Chapter 6

Multidimensional autoregression

The many applications of least squares to the one-dimensional convolution operator constitute the subject known as "**time-series analysis**." The ***autoregression*** filter, also known as the **prediction-error filter** (**PEF**), gathers statistics for us, not the autocorrelation or the spectrum directly but it gathers them indirectly as the inverse of the amplitude spectrum of its input. The PEF plays the role of the so-called "inverse-covariance matrix" in statistical estimation theory. Given the PEF, we use it to find missing portions of signals.

6.0.1 Time domain versus frequency domain

In the simplest applications, solutions can be most easily found in the frequency domain. When complications arise, it is better to use the time domain, to directly apply the convolution operator and the method of least squares.

A first complicating factor in the frequency domain is a required boundary in the time domain, such as that between past and future, or requirements that a filter be nonzero in a stated time interval. Another factor that attracts us to the time domain rather than the frequency domain is **weighting function**s.

Weighting functions are appropriate whenever a signal or image amplitude varies from place to place. Most of the literature on **time-series analysis** applies to the limited case of uniform weighting functions. Such time series are said to be "stationary." This means that their statistical properties do not change in time. In real life, particularly in the analysis of echos, signals are never stationary in time and space. A **stationarity** assumption is a reasonable starting assumption, but we should know how to go beyond it so we can take advantage of the many opportunities that do arise. In order of increasing difficulty in the frequency domain are the following complications:

1. A time boundary such as between past and future.

2. More time boundaries such as delimiting a filter.

3. More time boundaries such as erratic locations of missing data.

4. Nonstationary signal, i.e., time-variable weighting.

5. Time-axis stretching such as normal moveout.

We will not have difficulty with any of these complications here, because we will stay in the time domain and set up and solve optimization problems by use of the conjugate-direction method. Thus we will be able to cope with great complexity in goal formulation and get the right answer without approximations. By contrast, analytic or partly analytic methods can be more economical, but they generally solve somewhat different problems than those given to us by nature.

6.1 SOURCE WAVEFORM, MULTIPLE REFLECTIONS

Here we devise a simple mathematical model for deep **water bottom** multiple reflections.[1] There are two unknown waveforms, the source waveform $S(\omega)$ and the ocean-floor reflection $F(\omega)$. The water-bottom primary reflection $P(\omega)$ is the convolution of the source waveform with the water-bottom response; so $P(\omega) = S(\omega)F(\omega)$. The first multiple reflection $M(\omega)$ sees the same source waveform, the ocean floor, a minus one for the free surface, and the ocean floor again. Thus the observations $P(\omega)$ and $M(\omega)$ as functions of the physical parameters are

$$P(\omega) = S(\omega)\, F(\omega) \tag{6.1}$$
$$M(\omega) = -S(\omega)\, F(\omega)^2 \tag{6.2}$$

Algebraically the solutions of equations (1.1) and (1.2) are

$$F(\omega) = -M(\omega)/P(\omega) \tag{6.3}$$
$$S(\omega) = -P(\omega)^2/M(\omega) \tag{6.4}$$

These solutions can be computed in the Fourier domain by simple division. The difficulty is that the divisors in equations (1.3) and (1.4) can be zero, or small. This difficulty can be attacked by use of a positive number ϵ to **stabilize** it. For example, multiply equation (1.3) on top and bottom by $P'(\omega)$ and add $\epsilon > 0$ to the denominator. This gives

$$F(\omega) = -\frac{M(\omega)P'(\omega)}{P(\omega)P'(\omega) + \epsilon} \tag{6.5}$$

[1] For this short course I am omitting here many interesting examples of multiple reflections shown in my 1992 book, PVI.

where $P'(\omega)$ is the complex conjugate of $P(\omega)$. Although the ϵ stabilization seems nice, it apparently produces a nonphysical model. For ϵ large or small, the time-domain response could turn out to be of much greater duration than is physically reasonable. This should not happen with perfect data, but in real life, data always has a limited spectral band of good quality.

Functions that are rough in the frequency domain will be long in the time domain. This suggests making a short function in the time domain by local smoothing in the frequency domain. Let the notation $< \cdots >$ denote smoothing by local averaging. Thus, to specify filters whose time duration is not unreasonably long, we can revise equation (1.5) to

$$F(\omega) \quad = \quad - \frac{< M(\omega) P'(\omega) >}{< P(\omega) P'(\omega) + \epsilon >} \tag{6.6}$$

where it remains to specify the method and amount of smoothing. The goal of finding the filters $F(\omega)$ and $S(\omega)$ is to best model the multiple reflections so that they can be subtracted from the data, and thus enable us to see what primary reflections have been hidden by the multiples.

These frequency-duration difficulties do not arise in a time-domain formulation. Unlike in the frequency domain, in the time domain it is easy and natural to limit the duration and location of the nonzero time range of $F(\omega)$ and $S(\omega)$. First express (1.3) as

$$0 \quad = \quad P(\omega) F(\omega) + M(\omega) \tag{6.7}$$

To imagine equation (1.7) as a fitting goal in the time domain, instead of scalar functions of ω, think of vectors with components as a function of time. Thus $\mathbf{f}$ is a column vector containing the unknown sea-floor filter, $\mathbf{m}$ contains the "multiple" portion of a seismogram, and $\mathbf{P}$ is a matrix of down-shifted columns, each column being the "primary".

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \end{bmatrix} \quad = \quad \begin{bmatrix} p_1 & 0 & 0 \\ p_2 & p_1 & 0 \\ p_3 & p_2 & p_1 \\ p_4 & p_3 & p_2 \\ p_5 & p_4 & p_3 \\ p_6 & p_5 & p_4 \\ 0 & p_6 & p_5 \\ 0 & 0 & p_6 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} + \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \end{bmatrix} \tag{6.8}$$

To minimize $\mathbf{r}'\mathbf{r}$, we could use the conjugate-direction subroutine `cgmeth()` on page 48, but we would remove its call to the matrix multiply subroutine and replace it by a convolution subroutine with boundary conditions of our choice.

6.2 TIME-SERIES AUTOREGRESSION

Given y_t and y_{t-1}, you might like to predict y_{t+1}. The prediction could be a scaled sum or difference of y_t and y_{t-1}. This is called "**autoregression**" because a signal is regressed on itself. To find the scale factors you would optimize the fitting goal below, for the **prediction filter** (f_1, f_2):

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \begin{bmatrix} y_1 & y_0 \\ y_2 & y_1 \\ y_3 & y_2 \\ y_4 & y_3 \\ y_5 & y_4 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} - \begin{bmatrix} y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} \tag{6.9}$$

(In practice, of course the system of equations would be much taller, and perhaps somewhat wider.) A typical row in the matrix (1.9) says that $y_{t+1} \approx y_t f_1 + y_{t-1} f_2$ hence the description of f as a "prediction" filter. The error in the prediction is simply the residual. Define the residual to have opposite polarity and merge the column vector into the matrix, so you get

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \approx \quad \mathbf{r} \quad = \quad \begin{bmatrix} y_2 & y_1 & y_0 \\ y_3 & y_2 & y_1 \\ y_4 & y_3 & y_2 \\ y_5 & y_4 & y_3 \\ y_6 & y_5 & y_4 \end{bmatrix} \begin{bmatrix} 1 \\ -f_1 \\ -f_2 \end{bmatrix} \tag{6.10}$$

which is a standard form for autoregressions and prediction error.

Multiple reflections are predictable. It is the unpredictable part of a signal, the prediction residual, that contains the primary information. The output of the filter $(1, -f_1, -f_2) = (a_0, a_1, a_2)$ is the unpredictable part of the input. This filter is a simple example of a "prediction-error" (PE) filter. It is one member of a family of filters called "error filters."

The error-filter family are filters with one coefficient constrained to be unity and various other coefficients constrained to be zero. Otherwise, the filter coefficients are chosen to have minimum power output. Names for various error filters follow:

$(1, a_1, a_2, a_3, \cdots, a_n)$	**prediction-error (PE) filter**
$(1, 0, 0, a_3, a_4, \cdots, a_n)$	gapped PE filter with a gap
$(a_{-m}, \cdots, a_{-2}, a_{-1}, 1, a_1, a_2, a_3, \cdots, a_n)$	**interpolation-error (IE) filter**

We introduce a **free-mask matrix K** which "passes" the freely variable coefficients in the filter and "rejects" the constrained coefficients (which in this first example is merely the first coefficient $a_0 = 1$).

$$\mathbf{K} \quad = \quad \begin{bmatrix} 0 & . & . \\ . & 1 & . \\ . & . & 1 \end{bmatrix} \tag{6.11}$$

To compute a simple prediction error filter $\mathbf{a} = (1, a_1, a_2)$ with the CD method, we write (1.9) or (1.10) as

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \begin{bmatrix} y_2 & y_1 & y_0 \\ y_3 & y_2 & y_1 \\ y_4 & y_3 & y_2 \\ y_5 & y_4 & y_3 \\ y_6 & y_5 & y_4 \end{bmatrix} \begin{bmatrix} 0 & \cdot & \cdot \\ \cdot & 1 & \cdot \\ \cdot & \cdot & 1 \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} \tag{6.12}$$

Let us move from this specific fitting goal to the general case. (Notice the similarity of the free-mask matrix $\mathbf{K}$ in this filter estimation problem with the free-mask matrix $\mathbf{J}$ in missing data goal (3.3).) The fitting goal is,

$$\mathbf{0} \approx \mathbf{Ya} \tag{6.13}$$
$$\mathbf{0} \approx \mathbf{Y}(\mathbf{I} - \mathbf{K} + \mathbf{K})\mathbf{a} \tag{6.14}$$
$$\mathbf{0} \approx \mathbf{YKa} + \mathbf{Y}(\mathbf{I} - \mathbf{K})\mathbf{a} \tag{6.15}$$
$$\mathbf{0} \approx \mathbf{YKa} + \mathbf{Ya}_0 \tag{6.16}$$
$$\mathbf{0} \approx \mathbf{YKa} + \mathbf{y} \tag{6.17}$$
$$\mathbf{0} \approx \mathbf{r} = \mathbf{YKa} + \mathbf{r}_0 \tag{6.18}$$

which means we initialize the residual with $\mathbf{r}_0 = \mathbf{y}$ and then iterate with

$$\Delta\mathbf{a} \longleftarrow \mathbf{K}'\mathbf{Y}'\,\mathbf{r} \tag{6.19}$$
$$\Delta\mathbf{r} \longleftarrow \mathbf{YK}\,\Delta\mathbf{a} \tag{6.20}$$

Bringing this all together gives us subroutine `gdecon()`.

```
module gdecon_mod { # gapped deconvolution.  input=yy  output=rr
  use icaf1
  use cgstep_mod
  use solver_mod
contains
  subroutine gdecon( na, yy, rr, niter, gap) {
    integer,                intent( in)  :: na, niter, gap
    real,    dimension( :), pointer      :: yy
    real,    dimension( :), intent( out) :: rr
    real,    dimension( na)              :: aa # prediction-error filter
    logical, dimension( na)              :: mm # filter mask
    real,    dimension( size( yy))       :: dd # data
    dd = 0.                                    # initialize data
    aa = 0.; aa( 1) = 1.                       # initialize filter
    mm = .false.; mm( :gap) = .true.           # initialize mask
    call icaf1_init( yy, 1)                    # initialize convolution
    call solver( icaf1_lop, cgstep, aa, dd, niter,
                 x0= aa, res= rr, known= mm)
    call cgstep_close()
  }
}
```

My previous book, PVI, has many examples of gapped deconvolution on real and synthetic data. This book develops more two-dimensional examples.

6.3 PREDICTION-ERROR FILTER OUTPUT IS WHITE

The relationship between spectrum and PEF

Knowledge of an autocorrelation function is equivalent to knowledge of a spectrum. The two are simply related by Fourier transform. A spectrum or an autocorrelation function encapsulates an important characteristic of a signal or an image. Generally the spectrum changes slowly from place to place although it could change rapidly. Of all the assumptions we could make to fill empty bins, one that people usually find easiest to agree with is that the spectrum should be the same in the empty-bin regions as where bins are filled. In practice we deal with neither the spectrum nor its autocorrelation but with a third object. This third object is easily transformed to the spectrum. This third object is the Prediction Error Filter (PEF). Its spectrum is inverse to the data spectrum.

Undoing convolution in nature

Prediction-error filtering is also called "**deconvolution**". This word goes back to very basic models and concepts. In this model one envisions a random white-spectrum excitation function $\mathbf{x}$ existing in nature, and this excitation function is somehow filtered by unknown natural processes, with a filter operator $\mathbf{B}$ producing an *output* $\mathbf{y}$ in nature that becomes the *input* $\mathbf{y}$ to our computer programs. This is sketched in Figure 1.1. Then we design a prediction-error filter $\mathbf{A}$ on $\mathbf{y}$, which yields a white-

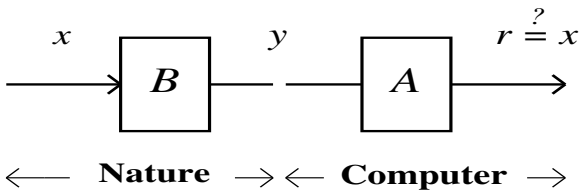


Figure 6.1: Flow of information from nature, to observation, into computer. mda-systems [NR]

spectrumed output residual $\mathbf{r}$. Because $\mathbf{r}$ and $\mathbf{x}$ theoretically have the same spectrum, the tantalizing prospect is that maybe $\mathbf{r}$ equals $\mathbf{x}$, meaning that the PEF $\mathbf{A}$ has *deconvolved* the unknown convolution $\mathbf{B}$.

Causal with causal inverse

Theoretically, a PEF is a causal filter with a causal inverse. This adds confidence to the likelihood that deconvolution of natural processes with a PEF might get the correct phase spectrum as well as the correct amplitude spectrum. Naturally, the PEF does not give the correct phase to an "all-pass" filter. That is a filter with a phase shift but a constant amplitude spectrum. (I think most migration operators are in this category.)

Theoretically we should be able to use a PEF in either convolution or polynomial division. There are some dangers though, mainly connected with dealing with data in small windows. Truncation phenomena might give us PEF estimates that are causal, but whose inverse is not, so they cannot be used in polynomial division. This is a lengthy topic in the classic literature. This old, fascinating subject is examined in my books, FGDP and PVI. A classic solution is one by John Parker Burg. We should revisit the Burg method in light of the helix.

PEF output tends to whiteness

The most important property of a **prediction-error filter** or **PEF** is that its output tends to a **white spectrum**. No matter what the input to this filter, its output tends to whiteness as the number of the coefficients $n \to \infty$ tends to infinity. Thus, the **PE filter** adapts itself to the input by absorbing all its **color**. This has important statistical implications and important geophysical implications.

Spectral estimation

The PEF's output being white leads to an important consequence: To specify a spectrum, we can give the spectrum (of an input) itself, give its autocorrelation, or give its PEF coefficients. Each is transformable to the other two. Indeed, an effective mechanism of spectral estimation, developed by John P. **Burg** and described in **FGDP**, is to compute a PE filter and look at the inverse of its spectrum.

Short windows

The power of a PE filter is that a short filter can often extinguish, and thereby represent, the information in a long resonant filter. If the input to the PE filter is a sinusoid, it is exactly predictable by a three-term recurrence relation, and all the color is absorbed by a three-term PE filter (see exercises). Burg's spectral estimation is especially effective in short windows.

Weathered layer resonance

That the output spectrum of a PE filter is **white** is also useful geophysically. Imagine the reverberation of the **soil** layer, highly variable from place to place, as the resonance between the surface and deeper consolidated rocks varies rapidly with surface location because of geologically recent fluvial activity. The spectral **color** of this erratic variation on surface-recorded seismograms is compensated for by a PE filter. Of course, we do not want PE-filtered seismograms to be white, but once they all have the same spectrum, it is easy to postfilter them to any desired spectrum.

6.3.1 PEF whiteness proof in 1-D

The basic idea of least-squares fitting is that the residual is orthogonal to the fitting functions. Applied to the PE filter, this idea means that the output of a PE filter is orthogonal to lagged inputs. The **orthogonality** applies only for lags in the past, because prediction knows only the past while it aims to the future. What we want to show is different, namely, that the output is uncorrelated with *itself* (as opposed to the input) for lags in *both* directions; hence the output spectrum is **white**.

In (1.21) are two separate and independent autoregressions, $\mathbf{0} \approx \mathbf{Y}_a\mathbf{a}$ for finding the filter $\mathbf{a}$, and $\mathbf{0} \approx \mathbf{Y}_b\mathbf{b}$ for finding the filter $\mathbf{b}$. Evidently, if the matrices were infinitely tall, or if the values in the columns of the matrices got small towards the top and bottom, we would find that $\mathbf{a} = \mathbf{b}$.

$$\mathbf{0} \approx \mathbf{r}_a = \begin{bmatrix} y_1 & 0 & 0 \\ y_2 & y_1 & 0 \\ y_3 & y_2 & y_1 \\ y_4 & y_3 & y_2 \\ y_5 & y_4 & y_3 \\ y_6 & y_5 & y_4 \\ y_7 & y_6 & y_5 \\ y_8 & y_7 & y_6 \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} \quad ; \qquad \mathbf{0} \approx \mathbf{r}_b = \begin{bmatrix} 0 & 0 & 0 \\ y_1 & 0 & 0 \\ y_2 & y_1 & 0 \\ y_3 & y_2 & y_1 \\ y_4 & y_3 & y_2 \\ y_5 & y_4 & y_3 \\ y_6 & y_5 & y_4 \\ y_7 & y_6 & y_5 \end{bmatrix} \begin{bmatrix} 1 \\ b_1 \\ b_2 \end{bmatrix} \tag{6.21}$$

When the energy $\mathbf{r}'\mathbf{r}$ of a residual has been minimized, the residual $\mathbf{r}$ is orthogonal to the fitting functions. For example, choosing a_2 to minimize $\mathbf{r}'\mathbf{r}$ gives $0 = \partial\mathbf{r}'\mathbf{r}/\partial a_2 = 2\mathbf{r}'\partial\mathbf{r}/\partial a_2$. This shows that $\mathbf{r}'$ is perpendicular to $\partial\mathbf{r}/\partial a_2$ which is the rightmost column of the $\mathbf{Y}_a$ matrix. Thus the vector $\mathbf{r}_a$ is orthogonal to all the columns in the $\mathbf{Y}_a$ matrix except the first (because we do not minimize with respect to a_0).

Our goal is a different theorem that is imprecise when applied to the three coefficient filters displayed in (1.21), but becomes valid as the filter length tends to infinity $\mathbf{a} = (1, a_1, a_2, a_3, \cdots)$ and the matrices become infinitely wide. Actually, all we require is that b_n tend to zero. This generally happens because as n increases, y_{t-n} becomes a weaker and weaker predictor of y_t.

The matrix $\mathbf{Y}_a$ contains all of the columns that are found in $\mathbf{Y}_b$ except the last (and the last one will turn out be irrelevant). This means that $\mathbf{r}_a$ is not only orthogonal to all of $\mathbf{Y}_a$'s columns (except the first) but $\mathbf{r}_a$ is also orthogonal to all of $\mathbf{Y}_b$'s columns except the last. Although $\mathbf{r}_a$ isn't really perpendicular to the last column of $\mathbf{Y}_b$, it doesn't matter because that column has hardly any contribution to $\mathbf{r}_b$ since $|b_n| << 1$. Because $\mathbf{r}_a$ is (effectively) orthogonal to all the components of $\mathbf{r}_b$, $\mathbf{r}_a$ is also orthogonal to $\mathbf{r}_b$ itself. (For any $\mathbf{u}$ and $\mathbf{v}$, if $\mathbf{r} \cdot \mathbf{u} = 0$ and $\mathbf{r} \cdot \mathbf{v} = 0$ then $\mathbf{r} \cdot (\mathbf{u} + \mathbf{v}) = 0$).

In choosing the example of Figure (1.21), I have shifted the two fitting problems by one lag. We could draw the two problems again shifted by two lags, three lags, and more, and we would find that $\mathbf{r}_b$ and $\mathbf{r}_a$ are always orthogonal. Actually, $\mathbf{r}_b$ and $\mathbf{r}_a$ both contain the same signal $\mathbf{r}$ but time-shifted. The orthogonality at all shifts

means that the autocorrelation of $\mathbf{r}$ vanishes at all lags. The autocorrelation does not vanish at zero lag, however, because $\mathbf{r}_a$ is not orthogonal to its first column (because we did not minimize with respect to a_0).

As we redraw $\mathbf{0} \approx \mathbf{r}_b = \mathbf{Y}_b\mathbf{b}$ for various lags, we may shift the columns only downward because shifting them upward would bring in the first column of $\mathbf{Y}_a$ and the residual $\mathbf{r}_a$ is not orthogonal to that. Thus we have only proven that one side of the autocorrelation of $\mathbf{r}$ vanishes. That is enough however, because autocorrelation functions are symmetric, so if one side vanishes, the other must also.

We also see where the proof would break if $\mathbf{a}$ and $\mathbf{b}$ were two-sided filters like $(\cdots, b_{-2}, b_{-1}, 1, b_1, b_2, \cdots)$. If $\mathbf{b}$ were two-sided, $\mathbf{Y}_b$ would catch the nonorthogonal column of $\mathbf{Y}_a$. Not only is $\mathbf{r}_a$ not proven to be perpendicular to the first column of $\mathbf{Y}_a$, but it cannot be orthogonal to it because a signal cannot be orthogonal to itself.

The consequence of this theorem is that the convolution of $\mathbf{y}$ and $\mathbf{a}$ is white noise (its autocorrelation is an impulse function) and that means that $\mathbf{y}$ and $\mathbf{a}$ have mutually inverse spectrums. In other words, $\mathbf{a}$ captures a fundamental statistical aspect of $\mathbf{y}$. Where information is missing we can use the PEF to guess it.

6.3.2 Simple dip filters

Convolution in two dimensions is just like convolution in one dimension except that convolution is done on two axes. The input and output data are planes of numbers and the filter is also a plane. A two-dimensional filter is a small plane of numbers that is convolved over a big data plane of numbers.

Suppose the data set is a collection of seismograms uniformly sampled in space. In other words, the data is numbers in a (t, x)-plane. For example, the following filter destroys any wavefront aligned along the direction of a line containing both the "+1" and the "−1".

$$\begin{matrix} -1 & \cdot \\ \cdot & \cdot \\ \cdot & 1 \end{matrix} \tag{6.22}$$

The next filter destroys a wave with a slope in the opposite direction:

$$\begin{matrix} \cdot & 1 \\ -1 & \cdot \end{matrix} \tag{6.23}$$

To convolve the above two filters, we can reverse either on (on both axes) and correlate them, so that you can get

$$\begin{matrix} \cdot & -1 & \cdot \\ 1 & \cdot & \cdot \\ \cdot & \cdot & 1 \\ \cdot & -1 & \cdot \end{matrix} \tag{6.24}$$

which destroys waves of both slopes.

A **two-dimensional filter** that can be a **dip-rejection filter** like (1.22) or (1.23) is

$$\begin{array}{cc} a & \cdot \\ b & \cdot \\ c & 1 \\ d & \cdot \\ e & \cdot \end{array} \tag{6.25}$$

where the coefficients (a, b, c, d, e) are to be estimated by least squares in order to minimize the power out of the filter. (In the filter table, the time axis runs vertically.)

Fitting the filter to two neighboring traces that are identical but for a time shift, we see that the filter coefficients (a, b, c, d, e) should turn out to be something like $(-1, 0, 0, 0, 0)$ or $(0, 0, -.5, -.5, 0)$, depending on the dip (stepout) of the data. But if the two channels are not fully coherent, we expect to see something like $(-.9, 0, 0, 0, 0)$ or $(0, 0, -.4, -.4, 0)$. To find filters such as (1.24), we adjust coefficients to minimize the power out of filter shapes, as in

$$\begin{array}{ccc} v & a & \cdot \\ w & b & \cdot \\ x & c & 1 \\ y & d & \cdot \\ z & e & \cdot \end{array} \tag{6.26}$$

With 1-dimensional filters, we think mainly of power spectra, and with 2-dimensional filters we can think of temporal spectra and spatial spectra. What is new, however, is that in two dimensions we can think of dip spectra (which is when a 2-dimensional spectrum has a particularly common form, namely when energy organizes on radial lines in the (ω, k_x)-plane). As a short (three-term) 1-dimensional filter can devour a sinusoid, we have seen that simple 2-dimensional filters can devour a small number of dips.

6.3.3 PEF whiteness proof in 2-D

A well-known property (see FGDP or PVI) of a 1-D PEF is that its energy clusters immediately after the impulse at zero delay time. Applying this idea to the helix in Figure 4.2 on page 88 shows us that we can consider a 2-D PEF to be a small halfplane like (4.6) on page 94 with an impulse along a side. These shapes are what we see here in Figure 1.2.

Figure 1.2 shows the input plane with a 2-D filter on top of it at two possible locations. The filter shape is a semidisk, which you should imagine being of infinitely large radius. Notice that semidisk A includes all the points in B. The output of disk A will be shown to be orthogonal to the output of disk B. Conventional least squares theory says that the coefficients of the filter are designed so that the output of the filter is orthogonal to each of the inputs to that filter (except for the input under

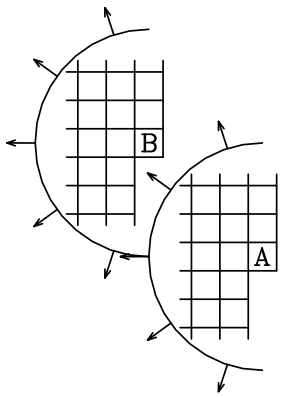


Figure 6.2: A 2-D whitening filter template, and itself lagged. At output locations "A" and "B," the filter coefficient is constrained to be "1". When the semicircles are viewed as having infinite radius, the B filter is contained in the A filter. Because the output at A is orthogonal to all its inputs, which include all inputs of B, the output at A is orthogonal to the output of B. mda-whitepruf [ER]

the "1," because any nonzero signal cannot be orthogonal to itself). Recall that if a given signal is orthogonal to each in a given group of signals, then the given signal is orthogonal to all linear combinations within that group. The output at B is a linear combination of members of its input group, which is included in the input group of A, which are already orthogonal to A. Therefore the output at B is orthogonal to the output at A. In summary,

residual	$\perp$	fitting function
output at A	$\perp$	each input to A
output at A	$\perp$	each input to B
output at A	$\perp$	linear combination of each input to B
output at A	$\perp$	output at B

The essential meaning is that a particular lag of the output **autocorrelation** function vanishes.

Study Figure 1.2 to see for what lags all the elements of the B filter are wholly contained in the A filter. These are the lags where we have shown the output autocorrelation to be vanishing. Notice another set of lags where we have proven nothing (where B is moved to the right of A). Autocorrelations are centrosymmetric, which means that the value at any lag is the same as the value at the negative of that lag, even in 2-D and 3-D where the lag is a vector quantity. Above we have shown that a halfplane of autocorrelation values vanishes. By the centrosymmetry, the other half must vanish too. Thus the autocorrelation of the PEF output is an impulse function, so its 2-D spectrum is white.

The helix tells us why the proper filter form is not a square with the "1" on the corner. Before I discovered the helix, I understood it another way (that I learned from John P. Burg): For a spectrum to be white, *all* nonzero autocorrelation lags must be zero-valued. If the filter were a quarter-plane, then the symmetry of autocorrelations would only give us vanishing in another quarter, so there would be two remaining quarter-planes where the autocorrelation was not zero.

Fundamentally, the white-output theorem requires a one-dimensional ordering to the values in a plane or volume, and the same filter must apply no matter where it is applied. Additionally, the filter must contain a halfplane of values so that symmetry gives the other half. It seems that we could design whitening autoregression filters for 45° rotations, and we could also design them for hexagonal coordinate systems.

6.3.4 The shape of the 3-D prediction-error filter

Imagine a 3-D PEF and through the "1.0" put a plane that has an infinitesimal tilt so as to get all zeros on one side of the plane and all fitting coefficients are on the other. Figure 1.3 shows the volume of free coefficients of the **three-dimensional prediction-error filter** . The top plane is the 2-D filter seen earlier. The top plane can be visualized as the area around the end of a helix. Above the top plane are zero-valued anticausal filter coefficients.

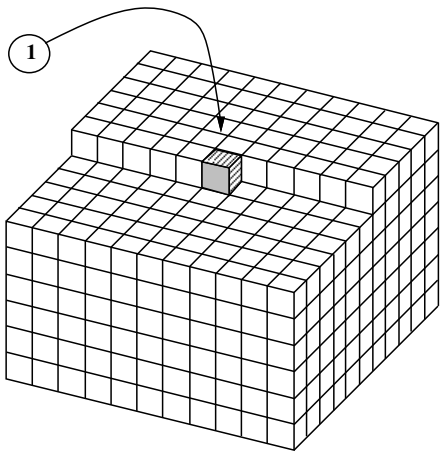


Figure 6.3: Three-dimensional prediction-error filter. mda-3dpef [NR]

Module `lagtable` on page 12 constructs a linear subsript of filter lags along the helix from the 3-D cartesian description. To print out the cartesian form of a helix filter, you can use module `format` on page 13.

```
module lagtable {          # Find 3-D PEF filter coef locations on a helix.
contains
  subroutine table3( n1,n2, center1,center2, a1,a2,a3, aa, lag) {
    integer, intent (in)            :: n1,n2, center1,center2, a1,a2,a3
    real,    dimension (:), pointer :: aa                  # needed by helicon.
    integer, dimension (:), pointer :: lag                 # needed by helicon.
    integer                         :: i,j, i1,i2,i3, na
    na = (a3-1)*a1*a2 + (a2-center2)*a1 + (a1-center1)
    allocate ( aa( na), lag( na));  aa = 0.          # a(center1,center2,1) = 1.
    do i3 = 1, a3 {
     do i2 = 1, a2 {    if( i3 == 1 .and. i2  < center2                 ) cycle
      do i1 = 1, a1 {   if( i3 == 1 .and. i2 == center2.and.i1 <= center1) cycle
          i = i1-center1 + (i2-center2)*a1 + (i3-1)*a1*a2
          j = i1-center1 + (i2-center2)*n1 + (i3-1)*n1*n2
```

```
          lag( i)  =  j                          # lag on helix for a(i1,i2,i3)
          }}}
    }
}
```

```
module format {
contains
  subroutine print3( n1, n2, center1, center2, a1, a2, a3, aa, lag) {
    integer,                intent( in) :: n1, n2, center1, center2,  a1,a2,a3
    real,    dimension( :), intent( in) :: aa
    integer, dimension( :), intent( in) :: lag
    integer                             :: ia, i1, i2, i3, i
    real, dimension( a1, a2, a3)        :: filt            # cartesian filter
    filt = 0.;    filt( center1, center2, 1) = 1.
    do ia = 1, size( lag) {
       i = lag( ia) + center1 + ( center2-1)*n1 - 1
       i1 = mod( i        , n1)
       i2 = mod( i/n1     , n2)
       i3 =      i/(n1*n2)
       filt( i1+1, i2+1, i3+1) = aa( ia)
    }
                    write( 0, *) "----------------------------------------"
    do i3 = 1, a3 {                                 # loop over planes
       if( i3 > 1)  write( 0, *) "++++++++++++++++++++++++++++++++++++++++"
       do i1 = 1, a1
                    write( 0, "(10f7.3)") filt( i1, :, i3) # print filter row
    }
                    write( 0, *) "----------------------------------------"
  }
}
```

A reasonable arrangement for a small filter is `a1=5 a2=3 a3=2 center1=1+a1/2=3 center2=1+a2/2=2`. I set all the filter coefficients to 2 except for a(3,2,1)=1. Then I invoked `print3`. The output is:

```
----------------------------------------
  0.000  0.000  2.000
  0.000  0.000  2.000
  0.000  1.000  2.000
  0.000  2.000  2.000
  0.000  2.000  2.000
++++++++++++++++++++++++++++++++++++++++
  2.000  2.000  2.000
  2.000  2.000  2.000
  2.000  2.000  2.000
  2.000  2.000  2.000
  2.000  2.000  2.000
----------------------------------------
```

where `i1` runs vertical, `i2` runs horizontal in a block, `i3` jumps between blocks.

6.3.5 Infill of 3-D seismic data from a quarry blast

Finding **missing data** (filling empty bins) requires use of a filter. In this subsection we will assume that the filter is known. In the next we will see how to find the filter. Except for using a 2-D convolution operator instead of a 1-D one, the theory is all the same in 2-D as it was in 1-D in Chapter 3, equations (3.3) to (3.11), and there is no reason to repeat it because the helix converts 2-D to 1-D. An open question is how many conjugate-direction iterations are needed in missing-data programs. When estimating filters, I set the **iteration count** `niter` at the number of free filter parameters. Theoretically, this gives me the exact solution. The number of free parameters in the missing-data estimation, however, could be very large. This fact implies impractical compute times for the exact solution (and thus calls for preconditioning). I find that where gaps are small, they fill in quickly. Where the gaps are large, they don't, and more iterations are required.

Figure 1.4 shows an example of replacing missing data by values predicted from a 3-D PEF. The data was recorded at **Stanford University** with a 13×13 array of independent recorders. The figure shows 12 of the 13 lines each of length 13. Our main goal was to measure the ambient night-time noise. By morning about half the recorders had dead batteries but the other half recorded a wave from a quarry blast. The raw data was distracting to look at because of the many missing traces so I interpolated it using parameters for the small filter mentioned with subroutine `print3()` on page 13.

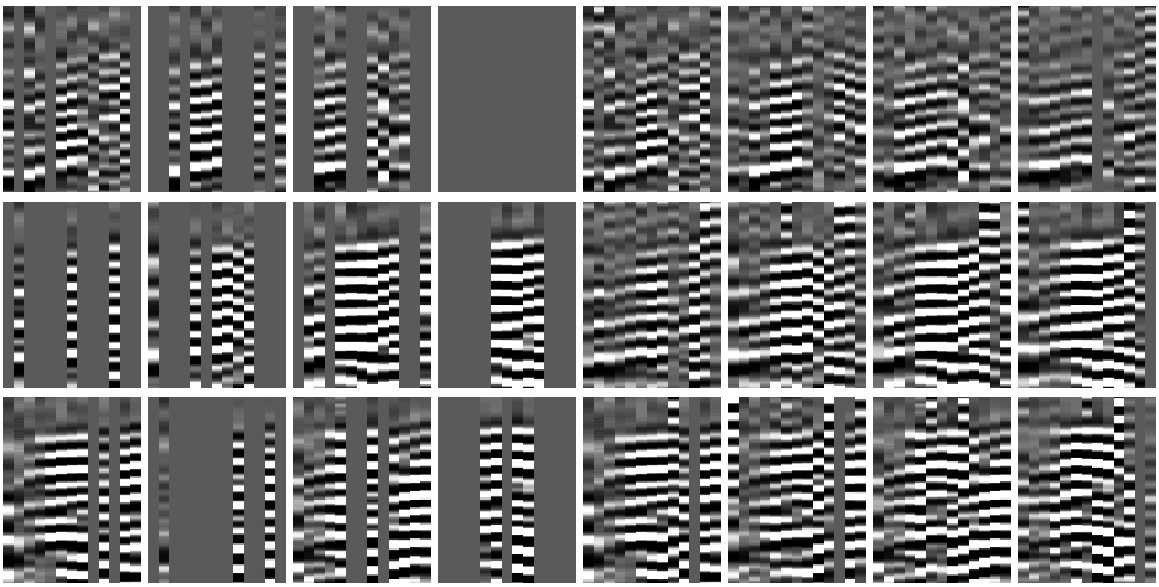


Figure 6.4: The left 12 panels are the inputs. The right 12 panels are outputs. mda-passfill90 [ER]

6.4 WEIGHTED PEF ESTIMATION

There are *practical* reasons for wanting **weighting function**s in fitting goals and there are *physical* reasons. When data points are missing for accidental reasons, they spoil the fitting equations they affect and it is best to simply weight those equations by zero.

Physically, echos get weaker with time, though the information content is unrelated to the signal strength. Echos also vary in strength as different materials are encountered by the outgoing wave. Programs for echo analysis typically divide the data by a scaling factor that is a smoothed average of the signal strength. This practice is nearly universal, although it is fraught with hazards. An example of **automatic gain control** (**AGC**) is when we compute the divisor by forming the absolute value of the signal strength and then smoothing by convolution with a triangle or damped exponential. Pitfalls are strange amplitude behavior surrounding the water bottom, and the overall loss of information contained in amplitudes. Personally, I have found that the gain function t^2 nearly always eliminates the need for AGC on raw field data, but without doubt, AGC is occasionally needed. (A theoretical explanation for t^2 is given in **IEI**.)

Equation (1.27) shows a tiny example of a weighted least-squares filtering goal. In real-life applications, the output is typically 1000-2000 points and the filter 5-50 points.

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \approx \quad \begin{bmatrix} w_1 & 0 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 & 0 \\ 0 & 0 & w_3 & 0 & 0 \\ 0 & 0 & 0 & w_4 & 0 \\ 0 & 0 & 0 & 0 & w_5 \end{bmatrix} \left(\begin{bmatrix} y_1 & 0 & 0 \\ y_2 & y_1 & 0 \\ y_3 & y_2 & y_1 \\ 0 & y_3 & y_2 \\ 0 & 0 & y_3 \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} \right) \tag{6.27}$$

The weights w_t are any positive numbers we choose. In "accidental" applications, we set weights to zero where data points are missing or bad. In physical applications, we choose the w_t so that the residual components are about equal in magnitude. (A generalization of weighting is where the weighting matrix is a filtering matrix or a product of weighting and filtering.)

Equation (1.27) is of the form $\mathbf{0} \approx \mathbf{W}(\mathbf{Ya} - \mathbf{d})$. To convert this to a new problem without weights we define a new data vector $\mathbf{Wd}$ and a new operator $\mathbf{WY}$ simply by carrying $\mathbf{W}$ through the parentheses to $\mathbf{0} \approx (\mathbf{WY})\mathbf{a} - \mathbf{Wd}$.

An earlier version of this book used a weighting function equal zero for fitting equations that lacked inputs. In this version of the book, we will accomplish the same goals with logical masks so that values to be multiplied by zero are never even computed.

6.5 TWO-STAGE LINEAR LEAST SQUARES

In Chapter 3 we filled empty bins by minimizing the energy output from the filtered mesh. In each case there was arbitrariness in the choice of the filter. Here we use the optimum filter, the PEF.

Here we have code in which **missing data** in a 2-D data plane is restored by a two-stage linear least squares process. In the first stage, we fit a 2-D prediction-error filter (PEF) to the given plane. Fitting equations that involve empty bins are weighted to zero before the least-squares problem is solved by conjugate direction descent. In a second stage, we take the PEF as known and find empty bin values in (or beyond) the plane so that the power out of the prediction-error filter is minimized. This two-stage method avoids the nonlinear problem we would otherwise face if we included the fitting equations containing both free data values and free filter values. Presumably, after two stages of linear least squares we are close enough to the final solution that we could switch over to the full nonlinear setup described near the end of this chapter.

Our subroutines for finding the PEF would have the pitfall that they would try to use the missing data to find the PEF, and hence they would get the wrong PEF. We can use a PEF finder that uses weighted least squares. Then we can use a weighting function that vanishes on those fitting equations that involve missing data (and is unity elsewhere). The `hconest` module is a convolution module that we will use. Instead of multiplying a convolution output by a zero, it simply does not compute the convolution output.

```
module hconest {             # masked helix convolution, adjoint is the filter.
  real,    dimension( :), pointer :: x
  logical, dimension( :), pointer :: bad        # output depends on bad inputs
  integer, dimension( :), pointer :: lag
#% _init( bad, x, lag)
#% _lop( a, y)
    integer  ia, ix, iy
    do ia = 1, size( a) {
         do iy = 1  + lag( ia), size( y) {    if( bad( iy)) cycle
            ix = iy - lag( ia)
             if( adj)
                    a( ia) = a( ia) + y( iy) * x( ix)
             else
                    y( iy) = y( iy) + a( ia) * x( ix)
         }
    }
}
```

Suppose that y_4 were a missing or a bad data value in the fitting goal (1.28). That would spoil the 4th, 5th and 6th fitting equations. Thus we would want to be sure

that w_4, w_5, and w_6 were zero.

$$\mathbf{0} \approx \mathbf{Wr} = \begin{bmatrix} w_1 & . & . & . & . & . & . & . \\ . & w_2 & . & . & . & . & . & . \\ . & . & w_3 & . & . & . & . & . \\ . & . & . & w_4 & . & . & . & . \\ . & . & . & . & w_5 & . & . & . \\ . & . & . & . & . & w_6 & . & . \\ . & . & . & . & . & . & w_7 & . \\ . & . & . & . & . & . & . & w_8 \end{bmatrix} \begin{bmatrix} y_1 & 0 & 0 \\ y_2 & y_1 & 0 \\ y_3 & y_2 & y_1 \\ y_4 & y_3 & y_2 \\ y_5 & y_4 & y_3 \\ y_6 & y_5 & y_4 \\ y_7 & y_6 & y_5 \\ y_8 & y_7 & y_6 \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} \tag{6.28}$$

When missing y_i are sprinkled around, which of the w_i should vanish? A simple solution (that we cannot use) is to put zeros in place of missing data and then take the product of every element in a row of the $\mathbf{Y}$ matrix. The product will vanish if any element in the row vanishes. We cannot use this solution because it would require us to have the matrix, when actually we have only an operator, a program pair that applies the matrix. A useful solution uses complementary logic: We prepare a template like $\mathbf{y}$ with ones where data is missing and zeros where the data is known. We prepare a template for $\mathbf{a}$ where all values are ones. We put the templates into (1.28) and apply the operator. Where the outputs are nonzero we have defective fitting equations and that is where we want the weights to be zero.

6.5.1 Internal boundaries to multidimensional convolution

Sometimes we deal with small patches of data. In order that boundary phenomena not dominate the calculation intended in the central region, we need to take care that input data is not assumed to be zero beyond the interval that the data is given. Examine figure 1.5 There are two things to understand about Fig 1.5. First, sliding the

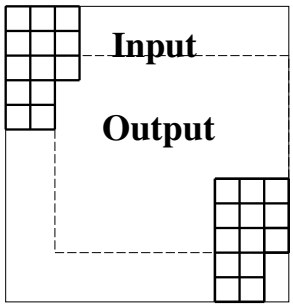


Figure 6.5: Domain of inputs and outputs of a two-dimensional filter like a PEF. mda-rabdomain [NR]

filter everywhere inside the outer box, we get outputs (under the 1 location) only in the inner box. Second, (the adjoint idea) crosscorrelating the inner and outer boxes gives us the 3×5 patch of information we use to build the filter coefficients. We need to be careful not to assume that signals vanish outside the region where they are defined. In a later chapter we will break data spaces into overlapping patches, separately analyze the patches, and put everything back together. We do this because crosscorrelations change with time and they are handled as constant in short time windows. There we

must be particularly careful that zero signal values not be presumed outside of the small volumes; otherwise the many edges and faces of the many small volumes can overwhelm the interior that we want to study.

In practice, the input and output are allocated equal memory, but the output residual is initialized to zero everywhere and then not computed except where shown in figure 1.5. Below is module `boundary` to build a selector for filter outputs that should be empty. There are two regions to think about: (1) at the beginning where the filter is moving onto the helix, (the left side in Fig 1.5). and (2) on the "backside" of the helix (top and bottom in Fig 1.5) where the filter moves off of one loop and onto the next. With Fig 1.5, you should be able to understand the two-dimensional parts of the the module `boundary`. (The 3-D part will test your visualization skills.)

```
module boundary {     # mark helix filter outputs where input is off data.
contains
  subroutine bound3 (n1, n2, lag1, lag2, a1, a2, a3, misin) {
    integer, intent (in)   :: a1, a2, a3, n1, n2, lag1, lag2
    logical, dimension (:) :: misin
    integer                :: i1, i2, i, n123
    n123 = size( misin)
    misin = .false.
                           # This is the transient onset onto the helix
    do i=1, a1-1        { misin(i)=.true.}    # 1-D
    do i=1,(a2-1)*n1    { misin(i)=.true.}    # 2-D
    do i=1,(a3-1)*n1*n2 { misin(i)=.true.}    # 3-D
                           # These disconnect each loop from the next.
    do i1 = 1, lag1-1       { do i =    i1, n123,    n1 { misin(i)=.true.}} #2-D
    do i1 = n1-a1+lag1+1,n1 { do i =    i1, n123,    n1 { misin(i)=.true.}} #2-D
    do i2 = 1, lag2-1       { do i = i2*n1, n123, n2*n1 { misin(i)=.true.}} #3-D
    do i2 = n2-a2+lag2+1,n2 { do i = i2*n1, n123, n2*n1 { misin(i)=.true.}} #3-D
  }
}
```

6.5.2 Finding the prediction-error filter

The first stage of the least-squares estimation is computing the **prediction-error filter**. The second stage will be using it to find the missing data. The input data space contains a mixture of known data values and missing unknown ones. For the first stage of finding the filter, we generally have many more fitting equations than we need so we can proceed by ignoring the fitting equations that involve missing data values. We ignore them everywhere that the missing inputs hit the filter, either by hitting the filter's "1" or by hitting the filter's adjustable values. To identify these less desirable fitting equations, we prepare a **mask** from two other masks. First we make a mask plane `dfre` that is the size of the data plane which has mask values being +1.0 where data is missing and being 0.0 where data is known. Likewise we make a filter mask `afre` equal +1.0 where there could be nonzero filter coefficients and equal 0.0 where filter coefficients are zero. (This mask is almost the same as the

`afre` previously defined, but we also need to reject equations where the filter's "1" hits a missing data value. Such a fitting equation might be linear but could involve both the unknown filter and also a missing data value under the filter's "1". Here we seek only fitting equations for the filter.)

```
module misinput {                       # find a mask of missing filter inputs
  use helicon
contains
  subroutine find_mask( dd, aa, lag, misin) {
    real,    dimension( :), intent( in)  :: dd
    logical, dimension( :), intent( out) :: misin
    real,    dimension( :), pointer      :: aa
    integer, dimension( :), pointer      :: lag
    real,    dimension( size( dd))       :: rr, dfre
    integer                              :: stat
    where( dd == 0.)
       dfre = 1.
    elsewhere
       dfre = 0.
    aa = 1.
    call helicon_init( aa, lag)
    stat = helicon_lop( .false., .false., dfre, rr)
    where ( rr > 0.)   misin = .true.
    aa = 0.
 }
}
```

The codes here do not address the difficulty that maybe too much data is missing so that all weights are zero. To add stabilization we could supplement the data volume with a synthetic "training dataset" or by a "prior filter". With things as they are, if there is not enough data to specify a prediction-error filter, you should encounter the error exit from `cgstep()` on page 45.

The synthetic data in Figure 1.6 is a superposition of two plane waves of different directions, each with a random (but low-passed) waveform. After punching a hole in the data, we find that the lost data is pleasingly restored, though a bit weak near the side boundary. This imperfection could result from the side-boundary behavior of the operator or from an insufficient number of missing-data iterations.

```
module pef {              # Find prediction-error filter (helix magic)
  use hconest
  use cgstep_mod
  use solver_mod
contains
  subroutine find_pef( dd, aa, lag, misin, niter) {
    integer,                intent( in)  :: niter
    real,    dimension(:), pointer      :: dd          # input data
    logical, dimension(:), pointer      :: misin       # where missing inputs
    integer, dimension(:), pointer      :: lag         # filter 1-D location.
    real,    dimension(:), intent( out) :: aa          # filter coefficient.
    call hconest_init( misin, dd, lag )
```

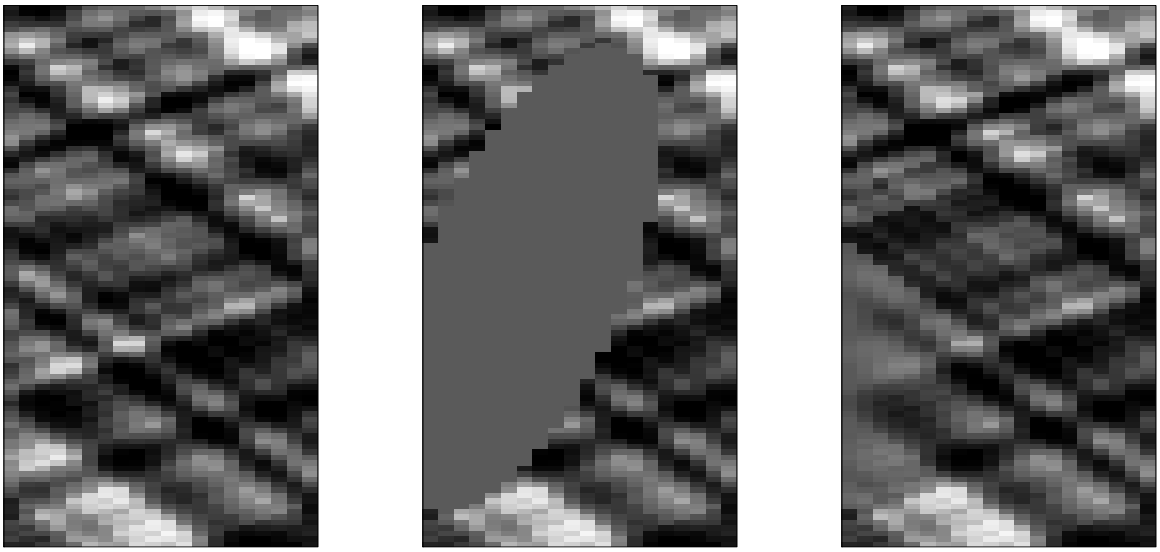


Figure 6.6: Original data (left), with a zeroed hole (center), and as restored (right). mda-hole90 [ER]

```
    call solver( hconest_lop, cgstep, aa, -dd, niter)
    call cgstep_close()
  }
}
```

6.5.3 Imposing prior knowledge of symmetry

Reversing a signal in time does not change its autocorrelation. In the analysis of stationary time series, it is well known (**FGDP**) that the filter for predicting forward in time should be the same as that for "predicting" backward in time (except for time reversal). When the short data are samples, however, a different filter may be found for predicting forward than for backward. Rather than average the two filters directly, the better procedure is to find the filter that minimizes the sum of power in two residuals. One is a filtering of the original signal, and the other is a filtering of a time-reversed signal, as in equation (1.29), where the top half of the equations represent prediction-error predicting forward in time and the second half is prediction backward.

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ \hline r_5 \\ r_6 \\ r_7 \\ r_8 \end{bmatrix} = \begin{bmatrix} y_3 & y_2 & y_1 \\ y_4 & y_3 & y_2 \\ y_5 & y_4 & y_3 \\ y_6 & y_5 & y_4 \\ \hline y_6 & y_5 & y_4 \\ y_5 & y_4 & y_3 \\ y_4 & y_3 & y_2 \\ y_3 & y_2 & y_1 \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} \tag{6.29}$$

In an (x, y) space, you might also expect rotational symmetry. In an (x, t) space, dips are preserved if t-axis reversal is accompanied by x-axis reversal. Many of my research codes include these symmetries, but I excluded them here. Nowhere did I prove that the symmetry presumption made a large difference, but in Fortran it always adds noticeable clutter, expanding the residual to a two-component *residual array*, which as usual needs to have three copies allocated.

Where a data sample grows exponentially towards the boundary, I expect that extrapolated data would diverge too. You can force it to go to zero (or any specified value) at some distance from the body of the known data. To do so, surround the body of data by missing data and surround that by specification of enough zeros. Enough is defined by the filter length.

6.6 SEABEAM: FILLING THE EMPTY BINS

Figure 1.7 shows a day's worth of data[2] collected at sea by **SeaBeam**, an apparatus for measuring water depth both directly under a ship, and somewhat off to the sides of the ship's track. The data is measurements of depth $h(x, y)$ at miscellaneous locations in the (x, y)-plane. The locations are scattered about, according to various aspects

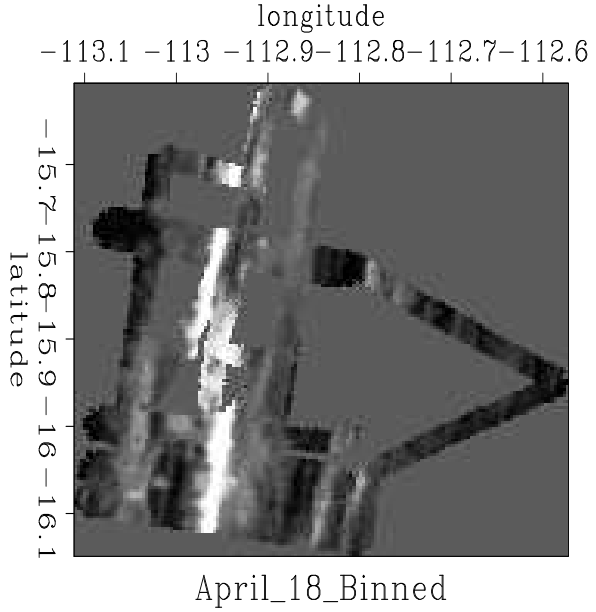


Figure 6.7: Depth of the ocean under ship tracks. mda-seabin90 [ER]

of the ship's navigation and the geometry of the SeaBeam sonic antenna. Figure 1.7 was made by binning with `bin2()` on page 13 and equation (1.15). The spatial spectra of the noise in the data could be estimated where tracks cross over themselves. More interesting are the empty mesh locations where no data is recorded. Here I left empty locations with a background value equal to the mean depth $\bar{h}$. After you study the water-depth map for a while, you come to realize that at many locations, your human intuition suggests some obvious extensions of the **ocean floor ridges**

[2]I'd like to thank Alistair Harding for this interesting data set named April 18.

(which are white) and we seek mathematical methods to guess what humans can guess, namely, a background depth $\bar{h}(x, y)$ more plausible than $\bar{h}$. Theoretically this amounts to finding and using the *model spectrum*. What we actually do is use a Prediction Error Filter (PEF) to fill in the missing data. Supposing the PEF to be the Laplacian operator ∇^2 and using module `mis2` on page 122 led to the result in Figure 1.8.

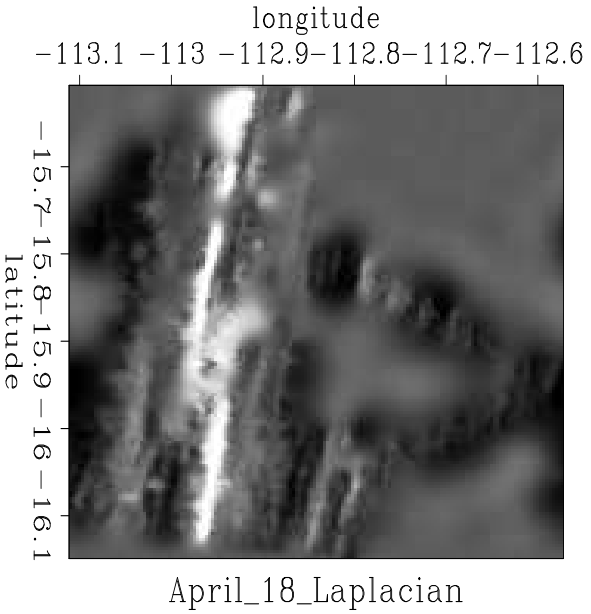


Figure 6.8: Depth of the ocean under ship tracks interpolated by minimizing the energy of the Laplacian of the water depth. mda-sealap90 [ER]

The problem we see with the Laplacian operator as an interpolator is that it smears things uniformly in all directions. From Figure 1.8 we see that we need an anisotropic interpolation oriented along the regional trends. To find a better PEF we apply module `pef` on page 19 to the image of Figure 1.7. After binning the data and finding this PEF, we do a second stage of linear-least-squares optimization with `mis2` on page 122, as we did for Figure 1.6, and we obtain the pleasing result in Figure 1.9.

6.7 BOTH MISSING DATA AND UNKNOWN FILTER

Recall the missing-data figures beginning with Figure 3.1. There the filters were taken as known, and the only unknowns were the missing data. Now, instead of having a predetermined filter, we will solve for the filter along with the missing data. The principle we will use is that the output power is minimized while the filter is constrained to have one nonzero coefficient (else all the coefficients would go to zero). We will look first at some results and then see how they were found.

In Figure 1.10 the filter is constrained to be of the form $(1, a_1, a_2)$. The result is pleasing in that the interpolated traces have the same general character as the given values. The filter came out slightly different from the $(1, 0, -1)$ that I guessed and tried in Figure 3.5. Curiously, constraining the filter to be of the form $(a_{-2}, a_{-1}, 1)$

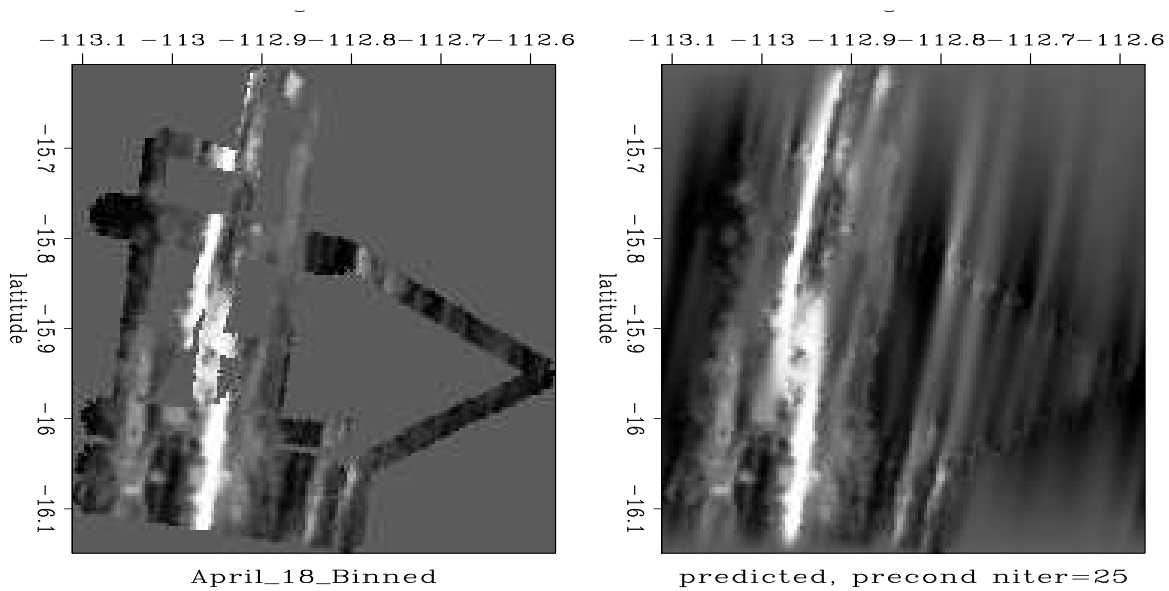
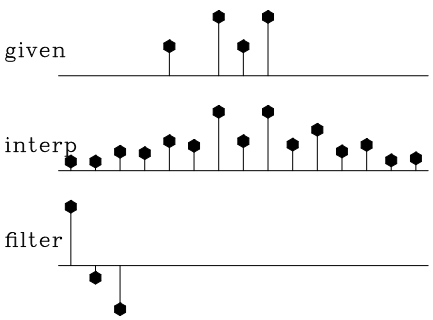


Figure 6.9: Depth of the ocean interpolated with a PEF. mda-seapef90 [ER]

Figure 6.10: Top is known data. Middle includes the interpolated values. Bottom is the filter with the leftmost point constrained to be unity and other points chosen to minimize output power. mda-misif90 [ER]



in Figure 1.11 yields the same interpolated missing data as in Figure 1.10. I understand that the sum squared of the coefficients of $A(Z)P(Z)$ is the same as that of $A(1/Z)P(Z)$, but I do not see why that would imply the same interpolated data; never the less, it seems to.

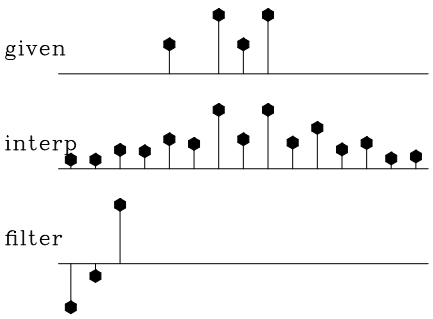


Figure 6.11: The filter here had its rightmost point constrained to be unity—i.e., this filtering amounts to backward prediction. The interpolated data seems to be identical to that of forward prediction. mda-backwards90 [ER]

6.7.1 Objections to interpolation error

In any data interpolation or extrapolation, we want the extended data to behave like the original data. And, in regions where there is no observed data, the extrapolated data should drop away in a fashion consistent with its **spectrum** determined from the known region.

My basic idea is that the spectrum of the missing data should match that of the known data. This is is the idea that the spectrum should be unchanging from a known region to an unknown region. A technical word to express the idea of spectra not changing is "**stationary**." This happens with the PEF (one-sided filter) because its spectrum tends to the inverse of that of the known data while that of the unknown data tends to the inverse of that of the PEF. Thus the spectrum of the missing data is the "inverse of the inverse" of the spectrum of the known. The PEF enables us to fill in the missing area with the spectral shape of the known area. (In regions far away or unpredictable, the spectral shape may be the same, but the energy drops to zero.)

On the other hand, the **interpolation-error filter**, a filter like $(a_{-2}, a_{-1}, 1, a_1, a_2)$, should fail to do the job because it has the wrong spectrum. Its spectrum tends to the inverse of the input spectrum (I am stating this fact without proof), so the spectrum of the IE filter tends to the inverse *squared* of the input spectrum.

To confirm and show these concepts, I prepared synthetic data consisting of a fragment of a damped exponential, and off to one side of it an impulse function. Most of the energy is in the damped exponential. Figure 1.12 shows that the spectrum and the extended data are about what we would expect. From the extrapolated data, it is impossible to see where the given data ends. For comparison, I prepared Figure 1.13. It is the same as Figure 1.12, except that the filter is constrained in the middle. Notice that the extended data does *not* have the spectrum of the given data—the

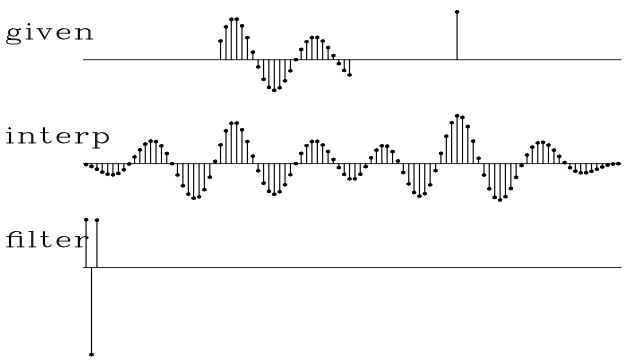


Figure 6.12: Top is synthetic data with missing portions. Middle includes the interpolated values. Bottom is the filter, a *prediction-error* filter which may look symmetric but is not quite. mda-exp90 [ER]

wavelength is much shorter. The boundary between real data and extended data is not nearly as well hidden as in Figure 1.12.

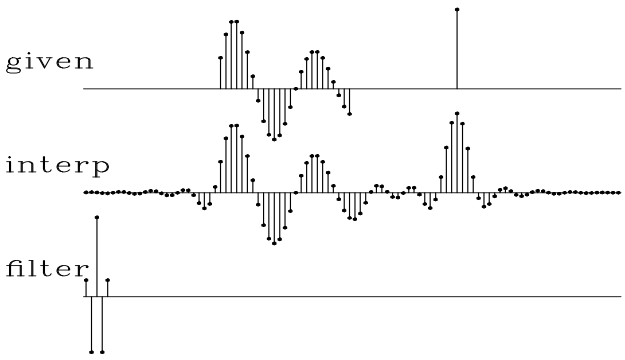


Figure 6.13: Top is the same synthetic data. Middle includes the interpolated values. Bottom is the filter, an *interpolation-error* filter. mda-center90 [ER]

6.7.2 Packing both missing data and filter into a vector

Now let us examine the theory and coding behind the above examples. Define a roughening filter $A(\omega)$ and a data signal $Y(\omega)$ at some stage of interpolation. The fitting goal is $0 \approx A(\omega)Y(\omega)$ where the filter $A(\omega)$ has at least one time-domain coefficient constrained to be nonzero and the data contains both known and missing values. Think of perturbations ΔA and ΔY. We neglect the nonlinear term $\Delta A\, \Delta Y$ as follows:

$$0 \;\approx\; (A + \Delta A)(Y + \Delta Y) \tag{6.30}$$

$$0 \;\approx\; A\,\Delta Y + Y\,\Delta A + AY + \Delta A\,\Delta Y \tag{6.31}$$

$$0 \;\approx\; A\,\Delta Y + Y\,\Delta A + AY \tag{6.32}$$

Linearizing a nonlinear fitting goal such as (1.32) seems reasonable. I linearized this way to first obtain the results of this chapter. Never-the-less, I find linearizing is a dangerous procedure that quickly leads to many problems. In reality, the term $\Delta A\, \Delta Y$ might not actually be small and the solution might not converge. You don't know how small is small, because these are not scalars but operators. The answer will depend on where you start from. Strange things happen. You can feel a sense

of frustration. There is good news: Two stage linear least squares offers relief. You should first use linear least squares to get close to your answer. Generally, when finding the filter A we have plenty of regression equations, so we can afford to neglect the ones that depend upon unknown data. Then you can fix the filter and in a second stage, find the missing data. If you don't have enough regression equations because your data is irregularly distributed, then you can use binning. Still not enough? Try coarser bins. The point is that nonlinear solvers will not work unless you first get sufficiently close to the solution, and the way you get close is by arranging first to solve a sensible (though approximate) linearized problem. Only as a last resort, after you have gotten as near as you can, should you use the nonlinear least-squares techniques described below.

Let us use matrix algebraic notation to rewrite the fitting goals (1.32). For this we need mask matrices (diagonal matrices with ones on the diagonal where variables are free and zeros where they are constrained i.e., where $\Delta a_i = 0$ and $\Delta y_i = 0$). The free-mask matrix for missing data is denoted $\mathbf{J}$ and that for the PE filter is $\mathbf{K}$. The fitting goal (1.32) becomes

$$\mathbf{0} \quad \approx \quad \mathbf{AJ}\Delta\mathbf{y} + \mathbf{YK}\Delta\mathbf{a} + (\mathbf{Ay} \text{ or } \mathbf{Ya}) \tag{6.33}$$

or

$$\mathbf{0} \quad \approx \quad \begin{bmatrix} \mathbf{AJ} & \mathbf{YK} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{y} \\ \Delta\mathbf{a} \end{bmatrix} + \mathbf{Ay} \tag{6.34}$$

For a 3-term filter and a 7-point data signal, the fitting goal (1.33) becomes

$$\begin{bmatrix}
a_0 & . & . & . & . & . & . & y_0 & . & . \\
a_1 & a_0 & . & . & . & . & . & y_1 & y_0 & . \\
a_2 & a_1 & a_0 & . & . & . & . & y_2 & y_1 & y_0 \\
. & a_2 & a_1 & a_0 & . & . & . & y_3 & y_2 & y_1 \\
. & . & a_2 & a_1 & a_0 & . & . & y_4 & y_3 & y_2 \\
. & . & . & a_2 & a_1 & a_0 & . & y_5 & y_4 & y_3 \\
. & . & . & . & a_2 & a_1 & a_0 & y_6 & y_5 & y_4 \\
. & . & . & . & . & a_2 & a_1 & . & y_6 & y_5 \\
. & . & . & . & . & . & a_2 & . & . & y_6 \\
. & . & . & . & . & . & . & . & . & .
\end{bmatrix}
\begin{bmatrix} \mathbf{J} & \mathbf{0} \\ \mathbf{0} & \mathbf{K} \end{bmatrix}
\begin{bmatrix} \Delta y_0 \\ \Delta y_1 \\ \Delta y_2 \\ \Delta y_3 \\ \Delta y_4 \\ \Delta y_5 \\ \Delta y_6 \\ \hline \Delta a_0 \\ \Delta a_1 \\ \Delta a_2 \end{bmatrix}
+
\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ 0 \end{bmatrix}
\approx \mathbf{0} \tag{6.35}$$

where $\mathbf{r} = \mathbf{Ay}$. In other words, r_t is the convolution of a_t with y_t, namely, $r_0 = y_0 a_0$ and $r_1 = y_0 a_1 + y_1 a_0$, etc. To optimize this fitting goal we first initialize $\mathbf{a} = (1, 0, 0, \cdots)$ and then put zeros in for missing data in $\mathbf{y}$. Then we iterate over equations (1.36) to (1.40).

$$\mathbf{r} \quad \longleftarrow \quad \mathbf{Ay} \tag{6.36}$$

$$\begin{bmatrix} \Delta\mathbf{y} \\ \Delta\mathbf{a} \end{bmatrix} \quad \longleftarrow \quad \begin{bmatrix} \mathbf{J'A'} \\ \mathbf{K'Y'} \end{bmatrix} \mathbf{r} \tag{6.37}$$

$$\Delta\mathbf{r} \quad \longleftarrow \quad \begin{bmatrix} \mathbf{AJ} & \mathbf{YK} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{y} \\ \Delta\mathbf{a} \end{bmatrix} \tag{6.38}$$

$$\mathbf{y} \longleftarrow \text{cgstep}(\mathbf{y}, \Delta\mathbf{y}) \tag{6.39}$$
$$\mathbf{a} \longleftarrow \text{cgstep}(\mathbf{a}, \Delta\mathbf{a}) \tag{6.40}$$

This is the same idea as all the linear fitting goals we have been solving, except that now we recompute the residual $\mathbf{r}$ inside the iteration loop so that as convergence is achieved (*if* it is achieved), the neglected nonlinear term $\Delta A \Delta Y$ tends to zero.

I no longer exhibit the nonlinear solver `missif` until I find an example where it produces noticibly better results than multistage linear-least squares.

6.8 LEVELED INVERSE INTERPOLATION

Eighteenth- and nineteenth- century mathematics literature gives us many methods of interpolating functions. These classical methods are generally based on polynomials. The user specifies some order of polynomial and the theory works out the coefficients. Today our interest is in both interpolating and extrapolating wavefields (which are solutions to low order differential equations) and we use methods that are much better behaved than polynomials when extrapolating data, methods which behave acceptably when faced with contradictory data values, and methods which also apply in two and three dimensions.

In Chapter 3, subroutine `invint1()` on page 119 solved the problem of inverse linear interpolation, which is, given scattered data points, to find a function on a uniform mesh from which linear interpolation gives the scattered data points. To cope with regions having no data points, the subroutine requires an input roughening filter. This is a bit like specifying a differential equation to be satisfied between the data points. The question is, how should we choose a roughening filter? The importance of the roughening filter grows as the data gets sparser or as the mesh is refined.

Figures 1.10-1.13 suggest that the choice of the roughening filter need not be subjective, nor a priori, but that the prediction-error filter (PEF) is the ideal roughening filter. Spectrally, the PEF tends to the inverse of its input hence its output tends to be "level". Missing data that is interpolated with this "leveler" tends to have the spectrum of given data.

6.8.1 Test results for leveled inverse interpolation

Figures 1.14 and 1.15 show the same example as in Figures 3.11 and 3.12. What is new here is that the proper PEF is not given but is determined from the data. Figure 1.14 was made with a three-coefficient filter $(1, a_1, a_2)$ and Figure 1.15 was made with a five-coefficient filter $(1, a_1, a_2, a_3, a_4)$. The main difference in the figures is where the data is sparse. The three-coefficient filter can fit only a single sinusoid and it does

that. The five-coefficient filter can fit only two sinusoids and it does something like that, a high-frequency sinusoid on the left and a lower one on the right. The data points in Figures 3.11, 1.14 and 1.15 are samples from a sinusoid.

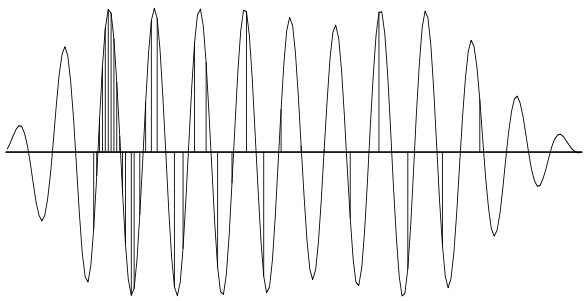


Figure 6.14: Interpolating with a three-term filter. The interpolated signal is fairly monofrequency. mda-subsine390 [ER]

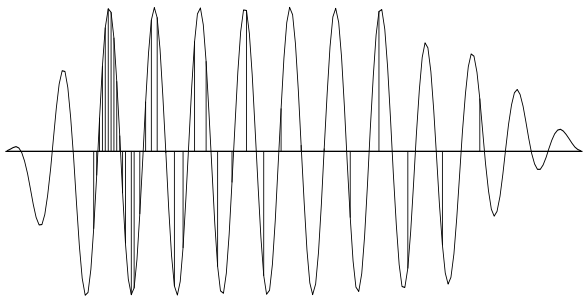


Figure 6.15: Interpolating with a five term filter. A low frequency component grows towards the right. mda-subsine590 [ER]

Comparing Figures 3.11 and 3.12 to Figures 1.14 and 1.15 we conclude that by finding and imposing the prediction-error filter while finding the model space, we have interpolated beyond **aliasing** in data space.

6.8.2 Analysis for leveled inverse interpolation

Here we see how the interpolation beyond aliasing was done. The first "statement of wishes" is that the observational data $\mathbf{d}$ should result from a linear interpolation $\mathbf{L}$ of the uniformly sampled model space $\mathbf{m}$; that is, $\mathbf{0} \approx \mathbf{Lm} - \mathbf{d}$. Expressing this as a change $\Delta\mathbf{m}$ gives the fitting goal in terms of the model change, $\mathbf{0} \approx \mathbf{L}\Delta\mathbf{m} + (\mathbf{Lm} - \mathbf{d}) = \mathbf{L}\Delta\mathbf{m} + \mathbf{r}$. The second wish is really an assertion that a good way to find missing parts of a function (the model space) is to solve for the function and its PEF at the same time. We are merging the fitting goal (3.21) for irregularly sampled data with the fitting goal (1.35) for finding the prediction-error filter.

$$\mathbf{0} \approx \mathbf{L}\Delta\mathbf{m} + (\mathbf{Lm} - \mathbf{d}) \tag{6.41}$$

$$\mathbf{0} \approx \mathbf{A}\Delta\mathbf{m} + \mathbf{MK}\Delta\mathbf{a} + (\mathbf{Am} \text{ or } \mathbf{Ma}) \tag{6.42}$$

Writing this out in full for 3 data points and 6 model values on a uniform mesh and a PEF of 3 terms, we have

$$\left[\begin{array}{cccccc|ccc} .8 & .2 & . & . & . & . & & & \\ . & . & 1 & . & . & . & & & \\ . & . & . & . & .5 & .5 & & & \\ \hline a_0 & . & . & . & . & . & m_0 & . & . \\ a_1 & a_0 & . & . & . & . & m_1 & m_0 & . \\ a_2 & a_1 & a_0 & . & . & . & m_2 & m_1 & m_0 \\ . & a_2 & a_1 & a_0 & . & . & m_3 & m_2 & m_1 \\ . & . & a_2 & a_1 & a_0 & . & m_4 & m_3 & m_2 \\ . & . & . & a_2 & a_1 & a_0 & m_5 & m_4 & m_3 \\ . & . & . & . & a_2 & a_1 & . & m_5 & m_4 \\ . & . & . & . & . & a_2 & . & . & m_5 \\ . & . & . & . & . & . & . & . & . \end{array}\right] \left[\begin{array}{cc} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{K} \end{array}\right] \left[\begin{array}{c} \Delta m_0 \\ \Delta m_1 \\ \Delta m_2 \\ \Delta m_3 \\ \Delta m_4 \\ \Delta m_5 \\ \Delta m_6 \\ \hline \Delta a_0 \\ \Delta a_1 \\ \Delta a_2 \end{array}\right] + \left[\begin{array}{c} r_{d0} \\ r_{d1} \\ r_{d2} \\ \hline r_{m0} \\ r_{m1} \\ r_{m2} \\ r_{m3} \\ r_{m4} \\ r_{m5} \\ r_{m6} \\ r_{m7} \\ 0 \end{array}\right] \approx \mathbf{0} \tag{6.43}$$

where r_m is the convolution of the filter a_t and the model m_t, where r_d is the data misfit $\mathbf{r} = \mathbf{Lm} - \mathbf{d}$, and where $\mathbf{K}$ was defined in equation (1.11).

Before you begin to use this **nonlinear fitting goal**, you need some starting guesses for $\mathbf{m}$ and $\mathbf{a}$. The guess $\mathbf{m} = 0$ is satisfactory (as explained later). For the first guess of the filter, I suggest you load it up with $\mathbf{a} = (1, -2, 1)$ as I did for the examples here.

6.8.3 Seabeam: theory to practice

I provide here a more fundamental theory for dealing with the Seabeam data. I originally approached the data in this more fundamental way, but with time, I realized that I paid a high price in code complexity, computational speed, and reliability. The basic problem is that the elegant theory requires a good starting model which can only come from the linearized theory. I briefly recount the experience here, because the fundamental theory is interesting and because in other applications, you will face the challenge of sorting out the fundamental features from the essential features.

The linear-interpolation operator carries us from a uniform mesh to irregularly distributed data. Fundamentally we seek to solve the inverse problem to go the other direction. A nonlinear approach to filling in the missing data is suggested by the one-dimensional examples in Figures 1.14–1.15, where the PEF and the missing data are estimated simultaneously. The nonlinear approach has the advantage that it allows for completely arbitrary data positioning, whereas the two-stage linear approach forces the data to be on a uniform mesh and requires there not be too many empty mesh locations.

For addressing the nonlinear problem, we follow the approach we used in one dimension. The idea is to fit the data with linear interpolation $\mathbf{0} \approx \mathbf{Lh} - \mathbf{d}$ as well as smooth the map by minimizing the energy output of its own PEF, $\mathbf{0} \approx \mathbf{Hu} = \mathbf{Uh}$.

(Details of the mathematics, fitting problem, and method of solution led to the one-dimensional equations (1.41) and (1.42).) As in 1-D, linearizing and introducing constraints brings us to

$$\begin{array}{rcrcl} \mathbf{0} & \approx & \mathbf{r}_d & = & \mathbf{L}\Delta\mathbf{h} + (\mathbf{Lh} - \mathbf{d}) \\ \mathbf{0} & \approx & \mathbf{r}_m & = & \mathbf{U}\ \Delta\mathbf{h} + \mathbf{HK}\ \Delta\mathbf{u} + (\mathbf{Uh} \text{ or } \mathbf{Hu}) \end{array} \tag{6.44}$$

where $\mathbf{d}$ is the data $d(x, y)$, $\mathbf{L}$ is linear interpolation, $\mathbf{h}$ is the modeled altitude $h(x, y)$, $\mathbf{H}$ is convolution with altitude, $\mathbf{u}$ is a PEF, $\mathbf{U}$ is the operator of convolution with a PEF, and $\mathbf{K}$ is an identity matrix but with a zero at the PEF's "1".

I have had considerable experience solving (1.44) and can report that bin filling is easier and works much more quickly and reliably. Eventually I realized that the best way to start the nonlinear iteration (1.44) is with the final result of bin filling. Then I learned that the extra complexity of the nonlinear iteration (1.44) offers little apparent improvement to the quality of the SeaBeam result. (This is not to say that we should not try more variations on the idea).

Not only did I find the binning method faster, but I found it to be *much* faster (compare a minute to an hour). The reasons for being faster (most important first) are,

1. Binning reduces the amount of data handled in each iteration by a factor of the average number of points per bin.

2. The 2-D linear interpolation operator adds many operations per data point.

3. The formulation (1.44) seems to require more iterations.

(Parenthetically, I later found that helix preconditioning speeds the Seabeam interpolation from minutes to seconds.)

The most serious criticisim of the nonlinear approach is that it does not free us from the linearized approaches. We need them to get a "close enough" starting solution to the nonlinear problem. I learned that the iteration (1.44), like most nonlinear sequences, behaves unexpectedly and badly when you start too far from the desired solution. For example, I often began from the assumed PEF being a Laplacian and the original map being fit from that. Oddly, from this starting location I sometimes found myself stuck. The iteration (1.44) would not move towards the map we humans consider a better one.

Having said all those bad things about iteration (1.44), I must hasten to add that with a different type of data set, you might find the results of (1.44) to be significantly better.

6.8.4 Risky ways to do nonlinear optimization

I have noticed that some geophysicists have adopted a risky method of nonlinear optimization, which is not advocated in the professional optimization literature. This risky method is to linearize a goal (with a multiparameter model space), then optimize the linearized goal, then relinearize, etc. The safer method is to relinearize after each step of CD.

An instructive example I learned about many years ago was earthquake epicenter location. Model space is latitude, longitude, and origin time. When people added a new variable, the depth, the solutions went wild until they learned to restrict the depth to zero until the other three parameters were stabilized. Apparently the instability stems from the fact that depth and origin time affect distant receivers in a similar way.

6.9 MULTIVARIATE SPECTRUM

A common **spectrum** is the Fourier spectrum. More fundamentally, a spectrum is a decomposition of a model space or data space into components. The components are in some sense independent; more specifically, the components are orthogonal to one another. Another well-known spectrum is provided by eigenvectors and eigenvalues. In statistical signal processing we handle a third type of spectrum, the multivariate spectrum.

Working in an optimization problem, we begin from residuals between theory and practice. These residuals can be scaled to make new optimization residuals before we start minimizing their energy. What scaling should we use? The scaling can be a simple weighting function or a filter. A filter is simply a weighting function in Fourier space.

The basic idea of common sense, which also comes to us as results proven by Gauss or from the theory of statistical signal processing, is this: The optimization residuals should be roughly of equal scale. This makes sense because squaring magnifies scale, and anything small will be ignored while anything large will dominate. Scaling optimization residuals to be in a common range makes them all equally influential on the final solution. Not only should optimization residuals be of like scale in physical space, they should be of like scale in Fourier space or eigenvector space, or any other space that we might use to represent the optimization residuals. This implies that the optimization residuals should be uncorrelated. If the optimization residuals were correlated, they would have a spectrum that was not white. Not white means of differing sizes in Fourier space. Residuals should be the same size as one another in physical space, likewise in Fourier space. Thus the optimization residuals should be orthogonal and of unit scale, much like Fourier components or as eigenvectors are orthonormal.

Let us approach the problem backwards. Suppose we have two random variables that we take to be the ideal optimization residuals x_1 and x_2. In reality the two may be few or trillions. In the language of statistics, the optimization residuals are expected to have zero mean, an idea that is formalized by writing $E(x_1) = 0$ and $E(x_2) = 0$. Likewise these ideal optimization residuals have equal energy, $E(x_1^2) = 1$ and $E(x_2^2) = 1$. Finally, these two optimization residuals are uncorrelated, a condition which is written as $E(x_1 x_2) = 0$. The expectation symbol $E()$ is like a summation over many instances of the random variable.

Now suppose there exists a transformation $\mathbf{B}$ from these ideal optimization residuals to two experimental residuals y_1 and y_2, say $\mathbf{y} = \mathbf{B}\mathbf{x}$ where

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{6.45}$$

The experimental residuals y_1 and y_2 are likely to be neither orthogonal nor equal in energy. From the column vector $\mathbf{y}$, the experimenter can form a square matrix. Let us also allow the experimenter to write the symbol $E()$ to denote summation over many trials or over many sections of data, ranges over time or space, over soundings or over receiver locations. The experimenter writes

$$\mathbf{R} = E(\mathbf{y}\mathbf{y}') \tag{6.46}$$
$$\mathbf{R} = E(\mathbf{B}\mathbf{x}\mathbf{x}'\mathbf{B}') \tag{6.47}$$

Given a random variable r, the expectation of $2r$ is simply $E(2r) = 2E(r)$. The $E()$ symbol is a summation on random variables, but constants like the coefficients of $\mathbf{B}$ pass right through it. Thus,

$$\mathbf{R} = \mathbf{B}\, E(\mathbf{x}\mathbf{x}')\, \mathbf{B}' \tag{6.48}$$
$$\mathbf{R} = \mathbf{B}\, E\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} x_1 & x_2 \end{bmatrix} \right) \mathbf{B}' \tag{6.49}$$
$$\mathbf{R} = \mathbf{B} \begin{bmatrix} E(x_1 x_1) & E(x_1 x_2) \\ E(x_2 x_1) & E(x_2 x_2) \end{bmatrix} \mathbf{B}' \tag{6.50}$$
$$\mathbf{R} = \mathbf{B}\mathbf{B}' \tag{6.51}$$

Given a matrix $\mathbf{R}$, there is a simple well-known method called the **Cholesky factorization** method that will factor $\mathbf{R}$ into two parts like $\mathbf{B}$ and $\mathbf{B}'$. The method creates for us either an upper or a lower triangular matrix (our choice) for $\mathbf{B}$. You can easily reinvent the Cholesky method if you multiply the symbols for two triangular matrices like $\mathbf{B}$ and $\mathbf{B}'$ and notice the procedure that works backwards from $\mathbf{R}$ to $\mathbf{B}$. The experimenter seeks not $\mathbf{B}$, however, but its inverse, the matrix that takes us from the experimental residuals to the ideal optimization residuals that are uncorrelated and of equal energies. The Cholesky factorization costs N^3 computations, which is about the same as the cost of the matrix inversion of $\mathbf{R}$ or $\mathbf{B}$. For geophysical maps and other functions on Cartesian spaces, the Prediction Error Filter (PEF)

accomplishes the same general goal and has the advantage that we have already learned how to perform the operation using operators instead of matrices.

The multivariate spectrum of experimental residuals $\mathbf{y}$ is the matrix $\mathbf{R} = E(\mathbf{y}\mathbf{y}')$. For optimum model finding, the experimental residuals (squared) should be weighted inversely (matrix inverse) by their multivariate spectrum.

6.9.1 What should we optimize?

Least-squares problems often present themselves as fitting goals:

$$\mathbf{0} \approx \mathbf{F}\mathbf{m} - \mathbf{d} \tag{6.52}$$

$$\mathbf{0} \approx \mathbf{m} \tag{6.53}$$

To balance our possibly contradictory goals we need weighting functions. The quadratic form that we should minimize is

$$\min_{m} \quad (\mathbf{F}\mathbf{m} - \mathbf{d})' \mathbf{A}_n' \mathbf{A}_n (\mathbf{F}\mathbf{m} - \mathbf{d}) + \mathbf{m}' \mathbf{A}_m' \mathbf{A}_m \mathbf{m} \tag{6.54}$$

where $\mathbf{A}_n'\mathbf{A}_n$ is the inverse multivariate spectrum of the noise (data-space residuals) and $\mathbf{A}_m'\mathbf{A}_m$ is the inverse multivariate spectrum of the model. In other words, $\mathbf{A}_n$ is a leveler on the data fitting error and $\mathbf{A}_m$ is a leveler on the model. There is a curious unresolved issue: What is the most suitable constant scaling ratio of $\mathbf{A}_n$ to $\mathbf{A}_m$?

6.9.2 Confusing terminology for data covariance

Confusion often stems from the mean of the data $E(\mathbf{d})$.

An experimentalist would naturally believe that the expectation of the data is solely a function of the data, that it can be estimated by averaging data.

On the other hand, a theoretician's idea of the expectation of the observational data $E(\mathbf{d})$ is that it is the theoretical data $\mathbf{F}\mathbf{m}$, that the expectation of the data $E(\mathbf{d}) = \mathbf{F}\mathbf{m}$ is a function of the model. The theoretician thinks this way because of the idea of noise $\mathbf{n} = \mathbf{F}\mathbf{m} - \mathbf{d}$ as having zero mean.

Seismological data is highly complex but also highly reproducible. In studies like seismology, the world is deterministic but more complicated than our ability to model. Thus, as a practical matter, the discrepancy between observational data and theoretical data is more realistically attributed to the theoretical data. It is not adequately modeled and computed.

This superficial difference in viewpoint becomes submerged to a more subtle level by statistical textbooks that usually define weighting functions in terms of variances instead of spectra. This is particularly confusing with the noise spectrum $\mathbf{A}_n'\mathbf{A}_n$. It

is often referred to as the "**data covariance**" defined as $E[(\mathbf{d}-E(\mathbf{d}))(\mathbf{d}-E(\mathbf{d}))']$. Clearly, the noise spectrum is the same as the data covariance only if we accept the theoretician's definition that $E(\mathbf{d})=\mathbf{Fm}$.

There is no ambiguity and no argument if we drop the word "variance" and use the word "spectrum". Thus, (1) the "inverse **noise spectrum**" is the appropriate weighting for data-space residuals; and (2) the "inverse **model spectrum**" is the appropriate model-space weighting. Theoretical expositions generally require these spectra to be given as "**prior information**." In this book we see how, when the model space is a map, we can solve for the "prior information" along with everything else.

The statistical words "covariance matrix" are suggestive and appealing, but I propose not to use them because of the ambiguity of $E(\mathbf{d})$. For example, we understand that people who say "data covariance" intend the "multivariate noise spectrum" but we cannot understand their meaning of "model covariance". They should intend the "multivariate model spectrum" but that implies that $E(\mathbf{m})=\mathbf{0}$, which seems wrong. Avoiding the word "covariance" avoids the problem.

Index

AGC, 15
aliasing, 28
autocorrelation, 11
automatic gain control, 15
autoregression, 1, 3

boundary module, 18
Burg, 7

Cholesky factorization, 32
color, 7
convolution
    two-dimensional, 9, 17

data covariance, 34
deconvolution, 6
dip-rejection filter, 10

FGDP, 7, 20
filter
    3-D prediction-error, 12
    dip-rejection, 10
    interpolation-error, 4, 24
    multidimensional, 9
    prediction, 4
    prediction-error, 4, 18
    two-dimensional, 9, 10
    unknown, 23
fitting goal
    nonlinear, 29, 31
format module, 13
free-mask matrix, 4

gdecon module, 5

Harding, Alistair, 21
hconest operator module, 16

IEI, 15
interpolation
    beyond aliasing, 28
    inverse, 27
interpolation-error filter, 4, 24
iteration count, 14

lagtable module, 12
leveled inverse interpolation, 27
linear interpolation instead of binning, 29

mask, 18
mask matrix, 26
misinput module, 19
missing data, 14, 16, 23, 25
model spectrum, 34
module
    `boundary`, out of bounds dependency, 18
    `format`, print out helix filter coefficients, 13
    `gdecon`, gapped decon filt, 5
    `lagtable`, constructing helix filter in 1-D, 2-D, and 3-D, 12
    `misinput`, mark bad regression equations, 19
    `pef`, estimate PEF on a helix, 19
multiple reflection, 2, 4

noise spectrum, 34
nonlinear fitting goal, 29
nonlinear optimization, 31

ocean floor ridges, 21
operator
    `hconest`, helix convolution, adjoint is the filter, 16
orthogonality, 8

PE filter, 7

PEF, 1, 7
PEF
    3D, 12
pef module, 19
prediction filter, 4
prediction-error filter, 1, 4, 7, 18
prediction-error filter
    3-D, 12
prior information, 34

quarry blast, 14

reversing a signal, 20

SeaBeam, 21
seabeam, 21
soil, 7
source waveform, 2
spectrum, 24, 31
spectrum
    white 2-D, 11
stabilize, 2
Stanford University, 14
stationarity, 1
stationary, 24
symmetry in time, 20

t squared, 15
time-series analysis, 1
two-dimensional filter, 10
two-stage
    least squares, 16
Two-stage linear least squares, 16

water bottom, 2
weighting function, 1, 15
white, 7, 8
white 2-D spectrum, 11
white spectrum, 7
whiteness
    multidimensional, 10