

Digital Postprocessing

“Son,” the old guy says, “no matter how far you travel, or how smart you get, always remember this: Some day, somewhere,” he says, “a guy is going to come to you and show you a nice brand-new deck of cards on which the seal is never broken, and this guy is going to offer to bet you that the jack of spades will jump out of this deck and squirt cider in your ear. But, son,” the old guy says, “do not bet him, for as sure as you do you are going to get an ear full of cider.”

—Damon Runyon, *The Idyll of Miss Sarah Brown*

17.1 INTRODUCTION

Everybody needs to know something about postprocessing data. There are two benefits of this: to be able to do it, of course, but as importantly, to gain good technical taste for what should be done in digital postprocessing and what must be done in the front end and the analog signal processing. Discerning that is a high and useful skill and should be cultivated.

In this chapter, we start with the raw numbers at the output of the digitizer and see how to produce real experimental results at the end. The focus is on strategies and algorithms, rather than on code. We'll explicitly ignore image processing, partly because it is a field unto itself, but more particularly because it is mainly good for pretty pictures and heavy processing such as feature extraction and machine vision, which are well beyond the scope of this book.

The basic thesis here is that postprocessing is not a substitute for good data. The best postprocessing strategy is one that makes few assumptions about the data and does no violence to them either; by and large that limits us to linear operations. Nonlinear processing can make nice pictures but you must use it very sparingly if quantitative data matter to you—enough processing can make a given picture look like anything at all. You might as well just multiply by 0 and add what you think it should look like.

Be thoughtful too about what order to do nonlinear processing in; for example, if you are measuring the phase of an RF signal directly, with a tracking phase detector, and then postprocessing the phase numbers as your signal, you should probably unwrap the phase before filtering. On the other hand, if the measurement is in rectangular coordinates (e.g., with an *I* and *Q* detector; see Section 13.8.7), you might want to filter and then unwrap, especially if you have lots of additive noise to contend with. If your intuition runs into

a wall, do some numerical experiments on canned data—and use real data if it’s in any way possible.

Choice of domain matters a lot. Certain operations are very much easier in one domain than other ones: filtering is easy in the frequency domain; image compression and some matrix operations in the wavelet domain; echo cancellation in the cepstral domain. We’ll begin with the simplest sorts of postprocessing (e.g., gain and offset correction), then state the basic theorems and use them to develop digital filtering, quantization effects, and the recovery of signals from noise. The emphasis is on interrelating these topics and showing how the lore is grounded in the theory, but if you’re new to the topic you’ll need a couple of good books, for example, Oppenheim and Shafer for signal processing theory and *Numerical Recipes in C* for algorithms and coding advice (see the Appendix).

17.2 ELEMENTARY POSTPROCESSING

Even instruments that don’t need a lot of filtering, deconvolution, and so on usually need to massage the raw digitizer output into a properly calibrated measurement.

17.2.1 Gain and Offset

The most basic postprocessing operation is to figure out where 0 and full scale are in measurement terms, and to apply a numerical gain and offset to transform the raw data into measurement units. For example, if a 12 bit digitizer was used with the turbidity meter of Example 13.11, and calibration data indicate that 0 signal corresponds to 8.3 ± 0.15 ADU and that a turbid extinction of $0.1\%/cm$ ($\pm 1\%$ of value) gives a reading of 3037.5 ± 2.2 ADU, the turbid extinction is given by

$$E = 0.1\%/cm \times \frac{(R - 8.3)}{(3037.5 - 8.3)}. \quad (17.1)$$

The standard error is found by differentiating with respect to all uncertain quantities, multiplying each partial derivative by the RMS error in the quantity, and taking the RMS sum of the result, just as we did in Section 10.6.2. Since all subsequent stages will depend on this calibration, you need to be a bit paranoid about it; in the turbidity meter example, clean water and $0.1\%/cm$ standard turbid water are probably not enough as calibration points; it needs some intermediate values. This improves accuracy as well as allowing sanity checks—if you have several independent measurements, one improperly mixed calibration solution won’t cause you to ship a miscalibrated instrument. Keep a close watch on how you could be fooled: for instance, in Section 13.11.5 we talked about making sure that there’s enough of an offset that 0 signal is guaranteed to give an on-scale reading. If you can’t possibly do that, for example, if you’re using preexisting equipment, then take several data points, use curve fitting to get the offset, and watch its drift vigilantly.

17.2.2 Background Correction and Calibration

A strategy involving more than a single measurement location (e.g., spectroscopy or imaging) has a somewhat more complicated calibration requirement, as the system gain

and offset may change with scan position. Of course, in designing instruments we try to make any such variation as small and as stable as possible, with stability being much the more important of the two. The simplest case is where the gain is close enough to constant, but the offset varies, for example, a mechanically scanned system where the background light depends on scan position; all we have to do then is subtract the background signal.

A more general and more common case is when the gain changes with scan position as well, as in a tunable-laser spectrometer, where the laser power depends on the tuning and there are etalon fringes in the sample cell. If the gain and offset are stable with time, we can usually produce calibration tables containing gains and offsets for each scan point, as we did in Section 3.9.19. The usual way is to make several calibration runs (from a minimum of about four to many thousands, depending on how expensive the calibration runs are and how much real data you intend to take). Remember that any residual noise from the calibration contributes fixed-pattern noise that will not go away when you signal-average your calibrated measurement data, so make sure the calibration noise is small compared to the residual random noise after signal averaging.

The other key to good calibrations is to calibrate often enough that background shifts do not corrupt the measurement. These two requirements are of course in direct conflict and must be traded off depending on the situation. A bit of ingenuity will usually produce some heuristics and sanity checks that you can use to determine when to calibrate and whether the current calibration is good. An instrument might be programmed to recalibrate itself any time the temperature changes by more than 0.87°C from last time, the system has recently been turned on and has now stabilized, 10 hours have elapsed, or a single-scan sanity check falls outside the 0.1% confidence limit.

As always in instrument design, make sure you calculate your signal-to-noise ratio carefully at each step; if you get stuck somewhere, that shows you where your unexamined assumptions are going to trip you up. Remember that we live and die by the SNR, so it is well worth your time to become clear, or to get some consulting help.

17.2.3 Frame Subtraction

There is a poor man's version of this careful correction procedure, one that may be unwise to even mention, as it sometimes constitutes a temptation to cutting corners unnecessarily. It is to make two runs where the background is the same but the signal is different, and simply subtract them. A legitimate example is a CCD with slow shuttering, for example, a frame transfer device, used in bright light. In that case, the bright background effectively gets shifted across the whole array during readout, making a big bright smear across the image. At the price of more or less serious dynamic range reduction, you can eliminate the smear caused by the slow shifting of the image into the storage array by taking two exposures, say, 2 s and 500 ms, and subtracting them. The smeared part of the image will be the same in both images (and so will cancel), but 60% of the unsmeared photons will survive. Providing that there's no bloom or nonlinearity to worry about, and that the image contrast is not too high,[†] summing several images taken in this way will recover the lost SNR.

[†]Why does the contrast matter?

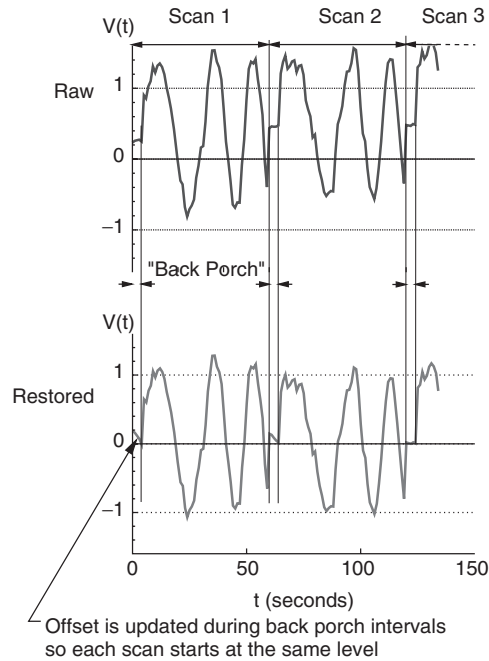


Figure 17.1. Baseline restoration.

17.2.4 Baseline Restoration

When the background fluctuates more rapidly than a canned calibration can deal with, we have to try to use internal cues instead, leading to the idea of baseline restoration (Figure 17.1). The prototype is the DC restore function of a TV signal. The video signal's instantaneous voltage controls the local picture intensity (black is most positive). However, the video detector output is AC-coupled, which might seem to make this difficult. At certain times in each scan line, reference levels are transmitted, which are sampled and used as references to adjust the offset line by line, establishing a stable zero point. The same idea is useful in many situations where the background fluctuates widely, on a time scale that is long compared to the line rate, but too fast for a static calibration to help.

Baseline restoration must be applied carefully, or it will introduce artifacts by subtracting the wrong DC value; make sure the baseline sampling is done at a time when the signal is known to be 0, for example, when the light source is cut off, the shutter is closed, or the sample is not present, and—crucially—allow enough settling time on both ends when switching between signal and baseline acquisition. Remember that for baseline restoration to be applicable at all, the rapidly drifting part of the background has to be independent of scan position, which is not always the case.

Baseline restoration can also be applied point-by-point in sufficiently slow measurements, in which case it turns into a digital version of auto-zeroing (Section 15.10.3).

Example 17.1: Baseline Correction and Deconvolution for a Pyroelectric Imager. In Sections 3.11.16 and 18.7.2 we talk about pyroelectric sensor design; here we'll look

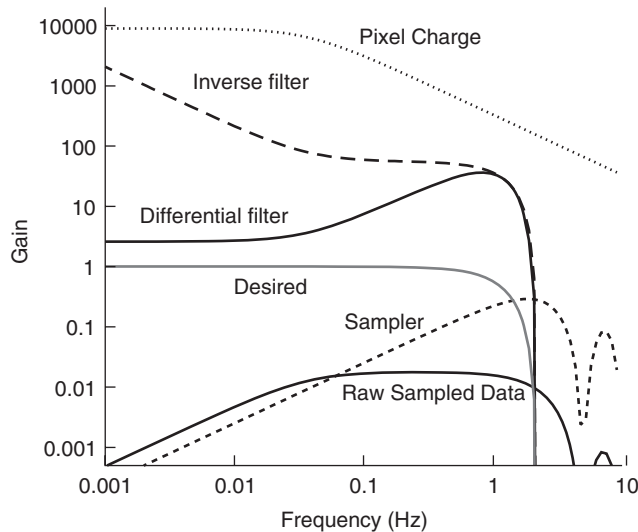


Figure 17.2. The transfer function of an unusual pyroelectric imager: the top curve shows the pixel charge response, which after sampling becomes the bottom curve. The two-stage inverse filter (dashed) is implemented as a nine-sample FIR filter for high frequency boost (solid black) followed by a running sum, and produces the well-behaved final response (solid grey). The ill-conditioning of the integration step is removed by adding a slow negative drift to the data and applying a positivity constraint: people are warmer than the floor.

at the DSP aspects. Pyroelectric detectors convert a temperature change into a charge, rather than a current as quantum detectors and bolometers do. That means that pyros are differentiating; the current is proportional to dT_{sensor}/dt , and when the charge is sampled periodically for measurement, a constant temperature corresponds to zero current. While this effect is fast, what we actually want to measure is scene temperature, not sensor temperature; due to the weak coupling and the thermal mass of the sensor, its response to scene temperature is a slow one-pole rolloff at $f_c = G/(2\pi M_{\text{th}})$, where G is total thermal conductance and M_{th} is the thermal mass of the detector element. Above f_c this rolloff cancels the differentiator slope, so that the response of a pyroelectric becomes flat for a bit, until thermal transient behavior becomes important.[†]

Thermal conduction acts exactly like a load resistor on a photodiode: it reduces the signal available and it adds thermal noise. Thus we would appear to win by insulating the sensor very well, so that its response is dominated by radiation, as Figure 17.2 shows. Unfortunately, this also slows the sensor down. What's more, the differentiating action means that forming a decent image is a bit of a puzzle; the inverse of a differentiator is an integrator, which applies infinite gain at zero frequency, where the sensor has an SNR of zero (not 0 dB, *zero*), which makes it hard to keep a stable background.

Each pixel has its own gain and offset, so we have to flat-field the sensors as in Section 3.9.19, but all pixels have the same thermal response. After applying the inverse filter, we still have to fix the DC problem, which we do by subtracting a slow linear ramp (equivalent to about -10 mK/s) from the data, and then forcing all negative ΔT values

[†]See Chapter 20, “Thermal Control,” available at <http://electrooptical.net/www/beos2e/thermal2.pdf>.

to zero. This *positivity constraint* and ramp force all the quiescent pixels to stay near $\Delta T = 0$. Interestingly, even the kTC noise is reduced by this approach; the noise charge has nowhere to go except the pixel capacitance, so it gets recycled into the following sample, and the running sum tends to make it cancel out in the final data.

17.2.5 Two-Channel Correction

If the background fluctuates too fast even for baseline restoration, you'll have to move your measurement to a frequency where the background is quiet (Section 10.9), or use a second detector that sees the background and not the signal. This is done in Section 18.6.3 in analog, but you can do something similar (though far less effective) by using a second digitizer channel to measure the background. It is crucial that both detectors see the same background, which may require some rejiggering of the apparatus, for example, a noisy arc lamp might have to be put in an integrating sphere. Unless your background measurement is somehow quieter than your signal measurement, two-channel correction will cost you at least 3 dB in ultimate SNR.

17.2.6 Plane Subtraction and Drift

Sometimes the error in adjacent scan lines is highly correlated, for example, a scanning tunneling microscope (STM) picture where there is a secular thermal drift in the z actuator, which controls tip-to-sample separation. The feedback loop will keep the tip-sample separation constant, so the z drift will instead show up in the control voltage, which is the measurement data. If the raster is regular and unidirectional, with a line period T_{row} and a pixel period T_{col} , a constant drift rate v will produce a topographic image that differs from the true topography by a linear function of the row and column indices i and j ,

$$Z_{\text{meas}}(x, y) = Z_{\text{true}}(x, y) + ai + bj + c, \quad (17.2)$$

where $a = v \cdot T_{\text{row}}$ and $b = v \cdot T_{\text{col}}$. The error terms are an analytical description of a plane, and so correcting for them is known as *plane subtraction*. Plane subtraction is less vulnerable to artifacts than line-by-line background restoration, since only one rate and one offset need be estimated, which takes a great deal less calibration data. It also has only two adjustable parameters, which increases our confidence in the measurement since you can't sweep much dirt under a rug that small.

17.2.7 More Aggressive Drift Correction

If plane subtraction is good for controlling drift and is pretty safe, how about using a higher order model? Sometimes you can measure the drift independently, for example, when you purposely build in a signal-free calibration time in each scan line to facilitate baseline restoration, and then more complicated background subtractions may be OK; however, in cases where the fitted background is coming from the data itself, it's a mistake. There are just too many unexpected ways of fooling yourself and others. Whatever operation you use to get the coefficients for the fancy fit, it will soon disappear from your consciousness, and may be silently generating reasonable-looking wrong answers long afterwards. There's no escaping it: you'll have to get better data.

17.3 DEAD TIME CORRECTION

We saw in Sections 3.6.2 and 3.6.4 that to get stable and accurate photon counting measurements, you have to add a bit of *dead time* τ after each photocount, which makes the measurement nonlinear, because some photons will be absorbed during the dead time and so be missed. The simplest way of looking at this is to say that at a photon detection rate λ' , the proportion of dead time is $\tau\lambda'$, so the true photon arrival rate λ is

$$\lambda = \lambda' / (1 - \tau\lambda'). \quad (17.3)$$

This approach gives excellent results as long as the photon arrival is a Poisson process with a constant rate (λ is constant during time τ) and $\tau\lambda \lesssim 0.2$; otherwise more thought is required.

17.4 FOURIER DOMAIN TECHNIQUES

This section really isn't an introduction to digital signal processing; it's an attempt to connect the digital realm to what we already know about time, frequency, and SNR, emphasizing rules you can hold on to and some nasty potholes to steer clear of. Readers unfamiliar with the basic material should consult Bracewell for a firm and intuitive grounding.

17.4.1 Discrete Function Spaces

The data set from our digitizer consists of a finite number of samples d_n , taken at known discrete times t_n . It can be looked at as an ordinary column vector, of the sort we're familiar with from linear algebra. We know that such a vector can be written as a linear combination of basis vectors, and that we can use any basis set we like as long as the basis vectors are linearly independent. Transforming from one basis set to another requires a matrix multiplication, and the results can be seriously corrupted by roundoff error if either the new or the old basis set is nearly linearly dependent. The best case is when both bases are orthonormal; that is, the basis vectors within each set are mutually orthogonal and have length 1.

Our data can similarly be represented as weighted sums of other functions, which is the idea of a function space. Continuous functions of infinite length give rise to complicated function space problems, but we don't have to worry about that here.

The data values consist of a mixture of signal, noise, and spurious products due to nonlinearity and the digitizing process itself; it turns out to be possible to use the functional decomposition of our data vector to separate the signal from the crud pretty well in most cases, unless we've allowed the spurs to land on top of the data.

By far the most useful basis set is the complex exponentials, $\phi_n = \exp(i2\pi nft)$, which lead to modeling signals as trigonometric polynomials (weighted sums of a fundamental and its harmonics). They are not only linearly independent, but orthogonal, both in the continuous- and discrete-time cases (where they are also called geometric sequences). Trig polynomials are intuitive and mathematically convenient, and have a close relationship to concepts we're familiar with from continuous-time analysis, such as bandwidth and SNR. A slightly subtler way of saying this is that they are the eigenfunctions of

linear, time-invariant discrete-time networks, which are expressible as systems of linear difference equations with constant coefficients. This “geometric in, geometric out” behavior is exactly analogous to the “exponential in, exponential out” property of linear circuits. (Note that the sampling rate has to be kept constant for the network to be time invariant.) A very useful property is that sines and cosines are the correct orthogonal functions for equal weighting on both the finite continuous interval *and* on an equally spaced grid of points, in any number of dimensions, so we can move easily between integral transforms and finite Fourier series.

Sines and cosines don’t exhaust the class of complex exponentials, of course. Do keep in mind that there are lots of functions that are not particularly easily represented this way, especially those with asymptotes, discontinuities, or rapid changes of behavior, and that other representations are better for some uses.

Sampled data can be represented as vectors, and linear operations are matrix multiplications (although they’re not implemented that way). However, what we care about is faithful representation and reproduction of the continuous-time input waveform, and furthermore, a finite length of sampled waveform has a continuous Fourier transform, so that we’re not in a pure linear algebra situation.

17.4.2 Finite Length Data

A function $g(t)$ that is nonzero only on the domain $[t_0, t_0 + P)$ is said to have *compact support*. (We can choose $t_0 = 0$ without loss of generality.) It will have a transform that is a continuous function of frequency, so that an uncountably infinite number of Fourier components are needed to represent it. This is not an attractive property, so we naturally seek ways around it. The most common way is to make it periodic, so that it is representable by a Fourier series, which is only countably infinite, and thus ever so much more practical. A function with compact support can be made periodic in many ways. The easiest is to join many copies together, defining $\hat{g}(t) \equiv g(t \bmod P)$. In the special case of compact support, this is equivalent to

$$\hat{g}(t) \equiv \sum_{n=-\infty}^{\infty} g(t + nP) = g * \text{III}\left(\frac{t}{P}\right), \quad (17.4)$$

which is a fruitful observation, as we’ll see.

Such a periodic function has a Fourier series representation,

$$\hat{g}(t) = \sum_{n=-\infty}^{\infty} a_n e^{i2n\pi t/P}, \quad a_n = \frac{1}{P} \int_0^P \hat{g}(t) e^{-i2n\pi t/P} dt. \quad (17.5)$$

Aside: Other Fourier-Type Series. The complex Fourier series is not the only possibility. For example, if we construct \hat{g} by alternating normal and time-reversed copies of g , we make an even function of period $2P$; the sine components of (17.5) go away, and we have the Fourier cosine series,

$$\hat{g}(t) = \sum_{n=0}^{\infty} b_n \cos \frac{n\pi t}{P}, \quad b_0 = \frac{1}{P} \int_0^P \hat{g}(t) dt, \quad b_n = \frac{2}{P} \int_0^P \hat{g}(t) \cos \frac{n\pi t}{P} dt. \quad (17.6)$$

If we multiply the time-reversed copies by -1 , \hat{g} becomes an odd function of t , producing the Fourier sine series,

$$\hat{g}(t) = \sum_{n=1}^{\infty} c_n \sin \frac{n\pi t}{P}, \quad c_n = \frac{2}{P} \int_0^P \hat{g}(t) \sin \frac{n\pi t}{P} dt. \quad (17.7)$$

The sine and cosine series have real coefficients at half the frequency spacing of the complex series. We aren't doing this for our health, but to look for a way to represent arbitrary functions using finite arithmetic on a computer; therefore we want a series representation that will give us accurate reconstruction of the data with the fewest coefficients (we assume that the Fourier coefficients of our signal will eventually die off with increasing frequency). If g has zero derivatives at the ends of the interval, the cosine series is probably best; if it goes to 0 linearly there, the sine series is worth a try; otherwise, stick with the complex series. Measurement data are seldom well enough behaved to make the sine or cosine series worth worrying about in practice.

17.4.3 Sampled Data Systems

The next question that comes to mind is: What did I do to my spectrum just now? The convolution relation (17.5) is the key. In Chapter 13, we saw that convolving two functions means multiplying their Fourier transforms; thus the spectrum of the periodic continuous function \hat{g} is the product of G and the transform of $\text{III}(t/P)$, which is $(1/P)\text{III}(fP)$. Thus as we see in Figure 17.3, the spectrum we wind up with is a sampled version of the spectrum G , with each sample multiplied by an appropriately placed δ -function to make the integral of \hat{g} come out right.

Functions known analytically can have their Fourier series computed as far as necessary, but we're building instruments—if we knew in advance what the data were going to look like, we could just sleep late instead. What we have to work with is a finite number of samples of a function of finite (but possibly great) extent in time. From elementary linear algebra, we know that you can't get more independent coefficients out of a basis transformation than you put in, so we accept that our knowledge even of the Fourier series will be limited. In fact, what we mean by *spectrum* is a bit problematical, since we saw that the way we go about making the function periodic affects the spectral information we get. We do have the intuition that if we sample a continuous function densely enough, we should be able to reproduce it accurately.[†]

Changing over to Fourier series also leaves us in a bit of a conceptual bind when we think about 1 Hz SNR and bandwidth generally. Both presuppose that frequency can vary continuously, whereas the Fourier series has infinite PSD at discrete frequencies and zero elsewhere. At the risk of getting slightly ahead of ourselves, we see from Figure 17.4 that all is not lost. If we take these spikes and spread them out over the areas in between, then increasing the sampling rate does seem to produce better spectral estimates. Figure 17.4 plots the N -point discrete and true transform of $g(x) = \text{sech}(\pi x)$, which is its own Fourier transform, over the range $-5 < x < 5$. Because the length of the interval in x is constant, the frequency spacing is too; the extra data points from the

[†]Errors resulting from truncating an infinite series, or sampling a continuous function, are generically called *truncation error*, to differentiate them from roundoff error, which is merely a consequence of imperfect arithmetic rather than of the algorithm itself.

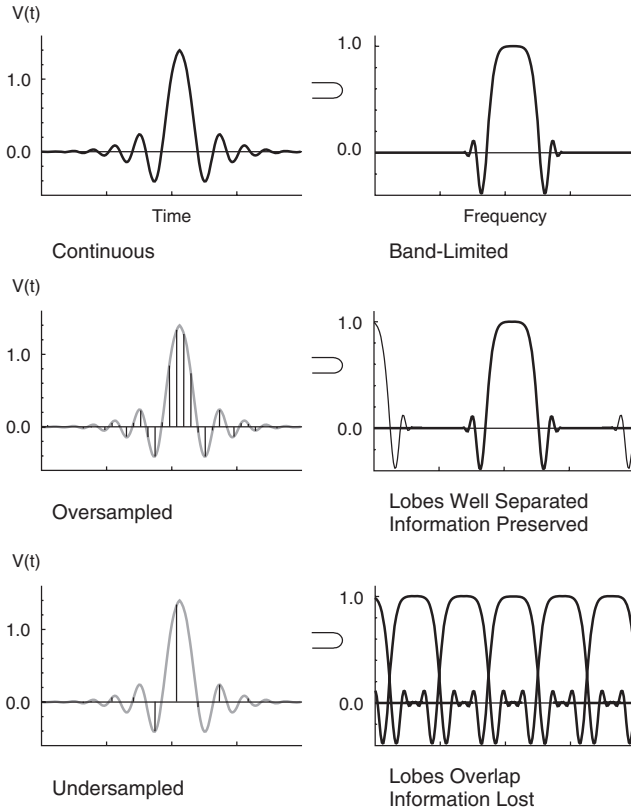


Figure 17.3. Sampled data in the Fourier domain.

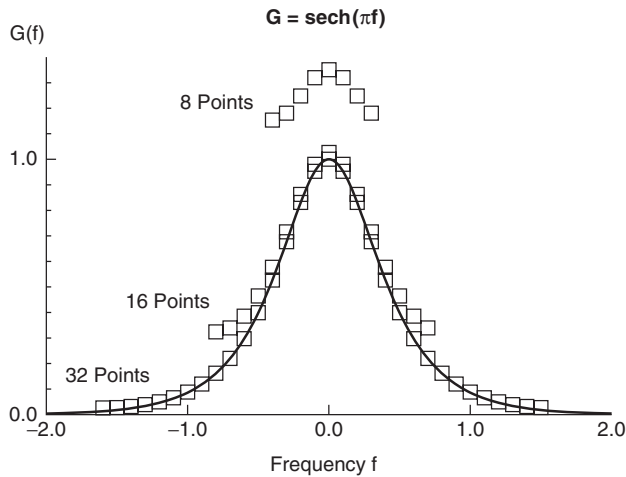


Figure 17.4. Transform of the function $g(x) = \text{sech}(\pi x)$, and the N -point DFTs of samples on the domain $-5 < x < 5$, for $N = 8, 16, 32$. Note how the spectral estimates improve with finer sampling.

finer sampling go into increasing the frequency range. We need to firm up these ideas, which is what we'll do next.

17.4.4 The Sampling Theorem and Aliasing

In the time domain, intuition tells us that if we take a sufficiently slowly varying continuous function, and sample it densely enough, we ought to be able to reconstruct it very accurately from the samples alone. Our experience with polynomial interpolation (e.g., Taylor's theorem with remainder) suggests that the error we commit might go as some high power of the spacing between samples, but in fact we're luckier than that. Nyquist's sampling theorem, which is the mathematical statement of this intuition, sets out conditions under which the reconstruction is perfect.

Sampling a function $g(t)$ at a rate $\nu = 1/\tau$ can be represented in the continuous-time domain as multiplication of the signal by an appropriately normalized *sha* function $\tau \text{III}(\nu t)$, which produces the multiple copies of the transform, as we just saw. This time the samples are in the time domain and the multiple copies in the frequency domain:

$$g(t)\tau\text{III}(\nu t) \supset \sum_{n=-\infty}^{\infty} G(f + n\nu). \quad (17.8)$$

In general, these copies of G will overlap, a phenomenon called *aliasing* that causes irreparable loss of information. We had a closely related problem with positive and negative frequency lobes overlapping when we discussed modulation—but that was just two spectral lobes; here there are ∞ . This effect is what gives rise to the inaccuracies in Figure 17.4—you can see the computed spectrum flattening out at the ends, where the overlap is occurring.

A component at $f = f_0 + n\nu$ will produce exactly the same set of samples as if it were at f_0 , making the two indistinguishable. On the other hand, if $G(f + n\nu) \equiv 0$ for $n \neq 0$, then no information is lost, and the true transform G is trivially recoverable from the sampled one. Since for real data, $G(-f) = G^*(f)$, we must center the allowable band at 0, so that $G \equiv 0$ for $|f| > \nu/2$. (Such a function is said to be *band limited*, that is, to have compact support in the frequency domain.) This is the Nyquist condition: the sampling rate must be more than twice the highest frequency present in the data. (Energy exactly at $\nu/2$ is a singular case: the cosine term is sampled correctly, but the sine term disappears.) Note that since we can recover the true Fourier transform from the sampled data, our continuous-time notions of bandwidth and 1 Hz SNR have survived sampling absolutely intact, odd as it may seem.

Really band-limited functions are extinct in the wild, and in fact finite bandwidth is incompatible with compact support in the time domain. We can do very well, though, with analog filters ahead of our sampler, and the expression (17.8) furnishes us with an error estimate in practical cases. The ADC contributes noise of its own, after any filtering has been accomplished. This noise of course aliases down into the fundamental interval $[-\nu/2, \nu/2]$.

In order to make the analog filter easier to build, tune, and test, it's worth sampling a fair bit faster than the Nyquist rate (Figure 17.5). The CD audio specification samples a 20 kHz bandwidth at 44.1 kHz, which is only 10% higher than Nyquist, and would need a really impressive analog filter if the sampling were really done that way (it's actually

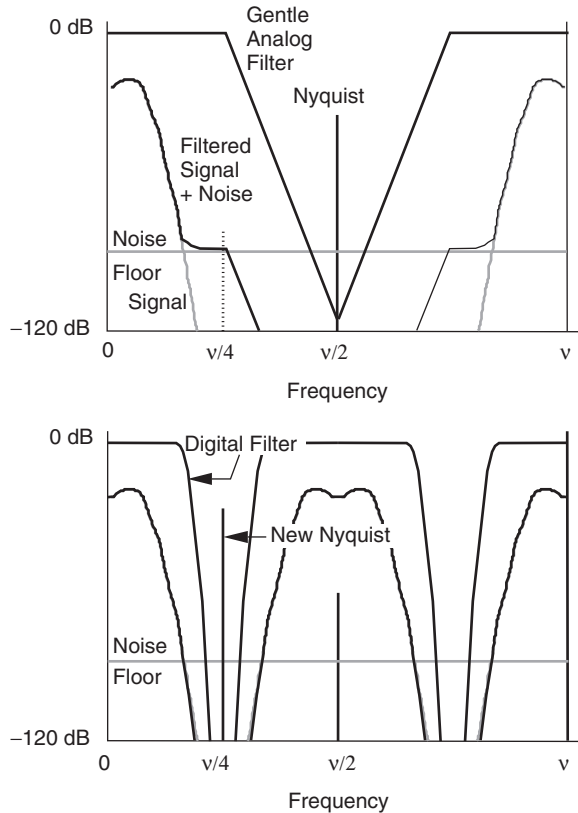


Figure 17.5. It's easier to filter gently, sample quickly, and then decimate. Here the decimation is only $2\times$, but $10\times$ is not uncommon.

done by decimation, see Section 17.8.1). Group delay isn't a big problem in audio,[†] but elsewhere it usually is, which makes the filter just that much harder to build. If we oversample by a factor of 2 or so, the analog filter becomes much easier, and $4\times$ or greater oversampling permits a really simple filter with loose tolerances. The explosion of nearly redundant data this causes can be controlled by the use of decimation, which we'll get to.

17.4.5 Discrete Convolution

We saw in Section 1.3.8 that filtering in the continuous-time case is a convolution operation between the input signal and the impulse response of the filter. The same is true in the discrete-time case, with suitable definitions of convolution; we can connect the two domains via the idea of a sampled function being equivalent to a continuous function times a III function. Convoluting two infinite trains of δ -functions is not pellucid to the

[†]A good deal of money has been made by appealing to people's vanity by telling them that true audiophiles can hear small phase differences. You may or may not have moral scruples about this.

intuition, but a diagram plus a simple limiting argument[†] can be used to show that the convolution of two sampled functions is

$$g(n) * h(n) = \sum_{m=-\infty}^{\infty} g(m)h(n-m) = \sum_{m=-\infty}^{\infty} g(n-m)h(m). \quad (17.9)$$

17.4.6 Fourier Series Theorems

In Section 1.3.8, we stated some of the basic theorems of Fourier transforms. Most of them can be carried along unchanged into the sampled data case, if we use the III function representation of our sampled data. The shifting theorem is identical; the power theorem becomes

$$\sum_{n=-\infty}^{\infty} y(n)x^*(n) = v \int_{-v/2}^{v/2} Y(f)X^*(f)df \quad (17.10)$$

and so does Rayleigh's theorem, which becomes the Parseval relation

$$\sum_{n=-\infty}^{\infty} y(n)y^*(n) = v \int_{-v/2}^{v/2} Y(f)Y^*(f)df. \quad (17.11)$$

Scaling is a bit more fraught, since the Nyquist criterion must be satisfied both before and after scaling[‡]; as long as we remain within that bound, and within the fundamental interval, it is the same. The derivative theorem is replaced by one for finite differences, which follows directly from the shift theorem:

$$\Delta g(n) \equiv g(n) - g(n-1) \supset (1 - e^{-i2\pi f\tau})G(f). \quad (17.12)$$

Aside: Trend Removal. It is of course possible to force the data to 0 at the ends, and even force one or more of its derivatives to be 0 as well, by subtracting a suitably chosen function (e.g., a polynomial). This approach can reduce the number of coefficients required to represent the function to a given accuracy. It is blameless in itself but has an unfortunate tendency to be forgotten—especially when we simply remove a trend line, to force the function to go to 0 at the ends. This is equivalent to subtracting the transform of a great big sawtooth wave from the frequency samples, which is a nontrivial modification. If we don't take account of the effects of the trend line removal on subsequent processing steps, we will be misled, as in the example of Figure 17.6, where removing the trend apparently doubles the frequency of the major component of the data.

If you do trend removal, don't just throw the linear part away—make sure to carry its coefficients along in subsequent steps. This isn't too hard, since we can usually apply the subsequent steps to it analytically, and so keep the true measurement data without being forced to use an unnecessarily large number of coefficients. Just be sure you keep

[†]Construct the two III function as limits of two trains of rectangles of unit area whose widths go to 0, and let one set go to 0 before the other one.

[‡]In case you're wondering why aliasing didn't destroy Rayleigh's theorem, when squaring the function doubles the width of its transform, it's because we're evaluating GG^* at 0, where the aliased copies don't quite reach.

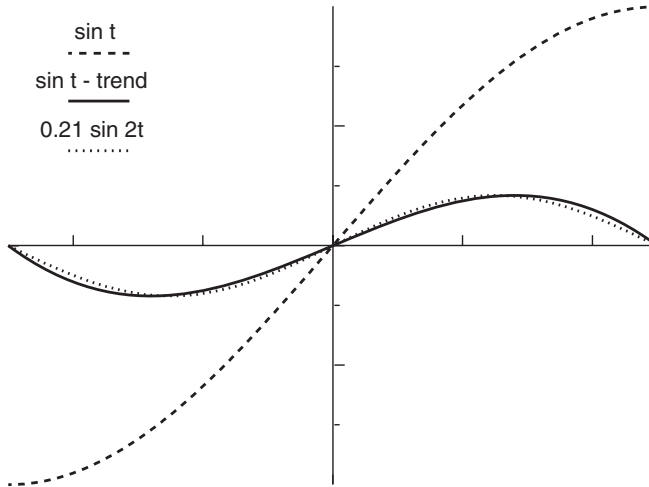


Figure 17.6. Trend line removal can create serious artifacts; here $\sin t$ is transformed nearly into $\sin 2t$.

your normalization straight, and check by comparing the data reconstructed from the big long Fourier series without trend removal to that from the compact form. When we get to spectral analysis with DFTs, having Fourier coefficients that fall off sufficiently rapidly will become important to us, and trend removal can help.

17.4.7 The Discrete Fourier Transform

So far we've used the frequency representation only implicitly. Sometimes we need to know the frequency spectrum of the data, and even more often, filtering is more efficient in the Fourier domain. We've seen that the Fourier coefficients a_n in (17.5) are correct samples of the true continuous transform G of the compactly supported function g , providing we manage to avoid aliasing. Two problems remain. The first one is purely practical: computing a_n appears to require a numerical integral for each sample point, which we probably can't afford; there's a clever algorithm that gets us round this. There is also a conceptual problem that is more worrisome: a finite run of data cannot have a compactly supported transform, so we cannot even in principle get exact samples of the continuous transform G from a finite length of data; the infinity of copies of G will leak into one another at least a bit. Our job is to keep the leakage small and bounded.

The real damage done here is not the approximate nature of our results—we're used to that—but the weakening of our intuition as to what these mysterious coefficients a_n really mean, so that we are less quick to spot nonsensical results. We'll work on keeping that intuition strong.

Once again, we have to change the problem definition to a special case. If we have a periodic, band-limited function $r(t)$, its true, continuous-time Fourier transform can be computed from samples making up exactly one period; the Nyquist criterion requires the number of samples to be at least equal to twice the number of harmonics present (including DC and the fundamental, of course). The result is the discrete Fourier transform, or

DFT:

$$D_n \equiv D(f_n) = DFT(d) = \sum_{m=0}^{N-1} d\left(\frac{mP}{N}\right) \exp\left(\frac{-i2\pi mn}{N}\right),$$

$$d_n \equiv d(t_n) = IDFT(D) = \frac{1}{N} \sum_{m=0}^{N-1} D\left(\frac{m}{NP}\right) \exp\left(\frac{i2\pi mn}{N}\right),$$
(17.13)

where $t_n = n\tau$ and IDFT is the inverse DFT. The IDFT is easier to motivate, in fact, since it's just the Fourier series of (17.5), truncated to reflect the band-limited nature of $d(t)$, and normalized slightly differently. By plugging the DFT definition into the IDFT definition, you can easily verify that the two are really inverses of each other. The normalization is chosen to preserve the useful property that the convolution of two functions is the transform of the product of their transforms.

There is a class of fast algorithms, generically called fast Fourier transforms (FFTs), which can compute the DFT and IDFT in $\alpha N \log_2 N$ operations, where α is around 1, but depends on the algorithm and on whether you count additions as well as multiplications. These algorithms are the most important nontrivial numerical algorithm in current use—the workhorses of DSP. They are fast only for highly composite N , and the simpler and more common algorithms assume that N is a power of 2.

Aside: FFT Routines. Don't write your own FFT routine if you can help it. Lots of smart people have worked on this problem and have spent a lot more time at it than you can spare. Use third party code for this work if possible, such as the open-source FFTW package, or one of several others available on the Internet. These codes all have a bit of a learning curve.[†] Alternatively, the lower performance but much more self-contained *Numerical Recipes in C* code can be a good choice when the FFTs are a small fraction of the execution time of your program. (Note that these routines are *not* in the public domain, but some others of the same sort are.)

Sharp-eyed readers may have noticed that the exponentials used in (17.13) do not obey the Nyquist criterion. They don't stop at $N/2$, but continue up to $N - 1$. What are the higher frequency coefficients for? In fact, they are the negative frequency components and alias down to frequencies between $-\nu/2$ and 0. Thus in order to identify DFT coefficients with frequency, we have to fold the DFT, that is, split the frequency samples at $N/2$, slide the upper $N/2$ coefficients down in frequency without disturbing them, and attach them to the left of 0 (i.e., sample $N - 1$ is really frequency sample -1 , and sample $N/2$ is really $-N/2$).

There's no deep magic here, it's just that the natural range of the index n is $[-N/2, N/2 - 1]$, whereas we parochially insist on using $[0, N - 1]$ (or even $[1, N]$ if we're crusty old Fortran IV bigots). The odd rearrangement of the data is the price we pay for this. If it weren't for the FFT algorithms' requirement that N be a power of 2, we might want to use an odd value of N to make the positive and negative frequency components more obviously symmetric about 0, and to avoid having to worry about the peculiar point at $N/2$, which is exactly at the Nyquist frequency.

The point $n = 0$ in the time domain corresponds to time $t = 0$. Because of this, if you have a function centered at $t = 0$, you have to do the same rearrangement in the time

[†]Watch out for license terms when using open-source code!

domain, if you want an even function of time to produce a purely real transform; before taking the DFT, split the data at $N/2$, and interchange the segments $[0, N/2 - 1]$ and $[N/2, N - 1]$. This is just a shift of time origin in the periodically extended function, so it isn't mysterious either.

The DFT has one or two important theorems associated with it, especially the DFT version of the Parseval theorem:

$$\sum_{n=0}^{N-1} x_n x_n^* = \frac{1}{N} \sum_{n=0}^{N-1} X_n X_n^*. \quad (17.14)$$

Aside: DFT Blunders. DFTs are very blunder-prone in practice. We get the folding wrong, load the data one sample off, or forget that the indices go from 0 to $N - 1$ with 0 counting as a positive frequency sample. Fortunately, it's easy to check that you've got the rearrangement right: try taking the DFT of a real, even function with no flat places, and make sure the transform is real and symmetric as well. If you've got 0 in the wrong place, the transform will have symmetric magnitude but will have a phase ramp on it, and if you've flopped the negative times or frequencies incorrectly, it won't be real and symmetric. Really bang on it to make sure you've got it right, and whatever you do, don't just poke things until it seems to work. Rearranging for even-order DFTs is easier—you just do a circular shift of $N/2$, that is, sample n in normal order is sample $(n + N/2) \bmod N$ in DFT order, and the operation is symmetrical—to shift back, just apply the same shift again. For odd-order DFTs, which are sometimes useful, it's a bit more subtle: forward and backward shifts can't be the same since they have to add up to an odd number. When the DFT returns, DC is at sample 0, whereas in normal order DC is at sample $(N - 1)/2$. Thus sample n in DFT order is sample $(n + (N - 1)/2) \bmod N$ in normal order, and sample n in normal order is sample $(n + (N + 1)/2) \bmod N$ in DFT order. (See Figure 17.7)

17.4.8 Does the DFT Give the Right Answer?

No, not exactly, unless you really live in the special case above (if you do, send a postcard); but as usual we can fix it up so that it does, in principle, to any accuracy we like. The main problem is spectral leakage, where energy that belongs in one bin appears in another. Leakage arises in two ways: overlap of the true Fourier transforms of the compactly supported data we actually have (aliasing), and frequency components that don't go through an integral number of cycles in the time P .

The aliasing problem is easily spotted in practice; generally speaking, if your computed DFT coefficients go to 0 faster than linearly as you approach $N/2$ from both sides, overlap leakage is not serious (taking more samples will help somewhat if it is). Serious aliasing usually leads to the a_n flattening out to some nonzero magnitude near $N/2$, although phase cancellations will often lead to peaks and nulls rather than a smooth approach.

Components at frequencies between frequency samples is usually a worse problem. It leads us to the idea of data windowing, which will solve both problems more or less completely.

17.4.9 Leakage and Data Windowing

Any given data set must contain a finite number of samples, drawn from a finite length of time, which violates the assumptions of Fourier analysis. Truncating the real measurement

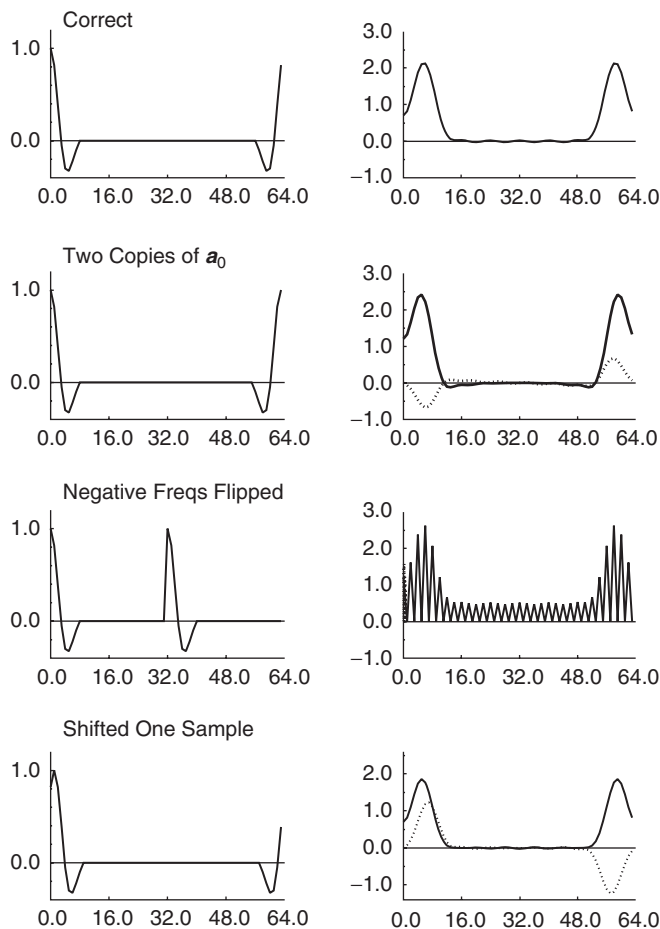


Figure 17.7. DFT blunders are easily spotted experimentally, using even and odd test functions and insisting that the theorems hold. Even big mistakes like these ones can easily go unnoticed with arbitrary data, which will turn your measurement into nonsense.

data in time this way is mathematically equivalent to multiplying it by a rectangle function; thus by the time we get our hands on it, the true continuous transform G has been convolved by the sinc function transform of the truncation window as well as by the shah function from the sampling. The sinc function convolution leads to nasty ripples and other artifacts in the transform, which are familiar in mathematical physics as the Gibbs phenomenon and in physical diffraction as Airy rings (see Section 5.6).

Convolution is commutative and associative, because it is multiplication in disguise; thus the transform we actually get our hands on out of the DFT is

$$DFT(g_n) = G * (Psinc(fP) * \text{III}(f\tau)). \quad (17.15)$$

The III function represents the wraparound, and the sinc function the leakage due to truncation in time. Thus each sample of the DFT is equivalent to the output of one

channel of a bank of identical filters, each with a sinc characteristic in frequency. If we slide the DFT—that is, recompute it every time a new time sample arrives (letting sample N fall off the end each time), we get N filter channels, each with an identically shaped sinc function filter characteristic, and each of which has the same data rate as the input.

The sinc function dies away only slowly as we move out in frequency and, in fact, is itself a source of significant wraparound as well as leakage into nearby channels. Figure 17.8 shows the problem; the sinc function centered on sample m is zero at all the other samples, so there is exactly 0 leakage provided that the function really contains only harmonics of $1/P$. However, we don't live in that case really, and as an estimator of the real continuous spectrum we care about, it stinks. There is a workaround for this problem, which is called *data windowing*.

17.4.10 Data Windowing

We saw that the output of the DFT is samples of the true spectrum G after corruption by a repeating sinc function, and that the sinc function arose from multiplying the true time series by $\text{rect}(t/P)$. If we don't like the sinc function, we'll have to change the rect function, and weight the data samples unequally, as shown in Figure 17.13a,b. A visual metaphor for what we're doing to the data is to imagine samples trooping by outside our office window, abruptly appearing at one edge and disappearing at the other; we need to make it appear more gradually by tinting the edges. Since this effectively narrows the sampling window somewhat, we can expect the FWHM frequency resolution to deteriorate accordingly, but that's a good trade if we can avoid wholesale smearing by the sinc function. As shown in Figures 7.9 and 17.10, there are a whole bunch of different windows you can use, which have different trade-offs of main lobe width (i.e., frequency resolution) versus close-in and distant sidelobe levels. Which is best depends on your intended use. A good general purpose window, and one that is easy to remember,

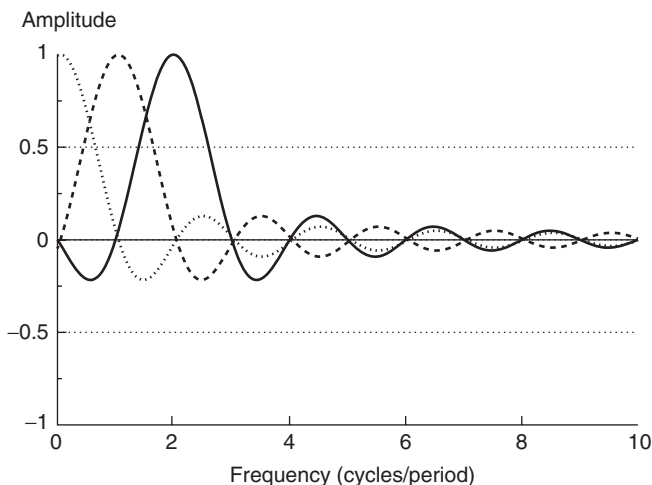


Figure 17.8. Each sample of a DFT is equivalent to the output of a filter whose transfer function is $\text{sinc}(f - n\tau/N)$. This function is 0 at all the other sampling points, but not in between them.

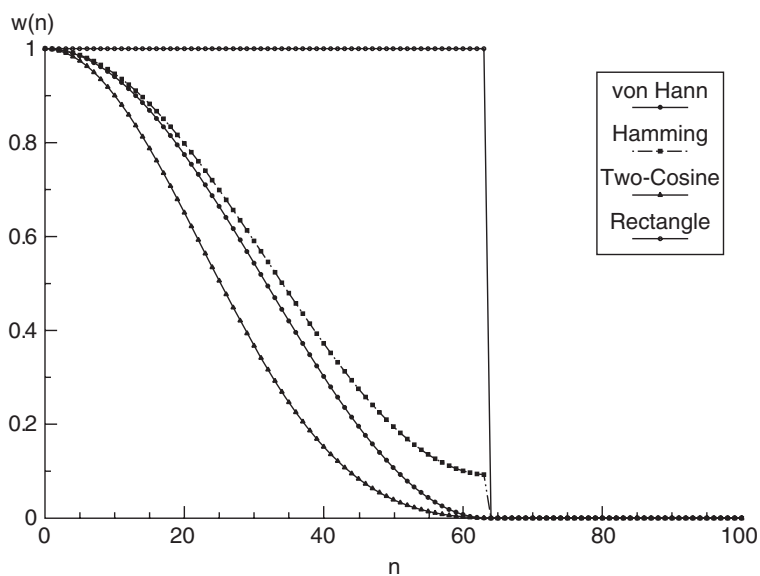


Figure 17.9. Window functions (128 points wide).

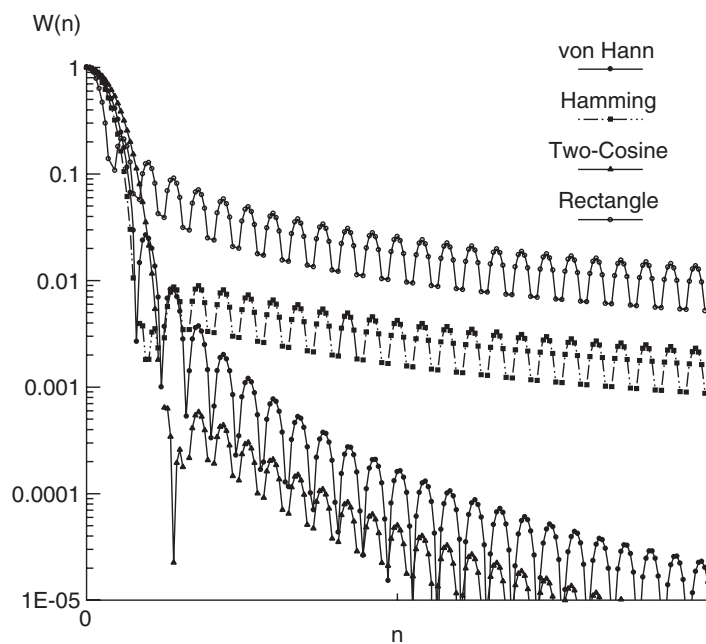


Figure 17.10. Transforms of the window functions of Figure 17.9 (log magnitude). The trade-off between main lobe width and sidelobe selectivity is apparent.

is the von Hann raised cosine

$$W_V(n) = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N}\right), \quad (17.16)$$

which goes to 0 quadratically at the ends, $n = 0$ and $n = N$ (remember the periodicity). The main lobe of its transform is about twice as wide as the sinc function, but because of its smoothness, its transform falls off strongly at large offsets.[†]

The first sidelobe is about -30 dB, much better than the -18 dB of the second sidelobe of the rectangle (which is at the same frequency offset), but no great shakes if you need to separate two closely spaced sinusoidal components of very different amplitude. To get better close-in sidelobes at the expense of ultimate rejection, try the Hamming window,

$$W_H(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right), \quad (17.17)$$

which is a linear combination of the rectangle and von Hann, chosen to cancel the first sidelobe of the von Hann. Its worst sidelobe is the third, which is more than 40 dB down. Elementary DSP books all have lots on windows, although usually with emphasis on their use in digital filter design. The two-cosine window is a linear combination the von Hann and its square, in the ratio of 26.5%:73.5%, chosen to minimize the largest sidelobe—it's an application of the Hamming technique to the second derivative at the patch point. This sort of window-cobbling procedure can help a lot in special situations.

The asymptotic falloff of the true continuous transform of the window function is governed by the window's order of continuity (including its ends); if the m th derivative is continuous, the transform falls off at least as $f^{-(m+1)}$. Removing a trend line from the data will get you an extra 1 in this exponent, at the expense of needing a separate scheme to keep track of what happened to that sawtooth wave in subsequent steps.

A properly windowed DFT, with enough samples that the transform has smoothly decreased by at least 60 dB by the time you reach $f = \pm\nu/2$, will give you good results most of the time; you can trust it about as well as a spectrum analyzer. Do pay attention to the normalization of your spectra, and remember that the spectral samples are no longer perfectly independent, as they were in the sinc function case; just because your one sinusoidal component is now spread over 3 bins doesn't necessarily mean that the total power is equal to the sum of the squares of the frequency samples, unless you've normalized your spectrum that way.

A final practical matter is that we don't really want to increase our data rate and computational burden N times by using a sliding DFT. Each frequency tap has its corresponding filter, and these filters are much narrower than $\nu/2$. Accordingly, they don't need to be sampled at the full rate ν . You can decide how often they need to be sampled by a straightforward aliasing argument, exactly as we did in the first place. A good window function will allow sampling at a rate of perhaps $4/P$ (i.e., $4/N$ times as fast as the incoming data), which corresponds to hopping the window over by $N/4$ samples at each DFT evaluation. This is somewhat surprising at first sight—we all know that you have to overlap lawn mower cuts by $1/4$ or so to do a good job, but that adds only 33% overhead; computing an FFT where $3/4$ of the points have been transformed before

[†]Here as elsewhere we assume that your data has not yet been rearranged for the DFT, if you're going to—always window first, or you'll get “an unexpected result” as they say in software manuals (i.e., garbage).

quadruples the computational burden, which seems very wasteful. Some windows and some problems allow overlaps of only $N/2$, but there's a significant accuracy trade-off, especially if you have a few strong components. If you aren't convinced, use your favorite math program to explore the consequences of reusing too few points, perhaps subtracting off a polynomial first to increase the order of continuity at the ends.

This leads to a piece of advice offered with great earnestness: whatever scheme you come up with, kick the tires hard, all four of them. Plot its expected performance out carefully in both time and frequency, SNR and signal amplitude, looking for aliasing due to the original sampling and to hopping the DFT, leakage due to inadequate window functions, scallop loss,[†] and (especially) blunders. Keep the results, and use them to check your real code for blunders and roundoff.

Aside: Negative Window Coefficients. The windows we have discussed are all positive. The frequency response of the equivalent filter can be sharpened up considerably by the use of negative coefficients as well (e.g., the sinc function, which has a brick wall characteristic in frequency). Should we use negative window coefficients? Well, yes, occasionally.

The noise gain of a digital filter (i.e., total power out over total power in, with a white noise input) is equal to the sum of the squares of its coefficients, while the signal gain goes as the square of the sum. Negative window coefficients thus tend to increase the relative error of the spectral estimates in the presence of noise, so you'll often hear it said that you should avoid them. There's less to that than meets the eye, though; Parseval's theorem says that the noise gain (see Section 13.1) in the time and frequency domains are equal, so for a good choice of window, this increased noise gain will mainly be due to squaring up the shoulder of the filter, which is what we want to happen (Figure 17.11).

In principle, a more general window can reduce the sampling rate required for the individual taps, and so reduce the total volume of data. You'll need to use longer transforms, though, because the main lobe of the window function will of course be much narrower to make room for the extra lobes. This increases the total computational burden, more or less eliminating the efficiency improvement we sought.

Sometimes, though, our data will characteristically exhibit a few very strong Fourier components, and our measurement requires measuring weak signals accurately, or we need to be able to measure the amplitudes of sine waves that aren't periodic in the sampling window with no scallop loss. In that case, the great improvement in the sidelobes that negative coefficients make possible will become very worthwhile, which is why they're used in FFT spectrum analyzers.

17.4.11 Interpolation of Spectra

One nice thing about the DFT is that you can get finer frequency sampling by zero-padding your array to the number of frequency samples you want, and then doing a bigger DFT (Figure 17.12). Since the frequency limits of the spectrum are always $\pm\nu/2$, the extra coefficients come from sampling the spectrum more finely. This introduces no

[†]Scallop loss is the error in the calculated channel power caused by the windowing, hopping, and the departure of the equivalent channel filters from being perfect rectangles. A plot of overlapped filter functions has a scalloped appearance about the peaks, as in Figure 17.8.

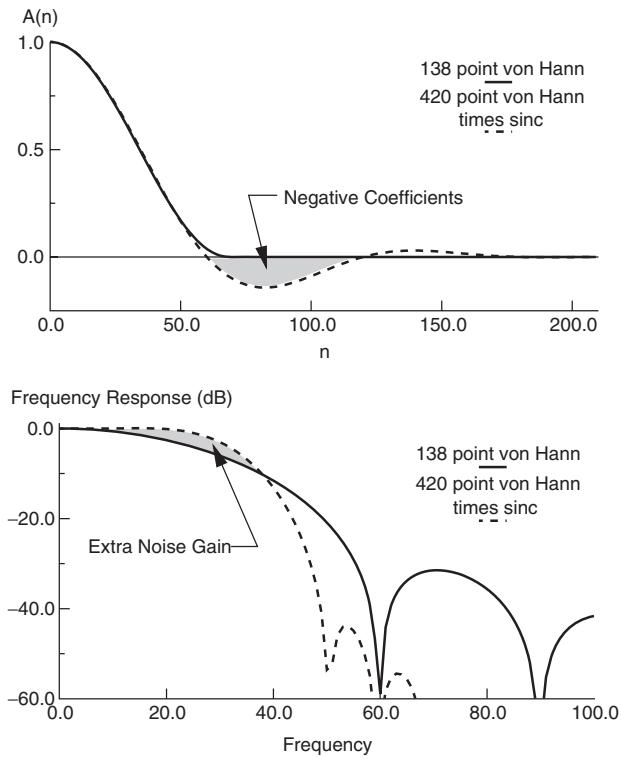


Figure 17.11. Negative coefficients increase the noise gain of the window, but can increase its selectivity. Here the extra noise gain comes from squaring up the shoulder of the frequency response and reducing its sidelobes.

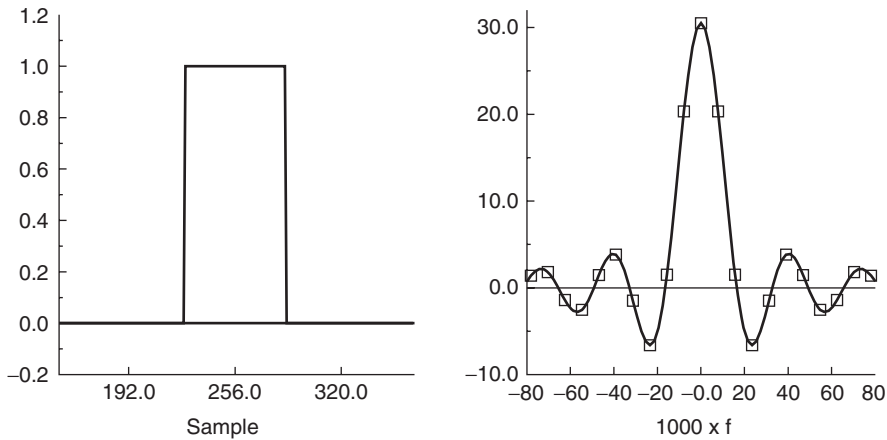


Figure 17.12. Interpolating with a zero-padded DFT: (a) a 31-sample rectangle function and (b) its DFT of 128 points (boxes) and 512 points (solid line).

additional artifacts whatsoever, but it may reveal some that were hiding, especially sinc function leakage due to not having windowed the data.[†]

The innocuous nature of zero-padding makes the usual FFT restriction that N be a power of 2 no problem in practical cases. Zero-padding doesn't mean you can window your 128 data points with a 2048 point Hamming window (why?), nor does it improve the intrinsic resolution of the measurement, which is of course limited by the transform of the window function you use. Zero-padding does reduce scallop loss, since each frequency sample has a filter function centered right on it.

A related benefit is that if we have two data sets that are slightly out of registration, we can shift one with subpixel resolution by taking the FFT, putting the appropriate linear phase ramp on the frequency samples (from the shift theorem), and transforming back. This is especially useful in registering images, or in permitting signal averaging when the scan range is drifting around a bit, due e.g. to Doppler shift.

17.5 POWER SPECTRUM ESTIMATION

Estimating the power spectrum of some signal is a very common thing to want to do. Color temperature meters, spectrometers, RF spectrum analyzers, and frequency counters all perform power spectrum estimation in different ways, with different sets of assumptions about the input function. With proper attention to normalization, you can compute n samples of the power spectrum of $2n$ data points by taking the squared modulus of the DFT (negative and positive frequencies give the same modulus so you only need half of them). The normalization will often involve dividing by the number of data points, but that depends on just what you're trying to compute; if you're interested in the total energy of a certain run of data, you'll just take the sum of the squared moduli.

The key limitation here is noise. Taking the DFT of a short run of your signal will give you the correct Fourier series representation for that particular run, but unfortunately that is quite different from the power spectrum of the underlying random process (signal plus noise). If you take zero-mean white Gaussian noise and square it, the standard deviation in each sample is equal to the mean value. The same is true of its DFT power spectrum: the standard deviation in each frequency bin of the PSD is equal to its mean regardless of how many data points you take; all the extra information you supply goes into sampling the power spectrum more densely, not increasing its accuracy. This makes naive use of the DFT useless in estimating the power spectrum of a noisy signal. (See Figure 17.13.)

Aside: Is There a Right Answer? Even without additive noise, power spectrum estimation is an ambiguous business. Say we have a continuous function of time that has no obvious beginning and ending, for example, Earth-surface solar irradiance on a 1 cm^2 patch at the North Pole. We can take only a finite number (though perhaps a large one) of samples of this function, with limited accuracy, and we want to know the power spectrum of the whole; we are willing to assume that our run of data is in some way typical. Typical in what way? A well-defined answer is not always easy to give. Our example has important time variations on all scales from the age of the Sun

[†]This may surprise some readers, who are used to hearing that the big jumps in the data from 0 to $g(0)$ and $g(N-1)$ cause artifacts—but consider taking the $2N$ point transform of N data points; (17.13) shows that sample $2n$ is identical with sample n of the N -point DFT.

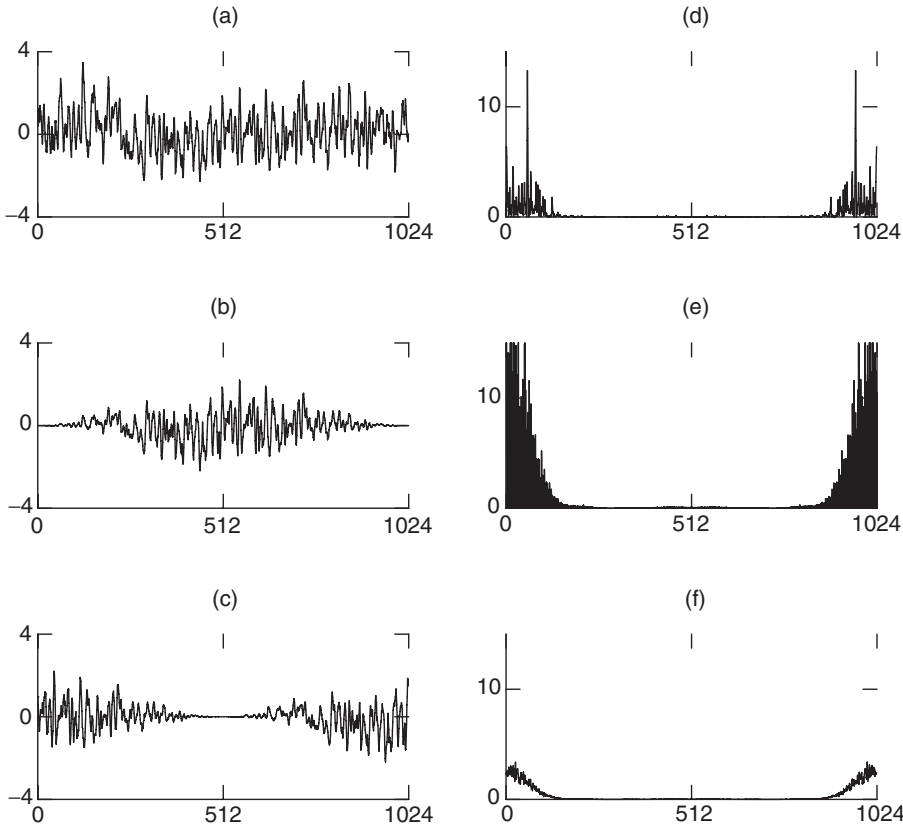


Figure 17.13. DFT power spectrum estimation: (a) short run, (b) von Hann windowed, (c) rearranged for DFT, (d) short-time PSD, (e) PSD of $64\times$ longer run, and (f) summed $64\times$.

and continental drift, through cloud motions and the shifting shadows of snowflakes, a frequency range of perhaps $10^{19}:1$. We can perhaps sample 1 part in 10^9 of the whole time series. Looked at too closely, even that short run is somewhat ill defined, because the surface moves around, for example, with snowfall and ice pack motion. Our strategy will obviously have to depend on what we want to get out of the data that we have—what is noise to radio communications people may be signal to astronomers.

The mathematical power spectrum estimation problem assumes that we have a finite number of data points, quantized at some resolution; that we want to estimate the power spectrum of the true continuous random function from which they were drawn, without reference to times before and after; and that we know that the signal's time autocorrelation function is well represented by one member of some limited class of model functions (trigonometric polynomials in the case of the DFT). We usually assume that the function is ergodic. This is a well-defined problem, which can be solved efficiently by a number of methods. The key thing to remember is that this isn't the same problem as the one we really want to solve, and unless we choose the sampling strategy and the set of model functions with care, the answers it gives will be different as well. Watch out for reasonable-looking wrong answers, here as elsewhere.

17.5.1 DFT Power Spectrum Estimation: Periodograms

One way to combat the noisiness of the power spectrum estimate is by averaging. There are two main ways to do this, either summing the energy in n adjacent bins in a big long $2N$ point windowed transform, or by shift-and-add: split up your data points into n chunks of $2m$ points (remember to overlap the chunks by 75% or so), window, transform each one, compute the squared modulus to get n estimates of the m point power spectrum, and sum the results. Shift-and-add evaluation of power spectra is also called the *method of modified periodograms*.

Both procedures reduce the variance by a factor of the number of independent data points, more or less. Both are reminiscent of a signal averager; summing adjacent bins is something like signal averaging the autocorrelation of the function in the time domain, whereas shift-and-add is like signal averaging the power spectrum in the frequency domain. Of the two, shift-and-add is modestly cheaper in CPU cycles and less wasteful of data, since instead of a single big wide window, you're using many overlapping copies of a narrower one; thus almost all the samples have a chance to be in the middle of a window, and so contribute equally to the measurement. Binning the spectrum is a lot easier to code, though, and less blunder-prone as well. Use shift-and-add, but use binning as a sanity check during debug.

If your data are unequally spaced, or you have gaps in them, these simple algorithms are inadequate. For those cases, try Lomb's method (see, e.g., Rabiner and Gold).

17.5.2 Maximum Entropy (All Poles) Method

There's nothing intrinsically more accurate about using trig polynomials as fitting functions. In approximating functions we usually prefer rational functions, because they are a lot better at reproducing asymptotes (vertical or horizontal) or other non-polynomial-ish features. Power spectra often have similar behavior (e.g., isolated Lorentzian peaks), so rational functions of $\exp(j2\pi ft)$ seem like good candidates for model functions. If we make our rational function the reciprocal of a trig polynomial, this idea leads to the *maximum entropy method* (MEM) or *all-poles method* of Figure 17.14, where

$$f(t) \approx \frac{1}{\left(\sum_{k=-N}^N a_k e^{j2\pi k f t}\right)}. \quad (17.18)$$

Note that this estimate is not band limited.

The maximum entropy method is a good way of pulling sharply peaked things out of noise; it amounts to putting in a priori knowledge that it's a few sharpish peaks in the frequency domain that you're looking for. If that's the case, you'll often get good estimates with relatively few terms, say, 5 or 10 times the number of peaks you expect. Don't go above the square root of the number of points, or you'll start getting a lot of spurious peaks—MEM is a bit like polynomial fitting that way. It is also very poor at fitting data sets containing pure tones—the condition number gets big. The *Numerical Recipes in C* MEM program doesn't tell you the condition number, so some numerical experimentation is required. Because the fit is nonlinear, the condition number depends on the data set. A reasonable check is to add some pseudorandom noise and recompute, looking for large deviations in the reconstructed spectrum.

If your data contains a strong pure tone which is cut off sharply at the ends of the interval, the true Fourier transform will have strong sidelobes, which will look like

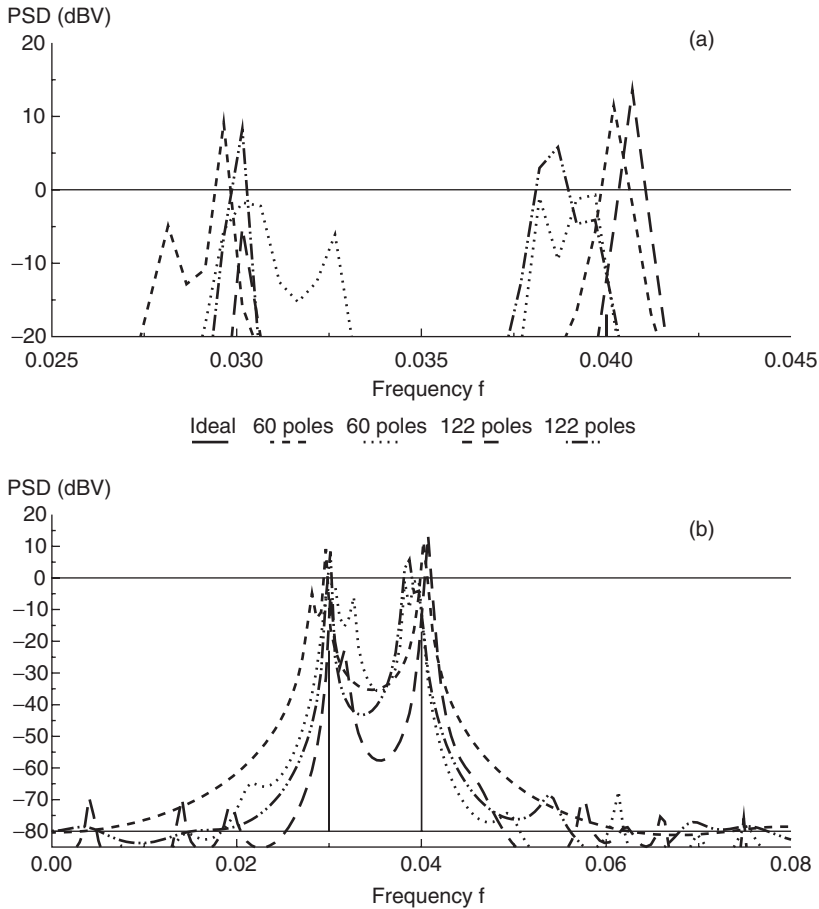


Figure 17.14. MEM power spectrum for 256 samples of a sum of sine waves, 0.1 V at $f = 0.03\nu$ and 0.2 V at $f = 0.04\nu$, plus $100\ \mu\text{V}$ rms noise, with 60 and 122 poles and two starting phases: (a) peaks and (b) full range.

additional poles and which will fall off slowly with frequency. Accordingly, negative and positive frequency lobes will overlap, causing peak shifts that are sensitive to the phase of the tone in the data. A good test is to move the data around a bit; if you have 377 data points, try comparing the MEM results from points 19 to 376 with those from 0 to 357. The phase shifts that result will shift the sidelobe overlap in the frequency domain. If the peak position moves significantly, you have an overlap problem.

Besides being ill conditioned, MEM takes quite a lot of computation and is not easily extended by averaging, periodogram-style, because the average of two functions with M poles is in general a function with $2M$ poles. If you have lots of data and are CPU limited, it may pay you to use a simpler technique on a bigger data set.

Example 17.2: Film Thickness Measurement with a Fiber Spectrometer. Thin dielectric films tend to produce optical spectra with lots of peaks and nulls, where the film thickness is an even or odd multiple of $0.25\lambda/n$, respectively. One good way to measure

film thickness in situ is to use an optical multichannel analyzer (OMA) spectrometer, fed with a fiber bundle (see Section 10.5.4). From a signal processing point of view, the difficulty is that the detection bins in the OMA are not equally spaced in frequency; they tend to be at equal intervals in wavelength instead. Over the 2:1 range of a typical spectrometer (limited by grating orders overlapping), the reciprocal relationship stretches the low optical frequencies out over more bins than the high frequencies get. There's nothing intrinsically wrong with this.

It is inconvenient for film thickness measurements, though, because a film produces fringes that are equally spaced in optical frequency, and when sampled at equally spaced wavelengths, the frequency is not constant, but chirped.[†] There are several techniques possible for pulling such a chirp out of the noise. The most straightforward conceptually is to resample the data onto equally spaced frequencies and do a DFT. This is not as easy as it sounds, since an OMA spectrometer's output really doesn't represent band-limited data in all probability, and there aren't that many data points (often 512), and the spectrum is not always oversampled. Make sure you window the data correctly, or sidelobe leakage will be troublesome. Resampling is somewhat fraught with difficulties, as we saw earlier, and is prone to give badly wrong answers if you don't oversample enough.

Other techniques are possible. You can also use brute force correlation and fit a mathematical chirp to the data, using some reasonable criterion such as minimizing the absolute value of the residuals, or maximizing the (normalized) cross-correlation. This takes a lot of CPU cycles compared to the DFT, especially if you don't already know something about the thickness and refractive index you expect. It also suffers from the tendency of correlation methods to exhibit multiple peaks and to jump about as the height of one peak passes that of the previous champion, perhaps due to a phase shift in the chirp. It will work well if there are many cycles in the chirps (i.e., thick films), so that the subsidiary peaks are well suppressed.

Still another approach is to resample and use the maximum entropy method to pull out the peaks. This seems to be less sensitive to the artifacts from resampling than the FFT method, but is much more CPU intensive, which may matter if you're doing fast measurements with a slow computer (industrial process control applications often have to run on some overloaded old PC that makes an 8088 seem fast).

If there is a whole lot more data available than you can process in the time available, you can use a simpler approach: filter gently to get rid of the high frequency noise (use a symmetrical FIR filter to avoid phase shifting the chirp), and then just use local interpolation to find the peaks of the chirp. The peak positions provide independent estimates of the film thickness, so that a peak that has been estimated incorrectly (perhaps due to noise) will stick out. Averaging a whole lot of these values may get you better measurements than a fancier approach to a small data set, especially if there is intrinsic variability to the sampled data—for example, in polishing substrates, where the thickness is bound to be somewhat nonuniform.

The best advice in a case like this is to live with your data set for a while and try several things. Look for average-case accuracy, but use enough data that you know about how often rogues come along. Save all the rogues to disc for testing your refined algorithms. Above all, don't be a signal processing snob—use what works, know why it works, and be able to prove experimentally that it works.

[†]Sinusoids whose frequencies change with time are called chirps after bird songs, which have this characteristic.

17.6 DIGITAL FILTERING

Most modern optical systems produce digital data at some point, and this trend will only continue. Digital filtering is so cheap, flexible, and powerful that it is an indispensable part of the equipment of a modern electro-optical systems designer. Newcomers to this art should go look at Hamming's undergraduate book on digital filters. He goes through this stuff with lots of hand-holding and comforting detail, which is very important since there are lots of ways to get digital filters wrong.[†]

For designing digital filters, commercial software is available, but nevertheless it is highly worthwhile to learn about z -transforms. They are not difficult to learn or to use, and in providing a conceptual bridge between the continuous-time and sampled domains, they really make digital signal processing easier to understand.

In this book, we'll just dip our toe into the subject and talk about designing *finite impulse response* (FIR) filters[‡] using windows, which is a very broadly useful method. FIR filters whose responses are symmetrical in time (e.g., a symmetrically windowed portion of $\text{sinc } x$) have exactly constant delay with frequency, which is an enormous benefit; a symmetric N -point filter has a delay of exactly $(N - 1)\tau/2$ seconds. Unlike recursive filters,[§] FIR filters raise no stability issues and exhibit no *limit cycles*[¶] due to roundoff error. In situations where phase linearity is not a requirement, recursive filters are more efficient, and the efficiency gain increases with increasing filter order. On the other hand, optimal FIR filters are more efficient than delay-equalized recursive filters.

It is usually best to use filters of odd order, so that the delay can be an integral number of samples, but some types of filters (e.g., differentiators) work better in even order. Unless your system has an embedded DSP, it's usually best to do your filtering in software on a host PC—it's much easier and more flexible.

Aside: Zero-Order Hold. One sometimes overlooked effect of converting from analog to digital and back again is time delay. We usually don't ignore the one-clock delay between the CONVERT command to the ADC and latching the data into a register, but sometimes forgotten are the delays caused by finite track/hold bandwidth (so that our sampling time is slightly before the track-to-hold transition) and the zero-order hold on DACs. Remember that our whole treatment of sampled-data systems was based on a series of mathematical impulses, whereas real systems use rectangular pulses of width τ instead, the *zero-order hold* of Figure 17.15; in sampled-data terms, a DAC convolves the ideal output with $\text{rect}(t - \tau/2)$, which is a filtering operation. As we know, this produces a delay of $\tau/2$ and a sinc function rolloff with frequency; this is a large effect up near the Nyquist frequency, so don't neglect it in your designs (especially with actuators).

[†]For example, computing the FFT, chopping off the high frequencies, and transforming back, or automatically removing a trend line before filtering. If it isn't obvious why this is silly, go read Hamming.

[‡]Finite impulse response means "finite-length impulse response," that is, a filter that can be implemented by discrete convolution with a finite-length function h .

[§]Nonrecursive (FIR) filters produce a weighted sum of their inputs, whereas recursive (IIR) filters use the previous values of their outputs as well; this is the equivalent of a linear circuit with feedback, and leads to poles in the transfer function, whereas FIR filters have only zeros.

[¶]A limit cycle is a self-sustaining output sequence in the absence of an input signal.

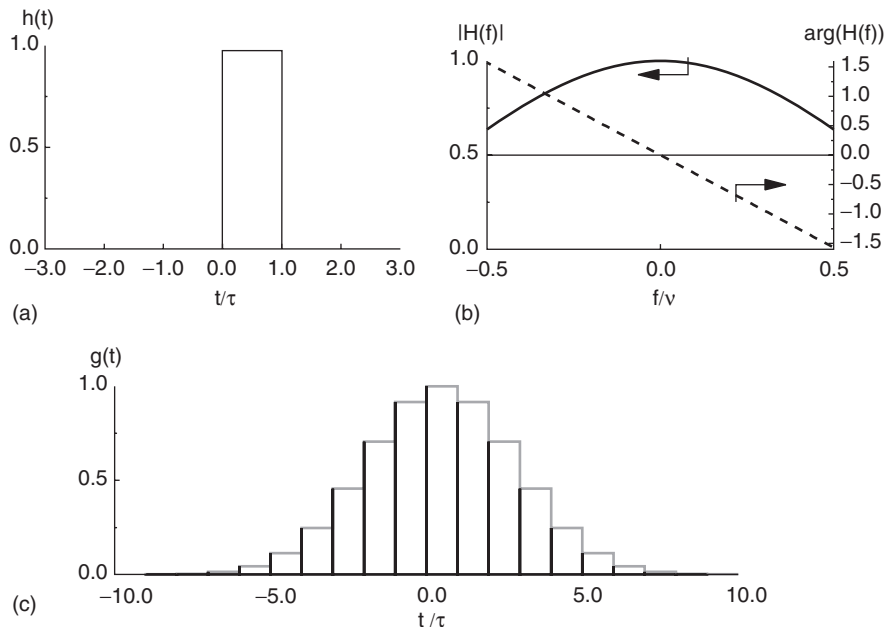


Figure 17.15. A zero-order hold produces a sinc rolloff and a half-sample delay: (a) the continuous-time smoothing function, (b) transfer function, and (c) result of the hold operation applied to delta-function samples.

17.6.1 Circular Convolution

We alluded to the use of the DFT plus multiplication to perform discrete convolutions, but we can't do an infinite sum with a DFT to evaluate (17.9), so how does that work?

Well, remember that the DFT gives correct samples of the aliased version of the Fourier transform of our continuous given function, assuming that the given function is periodic (the normalization is also changed to produce finite results). Thus we expect multiplying the DFTs of two functions together and taking the IDFT will produce correct samples of the convolution of the two periodic functions, which it does; the result is called the *circular convolution*. Since the width of the convolution is in general the sum of the widths of the functions, each period will smear out into two periods in general, producing serious wraparound errors everywhere.

In order for the circular convolution to produce the correct results for our discrete convolution, we'll need the functions to have compact support, and for them and their convolution to be able to fit in the N points available. Since the convolution of two sequences of lengths L and M is $L + M - 1$ samples long, the two must obey

$$L + M \leq N + 1, \quad (17.19)$$

or wraparound will occur (if it isn't obvious where the 1 comes from, consider convolving with a one-point function, which is the discrete version of a δ -function). Providing (17.19)

is obeyed, multiplying the N -point DFTs of two functions and taking the IDFT of the result yields their correct discrete convolution without wraparound.[†]

This discrete convolution algorithm is the basis of FIR digital filtering, which is convolution with the impulse response of the filter function, just as it was in analog. The way you do it in practice is to choose N large enough, zero-pad both functions up to length N , multiply the transforms, transform back, and unfold the resulting data. (Try it out with cases you know a priori before trusting that your code is correct—near enough is *not* good enough.)

Once again, apart from aliasing and the effects of windowing, the discrete convolution yields samples of the continuous convolution of the continuous functions g and h .

17.6.2 Windowed Filter Design

A brick-wall lowpass filter that cuts off at $\pm f_0$, $BW(f) = \text{rect}(f/(2f_0))$, has impulse response

$$g(t) = \frac{\sin(2\pi f_0 t)}{\pi t}. \quad (17.20)$$

Since this is a band-limited function, we can sample it without aliasing at any rate $f_s > 2f_0$, so obtaining an infinitely long impulse response. As a practical matter, we need to cut it off at a finite number N of coefficients. Chopping off the impulse response sharply leads to artifacts and ringing in the frequency domain as we saw when we discussed data windowing in Section 17.4.9, so as we did there, we make $g(t)$ go smoothly to 0 at $t = \pm T$, by multiplying by a window function (Figure 17.16). Providing T is at least a few times $1/f_0$, the approximation is quite reasonable, and the effects of windowing on the transfer function are easily checked via DFT. This is a good technique for generating FIR filters; the filters so obtained are not optimal, but they are great for getting something to work quickly and for making trade-offs of bandwidth, measurement speed, and noise. Once you get close, you can use some optimizing program to make the perfect filter.

17.6.3 Filtering in the Frequency Domain

We saw how to specify a filter in the frequency domain in Sections 1.8.1 and 13.8. Digital filter specs are similar, except that more attention is often paid to phase linearity, because we don't have the tuning problem with digital filters—what comes out of the design program is the filter itself, not just the L and C values. Also, at least with FIR filters, it's easy to get perfect phase linearity: just make the filter coefficients symmetrical, so that in an $(N + 1)$ -point filter, $C_n = C_{N-n}$.

Because the FFT convolution algorithm is fast, frequency-domain filtering is generally preferable. There are important special cases, though, where you should do it in the time domain: real-time systems, where it is unacceptable to wait N sample periods before producing output; filters with only a few coefficients, which take less than the several times $\log_2 N$ floating-point operations per point of FFT convolution; or where you need very sharp skirts, and don't care about the group delay, which strongly favors elliptic IIR filters with very long tails in time.

[†]You can also use a shift-and-add technique, which fixes up the wrapped parts afterwards, but there's no significant efficiency gain, and it's more blunder-prone.

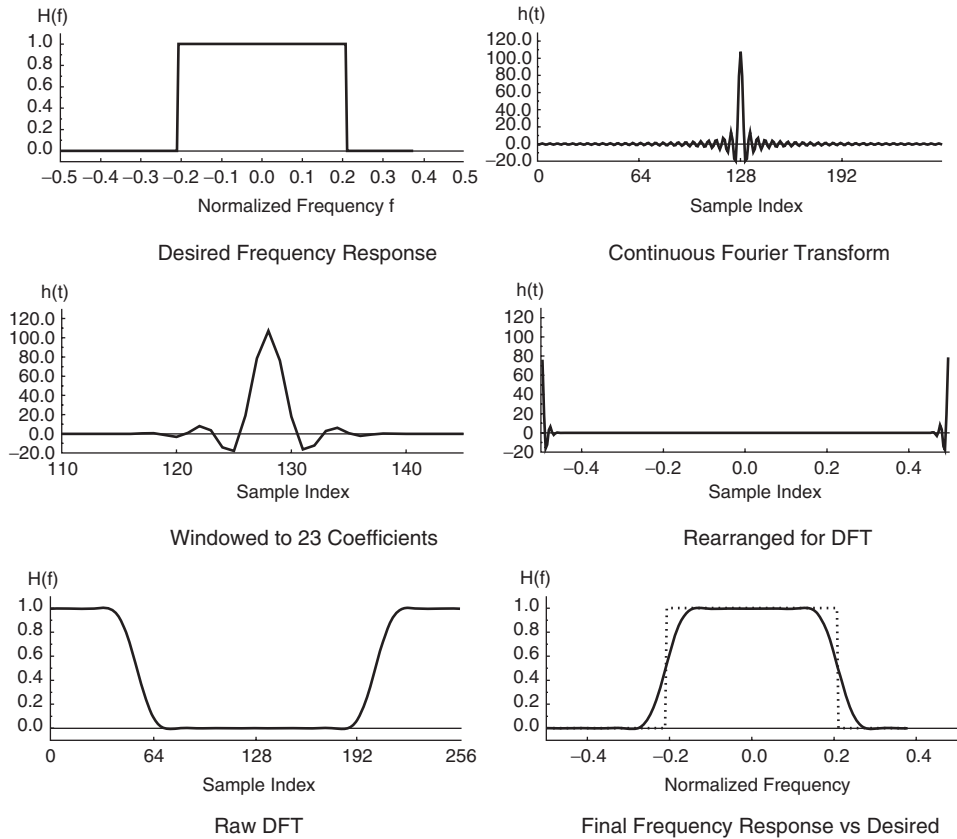


Figure 17.16. Windowed FIR filter design.

17.6.4 Optimal Filter Design

There are packages for digital filter design, which produce results that are optimal in some precise sense. Simple ones use a sampled-frequency representation, with the coefficients in the transition band left indeterminate; a numerical optimizer then optimizes the impulse response (i.e., the filter function in the time domain) by minimizing the number of coefficients required, handling ringing and overshoot, minimizing time delays, or whatever combination you specify. The most widely used FIR filter design program is the classical one by Parks and McClellan, originally published in Fortran but available as public-domain C code in various places on the Internet; search on “Parks–McClellan” and you’ll find it. It’s quite a good program, supporting multiple passbands and stopbands with different (specified) gains, and different weights for the rms error in each band.

More advanced packages can find the true optimum filter, including minimizing sensitivity to roundoff error. You just have to be careful what you wish for, because some of these theoretically optimal filters may have properties you don’t like, for example, deconvolution filters with sharp frequency peaks that make noise look like ringing.

17.7 DECONVOLUTION

17.7.1 Inverse Filters

Ordinary lowpass and bandpass digital filtering are the workhorses of DSP, but especially in electro-optical instruments, there is a uniquely valuable function digital filters can perform: eliminating obnoxious instrument functions. Calibration can be looked at as doing this in the real-space domain, for example, by removing the effects of etalon fringes or the $2.2\text{ }\mu\text{m}$ absorption band in the quartz bulb of a tungsten-halogen lamp. There are other systems such as phase-sensitive confocal microscopes (whose coherent transfer functions are nearly triangular instead of flat), or phase-sensitive systems suffering from defocus, where the instrument function is well known a priori, and the full complex data are available.

An inverse filter can produce results close to magic in these sorts of cases (Figure 17.17): just take the transfer function you want, divide it by the one you've got (window the result appropriately to avoid dividing by very small numbers), compute the Fourier transform of the result, take a whole bunch of samples, window appropriately, and you are in possession of a digital filter that can turn some ugly transfer function

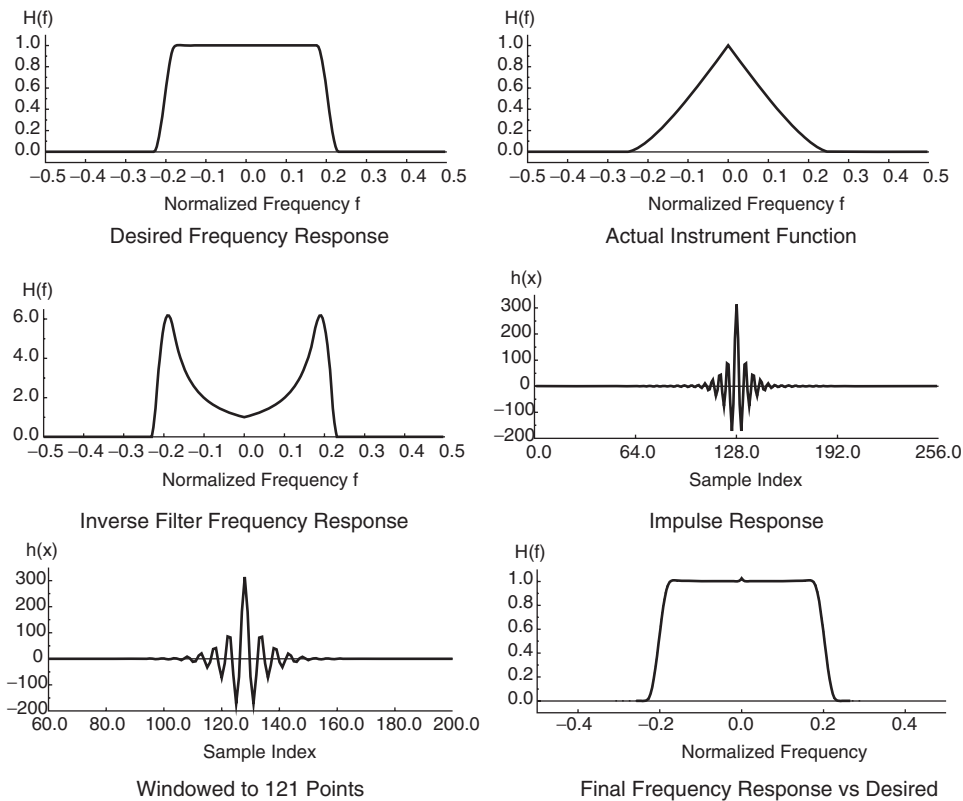


Figure 17.17. Inverse filter for a phase-sensitive confocal microscope.

to sheer beauty, at least part of the time. Used cautiously, this can make a factor of 2 difference to already-good microscope data,[†] or extend your depth of focus by 10×.

Be really careful about what you do with the edges of your instrument function if it's not fairly uniform; multiplying by the inverse of a Gaussian smoothing, for example, will lead to tears. Keep an eye on the noise gain, and watch especially for tall, narrow gain peaks—they make noise look like signal, which is a particularly obnoxious trait. Use numerical experiments with canned data and noisy fake data to make sure you've got this right.

17.7.2 Wiener Filters

Wiener filtering is a generalization of the inverse filter idea that explicitly takes account of finite SNR. It concerns how to get the best estimate of a function g that has been smeared by some (undesired) filter h and then afterwards corrupted by additive random noise r . The Wiener filter h_W is chosen to minimize the mean square discrepancy between g and $h_W \star (r + h \star g)$. Sometimes h is a δ -function, but often it isn't.

Construct a filter function

$$H_W = \frac{|G|^2}{H(|G|^2 + |R|^2)}, \quad (17.21)$$

and use that for your deconvolution. The Wiener filter provides most of the advantages of the inverse filter, with an automatic way of controlling the noise gain. (It is interesting to note that h_W , which acts on the signal amplitude, is obtained from the ratio of power spectra.) If your signal has a sharp falloff with frequency, this technique can help a fair amount and is well worth trying.

The Wiener filter is a least-squares optimum, and like most least-squares things it is nice for proving theorems but less useful in real life except to generate a reasonable first guess; controlling the deconvolution filter's shape by varying the length of the window used on the inverse filter is often a better bet unless your SNR changes a lot—and if it does, you'll need to think and test very carefully to make sure your subsequent processing doesn't go wrong when h_W changes.

17.8 RESAMPLING

Much of the time, the natural time synchronization for sampling the data is not the natural one for processing it. Correctly interpolating the data from one equally spaced mesh to another is easy—take the DFT, zero-pad it to the right length, and transform back; if you use the chirp- z transform algorithm (see e.g., Oppenheim & Shafer), N can even be a prime number (remember how the sinc functions overlap away from the original frequency samples, though). The chirp- z transform allows you to choose your frequency samples on any arc of a circular spiral in the complex f plane, so you can have any sampling resolution you like, and fine frequency samples can be had without computing the DFT over the entire unit circle at the same resolution, as you must with zero-padding.

[†]Philip C. D. Hobbs and Gordon S. Kino, Generalizing the confocal microscope with heterodyne interferometry and digital filtering. *J. Microsc.* **160** (3), 245–264 (December 1990), <http://electrooptical.net/www/confocal/confocal.pdf>

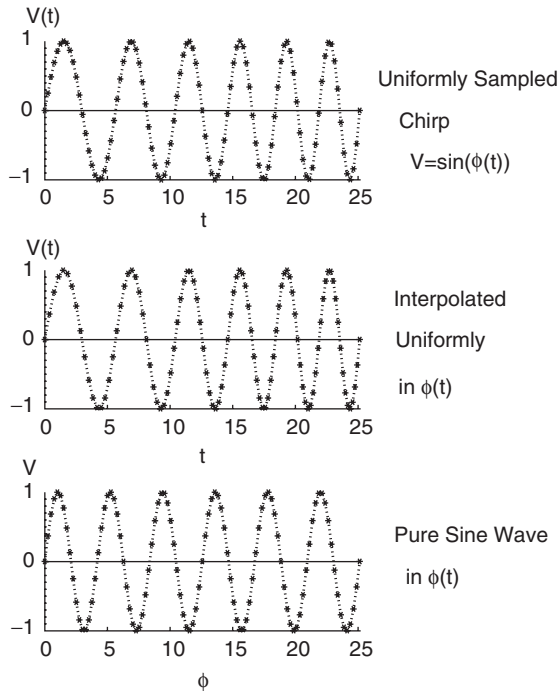


Figure 17.18. Resampling.

A seemingly similar but actually much harder problem arises when the data are not equally spaced in the right independent variable; an example is an OMA spectrometer of Example 17.2, where the output bins are equally spaced in λ and not f . This leads to the idea of resampling: using interpolation to move the sampling points from where they are to where we would like them. It is often possible to do this well, but great care must be used—resampling is an even more effective generator of reasonable-looking wrong answers than DFT power spectrum estimation. You have to multiply by the Jacobian, that is, the ratio of the bin widths before and after, and even that only levels the mean value; once you’ve stretched the bin widths, the noise floor will no longer be flat.

There are good and bad reasons for resampling. The three most common are: (1) it saves labor and expense in making the instrument linear in the first place (e.g., by using a rather costly f - θ lens on a scanner); (2) the signal has nice properties when resampled (e.g., a chirp could turn into a CW sinusoid as in Figure 17.18); or (3) the independent variable is inherently nonlinear in a way that must be compensated for in order to perform further processing.

Before using resampling, look hard at whether you can change the apparatus slightly to eliminate the need for it. You can vary the sampling clock rate to force scan data to be equally spaced, for example, or use a Fourier algorithm for unequally spaced data.[†]

The main danger in resampling is *truncation error* (see Section 17.4.2) due to the interpolation inherent in the procedure. Equally spaced data have a built-in translationally invariant interpolation method, that is, summing the sinc functions for each data point

[†]*Numerical Recipes in C* has a section on this.

in the set; as soon as you change the mesh, that stops being true, and your subsequent Fourier processing steps lose their mathematical foundation. Unequally spaced data have very different (and rather unintuitive) properties from equally spaced ones.

One way of seeing this is in the OMA example. The widths of the bins change under resampling, so each bin must be multiplied by its width before resampling occurs so that energy is conserved (this is the same as multiplying by the Jacobian). But this amplitude adjustment depends on where in the bin the real frequency component actually was; and that information is not present in the OMA output. This is obviously more serious when the original binning was coarse. You should thus ensure that your original data are well oversampled at the shortest frequency present in the signal, and filter really well in analog ahead of time to make doubly sure. Make sure that the same is true in the resampled data. Sometimes you can't do that, but do calculate the theoretical error bounds before proceeding.

Remember, you're making assumptions about how the data are to be reconstructed; the unequally spaced mesh may not have nice properties for reconstruction. Beware, test using many data sets, and once again, use *lots* of sample points.

17.8.1 Decimation

The previous section cautioned against ill-considered use of resampling. One safer special case is decimation, where the sample rate is divided by an integer N .[†] This is safer, since no interpolation is required. The samples of the fast data stream are first filtered digitally, so that the Nyquist criterion is satisfied at the new, slower, sampling rate, and then in every run of N samples, $N - 1$ are simply discarded.

17.9 FIXING SPACE-VARIANT INSTRUMENT FUNCTIONS

Sometimes our instrument function isn't even close to translationally invariant. Consider, for instance, a crude spectrometer using a tungsten-halogen bulb operated at different temperatures plus an unselective detector. The idea is that different bulb temperatures give different-colored light, so that, for instance, we might be able to subtract a measurement at 1500 K from one at 3400 K and get a measurement of sample absorption in the blue. However, not only is the Planck curve wide, but its width is proportional to its peak wavelength, so we can't just do a deconvolution.

In this case, we're back to needing a matrix operator; the smearing is some known matrix with no nice special properties. Okay, it will need some more CPU time, but we just multiply by the inverse, right? Well, sure, if the smearing is very gentle. Unfortunately, in the light bulb spectrometer example, the smearing function is $1\frac{1}{2}$ octaves wide[‡] and very smooth, so trying to get 512 samples over the visible spectrum from measuring at 512 different bulb temperatures is a tiny bit ill-conditioned: even for 10 bands over the visible, the condition number is on the order of 10^{15} . A 300 dB noise gain is not quite what we were looking for.

[†]The original Roman decimation was punishing a military unit for cowardice or dereliction by executing every tenth man. In digital signal processing, division by an integer works much better.

[‡]Why isn't it 2 octaves wide as in Section 2.4.1?

There is a method called *singular value decomposition* (SVD), which is a very general and well-conditioned way to solve linear systems (not necessarily square) by factoring the matrix into a row-orthogonal matrix, a diagonal matrix of singular values, and a column-orthogonal matrix. It can solve nonsingular square systems exactly, though somewhat less efficiently than *LU* decomposition. More importantly, it can also solve overdetermined, singular, or ill-conditioned systems approximately, yielding the minimum squared error solution to any rank we choose; by appropriate choice of the rank, the condition number can be controlled well.

If we are prepared to be less ambitious and abandon our idea of 512 spectral channels in favor of the three human cone pigment curves instead of 512 (see Figure 17.19), SVD will give us a pseudoinverse operator (the Moore–Penrose pseudoinverse) to solve the equation. The condition number is now more like 10^4 to 10^5 , depending on how close a fit we demand, and on how wide a range of black body temperatures we can use. These operators were calculated assuming that our detector cuts off exponentially near 800 nm.

The condition number is then equal to the ratio of the largest singular value to the smallest one; we select the rank of the pseudoinverse by discarding too-small singular values. The resulting pseudoinverse operator is the least-squares optimum for that rank. *Numerical Recipes in C* has some working code and a very good discussion of this technique, which is well worth your time if you have a variable instrument function like this.

Applying the technique is simple, though since it's an N^3 algorithm, it's expensive in CPU cycles: take your instrument function matrix, apply SVD, edit the singular values, calculate the Moore–Penrose operator, and left-multiply it by your desired bandshape. The result will be a matrix that takes your nonuniformly smeared data and produces one filter band output (for more bands, you have to left-multiply by each bandshape in succession).

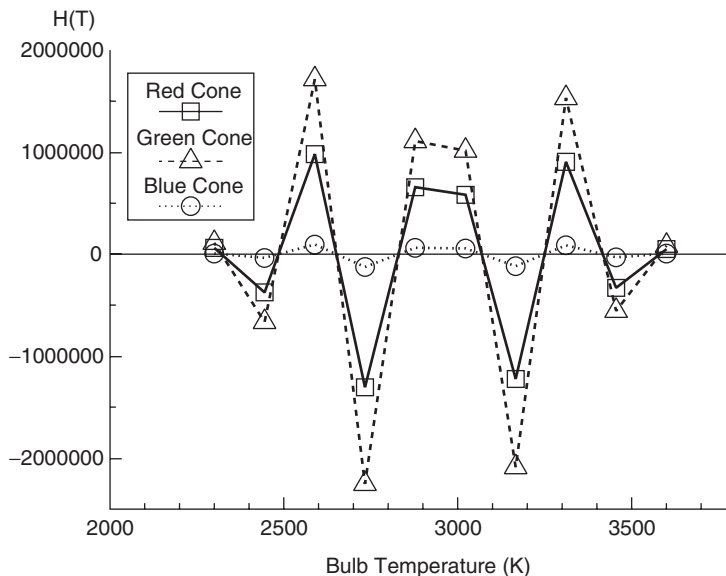


Figure 17.19. Least-squares operator to transform measurements taken with black body source into human cone responses. As you can see from the size of the vertical scale, this isn't too practical.

17.10 FINITE PRECISION EFFECTS

17.10.1 Quantization

The study of the effects of quantization is somewhat subtle, but most of the subtlety concerns doing arithmetic with short word lengths rather than signal quantization per se, so we can usually ignore it if we're using lots of bits, for example, 32 bit integers, or at least single precision floating point to do our arithmetic.

Provided the signal swing is at least several ADU p-p, the quantization noise is well represented as additive Gaussian white noise of rms value $1/\sqrt{12}$ ADU. This is intuitively reasonable, since we expect the LSBs to be less and less correlated between samples as the number of bits goes up. In fact, if our digitizer is good enough, we can get the effect of extra bits by adding analog white noise to smear out the steps and then averaging for a long time to get rid of the noise again. This puts fairly stringent limits on jitter and the DNL of our digitizer. Beware!

If you are using integers, then your adds and subtracts are exact (and hence noiseless), but your multiplies involve rounding the results to the nearest multiple of 2^{-b} , which (assuming the errors are uncorrelated) adds another $2^{-b}/\sqrt{12}$ noise signal after the multiplication. All these noise sources can be carried along to the filter output, and the rms sum of their values taken; this will yield a reasonable estimate of the output noise, unless you have limit cycles and so forth to contend with (see Section 17.6).

17.10.2 Roundoff

Some signal processing algorithms are very vulnerable to roundoff, because round-off errors during the computation can be magnified greatly by subsequent steps. Ill-conditioned matrix operations such as least-squares fitting of polynomials in x^n are one example, and maximum entropy method power spectrum estimation is another. For a given algorithm, the maximum possible amplification of roundoff error relative to the signal is called its condition number, and is a good number to know. It is equal to the ratio of the magnitudes of the largest and smallest magnitude eigenvalues of the operator.

17.10.3 Overflow

The overflow behavior of algorithms depends on whether they are implemented in integer or floating-point arithmetic. Integer overflow is not an error that will get caught; floating-point overflow will cause an exception unless you mask it off in your software. Fixed-point algorithms are more efficient and can use cheaper CPUs, but need close attention.

Integer overflow can lead to large, self-sustaining oscillations in the signal processor—that is, to disaster. Special-purpose processors sometimes use saturating adders for this reason (a saturating adder returns its maximum positive value for any positive-going overflow, and its maximum negative value for any negative-going one). This is a sensible approach that mimics the behavior of good analog networks, but adds a lot of software overhead unless your hardware supports it. (Customized arithmetic is a good application for an FPGA.)

17.11 PULLING DATA OUT OF NOISE

We're always trying to squeeze the most out of our measurements with digital post-processing, which is a great idea. Unfortunately, lots of people neglect the essential precondition, dynamic range reduction in analog (see Chapters 10 and 15). Digitizing prematurely does serious violence to your dynamic range, especially in slow measurements. Most of us have had signals that were perfectly adequate for a measurement but were invisible on a scope—that's your measurement if you digitize too soon.

17.11.1 Shannon's Theorem

It is intuitively reasonable that a given, fixed-bandwidth channel can carry more information if the SNR is high. Even a 1 Hz wide channel can be used to transmit a lot of data if the data are encoded as very small frequency shifts; for example, a frequency counter can measure the frequency of a 1 Hz signal to 8 decimal digits in 1 s, by relying on the stability of the zero crossings in high SNR situations, an equivalent data rate of $8 \log_2(10) \approx 26.6$ bits/s. Remember from Section 13.6.9 that the phase error in radians is

$$\langle \Delta\phi \rangle = \frac{1}{\sqrt{2}} \sqrt{\frac{P_N}{P_C}}, \quad (17.22)$$

so a frequency measurement of 1 cycle at a given SNR (equivalent to CNR in this case) will yield b bits' accuracy, where b is

$$b = \frac{1}{2} \log_2(\text{SNR}) + \log_2(2\pi). \quad (17.23)$$

The time taken for the measurement will depend on how a particular filter settles to b bits' accuracy, but for a given filter shape, it will scale linearly with bandwidth. This is not the optimal measurement situation, of course, since it ignores what happens in between zero crossings, but it shows that at high SNR the channel capacity C (the maximum data rate in bits/s) is proportional to $\text{BW} \cdot \log_2(\text{SNR})$. On the other hand, a signal averager needs N sweeps to improve the SNR by a factor of N , so at very low SNR we expect C to go as the SNR. Shannon's theorem is the mathematical statement that, for any network, the maximum rate of information transfer cannot exceed

$$C = \text{BW} \cdot \log_2(1 + \text{SNR}), \quad (17.24)$$

which has the right asymptotic behavior (Figure 17.20) and also takes into account the time–bandwidth product issues that govern how often we can make independent measurements of a given accuracy. (Note that this is the electrical and not the optical SNR—see Section 3.3). Shannon's theorem sets the upper limit for the amount of information our measurement contains, and so tells us how close we are to the theoretical optimum. This is very valuable, since in the design of a complicated instrument, we may make a blunder somewhere and wind up far from the optimum; Shannon's theorem helps us catch it, just as keeping an eye on the shot noise limit helps us in Chapter 18. If you don't watch the Shannon limit closely, you'll wind up attempting recreational impossibilities on one hand, or turning a silk purse back into a sow's ear on the other.

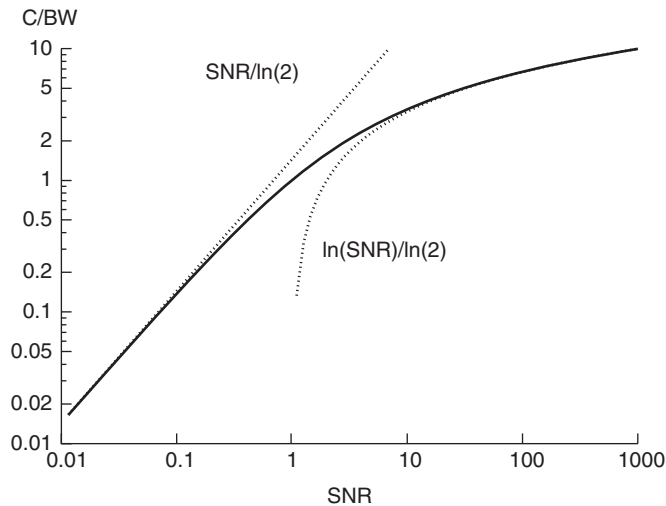


Figure 17.20. Shannon limit.

Aside: Shannon's Theorem and the Bode Limit. In Section 18.5.3, we saw that for a given RC product, there was a trade-off between bandwidth and the logarithm of the matching efficiency, which is analogous to Shannon's theorem for signaling.

17.11.2 Model Dependence

The optimal data recovery technique is one in which the signal energy is all concentrated into one sample when you're done, and the noise is spread out. An example would be a chirped signal in additive noise; resampling it so that the chirp became a pure sinusoid right at one of the sample frequencies of a DFT will do this. Fitting some sine wave to the data and extracting its amplitude, frequency, and phase is another example. Both of these are, of course, nonsense unless the model represents the signal well.

17.11.3 Correlation Techniques

In Section 13.8.10 we encountered the idea of a matched filter, whose transfer function was the complex conjugate of our signal; this undid any phase funnies and resulted in the highest received SNR. In the digital domain, there's a cheat we can use: flip the actual data around in time, and use that as the filter—that is, compute its time autocorrelation. Noise has good autocorrelation properties, but we're not exactly computing the true statistical autocorrelation, since there's no ensemble to average over, so we have to be a bit careful here. *Uncorrelated* noise sources have zero ensemble-averaged cross-correlations, but because of finite bandwidth, that is not true of the time autocorrelations of samples of noise.

With sufficiently small noise, this would be close to the matched filter and would therefore provide near-optimal response. It gets worse fairly fast at low SNR, of course, since the cross-correlation of the noise and the data is unlikely to be 0, and the cross-correlation will shrink more slowly with signal amplitude than the desired signal's autocorrelation.

On the other hand, it is a very good way of pulling out a narrowband signal in wide-band noise; for example, in a backscatter interferometer where the Doppler frequency is unknown in advance, you can find tone bursts in white Gaussian noise. On the other hand, if your data set contains lots of nonrandom background signals, or your noise has structure of its own, autocorrelation methods will generate artifacts all over the place.

Cross-correlations between what you know the signal ought to look like and the actual data can be useful in estimating the delay of your signal; this is often used for range measurements in radar. If you don't have the luxury of designing your own signal shape (as radar people generally do), then when you compute the cross-correlation, there will usually be more than one peak—especially with short data runs. Reliably finding the right peak is nontrivial in general, because of the large discontinuities that occur when one peak grows up to pass its neighbor. Usually you're really interested in the envelope of the cross-correlation, which means you have to fit a function to the peaks, or else try Hilbert transforming, which is hard with short data runs. (You can design a Hilbert transformer using windows, as we did with lowpass filters.)

17.11.4 Numerical Experiments

It is not always trivial to know what the condition number of your algorithm is. We've managed to preserve our continuous-time ideas of bandwidth and SNR through all our Fourier-type processes; DFTs and digital filters have well-defined noise gains (Section 13.1), so that's simple enough. However, once you start cascading signal processing components and fancy digital algorithms (e.g., SVD), things become less clear very rapidly. For example, an analog filter with a big peak in it at a frequency that doesn't occur in the data will in practice boost the total noise much less than you might expect from its noise gain. This can be estimated analytically, and simulated afterwards, but both of these approaches can hide subtle blunders. A combination of physical and numerical experiments is especially necessary in these situations.

17.11.5 Signal Averaging

Signal averaging was encountered in Chapter 13 as a good way to narrow the measurement bandwidth without concentrating all the signal power down in the $1/f$ noise region. Now that we're a bit more sophisticated about digital processing, we can investigate what the actual passband of a signal averager looks like.

Suppose we want to average K scans of N points each, and imagine that we have all NK data points in a big array like a TV raster, as shown in Figure 17.21. At each scan point x_n , the raw data are a time series, $y_n^k(t_k)$, where $t_n^k = t_n^0 + kP$ and P is the scan period; in other words, y_n^k is the n th data value from the k th scan.

Now let's look at the n th bin, as a function of k , from a Fourier analysis perspective, with an eye to keeping it uncorrupted by strong spurious signals (e.g., harmonics of 60 Hz). The n th bin is sampled with period P , which of course is horribly undersampled; however, because the true measurement data are periodic with period P , all the desired components of bin n are aliased right down to DC (which is just what we want, since we're about to average them). The filtering operation is a rectangular-windowed average, whose frequency response is a sinc function,

$$|H(f)| = \text{sinc}(fT), \quad (17.25)$$

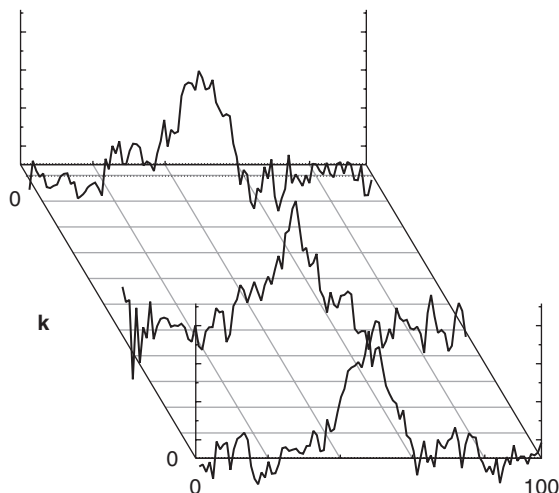


Figure 17.21. Partial tableau of a signal averaging scheme.

which is replicated at every harmonic of $f_{\text{scan}} = 1/P$. This is a lousy window, as we saw, so for better spur rejection we can use data windowing.

This is a different application of windowing than in spectrum analysis or FIR filter design. We want to multiply the points y_n^k by a window function W —but now $W(k)$ the scan line index, *not* $W(n)$, the index of the point within the scan line. We're weighting the average of the scan lines. Figure 17.22 shows one of the passbands near 60 Hz of 10 averages, 100 points/scan, $\nu = 1061$ Hz, and $1/P = 10.61$ Hz (i.e., no retrace delay);

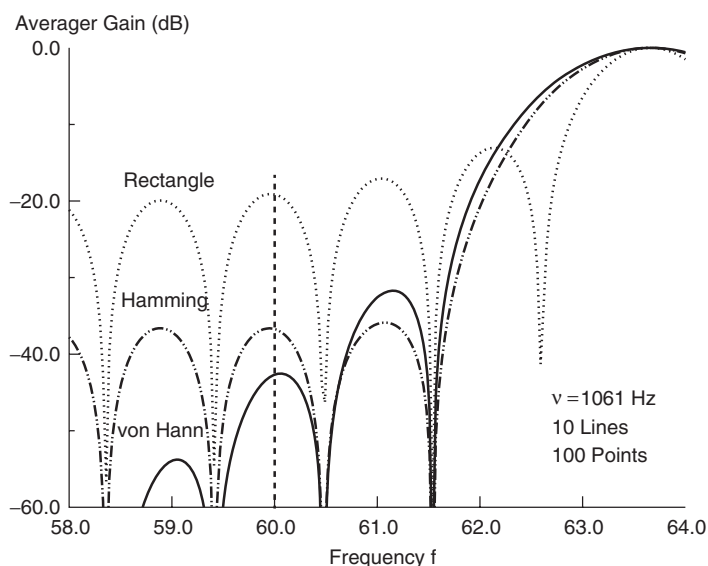


Figure 17.22. Detail of the frequency response of a signal averager, with spur rejection improved by *transverse* data windowing.

note the improvement in the sidelobes due to windowing in k , at the price of a wider main lobe, that is, poorer wideband noise rejection.[†] Another way of looking at this is that windowing in n will force the averaged data to go to 0 at the ends of the scan, and so improve the resolution of spectral estimates of the signal and residual noise and aliased interference, which isn't what we want. On the other hand, windowing in k will affect each averaged point the same way, and will improve the *rejection* of isolated spurious signals such as hum.

17.11.6 Two-Point Correlation

If you're trying to measure an aperiodic signal (e.g., noise), so that signal averaging isn't applicable, it isn't easy to separate out the instrument contribution. If the statistics of the signal are stationary, the signal magnitude can be measured using *two-point correlation* (see Section 15.3). In this trick, we measure the signal using two or more voltmeters simultaneously. The instrument noise is assumed to be uncorrelated. With M voltmeters in parallel measuring a true signal $v(t)$, the measurements that result are

$$v_j(t) = v(t) + v_{Nj}(t) + Z \cdot \sum_{j=1}^M i_{Nj}(t), \quad (17.26)$$

where $i_{Nj}(t)$ and $v_{Nj}(t)$ are the instantaneous noise current and voltage of the j th voltmeter. The current noises add because they're wired in parallel. The signal power can be obtained as the time average of $v_j(t)^2$, but that's polluted by voltage and current noise. However, if we compute the time average of $v_j \cdot v_k$, the cross terms in v_N drop out, and only the current noise is left. The current noise can be reduced to very low levels with MOSFET buffers, and so signal statistics can be measured accurately with the correlation technique even if they're below the level of the technical noise.[‡] This is a good way to measure the absolute noise of a low value resistor, for example. We can do better still by using more than two meters; the number of independent measurements is $M(M-1)/2$, so for large M our rms noise voltage goes down like $M/\sqrt{2}$, which is faster than the usual square-root dependence of averaging. (In Section 10.10 we discussed a generalization of this idea, called *closure*.)

17.12 PHASE RECOVERY TECHNIQUES

17.12.1 Unwrapping

Phase data are nearly always obtained *modulo* 2π , unless some tracking device was employed, or other information is available, for example, two phase detectors, one with frequency dividers ahead of it so that its range is not $\pm\pi/2$ but $\pm N\pi/2$. Such phase data are said to have been wrapped into the fundamental range, usually $[-\pi, \pi)$, and must be unwrapped to obtain the true phase data.[§]

[†]The Hamming window is unimpressive here because there aren't enough data samples. It's much better with 100 lines than with 10.

[‡]This obviously fails to get rid of the DC contributions, which are precisely those that survive time averaging.

[§]Some instruments use $[0, 2\pi)$, which makes life interesting when (as often) we want to servo around 0—it wraps continually, producing an average value of π , which is 180° from the right answer.

Unwrapping 1D phase data is usually easy; a phase wrap has occurred whenever two adjacent points differ by more than $3\pi/2$ radians (or π radians if you can be sure there's no zero in between, which will produce a sudden π phase shift but is not a wrap). The only times you get into big trouble are when the data are undersampled, when a tracking phase detector has inadequate bandwidth and so smears the wrap out, or when the data contains big jumps that are real, for example, a phase shifting interferometer looking at a sample with steep slopes (where the data drop out) that may be several wavelengths tall.

The 2D case is considerably harder and is usually handled by minimizing the L2 norm of the slope of the phase, via an iterative algorithm, or by a Green's function approach.[†]

Some kinds of phase data, such as synthetic-aperture radar and full-field phase shifting interferometry, often lead to phase maps with inconsistencies; these can be handled with the method of residues.[‡] This task will burn lots of cycles but will give you the best results for noisy or inconsistent data, as you might get from an undersampled data set, or one that was changing with time as the data were taken (as it often will be).

If the data are of good quality, you can do it by numerical crystallization, which is like tiling a floor: you start from somewhere in the interior, and add points to the periphery of your unwrapped data set by forcing the phase jumps at the edges to be less than $3\pi/2$ radians. You can often find inconsistencies this way, especially at boundaries where no data are available.

If you have an undersampled measurement, for example, a phase shifting interferometer using a CCD with a very small fill factor and a pixel pitch $\Delta > 0.5\lambda/\text{NA}$, you can still unwrap the phase successfully in the presence of multiples of 2π shift per pixel, if you know a priori that the wavefront phase is smooth. In that case, instead of keeping $|\partial\phi/\partial x| < 2\pi/\Delta$, as we did before, we require that

$$\left| \frac{\partial^2 \phi}{\partial x^2} \right| < \frac{2\pi}{\Delta^2}. \quad (17.27)$$

17.12.2 Phase Shifting Measurements

If you have measurements of the quadrature amplitudes I and Q , you can turn them into amplitude and phase easily; most compilers provide canned functions for this. When I and Q have been corrupted by noise, spurs, and imperfect phase shifting between them, life gets more difficult; the arctangent is a pretty well-behaved function, but its error sensitivity is not 0. Many algorithms have been proposed for getting A and ϕ from N measurements spaced around the unit circle, each with different sensitivity to noise, harmonics, and errors in the phase step size, of which the best known are the Carré and Hariharan algorithms. If you're doing phase shifting interferometry, your algorithm needs some attention. Which is best depends very much on your situation, and on what the dominant source of noise and errors.

If your system has significant distortion of the sinusoidal phase dependence of the signal, the situation becomes much more difficult, as this distortion generally becomes worse as the signal level increases. This is the case, for example, when using RF mixers as phase detectors over a wide range of phases; all the harmonics and spurious mixing

[†]Gianfranco Fornaro, Giorgio Franceschetti, Riccardo Lanari, and Eugenio Sansosti, Robust phase unwrapping techniques: a comparison. *J. Opt. Soc. Am. A* **13**(12), 2355–2366 (1996)

[‡]Dennis C. Ghiglia and Mark D. Pritt, *Two Dimensional Phase Unwrapping*. Wiley, Hoboken, NJ, 1998.

products that would normally be filtered out are now right on top of your signal, since any harmonic of DC is still DC.

Be suspicious of optimal algorithms, because one of two things will be true: either the optimum is not sharp, in which case optimality doesn't buy you that much; or it is sharp, in which case minor failures to observe the assumptions on which it was derived will have large consequences. A big calibration table is probably better than any of these algorithms, if you have the time and space.[†]

17.12.3 Fienup's Algorithm

We're right at the edge of our scope, but there is another class of phase recovery algorithms that's worth mentioning. They use a priori information about an (intensity) image to recover its corresponding phase image. There are several algorithms for this, of which the best known is the Fienup phase retrieval algorithm.[‡] They all work by iteratively applying constraints in real space and \mathbf{k} -space; they're all a bit flaky, but when they work, they're amazing.

[†]There are good discussions of these algorithms in Daniel Malacara-Doblado et al., *Optical Eng.* **36**(7), 2086–2091 (1997); and Yves Sirel, *Appl. Optics* **36**(1), 271–276 (January 1, 1997).

[‡]J. R. Fienup, *Appl. Optics* **21**(15), 2758–2769 (1982).