

# **ELEC 421**

## **Digital Signal and Image Processing**



**Siamak Najarian, Ph.D., P.Eng.,**  
Professor of Biomedical Engineering (retired),  
Electrical and Computer Engineering Department,  
University of British Columbia

## Course Roadmap for DSP

Lecture	Title
Lecture 0	Introduction to DSP and DIP
Lecture 1	Signals
Lecture 2	Linear Time-Invariant System
Lecture 3	Convolution and its Properties
Lecture 4	The Fourier Series
Lecture 5	The Fourier Transform
Lecture 6	Frequency Response
Lecture 7	Discrete-Time Fourier Transform
Lecture 8	Introduction to the z-Transform
Lecture 9	Inverse z-Transform; Poles and Zeros
Lecture 10	The Discrete Fourier Transform
Lecture 11	Radix-2 Fast Fourier Transforms
Lecture 12	The Cooley-Tukey and Good-Thomas FFTs
Lecture 13	The Sampling Theorem
Lecture 14	Continuous-Time Filtering with Digital Systems; Upsampling and Downsampling
Lecture 15	MATLAB Implementation of Filter Design

# Lecture 10: The Discrete Fourier Transform

## Table of Contents

---

- Review of the 4 Fourier transforms
- The DFT's place
- Recall the Fourier Series
- Discrete-time exponentials are periodic
- Definitions: the DFT and inverse DFT
- The  $W_N$  notation
- Thinking of the DFT as a change of coordinates
- Writing the DFT as a matrix-vector product
- The Fourier matrix  $F$
- How are the DTFT and DFT related?
- The DFT samples the DTFT at equally spaced frequencies
- Examples of computing the DFT
- DFT of delta function
- DFT of a constant
- The orthogonality principle
- A pulse: the DTFT vs. the DFT
- MATLAB demonstration of how the DFT samples the DTFT
- DFT properties
- Cyclic convolution
- Representing cyclic convolution as a matrix-vector product
- Representing normal convolution as a matrix-vector product
- Computing normal convolution as cyclic convolution with zero-padding
- Block diagram for zero padding

## Review of the 4 Fourier transforms; The DFT's place

So FAR

	CONTINUOUS	DISCRETE
PERIODIC	FS	DFT (DISCRETE FOURIER TRANSFORM) * DTFS (DT FOURIER SERIES) FFT (FAST FOURIER TRANSFORM)
APERIODIC	FT	DTFT

- The discrete-time Fourier transform (DTFT) is one of the most critical concepts for understanding what is really going on inside MATLAB or inside a circuit that is doing **Fourier Analysis**.
- We drew this table before and now we are adding more to it. In the columns, we have a continuous-time signal and a discrete-time signal. In the rows, we have periodic vs. non-periodic (aperiodic).
- For a continuous-time periodic signal, we can take the Fourier series (FS). For a continuous-time non-periodic signal, we can take the Fourier transform (FT). For a discrete-time non-periodic signal, we can take the discrete-time Fourier transform (DTFT).
- The topic of today's lecture is what goes into the box of discrete-time periodic signals (\*), which is variously called the **DFT** in the case of finite-length digital signals (the discrete FT). In (\*), we can also put the series representation of a periodic discrete-time signal, the discrete-time Fourier series (**DTFS**). Fast Fourier transform (**FFT**) can also be placed in (\*). The result of DFT and FFT is the same, but FFT is **customized** and **optimized** to make the DFT process much more efficient.

## Recall the Fourier Series

INTUITION: THE FS TAKES A CONTINUOUS, PERIODIC SIGNAL AND REPRESENTS IT AS A SUM OF (COMPLEX) SINES/COSINES:

$$x(t) = \sum_{k=-\infty}^{\infty} \underbrace{X[k]}_{a_k} e^{jk \frac{2\pi}{T} t} \quad (1) \quad t \in [0, T]$$

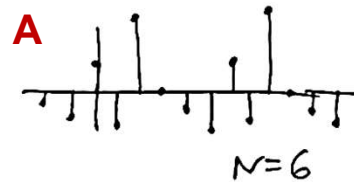
$\nwarrow \omega_0$

- **What is the intuition here?** To make an analogy to the Fourier series, the intuition is that the Fourier series takes a continuous periodic signal and represents it as a sum of complex sines and cosines. And what that means is that in our formula for the Fourier series, **(1)**, we take the periodic signal and we write it like the sum of a bunch of coefficients. Formerly, we used  $a_k$ 's but just to make an analogy, let us use  $X[k]$  for the coefficients.
- In **(1)**,  $\omega_0$  was the fundamental frequency and  $t$  was some number between **0** and the period of the signal, **T**. Since this signal may oscillate infinitely fast, we may need infinitely many of these  $a_k$ 's to make the sum of this approximation look close to the original signal.
- The intuition is what if we want to do the same kind of thing in discrete time? So, in this lecture, we are only going to be considering **periodic discrete-time signals**.

## Recall the Fourier Series

INTUITION: THE FS TAKES A CONTINUOUS, PERIODIC SIGNAL AND REPRESENTS IT AS A SUM OF (COMPLEX) SINES/COSINES:

$$x(t) = \sum_{k=-\infty}^{\infty} \frac{X[k]}{a_k} e^{jk \left( \frac{2\pi}{T} \right) t} \quad (1) \quad t \in [0, T]$$



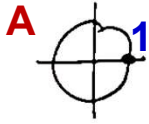
COULD WE WRITE?

$$x[n] = \sum_{k=-\infty}^{\infty} X[k] e^{jk \frac{2\pi}{N} n} \quad (2) \quad n = 0, 1, \dots, N-1$$

- **What do we mean by periodic discrete-time signal?** One example is shown in graph **A**. Here, we have a discrete signal that is going to repeat every once in a while.
- **How would we make the analogy with  $x(t)$ ?** Remember that there are only a fixed number of discrete-time sinusoids and we can only make it oscillate between **+1** and **-1** (or  **$-\pi$**  and  **$+\pi$** ). Also, the highest frequency is  **$\pi$** . The analogy is that maybe we can get away with writing an equation like **(2)**. This equation is not the actual DFT, but a good starting point.
- One of the reasons that **(2)** is not quite right is that our periodic signal,  **$x[n]$** , going from  **$n = 0$**  to  **$n = N-1$**  only has  **$N$**  unique values, but we are using an infinite number of sines and cosines to represent it (due to  **$k = -\infty$**  to  **$+\infty$** ). That does not seem quite right. We can get around that by taking a closer look at the exponential term,  **$\exp(jk2\pi n/N)$** .

## Discrete-time exponentials are periodic

$$\begin{aligned}
 e^{jk\frac{2\pi}{N}(n+N)} &= e^{jk\frac{2\pi}{N}n} \overbrace{e^{jk2\pi}}^{(*)} \\
 &= e^{jk\frac{2\pi}{N}n}
 \end{aligned}$$



THERE ARE ONLY  $N$  UNIQUE COMPLEX  
EXPONENTIALS OF PERIOD  $N$ .

- If we put in an  $N$  in  $(jk2\pi/N)(n+N)$ , i.e., we increment  $n$  by  $N$ , we get **(1)**. In term **(\*)**, since  $k2\pi$  is an integer, no matter what  $k$  is, we are always at the same point “1” on the unit circle, shown in **A**. What that means is that there are only  $N$  unique complex exponentials of period  $N$ , and that is good. This means that if we have  $N$  numbers coming in as our input signal, we should have some coefficients on  $N$  numbers coming out, which are the coefficients on these  $N$  unique exponentials.



## Definitions: the DFT and inverse DFT; The $W_N$ notation

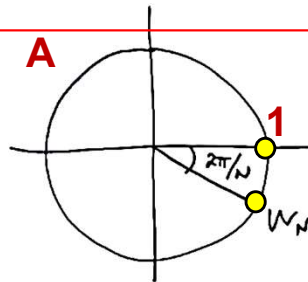
DEFINE THE DISCRETE FOURIER TRANSFORM.  
(DFT) AS:

$$\text{DFT: } X[k] = \sum_{n=0}^{N-1} x[n] e^{-jk\frac{2\pi}{N}n} \quad k=0,1,\dots,N-1 \quad (1)$$

$$\text{IDFT: } x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{+jk\frac{2\pi}{N}n} \quad n=0,1,\dots,N-1 \quad (2)$$

$$(3) \quad W_N = e^{-j\frac{2\pi}{N}}$$

$$(4) \quad (W_N)^N = 1$$



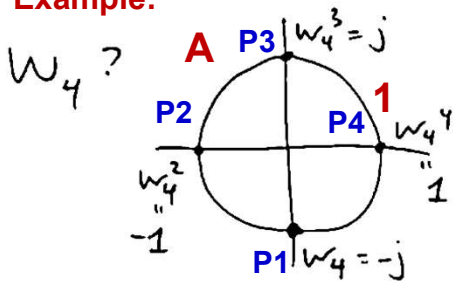
- Let us now write the actual definitions. Equation (1) is the **discrete Fourier transform (DFT)**. Here,  $x[n]$  is the input signal and  $X[k]$  is the **DFT**. This is the **forward way** of doing it. If we want to reconstruct the original signal,  $x[n]$ , from the DFT, we use (2), i.e., the **inverse DFT**. Note that in both (1) and (2),  $n$  and  $k$  both ranges from 0 to  $N-1$ . Equation (2) gives us the **reconstructed** form of the original signal,  $x[n]$ .
- These equations tell us how to go from  $N$  number of  $x[n]$  to  $N$  number of  $X[k]$ , and vice versa. The factors of  $\exp[(-jk2\pi/N)n]$  depend on **the length of the signal,  $N$** . And, we are going to define a special number,  $W_N$ , to make all this notation easier (3).
- What is  $W_N$ ?**  $W_N$  is a number that is somewhere on the unit circle (shown in A), where the angle is  $2\pi/N$ . This is going to be important because we can show that  $W_N$  is what we would call **an  $N$ th root of 1**, shown in (4). This is because if we take  $W_N$  to the  $N$  power, we get  $[\exp(-j2\pi/N)]^N$ , which brings us back to 1 on the unit circle.

## The $W_N$ notation

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad (1) \text{ Forward}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \quad (2) \text{ Backward}$$

Example:



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

$$(W_N)^N = 1$$

$$(W_N)^N = 1 \xrightarrow{N=4} (W_4)^4 = 1 \xrightarrow{z=W_4} (z)^4 = 1 \rightarrow z^4 = 1$$

THE 4 roots of 1

$(z^4 = 1)$	$z$	$z^4$
$W_4$	$= e^{-j\frac{2\pi}{4}} = e^{-j\frac{\pi}{2}}$	$e^{-2\pi j}$
$W_4^2$	$= e^{-j\pi}$	$e^{-4\pi j}$
$W_4^3$	$= e^{-j\frac{3\pi}{2}}$	$e^{-6\pi j}$
$W_4^4$	$= e^{-j2\pi}$	$e^{-8\pi j}$

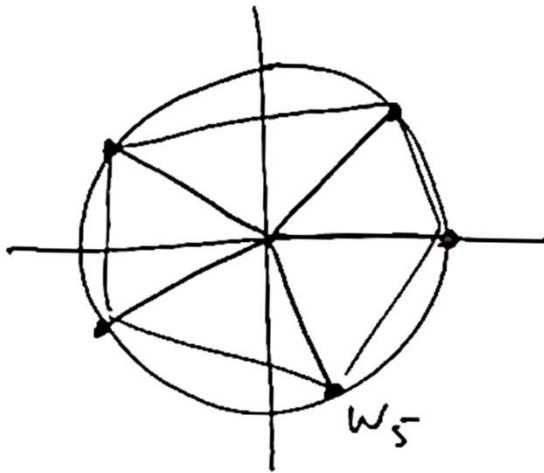
(3)

- Here, we have rewritten the same equations in terms of this special complex number,  $W_N$ . As an example, let us look at  $W_4$ .
- What is  $W_4$ ?**  $W_4$  or  $W_4^1$  is like a root of 1. It is a complex number that has a magnitude of 1. It is located at **P1**. When we square it,  $W_4^2$ , all we are doing is doubling the angle. So, we go from point **P1** to point **P2**, where the angle is doubled. When we take the 4<sup>th</sup> power of  $W_4$ ,  $W_4^4$  is the complex number that has the smallest angle and will bring us back to 1. That is, when we do  $W_4^4$ , it brings us back to 1. When we take the 4<sup>th</sup> power, there are other four roots of 1, from **P1** to **P4**. That is, we have -j, -1, +j and +1.
- There are four roots of 1 that can satisfy the equation of  $z^4 = 1$ . They are shown in (3). For example,  $W_4^2$  is one of the roots that satisfies the equation of  $z^4 = 1$ .
- Conclusion:** We can generate all of the roots of a number by looking at this special complex number,  $W_N$ .

## The $W_N$ notation

Example:

$W_5$ ?



- Let us plot  $W_5$ . We are looking for the **5<sup>th</sup>** roots of **1**. The points on the unit circle form an equal-sided pentagon. So, the **N<sup>th</sup>** roots of **1** are going to form this kind of connection. They are going to make this equal sided-polygon inside the unit circle.

# The $W_N$ notation

```
>> % Number of roots of unity (N=5)
N = 5;

% Calculate the N-th roots of unity
theta = 2 * pi * (0:N-1) / N; % Angles for roots of unity
roots_of_unity = exp(1j * theta); % Compute roots using Euler's formula

% Create a figure
figure;
hold on;
axis equal;
title('5th Roots of Unity on the Complex Plane');
xlabel('Real Axis');
ylabel('Imaginary Axis');

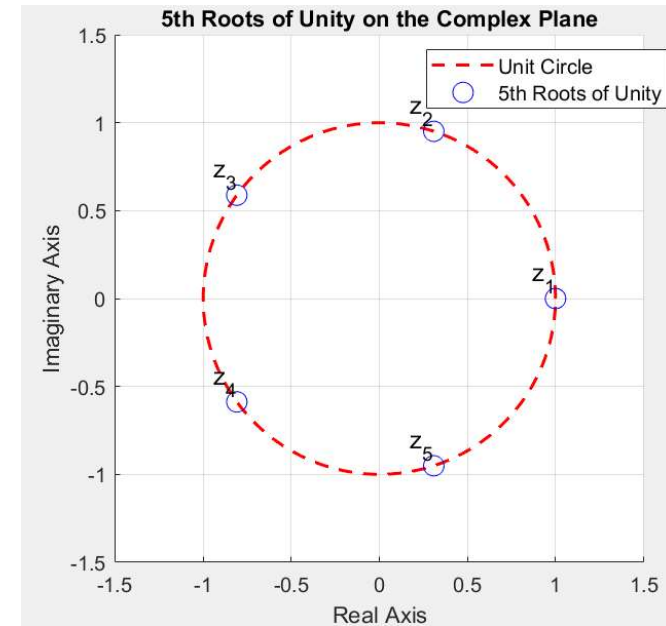
% Plot the unit circle
t = linspace(0, 2*pi, 100); % Parameter for unit circle
plot(cos(t), sin(t), 'r--', 'LineWidth', 1.5, 'DisplayName', 'Unit Circle'); % Unit circle

% Plot the roots of unity
plot(real(roots_of_unity), imag(roots_of_unity), 'bo', 'MarkerSize', 10, 'DisplayName', '5th Roots of Unity');

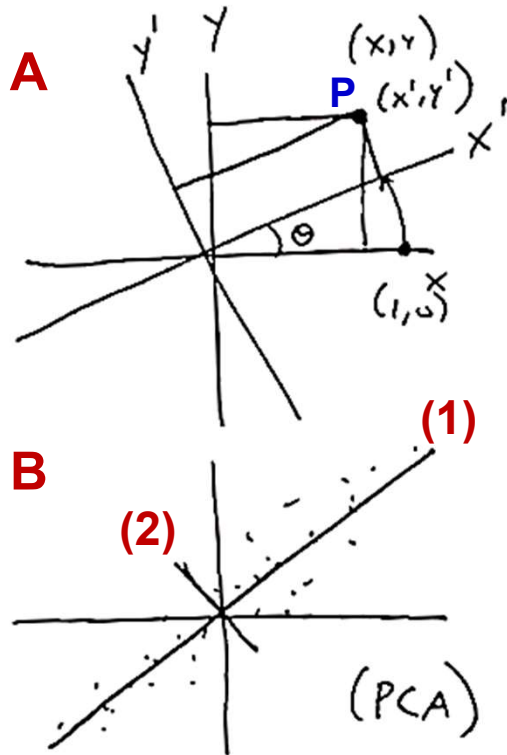
% Annotate the points as z_1 to z_5
for k = 1:N
    text(real(roots_of_unity(k)), imag(roots_of_unity(k)), sprintf('z_%d', k), ...
        'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'FontSize', 12);
end

% Set axis limits
xlim([-1.5 1.5]);
ylim([-1.5 1.5]);

% Add legend and grid
legend show;
legend('Location', 'best', 'FontSize', 10); % Set legend font size to 10
grid on;
hold off;
```

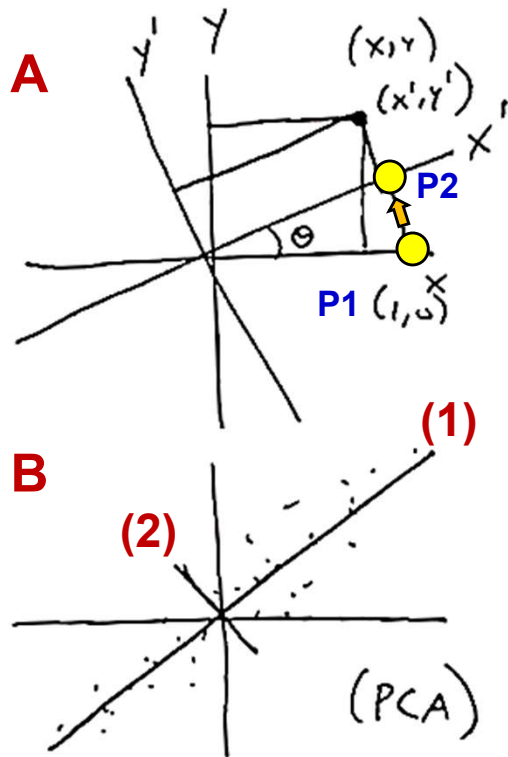


## Thinking of the DFT as a change of coordinates



- Let us briefly review an important concept in **linear algebra**. One of the most intuitive ways of thinking about transformation of **N** numbers is a **change of basis**.
- **What is a change of basis?** A normal or regular **xy**-coordinates is shown in **A**. We can always define a new set of coordinates axes (rotated by angle  $\theta$ ), such as  $x'$  and  $y'$ . Point **P** can be represented in both the regular and the rotated axes. These are equivalent ways of representing the same point in 2D space.
- We conventionally use the Cartesian coordinate system, but there are some cases where it is much easier to represent a signal in a different rotated coordinate system. For example, if we see a bunch of data in the **xy**-plane (shown in graph **B**), we might do something called **principal component analysis (PCA)**, which basically means finding the direction where there is the most variation data, such as **(1)**, and the direction where there is the least variation data, such as **(2)**. And then, we represent the signals in terms of these two new orthogonal axes.
- **Conclusion:** We might need to change the coordinates of the system in a way that is more favourable to us. And, that is exactly what is happening when we are doing the DFT. A change of coordinates is nothing more than a matrix that multiplies a vector.

## Thinking of the DFT as a change of coordinates

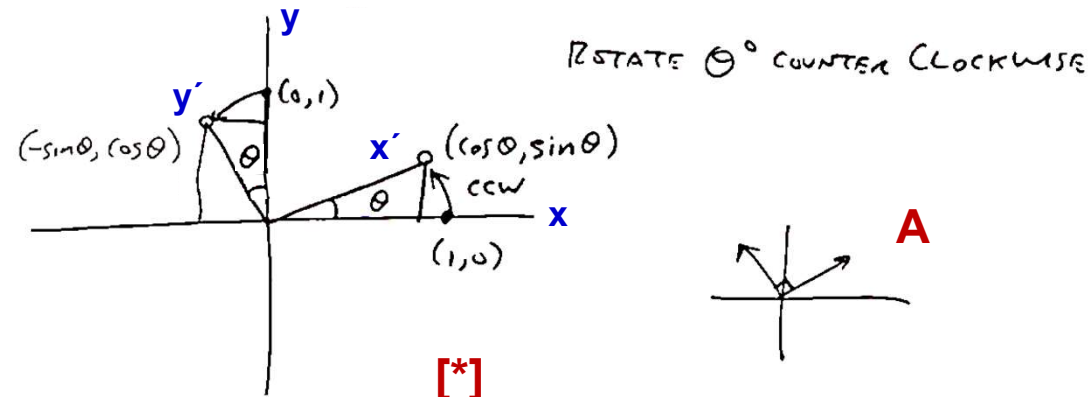


- **What would be location of a point in the new coordinate system?** The way that we would figure out this would be to use matrix operations. Equation (3) is saying that the  $x'$  and  $y'$  are going to be some matrix  $[*]$  that is multiplied by a matrix of  $x$  and  $y$  (in this case, a column vector). For example, let us see what happens to **P1** with the coordinates of **(1,0)**. So, the **old coordinate** is **(1,0)**. If we were to rotate the world (i.e., **xy**-plane), now we are looking at the same point, this time located at the new coordinate shown by **P2**.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} * \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (3)$$



## Thinking of the DFT as a change of coordinates



$$\begin{aligned}
 \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \\
 &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ c \end{bmatrix} \quad (1)
 \end{aligned}$$

$$\begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix} \quad \begin{bmatrix} -\sin \theta \\ \cos \theta \end{bmatrix}$$

unit vector

- Let us say we rotate **xy**-plane by  $\theta$  degrees in the counterclockwise direction. In equation (1), **a** and **c** in the matrix on the right-hand side are the new coordinates of the point in the **x'y'**-plane. This is a type of **geometric transformation**.
- Matrix **[\*]** is a special matrix, in the sense that, for one thing, its elements are the new vectors that give us the **x** and **y** axes of the coordinates system. These vectors have **unit length**, because  $\sqrt{(\cos \theta)^2 + (\sin \theta)^2} = 1$ .
- The other thing is if we look at the **dot product** of these unit vectors, i.e., if we compute  $(\cos \theta) \cdot (-\sin \theta) + (\sin \theta)(\cos \theta)$ , it will be **0**. The dot product being zero tells us that they are orthogonal (meaning perpendicular), as shown in (A). So, we take the old coordinates system that is right angles and turn them to the new coordinates system, which are also right angles.
- Conclusion:** It turns out that the DFT is exactly doing the above thing, except instead of two dimensions, we are doing it in **n**-dimensions, and instead of doing it in the plane, we are doing it in the complex world.

## Writing the DFT as a matrix-vector product

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

$$\begin{bmatrix} X[0] \\ X[1] \\ \vdots \\ X[N-1] \end{bmatrix} = \begin{bmatrix} W_N^0 & W_N^0 & \dots \\ W_N^0 & W_N^1 & \dots \\ W_N^0 & W_N^2 & \dots \\ \vdots & \vdots & \ddots \\ W_N^0 & W_N^{N-1} & \dots \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} \quad (2)$$

$N \times 1$                        $N \times N$                        $N \times 1$

The DFT coefficients                      The N old signal values

- Equation (1) is saying that we are taking the **N** old values of **x**, i.e., **x[n]** and we are producing the **N** new values of **x**, i.e., **X[k]**.
- How would we write equation (1) in terms of linear algebra?** This is shown in (2). So, the DFT coefficients on the left-hand side, which are the new values of **x[n]** or **X[k]**, are obtained when we multiply the matrix of **W<sub>N</sub><sup>kn</sup>** by the vector of the old values of **x**, i.e., **x[n]**. The elements of matrix **W<sub>N</sub><sup>kn</sup>** are called the **basis functions**.
- Each DFT coefficient corresponds to the strength of a specific frequency component in the signal.



## The Fourier matrix F

$$\begin{bmatrix} X[0] \\ X[1] \\ \vdots \\ X[N-1] \end{bmatrix} = \begin{bmatrix} W_N^0 & W_N^0 & \dots \\ W_N^0 & W_N^1 & \dots \\ W_N^0 & W_N^2 & \dots \\ \vdots & \vdots & \ddots \\ W_N^0 & W_N^{N-1} & \dots \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix}$$

$N \times 1$                        $N \times N$                        $N \times 1$   
 COMPLEX                      COMPLEX MATRIX

- Even though the input maybe real-valued, both  $\mathbf{X}[k]$  and  $\mathbf{W}_N^{kn}$  are, in general, a complex matrix or vector.
- A compact way to show the matrixes would be (1). Here, the output vector,  $\mathbf{X}$ , is equal to the **DFT matrix** times the input vector,  $\mathbf{x}$ . Matrix  $\mathbf{F}$  is a kind of special change of coordinates matrix.  $\mathbf{F}$  is orthogonal. What that means is that its columns all have the length  $N$  and are all perpendicular. This matrix is called the **Discrete Fourier Transform (DFT) matrix** or simply the **Fourier matrix F**.

$$(1) \quad \begin{matrix} X \\ N \times 1 \end{matrix} = \begin{matrix} F \\ N \times N \end{matrix} \begin{matrix} x \\ N \times 1 \end{matrix}$$

$F$  IS ORTHOGONAL:  
 COLUMNS ALL HAVE  
 LENGTH  $N$ , ARE  
 ALL PERPENDICULAR

or

$$X = Fx$$

## The Fourier matrix F

```
>> F = dftmtx(4)
```

```
1.0000 + 0.0000i    1.0000 + 0.0000i    1.0000 + 0.0000i    1.0000 + 0.0000i
1.0000 + 0.0000i    0.0000 - 1.0000i   -1.0000 + 0.0000i    0.0000 + 1.0000i
1.0000 + 0.0000i   -1.0000 + 0.0000i    1.0000 + 0.0000i   -1.0000 + 0.0000i
1.0000 + 0.0000i    0.0000 + 1.0000i   -1.0000 + 0.0000i    0.0000 - 1.0000i
```

**A****B**

1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i
1.0000 + 0.0000i	0.0000 - 1.0000i	-1.0000 + 0.0000i	0.0000 + 1.0000i
1.0000 + 0.0000i	-1.0000 + 0.0000i	1.0000 + 0.0000i	-1.0000 + 0.0000i
1.0000 + 0.0000i	0.0000 + 1.0000i	-1.0000 + 0.0000i	0.0000 - 1.0000i

```
>> F' * F
```

```
ans =
```

```
4      0      0      0
0      4      0      0
0      0      4      0
0      0      0      4
```

**C**

- **How we can see that in MATLAB?** There is a function in MATLAB that will give us the DFT matrix for a given order. For example, if we want to know what the DFT matrix is for an order of **4**, i.e., a **4x4** DFT matrix, we type in **F = dftmtx(4)**. Here, the length of all the column vectors is **4**.
- If we would take the dot product of any of these two vectors, such as **A** and **B**, we would get **0**. One way to say this really fast (i.e., to test this) is that if we would take the transpose of **F** (use **F'**) times **F**, we should get matrix **C** with special elements. In **C**, when we are off the diagonal, we get **0** elements, and when we are on the diagonal, we get **the length of that vector**, which is **4**.

## The Fourier matrix F

```
>> F = dftmtx(4)
```

```
1.0000 + 0.0000i  1.0000 + 0.0000i  1.0000 + 0.0000i  1.0000 + 0.0000i
1.0000 + 0.0000i  0.0000 - 1.0000i -1.0000 + 0.0000i  0.0000 + 1.0000i
1.0000 + 0.0000i -1.0000 + 0.0000i  1.0000 + 0.0000i -1.0000 + 0.0000i
1.0000 + 0.0000i  0.0000 + 1.0000i -1.0000 + 0.0000i  0.0000 - 1.0000i
```

(1)

---

```
>> F = dftmtx(2)
```

```
F =
```

```
1    1
1   -1
```

(2)

---

- One thing that we can immediately see when we look at the DFT matrix for order **4**, shown in (1), is that it is a pretty simple matrix. Here, there is no actual heavy multiplication by any weird numbers. This suggests that we could do the DFT for length-**4** signals without really doing a lot of multiplications.
- We see the same thing for order **2**, shown in (2). In these matrix operations, there are lots of zeros and lots of ones. So, the thing that we are going to talk about later is how we can make our life easier by taking what looked like complicated DFT's and reducing them down to really simple DFT's like (1) or (2). That is what is at the heart of the **Fast Fourier Transform** algorithms or **FFT** algorithms.

## How are the DTFT and DFT related?

How is this connected to the DTFT?

Say we have a finite-length DT signal.

$$x[0], x[1], \dots, x[N-1] \quad (\text{other } x[n] = 0)$$

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n} \quad (1) \quad \omega \in [-\pi, \pi]$$

$$\text{DTFT} \quad = \sum_{n=0}^{N-1} x[n] e^{-j\omega n} \quad (2)$$

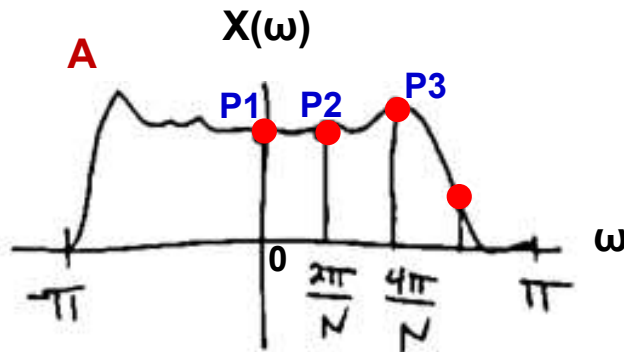
$$\left[ \text{DTFT} \quad X(\omega) = \sum_{n=0}^{N-1} x[n] e^{-j\omega n} \quad \omega \in [-\pi, \pi] \quad (3) \right.$$

$$\left. \text{DFT} \quad X[k] = \sum_{n=0}^{N-1} x[n] e^{-jk \frac{2\pi}{N} n} \quad k=0, 1, \dots, N-1 \quad (4) \right]$$

$$\boxed{X[k] = X(\omega) \Big|_{\omega = \frac{2\pi k}{N}}} \rightarrow X[k] = X\left(\frac{2\pi k}{N}\right) \quad (5)$$

- How is the DTFT,  $X(\omega)$ , connected to DFT,  $X[k]$ ? We know that we can take the DTFT of any discrete signal that we want. Let us suppose we have a finite length discrete-time signal,  $x[n]$ . Also, suppose we have  $N$  non-zero values of  $x[0]$ ,  $x[1]$ , etc.
- We can take the normal DTFT of this signal, i.e., we can find  $X(\omega)$  using equation (1), as we discussed before. The range is always  $2\pi$ -periodic. Normally, when we take the DTFT of some discrete-time signal,  $x[n]$ , we get a continuous function,  $X(\omega)$ , that is periodic in the range of  $-\pi$  to  $+\pi$ .
- Let us simplify (1). This is like saying since there is only  $N$  non-zero values of  $x[n]$ , we can turn the sum into (2). Going back to  $X[k]$  for DFT, (4), and by comparing it with (3), we see that they look very similar. This is saying this DFT, (4), is the DTFT, (3), if we were to evaluate (3) for a particular omega that was equal to  $2\pi k/N$ . This leads to (5).
- **Conclusion:**  $X[k]$  for DFT is like  $X(\omega)$  for DTFT that is evaluated at  $2\pi k/N$ .

## The DFT samples the DTFT at equally spaced frequencies



$$X[k]_{\text{DFT}} = X(\omega)_{\text{DTFT}} \Big|_{\omega = \frac{2\pi k}{N}}$$

$$X[k]_{\text{DFT}} = X\left(\frac{2\pi k}{N}\right) \quad (5)$$

- Normally, we have a DTFT,  $X(\omega)$ , that looks like graph **A**, which is between  $-\pi$  and  $+\pi$ . This is saying that we are going to get  $N$  values of the DFT, **(5)**, that are equally-spaced as shown by dots in **A**. The first value is going to be at **0** or **P1**, the second value at **P2** or  $2\pi/N$ , and the third value at **P3** or at  $4\pi/N$ , etc.
- So, what we are really doing here is we are taking the DTFT,  $X(\omega)$ , and we are **sampling** it at  $N$  equally-spaced points to get the DFT signals,  $X[k]$ .
- What we really care about is the continuous time  $X(\omega)$ . In MATLAB, we can sample it at different equally-spaced values of omega. If we want to sample the continuous DTFT with finer frequency resolution, we can increase the value of  $N$  in the DFT.

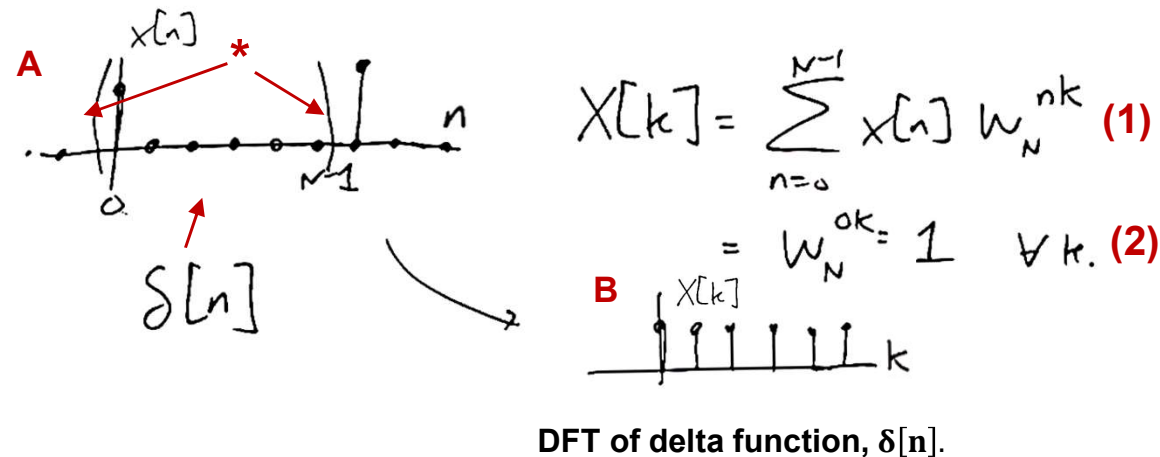
## The DFT samples the DTFT at equally spaced frequencies

INSIGHT: DFT IS THE DTFT OF  
A DISCRETE-TIME, FINITE-LENGTH SIGNAL  
EVALUATED AT  $N$  EQUALLY-SPACED  
POINTS  $\omega = \frac{2\pi k}{N}$ ,  $k = 0, 1, \dots, N-1$ .

- **Conclusion:** The insight is that the DFT,  $X[k]$ , is a sampled version of the DTFT,  $X(\omega)$ , of a finite-length discrete-time signal,  $x[n]$ , evaluated at  $N$  equally-spaced points of  $\omega$  that are equal to  $2\pi k/N$ , where  $k = 0, 1, 2, \dots, N-1$ .

## Examples of computing the DFT; DFT of delta function

### Example 1:



→  $X[k] = 1 \quad \forall k$

- Let us do some examples of DFT's and see how this **sampling property** works.
- Example 1:** If we have a delta function, one thing to be very careful of is that **in the DFT world everything is periodic**. So, when we talk about, for example, a delta function like **A**, what we are really talking about is that, it is a delta function inside the bracket **\***, but if we were to think about this signal outside of the bracket, it would be periodically repeating. When we compute the DFT of  $\delta[n]$ , the result,  $X[k]$ , equals 1 for all  $k$ , which demonstrates this periodicity. This is a representation similar to a **discrete-time Fourier series**. The assumption is that the input is repeating every  $N$  units. So, that part may be a little bit weird, because clearly some signals that we care about are not repeating. So, how we are going to deal with that?

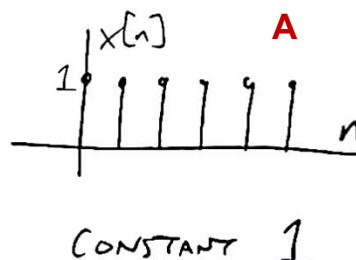
- Supposed that we just have the part within the range from zero to  $0$  and  $N-1$ . In that case, what is the DFT of that, i.e.,  $X[k]$ ? Starting from **(1)**, we get **(2)**. The sum is going to be non-zero when  $x[n]$  fires at  $n = 0$ , and hence, we get  $w_N^{0k} = 1$ , for all  $k$ . So, we put the delta function **A** in, what comes out is a constant, i.e., graph **B**.



## DFT of a constant

### Example 2:

**A**



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


$$= \sum_{n=0}^{N-1} w_N^{nk} \quad (1)$$

$$= w_N^0 + w_N^k + \dots + w_N^{(N-1)k} \quad (2)$$

**(3)**

$$= \frac{1 - (w_N^k)^N}{1 - w_N^k} = \frac{1 - w_N^{kN}}{1 - w_N^k} = \frac{0}{\neq 0} = 0$$

**B**



IF $k=0$	$X[k] = N$
IF $k \neq 0$	$X[k] = 0$



DFT of a constant function, 1.

- **Example 2:** What should we expect for the DFT when we put a constant in? Here,  $x[n]$  is going to be always equal to 1.  $X[k]$  in (1) is a **finite geometric sum**. So, we should be able to use the geometric sum formula (3).
- If  $k = 0$ , in (2), it is just like saying we are adding up  $1+1+\dots+1$ . So, we are adding  $N$  1's, or simply,  $x[k] = N \times 1 = N$ .
- If  $k \neq 0$ , in (3), the denominator is just some complex number that is non-zero. The term  $w_N^{kN}$  means that we are going  $N$  number of times around the unit circle, **B**. So, we are hitting the point **1**,  $k$  times, and we are always going to end up at **1**. So,  $w_N^{kN}$  is always equal to 1. So, for  $k \neq 0$ ,  $X[k] = 0$ . The signal will look like **C**.
- **Conclusion:** In the time domain, when we have a delta function, we get a constant in frequency domain. If we have a constant in the time domain, we get a delta function in the frequency domain. So, this kind of **duality**, where things look the same in both domains, still holds for the DFT.



## The orthogonality principle

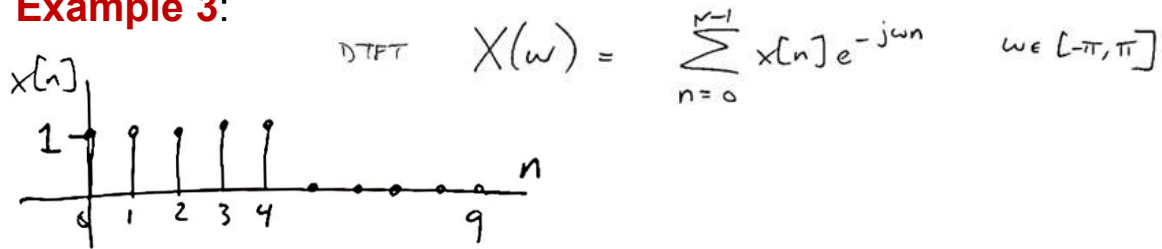
ORTHOGONALITY PROPERTY:

$$\sum_{n=0}^{N-1} W_N^{Mn} = \begin{cases} N & ; \text{ IF } M \text{ IS AN INTEGER MULT. OF } N \\ 0 & ; \text{ OTHERWISE.} \end{cases}$$

- **Orthogonality Property:** We actually proved and used this property in the previous example. We showed that if **M** is an integer multiple of **N** (including **M = 0**), we are adding a bunch of ones and the outcome of the summation would be **N**. When **M** is not an integer multiple of **N**, we get **0**, indicating that the DFT basis functions are **orthogonal**.

## A pulse: the DTFT vs. the DFT

### Example 3:



$$\underset{\text{DTFT}}{X(\omega)} = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n} = \sum_{n=0}^4 e^{-j\omega n} = \frac{1 - e^{-5j\omega}}{1 - e^{-j\omega}} \longrightarrow$$

$$\frac{e^{-5j\omega/2} (e^{5j\omega/2} - e^{-5j\omega/2})}{e^{-j\omega/2} (e^{j\omega/2} - e^{-j\omega/2})} = e^{-2j\omega} \frac{\sin(\frac{5\omega}{2})}{\sin(\frac{\omega}{2})} \longrightarrow$$

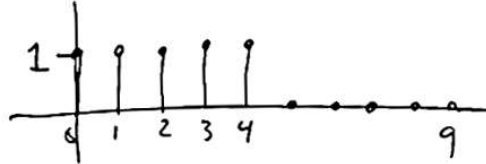
$$\longrightarrow \boxed{\underset{\text{DTFT}}{X(\omega)} = e^{-2j\omega} \frac{\sin(\frac{5\omega}{2})}{\sin(\frac{\omega}{2})}}$$

- Let us find the regular DTFT of the given signal. In our signal, we have **5 pulses ON** and **5 pulses OFF**.

## A pulse: the DTFT vs. the DFT

### Example 4:

$$\text{DFT } X[k] = \sum_{n=0}^{N-1} x[n] e^{-jk \frac{2\pi}{N} n} \quad k=0, 1, \dots, N-1$$

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


- Suppose we treated the previous signal as a length-10 discrete time signal. Let us find the **length-10 DFT signal**. After the mathematical manipulations that are shown below, we will arrive at (1).

INSTEAD, TAKE LENGTH 10 DFT:

$$X[k] = \sum_{n=0}^9 x[n] e^{-j \frac{2\pi}{10} kn}$$

$$= \sum_{n=0}^4 e^{-j \frac{2\pi}{10} kn} = \sum_{n=0}^4 (w_{10})^{kn}$$

$$= \frac{1 - w_{10}^{5k}}{1 - w_{10}} = \frac{1 - e^{-jk \frac{2\pi \cdot 5}{10}}}{1 - e^{-jk \frac{2\pi}{10}}}$$

$$= \frac{(e^{jk\pi/10} - e^{-jk\pi/10})(e^{-jk\pi/10})}{e^{-jk\pi/10}(e^{jk\pi/10} - e^{-jk\pi/10})} = e^{-j4k\pi/10} \frac{\sin \frac{\pi k}{2}}{\sin \frac{\pi k}{10}}$$

Multiply by  $\frac{e^{-j4k\pi/10}}{e^{-j4k\pi/10}}$

$k=0, 1, \dots, 9$

Factor out  $e^{-j4k\pi/10}$  both in the numerator and the denominator

$$X[k] = e^{-j4k\pi/10} \frac{\sin \frac{\pi k}{2}}{\sin \frac{\pi k}{10}} \quad k=0, 1, \dots, 9 \quad (1)$$

## A pulse: the DTFT vs. the DFT

### Summary for Examples 3 and 4:

#### Example 3:

$$\text{DTFT } X(\omega) = \sum_{n=0}^{N-1} x[n] e^{-j\omega n} = e^{-2j\omega} \frac{\sin(\frac{5\omega}{2})}{\sin(\frac{\omega}{2})}$$

#### Example 4:

$$\text{DFT } X[k] = \sum_{n=0}^{N-1} x[n] e^{-jk \frac{2\pi}{N} n} \quad k=0,1,\dots,N-1 = e^{-j4k\frac{\pi}{10}} \frac{\sin \frac{\pi k}{2}}{\sin \frac{\pi k}{10}}$$

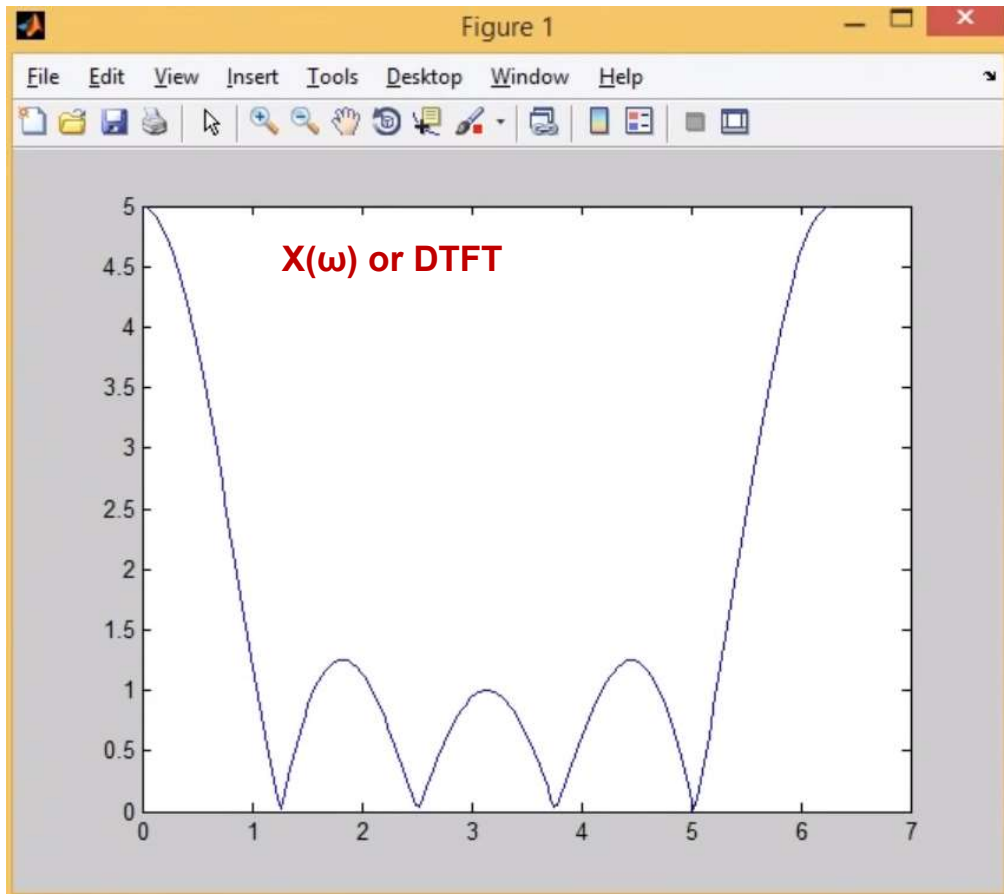
$k=0,1,\dots,9$

$$\text{DTFT @ } \omega = \frac{2\pi k}{10} = e^{-2j \frac{2\pi k}{10}} \frac{\sin(\frac{5 \cdot \frac{2\pi k}{10}}{2})}{\sin(\frac{\frac{2\pi k}{10}}{2})} = e^{-2j \frac{2\pi k}{10}} \frac{\sin(\frac{\pi k}{2})}{\sin(\frac{\pi k}{10})} = e^{-j4k\frac{\pi}{10}} \frac{\sin \frac{\pi k}{2}}{\sin \frac{\pi k}{10}}$$

- Let us now compare the results of these two examples.
- We now see that if we were to evaluate the DTFT,  $X(\omega)$ , in **Example 3** at multiples of  $2\pi/10$  or at  $\omega = 2\pi k/10$ , we will get DFT,  $X[k]$ , in **Example 4**.
- **Conclusion:** When we sample the DTFT, we get the DFT values.

## MATLAB demonstration of how the DFT samples the DTFT

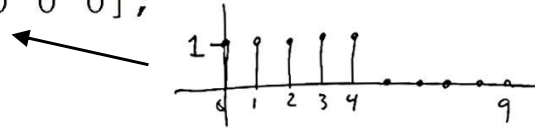
```
>> w = linspace(0,2*pi,200); ←  $\omega$ 
>> dtfft = exp(-2*j*w).*(sin(5*w/2)./sin(w/2));
      ↙  $X(\omega)$  or DTFT
>> plot(w,abs(dtfft))
```



- Let us do the same example in MATLAB. Here,  $X(\omega)$  or **DTFT** is the continuous function of  $\omega$ , and it is  $2\pi$ -periodic. It is represented by “**dtfft**”. The plot of the absolute value of the DTFT is shown in **A**.
- The MATLAB command **w = linspace(0,2\*pi,200)** generates a linearly spaced vector of **200** elements between **0** and **2\*pi** (inclusive). Here is a breakdown of what each part does:
  - **w = ...**: This assigns the result of the expression on the right-hand side to a variable named **w**.
  - **linspace(0,2\*pi,200)**: This is the function that creates the vector.
    - ✓ **linspace**: This is a built-in MATLAB function used for generating linearly spaced vectors.
    - ✓ **(0,2\*pi)**: This defines the interval for the generated values. The vector will range from **0** (inclusive) to **2\*pi** (inclusive).
    - ✓ **200**: This specifies the number of elements in the vector. The function will create **200** evenly spaced values within the given interval.

## MATLAB demonstration of how the DFT samples the DTFT

```
>> s = [1 1 1 1 1 0 0 0 0 0];
>> dft = fft(s);
>> whos
```

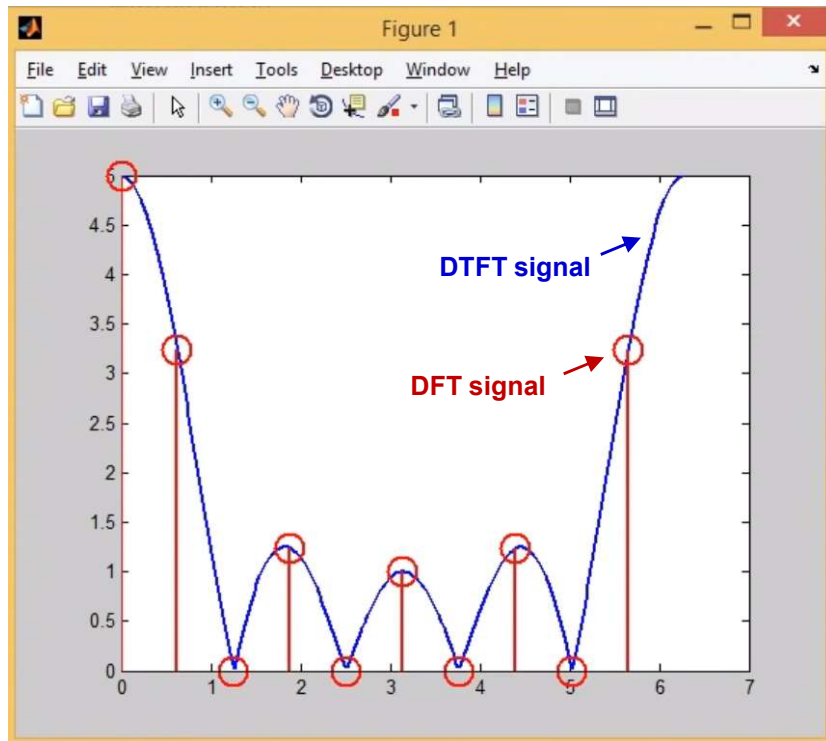


```
dft      1x10 (1)      160 double complex
```

- Let us see what we would get if we were to just take the **DFT** of the original signal, **s = [1 1 1 1 1 0 0 0 0 0]**. Let us find the **dft**, using **fft(s)** command. We can see the DFT is a length-**10** vector, **(1)**.
- The **fft function** is the primary tool for computing the DFT of a finite-length discrete-time signal in MATLAB. The **fft** function actually implements the Fast Fourier Transform (FFT) algorithm, which is a computationally efficient way to calculate the DFT.
- The **whos command** in MATLAB is used to display information about the variables currently in our workspace or within a specific MAT-file. It provides details like the name, size, type, and bytes used in memory for each variable.

## MATLAB demonstration of how the DFT samples the DTFT

```
>> ws = 2*pi/10*[0:9]; (1)
>> hold on (2)
>> stem(ws, abs(dft), 'r') (3)
```

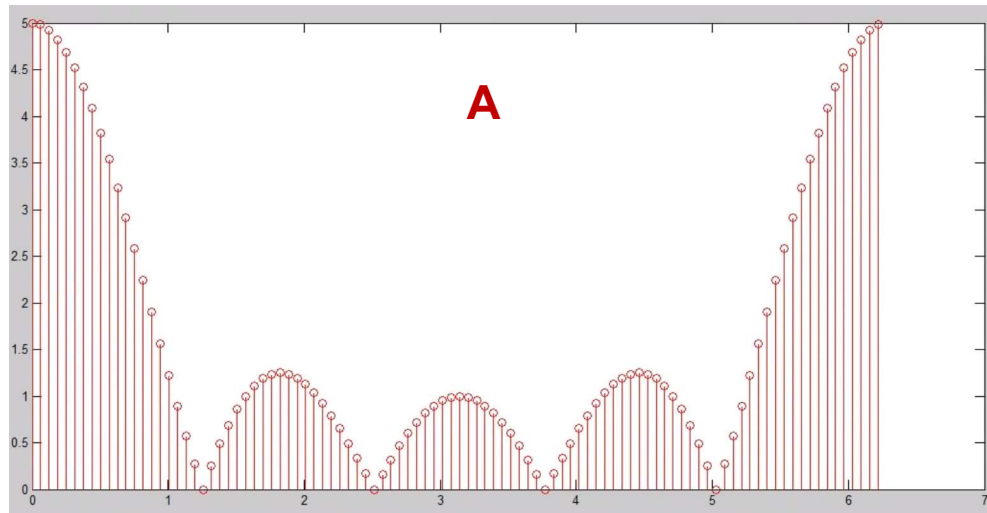


- Our omega samples,  $\omega_s$  (or **ws**), are going to be  $2\pi/10$  times **[0:9]**, which means that these are equally-spaced from **0** to **9**. The command for **ws** is shown in **(1)**. These are going to be our samples. Now, we **hold on**, **(2)**, and then we stem the frequencies with the absolute value of the DFT values, but in **red**, **(3)**.
- What we get is exactly what is happening. The graph shows that **the DFT values are indeed sampling the continuous values**. We can see that the **DFT samples** (the **red circles**) are just hitting the **DTFT signal** (the **continuous blue curve**) at equally-spaced values.

## MATLAB demonstration of how the DFT samples the DTFT

```
>> s = [ones(1,5), zeros(1,95)]; (1)
>> dft = fft(s); (2)

>> plot(w,abs(dtft)) (3)
>> ws = 2*pi/100*[0:99]; (4)
>> stem(ws, abs(dft),'r')
```

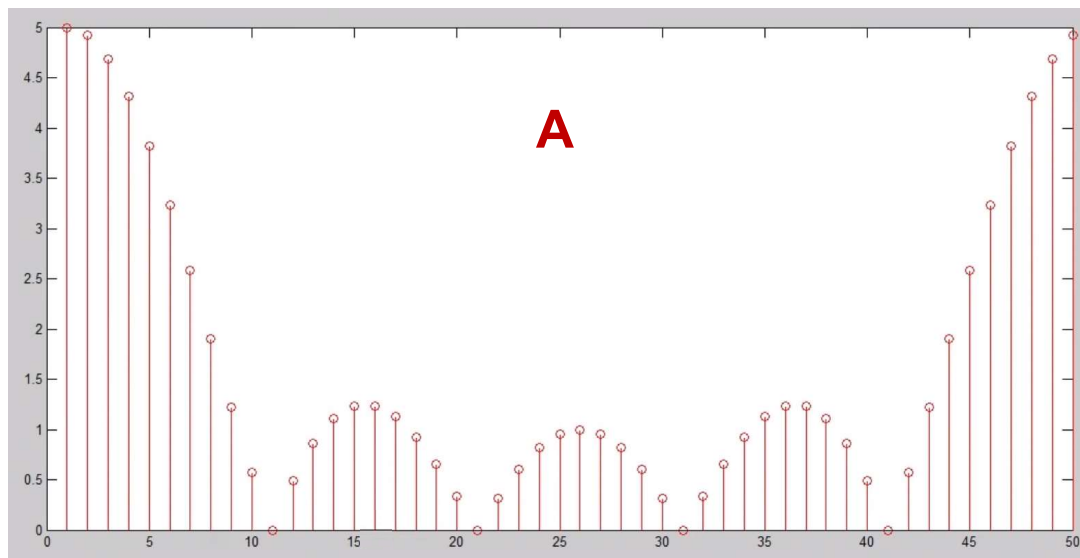


- If we were to amp up the length of the DFT (because we wanted to get a better or a finer resolution DFT), what we could do is called **zero padding**.
- So, instead of thinking about the original signal as a length-**10** signal, we are going to think about this as a length-**100** signal. In MATLAB, we are going to take five **1**'s and then stick ninety five **0**'s at the end of that, **(1)**. Now, we are going to take the DFT of that, **(2)**, and again, plot the original DTFT, **(3)**. Our new frequencies are going to be sampled every  **$2\pi/100$** , **(4)**. Based on graph **A**, now we see that the DFT is sampling that DTFT **very finely**.



## MATLAB demonstration of how the DFT samples the DTFT

```
>> ws = 2*pi/100*[0:99]; (1)  
      (No need to define the vector ourself.)  
>> dft = fft(s,50); (2)  
>> stem(abs(dft), 'r')
```



- **Conclusion:** The idea here is that we can get an arbitrarily close approximation to the underlying continuous signal by taking a longer and longer DFT. Actually, this idea is built into MATLAB. We do not have to make this long vector ourself by using (1). Instead, we can use (2), which says that, we just say we want the DFT to be of length **50** (or any other length that we want). The graph for a length of **50** is shown in **A**. The longer the DFT is, the more finely we are sampling the underlying DTFT, which is usually what we want to do.

## DFT properties

PROPERTIES:

- LINEARITY

Linearity Property:

$$X_1[k] = \text{DFT}(x_1[n])$$

$$X_2[k] = \text{DFT}(x_2[n])$$

$$\text{DFT}(ax_1[n] + bx_2[n]) = a \cdot \text{DFT}(x_1[n]) + b \cdot \text{DFT}(x_2[n])$$

$$\text{DFT}(ax_1[n] + bx_2[n]) = aX_1[k] + bX_2[k]$$

- SHIFT

$$x[n-m] \xleftrightarrow{\text{DFT}} W_N^{km} X[k]$$

- **Linearity:** Why is Linearity Important? The linearity property is important because it allows for **superposition** in signal analysis and processing. Many real-world signals are combinations of simpler signals, and linearity means we can analyze each component separately. This simplifies tasks like filtering, modulation, and signal decomposition.
- **Practical Applications:**
  - **Signal Filtering:** The linearity of the DFT allows for filtering operations by linearly combining different frequency components.
  - **Modulation and Demodulation:** In communications, linearity helps when analyzing the superposition of signals for encoding/decoding.
  - **Additive Synthesis:** In audio processing, complex sounds are often synthesized by adding simpler waveforms, and linearity simplifies frequency-domain analysis.
- **Shift:** The shifting is always taking place in a cyclic world. This is called **Cyclic Shift**.

## DFT properties

### The Modulo Operation in MATLAB:

- The modulo operation in MATLAB, represented by the **mod function**, returns the **remainder** after division of one number by another.

### Syntax in Matlab:

**b = mod(a, m)**, which means **a mod m = b**

Where:

- a** is the dividend
- m** is the divisor
- b** is the remainder

### Example in Matlab:

**result = mod(17, 5)**

This will output **2** because **17** divided by **5** leaves a remainder of **2**.

```
>> d=mod(17,5)
```

```
d =
```

```
2
```

### Key Points:

- The result of the modulo operation always has the same sign as the divisor.
- The mod function can be used with both positive and negative numbers.
- In the context of cyclic shifts, the modulo operation is used to ensure that the shift index is within the bounds of the array.
- In summary, the **mod** function in MATLAB provides a convenient way to calculate the remainder after division, which is essential for various mathematical and engineering applications, including cyclic shifts.

## DFT properties

### The Cyclic Shift Property:

- **The cyclic shift property** is an important concept in the context of the Discrete Fourier Transform (DFT). It describes how shifting a signal in the time domain affects its representation in the frequency domain.

### Understanding the Property:

- Imagine a discrete-time signal  $x[n]$ . Its DFT is represented by  $X[k]$ .
- If we perform a cyclic shift of the signal by  $d$  positions, the new signal becomes  $x[n - d]$ . Here,  $n$  and  $d$  are integers representing time indices.
- The cyclic shift property states that the DFT of the shifted signal,  $X'[k]$ , is obtained by rotating the original DFT,  $X[k]$ , by  $d$  positions in the circular frequency domain.

### Circular Analogy:

- Think of the frequency domain as a circle. Each point on the circle represents a different frequency component of the signal. The DFT coefficients  $X[k]$  are like values placed at specific points on this circle.
- When we perform a cyclic shift in the time domain, it is like rotating the entire signal along the time axis.
- In the frequency domain, this rotation translates to a **circular shift** of the DFT coefficients  $X[k]$  by  $d$  positions around the frequency circle.

## DFT properties

### Example:

Consider a signal  $\mathbf{x}[n]$  with the following DFT:

- $\mathbf{X}[k] = [10, 5, 2, 1, 4, 3]$  (where  $k = 0$  to  $5$ ;  $N = 6$ ) ; For example:  $\mathbf{X}[0] = 10$ .
- Let us say we perform a cyclic shift of the signal by  $2$  positions ( $d = 2$ ). The shifted signal would be  $\mathbf{x}[n - 2]$ .
- According to the cyclic shift property, the DFT of the shifted signal,  $\mathbf{X}'[k]$ , is obtained by rotating  $\mathbf{X}[k]$  by  $2$  positions on the frequency circle. Here is the result:
- $\mathbf{X}'[k] = [4, 3, 10, 5, 2, 1]$  (original values rotated by  $2$  positions)

### Visualization:

- Imagine the DFT coefficients  $\mathbf{X}[k]$  plotted as **red** dots around a circle, with each point representing a frequency. Shifting the signal in the time domain corresponds to rotating these dots around the circle by the amount of the shift.

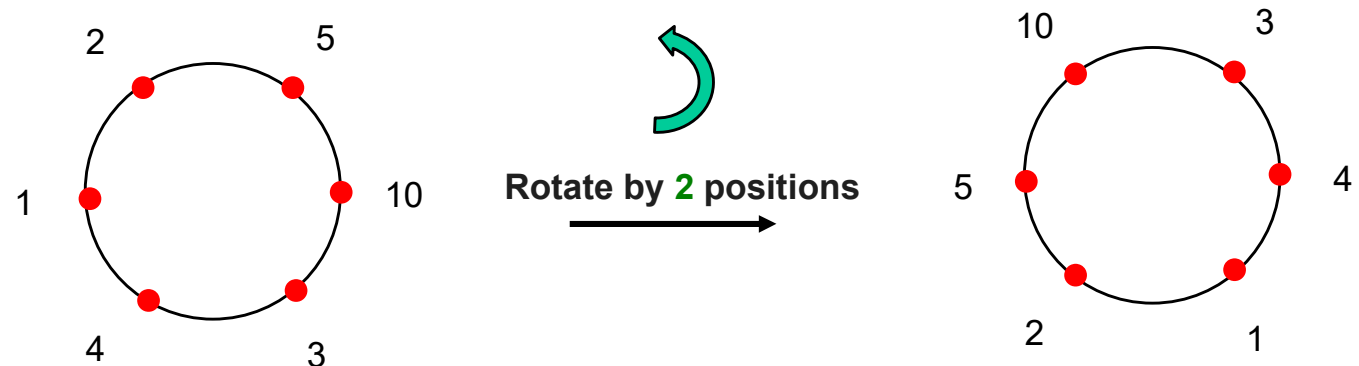
### Sample Calculations:

$$\mathbf{X}'[k] = \mathbf{X}[(k - d) \bmod N]$$

$$\vdots$$

$$\mathbf{X}'[2] = \mathbf{X}[(2 - 2) \bmod 6] = \mathbf{X}[0] = 10$$

$$\mathbf{X}'[3] = \mathbf{X}[(3 - 2) \bmod 6] = \mathbf{X}[1] = 5$$

$$\vdots$$


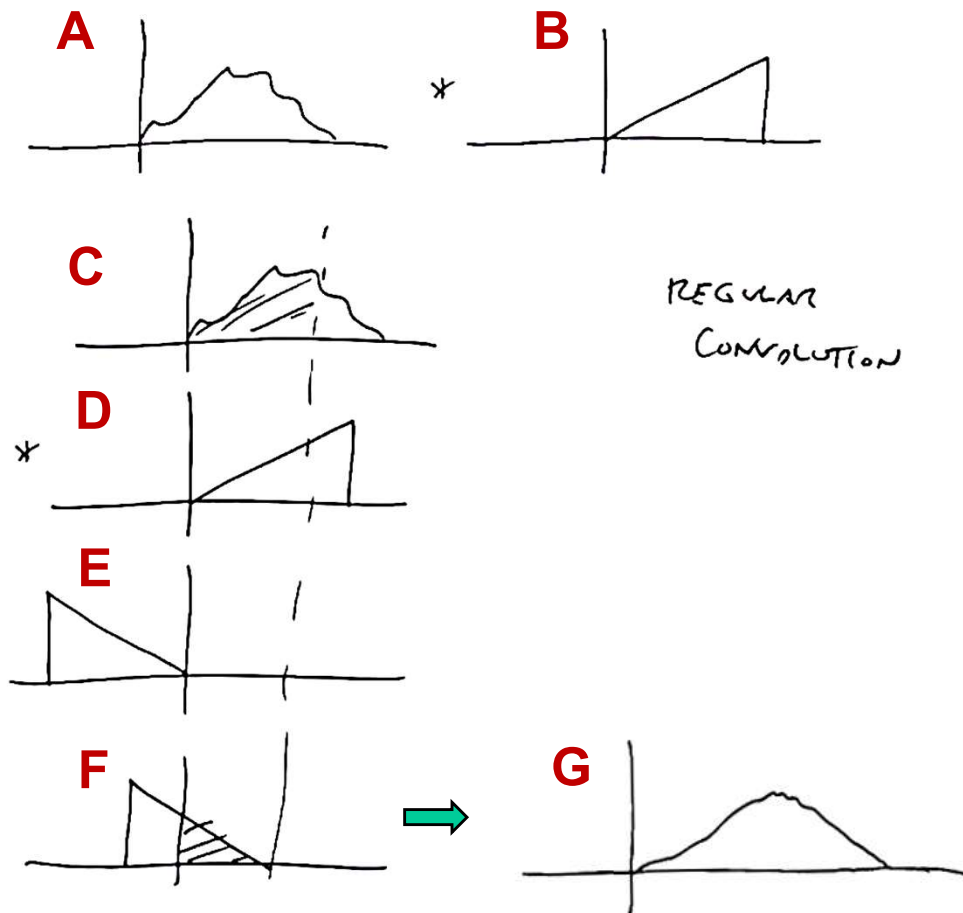
## Cyclic convolution

### • CYCLIC CONVOLUTION



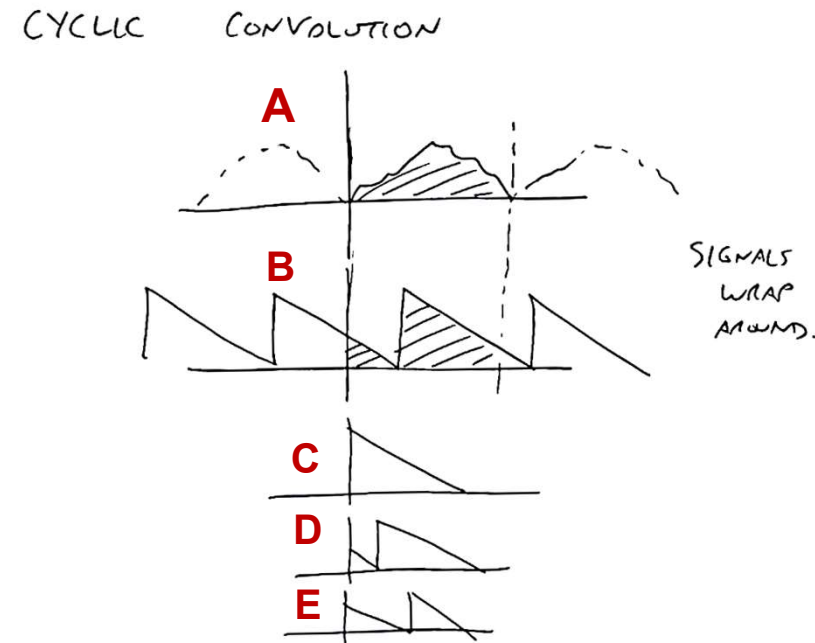
Notation used for  
cyclic convolution

### REGULAR CONVOLUTION



- In a similar way, when we talk about convolving two signals, we are talking about this new concept called **cyclic convolution**. We use an asterisk in a circle to connote that what we mean is cyclic, and not regular convolution, which is usually shown just by an asterisk. Cyclic convolution is also known as **circular convolution**.
- Let us start with **regular convolution**. Regular convolution would be like saying, we want to convolve a given signal, **A**, with another signal, **B**. What we would do first is to flip around signal **B** (now shown as signal **D**). We now have signal **E**. Then we start to drag signal **E** across signal **A**, and in places where there is an overlap, we would integrate up these two signals, shown by shaded area in **C** and **F**. The final convolution would like **G**, where when we start from the left side of graph **G**, there is nothing for a while (to the left of the origin), and then it would come to some peak, then come down, and then it would be nothing again. That is the world of regular convolution.

## Cyclic convolution



- **Cyclic Convolution:** Instead of the previous signal for regular convolution that we had one period of the signal, everything here is assumed to be periodic. If we want to convolve signal **A** with that triangle again, what we have is copies of that triangle (shown in **B**) that are going on like what we see in graph **B**. And, what we do is, we look at the integral as we shift this triangle across, bump by bump.
- So, maybe the first value is like graph **C**, the next value is like graph **D**, and the next value is like graph **E**. This is as if the triangle never falls off the page, and it is constantly shifting across and copies of it are coming in from the left. This means that the convolution is also going to be periodic. Here, the signals wrap around in a way that they do not for the regular convolution.
- The main difference between DFT-based convolution (cyclic) and regular convolution is the periodic nature of cyclic convolution. In cyclic convolution, the **signals wrap around instead of terminating**. The DFT uses this property for convolution in the frequency domain. If we convolve two signals in the time domain using DFT, we are essentially performing cyclic convolution.
- **Conclusion:** The way convolution works for the DFT is different from that of the regular convolution. The convolution property still holds, but in the cyclic convolution sense.

## Cyclic convolution

FOR THE DFT, WE HAVE THE SAME KIND OF CONVOLUTION  $\Leftrightarrow$  MULTIP. PROPERTIES, BUT THEY ARE CYCLIC.

$$x[n] \circledast h[n] \xleftrightarrow{\text{DFT}} X[k] H[k] \quad (1)$$

$\nwarrow \quad \nearrow$   
 LENGTH N  
 SIGNALS

HOW TO GET THE "REGULAR" CONVOLUTION WE NEED FOR LTI SYSTEMS?

- For the DFT, we have the same kind of **convolution  $\Leftrightarrow$  multiplication** property, but they are cyclic. What that means is that if we have,  **$x[n]$  circularly convolved** with  **$h[n]$** , and these are both length-**N** signals, then in the frequency domain, we have the product of  **$X[k]$**  and  **$H[k]$** , which is what we want, **(1)**. This is simply  **$X[k]$**  times  **$H[k]$** .
- That is a little bit unsettling, because usually what we really care about is, that for linear time invariant systems, we do not want this weird cyclic convolution. Instead, we want the regular convolution. That is what we grew up on!
- **So, how do we turn cyclic convolution into regular convolution?** The answer again lies in the idea of **zero padding the signal**. It turns out that we **can** get away with this by using zero padding. Next, we will look at this trick of how to get the regular convolution we need for LTI systems by using zero padding the signal.



# Representing cyclic convolution as a matrix-vector product

SUPPOSE  $x$  AND  $h$  ARE LENGTH- $N$  SIGNALS. WHAT IS  $x \otimes h$ ?

$$\begin{array}{c}
 (1) \left[ \begin{array}{cccc} x[0] & x[1] & \dots & x[N-1] \\ h[2] & h[1] & & h[0] \end{array} \right] \left[ \begin{array}{c} y[0] \\ y[1] \\ \vdots \end{array} \right] \\
 \begin{array}{c} (3) \\ (4) \end{array} \left[ \begin{array}{cc} h[1] & h[0] \\ h[3] & h[2] \end{array} \right] \left[ \begin{array}{c} y[1] \\ y[2] \\ \vdots \end{array} \right] \\
 (5) \left[ \begin{array}{l} y[0] = h[0]x[0] + h[N-1]x[1] + \dots + h[1]x[N-1] \\ y[1] = h[1]x[0] + h[0]x[1] + \dots + h[2]x[N-1] \\ \vdots \\ y[N-1] = h[N-1]x[0] + \dots + h[0]x[N-1] \end{array} \right]
 \end{array}$$

- Now, let us look at the trick. Let us suppose we have  $x[n]$  and  $h[n]$  that are length- $N$  signals. **What is  $x$  circularly convolved with  $h$  or  $x \otimes h$ ?**
- Here, we have the set of  $x[n]$ 's in (1) and the set of  $h[n]$ 's in the second row. Remember that everything is cyclic. That means that we have this kind of wrap around property that things wrap around in the second row of the diagram.
- When we flip  $h[n]$  around, we will have (2). Now we put ...,  $h[2]$ ,  $h[1]$ ,  $h[0]$  as shown in (3). This will give us  $y[0]$ , which is the sum of the product of all these things in the **blue brackets**.
- Next, when we click  $h[n]$  over to the right, as shown in the third row, things will kind of **fall off the side** as shown in (4). This will give us  $y[1]$ , and so on.
- Every time we click  $h[n]$  one more unit, we get a different product of  $h[n]$ 's with the  $x[n]$ 's. We are going to get  $N$  different products as we shift  $h[n]$ ,  $N$  different times. This is a **linear transformation of the input,  $x[n]$** . So, for each of the  $y[n]$  values, we are getting a combination of the  $N$ ,  $x[n]$  values, and  $N$ ,  $h[n]$  values, as shown in (5).

## Representing cyclic convolution as a matrix-vector product

$$\begin{bmatrix} y[0] \\ y[1] \\ \vdots \\ y[N-1] \end{bmatrix} = \begin{bmatrix} h[0] & h[N-1] & & & \\ h[1] & h[0] & & & \\ h[2] & & h[0] & & \\ \vdots & & & \ddots & \\ h[N-1] & h[N-2] & h[2] & h[1] & h[0] \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix}$$

CIRCULANT MATRIX

THIS MATRIX-VECTOR PRODUCT  
CORRESPONDS TO CIRCULAR CONVOLUTION

- We could think of this operation as a **matrix-vector product**. So, we can write this whole convolution in a **linear algebra** format. Here, the matrix of  $h[n]$ 's is a big  $N$  by  $N$  matrix that is multiplying by the input  $x[n]$ 's to give us the output,  $y[n]$ 's.
- The matrix of  $h[n]$  is a special matrix. What we see in this matrix is that **all the diagonal elements are the same**. For example, all the elements on line (1) are  $h[1]$ . This matrix has a very special structure, and is called a **Circulant Matrix**. This is because there is really only  $N$  unique values that are being circulated along the diagonals.
- Now, let us put it all together. **How do we get the linear convolution that we want?** We know that this operation is a type of circular convolution. Therefore, this kind of **matrix-vector product** corresponds to circular convolution.

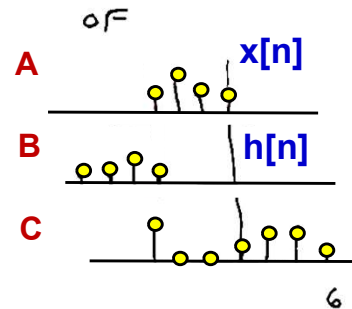
# Representing normal convolution as a matrix-vector product

## Example:

CONSIDER LINEAR CONVOLUTION OF

$$x[0] \dots x[3]$$

$$h[0] \dots h[3]$$



$$y[0] = x[0] h[0]$$

$$y[1] = x[0] h[1] + x[1] h[0]$$

$$y[2] = x[0] h[2] + x[1] h[1] + x[2] h[0]$$

$$7 \times 1 \begin{bmatrix} y[0] \\ \vdots \\ y[6] \end{bmatrix} = \begin{bmatrix} h[0] & 0 & 0 & 0 \\ h[1] & h[0] & 0 & 0 \\ h[2] & h[1] & h[0] & 0 \\ h[3] & h[2] & h[1] & h[0] \\ 0 & 0 & 0 & h[3] \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix} \quad (1)$$

- What do we want when we have **linear convolution**? As an example, let us consider a linear convolution of a length-4 vector input signal,  $x[0]$  to  $x[3]$ , with a length-4 vector of an impulse response,  $h[0]$  to  $h[3]$ .
- We will use the usual flipping and shifting of the input, no circular or anything. We can **also** think about this process as a kind of a matrix-vector product. As shown in **A** and **B**, we are going to have 4 values of  $x[n]$  and 4 values of  $h[n]$ . Now, we are just going to run  $h[n]$  across in graph **B** to the right. After that, we are going to have 7 total values. As shown in **C**, the end of this process is going to be when  $h[n]$  runs off the end of  $x[n]$ .
- Eventually, we are going to have  $y[0]$  to  $y[6]$ . The whole process can be re-written in matrix form, **(1)**. We can see here that **(1)** actually looks similar to what we had before, except none of the values are wrapping around and also the matrix of  $h[n]$ 's is not square.

# Computing normal convolution as cyclic convolution with zero-padding

(1)

$$\begin{matrix} 7 \times 1 \\ \begin{bmatrix} y[0] \\ \vdots \\ y[6] \end{bmatrix} \end{matrix} = \begin{matrix} 7 \times 4 \\ \begin{bmatrix} h[0] & 0 & 0 & 0 \\ h[1] & h[0] & 0 & 0 \\ h[2] & h[1] & h[0] & 0 \\ h[3] & h[2] & h[1] & h[0] \\ 0 & 0 & 0 & h[3] \end{bmatrix} \end{matrix} \begin{matrix} 4 \times 1 \\ \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix} \end{matrix}$$

(2)

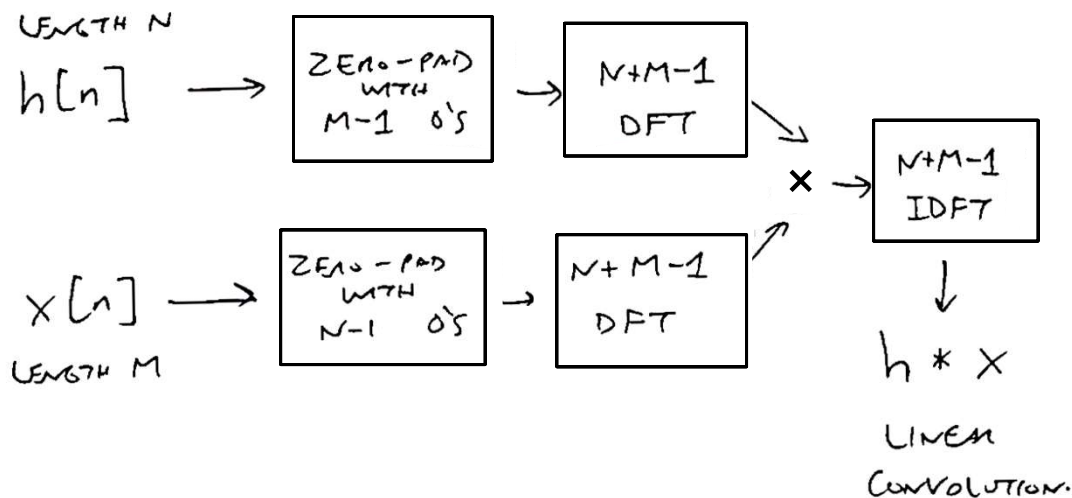
$$\begin{matrix} \begin{bmatrix} y[0] \\ \vdots \\ y[6] \end{bmatrix} \end{matrix} = \begin{matrix} \begin{bmatrix} h[0] & 0 & 0 & 0 & h[3] & h[2] & h[1] \\ h[1] & h[0] & 0 & 0 & h[3] & h[2] \\ h[2] & h[1] & h[0] & 0 & h[3] \\ h[3] & \mathbf{A} & h[0] \\ h[3] & h[1] \\ 0 & 0 & 0 & h[3] & h[2] & h[1] & h[0] \end{bmatrix} \end{matrix} \begin{matrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \\ 0 \\ 0 \\ 0 \end{bmatrix} \end{matrix}$$

circular matrix

- If we wanted to, we could add some **extra columns** to matrix of  $\mathbf{h}[n]$  and also some **extra zeros** to the matrix of  $\mathbf{x}[n]$ . Note that matrix  $\mathbf{y}[n]$  is basically a 7 by 1 vector, matrix  $\mathbf{h}[n]$  is a 7 by 4 matrix, and matrix  $\mathbf{x}[n]$  is a 4 by 1 vector. What we could do here is to just add a bunch of zeros to the bottom end of  $\mathbf{x}[n]$  matrix and a bunch of columns to the right side of  $\mathbf{h}[n]$  matrix and then multiply those **to get the same result**. So, instead of (1), we will get (2).
- For  $\mathbf{x}[n]$  matrix, we basically added a few zeros. To generate a **circular matrix**, we start with the original matrix of  $\mathbf{h}[n]$ . For matrix  $\mathbf{h}[n]$ , on the left part of this matrix, shown by **A**, we are going to put the same matrix of  $\mathbf{h}[n]$  that we have. The remaining elements are going to be put in a way so that the whole new matrix of  $\mathbf{h}[n]$  may look like the circular matrix. These added elements are placed in the **red dashed-line boxes**.
- What we did here will not affect the results of  $\mathbf{y}[n]$ 's and it is just another way of showing the same matrix operation. Now that we have this new  $\mathbf{h}[n]$  matrix, i.e., the circular matrix, we can do the linear convolution (the regular convolution) that we want by zero padding the input and the impulse response.

## Block diagram for zero padding

CONCLUSION: To do LINEAR CONVOLUTION WITH THE DFT:



- Conclusion:** To do linear convolution with the DFT, we would follow the procedure shown in the block diagram. Let us suppose we have  $h[n]$  with length  $N$ , and the input  $x[n]$  with length  $M$ . We would zero-pad  $h[n]$  with  $M-1$  zeros and we would also zero-pad  $x[n]$  with  $N-1$  zeros. Now these results are both  $N+M-1$  DFT. Next, we multiply these two things together shown by  $\times$  in the diagram. Then we take the  $N+M-1$  inverse DFT (IDFT). What comes out of this is  $h * x$ , i.e.,  $h$  convolved with  $x$ , which is a **regular linear convolution**.
- Whenever we have a convolution in MATLAB, what is happening under the hood in this platform is a DFT. Even if we are convolving very short signals, MATLAB is using its Fourier transform algorithm to compute that convolution. But before it does, it is making sure that each of these vectors,  $h[n]$  and  $x[n]$ , has the same length. And then it gets the DFT using that special length.

## Block diagram for zero padding

- **Example:** If we have a **100** length vector convolved with a **49** length vector, these two vectors are both getting padded to what length before we do the convolution?

### Zero-Padding for Convolution (Determining the Padded Length):

- **Understanding the Requirement:**
  - When performing convolution using the Fast Fourier Transform (FFT), it is essential to zero-pad the input signals **to avoid circular convolution artifacts**. The question is: to what length should we pad each signal?
- **Determining the Padded Length:**
  - To prevent overlap in the circular convolution, the combined length of the two zero-padded signals must be **greater than or equal to** the sum of the lengths of the original signals **minus one**.
- **Mathematically:**
  - Let **N1** be the length of the first signal (**100** in our example).
  - Let **N2** be the length of the second signal (**49** in our example).
  - The minimum padded length **N** for both signals should satisfy:
$$N \geq N1 + N2 - 1$$
- **In our specific case:**
  - **N**  $\geq 100 + 49 - 1$
  - **N**  $\geq 148$

Therefore, both the **100**-length and **49**-length vectors should be zero-padded to a length of at least **148** before performing the convolution using FFT.

- **Common Padding Lengths:**
  - While the minimum required length is **148**, often in practice, we choose powers of **2** for the padded length to efficiently utilize FFT algorithms. **In this case, the nearest power of 2 greater than 148 is  $2^8 = 256$** . So, both signals would be zero-padded to a length of **256**. Note that  $2^7$  is **128**.
- **In conclusion**, to avoid circular convolution artifacts and obtain the correct linear convolution result, the zero-padded length should be greater than or equal to the sum of the original signal lengths minus one. Common practice is to choose a power of **2** for computational efficiency.

# End of Lecture 10