DSP notes 1

Malcolm

Started April 11th 2024

Contents

0.1 Convolution algorithm

We consider two algorithms, the first *input* algorithm considers convolution from the viewpont of the input signal—how each input sample contributes to multiple points. The second considers *output* algorithm does the same but for the output signal—how an output sample has received information from multiple input samples.

The first perspective demonstrates the conceptual understanding of convolution, while the second the mathematics. Recall the (continuous) definition of the convolution:

$$(f * g)(t) = \int_{0^-}^{t^+} f(\tau)g(t-\tau)d\tau \quad \text{for } t > 0$$

Input side algorithm
Consider a 9 point input signal $x[n]$ passed through a system with a 4 point impulse response $h[n]$. The input $x[n]$ is convolved with $h[n]$ to produce $y[n]$, the system respose to $x[n]$. The convolution amounts to *scaling and shifting the impulse response according to each input sample*, and then *adding them together*.

For example, lets say input sample number 4 is $x[4] = 1.4$; this contributes an output component of $1.4h(n-4)$, essentially scale the impulse response by 1.4 and shift it four samples to the right. (The input can be represented as $1.4\delta[n = 4]$, which after passing through the system becomes $1.4h[n-4]$ (linearity)).

This is repeated for all the input points, so we have 9 output components for 9 input points. We add these components together to get the output.
(next page)

Consider these figures, squares represent the data points that come from the shifted and scaled input response, and diamonds for the added zeros:

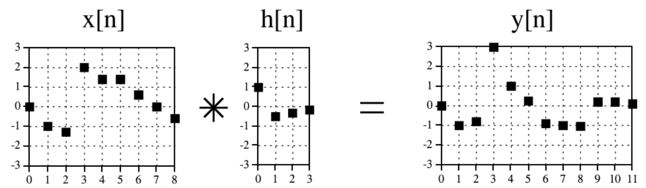

FIGURE 6-5
Example convolution problem. A nine point input signal, convolved with a four point impulse response, results in a twelve point output signal. Each point in the input signal contributes a scaled and shifted impulse response to the output signal. These nine scaled and shifted impulse responses are shown in Fig. 6-6.

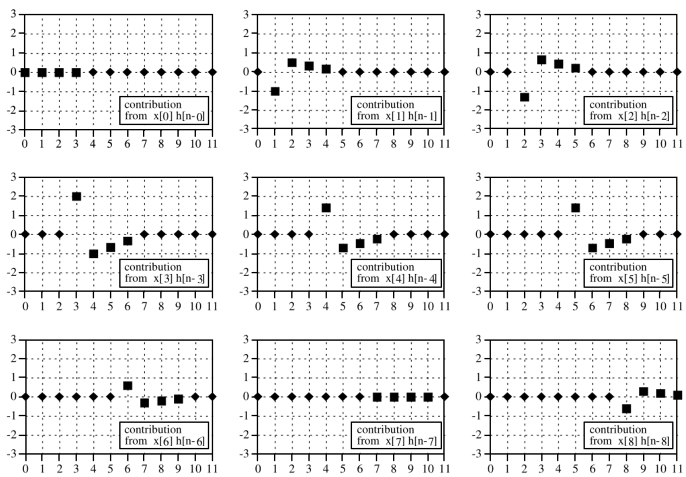

FIGURE 6-6
Output signal components for the convolution in Fig. 6-5. In these signals, each point that results from a scaled and shifted impulse response is represented by a square marker. The remaining data points, represented by diamonds, are zeros that have been added as place holders.

The convolution is the resulting sum of the nine scaled and translated impulse responses.

To relate this back to the math, intuitively the fourth output point depends on the fourth point of the impulse response scaled by the first point of the input, and the third point of the impulse response scaled by the second point of the input and so on.
(next page)

Input side program

The program shown here convolves an 81 point input signal held in array X with a 31 point impulse response held in array H to produce output array Y:

```
100  'CONVOLUTION USING THE INPUT SIDE ALGORITHM
110                                  '
120 DIM X[80]                        'The input signal, 81 points
130 DIM H[30]                        'The impulse response, 31 points
140 DIM Y[110]                       'The output signal, 111 points
150                                  '
160 GOSUB XXXX                       'Mythical subroutine to load X[ ] and H[ ]
170                                  '
180 FOR I% = 0 TO 110                'Zero the output array
190   Y(I%) = 0
200 NEXT I%
210                                  '
220 FOR I% = 0 TO 80                 'Loop for each point in X[ ]
230   FOR J% = 0 TO 30               'Loop for each point in H[ ]
240     Y[I%+J%] = Y[I%+J%] + X[I%]*H[J%]
250   NEXT J%
260 NEXT I%                          '(remember, * is multiplication in programs!)
270                                  '
280 GOSUB XXXX                       'Mythical subroutine to store Y[ ]
290                                  '
300 END
```

TABLE 6-1

See that the first for loop loops through the *input*, so it looks at each input sample, then scales each point of the impulse response (in the second for loop) by that input sample, accordingly incrementing the related points in the output array.

Lastly, since the convolution is commutative, we can swap the input and the impulse response and get the same solution:

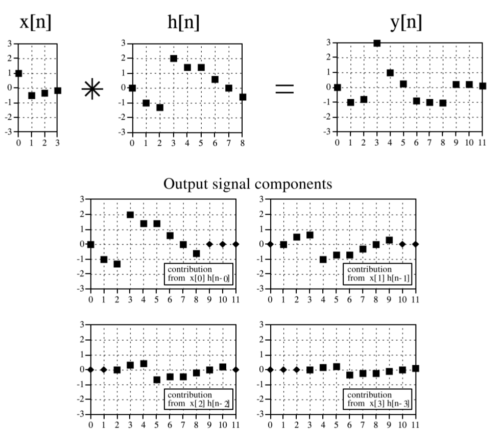

FIGURE 6-7
A second example of convolution. The waveforms for the input signal and impulse response are exchanged from the example of Fig. 6-5. Since convolution is commutative, the output signals for the two examples are identical.

(next page)

Output side algorithm
Now we consider the output side algorithm—how a single output point receives contribution from multiple input points. First recall the discrete convolution definition:

$$y[i] = (x * h)[i] = \sum_{j=0}^{M-1} h[j]x[i-j]$$

Where h is an M point signal running from 0 to $M-1$. (Recall this idea: intuitively the nth output point depends on the nth point of the impulse response scaled by the first point of the input (meaning $h[0] \cdot x[n-0]$), and the $(n-1)$th point of the impulse response scaled by the second point of the input (meaning $h[1] \cdot x[n-1]$) and so on.

Now see that this can be represented as a 'convolution machine', where the inpulse response is *flipped left-for-right* and the dot product with the input is taken such that its output aligns with the output sample being calculated:

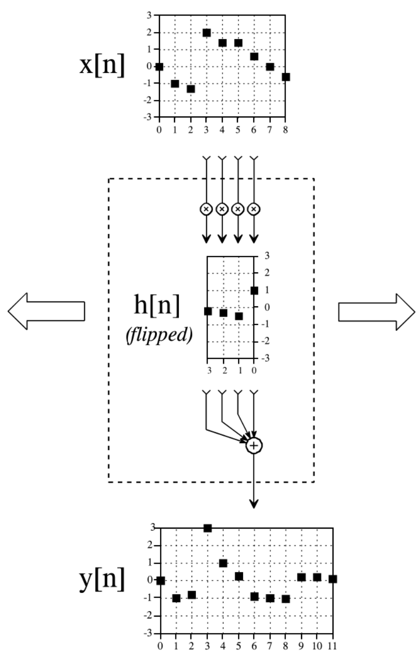

FIGURE 6-8
The convolution machine. This is a flow diagram showing how each sample in the output signal is influenced by the input signal and impulse response. See the text for details.

In this instance, $y[6]$ is being calculated from the input samples $x[3], x[4], x[5]$ and $x[6]$. To calculate $y[7]$, the 'convolution machine' moves one sample to the right.
(next page)

Nonexistent samples

See a complcation that arises: when the 'convolution machine' is located fully to the left or right, say $y[0]$, it is trying to receive input from samples that don't exist:

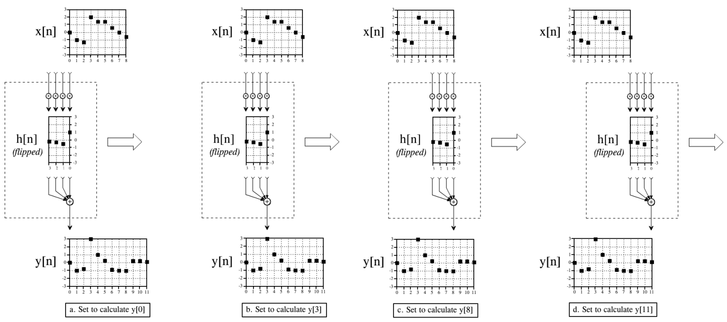

FIGURE 6-9
The convolution machine in action. Figures (a) through (d) show the convolution machine set to calculate four different output signal samples, y[0], y[3], y[8], and y[11].

Figure 6-9 (continued)

This can be handled by inventing the nonexistent samples or zero padding.

Output side program

Appendix A

Mathematics (copied from appendices)

A.0.1 Convolution—Definition and properties

Definition
The *convolution* of two functions f and g, denoted by $f * g$, is defined as

$$(f * g)(t) = \int_{0^-}^{t^+} f(\tau)g(t-\tau)d\tau \quad \text{for } t > 0$$

This is a *one-sided* convolution, which is only concerned with functions on the interval $(0^-, \infty)$.

Properties
Linearity; for functions f_1, f_2, g and constants c_1, c_2:

$$(c_1 f_1 + c_2 f_2) * g = c_1(f_1 * g) + c_2(f_2 * g)$$

This follows from the linearity of integration.

Commutivity: $f * g = g * f$. This follows from the change of variable $v = t - \tau$; the limits become

$$\tau = 0^- \implies t - \tau = t^+ \quad \text{and} \quad \tau = t^+ \implies t - \tau = 0^-$$

Associativity: $f * (g * h) = (f * g) * h$. Showing this amounts to changing the order of integration. First consider the discrete case:

$$
\begin{aligned}
((f * g) * h)(n) &= \sum_{k=0}^{n} (f * g)(k) h(n-k) \\
&= \sum_{k=0}^{n} \left(\sum_{i=0}^{k} f(l) g(k-l) \right) h(n-k) \\
&= \sum_{0 \leq l \leq k \leq n} f(l) g(k-l) h(n-k) \\
&= \sum_{l=0}^{n} \sum_{k=l}^{n} f(l) g(k-l) h(n-k) \\
&= \sum_{l=0}^{n} f(l) \left(\sum_{k=0}^{n-1} g(k) h(n-k-l) \right) \\
&= \sum_{l=0}^{n} f(l) (g * h)(n-l) \\
&= (f * (g * h))(n)
\end{aligned}
$$

(next page)

Properties cont.
The continuous case is analogously:

$$\begin{aligned}
((f*g)*h)(t) &= \int_0^t (f*g)(s)h(t-s)ds \\
&= \int_{s=0}^t \left(\int_{u=0}^s f(u)g(s-u)du\right) h(t-s)ds \\
&= \iint_{0\le u\le s\le t} f(u)g(s-u)h(t-s)du\,ds \\
&= \int_{u=0}^t f(u)\left(\int_{s=0}^{t-u} g(s)h(t-u-s)ds\right) du \\
&= \int_{u=0}^t f(u)(g*h)(t-u)du \\
&= (f*(g*h))(t)
\end{aligned}$$

Delta functions
See that

$$(\delta * f)(t) = f(t) \quad \text{and} \quad (\delta(t-a)*f)(t) = f(t-a)$$

These can be shown via direct computation. Recall that for $b > 0$

$$\int_{0^-}^b \delta(\tau)f(\tau)d\tau = f(0)$$

So it follows that for $t \ge 0$

$$\begin{aligned}
(\delta * f)(t) &= \int_{0^-}^{t^+} \delta(\tau)f(t-\tau)d\tau = f(t-0) = f(t) \\
(\delta(t-a)*f)(t) &= \int_{0^-}^{t^+} \delta(\tau-a)f(t-\tau)d\tau = f(t-a)
\end{aligned}$$

(for the second statement see that the delta function only has magnitude for $\tau = a$)

A.0.2 Green's Formula

Definition
Suppose we have the linear time invariant system with rest initial conditions:

$$p(D)y = f(t), \quad y(t) = 0 \text{ for } t < 0$$

Suppose that $w(t)$ is the unit impulse response (also called the *weight* function) for the above system. That is, $w(t)$ satisfies $p(D)w = \delta(t)$, with rest initial conditions. *Green's formula* states that for any input $f(t)$ the solution to that system is given by

$$y(t) = (f * w)(t) = \int_{0^-}^{t^+} f(\tau)w(t-\tau)d\tau$$

This means we can find the response to *any* input once we know the unit impulse response. It is also an integral, which can be computed numerically if necessary.
Intuition
Recall that linear time invariance means

$$y(t) \text{ solves } p(D)y = f(t) \implies y(t-a) \text{ solves } p(D)y = f(t-a)$$

(If $y(t)$ is the response to input $f(t)$ then $y(t-a)$ is the response to input $f(t-a)$.) First consider partitioning time into intervals of width Δt; so $t_0 = 0, t_1 = \Delta t, t_2 = 2\Delta t$, etc.:

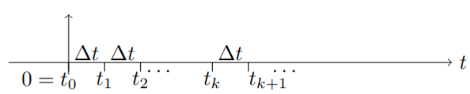

Figure 1: Division of the t-axis into small intervals.

Next we decompose the input signal $f(t)$ into packets over each interval. The kth signal packet, $f_k(t)$ coincides with $f(t)$ between t_k and t_{k+1} and is 0 elsewhere:

$$f_k(t) = \begin{cases} f(t) & \text{for } t_k < t < t_{k+1} \\ 0 & \text{elsewhere} \end{cases}$$

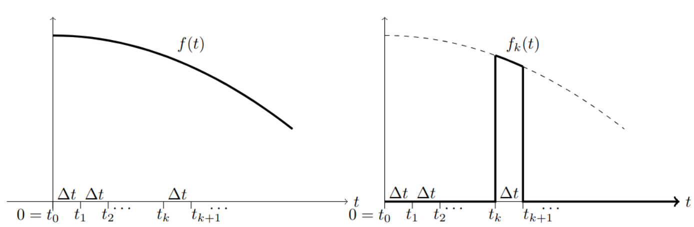

Figure 2: The signal packet $f_k(t)$.

(next page)

Intuition cont.
Naturally for $t > 0$ we have $f(t)$ as the sum of the packets

$$f(t) = f_0(t) + f_1(t) + \ldots + f_k(t) + \ldots$$

Given that a single packet $f_k(t)$ is concentrated entirely in the small neighbourhood of t_k, they can be approximated as an impulse with the same size as the area under $f_k(t)$, we also approximate the area as a rectangle, (like a riemann sum) so

$$f_k(t) \approx (f(t_k)\Delta t)\delta(t - t_k)$$

The weight function $w(t)$ is the response to $\delta(t)$. So by linear time invariance the response to $f_k(t)$ is

$$y_k(t) \approx (f(t_k)\Delta t)w(t - t_k)$$

Say we want to find the response at a fixed time T, we know f (input) is the sum of f_k so by *superposition* we have y (output) as the sum of y_k. At time T:

$$\begin{aligned} y(T) &= y_0(T) + y_1(T) + \ldots \\ &\approx (f(t_0)w(T - t_0) + f(t_1)w(T - t_1) + \ldots)\,\Delta t \end{aligned}$$

We can ignore all the terms where $t_k > T$ as $T - t_k < 0$ so $\delta(T - t_k) = 0$ and $w(T - t_k) = 0$. So if n is the last index where $t_k < T$ we have

$$y(T) \approx (f(t_0)w(T - t_0) + f(t_1)w(T - t_1) + \ldots + f(t_n)w(T - t_n))\,\Delta t$$

This is a riemann sum; as $\Delta t \to 0$ it tends to the integral

$$y(T) = \int_0^T f(t)w(T - t)dt$$

which is the convolution $(y * w)(T)$—the system response is the convolution of the input with the impulse response.