

# Convolution, correlation, and the Wiener-Hopf equations

Sebastian Seung

9.29 Lecture 2: February 7, 2002

In this lecture, we'll learn about two mathematical operations that are commonly used in signal processing, *convolution* and *correlation*. The convolution is used to linearly filter a signal, for example to smooth a spike train to estimate probability of firing. The correlation is used to characterize the statistical dependencies between two signals.

When analyzing neural data, the firing rate of a neuron is sometimes modeled as a linear filtering of the stimulus. Alternatively, the stimulus is modeled as a linear filtering of the spike train. To construct such a model, the optimal filter must be determined from the data. This problem was studied by the famous mathematician Norbert Wiener in the 1940s. It requires the solution of the famous Wiener-Hopf equations.

## 1 Convolution

Let's consider two time series,  $g_i$  and  $h_i$ , where the index  $i$  runs from  $-\infty$  to  $\infty$ . The convolution of these two time series is defined as

$$(g * h)_i = \sum_{j=-\infty}^{\infty} g_{i-j} h_j \quad (1)$$

This definition is applicable to time series of infinite length. If  $g$  and  $h$  are finite, they can be extended to infinite length by adding zeros at both ends. After this trick, called zero padding, the definition in Eq. (1) becomes applicable. For example, the sum in Eq. (1) becomes

$$(g * h)_i = \sum_{j=0}^{n-1} g_{i-j} h_j \quad (2)$$

for the finite time series  $h_0, \dots, h_{n-1}$ . Another trick for turning a finite time series into an infinite one is to repeat it over and over. This is sometimes called periodic boundary conditions, and will be encountered later in our study of Fourier analysis.

The convolution operation, like ordinary scalar multiplication, is both commutative  $g * h = h * g$  and associative  $f * (g * h) = (f * g) * h$ . Although  $g$  and  $h$  are treated symmetrically by the convolution, they generally have very different natures. Typically,

one is a signal that goes on indefinitely in time. The other is concentrated near time zero, and is called a filter or convolution kernel. The output of the convolution is also a signal, a filtered version of the input signal.

In Eq. (2), we chose  $h_i$  to be zero for all negative  $i$ . This is called a *causal filter*, because  $g * h$  is affected by  $h$  in the present and past, but not in the future. In some contexts, the causality constraint is not important, and one can take  $h_{-M}, \dots, h_M$  to be nonzero, for example.

Formulas are nice and compact, but now let's draw some diagrams to see how this works. Let  $m$  and  $n$  be the dimensions of  $g$  and  $h$  respectively. For simplicity, assume zero-offset indexing, so that the first components of  $g$  and  $h$  are  $g_0$  and  $h_0$  (not  $g_1$  and  $h_1$  as in MATLAB). Then  $(g * h)_0$  is given by summing  $g_{-j}h_j$  over  $j$ , which can be visualized as

$$\begin{array}{ccccccccccc} \cdots & g_{m-1} & \cdots & g_1 & g_0 & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{n-1} & 0 & \cdots \end{array}$$

Next,  $(g * h)_1$  is found by summing  $g_{1-j}h_j$  over  $j$ , which can be visualized as

$$\begin{array}{ccccccccccc} \cdots & 0 & g_{m-1} & \cdots & g_1 & g_0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{n-1} & 0 & \cdots \end{array}$$

The rest of the components of  $g * h$  are generated by sliding the  $g$  vector to the right. The last nonzero component  $(g * h)_{m+n-2}$  can be visualized as

$$\begin{array}{ccccccccccc} \cdots & 0 & 0 & \cdots & g_{m-1} & \cdots & g_1 & g_0 & 0 & \cdots \\ \cdots & h_0 & h_1 & \cdots & h_{n-1} & \cdots & 0 & 0 & 0 & \cdots \end{array}$$

Therefore  $g * h$  has  $m + n - 1$  nonvanishing components, which is why the MATLAB function `conv` returns an  $m + n - 1$  dimensional vector.

## 2 Probability of firing

The spike train  $\rho_i$  is a binary-valued time series. Since linear models are best suited for analog variables, it is helpful to replace  $\rho_i$  with a probability  $p_i$  of firing per time bin. Many methods for doing this can be expressed in the convolutional form

$$p_i = \sum_j \rho_{i-j} w_j$$

where  $w$  satisfies the constraint  $\sum_j w_j = 1$ . According to this formula,  $p_i$  is the weighted average of  $\rho_i$  and its neighbors, so that  $0 \leq p_i \leq 1$ .

There are many different ways to choose  $w$ , depending on the particulars of the application. For example,  $w$  could be chosen to be of length  $n$ , with nonzero values equal to  $1/n$ . This is sometimes called a "boxcar" filter. MATLAB comes with a lot of other filter shapes. Try typing `help bartlett`, and you'll find more information about the Bartlett and other types of windows that are good for smoothing. Depending on the context, you might want a causal or a noncausal filter for estimating probability of firing.

Another option is to choose the kernel to be a decaying exponential,

$$w_j = \begin{cases} 0, & j < 0 \\ \gamma(1 - \gamma)^j, & j \geq 0 \end{cases}$$

This is causal, but has infinite duration. As an exercise, you could try proving that this is equivalent to

$$p_i = (1 - \gamma)p_{i-1} + \gamma\rho_i$$

The probability  $p$  of firing in a time bin is closely related to frequency  $\nu$  of firing by  $p = \nu\Delta t$ , where  $\Delta t$  is the sampling interval. Probabilistic models of neural activity will be treated more formally in a later lecture.

### 3 Correlation

The correlation of two time series is

$$\text{Corr}[g, h]_j = \sum_{i=-\infty}^{\infty} g_i h_{i+j}$$

The case  $j = 0$  corresponds to the correlation that was defined in the first lecture. The difference here is that  $g$  and  $h$  are correlated at times separated by the lag  $j$ .<sup>1</sup> As with the convolution, this definition can be applied to finite time series by using zero padding. Note that  $\text{Corr}[g, h]_j = \text{Corr}[h, g]_{-j}$ , so that the correlation operation is not commutative. Typically, the correlation is applied to two signals, while its output is concentrated near zero.

If  $g$  and  $h$  are  $n$ -dimensional vectors, then the MATLAB command `xcorr(g, h)` returns a  $2n - 1$  dimensional vector, corresponding to the lags  $j = -n$  to  $n$ . Lags beyond this range are not included, as the correlation vanishes. The zero lag case looks like

$$\begin{array}{cccccccc} \cdots & 0 & g_1 & g_2 & \cdots & g_n & 0 & \cdots \\ \cdots & 0 & h_1 & h_2 & \cdots & h_n & 0 & \cdots \end{array}$$

and the other lags correspond to sliding  $h$  right or left. A maximum lag can also be given, `xcorr(g, h, maxlag)`, restrict the range of lags computed to `-maxlag` to `maxlag`. The default is the unnormalized correlation given above, but there are other options too.

The autocorrelation is a special case of the correlation, with  $g = h$ . If  $g \neq h$ , the correlation is sometimes called the crosscorrelation to distinguish it from the autocorrelation. In the first lecture, we distinguished between correlation and covariance. The covariance was defined as the correlation with the means subtracted out. Similarly, the cross-covariance can be defined as the correlation left between two time series after subtracting out the means. The auto-covariance is a special case. The command `xcov` can be used for this purpose.

---

<sup>1</sup>Warning: This is the convention followed by Dayan and Abbott, and by MATLAB. Some other books, like *Numerical Recipes*, call the above sum  $\text{Corr}[h, g]_j$

## 4 Spike-triggered average

Demonstration of these ideas:

- Convolve spike train  $\rho$  with filter to find firing rate
- Autocorrelation of stimulus
- Autocorrelation of spike train
- Cross-correlation of spike train and stimulus

## 5 The Wiener-Hopf equations

Suppose that we'd like to model the time series  $y_i$  as a filtered version of  $x_i$ , i.e. find the  $h$  that optimizes the approximation

$$y_i \approx \sum_j h_j x_{i-j}$$

We assume that both  $x$  and  $y$  have had their means subtracted out, so that no additive constant is needed in the model. Also,  $h_j$  is assumed to be zero for  $j < M_1$  or  $j > M_2$ . This constrains how far forward or backward in time the kernel extends. For example,  $M_1 = 0$  corresponds to the case of a causal filter.

The best approximation in the least squares sense is obtained by minimizing the squared error

$$E = \frac{1}{2} \sum_i \left( y_i - \sum_{j=M_1}^{M_2} h_j x_{i-j} \right)^2$$

relative to  $h_j$  for  $j = M_1$  to  $M_2$ . This is analogous to the squared error function for linear regression, which we saw in the first lecture.

The minimum is given by the equations,  $\partial E / \partial h_k = 0$ , for  $k = M_1$  to  $M_2$ . These are the famous Wiener-Hopf equations,

$$C_k^{xy} = \sum_{j=M_1}^{M_2} h_j C_{k-j}^{xx} \quad k = M_1, \dots, M_2 \quad (3)$$

where the shorthand notation

$$C_k^{xy} = \sum_i x_i y_{i+k} \quad C_l^{xx} = \sum_i x_i x_{i+l}$$

has been used for the cross-covariance and auto-covariance. You'll be asked to prove this in the homework. This is a set of  $M_2 - M_1 + 1$  linear equations in  $M_2 - M_1 + 1$  unknowns, so it typically has a unique solution. For our purposes, it will be sufficient to solve them using the backslash (`\`) and `toeplitz` commands in MATLAB. If you're

worried about minimizing computation time, there are more efficient methods, like Levinson-Durbin recursion.

Recall that in simple linear regression, the slope of the optimal line times the variance of  $x$  is equal to the covariance of  $x$  and  $y$ . This is a special case of the Wiener-Hopf equations. In particular, linear regression corresponds to the case  $M_1 = M_2 = 0$ , for which

$$h_0 = C_0^{xy} / C_0^{xx}$$

## 6 White noise analysis

If the input  $x$  is Gaussian white noise, then the solution of the Wiener-Hopf equation is trivial, because  $C_{k-j}^{xx} = C_0^{xx} \delta_{kj}$ . Therefore

$$h_k = \frac{C_k^{xy}}{C_0^{xx}} \quad (4)$$

So a simple way to model a linear system is to stimulate it with white noise, and correlate the input with the output. This method is called reverse correlation or white noise analysis.

If the input  $x$  is not white noise, then you must actually do some work to solve the Wiener-Hopf equations. But if the input  $x$  is close to being white noise, you might get away with being lazy. Just choose the filter to be proportional to the  $xy$  cross-correlation,  $h_k = C_k^{xy} / \gamma$ , as in the formula (4). The optimal choice of the normalization factor  $\gamma$  is

$$\gamma = \frac{\sum_{jl} C_j^{xy} C_{j-l}^{xx} C_l^{xy}}{\sum_m C_m^{xy} C_m^{xy}}$$

where the summations run from  $M_1$  to  $M_2$ . Note this reduces to  $\gamma = C_0^{xx}$  in the case of white noise, as in Eq. (4).