Chapter 7

Multidimensional autoregression

Occam's razor says we should try understand the world by the simplest explanation. So, how do we decompose a complicated thing into its essential parts? That's far too difficult a question, but the word "covariance" points the way. If things are found statistically connected (covary), the many might be explainable by a few. For example a one-dimensional waveform can excite a wave equation filling a 3-D space. The values in that space will have a lot of covariance. In this chapter we take multidimensional spaces full of numbers and answer the question, "what causal differential (difference) equation might have created these numbers?" Our answer here, an autoregressive filter, does the job imperfectly, but it is a big step away from complete ignorance. As the book progresses we find three kinds of uses: (1) filling in missing data and uncontrolled parts of models, (2) preparing residuals for data fitting, (3) providing "prior" models for preconditioning and estimation.

Recall that residuals (and preconditioning variables) should be Independent, and Identically Distributed (IID). In practice the "ID" means all residuals should have the same variance, and the preceding "I" means likewise in Fourier space (whiteness). This is the "I" chapter. Conceptually we might jump in and out of Fourier space, but here we learn processes in physical space that whiten in Fourier space. In earlier chapters we transformed from a physical space to something more like an IID space when we said, "Topography is smooth, so let us estimate and view instead its derivative." In this chapter we go beyond roughening with a guessed derivative.

The branch of mathematics introduced here is young. Physicists seem to know nothing of it, perhaps because it begins with time not being a continuous variable. About 100 years ago people looked at market prices and wondered why they varied from day to day. To try to make money from the market fluctuations they schemed to try to predict prices. That is a good place to begin. The subject is known as "**time-series analysis**." In this chapter we define the ***autoregression*** filter, also known as the **prediction-error filter** (**PEF**). It gathers statistics for us. It gathers not the autocorrelation or the spectrum directly but it gathers them indirectly as the inverse of the amplitude spectrum of its input. Although time-series analysis is a one dimensional study, we naturally use the helix to broaden it to multidimensional space. The PEF leads us to the "inverse-covariance matrix" of statistical estimation theory. Theoreticians tell us we need this before we can properly find a solution. Here we go after it.

7.0.11 Time domain versus frequency domain

In the simplest applications, solutions can be most easily found in the frequency domain. When complications arise, it becomes necessary to use time and space domains, where we may cope with boundaries, scale by material properties, convolve differential operators, and apply statistical weighting functions and filters.

Recall Vesuvius in Chapter 2. We solved for altitude using only the phase of the data. (The given data was in (ω_0, x, y)-space.) There was a marvelously fast solving method in the (k_x, k_y) Fourier space. It worked so long as we were satisfied that each data value in (x, y) was as good as any other. But when we recognized data quality varied with location in (x, y) in proportion to the amplitude of the signal, we needed a **weighting function** in (x, y). Without it we had a limited quality solution, perhaps a good starting solution for using weights and finite differences in (x, y).

Recall some of the "magic tricks" we did in Chapter 4 with spectral factorization, finding the impulse response of the sun, blind deconvolution, and others. There we required a full mesh of regularly sampled data. Here we allow in the mesh missing information somewhat arbitrarily distributed. Being out of fourier space, in the physical domain we can gather spectral information on small grids, irregularly shaped.

This is a general fact of science. Homogeneity in time and space enables Fourier methods. Fourier methods give insight because they may be roughly correct in real life. But when we have space variable coefficients, either physically, as seismic velocity, or statistically, as with Vesuvius, we are back to solving problems in physical space. Seismology has the delightful aspect that the earth is unchanging in time, so Fourier analysis is generally applicable for physical modeling, but like the space axes, statistical qualities of data will be variable with time so when reconstructing models from data we are thrown out of the frequency domain.

7.1 SOURCE WAVEFORM, MULTIPLE REFLECTIONS

Deep water multiple reflection[1] is a simple geometry where the fourier formulation readily converts to the the physical domain. There are two unknown waveforms, the source waveform $S(\omega)$ and the ocean-floor reflection $F(\omega)$ which may include the upper mud layers. 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 reflection coefficient at the water surface, and the ocean floor again. Thus the observations $P(\omega)$ and $M(\omega)$ as functions of the physical parameters $S(\omega)$ and $F(\omega)$ are:

$$P(\omega) = S(\omega)\, F(\omega) \tag{7.1}$$

$$M(\omega) = -S(\omega)\, F(\omega)^2 \tag{7.2}$$

Algebraically the solutions of equations (7.1) and (7.2) are:

$$F(\omega) = -M(\omega)/P(\omega) \tag{7.3}$$

$$S(\omega) = -P(\omega)^2/M(\omega) \tag{7.4}$$

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

These solutions can be computed in the Fourier domain by simple division. The difficulty is that the divisors in equations (7.3) and (7.4) can be zero, or small. This difficulty can be attacked by use of a positive number ϵ to **stabilize** it. For example, multiply equation (7.3) on top and bottom by $P(\omega)^*$ and add $\epsilon > 0$ to the denominator. This gives

$$F(\omega) \quad = \quad - \frac{M(\omega) P(\omega)^*}{P(\omega) P(\omega)^* + \epsilon} \tag{7.5}$$

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 (7.5) to

$$F(\omega) \quad = \quad - \frac{< M(\omega) P(\omega)^* >}{< P(\omega) P(\omega)^* >} \tag{7.6}$$

where instead of deciding a size for ϵ we need to decide how much smoothing. I find that smoothing has a simpler physical interpretation than choosing ϵ. 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 (7.3) as

$$0 \quad = \quad P(\omega) F(\omega) + M(\omega) \tag{7.7}$$

Recall the convolution operator from Chapter 1. Express the frequency functions in equation (7.7) as polynomials in $Z = e^{i\omega\Delta t}$. The coefficient of each power of Z gives one of the time-domain regression equations below. The column vector $\mathbf{f}$ contains the unknown sea-floor filter. The column vector $\mathbf{m}$ contains the multiple reflection. The matrix $\mathbf{P}$ has down-shifted columns of the primary reflection.

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \end{bmatrix} \quad = \quad \begin{bmatrix} p_1 & 0 & 0 \\ p_2 & p_1 & 0 \\ p_3 & p_2 & p_1 \\ p_4 & p_3 & p_2 \\ p_5 & p_4 & p_3 \\ p_6 & p_5 & p_4 \\ 0 & p_6 & p_5 \\ 0 & 0 & p_6 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} + \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \end{bmatrix} \tag{7.8}$$

7.2 TIME-SERIES AUTOREGRESSION

Historically, the earliest application of the ideas in this chapter came in the predictions of markets. Prediction of a signal from its past is called "**autoregression**", because a signal is regressed on itself hence "auto". The regression below finds for us the **prediction filter** (f_1, f_2). With it we have prediction of d_t from its past d_{t-1} and d_{t-2}.

$$\mathbf{0} \approx \mathbf{r} = \begin{bmatrix} d_1 & d_0 \\ d_2 & d_1 \\ d_3 & d_2 \\ d_4 & d_3 \\ d_5 & d_4 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} - \begin{bmatrix} d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} \tag{7.9}$$

(In practice, of course the system of equations would be much taller, and likely somewhat wider.) A typical row in the matrix (7.9) says that $d_{t+1} \approx d_t f_1 + d_{t-1} f_2$ hence the description of f as a "prediction" filter. The error in prediction defines the residual. Let the residual have opposite polarity and merge the column vector into the matrix getting

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \approx \mathbf{r} = \begin{bmatrix} d_2 & d_1 & d_0 \\ d_3 & d_2 & d_1 \\ d_4 & d_3 & d_2 \\ d_5 & d_4 & d_3 \\ d_6 & d_5 & d_4 \end{bmatrix} \begin{bmatrix} 1 \\ -f_1 \\ -f_2 \end{bmatrix} = \mathbf{Da} \tag{7.10}$$

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
$(a_{-m}, \cdots, a_{-2}, a_{-1}, 1, a_1, a_2, a_3, \cdots, a_n)$	**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 $a_0 = 1$).

$$\mathbf{K} = \begin{bmatrix} 0 & . & . \\ . & 1 & . \\ . & . & 1 \end{bmatrix} \tag{7.11}$$

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

(7.9) or (7.10) as

$$\mathbf{0} \approx \mathbf{r} = \begin{bmatrix} d_2 & d_1 & d_0 \\ d_3 & d_2 & d_1 \\ d_4 & d_3 & d_2 \\ d_5 & d_4 & d_3 \\ d_6 & d_5 & d_4 \end{bmatrix} \begin{bmatrix} 0 & \cdot & \cdot \\ \cdot & 1 & \cdot \\ \cdot & \cdot & 1 \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} \tag{7.12}$$

Let us move from this specific fitting goal to the general case. Let $\mathbf{D}$ be the matrix in equation 7.10. (Notice the similarity of the free-mask matrix $\mathbf{K}$ in this filter estimation application with the free-mask matrix $\mathbf{J}$ in missing data goal (3.3).) Rewriting equation 7.12 the fitting goal is

$$\mathbf{0} \approx \mathbf{Da} \tag{7.13}$$
$$\mathbf{0} \approx \mathbf{D(I - K + K)a} \tag{7.14}$$
$$\mathbf{0} \approx \mathbf{DKa + D(I - K)a} \tag{7.15}$$
$$\mathbf{0} \approx \mathbf{DKa + Da}_0 \tag{7.16}$$
$$\mathbf{0} \approx \mathbf{DKa + y} \tag{7.17}$$
$$\mathbf{0} \approx \mathbf{r} = \mathbf{DKa + r}_0 \tag{7.18}$$

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

$$\Delta\mathbf{a} \longleftarrow \mathbf{K}^*\mathbf{D}^*\,\mathbf{r} \tag{7.19}$$
$$\Delta\mathbf{r} \longleftarrow \mathbf{DK}\,\Delta\mathbf{a} \tag{7.20}$$

7.3 PREDICTION-ERROR FILTER OUTPUT IS WHITE

In Chapter 5 we learned that least squares residuals should be IID (Independent, Identically Distributed) which in practical terms means in both Fourier space and physical space they should have a uniform variance. Further, not only should residuals have the IID property, but we should choose a preconditioning transformation so that our unknowns have the same IID nature. For example echos get weaker in time. Multipying by some constant function of time such as t or t^2 will tend to uniformize (flatten) the variance with time. We should also transform to flatten them in Fourier space. Prediction-error filters (PEFs) accomplish that. Next we see why and how.

> Residuals and preconditioned models should be white. PEFs can do it.

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 the Prediction Error Filter (PEF), the filter in equation (7.10).

Take equation (7.10) for $\mathbf{r}$ and multiply it by the adjoint $\mathbf{r}^*$ getting a quadratic form for $\mathbf{r} \cdot \mathbf{r}$. The matrix of the quadratic form contains the autocorrelation of the original data d_t, not on the data d_t itself. Solving gives the PEF. Changing the polarity of the data or time reversing it leaves the autocorrelation unchanged, so it leaves the PEF unchanged. Thus knowledge of the PEF is equivalent to knowledge of the autocorrelation or the spectrum.

7.3.1 Why 1-D PEFs have white output

The basic idea of least-squares fitting is that the residual is orthogonal to each of the fitting functions. Applied to the prediction error filter (PEF), this idea means that the output of the PEF 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 soon see here 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**. This has many, many applications with examples coming up soon. (Surprisingly, the output of an *interpolation-error* filter is usually non-white.)

Let $\mathbf{d}$ be a vector whose components contain a time function. Let $Z^n\mathbf{d}$ represent shifting the components to delay the signal in $\mathbf{d}$ by n samples. The definition of a prediction-error filter (PEF) is that it minimizes $||\mathbf{r}||$ by adjusting filter coefficients a_τ. The PEF output is:

$$\mathbf{r} = \mathbf{d} + a_1 Z^1 \mathbf{d} + a_2 Z^2 \mathbf{d} + a_3 Z^3 \mathbf{d} + \cdots \tag{7.21}$$

We set out to choose the best a_τ by setting to zero the derivative of $(\mathbf{r} \cdot \mathbf{r})$ by a_τ. After the best a_τ are chosen, the residual is perpendicular to each of the fitting functions:

$$0 = \frac{d}{da_\tau} (\mathbf{r} \cdot \mathbf{r}) \tag{7.22}$$

$$0 = \mathbf{r} \cdot \frac{d\mathbf{r}}{da_\tau} = \mathbf{r} \cdot Z^\tau \mathbf{d} \qquad \text{for } \tau > 0. \tag{7.23}$$

Given that $0 = \mathbf{r} \cdot Z^\tau \mathbf{d}$ we examine $0 = \mathbf{r} \cdot Z^\tau \mathbf{r}$. Using equation (7.21) we have for any autocorrelation lag $k > 0$,

$$\begin{aligned} \mathbf{r} \cdot Z^k \mathbf{r} &= \mathbf{r} \cdot (Z^k \mathbf{d} + a_1 Z^{k+1} \mathbf{d} + a_2 Z^{k+2} \mathbf{d} + ...) \\ &= \mathbf{r} \cdot Z^k \mathbf{d} + a_1 \mathbf{r} \cdot Z^{k+1} \mathbf{d} + a_2 \mathbf{r} \cdot Z^{k+2} \mathbf{d} + ... \\ &= 0 + a_1 0 + a_2 0 + ... \\ &= 0 \,. \end{aligned}$$

Since the autocorrelation is symmetric $\mathbf{r} \cdot Z^{-k} \mathbf{r}$ is also zero for $k < 0$, so the autocorrelation of $\mathbf{r}$ is an impulse. In other words, the spectrum of the time function r_t is white. Thus $\mathbf{d}$ and $\mathbf{a}$ have mutually inverse spectra.

> Since the output of a PEF is white, the PEF itself has a spectrum inverse to its input.

An important application of the PEF is in missing data interpolation. We'll see examples later in this chapter. My third book, PVI, has many examples in one dimension with both synthetic data and field data including the gap parameter. Here we next extend these ideas to two (or more) dimensions.

In practice the degree of whiteness is limited by the number of lags we take in the PEF. This will not be infinity so the autocorrelation will be non-zero for the lags we have omitted. In most applications long-lag correlations tend to be small. This because predictions tend to degrade with time lag. There are exceptions, however. To predict unemployment next month, it helps a lot to know the unemployment this month. On the other hand, because of seasonal effects, the unemployment from a year before next month (11 months back) might provide even better prediction. But mostly, older data has diminishing ability to enhance prediction.

Finite-difference equations resemble PEFs, and they use only a short range of lags, for example, a wave equation containing only the three lags intrinsic to $\partial^2/\partial t^2$. So, short PEFs are often quite analogous to differential equations, hence very powerful, short lags enabling prediction over long intervals.

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** (to be proven here). 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.

Undoing convolution in nature

Prediction-error filtering is called "**blind deconvolution**". In the exploration industry it is simply 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 7.1. Then we design a prediction-error filter $\mathbf{A}$ on

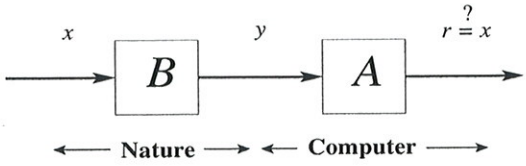


Figure 7.1: Flow of information from nature, to observation, into computer. (y is data $\mathbf{d}$.) VIEW mda/. systems

$\mathbf{y}$, which yields a white-spectrum 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 suggests that deconvolution of natural processes with a PEF might get the correct phase spectrum as well as the correct amplitude spectrum. Naturally, the PEF could not give the correct phase to an "all-pass" filter. That is a filter with a phase shift but a constant amplitude spectrum. (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 older books, FGDP and PVI.

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. A classic PEF estimation technique is named for Norman Levinson found in an appendix of a classic test by Norbert Wiener. Those methods assume the autocorrelation is given. Starting instead from a truncated signal series is another classic method by John Parker Burg. These are described in considerable detail in my web based book FGDP. Having the PEF and its FT, the signal spectrum is simply the inverse the PEF's 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 method supercedes Levinson's in short data windows. Burg's method also ensures a causal inverse, something we will not ensure here. His method should be reviewed in light of the helix.

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 shallow more-consolidated soil layers varies rapidly with surface location because of geologically recent fluvial activity. The spectral **color** of this erratic variation on surface-recorded seismograms is compensated by a PE filter. Usually 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.

7.4 2-D 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".

$$\begin{matrix} -1 & \cdot \\ \cdot & \cdot \\ \cdot & 1 \end{matrix} \tag{7.24}$$

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

$$\begin{matrix} \cdot & 1 \\ -1 & \cdot \end{matrix} \tag{7.25}$$

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

$$\begin{matrix} \cdot & -1 & \cdot \\ 1 & \cdot & \cdot \\ \cdot & \cdot & 1 \\ \cdot & -1 & \cdot \end{matrix} \tag{7.26}$$

which destroys waves of both slopes.

A **two-dimensional filter** that can be a **dip-rejection filter** like (7.24) or (7.25) is

$$\begin{matrix} a & \cdot \\ b & \cdot \\ c & 1 \\ d & \cdot \\ e & \cdot \end{matrix} \tag{7.27}$$

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 (7.26), we adjust coefficients to minimize the power out of filter shapes, as in

$$\begin{matrix} v & a & \cdot \\ w & b & \cdot \\ x & c & 1 \\ y & d & \cdot \\ z & e & \cdot \end{matrix} \tag{7.28}$$

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.

7.4.1 Why 2-D PEFs have white output

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 shows us that we can consider a 2-D PEF to be a small halfplane like 4.9 with an impulse along a side. These shapes are what we see here in Figure 7.2.

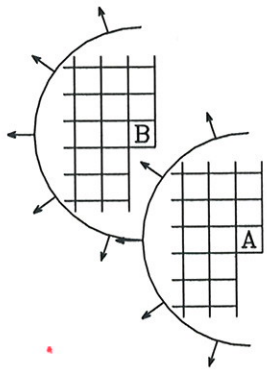


Figure 7.2: A 2-D whitening filter template, and itself lagged. At output locations "A" and "B," the filter coefficient is constrained to be "1". When the semicircles are viewed as having infinite radius, the B filter is contained in the A filter. Because the output at A is orthogonal to all its inputs, which include all inputs of B, the output at A is orthogonal to the output of B. VIEW mda/. whitepruf

Figure 7.2 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 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 and the output of B
output at A	$\perp$	linear combination of all parts of B
output at A	$\perp$	output at B

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

Study Figure 7.2 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. The filter must contain a halfplane of values so that symmetry gives the other half.

You will notice some nonuniqueness. We could embed the helix with a 90° rotation in the original physical application. Besides the difference in side boundaries, the 2-D PEF would have a different orientation. Both PEFs should have an output that tends to whiteness as the filter is enlarged. It seems that we could design whitening autoregression filters for 45° rotations also, and we could also design them for hexagonal coordinate systems. In some physical applications, you might find the nonuniqueness unsettling. Does it mean the "final solution" is nonunique? Usually not, or not seriously so. Recall even in one dimension, the time reverse of a PEF has the same spectrum as the original PEF. When a PEF is used for regularizing a fitting application, it is worth noticing that the quadratic form minimized is the PEF times its adjoint so the phase drops out. Likewise, a missing data restoration also amounts to minimizing a quadratic form so the phase again drops out.

7.5 Basic blind deconvolution

Here are the basic definitions of blind deconvolution: If a model m_t (with FT M) is made of random numbers and convolved with a "source waveform" (having FT) F^{-1} it creates data D. From data D you find the model M by $M = FD$. Trouble is, you typically do not know F and need to estimate (guess) it hence the word "blind."

Suppose we have many observations or many channels of D so we label them D_j. We can define a model M_j as

$$M_j \;=\; \frac{D_j}{\sqrt{\sum_j D^* D}} \tag{7.29}$$

so blind deconvolution removes the average spectrum.

Sometimes we have only a single signal D but it is quite long. Because the signal is long, the magnitude of its Fourier transform is rough, so we smooth it over frequency, and

denote it thus:

$$M = \frac{D}{\sqrt{\ll D^* D \gg}} \tag{7.30}$$

Smoothing the spectrum makes the time function shorter. Indeed, the amount of smoothing may be chosen by the amount of shortness wanted.

The above preliminary models are the most primative forms of deconvolved data. They deal only with the amplitude spectrum. Most deconvolutions involve also the phase. The examples we'll show next do include the phase. That's sometimes significant, sometimes not. Averaging occurs because the PEF is smaller than the data.

$$m = d * \text{PEF} \tag{7.31}$$

7.5.1 Examples of modeling and deconvolving with a 2-D PEF

Here we examine elementary signal-processing applications of 2-D prediction-error filters (PEFs) on both everyday 2-D textures and on seismic data. Some of these textures are easily modeled with prediction-error filters (PEFs) while others are not. All figures used the same 10×10 filter shape. No attempt was made to optimize filter size or shape or any other parameters.

Results in Figures 7.3-7.9 are shown with various familiar textures[2] on the left as training data sets. From these training data sets, a prediction-error filter (PEF) is estimated using module `pef`. The center frame is simulated data made by deconvolving (polynomial division) random numbers by the estimated PEF. The right frame is the more familiar process, convolving the estimated PEF on the training data set. Theoretically, the right frame tends towards a white spectrum.

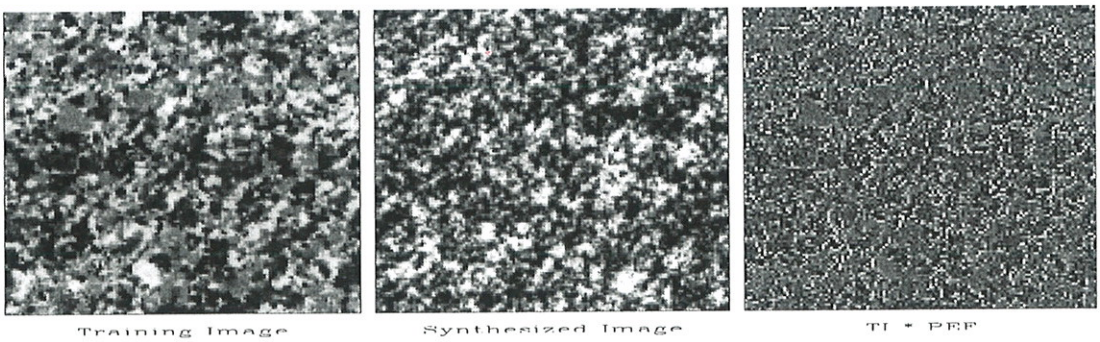


Figure 7.3: Synthetic granite matches the training image quite well. The prediction error (PE) is large at grain boundaries so it almost seems to outline the grains. VIEW mda/. granite

Since a PEF tends to the inverse of the spectrum of its input, results similar to these could likely be found using Fourier transforms, smoothing spectra, etc. We used PEFs because of their flexibility. The filters can be any shape. They can dodge around missing

[2] I thank Morgan Brown for finding these textures.

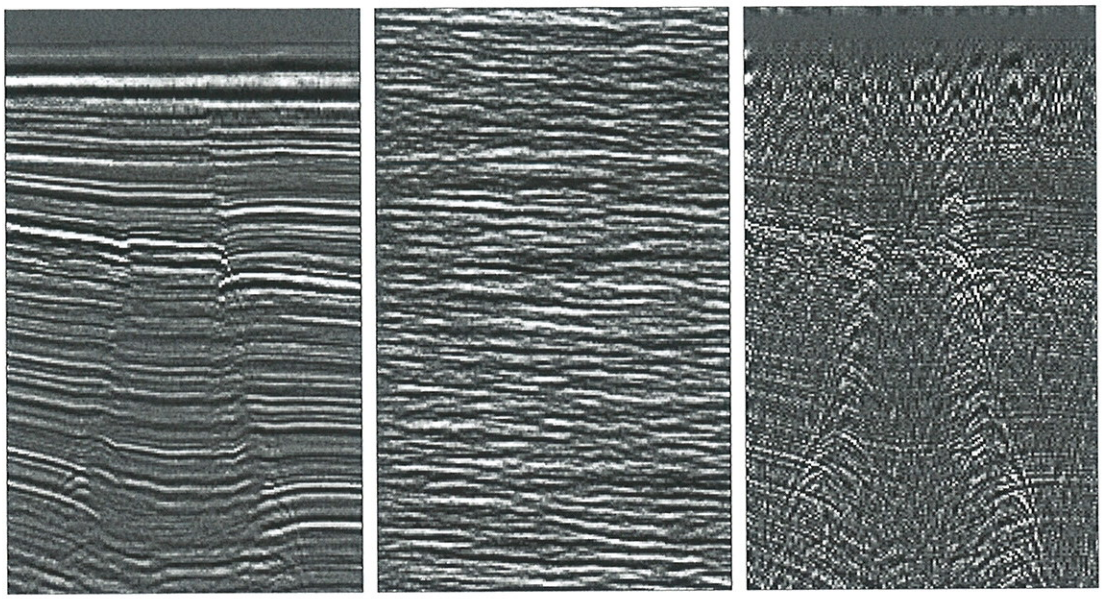


Figure 7.9: Gulf of Mexico seismic section, modeled, and deconvolved. Do you see any drilling prospects in the simulated data? In the deconvolution, the strong horizontal layering is suppressed giving a better view of the hyperbolas. The decon filter has the same 10×10 size used on the everyday textures. VIEW mda/. WGstack

data, or we can use them to estimate missing data. PEFs with a helix have periodic boundary assumptions on all axes but one, while FTs have periodic boundaries on all axes. The PEFs are designed only internal to known data, not off edges so they are readily adaptable small data samples and to nonstationarity. Thinking of these textures as seismic time slices, the textures could easily be required to pass thru specific values at well locations.

7.5.2 Seismic field data examples

Figures 7.10-7.12 are based on exploration seismic data from the Gulf of Mexico deep water. A ship carries an air gun and tows a streamer with some hundreds of geophones. First we look at a single pop of the gun. We use all the hydrophone signals to create a single 1-D PEF for the time axis. This changes the average temporal frequency spectrum as shown in Figure 7.10. Signals from 60 Hz to 120 Hz are boosted substantially. The raw data has evidently been prepared with strong filtering against signals below about 8 Hz. The PEF attempts to recover these signals, mostly unsuccessfully, but it does boost some energy near the 8 Hz cutoff. Choosing a longer filter would flatten the spectrum further. The big question is, "Has the PEF improved the appearance of the data?"

The data itself from the single pop, both before and after PE-filtering is shown in Figure 7.11. For reasons of aesthetics of human perception I have chosen to display a mirror image of the PEF'ed data. To see a blink movie of superposition of before-and-

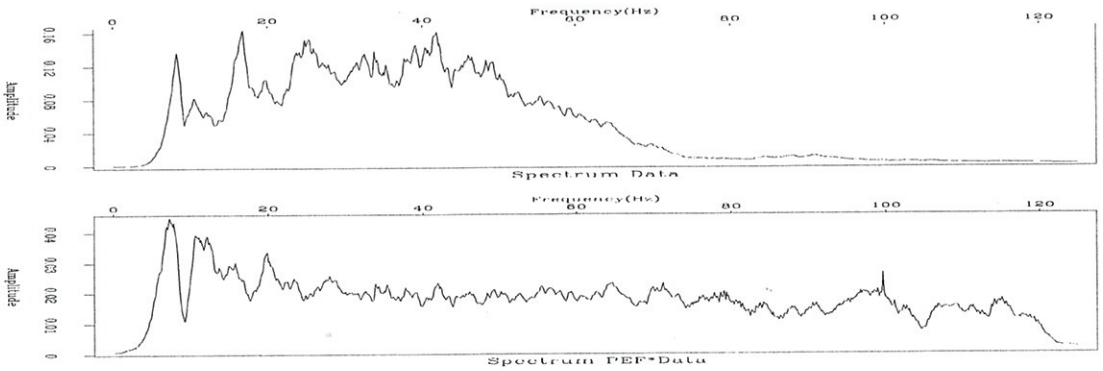


Figure 7.10: ω spectrum of a shot gather of Figure 7.11 before and after 1-D decon with a 30 point filter. VIEW mda/. antoinedecon1

after images you need the electronic book (which technology does not enable me to deliver in 2014). We notice that signals of high temporal frequencies indeed have the expected hyperbolic behavior in space. Thus, these high-frequency signals are wavefields, not mere random noise.

Given that all visual (or audio) displays have a bounded range of amplitudes, increasing the frequency content (bandwidth) means that we will need to turn down the amplification so we do not wish to increase the bandwidth unless we are adding signal.

> Increasing the spectral bandwidth always requires us to diminish the gain.

The same ideas but with a two-dimensional PEF are in Figure 7.12 (the same data but with more of it squeezed onto the page.) After the PEF, we tend to see equal energy in dips in all directions. We have strongly enhanced the "backscattered" energy, those events that arrive later at *shorter* distances.

We have been thinking of the PEF as a tool for shaping the spectrum of a display. But does it have a physical meaning? What might it be? Referring back to the beginning of the chapter we are inclined to regard the PEF as the convolution of the source waveform with some kind of water-bottom response. In Figure 7.12 we used many different shot-receiver separations. Since each different separation has a different response (due to differing moveouts) the water bottom reverberation might average out to be roughly an impulse. Figure 7.13 is a different story. Here for each shot location, the distance to the receiver is constant. Designing a single channel PEF we can expect the PEF to contain both the shot waveform and the water bottom layers because both are nearly identical in all the shots. We would rather have a PEF that represents only the shot waveform (and perhaps a radiation pattern).

Let us consider how we might work to push the water-bottom reverberation out of the PEF. This data is recorded in water 600 meters deep. A consequence is that the sea bottom is made of fine-grained sediments that settled very slowly and rather similarly from place to place. In shallow water the situation is different. The sands near estuaries are always shifting. Sedimentary layers thicken and thin. They are said to "on-lap and off-lap." Here

I notice where the water bottom is sloped, the layers thin a little. To push the water bottom layers out of the PEF, our idea is to base its calculation not on the raw data, but on the spatial prediction error of the raw data. On a perfectly layered earth a perfect spatial prediction error filter would zero all traces but the first one. Since a 2-D PEF includes spatial prediction as well as temporal prediction, we can expect it to contain much less of the sea-floor layers than the 1-D PEF. If you have access to the electronic book, you can blink the figure back and forth with various filter shapes.

7.6 PEF ESTIMATION WITH MISSING DATA

If we are not careful, our calculation of the PEF could have the pitfall that it would try to use the missing data to find the PEF, and hence it would get the wrong PEF. To avoid this pitfall, imagine a PEF finder that uses weighted least squares where the weighting function vanishes on those fitting equations that involve missing data. The weighting would be unity elsewhere. Instead of weighting bad results by zero, we simply will not compute them. The residual there will be initialized to zero and never changed. Likewise for the adjoint, these components of the residual will never contribute to a gradient. So now we need a convolution program that produces no outputs where missing inputs would spoil it.

Recall there are two ways of writing convolution, equation (1.4) when we are interested in finding the filter *inputs*, and equation (1.5) when we are interested in finding the *filter itself*. We have already coded equation (1.4), operator `helicon`. That operator was useful in missing data applications. Now we want to find a prediction-error filter, so we need the other case, equation (1.5), and we need to ignore the outputs that will be broken because of missing inputs. The operator module `hconest` does the job.

helix convolution.lop

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

We are seeking a prediction error filter $(1, a_1, a_2)$ but some of the data is missing. The data is denoted $\mathbf{y}$ or y_i above and x_i below. Because some of the x_i are missing, some of the regression equations in (7.32) are worthless. When we figure out which are broken, we will put zero weights on those equations.

$$\mathbf{0} \approx \mathbf{r} = \mathbf{WXa} = \begin{bmatrix} w_1 & . & . & . & . & . & . & . \\ . & w_2 & . & . & . & . & . & . \\ . & . & w_3 & . & . & . & . & . \\ . & . & . & w_4 & . & . & . & . \\ . & . & . & . & w_5 & . & . & . \\ . & . & . & . & . & w_6 & . & . \\ . & . & . & . & . & . & w_7 & . \\ . & . & . & . & . & . & . & w_8 \end{bmatrix} \begin{bmatrix} x_1 & 0 & 0 \\ x_2 & x_1 & 0 \\ x_3 & x_2 & x_1 \\ x_4 & x_3 & x_2 \\ x_5 & x_4 & x_3 \\ x_6 & x_5 & x_4 \\ 0 & x_6 & x_5 \\ 0 & 0 & x_6 \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} \tag{7.32}$$

Suppose that x_2 and x_3 were missing or known bad. That would spoil the 2nd, 3rd, 4th, and 5th fitting equations in (7.32). In principle, we want w_2, w_3, w_4, and w_5 to be zero. In practice, we simply want those components of $\mathbf{r}$ to be zero.

What algorithm will enable us to identify the regression equations that have become defective, now that x_2 and x_3 are missing? Take filter coefficients $(a_0, a_1, a_2, \ldots)$ to be all ones. Let $\mathbf{d}_{\text{free}}$ be a vector like $\mathbf{x}$ but containing 1's for the missing (or "freely adjustable") data values and 0's for the known data values. Recall our very first definition of filtering showed we can put the filter in a vector and the data in a matrix or vice versa. Thus $\mathbf{Xa}$ above gives the same result as $\mathbf{Ax}$ below.

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 2 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \mathbf{A}\mathbf{d}_{\text{free}} \tag{7.33}$$

The numeric value of each m_i tells us how many of its inputs are missing. Where none are missing, we want unit weights $w_i = 1$. Where any are missing, we want zero weights $w_i = 0$. The desired residual under partially missing inputs is computed by module `misinput`.

mark bad regression equations.r90

```
module misinput {                          # find a mask of missing filter inputs
  use helicon
contains
  subroutine find_mask( known, aa) {
    logical, intent( in)           :: known(:)
    type( filter)                  :: aa
    real, dimension( size (known)) :: rr, dfre
    integer                        :: stat
    where( known) dfre = 0.
    elsewhere     dfre = 1.
    call helicon_init( aa)
    aa%flt = 1.
    stat = helicon_lop( .false., .false., dfre, rr)
```

```
    aa%flt = 0.
    where ( rr > 0.)    aa%mis = .true.
    }
}
```

7.6.1 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.

The two little triangular patches of zeros in the convolution matrix in equation (7.32) describe end conditions where it is assumed that the data y_t vanishes before $t = 1$ and after $t = 6$. Alternately we might not wish to make that assumption. Thus the triangles filled with zeros could be regarded as missing data. In this one-dimensional example, it is easy to see that the filter, say `yy%mis()` should be set to `.TRUE.` at the ends so no output would ever be computed there. We would like to find a general multidimensional algorithm to correctly specify `yy%mis()` around the multidimensional boundaries. This proceeds like the missing data algorithm, i.e. we apply a filter of all ones to a data space template that is taken all zeros except ones at the locations of missing data, in this case y_0, y_{-1} and y_7, y_8. This amounts to surrounding the original data set with some missing data. We need padding the size of the filter on all sides. The padded region would be filled with ones (designating missing inputs). Where the convolution output is nonzero, there `yy%mis()` is set to `.TRUE.` denoting an output with missing inputs.

The two-dimensional case is a little more cluttered than the 1-D case but the principle is about the same. Figure 7.14 shows a larger input domain, a 5×3 filter, and a smaller output domain. There are two things to notice. First, sliding the 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. A chapter, possibly not included with this version of the book (for reasons of clutter) breaks data spaces into overlapping patches, separately analyzes the patches, and puts everything back together. This is needed when the crosscorrelation changes with time. That is 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;

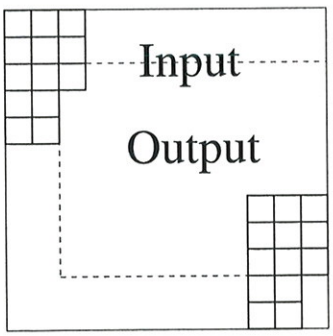


Figure 7.14: Domain of inputs and outputs of a two-dimensional filter like a PEF. VIEW mda/. rabdomain

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 7.14. Below is module bound to build a selector for filter outputs that should never be examined or even computed (because they need input data from outside the given data space). Inputs are a filter aa and the size of its cube na = (na(1),na(2),...). Also input are two cube dimensions, that of the data last used by the filter nold and that of the filter's next intended use nd. (nold and nd are often the same). Module bound begins by defining a bigger data space with room for a filter surrounding the original data space nd on all sides. It does this by the line nb=nd+2*na. Then we allocate two data spaces xx and yy of the bigger size nb and pack many ones in a frame of width na around the outside of xx. The filter aa is also filled with ones. The filter aa must be regridded for the bigger nb data space (regridding merely changes the lag values of the ones). Now we filter the input xx with aa getting yy. Wherever the output is nonzero, we have an output that has been affected by the boundary. Such an output should not be computed. Thus we allocate the logical mask aa%mis (a part of the helix filter definition in module helix and wherever we see a nonzero value of yy in the output, we designate the output as depending on missing inputs by setting aa%mis to .true..

out of bounds dependency.r90

```
module bound {          # mark helix filter outputs where input is off data.
use cartesian
use helicon
use regrid
contains
  subroutine boundn ( nold, nd, na, aa) {
    integer, dimension( :), intent( in) :: nold, nd, na     # (ndim)
    type( filter)                       :: aa
    integer, dimension( size( nd))      :: nb, ii
    real,    dimension( :), allocatable :: xx, yy
    integer                             :: iy, my, ib, mb, stat
    nb = nd + 2*na;   mb = product( nb)     # nb is a bigger space to pad into.
    allocate( xx( mb), yy( mb))             # two large spaces, equal size
    xx = 0.                                 #              zeros
    do ib = 1, mb {                         # surround the zeros with many ones
       call line2cart( nb, ib, ii)          # ii( ib)
       if( any( ii <= na  .or.  ii > nb-na))   xx( ib) = 1.
       }
    call helicon_init( aa)                  # give aa pointer to helicon.lop
    call regridn( nold, nb, aa); aa%flt = 1.          # put all 1's in filter
    stat = helicon_lop( .false., .false., xx, yy)     # apply filter
    call regridn(   nb, nd, aa); aa%flt = 0.    # remake filter for orig data.
    my = product( nd)
    allocate( aa%mis( my))              # attach missing designation to y_filter
    do iy = 1, my {                     # map from unpadded to padded space
       call line2cart( nd, iy, ii )
       call cart2line( nb,     ii+na, ib )          # ib( iy)
       aa%mis( iy) =             ( yy( ib) > 0.)    # true where inputs missing
       }
    deallocate( xx, yy)
    }
}
```

In reality one would set up the boundary conditions with module bound before identifying locations of missing data with module misinput. Both modules are based on the same concept, but the boundaries are more cluttered and confusing which is why we examined them later.

7.6.2 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 PEF 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.

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 "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 will get a zero filter, or you might encounter the error exit from cgstep().

estimate PEF on a helix.r90

```
module pef {                    # Find prediction-error filter (helix magic)
  use hconest
  use cgstep_mod
  use solver_smp_mod
contains
  subroutine find_pef( dd, aa, niter) {
    integer,          intent( in)  :: niter          # number of iterations
    type( filter)                  :: aa             # filter
    real,    dimension(:), pointer :: dd             # input data
    call hconest_init( dd, aa)
    call solver_smp(m=aa%flt, d=-dd, Fop=hconest_lop, stepper=cgstep, &
                                                      niter=niter, m0=aa%flt)
    call cgstep_close()
    }
}
```

7.7 TWO-STAGE LINEAR LEAST SQUARES

In Chapter 3 and Chapter 5 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 find and use the optimum filter, the PEF.

The first stage is that of the previous section, finding the optimal PEF while carefully avoiding using any regression equations that involve boundaries or missing data. For the second stage, we take the PEF as known and find values for the empty bins so that the power out of the prediction-error filter is minimized. To do this we find missing data with module mis2().

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.

The synthetic data in Figure 7.15 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.

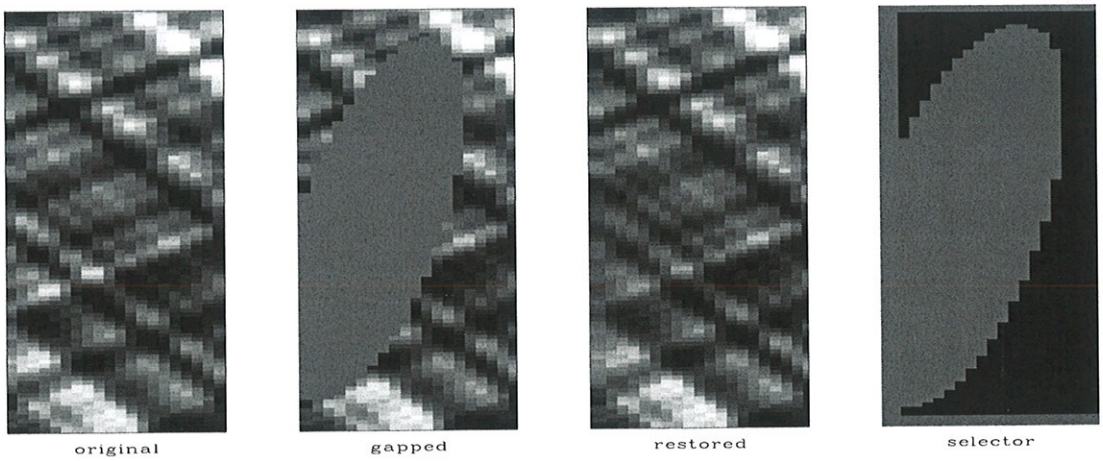


Figure 7.15: Original data (left), with a zeroed hole, restored, residual selector (right). VIEW mda/. hole90

The residual selector in Figure 7.15 shows where the filter output has valid inputs. From it you can deduce the size and shape of the filter; namely that it matches up with Figure 7.14. The ellipsoidal hole in the residual selector is larger than that in the data because we lose regression equations not only at the hole, but where any part of the filter overlaps the hole.

The results in Figure 7.15 are essentially perfect representing the fact that that synthetic example fits the conceptual model perfectly. Before we look at the many examples in Figures 7.16-7.19 we examine another gap-filling strategy.

7.7.1 Adding noise (Geostat)

In chapter 3 we restored missing data by adopting the philosophy of minimizing the energy in filtered output. In this chapter we learned about an optimum filter for this task, the prediction-error filter (PEF). Let us name this method the "minimum noise" method of finding missing data.

A practical application with the minimum-noise method is evident in a large empty hole such as in Figures 7.16-7.17. In such a void the interpolated data diminishes greatly. Thus we have not totally succeeded in the goal of "hiding our data acquisition footprint" which we would like to do if we are trying to make pictures of the earth and not pictures of our

data acquisition footprint.

What we will do next is useful in some applications but not in others. Misunderstood or misused it is rightly controversial. We are going to fill the empty holes with something that looks like the original data but really isn't. I will distinguish the words "**synthetic data**" (that derived from a physical model) from "**simulated data**" (that manufactured from a statistical model). We will fill the empty holes with simulated data like what you see in the center panels of Figures 7.3-7.9. We will add just enough of that "wall paper noise" to keep the variance constant as we move into the void.

Given some data $\mathbf{d}$, we use it in a filter operator $\mathbf{D}$, and as described with equation (7.32) we build a weighting function $\mathbf{W}$ that throws out the broken regression equations (ones that involve missing inputs). Then we find a PEF $\mathbf{a}$ by using this regression:

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \mathbf{WDa} \tag{7.34}$$

Because of the way we defined $\mathbf{W}$, the "broken" components of $\mathbf{r}$ vanish. We need to know the variance σ of the nonzero terms. It can be expressed mathematically in a couple different ways. Let $\mathbf{1}$ be a vector filled with ones and let $\mathbf{r}^2$ be a vector containing the squares of the components of $\mathbf{r}$.

$$\sigma \quad = \quad \sqrt{\frac{1}{N}\sum_i^N r_i^2} \quad = \quad \sqrt{\frac{\mathbf{1'Wr}^2}{\mathbf{1'W1}}} \tag{7.35}$$

Let us go to a random number generator and get a noise vector $\mathbf{n}$ filled with random numbers of variance σ. We'll call this the "added random noise". Now we solve this new regression for the data space $\mathbf{d}$ (both known and missing):

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \mathbf{Ad} - \mathbf{n} \tag{7.36}$$

keeping in mind that known data is constrained (as detailed in chapter 3).

To understand why this works, consider first the training image, a region of known data. Although we might think that the data defines the white noise residual by $\mathbf{r} = \mathbf{Ad}$, we can also imagine that the white noise determines the data by $\mathbf{d} = \mathbf{A}^{-1}\mathbf{r}$. Then consider a region of wholly missing data. This data is determined by $\mathbf{d} = \mathbf{A}^{-1}\mathbf{n}$. Since we want the data variance to be the same in known and unknown locations, naturally we require the variance of $\mathbf{n}$ to match that of $\mathbf{r}$.

A very minor issue remains. Regression equations may have all of their required input data, some of it, or none of it. Should the $\mathbf{n}$ vector add noise to every regression equation? First, if a regression equation has all its input data that means there are no free variables so it doesn't matter if we add noise to that regression equation because the constraints will overcome that noise. I don't know if I should worry about how *many* inputs are missing for each regression equation.

It is fun making all this interesting "wall paper", noticing where it is successful and where it isn't. We cannot help but notice that it seems to work better with the genuine geophysical data than it does with many of the highly structured patterns. Geophysical data is expensive to acquire. Regrettably, we have uncovered a technology that makes counterfeiting much easier.

Examples are in Figures 7.16-7.19. In the electronic book, the right-side panel of each figure is a movie, each panel being derived from different random numbers. Unfortunately, in 2014 I am not able to deliver the electronic book on the internet.

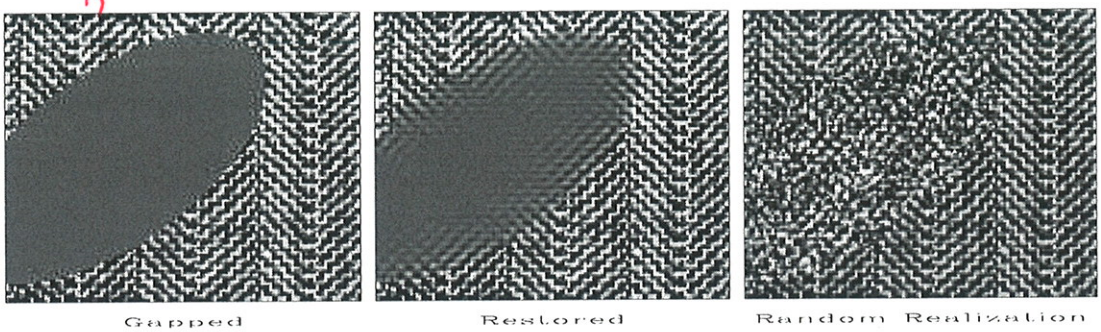


Figure 7.16: The herringbone texture is a patchwork of two textures. We notice that data missing from the hole tends to fill with the texture at the edge of the hole. The spine of the herring fish, however, is not modeled at all. VIEW mda/. herr-hole-fillr

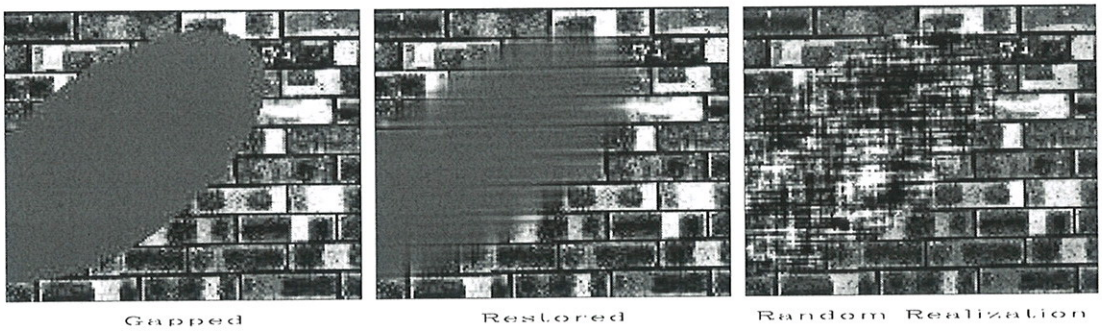


Figure 7.17: The brick texture has a mortar part (both vertical and horizontal joins) and a brick surface part. These three parts enter the empty area but do not end where they should. VIEW mda/. brick-hole-fillr

The seismic data in Figure 7.19 illustrates a fundamental principle: In the restored hole (center) we do not see the same spectrum as we do on the other panels. This is because the hole is filled, not with all frequencies (or all slopes) but with those that are most predictable. The filled hole is devoid of the unpredictable noise that is a part of all real data.

Figure 7.20 is an interesting seismic image showing ancient river channels now deeply buried. Such river channels are often filled with sand, hence are good petroleum prospects. Prediction error methodology fails to simulate these channels. We can give the reason that the channels are not statistically stationary. The methodology also fails to extrapolate them very far from a known region into a hidden region.

7.7.2 Inversions with geostat

In geophysical estimation (inversion) we use model styling (regularization) to handle the portion of the model that is not determined by the data. This results in the addition of

minimal noise. Alternately, like in Geostatistics, we could make an assumption of statistical stationarity and add much more noise so the signal variance in poorly determined regions matches that in well determined regions. Here is how to do this. Given the usual data fitting and model styling goals

$$\mathbf{0} \approx \mathbf{Lm} - \mathbf{d} \tag{7.37}$$
$$\mathbf{0} \approx \mathbf{Am} \tag{7.38}$$

We introduce a sample of random noise $\mathbf{n}$ and fit instead these regressions

$$\mathbf{0} \approx \mathbf{Lm} - \mathbf{d} \tag{7.39}$$
$$\mathbf{0} \approx \mathbf{Am} - \mathbf{n} \tag{7.40}$$

Of course you get a different solution for each different realization of the random noise. You also need to be a little careful to use noise $\mathbf{n}$ of the appropriate variance. **Bob Clapp** developed this idea at SEP and also applied it to interval velocity estimation, the example of Figures 5.3-5.5.

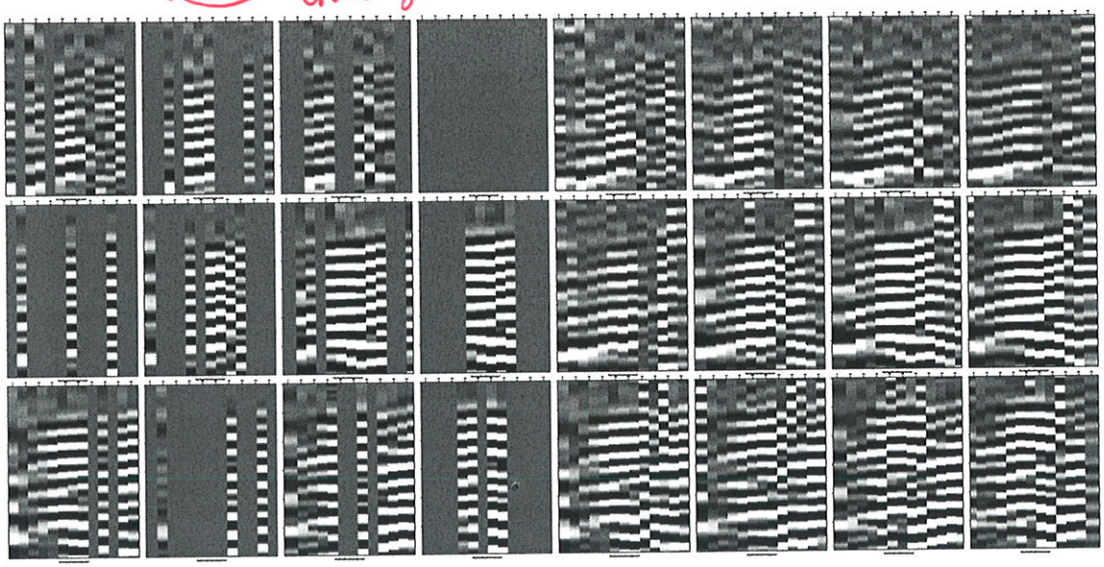


Figure 7.21: The left 12 panels are the inputs. The right 12 panels are outputs. VIEW mda/. passfill90

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

Finding **missing data** (filling empty bins) requires use of a filter. Because of the helix, the codes work in spaces of all dimensions.

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 but sometimes I run double the number of iterations to be sure. The missing-data estimation, however, is a completely different story. The number of free parameters in the missing-data estimation could be

very large. This often implies impractically long compute times for the exact solution. In practice I experiment carefully with `niter` and hope for the best. I find that where gaps are small, they fill in quickly. Where the gaps are large, they don't, and more iterations are required. Where the gaps are large is where we should experiment with preconditioning.

Figure 7.21 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 with a small 3-D filter. That filter was a PEF. It may seem strange that an empty panel is filled by interpolation. That information came from the panels on either side of the empty panel.

7.8 SEABEAM: FILLING THE EMPTY BINS WITH A PEF

In chapter 5 empty bins in an image of the ocean bottom were filled using the laplacian operator. It is shown in Figure 5.10.

The problem with the Laplacian operator as an interpolator is that it smears information uniformly in all directions. We see we need an anisotropic interpolation oriented along the regional trends. What we need is a PEF in place of the Laplacian. To get it, we apply module `pef` on page 200. After binning the data and finding this PEF, we do a second stage of linear-least-squares optimization as we did for Figure 7.15, and we obtain the pleasing result in Figure 7.22.

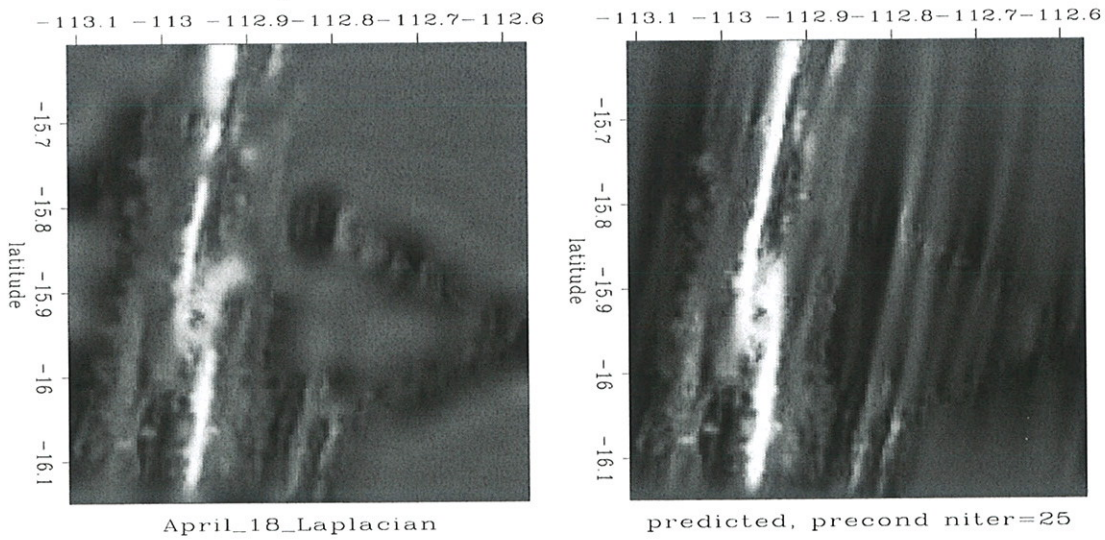


Figure 7.22: Depth of the ocean (Figure 5.10) as filled with a laplacian (left) and with a PEF (right). mda/. seamda

7.8.1 The bane of PEF estimation

This is the place where I would like to pat myself on the back for having "solved" the problem of missing data. Actually, an important practical problem remains. The problem arises when there is too much missing data. Then *all* the regression equations disappear. The nonlinear methods are particularly bad because if they don't have a good enough starting location, they can and do go crazy. My only suggestion is to begin with a linear PEF estimator. Shrink the PEF and coarsen the mesh in model space until you do have enough equations. Starting from there, hopefully you can refine this crude solution without dropping into a local minimum.

The bane of PEF estimation is too much missing data.

7.9 MADAGASCAR: merging bidirectional views

Gravity of mountains on the ocean bottom pulls water towards them raising sea level above them. Kilometer high topography on the sea floor creates 10cm topography on the sea floor that can be dug out from the many stronger oceanographic effects.

A satellite points a radar at the ground and receives echoes we investigate here. These echoes are recorded only over the ocean. The echo tells the distance from the orbit to the ocean surface. After various corrections are made for earth and orbit ellipticities the residual shows tides, wind stress on the surface, and surprisingly a signal proportional to the depth of the water.

The raw data investigated here[3] had a strong north-south tilt ~~which~~ I[4] removed at the outset. Figure 7.23 gives our first view of altimetry data (ocean height) from southeast of the island of Madagascar. About all we can see is satellite tracks. The satellite flies a circular orbit, effectively a polar orbit, south to north, then north to south. Earth at the center of the circle rotates east to west. To us the sun seems to rotate east to west as does the circular orbit. Consequently, when the satellite moves northward it is measuring altitude along a line running SE→NW. When it moves southward we get measurements along a NE→SW line. This data is from the cold war era. At that time dense data above the $-30°$ parallel was secret although sparse data was available. (The restriction had to do with precision guidance of missiles. Would the missile hit the silo? or miss it by enough to save the retaliation missile? Knowledge of regional gravity in the northern hemisphere was essential.)

Here are some definitions: Let components of $\mathbf{d}$ be the data, altitude measured along a satellite track. The model space is $\mathbf{h}$, altitude on portion of the earth surface, that surface flattened to an (x, y)-plane. Let $\mathbf{L}$ denote the 2-D linear interpolation operator from the plane to a track. Let $\mathbf{H}$ be the helix derivative, a filter with response $\sqrt{k_x^2 + k_y^2}$. Except where otherwise noted, the roughened image $\mathbf{p}$ is the preconditioned variable $\mathbf{p} = \mathbf{Hh}$. The

[3] I wish to thank David T. Sandwell http://topex.ucsd.edu/ for providing me with this subset of satellite altimetry data, commonly known as Topex-Posidon data. Readers may also enjoy oceanographic observation on internet video.

[4] The calculations here were all done for us by Jesse Lomask.

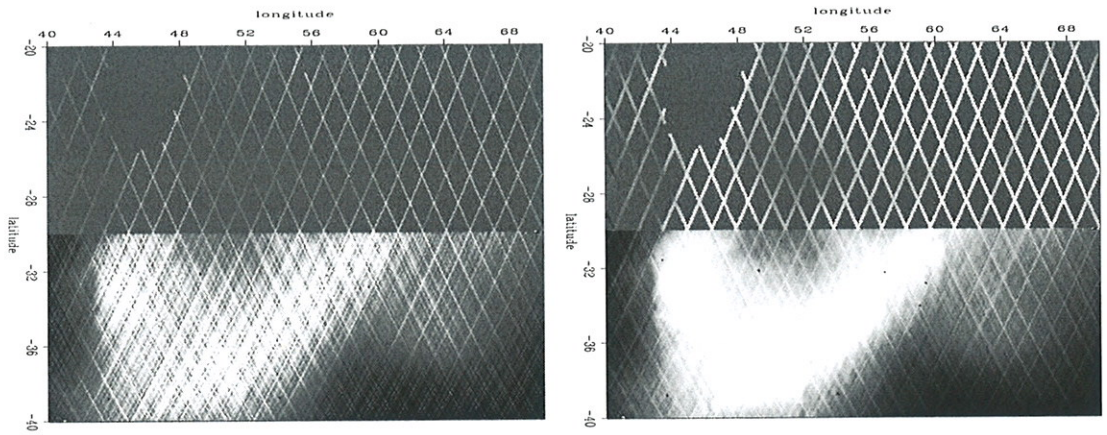


Figure 7.23: Sea height under satellite tracks. The island of Madagascar is in the empty area at $(46°, -22°)$. Left is the adjoint $\mathbf{L}^*\mathbf{d}$. Right is the adjoint normalized by the bin count, $\mathbf{diag}(\mathbf{L}^*\mathbf{1})^{-1}\mathbf{L}^*\mathbf{d}$. You might notice a few huge, bad data values. Overall, the topographic function is too smooth, suggesting we need a roughener. VIEW mda/. jesse1

derivative along a track in data space is $\frac{d}{dt}$. A weighting function that vanishes when any filter hits a track end or a bad data point is $\mathbf{W}$.

Figure 7.24 shows the entire data space, over a half million data points (actually 537974). Altitude is measured along many tracks across the image. In Figure 7.24 the tracks are placed end-to-end, so it is one long vector (displayed in about 50 signal rows). A vector of equal length is the missing data marker vector. This vector is filled with zeros everywhere except where data is missing or known bad or known to be at the ends of the tracks. The long tracks are the ones that are sparse in the north.

Figure 7.25 brings this information into model space. Applying the adjoint of the linear interpolation operator $\mathbf{L}^*$ to the data $\mathbf{d}$ gave our first image $\mathbf{L}^*\mathbf{d}$ in model space in Figure 7.23. The track noise was so large that roughening it made it worse (not shown). A more inviting image arose when I normalized the image before roughening it. Put a vector of all ones $\mathbf{1}$ into the adjoint of the linear interpolation operator $\mathbf{L}^*$. What comes out $\mathbf{L}^*\mathbf{1}$ is roughly the number of data points landing in each pixel in model space. More precisely, it is the sum of the linear interpolation weights. This then, if it is not zero, is used as a divisor. The division accounts for several tracks contributing to one pixel. In matrix formalism this image is $\mathbf{diag}(\mathbf{L}^*\mathbf{1})^{-1}\mathbf{L}^*\mathbf{d}$. In Figure 7.25 this image is roughened with the helix derivative $\mathbf{H}$.

There is a simple way here to make a nice image—roughen along data tracks. This is done in Figure 7.26. The result is two attractive images, one for each track direction. Unfortunately, there is no simple relationship between the two images. We cannot simply add them because the shadows go in different directions. Notice also that each image has noticeable tracks that we would like to suppress further.

A geological side note: The strongest line, the line that marches along the image from southwest to northeast is a sea-floor spreading axis. Magma emerges along this line as a source growing plates that are spreading apart. Here the spreading is in the north-south

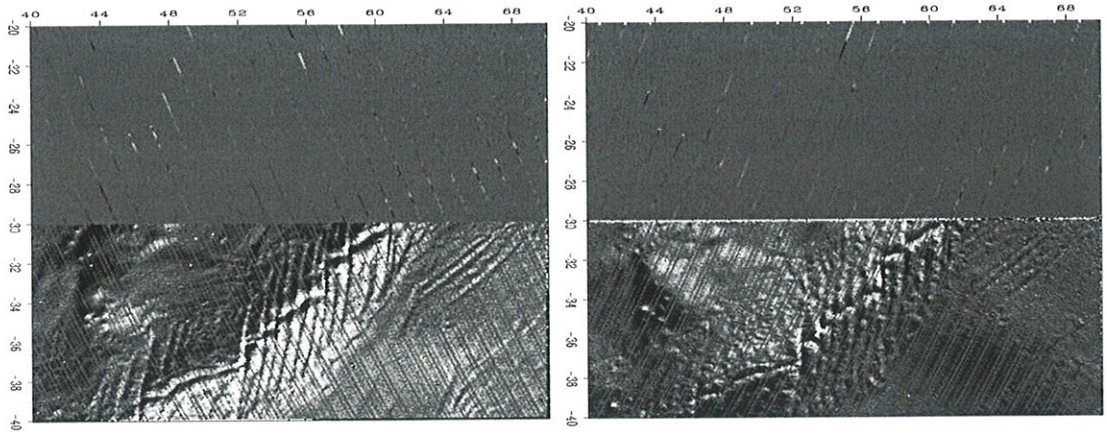


Figure 7.26: With a simple roughening derivative in data space, model space shows two nice topographic images. Let $\mathbf{n}$ denote ascending tracks. Let $\mathbf{s}$ denote descending tracks. Left is $\mathbf{L}^* \frac{d}{dt}\mathbf{n}$. Right is $\mathbf{L}^* \frac{d}{dt}\mathbf{s}$. VIEW mda/. jesse3

direction. The many vertical lines in the image are called "transform faults".

Fortunately, we know how to merge the data. The basic trick is to form the track derivative not on the data (which would falsify it) but on the residual which (in Fourier space) can be understood as choosing a different weighting function for the statistics. A track derivative on the residual is actually two track derivatives, one on the observed data, and the other on the modeled data. Both data sets are changed in the same way. Figure 7.27 shows the result. The altitude function remains too smooth for nice viewing by variable

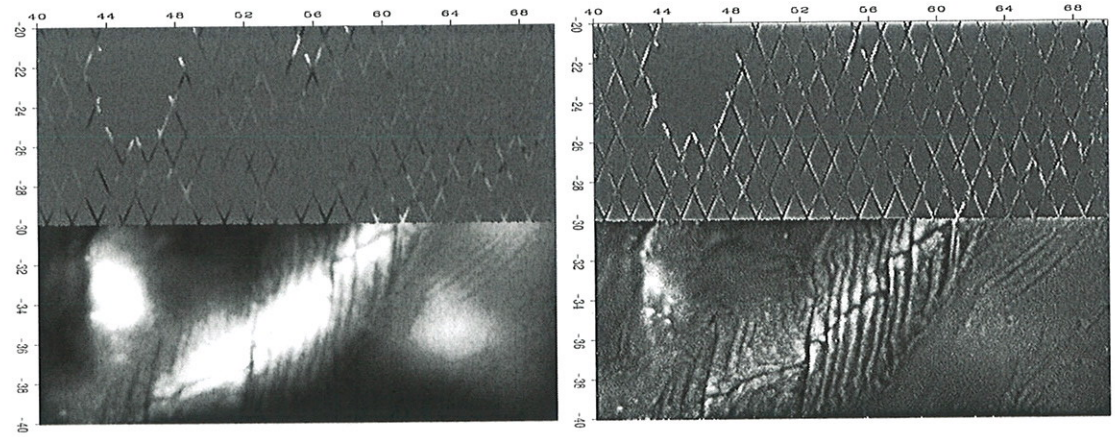


Figure 7.27: All data merged into a track-free image (hooray!) by applying the track derivative, not to the data, but to the residual. Left is $\mathbf{h}$ estimated by $\mathbf{0} \approx \mathbf{W}\frac{d}{dt}(\mathbf{Lh} - \mathbf{d})$. Right is the roughened altitude, $\mathbf{p} = \mathbf{Hh}$. VIEW mda/. jesse10

brightness, but roughening it with $\mathbf{H}$ makes an attractive image showing, in the south, no visible tracks.

The north is another story. We would like the sparse northern tracks to contribute to

our viewing pleasure. We would like them to contribute to a northern image of the earth, not to an image of the data acquisition footprint. This begins to happen in Figure 7.28.

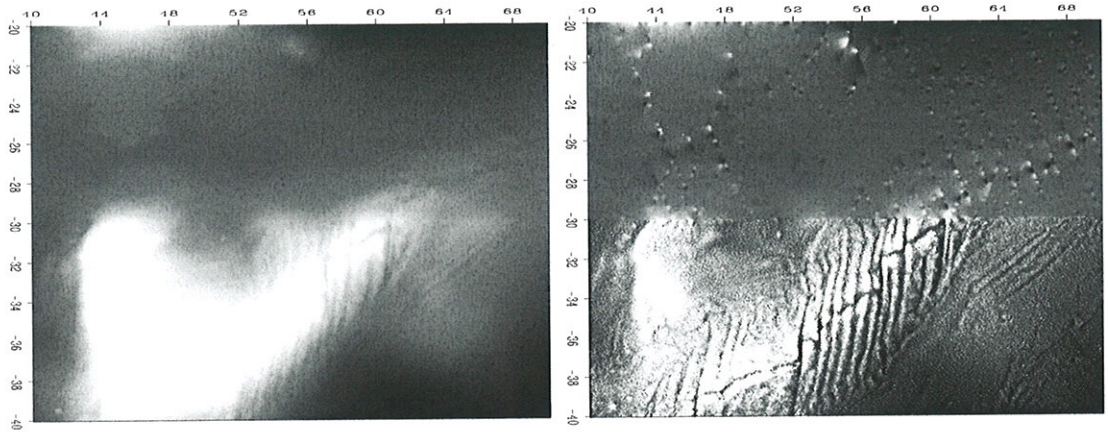


Figure 7.28: Using the track derivative in residual space and helix preconditioning in model space we start building topography in the north. Left is $\mathbf{h} = \mathbf{H}^{-1}\mathbf{p}$ where $\mathbf{p}$ is estimated by $\mathbf{0} \approx \mathbf{W}\frac{d}{dt}(\mathbf{L}\mathbf{H}^{-1}\mathbf{p} - \mathbf{d})$ for only 10 iterations. Right is $\mathbf{p} = \mathbf{H}\mathbf{h}$. VIEW mda/. jesse8

The process of fitting data by choosing an altitude function $\mathbf{h}$ would normally include some regularization (model styling), such as $\mathbf{0} \approx \nabla\mathbf{h}$. Instead we adopt the usual trick of changing to preconditioning variables, in this case $\mathbf{h} = \mathbf{H}^{-1}\mathbf{p}$. As we iterate with the variable $\mathbf{p}$, we watch the images of $\mathbf{h}$ and $\mathbf{p}$ and quit either when we are tired, or more hopefully, when we are best satisfied with the image. This subjective choice is rather like choosing the ϵ that is the balance between data fitting goals and model styling goals. Chapter 5 explains the logic. The result in Figure 7.28 is pleasing. We have begun building topography in the north that continues in a consistent way with what is in the south. Unfortunately, this topography does fade out rather quickly as we get off the data acquisition tracks.

If we have reason to suspect that the geological style north of the 30th parallel matches that south of it (the stationarity assumption), we can compute a PEF on the south side and use it for interpolation on the north side. This is done in Figure 7.29. The final image contrasts delightfully from earlier ones. Our fractured ridge continues nicely into the north. Unfortunately, we have imprinted the fractured ridge texture all over the northern space, but that's the price we must pay for relying on the stationarity assumption.

The fitting residuals are shown in Figure 7.30. The physical altitude residuals tend to be rectangles, each the duration of a track. While the satellite is flying over the backside of the earth, the ocean surface changes altitude because of tides and the depressed centers of moving eddies. The fitting residuals (right side) are very fuzzy. They appear to be "white," though with ten thousand points crammed onto a line a couple inches long, we cannot be certain. We could inspect this further. If the residuals turn out to be significantly non-white, we might do better to change $\frac{d}{dt}$ to a PEF along the track.

7.10 MORE IDEAS AND EXAMPLES

7.10.1 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 (**FGDR**) 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 data samples are short, 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 (7.41), where the top half of the equations represent prediction-error predicting forward in time and the second half is prediction backward.

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ \hline r_5 \\ r_6 \\ r_7 \\ r_8 \end{bmatrix} = \begin{bmatrix} y_3 & y_2 & y_1 \\ y_4 & y_3 & y_2 \\ y_5 & y_4 & y_3 \\ y_6 & y_5 & y_4 \\ \hline y_1 & y_2 & y_3 \\ y_2 & y_3 & y_4 \\ y_3 & y_4 & y_5 \\ y_4 & y_5 & y_6 \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} \tag{7.41}$$

To get the bottom rows from the top rows, we simply reverse the order of all the components within each row. That reverses the input time function. (Reversing the order within a column would reverse the output time function.) Instead of the matrix being diagonals tipping 45° to the right, they tip to the left. We could make this matrix from our old familiar convolution matrix and a time-reversal matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

It is interesting to notice how time-reversal symmetry applies to Figure 7.15. First of all, with time going both forward and backward, the residual space gets twice as big. The time-reversal part gives a selector for Figure 7.15 with a gap along the right edge instead of the left edge. Thus, we have acquired a few new regression equations.

Some of my research codes include these symmetries, but I excluded them here. Nowhere did I see that the reversal symmetry made noticeable difference in results, but in coding, it makes a noticeable clutter by expanding the residual to a two-component *residual array*.

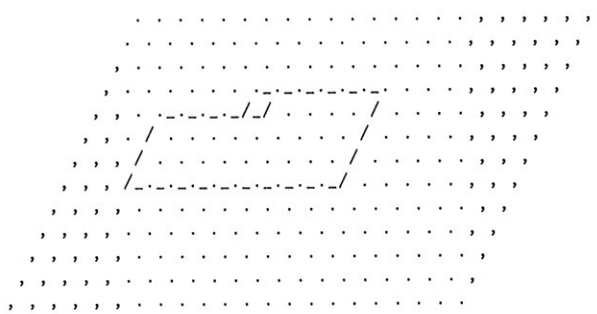
Where a data sample grows exponentially towards the boundary, I expect that extrapolated data would diverge too. You can force it to go to zero (or any specified value) at some distance from the body of the known data. To do so, surround the body of data by missing data and surround that by specification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

7.10.2 Hexagonal coordinates

In a two-dimensional plane it seems that the one-sidedness of the PEF could point in any direction. Since we usually have a rectangular mesh, however, we can only do the calculations along the axes so we have only two possibilities, the helix can wrap around the 1-axis, or it can wrap around the 2-axis.

Suppose you acquire data on a hexagonal mesh as below

and some of the data values are missing. How can we apply the methods of this chapter? The solution is to append the given data by more missing data shown by the commas below:



Now we have a familiar two-dimensional coordinate system in which we can find missing values, as well as perform signal and noise separations as described in a later chapter.

7.10.3 Interpolations with PEF do not depend on direction of time

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 7.31 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 $(a_{-2}, a_{-1}, 1)$ in Figure 7.32 yields the

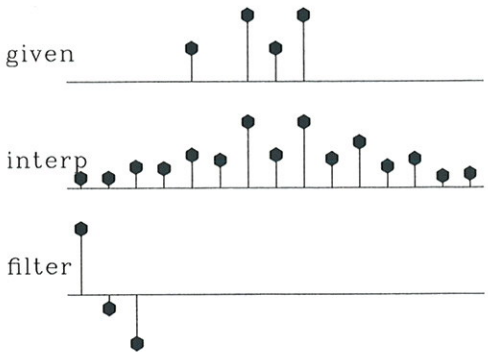


Figure 7.31: Top is known data. Middle includes the interpolated values. Bottom is the filter with the leftmost point constrained to be unity and other points chosen to minimize output power. VIEW mda/. misif90

same interpolated missing data as in Figure 7.31. 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.

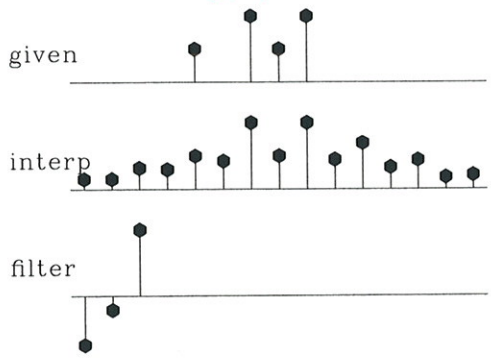


Figure 7.32: The filter here had its rightmost point constrained to be unity—i.e., this filtering amounts to backward prediction. The interpolated data seems to be identical to that of forward prediction. VIEW mda/. backwards90

7.10.4 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 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 tends to 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. As we saw in figure 7.16 non predictable signal such as white noise may be in the training data without being extended into the missing region.)

On the other hand, the **interpolation-error filter**, a filter like $(a_{-2}, a_{-1}, 1, a_1, a_2)$, fills with the wrong spectrum. To confirm this, 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 7.33 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 7.34. It

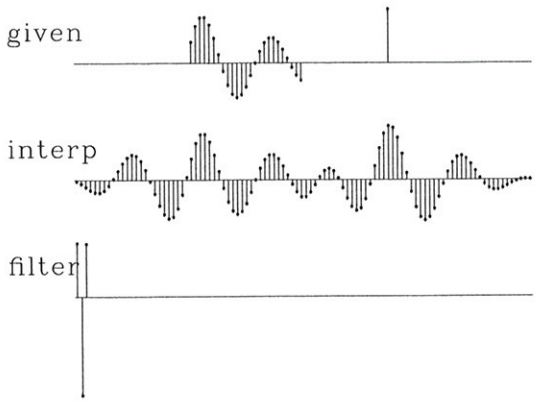


Figure 7.33: Top is synthetic data with missing portions. Middle includes the interpolated values. Bottom is the filter, a *prediction-error* filter which may look symmetric but is not quite. VIEW mda/. exp90

is the same as Figure 7.33, except that the filter is constrained in the middle. Notice that the extended data does *not* have the spectrum of the given data—the wavelength is much shorter. The boundary between real data and extended data is not nearly as well hidden as in Figure 7.33.

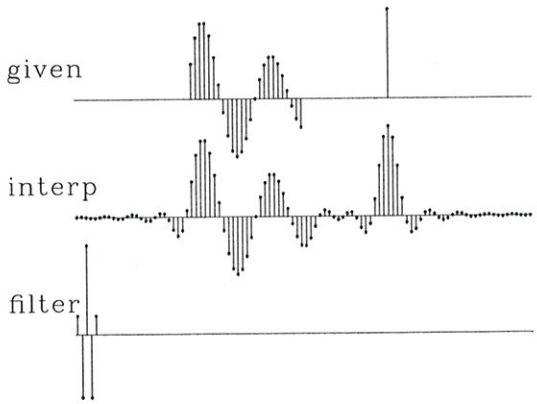


Figure 7.34: Top is the same synthetic data. Middle includes the interpolated values. Bottom is the filter, an *interpolation-error* filter. VIEW mda/. center90

7.10.5 Hermeneutics

> In seismology, the data is usually better than the theory. Data misfit alerts us to opportunity. The earth knows something we have not yet learned.

Hermeneutics is the study of the methodological principles of interpretation. Historically, it refers to bible study. Never-the-less, it seems entirely appropriate for Geophysical Estimation. If Albert's book is "Inverse Problem Theory" and mine is "Inverse Problem Practice", and if the difference between theory and practice is smaller in theory than it is in practice, then there are two fundamental questions:

1. In theory, what is the difference between theory and practice? In theory, the difference is data error.
2. In practice, what is the difference between theory and practice? One suggestion is that the discrepancy is entirely due to inadequate modeling. It is well known that geophysical data is highly repeatable. The problem is that the modeling neglects far too much.

Here is a perspective drawn from analysis of the human genome: "The problem is that it is possible to use empirical data to calibrate a model that generates simulated data similar to the empirical data. The point of using such a calibrated model is to be able to show how strange certain regions are if they don't fit the simulated distribution, which is based on the empirical distribution." In other words, "inversion" is just the process of calibrating a model. To learn something new we track down the *failures* of such models.

7.11 NONSTATIONARY OPERATORS

Nonstationary data is that with spectra changing in time or space. Nonstationary data usually calls for nonstationary operators. We need those to get our residual white.

My past work did not meet my standards for this book; but now I think I know what I should have done. This omission disappoints me because nonstationary data is so very prevalent.

7.11.1 Time-variable 1-D filter

My first go at nonstationarity was a time-variable prediction-error filter. Unfortunately at the present state of computer hardware the method is not suitable for multidimensional data. This method did work well in one dimension. Figure 7.35 shows synthetic data with time-variable deconvolution. (Details are in the document labeled "Unfinished" at my web site.)

The method is simple. Every point on the signal has its own filter. Since each data point has a multipoint filter the PEF-design regression is severely underdetermined, but a workable regularization is forcing filters to change slowly. I minimized their gradient.

As we hope for deconvolution, events are compressed. The compression is fairly good, even though each event has a different spectrum. What is especially pleasing is that satisfactory results are obtained in truly small numbers of iterations (about three). The example is for two free filter coefficients $(1, a_1, a_2)$ per output point.

Dip spectra are commonly time and space variable. In multidimensional spaces we mostly struggle for data memory. Needing a filter array for each data point is abhorrent.

7.11.2 Patching

My second go at nonstationarity was patching. A big block of data is chopped into overlapping little blocks. The adjoint operation merges the little blocks back into a big block.

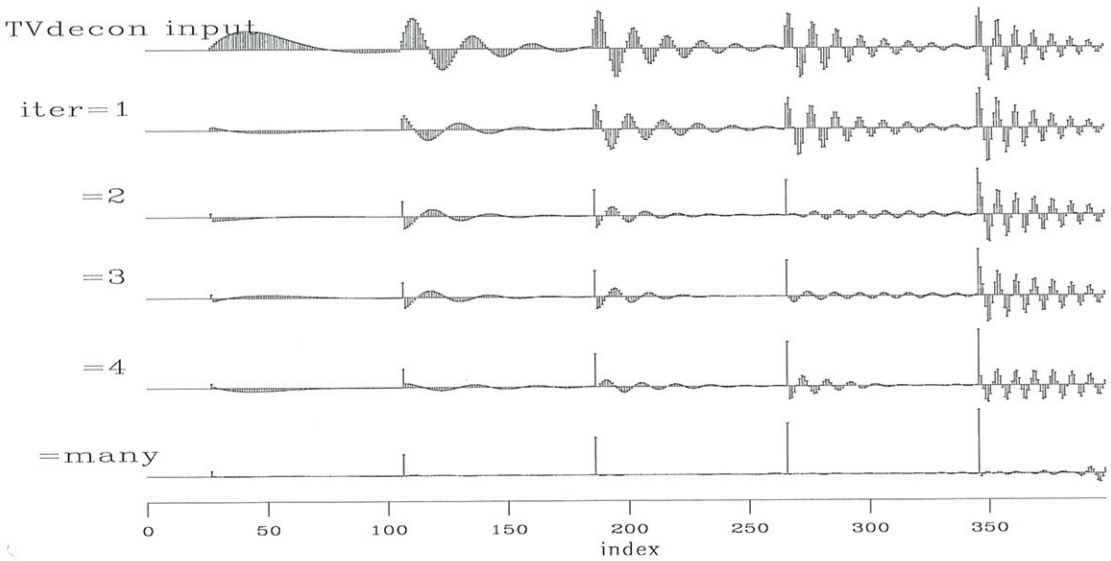


Figure 7.35: Time variable deconvolution with two free filter coefficients and a gap of 6. mda/. tvdecon90

The inverse operator is easily found by passing a big plane full of ones thru the operator and back. This gives a measure of overlap, i.e. finds a bin count for a divisor to convert the adjoint to an inverse. Weighting functions of space may also be introduced and the inverse likewise calculated. Patching would appear to be well suited to modern parallel computer architectures.

Patches need not be equal in size. They need not be rectangular. Reflection seismologists immediately recognize the need for wedge-shaped patches in the space of time and source-receiver offset.

This method does work, but there are drawbacks. A big drawback is the many parameters required to specify patch sizes and overlaps. When PEF's are designed in blocks, then care must be taken to use internal filtering and attend to the fact that output lengths are shorter than input lengths. You live in fear that patch boundaries may be visible in your output. The many parameters increase the likelihood of miscommunication between the coder and the user. The many parameters also require effort and experience to optimize (tune).

7.11.3 Store the filter on a coarser mesh

The first coarse-mesh-filter idea is to keep the filter constant over a range of values in time and space. Such a filter would be easily stored on a coarser mesh, so the memory devoted to filters could be significantly less than the data. But this idea evokes fear that we will see the blocky boundaries in outputs.

Bob Clapp (who has exercised nonstationary filtering in large scale environments) suggests we should linearly interpolate filters from the coarser mesh. It can become costly,

but economics are hard to figure in this age of rapidly changing computer architectures. Whether and how the coarse-mesh-filter idea is integrated with the helix transform is a topic which to my knowledge has not yet been attacked. The challenge for the analyst/coder is to produce filters interpolated from a grid in an environment that can be widely shared among many applications and with many people.

7.12 IRREGULARLY SPACED SIGNALS CALL FOR SPARSE MATRICES

We have accomplished much using operators (function pairs) instead of matrices (data structures). But seismological data suggests a need for representing operators as sparse matrices. Often we move data from irregular meshes to regular ones. We want the regular mesh for data viewing, for filtering, and for Fourier transformation. When we have data signals instead of simply data values we could use the same program at each point in time. But sparse matrices might be vastly faster. Mathematically, think of a large collection of least-squares regressions (N time points) each with the same time-independent operator $\mathbf{F}^*\mathbf{F}$.

$$\mathbf{F}^*\,\mathbf{F}\,[\mathbf{m}_1\mathbf{m}_2\mathbf{m}_3\cdots\mathbf{m}_N] \quad\approx\quad [\mathbf{b}_1\mathbf{b}_2\mathbf{b}_3\cdots\mathbf{b}_N] \tag{7.42}$$

Instead of iterating the same operator at each time, efficiency might be gained by approximating $(\mathbf{F}^*\mathbf{F})^{-1}$. How big is $\mathbf{F}$? Data here has these sizes:

Galilee	132,044
Seabeam	368,945
Madagascar	507,961
Vesuvius	490,000

Model sizes are typically larger because of zero padding. These are toy problems solvable in a few minutes using operators. Each would be an awesome big matrix, but if represented sparsely, special techniques become applicable.[5]

A sparse matrix $F_{i,j}$ is a list of 3 columns and K rows. Each row contains (matrix element, i value, j value). Observe how we multiply a sparse matrix times a vector, say $\mathbf{d} = \mathbf{F}\mathbf{m}$ or $d_i = \sum_j F_{i,j} m_j$.

```
do k=1,K
     data(i(k)) += matrix(k) * model(j(k))
```

Now review: Let $m(x_i, y_j)$ be unknown scalar values on a regular 2-D grid packed into a model vector $\mathbf{m}$. Among these scalar model values are data values d_k packed into a data vector $\mathbf{d}$. The linear interpolation operator $\mathbf{L}$ creates synthetic data by $\mathbf{L}\mathbf{m} = \mathbf{d}_{\text{modeled}}$. We have real data $\mathbf{d}$ and seek the model $\mathbf{m}$. We minimize $\|\mathbf{d} - \mathbf{d}_{\text{modeled}}\|$ by least squares. Additionally, since the model mesh is dense having many more points than the data space, we need a regularization operator we'll call $\mathbf{A}$. It might be gradient, laplacian, or a 2-D

[5] I thank Michael Saunders for explaining this to me. He suggests the technique known as sparse QR and recommends a program by Tim Davis. http://www.cise.ufl.edu/research/sparse/SPQR/

filter we call a prediction error filter (PEF). For finding $\mathbf{m}$ we have two goals, a data fitting goal $\mathbf{0} \approx \mathbf{Lm} - \mathbf{d}$ and a regularization goal (model styling goal) $\mathbf{0} \approx \mathbf{Am}$.

$$\begin{aligned} \mathbf{0} &\approx \mathbf{Lm} - \mathbf{d} \\ \mathbf{0} &\approx \epsilon\, \mathbf{Am} \end{aligned} \tag{7.43}$$

To see the regression (7.42) we need $\mathbf{b} = \mathbf{L}^*\mathbf{d}$ and $\mathbf{F}$

$$\mathbf{F} = \begin{bmatrix} \mathbf{L} \\ \epsilon\, \mathbf{A} \end{bmatrix} \tag{7.44}$$

A singular contribution of this book is multidimensional PEFs. Unlike gradient and Laplacian they are easily invertible offering solution by preconditioner $\mathbf{p}$ where $\mathbf{m} = \mathbf{A}^{-1}\mathbf{p}$. In the simplest case $\mathbf{A}^{-1}$ would be leaky integration, trivially implemented with recursion. Recursion allows easy solution to these huge problems. In the QR algorithm recursion appears as the triangular matrix $\mathbf{R}$.

Well my friends, we have come a long way; we have made much progress; but I have become old. I can help a while longer, but from here on, it is for you to carry the ball.

Chapter 8

Future work

In wrapping up this book, I found myself with a collection of clever but inadequately finished projects. I also found two major directions I wish to have run. First, on nonstationary data because it is so very prevalent. Second, while we have covered unevenly spaced data, unevenly spaced signals invite new techniques. Here lies the trail ahead.

8.1 NONSTATIONARY OPERATORS

Nonstationary data is that with spectra changing in time or space. Nonstationary data usually calls for nonstationary operators. We need those to accelerate solutions and to transform residuals to whiteness (IID).

NOTE: This is all discussed almost verbatim in 7.11.1

8.1.1 Time-variable 1-D filter

PEF

My first go at nonstationarity was a time-variable ~~prediction-error filter~~. Unfortunately at the present state of computer hardware, the method is not suitable for multidimensional data. This method did work well in one dimension. Figure 8.1 shows synthetic data with time variable deconvolution. (Details are in the document labeled "Unfinished" ~~at~~ my web site.)

one word website

The method is simple. Every point on the signal has its own filter. ~~Since~~ each data point has a multipoint filter, the PEF-design regression is severely underdetermined, but a workable regularization is forcing filters to change slowly. I minimized their gradient.

Because

As we hope for deconvolution, events are compressed. The compression is fairly good, even though each event has a different spectrum. What is especially pleasing is that satisfactory results are obtained in truly small numbers of iterations (~~about~~ three). The example is for two free filter coefficients $(1, a_1, a_2)$ per output point.

approximately

Dip spectra are commonly time and space variable. In multidimensional spaces, we ~~mostly~~ struggle for machine memory. Needing a filter array for each data point is abhorrent.

primarily

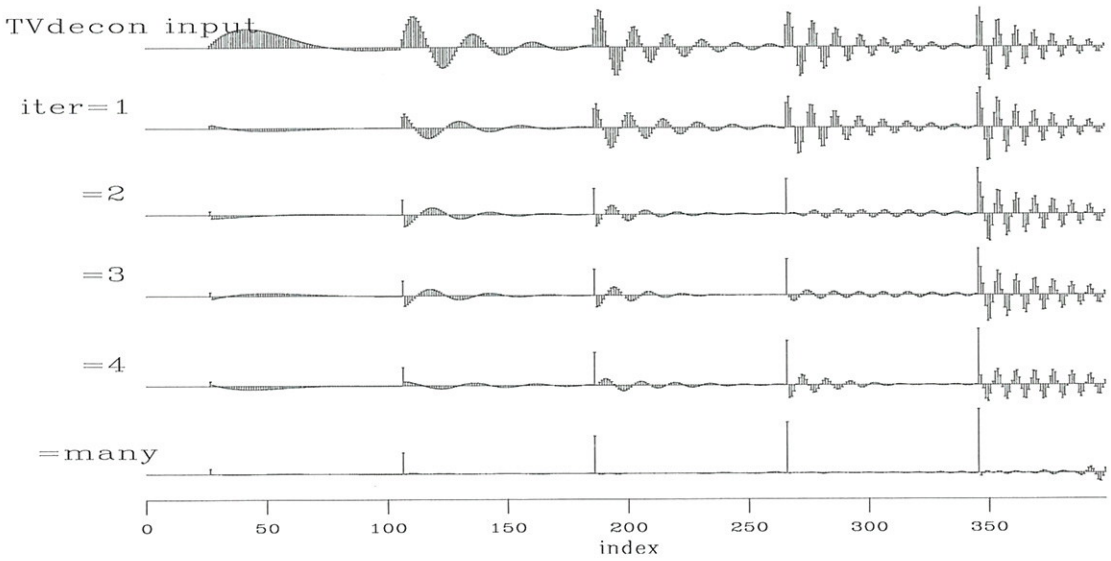


Figure 8.1: Time variable deconvolution with two free filter coefficients and a gap of 6. mda/. tvdecon90

NOTE: Verbatim from 7.11.2

8.1.2 Patching

My second go at nonstationarity was patching. A big block of data is chopped into overlapping little blocks. The adjoint operation merges the little blocks back into a big block. The inverse patching operator is easily found by passing a big plane full of ones thru the operator and back. This gives a measure of overlap, i.e. finds a bin count for a divisor to convert the adjoint to an inverse. Weighting functions of space may also be introduced and the inverse likewise calculated. Patching would appear to be well suited to modern parallel computer architectures.

Patches need not be equal in size. They need not be rectangular. Reflection seismologists immediately recognize the need for wedge-shaped patches in the space of time and source-receiver offset.

This method does work, but there are drawbacks. A big drawback is the many parameters required to specify patch sizes and overlaps. When PEFs are designed in blocks, then care must be taken to use internal filtering and attend to the fact that output lengths are shorter than input lengths. You live in fear that patch boundaries may be visible in your output. The many parameters increase the likelihood of miscommunication between the coder and the user. The many parameters also require effort and experience to optimize (tune).

NOTE: Verbatim from 7.11.3

8.1.3 Store the filter on a coarser mesh

The first coarse-mesh-filter idea is to keep the filter constant over a range of values in time and space. Such a filter would be easily stored on a coarser mesh, so the memory devoted to filters could be significantly less than the data. But this idea evokes fear that we will see

the blocky boundaries in outputs.

Bob Clapp (who has exercised nonstationary filtering in large scale environments) suggests we should linearly interpolate filters from the coarser mesh. It can become costly, but economics are hard to figure in this age of rapidly changing computer architectures. Whether and how the coarse-mesh-filter idea is integrated with the helix transform is a topic which to my knowledge has not yet been attacked. The challenge for the analyst/coder is to produce filters interpolated from a grid in an environment that can be widely shared among many applications and with many people.

8.2 SCATTERED SIGNALS CALL FOR SPARSE MATRICES

We have accomplished much using operators (function pairs) instead of matrices (data structures). We soon here see that scattered seismic data suggests a need for switching from operators to sparse matrices. Often we move scattered data to a regular mesh. We want the regular mesh for data viewing, filtering, correlation, and for Fourier transformation. When we have data signals instead of simply data values it seems we must repeatedly use the same iterative program at each point in time. But sparse matrices might be vastly faster. To see why, represent a large collection of least-squares regressions (N time points) each with the same time-independent operator $\mathbf{F}^*\mathbf{F}$.

$$\mathbf{F}^*\,\mathbf{F}\,[\mathbf{m}_1\mathbf{m}_2\mathbf{m}_3\cdots\mathbf{m}_N] \quad\approx\quad [\mathbf{b}_1\mathbf{b}_2\mathbf{b}_3\cdots\mathbf{b}_N] \tag{8.1}$$

Instead of iterating the same operator at each time, efficiency might be gained by approximating $(\mathbf{F}^*\mathbf{F})^{-1}$. How big is $\mathbf{F}$? Data here has these sizes:

Galilee	132,044
Seabeam	368,945
Madagascar	507,961
Vesuvius	490,000

Model sizes are typically larger because of zero padding. These are toy problems solvable in a few minutes using operators. Industrial settings have comparable numbers of signals as we had values. Luckily industrial and academic settings also have signal clusters in much smaller numbers, say hundreds to a few thousands. I asked Michael Saunders for an approach using operators. He suggested instead we consider sparse matrices, in particular a technique known as sparse QR.[1]

Think of a sparse matrix $F_{i,j}$ as a list of 3 columns and K rows. Each row contains (matrix element, i value, j value). Observe how to multiply a sparse matrix times a vector, say $\mathbf{d} = \mathbf{F}\mathbf{m}$ or $d_i = \sum_j F_{i,j} m_j$.

```
do k=1,K
     data(i(k)) += matrix(k) * model(j(k))
```

[1] Michael Saunders recommends http://www.cise.ufl.edu/research/sparse/SPQR/ a sparse QR method and code by Tim Davis.

Here follows a review to connect equation(8.1) more explicitly to this book: Let $m(x_i, y_j)$ be unknown scalar values on a regular 2-D grid packed into a model vector $\mathbf{m}$. Among these scalar model values are data values d_k packed into a data vector $\mathbf{d}$. The linear interpolation operator $\mathbf{L}$ creates synthetic data by $\mathbf{Lm} = \mathbf{d}_{\text{modeled}}$. We have real data $\mathbf{d}$ and seek the model $\mathbf{m}$. We minimize $\|\mathbf{d} - \mathbf{d}_{\text{modeled}}\|$ by least squares. Additionally, since the model mesh is dense, having many more points than the data space, we need a regularization operator we'll call $\mathbf{A}$. It might be gradient, laplacian, or a 2-D filter we call a prediction error filter (PEF). For finding $\mathbf{m}$ we have two goals, a data fitting goal $\mathbf{0} \approx \mathbf{Lm} - \mathbf{d}$ and a regularization goal (model styling goal) $\mathbf{0} \approx \mathbf{Am}$.

$$\begin{aligned} \mathbf{0} &\approx \mathbf{Lm} - \mathbf{d} \\ \mathbf{0} &\approx \epsilon\, \mathbf{Am} \end{aligned} \tag{8.2}$$

To see the regression (8.1) we need $\mathbf{b} = \mathbf{L}^*\mathbf{d}$ and $\mathbf{F}$

$$\mathbf{F} = \begin{bmatrix} \mathbf{L} \\ \epsilon\, \mathbf{A} \end{bmatrix} \tag{8.3}$$

A singular contribution of this book is multidimensional PEFs. Unlike gradient and Laplacian they are easily invertible offering solution by preconditioner $\mathbf{p}$ where $\mathbf{m} = \mathbf{A}^{-1}\mathbf{p}$. In the simplest case $\mathbf{A}^{-1}$ would be leaky integration, trivially implemented with recursion. Recursion allows easy solution to these huge problems. In the QR algorithm recursion appears as the triangular matrix $\mathbf{R}$.

Well my friends, we have come a long way. We have made much progress, and we see another path for more. Meanwhile, I have become old. I can help a while longer, but from here on, it is for you to carry the ball.

almost all verbatim from 7.12

Chapter 8

Industrial seismology sampler

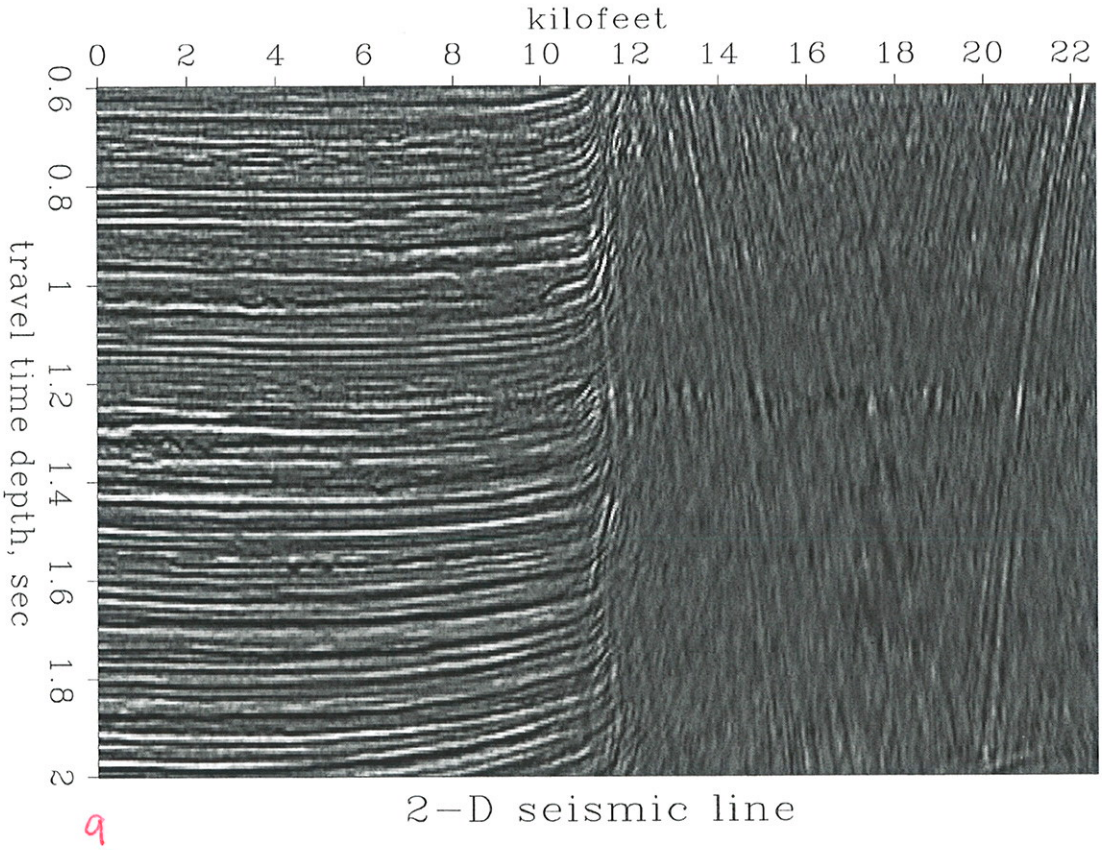


Figure 8.1: A 2-D seismic survey line. Left half is layers. Right half is a salt dome. Salt flows upwards, dragging hence bending upwards the adjacent layers. There are no reflections inside the salt. In the salt are only artefacts of data processing. rez/. line

Industrial seismology is a big consumer of technologies developed in this book. This book steers away from seismology because of its complexity (and because I have written other books devoted to seismology). Figure 8.1 is a traditional single survey line of the kind that dominated the industry in the 1960s..

This book is merely a "warm up" to today's industry. In earlier chapters you saw tiny data sets manageable in a small desktop computer. Industrial seismology is done both on land and at sea. These examples are marine. Receivers measure hydrophone voltage in a five dimensional data space, two surface coordinates (x_s, y_s) for each source pop, two more (x_r, y_r) for each receiver, and the echo delay time t. It has the 3-D model space of our world (x, y, z), though on the cube here we do not see z, but t, the vertical seismic travel time.

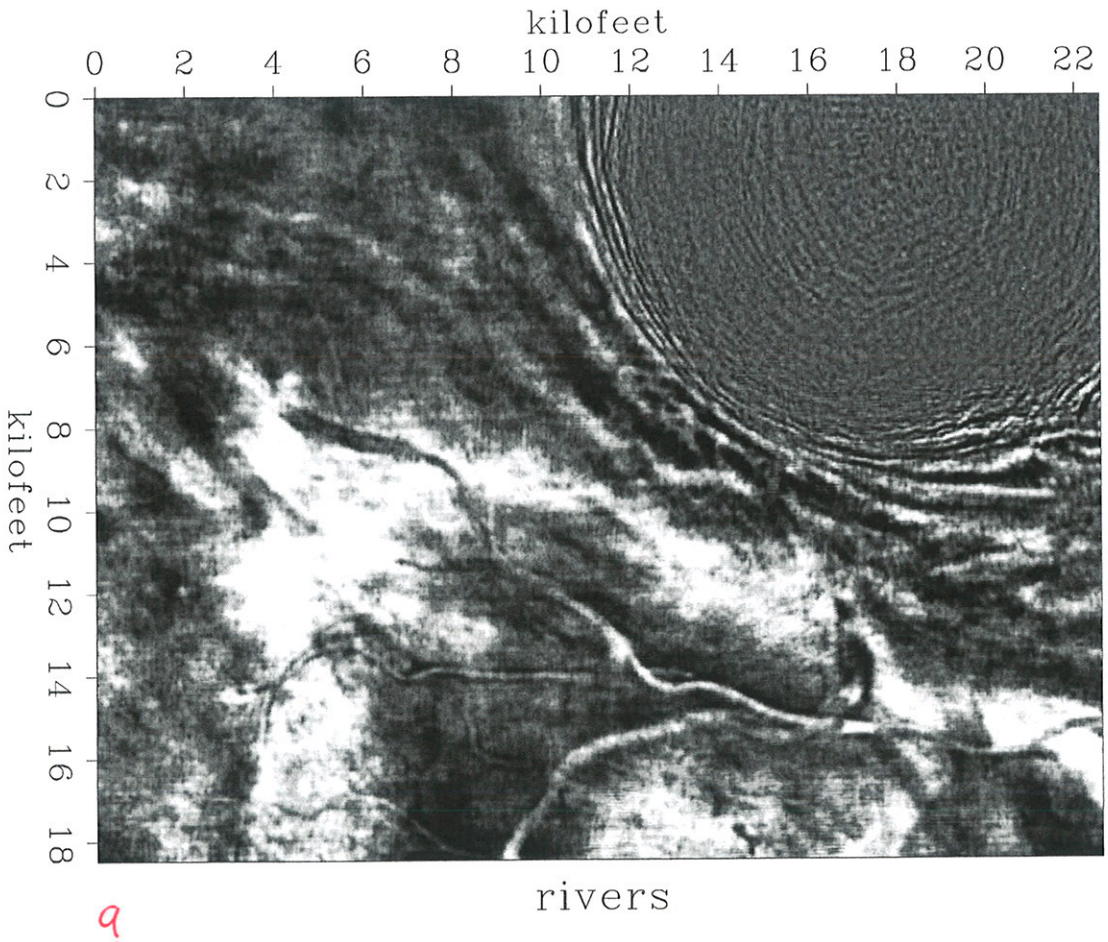


Figure 8.2: At t =1.387s (about 1.4km depth): The upper right circular corner is a salt dome. River meanders from about a million years ago. River meanders are a common sight in 3-D reflection seismic images. Rivers typically migrate significant distances in the 7000 years between our resolution slices. Some depth ranges contain no rivers. Such correspond to eras when these layers were being laid down lay beneath the sea. rez/. rivers

Illustrations here may look like data, but they are slices from model space, On figure 8.1 the alternating voltages in the seismic microphone suggest black-white physical layering in the earth. While this is surely indicative, higher frequency filtration would yield more layers. Keep this in mind as you examine Figure 8.2, a horizontal slice inside the earth at a constant depth (travel-time depth t = 1.387sec). Local outline shapes are truly meaningful here while black/white polarities hardly so. Whether a river is white in a black background or black in a white background is an accidental function of overall travel time and spectrum. What

The upper right corner of the constant depth slices shows a circular region. This is salt. Salt, like ice, seems brittle, but under pressure it flows like a liquid. Before the past million years ago before the sediments of this cube were laid down there was a salt lake here that eventually dried and was buried beneath the sand, shale, and carbonates that became this cube. Salt is lighter than rock, and so eventually it erupted like a pimple on the face of the earth, a pimple two miles wide. No oil in here, but the bent up layers aside it seen in Figure 8.1 are excellent prospects. Salt flow is a dominant feature in the Gulf of Mexico.

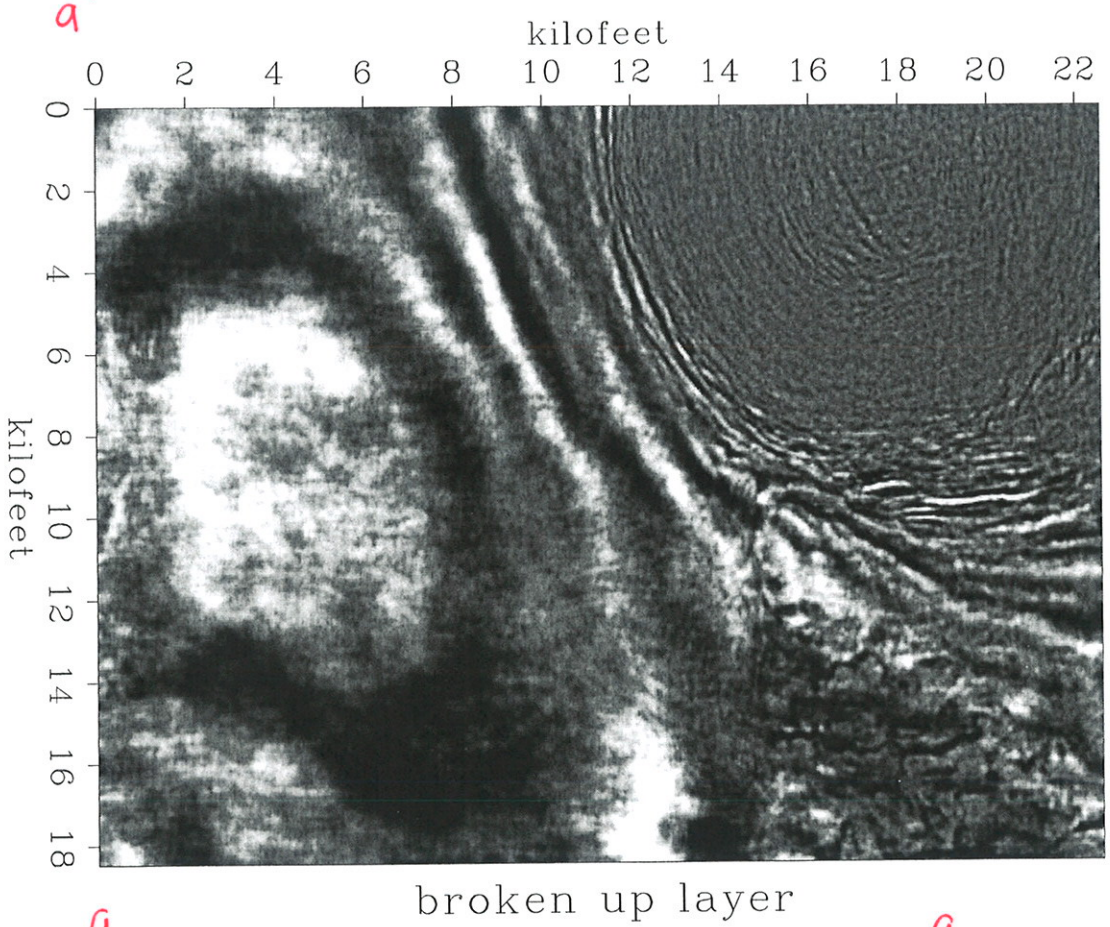


Figure 8.5: At t=1.938s: To the east of the fault noted already in Figure 8.4 is a broken up layer with a "wormy" appearance. I do not know what it is. Curiously it is found only on one side of the fault. rez/. worms

This data cube (actually model space) is about 20 years old. It came from Chevron via David Lumley to James Rickett. It is textbook quality 3-D data from the Gulf of Mexico. It would have taken the survey company about a month to acquire, and it would cost the oil company (group maybe) about ten million dollars.

A ship with an air gun towed a 7km long cable with a thousand hydrophones. Today there would be several gun boats. The recording ship would trail about a dozen streamers separated about 150 meters. World-wide there are about 50 marine survey teams working continuously. The half dozen largest seismic survey companies together sell about ten billion

is significant is the rings surrounding the dome. These are a consequence of the upward bending layers you saw in figure 8.1.

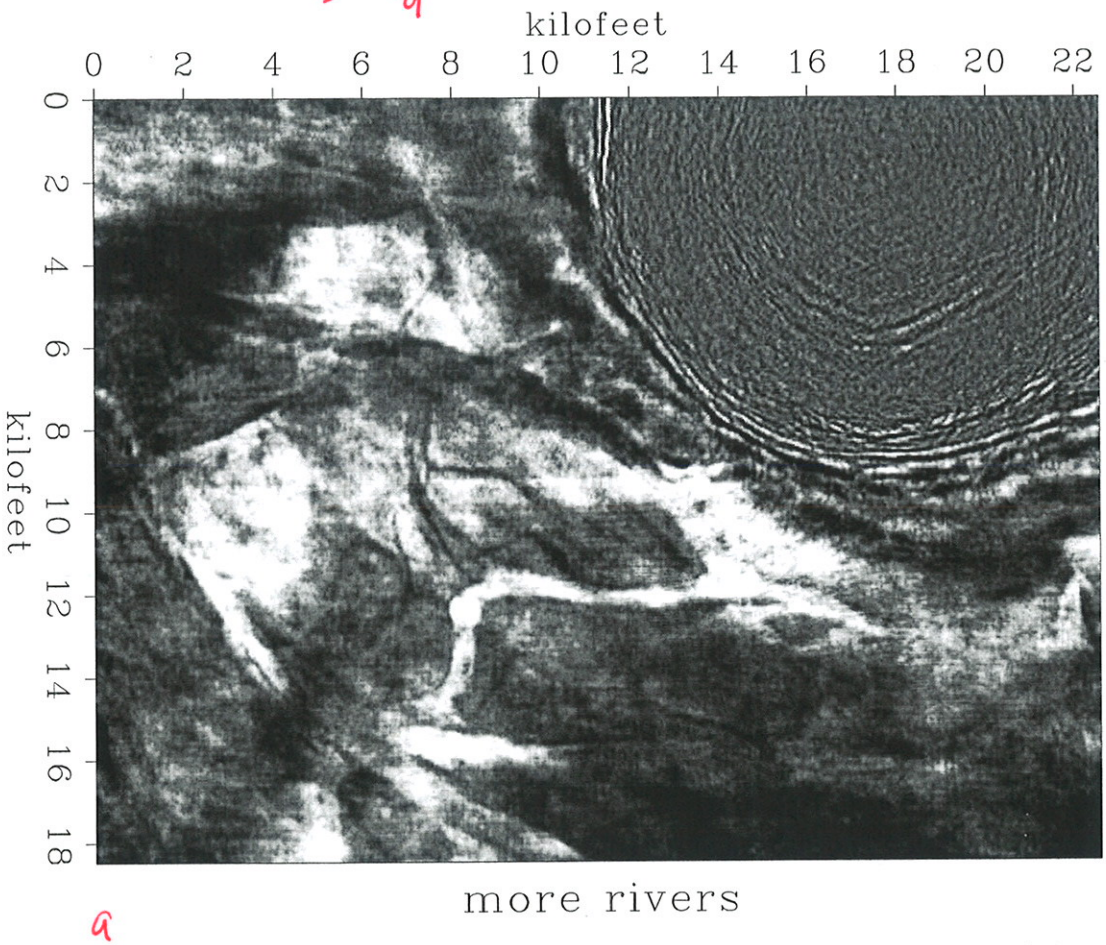


Figure 8.3: At t=0.888s: The (x, y) plane shown here is grabbed from a volume of slices separated by 6ms, about 18 feet. Slice to slice represents about 7,000 years of sedimentary deposition in the Gulf of Mexico. Top to bottom is about a million years (about the age of the human species). Think of the creatures in all those rivers, their ancient worlds. Awesome, isn't it? rez/. rivers2

Seismic waves here are a little faster than 2 km/sec, but they must go both down into the earth and up again, so the bottom of the time axis is a little more than 2 km deep. A ship sails from west to east creating an x-axis, 22 kilofeet long, a little under 5 miles. Where the vertical axis is not north-south it is travel time. Typically that axis might run to 5 seconds. Here for space limitations, it runs less than 2s. All the planes you see in this chapter come from one $292 \times 451 \times 551$ cube of 72 megapixels, a subset of a larger volume of model space.

You may be seeing paper or images of what's on paper, but what you see is merely two-dimensional slices thru the 3-D model space. I can plunge into these volumes, panning and zooming. Thanks to my colleague Bob Clapp and others like him, after some years we may escape the constraints of PDF files and deliver such experiences to readers outside our lab.