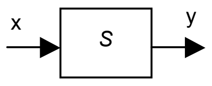
Convolution as useful tool for Engineer

Author: Gianluca Biccari
Github: gl051

INTRODUCTION

Consider a system where *x* is the input signal, *y* is the output signal (both can be continuos-time, discrete time, finite energy or any mixture of these) and *S* is a generic operator to define the system behavior:



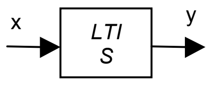
y = *S*(x)

A time invariant system (abr. LTI) is a subclass of *S* that is both linear and time/translation invariant:

- LINEAR
  $S(\alpha x_1 + \beta x_2) = \alpha S(x_1) + \beta S(x_1) = \alpha\, y_1 + \beta\, y_1$

- TIME INVARIANT
  $y(t) = S(x(t))$
  $y(t-\tau) = S(x(t-\tau))$

Most of real (physical) systems where an electrical engineer is involved are linear and time invariant, they are widely applied in pratical applications.
For an LTI system the convolution is the fundamental way to describe its behaviour in the time domain (in the same way as it is Fourier transform for frequency domain)[1]:



y = *S*(x) = x(t) * h(t)

Where h(t) is called the impulsive respose of the system (defined as $x(t) \equiv \delta(t) \leftrightarrow y(t) \equiv h(t)$).
In other words convolution provides the mathematical backbone for system time analysis.
In the next paragraph is proposed a mathematical definition of convolution and its properties. Then the focus will be on discrete signals and the methods to compute convolution, as last we'll analyze how matlab compute convolution.

CONVOLUTION

Convolution is a formal mathematical operation, just as multiplication, addition and integration that take two function, x(t) and h(t), both integrable on ***R*** space, and produces a third function y(t), the formula is:

$$y(t) = x(t) * h(t) = \int_{-\infty}^{+\infty} h(\tau)\, x(t-\tau)\, d\tau$$

What follows from this definition is that a convolution between x(t) and h(t), in time domain, is equivalent to a product of X(f) and H(f) in frequency domain [2]:

$$y(t) = x(t) * h(t) \quad \Leftrightarrow \quad Y(f) = X(f)H(f)$$

(Time domain) *(Frequency domain)*

CONVOLUTION PROPERTIES

- DELTA FUNCTION
  $x(t) * \delta(t) = x(t)$
  $x(t) * k\,\delta(t) = k\,x(t)$
  $x(t) * \delta(t+k) = x(t+k)$

- COMMUTATIVE PROPERTY
  $x(t) * h(t) = h(t) * x(t)$

- ASSOCIATIVE PROPERTY
  $a(t) * [b(t) * c(t)] = [a(t) * b(t)] * c(t)$

- DISTRIBUTIVE PROPERTY
  $a(t) * b(t) + a(t) * c(t) = a(t) * (b(t) + c(t))$

- SCALINE PROPERTY
  $y(t) = h(t) * x(t)$
  then
  $h(t/k) * x(t/k) = |k|\, y(t/k)$

- FOURIER PROPERTY
  $x(t) * h(t) = H(f) * X(f)$

- LINEAR CHANGE BETWEEN INPUT AND OUTPUT
  $y(t) = h(t) * x(t)$
  L linear change (i.e. derivate)
  $L(y(t)) = L(h(t)) * x(t) = h(t) * L(x(t))$
  (i.e. → $y'(t) = x'(t) * h(t) = x(t) * h'(t)$)

DISCRETE CONVOLUTION

If we think to x(t) and h(t) as signals instead of functions and we sample them at the same rate for a finite period of time, we get a couple of array where x[n] is a *NX* points vector, and h[n] is an *NH* points vector. The convolution of the two is the *NX*+*NH*-1 points vector y[n]:

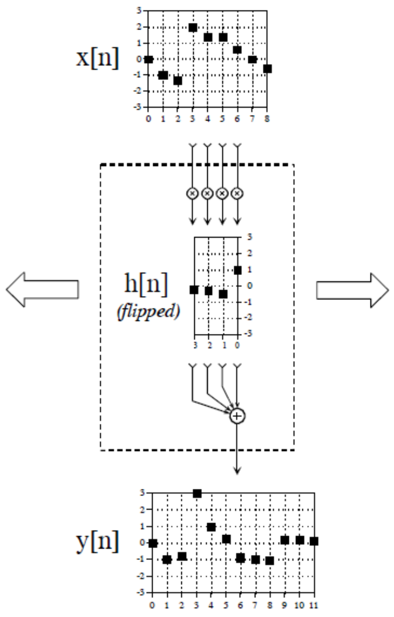
$$y(n) = x(n) * h(n) = \sum_{j=0}^{j=NH-1} h(j)\,x(n-j)$$

x[n] @ *NX* points
h[n] @ *NH* points
y[n] @ *NX*+*NH*-1 points
(n runs from 0 to length-1)

This equation is called the **convolution sum**, it provides a description of how a discrete system works on any sampled (digital) input signal. The system is completely characterized by its h(n) which is often called **impulse response** or **convolution kernel**.

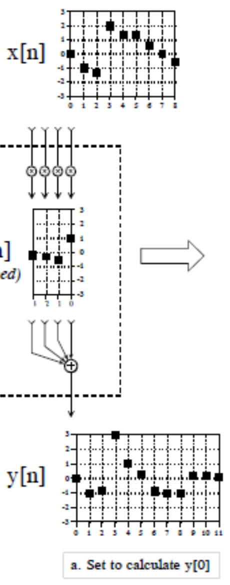
CONVOLUTION MACHINE

To understand how convolution works on a couple of signals let us borrow the convolution machine describe in [3]. We need to look at the next figure and think of the input signal x[n] and output signal y[n] as fixed on the page. The convolution machine, everything inside the dashed box, is free to move left and right as needed. It must be positioned so that its output is aligned with the output sample being calculated. The input values are multiplied by the indicated samples in the impulse response, and the products are added. This produce the value for the output signal, which drops into its proper place. The next output sample will be calculated after a one right shift of the convolution machine.



(fig 1 Convolution machine)

The arrangement of the impulse response inside the convolution machine is very important, the impulse response must be flipped left-to-right, the filp falls out of the mathematical definition of convolution.
At the starting point of convolution process, the convolution machine is located fully to the left with its output aligned at y[0],in this position is trying to receive input from samples with negative index. One way to handle this problem is by inventing the nonexistent samples, this is called **padding** the signal with a well know value (most of the time this value is zero).



(fig.2 Padding of x[-3], x[-2], x[-1] at zero)

HOW TO COMPUTE DISCRETE CONVOLUTION

There are several different approaches that may be used to perform convolution and the one that is the easiest will depend upon the form and type of sequences that are to be convolved.

• **DIRECT EVALUATION**

When the sequences that are being convolved may be described by simple closed-form mathematical expressions, the convolution is often most easily performed by directly evaluating the convolution sum.
Let's see an example:

x(n) = a^n , n>0
x(n) = 0 , otherwise

h(n) = 1 , n>0

with the direct evaluation of the convolution sum we find:

$$y(n) = \sum_{j=0}^{j=\infty} h(j)x(n-j) =$$

$$= \sum_{j=0}^{j=\infty} x(j)h(n-j) =$$

$$= \sum_{j=0}^{\infty} a^j h(n-j) = \sum_{j=0}^{n} a^j = \frac{1-a^{n+1}}{1-a}$$

- **CONVENTIONAL METHOD**

it consists of implementing the convolution machine:
1) flip h(n)
2) Shift it on x(n) to the right by n
3) Sum up the term-by-term of x(j) and h(n-j) to get y(n)

Let's see an example:

x = {1, -2, 3, 5}
h = {1, 3, 4}
y(2) = ?

1) h = {4, 3, 1}

2) 1 -2 3 5
    4   3 1

3) y(2) = 4 - 6 + 4 = 2

- **MULTIPLICATION METHOD**

If h(n) and x(n) are both of finite duration it's possible to compute the whole convolution by adopting a very efficient method that consist in proceding with a multiplication scheme without borrow:

$$\begin{array}{cccccccc}
 & & & x_0 & x_1 & \dots & x_{NX-2} & x_{NX-1} \\
 & & & & h_0 & h_1 & \dots & h_{NX-1} \\
\hline
 & & & p_0 & p_1 & \dots & p_{NX-2} & p_{NX-1} \\
 & & p_0 & p_1 & \dots & p_{NX-2} & p_{NX-1} & - \\
 & & \dots & & \dots & & & \\
p_0 & p_1 & \dots & p_{NX-2} & p_{NX-1} & - & & \\
\hline
y_0 & y_1 & y_3 & \dots & & & & y_{NX+NH-1}
\end{array}$$

Let's see an example:

x = {1, -2, 3, 5}
h = {1, 3, 4}

$$\begin{array}{ccccccl}
 & & 1 & -2 & 3 & 5 & \leftarrow x(n) \\
 & & & 1 & 3 & 4 & \leftarrow h(n) \\
\hline
 & & 4 & -8 & 12 & 20 & \\
 & 3 & -6 & 9 & 15 & - & \\
1 & -2 & 3 & 5 & - & & \\
\hline
1 & 1 & 1 & 6 & 27 & 20 & \leftarrow y(n)
\end{array}$$

CONVOLUTION IN MATLAB

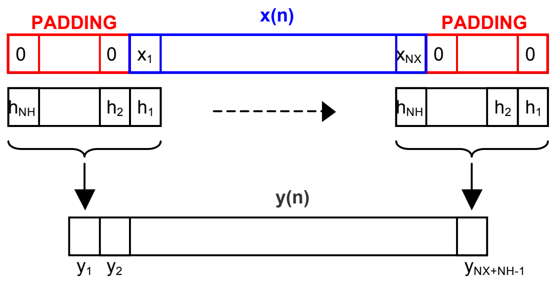
Matlab is one of the most important program used by engineers during the flow design of digital system. It can be a good exercise to understand how Matlab performs convolution calculus.
First of all we have to keep in mind that *x* and *y* are vector indexed by positive integers (running from one and not zero as seen until now). After this consideration the convolution sum can be re-written as:

$$y(n) = x(n) * h(n) = \sum_{j=1}^{j=NH} h(j)x(n-(j-1))$$

and it is performed by the *conv* function:

y = conv(x,h)

which implements a convolution machine scheme:

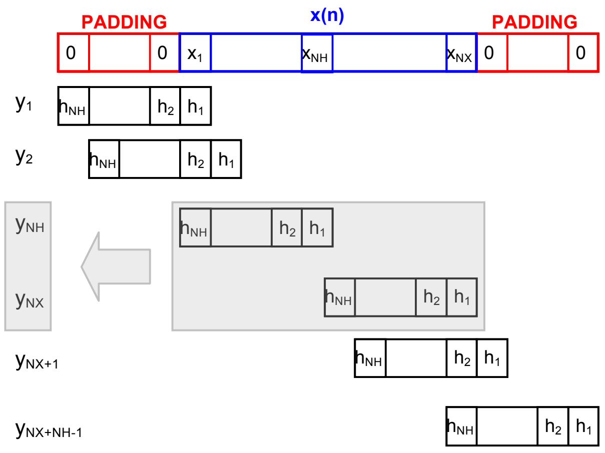


(fig.3 conv function)

Most of the time convolution is used to describe a signal processing performed by a (digital) system, in this case we would like to work with signal of the same length. That means that we'd like to get the y output with *NX* points as it is for x. We are only interest in how convolution modify the original input signal x, for example think of a kernel function, h(n), where all elements are equal, it is the case of an average filter on the input signal x where we only care to the convolved signal y as a filtered version of x.
Upon this consideration can be useful look back at convolution machine and remember that *conv* entails:

- y is length *NX*+*NH*-1
- y(1) is the first element of y
- y(end) = y(*NX*+*NH*-1) is the last element of y

If we look at the figure 4 we can assert that the first valid sample of y (where not padding values are being used to compute y output), it will be at *NH* position of y(n) vector. On the other side, the last valid value of y it will be at *end – (NH-1) = NX* position.



(fig.4 conv function)

- y_{NH} *:* is the first sample valid.
- y_{NX} *:* is the last sample valid.

The really "good" values in y are between *NH* and *NX* index, so we have *NX-NH*+1 points calculated with not padded value. To obtain an *NX* points vector in output it's necessary to add *NH*-1 points in the head, for example

y @ *NX* points := [y(*NH*)*ones(1,*NH*-1),y(*NH*:*NX*)]

or

y @ *NX* points := [y(1:*NX*)]

or

y @ *NX* points := [y(1:*NX*)] , with x padded with x(1) value.

If you think at impulse response as a set of weighing coefficients to apply to x vector each sample in the output signal is equal to a sum of weigthed inputs, so each sample in the output signals is influenced by a region of samples in the input signal. Viewing the convolution machine in this way , the coefficients can be choosen odd and symmetrically:

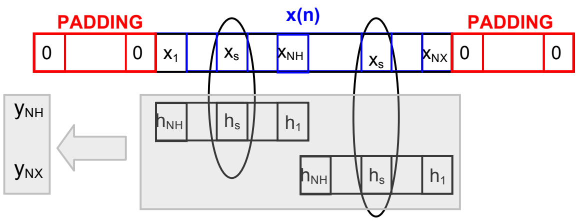
h @ *NH* points, where *NH* is odd

H*NH* := (*NH*-1)/2

s := H*NH* + 1

h(s+1) = h(s-1)
h(s+2) = h(s-2)
…
h(s+H*NH*) = h(s-H*NH*)

Anyway mathematically there is only one concept of convolution, what is different is the approach to interpret the output:



(fig.5 symmetric kernel)

The convolution output is always the same, it's different the way to think about the input sample processed by the convolution machine, in this case:

- y_{NH} *:* is obtained by the x_s weigted with the kernel coefficients value symmetrically spread around it.
- y_{NX} *:* is obtained by the $x_{NX\text{-}S+1}$ weigted with the kernel coefficients points symmetrically spread around it.

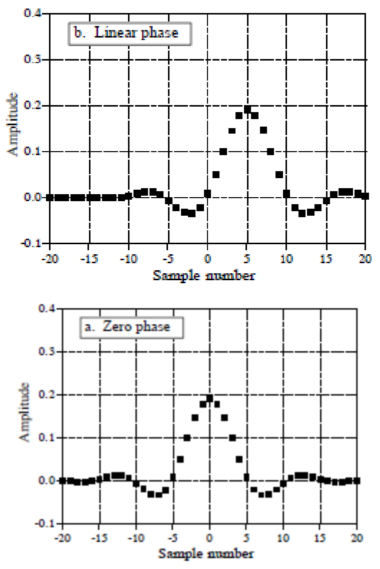
The really "good" values are still *NX-NH*+1 points, but it's different how to obtain an *NX* points in output:

y @ *NX* points :=
[y(*NH*)* ones(1,H*NH*), y(*NH*:*NX*), y(*NX*)* ones(1,H*NH*)]

or

y @ *NX* points := [y(S:*NX*+S)],
with x padded with x(1) and x(*NX*) values.

In the case of a right-left symmetry, around some point other than zero the system is called **linear phase**, that means that the output signal is shifted by the same quantity respect of input signal, the amount of this shift is H*NH*.
To change the system behaviour in a zero phase it's only necessary to shift the kernel sample so that there is left-right symmetry around sample zero (figure 6). This solution it's not possible in Matlab where all indexes must be real and positive, but you can always shift output signal so that you will have the same attended result.



(fig.6 linear phase and zero phase kernel)

Bibliography

[1] **Signal and System Modelling**, Gianluca Biccari

[2] **The transforms and applications handbook**, Alexander D.Poularikas, CRC PRESS, IEEE PRESS.

[3] **The Scientist and Engineer’s Guide to Digital Signal Processing**, Steven W. Smith, California Technical Publishinf (San Diego).