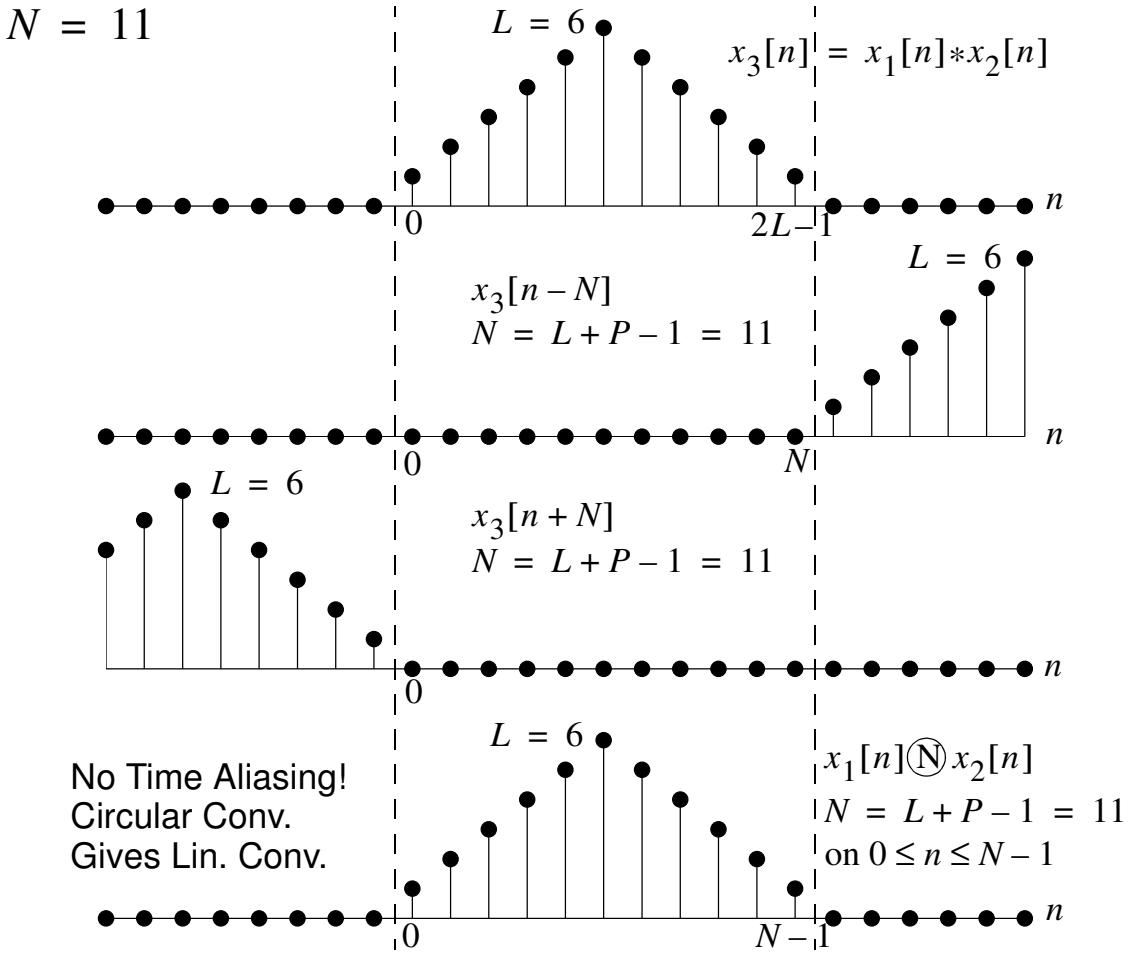
Plots of (a) $x_1[n] = x_2[n]$ and (b) $x_3[n] = x_1[n] * x_2[n]$

- The $N = 6$-point circular convolution $x_1[n] \circledast_N x_2[n]$ can be obtained on the interval $0 \leq n \leq N-1$ by considering $x_3[n] + x_3[n+N]$



- To avoid time aliasing we need to choose $N \geq L + P - 1$ or $N \geq 11$, then

$$x_1[n] \circledast_N x_2[n] = x_1[n] * x_2[n], \; 0 \leq n \leq N-1$$



12-point circular convolution

7.4.7 Multiplication Theorem

$$x_1[n]x_2[n] \overset{\mathcal{DFT}}{\longleftrightarrow} \frac{1}{N}\sum_{m=0}^{N-1} X_1[m]X_2[((k-m))_N]$$

- Note that this is just the dual of the convolution theorem

7.4.8 Parseval's Theorem

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2$$

7.4.9 Summary of DFT Properties

	Finite - Length (N) Sequence	N - point DFT
1.	$ax_1[n] + bx_2[n]$	$aX_1[k] + bX_2[k]$
2.	$X[n]$	$Nx[((-k))_N]$
3.	$x[((n-m))_N]$	$W_N^{km} X[k]$
4.	$W_N^{-ln} x[n]$	$X[((k-l))_N]$
5.	$\sum_{m=0}^{N-1} x_1[m]x_2[((n-m))_N]$	$X_1[k]X_2[k]$
6.	$x_1[n]x_2[n]$	$\frac{1}{N} \sum_{l=0}^{N-1} X_1[l]X_2[((k-l))_N]$
7.	$x^*[n]$	$X^*[((-k))_N]$
8.	$x^*[((-n))_N]$	$X^*[k]$
9.	For $x[n]$ real:	$\begin{cases} X[k] &= X^*[((-k))_N] \\ \lvert X[k]\rvert &= \lvert X[((-k))_N]\rvert \\ \angle X[k] &= -\angle X[((-k))_N] \end{cases}$

7.5 Implementation of LTI Systems using the DFT

Suppose we have an L-point input sequence $x[n]$ and a P-point impulse response $h[n]$. To perform linear convolution of these sequences using an N-point circular convolution requires that $N \geq$

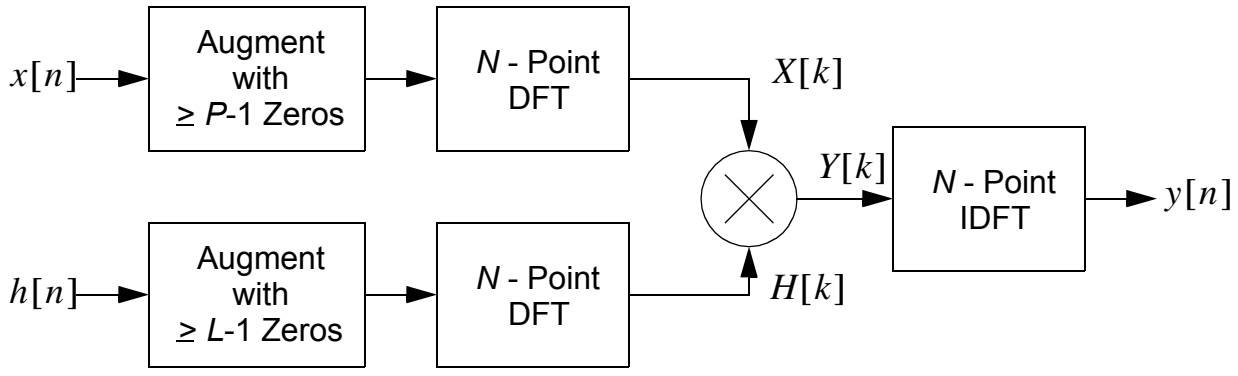
$L + P - 1$. The circular convolution can also be performed by multiplying the DFTs of $x[n]$ and $h[n]$ and then inverse transforming.

- The first step in using the DFT to perform linear convolution is to augment both $x[n]$ and $h[n]$ with zeros to make their lengths $N \geq L + P - 1$

$$\{x[n]\} \rightarrow \{x[0],\ x[1], \ldots,\ x[L-1],\ 0, \ldots,\ \underbrace{0}_{N-1}\}$$

$$\{h[n]\} \rightarrow \{h[0],\ h[1], \ldots,\ h[P-1],\ 0, \ldots,\ \underbrace{0}_{N-1}\}$$

- The second step is to calculate the N-point DFT of each augmented sequence
- The sequences $X[k]$ and $H[k]$ are then multiplied together to form $Y[k]$
- Finally calculating the IDFT of $Y[k]$ results in $y[n] = x[n] * h[n]$

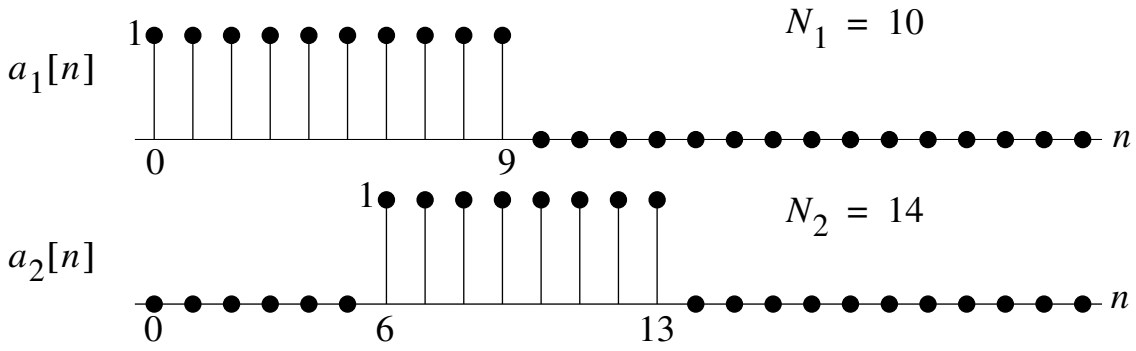


Using the DFT to convolve two finite-length sequences

Example 7.4: Transform Domain Convolution in Python

In this example Python is used to perform linear convolution of two rectangular pulse sequences via the DFT/IDFT. The results are compared with direct linear convolution. The amount of zero padding included in the transform domain (circular convolution) based method is varied so that the time domain aliasing can be studied.

- The input sequeneces $a_1[n]$ and $a_2[n]$ are shown below



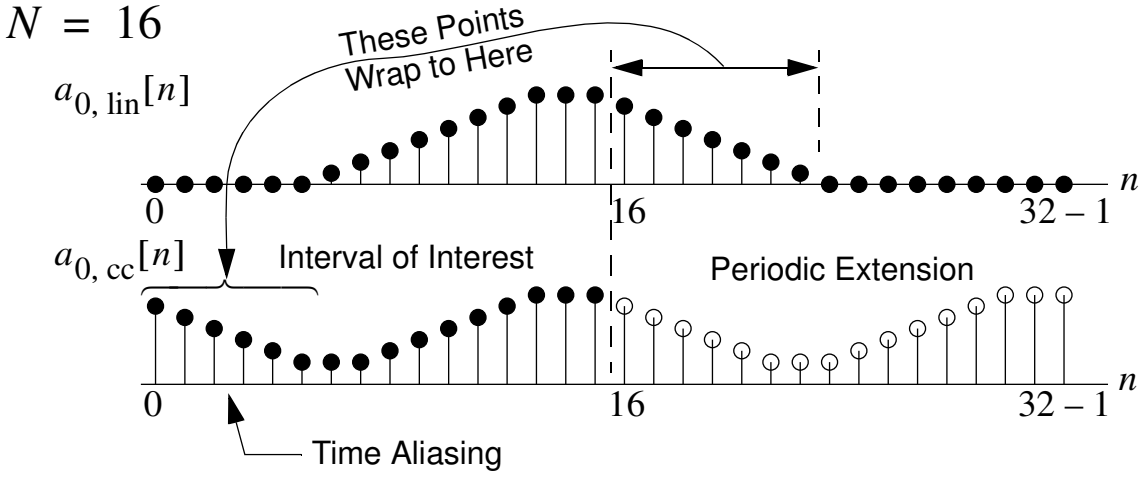
Input sequences $a_1[n]$ and $a_2[n]$

- The Python commands used to produce the linear convolution result using `filter()` and the circular convolution result with $N = 16$ are given below

```
# Define the sequences
N = 16 # try 20 and 24 to explore need for N >= 23
nN = arange(2*N)
a1 = hstack((ones(10), zeros(len(nN)-10)))
a2 = array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1])
# Linear convolution
a0 = signal.lfilter(a2,1,a1) # linear conv
subplot(211)
stem(nN,a0)
title(r'Linear vs Circular Convolution for $N_1+N_2-1=23$ &
   $N=$%d'% (N,))
```

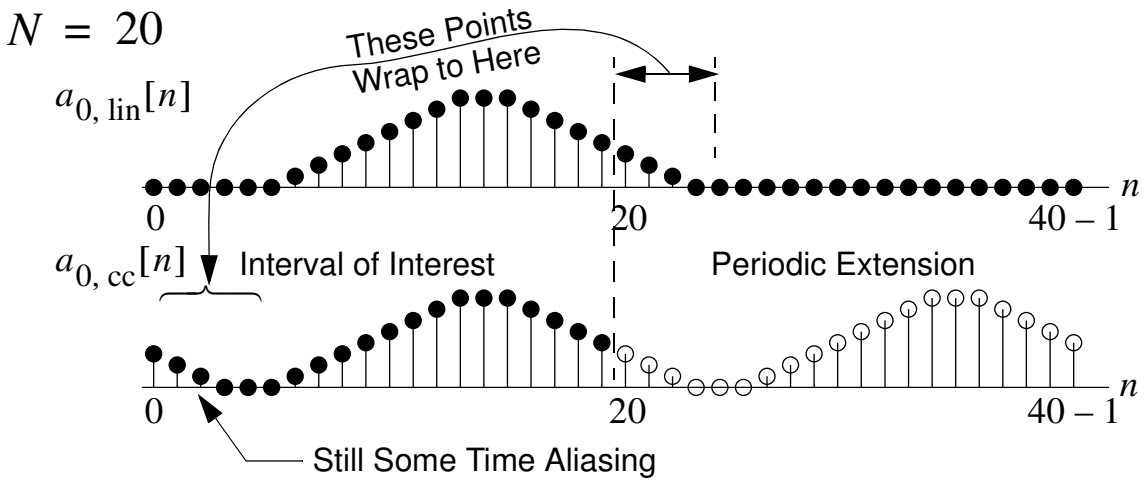
```
ylabel(r'$a_{0,lin}[n]$')
# Circular convolution using the FFT/IFFT (DFT/IDFT)
# Take real-part since we know convolution is real &
# a very small imaginary part may (will) exist
a0_circ = real(fft.ifft(fft.fft(a1,N)*fft.fft(a2,N)))
subplot(212)
stem(nN[:N],a0_circ)
stem(nN[N:],a0_circ,'g', markerfmt='go')
ylabel(r'$a_{0,cc}[n]$')
xlabel(r'Index $n$')
tight_layout()
```

- The input sequence lengths are effectively $N_1 = 10$ and $N_2 = 14$ respectively
- For circular convolution to perform linear convolution we expect that we need $N \geq N_1 + N_2 - 1 = 23$
- Results given below plot the output, $a_0[n]$, for the $N = 16$ case



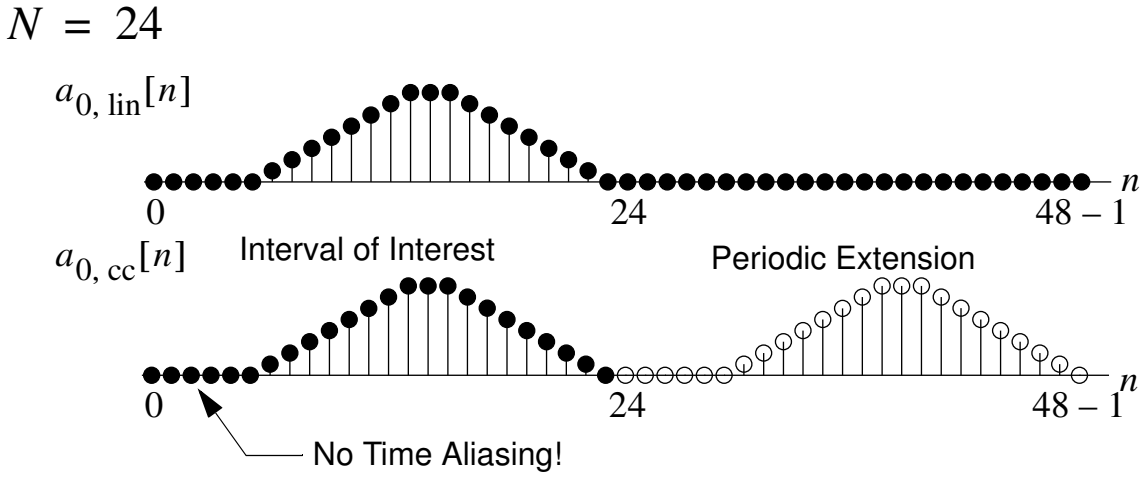
Linear and circular convolutions compared for $N = 16$

- As expected there is considerable *time aliasing*
- We now increase N to 20



Linear and circular convolutions compared for $N = 20$

- With $N = 20$ there is still a noticeable amount of time aliasing
- By setting $N = 24$ we should solve the aliasing problem



Linear and circular convolutions compared for $N = 24$

7.5.1 Filtering a Sequence of Possibly Infinite Duration

This is an extension to the convolution problem discussed above. Here $x[n]$ may be a sequence of possibly infinite duration, perhaps a speech signal. The second sequence, $h[n]$, is assumed to be of finite duration, P, such as the impulse response of an FIR filter.

The approach is to segment $x[n]$ into sections of finite length L, then filter (convolve) each segment using the DFT/IDFT. Afterwards the filtered sections are summed together.

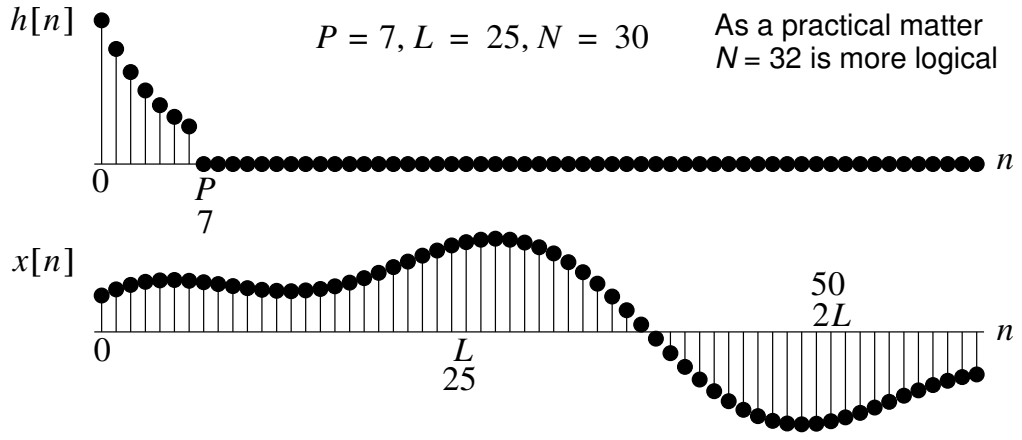
- Assuming $x[n] = 0,\ n < 0$, then we may decompose $x[n]$ such that

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

where

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

- The sequences are shown below



- The convolution of $x[n]$ with $h[n]$ can be written as

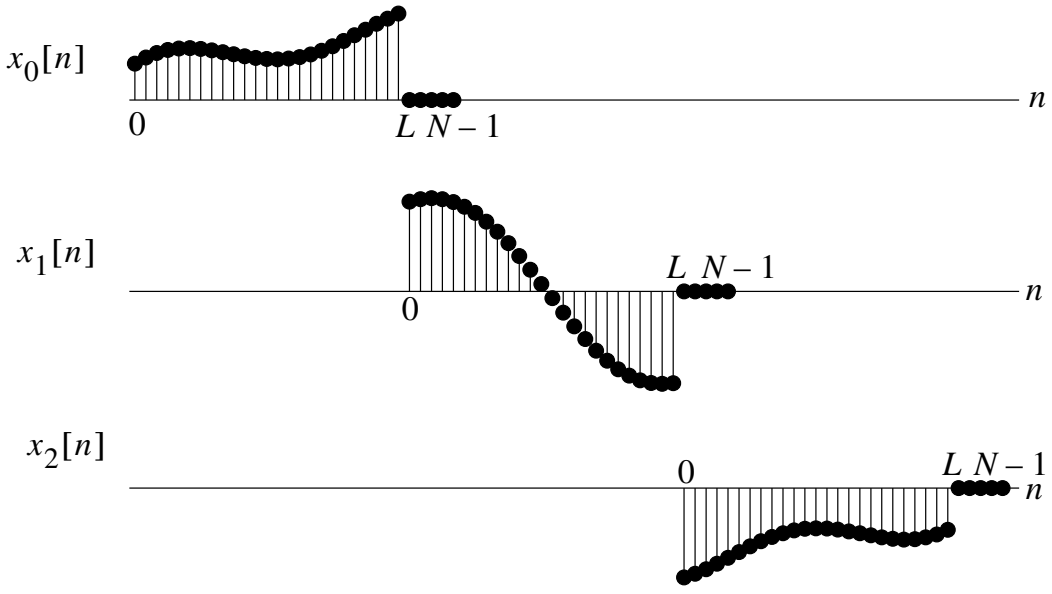
$$x[n] * h[n] = \sum_{r=0}^{\infty} x_r[n - rL] * h[n]$$

- Each convolution $x_r[n] * h[n]$ can be obtained using an $N \geq L + P - 1$-point DFT/IDFT which gives $y_r[n]$

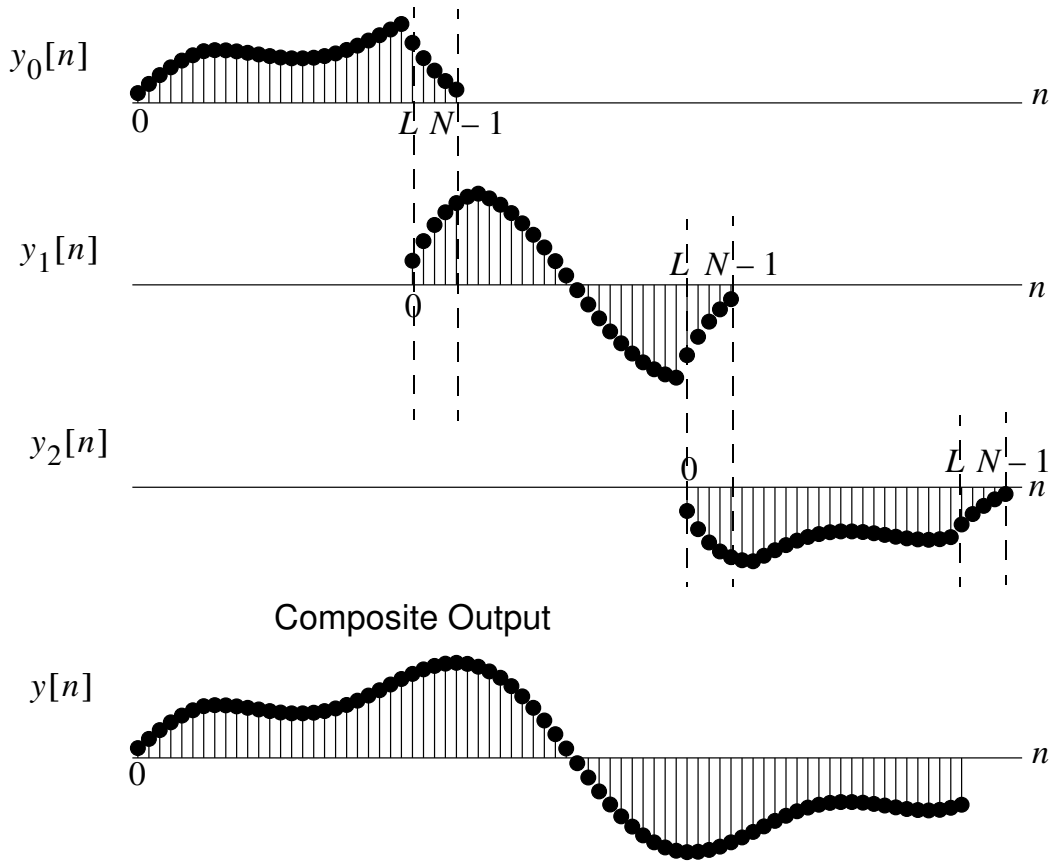
- The output $y[n]$ is then obtained by summing the output subsequences

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

- The method is summarized in the following figures:



Decomposition of $x[n]$ into nonoverlapping sections



The result of convolving each section with $h[n]$

7.6 Computation of the Discrete Fourier Transform

We have seen that the N-point DFT computes the Fourier transform of a sequence at N equally spaced frequencies on $0 \leq \omega < 2\pi$. Efficient algorithms for computation of the DFT are called *fast Fourier transform (FFT) algorithms*.

In general FFT algorithms calculate all N values of the DFT. When the DFT is not required over the entire frequency range $0 \leq \omega < 2\pi$, then the FFT may not be the most efficient algorithm.

To measure the computational complexity of an FFT algorithm

we will use the number of multiplications and additions. These are not the only useful measures, but for many applications they provide an indication of an algorithm's computational speed.

7.6.1 Direct Evaluation of the DFT

By definition the DFT of an N-point sequence is

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

- Each DFT point requires N complex multiplies and $N-1$ complex additions
- To compute all N points requires N^2 multiplies and $N(N-1)$ additions
- Each complex multiply is of the form

$$\begin{aligned} x[n]W_N^{kn} = &\mathrm{Re}\{x[n]\}\mathrm{Re}\{W_N^{kn}\} - \mathrm{Im}\{x[n]\}\mathrm{Im}\{W_N^{kn}\} \\ &+ j\left[\mathrm{Re}\{x[n]\}\mathrm{Im}\{W_N^{kn}\} + \mathrm{Im}\{x[n]\}\mathrm{Re}\{W_N^{kn}\}\right] \end{aligned}$$

  Thus each complex multiply requires four real multiplies and two real additions

- Each complex addition requires two real additions

7.6.2 Decimation-In-Time FFT Algorithms

Significant computational savings can be realized by decomposing the DFT into smaller DFT computations. With the *decimation-in-time algorithms* decomposition is performed by decimating the discrete-time sequence $x[n]$.

By choosing N to be a power of two (i.e. $N = 2^{\nu}$) the *radix-2* decimation-in-time algorithm can be developed.

- For the radix-2 algorithm N is even, thus we can separate $x[n]$ into even and odd $N/2$ point sequences

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

- In the first sum let $n = 2r$ and in the second let $n = 2r + 1$

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

or

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

- Note that

$$W_N^2 = e^{-2j(2\pi/N)} = e^{-j2\pi/(N/2)} = W_{N/2}$$

thus we can write

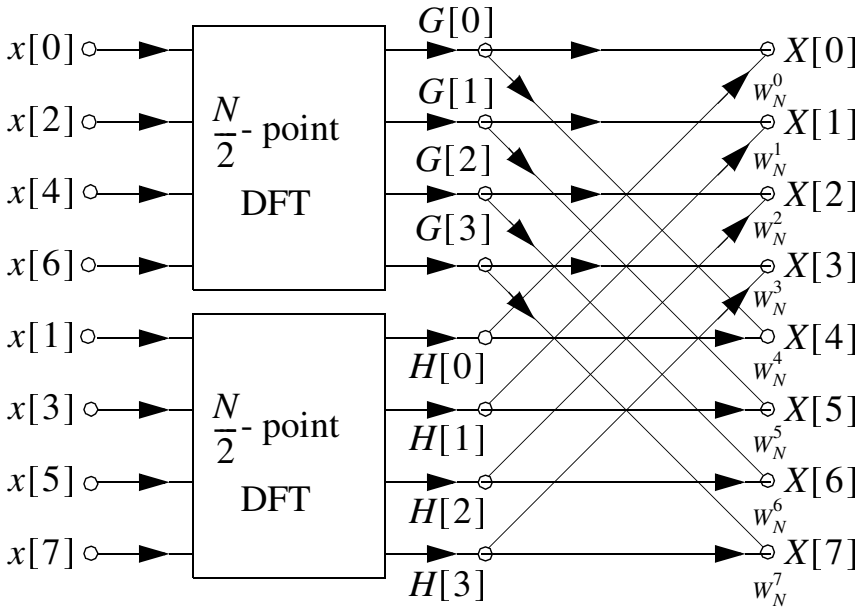
$$\begin{aligned} X[k] &= \sum_{r=0}^{(N/2)-1} x[2r]W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1]W_{N/2}^{rk} \\ &= G[k] + W_N^k H[k] \end{aligned}$$

where the terms $G[k]$ and $H[k]$ are $N/2$ point DFTs of the even and odd points of the original sequence $x[n]$

- The complex coefficients W_N^k are known as *twiddle factors*

- Observe that the index k runs from $k = 0,\ 1,\ \ldots,\ N-1$, but since $G[k]$ and $H[k]$ are only $N/2$-point DFTs they are only computed for $k = 0,\ 1,\ \ldots,\ (N/2)-1$

- The flow graph for an $N = 8$ DFT after one stage of decomposition is the following:



Eight point DFT flow graph after one stage of time decomposition

- The computation count for this flow graph is:
  - $\underbrace{2(N/2)^2}_{2\ N/2\text{ pt. DFTs}} + \underbrace{N}_{\text{DFT combining}}$ complex multiplications
  - $\underbrace{\approx 2(N/2)^2}_{2\ N/2\text{ pt. DFTs}} + \underbrace{N}_{\text{DFT combining}}$ complex additions
  - In summary at most $N + N^2/2$ complex multiplications and additions are required

  - Note that $N + N^2/2 < N^2$ for $N > 2$

- Since we have assumed that N is a power of two, we may repeat the decomposition process on each of the $N/2$ point DFTs to obtain $N/4$-point DFTs, i.e. we can write

$$G[k] = \sum_{l=0}^{(N/4)-1} x[4l]W_{N/2}^{2lk} + \sum_{l=0}^{(N/4)-1} x[4l+2]W_{N/2}^{(2l+1)k}$$

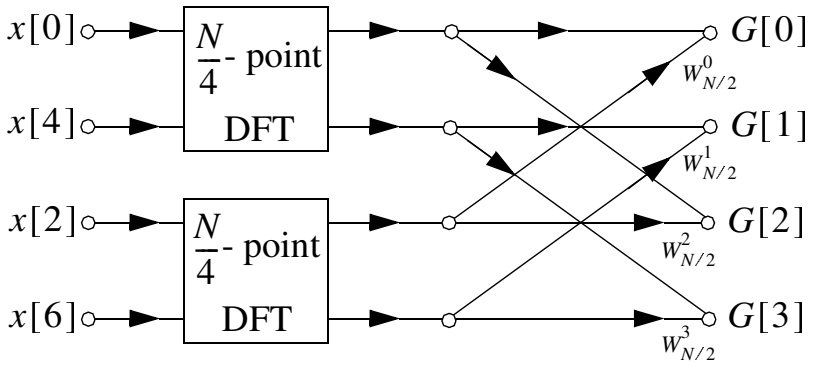
or

$$G[k] = \sum_{l=0}^{(N/4)-1} g[2l]W_{N/4}^{lk} + W_{N/2}^{k} \sum_{l=0}^{(N/4)-1} g[2l+1]W_{N/4}^{lk}$$

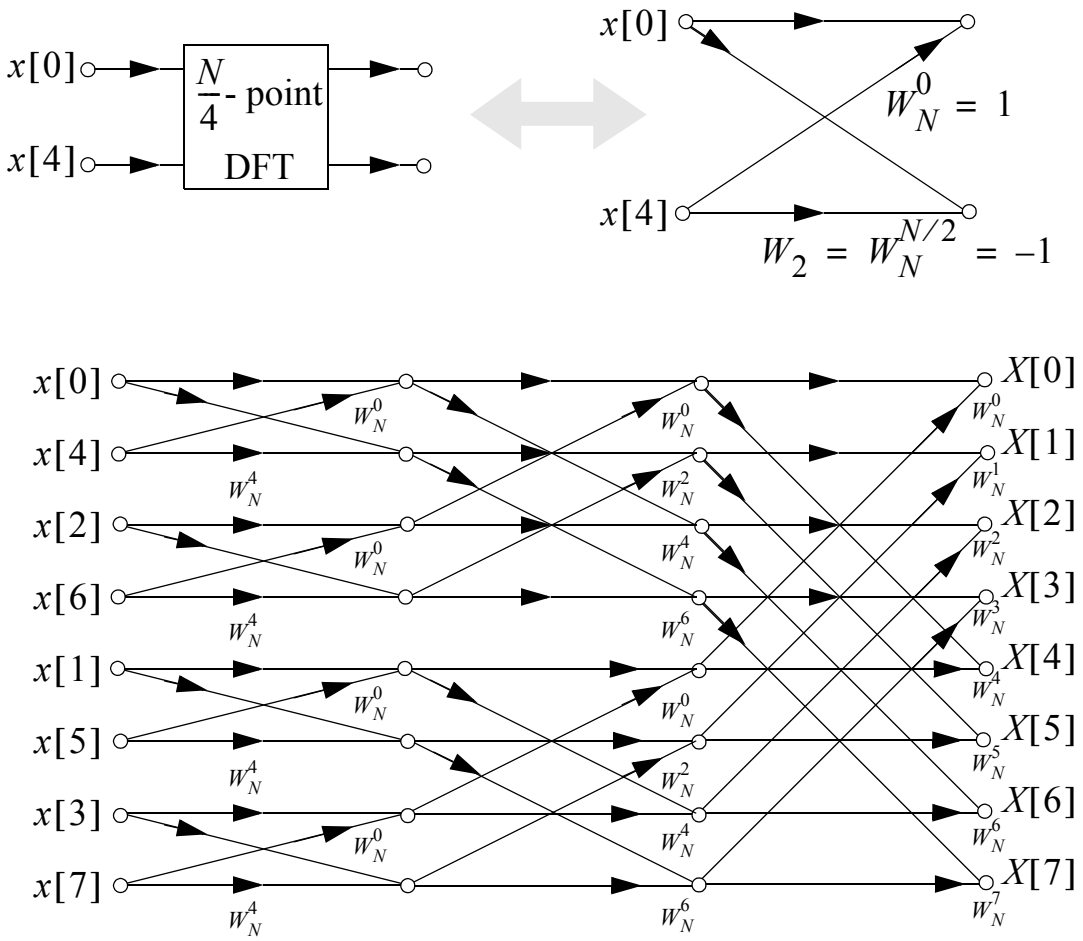
and

$$H[k] = \sum_{l=0}^{(N/4)-1} x[4l+1]W_{N/4}^{lk} + W_{N/2}^{k} \sum_{l=0}^{(N/4)-1} x[4l+3]W_{N/4}^{lk}$$

- Each $N/4$-point DFT is similar to the following:



- The flow graph for an 8-point DFT after two stages of decomposition is the following (note that the $N/4$-point DFTs are simply 2-point DFTs in this case)



Eight point DFT after two stages of time decomposition

- For an arbitrary power of two, the decomposition continues until only 2-point transforms remain. This requires $\nu = \log_2 N$ stages of decomposition
- Computation count after 2-stages of decomposition:
  - $4(N/4)^2 + N + N$ complex multiplications
  - Approximately $4(N/4)^2 + 2N$ additions
- After $\nu = \log_2 N$ stages of decomposition the number of multiplications and additions is exactly $N \log_2 N$ since each stage requires N multiplications and N additions

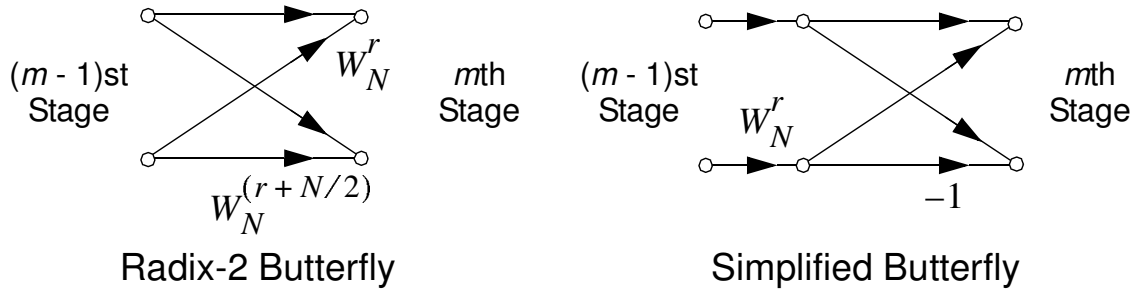
- As a simple comparison suppose $N = 2^{10} = 1024$

$$\begin{aligned} \text{Direct Evaluation} \Longrightarrow \quad & 2^{20} = 1,048,576 \\ \text{Radix-2 FFT} \Longrightarrow \quad & 10,240 \end{aligned}$$

  A reduction of about two orders of magnitude

- Further reduction can be obtained by noting that the basic computational element of the FFT, called a *butterfly*, can be simplified using the fact that

$$W_N^{r+N/2} = \underbrace{W_N^{N/2}}_{e^{-j\pi}} W_N^r = -W_N^r$$



- Using the simplified butterfly the computations count is reduced by one-half i.e. $(N/2)\log_2 N$

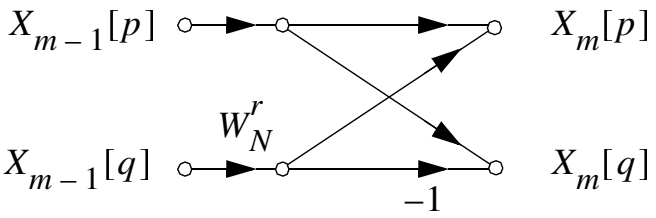
In-Place Computations

A property of the decimation-in-time radix-2 FFT as derived above is that all computations can be performed *in-place*.

- At each stage of the FFT the butterfly takes two numbers from the $(m-1)$st stage and uses them to compute corresponding

values of the mth stage

$$X_m[p] = X_{m-1}[p] + W_N^r X_{m-1}[q]$$
$$X_m[q] = X_{m-1}[p] - W_N^r X_{m-1}[q]$$



- Only one N-point complex array is needed to transform the $x[n]$ array to the $X[k]$ array

- A consequence of the in-place computation is the need for the input array of $x[n]$ values to be in *bit-reversed* order i.e. for $N = 8$ the input to output mapping is

$$x[0] = x[000] \Rightarrow X[000] = X[0]$$
$$x[4] = x[100] \Rightarrow X[001] = X[1]$$
$$x[2] = x[010] \Rightarrow X[010] = X[2]$$
$$x[6] = x[110] \Rightarrow X[011] = X[3]$$
$$x[1] = x[001] \Rightarrow X[100] = X[4]$$
$$x[5] = x[101] \Rightarrow X[101] = X[5]$$
$$x[3] = x[011] \Rightarrow X[110] = X[6]$$
$$x[7] = x[111] \Rightarrow X[111] = X[7]$$

Summary of Computational Requirements for a Direct DFT versus a Radix-2 FFT

Number of points N	Complex Mult. in Direct Comp. N^2	Complex Mult. in FFT Algorithm $(N/2)\log_2 N$	Speed Improvement Factor
4	16	4	4.0
8	64	12	5.3
16	256	32	8.0
32	1,024	80	12.8
64	4,096	192	21.3
128	16,384	448	36.6
256	65,536	1,024	64.0
512	262,144	2,304	113.8
1,024	1,048,576	5,120	204.8

Example 7.5: Radix-2 DIT FFT in ANSI C

```
1  # include <math.h>
2
3  typedef struct {
4          float u;
5          float v;
6          } complex_num;
7
8  void cfft(int log2n, complex_num *data, int ntype)
9  {
10 int     n,n1,n2,i,j,k,l,le,lel,id,ip;
11 double ur,ui,wr,wi,tr,ti,
12     pi = 4. * atan(1.);
13 float  fm,fl,sign,in;
14
15 /* ntype = 1 for DFT and ntype = -1 for IDFT */
16 sign = -1.; if (ntype < 0) sign = 1;
17 fm = log2n; n = pow(2.,fm);
18 n2 = n/2; n1 = n-1; j = 1;
```

```
19 for (i=1; i<=n1; i++) {
20     ip = i-1;
21     if (i<j) {
22         id = j-1;
23         tr = (*(data+id)).u; ti = (*(data+id)).v;
24         (*(data+id)).u = (*(data+ip)).u;
25         (*(data+id)).v = (*(data+ip)).v;
26         (*(data+ip)).u = tr; (*(data+ip)).v = ti;
27     }
28     k = n2;
29     while (k<j) {
30         j = j-k; k = k/2;
31     }
32     j = j+k;
33 }
34
35 for (k=1; k<=log2n; k++) {
36     fl = k; le = pow(2.,fl);
37     lel = le/2; ur = 1.; ui = 0.;
38     wr = cos(pi/lel);
39     wi = sign*sin(pi/lel);
40     for (j=1; j<=lel; j++) {
41         for (i=j; i<=n; i=i+le) {
42             ip = i-1; id = ip+lel;
43             tr = (*(data+id)).u*ur - (*(data+id)).v*ui;
44             ti = (*(data+id)).u*ui + (*(data+id)).v*ur;
45             (*(data+id)).u = (*(data+ip)).u - tr;
46             (*(data+id)).v = (*(data+ip)).v - ti;
47             (*(data+ip)).u = (*(data+ip)).u + tr;
48             (*(data+ip)).v = (*(data+ip)).v + ti;
49         }
50     tr = ur*wr - ui*wi;
51     ti = ur*wi + ui*wr;
52     ur = tr; ui = ti;
53     }
54 }
55 if (ntype < 0) {
56     in = 1./n;
57     for (i=0; i<n; i++) {
58     (*(data+i)).u = (*(data+i)).u * in;
59     (*(data+n)).v = (*(data+n)).v * in;
60     }
61 }
62 return;
63 }
```

7.6.3 Decimation-In-Frequency FFT Algorithms

With the *decimation-in-frequency (DIF)* algorithms decomposition is performed by decimating the output sequence $X[k]$. A radix-2 DIF algorithm can be developed by considering $X[k]$ as consisting of even and odd numbered frequency samples

- Begin by writing $X[k]$ as two sums

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

Let $r = n - N/2$ in the second sum

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

but $W_N^{k\,N/2} = (-1)^k$ so,

$$X[k] = \sum_{n=0}^{(N/2)-1} \left[x[n] + (-1)^k x[n + N/2]\right] W_N^{nk}$$

- Consider k = even, then $k = 2r,\ r = 0,\ 1,\ \ldots,\ N/2 - 1$ and

$$X[2r] = \sum_{n=0}^{(N/2)-1} \underbrace{[x[n] + x[n + N/2]]}_{g[n]} W_{N/2}^{nr}$$

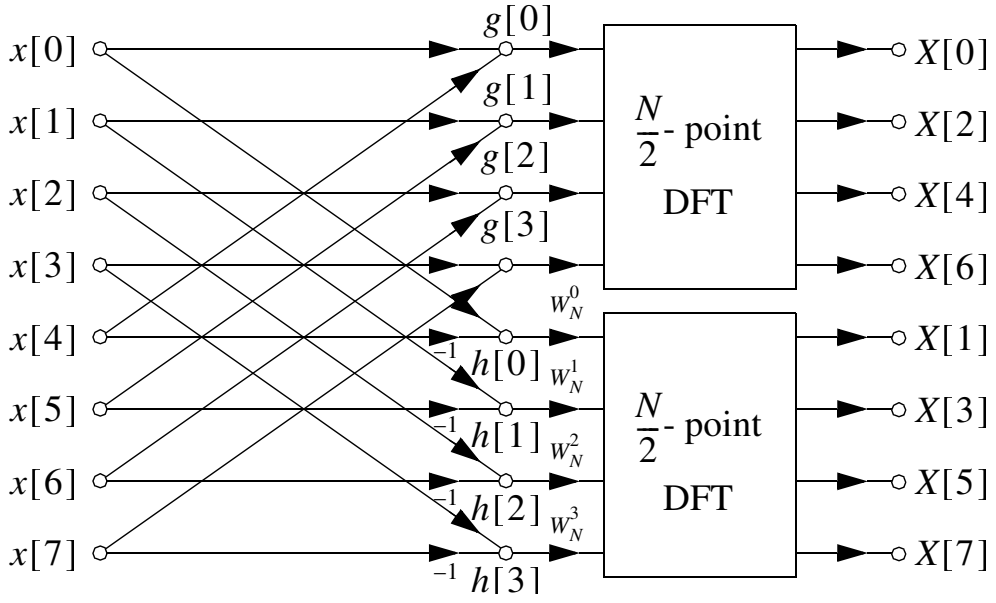
- Consider k = odd, then $k = 2r + 1,\ r = 0,\ 1,\ \ldots,\ N/2 - 1$ and

$$X[2r+1] = \sum_{n=0}^{(N/2)-1} \underbrace{[x[n] - x[n+N/2]]}_{h[n]} W_N^n W_{N/2}^{nr}$$

- Note that $g[n]$ and $h[n]$ are $N/2$ point sequences formed by operating on $x[n]$ using a simple combining algebra

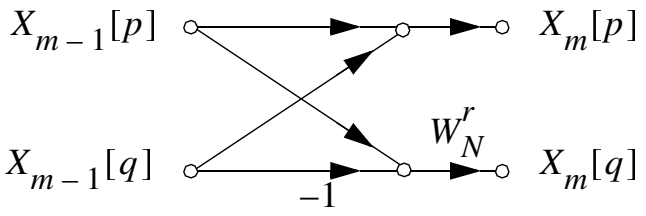
- The output sequence $X[k]$ is then obtained by computing the $N/2$ point DFT of the sequences $g[n]$ and $h[n]W_N^n$

- The flow graph for an $N = 8$ DFT after one stage of decomposition is the following:



- By further decomposing the sequences $g[n]$ and $h[n]$ into even and odd parts, the DFT is reduced down to $N/4$ point transforms with appropriate combining algebra on the inputs

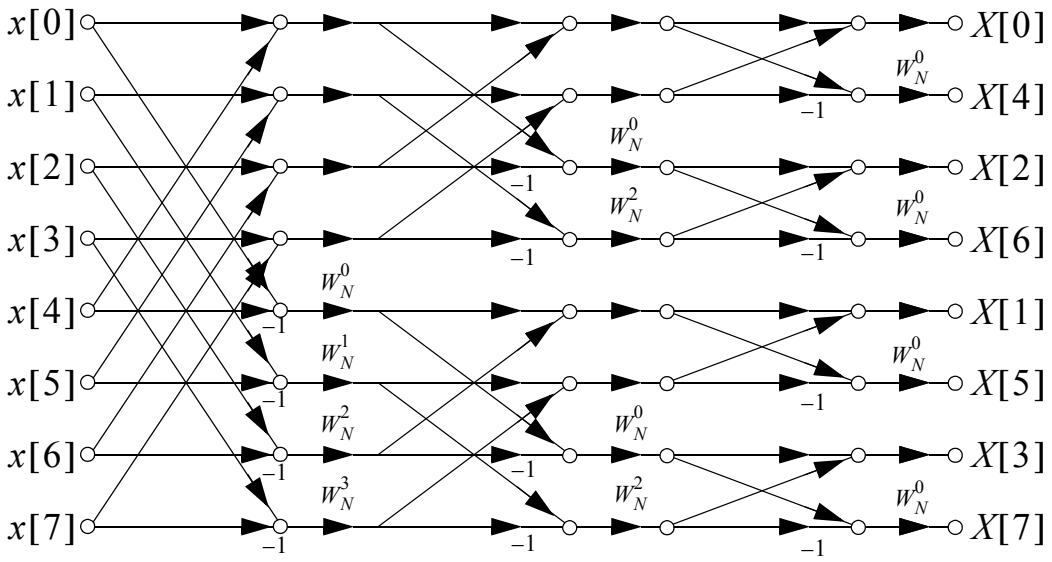
- After $\log_2 N$ decompositions the basic computational element is again a 2-point DFT



Simplified Radix-2 DIF butterfly

- The total computation count when using the butterfly shown above is once again $(N/2)\log_2 N$ complex multiplications and $N \log_2 N$ complex additions

- The flow graph for an 8-point DIF DFT is shown below:



- Note: The decimation-in-time (DIT) and DIF flow graphs are actually related by network transposition.

- DIF radix-2 FFT Observations:

- The input is in "normal order" while the ouput is in bit reversed order (the opposite of the DIT FFT)
- The butterfly structure allows in-place computations

7.6.4 IDFT Computation using an FFT

To compute the IDFT using one of the FFT algorithms just described is quite simple.

- The computation is identical to the DFT except for a $1/N$ scale factor, and the twiddle factors must be conjugated (i.e. $W_N^k \rightarrow W_N^{-k}$)
- Another way of looking at this is

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

$$= \frac{1}{N}\underbrace{\left[\sum_{k=0}^{N-1} X^*[k]W_N^{kn}\right]}_{\text{DFT}\{X^*[k]\}}^* = \frac{1}{N}\left[\text{DFT}\{X^*[k]\}\right]^*$$

7.7 Applications of FFT Algorithms

In this section a brief overview of FFT (DFT) applications will be given. It is the efficiency with which the DFT can be computed via the FFT that makes many applications attractive.

7.7.1 The DFT of Two Real N-Point Sequences using an N-point FFT

The FFT is by definition designed to operate on complex input data. The input may be real, but the weights W_N^k are in general complex, thus the array used for in-place computations must always be complex.

- Suppose $x_1[n]$ and $x_2[n]$ are two real N-point sequences
- Define the complex sequence $x[n]$ to be

$$x[n] = x_1[n] + jx_2[n], \;\; 0 \leq n \leq N-1$$

- Since the DFT is a linear operator it follows that

$$X[k] = X_1[k] + jX_2[k]$$

  where $X_1[k]$ and $X_2[k]$ are of course complex quantities

- To extract $X_1[k]$ and $X_2[k]$ from $X[k]$ first note that

$$x_1[n] = \frac{1}{2}\{x[n] + x^*[n]\}$$
$$x_2[n] = \frac{1}{2j}\{x[n] - x^*[n]\}$$

- Taking the DFT of both sides of the above equations results in

$$X_1[k] = \frac{1}{2}\left[\text{DFT}\{x[n]\} + \text{DFT}\{x^*[n]\}\right]$$
$$X_2[k] = \frac{1}{2j}\left[\text{DFT}\{x[n]\} - \text{DFT}\{x^*[n]\}\right]$$

- Using the DFT property that $x^*[n] \overset{\mathcal{DFT}}{\longleftrightarrow} X^*[N-k]$ we can write

$$X_1[k] = \frac{1}{2}\left\{X[k] + X^*[N-k]\right\}$$
$$X_2[k] = \frac{1}{2j}\left\{X[k] - X^*[N-k]\right\}$$

  for $0 \leq k \leq N-1$

7.7.2 The DFT of a Real $2N$-Point Sequence using an N-point FFT

In this section we will show that it is possible to compute the DFT of a $2N$-point real sequence using an N-point FFT.

- Let $y[n]$ be a real $2N$-point sequence. Map the even indexed points to the N-point sequence $x_1[n]$ and the odd indexed points to the N-point sequence $x_2[n]$ i.e.

$$x_1[n] = y[2n], \qquad x_2[n] = y[2n+1], \quad 0 \leq n \leq N-1$$

- Using the results of the previous section for two real sequences, let $x[n]$ be the complex N-point sequence

$$x[n] = x_1[n] + jx_2[n]$$

- We know that

$$X_1[k] = \frac{1}{2}\{X[k] + X^*[N-k]\}$$
$$X_2[k] = \frac{1}{2j}\{X[k] - X^*[N-k]\}$$

- Direct evaluation of the $2N$-point DFT of $y[n]$ yields

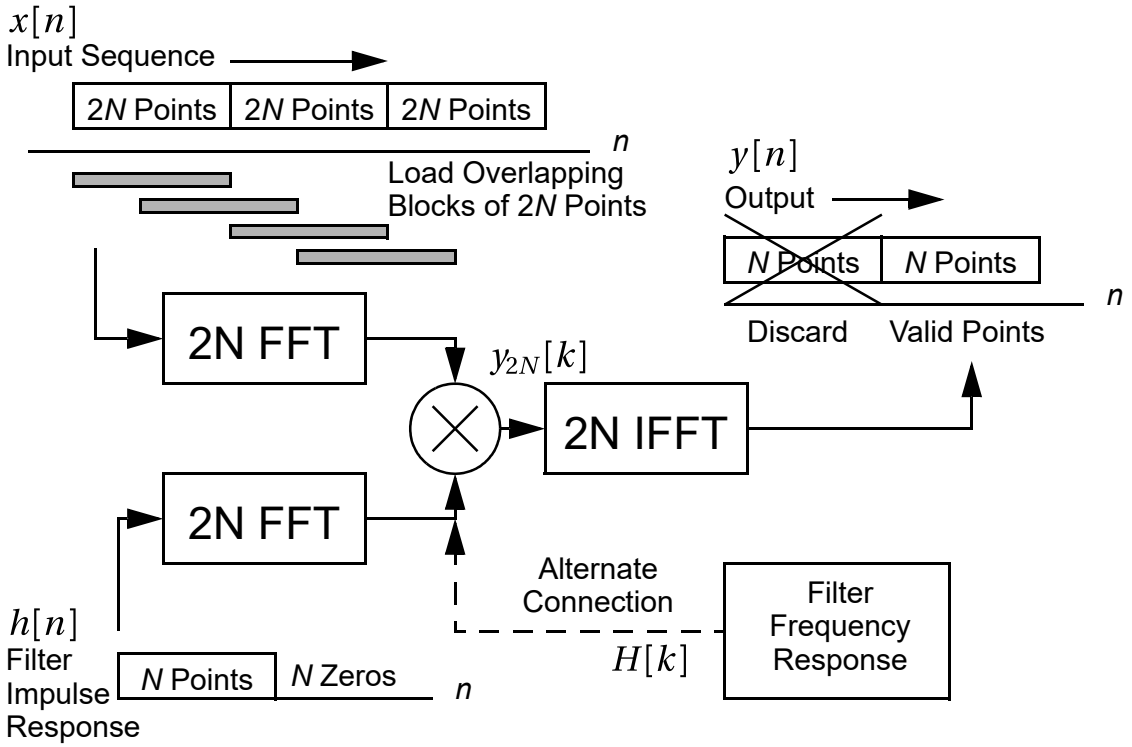
$$\begin{aligned} Y[k] &= \sum_{n=0}^{2N-1} y[n]W_{2N}^{nk} \\ &= \sum_{n=0}^{N-1}\left[y[2n]W_{2N}^{2nk} + y[2n+1]W_{2N}^{(2n+1)k}\right] \\ &= \sum_{n=0}^{N-1} x_1[n]W_N^{nk} + W_{2N}^k \sum_{n=0}^{N-1} x_2[n]W_N^{nk} \\ &= \mathrm{DFT}_N\{x_1[n]\} + W_{2N}^k\,\mathrm{DFT}_N\{x_2[n]\} \end{aligned}$$

- From the above analysis we conclude that

$$Y[k] = \begin{cases} X_1[k] + W_{2N}^k X_2[k], & 0 \le k \le N-1 \\ X_1[k-N] + W_{2N}^k X_2[k-N], & N \le k \le 2N-1 \end{cases}$$

7.7.3 Transform Domain Linear Filtering

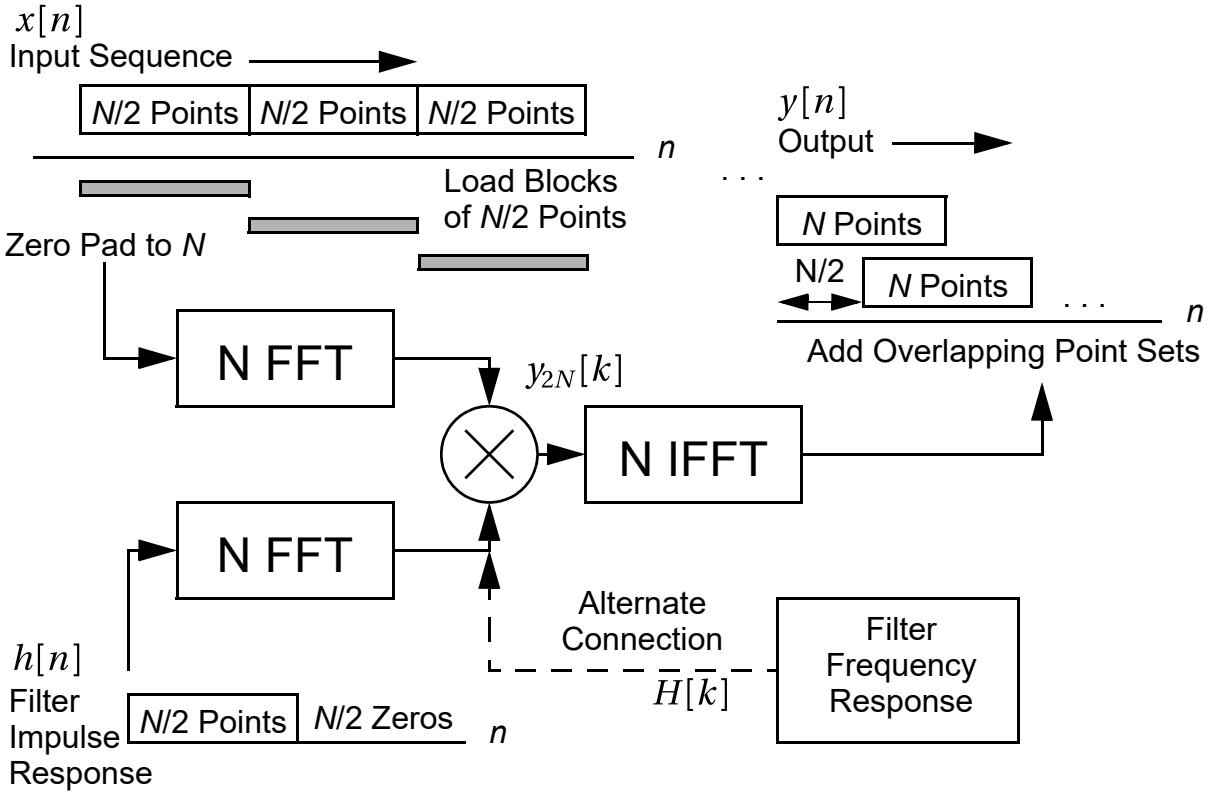
- Formalizing Section 7.5.1 *Filtering a Sequence of Possibly Infinite Duration* we consider two techniques for using FFT/IFFT pair to perform transform domain linear filtering
- As in the earlier discussion the basic assumption here is that the signal sample stream we wish to filter may have infinite duration, but the impulse response of the filter must be finite
- One standard approach is known as the *overlap and save* method of FIR filtering



Overlap and Save Filtering

- As the impulse response length grows the transform domain approach becomes more efficient than the time domain convolution sum, implemented in say a direct form FIR structure

- Another way of performing a transform domain convolution is with the overlap and add method



Overlap and Add Filtering

- The scikit-dsp-comm module `sigsys` contains overlap and add and overlap and save filtering functions:

```
Signature: ss.os_filter(x, h, N, mode=0)
Docstring:
Overlap and save transform domain FIR filtering.

This function implements the classical overlap and save method of
transform domain filtering using a length P FIR filter.

Parameters
----------
x : input signal to be filtered as an ndarray
h : FIR filter coefficients as an ndarray of length P
N : FFT size > P, typically a power of two
```

```
mode : 0 or 1, when 1 returns a diagnostic matrix

Signature: ss.oa_filter(x, h, N, mode=0)
Docstring:
Overlap and add transform domain FIR filtering.

This function implements the classical overlap and add method of
transform domain filtering using a length P FIR filter.

Parameters
----------
x : input signal to be filtered as an ndarray
h : FIR filter coefficients as an ndarray of length P
N : FFT size > P, typically a power of two
mode : 0 or 1, when 1 returns a diagnostic matrix
```

Example 7.6: Prototype Overlap and Save in Python

- Referring to the overlap and save block diagram we implement a real FIR filtering routine for real coefficients using `fft.fft` and `fft.ifft` from the numpy FFT submodule
- Below is a listing extracted from a Jupyter notebook:

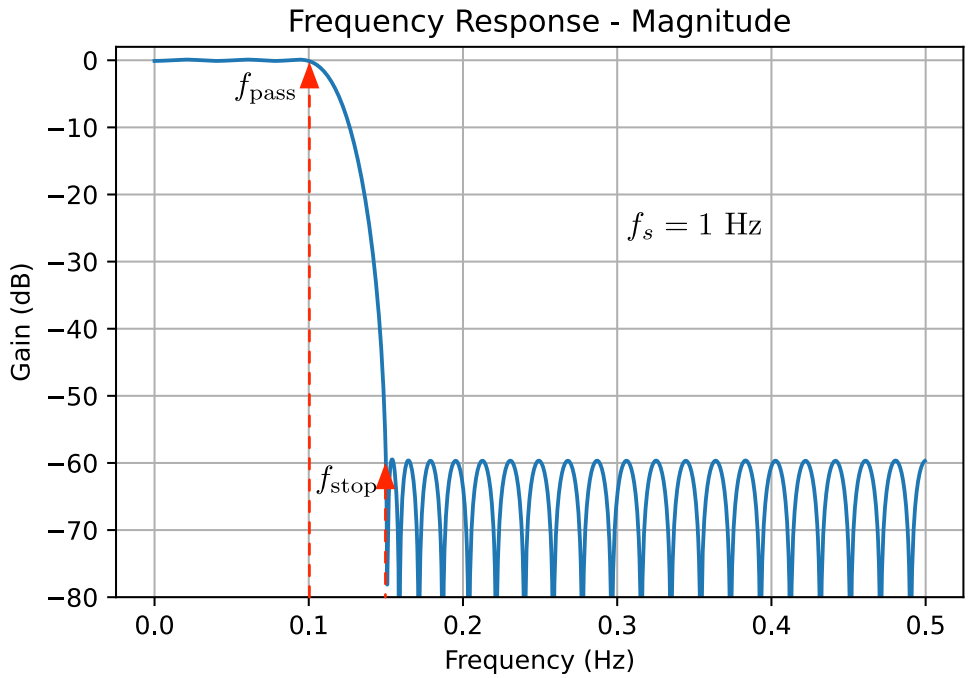
```
1  def fft_os_fir(x,N_fft,b):
2  """
3  Overlap and Save FIR filtering
4
5  y = fft_os_fir(x,N_fft,b)
6
7  x = real input signal ndarray
8  N_fft = FFT half length, i.e. overlap length N
9  b = real FIR  filter coefficients of length <= N_fft
10
11 y = filtered output (real as input is real)
12
13 Mark Wickert October 2021
14 """
15 b_zp = zeros(2*N_fft) # zero pad coeff. array
16 b_zp[:len(b)] = b # place coeff. at the start
17 B_zp = fft.fft(b_zp) # transform filter coeff.
```

```
N_out_buf = len(x)//(N_fft) # Number of output buffers
y = zeros(N_fft*N_out_buf) # define the output array
x_state = zeros(N_fft) # state array to hold past inputs
for k in range(N_out_buf):
    # Fill 2*N_fft array with old and new subarrays
    x_in = hstack((x_state,x[k*N_fft:(k+1)*N_fft]))
    Y_all = B_zp*fft.fft(x_in) # freq. domain filter
    y_all = fft.ifft(Y_all) # back to time domain
    # save the upper index points real-part
    y[k*N_fft:(k+1)*N_fft] = y_all[N_fft:].real
    x_state = x[k*N_fft:(k+1)*N_fft]
return y
```

- To test the `fft_os_fir()` function we design an equal-ripple lowpass filter with sampling rate, $f_s = 1$ Hz, as characterized below:
  - The passband cutoff is 0.1 Hz
  - The stopband cutoff is 0.15 Hz
  - The passband ripple is 0.1 dB
  - The stopband attenuation is 60 dB
- The number of FIR taps is 53, so from the block diagram we can choose N to be as small as 64, as this is nearest power of two

```
import sk_dsp_comm.fir_design_helper as fir_h

b = fir_h.fir_remez_lpf(0.1,0.15,0.1,60,n_bump=3)
# Plot the frequency response using freq_resp_list
fir_h.freqz_resp_list([b],[1],'dB',fs=1.0)
ylim([-80,2])
grid();
len(b)
53
```

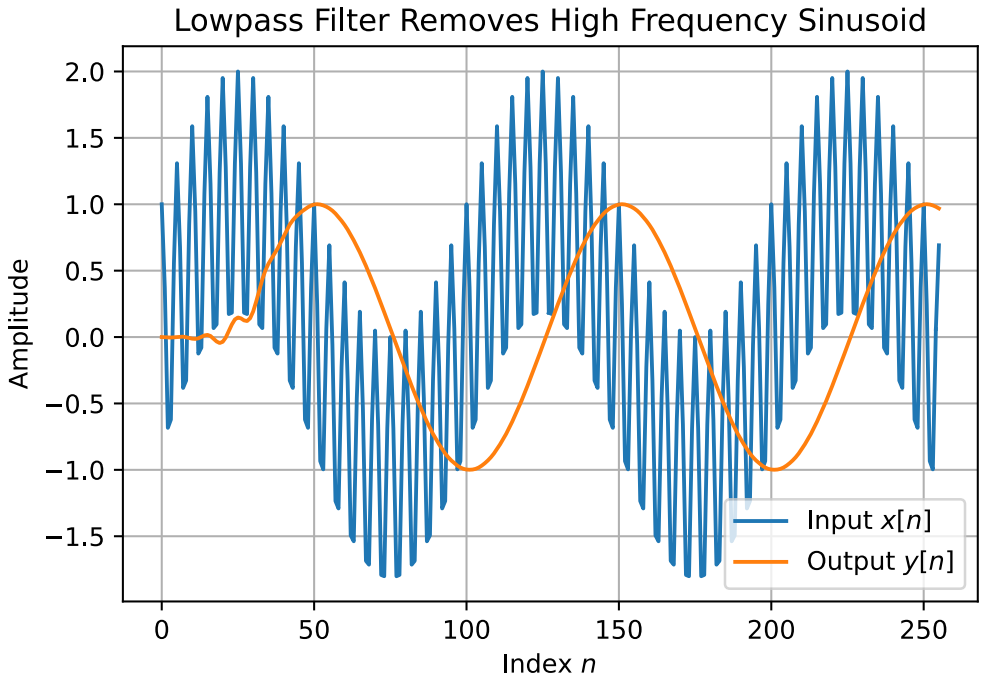


- The overlap and save filter is tested with a sum of two sinusoids signal having frequencies of 0.01 and 0.2 Hz; the lowpass filter should push the gain of the 0.2 Hz term down by 60 dB

```
n = arange(0,256)
x = cos(2*pi*0.2*n) + sin(2*pi*0.01*n)
```

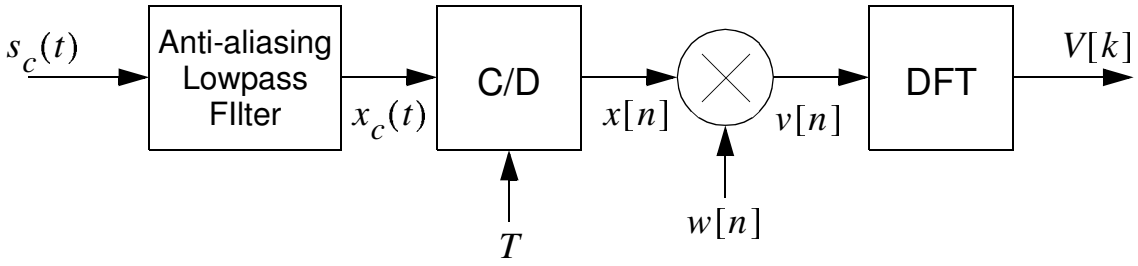
- The simulation code below pass the signal $x[n]$ to the input and collects the output in $y[n]$

```
y = fft_os_fir(x,64,b)
plot(n,x)
plot(n,y)
xlabel(r'Index $n$')
ylabel(r'Amplitude')
title(r'Lowpass Filter Removes High Frequency Sinusoid')
legend(('Input $x[n]$','Output $y[n]$'),loc='best')
grid();
```



7.7.4 Fourier Analysis of Continuous-Time Signals

A very popular FFT (DFT) application is performing spectral analysis of continuous-time signals. The block diagram of a basic Fourier analysis processor is shown below. $s_c(t)$ is the continuous time input signal and $w[n]$ is a finite-duration window sequence.



- The antialiasing filter $H_{aa}(j\Omega)$ is used to remove (minimize) aliasing that results from uniform sampling

- Multiplication of $x[n]$ by the window sequence $w[n]$ is required to match a possibly infinite duration $x[n]$ to the FFT

- In the frequency domain $x_c(t)$ is related to $x[n]$ by

$$X(e^{j\omega}) = \frac{1}{T}\sum_{r=-\infty}^{\infty} X_c\left(j\frac{\omega}{T} + j\frac{2\pi r}{T}\right)$$

- Since the anti-aliasing filter must be realizable some aliasing or spectral overlap of $X_c(j\,\Omega)$ translates can be expected

- To minimize aliasing the design may utilize a sharp roll-off antialiasing filter or/and use oversampling followed by a sharp roll-off digital filter and decimator (downsampler) to reduce the sampling rate

- In practice $x[n]$ will actually be a digital signal which means that it also contains quantization error (noise). The signal-to-quantization noise ratio can be made negligible if the number of quantizer bits is large.

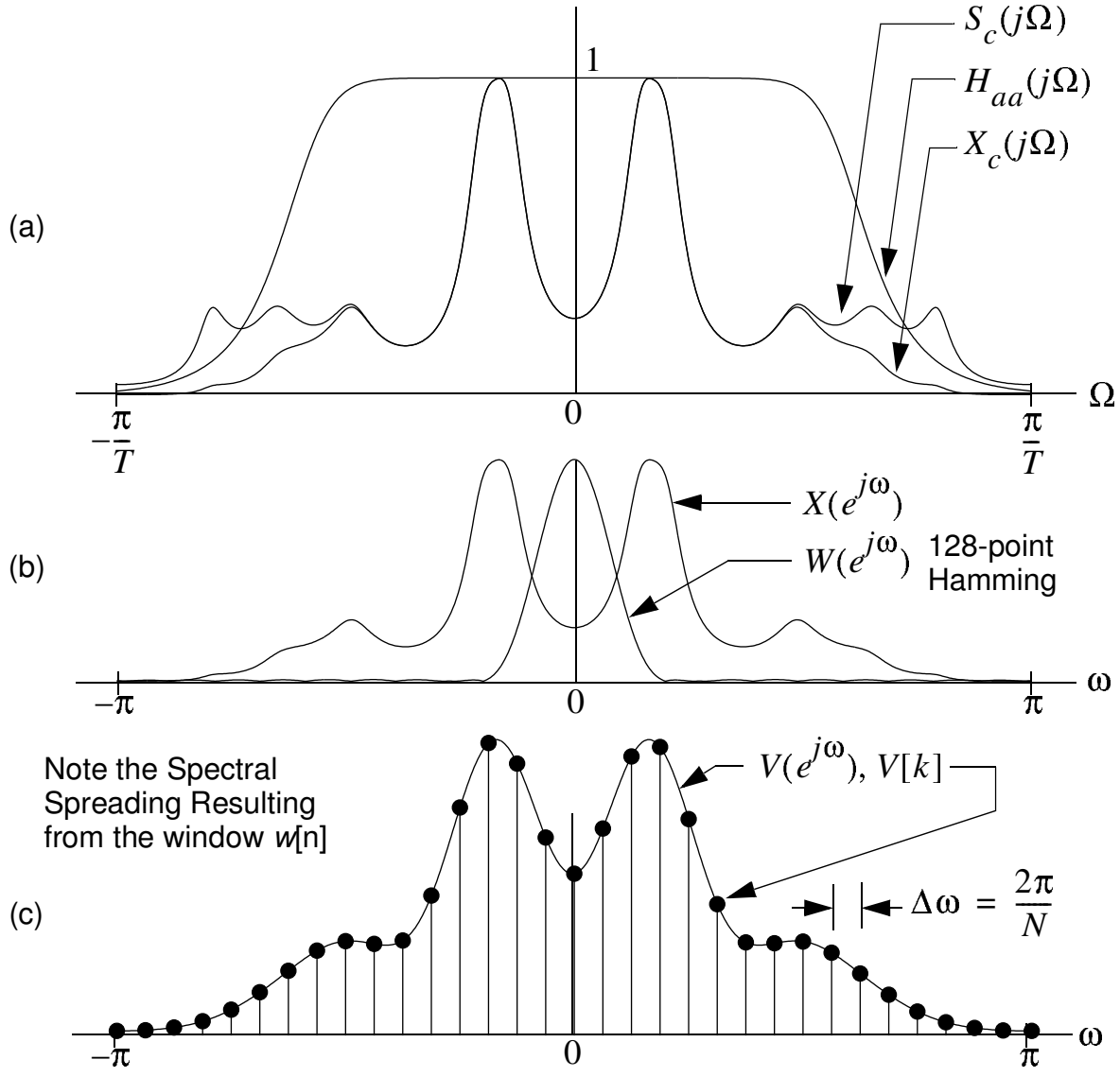
- Since $v[n] = x[n]w[n]$, $V(e^{j\omega})$ is the periodic convolution of $X(e^{j\omega})$ with $W(e^{j\omega})$ i.e.

$$V(e^{j\omega}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\theta})W(e^{j(\omega-\theta)})\,d\theta$$

- The FFT (DFT) of $v[n]$ is just

$$V[k] = V(e^{j\omega})|_{\omega=2\pi k/N},\ \ 0 \le k \le N-1$$

  where N is the FFT length



Fourier transforms in the Fourier analysis processor shown earlier. (a) Input spectrum, Anti-aliasing filter frequency response, and Filtered continuous-time spectrum; (b) Spectrum of the sampled signal and Fourier transform of the window sequence; (c) Spectrum of the windowed discrete-time signal.

- The frequency locations and spacing between FFT samples in terms of the continuous-time frequency variable Ω is

$$\Omega_k = \frac{2\pi k}{NT} \text{ rad/sec}, \quad 0 \leq k \leq N-1$$
$$\Delta\Omega = \frac{2\pi}{NT} \text{ rad/sec}$$

  In terms of the frequency variable $f = \Omega/(2\pi)$ the equivalent values are

$$f_k = \frac{k}{NT} = \frac{kf_s}{N} \text{ Hz}, \quad 0 \leq k \leq N-1$$
$$\Delta f = \frac{1}{NT} = \frac{f_s}{N} \text{ Hz}$$

FFT Analysis of Sinusoidal Signals

In this section we will discuss the effects of windowing and zero padding on the FFT analysis of sinusoids.

Consider the continuous-time signal

$$s_c(t) = A_0 \cos(\Omega_0 t + \theta_0) + A_1 \cos(\Omega_1 t + \theta_1)$$

- The Fourier transform of $s_c(t)$ is

$$S_c(j\Omega) = \pi \left\{ A_0 \left[e^{j\theta_0} \delta(\Omega - \Omega_0) + e^{-j\theta_0} \delta(\Omega + \Omega_0) \right] + A_1 \left[e^{j\theta_1} \delta(\Omega - \Omega_1) + e^{-j\theta_1} \delta(\Omega + \Omega_1) \right] \right\}$$

- If we assume ideal sampling with no aliasing (i.e. $\Omega_0 T,\ \Omega_1 T < \pi$) and no quantization error, then the sampler output in the Fourier analysis processor is

$$x[n] = A_0 \cos(\omega_0 n + \theta_0) + A_1 \cos(\omega_1 n + \theta_1)$$

  where $\omega_0 = \Omega_0 T$ and $\omega_1 = \Omega_1 T$

- The Fourier transform of $x[n]$ on $[-\pi, \pi)$ is just

$$X(e^{j\omega}) = \pi \left\{ A_0 \left[e^{j\theta_0} \delta(\omega - \omega_0) + e^{-j\theta_0} \delta(\omega + \omega_0) \right] + A_1 \left[e^{j\theta_1} \delta(\omega - \omega_1) + e^{-j\theta_1} \delta(\omega + \omega_1) \right] \right\}$$

The Effect of Windowing – Resolution versus Leakage

Due to the fact that the FFT operates on a finite record of "data" and that in practice we can only collect a finite number of signal samples, the observed spectrum, $V(e^{j\omega})$ is a "distorted" version of $X(e^{j\omega})$. The spectral distortion or spreading is commonly referred to as *spectral leakage*. Specifically for the case of two sinusoidal sequences the sifting property of the delta function gives

$$V(e^{j\omega}) = \frac{A_0}{2} \left[e^{j\theta_0} W(e^{j(\omega-\omega_0)}) + e^{-j\theta_o} W(e^{j(\omega+\omega_0)}) \right] + \frac{A_1}{2} \left[e^{j\theta_1} W(e^{j(\omega-\omega_1)}) + e^{-j\theta_1} W(e^{j(\omega+\omega_1)}) \right]$$

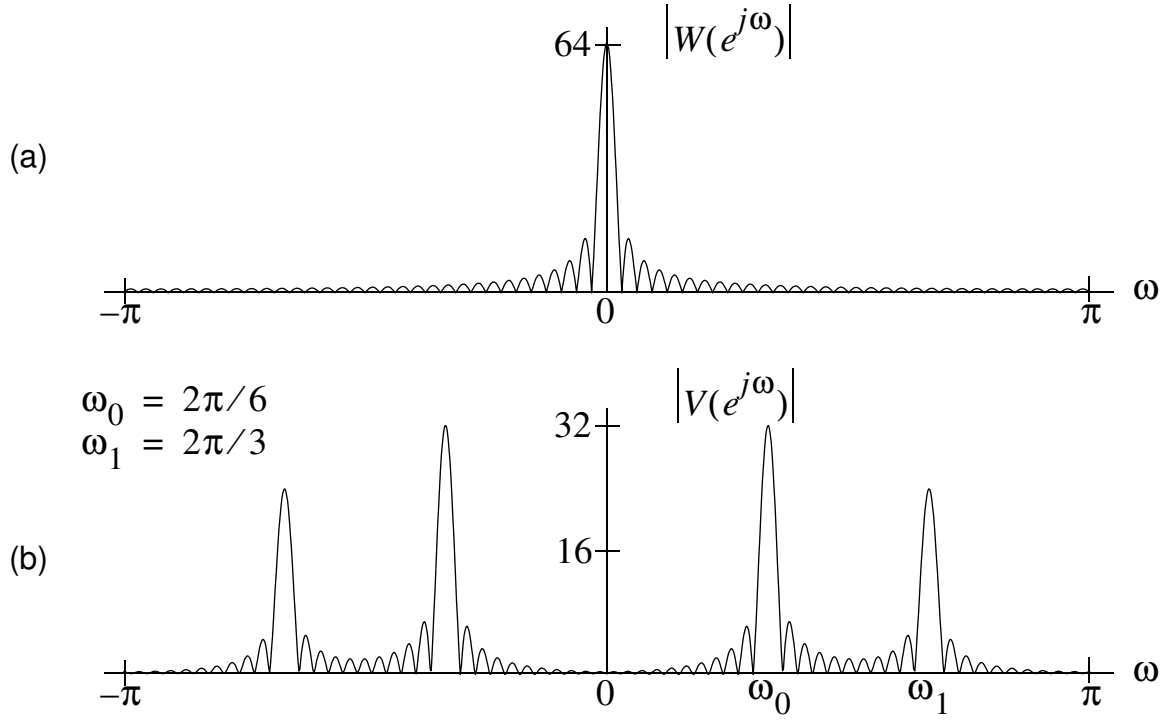
Note that the window Fourier transform is simply replicated at $\pm\omega_0$ and $\pm\omega_1$.

Example 7.7: **Two Sinusoids**

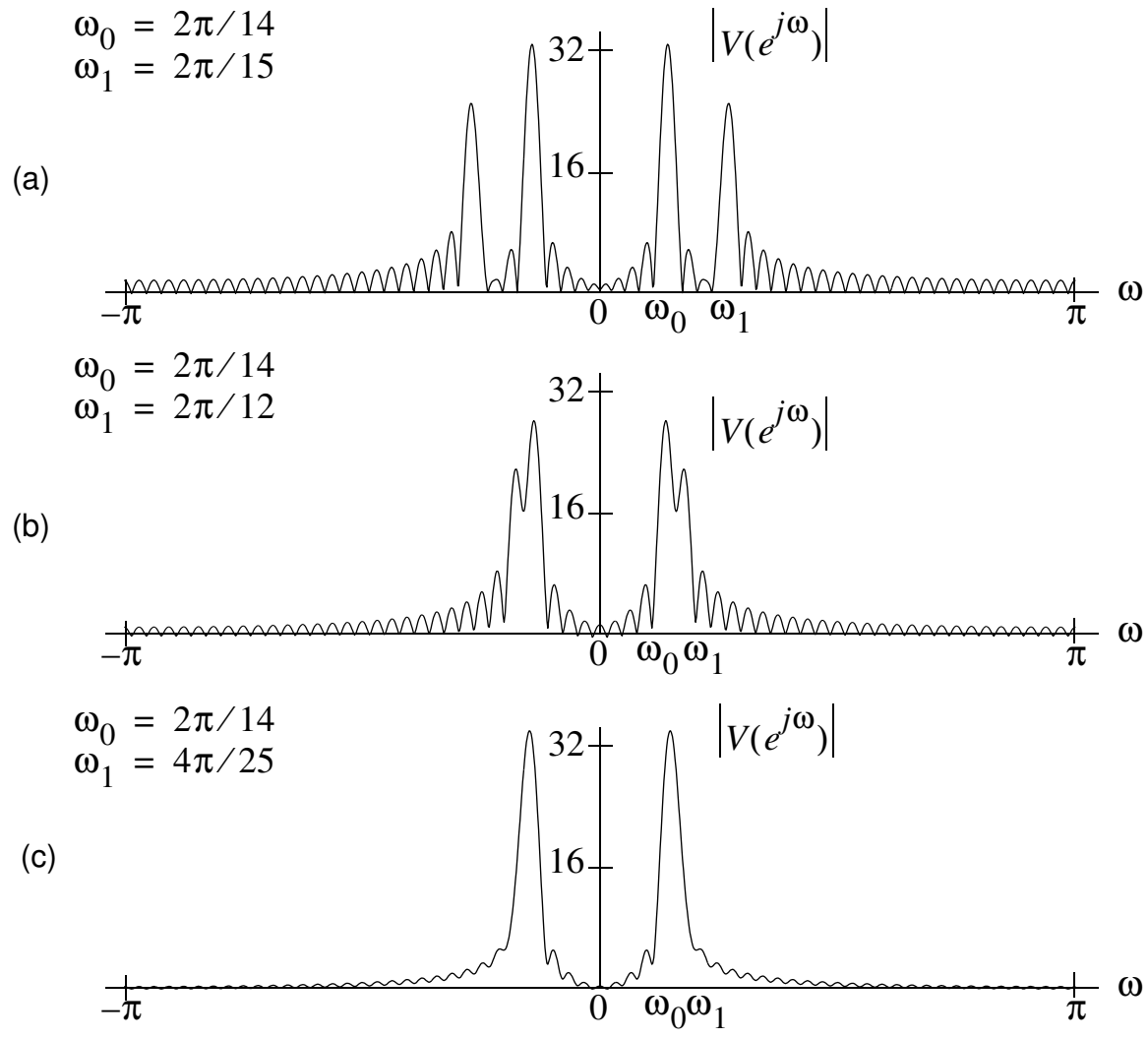
For the case of two sinusoids suppose that $f_s = 1/T = 10$ kHz, $w[n]$ is a rectangular window of length 64 i.e.

$$w[n] = \begin{cases} 1, & 0 \le n \le 63 \\ 0, & \text{otherwise} \end{cases}$$

with $A_0 = 1$ and $A_1 = 0.75$. Choose $\theta_0 = \theta_1 = 0$. In the figures that follow $|W(e^{j\omega})|$ is shown along with $|V(e^{j\omega})|$ for different values of f_0 (Ω_0) and f_1 (Ω_1).



(a) $|W(e^{j\omega})|$ for $N = 64$; (b) $|V(e^{j\omega})|$ for $f_0 = 5/3 = 1.667$ kHz and $f_1 = 10/3 = 3.333$ kHz.



(a) $|V(e^{j\omega})|$ for $f_0 = 5/7(0.714)$ kHz and $f_1 = 4/3 = 1.333$ kHz; (b) $|V(e^{j\omega})|$ for $f_0 = 5/7 = 0.714$ kHz and $f_1 = 5/3 = 0.833$ kHz; (c) $|V(e^{j\omega})|$ for $f_0 = 5/7 = 0.714$ kHz and $f_1 = 4/5 = 0.8$ kHz.

- The rectangular window has Fourier transform

$$W_r(e^{j\omega}) = \sum_{n=0}^{L-1} e^{-j\omega n} = e^{-j\omega(L-1)/2}\frac{\sin(\omega L/2)}{\sin(\omega/2)}$$

- Since $W(1) = L = 64$ we see that the peak spectral amplitudes in part (b) of the above figure are $32A_0 = 32$ and $32A_1 = 24$

- Notice that as the frequency difference decreases (b) $\rightarrow$ (e) and the leakage of one sinusoid into the other becomes more pronounced

- In particular in (d) we see that the sidelobes add out of phase to reduce the height of the spectral peaks

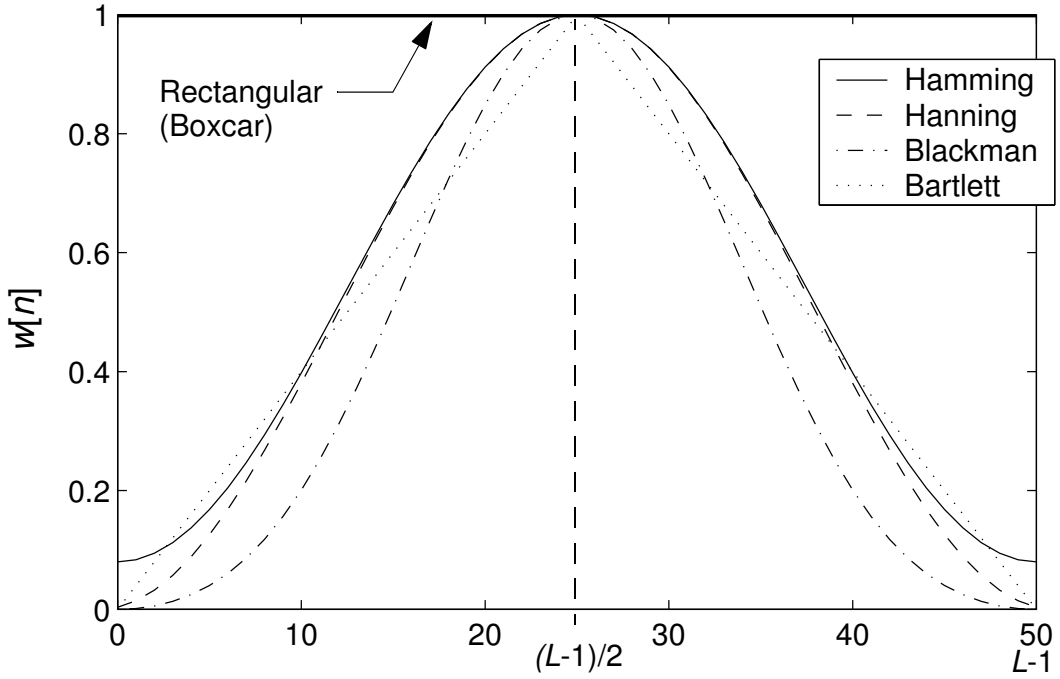
- In (e) the overlap of sidelobes is so great that the two peaks have become one. Of all possible window functions the rectangular window has the narrowest mainlobe for a given length, thus to resolve the two frequencies in this case will require a longer window.

- To reduce the spectral leakage with the penalty of decreased spectral resolution (wider mainlobe) nonrectangular window functions may be employed

A Short Catalog of Window Functions

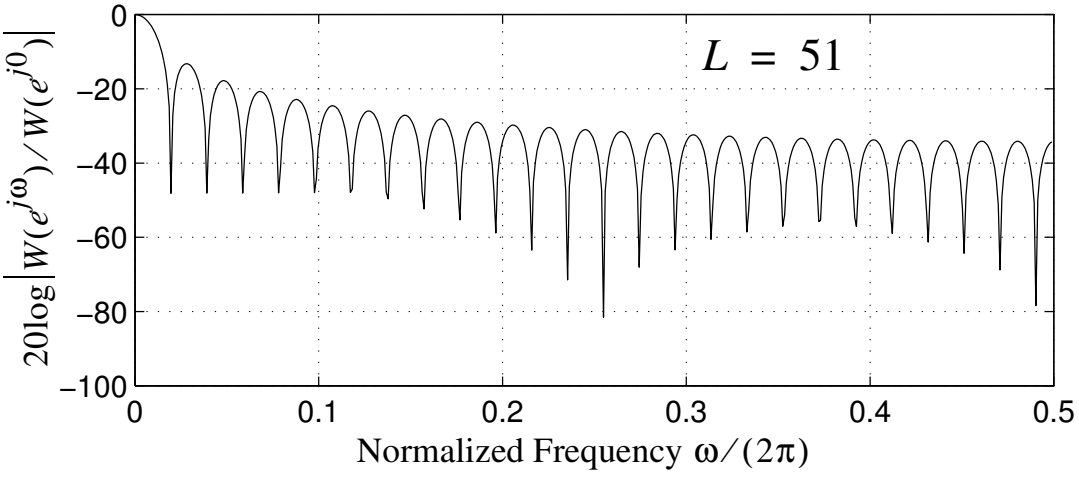
The rectangular window, which is inherent in the FFT (DFT), has a peak sidelobe down only 13 dB from the mainlobe and a rolloff rate of only 6 dB per octave. Here the ability of the windowed transform to resolve a weak signal close to a strong signal, is limited in dynamic range depending upon the relative frequency spacing. Nonrectangular window functions offer reduced leakage (increased dynamic range) at the expense of a wider mainlobe. With a wider mainlobe the ability to resolve closely spaced near equal amplitude

sinusoids is reduced. The designer is thus faced with a resolution versus leakage (dynamic range) tradeoff.



Rectangular

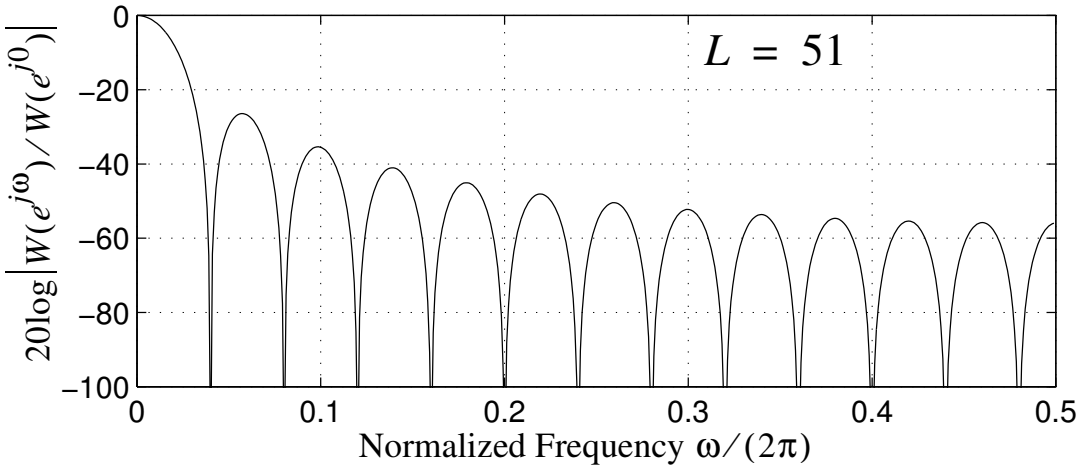
$$w[n] = \begin{cases} 1, & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$$



Rectangular window frequency response for $L = 51$

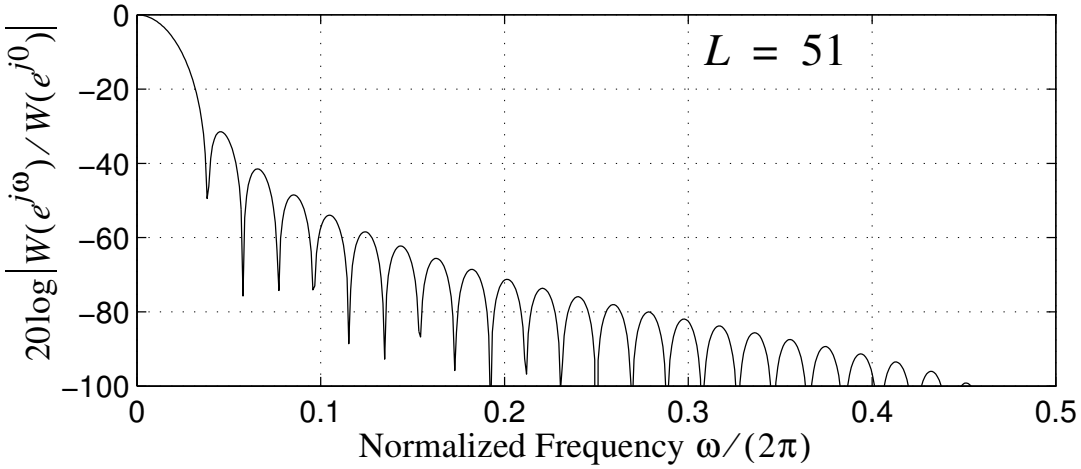
Bartlett (triangular)

$$w[n] = \begin{cases} 2n/(L-1), & 0 \le n \le (L-1)/2 \\ 2 - 2n/(L-1), & (L-1)/2 < n \le L-1 \\ 0, & \text{otherwise} \end{cases}$$



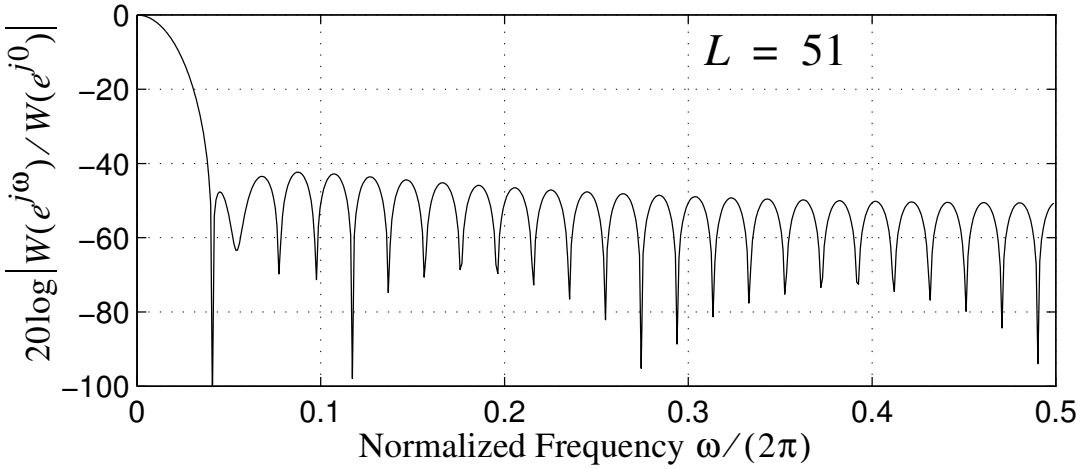
Hanning

$$w[n] = \begin{cases} 0.5\{1 - \cos[2\pi n/(L-1)]\}, & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$$



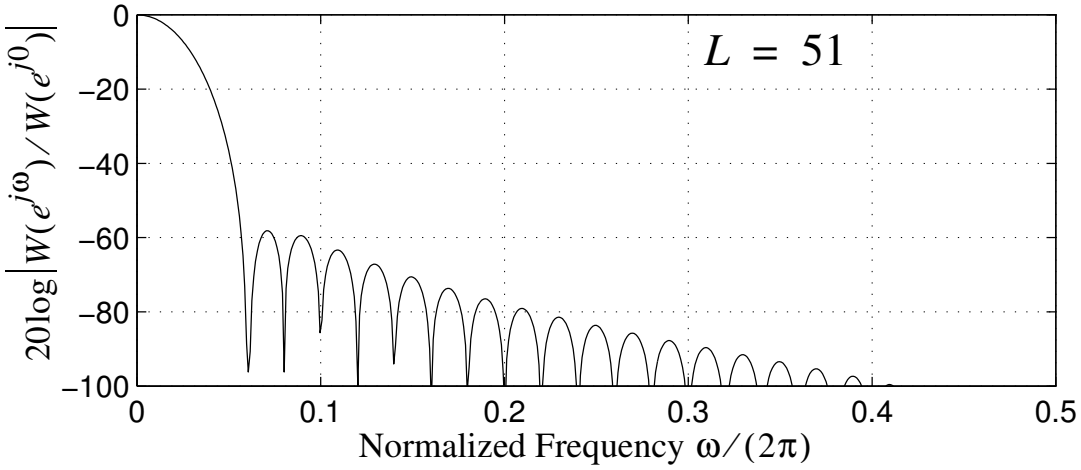
Hamming

$$w[n] = \begin{cases} 0.54 - 0.46\cos[2\pi n/(L-1)], & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$$



Blackman

$$w[n] = \begin{cases} 0.42 - 0.5\cos[2\pi n/(L-1)] \\ +0.08\cos[4\pi n/(L-1)], & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$$



Kaiser (Kaiser-Bessel)

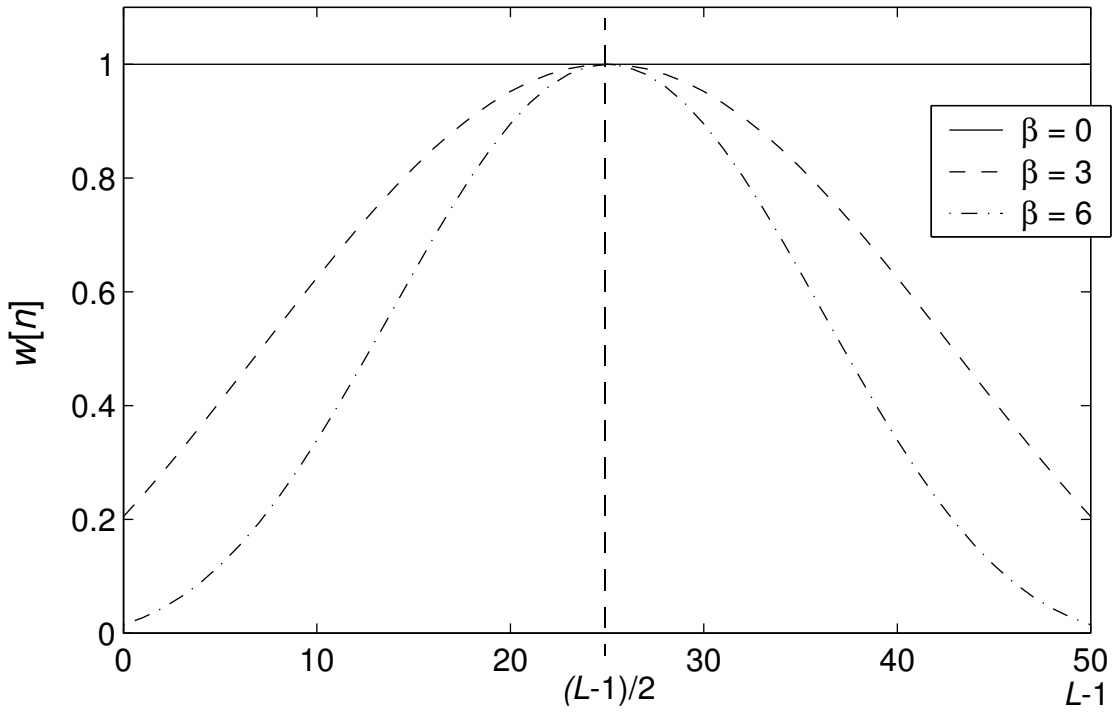
Window design in general involves a tradeoff between mainlobe width and sidelobe area or attenuation. The Kaiser window is designed to concentrate nearly all of its energy about $f = 0$ in the frequency domain

$$w[n] = \begin{cases} \frac{I_0\{\beta(1-[(n-\alpha)/\alpha]^2)^{L/2}\}}{I_0(\beta)}, & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$$

where $\alpha = (L-1)/2$ and $I_0(\cdot)$ is the zeroth-order modified Bessel function of the first kind.

- Note:

$$I_0(x) = \sum_{k=0}^{\infty} \left[\frac{(x/2)^k}{k!} \right]^2$$



Kaiser windows for $L = 51$ and $\beta = 0$, 3, and 6

- The two parameters, L and β, are used to trade between mainlobe width and sidelobe relative amplitude

- Note that when $\beta = 0$ the Kaiser window becomes a rectangular window

- Let Δ_{ml} be the null-to-null bandwidth of the mainlobe, and A_{sl} be the maximum sidelobe level in dB with respect to the mainlobe amplitude

- Sidelobe level is approximately independent of window length

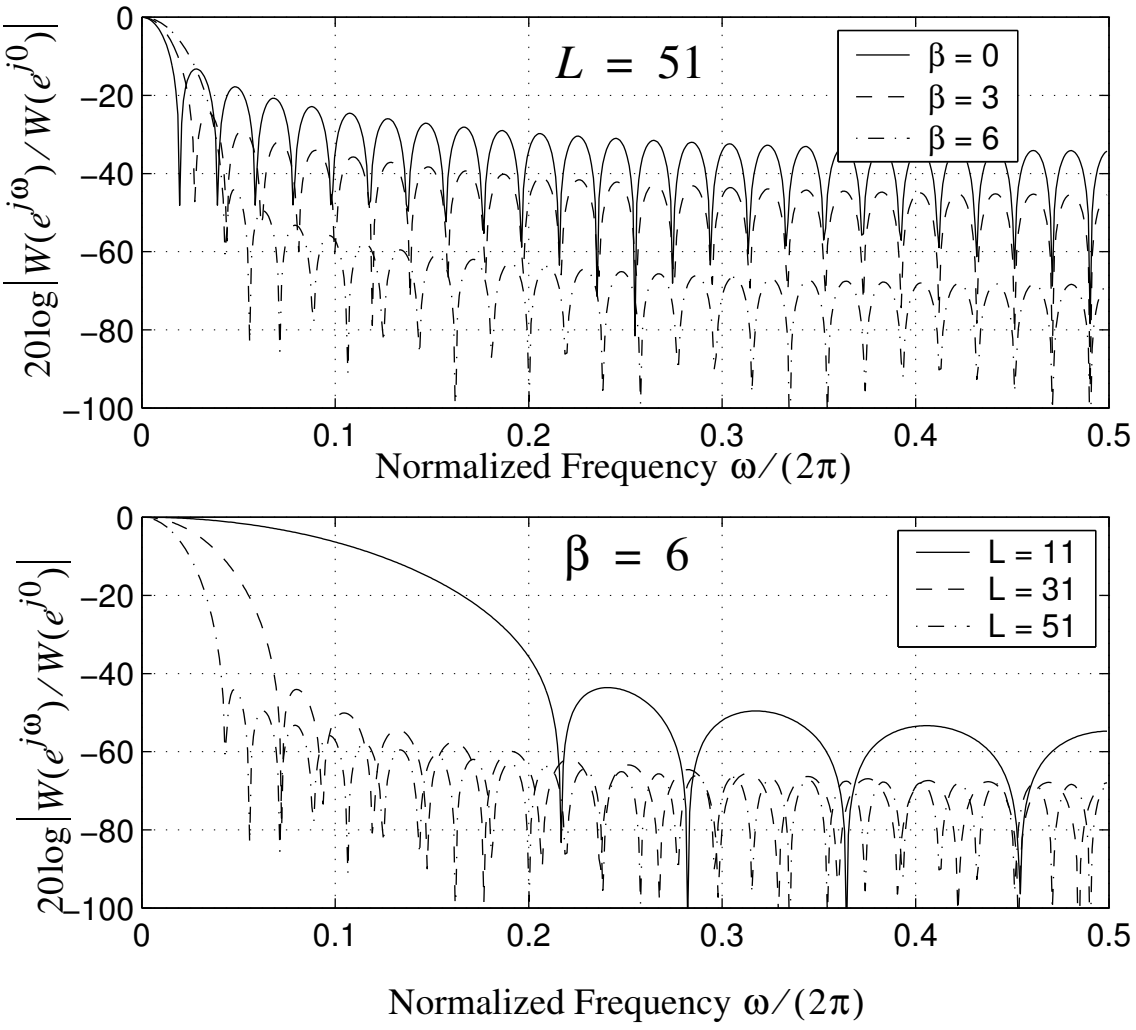
- A design formula[1] for β, accurate to within 0.36% for $13.26 < A_{sl} < 120$, is

$$\beta = \begin{cases} 0, & A_{sl} < 13.26 \\ .76609(A_{sl} - 13.26)^{0.4} + .09834(A_{sl} - 13.26), & 13.26 < A_{sl} < 60 \\ .12438(A_{sl} + 6.3), & 60 < A_{sl} < 120 \end{cases}$$

- An approximate formula which relates L, A_{sl}, and Δ_{ml} is[1]

$$L \simeq \frac{24\pi(A_{sl} + 12)}{155\Delta_{ml}} + 1$$

[1] J. F. Kaiser and R.W. Schafer, "On the Use of the I_0-Sinh Window for Spectrum Analysis,"*IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-28, No. 1, pp. 105-107, Feb. 1980.



Kaiser window frequency response, (top) $L = 51$ and $\beta =$ 0, 3, and 6; (bottom) $\beta = 6$ and $L =$ 11, 31, 51.

Window Comparisons[2]

Window Type	6 dB point (bins)	Full width (bins) of central peak	Highest sidelobe (dB)	Rate of sidelobe fall-off (db/oct.)	Equiv. noise BW. (bins)
Rect.	1.21	2.0	-13	-6	1.00
Bartlett	1.78	4.0	-27	-12	1.33
Hanning	2.00	4.0	-32	-18	1.50
Hamming	1.81	5.0(zero)	-43	-6	1.36
Blackman	2.35	7.0(zero) 6.0(-40dB)	-58	-18	1.73
Kaiser $\beta = 3.0$	2.39	7.0(-70dB)	-69	-6	1.80

Table parameter definitions:

- In an L point DFT a *bin* is equal to the smallest frequency sample spacing i.e. $\Delta\Omega = \Omega_s/L$ or $\Delta f = f_s/L$
- The 6 dB point is the bandwidth in bins from the mainlobe center (peak) to where it has dropped in amplitude 6 dB e.g. for a 6 dB point of 1.21, $L = 256$, and $f_s = 10$ MHz, then the 6 dB point is at 47.266 kHz
- The equivalent noise bandwidth (ENBW) is the two-sided bandwidth in bins of a rectangular filter with power gain $|W(1)|^2$ that passes the same noise power as the window magnitude squared frequency response. In Elliot and Rao[3] it is shown

[2]F.J. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proc. IEEE*, Vol. 66, pp. 51-83, Jan. 1978.

[3]D.F. Elliot and K.R. Rao, *Fast Transforms Algorithms, Analysis, Applications*, Academic Press, Inc., 1982.

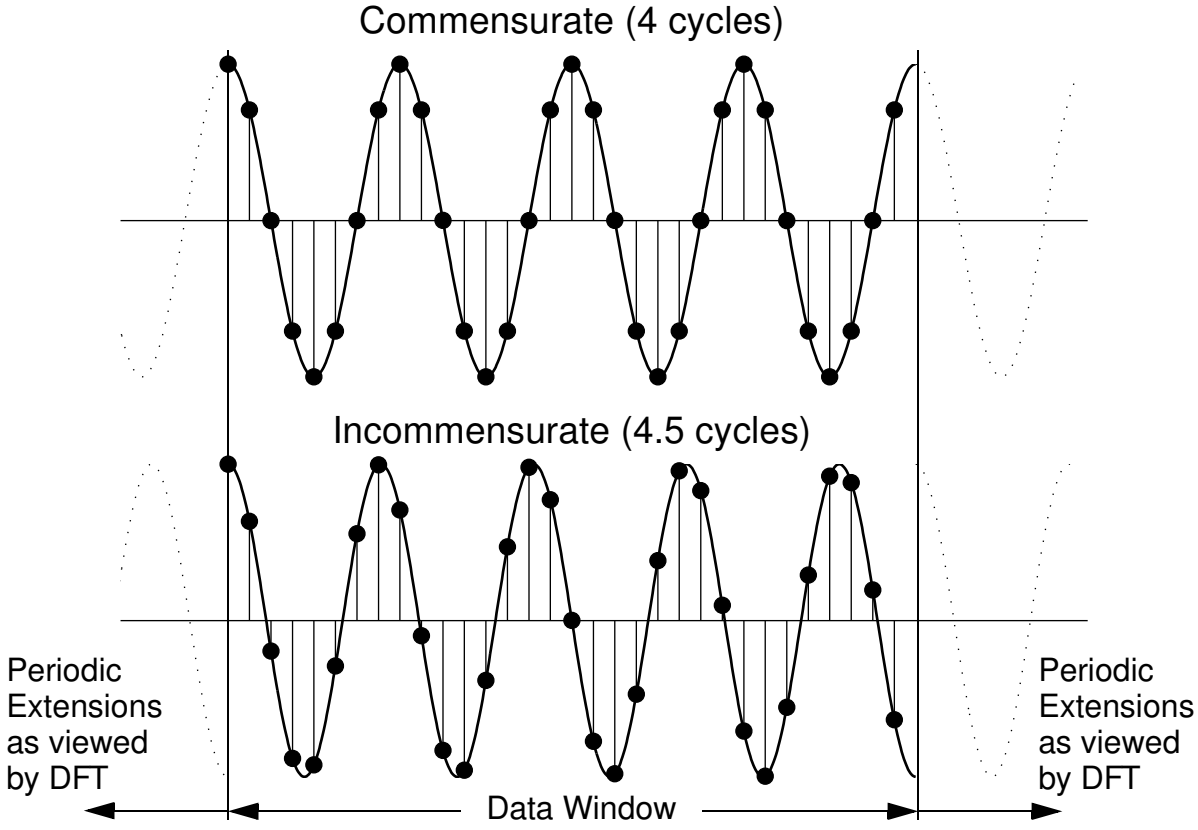
that

$$\text{ENBW} = \frac{L \sum_{n=0}^{L-1} w^2[n]}{\left[\sum_{n=0}^{L-1} w[n]\right]^2}$$

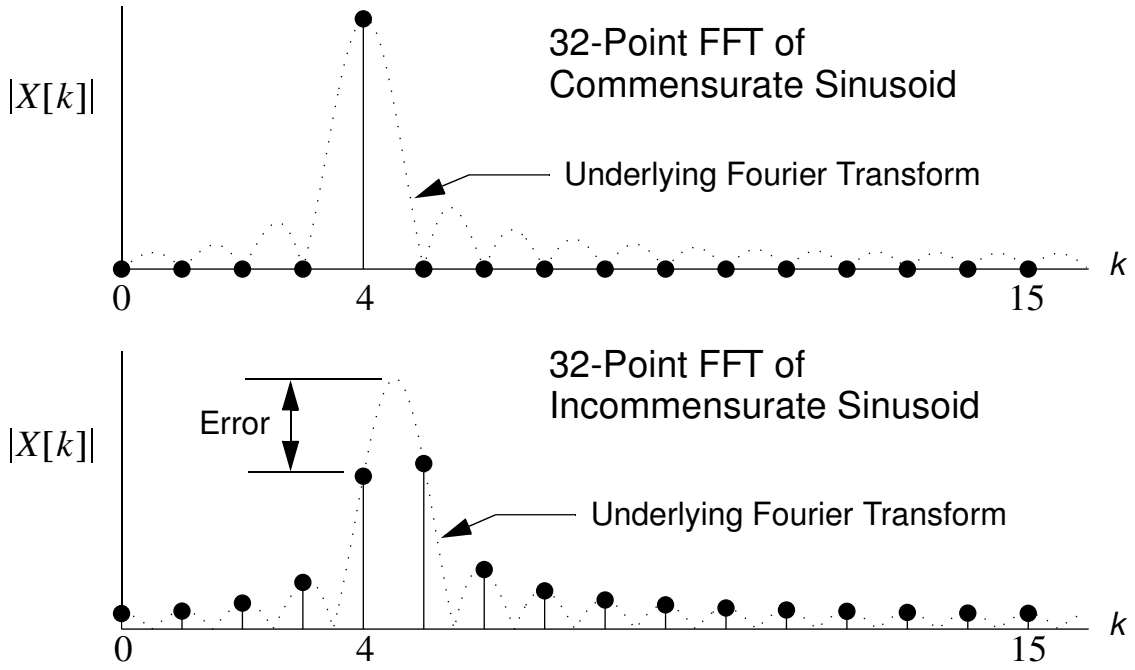
The Effect of Spectral Sampling – Zero Padding

The spectral sampling performed by the FFT (DFT) results in reduced signal amplitudes for sinusoid frequencies off the 'bin centers'. Note that the bin centers are the FFT sample frequencies.

- To insure that the sinusoid frequency lies at a bin center an integer number of signal cycles must be contained in the window. The sinusoid is then said to be *commensurate* with the window.

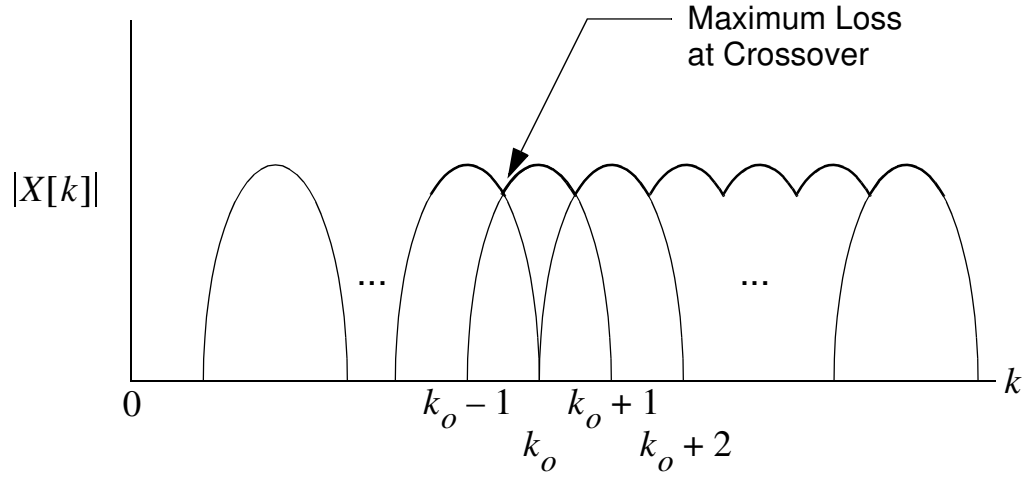


Examples of commensurate and incommensurate signals



DFT spectra: commensurate & incommensurate sinusoids

- The maximum loss (amplitude error) occurs when the sinusoid frequency is halfway between two bins e.g. $f = f_s/2 + \Delta f$
- The amplitude loss is referred to as *scalloping loss* or the *picket-fence effect* since the spectrum is actually viewed through a picket fence of impulses



- The scalloping loss is a function of the window type

- If we assume that the sinusoid frequency is equally likely to lie between bin centers, then the average scalloping loss is the appropriate performance parameter

Example 7.8: 64-Point Signal with Zero-Padding

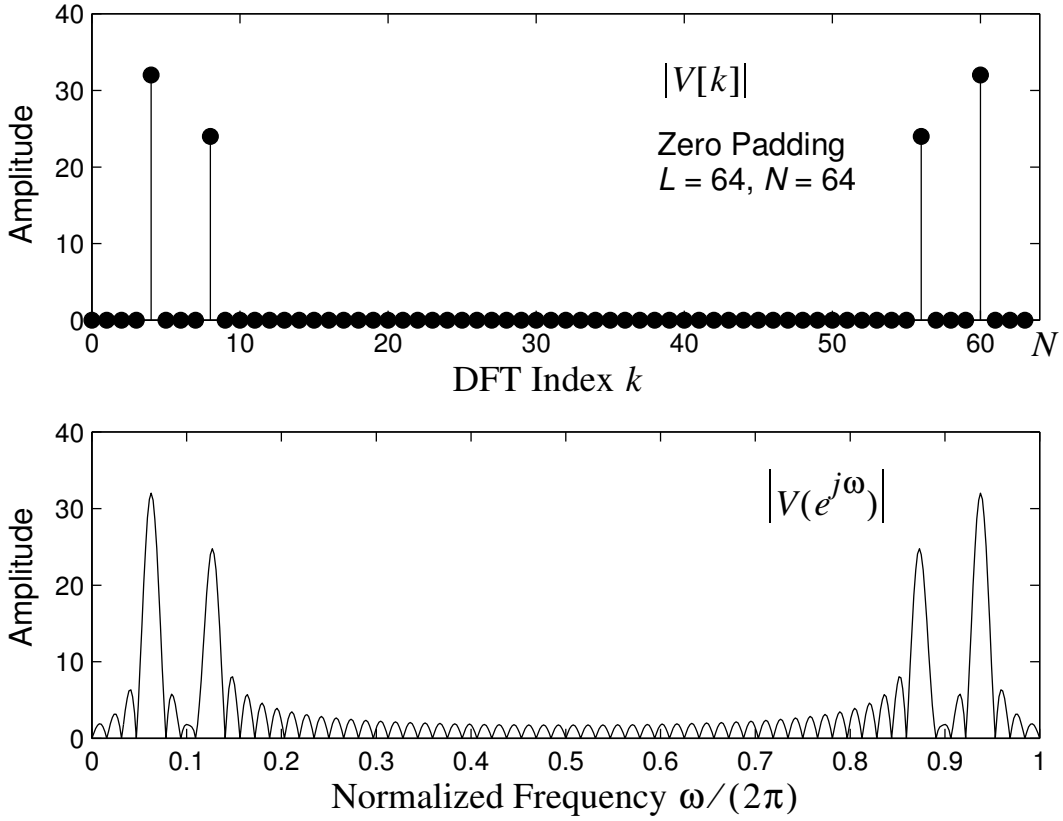
Consider the $N = 64$-point FFT of the sequence

$$v[n] = \begin{cases} \cos\left(\frac{2\pi}{16}n\right) + 0.75\cos\left(\frac{2\pi}{8}n\right), & 0 \leq n \leq 63 \\ 0, & \text{otherwise} \end{cases}$$

- Implicitly a rectangular window of length $L = 64$ is being used. The DFT bin frequencies are $\omega_k = 2\pi k/N = 2\pi k/64$, $0 \leq k \leq 63$.

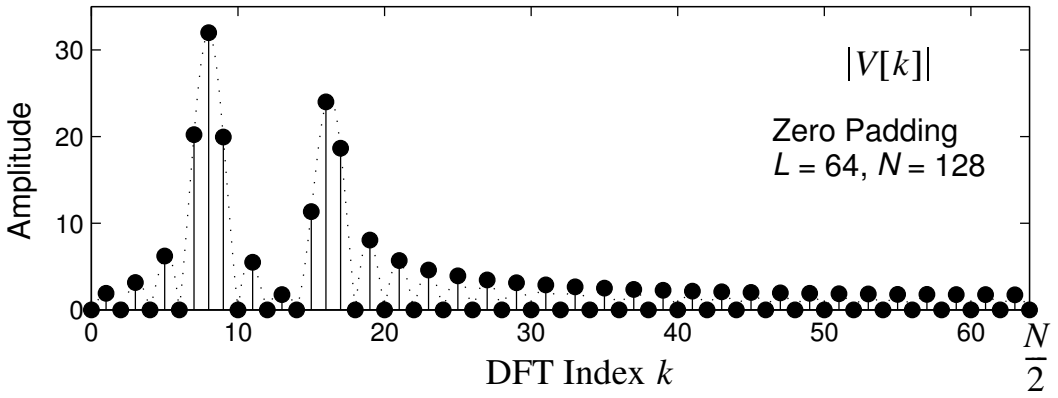
- Both sinusoids are commensurate with the window since

$$\omega_0 = \frac{2\pi}{16} = 4\frac{2\pi}{64}$$
$$\omega_1 = \frac{2\pi}{8} = 8\frac{2\pi}{64}$$



Spectral analysis of two sinusoids. (a) 64-point DFT magnitude $|V[k]|$; (b) Fourier transform magnitude $|V(e^{j\omega})|$.

- From the DFT plot $|V[k]|$ the two spectral lines standout very clearly, with no apparent leakage
- Suppose $v[n]$ is zero padded to 128 points and an $N = 128$-point DFT is then calculated



- With twice as many points sampling the Fourier transform of $v[n]$ the spectral leakage is now apparent
- Zero padding **does not** improve the resolution, it only allows the DFT to evaluate the Fourier transform over more closely spaced frequencies
- With zero padding Δf is decreased, hence problems associated with the picket fence effect are reduced
- To obtain a true increase in resolution for a given window function, the window length **must** be increased

Example 7.9: Three Sinusoids with 80 dB Dynamic Range

In this example we use MATLAB to investigate the windowed spectrum of three sinusoids. The sampling rate is chosen to be 10 kHz. The input signal model is

$$s_c(t) = 10\cos[2\pi(1000)t] + 10\cos[2\pi(1100)t] + 0.001\cos[2\pi(3000)t]$$

The first two sinusoids are at 1 kHz and 1.1 kHz respectively. The third sinusoid is at 3 kHz and amplitude attenuated by 80 dB with respect to the first two sinusoids.

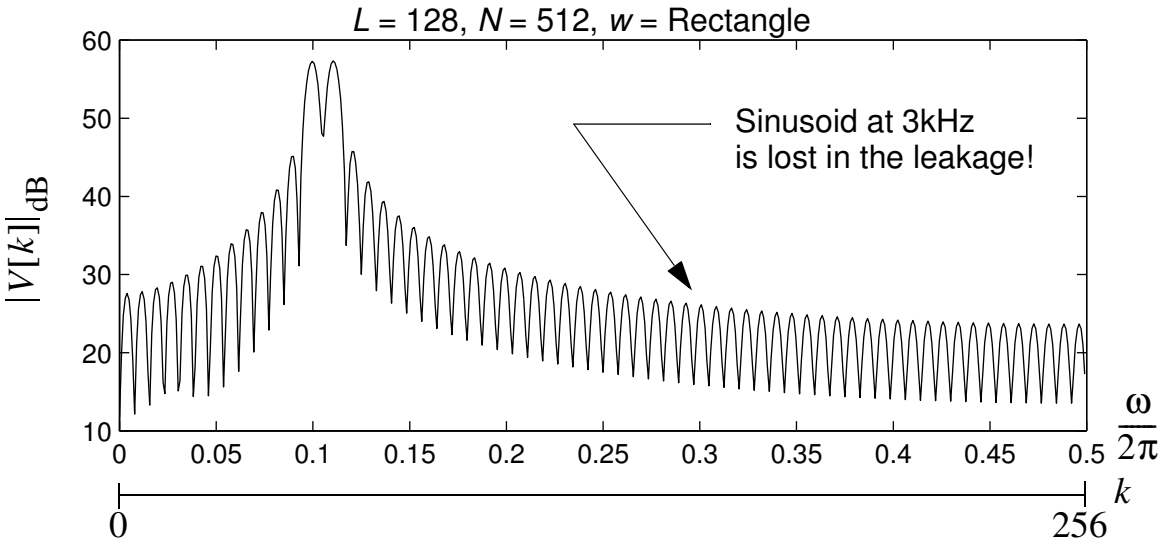
Spectrum analysis parameters:

- An initial window length of $L = 128$
- The FFT length is fixed at $N = 512$ (zero padding with 512 - 128 points)

  - In MATLAB we zero pad an FFT by including a second parameter, e.g.,

```
» X = fft(x); % No zero padding just length(x) FFT
» X = fft(x,N); % Zero padded N > length(x) or truncated FFT
```

- The windows chosen are the rectangular and Hanning functions



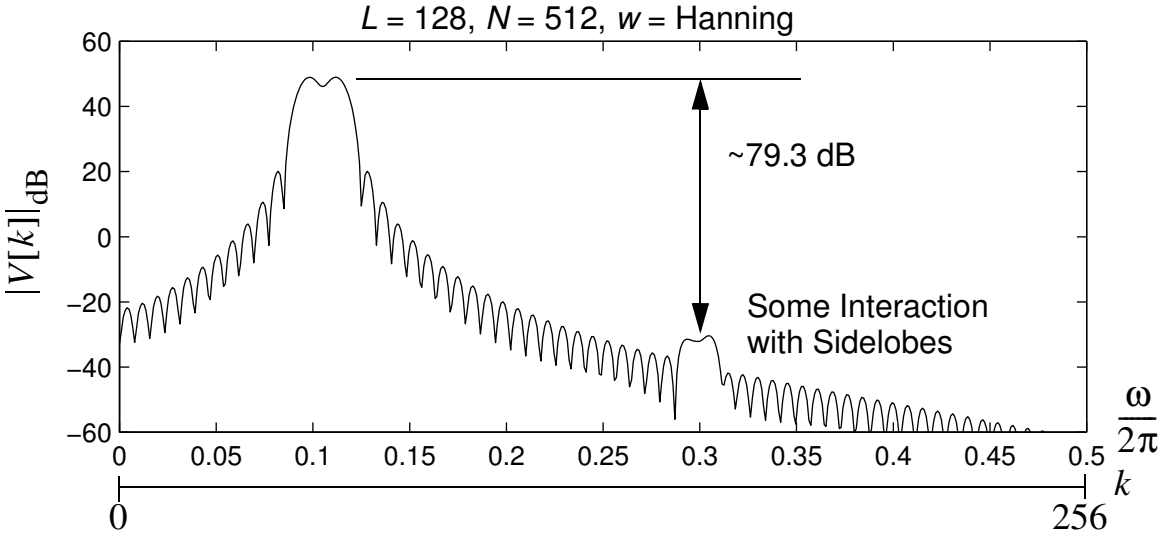
Zero-padded spectrum with the rectangular window with $L = 128$ and $N = 512$

Observations:

- When using the rectangular window the closely spaced sinusoids are well resolved, but the 80 dB down 3 kHz tone is masked over by the spectral leakage of the stronger tones

- When we use the Hanning window the closely spaced sinusoids are just barely resolvable, but the -18 dB/octave roll-off of the sidelobes allows the weak tone to appear

- Note there is still some interference from the sidelobes of the strong tones



Zero-padded spectrum with the Hanning window with $L = 128$ and $N = 512$

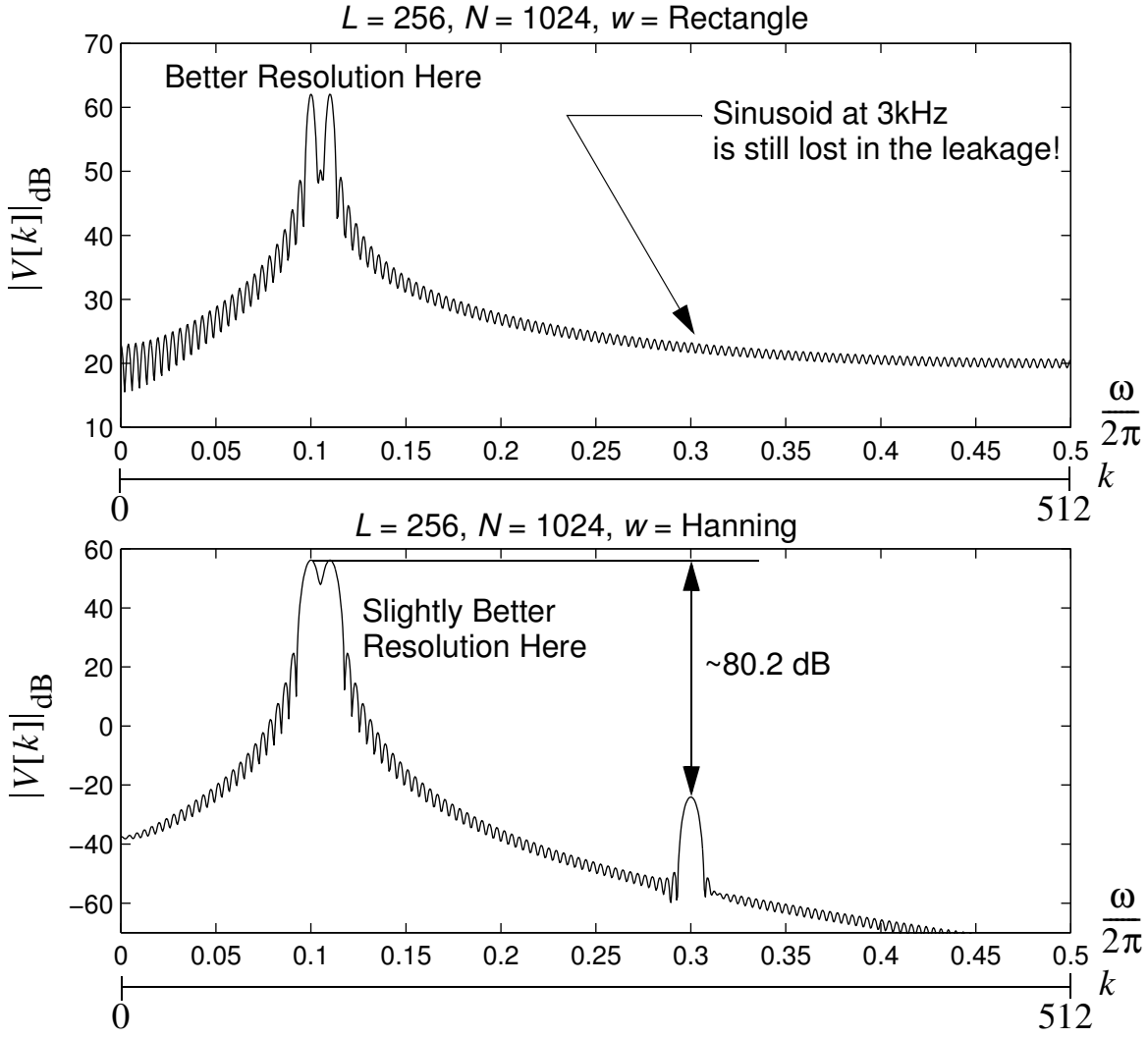
- This example clearly illustrates the resolution versus leakage trade-off involved in window selection

- For a fixed length window we get the best resolution with rectangular window, but the dynamic range is severely limited by the small -6 dB/octave sidelobe roll-off rate and the associated spectral leakage

- Switching to a faster roll-off rate window increases the dynamic range, but the wider main lobe blurs closely spaced tones

- Suppose we can collect $L = 256$ signals values and zero pad up to $N = 1024$-points

- The rectangular and Hanning windows produce the following spectral results



Zero-padded spectrum with the rectangular and Hanning windows when $L = 256$ and $N = 1024$

.