for specific situations, and arm themselves with a variety of other tricks. We suggest that you do likewise, as your projects demand.

CITED REFERENCES AND FURTHER READING:

Hamming, R.W. 1983, *Digital Filters*, 2nd ed. (Englewood Cliffs, NJ: Prentice-Hall).

Antoniou, A. 1979, *Digital Filters: Analysis and Design* (New York: McGraw-Hill).

Parks, T.W., and Burrus, C.S. 1987, *Digital Filter Design* (New York: Wiley).

Oppenheim, A.V., and Schafer, R.W. 1989, *Discrete-Time Signal Processing* (Englewood Cliffs, NJ: Prentice-Hall).

Rice, J.R. 1964, *The Approximation of Functions* (Reading, MA: Addison-Wesley); also 1969, *op. cit.*, Vol. 2.

Rabiner, L.R., and Gold, B. 1975, *Theory and Application of Digital Signal Processing* (Englewood Cliffs, NJ: Prentice-Hall).

13.6 Linear Prediction and Linear Predictive Coding

We begin with a very general formulation that will allow us to make connections to various special cases. Let $\{y'_\alpha\}$ be a set of measured values for some underlying set of true values of a quantity y, denoted $\{y_\alpha\}$, related to these true values by the addition of random noise,

$$y'_\alpha = y_\alpha + n_\alpha \tag{13.6.1}$$

(compare equation 13.3.2, with a somewhat different notation). Our use of a Greek subscript to index the members of the set is meant to indicate that the data points are not necessarily equally spaced along a line, or even ordered: they might be "random" points in three-dimensional space, for example. Now, suppose we want to construct the "best" estimate of the true value of some particular point $y_\star$ as a linear combination of the known, noisy, values. Writing

$$y_\star = \sum_\alpha d_{\star\alpha} y'_\alpha + x_\star \tag{13.6.2}$$

we want to find coefficients $d_{\star\alpha}$ that minimize, in some way, the *discrepancy* $x_\star$. The coefficients $d_{\star\alpha}$ have a "star" subscript to indicate that they depend on the choice of point $y_\star$. Later, we might want to let $y_\star$ be one of the existing y_α's. In that case, our problem becomes one of optimal filtering or estimation, closely related to the discussion in §13.3. On the other hand, we might want $y_\star$ to be a completely new point. In that case, our problem will be one of *linear prediction*.

A natural way to minimize the discrepancy $x_\star$ is in the statistical mean square sense. If angle brackets denote statistical averages, then we seek $d_{\star\alpha}$'s that minimize

$$\begin{aligned}\left\langle x_\star^2\right\rangle &= \left\langle \left[\sum_\alpha d_{\star\alpha}(y_\alpha + n_\alpha) - y_\star\right]^2 \right\rangle \\ &= \sum_{\alpha\beta}(\langle y_\alpha y_\beta\rangle + \langle n_\alpha n_\beta\rangle) d_{\star\alpha} d_{\star\beta} - 2\sum_\alpha \langle y_\star y_\alpha\rangle\, d_{\star\alpha} + \left\langle y_\star^2\right\rangle\end{aligned} \tag{13.6.3}$$

Here we have used the fact that noise is uncorrelated with signal, e.g., $\langle n_\alpha y_\beta \rangle = 0$. The quantities $\langle y_\alpha y_\beta \rangle$ and $\langle y_\star y_\alpha \rangle$ describe the autocorrelation structure of the underlying data. We have already seen an analogous expression, (13.2.2), for the case of equally spaced data points on a line; we will meet correlation several times again in its statistical sense in Chapters 14 and 15. The quantities $\langle n_\alpha n_\beta \rangle$ describe the autocorrelation properties of the noise. Often, for point-to-point uncorrelated noise, we have $\langle n_\alpha n_\beta \rangle = \langle n_\alpha^2 \rangle \delta_{\alpha\beta}$. It is convenient to think of the various correlation quantities as comprising matrices and vectors,

$$\phi_{\alpha\beta} \equiv \langle y_\alpha y_\beta \rangle \qquad \phi_{\star\alpha} \equiv \langle y_\star y_\alpha \rangle \qquad \eta_{\alpha\beta} \equiv \langle n_\alpha n_\beta \rangle \text{ or } \langle n_\alpha^2 \rangle \delta_{\alpha\beta} \tag{13.6.4}$$

Setting the derivative of equation (13.6.3) with respect to the $d_{\star\alpha}$'s equal to zero, one readily obtains the set of linear equations,

$$\sum_\beta \left[\phi_{\alpha\beta} + \eta_{\alpha\beta}\right] d_{\star\beta} = \phi_{\star\alpha} \tag{13.6.5}$$

If we write the solution as a matrix inverse, then the estimation equation (13.6.2) becomes, omitting the minimized discrepancy $x_\star$,

$$y_\star \approx \sum_{\alpha\beta} \phi_{\star\alpha} \left[\phi_{\mu\nu} + \eta_{\mu\nu}\right]^{-1}_{\alpha\beta} y'_\beta \tag{13.6.6}$$

From equations (13.6.3) and (13.6.5) one can also calculate the expected mean square value of the discrepancy at its minimum, denoted $\langle x_\star^2 \rangle_0$,

$$\langle x_\star^2 \rangle_0 = \langle y_\star^2 \rangle - \sum_\beta d_{\star\beta}\phi_{\star\beta} = \langle y_\star^2 \rangle - \sum_{\alpha\beta} \phi_{\star\alpha} \left[\phi_{\mu\nu} + \eta_{\mu\nu}\right]^{-1}_{\alpha\beta} \phi_{\star\beta} \tag{13.6.7}$$

A final general result tells how much the mean square discrepancy $\langle x_\star^2 \rangle$ is increased if we use the estimation equation (13.6.2) not with the best values $d_{\star\beta}$, but with some other values $\widehat{d}_{\star\beta}$. The above equations then imply

$$\langle x_\star^2 \rangle = \langle x_\star^2 \rangle_0 + \sum_{\alpha\beta} (\widehat{d}_{\star\alpha} - d_{\star\alpha}) \left[\phi_{\alpha\beta} + \eta_{\alpha\beta}\right] (\widehat{d}_{\star\beta} - d_{\star\beta}) \tag{13.6.8}$$

Since the second term is a pure quadratic form, we see that the increase in the discrepancy is only second order in any error made in estimating the $d_{\star\beta}$'s.

Connection to Optimal Filtering

If we change "star" to a Greek index, say γ, then the above formulas describe optimal filtering, generalizing the discussion of §13.3. One sees, for example, that if the noise amplitudes n_α go to zero, so likewise do the noise autocorrelations $\eta_{\alpha\beta}$, and, canceling a matrix times its inverse, equation (13.6.6) simply becomes $y_\gamma = y'_\gamma$. Another special case occurs if the matrices $\phi_{\alpha\beta}$ and $\eta_{\alpha\beta}$ are diagonal. In that case, equation (13.6.6) becomes

$$y_\gamma = \frac{\phi_{\gamma\gamma}}{\phi_{\gamma\gamma} + \eta_{\gamma\gamma}} y'_\gamma \tag{13.6.9}$$

which is readily recognizable as equation (13.3.6) with $S^2 \rightarrow \phi_{\gamma\gamma}$, $N^2 \rightarrow \eta_{\gamma\gamma}$. What is going on is this: For the case of equally spaced data points, and in the Fourier domain, autocorrelations become simply squares of Fourier amplitudes (Wiener-Khinchin theorem, equation 12.0.12), and the optimal filter can be constructed algebraically, as equation (13.6.9), without inverting any matrix.

More generally, in the time domain, or any other domain, an optimal filter (one that minimizes the square of the discrepancy from the underlying true value in the presence of measurement noise) can be constructed by estimating the autocorrelation matrices $\phi_{\alpha\beta}$ and $\eta_{\alpha\beta}$, and applying equation (13.6.6) with $\star \rightarrow \gamma$. (Equation 13.6.8 is in fact the basis for the §13.3's statement that even crude optimal filtering can be quite effective.)

Linear Prediction

Classical *linear prediction* specializes to the case where the data points y_β are equally spaced along a line, y_i, $i = 1, 2, \ldots, N$, and we want to use M consecutive values of y_i to predict an $M + 1$st. Stationarity is assumed. That is, the autocorrelation $\langle y_j y_k \rangle$ is assumed to depend only on the difference $|j - k|$, and not on j or k individually, so that the autocorrelation ϕ has only a single index,

$$\phi_j \equiv \langle y_i y_{i+j} \rangle \approx \frac{1}{N-j} \sum_{i=1}^{N-j} y_i y_{i+j} \tag{13.6.10}$$

Here, the approximate equality shows one way to use the actual data set values to estimate the autocorrelation components. (In fact, there is a better way to make these estimates; see below.) In the situation described, the estimation equation (13.6.2) is

$$y_n = \sum_{j=1}^{M} d_j y_{n-j} + x_n \tag{13.6.11}$$

(compare equation 13.5.1) and equation (13.6.5) becomes the set of M equations for the M unknown d_j's, now called the *linear prediction (LP) coefficients*,

$$\sum_{j=1}^{M} \phi_{|j-k|} d_j = \phi_k \qquad (k = 1, \ldots, M) \tag{13.6.12}$$

Notice that while noise is not explicitly included in the equations, it is properly accounted for, *if* it is point-to-point uncorrelated: ϕ_0, as estimated by equation (13.6.10) using *measured* values y'_i, actually estimates the diagonal part of $\phi_{\alpha\alpha} + \eta_{\alpha\alpha}$, above. The mean square discrepancy $\langle x_n^2 \rangle$ is estimated by equation (13.6.7) as

$$\langle x_n^2 \rangle = \phi_0 - \phi_1 d_1 - \phi_2 d_2 - \cdots - \phi_M d_M \tag{13.6.13}$$

To use linear prediction, we first compute the d_j's, using equations (13.6.10) and (13.6.12). We then calculate equation (13.6.13) or, more concretely, apply (13.6.11) to the known record to get an idea of how large are the discrepancies x_i. If the discrepancies are small, then we can continue applying (13.6.11) right on into the future, imagining the unknown "future" discrepancies x_i to be zero. In this application, (13.6.11) is a kind of extrapolation formula. In many situations, this extrapolation turns out to be vastly more powerful than any kind of simple polynomial extrapolation. (By the way, you should not confuse the terms "linear prediction" and "linear extrapolation"; the general functional form used by linear prediction is *much* more complex than a straight line, or even a low-order polynomial!)

However, to achieve its full usefulness, linear prediction must be constrained in one additional respect: One must take additional measures to guarantee its *stability*. Equation (13.6.11) is a special case of the general linear filter (13.5.1). The condition that (13.6.11) be stable as a linear predictor is precisely that given in equations (13.5.5) and (13.5.6), namely that the characteristic polynomial

$$z^N - \sum_{j=1}^{N} d_j z^{N-j} = 0 \qquad (13.6.14)$$

have all N of its roots inside the unit circle,

$$|z| \leq 1 \qquad (13.6.15)$$

There is no guarantee that the coefficients produced by equation (13.6.12) will have this property. If the data contain many oscillations without any particular trend towards increasing or decreasing amplitude, then the complex roots of (13.6.14) will generally all be rather close to the unit circle. The finite length of the data set will cause some of these roots to be inside the unit circle, others outside. In some applications, where the resulting instabilities are slowly growing and the linear prediction is not pushed too far, it is best to use the "unmassaged" LP coefficients that come directly out of (13.6.12). For example, one might be extrapolating to fill a short gap in a data set; then one might extrapolate both forwards across the gap and backwards from the data beyond the gap. If the two extrapolations agree tolerably well, then instability is not a problem.

When instability *is* a problem, you have to "massage" the LP coefficients. You do this by (i) solving (numerically) equation (13.6.14) for its N complex roots; (ii) moving the roots to where you think they ought to be inside or on the unit circle; (iii) reconstituting the now-modified LP coefficients. You may think that step (ii) sounds a little vague. It is. There is no "best" procedure. If you think that your signal is truly a sum of undamped sine and cosine waves (perhaps with incommensurate periods), then you will want simply to move each root z_i onto the unit circle,

$$z_i \;\rightarrow\; z_i / |z_i| \qquad (13.6.16)$$

In other circumstances it may seem appropriate to reflect a bad root across the unit circle

$$z_i \;\rightarrow\; 1/z_i{}^* \qquad (13.6.17)$$

This alternative has the property that it preserves the amplitude of the output of (13.6.11) when it is driven by a sinusoidal set of x_i's. It assumes that (13.6.12) has correctly identified the spectral width of a resonance, but only slipped up on identifying its time sense so that signals that should be damped as time proceeds end up growing in amplitude. The choice between (13.6.16) and (13.6.17) sometimes might as well be based on voodoo. We prefer (13.6.17).

Also magical is the choice of M, the number of LP coefficients to use. You should choose M to be as small as works for you, that is, you should choose it by experimenting with your data. Try $M = 5, 10, 20, 40$. If you need larger M's than this, be aware that the procedure of "massaging" all those complex roots is quite sensitive to roundoff error. Use double precision.

Linear prediction is especially successful at extrapolating signals that are smooth and oscillatory, though not necessarily periodic. In such cases, linear prediction often extrapolates accurately through *many cycles* of the signal. By contrast, polynomial extrapolation in general becomes seriously inaccurate after at most a cycle or two. A prototypical example of a signal that can successfully be linearly predicted is the height of ocean tides, for which the fundamental 12-hour period is modulated in phase and amplitude over the course of the month and year, and for which local hydrodynamic effects may make even one cycle of the curve look rather different in shape from a sine wave.

We already remarked that equation (13.6.10) is not necessarily the best way to estimate the covariances ϕ_k from the data set. In fact, results obtained from linear prediction are remarkably sensitive to exactly how the ϕ_k's are estimated. One particularly good method is due to Burg [1], and involves a recursive procedure for increasing the order M by one unit at a time, at each stage re-estimating the coefficients d_j, $j = 1, \ldots, M$ so as to minimize the residual in equation (13.6.13). Although further discussion of the Burg method is beyond our scope here, the method is implemented in the following routine [1,2] for estimating the LP coefficients d_j of a data set.

```
#include <math.h>
#include "nrutil.h"

void memcof(float data[], int n, int m, float *xms, float d[])
```
Given a real vector of `data[1..n]`, and given `m`, this routine returns `m` linear prediction coefficients as `d[1..m]`, and returns the mean square discrepancy as `xms`.
```
{
    int k,j,i;
    float p=0.0,*wk1,*wk2,*wkm;

    wk1=vector(1,n);
    wk2=vector(1,n);
    wkm=vector(1,m);
    for (j=1;j<=n;j++) p += SQR(data[j]);
    *xms=p/n;
    wk1[1]=data[1];
    wk2[n-1]=data[n];
    for (j=2;j<=n-1;j++) {
        wk1[j]=data[j];
        wk2[j-1]=data[j];
    }
    for (k=1;k<=m;k++) {
        float num=0.0,denom=0.0;
        for (j=1;j<=(n-k);j++) {
            num += wk1[j]*wk2[j];
            denom += SQR(wk1[j])+SQR(wk2[j]);
        }
        d[k]=2.0*num/denom;
```

```
        *xms *= (1.0-SQR(d[k]));
        for (i=1;i<=(k-1);i++)
            d[i]=wkm[i]-d[k]*wkm[k-i];
            The algorithm is recursive, building up the answer for larger and larger values of m
            until the desired value is reached. At this point in the algorithm, one could return
            the vector d and scalar xms for a set of LP coefficients with k (rather than m)
            terms.
        if (k == m) {
            free_vector(wkm,1,m);
            free_vector(wk2,1,n);
            free_vector(wk1,1,n);
            return;
        }
        for (i=1;i<=k;i++) wkm[i]=d[i];
        for (j=1;j<=(n-k-1);j++) {
            wk1[j] -= wkm[k]*wk2[j];
            wk2[j]=wk2[j+1]-wkm[k]*wk1[j+1];
        }
    }
    nrerror("never get here in memcof.");
}
```

Here are procedures for rendering the LP coefficients stable (if you choose to do so), and for extrapolating a data set by linear prediction, using the original or massaged LP coefficients. The routine `zroots` (§9.5) is used to find all complex roots of a polynomial.

```
#include <math.h>
#include "complex.h"
#define NMAX 100                          Largest expected value of m.
#define ZERO Complex(0.0,0.0)
#define ONE Complex(1.0,0.0)

void fixrts(float d[], int m)
Given the LP coefficients d[1..m], this routine finds all roots of the characteristic polynomial
(13.6.14), reflects any roots that are outside the unit circle back inside, and then returns a
modified set of coefficients d[1..m].
{
    void zroots(fcomplex a[], int m, fcomplex roots[], int polish);
    int i,j,polish;
    fcomplex a[NMAX],roots[NMAX];

    a[m]=ONE;
    for (j=m-1;j>=0;j--)                  Set up complex coefficients for polynomial root
        a[j]=Complex(-d[m-j],0.0);            finder.
    polish=1;
    zroots(a,m,roots,polish);             Find all the roots.
    for (j=1;j<=m;j++)                    Look for a...
        if (Cabs(roots[j]) > 1.0)         root outside the unit circle,
            roots[j]=Cdiv(ONE,Conjg(roots[j]));   and reflect it back inside.
    a[0]=Csub(ZERO,roots[1]);             Now reconstruct the polynomial coefficients,
    a[1]=ONE;
    for (j=2;j<=m;j++) {                  by looping over the roots
        a[j]=ONE;
        for (i=j;i>=2;i--)                and synthetically multiplying.
            a[i-1]=Csub(a[i-2],Cmul(roots[j],a[i-1]));
        a[0]=Csub(ZERO,Cmul(roots[j],a[0]));
    }
    for (j=0;j<=m-1;j++)                  The polynomial coefficients are guaranteed to be
        d[m-j] = -a[j].r;                     real, so we need only return the real part as
}                                             new LP coefficients.
```

```
#include "nrutil.h"

void predic(float data[], int ndata, float d[], int m, float future[],
    int nfut)
```
Given `data[1..ndata]`, and given the data's LP coefficients `d[1..m]`, this routine applies equation (13.6.11) to predict the next `nfut` data points, which it returns in the array `future[1..nfut]`. Note that the routine references only the last `m` values of `data`, as initial values for the prediction.
```
{
    int k,j;
    float sum,discrp,*reg;

    reg=vector(1,m);
    for (j=1;j<=m;j++) reg[j]=data[ndata+1-j];
    for (j=1;j<=nfut;j++) {
        discrp=0.0;
```
This is where you would put in a known discrepancy if you were reconstructing a function by linear predictive coding rather than extrapolating a function by linear prediction. See text.
```
        sum=discrp;
        for (k=1;k<=m;k++) sum += d[k]*reg[k];
        for (k=m;k>=2;k--) reg[k]=reg[k-1];
        future[j]=reg[1]=sum;
    }
    free_vector(reg,1,m);
}
```
[If you want to implement circular arrays, you can avoid this shifting of coefficients.]

Removing the Bias in Linear Prediction

You might expect that the sum of the d_j's in equation (13.6.11) (or, more generally, in equation 13.6.2) should be 1, so that (e.g.) adding a constant to all the data points y_i yields a prediction that is increased by the same constant. However, the d_j's do not sum to 1 but, in general, to a value slightly less than one. This fact reveals a subtle point, that the estimator of classical linear prediction is not *unbiased*, even though it does minimize the mean square discrepancy. At any place where the measured autocorrelation does not imply a better estimate, the equations of linear prediction tend to predict a value that tends towards zero.

Sometimes, that is just what you want. If the process that generates the y_i's in fact has zero mean, then zero is the best guess absent other information. At other times, however, this behavior is unwarranted. If you have data that show only small variations around a positive value, you don't want linear predictions that droop towards zero.

Often it is a workable approximation to subtract the mean off your data set, perform the linear prediction, and then add the mean back. This procedure contains the germ of the correct solution; but the simple arithmetic mean is not quite the correct constant to subtract. In fact, an unbiased estimator is obtained by subtracting from every data point an autocorrelation-weighted mean defined by [3,4]

$$\overline{y} \equiv \sum_{\beta} [\phi_{\mu\nu} + \eta_{\mu\nu}]^{-1}_{\alpha\beta}\, y_\beta \Big/ \sum_{\alpha\beta} [\phi_{\mu\nu} + \eta_{\mu\nu}]^{-1}_{\alpha\beta} \tag{13.6.18}$$

With this subtraction, the sum of the LP coefficients should be unity, up to roundoff and differences in how the ϕ_k's are estimated.

Linear Predictive Coding (LPC)

A different, though related, method to which the formalism above can be applied is the "compression" of a sampled signal so that it can be stored more compactly. The original form should be *exactly* recoverable from the compressed version. Obviously, compression can be accomplished only if there is redundancy in the signal. Equation (13.6.11) describes one kind of redundancy: It says that the signal, except for a small discrepancy, is predictable from its previous values and from a small number of LP coefficients. Compression of a signal by the use of (13.6.11) is thus called *linear predictive coding*, or *LPC*.

The basic idea of LPC (in its simplest form) is to record as a compressed file (i) the number of LP coefficients M, (ii) their M values, e.g., as obtained by `memcof`, (iii) the first M data points, and then (iv) for each subsequent data point only its residual discrepancy x_i (equation 13.6.1). When you are creating the compressed file, you find the residual by applying (13.6.1) to the previous M points, subtracting the sum from the actual value of the current point. When you are reconstructing the original file, you add the residual back in, at the point indicated in the routine `predic`.

It may not be obvious why there is any compression at all in this scheme. After all, we are storing one value of residual per data point! Why not just store the original data point? The answer depends on the relative sizes of the numbers involved. The residual is obtained by subtracting two very nearly equal numbers (the data and the linear prediction). Therefore, the discrepancy typically has only a very small number of nonzero bits. These can be stored in a compressed file. How do you do it in a high-level language? Here is one way: Scale your data to have integer values, say between $+1000000$ and -1000000 (supposing that you need six significant figures). Modify equation (13.6.1) by enclosing the sum term in an "integer part of" operator. The discrepancy will now, by definition, be an integer. Experiment with different values of M, to find LP coefficients that make the range of the discrepancy as small as you can. If you can get to within a range of ± 127 (and in our experience this is not at all difficult) then you can write it to a file as a single byte. This is a compression factor of 4, compared to 4-byte integer or floating formats.

Notice that the LP coefficients are computed using the *quantized* data, and that the discrepancy is also quantized, i.e., quantization is done both outside and inside the LPC loop. If you are careful in following this prescription, then, apart from the initial quantization of the data, you will not introduce even a single bit of roundoff error into the compression-reconstruction process: While the evaluation of the sum in (13.6.11) may have roundoff errors, the residual that you store is the value which, when added back to the sum, gives *exactly* the original (quantized) data value. Notice also that you do not need to massage the LP coefficients for stability; by adding the residual back in to each point, you never depart from the original data, so instabilities cannot grow. There is therefore no need for `fixrts`, above.

Look at §20.4 to learn about *Huffman coding*, which will further compress the residuals by taking advantage of the fact that smaller values of discrepancy will occur more often than larger values. A very primitive version of Huffman coding would be this: If most of the discrepancies are in the range ± 127, but an occasional one is outside, then reserve the value 127 to mean "out of range," and then record on the file (immediately following the 127) a full-word value of the out-of-range discrepancy. §20.4 explains how to do much better.

There are many variant procedures that all fall under the rubric of LPC.

- If the spectral character of the data is time-variable, then it is best not to use a single set of LP coefficients for the whole data set, but rather to partition the data into segments, computing and storing different LP coefficients for each segment.
- If the data are really well characterized by their LP coefficients, and you can tolerate some small amount of error, then don't bother storing all of the residuals. Just do linear prediction until you are outside of tolerances, then reinitialize (using M sequential stored residuals) and continue predicting.
- In some applications, most notably speech synthesis, one cares only about the spectral content of the reconstructed signal, not the relative phases. In this case, one need not store any starting values at all, but only the LP coefficients for each segment of the data. The output is reconstructed by driving these coefficients with initial conditions consisting of all zeros except for one nonzero spike. A speech synthesizer chip may have of order 10 LP coefficients, which change perhaps 20 to 50 times per second.
- Some people believe that it is interesting to analyze a signal by LPC, even when the residuals x_i are *not* small. The x_i's are then interpreted as the underlying "input signal" which, when filtered through the all-poles filter defined by the LP coefficients (see §13.7), produces the observed "output signal." LPC reveals simultaneously, it is said, the nature of the filter *and* the particular input that is driving it. We are skeptical of these applications; the literature, however, is full of extravagant claims.

CITED REFERENCES AND FURTHER READING:

Childers, D.G. (ed.) 1978, *Modern Spectrum Analysis* (New York: IEEE Press), especially the paper by J. Makhoul (reprinted from *Proceedings of the IEEE*, vol. 63, p. 561, 1975).

Burg, J.P. 1968, reprinted in Childers, 1978. [1]

Anderson, N. 1974, reprinted in Childers, 1978. [2]

Cressie, N. 1991, in *Spatial Statistics and Digital Image Analysis* (Washington: National Academy Press). [3]

Press, W.H., and Rybicki, G.B. 1992, *Astrophysical Journal*, vol. 398, pp. 169–176. [4]

13.7 Power Spectrum Estimation by the Maximum Entropy (All Poles) Method

The FFT is not the only way to estimate the power spectrum of a process, nor is it necessarily the best way for all purposes. To see how one might devise another method, let us enlarge our view for a moment, so that it includes not only real frequencies in the Nyquist interval $-f_c < f < f_c$, but also the entire complex frequency plane. From that vantage point, let us transform the complex f-plane to a new plane, called the *z-transform plane* or *z-plane*, by the relation

$$z \equiv e^{2\pi i f \Delta} \tag{13.7.1}$$

where Δ is, as usual, the sampling interval in the time domain. Notice that the Nyquist interval on the real axis of the f-plane maps one-to-one onto the unit circle in the complex z-plane.