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

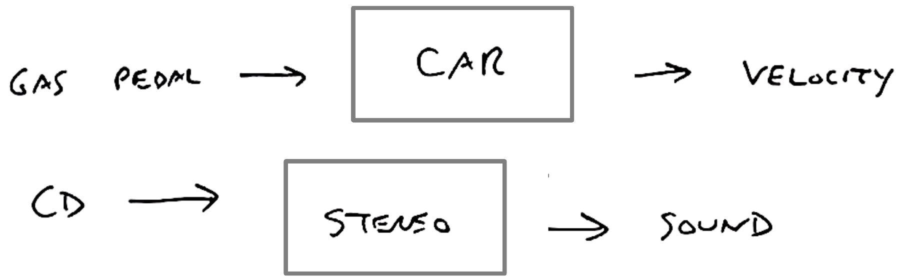
Lecture 1:
Signals

Table of Contents

What is a signal? What is a system?

SIGNALS COME FROM SENSORS

SYSTEMS PROCESS SIGNALS TO PRODUCE OTHER SIGNALS

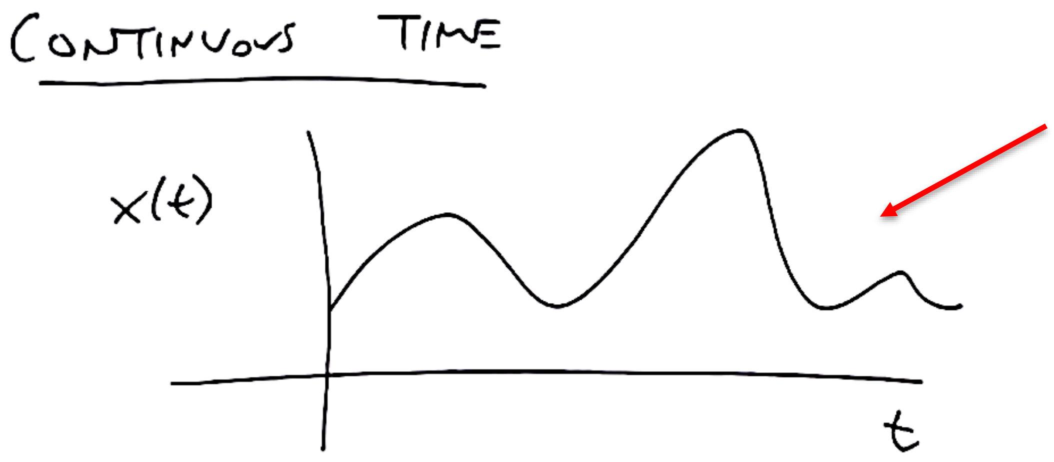


- **Signals:** In Digital Signal Processing (DSP), a signal is defined as a physical quantity that varies with time, space, or any other independent variable and carries information. Specifically, in DSP, signals are typically represented as discrete-time signals, which means they are sequences of numbers sampled at regular intervals. Signals come from sensors.
- **Sensors Systems:** For example, thermometers, voltmeters, microphones, and cameras.

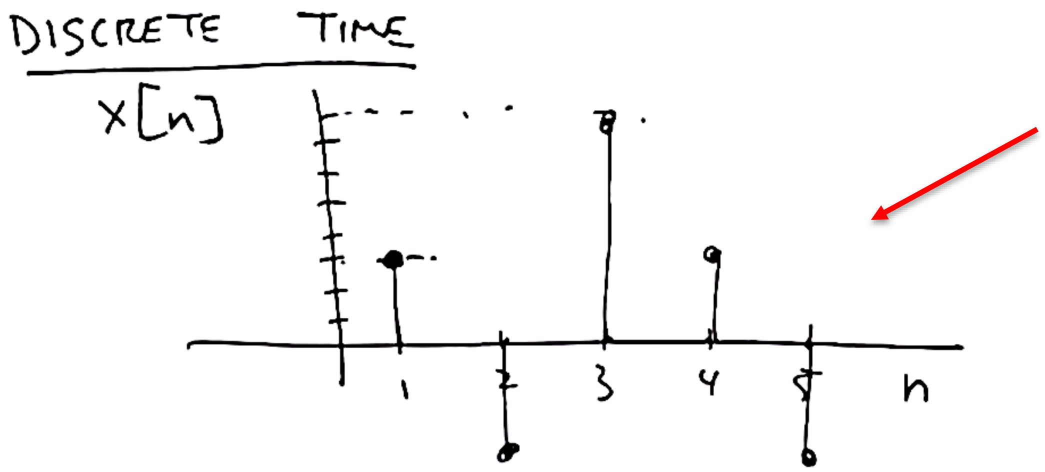
What is a signal? What is a system?

- In Digital Signal Processing (DSP), a **system** refers to any process that transforms a discrete-time input signal into a discrete-time output signal.
- Here is a breakdown of this definition:
  - **Discrete-time signal:** Signals in DSP are represented by a sequence of numbers, where each number corresponds to the amplitude of the signal at a specific point in time.
    - ✓ These points in time are equally spaced, unlike continuous-time signals which can have any value on the time axis.
  - **Transformation:** The system takes the input signal and performs some kind of operation on it, resulting in a new signal with potentially different characteristics.
    - ✓ This operation can involve various manipulations like filtering, amplification, delaying, or more complex computations.

Continuous time vs. discrete time (analog vs. digital)



- The **unbroken line** represents a signal at any point in time.



- We do not have anything that occurs between the gaps of these **stem plots**, and so whenever we draw a **digital signal**, it is usually going to be like the following where we have little stem plots that represent what is happening at every point in time.

- We do not have infinite resolution on the **y**-axis either. On the **y**-axis, we may only have some discrete locations where the sticks have to fall exactly on one of the ***ticks***. That is called **quantization**.

Continuous time vs. discrete time (analog vs. digital)

- There is a lot of theory behind how we go from a continuous signal to a discrete signal in such a way that we do not lose information.
    - That is what the **sampling theorem** is all about.
- We are going to cover that in this class too.
- In certain conditions we can actually sample continuous time signals, so the samples are good enough to reconstruct the original continuous time signals.

Signal transformations

- **Signal transformations** in DSP encompass a wide range of operations that modify a signal's characteristics.
- Here is some common transformations:
  - **Flipping**
  - **Scaling**
  - **Shifting**
- **Note:** In some references, we may see the following definitions:

Flipping (Sign Inversion):

- **Effect:** This transformation multiplies each sample of the signal by **-1**. In the time domain, it flips the signal about the horizontal (**x**) axis, inverting its polarity.
- **Mathematical representation:** **$y(n) = -x(n)$**, where **$x(n)$** is the original signal and **$y(n)$** is the flipped signal.

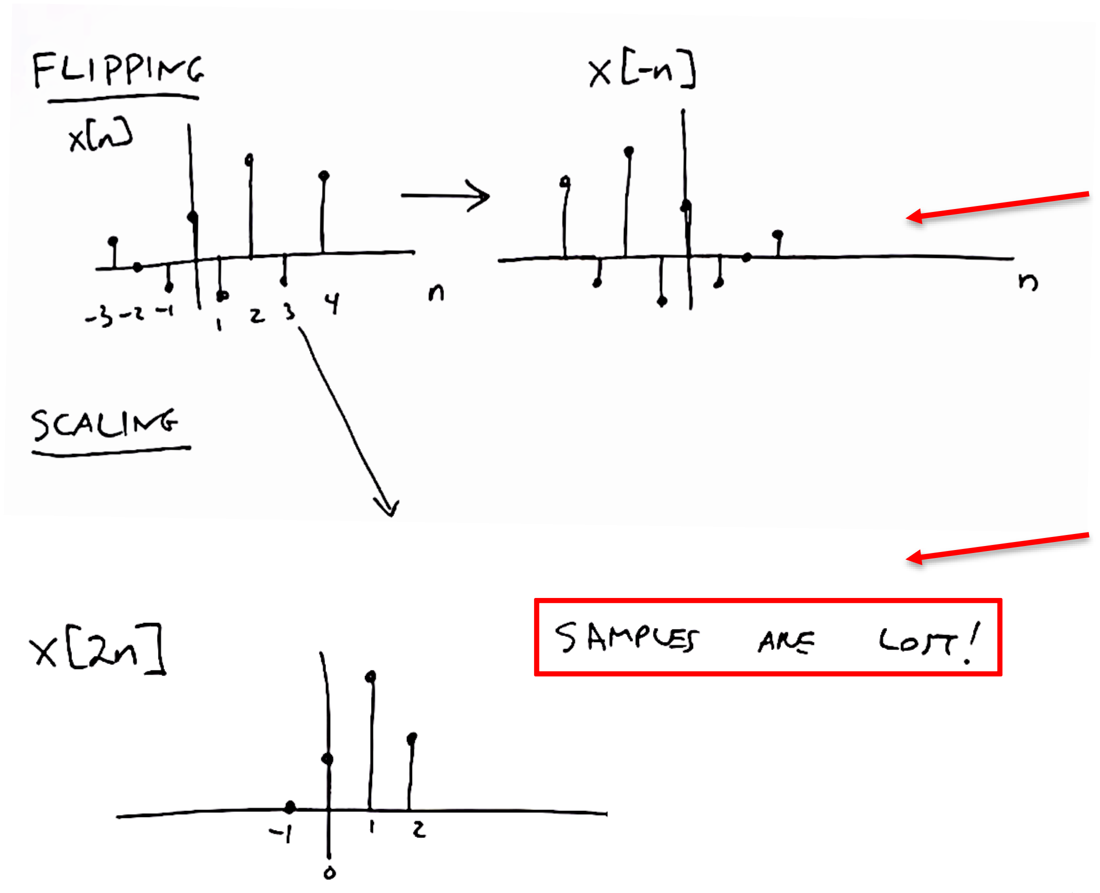
Time Reversal:

- **Effect:** This transformation reverses the order of the samples in the signal. In the time domain, it creates a mirror image of the signal along the vertical (**y**) axis.
- **Mathematical representation:** **$y(n) = x(-n)$**, where **$x(n)$** is the original signal and **$y(n)$** is the time-reversed signal.

- In this course, we will use the term ***flipping as synonymous to time reversal***, and it is defined as **$y(n) = x(-n)$**.

Flipping/time reversal and Scaling

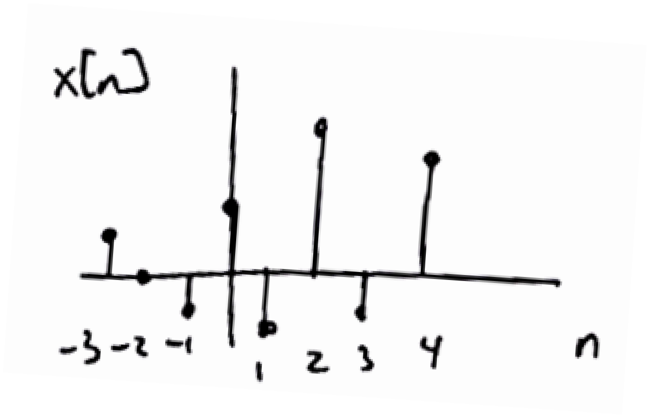
- What are the kinds of basic operations that we can do to a signal (**Signal Processing**)?



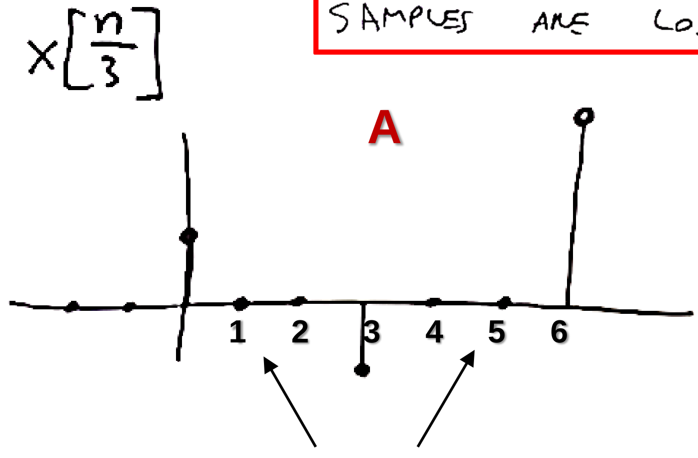
SAMPLES ARE LOST!

- **Flipping:** All that means is we are flipping the samples around the **y**-axis.
- **Scaling x[2n]:** If we imagine this is an audio signal, then we are playing the signal twice as fast. That means that the overall duration of the signal should decrease by a factor of two. Put differently, we are taking every other sample. One thing that is definitely true in DSP that was not true in continuous time signal processing is that we lose information. So after we do this scaling, we have thrown away the samples in between the ones that we had, and we might never get them back.

Flipping/time reversal and Scaling



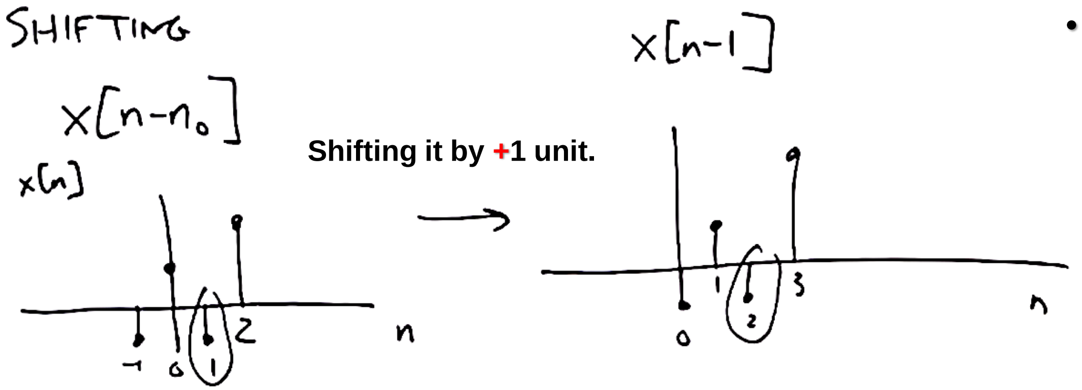
SAMPLES ARE LOST!



middle locations

- **Scaling x[n/3]:** We are slowing down the signal by a factor of **3**. It means that it should get broader. On the **x**-axis, what used to be **1** becomes **3**. And what used to be **2** becomes **6**. If we do not do anything else, our expanded signal looks like graph **A**. That is, we now do not have anything else to put in the ***middle locations*** because we do not have any samples to grab from.
- We can use the ***sampling theorem*** to solve this kind of problem.

Shifting



Shifting it by +1 unit.

$y[n] = x[n-1]$

$y[2] = x[1]$

- You can always confirm the values in the shifted graph by plugging in for the value of **n**.

- **Shifting:** If we are shifting a signal by some amount, it has to be by an ***integral amount*** (i.e., an ***integer number***).

x[n]: The original signal

y[n] = x[n-1]: The transformed signal

- The way we can think about $\mathbf{n_0}$ (here, it is **+1**) is that this number tells us how we are ***delaying*** the signal. This is like saying the signal should start one unit later.

Combining transformations; order of operations

$x[-2n+3]$ **(1)**

SHIFT, FLIP, SCALE

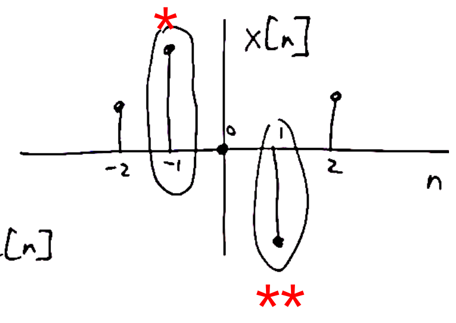
$z[n] = x[n+3]$ **(2)**

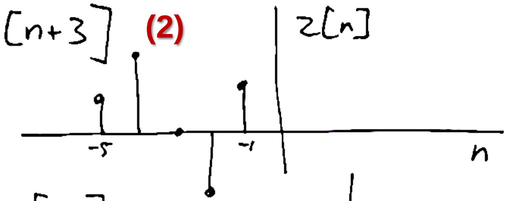
$w[n] = z[-n]$ **(3)**

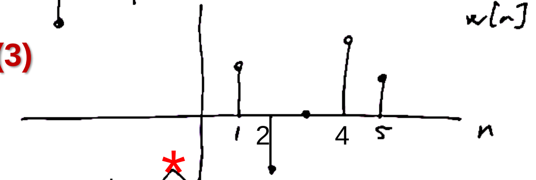
$y[n] = w[2n]$ **(4)**

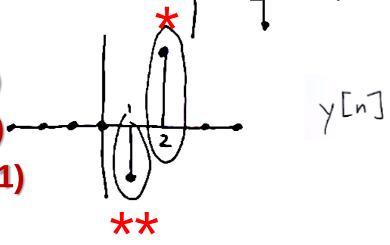
$= z[-2n]$ **(5)**

$= x[-2n+3]$ **(1)**









Checking:

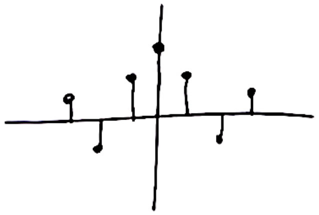
$x[-2n+3] \rightarrow$ $y[2] = x[-1]$ *

$y[1] = x[1]$ **

- We can combine all the stuff together.
- We have to ***do these steps in a certain order***, otherwise we might get the wrong answer. The order is Shift, Flip, and Scale.
- Equation **(2)** is the shifting operation. In the left hand side of **(2)**, we plug in "**-n**" for "**n**", and we get **(3)**, which is the flipping operation. In the left hand of **(3)**, we plug in "**2n**" for "**n**", and we get **(4)**, which is the scaling operation. Here is what wanted, i.e., **y[n]** or **w[2n]**.
- To check, in the left hand side of **(3)**, we plug in "**2n**" for "**n**" in **w[n]**, and we get **z[-2n]** from the right hand side or **(5)**. In the left hand side of **(2)**, we plug in "**-2n**" for "**n**" in **z[n]**, and from the right hand side we get **x[-2n+3]** or **(1)**.
- For **w[2n]**, we go from **1** to **5**. Then we are going to scale that by **2**, which means we are going to compress it. So, zero stays zero. What used to be **2** becomes **1**. What used to be **4** becomes **2** and everything else goes away.

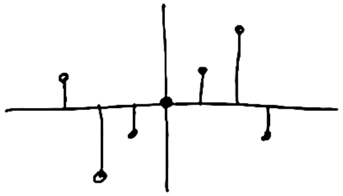
Signal properties; Even and odd

EVEN: $x[n] = x[-n]$



ODD $x[n] = -x[-n]$

$x[0] = -x[0] = 0$



EVERY SIGNAL HAS EVEN AND ODD PARTS

$$Ev(x[n]) = \frac{1}{2}(x[n] + x[-n])$$

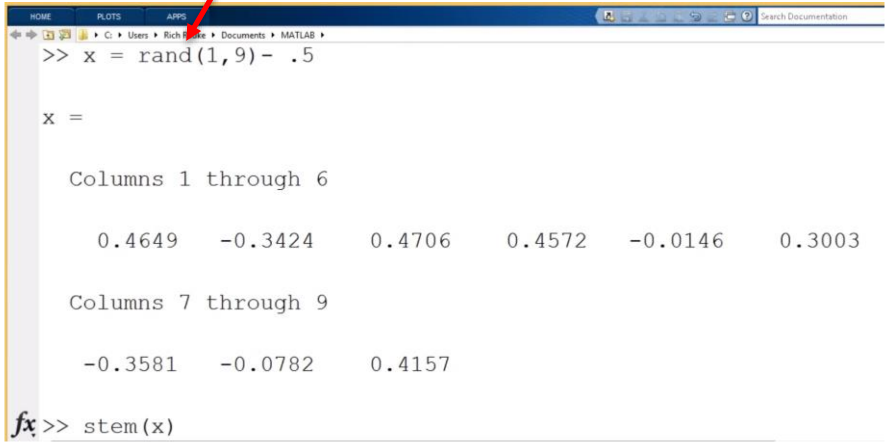
$$Od(x[n]) = \frac{1}{2}(x[n] - x[-n])$$

- **Even Signal:** A signal is even if it looks the same as itself flipped around the **y**-axis.
- **Odd Signal:** A signal is odd if it is looks the negative itself flipped across the **y**-axis (anti-symmetric). By definition, if we plug in **0**, the equation tells us that the middle entry of an odd signal always has to be **0** and then it is a mirror image of itself around the **y**-axis.
- The reason we care about even and odd signals will be important when we get to things like understanding the **Fourier transform** and how the real and imaginary parts of the Fourier transform are related in signals.

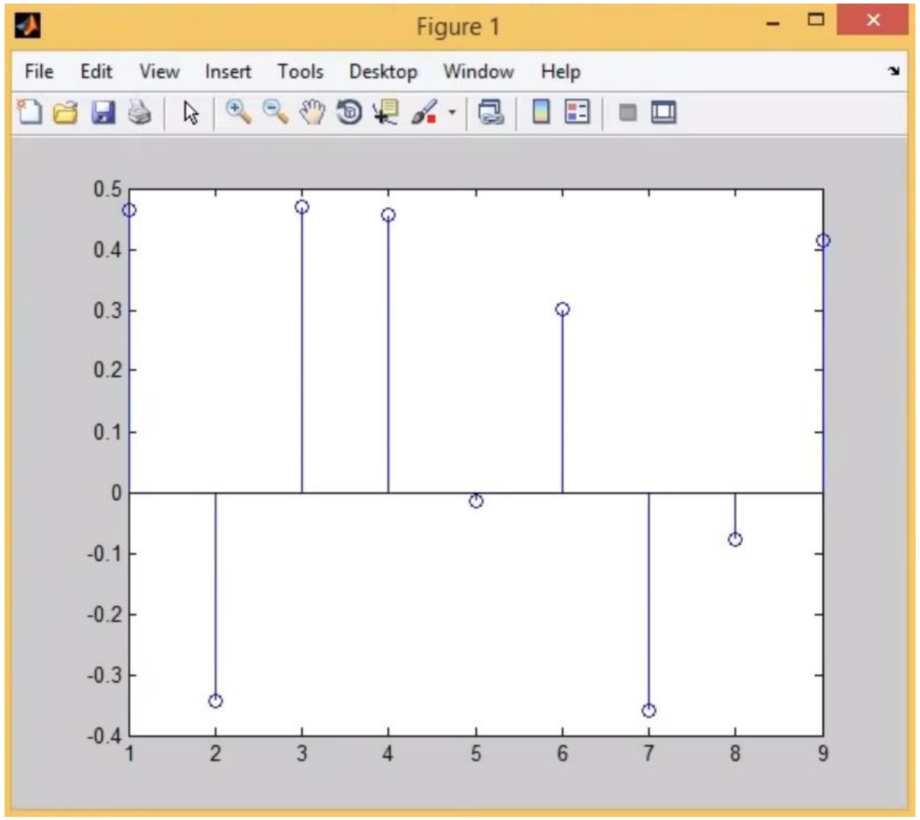
Decomposing a signal into even and odd parts (with MATLAB demo)

- An example of even and odd signal in **MATLAB**.

random signal



```
>> x = rand(1,9)- .5

x =

  Columns 1 through 6

    0.4649   -0.3424    0.4706    0.4572   -0.0146    0.3003

  Columns 7 through 9

   -0.3581   -0.0782    0.4157

fx >> stem(x)
```



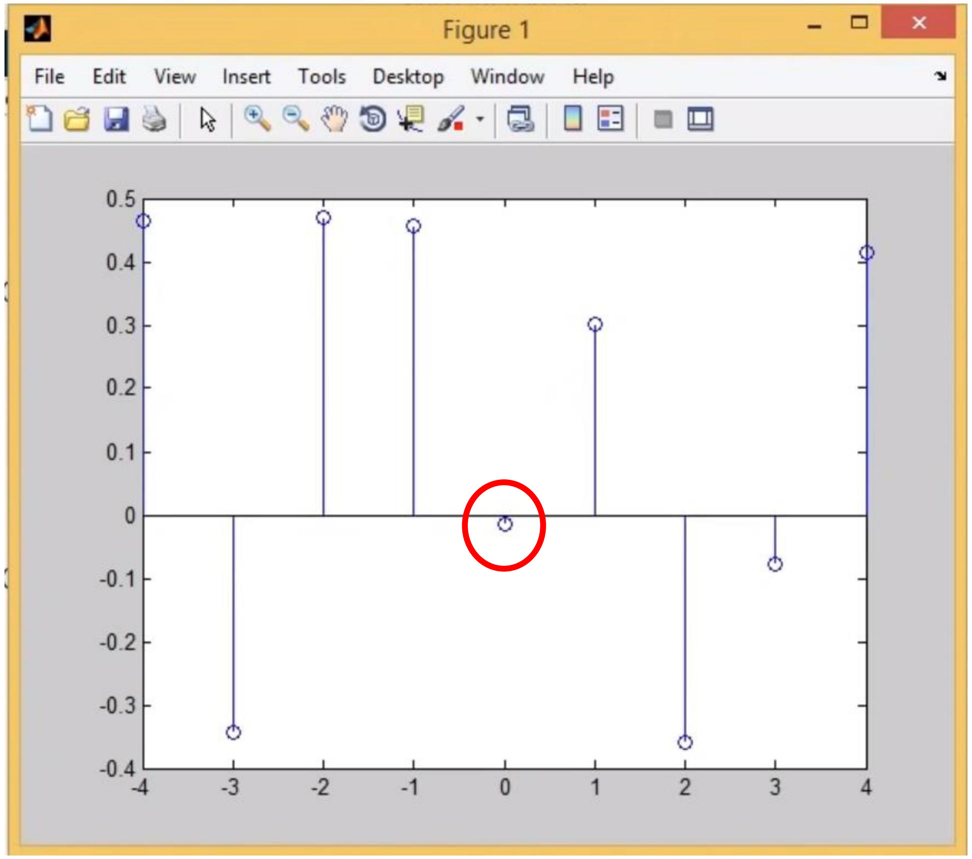
- **Stem (x)**: This is a built-in function in MATLAB used to create stem plots, which are a specific type of plot used to visualize discrete-time signals.

- This signal is neither even nor odd.

Decomposing a signal into even and odd parts (with MATLAB demo)

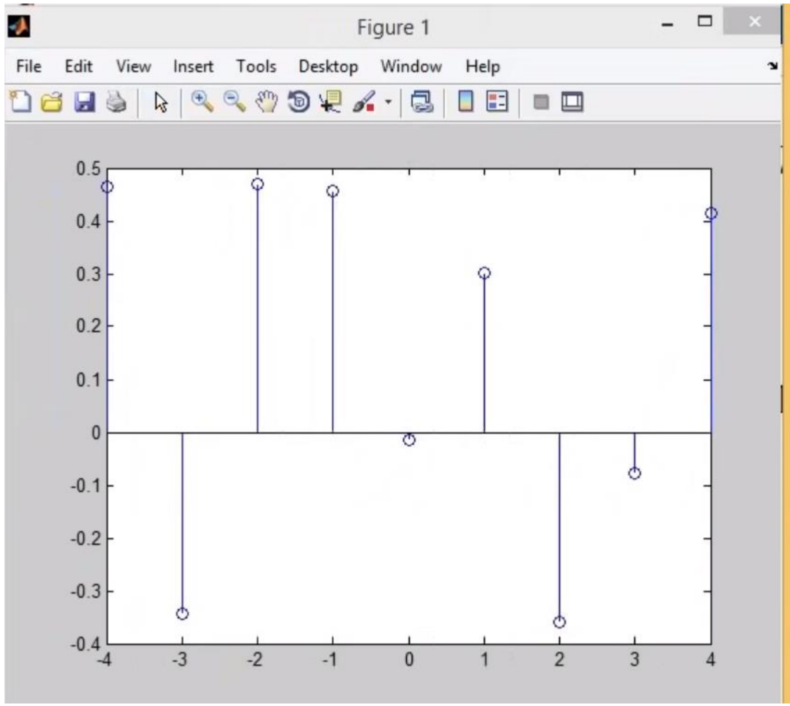
```
>> stem([-4:4],x)
```

- By using the above command, we make the middle value zero.

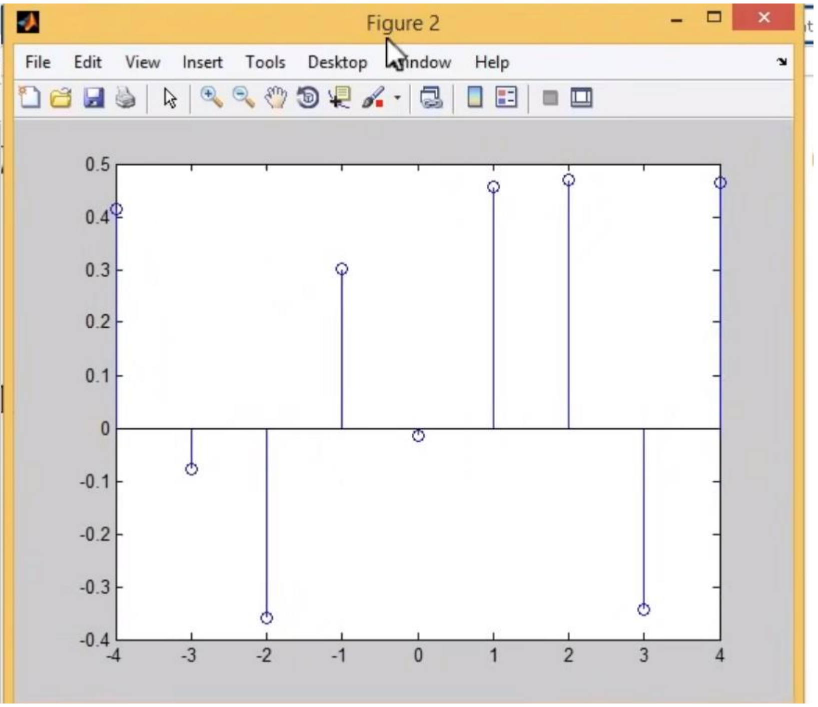


Decomposing a signal into even and odd parts (with MATLAB demo)

```
>> stem(x)
>> stem([-4:4],x)
>> negx = fliplr(x);
>> figure(2)
>> stem([-4:4], negx)
```



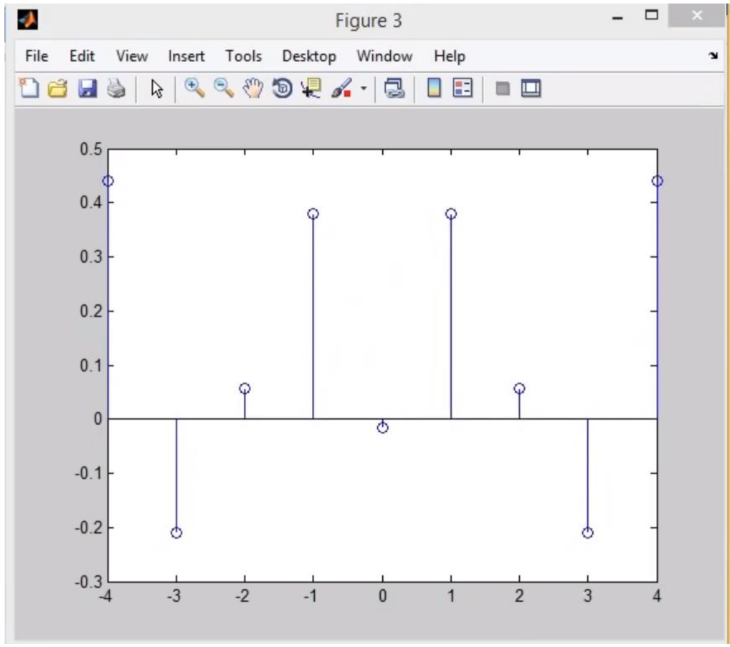
Original signal (between -4 to 4)



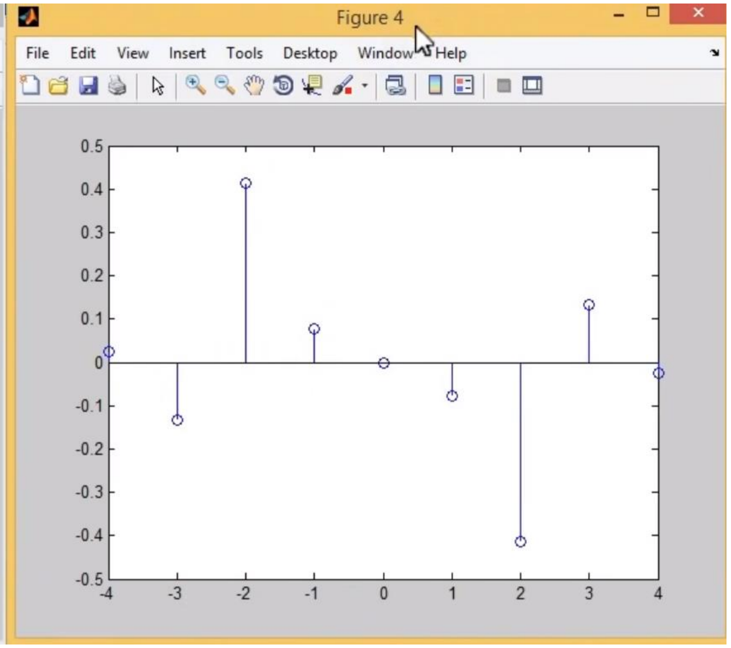
Flipped signal

Decomposing a signal into even and odd parts (with MATLAB demo)

```
>> evx = (x + negx)/2;
>> odx = (x - negx)/2;
>> stem([-4:4], evx);

>> stem([-4:4], odx);
```



Even part of x

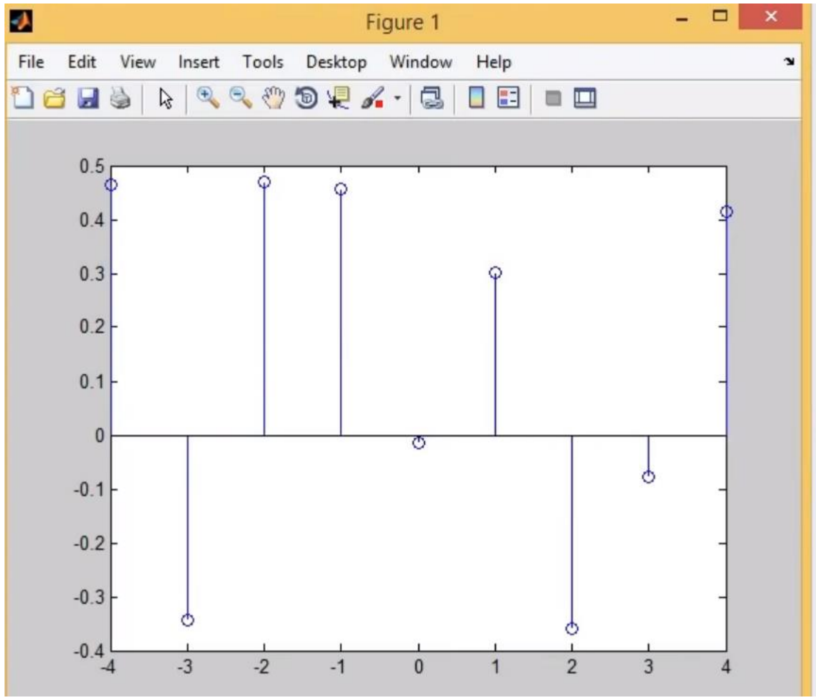


Odd part of x

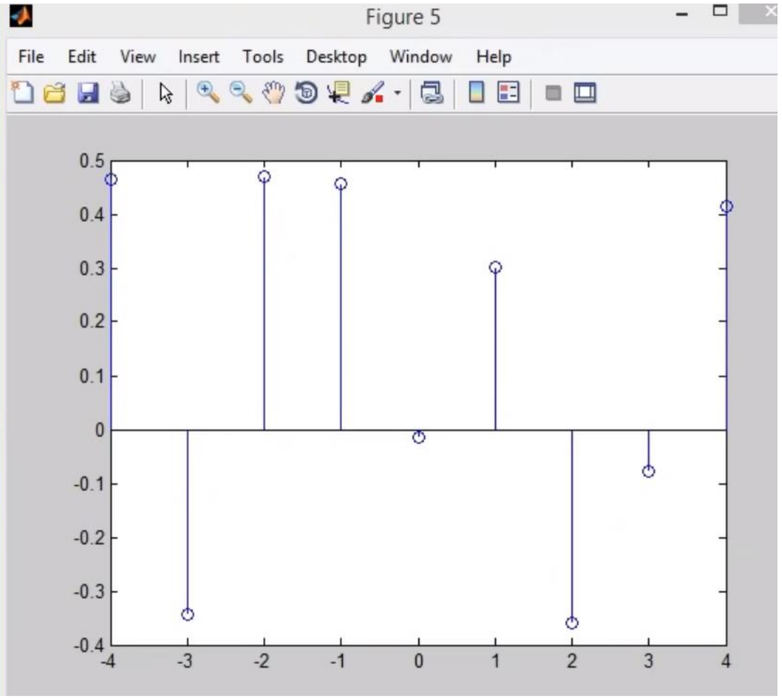
Decomposing a signal into even and odd parts (with MATLAB demo)

```
>> q = evx + odx;
>> figure(5)
>> stem([-4:4], q);
```

- **Checking:** When we add the even part and the odd part of the signal back together, we get the original signal.



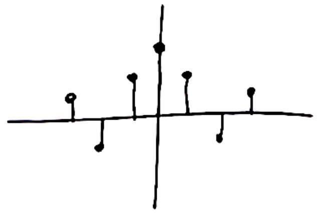
Original signal



Checking by adding evx to odx

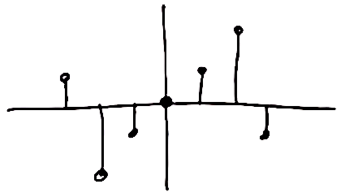
Decomposing a signal into even and odd parts (with MATLAB demo)

EVEN: $x[n] = x[-n]$



ODD $x[n] = -x[-n]$

$x[0] = -x[0] = 0$

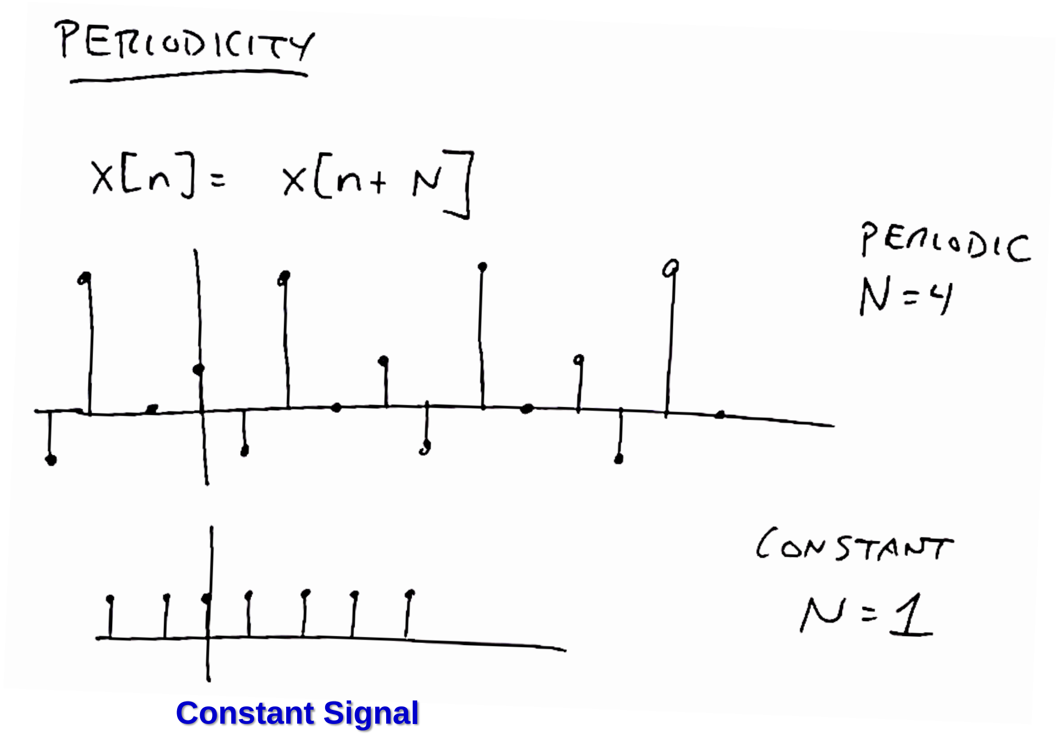


EVERY SIGNAL HAS EVEN AND ODD PARTS

$$\left.\begin{aligned} Ev(x[n]) &= \tfrac{1}{2}\left(x[n] + x[-n]\right) \\ Od(x[n]) &= \tfrac{1}{2}\left(x[n] - x[-n]\right) \end{aligned}\right\}$$

- **Note:** We see that when we add **Ev(x[n])** to **Od(x[n])**, the **+x[-n]** and **-x[-n]** parts are going to cancel out and eventually add up to just the **x[n]** again.

Periodicity



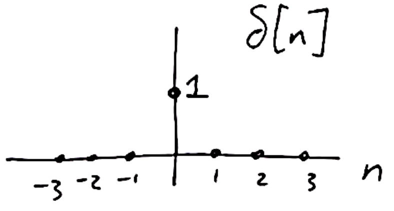
Constant Signal

- **Periodicity:** It means that the signal repeats itself after a certain number of integer steps.
- A **constant signal** is also periodic. It is a single value all the time. So, by its nature, it has a period of **1**.
- Periodic signals are going to be very important for us, largely because we are going to be dealing with different kinds of discrete-time sines and cosines.

Special signals; The delta function; The unit step function

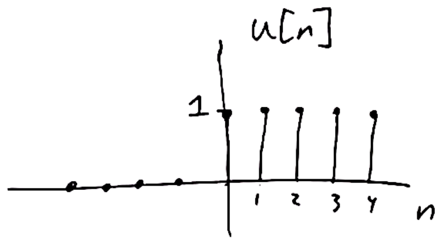
SPECIAL SIGNALS

$$\delta[n] = \begin{cases} 1 & n=0 \\ 0 & n\neq 0 \end{cases}$$



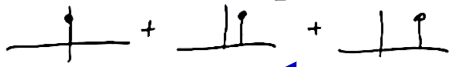
Delta Function

$$u[n] = \begin{cases} 1 & n\geq 0 \\ 0 & n<0 \end{cases}$$



Unit Step Function

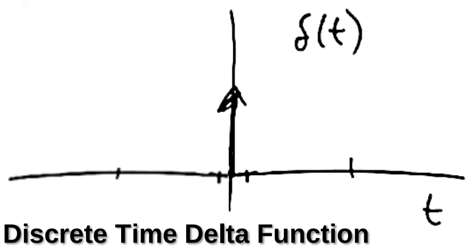
$$u[n] = \delta[n] + \delta[n-1] + \delta[n-2] + \dots = \sum_{k=0}^{\infty} \delta[n-k]$$



- The unit step function and the delta function are related to each other.
- The unit step function can be thought of to be made up of a bunch of delta functions.
- That is like taking apart the signal into just its **stick pieces**.

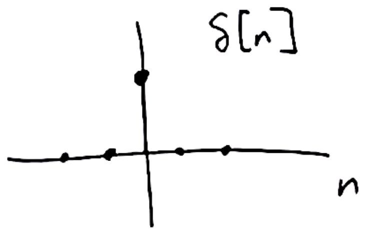
The relationship between the delta and step functions

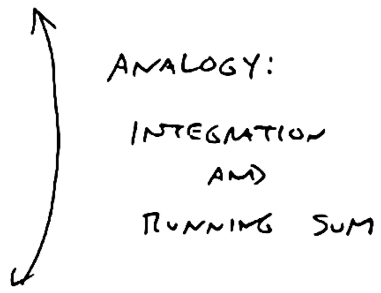
Continuous Time Delta Function



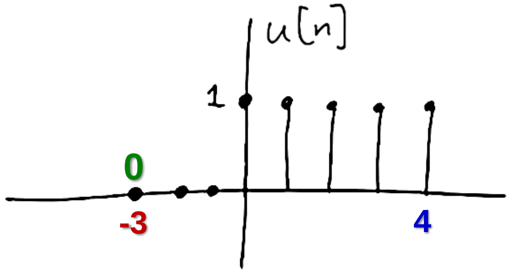
$$\int_{\tau=-\infty}^{t} \delta(t)\,d\tau = u(t)$$

Discrete Time Delta Function





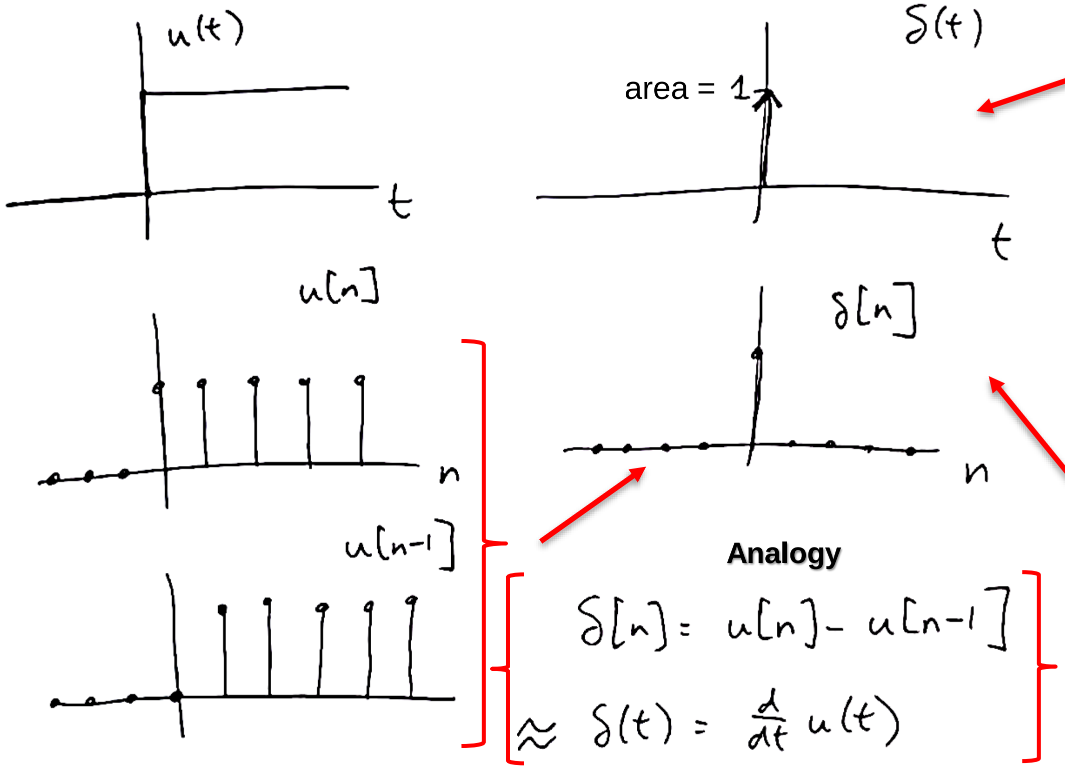
$$u[n] = \sum_{k=-\infty}^{n} \delta[k]$$



- **What is the value of the step function at, say, -3?** We plug in **n = -3** in the infinite sum. We add up all the values of the delta function from minus infinity up to **-3**. After we add up those values, we get **0**. So, nothing has happened yet. We can put down **0** in the **u[n]** graph at **-3**. From negative infinity to **0**, suddenly we add this one unit of sum and we jump up to **1**. For **n = 4**, we add everything up for the left hand side all the way up to **4**. We have only got that one unit. So, after we bump over the **1** piece in the middle, we never accumulate anything else.

- In continuous time, we got the step function by integrating the delta function up to some point. The same thing is true in discrete time. But, we cannot integrate in discrete time. The only thing we can do is to add up the terms. This is the ***analogy*** between integration and what we can call a **running sum**.

The relationship between the delta and step functions



Analogy

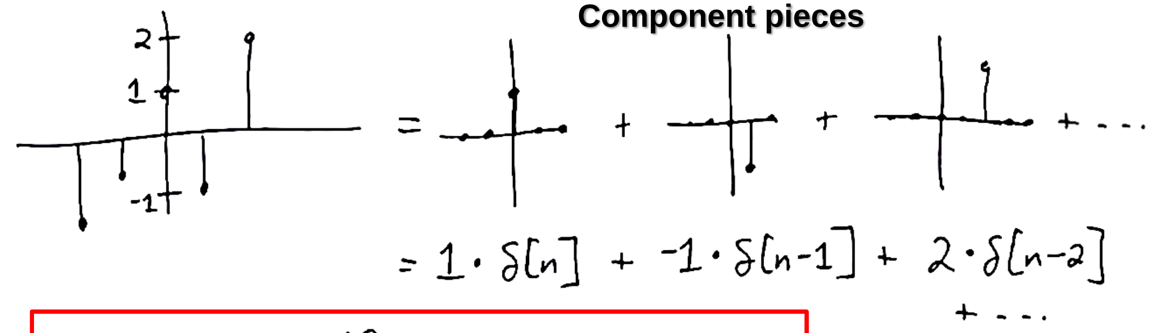
$$\delta[n] = u[n] - u[n-1]$$

$$\approx \delta(t) = \frac{d}{dt} u(t)$$

- In continuous world, by taking the **derivative** of the step function, we get the delta function. The derivative almost everywhere is **0**. Wherever this function is flat, we get a **0** derivative. The only place we do not get a **0** derivative is where the function jumps up to **1** and that is why this delta function jumps up to **1**. Technically, the derivative of this function at **0** is not well defined. That is, it is infinity. The slope is infinitely high and that is why we have the **arrow** instead of an actual dot.
- To obtain the the digital version of the delta function, we do the discrete time version of differentiation. That is like taking a **difference**. We take a signal and we shift it a little bit and we subtract them and then we make that shift smaller and smaller.

Decomposing a signal into delta functions

MAKING A SIGNAL OUT OF DELTAS:



$$= 1 \cdot \delta[n] + -1 \cdot \delta[n-1] + 2 \cdot \delta[n-2] + \dots$$

$$\boxed{x[n] = \sum_{k=-\infty}^{\infty} x[k]\,\delta[n-k]}$$

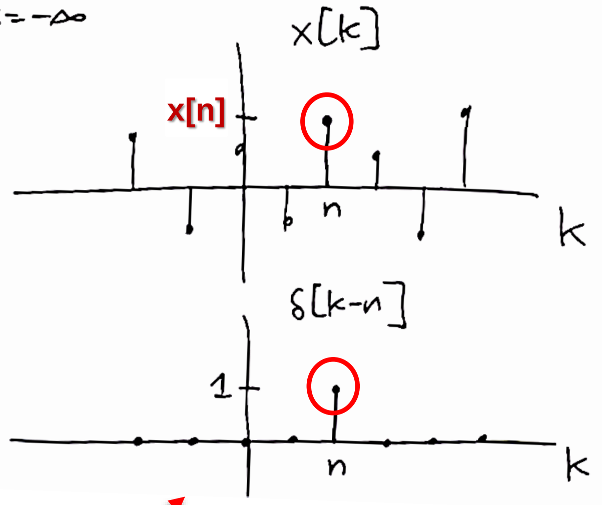
- Just like we made the step function up out of delta function pieces, ***we can do that with any signal***. That is, we can make any signal out of delta function.
- Each of the **component pieces** is like a scaled and shifted delta function.

- This equation means that at every position (or at every value of **k**), we multiply the delta function by the corresponding value of **x** and then we add all those terms up to find **x[n]**.
- We are going to use the above formalism a lot when we come to things like **convolution**.

The sampling property of delta functions

SAMPLING PROPERTY:

$$\sum_{k=-\infty}^{\infty} x[k]\,\delta[k-n] = x[n] \quad \text{(1)}$$



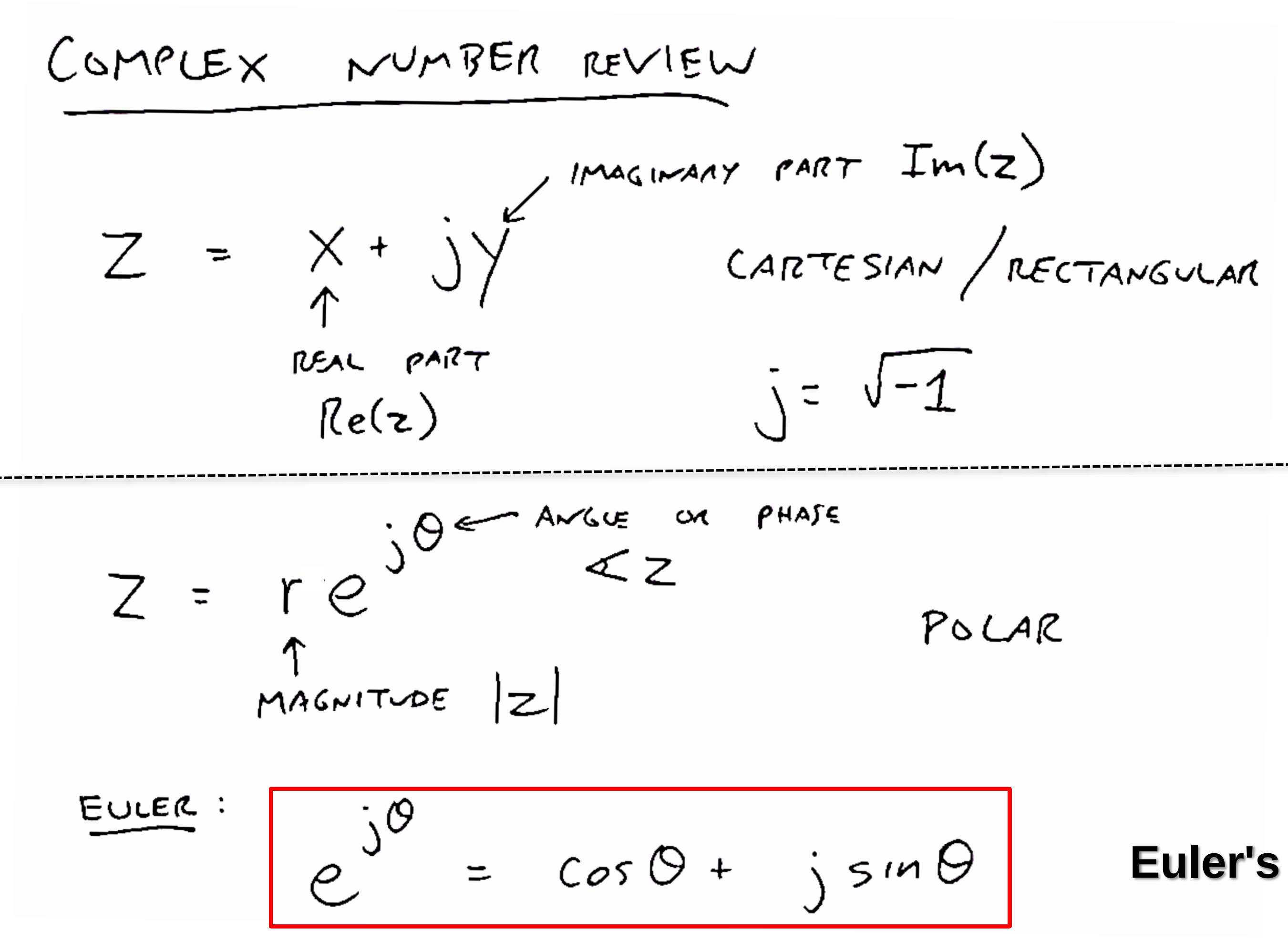
- This is a delta function that is shifted or delayed to start at **n**.

- Let us see what **(1)** means. In **A**, we have our signal, **x[k]** vs. **k**. In **B**, we have got a delta function that is shifted or delayed to start at **n**.

- **What do we get when we multiply these two signals together and add them all up?** We are going to be multiplying a lot of zeros, which simply leads to a lot of zeros. The only time we are not going to get a non-zero answer is at the point where the delta function is firing, at **k = n**. Here, we get whatever **x[n]** value is at **k = n**, shown by a **red** circle in graph **A**, multiplied by the value **1**, also shown by a **red** circle, at **k = n** in graph **B**. The result will be **1.x[n]**, or **x[n]**, as shown on the right hand side of equation **(1)**.

- **Conclusion:** We can use the delta function to **pick off** any value of our signal that we want (i.e., **x[n]**) by multiplying the delta function with **x[k]**.

Complex number review (magnitude, phase, Euler's formula)



COMPLEX NUMBER REVIEW

IMAGINARY PART $Im(z)$

$$Z = \underset{\substack{\uparrow \\ \text{REAL PART} \\ Re(z)}}{x} + jy$$

CARTESIAN / RECTANGULAR

$$j = \sqrt{-1}$$

ANGLE OR PHASE

$\measuredangle Z$

$$Z = \underset{\substack{\uparrow \\ \text{MAGNITUDE } |z|}}{r}\, e^{j\theta}$$

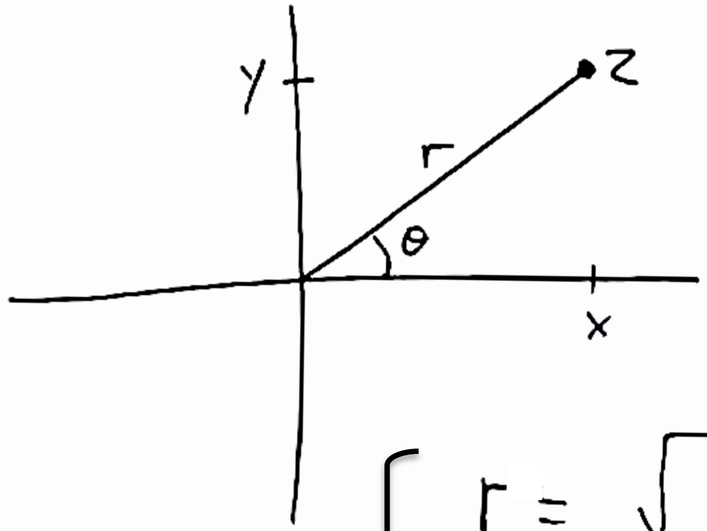
POLAR

EULER :

$$\boxed{e^{j\theta} = \cos\theta + j\sin\theta}$$

Euler's formula

Complex number review (magnitude, phase, Euler's formula)



$x + jy$

$re^{j\theta}$

$$\begin{cases} r = \sqrt{x^2 + y^2} \\ \theta = \tan^{-1}(y/x) \end{cases}$$

Radius

Angle

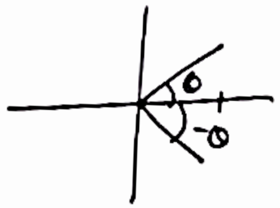
$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases}$$

Complex number review (magnitude, phase, Euler's formula)

$$\cos\theta = \frac{1}{2}\left(e^{j\theta} + e^{-j\theta}\right)$$

$$= \frac{1}{2}\left(\cos\theta + j\sin\theta + \cos(-\theta) + j\sin(-\theta)\right)$$

$$= \frac{1}{2}\left(2\cos\theta\right) = \cos\theta$$



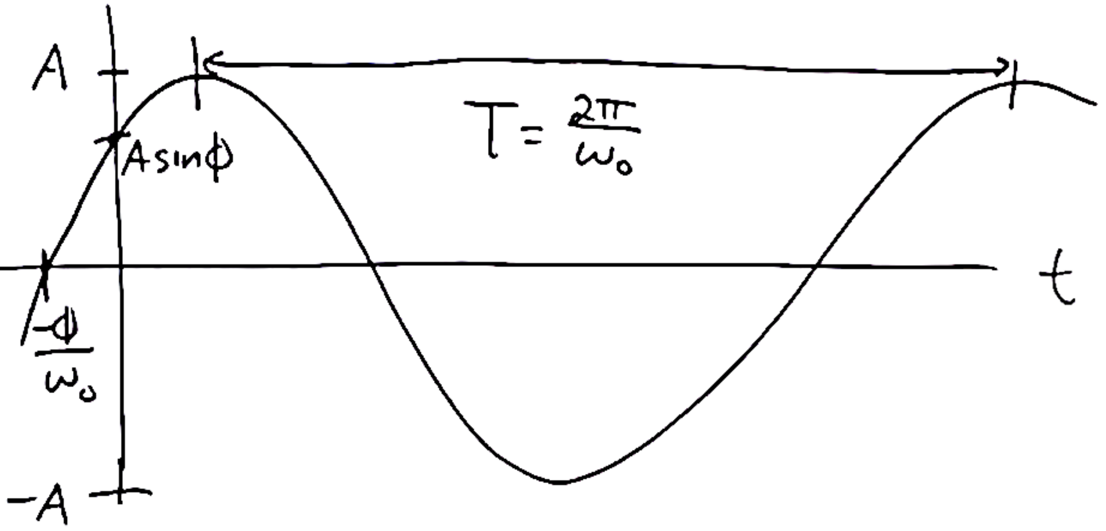
$$\sin\theta = \frac{1}{2j}\left(e^{j\theta} - e^{-j\theta}\right)$$

Real sinusoids (amplitude, frequency, phase)

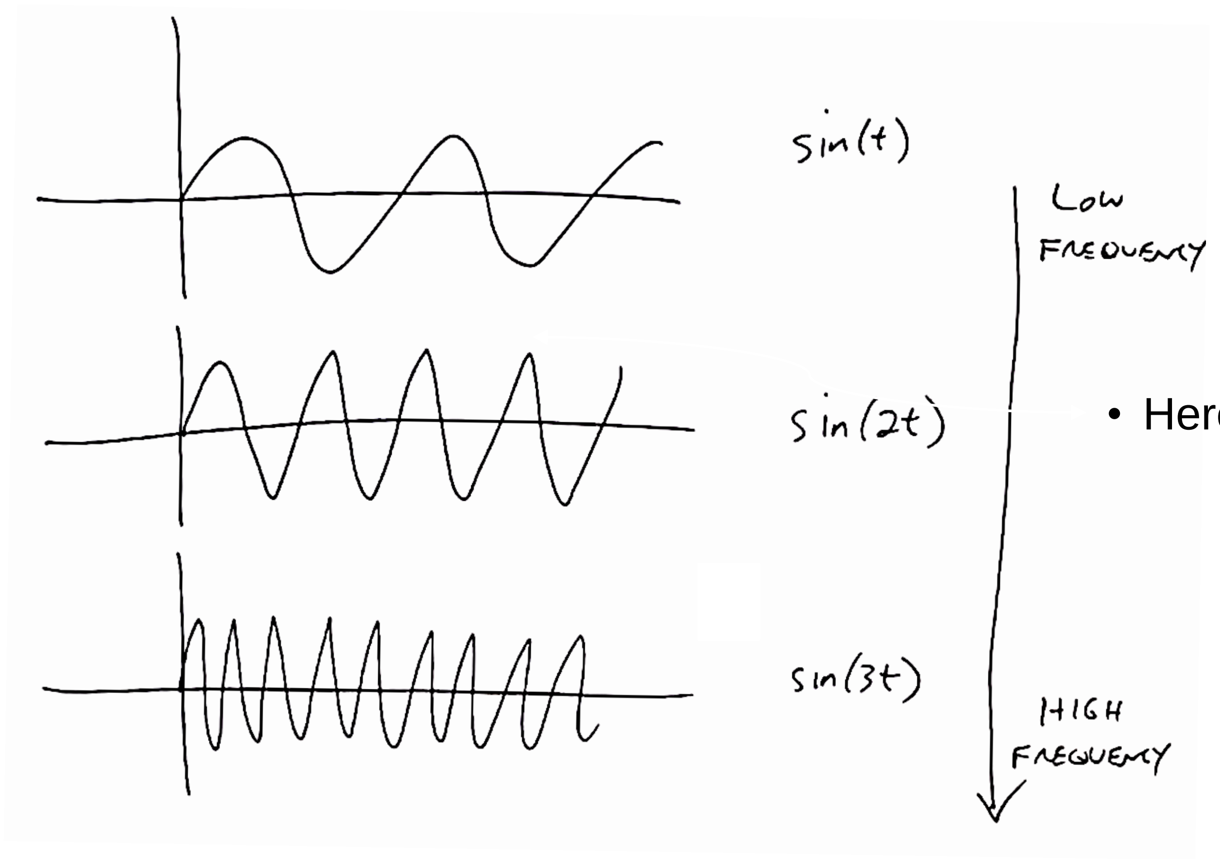
SINUSOIDS

AMPLITUDE FREQUENCY PHASE

$$x(t) = A \sin(\omega_0 t + \phi)$$

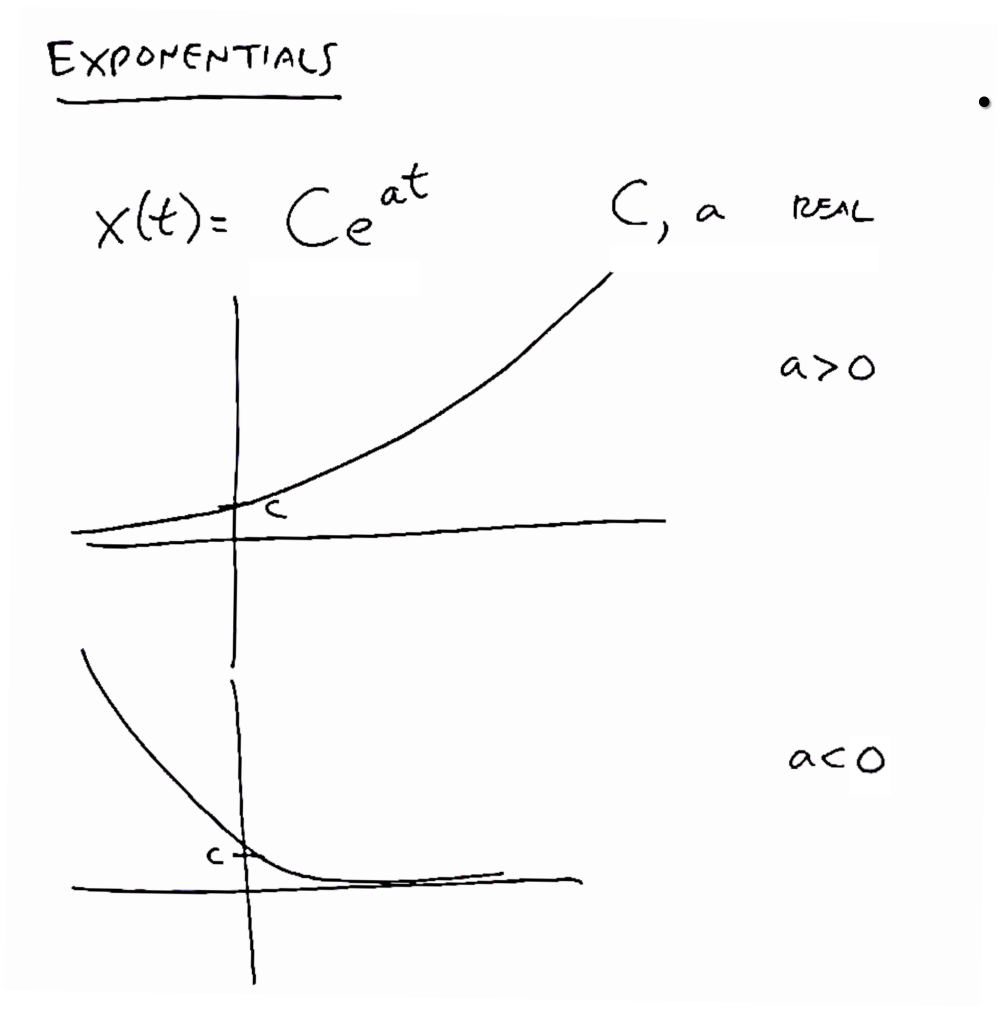


Real sinusoids (amplitude, frequency, phase)



- Here, **sin(2t)** wiggles twice as fast.

Real exponential signals



- **Exponentials** are functions that grow or shrink in a certain way.

Complex exponential signals

WHAT ABOUT $x(t) = Ce^{at}$ WHEN

C AND a ARE COMPLEX?

$(2+3j)\, e^{(-1-2j)t}$?

$C = |C| e^{j\theta}$ POLAR

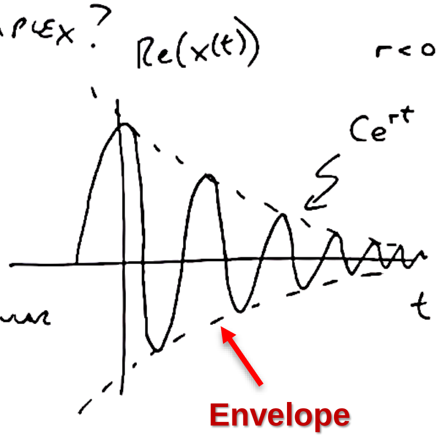
$a = r + j\omega_0$ RECTANGULAR

$$x(t) = |C| e^{j\theta} e^{(r+j\omega_0)t}$$

$$= |C| e^{rt} e^{j(\omega_0 t + \theta)} \longrightarrow$$

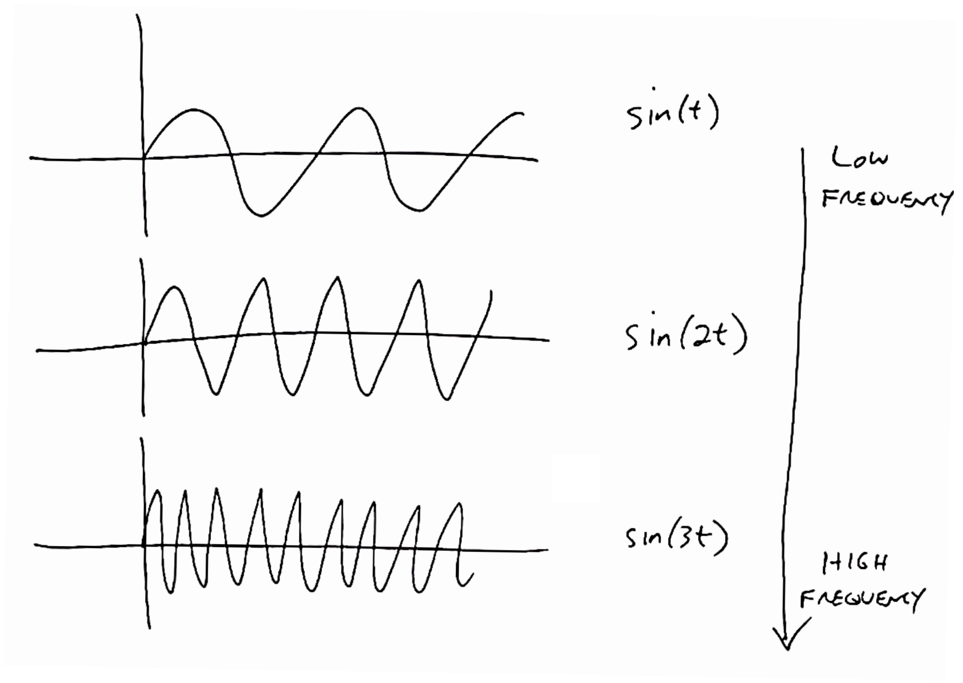
$$x(t) = \underline{\underline{|C| e^{rt}}} \left[\cos(\omega_0 t + \theta) + j \sin(\omega_0 t + \theta) \right] \quad \textbf{(1)}$$

Envelope



- The best way to show **x(t)** graphically is to convert **C** from Cartesian into polar coordinates.
- The real part of the signal is some cosine that is oscillating within the **envelope**.
- If **r = 0**, then we do not have any oscillation at all, just a regular cosine or sine.
- If **r** is negative or positive, then we either have an expanding cosine or contracting cosine.
- So, we have a complex sinusoid, $\mathbf{e^{at}}$, modulated by the exponential function, $\mathbf{|C|\, e^{rt}}$. But $\mathbf{e^{at}}$ has a notion of a frequency ($\boldsymbol{\omega_0}$) in it, as shown in **(1)**.
- We cannot really say this function is periodic, because it does not strictly repeat itself (it is changing its amplitude). But there is this sense of a period inside the envelope.

Complex exponential signals



- Let us look at the **periodicity** in the continuous time world one more time.
- Here (on the left graph), we have sines (or cosines). We can just keep on increasing the frequency. There is nothing to prevent us from continuing to oscillate faster and faster. We can talk about cosine of 40,000 or 400,000 Hertz.
- But that is not true in the discrete time world. The discrete time sinusoids are a little bit different.

Complex exponential signals

IN A SIMILAR WAY,

$$x[n] = C\alpha^n = Ce^{\beta n} \; ; \quad \alpha = e^{\beta}$$

$$= |C||\alpha|^n\left(\cos(\omega_0 n+\phi) + j\sin(\omega_0 n+\phi)\right)$$

DISCRETE TIME SINUSOIDS ARE DIFFERENT:

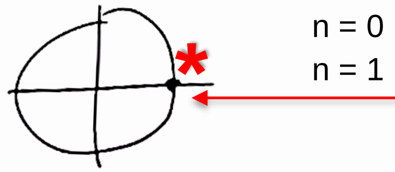
$e^{j\omega_0 n}$ WHAT IS $e^{j(\omega_0+2\pi)n}$?

$$= e^{j\omega_0 n + j2\pi n} =$$

$$= \overset{1}{\cancel{e^{j2\pi n}}}\, e^{j\omega_0 n}$$

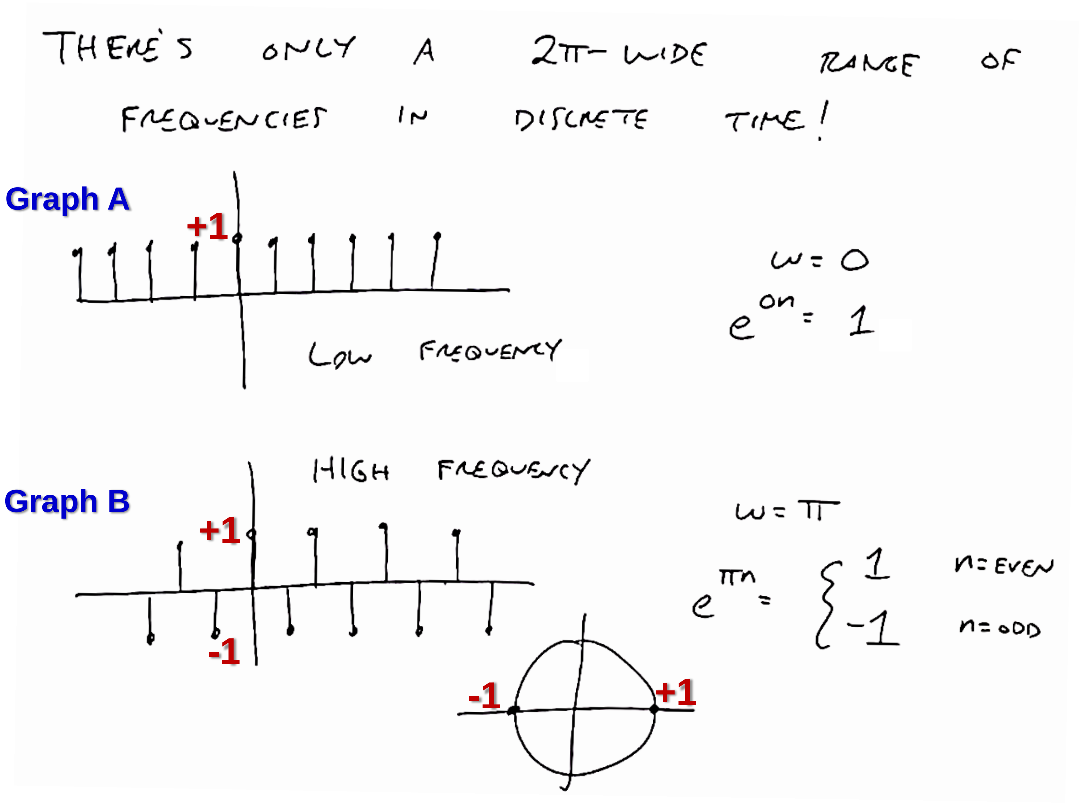
$$= e^{j\omega_0 n}$$

n IS AN INTEGER



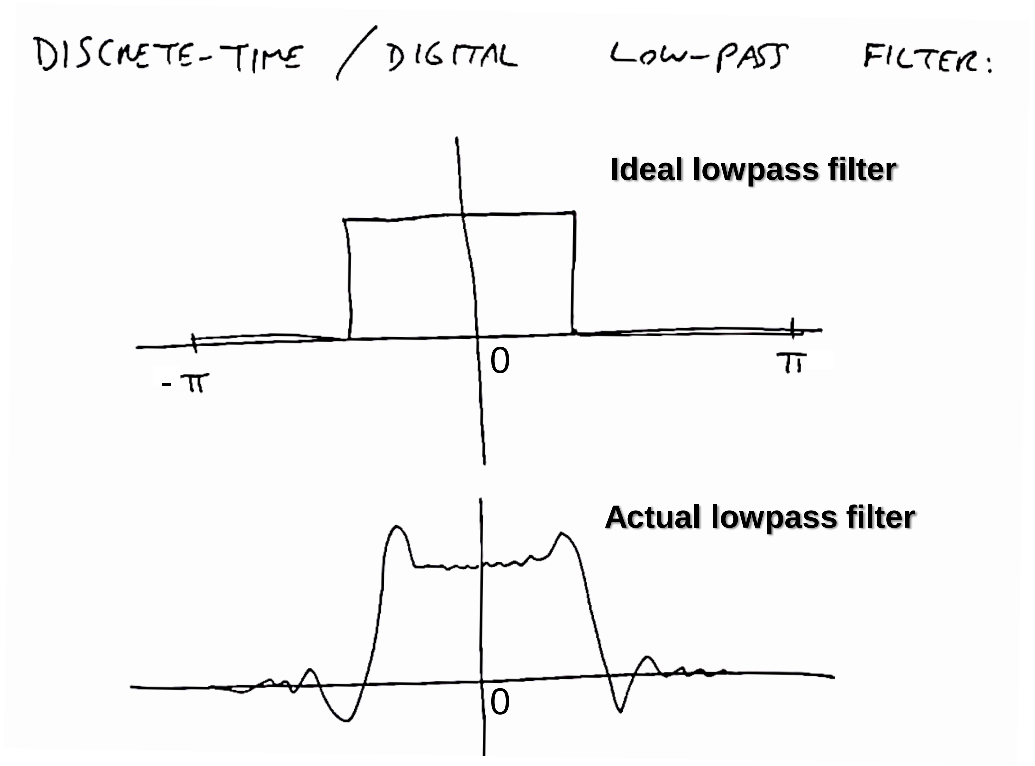
- In theory, in the continuous-time world, by adding a value, such as **2π** to **ω_0**, we would make the signal faster. That is, it would lead to a higher frequency sinusoid.
- In the discrete-time world, if **n = 0**, then **exp(j2nπ) = exp(j2π×0) = 1** and that puts us on point * on the circle. If **n = 1**, that means we come around and we have gotten one full circle. We get **exp(j2π×1) = 1** (i.e., that is also **1**). If **n = 2**, we are going around twice. That is also **1**. So, **exp(j2nπ) is always 1**.
- **Conclusion:** In summary, when we add **2π** to any frequency, we end up getting the same frequency again. There is no infinitely high frequencies in discrete-time world. We can only go up to a certain amount.

Discrete-time sinusoids are 2π-periodic


THERE'S ONLY A 2π-WIDE RANGE OF FREQUENCIES IN DISCRETE TIME!
Graph A
+1
LOW FREQUENCY
ω = 0
e^{0n} = 1
Graph B
HIGH FREQUENCY
+1
-1
ω = π
e^{πn} = { 1 n=EVEN; -1 n=ODD
-1
+1

- The **lowest frequency** we can get is just a constant (**Graph A**). That is, for **ω = 0**, the height of the signal is constant.
- What about the **highest frequency** that we could get? The highest frequency is something that basically is flipping back and forth as fast as we could possibly go (**Graph B**). This is about as fast as we can make a digital sinusoid go. It alternates between its maximum and minimum amplitude as quickly as possible. And this is corresponding to **ω = π**. So, the signal is bouncing back and forth between **+1** and **-1** points on the complex plane.
- In DSP, we no longer have to draw charts of frequency that go arbitrarily far out on the **x**-axis.

Discrete-time sinusoids are 2π-periodic



- There is a symmetry that basically will say for most of the filters that we care about, the frequency response is symmetric around zero.
- This means that we could have a filter which looks like the **ideal lowpass filter**.
- In practice, we are going to talk about how we can obtain this lowpass filter with limited length or finite impulse response filter. So, in practice, the filter that we design may look something that is more funky like the bottom graph (**actual lowpass filter**).
- Later on, we will talk about how we can make our actual designed filter frequency response look as close as possible to that of the ideal one.

When are complex sinusoids periodic?

WHEN IS $e^{j\omega_o n}$ PERIODIC (DISCRETE TIME)?

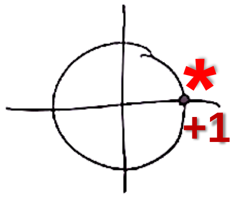
$e^{j\omega_o(n+N)} = e^{j\omega_o n}$ FOR SOME INTEGER N.

$\downarrow$

$e^{j\omega_o(n+N)} = e^{j\omega_o n} e^{j\omega_o N}$ $\longrightarrow$ $e^{j\omega_o N} = 1$

$\omega_o N = 2\pi k$ FOR INTEGER k. **(1)**

$\boxed{\omega_o = \frac{2\pi k}{N}}$ or $\boxed{N = \frac{2\pi k}{\omega_o}}$



- We are probably thinking that any cosine is periodic because that is what cosines do. That is not exactly true!
- Here, **exp(jω_oN)** should be equal to **+1**. That means that we need to land on point ***** on the complex plane. So, **ω_oN** has to be some even multiple of **π**, as shown in equation **(1)**. That means **ω_oN** has to be either **0**, **2π**, **4π** or some integer. All these points coincide with point *****.

When are complex sinusoids periodic?

Example:

FOR INTEGER k.

$$N = \frac{2\pi k}{\omega_0}$$

$$x[n] = \cos\left(\frac{4\pi}{5} n\right) \qquad N = \frac{2\pi k}{4\pi/5} = \frac{10\pi k}{4\pi} = \frac{5}{2}k$$

MUST BE INTEGER

$$k = 2, \quad N = 5$$

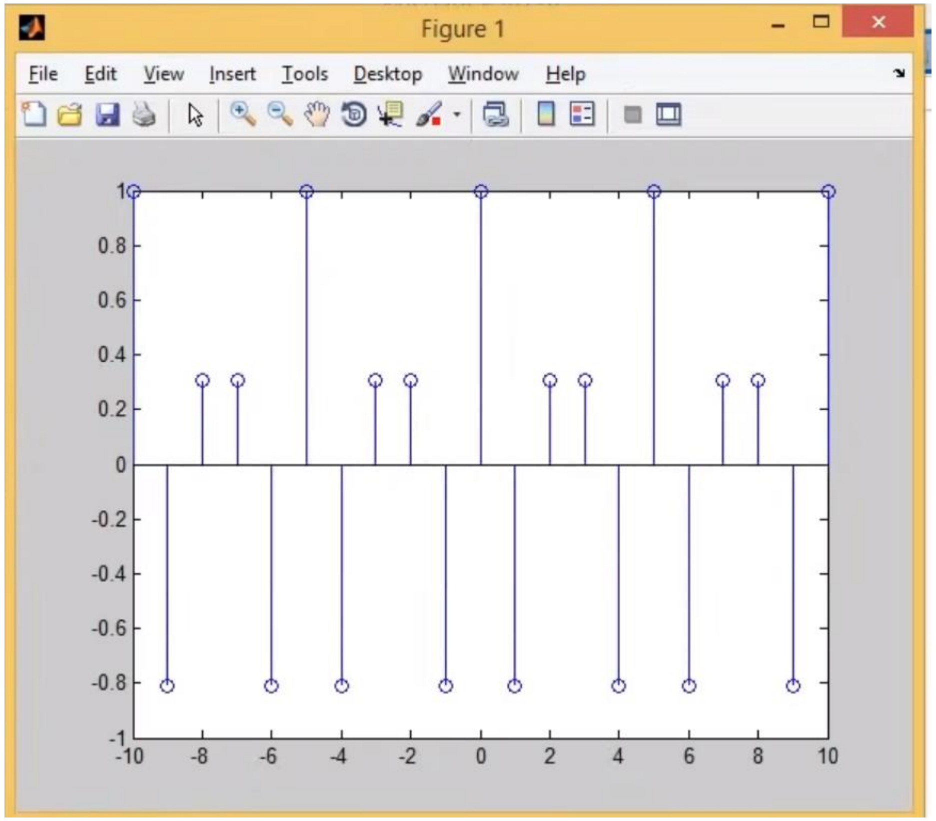
$$\boxed{\text{PERIODIC WITH PERIOD } 5}$$

- Is the given digital signal, **x[n]**, periodic? What is its period?
- Following the method shown, we can conclude that the signal is periodic with period **5** or **N = 5**.
- We will verify the above conclusion using MATLAB.

When are complex sinusoids periodic?

```
>> close all
>> n = [-10:10];
>> x = cos(4*pi/5*n);
>> stem(n,x)
```

Time domain



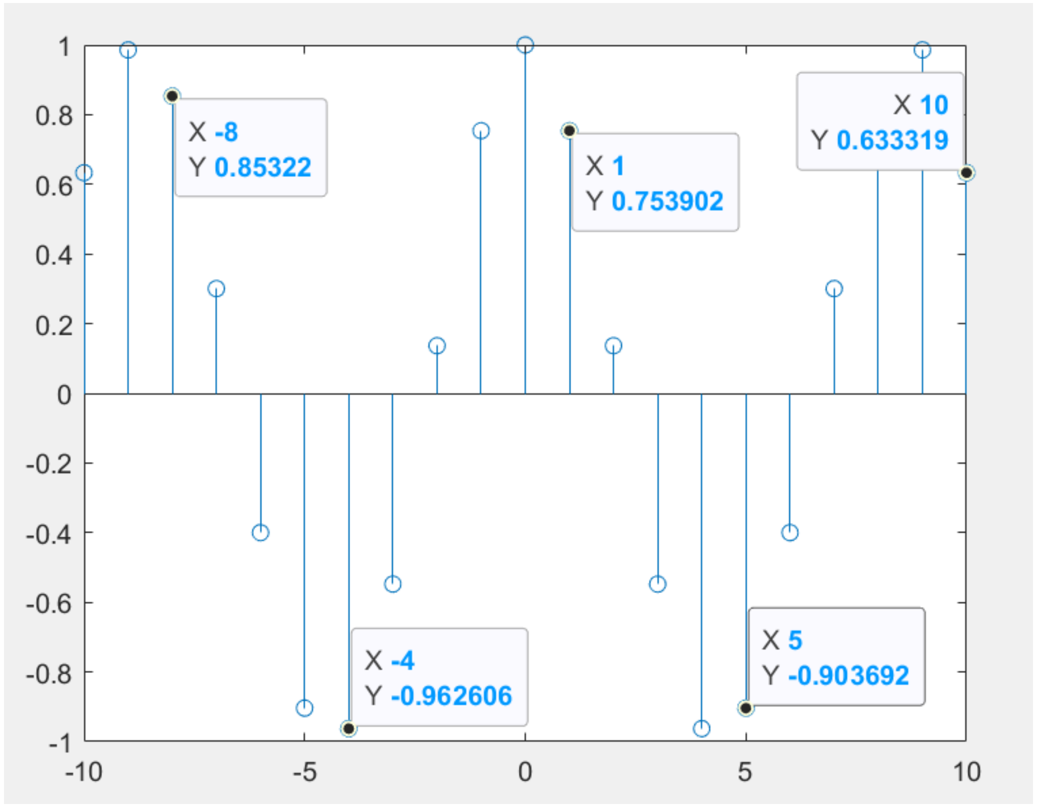
- We can see here that this is indeed periodic with period **5**. It repeats every **5** units starting at **n = 0** and then we get some behavior and then we get this repeat at **n = 5**.

When are complex sinusoids periodic?

Example:

$x[n] = \cos(7n) \qquad N = \frac{2\pi k}{7}$

```
>> stem(n,cos(7*n))
```



- But not every discrete-time signal that looks like a cosine is periodic.
- For the given **x[n]**, if it were in continuous-time world, we would say it is periodic with a period of **2π/7**.
- After isolating **N**, we see that there is no integer (no **k**-value) that will make **2πk/7** an integer value.
- When we plot this signal in MATLAB, we can see that things are kind of periodic, but not exactly. If you were to zoom in on the stem plot, there is no exact repeat here.
- We can also see the numbers on the graph for more verification.
- A sample data point is **n = 10** (or **X = 10**) and **x[10] = 0.633319** (or **Y = 0.633319**).

When are complex sinusoids periodic?

- We can also see the list of numbers using **(1)**.
- For column **21**, shown below in **red** box, the data is **n = 10** (or **X = 10**) and **x[10] = 0.633319** (or **Y = 0.633319**).

```
>> cos(7*n)    (1)

ans =

  Columns 1 through 11

    0.6333    0.9859    0.8532    0.3006   -0.4000   -0.9037   -0.9626   -0.5477    0.1367    0.7539    1.0000

  Columns 12 through 21

    0.7539    0.1367   -0.5477   -0.9626   -0.9037   -0.4000    0.3006    0.8532    0.9859    0.6333
```

- In summary, and in practice, we care a lot about particular cosines that ***are*** periodic in discrete-time world.
- In this regard, and at a later time, we are going to discuss the Fourier transform and the Fourier series for digital signals.

End of Lecture 1