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

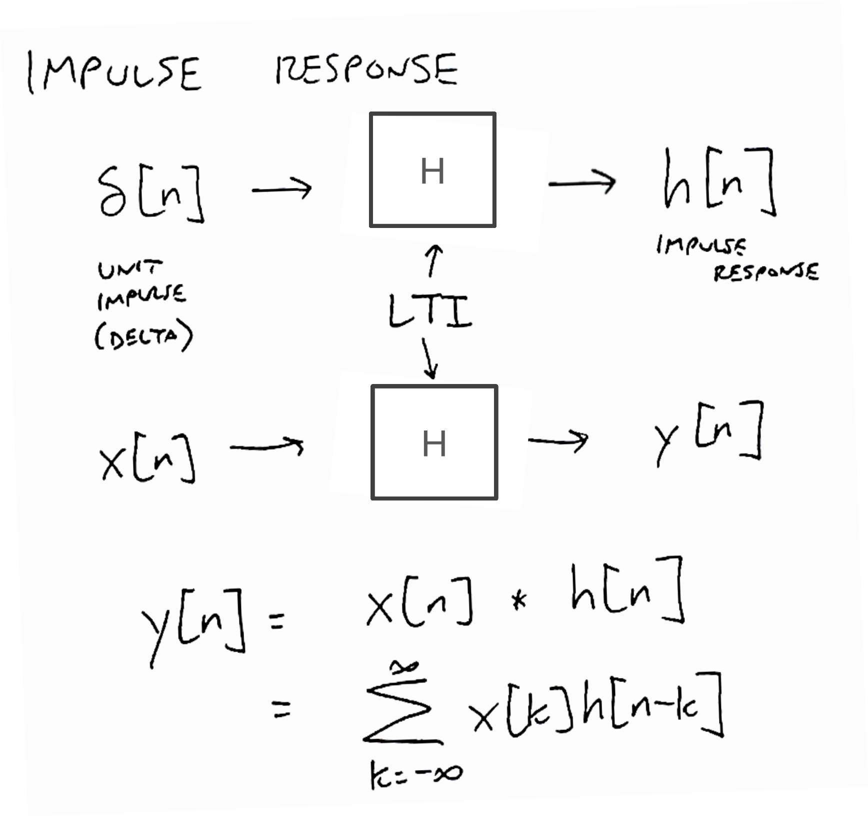
Lecture	Title
Lecture 0	Introduction to DSP and DIP
Lecture 1	Signals
Lecture 2	Linear Time-Invariant System
Lecture 3	Convolution and its Properties
Lecture 4	The Fourier Series
Lecture 5	The Fourier Transform
Lecture 6	Frequency Response
Lecture 7	Discrete-Time Fourier Transform
Lecture 8	Introduction 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 3:
Convolution and its Properties

Table of Contents

- Review of impulse response
- Running example: computing a system's response to a signal in different ways
- Direct computation
- Using the convolution sum: adding up shifted and scaled copies of the impulse response
- Flipping and sliding one signal against the other
- MATLAB example of flipping and sliding
- Understanding h[n-k]
- The convolution array (a fast method for convolving short signals)
- Convolving infinite-length signals
- The sum of a finite geometric series
- Symbolic sum of a series in MATLAB
- Properties of convolution/LTI systems
- Commutative property
- Distributive property
- Associative property
- Causality and the impulse response
- The step response and its relationship to the impulse response
- Differential and difference equations

Review of impulse response



$$y[n] = x[n] * h[n]$$
$$= \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

- **Goal:** We want to know what would happen if we put an arbitrary signal into the system.
- We showed that the output was called the **convolution of the input with the impulse response**.
- For the above to be true, the system should be LTI.

Running example: computing a system's response to a signal in different ways; Direct computation

Example:

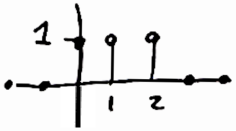
CONSIDER THIS LTI SYSTEM

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

- We would use **double underlined** to remind ourself where **x[0]** is. We can also use a vertical arrow.

WHAT IS THE RESPONSE TO

$x[n] =$



?

$$[\underset{\uparrow\, n=0}{\underline{\underline{1}}}\quad 1\quad 1]$$

Shorthand for input signal.

1) <u>DIRECT</u>

$y[-1] = 0$ ALSO $-2, -3 \ldots$

$y[0] = 1$

$y[1] = 1 - 2(1) = -1$

$y[2] = 1 - 2 + 3 = 2$

$y[3] = -2 + 3 = 1$

$y[4] = 3$

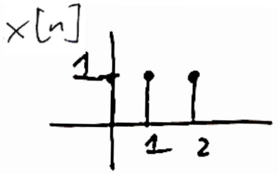
$$\boxed{[\underline{\underline{1}}\quad -1\quad 2\quad 1\quad 3]}$$

Shorthand for output signal.

Using the convolution sum: adding up shifted and scaled copies of the impulse response

2) USING CONVOLUTION SUM

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]\,h[n-k]$$



WHAT IS THE IMPULSE RESPONSE?

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

$$\delta[n] = [\underline{\underline{1}}]$$

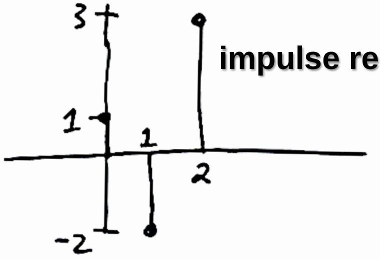
$$h[0] = 1$$

$$h[1] = 0 - 2 = -2$$

$$h[2] = 0 - 0 + 3 = 3$$

$$h[3] = 0$$

$$[\underline{\underline{1}} \quad -2 \quad 3]$$



impulse response

$$[\underline{\underline{1}} \quad -2 \quad 3]$$

$$y[n] = 1x[n] - 2x[n-1] + 3x[n-2] \quad (1)$$

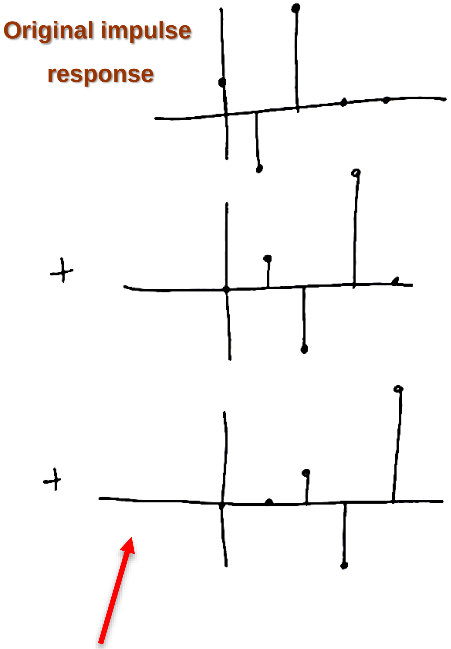
- We can just read off the impulse response, **1**, **-2**, **3** in the shorthand, which is the same as **1**, **-2**, **3** in equation **(1)**. We are simply writing down the **coefficients** on each of the delay versions of the signal.

Using the convolution sum: adding up shifted and scaled copies of the impulse response

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \quad \textbf{(1)}$$

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

- What we are doing here is weighting the delayed versions of the impulse response, **(1)**. The weighting factors are **x[k]**'s.
- For example, for **x[1]h[n-1]**, the weighting is **x[1]** (or the coefficient of the impulse signal) and the delayed version of the impulse signal is **h[n-1]**.



$$[\underline{\underline{1}}\;\; -2\;\; 3]$$

Start here

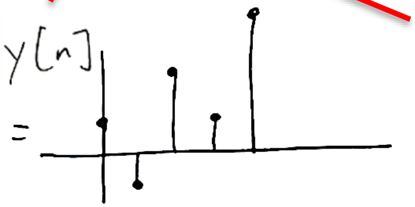
$$+\,[\underline{\underline{0}}\;\; 1\;\; -2\;\; 3]$$

Shift by 1

$$+\,[\underline{\underline{0}}\;\; 0\;\; 1\;\; -2\;\; 3]$$

Shift by 1

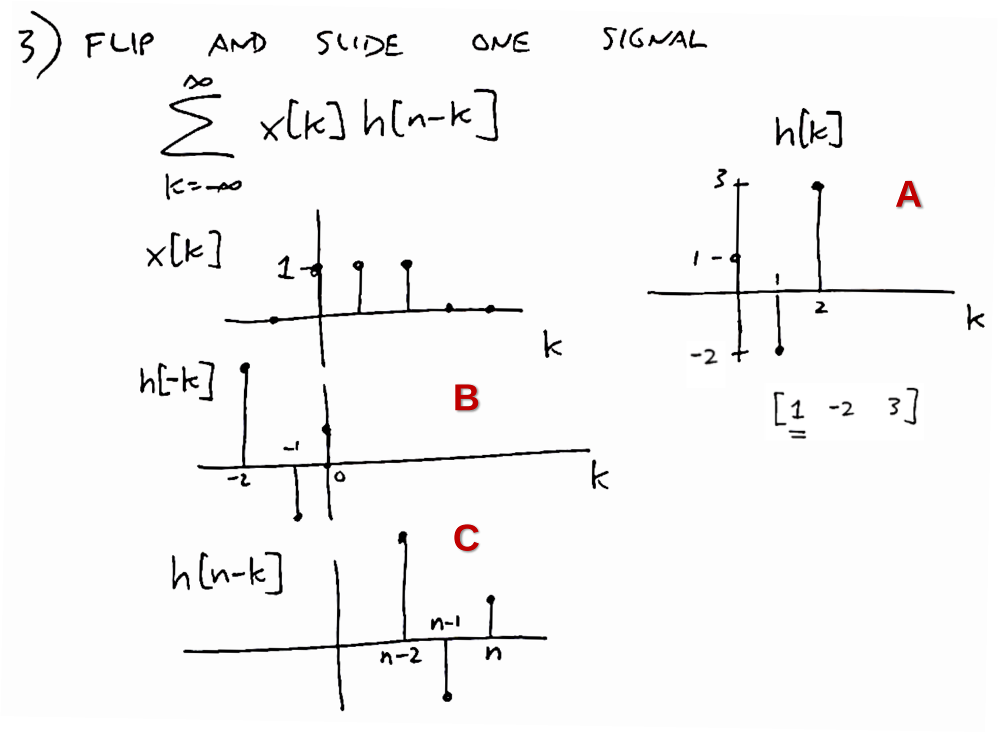
$$\boxed{[\underline{\underline{1}}\;\; -1\;\; 2\;\; 1\;\; 3]}$$



- Zero element is right here (at **y[0] = 1**).

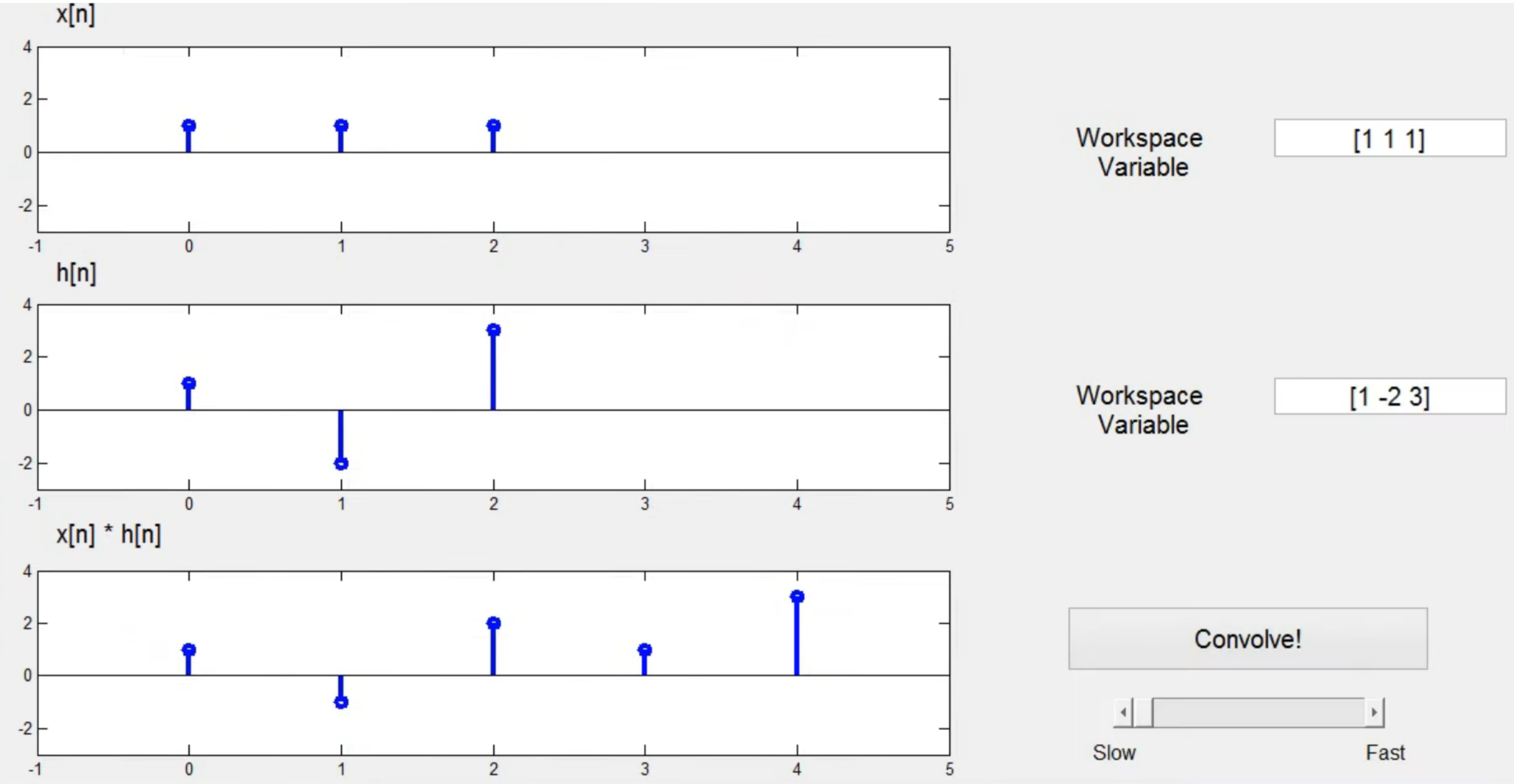
- We are adding up the ***corresponding sticks***.

Flipping and sliding one signal against the other


3) FLIP AND SLIDE ONE SIGNAL
$\sum_{k=-\infty}^{\infty} x[k]\,h[n-k]$
x[k]
1
k
h[k]
3
1
-2
1
2
k
A
[1 -2 3]
h[-k]
-1
-2
0
k
B
h[n-k]
n-1
n-2
n
C

- Here, **k** is like our dependent variable and **n** is like a constant.
- **h[-k]**, **B**, is the mirror image of **h[k]**, **A**.
- **h[n-k]**, **C**, looks like **h[-k]** slid to the right by **n** units. In **h[n-k]** graph, what used to be **0** will begin at **n**.
- The convolution sum tells us that first we need to find out **x[k]**, then find out **h[n-k]** and finally multiply corresponding entries up and sum them up together. That gives us one value of **y**.

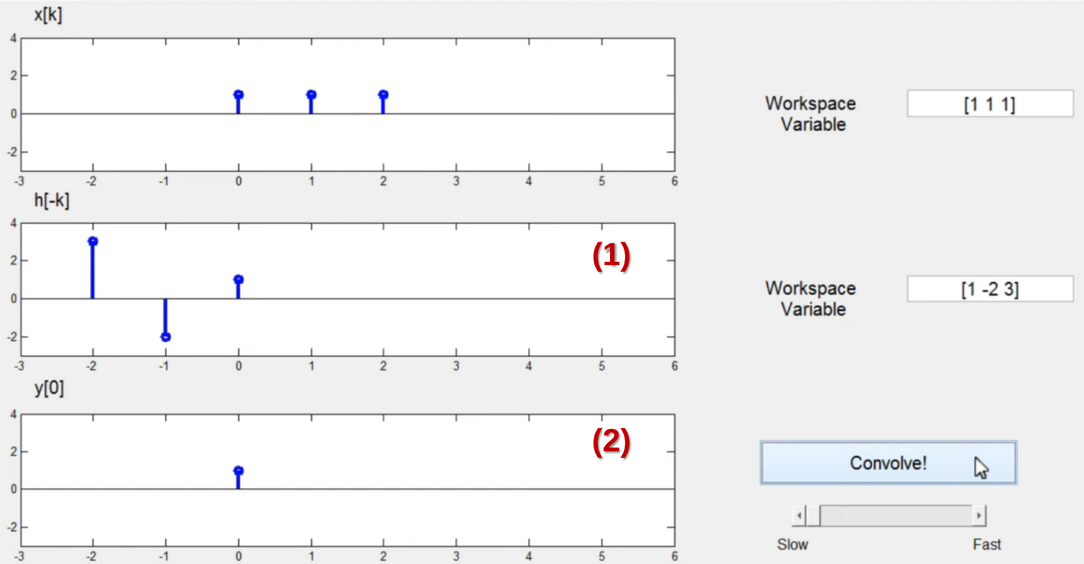
MATLAB example of flipping and sliding



Workspace Variable [1 1 1]

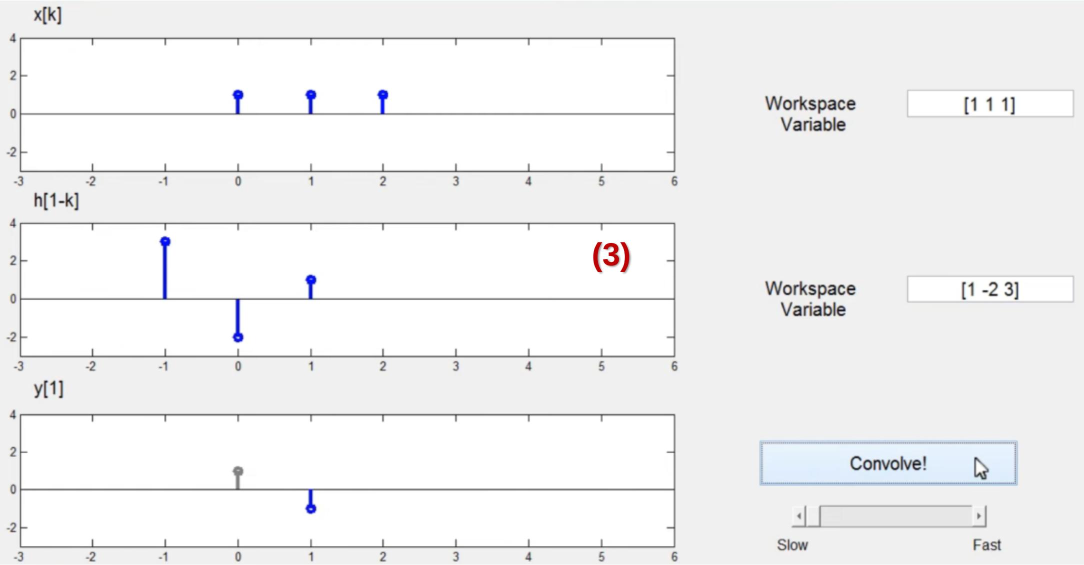
Workspace Variable [1 -2 3]

MATLAB example of flipping and sliding



Workspace Variable [1 1 1]

Workspace Variable [1 -2 3]

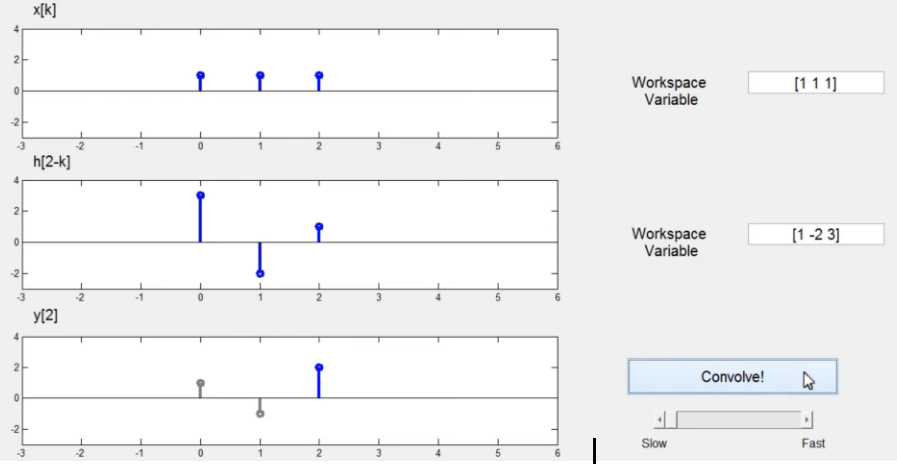


Workspace Variable [1 1 1]

Workspace Variable [1 -2 3]

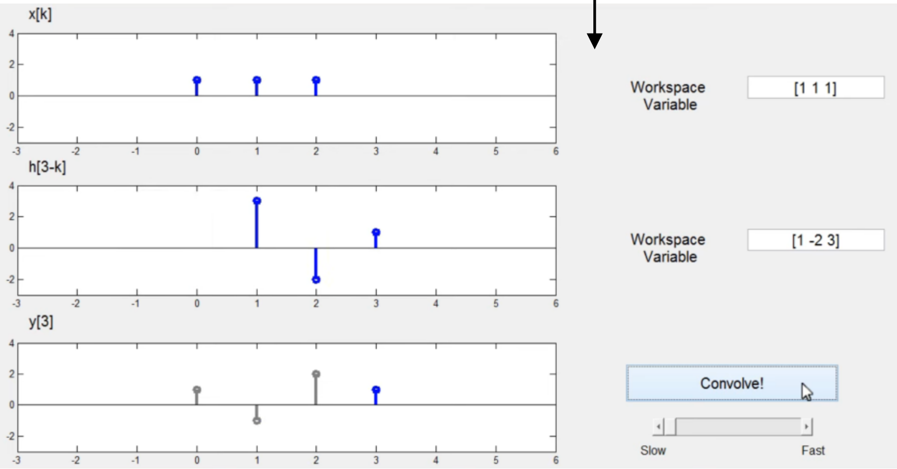
- **How does the convolution work?** What is going to happen is we are going to see that **h[k]** is going to flip around. Then we are going to start click it across the **x[k]** and at each point we multiply the corresponding entries at the bottom and add them up.
- Here, **h[-k]** is the filliped version, i.e., graph **(1)**. In **(2)**, we get one entry (**y[0]**). Then we click the signal **h[-k]** over by **1**. Now, we get **h[1-k]** or graph **(3)**. We add the corresponding entries up and we get another entry, **y[1]**, and so on.

MATLAB example of flipping and sliding



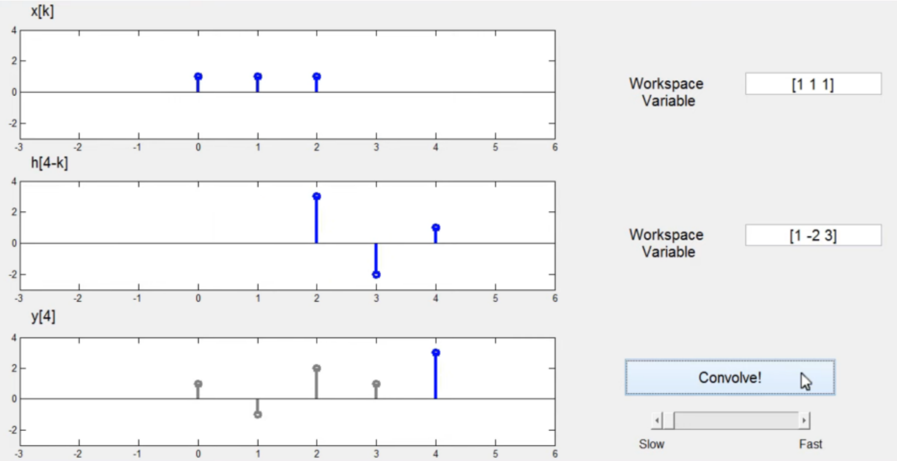
Workspace Variable [1 1 1]

Workspace Variable [1 -2 3]



Workspace Variable [1 1 1]

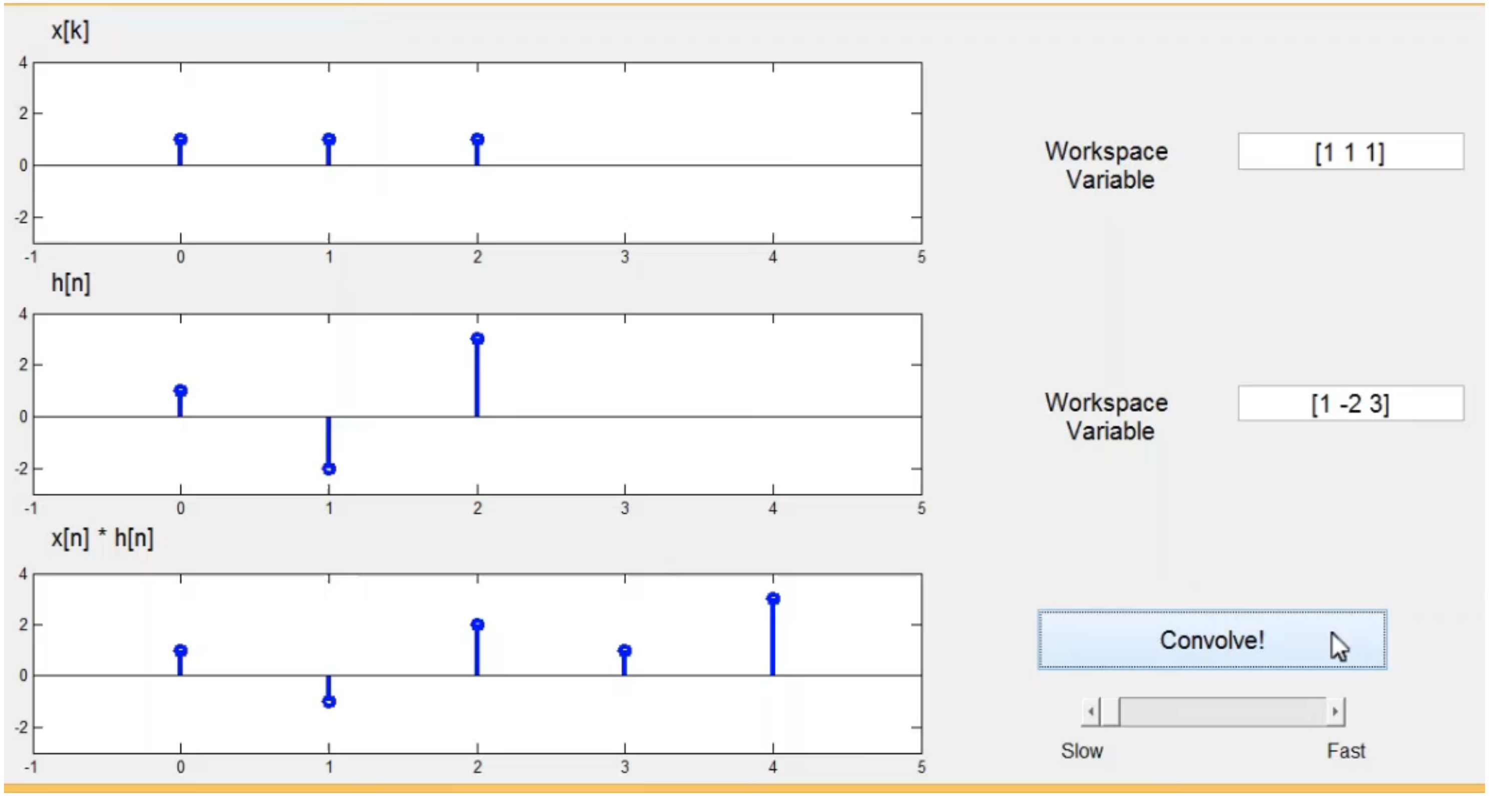
Workspace Variable [1 -2 3]



Workspace Variable [1 1 1]

Workspace Variable [1 -2 3]

MATLAB example of flipping and sliding

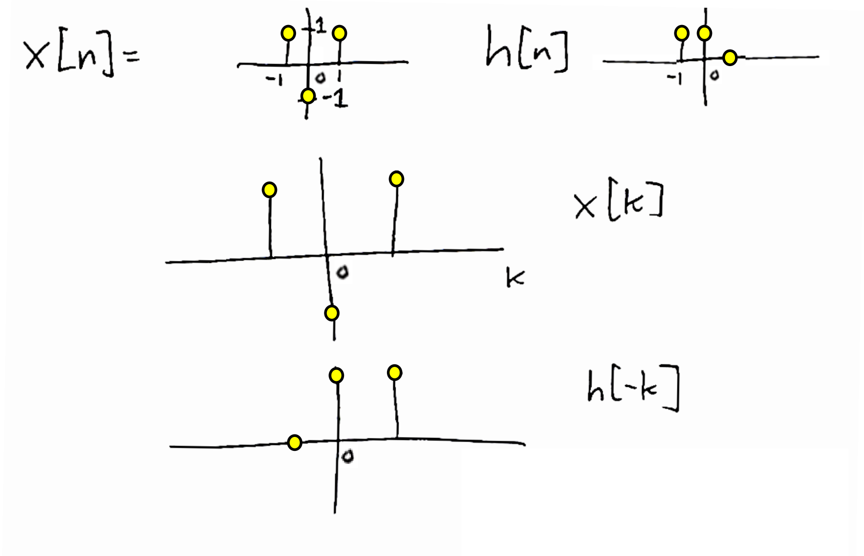


Workspace Variable [1 1 1]

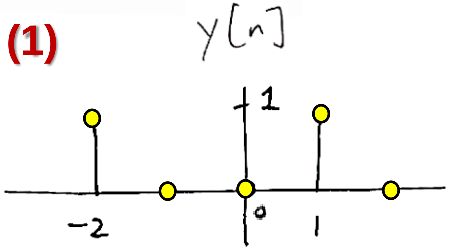
Workspace Variable [1 -2 3]

Understanding h[n-k]

Example:



(1)



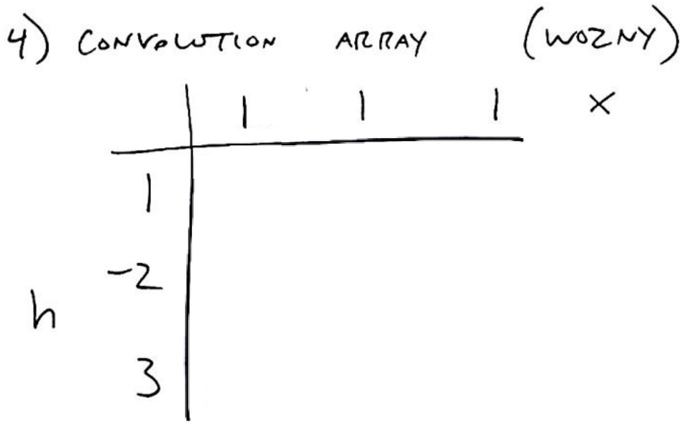
- In MATLAB, the **disp(y)** command is used to display the value of the variable **y** in the **Command Window**. This function shows the value without printing the variable name, making it useful for displaying results or messages in a clean format.

- Can we still use the previous process when there is an impulse response beyond zero on the left hand side? The answer is "yes".
- So, if we have negative values like **h[-1]**, the same idea applies. We do not usually like systems like that, but the same flipping, shifting computation method is applied. And that is where it is really important to keep track of the zero element (see the example).
- In this example, we also have to think about what happens when we move **h[-k]**. That means we may get some **y**-values that are non-zero for negative **n**'s.
- The final output signal is given in graph **(1)**.

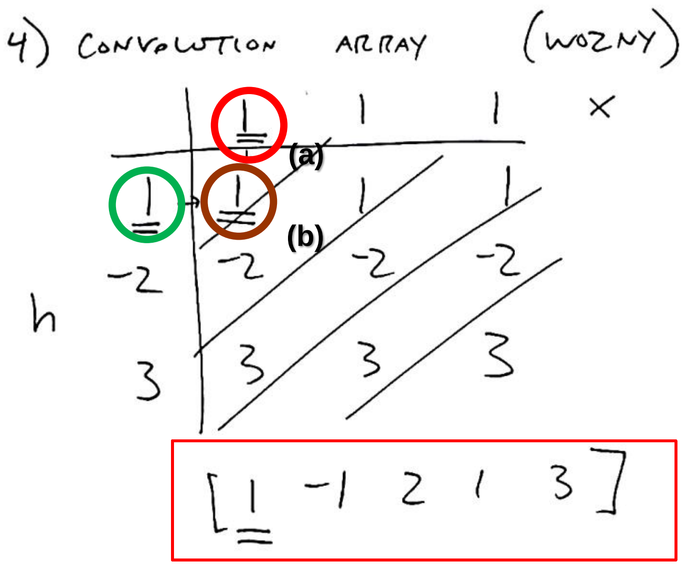
MATLAB Code for Convolution:

```
>> x = [1 -1 1];
h = [1 1 0];
y = conv(x, h);
disp(y);
     1     0     0     1     0
                 ↑
```

The convolution array (a fast method for convolving short signals)



A



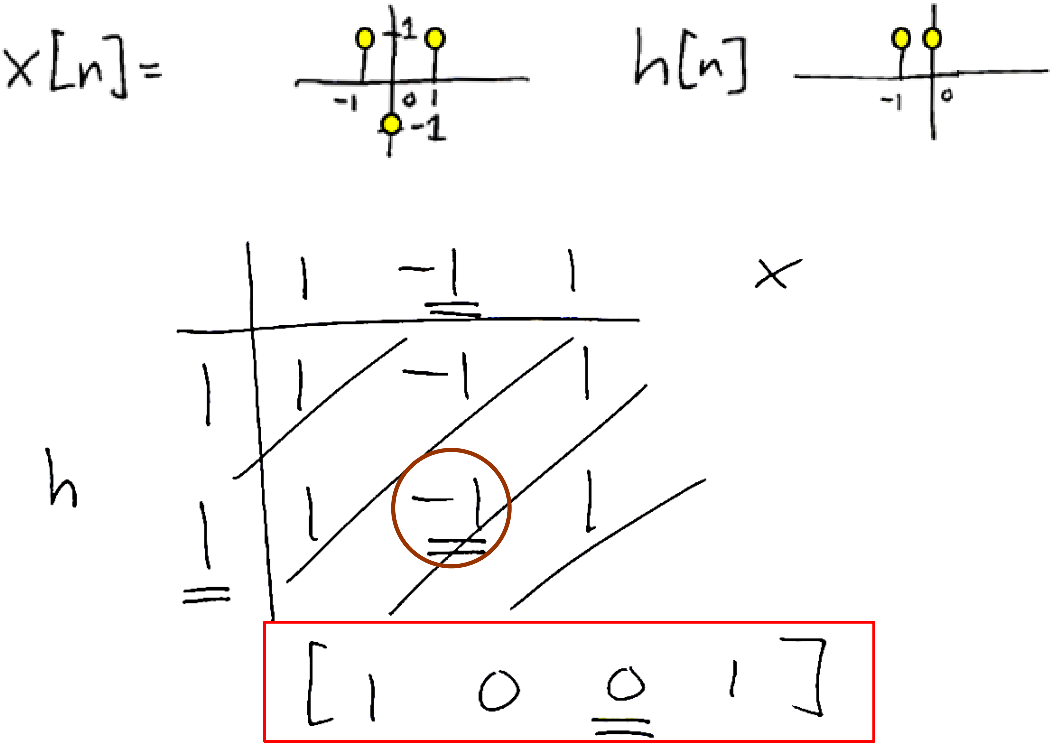
B

- **Convolution Array Method:** We take the elements of **x** and the elements of **h** and we make a table, **A**. Then we are going to fill this table up with products of values of **x** and **h**, as shown in **B**. For example, **1** (encircled **green**)×**1** (encircled **red**) is **1** (encircled **brown**), and **-2×1** is **-2**, etc. Then we look at the diagonals of this table and we add up corresponding values. For example, on the diagonal line of "**(a)**", we have only one element: **(1) = 1** and on the diagonal line of "**(b)**", we have: **(-2) + (1) = -1**, etc. So, here we get **[1, -1, 2, 1, 3]**.

- **How do we know what is the zero element of the result?** First, we double-underline both the zero element of the input (**x**) and the zero element of the impulse response (**h**) and where those two things cross is the zero element of the output. That is, wherever that double-underline crossing lands is where we **n = 0** for **y**. In this case, they land on the **1** encircled **brown**. Because the brown element is on the first diagonal, so the result of our calculations for the first diagonal will give us the location of the zero-value element. In this example, the first element in the final output sequence is our zero-element.

The convolution array (a fast method for convolving short signals)

Example:

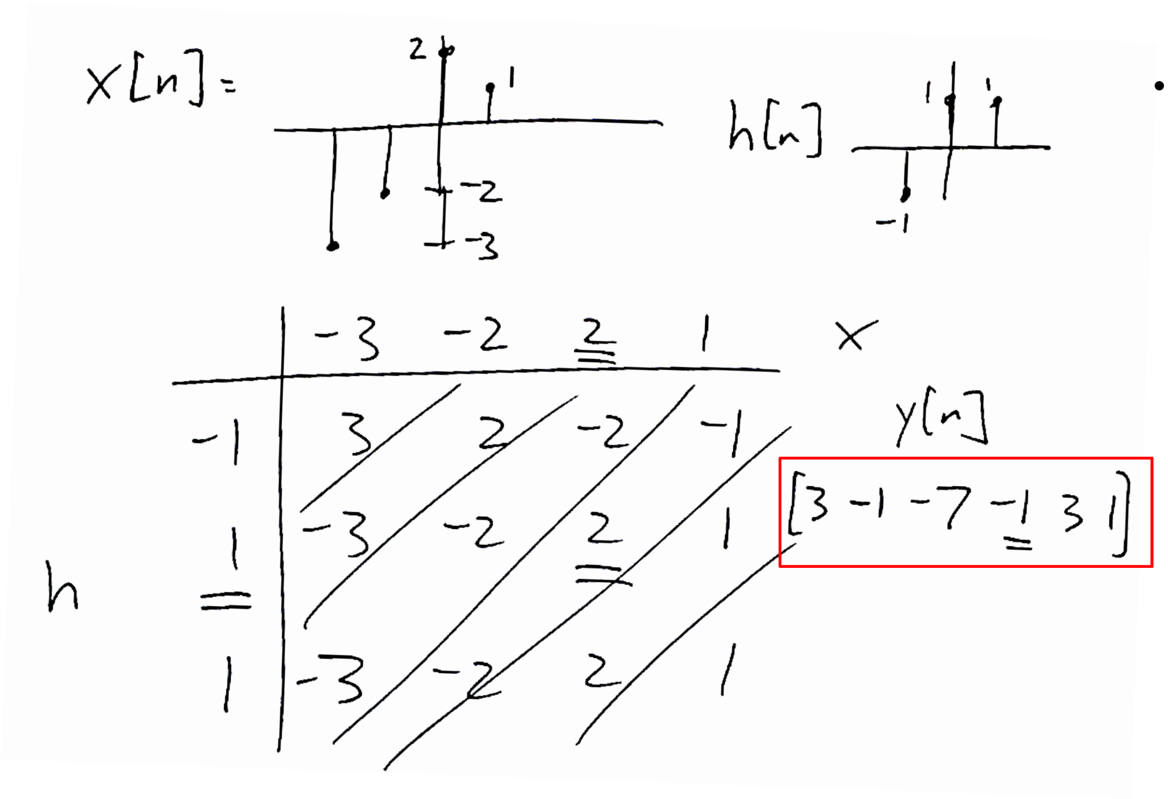


- This is the zero-value element.

- Let us use the **Convolution Array Method** to do this example.
- We see that this method is much easier than any of the previous methods we covered so far.
- The crossing of the double-underlined **-1** and **1** gives us an element that is encircled by the **brown** circle, i.e., **-1**. This means that the zero element will be the outcome of our computation for this particular diagonal.

The convolution array (a fast method for convolving short signals)

Example:



	-3	-2	2	1
-1	3	2	-2	-1
1	-3	-2	2	1
1	-3	-2	2	1

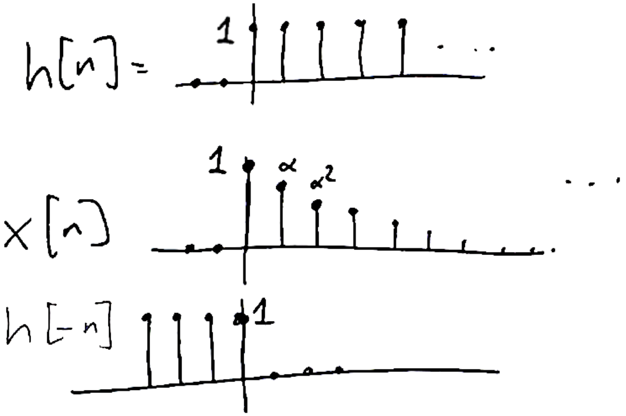
- Another example that is solved by the **Convolution Array Method**.

Convolving infinite-length signals

Example:

$x[n] = \alpha^n u[n] \qquad \alpha \in (0,1)$

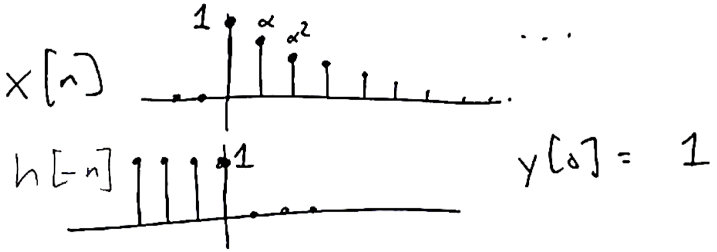
$h[n] = u[n]$



- **Note:** We cannot use the convolution array method for every input signal, specifically, the ones that are infinitely long.
- In this example, instead of having just solid numbers, we also have ***symbols***, i.e., $\boldsymbol{\alpha}$.
- But this exponential decay goes on forever and for any value of **n**, no matter how large, we are still going to get a little non-zero value of **x**. So, let us investigate what is going to be the convolution between these two things.
- One nice thing about convolution is that it does not matter which signal we flip and shift first. We can either choose **x** or we can choose **h**. In this case, it is going to be a little bit easier for us to choose **h**.

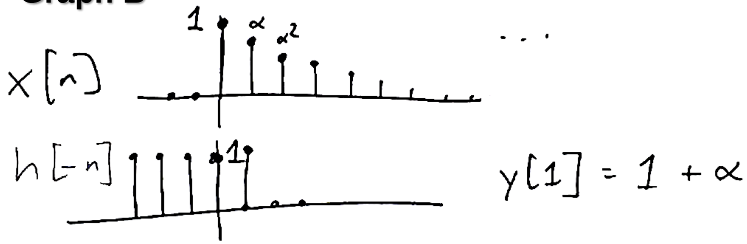
Convolving infinite-length signals

Graph A



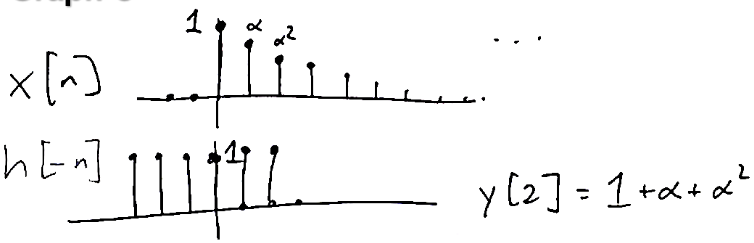
- In **Graph A**, when we are at **0** in **h[-n]** plot, for **y[0]**, we get **1** from **x[n]** plot times **1** from **h[-n]** plot. So, **y[0] = 1×1 = 1**.

Graph B



- In **Graph B**, for **h[-n]** plot, we click **1** unit over and then we repeat the same process. So, **y[1] = 1 +** $\boldsymbol{(\alpha)(1)}$ **= 1+** $\boldsymbol{\alpha}$.

Graph C



- In **Graph C**, for **h[-n]** plot, we click **1** unit over and then we repeat the same process. So, **y[2] = 1 +** $\boldsymbol{\alpha}$ **+** $\boldsymbol{\alpha^2(1)}$ or **y[2] = 1 +** $\boldsymbol{\alpha}$ **+** $\boldsymbol{\alpha^2}$.

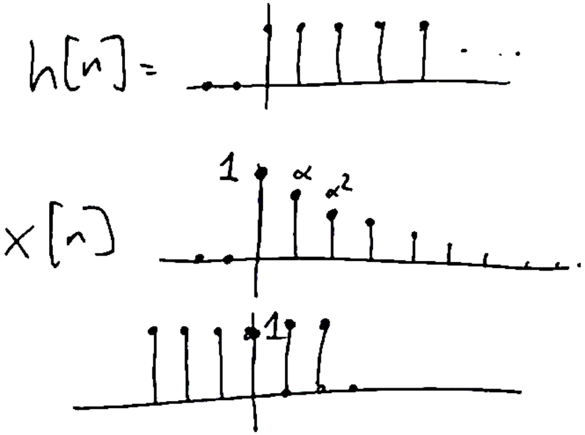
Convolving infinite-length signals

Finding the Pattern:

$$x[n] = \alpha^n u[n] \qquad \alpha \in (0,1)$$

$$h[n] = u[n]$$

- Now we see there is a ***pattern***.



$\vdots$

- All the elements before **y[-1]** are zero.

$$y[-1] = 0$$

$$y[0] = 1$$

$$y[1] = 1 + \alpha$$

$$y[2] = 1 + \alpha + \alpha^2$$

Convolving infinite-length signals

The Pattern:

$$y[n] = \begin{cases} 0 & n < 0 \\ \sum_{k=0}^{n} \alpha^k & n \geq 0 \end{cases}$$

$$\longrightarrow \boxed{y[n] = \left(\sum_{k=0}^{n} \alpha^k\right) u[n]}$$

The sum of a finite geometric series

A Sidebar:

REMEMBER:

$$0<\alpha<1$$

$$\sum_{k=0}^{\infty} \alpha^k = \frac{1}{1-\alpha}$$

Proof:

$$(1+\alpha+\alpha^2+\alpha^3+\dots)(1-\alpha) = \begin{array}{l} 1+\alpha+\alpha^2+\alpha^3+\dots \\ \quad -\alpha-\alpha^2-\alpha^3\dots = 1 \end{array}$$

$$\sum_{k=0}^{n} \alpha^k = \sum_{k=0}^{\infty} \alpha^k - \boxed{\sum_{k=n+1}^{\infty} \alpha^k}$$

$$\downarrow \quad \alpha^{n+1} \sum_{k=0}^{\infty} \alpha^k$$

$$= \frac{1}{1-\alpha} - \frac{\alpha^{n+1}}{1-\alpha}$$

$$= \frac{1-\alpha^{n+1}}{1-\alpha}$$

$$0<\alpha<1$$

$$\sum_{k=0}^{\infty} \alpha^k = \frac{1}{1-\alpha}$$

$$0<\alpha<1$$

$$\sum_{k=0}^{n} \alpha^k = \frac{1-\alpha^{n+1}}{1-\alpha}$$

The sum of a finite geometric series

- Using the pervious formulas, we get the following:

$$y[n] = \begin{cases} 0 & n < 0 \\ \sum_{k=0}^{n} \alpha^k & n \geq 0 \end{cases} \qquad \longrightarrow$$

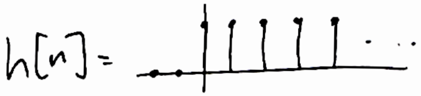
$$y[n] = \left(\sum_{k=0}^{n} \alpha^k \right) u[n] \qquad \longrightarrow$$

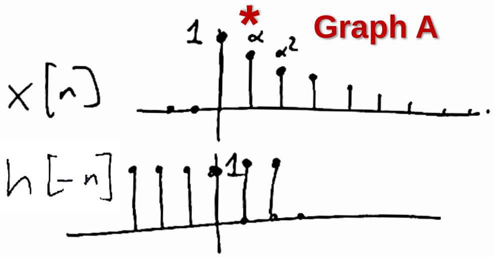
$$\boxed{y[n] = \left(\frac{1 - \alpha^{n+1}}{1 - \alpha} \right) u[n]}$$

The sum of a finite geometric series

$x[n] = \alpha^n u[n] \qquad \alpha \in (0,1)$

$h[n] = u[n]$





$y[-1] = 0$

$y[0] = 1$

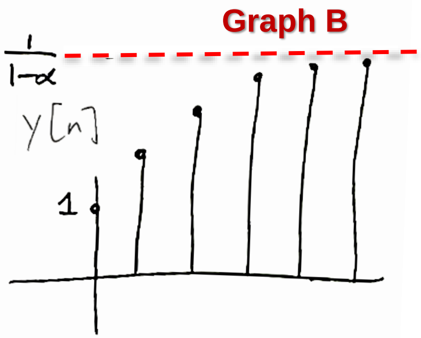
$y[1] = 1 + \alpha$

$y[2] = 1 + \alpha + \alpha^2$

$$y[n] = \begin{cases} 0 & n < 0 \\ \sum_{k=0}^{n} \alpha^k & n \geq 0 \end{cases}$$

$$y[n] = \left(\sum_{k=0}^{n} \alpha^k \right) u[n]$$

$$y[n] = \left(\frac{1 - \alpha^{n+1}}{1 - \alpha} \right) u[n]$$



- One way to think about **y[n]** is that we are making a cumulative sum of the values of **x**. In the beginning, we are adding relatively larger sticks to our sum (shown by the region of * in **Graph A**). So, the sum will grow faster in the beginning and then as we move our "**h[-n]**" along the signal **x[n]**, the next little bit that we add from the original **x[n]** is pretty small. It gets smaller and smaller, and finally, we are going to reach a point of diminishing returns and the output signal would look like **Graph B**.
- Eventually, the graph tapers off and there is going to be an **asymptote** (shown by horizonal dashed-line) that we never quite reach.

The sum of a finite geometric series

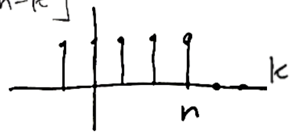
$$x[n] = \alpha^n u[n] \qquad \alpha \in (0,1)$$

$$h[n] = u[n]$$

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

$$= \sum_{k=-\infty}^{\infty} \alpha^k u[k]\, u[n-k] \qquad \textbf{(1)}$$

$$= \sum_{k=0}^{\infty} \alpha^k \underbrace{u[n-k]}_{=1 \text{ For } k \le n} \qquad \textbf{(2)}$$



$$\boxed{y[n] = \left(\sum_{k=0}^{n} \alpha^k\right) u[n]} \qquad \textbf{(3)}$$

- Let us now use the **Convolution Sum**.
- What does the product of the two step functions in **(1)** look like?
- Here, everything is a function of **k** and when **k** is negative, the step function of **u[k]** is **0**. So, this is like a step function that does not turn on until **k = 0**. It means that we can immediately remove **u[k]** entirely and the sum now turns into a sum that starts at **k = 0**. This is because we know nothing can be happening before **0**.
- In **(2)**, **u[n-k]** is like a step function that is flipping and then shifting **n** units. It is equal **1** for **k** ≤ **n**. Based on **(3)**, we managed to get rid of **u[n-k]** in **(2)**.

Symbolic sum of a series in MATLAB

MATLAB built-in function symsum(f, k, a, b):

- The **symsum** function is powerful for symbolic computations, allowing us to work with exact expressions rather than numerical approximations. The **symsum(f, k, a, b)** function in MATLAB computes the symbolic sum of a series. Here is a breakdown of its usage:
  - ➢ **f**: The expression defining the terms of the series.
  - ➢ **k**: The summation index.
  - ➢ **a**: The lower bound of the summation.
  - ➢ **b**: The upper bound of the summation.

Example 1:

- To find the sum of (**k^2**) from (**k = 1**) to (**k = 10**):

syms k; F = symsum(k^2, k, 1, 10); disp(F);

This will output: **385**

Example 2:

syms k n; F = symsum(k^2, k, 1, n); disp(F);

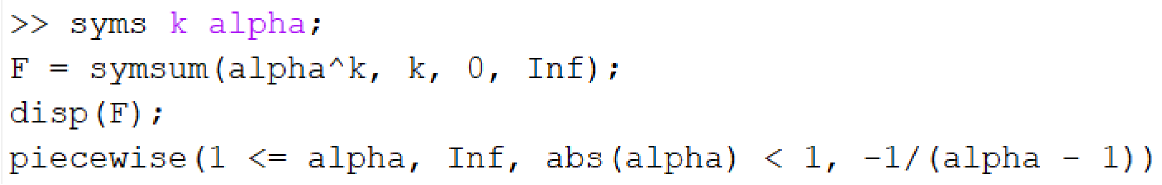
This will output: **(n*(n + 1)*(2*n + 1))/6**

Example 3 (Indefinite Sum):

- The following **symsum** returns the indefinite sum (antidifference):

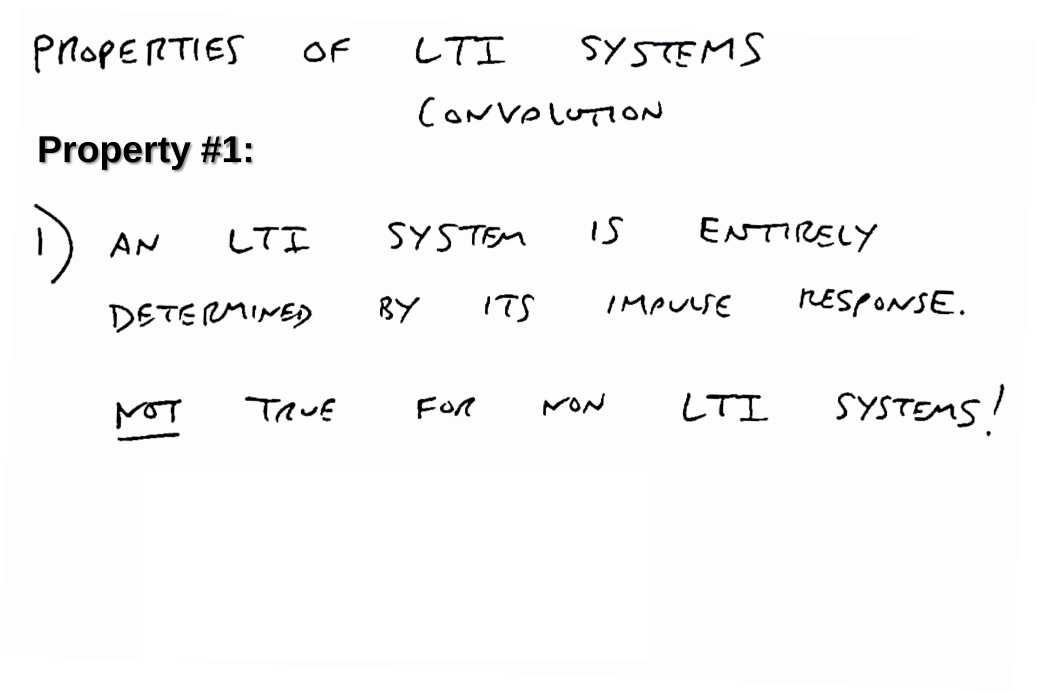
syms k alpha; F = symsum(alpha^k, k, 0, Inf); disp(F);

This will output: **1/(1 - α)** with the assumption that 0 < α < 1.



```
>> syms k alpha;
F = symsum(alpha^k, k, 0, Inf);
disp(F);
piecewise(1 <= alpha, Inf, abs(alpha) < 1, -1/(alpha - 1))
```

Properties of convolution/LTI systems



PROPERTIES OF LTI SYSTEMS

CONVOLUTION

Property #1:

1) AN LTI SYSTEM IS ENTIRELY DETERMINED BY ITS IMPULSE RESPONSE.

NOT TRUE FOR NON LTI SYSTEMS!

- A real world system is usually composed of lots of little subsystems put together either in series or in parallel.
- So, we need to know how we can analyze those systems. We want to have an **effective impulse response for the entire system** and not getting involved in computing the convolution from one response and then go to the next one, and so on. That would be challenging and tedious.
- **Property #1 or Impulse Response:** This is basically saying that as long as we know **h[n]**, all we have to do next is to convolve **x** with **h** and then we will know how the system acts for any input signal.

Properties of convolution/LTI systems

Example:

$y[n] = nx[n]$

- **How can we test if $y[n] = nx[n]$ is an LTI system?** Here, instead of a general proof method, let us use the method we discussed before, i.e., the method known as the **counterexample**, which relies on the fact that a single counterexample that violates the claim is sufficient (enough) to prove that the claim is wrong.
- Assume that the system was LTI, (the claim is that system is LTI). Then consider the following inputs $x_1[n] = \delta[n]$ and $x_2[n] = \delta[n-1]$ with corresponding outputs $y_1[n] = 0\cdot\delta[n] = 0$ for all n and $y_2[n] = 1.\delta[n-1] = \delta[n-1]$.
- Now if the system is actually LTI, then its output should satisfy $y_2[n] = y_1[n-1]$. However, one can see that this is not the case, as $y_2[n] = \delta[n-1] \neq y_1[n-1] = 0$. Hence, this counterexample has proved that the claim was wrong; i.e., the system is not LTI.
- In fact, what we have actually shown is that the system is not time-invariant, but then the fact that it is also not LTI follows as a consequence.
- Here, we used **the sifting property of the impulse function**, i.e., $f[n]\delta[n-a] = f[a]\delta[n-a]$, which becomes specifically as $n.\delta[n] = 0.\delta[n] = 0, \forall n$.

Commutative property; Distributive property

Property #2:

2) COMMUTATIVE

$$x[n] * h[n] = h[n] * x[n]$$

$$\downarrow$$

$$\sum_{k=-\infty}^{\infty} x[k]\, h[\overset{\overset{m}{\downarrow}}{n-k}] \qquad \begin{aligned} m &= n-k \\ k &= n-m \end{aligned}$$

$$\hookrightarrow \sum_{m=-\infty}^{\infty} x[n-m]\, h[m]$$

Property #3:

3) DISTRIBUTIVE

$$x[n] * \left(h_1[n] + h_2[n]\right)$$

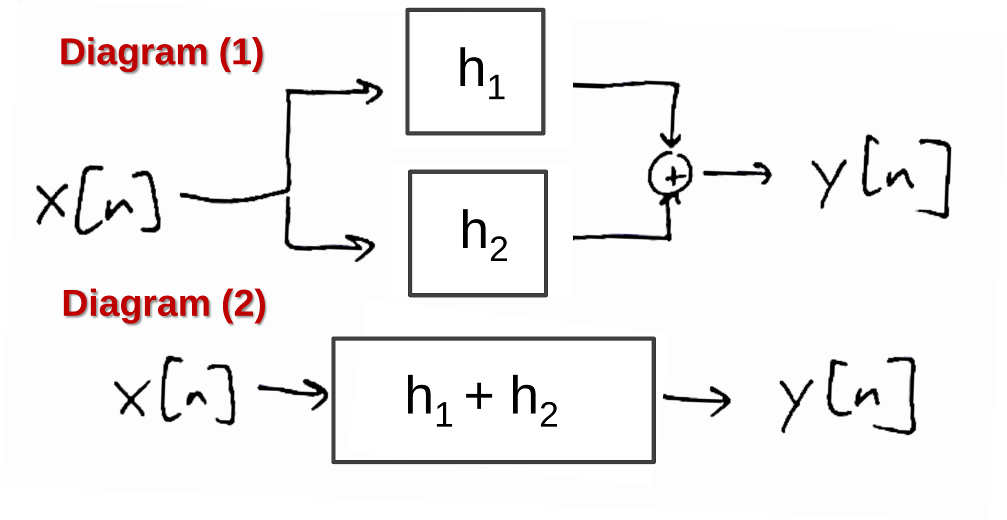
$$= x[n] * h_1[n] + x[n] * h_2[n]$$

- **Property #3 or Distributive Property:** This states that the convolution of a signal **x[n]** with the sum of two signals **h_1[n]** and **h_2[n]** is equal to the sum of the individual convolutions of **x[n]** with **h_1[n]** and **h_2[n]**. It has applications in parallel systems.

- **Property #2 or Commutative Property:** This basically says that if we were to interchange the order of the signals, when we convolve them, we get the same answer.
- We can keep **x** the way it is and we flip **h** and we run **h** across the signal. We do not have to flip **h**, we can flip **x** instead and run it around across **h**.
- Let us show that why that is true. We can do that by a simple change of variables. Let **m = n - k**. Here, **k** goes from minus infinity to plus infinity. That means **m** also goes from minus infinity to plus infinity. That is the same thing as if we were to interchange the **signal order**.
- In the real world engineering, we may find that it is easier to flip one signal than to flip the other signal.

Distributive property

Example for Distributive Property:



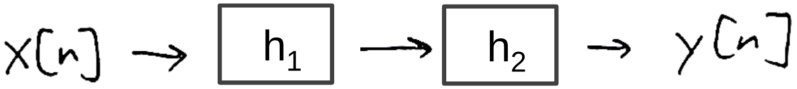
- If we have a signal, **x[n]**, that goes through two pieces, **Diagram (1)**, and then we add those pieces up later to get **y[n]**, an equivalent system is what is shown in **Diagram (2)**.

Associative property

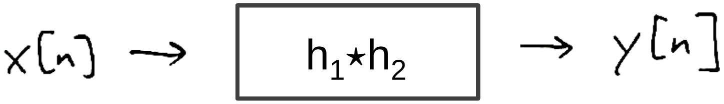
4) ASSOCIATIVE

$$x[n] * (h_1[n] * h_2[n]) \quad \textbf{(1)}$$
$$= (x[n] * h_1[n]) * h_2[n])$$

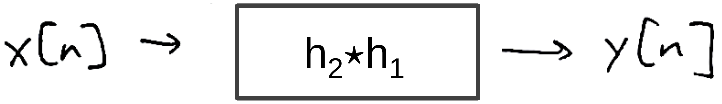
Examples for Associative Property:



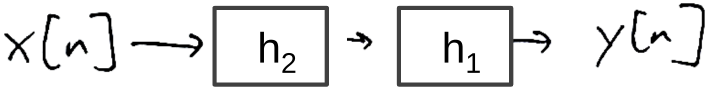
(2)



(3)



(4)

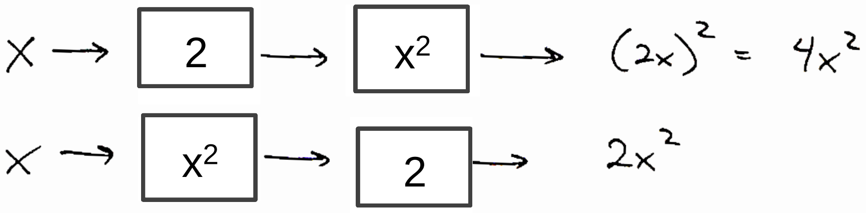


(5)

- **Property #4 or Associative Property:** This states that in equation **(1)**, we can put the parentheses wherever we want. We could basically say that we can move these parentheses around or we could remove the parentheses altogether. And, this is important in terms of the way that systems are **composed**.
- **Note: Composed Systems** refer to systems that are constructed by connecting multiple basic digital signal processing building blocks or subsystems together.
- In **(2)**, we put a signal through a system with one impulse response and then we put it through some other system and we get our result, **y[n]**. We could equivalently put **x[n]** through a single system, shown by **(3)**, whose impulse response is the convolution of $\mathbf{h_1}$ and $\mathbf{h_2}$. And, if we combine this with the ***communitive property***, shown in **(4)**, we could also interchange the order of these two **h**'s (because we would get the same answer). Finally, as shown in **(5)**, we could take this apart again and say that it does not matter what order we put the systems in. That is, we can switch the systems themselves.
- In summary, we can effectively combine all our LTI systems into one glove and also we can put the systems in any order that we want (we should get the same answer).

Associative property; Causality and the impulse response

Example for which Associative Property fails:

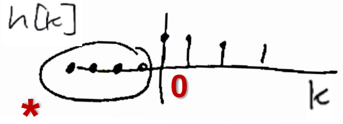


5) CAUSALITY

y[n] DOESN'T DEPEND ON x[n+k] FOR k>0.

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]\,x[n-k] \qquad \textbf{(1)}$$

FOR A CAUSAL SYSTEM $h[k] = 0$ FOR $k < 0$.



Graph A

- What we discussed about the associative property ***only works for LTI systems***. Namely, the same is definitely not true for nonlinear systems.
- Let us look at an example where we have a system in which, first, we multiply **x** by **2** and then we square. What we get out would be **$4x^2$**. Whereas if we were to square the signal first, then multiply by **2**, what we would get would be **$2x^2$**. So, here, this is a situation where interchanging these two blocks does not give us the same answer. The reason is that **x^2** block is not linear. In this case, we cannot use the LTI properties. The **system interchangeability** only applies to LTI systems.

- **Property #5 or Causality Property:** Causality means that **y[n]** does not depend on **x[n+k]** for **k > 0**. In the convolution sum equation, **(1)**, when **k < 0** in **x[n-k]**, it means we are looking to the future of **x**. So, if we have a causal system, for this to work, the corresponding impulse response value has to be **0** after multiplying future values of **x** with **0**. The impulse response has to look like **Graph A**. They do not start before **0**. The system cannot be doing stuff before it sees the impulse. All the values shown in region *****, for **h[k]** vs. **k** graph, should be **0**.

The step response and its relationship to the impulse response

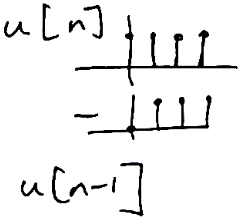
6) STEP RESPONSE

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

$$\downarrow \qquad \downarrow$$

$$h[n] = s[n] - s[n-1]$$

↑ STEP RESPONSE



$$u[n] = \sum_{k=-\infty}^{n} \delta[k] \quad \textbf{(1)}$$

$$s[n] = \sum_{k=-\infty}^{n} h[k] \quad \textbf{(2)}$$

$$s[n] = H(u[n])$$

s[n] = u[n] * h[n]

STEP RESPONSE ALSO CHARACTERIZES THE LTI SYSTEM.

- **Property #6 or Step Response:** Since the step signal **u[n]** looks like the sum of the delta functions up to a certain point, **(1)**, then the **step response s[n]** is equivalently the sum of values of the impulse response **h[k]** up to a certain point, **(2)**. So, if we know this impulse response, we can get the step response and vice versa.
- **Conclusion:** If we know what the step response is, we can go back and compute the impulse response and thus compute the response to anything.

- **Practical Application of these Properties:** The reason we are studying all these properties is that it simplifies the computational work. They will come in handy in cases where we have to compute what may look like tricky parallel or serial combinations of systems. Before we just brute force our way through a lot of convolutions, we should think about “Can we take pieces of the system and combine them in ways that could possibly make our life easier?”.

Differential and difference equations

MANY ELECTRICAL / MECHANICAL SYSTEMS ARE DESCRIBED BY DIFF'L EQUATIONS.

DISCRETE VERSIONS OF THESE ARE CALLED <u>DIFFERENCE EQUATIONS</u>.

(1) $$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k]$$

(2) $$y[n] = y_h[n] + y_p[n]$$

(3) $$y[n] = \sum_{k=0}^{M} b_k x[n-k] \quad \text{FIR} \quad \text{FINITE IMPULSE RESPONSE}$$

- In **(1)**, **a_k**'s and **b_k**'s are called **linear constant coefficient**.
- We solve the difference equations such as **(1)** the same way that we solve differential equations.
- As shown in **(2)**, the solution is going to be a ***homogeneous*** solution plus a ***particular*** solution.
- As a special case of **(1)**, we can have **(3)**. Here, we only have one term, **y[n]**, on the left and other terms are on the right. That is like the systems that we were talking about so far. That is called a **finite impulse response system (FIR system)**. Here, there is really nothing to solve necessarily. We just use convolution to get the answer, whereas once we start combining previous values of **y[n]**, then we have to use **(1)**, the **recursive solution**, to get the difference equation answer. This is where we have to go through the machinery of homogeneous solution and particular solution.

Differential and difference equations

- Later on, we will see that convolution and the responses of LTI systems can be greatly simplified with the tools of frequency domain analysis, such as, **Fourier transform**, and **z-transform**.
- To solve differential equations, the **Laplace transform** was one of the best ways to do it. The same thing is true for difference equations. Here, the z-transform is a powerful tool.

End of Lecture 3