# Review of Linear Shift-invariant Systems Theory

The following is a (*very*) brief review of linear systems theory, convolution, and Fourier analysis. I work primarily with discrete signals, but each result developed in this review has a parallel in terms of continuous signals and systems. I assume the reader is familiar with linear algebra (as reviewed in the handout *Geometric Review of Linear Algebra*), and least squares estimation (as reviewed in the handout *Least Squares Optimization*).

## 1 Linear shift-invariant systems

A system is **linear** if (and only if) it obeys the principle of superposition: the response to a weighted sum of any two inputs is the (same) weighted sum of the responses to each individual input.

A system is **shift-invariant** (also called **translation-invariant** for spatial signals, or **time-invariant** for temporal signals) if the response to any input shifted by any amount $\Delta$ is equal to the response to the original input shifted by amount $\Delta$.

These two properties are completely independent: a system can have one of them, both or neither [think of an example of each of the 4 possibilities].

The rest of this review is focused on systems that are *both* linear and shift-invariant (known as **LSI** systems). The diagram below decomposes the behavior of such an LSI system. Consider an arbitrary discrete input signal. We can rewrite it as a weighted sum of impulses (also called "delta functions"). Since the system is linear, the response to this weighted sum is just the weighed sum of responses to each individual impulse. Since the system is shift-invariant, the response to each impulse is just a shifted copy of the response to the first one. The response to the impulse located at the origin (position 0) is known as the system's **impulse response**.

Putting it all together, the full system response is the weighted sum of shifted copies of the impulse response. Note that the system is fully characterized by the impulse response: This is all we need to know in order to compute the response of the system to any input!

To make this explicit, we write an equation that describes this computation:

$$y[n] = \sum_m x[m]r[n-m]$$

This operation, by which input $x$ and impulse response $r$ are combined to generate output signal $y$ is called a **convolution**. It is often written using a more compact notation: $y = x * r$. Although we think of $x$ and $r$ playing very different roles, the operation of convolution is actually commutative: substituting $k = n - m$ gives:

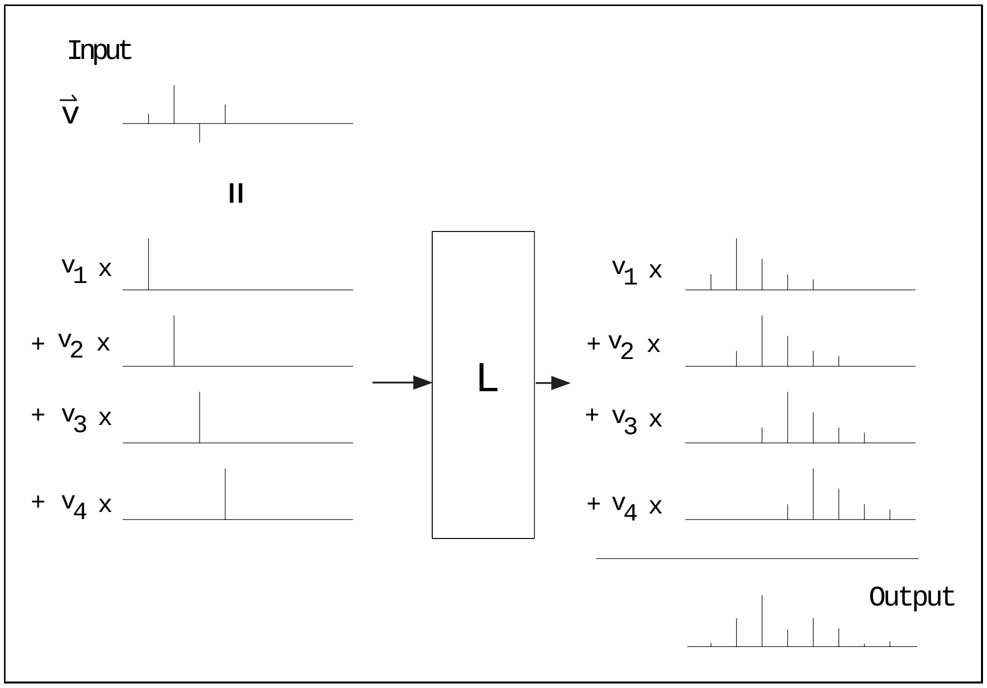
$$y[n] = \sum_k x[n-k]r[k]$$

• Author: Eero Simoncelli, Center for Neural Science, and Courant Institute of Mathematical Sciences, New York University.

• Thanks to Jonathan Pillow for generating some of the figures.

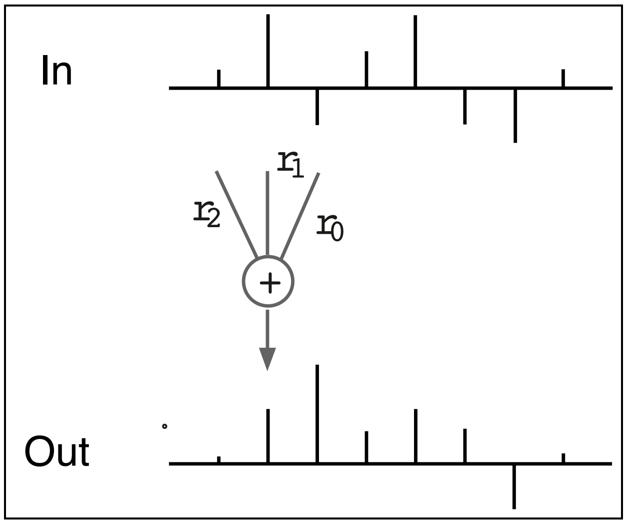
• Created: Fall 2001. Last revised: October 23, 2022.

• Send corrections or comments to eero.simoncelli@nyu.edu

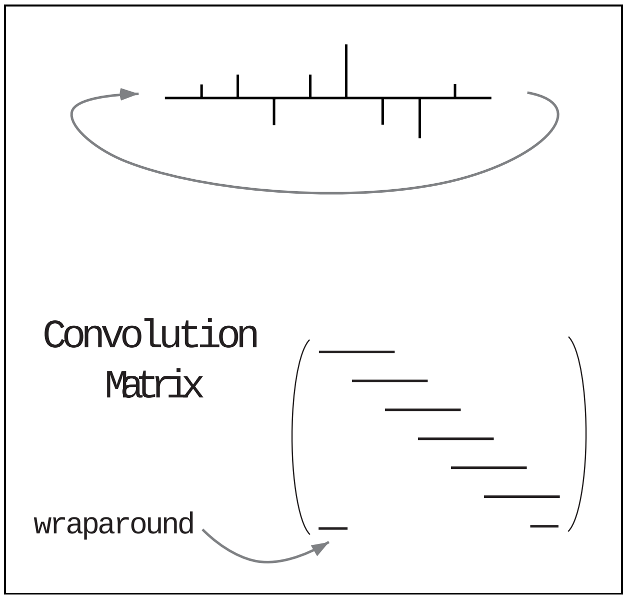


which is just $r * x$. It is also easy to see that convolution is associative: $a * (b * c) = (a * b) * c$.

And finally, convolution is distributive over addition: $a * (b + c) = a * b + a * c$.



Back to LSI systems. The impulse response $r$ is also known as a "convolution kernel" or "linear filter". Looking back at the definition, each component of the output $y$ is computed as an inner product of a chunk of the input vector $x$ with a *reverse-ordered* copy of $r$. As such, the convolution operation may be visualized as a weighted sum over a window that slides along the input signal.



For finite-length discrete signals (i.e., vectors), one must specify how convolution is handled at the boundaries (altenatively, one must define "shift-invariance" so as to explain what happens to samples that are "shifted" to locations beyond the last element of the vector). One solution is to consider each vector to cover one period of an infinitely periodic signal. Thus, for example, when the convolution operation would require one to access an element just beyond the last element of the vector, one need only "wrap around" and use the first element. This is referred to as *circular* or *periodic* boundary handling. There are other methods of handling boundaries. For example, one can pad the signal with zeros (implicitly assumed in Matlab), or *reflect* or *extrapolate* it about the boundary elements.

The convolution operation may be naturally generalized to multidimensional signals. For example, in 2D, both the signal and convolution kernal are two-dimensional arrays of numbers (eg., digital images), and the operation corresponds to taking sums over a 2D window of the signal, with weights specified by the kernel.

## 2 Sinusoids and Convolution

The **sine** function, $\sin(\theta)$, gives the $y$-coordinate of the points on a unit circle, as a function of the angle $\theta$. The **cosine** function $\cos(\theta)$, gives the $x$-coordinate. Thus, $\sin^2(\theta)+\cos^2(\theta) = 1$. The angle, $\theta$, is (by convention) assumed to be in units of radians, and counter-clockwise relative to the horizontal axis. A full sweep around the unit circle corresponds to an angle of $2\pi$ radians.

Now we consider sinusoidal *signals*. A discretized sinusoid can be written as: $x[n] = A\cos(\omega n - \phi)$. Here, $n$ is an integer position within the signal, $\omega$ is the frequency of oscillations (in radians per sample), and $\phi$ is the phase (in radians).

These sinusoidal functions have a unique behavior with respect to LSI systems. Consider input signal $x[n] = \cos(\omega n)$. The response of an LSI system with impuse response $r[n]$ is:

$$\begin{aligned} y[n] &= \sum_k r[k]x[n-k] \\ &= \sum_k r[k]\cos(\omega(n-k)) \\ &= \sum_k r[k][\cos(\omega n)\cos(\omega k) + \sin(\omega n)\sin(\omega k)] \\ &= \left[\sum_k r[k]\cos(\omega k)\right]\cos(\omega n) + \left[\sum_k r[k]\sin(\omega k)\right]\sin(\omega n) \end{aligned}$$

where the third line is achieved using the trigonometric identity $\cos(a-b) = \cos(a)\cos(b) + \sin(a)\sin(b)$. The two sums (in brackets) are just inner products of the impulse response with the cosine and sine functions at frequency $\omega$, and we denote them as $c_r(\omega) = \sum_k r[k]\cos(\omega k)$ and $s_r(\omega) = \sum_k r[k]\sin(\omega k)$. If we consider these two values as coordinates of a two-dimensional vector, we can rewrite them in polar coordinates by defining vector length (amplitude) $A_r(\omega) = \sqrt{c_r(\omega)^2 + s_r(\omega)^2}$ and vector angle $\phi_r(\omega) = \tan^{-1}(s_r(\omega)/c_r(\omega))$. Substituting back into our expression for the LSI response gives:

$$\begin{aligned} y[n] &= c_r(\omega)\cos(\omega n) + s_r(\omega)\sin(\omega n) \\ &= A_r(\omega)\cos(\phi_r(\omega))\cos(\omega n) + A_r(\omega)\sin(\phi_r(\omega))\sin(\omega n) \\ &= A_r(\omega)[\cos(\omega n)\cos(\phi_r(\omega)) + \sin(\omega n)\sin(\phi_r(\omega))] \\ &= A_r(\omega)\cos(\omega n - \phi_r(\omega)) \end{aligned}$$

where the last line is achieved by using the same trigonometric identity as before (but in the opposite direction). Thus: **The response of an LSI system to a sinusoidal input signal is a sinusoid of the same frequency, but (possibly) different amplitude and phase**. The amplitude is multiplied by $A_r(\omega)$, and the phase is shifted by $\phi_r(\omega)$, both of which are derived from the system impulse response $r[n]$. This is true of all LSI systems, and all sinusoidal signals.

**Sinusoids as eigenfunctions of LSI systems.** The relationship between LSI systems and sinusoidal functions may be expressed more compactly (and completely) by bundling together a sine and cosine function into a single complex exponential:

$$e^{i\theta} \equiv \cos(\theta) + i\sin(\theta)$$

where $i = \sqrt{-1}$ is the imaginary number. This rather mysterious relationship (known as Euler's Formula) can be derived by expanding the exponential in a Taylor series, and noting that the even (real) terms form the series expansion of a cosine and the odd (imaginary) terms form the expansion of a sine.

The use of complex numbers may seem unnecessarily abstract. But it allows changes in the amplitude and phase of a sinusoid, and thus the responses of an LSI system to a sinusoid, to be expressed more compactly. Consider input signal $x[n] = e^{i\omega n}$. The response of an LSI system with impuse response $r[n]$ is now:

$$\begin{aligned} y[n] &= A_r(\omega)\cos(\omega n - \phi_r(\omega)) + iA_r(\omega)\sin(\omega n - \phi_r(\omega)) \\ &= A_r(\omega)e^{i(\omega n - \phi_r(\omega)} \\ &= A_r(\omega)e^{-i\phi_r(\omega)}e^{i\omega n} \\ &= A_r(\omega)e^{-i\phi_r(\omega)}x[n]. \end{aligned}$$

The action of an LSI system on the complex exponential function is to multiply it by a single complex number, $A_r(\omega)e^{-i\phi_r(\omega)}$. That is, the complex exponentials are *eigenfunctions* of LSI systems!

## 3 Fourier Transform(s)

A collection of sinusoids may be used as a linear basis for representing signals. The transformation from the standard representation of the signal (eg, as a function of time) to a set of coefficients representing the amount of each sinusoid needed to create the signal is called the **Fourier Transform** (F.T.).

There are really four variants of Fourier transform, depending on whether the signal is continuous or discrete, and on whether its F.T. is continuous or discrete:

| Signal Transform | continuous | discrete |
|---|---|---|
| continuous | **Fourier Transform** | **Discrete-Time(Space) Fourier Transform** |
| discrete | **Fourier Series** | **Discrete Fourier Transform** |

In addition, when a signal/F.T. is discrete, the F.T./signal is *periodic*.

For our purposes here, we'll construct the Discrete Fourier Transform (DFT), which is both periodic and discrete, in both the signal and transform domains. Consider the input domain to be vectors of length $N$, which represent one period of a periodic discrete input signal. We can construct a set of $N$ sinusoidal vectors that are orthogonal to each other [verify]:

$$\begin{aligned} c_k[n] &\equiv \cos\left(\frac{2\pi k}{N}n\right), \qquad k = 0, 1, \ldots \frac{N}{2} \\ s_k[n] &\equiv \sin\left(\frac{2\pi k}{N}n\right), \qquad k = 1, 2, \ldots \frac{N}{2} - 1 \end{aligned}$$

A few comments about this set:

- The vectors above have a squared norm of $N/2$ ($N$ for $k=0$), so dividing them by $\sqrt{N/2}$ ($\sqrt{N}$ for $k=0$) would make them unit vectors. The matrix $F$ formed with these normalized vectors as columns would be orthogonal, with an inverse equal to its transpose. But many definitions/implementations of the DFT choose to normalize the vectors differently. For example, in matlab, the fft function does not include any normalization factor, but the inverse (ifft) function then normalizes by $2/N$ ($1/N$ for $k=0$).

- these vectors come in sine/cosine pairs for each frequency *except* for the first and last frequencies ($k=0$, and $k=N/2$), for which the sine vector would be zero). If $N$ is odd, the vector at frequency $k=N/2$ is omitted.

- Notice that if we included vectors for additional values of $k$, they would be redundant. In particular, the vectors associated with any particular $k$ are the same as those for $k+mN$ for any integer $m$. That is, the DFT, indexed by $k$, is periodic with period $N$. Moreover, the vectors associated with $-k$ are the same as those associated with $k$, except that all of the the sine functions are negated (sine is an *anti-symmetric* function).

Since this set of $N$ sinusoidal vectors are orthogonal to each other, they span the full input space, and we can write any vector $\vec{v}$ as a weighted sum of them:

$$v[n] = \sum_{k=0}^{N/2} a_k c_k[n] + \sum_{k=1}^{N/2-1} b_k s_k[n]$$

Since the basis is orthogonal, the Fourier coefficients $\{a_k, b_k\}$ may be computed by projecting the input vector $\vec{v}$ onto each basis function:

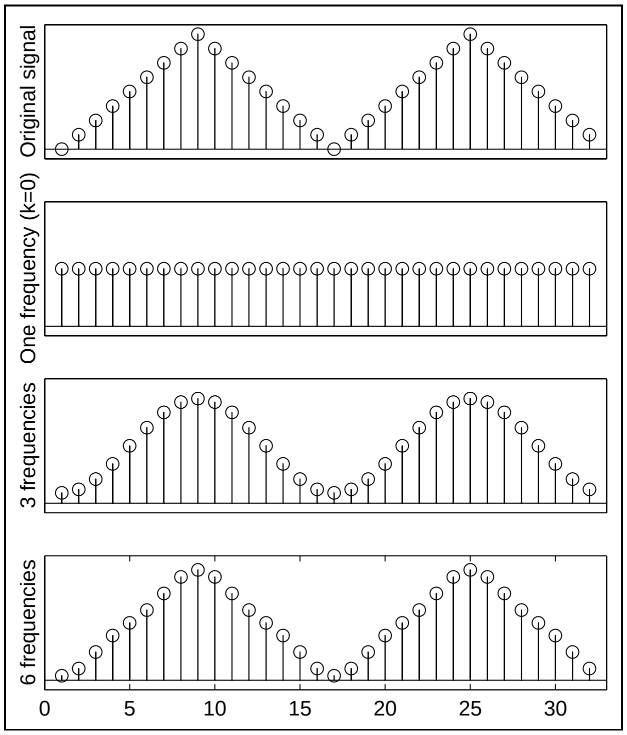
$$\begin{aligned} a_k &= \sum_{n=0}^{N-1} v[n] c_k[n] \\ b_k &= \sum_{n=0}^{N-1} v[n] s_k[n] \end{aligned}$$

In matrix form, we can write $\vec{v} = F(F^T \vec{v})$ (assuming normalization as described in the first bullet above).

Now, using the properties of sinusoids developed earlier, we can combine cosine and sine terms into a single phase-shifted sinusoid:

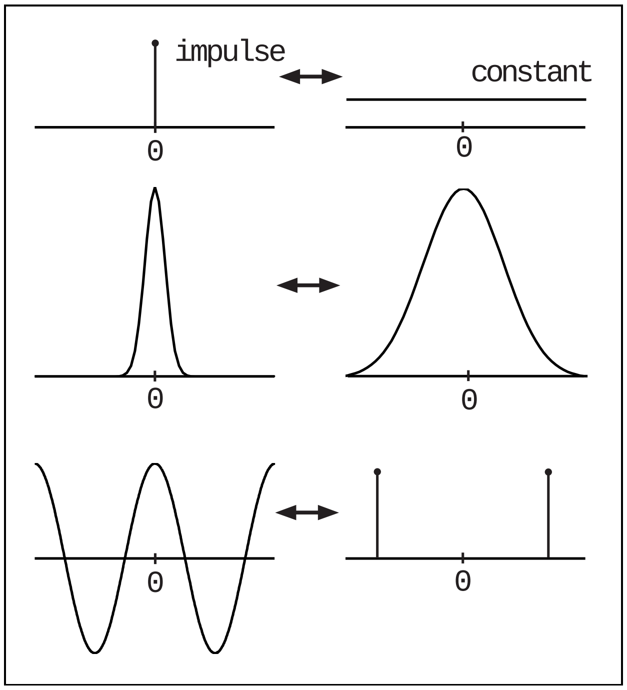
$$v[n] = \sum_{k=0}^{N/2} A_k \sin\left(\frac{2\pi k}{N} n - \phi_k\right)$$

with amplitudes $A_k = \sqrt{a_k^2 + b_k^2}$, and phases $\phi_k = \tan^{-1}(b_k/a_k)$. These are are referred to as the **Fourier amplitudes** (or magnitudes) and **Fourier phases** of the signal $v[n]$. Again, this is just a polar coordinate representation of the original values $\{a_k, b_k\}$.



At the right is an illustration of successive approximations of a triangle wave with sinusoids. The top panel shows the original signal. The next shows the approximation one gets by projecting onto a single zero-frequency sinusoid (i.e., the constant function). The next shows the result with three frequency components, and the last panel shows the result with six. [Try matlab's `xfourier` to see a live demo with square wave.]

The standard representation of the Fourier coefficients uses a complex-valued number to represent the amplitude and phase of each frequency component, $A_k e^{i\phi_k}$. The Fourier amplitudes and phases correspond to the amplitude and phase of this complex number.



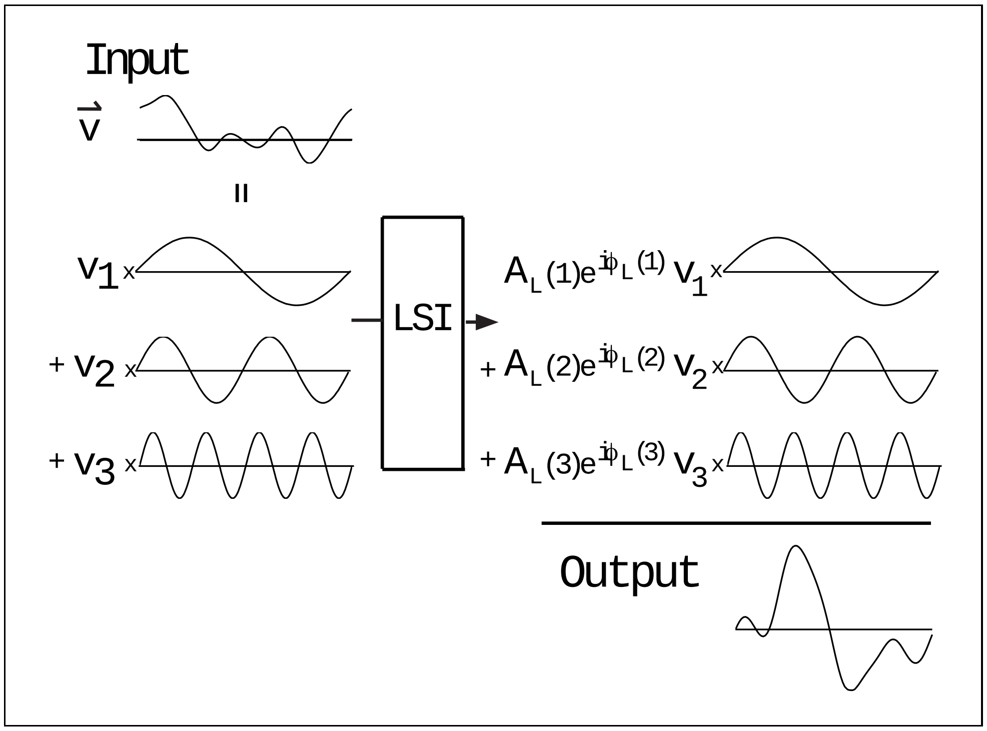
Shown in the left column are three simple signals: an impulse, a Gaussian, and a cosine function. In the right column are their Fourier amplitude spectra (the amplitudes plotted as a function of frequency number $k$) for These are shown here symmetrically arranged around frequency $k = 0$, but some authors plot only the positive half of the frequency axis. Note that the cosine function is constructed by adding a complex exponential to its frequency-negated cousin - this is why the Fourier transform shows two impulses.

**Stretch (dilation) property.** If we stretch the input signal (i.e., rescale the x-axis by a factor $\alpha$), the Fourier transform will be compressed by the same factor (i.e., rescale the frequency axis by $1/alpha$). Consider a Gaussian signal. The Fourier amplitude is also a Gaussian, with standard deviation *inversely* proportional to that of the original signal.

**Shift property.** When we shift an input signal, each sinusoid in the Fourier representation must be shifted. Specifically, shifting by $m$ samples means that the phase of each sinusoid changes by $\frac{2\pi k}{N}m$, while the amplitude is unchanged. Note that the phase change is different for each frequency $k$.

## 4 Convolution Theorem

The most important property of the Fourier representation is its relationship to LSI systems and convolution. To see this, we need to combine the eigenvector property of complex exponentials with the Fourier transform. The diagram below illustrates this. Consider applying an LSI system to an arbitrary signal. Use the Fourier transform to rewrite it as a weighted sum of sinusoids. The weights in the sum may be expressed as complex numbers, $A_k e^{i\phi_k}$, representing the amplitude and phase of each sinusoidal component. Since the system is linear, the response to this weighted sum is just the weighted sum of responses to each of the individual sinusoids. But the action of an LSI on a sinusoid with frequency number $k$ will be to multiply


Input
$\vec{v}$
=
$v_1 \times$
$+ v_2 \times$
$+ v_3 \times$
LSI
$A_L(1)e^{i\phi_L(1)}\, v_1 \times$
$+ A_L(2)e^{i\phi_L(2)}\, v_2 \times$
$+ A_L(3)e^{i\phi_L(3)}\, v_3 \times$
Output

the amplitude by a factor $A_r(k)$ and shift the phase by an amount $\phi_r(k)$. Finally, the system response is a sum of sinusoids with amplitudes/phases corresponding to

$$(A_r(k)A_k)e^{i(\phi_r(k)+\phi_k)} = (A_r(k)e^{i\phi_r(k)})(A_k e^{i\phi_k}).$$

Earlier, using a similar sort of diagram, we explained that LSI systems can be characterized by their impulse response, $r[n]$. Now we have seen that the complex numbers $A_r(k)e^{i\phi_r(k)}$ provide an alternative characterization. We now want to find the relationship between these two characterizations. First, we write an expression for the convolution (response of the LSI system):

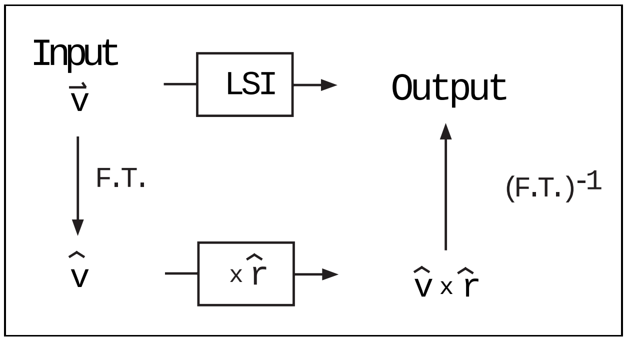
$$y[n] = \sum_m r[m]x[n-m]$$

Now take the DFT of both sides of the equation:

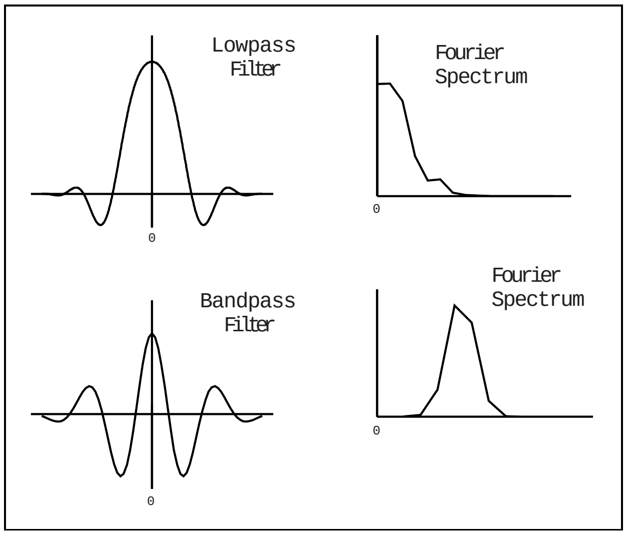
$$\begin{aligned}
Y[k] &= \sum_n y[n]e^{i2\pi nk/N} \\
&= \sum_n \sum_m r[m]x[n-m]e^{i2\pi nk/N} \\
&= \sum_m r[m] \sum_n x[n-m]e^{i2\pi nk/N} \\
&= \sum_m r[m]e^{i2\pi mk/N} X[k] \\
&= R[k]X[k]
\end{aligned}$$

This is quite amazing: the DFT of the LSI system response, y[n], is just the product of the DFT of the input and the DFT of the impulse response! That is, the complex numbers $A_r(k)e^{i\phi_r(k)}$ correspond to the Fourier transform of the function $r[n]$.

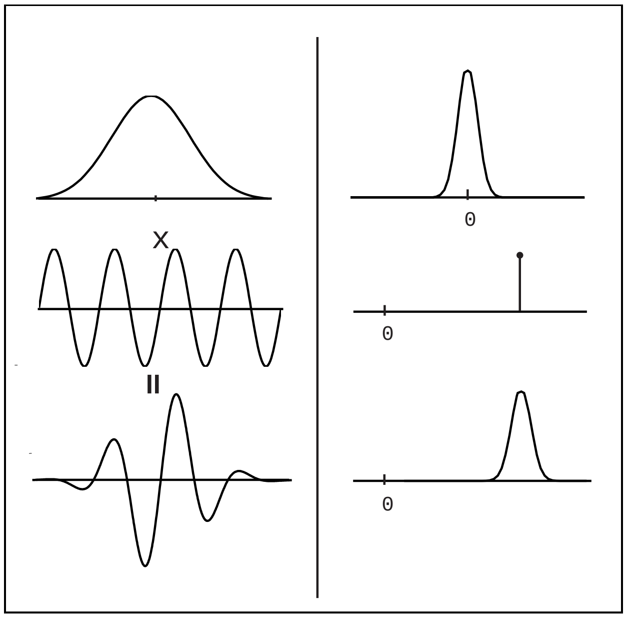
Summarizing, the response of the LSI system may be computed by a) Fourier-transforming the input signal, b) multiplying each Fourier coefficient by the associated Fourier coefficient of the impulse response, and c) Inverse Fourier-transforming. A more colloquial statement of this **Convolution theorem** is: "convolution in the signal domain corresponds to multiplication in the Fourier domain". Reversing the roles of the two domains (since the inverse transformation is essentially the same as the forward transformation) means that "multiplication in the signal domain corresponds to convolution in the Fourier domain".



Why would we want to bother going through three sequential operations in order to compute a convolution? Conceptually, multiplication is easier to understand than convolution, and thus we can often gain a better understanding of an LSI by thinking about it in terms of its effect in the Fourier domain. More practically, there are very efficient algorithms for the Discrete Fourier Transform, known as the *Fast Fourier Transform (DFT)*, such that this three-step computation may be more computationally efficient than direct convolution.



As an example of conceptual simplification, consider two impulse responses, along with their Fourier amplitude spectra. It is often difficult to anticipate the behavior of these systems solely from their impulse responses. But their Fourier transforms are quite revealing. The first is a **lowpass** filter meaning that it discards high frequency sinusoidal components (by multiplying them by zero). The second is a **bandpass** filter - it allows a central band of frequencies to pass, discarding the lower and higher frequency components.



As another example of conceptual simplification, consider an impulse response formed by the product of a Gaussian function, and a sinusoid (known as a **Gabor function**). How can we visualize the Fourier transform of this product? We need only compute the convolution of the Fourier transforms of the Gaussian and the sinusoid. The Fourier transform of a Gaussian is a Gaussian. The Fourier transform of a sinusoid is an impulse at the corresponding frequency. Thus, the Fourier transform of the Gabor function is a Gaussian centered at the frequency of the sinusoid.