# Lecture 25
# Application of FFTs

- Convolution & deconvolution.
- Filtering.

**Convolution and deconvolution:**

$$s * r = \int_{-\infty}^{\infty} s(\tau) r(t-\tau) d\tau$$

$$s * r \Leftrightarrow S(f) R(f) \qquad \text{"Convolution theorem"}$$

Although the operation is symmetric in s,r, normally they have very different meanings: usually s is a “signal” that goes on forever, whereas r is a “response” that is usually peaked and dies off rapidly.

In the discrete case, both s and r are vectors of numbers. The response vector has the following interpretation:

- $r_0$ gives the amount of the signal $s_j$ that will be copied into bin j of the output.
- $r_1$ gives the multiple of the signal $s_j$ that will be copied into bin j+1 of the output.

Example: if $r_0$=1 and all others zero, the result is the original signal unmodified. If $r_{14}$=2 and all the others zero the result is the original signal shifted by 14 bins and doubled in amplitude.

What we have done is define in words the discrete convolution,

$$(r * s)_j \equiv \sum_{k=-M/2+1}^{M/2} s_{j-k}\, r_k$$

Which has a discrete convolution theorem. If a signal is periodic with period N and the response function is of finite duration N, then,

$$\sum_{k=-N/2+1}^{N/2} s_{j-k}\, r_k \quad \Longleftrightarrow \quad S_n R_n$$

The assumptions look a bit strong. The second one is not, normally one is interested in response functions that are shorter than the signal. One then simply pads with zeros the response function so it is of duration N.

The first assumption is trickier. Since the theorem assumes the function is periodic, if it is not it means that the portion of the convolution involving the ends of the function gets contaminated.

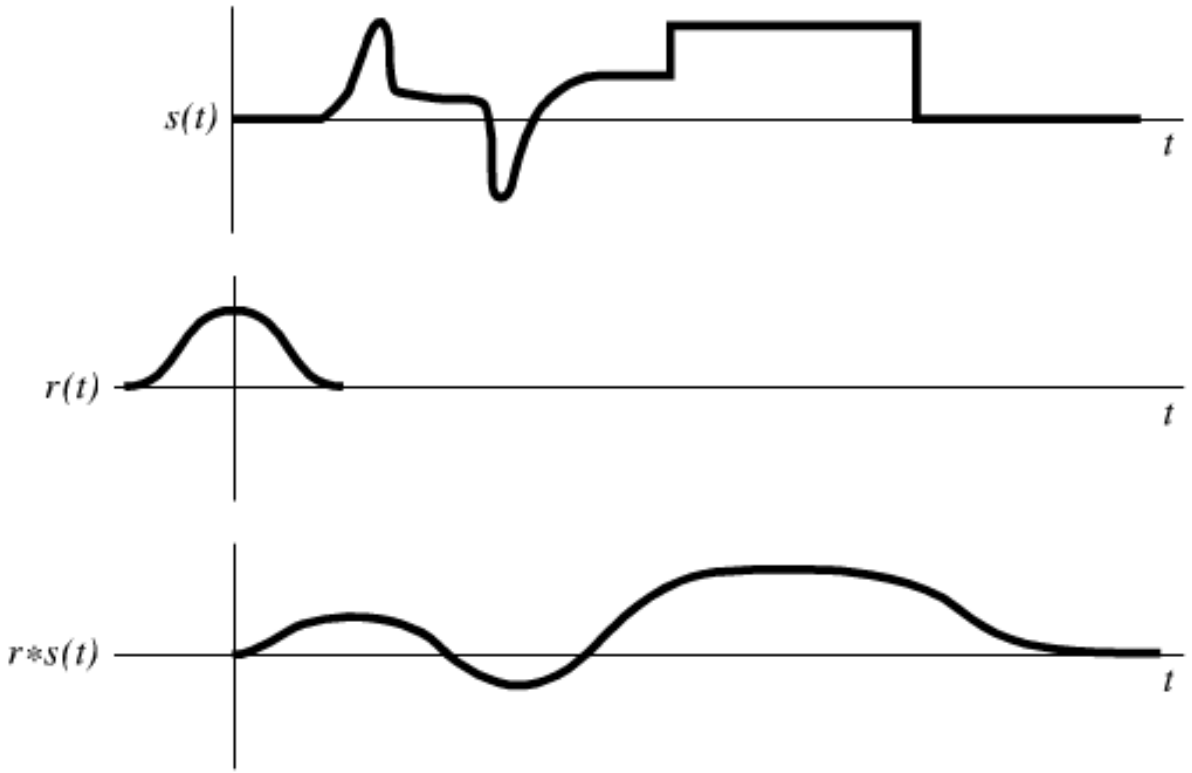



Figure 13.1.1. Example of the convolution of two functions. A signal $s(t)$ is convolved with a response function $r(t)$. Since the response function is broader than some features in the original signal, these are "washed out" in the convolution. In the absence of any additional noise, the process can be reversed by deconvolution.

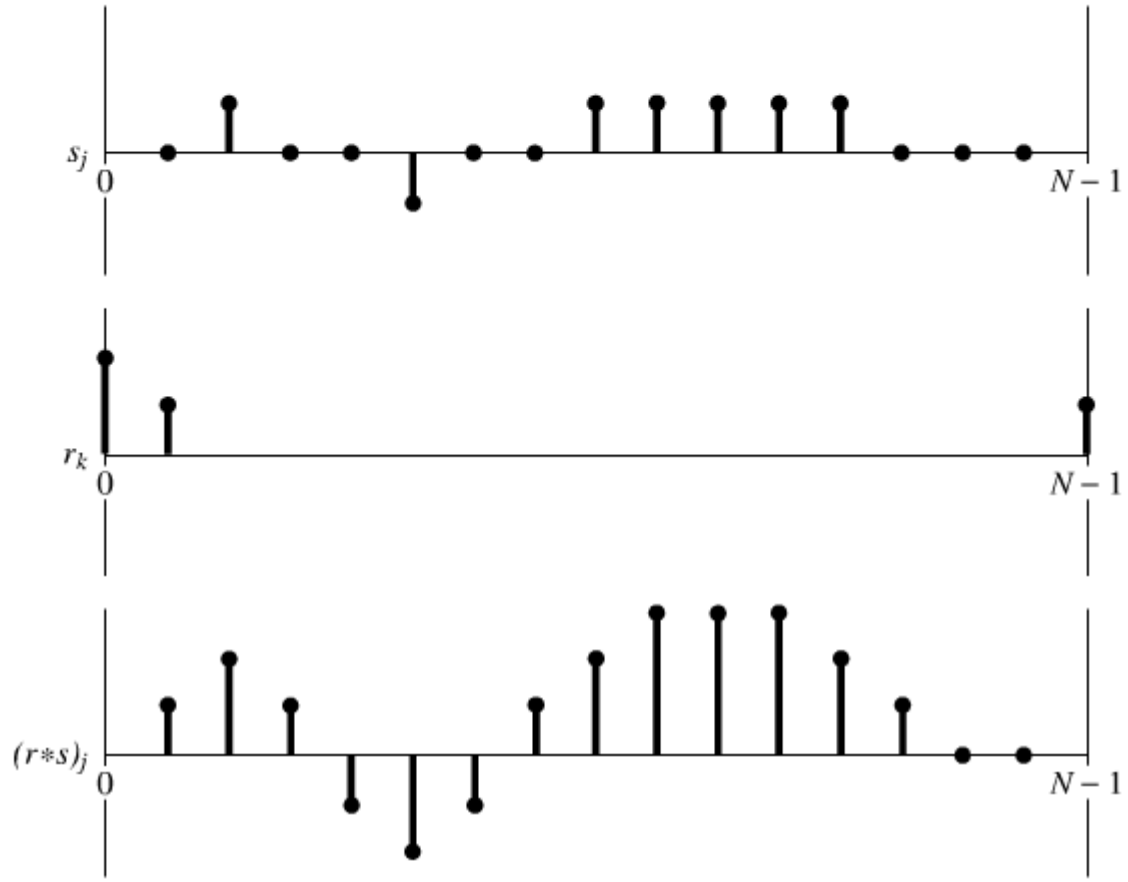
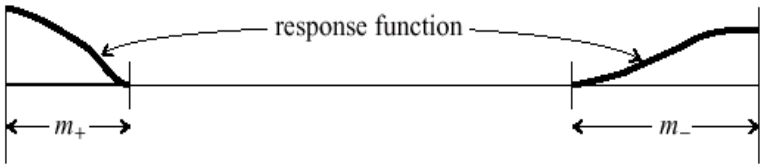
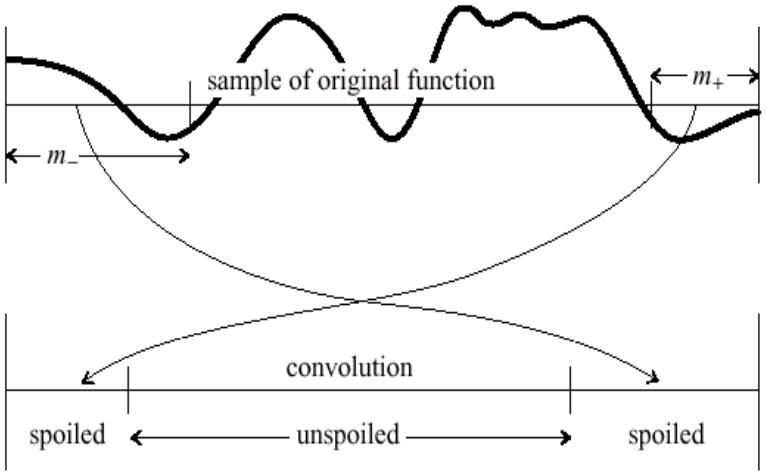



Figure 13.1.2. Convolution of discretely sampled functions. Note how the response function for negative times is wrapped around and stored at the extreme right end of the array $r_k$.

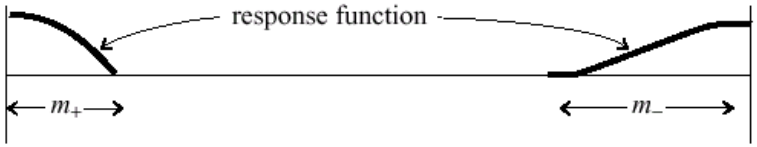


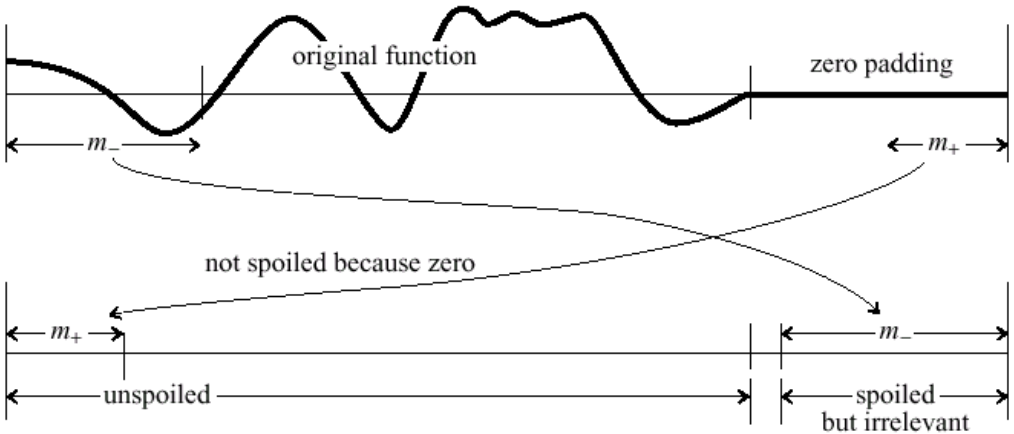




The fix for this is also zero padding. The extra zeros do not contaminate the convolution in the domain of interest.

To understand the plot, remember that we are representing the functions in “wrap around form” with the zero frequency in the middle and the Nyquist frequency at the ends.

As the response function “slides along the signal” in performing the convolution, the superposition with the ends will be blatantly wrong.



To do a convolution is (apart from the previous caveats) trivial using FFTs. One simply transforms both functions, multiplies them component by component (as complex numbers) and that gives the transform of the convolution.

The equation for the deconvolution is the same as for the convolution, but now one considers the left-hand-side as a given. One is then left with a set of coupled linear equations for the unknown signal, the coefficients determined by the components of the response function.

For large sets of data, it is impractical to solve these equations (the matrix has many zeros but also lots of non-zero components). It is much easier to take the transform of the convolution and divide it by the transform of the response function. That gives the transform of the signal, and one simply does an inverse transform.

The process can fail if a component of the transform of the response function is zero. Then the convolution means one has lost all information about the signal for that particular frequency.

Due to that kind of sensitivity, the deconvolution operation is delicate, particularly with noisy data. It might pay to filter the data first, as we will discuss later on.

```
 subroutine convlv(data,n,respns,m,isign,ans)
      parameter(nmax=8192)
      dimension data(n),respns(n)
      complex fft(nmax),ans(n)
      do 11 i=1,(m-1)/2
        respns(n+1-i)=respns(m+1-i)
11    continue
      do 12 i=(m+3)/2,n-(m-1)/2
        respns(i)=0.0
12    continue
      call twofft(data,respns,fft,ans,n)
      no2=n/2
      do 13 i=1,no2+1
        if (isign.eq.1) then
          ans(i)=fft(i)*ans(i)/no2
        else if (isign.eq.-1) then
          if (cabs(ans(i)).eq.0.0) pause 'deconvolving at a response zero'
          ans(i)=fft(i)/ans(i)/no2
        else
          pause 'no meaning for isign'
        endif
13    continue
      ans(1)=cmplx(real(ans(1)),real(ans(no2+1)))
      call realft(ans,no2,-1)
      return
      end
```

Convert response to length N, padding with zeros.

Transform

Convolve or deconvolve depending on isign.

Inverse transform.

Copy first component.

If one is dealing with very large data sets such that they cannot fit in the computer memory one must break them into sections and convolve each section separately. Apart from the wrap around effects we discussed, you will have to deal that each section of data should have been influenced by the preceding and following section of data, but were not.

There are two ways to deal with this. The first one is the overlap-save method. You pad only the very beginning of the data with enough zeros to avoid the wraparound problem. Throw away the points at each end that are polluted by wrap around effects. Output only the remaining good points in the middle. Now bring in the next section of data, but not all new data. The first points in each new section overlap with the last points from the preceding section of data.

The second technique is to pad at both ends with zeros and then overlap and add the results.

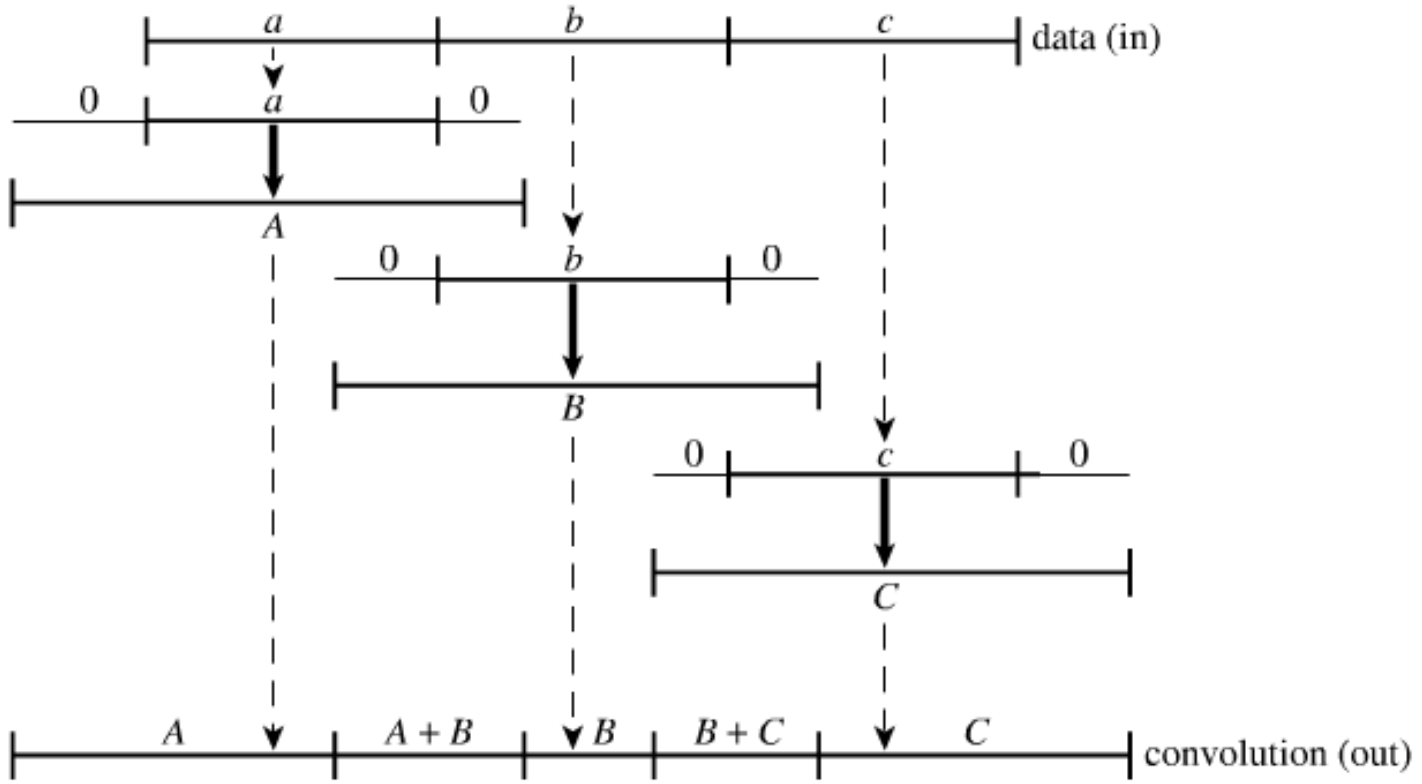



Figure 13.1.5. The overlap-add method for convolving a response with a very long signal. The signal data is broken up into smaller pieces. Each is zero padded at both ends and convolved (denoted by bold arrows in the figure). Finally the pieces are added back together, including the overlapping regions formed by the zero pads.

Correlations go more or less in the same way. There is a discrete correlation theorem that holds for both functions being periodic,

$$\mathrm{Corr}(g,h)_j \equiv \sum_{k=0}^{N-1} g_{j+k} h_k \qquad\qquad \mathrm{Corr}(g,h)_j \Longleftrightarrow G_k H_k{}^*$$

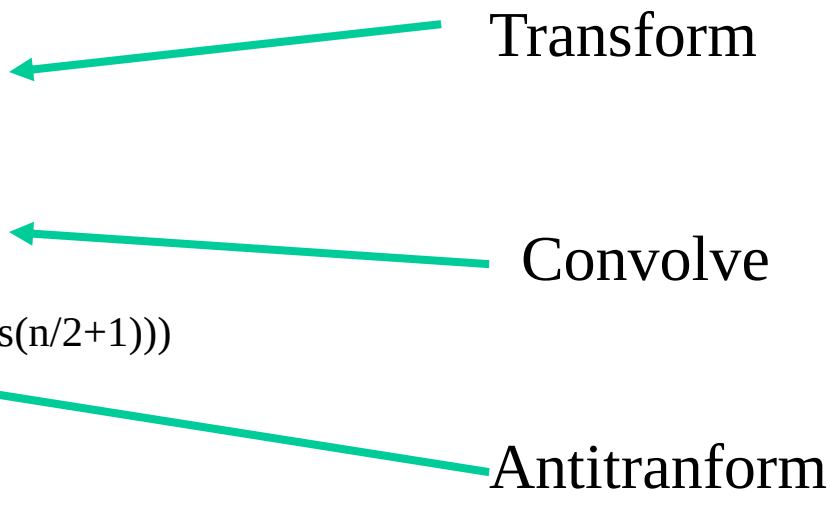
The correlation is a measure of how much one functions lags the other. If the correlation is large for large values of t, it means that one of the functions resembles the other but for later values of its variable. Similarly if it is large for negative values of t it means one functions resembles the other but leads it.

Because correlations are normally computed among functions that are similar, we do not have the problems we had with the response function in convolutions.

If the data is not periodic, you need to pad with zeros. How many? As many as the length of the correlation you wish to measure (that is, how large a value of t are you interested in).

The routine that implements it is quite simple,

```
subroutine correl(data1,data2,n,ans)
    parameter(nmax=8192)
    dimension data1(n),data2(n)
    complex fft(nmax),ans(n)
    call twofft(data1,data2,fft,ans,n)
    no2=float(n)/2.0
    do 11 i=1,n/2+1
      ans(i)=fft(i)*conjg(ans(i))/no2
11    continue
    ans(1)=cmplx(real(ans(1)),real(ans(n/2+1)))
    call realft(ans,n/2,-1)
    return
    end
```




In all these routines, when we use twofft one has to be careful that the two functions transformed are roughly of the same magnitude. Otherwise, roundoff error will dominate. In such cases, one should substitute the call by two calls to real Fourier transform routines.

**Filtering:**

In real life one deals with “corrupted” signals. There are two main ways that the signal u(t) can become corrupted. One is that the measuring apparatus is not “faithful”. This means that the measuring process is not a convolution with a Dirac delta function (a perfect measuring apparatus),

$$s(t) = \int_{-\infty}^{\infty} r(t-\tau)u(\tau)\,d\tau \quad \text{or} \quad S(f) = R(f)U(f)$$

If we know the details of the response r(t) of the measuring apparatus, we already learnt how to fix this problem: deconvolve.

The second way the signal can get corrupted is through added noise,

$$c(t) = s(t) + n(t)$$

If the initial signal is corrupted this way we cannot simply deconvolve it to eliminate the influence of the measuring device. The technique used to deal with this is *filtering*.

We apply the filter, represented by a function $\phi(t)$ (or $\Phi(f)$) and then deconvolve,

$$\widetilde{U}(f) = \frac{C(f)\Phi(f)}{R(f)}$$

And we require that the resulting signal be close to the correct one in the least square sense,

$$\int_{-\infty}^{\infty} |\widetilde{u}(t) - u(t)|^2 \, dt = \int_{-\infty}^{\infty} \left|\widetilde{U}(f) - U(f)\right|^2 \, df \quad \text{is minimized.}$$

If we now assume that the signal and the noise have zero correlation (definition of noise), we get,

$$\int_{-\infty}^{\infty} \left| \frac{[S(f)+N(f)]\Phi(f)}{R(f)} - \frac{S(f)}{R(f)} \right|^2 df$$

$$= \int_{-\infty}^{\infty} |R(f)|^{-2} \left\{ |S(f)|^2 |1-\Phi(f)|^2 + |N(f)|^2 |\Phi(f)|^2 \right\} df$$

Which is the formula for the “optimal” (Wiener) filter.

And if we differentiate with respect to the noise and set the result to zero we get,

$$\Phi(f) = \frac{|S(f)|^2}{|S(f)|^2 + |N(f)|^2}$$

An important aspect of the previous formula is that it only requires knowledge of the signal “smeared with noise” S(t). Otherwise we would not have information to evaluate it!

It depends, however, on the noise N(t). To estimate this we can sample the signal over a long stretch of data and compute its power spectral density. Again assuming that signal and noise are uncorrelated, the power spectral density of the corrupted signal c(t) is,

$$|S(f)|^2 + |N(f)|^2 \approx P_c(f) = |C(f)|^2 \qquad 0 \leq f < f_c$$

Spectrally, the signal will usually be very different from the noise. The signal usually will have a peak at its characteristic frequency, whereas the noise will have components all across the spectrum. The particular spectrum of the noise may have peculiarities, like “tilts”, increasing or decreasing with frequency, but one ought to be able to separate it from the signal. If this is impossible, then one is out of luck in applying an optimal filter.

The filter is close to one where there is no noise and close to zero where the noise dominates. That is how it does its job!

The minimization we used to determine the filter implies that discrepancies between the noise we use and the real noise only enter at second order. That is why even eyeballing the noise in the power spectrum can be good enough as an estimate.

One can be tempted to iterate the technique. Having the filtered signal, treat it as a fresh signal and go over the process again. Unfortunately application of a filter attenuates the signal (due to the errors in determining the noise). Repeating the procedure leads to a process that is convergent… to S(f)=0!

With the routines we have, applying a filter is straightforward. One applies it in the frequency domain after FFT'ing the data. If one is also deconvolving, we can modify the routine we discussed before to apply the filter before doing the deconvolution.

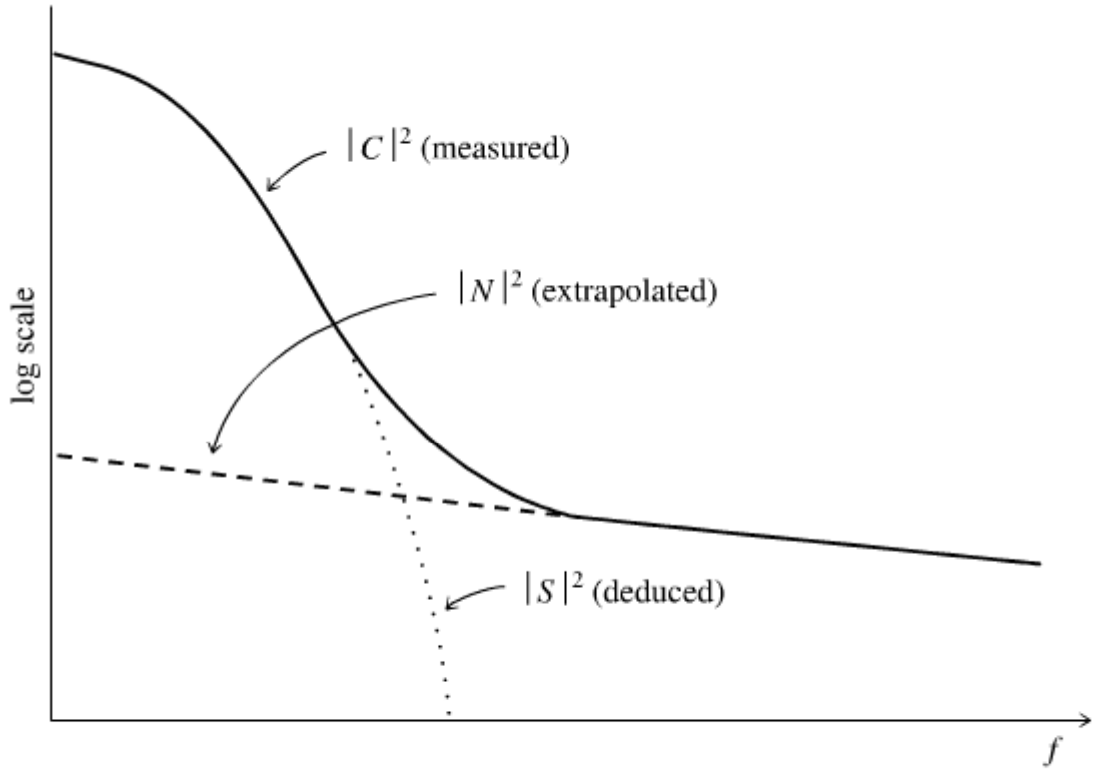



Figure 13.3.1. Optimal (Wiener) filtering. The power spectrum of signal plus noise shows a signal peak added to a noise tail. The tail is extrapolated back into the signal region as a "noise model." Subtracting gives the "signal model." The models need not be accurate for the method to be useful. A simple algebraic combination of the models gives the optimal filter (see text).

# Summary

- Convolution and deconvolution are straightforward but watch for wraparound effects.
- Optimal filtering only needs an approximate knowledge of the noise.