SEISMIC TRACE INTERPOLATION WITH NONSTATIONARY PREDICTION-ERROR FILTERS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF GEOPHYSICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Sean Crawley
June 2001

Abstract

Theory predicts that time and space domain prediction-error filters (PEFs) may be used to interpolate aliased signals. I explore the utility of the theory, applying PEF-based interpolation to aliased seismic field data, to dealias it without lowpass filtering by inserting new traces between those originally recorded. But before theoretical potential is realized on 3-D field data, some practical aspects must be addressed.

Most importantly, while PEF theory assumes stationarity, seismic data are not stationary. We can divide the data into assumed-stationary patches, as is often done in other interpolation algorithms. We interpolate with PEFs in patches, and get near-perfect results in those parts of the data where events are mostly local plane waves, lying along straight lines. However, we find that the results are unimpressive where the data are noticeably curved. As an alternative to assumed-stationary patches, I calculate PEFs everywhere in the data, and force filters which are calculated at adjacent coordinates in data space to be similar to each other. The result is a set of smoothly-varying PEFs, which we call adaptive or nonstationary. The coefficients of the adaptive PEFs constitute a large model space. Using SEP's helical coordinate, we precondition the filter calculation problem so that it converges in manageable time.

To address the difficult problem of curved events not fitting the plane wave model, we can control the degree of smoothness in the filters as a function of direction in data coordinates. To get statistically robust filter estimates, we want to maximize the area in data space over which we estimate a filter, while still approximately honoring stationarity. The local dip spectrum on a CMP gather is nearly constant in a region which is elongated in the radial direction, so I estimate PEFs that are smooth along radial lines but which may vary quickly with radial angle. In principle that addresses the curvature issue, and I find it performs well in practice on strongly curved data, noisy data, and data with somewhat irregular acquisition geometry or statics.

Preface

All of the figures in this thesis are marked with one of the three labels: [ER], [CR], and [NR]. These labels define to what degree the figure is reproducible from the data directory, source code and parameter files provided on the web version of this thesis [1].

ER denotes Easily Reproducible. The author claims that you can reproduce such a figure from the programs, parameters, and data included in the electronic document. We assume you have a UNIX workstation with Fortran90, X-Window system, and SEP's freely distributed software at your disposal. Before the publication of the electronic document, someone other than the author tests the author's claim by destroying and rebuilding all ER figures.

CR denotes Conditional Reproducibility. The author certifies that the commands are in place to reproduce the figure if certain resources are available. To find out what the required resources are, you can inspect a corresponding warning file in the document's result directory. For example, you might need a large or proprietary data set. You may also need a super computer, or you might simply need a large amount of time on a workstation.

NR denotes Non-Reproducible. This class of figure is considered non-reproducible. Figures in this class are scans and artists' drawings. Outputs of interactive processing are labeled NR.

[1] http://sepwww.stanford.edu/public/docs/sep101

Many of the figures in Chapter 2 do not lend themselves easily to automatic reproduction, because several of the standard processing steps, like NMO, Radon, and stack, were done with the commercial package ProMAX, which favors pointing and clicking. These figures are all marked [NR].

Figures in Chapter 3 and 4 are mostly marked [ER]. The software requires a Fortran90 compiler. While the figures generally take less than a half hour to generate on a single processor, they tend to consume a large amount of memory, so that reproducing them on a small workstation may be impractical. Some of the figures in Chapter 4 compare results of time-domain interpolation [ER] and ProMAX's FX interpolation [NR]. These are marked [ER], but only the time-domain panels actually reproduce themselves.

Acknowledgements

It took me long enough to get this done, but it would have taken forever without help, both in the form of encouragement and emotional support, and in the form of other people physically performing tasks that I would otherwise have had to do.

I owe thanks to Sergey Fomel and Bob Clapp, who have worked on tracks parallel to the one in this thesis at various points over the last few years. I benefited from many problem-solving discussions with Bob. I also benefited by stealing much of my early software from Sergey.

Thanks also to James Rickett and Morgan Brown, who (along with Sergey and Bob) spent a lot of their time fixing and maintaining the machines and software at SEP, and who also had a number of useful things to say about my thesis over the last several years.

Thanks to Jon Claerbout, who originally suggested the change from blob smoothing to radial smoothing. That suggestion turned out to be really useful to this thesis.

And much thanks to everyone who has been supportive and patient towards me over the last six years, especially my family, my office mates Bob and Marie, my friends Shilliday and Woodburn, and my cats Ellis and Betty. Most of all to my wife Colleen, who is amazingly patient and supportive.

Contents

List of Figures

Chapter 1

Introduction

Reflection seismic data are nearly always adequately sampled in time. Four millisecond sampling is typical, giving a temporal Nyquist of 125 Hz, more than adequate since the signal tends to die out somewhere in the 60-100 Hz range. In contrast, seismic data is often inadequately sampled along various spatial axes. If there is more signal, improving temporal sampling is easy. Improving spatial sampling is expensive, because it requires more equipment, time and/or manpower in the field. Seismic acquisition is expensive. Moreover, reflection surveys tend to be designed with the goal of maximizing poststack or image resolution for a fixed cost (fixed cost $\approx$ fixed number of traces), at the expense of fold. This represents the choice to spend money on well-sampled geology, in order to get more precise subsurface information, rather than on a well-sampled wavefield which would be easier to process.

All of the multichannel steps in seismic data processing depend to some degree on adequately sampled data. Inputting spatially aliased data can produce results with various sorts of artifacts and poor spatial resolution. Marfurt et al. (1996) give examples of artifacts that appear in Radon transforms of insufficiently sampled data. Spitz (1991) shows differences between migration results on well sampled and poorly sampled data.

INTERPOLATION AND ALIASING IN 3-D MARINE

Seismic data is expensive to acquire, and spatial sampling is an important limiting factor in processing. Taken together, that means it should be easy to find examples where it is useful to make more data, without going to the expense of actually recording it. It turns out that in most 3-D marine acquisition (which includes most of the seismic data recorded in recent years), there is systematic undersampling along a couple of different axes. The inline shot axis is one that tends to be undersampled. The resulting aliasing can have a significant impact on multiple suppression in particular.

Multiple suppression is a motivating example of a multichannel process that can be severely hindered by data aliasing. Interpolating data does not, in and of itself, do anything to remove multiples, but it can significantly improve the results of various multiple suppression algorithms. Radon demultiple, for example, is affected in a fairly intuitive fashion by poor sampling in offset. Where data are aliased, a single reflection event can appear to have more than one slope. Radon demultiple algorithms (and other moveout discrimination methods) are affected because at some level they amount to dip filtering the data. An aliased multiple event will have components at the expected dip value and at various other dips as well, making it difficult to remove. Figures 1.1 and 1.2 show well-sampled and poorly-sampled CMP gathers, and hyperbolic Radon transforms. Figure 1.1 shows a well-sampled gather and the same gather with one third the sampling in offset. Figure 1.2 shows hyperbolic Radon transforms of the gathers in Figure 1.1. The horizontal axis is given as milliseconds of moveout at 3000 meters of offset, instead of slowness. The well-sampled gather transforms fairly cleanly; there is not significant energy visible in areas of the transform that do not correspond to physically reasonable times and velocities. The subsampled gather produces a transform that does have the same trend, but also has artifact energy spread over most of the transform.

Radon multiple suppression (and multiple suppression in general) is often used as motivation in interpolation papers, because data aliasing occurs systematically in marine acquisition. Marine acquisition commonly looks like Figure 1.3, where a boat tows two airgun sources and three (or more) receiver cables. The number of receiver cables can be significantly larger than three, with the number of midpoints increasing proportionately, but with the pattern of

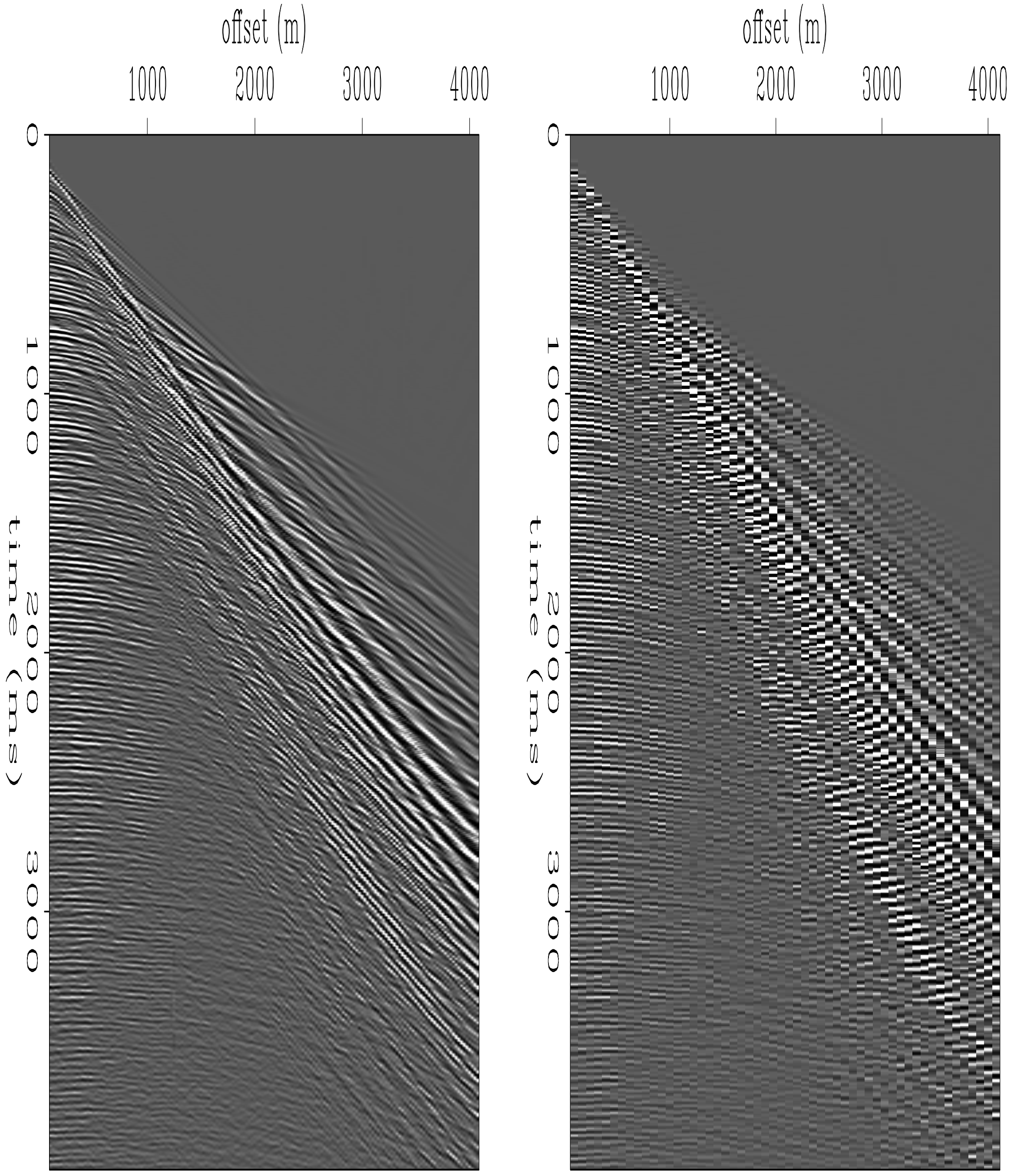


Figure 1.1: Unaliased, well-sampled data and aliased, poorly sampled data, before hyperbolic Radon transform. These gathers are Radon transformed to make the panels in Figure 1.2. int-radExpre [NR]

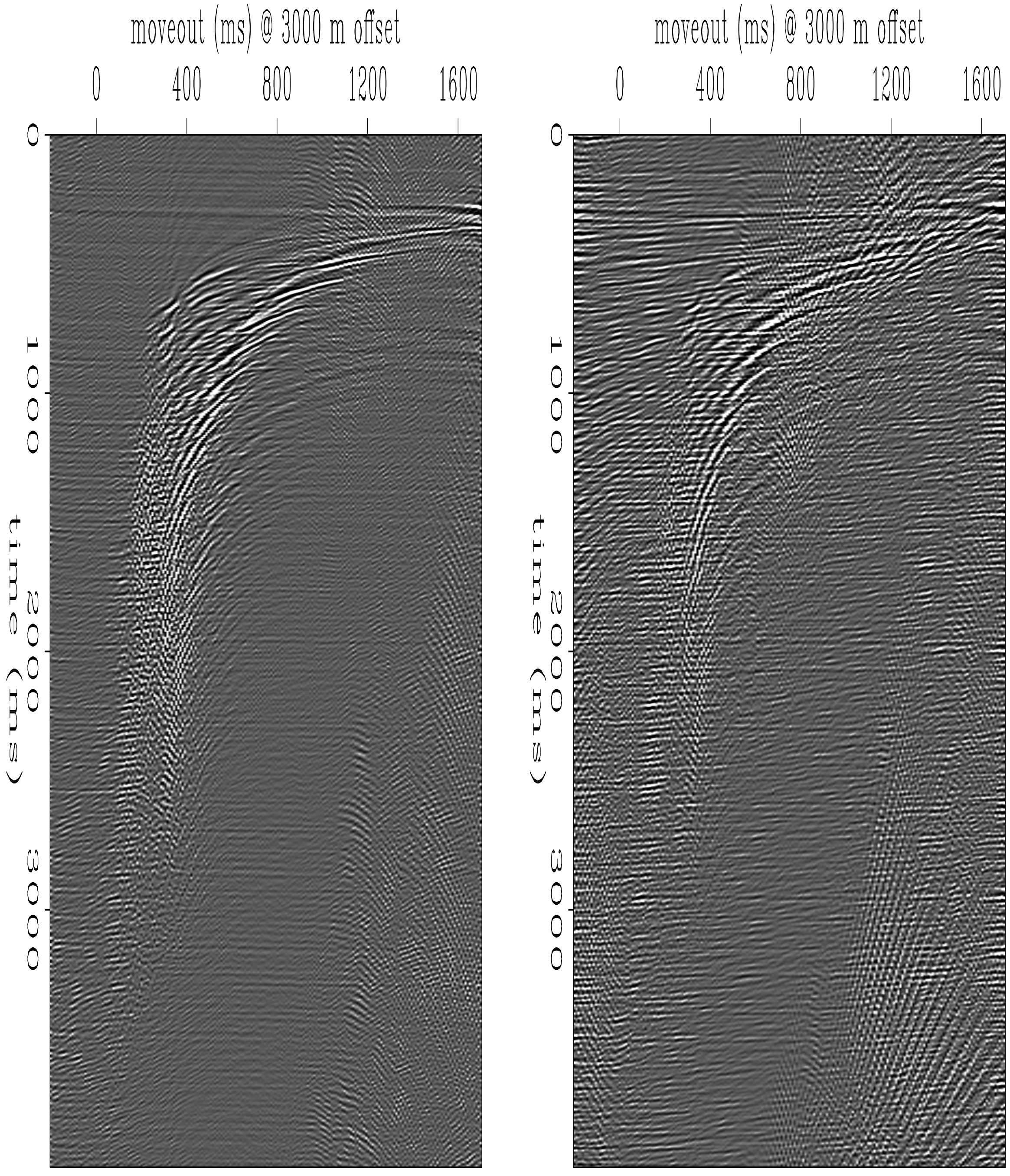


Figure 1.2: Unaliased and aliased data, after hyperbolic Radon transform. The horizontal axis is in milliseconds of moveout at 3000 meters offset, instead of slowness. The unaliased data transforms cleanly (left panel). The Radon transform of the aliased data (right panel) shows the same main trend as that of the unaliased data, but also has artifact energy spread over most of the transform. int-radExpost [NR]

midpoints staying the same. At each shot location, one or the other of the two sources fires, and the two alternate in what we call a “flip/flop” mode. Shot spacing for the pair considered together is normally equal to the receiver spacing, so that for a single inline the shot spacing is double the receiver spacing. The sources are positioned in such a way that they fire into interleaved sets of midpoints. This means there is an increase in the number of crossline midpoints, which is good for lateral resolution of the subsurface, but it also means that the fold decreases. Specifically, a given midpoint only gets a trace from every second shot, so the common-midpoint (CMP) fold is cut in half and effective source and receiver spacing is doubled.

In the older single source acquisition, CMP fold was already half the number of receivers, spread over the same offset range, meaning that unaliased shot gathers could be sorted into aliased CMP gathers. This is the so-called “sampling paradox” (Vermeer, 1990). The additional halving due to the “flip/flop” acquisition means that the trace spacing in a common midpoint gather is four times the trace spacing in a common shot gather. Figures 1.4 and 1.5 illustrate the sampling for CMP gathers in a single inline in a “flip/flop” marine survey and in a survey with a single source. With a single source, shot spacing and receiver spacing are the same, and offset spacing within a CMP gather is double that. With two sources, shot spacing is twice the receiver spacing, and offset spacing within a CMP gather is quadruple the receiver spacing. Jakubowicz (1994) gives an alternate explanation in terms of spatial frequencies. Shot gathers are antialiased in the inline direction by receiver arrays in the streamer cables, but it is not surprising for seismic events to be aliased in a CMP gather.

Other multiple removal schemes depend equally on data sampling. Seafloor-related multiples can be modeled from the recorded data by upward continuation (Berryhill and Kim, 1986). This process models first order multiples from primaries, second order multiples from first order multiples, and so on. The recorded data minus the modeled multiples yields the desired primaries. Upward continuation is a migration-type operator, and not surprisingly creates artifacts if the data are not well sampled. The artifacts make it harder to match the modeled multiples to the recorded data. Another method for modeling multiples from primaries is the surface related multiple elimination (SRME) method of Delft (Berkhout and Verschuur, 1997; Verschuur and Berkhout, 1997). In this method, the recorded traces function both as input data

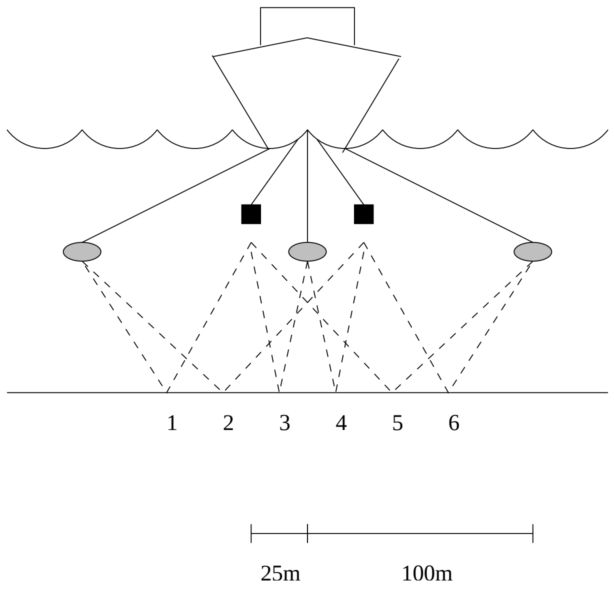


Figure 1.3: Seismic acquisition boat towing two sources and three streamers. The number of streamers (and similarly the number of midpoints) may be much larger, but the pattern of midpoints stays the same. int-boat [NR]

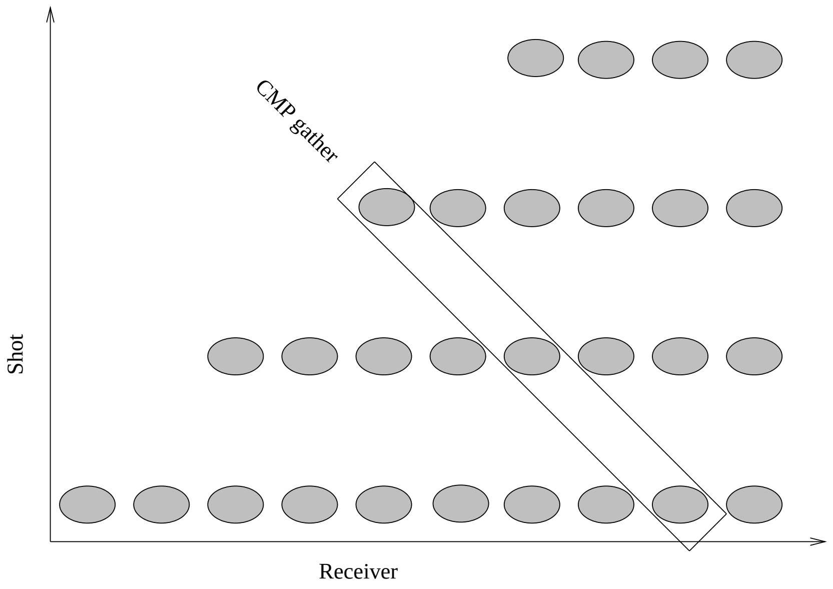


Figure 1.4: Stacking diagram for marine survey with a single source. int-stackingchart1 [NR]

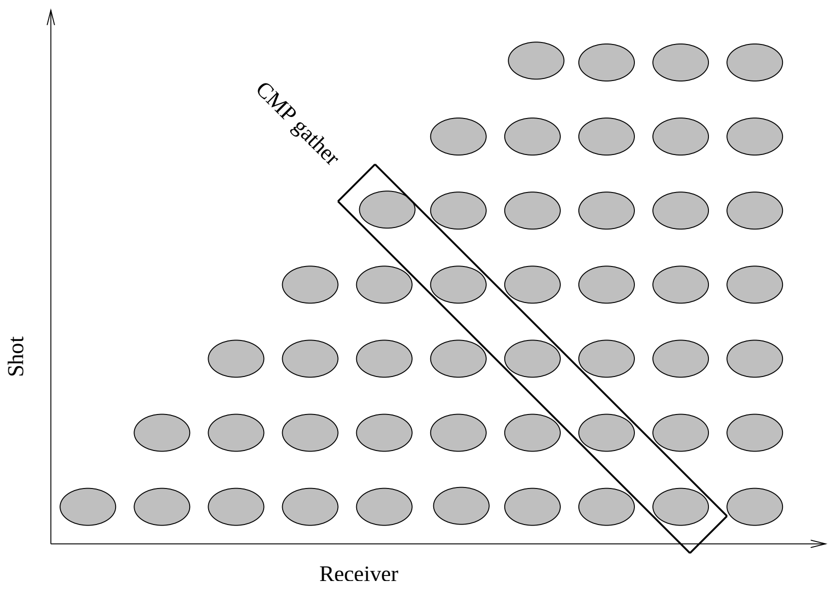


Figure 1.5: Stacking diagram for marine survey with two sources. int-stackingchart2 [NR]

and as a convolutional operator, both data aliasing and operator aliasing result from insufficient sampling. These two methods rely on reciprocity as well. Marine acquisition is often missing near source receivers, and sources are offset in the crossline direction from the receivers, so it is impossible to construct reciprocal gathers from the recorded data. This introduces an additional sort of missing data problem.

The crossline receiver axis is also poorly sampled, as suggested by Figure 1.3. The physical problems of towing something through the water make it difficult to hope for a dense crossline arrangement of receivers. Modern boats may have a dozen or more streamers, but these extra crossline receivers tend to be used to widen the aperture rather than to refine the sampling. The scarcity of crossline receiver sampling makes various kinds of potentially interesting operations difficult. True 3-D multiple suppression and 3-D full wave equation migration would both (eventually) want better crossline sampling.

Regularity versus predictability

Multiple suppression provides a good context for interpolating seismic data, because the problems related to sampling are well known and the data usually have a very regular acquisition geometry and high signal-to-noise ratio. While cable feathering makes the data irregular in physical (x, y) coordinates, it tends to be perfectly regular in (s, g).

The shape of a streamer, while not straight, tends to change slowly from one shot to the next, since the currents tend to vary with tidal cycle at the fastest. With modern GPS and navigation controls, the source locations generally follow lines which vary quite slowly in relation to the shot spacing. This regularity makes marine data suitable for a wide range of algorithms. In particular, filtering is sensible because the shot and channel regularity and the slowly changing shapes of streamer cables make it easy to put the data on a meaningful grid. Figure 1.6 shows the layout of receivers during numerous shots. The figure is vertically exaggerated to show the irregularities in the geometry. The streamer is not straight, so the data are not exactly regular in terms of shot and receiver surface locations. However, the data are regular in shot and receiver number coordinates, and because the streamer wiggles in a smooth fashion, they should also be predictable.

Land data often do not have this property. Receiver and source positions can make sharp turns, particularly when acquisition is constrained by permits and terrain. Adjacent receiver lines can have different numbers of receivers. The data are typically much noisier than marine data. Nonetheless, land data presents an interesting interpolation challenge because it is more difficult and expensive to acquire than marine data.

WHAT TO DO WITH ALIASED DATA

There are a couple of things to do about aliasing. In the 1-D case, the choices are bandpass filter the data before digitizing it, or live with it. With more dimensions come more possibilities, in particular the option to create more data. Most multichannel operators have some notion of operator antialiasing, which is essentially a bandpass on the impulse response of the operator, intelligently applied so that the maximum frequency is only as low as it has to be, changing

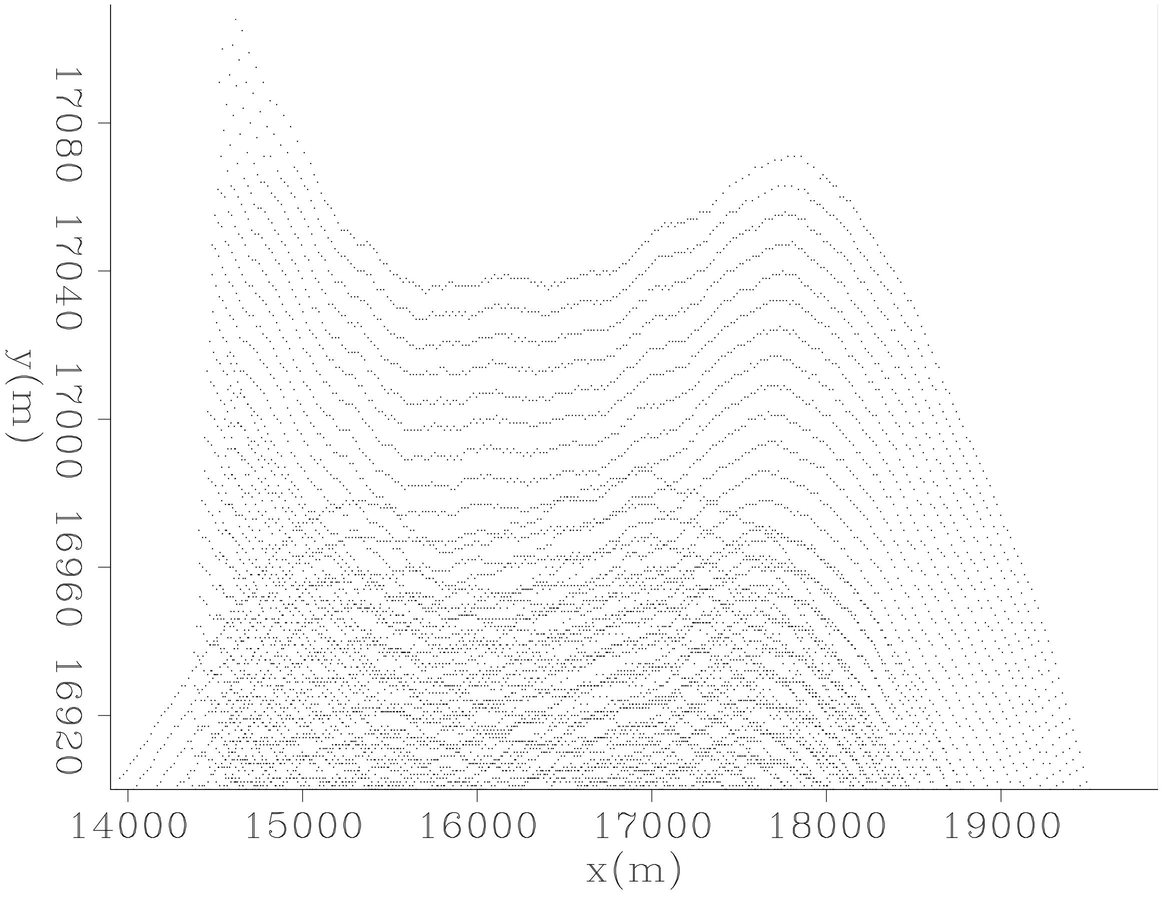


Figure 1.6: Receiver positions for a number of shots from a 3-D survey. Vertical exaggeration is around 50:1. The ship and the towed cables do not go precisely straight, but deviate in a smooth manner. The data are not precisely regular in shot and receiver location coordinates, but they are predictable, and they are regular in shot and receiver number coordinates. int-recs2 [NR]

with the dip of the operator.

Data aliasing can also, of course, be dealt with by bandpassing, but it would be foolish to throw away the useful information contained in the high frequencies. An alternative is to make more data. Aliasing in seismic data can almost always be thought of as not having traces spaced closely enough, along some axis. Interleaving a set of appropriately synthesized traces removes the problem without giving up the high frequencies. Naively interleaving traces (for instance by linear interpolation or by FFT and zero pad), will produce more data but with incorrect dips.

A simple example is shown in Figure 1.7. An almost-aliased diagonal event and its Fourier transform appear in Figure 1.7a. When the data are reduced to every second trace (1.7b), the data are obviously aliased, such that the dipping event appears to be a series of horizontal events. The Fourier transform exhibits the expected wraparound. Bandpassing removes the aliasing wraparound, as shown in Figure 1.7c, but is not an attractive option because half the information is gone. Linear interpolation, shown in Figure 1.7d, extrapolates both the desired energy and the aliases in the frequency domain, so that the resampled data have the wrong dip. Zero padding in the frequency domain gives a similarly bad result. This is an excellent method when the data are not aliased. When the data are aliased, it does not really accomplish anything, as shown in Figure 1.7e. Restoring the data so that it looks like Figure 1.7a requires "unwrapping" the aliased energy visible in the second row, and not just zeroing or extrapolating it.

A number of algorithms exist that attempt to do exactly that, with varying degrees of success, expense, and robustness. The most well known is the (f,x) interpolation method of Spitz (1991; 1990). At each frequency, a spatial prediction filter is calculated from the known data, absorbing a set of constant linear dips to represent the data. A given filter is then used as an interpolation operator at double the frequency, where the spectrum it has captured from the known data corresponds to the same linear slopes at half the trace spacing.

A similar method is described by Claerbout (Claerbout, 1992b; Claerbout and Nichols, 1991) , who estimates a PEF in the (t,x) domain and uses it as the interpolation operator. Where the frequencies are scaled in Spitz's (f,x) method, here Claerbout scales the physical

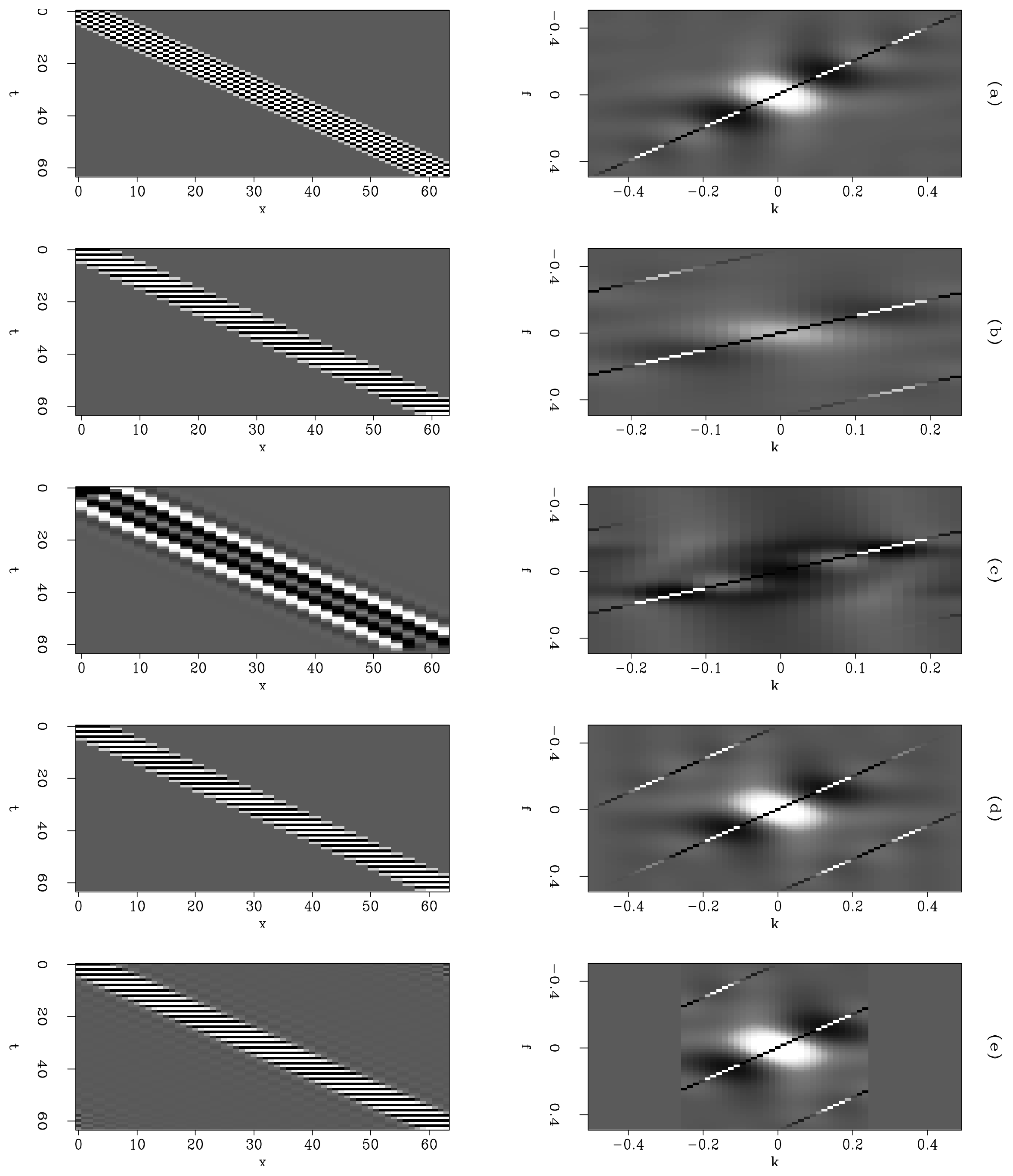


Figure 1.7: How not to interpolate aliased data. $\boxed{\text{int-introex}}$ [ER]

axes of the PEF. The lag of a filter coefficient is scaled equally along all axes in order to represent the same slope at different trace spacings.

Basic to both of these methods is the assumption that seismic data are made up of linear events with constant slopes. This is clearly untrue for any large amount of data. Changes in structure, velocity, offset and time cause changes in the slope of seismic events. To get around this problem, we typically divide the data into small regions (from here on referred to as patches) where we assume that the data do have constant slopes. We estimate filters and interpolate traces in each patch. At the end we reassemble the patches, usually with some reasonable amount of overlapping, to make a full dataset.

Another group of methods are based on local Radon transforms. These methods make a similar assumption to the linear event assumption that the filtering methods above make, though whether the events are really assumed to be linear, parabolic, hyperbolic or whatever, depends on the particular Radon transform being used. In this case the Radon transform is the estimate of local dip, and the modeling adjoint of the transform stacking operator is the interpolator. Cabrera (1984) and Hugonnet (1997) describe variations on this theme. Novotny (1990) instead picks the strongest dip in the transform and linearly interpolates along it.

All the methods listed above are two-step methods, where the first step is to gather local information from regions of the data, and the second step is to use the information to synthesize new data with similar properties. Still another group of methods exist, which are based on migration-like operators. For example, Ronen (1987) applies DMO and inverse DMO iteratively to regrid spatially aliased data, and Chemingui (1996) uses AMO.

Dealiasing versus antialiasing

It is apparent from the number of interpolation algorithms that people have invented that aliasing is an old problem, and so naturally many previous authors have used terms like “dealias” and “antialias,” with slightly different connotations. In this thesis, I follow the examples of numerous SEP alumni (and others), and use the term “antialias” to refer to methods which remove aliasing without changing sampling rates, by lowpass filtering the data in some manner (Claerbout, 1992a; Lumley et al., 1994). I use the term “dealias” to refer to methods which maintain frequency and change sampling, by inventing some hypothetical data to represent or supplement the original (Manin and Spitz, 1995; Ronen, 1990; Nichols, 1992). This seems as good a convention as any, though counter examples can be turned up (Beasley and Mobley, 1998). Finally, where data are said to be unaliased, I simply mean that in their original state, without any particular sort of manipulation, the data are not aliased.

THESIS OVERVIEW

Interpolation with locally stationary PEFs

In Chapter 2, I describe how a PEF can be used to interpolate stationary, aliased data in two steps of least squares, similar to Claerbout (1992b). The first step calculates a PEF from the recorded data. The PEF "learns" the dips in the data; its spectrum tends towards the inverse of the data spectrum (in the limit as the number of filter coefficients approaches the number of data samples). The data spectrum is aliased, and so the PEF spectrum is as well. But taking advantage of the scale invariant properties of the PEF allows us to easily "unwrap" its aliased spectrum. The PEF then has a dealiased representation of the data spectrum.

In the second step of the interpolation, a closely related problem is solved, except that the filter is known and the missing data is unknown. By "missing" we mean all the hypothetical data that was not included in the acquisition geometry. Typically this translates into every other trace, when the offset sampling interval or the source sampling interval is being halved in the interpolation. Whereas the PEF acquired the inverse of the aliased data spectrum in the first step, the data winds up with the inverse of the dealiased PEF spectrum in the second step. The result is a version of the original data, dealised by virtue of having double (or some other integer multiple) the spatial sampling rate of the original.

Of course, seismic data do not tend to be stationary, so the data is divided into small patches, and assumed to be stationary within a patch. Each patch is a separate, independent problem.

Interpolation with nonstationary PEFs

In Chapter 3, I present a new method which attempts to take advantage of the fact that events in prestack seismic data do not tend to have dips that are distibuted independently over any particularly-sized region. Rather, events tend to be large curving things, with dips that change gradually as you observe them at different offsets and times.

Instead of dividing the data into assumed-stationary patches, I assume the data have gradually varying slopes, and formulate the problem so that PEFs calculated from adjacent portions of the data look similar to each other. I estimate a PEF for every data sample, or on very small patches, so that an individual patch is too small to determine all the coefficients of a single PEF. The problem becomes underdetermined. To control the null space, I use a smoothing operator as a preconditioner to encourage a smoothly-varying batch of PEFs. The smoothly-varying PEFs suit the smoothly-varying dip spectrum of the data.

I estimate a large number of PEFs, so the model space is enormous. Nonetheless, the filter estimation step converges quickly, in about ten iterations. In the tests I have run, it takes longer to estimate a smaller number of PEFs in independent patches. The cost of convolutions of the PEFs across the data do not depend on the number of filters, but only on their size, so the convolution operator is not any more expensive for many filters than for a single filter. The preconditioning operator is a smoother, which does add some cost. However, the cost of the smoother is small, because it is implemented as division (deconvolution) by a tiny three point roughening filter, rather than convolution with a larger smoothing filter. Using Claerbout's helical coordinate (Claerbout, 1998) allows a tiny operator to have a large, multidimensional impulse response. The large set of smoothly-varying PEFs produce more accurate interpolation results than other methods I tested, particularly in data with complicated moveouts.

Noisy data and land data

The examples in this thesis could be done in the (f,x) domain, using about the same theory, but somewhat faster. Aside from simply being cavalier about computational cost, there are some arguments in favor of the time domain. In particular, Abma (1995) argues that time domain filters are more resistant to noise. Abma develops the point that filters calculated in the (ω, x) domain are effectively long in the time domain, because there is a separate filter for each frequency. The extra degrees of freedom make the frequency domain filters more likely to predict the noise. This leads to interpolation of ostensibly incoherent noise.

As noise makes data harder to predict, and thus interpolate, so does irregular acquisition. In the marine case, motivation to interpolate comes largely from the multiple suppression problem. In the land case, the expense and difficulty of data acquisition provides a more general motivation. However, the land problem is also harder. While marine geometry tends to be very regular, land data often has jagged, discontinuous arrangements of shots and receivers, where the survey is forced to work around and over surface features. Land data also tends to be much noisier than marine data. Slow variations in geometry and relative lack of noise mean that a given trace or shot gather in a marine survey is similar to (and thus predictable from) its neighboring shot gathers. Land data is less predictable, so it is harder to interpolate.

In Chapter 4, I discuss the effects of noise and erratic geometry on the interpolation method. I show some land data interpolation results, and compare (f, x) and (t, x) interpolation results on noisy data.

Chapter 2

Interpolation with locally stationary PEFs

In this chapter I show how to interpolate seismic data using time and space domain prediction error filters. I describe the scale invariant properties of PEFs and describe the first of two methods for dealing with nonstationarity in the data. The first method is patching, dividing the data into small regions where stationarity can be reasonably assumed. I use patching in a synthetic data example where data aliasing degrades the results of Radon demultiple. Interpolating the data dealiases it, and improves the demultiple results. The interpolation is successful in the sense that it improves the result, but the accuracy of the interpolation is less than perfect, especially in areas of complicated moveout, where the stationarity assumption does not hold. With that in mind, I broach the second method of dealing with nonstationarity, a new smoothing-based method, which is described in detail in the third chapter.

PREDICTION ERROR FILTERS

To paraphrase Claerbout (1992b), one way to synthesize new seismic traces is to ensure that, after specified filtering, they have minimum power. Filtering is spectral multiplication, so the specified filtering is a way of prescribing a spectrum for the new traces. You are free to choose a spectrum to prescribe. A sensible choice is the spectrum of the recorded data, which can be captured, in a sense, by finding the data's PEF.

PEF spectrum equals the inverse of the data spectrum

PEFs have the important property that they whiten the data they are designed on. Since time-domain convolution is frequency-domain multiplication, this implies that the PEF has a spectrum which is inverse of the input data. This property of PEFs is what makes the interpolation scheme described in this thesis function. There are various ways to prove the whitening property of PEFs; here I follow Jain (1989) and Leon-Garcia (1994).

The minimum mean square error prediction of the nth value in a stationary zero-mean data series $u(n)$, based on the previous p data values, is

$$\hat{u}(n) = \sum_{k=1}^{p} a(k)u(n-k) \tag{2.1}$$

The coefficients $a(n)$ make up the bulk of the prediction error filter for $u(n)$, which is defined in the Z-transform domain by

$$A_p(z) = 1 - \sum_{n=1}^{p} a(n)z^{-n} \tag{2.2}$$

The coefficients $a(n)$ generate a prediction of the data. Convolving the entire prediction error filter (2.2) on the input yields the prediction error $\epsilon(n)$, which is then just the difference between the estimate $\hat{u}(n)$ and the known data $u(n)$.

$$u(n) - \hat{u}(n) = \epsilon(n) \tag{2.3}$$

The filter coefficients $a(n)$ are determined from the input data $u(n)$ by minimizing the mean square of the prediction error $\epsilon(n)$.

A fundamental principle from estimation theory says that the minimum mean square prediction error is orthogonal to the known data and to the prediction. It turns out that this implies the most important property of PEFs, and their utility in finding missing data.

First, to develop the orthogonality condition, we consider the data $u(n)$ and the estimate

$\hat{u}(n)$ as random variables. The estimate is the expectation of the true data $u(n)$ based on the rest of the data, the random sequence $\mathbf{u} = (u(1), u(2), u(3), ...)$, not including $u(n)$. Taking $f(\mathbf{u})$ to be any function of $\mathbf{u}$, and using $E\,[]$ to denote expectation, we can write

$$\begin{aligned} E\left[\hat{u}(n) f(\mathbf{u})\right] &= E\left[E(u(n)|\mathbf{u}) f(\mathbf{u})\right] && (2.4) \\ &= E\left[E(u(n) f(\mathbf{u})|\mathbf{u})\right] && (2.5) \\ &= E\left[u(n) f(\mathbf{u})\right] && (2.6) \end{aligned}$$

Since $\hat{u}(m)$ and $u(m)$ are functions of $\mathbf{u}$, that implies the orthogonality conditions

$$E\left[(u(n) - \hat{u}(n))u(m)\right] = E\left[\epsilon(n)u(m)\right] = 0, \qquad m \neq n \tag{2.7}$$

and

$$E\left[(u(n) - \hat{u}(n))\hat{u}(m)\right] = E\left[\epsilon(n)\hat{u}(m)\right] = 0, \qquad m \neq n. \tag{2.8}$$

With certain provisos, these orthogonality conditions imply that the prediction error is white:

$$E\left[\epsilon(n)\epsilon(m)\right] = \sigma_\epsilon^2 \delta(n-m) \tag{2.9}$$

where σ_ϵ^2 is the variance of $\epsilon(n)$.

For proof, replace n with $m-k$, and

$$\begin{aligned} E\left[\epsilon(m)\epsilon(m-k)\right] &= E\left(\left[\epsilon(m)u(m-k)\right] - \left[\epsilon(m)\hat{u}(m-k)\right]\right) && (2.10) \\ &= E\left[\epsilon(m)u(m-k)\right] - E\left[\epsilon(m)\sum_i a(i)u(n-k-i)\right] && (2.11) \end{aligned}$$

The first term on the right side of equation (2.11) is a delta function with amplitude equal to the variance of the prediction error, because

$$\begin{aligned}
E\left[\epsilon(m)u(m)\right] &= E\left[\epsilon(m)\epsilon(m)\right] + E\left[\epsilon(m)\hat{u}(m)\right] && (2.12)\\
&= E\left[\epsilon(m)\epsilon(m)\right] + E\left[\epsilon(m)\sum_i a(i)u(m-i)\right] && (2.13)\\
&= \sigma_\epsilon^2, i \neq 0 && (2.14)
\end{aligned}$$

The second term in the RHS of equation (2.11) is basically the same, but the delta function appears inside the sum. Rewriting the RHS, equation (2.11) turns into

$$E\left[\epsilon(m)\epsilon(m-k)\right] = \sigma_\epsilon^2 \delta(k) - \sigma_\epsilon^2 \sum_i a(i)\delta(k+i) \tag{2.15}$$

If the filter A_p is causal, then $i = 1, 2, 3, ..., p$. The sum in the RHS of equation (2.15) is zero, because k is an autocorrelation lag, meaning $k \geq 0$. This means the prediction error is white:

$$E\left[\epsilon(m)\epsilon(m-k)\right] = \sigma_\epsilon^2 \delta(k). \tag{2.16}$$

If the filter A_p is not causal but has $i = -p, ..., 0, ..., p$, then the sum does not vanish, and the prediction error is not white. In this case, using $a(0) = -1$ to fit with equation (2.2),

$$E\left[\epsilon(m)\epsilon(m-k)\right] = -\sigma_\epsilon^2 \sum_{i=-p}^{p} a(i)\delta(k+i) \tag{2.17}$$

The prediction error is the output of the convolution of PEF and data, so if the prediction error is white, then the PEF spectrum tends to the inverse of the data spectrum. This is the most important thing about PEFs; they give an estimate of the inverse data spectrum. This is only true for a causal prediction.

The development above uses one-dimensional data series. In this thesis I deal with predicting missing trace data from other known traces, so two and more dimensions are necessary. Thinking in helical coordinates (Claerbout, 1998) allows extension to arbitrarily many dimensions. Jain (1989) develops the same arguments with more dimensions. Also, Claerbout (1997) gives alternative whiteness proofs in one dimension and two dimensions, attributed to

John Burg.

Causality in multiple dimensions

In order for the PEF to contain the inverse of the data spectrum, it has to be causal. The notion of causality is most obvious when there is a single axis labeled “time,” but it extends readily enough to two or more dimensions. Your eye scans the page from left to right and top to bottom; the words previous to the current one are to the left on the same line and everywhere on the lines above. This gives a sort of two-dimensional causal region. Including all the words on previous pages gives a sort of three-dimensional causal region. This is somewhat arbitrary. In some other language the fast and slow axes might be swapped or reversed. The important thing is that along a line parallel to any axis and going through the current point (the zero lag, the value $1z^0$ in equation (2.2)), a causal region lies only to one side of the current location. Figure 2.1 shows an illustration of a two-dimensional causal region. We can make a causal prediction of the data value in the square labeled “1” from any or all of the other shaded squares. The shaded squares are labeled to show the layout of of a PEF with four adjustable coefficients. The “1” is the value $1z^0$ in equation (2.2), and the $a(n)$ are the adjustable coefficients.

Figure 2.2 shows a picture of a 2-D PEF, and Figure 2.3 shows a picture of a 3-D PEF. In both cases, the dark shaded block holds the 1, and the lighter blocks are the coefficients $a(n)$ calculated from the data.

Scale invariance and filling in the missing data

The PEF that minimizes the causal prediction error turns out to have the inverse of the data spectrum as its spectrum. With such a filter, Claerbout’s statement at the beginning of this chapter tells how to then fill in the missing traces. However, interpolation is interesting only when the data are aliased. And when the data are aliased, the data spectrum is *not* really a sensible choice. The data aliasing is expressed in the Fourier domain by the wraparound artifacts, which need to be unwrapped for the interpolation to be useful. Otherwise, we could

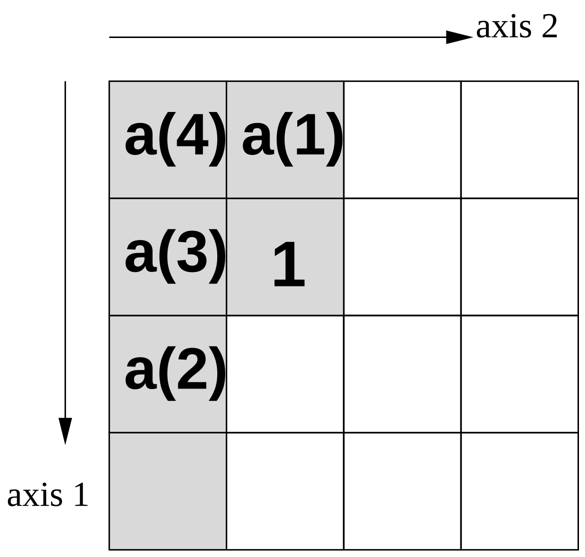


Figure 2.1: A two-dimensional causal region. We can predict the data value in the square labeled “1” from any or all of the other shaded squares. bp-causal [NR]

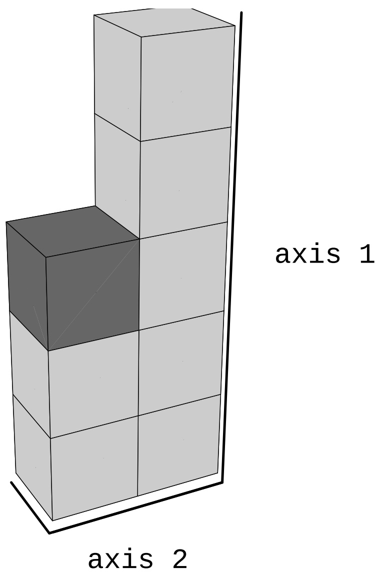


Figure 2.2: Form of a 2-D prediction error filter. The shaded box holds the zero-lag coefficient, with a fixed value of 1. bp-2dpef [NR]

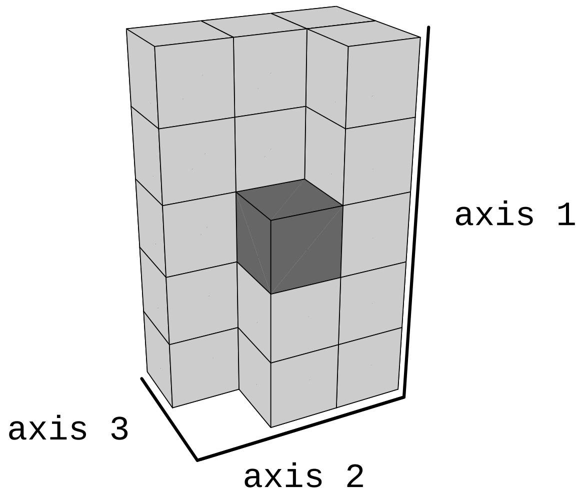


Figure 2.3: Form of a 3-D prediction error filter. The shaded box holds the zero-lag coefficient, with a fixed value of 1. bp-3dpef [NR]

do everything instantly in the Fourier domain by zero padding or something similar, and not bother with filter estimation at all. This complication turns out to illustrate another important PEF property, scale invariance.

The data that we use to calculate PEFs is incomplete. We can imagine the missing traces that we want to eventually fill in as being composed of zeroes. We can rearrange the recorded data and the zeroes in various ways by sorting the data. Continuing with the idea of missing shots in marine acquisition, we can sort the data into common receiver gathers and form a checkerboard of recorded and zero traces, as shown in Figure 2.4. We can choose to leave them as shot gathers, and have the arrangement shown in Figure 2.5.

We do not want those zeroes to influence our filter estimation, so we scale the axes of the filter. Figure 2.6 shows the PEF from Figure 2.3, but with all its axes scaled by two. Again the dark box is the fixed 1 and the lighter boxes hold the adjustable coefficients. The empty spaces between boxes are lags with no coefficients in them. Convolving the filter in Figure 2.3 across either of the data cubes in Figures 2.4 and 2.5 results in predicting zeros from known data values, or known data values from zeros, depending on the position of the filter in the data cube. Convolving the filter in Figure 2.6 across the same data cubes results in predicting zeros

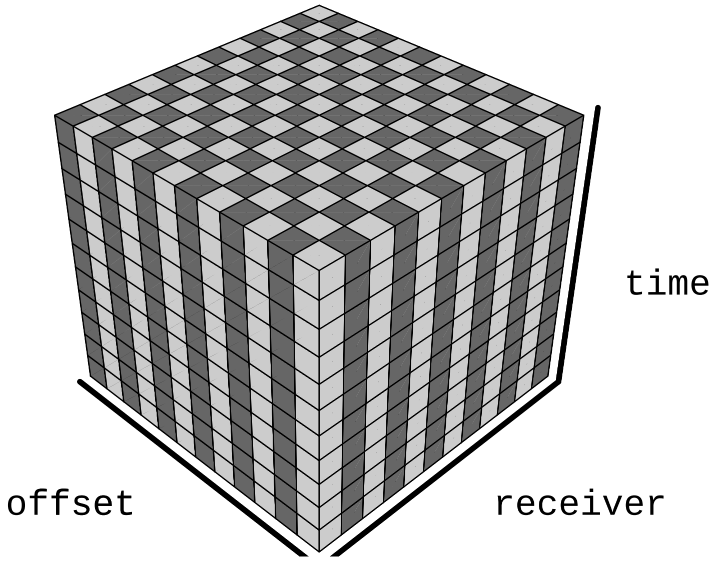


Figure 2.4: Arrangement of recorded and missing traces in one inline of a dual-source marine experiment, sorted into common receiver gathers. One shade represents recorded traces and the other missing traces. bp-traces-crg [NR]

from zeros, or known data from known data only. That is what we want. The PEF should find the spectrum of just the recorded data.

All the axes are scaled equally. It looks as though we only need to scale the spatial axes. However, if we only scale the spatial axes, then the dips represented by the original-size filter (Figure 2.3) will not be the same as the dips represented by the scaled filter (Figure 2.6). We need to return the filter to its original size in order to fill in the missing traces. As an aside, scaling the filter halves the temporal Nyquist of the filter. If the input data contain frequencies above the half-Nyquist, it is important to resample the time axis. This is discussed in more detail later.

Calculating data values to put in the missing traces is exactly like calculating filter coefficients. Minimize the mean square of the prediction error ϵ, except instead of changing the filter coefficients $a(n)$, change the data values $u(n)$. Naturally the recorded data should not change, so change only the $u(n)$ that were not part of the original data set, those corresponding

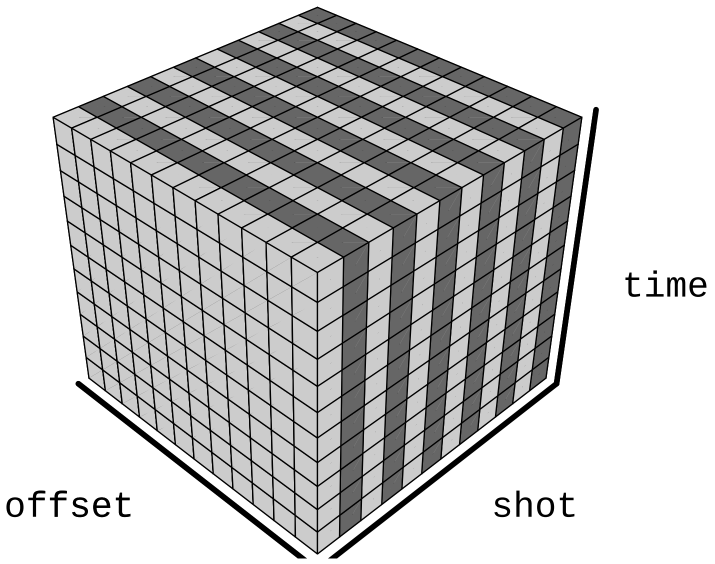


Figure 2.5: Arrangement of recorded and missing traces in one inline of a dual-source marine experiment, sorted into common shot gathers. One shade represents recorded traces and the other missing traces. $\boxed{\text{bp-traces-csg}}$ [NR]

to the zero traces in Figures 2.4 and 2.5.

The filter with scaled axes, shown in Figure 2.6, does not predict missing values from known ones, so there is no criteria for assigning values to the missing data. In order to move energy between the known and missing traces, we rescale the axes so that the filter is compressed to its original size, as in Figure 2.3. Happily, the PEF's wrapped (aliased) spectrum that is calculated from the aliased data seems to come unwrapped; depending on how aliased the original data is, the rescaled filter is less aliased or not aliased at all. The PEF coefficients were calculated so that the PEF's spectrum was the inverse of the data spectrum. By the same method, the adjustable data values are calculated to have the inverse of the PEF spectrum, which is just the original data spectrum again, but with aliasing wrap-around removed. Where before the filter took on the inverse of the aliased data spectrum, now the data takes on the inverse of the unaliased filter spectrum.

As an example, Figures 2.7 and 2.8 show a simple diagonal example similar to the one in

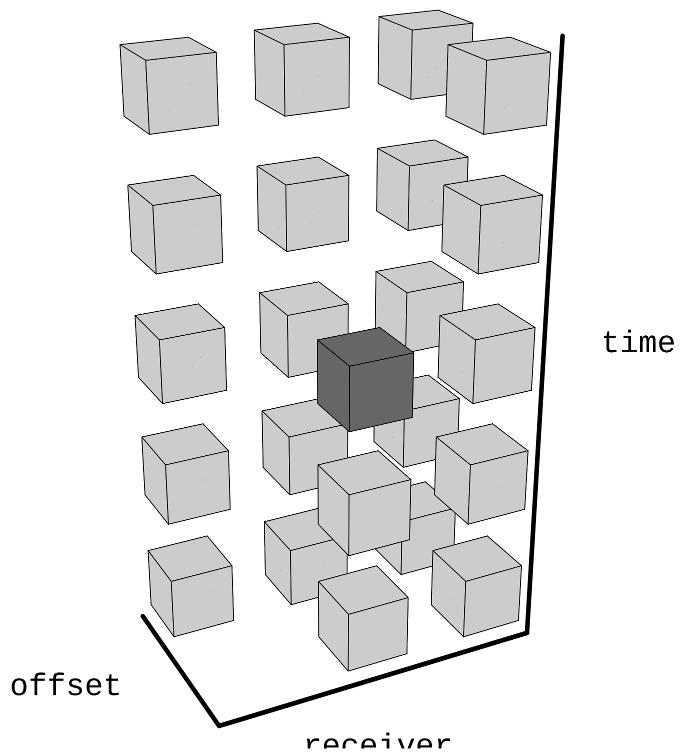


Figure 2.6: A 3-D prediction error filter with all axes scaled by two. bp-exp [NR]

the introduction. Figure 2.7 shows the aliased input data (2.7a), its Fourier transform (2.7b), the impulse response of the inverse of the axis-scaled PEF calculated from the data (2.7c), and the Fourier transform of that (2.7d). The axis-scaled PEF whitens the aliased input data, knocking down the dominant events in the data. Accordingly, the same four strong events are visible in the spectrum of the input and of the inverse PEF. Oddly enough, the inverse PEF has some additional events in its spectrum, at high frequencies where the data spectrum is all zero. The inverse PEF is the estimate of the spectrum of the data, so it seems strange to see energy where the data spectrum is known to be zero. However, those components of the model (the PEF) are in null space of the operator (convolution with the data), since multiplying anything by the zeroes in the data spectrum gives zero.

Figure 2.8 shows the impulse response of the inverse PEF with its axes returned to normal size (2.8a), the Fourier transform (2.8b), the interpolated data (2.8c), and the interpolated data's Fourier transform (2.8d). When the PEF is compressed to normal size, its spectrum is windowed down to a less-aliased region, which parenthetically happens to get rid of those surprising high-frequency events in the PEF. The interpolated data then takes on the spectrum of the compressed PEF. There is still an obvious alias in this particular PEF's spectrum, but it again is outside the band of the data.

Implementation

There are two steps. The first calculates a PEF and the second calculates missing data values. Both are linear least squares problems, which I solve using conjugate gradients.

The first step we can write

$$\mathbf{0} \approx \mathbf{YCa} + \mathbf{r}_0 \tag{2.18}$$

where **a** is a vector containing the PEF coefficients, **C** is a filter coefficient selector matrix, and **Y** denotes convolution with the input data. The coefficient selector **C** is like an identity matrix, with a zero on the diagonal placed to prevent the fixed 1 in the zero lag of the PEF from changing. The $\mathbf{r}_0$ is a vector that holds the initial value of the residual, $\mathbf{Ya}_0$. If the unknown filter coefficients are given initial values of zero, then $\mathbf{r}_0$ contains a copy of the input data. $\mathbf{r}_0$ makes up for the fact that the 1 in the zero lag of the filter is not included in the convolution (it is knocked out by **C**).

The second step is almost the same, except we solve for data values rather than filter coefficients, so knowns and unknowns are reversed. We use the entire filter, not leaving out the fixed 1, so the **C** is gone, but we need a different selector matrix, this time to prevent the originally recorded data from changing. Split the data into known and unknown parts with an unknown data selector matrix **U** and a known data selector matrix **K**. Between the two of them they select every data sample once: $\mathbf{U} + \mathbf{K} = \mathbf{I}$. Now to fill in the missing data values, solve

$$\begin{aligned} \mathbf{0} &\approx \mathbf{A}(\mathbf{U} + \mathbf{K})\mathbf{y} && (2.19) \\ &= \mathbf{AUy} + \mathbf{AKy} && (2.20) \\ &= \mathbf{AUy} + \mathbf{r}_0 && (2.21) \end{aligned}$$

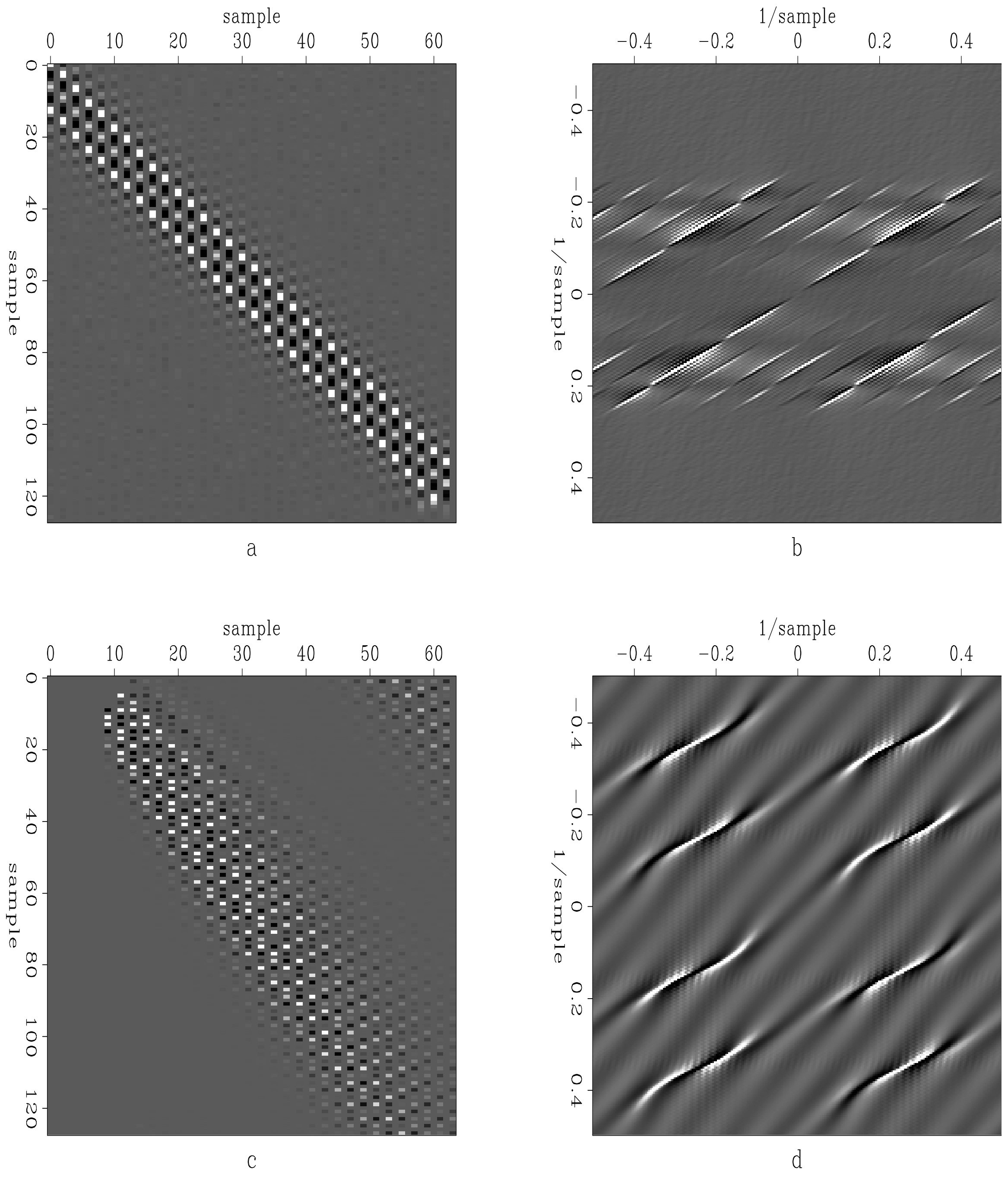


Figure 2.7: Example input, and expanded PEF calculated from the input. Panels show (a) aliased input data, (b) Fourier transform of input, (c) PEF calculated from input, (c) inverse of the envelope of the Fourier transform of PEF. bp-diagonalin2 [ER]

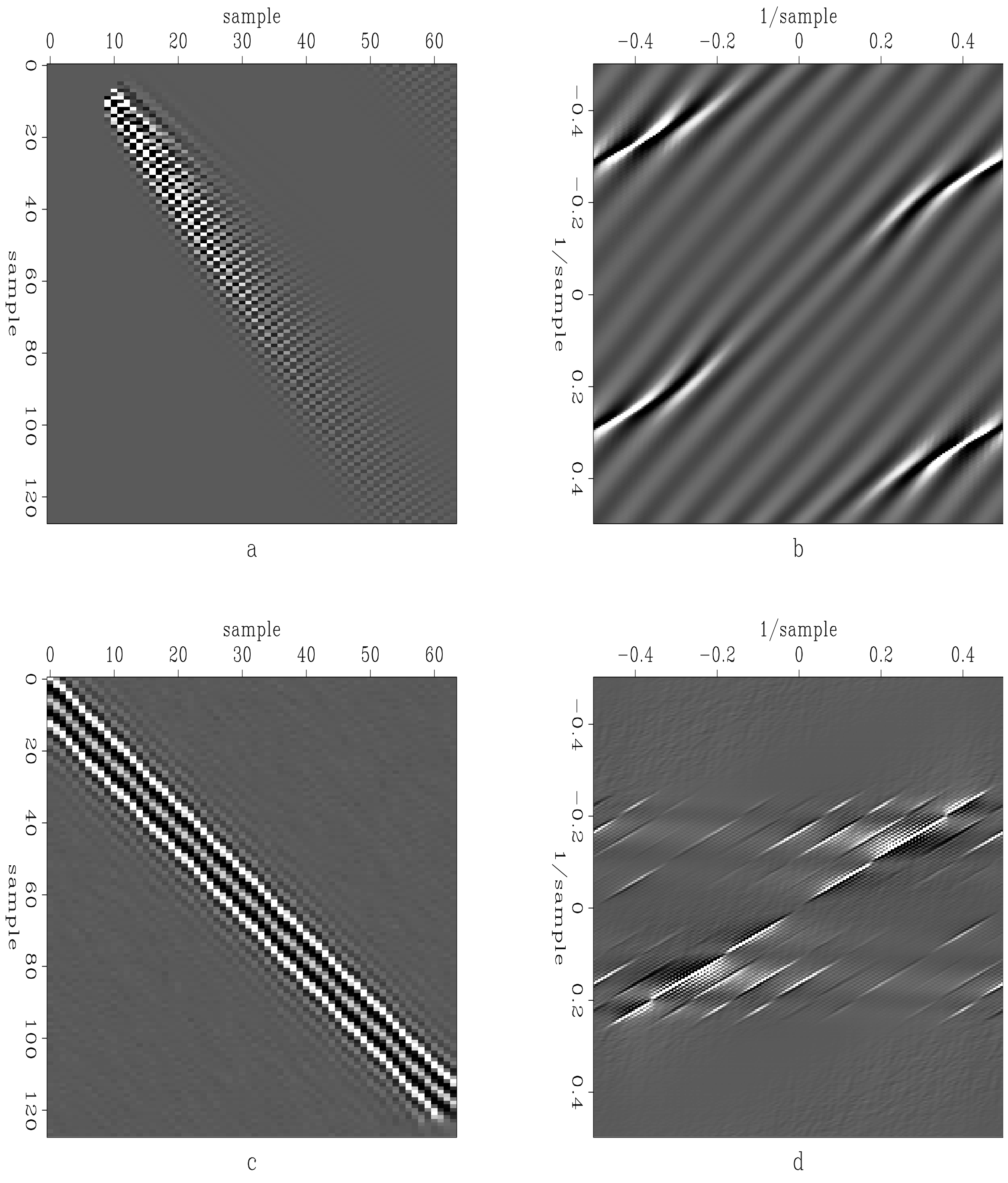


Figure 2.8: Example output, and compressed PEF. Panels show (a) compressed PEF, (b) inverse of the envelope of the compressed PEF's Fourier transform, (c) interpolated output, (d) Fourier transform of output. bp-diagonalout2 [ER]

A now denotes the convolution operator, and the vector **y** holds the data, with the selector **U** preventing any change to the known, originally recorded data values. The $\mathbf{r}_0$ in (2.21) is calculated just like the $\mathbf{r}_0$ in (2.18), except that now the adjustable filter coefficients are presumably not zero. Where before $\mathbf{r}_0$ wound up holding a copy of the data (assuming the adjustable filter coefficients start out zero), here it holds the initial prediction error $\mathbf{AKy}_0$.

There is some wasted effort implied here. There is no sense, for instance, in actually running the filter over all those zero-valued missing traces in the PEF calculation step, so they can be left out.

SAMPLING IN TIME

Dips do not change when the PEF's axes are scaled, but frequencies do. This has two important consequences. The first relates to the data's temporal nyquist. The filter that we calculate, shown in Figure 2.6, has half the temporal Nyquist of the filter in Figure 2.3. So even though seismic data is almost never temporally aliased, we can have a problem estimating filter coefficients if the data contains frequencies above the half-Nyquist. Components at certain dips and frequencies will simply slip through the gaps in the axis-scaled filter, and be effectively invisible.

This is not a problem in principle, because since the data are not temporally aliased, it is easy to resample the time axis and make sure the signal is below the half-Nyquist. But it is something that is important to remember in practice. Figures 2.9 and 2.10 show an example of the difference that resampling can make. Figure 2.9a shows a slice from a small cube of seismic data, bandpassed and sampled at 8 ms so that the data band just fills all the available frequencies, and Figure 2.9b shows its power spectrum. Figure 2.9c shows the same data, resampled to 4 ms so that the data band fills half the available frequencies, with zeroes above the half-Nyquist, and Figure 2.9d shows its power spectrum. Neither dataset is temporally aliased. Figure 2.10 shows results of zeroing half the traces in those two inputs, and then interpolating to fill the zeroed traces back in. The 8 ms data gives a poor interpolation result, shown in Figure 2.10a. In some areas it is fine, and in some it is terrible. The badly interpolated regions correspond to areas where a significant component of the data slips through the gaps in the axis-scaled filter. The 4 ms data (with signal at or below the half-Nyquist) produces a good result, shown in Figure 2.10b.

So this is a manageable issue, but it does have some important consequences later, particularly with regards to sorting the input data. In particular, it is not hard to think of a situation where you would rather interpolate to a sampling three or four times denser than the input, rather than double as in most of the examples in this thesis. It is easy enough to scale the filter's axes by three or four, but that requires more resampling of the time axis, to be sure that the filters do not miss the higher frequencies in the data. That in turn requires many more filter coefficients to cover the same range of slopes in the data. Too many filter coefficients (degrees of freedom) is often a problem. In fact, even with very low frequency input, my experience has been that results turn out to be better when the data are interpolated in several smaller steps, rather than one large step. An example is shown in the next chapter.

A second potential problem associated with scaling the PEF's axes relates to dispersive data. Scaling PEF axes assumes that events lie along lines in Fourier space, so that time dip is independent of frequency. Dispersive waves have velocities which vary with frequency. If the slope (velocity) of a seismic event is different at different frequencies, then the data's PEF at one scale may not predict that data at a different scale. In real data examples, I have not found it to be a noticeable problem. Ground roll is dispersive, but can typically be interpolated. An illustration of the potential problem, and the reason it may not be a serious issue in real data, is shown in Figure 2.11.

Figure 2.11 shows three data sets with dips varying with frequency. Each row shows a data set's 2-D Fourier transform, original time-domain data, and the result of subsampling and then interpolating the data.

In the first example, the data are just two sinusoids with different frequencies. It seems to be a simple example to interpolate, but the result in the top right panel is terrible. The estimated PEF's spectrum needs to match the data at only a few points in Fourier space, and those points effectively move when the filter is rescaled. The result is interpolated data with the wrong dips.

The second and third examples are synthetics intended to simulate dispersive waves. In

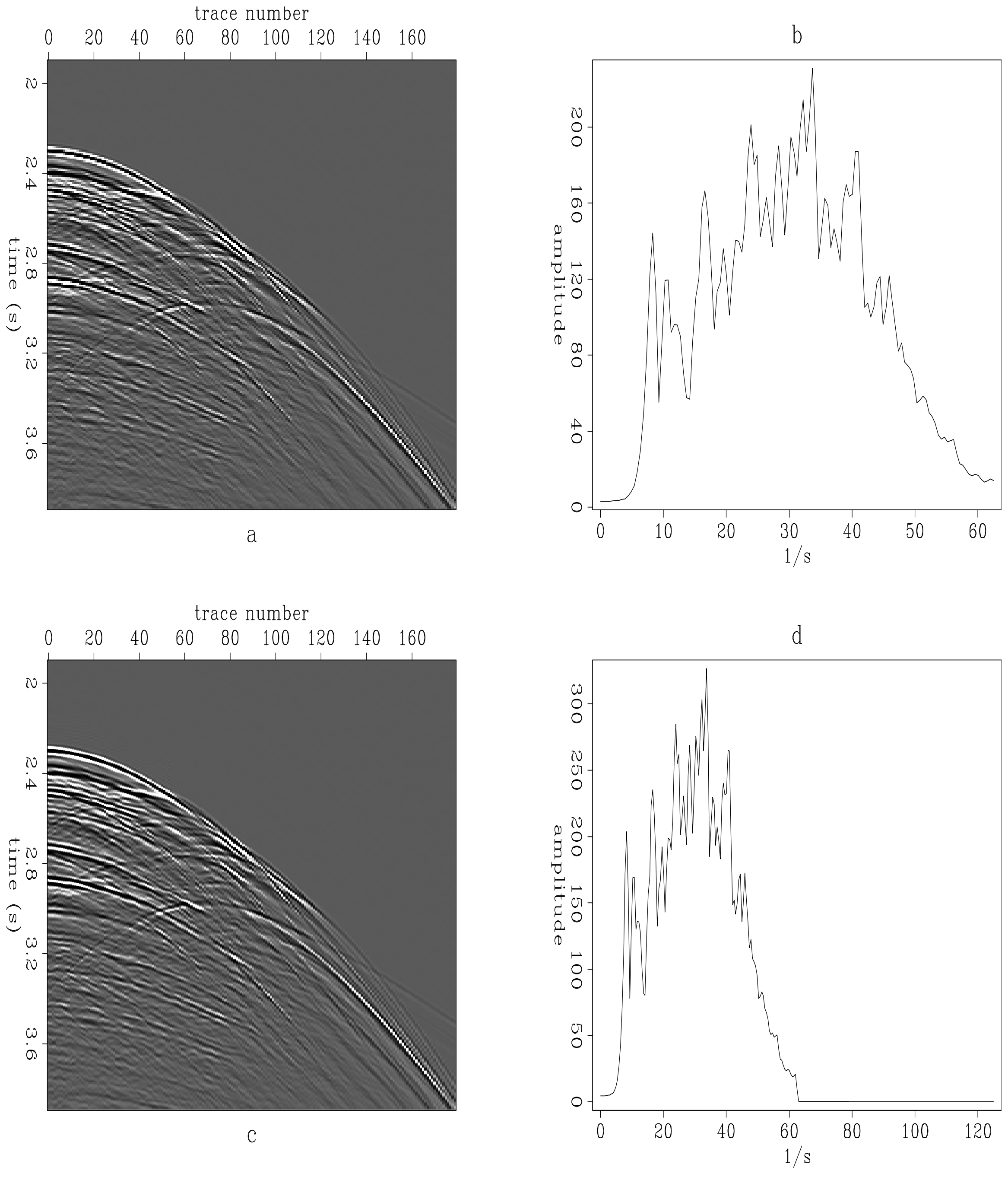


Figure 2.9: Samples of input data. The panels show (a) 8 ms input data and (b) its power spectrum; and (c) 4 ms input data and (d) its power spectrum. bp-nyqin [ER]

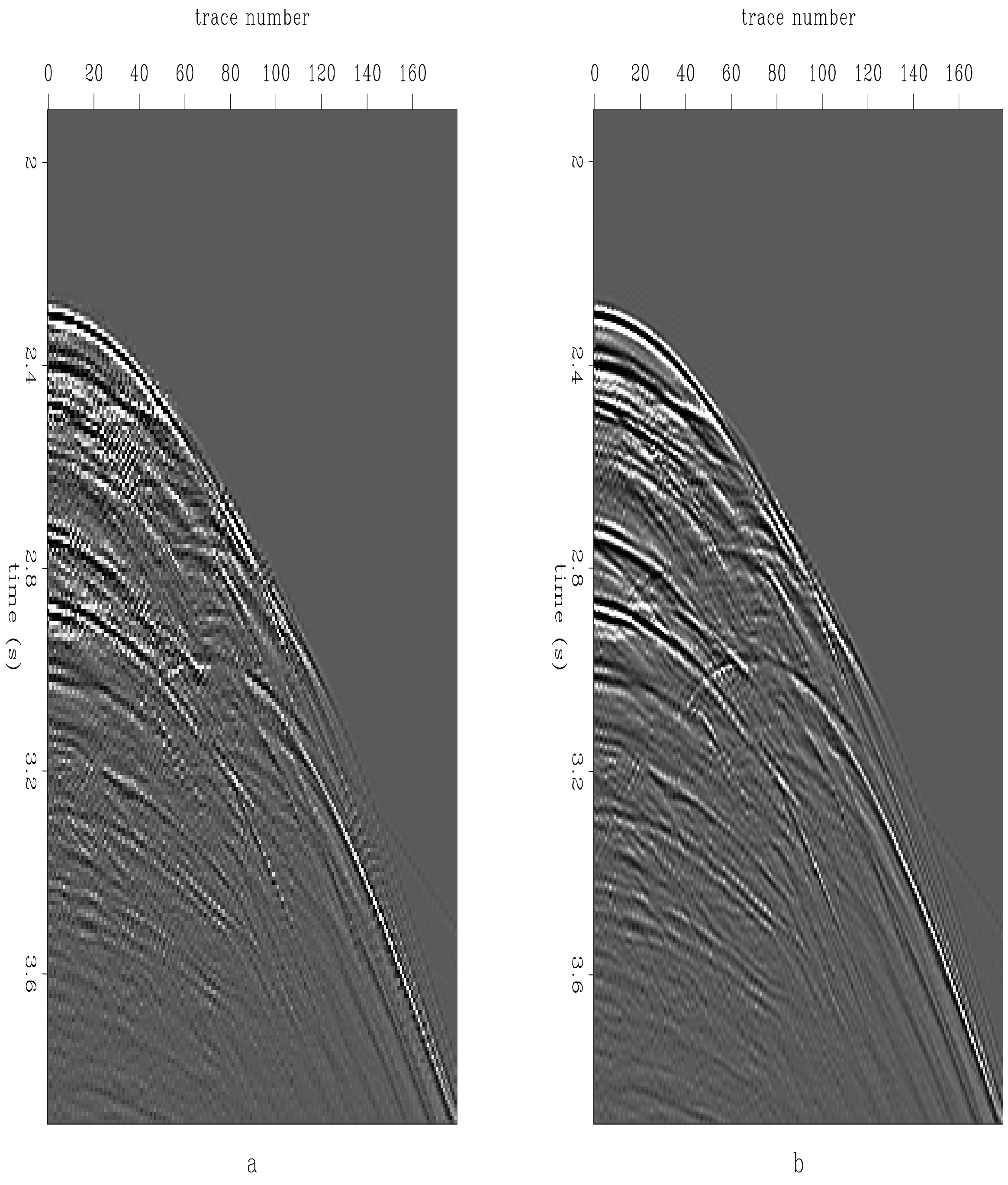


Figure 2.10: Samples of output data. The same as Figure 2.9, after zeroing half the traces and interpolating to fill the zeroed traces back in. The panels show (a) 8 ms interpolation output and (b) 4 ms interpolation output. $\boxed{\text{bp-nyqout}}$ [ER]

these cases, time dip of the events is a smooth function of their frequency. The interpolations are somewhat successful, though not perfect (as we would hope on a simple, noise-free synthetic). The example in the second row uses a single PEF, the example in the bottom row uses nonstationary PEFs. The example in the bottom row is harder because the amplitude in Fourier space does not decrease noticeably as the event bends. A single filter does not produce a good interpolation result in that example. Using multiple filters is effectively just a way to take some of the curvature out of Fourier space. In the middle row example, the energy dies off more quickly as the event curves.

Parenthetically, this example also points out that the interpolation process as a whole is nonlinear in the amplitude of the data, though the two individual steps of least squares are both linear. For instance, in the top row, the input data is the combination of two data sets, each with a single dip. Either of the two can be interpolated perfectly by itself. This is an issue if we think of starting from a nonzero solution for the missing data samples. Something like linear interpolation might seem to be a sensible way to get an initial guess, but Figure 1.7 points out that that is not going to be useful if the data are not already well sampled. Nonlinearity adds the prospect that linear interpolation or some other simple method may ultimately worsen the result.

NONSTATIONARITY

The theory in this chapter assumes that the data are *wide-sense stationary*. A data sequence is considered to be wide-sense stationary if it has constant mean and if its autocorrelation is strictly a function of lag

$$E\left[u(n_1)u(n_2)\right] = a(n_1, n_2) = a(n_1 - n_2) \tag{2.22}$$

Throughout this thesis, when data are said to be stationary, I mean that they are wide-sense stationary. This is as opposed to *strict-sense stationary*, which means that the data have the same probability mass function everywhere.

We want to look at data with more than one dimension. Thinking in helical coordinates,

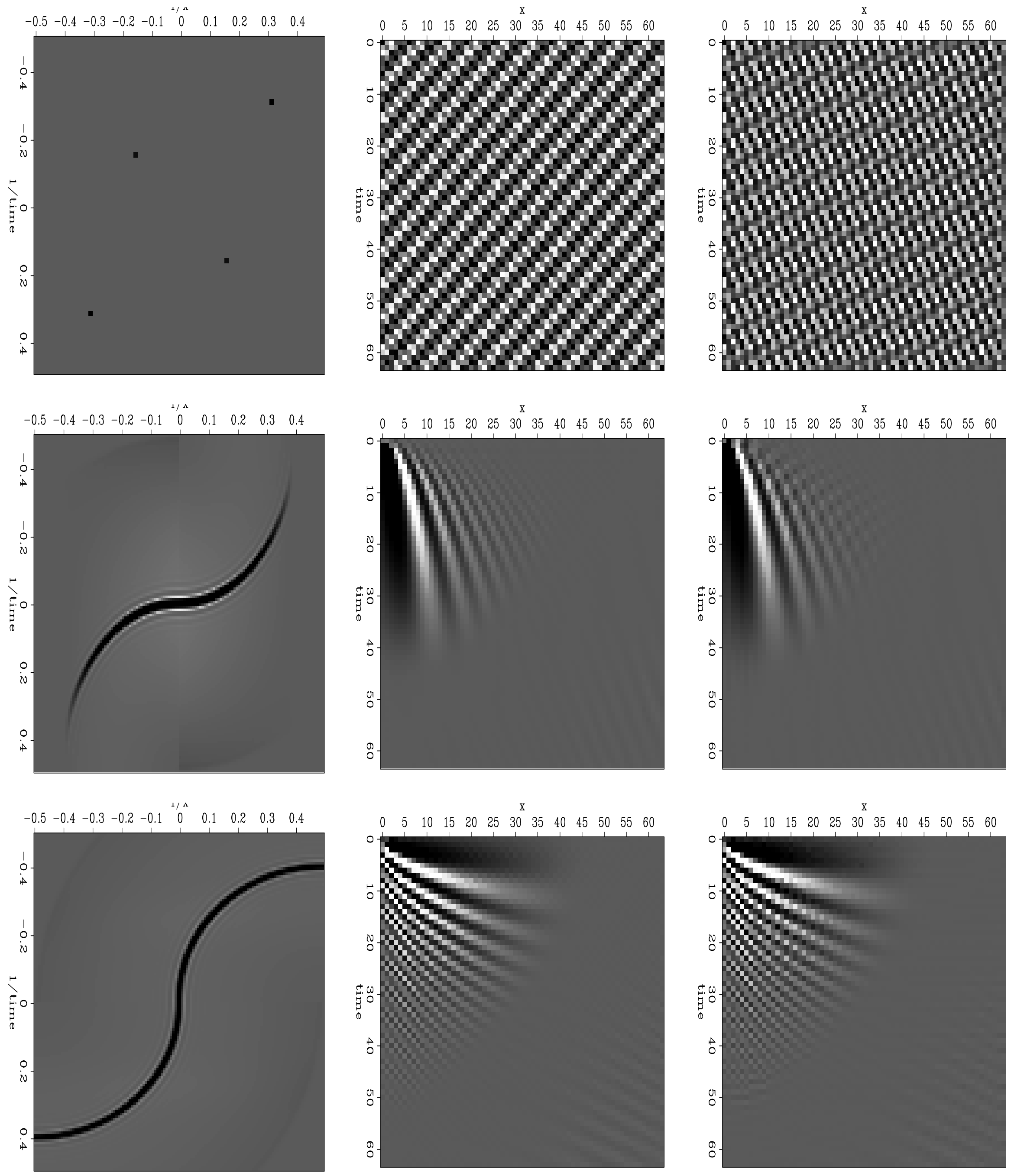


Figure 2.11: Data whose dips vary with frequency and attempts to interpolate it. Columns show Fourier transforms, input data panels, and results of subsampling and interpolating the 2-D data. Where the data are single-frequency sine waves with two different dips (top row), the result is terrible. Where the data contain dispersive events, changing coninuously with frequency, the results are somewhat better. bp-disp2 [ER]

we can view a set of traces as a super trace and take its autocorrelation. Equation (2.22) says that we should be able to get the same autocorrelation function for different subsets of the data (different starting samples n_1). Intuitively, this coincides with the statement in the introduction (taken from Spitz (1991)) that the data should be made up of linear events of constant slope. Figures 2.12 and 2.13 show examples. Figure 2.12 shows a window from the far offsets of a synthetic CMP gather, and autocorrelation sequences made from identical numbers of samples, starting at different places within the window. Figure 2.13 shows a window from the inner offsets of the same gather, and autocorrelation sequences. The far offsets, where data tends to be made up of linear events as hyperbolas approach their asymptotes, are by appearance nearly stationary, and the autocorrelations are approximately equal except for a scale factor. The inner offsets, where the events are obviously changing dip, do not look stationary and do not have approximately equivalent autocorrelation functions.

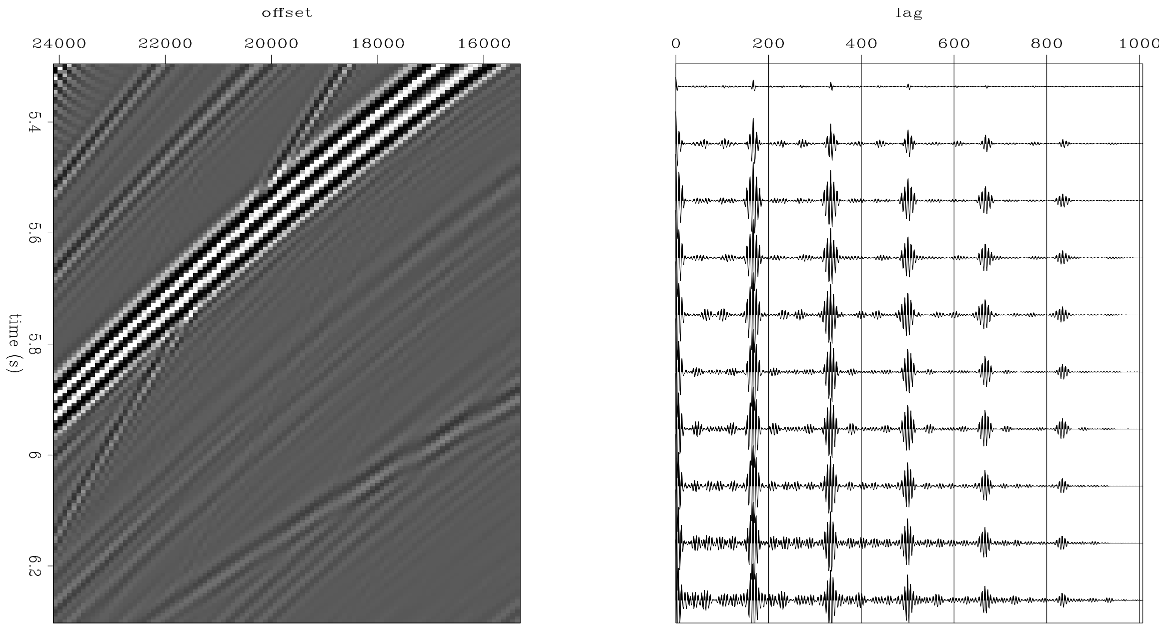
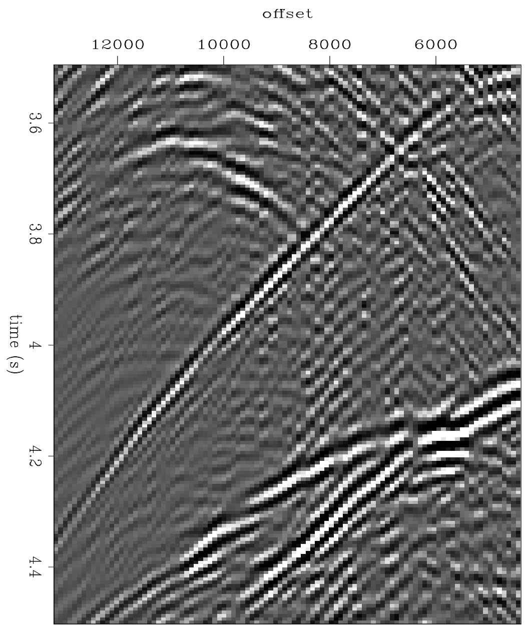


Figure 2.12: Synthetic seismic data, and autocorrelation functions calculated from different subsets of the data. The data are approximately wide-sense stationary, so the autocorrelations are approximately identical, except for a scale factor. bp-stationdata [ER]

Seismic data are not, as a rule, stationary. The slopes of seismic events change with offset and time, and when the geology is nontrivial, with surface locations also. This just means that



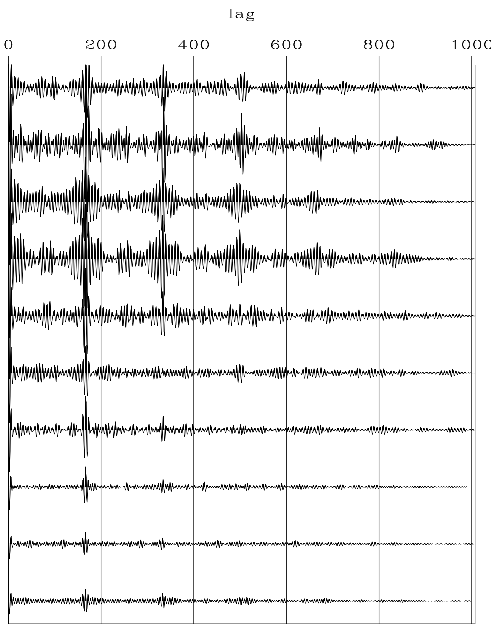


Figure 2.13: Synthetic seismic data, and autocorrelation functions calculated from different subsets of the data. The data are nonstationary, so the autocorrelations are significantly different. $\boxed{\text{bp-nonstationdata}}$ [ER]

we can not use a single PEF to represent the spectrum of all the data, but only as much data as we can reasonably assume to be stationary.

Patching

The most common way to deal with nonstationarity is to divide the data into smaller regions, called "patches" in this thesis (after Claerbout (1997)), though "design gate" and "analysis window" and other terms are common. We assume the data are stationary within a patch.

Each patch is treated as an individual problem. For each patch, we calculate a PEF and then missing data values. After all the patches are interpolated, we reassemble them to form a complete data set. To make the patches fit together without visible seams, and to hide the boundary conditions of the convolution operators, we choose the patches so that they overlap, and reassemble the patches with some simple normalization.

The BP synthetic example which follows uses patching, and within each patch, the implementation described above.

Smoothing

For the most part, the results in the BP synthetic example later in this chapter are good. However, they could improve. Near offsets and wherever the data has strong curvature tend to be poorly interpolated. The assumption that we can divide the data into rectangular regions where it is composed of linear events with constant dip seems like a bad assumption. Events are strongly curved, their dips change over short ranges of offset, but they don't change abruptly. The dips change continuously and smoothly. In order to have a set of assumptions which better fit this observed quality of the data, I extend the notion of time-variable deconvolution (Claerbout, 1997) to time- and space-variable interpolation. As an alternative to dividing the data into independent regions, we can estimate a PEF for every data sample, and use a smoothness criterion to calculate smoothly-varying filters to fit the smoothly-varying dips in the input data. This is the subject of the next chapter.

BP SYNTHETIC EXAMPLE

The first nontrivial interpolation example uses a synthetic dataset provided by British Petroleum. The dataset is 2D, and was modeled using elastic finite differences on a tabular salt model. The velocity model is shown in figure 2.14. The water layer and the salt give rise to multiples, which become aliased when the data are subsampled to simulate a dual-source marine geometry. The presence of the salt leads to complicated moveouts and a wide angular band in much of the data, making it an interesting test for interpolation. A representative shot gather from over the the salt body is shown in Figure 2.15.

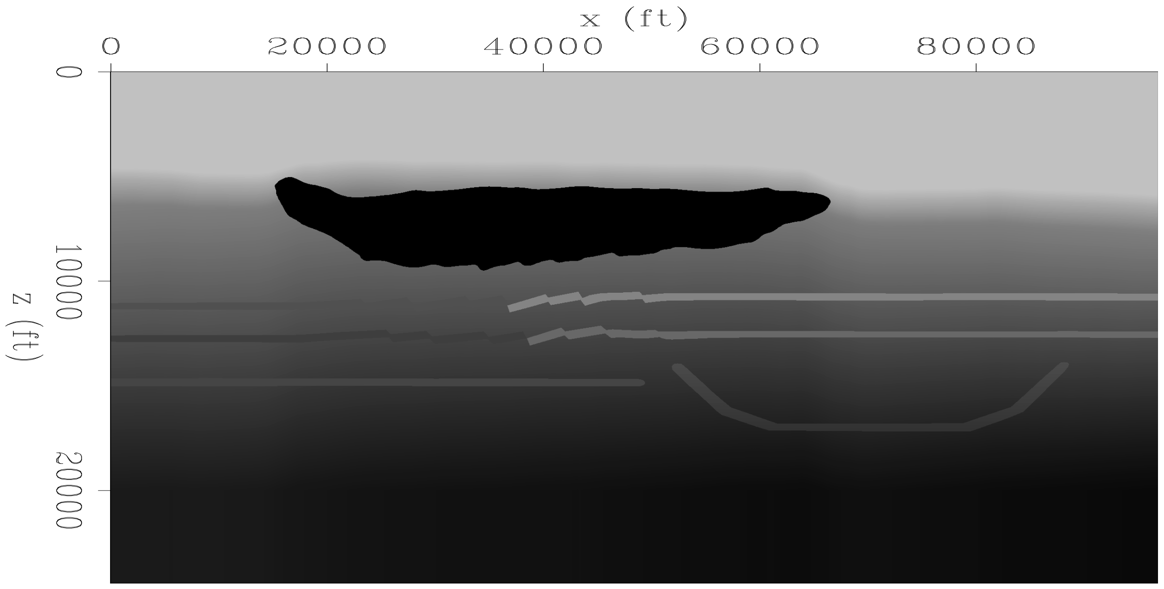


Figure 2.14: Velocity model for the BP synthetic data. The dark irregular feature is a salt body. bp-bpvel [ER]

Subsampling the BP synthetic data

The data are modeled with equal source and receiver spacings. We can throw out every other shot, and thus simulate a single inline from a multi-source 3D survey. Alternatively, we can throw out every other receiver surface location, which in this case happens to be more convenient, but has the same effect on CMP sampling. As an example, Figures 1.4 and 1.5 show the difference in CMP sampling for a survey with equal shot and receiver sampling, and with the shot sampling interval doubled. Instead of removing rows from Figure 1.4 (as we did to make Figure 1.5), we could remove columns, and the CMP gathers would be undersampled in the same way. With half the data removed, the CMP gathers lose alternating offsets, so that the offset spacing is four times the offset spacing of a shot gather. The decrease in sampling causes many of the events to become aliased. Figure 2.16a shows a CMP gather constructed from the original data, with shot spacing equal to receiver spacing, and Figure 2.16b shows the CMP gather for the same midpoint, constructed from the subsampled input. This data contains apparently asymmetric events which make it appear not to be a CMP gather at all (for example at 5 seconds and 5000 meters offset), but these are peg-leg multiples with multiple

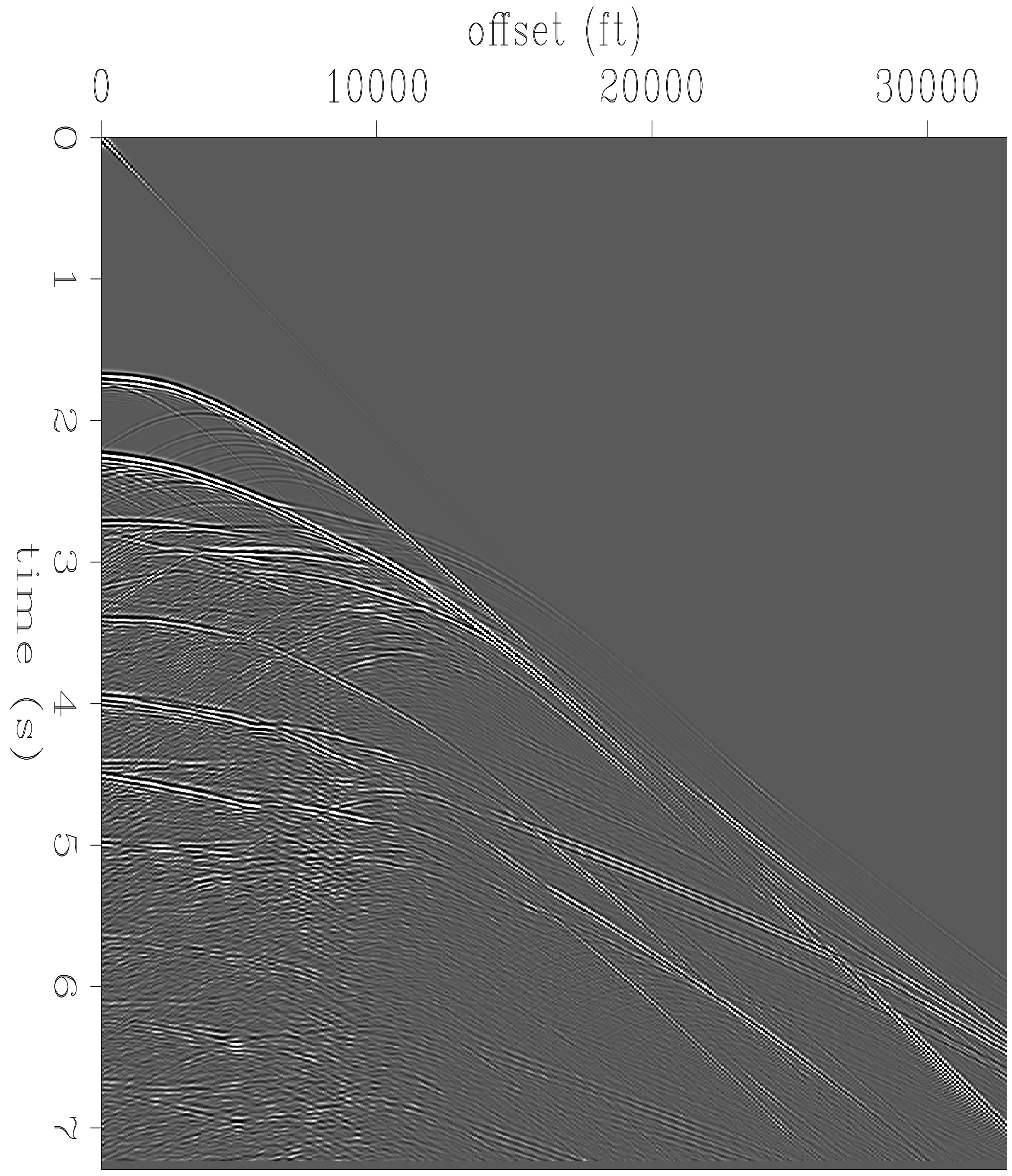


Figure 2.15: Representative shot gather from the BP synthetic survey. $\boxed{\text{bp-bpshot}}$ [ER]

paths, which would turn out to be symmetric if the negative offsets had been modeled. The mechanism is explained in detail by Thorson (1984). The steep events, which are not aliased or just slightly aliased in Figure 2.16a, are strongly aliased in Figure 2.16b. While the data do appear to be somewhat aliased in the Figure 2.16a, applying a slight moveout removes the aliasing. Moving out the data in the right panel does not remove the aliasing, because events curving upward begin to be aliased before events curving downwards cease to be aliased. Figures 2.17a and 2.17b show the same two panels with moveout applied. The velocity chosen is between the primary and multiple velocity, so that on average everything will be relatively flat (not stackably flat, but hopefully flat enough that they are not aliased). The left panel, Figure 2.17a, does not appear to be aliased at all, but in Figure 2.17b some of the events curving up and some of the events curving down have ambiguous dips.

The dip ambiguity in Figure 2.17b leads to artifacts in the Radon transform. Parabolic Radon transforms of both panels appear in Figure 2.18. Figure 2.18a shows the Radon transform of the unaliased gather from 2.17a, and Figure 2.18b shows the Radon transform of the aliased gather from 2.17b. The horizontal axis displays milliseconds of moveout at 10000 feet of offset, rather than velocity or slowness. There are obvious artifacts, due to aliasing, streaking across Figure 2.18b. These artifacts make it difficult to remove multiples, because energy from both primaries and multiples is spread throughout the Radon transform domain.

The goal of the interpolation in this case is to generate dealiased data from the aliased data so that we wind up with a Radon transform similar to Figure 2.18a and not Figure 2.18b.

Interpolating the BP synthetic data

The subsampled BP synthetic data were used as input to a two-stage, PEF-based interpolation algorithm. The first stage calculated a PEF using equation (2.18), the second stage calculated missing data using equation (2.21). Both steps were solved with conjugate gradients. To deal with nonstationarity, the data were divided into 3-D rectangular patches, chosen to overlap so that every data sample (barring those along the edges of the input data cube) was in several patches. Along a single axis, a data point is typically in 1.5 to 2 patches, which translates to 3 to 8 3-D patches.

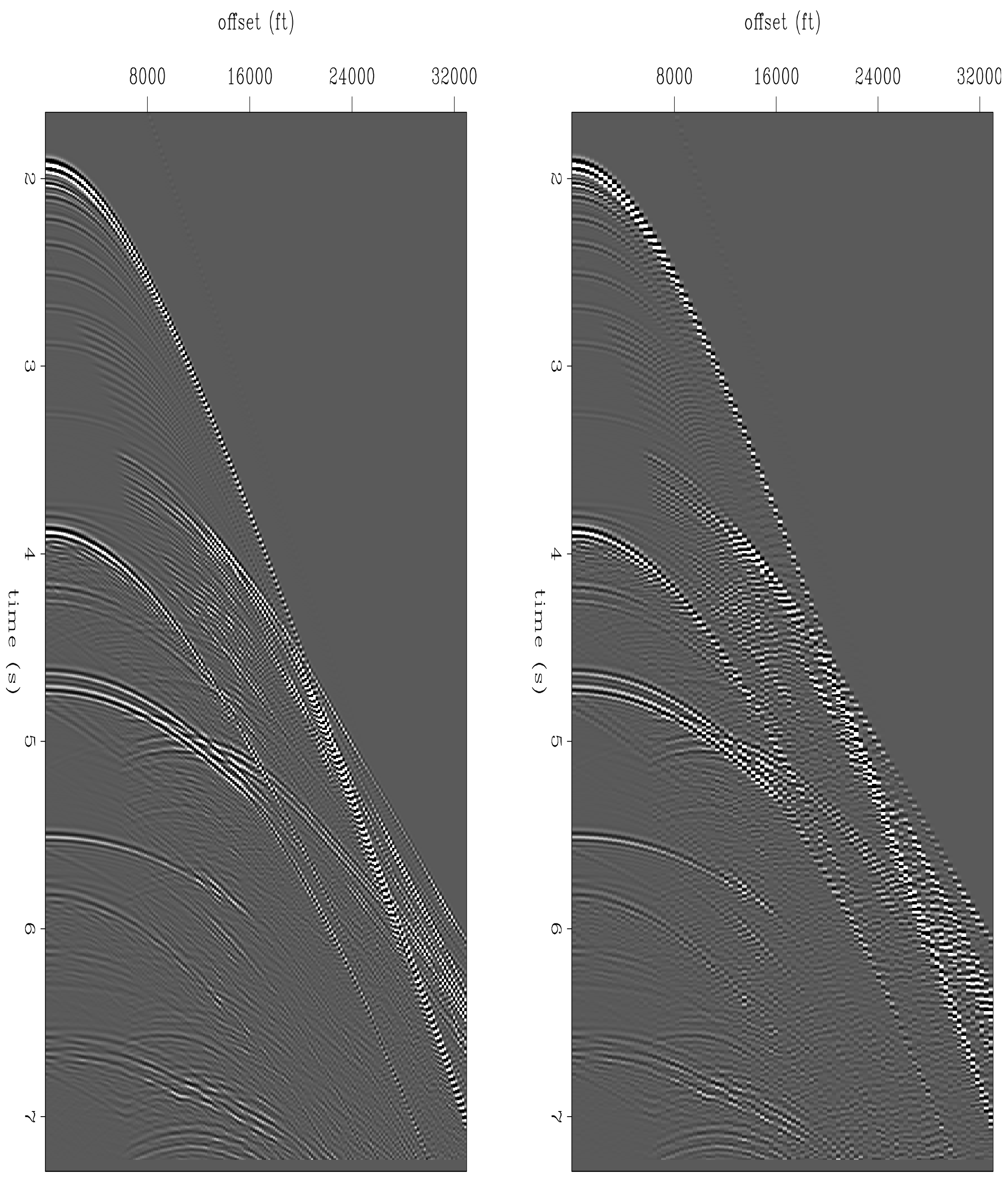
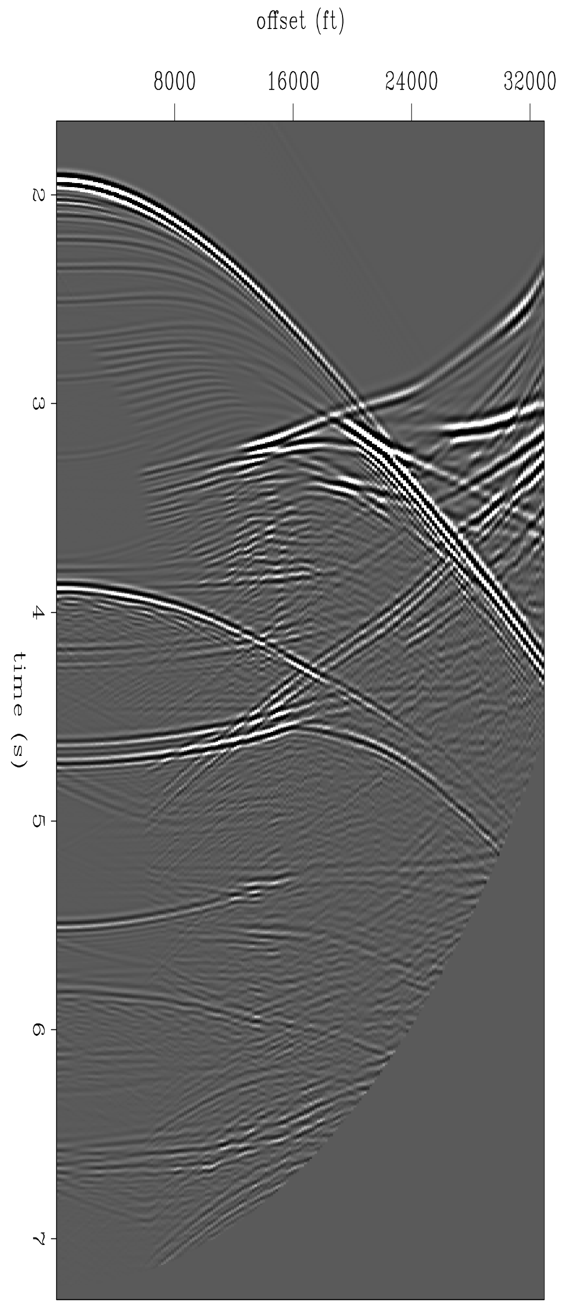
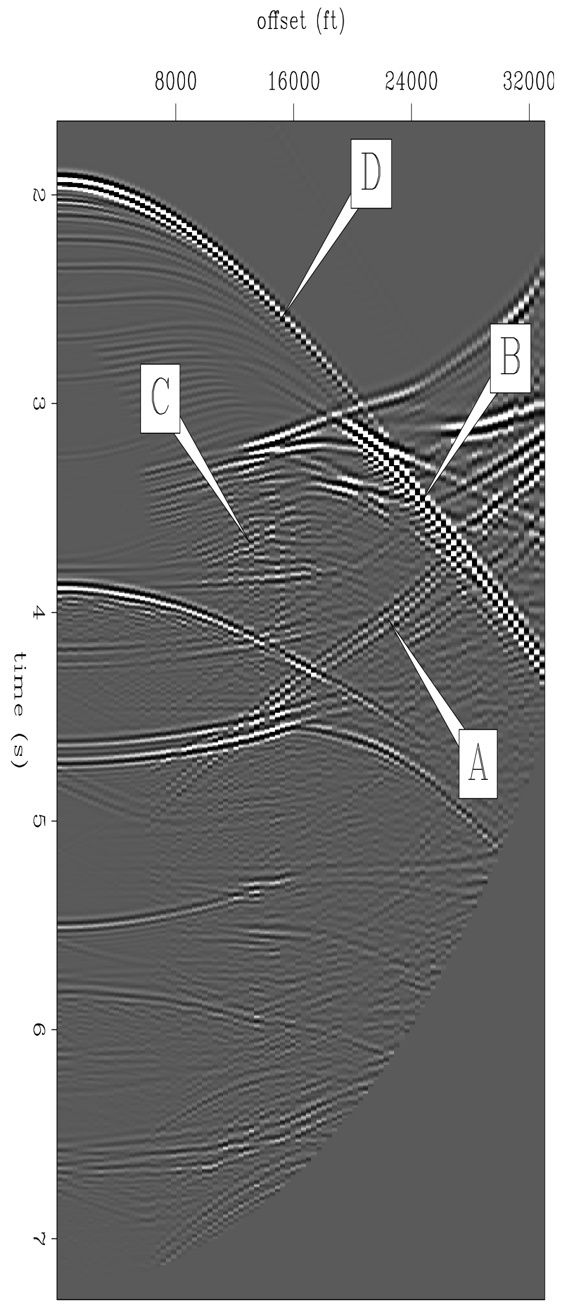


Figure 2.16: CMP gathers from the BP synthetic. The left panel is constructed from the full data set, the right is constructed from the subsampled data. bp-cmps1 [ER]



a



b

Figure 2.17: The CMP gathers from Figure 2.16 with moveout applied to remove aliasing effects. The well-sampled CMP gather shows no aliasing, the subsampled gather still shows some aliasing. The arrows labeled **A** through **D** point to events that appear to have more than one dip. $\boxed{\text{bp-cmps1nmo}}$ [ER]

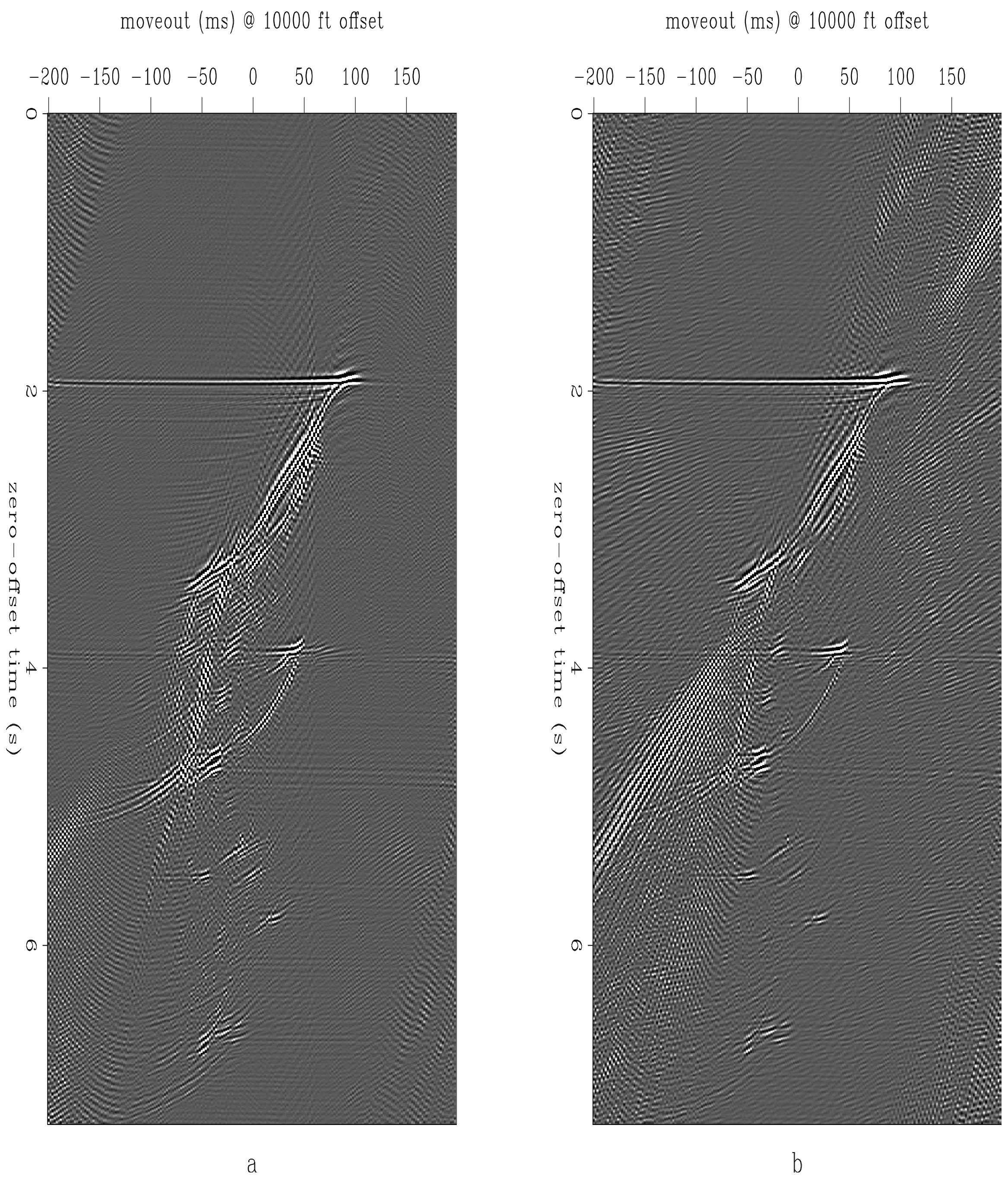


Figure 2.18: Parabolic Radon transforms of the gathers from Figure 2.17. The aliasing in the subsampled gather gives rise to strong artifacts in the Radon transform. The horizontal axis displays milliseconds of moveout at 10000 feet of offset, rather than velocity or slowness. $\boxed{\text{bp-rads1}}$ [NR]

The results, shown in the next few sections in the prestack, stack, and radon domains, are mostly very good. After interpolation, Radon transforms and stacks of the data are basically indistinguishable from the same results using the full original data, and significantly better than the same results using the subsampled data.

Results in the prestack domain

The next several figures show interpolation results and difference panels on the prestack data.

Figure 2.19 shows two CMP gathers. The left, Figure 2.19a, is from the original data, and the right, Figure 2.19b is from the subsampled and interpolated data. The two are difficult to tell apart at this scale. Figure 2.20 shows close ups on two portions of the original CMP gather from Figure 2.19a, and close ups on the same two portions of the difference between the original and interpolated CMP gather. Figures 2.20a and 2.20b are close ups of the far offset, late time corner of the CMP gather. Here the data are mostly linear events, which are ideal for prediction with PEFs. Not surprisingly, the difference panel shows little. The data in this section of the data are very predictable, and the difference between the interpolated data and the original data is small.

Figures 2.20c and 2.20d are close ups on a near offset, intermediate time window of the CMP gather. Here the data are more complicated. There are strong bending events, and dips generally change significantly from one region to the next. These data are harder to predict with a PEF than the data in Figure 2.20a, and this comes out in the difference panel, Figure 2.20d. This difference panel has much more energy than that for the more predictable data.

Results in the Radon domain

The purpose for resampling the CMP gathers in this case is to remove the aliasing artifacts from the Radon transforms. Figure 2.18 shows Radon transforms of the original data and of the subsampled data which was the input to the interpolation. The interpolation results were transformed the same way for comparison. Figure 2.21 shows the Radon transform of the interpolated data, and a difference panel. Figure 2.21a shows the transform of the same CMP

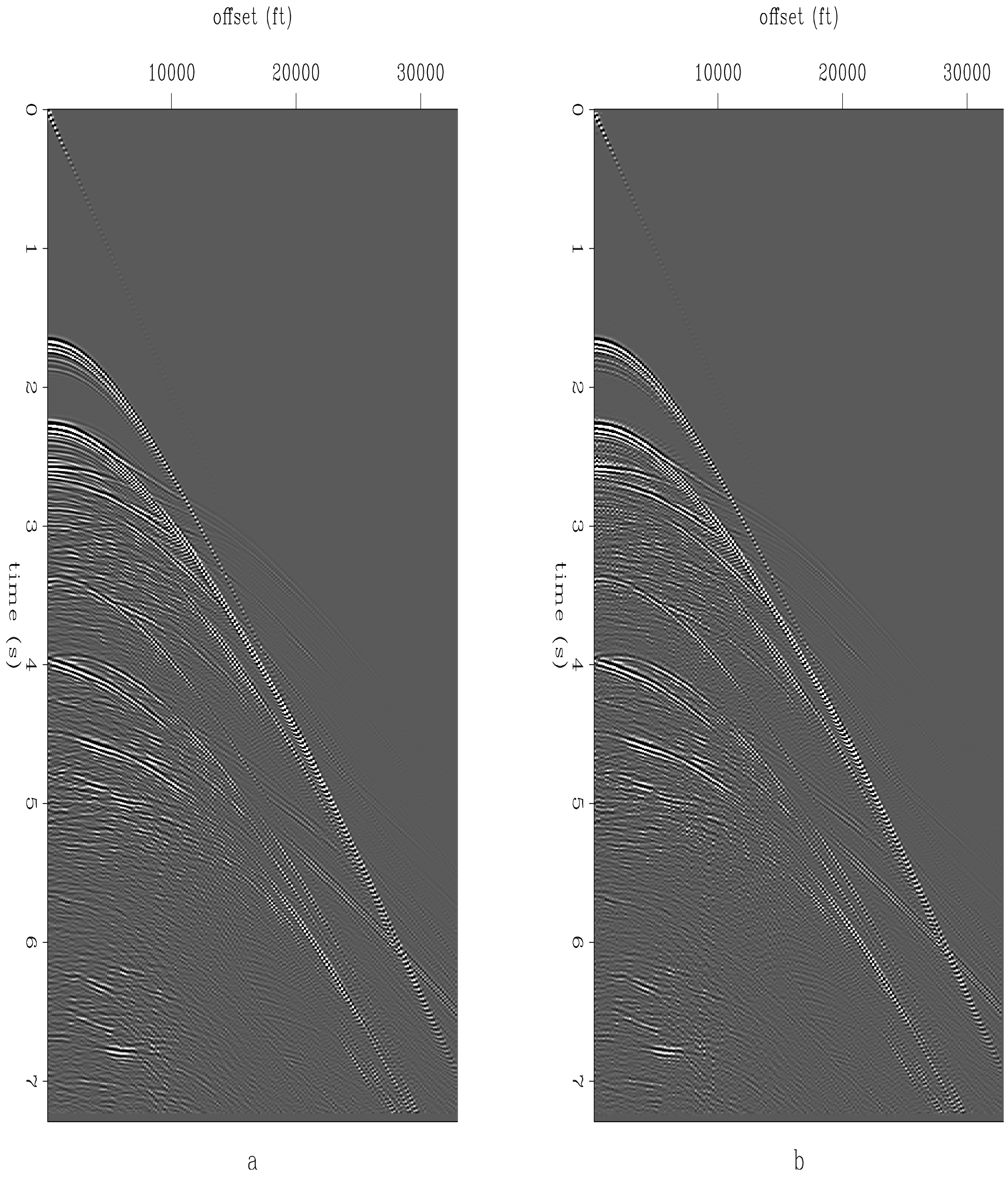


Figure 2.19: CMP gathers. The left is from the original data, the right from the subsampled and then interpolated data. Difference panels are shown in close up in Figure 2.20. bp-bpcmps [CR]

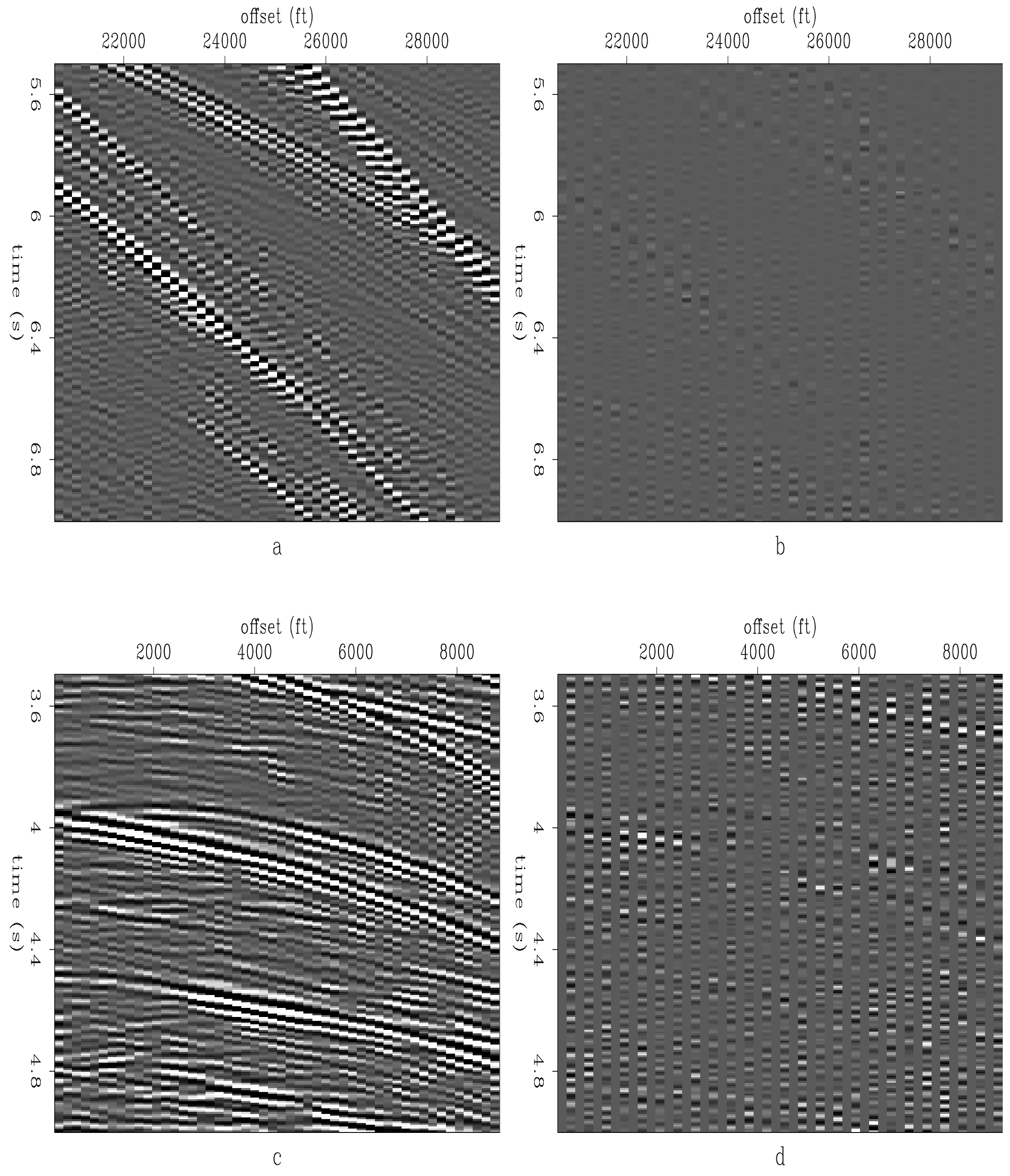
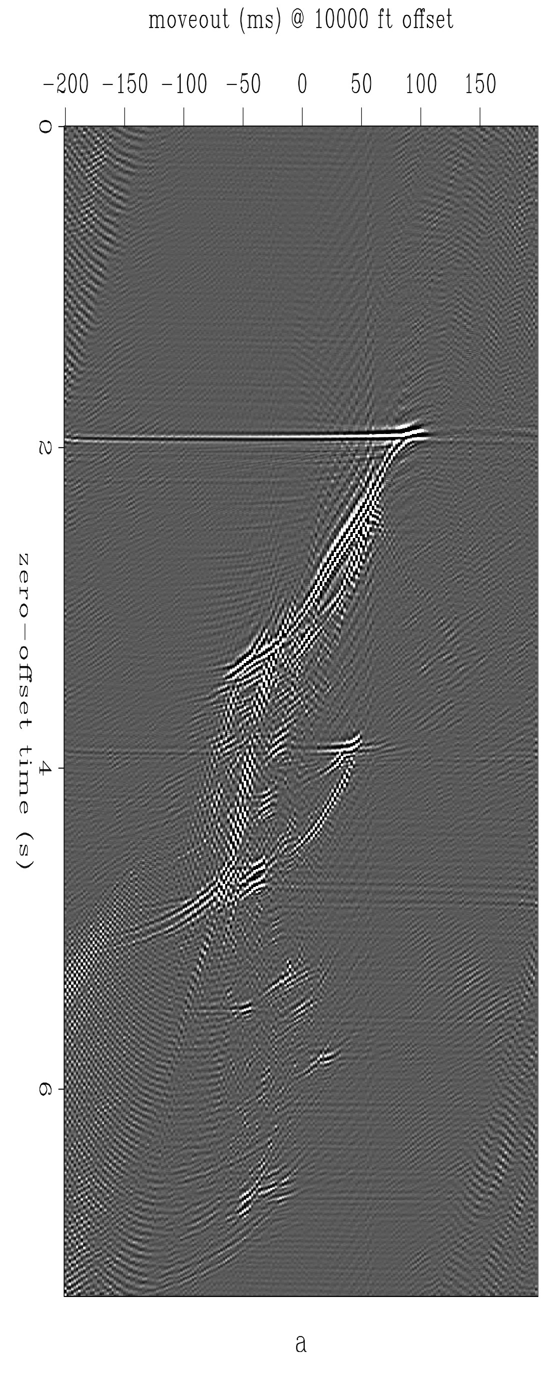


Figure 2.20: CMP difference panels in closeup. The left two panels show close ups on portions of the original CMP gather, shown in Figure 2.19. The right panels show the difference between the interpolation result and the original data in those two portions. The differences are noticeably smaller where the events are mostly linear. bp-bpcmpdiff [CR]

gather used in Figure 2.18, but constructed from the interpolated survey. Figure 2.21b shows the result of subtracting 2.21a from Figure 2.18a. The difference is very small, implying that the difference between the results of Radon demultiple on the full data and on the interpolated data would be small. Most importantly, results on the interpolated data should be better than on the subsampled input data, because the artifacts which are obvious in Figure 2.18b do not appear in Figure 2.21a.

Results in the poststack domain

Figure 2.22 shows a subsalt portion of a stacked section. The left panel shows the result of radon demultiple and stack on the decimated data, the right panel shows the result of simple stack on the interpolated data. The reason for the difference in flow is that simply stacking the data which has been dealiased by interpolation (leaving out the innermost few traces) removes most of the multiple energy, but doing a similar operation on the decimated data (left panel) gives a terrible result, because the aliased nature of the multiples causes them to come through in the stack as a short-period (50ms or so) series of seafloor multiples that completely obscures the section. At any rate, the right panel is approximately identical with or without going to the trouble of radon transform. Strong multiple events (examples at **A**) are suppressed more or less equivalently in the two sections; arguably somewhat better in the interpolated data. Some important primaries at **B** are significantly attenuated in the decimated data, as shown by comparison with the interpolated data. Also, throughout the section, most visible in the neighborhood of **C**, aliasing results in layer-like artifacts. Finally, many interesting arrivals, especially diffractions, in the faulted area around **D** are strongly attenuated in the decimated data.



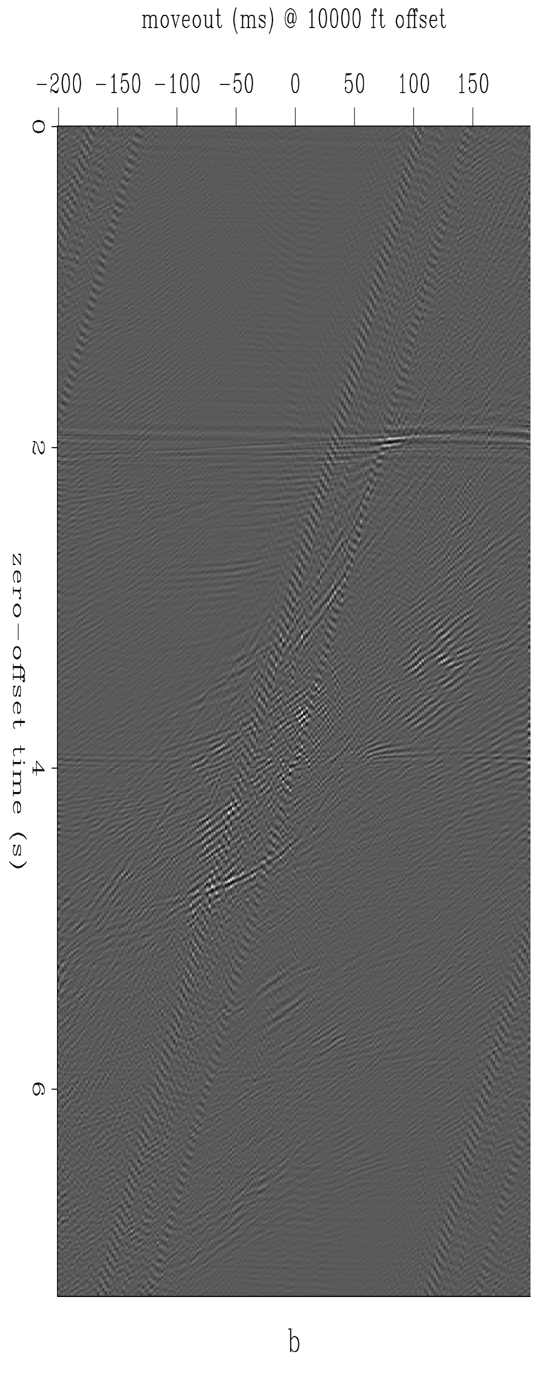


Figure 2.21: Radon transform of interpolated data, and difference panel. The left panel is the Radon transform of a CMP gather constructed from the subsampled, interpolated data. Ideally, it should be the same as the Radon transform of the original data. The right panel is the (thankfully small) difference. $\boxed{\text{bp-interprads}}$ [NR]

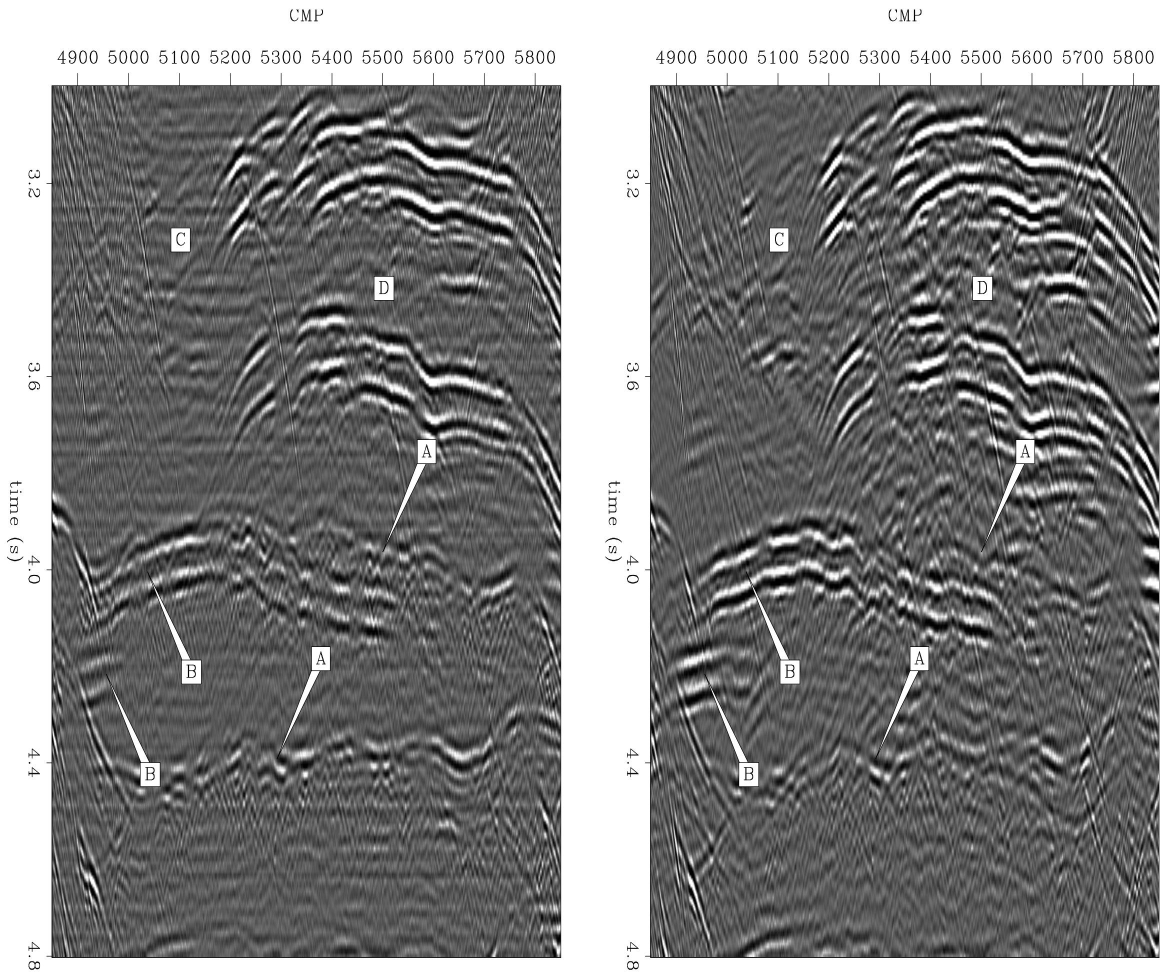


Figure 2.22: Portion of unmigrated stacked sections. The left panel shows decimated data after demultiple and stack, the right panel shows same data after interpolation, demultiple, and stack. Multiples (**A**) are arguably better attenuated in the interpolated data. The decimated data shows attenuated primary events (**B**), attenuated diffractions (**D**), and layer-like artifacts (**C**). bp-stacks [NR]

Chapter 3

Interpolation with adaptive PEFs

In this chapter, I introduce a new method for dealing with the nonstationarity of the input data. In the previous chapter, the data are divided into rectangular patches and assumed to be stationary within a patch. Each patch is an independent interpolation problem. This implies that seismic events have piecewise constant dips in prestack data. In general that is not true, though it can be nearly true in cases, particularly at long offsets, where seismic events tend to approach their asymptotes. In this chapter, instead of independent patches, we divide the data into smaller regions, as small as a single data sample, and assume that the dip of seismic events varies gradually with location in the data. We refer to the new, smaller regions as "micropatches." They are no longer independent problems, but a series of related problems. Following the literature, we say we have moved from block estimation to adaptive or nonstationary estimation.

Instead of independent problems in patches, we smooth between micropatches. We apply a smoother between values at identical lags of different PEFs, in such a way that PEFs which are applied at similar data coordinates are averaged together. For statistical robustness, we want to smooth as much as possible, but we want to avoid making a roundabout assertion of stationarity where the data are not stationary. To maximize the area in data space over which PEFs are averaged, without averaging nonstationary regions together, we can choose the directionality of our smoother. CMP gathers tend to have nearly constant dips in regions along radial lines, so we choose the radial direction.

The filter calculation part of the problem becomes large and underdetermined. Happily, however, it converges in few iterations, and turns out to be cheaper and lead to better interpolation results than interpolation in independent patches.

ARGUMENTS AGAINST PATCHING

Independent patches work reasonably well in many cases, but there are some arguments against them. One easy argument against patching is that it effectively increases the size of the data. Patches usually need to have significant overlap in order to get a good interpolation result at the end. Along a single axis, a given data point is contained in 1.5 to 2 patches on average. For a 3-D cube of input data, as in a prestack 2-D survey or a prestack 3-D survey considered as individual source/streamer combinations, that effectively increases the data volume by 3.5 to 8 times.

Using adaptive PEFs means a larger volume of PEFs rather than of data. The larger volume of PEFs does not add to the number of computations required the way that a larger data volume does. The computational cost of convolutions increases with the size of the data and the number of coefficients that multiply each data point (the size of a single filter), but is indifferent to whether the coefficients that multiply two different data points belong to the same filter or different filters.

More important than the cost, there is the argument that the data are not really aligned along linear, constant slopes. There are often large portions of the data that nearly are, usually at late times and large offsets, where events approach their asymptotes. However, there are also portions of the data that do not fit that model, especially near the apex of hyperbolas, where events have the most curvature. As shown in Figure 2.20, where events have significant curvature, interpolation results suffer. Modifying our assumptions to fit curvy data should help us get better interpolation results.

Also, while there is some convenience in the notion of dividing the interpolation into many small independent problems, it ignores some potentially useful information. Dips in prestack seismic data are not independently distributed throughout the data volume. The seismic data response of a point reflectivity anomaly is large relative to an interpolation patch, and of a somewhat predictable shape, even with variable velocities. In other words, even when events are nonhyperbolic, they are likely to be nearly hyperbolic, in some rough sense.

Instead of filters in independent patches, in this chapter we estimate sets of filters in smaller, non-overlapping, non-independent micropatches. Micro-patches which are near each other in the data volume are assumed to have similar sets of dips, and thus similar filter coefficients. We assume that dips of seismic events change gradually as we move around in the data. The micropatches are small enough so that events with tight curvature can be still be reasonably resolved into linear events, but large enough to avoid excessive memory consumption brought on by allocating more filters than necessary.

IMPLEMENTATION OF ADAPTIVE PEFS

Interpolating with adaptive PEFs means calculating a large volume of filter coefficients. It is possible to estimate all these filter coefficients by the same formulation as in the previous chapter, supplemented with some damping equations, like

$$\mathbf{0} \approx \mathbf{YKa} + \mathbf{r}_0 \tag{3.1}$$

$$\mathbf{0} \approx \epsilon\, \mathbf{Ra} \tag{3.2}$$

where **R** is a roughening operator, **Y** is convolution with the data, and **K** is an adjustable filter coefficient selector. **R** does not roughen between coefficients within a single filter, but between coefficients at the same lag in different filters.

When the roughening operator **R** is a differential operator, the number of iterations can be large. To speed the calculation immensely, we can precondition the problem. Define a new variable **p** by $\mathbf{a} = \mathbf{Sp}$ and insert it into (3.1) and (3.8) to get

$$\mathbf{0} \approx \mathbf{YKSp} + \mathbf{r}_0 \tag{3.3}$$

$$\mathbf{0} \approx \epsilon\, \mathbf{RSp} \tag{3.4}$$

Now, because the smoothing and roughening operators are somewhat arbitrary, we may as well replace **RS** by **I** and get

$$\begin{aligned} \mathbf{0} &\approx \mathbf{YKSp} + \mathbf{r}_0 \qquad (3.5) \\ \mathbf{0} &\approx \epsilon\, \mathbf{Ip} \qquad (3.6) \end{aligned}$$

We solve for **p** using conjugate gradients. To get **a**, just use $\mathbf{a} = \mathbf{Sp}$. Once **a** is calculated, the missing traces are filled in as before. To simplify the formulation, one could drop the damping (3.6) and keep only (3.5); then to control the null space, start from a zero solution and just limit the number of iterations. This is the way most of the examples later in this chapter are calculated.

Previously we solved for the PEFs **a**, which have a fixed coefficient that is defined to have the value 1. We instead estimate **p**, which is related by $\mathbf{a} = \mathbf{Sp}$. It appears troublesome that we do not necessarily know the fixed coefficient of **p**. We can begin by applying $\mathbf{p}_0 = \mathbf{S}^{-1}\mathbf{a}_0$, putting some other value in the fixed coefficient of **p**, that will be integrated by **S** to give 1's. But it is a hassle to then apply **p** to the data because our software has the value 1 built in. Luckily, the problem disappears by itself. Wherever the forward operator is applied, it looks like **YKSp**, which is the same as **YKa**. We only need to apply **S** to the adjustable coefficients of **p**, because we know the fixed coefficient of **Sp** equals one, even if we do not know the fixed coefficient of **p**. We do not need to know or store the fixed coefficients of **p**.

RADIAL SMOOTHING

We have to choose **S**. Inserting a smoother signifies the assertion that the dips in seismic data should change in a gradual way. Choosing an isotropic smoother means we expect the dips to vary similarly in all directions. However, we know that the dip spectrum of the data probably changes more quickly in some directions than in others. We want to smooth most heavily along directions where the dip is nearly constant.

In a constant-velocity, flat-layered earth, events fall along hyperbolas like

$$t^2 = \tau^2 + \frac{x^2}{v^2},$$

where x is offset, v is stacking velocity, τ is zero-offset time. The time dip of an event is dt/dx. If velocity is constant, differentiating gives

$$\frac{dt}{dx} = \frac{x}{v^2 t},$$

which means that the dip does not change along radial lines, where x/t is constant. In a real earth, we suppose that dips will change, but slowly. Real earth velocity may change quickly in depth, but hyperbola trajectories are functions of RMS velocity, which is smooth.

We want a smoother with an impulse response which is highly elongated in the radial direction. To get a big impulse response cheaply, I apply the inverse of a directional derivative, pointed in the radial direction. To directly apply the inverse, the roughener has to be causal, which means that the inverse will only smooth in one direction. We want **S** to have an impulse response which is smoothed both in towards zero radius and out towards large radius, so we make it the cascade of the causal smoother and its anticausal adjoint.

Patches, micropatches, pixels

Having chosen the radial direction, we can think of some different ways of implementing our radially-smoothed filters. An obvious one is putting a PEF at every point on the data grid, and devising a derivative filter which adjusts its direction to point at the origin. An alternative is to overlay a radial grid on the data grid, and arrange PEFs on the radial grid. Here we compare the two smoothing schemes.

Our goal is to assume stationarity in a small enough region that we can interpolate well where the data do not fit a plane-wave model. In the method of independent patches, individual patches are treated as separate problems. A patch can not be arbitrarily small, because it must provide enough fitting equations that the filter coefficients are well overdetermined. In 1-D, filtering with a PEF looks like this:

$$\hat{u}(i) = \sum_{k=0}^{p} a(k)u(i-k), \qquad p + i_{\text{patchmin}} \le i \le i_{\text{patchmax}}. \tag{3.7}$$

The patch boundaries are i_{patchmin} and i_{patchmax}, **p** is the number of adjustable coefficients, $a(0) = 1$. The lower limit on i is to prevent the filter from running off the end of the data and encountering implicit zero values. The patches are designed to overlap, and the outputs are normalized to hide the patch boundaries.

In moving to the method of gradually-varying PEFs, we replace the notion of extracting a subset of the data with that of dereferencing the data coordinates to find the appropriate filter, as in

$$\hat{u}(i) = \sum_{k=0}^{p} a_i(k)u(i-k), \qquad p + 1 \le i \le i_{\text{datamax}}. \tag{3.8}$$

The index of the data sample i dereferences the set of filters $a_i(k)$. The data boundaries $i = 1$ and i_{datamax} replace the patch boundaries.

We have lots of freedom in dereferencing a_i. In the limiting cases, all the data may share one PEF, or we can choose a_i to be a different set of coefficients for each data point. In the case where we have a PEF at every data point, we call **S** pixel-wise smoothing.

Choosing a separate PEF for every input sample is a possibility, but not necessary. Our motivation for moving away from independent patches was to use one PEF in a region small enough that we do not have trouble with nonstationarity. Some amount of patching may still make sense, provided the patches may be small. It is easy to implement small patches as a generalization of the case above where each data sample has its own PEF. A particular a_i can be the same for any number of values of i without complication. Because they may be small, we refer to the new patches as “micropatches” to distinguish between them and the independent patches of the previous chapter.

To subdivide a CMP gather into micropatches, we choose a web-like grid made up of radial lines and circular lines. Radial lines are a natural choice because we want to smooth in the radial direction. Circles are somewhat arbitrary; we could choose flat lines or reflection-like hyperbolas to cross the radial lines. Circles have the attractive property that they make equal-area micropatches at a given radius.

Smoothing pixels versus smoothing micropatches

We can choose pixel-wise smoothing or micropatch smoothing. An easy argument favoring micropatches over pixel-wise smoothing says that putting a filter at every data sample is a tremendous waste of memory. If the data are predictable at all, they are probably not so nonstationary that they need a separate PEF at each sample. A single 3-D PEF has easily 20 or more adjustable coefficients, so allocating the set of PEFs requires 20 times the storage of the input data. Even very small micropatches require much less memory.

Micropatches also have some simplifying side effects that make them preferable to pixel-wise smoothing. One is apparent from examining Figures 3.1 and 3.2. Figure 3.1 shows smoothing in micropatches and Figure 3.2 shows pixel-wise smoothing. In each figure, the values represent filter coefficients displayed in data coordinates. The axes are time and offset. The top halves show a set of impulses, which I label $\mathbf{d}$. $\mathbf{M}$ is the operator which bins the impulses into micropatches, while $\mathbf{P}$ bins into pixels (naturally, $\mathbf{Pd} = \mathbf{d}$). The variously binned impulses are shown in the panels labeled $\mathbf{Md}$ and $\mathbf{Pd}$. $\mathbf{F}$ and $\mathbf{F}'$ are pixel-wise smoothing operators pointed towards and away from zero radius, respectively. $\mathbf{C}$ and $\mathbf{C}'$ are the corresponding micropatched smoothers. The impulses are smoothed using the pixel-wise operator in the panels labeled $\mathbf{FPd}$ and $\mathbf{F}'\mathbf{FPd}$; smoothing with the micropatched operator is in panels $\mathbf{CPd}$ and $\mathbf{C}'\mathbf{CPd}$. In this case, the two are similar, though the pixel-wise smoother obviously produces a higher-resolution picture (though the micropatches could be made much smaller).

The same exercise is repeated in the bottom halves of the two figures, this time using a constant function instead of impulses. $\mathbf{M1}$ has an angular limit applied. Pixel-wise smoothing creates some very large ridge artifacts, visible in $\mathbf{FP1}$ and $\mathbf{F}'\mathbf{FP1}$, where the angle between a data sample and the origin corresponds to an integer slope. Also, where the constant function is smoothed in towards zero radius, $\mathbf{FP1}$, energy concentrates in a huge spike at the origin.

Whichever smoother ($\mathbf{F}'\mathbf{F}$ or $\mathbf{C}'\mathbf{C}$) we choose will be used for $\mathbf{S}$ in equation (3.5). $\mathbf{F}'\mathbf{F}$

and $\mathbf{C}'\mathbf{C}$ can be thought of as weighting functions. It is desirable to have the flatter weighting function $\mathbf{C}'\mathbf{C}$. It is also simpler to implement. Producing the many different angles in Figure 3.2 requires that the smoothers $\mathbf{F}$ and $\mathbf{F}'$ be made up of many different filters, oriented in a continuous sweep between a spatial derivative and a time derivative. $\mathbf{C}$ and $\mathbf{C}'$ produce the same range of angles in Figure 3.1 using a single, radial derivative filter. The PEFs in micropatches are regularly gridded in angle and radius, so they are easily smoothed in those directions with old-fashioned stationary 1-D derivative filters. PEFs at every pixel are instead regularly sampled in time and offset, so working in polar coordinates requires some work. $\mathbf{F}$ uses many coefficients, $\mathbf{C}$ uses two.

Is smoothing necessary?

An unexplored alternative to radial smoothing may be to simply lengthen the micropatches in the radial direction, and not bother smoothing at all. I have not tested this direction very thoroughly, though in the tests I have done, it seems that smoothing may have some important effects beyond just statistically compensating for the small size of a micropatch. Even with very elongated patches, my experience has been that smoothing noticeably improves the final result, particularly where the data are noisy. One possible explanation is that where the data are incoherent, the change in a particular filter coefficient at each iteration is just an average of noisy data samples, which is approximately zero. With the addition of the smoother, the change in nearby filter coefficients fills in.

Radial smoothing and variable velocity

We chose the radial direction because dip of seismic events is constant along radial lines. But that is only true in ideal circumstances. On radial lines, dip is constant if stacking velocity and geologic dip are constant. Naturally, things are more complicated in real life. Velocity is generally not constant. Multiples and complex structure can introduce all manner of unpredictable dips, in all types of gathers. Nonetheless, radial smoothers work. My experience has been that the radial smoother works as well as or better than an approximately isotropic blob smoother, on common-shot, common-receiver, and CMP gathers. Figure 3.3 shows an

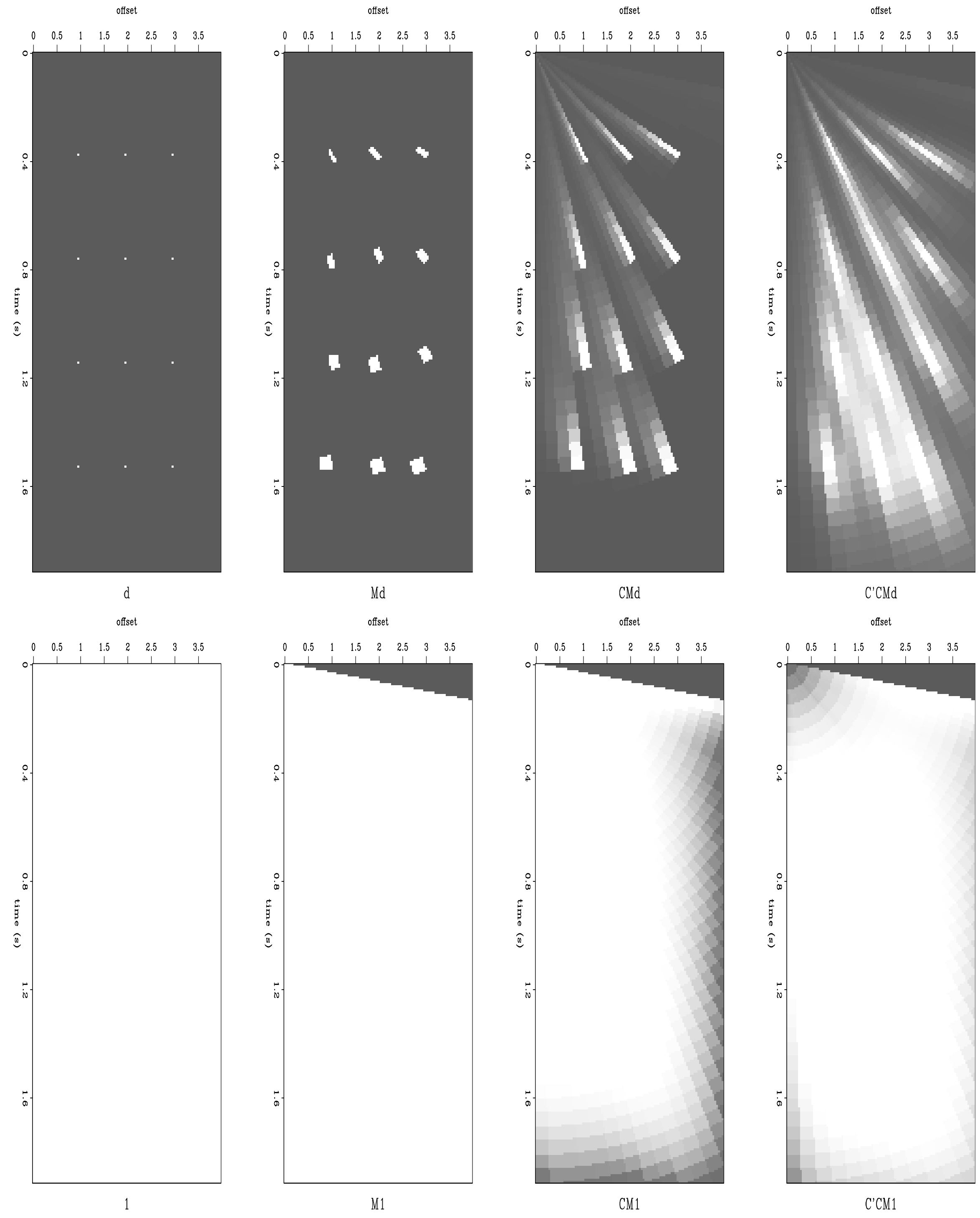


Figure 3.1: Illustration of micropatched radial filter coefficient smoothing. sm-curtSmear8 [ER]

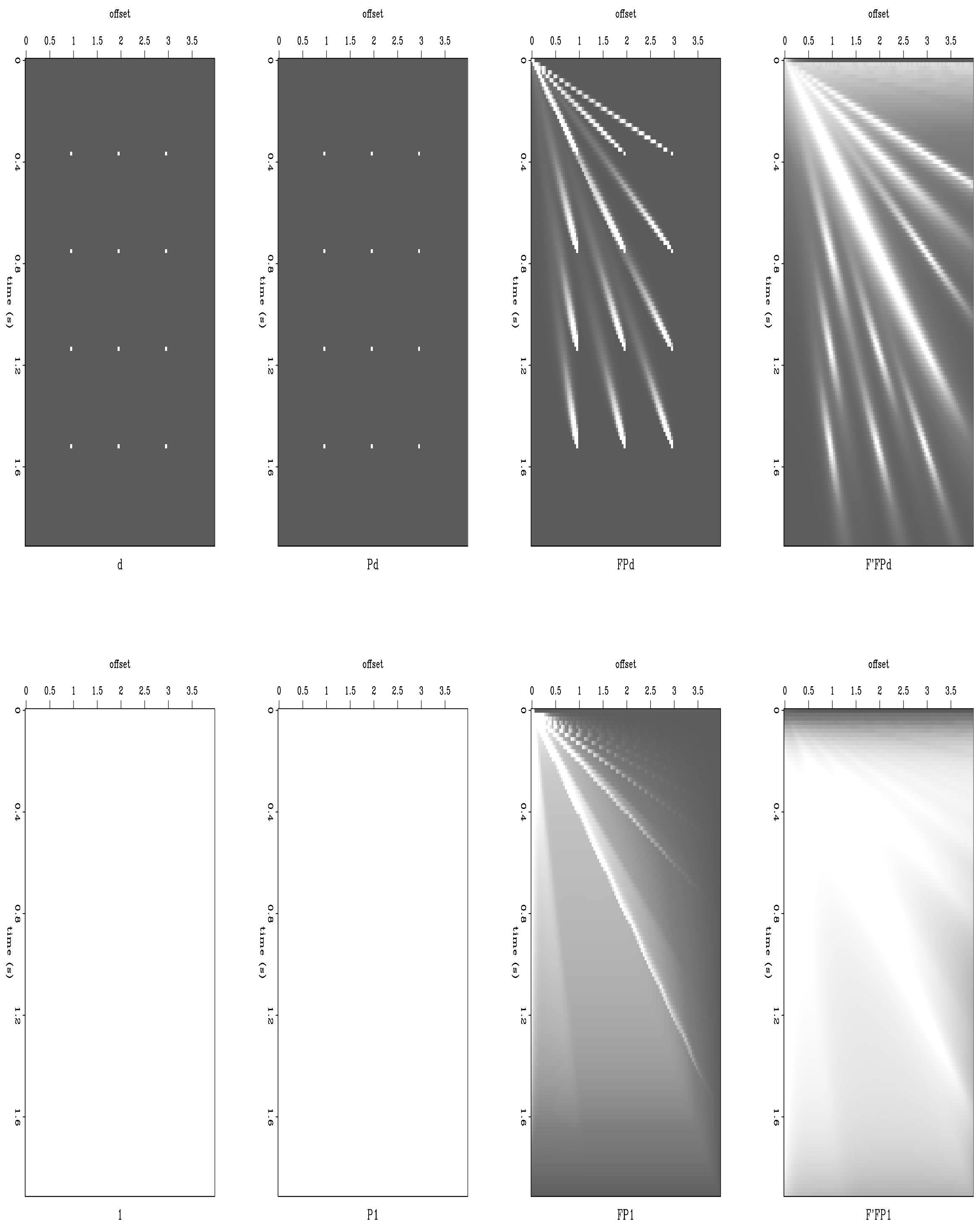


Figure 3.2: Illustration of pixel-wise radial filter coefficient smoothing. $\boxed{\text{sm-random8}}$ [ER]

example of an interpolation result using radially-smoothed PEFs to interpolate, and the same result using isotropically smoothed PEFs. The results are similar, but the radially-smoothed PEFs did a better job at interpolating the events dipping back towards zero offset. Inspection of the figures shows that all those events have about the same dip. The isotropic smoother had a similar number of coefficients, but did not do as well. Smoothing in all directions implies smoothing along events where their dip is changing, while smoothing radially implies smoothing along events only where they reach their asymptotes. A PEF can predict more than one dip, but not too many. In the isotropic case, a filter may be averaged over a region containing all manner of dips. For instance, near the apex of a hyperbola there is a range of positive and negative dips. At any rate, it is pleasing that the radial scheme works well.

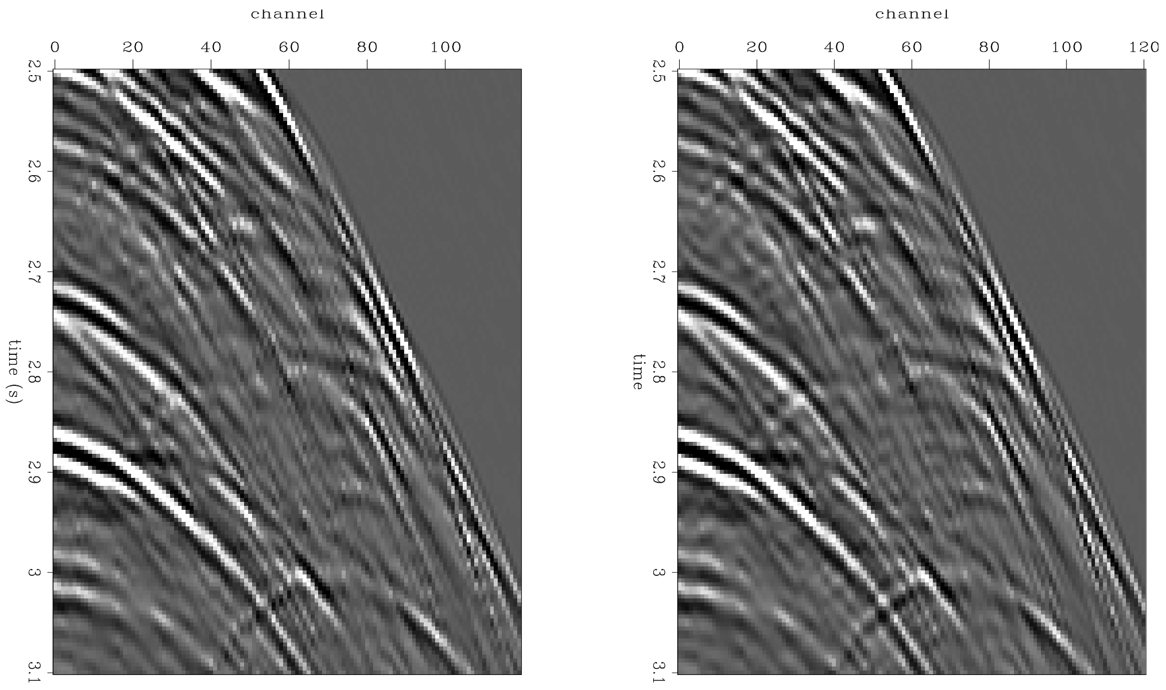


Figure 3.3: Comparison of interpolation results using isotropic and radial filter smoothing. Both results are reasonably good, but the radial smoothing result (right panel) is better where events are flat or dip back towards zero offset. sm-radblob [ER]

PARAMETER SPACE

The parameters for smoothing and micropatching, the size and shape of the PEFs, and the number of iterations to spend on filter calculation and missing data calculation all make for a large parameter space. And unfortunately, results tend to be sensitive to at least a few of the parameters.

The numbers of iterations are important parameters, especially the number of filter calculation iterations. Too many iterations during the filter calculation step results in a noticeably degraded final result. A small amount of damping is very good at reducing sensitivity to the numbers of iterations, though the damping parameter ϵ must be chosen, and a bad choice can also contribute to a bad result. Either way, a little experimentation usually will provide the answer. Most of the examples in this and the next chapter are done with $\epsilon = 0$.

The size, shape, and density of PEFs can also lead to bad final results, if chosen poorly. In my experience the effectiveness of a set of filter parameters is largely independent of the lengths of the slow axes (shot number, midpoint number, etc.) of the input data. Put another way, the settings that work for one or two input gathers will generally work for any number of gathers, so it easy to test parameters quickly.

Another parameter that can significantly affect the outcome is the ratio of original spacing to interpolated spacing. In principle, it can be any integer, but in practice it should usually be two (as it is in most of the examples). There is no reason not to interpolate several times in a row, and this is usually the best way to get some larger ratio of sampling intervals. One reason is that scaling the axes of a PEF by a large factor decreases the range of frequencies that the PEF can reliably detect. The fix is to resample the time axis of the data, but that also means lengthening the PEF in time, in order to maintain the same range of dips. The extra degrees of freedom in the longer filter can result in spurious events and so forth. Even in simple tests, where the data is low frequency and does not require temporal resampling, my experience has been that interpolating in multiple steps works better than quadrupling (or more) the data in one step. As an example, Figure 3.4 compares interpolation results where some very low frequency data have their offset sampling density quadrupled in two steps and in one step. The left panel shows the original data. The original data were lowpassed so that energy would lie below one quarter of its temporal Nyquist. The center panel shows the result of interpolating twice, doubling the data each time. The right panel shows the result of interpolating a single time, quadrupling the data. Something has obviously gone wrong in the right panel.

Figure 3.5 shows what happens in Fourier space. All the panels in this figure are cropped on the temporal frequency axis for clarity. The top left panel is the 2-D Fourier transform of the original, well-sampled data. The top center panel is the Fourier transform of the data after replacing three out of four traces with zeros. Zeroing the traces is a sampling operation, which leads to replication in the Fourier domain. The top right panel is the Fourier transform after attempting to interpolate all the zero traces at once. The interpolator needs to zero all the replicate versions of the input data's spectrum, and leave the originals alone. However, PEF spectra tend to be very simple (as in Figures 2.7 and 2.8), since a PEF is made up of a handful of coefficients. The PEF that passes the original part of the spectrum also passes the replicate spectrum that is aligned approximately along the same diagonal line in Fourier space. The two blobs in Fourier space that line up with the original data spectrum are only partially attenuated in the top right panel. Those events correspond to the same temporal frequencies, but higher wavenumbers, and so in the time domain the result has anomalously high dips (shown in the right panel in Figure 3.4).

The bottom three panels of Figure 3.5, show the process of interpolating in two steps. The bottom left panel is the Fourier transform of the data with zero traces, after removing half the zeros. This panel is just like the top center, with the outermost spatial frequencies removed. In this case there is no trouble with original and replicate spectra aligning, and it is straightforward to interpolate. The bottom center panel shows the halfway point in the two steps of interpolation. This is the Fourier transform after interpolating and then reinserting the zeros that were removed. Again there is a replicate of the original spectrum to remove, but it does not line up with the original, so it is straightforward to remove it. The Fourier transform of the interpolation result is in the bottom right panel, the result in the time-domain is shown at the right side of Figure 3.4.

Manin and Spitz (1995) may allude to something similar, when they suggest that if fold is very low, the data should be interpolated in multiple steps along different directions.

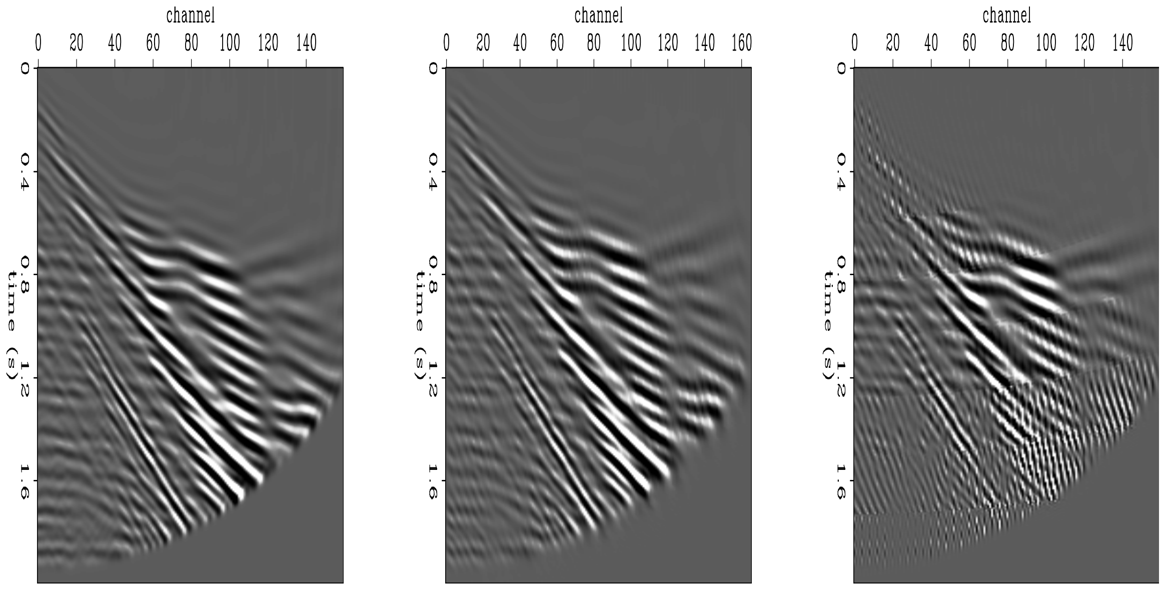


Figure 3.4: The difference between refining data sampling in one step and in multiple steps. The data in the left panel were subsampled by keeping every fourth offset, and zeroing the other three. The center panel shows the result of interpolating the data in two steps. The right panel shows the result of interpolating the data in one step. sm-curtlow [ER]

SHOTS VERSUS RECEIVERS, 2-D VERSUS 3-D

There are a couple of choices to make concerning how to arrange the input data. The first is really a matter of sorting. Since the data are not really being addressed by physical coordinates but rather by shot number and channel and so forth, it makes sense that the way the data are sorted might affect the output, because sorting changes the distribution of recorded and nonrecorded traces. This is shown in the previous chapter, in Figures 2.4 and 2.5. One cube can be made from the other by sorting. It is intuitively appealing to have each missing trace surrounded on all sides by known data, as in Figure 2.4.

Another important choice is how much data to work on at one time, and whether to split the data in such a way that its dimensionality is reduced. Some of the data sets used in the examples are 2-D data sets, and some are 3-D. In the case of marine data, there really is very little difference between 2-D data and 3-D data, and the same routines are used. In the case of land data, 3-D can be vastly more complicated than 2-D.

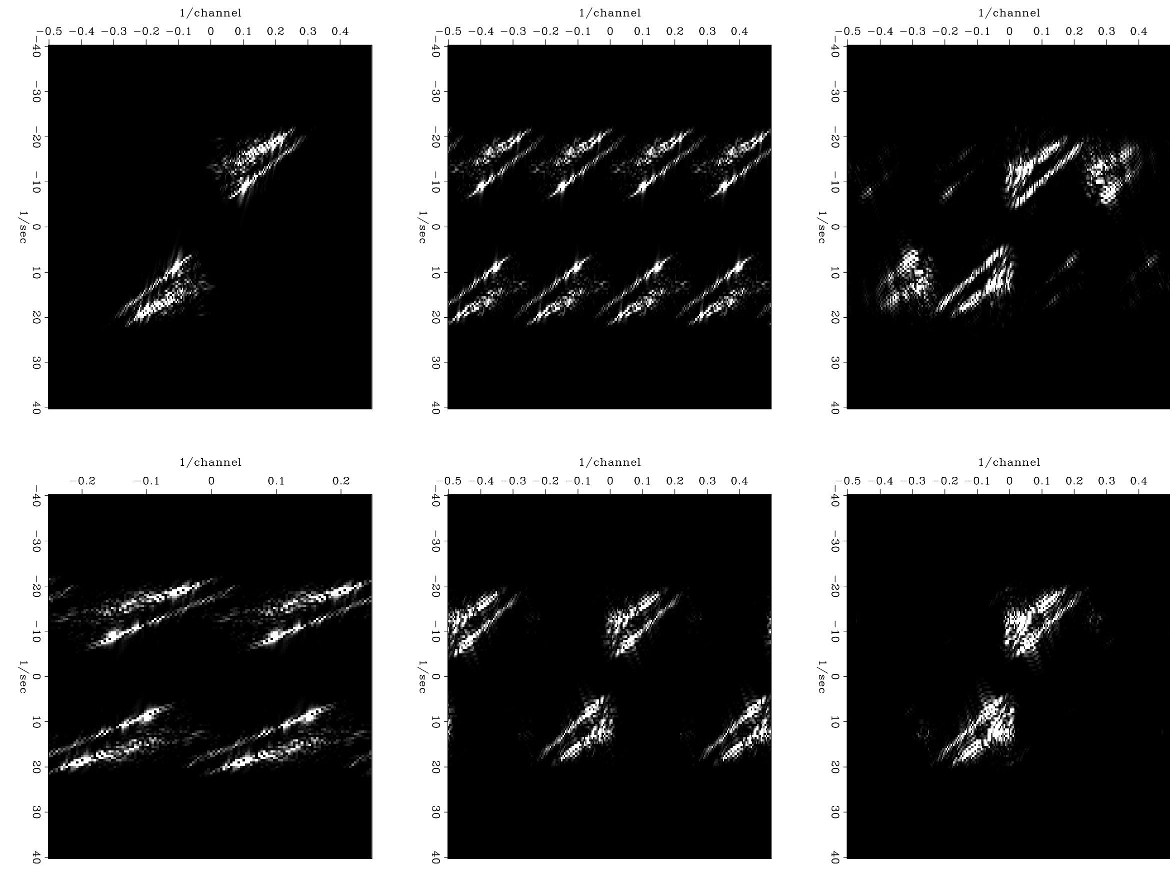


Figure 3.5: Fourier transforms illustrating the difference between refining data in one step and in multiple steps. The top left is the spectrum of the original data. The top center is the spectrum after replacing three fourths of the traces with zeroes. The top right is the result of interpolating all the empty traces at once, it is not a good facsimile of the original spectrum at top left. The bottom left is the spectrum after removing two thirds of the zero traces. The bottom center is the spectrum after interpolating the remaining zero traces, and reinserting the zero traces which were removed. The bottom right is the spectrum of the final interpolation, and it is a much better estimate of the top left panel. $\boxed{\text{sm-curtlowspecs}}$ [ER]

A 2-D prestack data set is a 3-D cube, with axes for time, midpoint, and offset; or time, shot, and receiver. A 3-D prestack data set is a 5-D cube of data, where the surface axes each have an inline and crossline component. In marine acquisition, the boat typically tows only two sources, and so it is easy enough to consider a sail line to be two 4-D cubes of seismic data. Because the crossline receiver aperture may be small, with only a few streamers, it makes sense to decompose each of those two 4-D cubes into a series of n_{streamer} 3-D cubes. It is helpful to pad a couple of zeroes onto each axis during the filter calculation step. If the data are only a few points wide along one axis, then padding that axis can easily double the size of the data. It also significantly increases the number of filter coefficients, if the filter has some width along the extra axis. Thus for a boat with few streamers, including the crossline receiver axis can easily double the size of the input data, making convolutions that much more expensive, but probably not improving the final results noticeably.

There are benefits to using more dimensions. Having an extra direction for things to be predictable in often means a better interpolation result. Calculating a PEF from a 2-D plane of white noise will produce a filter that works like an identity. That filter will then not insert anything into any missing samples. If the plane of white noise is a slice from a cube or hypercube that contains coherent energy along some other axis, then a PEF that is calculated with some coefficients spread along that coherent axis will be a good interpolator.

My experience has been that a three dimensional input cube (many 2-D gathers) works noticeably better than a single two dimensional gather. Four dimensions work somewhat better than three, but the returns diminish as the misfit gets small. At any rate, on most examples I have done, three dimensions turns out to be plenty to get a good interpolation result. Nearly all of the examples in this thesis are done with three dimensions of input. If there are only a few streamers, working in 4-D brings many extra computations because of padding, and because adding one to the dimensionality of the PEFs significantly increases the number of coefficients in each filter. It is probably not worth the extra computations if a good result comes from 3-D. If there are many streamers, as is the case with some modern boats which may have 10 or 20, or if results with three dimensions are not good enough, then it would make sense to use a 4-D input.

As a final note, the notion of deciding whether or not to filter along the crossline receiver axis assumes that that is not the axis you want to resample. However, one can certainly imagine wanting to interpolate new streamers, to refine the typically sparse crossline receiver sampling. In that case, you would naturally want to predict in that crossline receiver direction. This is done in an example later in this chapter.

DATA EXAMPLES

Marine 2-D shot interpolation example

Figure 3.6 shows a slice from a cube of 2-D shot gathers. The data have long-period multiples, and complicated moveouts caused by a tabular salt body. The geologic structure in the area is similar to the model used to generate the synthetic data in the previous chapter, though this data has the virtue and added complications of being real field data. This data set is 2-D, with shot sampling and receiver sampling equal. As in the last chapter, we can throw out alternating receivers in a checkerboard pattern to simulate common receiver gathers formed from a single inline from a 3-D survey with flip/flop acquisition. We can also just throw out alternating shots to simulate common shot gathers from the same survey.

As a test, we interpolate both ways. Figure 3.7 shows a closeup of the input, zoomed in on regions between 2 and 4 seconds in Figure 3.6. Figure 3.8 shows the interpolation result on common shot gathers. Figure 3.9 shows the interpolation result on simulated common receiver gathers. The traces shown in Figure 3.9 are the same as in the other figures, but traces were zeroed in a checkerboard pattern (as in Figure 2.4) to simulate the pattern of known and missing traces that arises from forming common receiver gathers after throwing out every second shot. In the other case, where the data are just representing shot gathers, every second slice in the cube was zeroed (as in Figure 2.5).

The most obvious difference between the results in Figures 3.8 and 3.9 is in the events which dip back towards zero offset. The data which are interpolated in slices have lost most of these events, while the data which are interpolated in the checkerboard pattern restored those somewhat. In the best case, just under 95% of the variance of the original data is predicted.

An unexplored alternative to resorting the data may be to rotate the filters by 45 degrees.

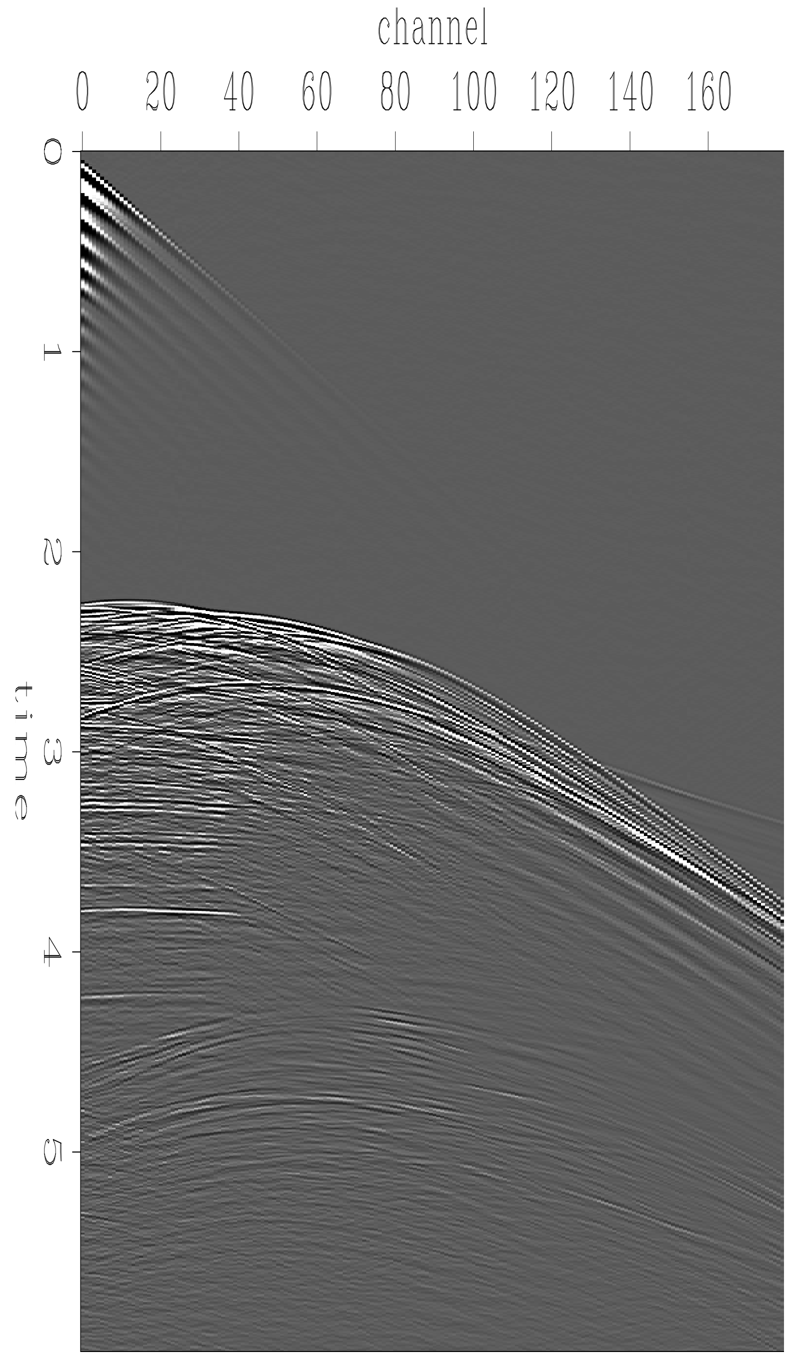


Figure 3.6: Sample shot gather from the original data. sm-355exshotgather [ER]

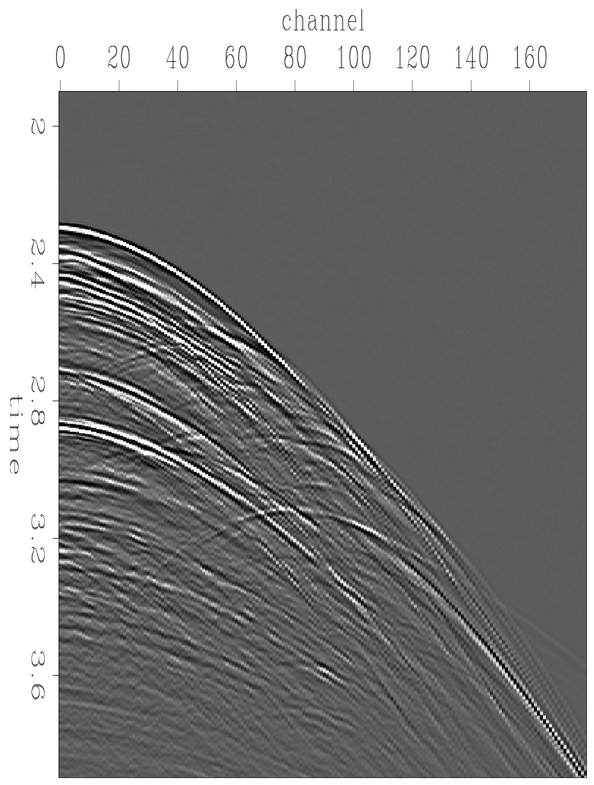


Figure 3.7: Portion of a marine shot gather. Part of the original data that is subsampled and interpolated to make the next figures. sm-355start.shot [ER]

Marine 3-D shot interpolation example

Figure 3.10 shows another shot gather, this from one a 3-D marine survey. While in previous examples I threw out portions of a 2-D survey, to simulate a flip/flop acquisition, this data actually was collected with a flip/flop acquisition. Thus we can not do exactly the same experiments as with the previous 2-D examples. On the other hand, the dips in this data are more routine than those in the previous data, so it should be subsampled more to make an interesting test. Instead we start from the slightly subsampled data (using one source from a two-source survey), and remove half the shots again, so the shot interval is four times the receiver interval.

Figure 3.10 shows one of the shot gathers that was removed, and the interpolated version of that shot gather is shown in Figure 3.11. The interpolated shot gather is extremely faithful to the original, predicting 94% of its variance. The most noticeable differences are back-scattered noises at late times and short offsets which are not interpolated well. In the electronic version of this document, Figure 3.10 is a two-frame movie, comparing the original and interpolated

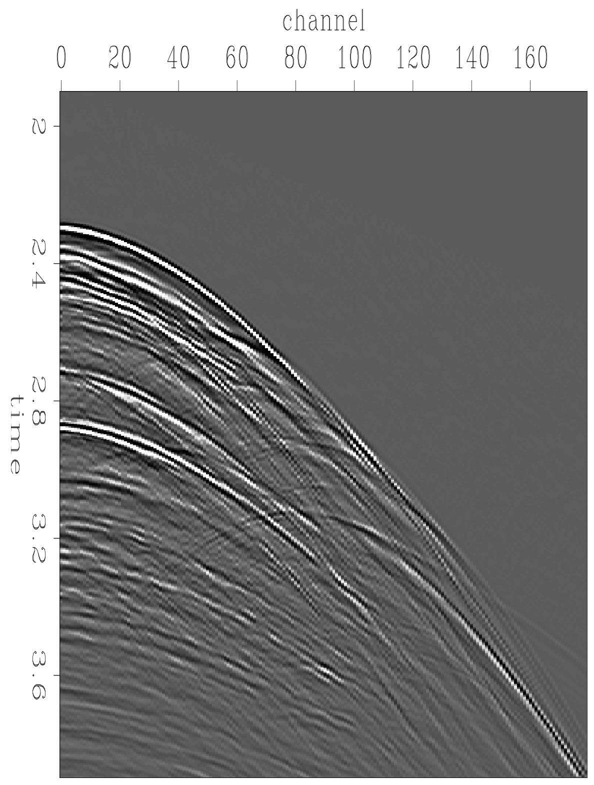


Figure 3.8: Portion of an interpolated marine shot gather. This is the result of subsampling the shot axis and interpolating with the data sorted as shot gathers. sm-355shotsub.out.shot [ER]

gathers.

Figures 3.12 and 3.13 show another view. These are time slices through cubes made up of one source and one streamer. The three slices in each figure correspond to the three steamers in the survey. Not surprisingly, the time slices appear slightly smoothed along the shot number axis, where the data were subsampled. The visible events are all continued in a coherent manner, but may be a bit too coherent in areas where some high-wavenumber fluctuations of events were removed by the subsampling, and then replaced in a smooth fashion by the interpolation.

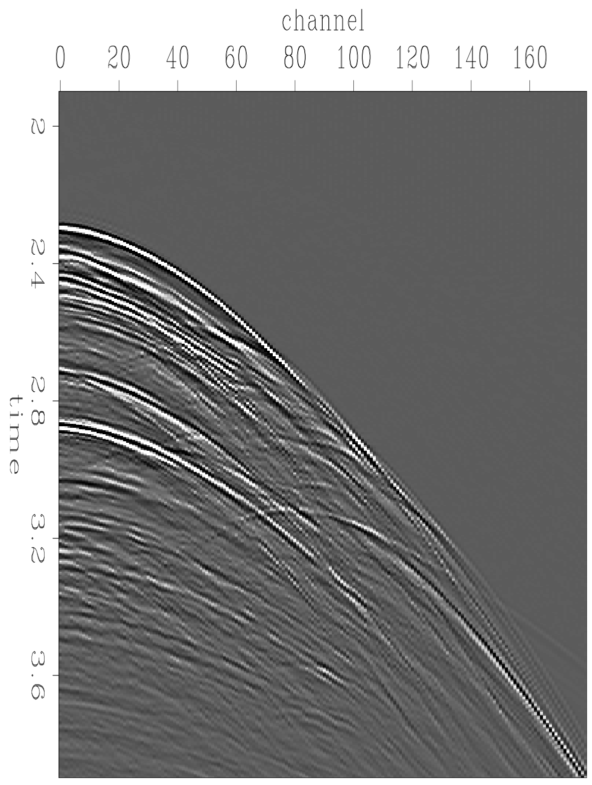


Figure 3.9: Portion of an interpolated shot gather. This result simulates subsampling the shot axis and interpolating with the data sorted as receiver gathers. sm-355recsub.out.shot [ER]

Interpolating crossline in the 3-D marine case

For some applications, it may make sense to refine the crossline receiver sampling. Prestack datuming, for instance, can be hampered in 3-D because the relatively sparse crossline sampling requires much stronger antialiasing on energy moving in the crossline direction than on energy moving in the inline direction. The SRME demultiple method requires many more receivers than are recorded, in order to model multiples that bounce at surface locations between streamers. In both cases, there are other issues as well. In particular, sources tend to be placed nearly at the center of the crossline range of the receiver spread, and should be extrapolated outward in the crossline direction, for reciprocity's sake. That should probably be done with some other algorithm, because the very narrow dip range that should exist between the two nearly central, only slightly separated sources is not likely to have information necessary to model sources far away. Put another way, PEFs can always interpolate, but can only extrapolate along straight lines.

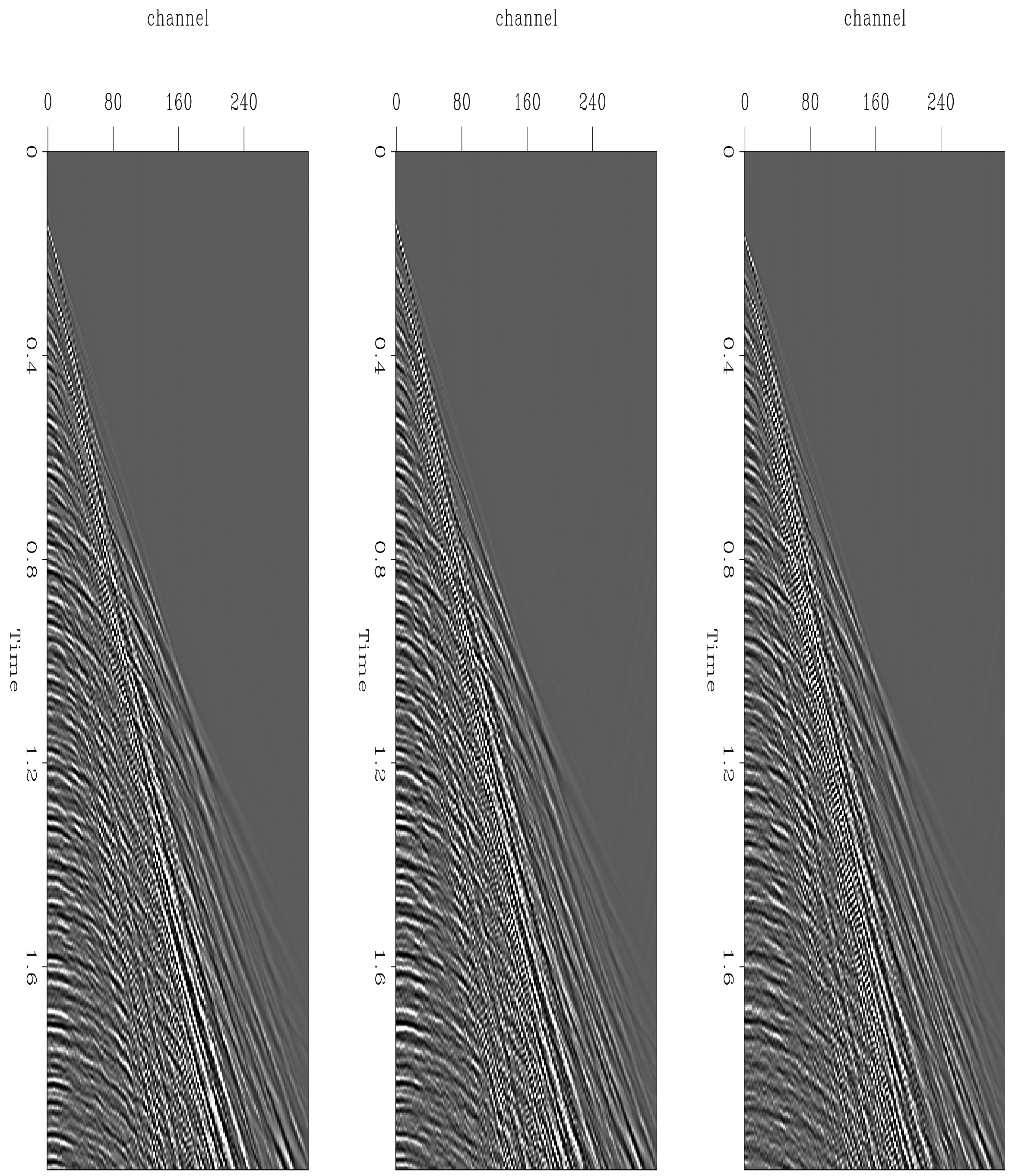


Figure 3.10: 3-D shot gather. This shot gather was among the half removed from the input data, and then reinterpolated. The interpolated version of this shot gather is shown in Figure 3.11. $\boxed{\text{sm-curt.in}}$ [ER]

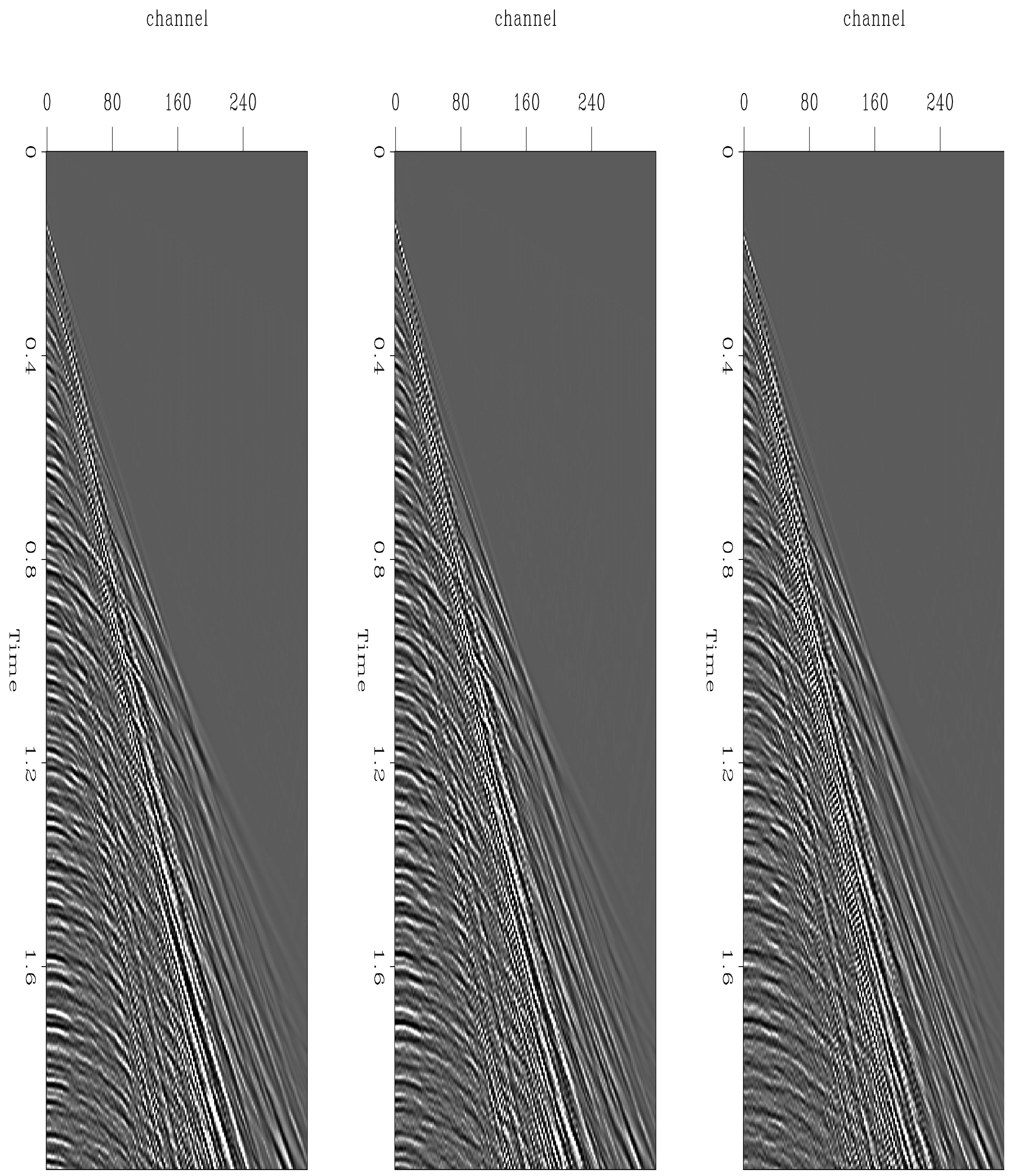


Figure 3.11: Recreated 3-D shot gather. The shot gather in Figure 3.10 was removed and interpolated from nearby shots to produce this shot gather. sm-curt.out [ER]

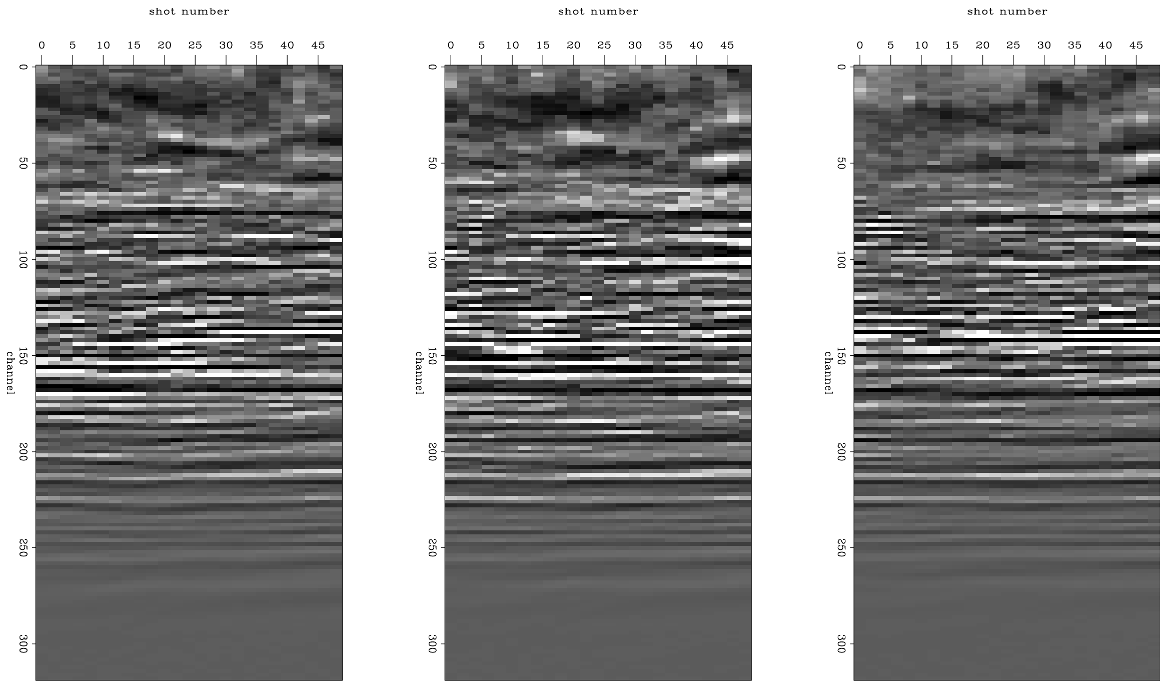


Figure 3.12: Time slice from a 3-D survey. This is a slice through a cube made up of shots from one of two sources, and receivers from three streamers. Each panel corresponds to one streamer. sm-curt.in.ts [ER]

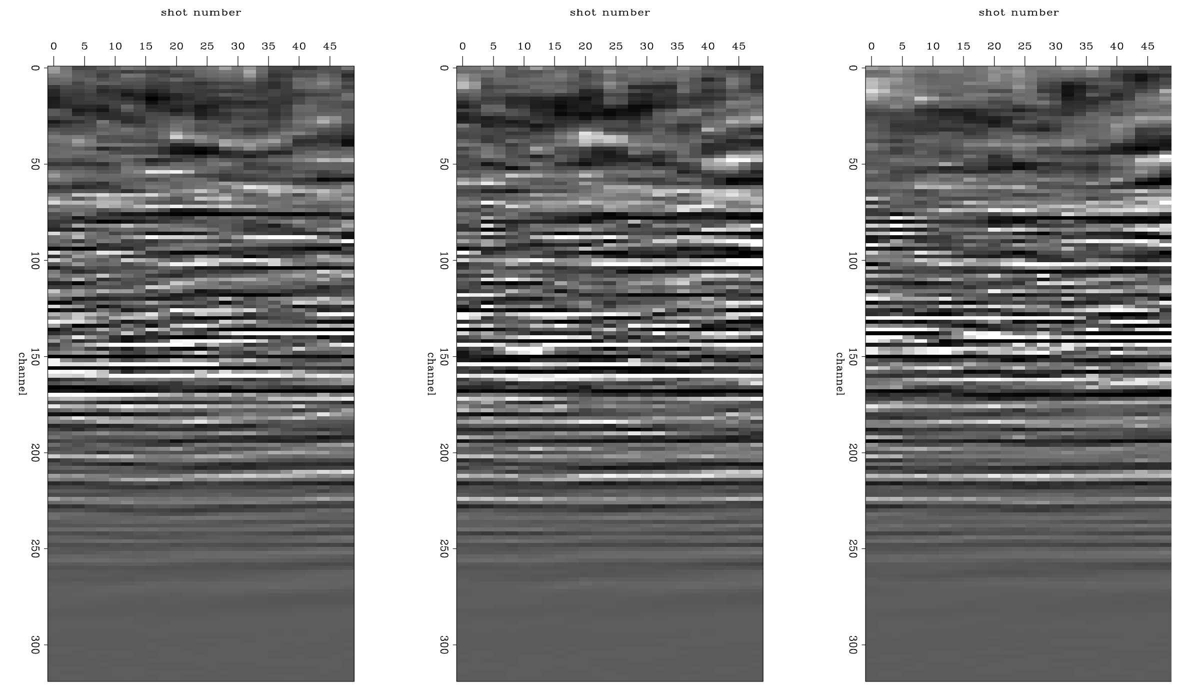


Figure 3.13: Recreated time slice from a 3-D survey. Every second shot from the data shown in Figure 3.12 was removed and then reinterpolated. This time slice is from the interpolated cube, at the same time sample as above. $\boxed{\text{sm-curt.out.ts}}$ [ER]

In the 3-D field data example above, there are only three streamers, with a narrow aperture, so it does not seem especially interesting to interpolate between them. Energy moving in the crossline direction probably becomes more important when there is a larger crossline aperture, as is the case with newer seismic acquisition boats which may tow a dozen or more cables. Unfortunately, no data from those boats was available. Figures 3.14 and 3.15 show a shot gather from the SEG-AEG 3-D salt model synthetic, as modeled in the case of Figure 3.14 and after interpolating streamers between the original streamers in the case of Figure 3.15. In this case, 20 or so shot gathers were used in the test. The interpolated data in this case was not modeled, so there are no difference panels, but by inspection the interpolated data seems reasonable. The front face of the two figures do not show exactly the same inline. Instead, Figure 3.14 shows a particular streamer, and Figure 3.15 shows a streamer that was interpolated next to it.

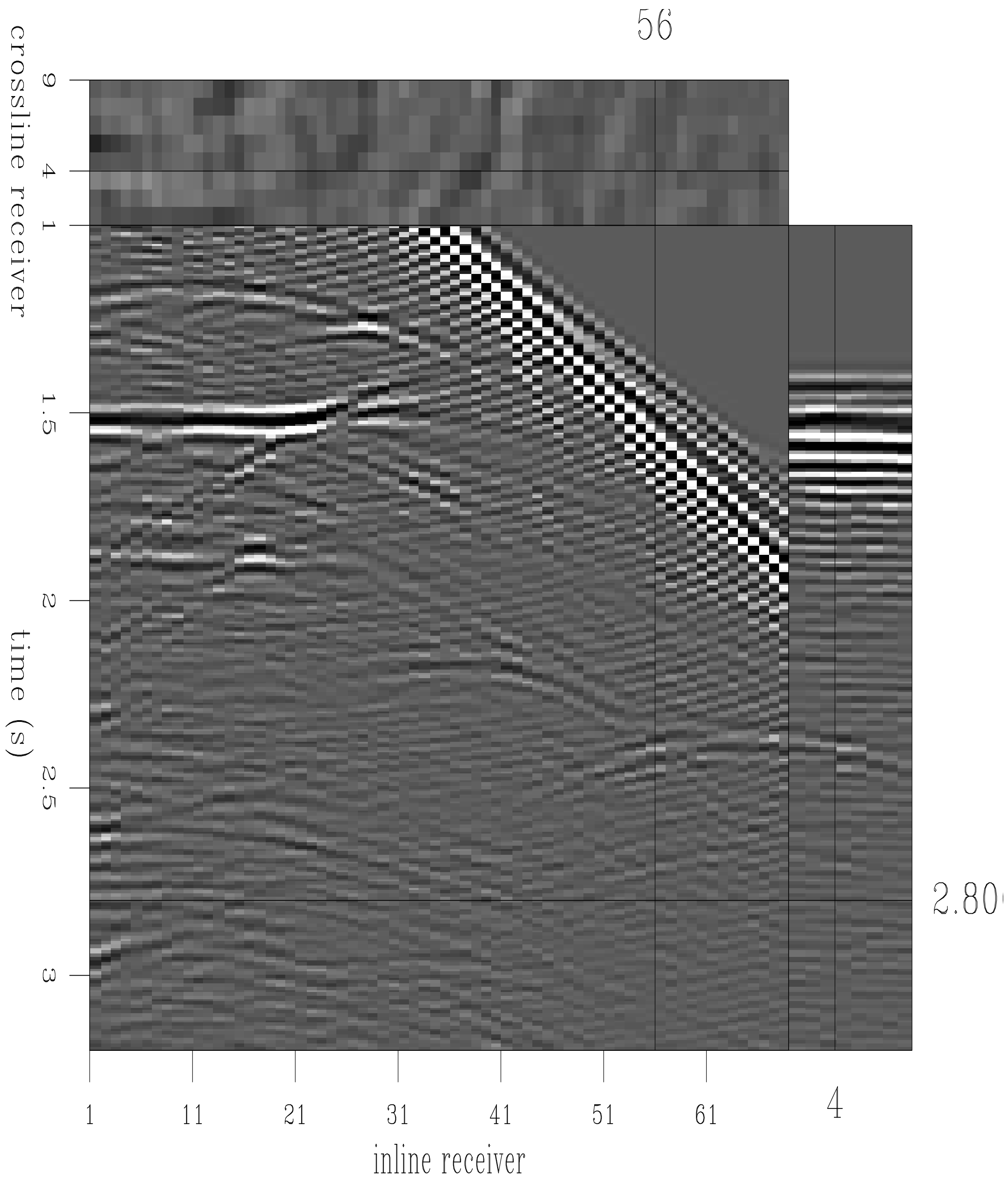


Figure 3.14: Closeup on a marine 3-D shot gather. Input to the crossline interpolation. sm-m33in [ER]

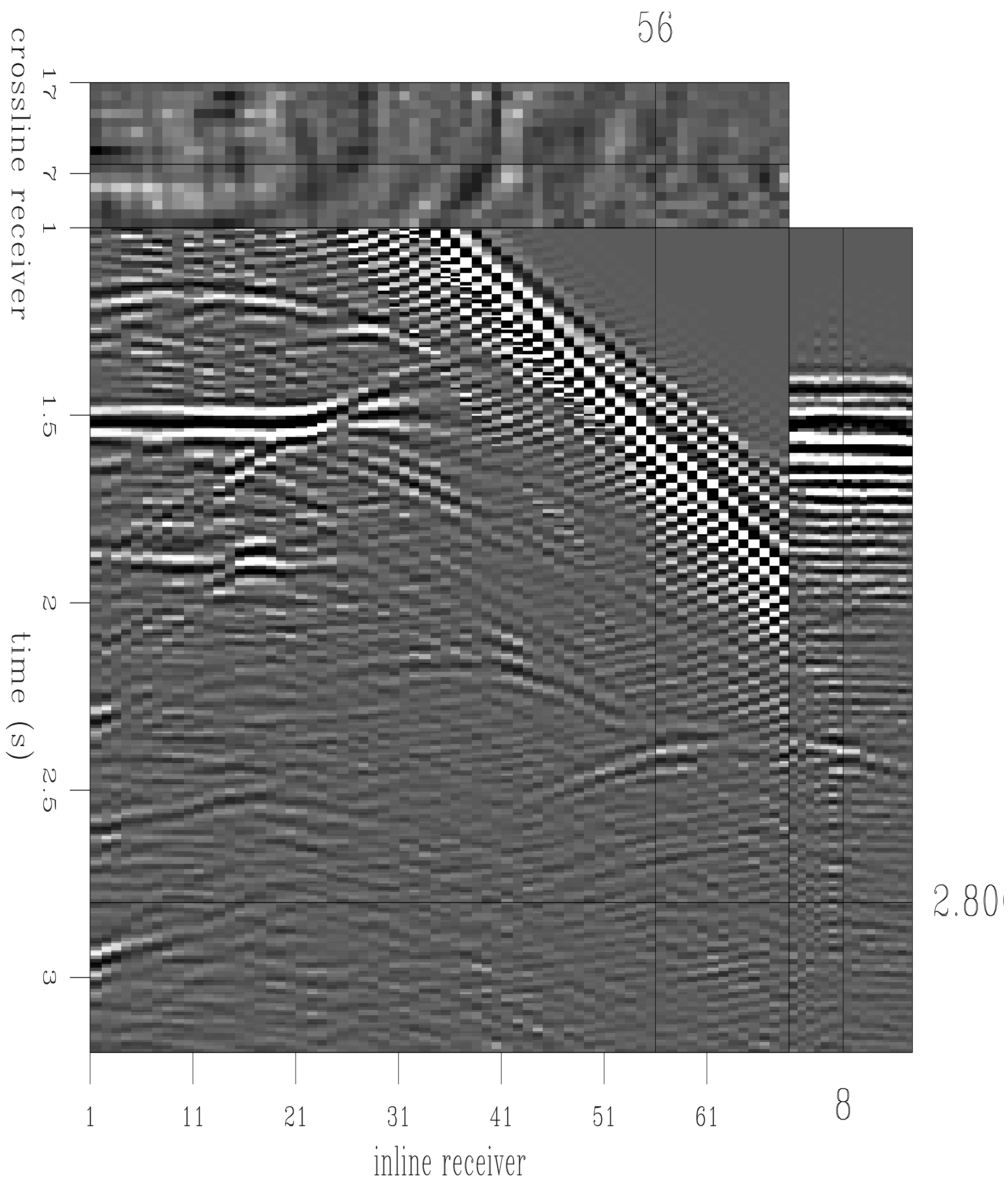


Figure 3.15: Closeup on a marine 3-D shot gather. Output of the crossline interpolation. sm-m33out [ER]

SMOOTHING AND DAMPING, ACCURACY AND CONVERGENCE

Estimating a set of adaptive PEFs tends to create an underdetermined problem, at least in those regions of the data (if any) where the filters are placed close together. The filter calculation then requires some damping equations or some method of controlling the null space in order to get satisfactory results. Several prescriptions exist. In one dimension, Claerbout (1997) preconditions the filter estimation with a smoother, and limits the number of iterations to control the null space. Shoepp and Margrave (1998) use an estimate of attenuation of the input trace to characterize the time-varying behavior of the input data. In two and three dimensions, Brown (1999) regularizes the filters with a Laplacian. Clapp (1999) preconditions with a smoother and also damps the preconditioned (roughened) model variable.

In this section we compare some strategies for controlling the null space of the underdetermined filter calculation problem. Convergence of the filter calculation by itself is not very informative. Filter calculation is the first step in a sequence of two linear optimization steps, and solving either step by any of several methods will guarantee convergence. There is no guarantee of convergence for the real quantity of interest, which is the difference between the true data (unknown in principle, but known for test cases) and the output of the second step, the interpolated data. Here we track that difference, as well as the residuals of the two individual least squares steps, varying the filter calculation strategy.

Tests of convergence and accuracy

A cube of seismic data with an interesting set of dips was used as a test case. Half the traces were replaced by zeroes and input to four variations on the adaptive interpolation scheme of this chapter.

These tests use filters estimated in radial micropatches. The density of filters was chosen to give a good interpolation result, and to give patches that create an overdetermined problem at longer offsets and later times where the patches are largest, and an underdetermined problem where the patches are smaller.

Figure 3.16 shows the rms amplitude of the data-fitting residuals for the filter calculation step versus iteration number. The four curves correspond to four variations on the filter calculation scheme. The curve labels are:

Smoothed means that the filters were calculated using the preconditioned optimization of equation (3.5), but that ϵ was zero, and so the damping (equation (3.6)) was effectively turned off.

Both is similar but with $\epsilon \neq 0$.

Damped used equations (3.1) and (3.6), except the damping equation really reads $\mathbf{0} \approx \mathbf{Ia}$, because there is no preconditioning, and so no change of model variables.

Neither was estimated using equation (3.1) alone, which is fine so long as the problem is overdetermined.

The curves all start off about the same, with the three curves associated with any sort of smoothing and/or damping flattening out earlier and higher than the fourth. The residual of the filter calculation step goes down fastest when there is no restriction on the filters (the *Neither* curve). This is not surprising. Unfortunately, a small filter estimation residual is not necessarily good. In this case, it just means the filters have too many degrees of freedom.

The rms amplitude of the residual for the missing data calculation step is shown in Figure 3.17. A pleasing thing about all of these curves is that they converge after a handful of iterations. The empty spaces in the data are small, so they do not take long to fill in. Of course, some of the curves do not converge to very good answers; that depends on the PEFs calculated in the first stage.

Figure 3.18 shows the real quantity of interest. Each curve is the norm of the difference between the interpolated data and the true data as a function of iteration in the second step of the interpolation. The difference increases at some point in most cases. In principle the true data are unknown, and there is no reason for the difference not to increase. Significantly, the misfit starts to go up about the time that the residual from Figure 3.17 bottoms out. The missing data residual bottoms out after about 6 iterations, and is guaranteed not to increase.

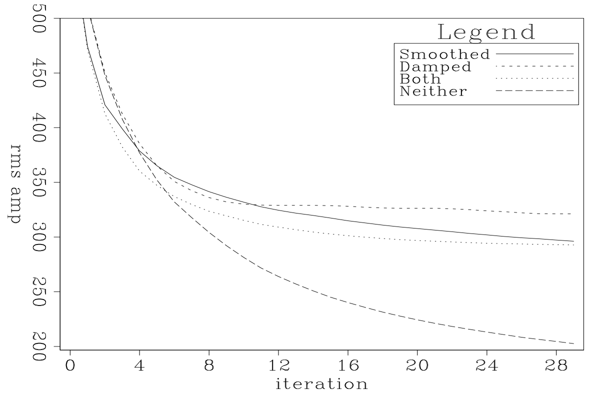


Figure 3.16: Filter calculation residual as a function of iteration. Curves represent different filter calculation schemes. sm-curves.nrp [CR]

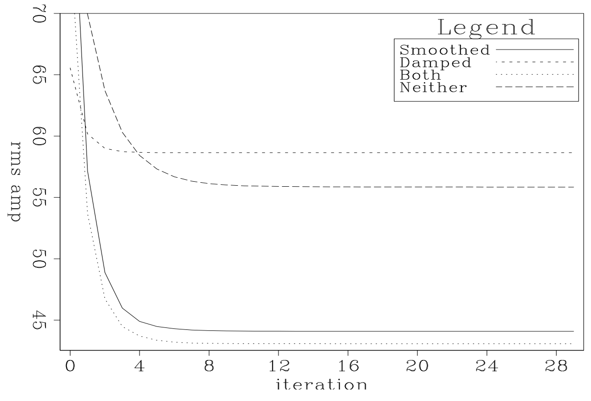


Figure 3.17: Missing data residual as a function of iteration. Curves represent different filter calculation schemes. sm-curves.nrd [CR]

The difference between the interpolated and true data begins to climb after the missing data problem has converged. Luckily, it does not climb very far.

The algorithm producing the curves labeled *Smoothed* and that producing the curves labeled *Both* are closely related. The former is just a special case of the latter, with $\epsilon = 0$. With $\epsilon = 0$, there is no damping to control the null space of the filter calculation step; control comes from not running too many iterations. Figure 3.19 shows how, with smoothing but no damping, the number of iterations spent calculating filters affects the misfit between the final interpolation result and the true data. The vertical axis is the sum of squares in the difference. The horizontal axis is the number of iterations spent filling in missing data. The five curves correspond to different numbers of iterations spent calculating filters. For example, the curve labeled *niter=30* shows the rms amplitude of the difference between interpolated and true data as a function of missing-data iteration, for the set of PEFs calculated after 30 filter-calculation iterations. As in Figure 3.18, after a sufficient number of missing data iterations, the difference between interpolated and true data begins to increase. Figure 3.19 shows that the size of the misfit also goes up if too many iterations are spent on the filter calculation. After spending 80 iterations calculating PEFs, the smallest possible result is larger than if only 30 iterations were spent on the PEFs.

Figures 3.19, 3.20, and 3.21 point to the utility of damping. Figure 3.20 shows the same curves as are shown in Figure 3.19, except that a small amount of damping is applied, with $\epsilon = 2$. That value was chosen by trial and error. Figure 3.21 is also thematically similar to Figure 3.19, except that the number of filter calculation iterations is held constant (at 30) and the value of ϵ is incremented.

Figure 3.20 shows that choosing a reasonable value of ϵ helps to reduce the sensitivity of the final result to the number of iterations in the filter calculation step. Figure 3.19 shows that increasing the number of iterations from 30 to 80 with no damping causes a noticeable increase in the misfit, While Figure 3.20 shows that the same change in number of iterations with damping has a smaller effect.

Figure 3.21 shows that while there is a choice between choosing the number of iterations and choosing ϵ, there appears to be an advantage to choosing ϵ. Changing the number of iterations by a factor of about two (again, from 30 to 80), with no damping, showed a significant effect in Figure 3.19. Changing the value of ϵ by a larger factor , while keeping iterations constant, produces a very small effect in Figure 3.21.

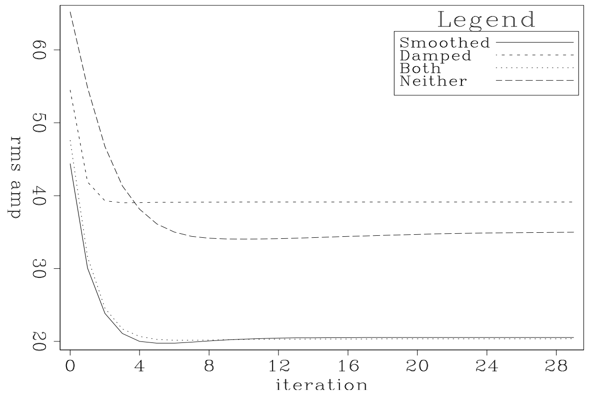


Figure 3.18: Norm of the misfit between the true data and the interpolated data. Horizontal axis displays number of iterations in the missing data calculation step. Curves represent different filter calculation schemes. sm-curves.nm [CR]

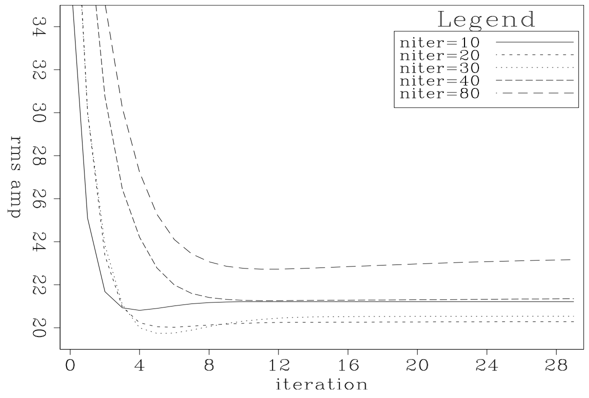


Figure 3.19: Norm of the misfit between the true data and the interpolated data. Horizontal axis displays number of iterations in the missing data calculation step. Curves represent different numbers of iterations in the filter calculation step. sm-curves.nm.filtniter [CR]

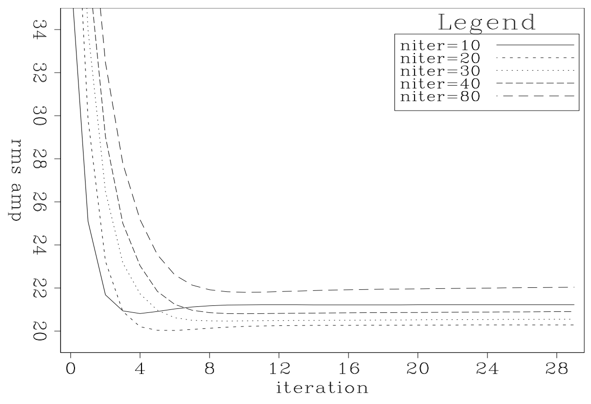


Figure 3.20: Norm of the misfit between true data and interpolated data. Curves represent different numbers of iterations in the filter calculation step. sm-curves.nm.slightdamp.fn [CR]

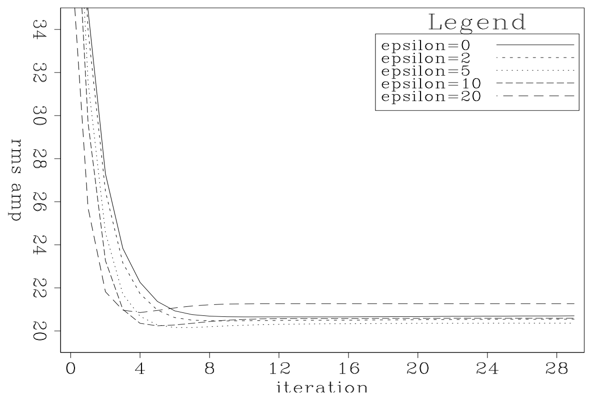


Figure 3.21: Norm of the misfit between true data and interpolated data. Curves represent different amounts of damping. sm-curves.nm.muchdamp [CR]

Chapter 4

Noisy data and land data

In previous chapters, all of the interpolation examples were run on marine data. Interpolating marine data is a good application because of the trouble with aliased multiples. Marine data is also relatively easy to interpolate, because it typically has little noise and regular geometry. Land data is much more difficult. Land data often has discontinuous arrangements of shots and receivers, where the survey is forced to work around and over surface features. Land data also tends to be noisier. Noise and statics make it difficult to predict a seismic trace from its neighbors, so it is more difficult to interpolate. Nonetheless, because of the expense and effort of acquiring land data, it is worthwhile to try interpolating it.

Statics and irregular geophone placement can significantly reduce the window size that can be realistically assumed to be stationary. That makes it attractive to use tiny micropatches rather than patches large enough to calculate a PEF independently. However, it also suggests that PEFs should not be gradually varying, because the dips of events might change very quickly where there are statics.

In this chapter, I apply the method of the previous chapter to interpolate noisy marine data and land data with noise and irregular geometry. The problems are more difficult than earlier examples, but the results are still encouraging.

NOISY DATA

On data with complicated dips, nonstationary filters work better than stationary filters. Noise is a problem related to the problem of complicated dips. Regone (1998) points out that in many cases, noise which appears to be random is completely repeatable, though incoherent. Energy may be scattered by near surface features back to receivers at many different angles and amplitudes simultaneously. This superposition of waves at all angles may produce what looks like random noise, but as long as the shot and receiver positions do not change, the same scattered energy will be produced by successive shots, and the apparently random noise will be recreated.

That may mean that apparently random noise should be interpolated, because it is just many superimposed plane waves. On the other hand, maybe genuinely random noise will be interpolated if we allow our filters too much freedom by making them large, since the number of plane waves predicted by a PEF depends on its size. In the end the important thing is to interpolate all of the coherent events that exist in the data, and to not create any new coherent events from random noise.

We do not attempt any distinction between signal and coherent noise. Coherent noise should be interpolated. In the case of multiples, this is often exactly the point of the interpolation. Interpolating the multiples dealiases them and makes them easier to suppress.

Noisy data examples: confirmation of Abma's result

Abma (1995) shows that (t,x) PEFs are less likely to create spurious events in the presence of noise than (f,x) PEFs, because calculating a filter at each frequency is the time domain equivalent of calculating a filter that is long on the time axis. This effective time length of the (f,x) filter gives it sufficient freedom to predict ostensibly random noise.

As an example, the data used in Figures 3.7 through 3.9 is used again, except noise is added, and the PEFs used for interpolation are only two dimensional. The other examples in this thesis have three or more dimensions, but the (f,x) interpolation that I happened to have available for comparison uses two dimensions in the input, so for this test the interpolation is

2-D in both domains. My experience has been that (t,x,y) interpolation nearly always gives better results than just (t,x); presumably the same is true for (f,x,y) and (f,x). So in both domains the results could be better, but they should still make a meaningful comparison.

The data as it was recorded has little noise. On noise-free data, time- and frequency-domain implementations both produce fine results. Figure 4.1 shows closeups on results of interpolating noise-free data using (t,x) interpolation on the left, and (f,x) on the right. The data were subsampled on the x-axis simulating receiver gathers with alternate shots missing, and then reinterpolated. The time-domain interpolation uses the PEF smoothing scheme described in chapter 3. The panels in Figure 4.1 are not identical, but either is a good result.

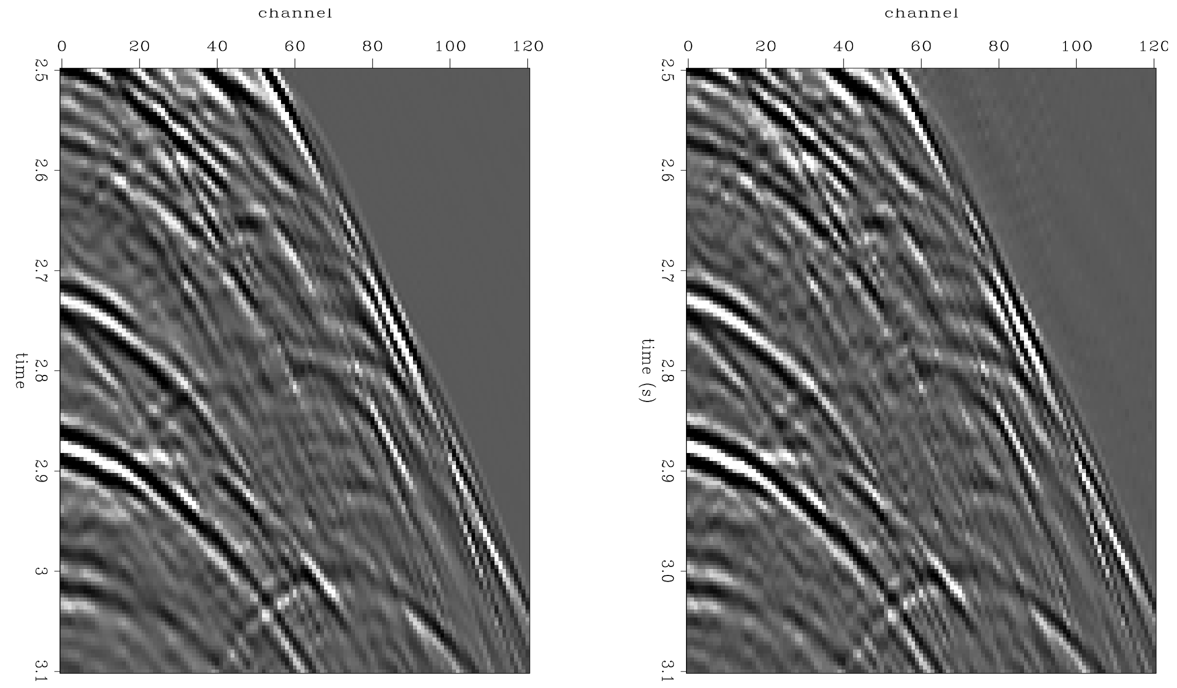


Figure 4.1: Time-domain and frequency-domain interpolation results using noise-free input. The panels are not identical, but either is a good result. lnd-102.smooth.both.out.closeup [NR]

With increasing amounts of noise, the difference between time-domain and frequency-domain interpolation becomes significant. Random noise was added to the same data and interpolated again, to produce the results in Figure 4.2. Again, time domain is on the left, frequency on the right. Where either domain did a satisfactory job on the noise-free data, the time domain produced a significantly better result on the noisy data. Many of the strong events are better interpolated in the time domain. Also, as a confirmation of Abma's observation, in the time domain there is almost no interpolation where there are no coherent seismic events, such as above the seafloor reflection where the only energy is the added noise. Where there is nothing but white noise to interpolate, nothing happens, because PEFs are just whiteners. In the frequency domain, the noise is interpolated. The same amount and distribution of noise is added in both cases, but the frequency-domain result looks noisier because more noise is interpolated into the new traces.

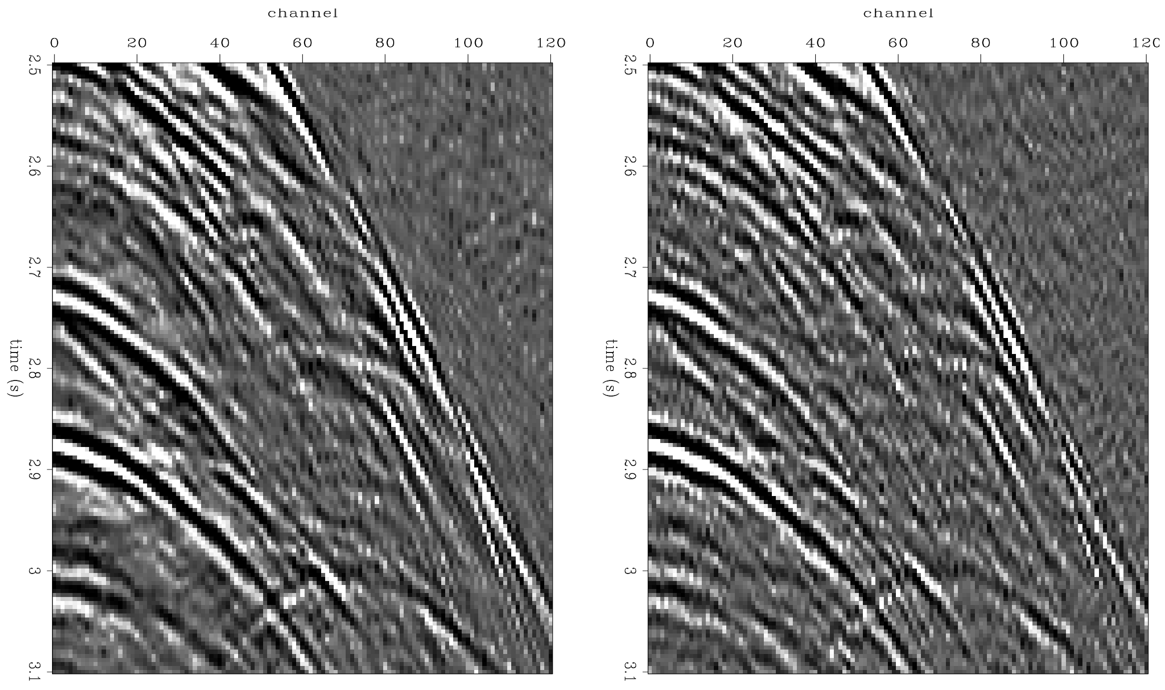


Figure 4.2: Time-domain and frequency-domain interpolation results using noisy input. Several of the strong events are interpolated better in the time-domain. In the region above the first breaks, where there is only noise, the time-domain puts less energy in the interpolated traces. $\boxed{\text{lnd-102.sn.both.cp}}$ [NR]

The differences between the two results are easier to see in Figure 4.3. The top half of this figure shows the same two panels as Figure 4.2, with the known half of the traces removed. The top left shows just the energy interpolated into the missing traces in the time domain. The top right shows just the energy interpolated into the missing traces in the frequency domain. The time domain method adds much less energy above the first breaks, where there is nothing to interpolate, though both methods add a slight artifact above the first breaks and parallel to them. More importantly, the time domain method does a better job at interpolating dips pointed both out towards increasing offset and in towards zero offset. For example, the event with its apex at about channel 70, at time 3.0 s, is better interpolated in the time domain. The near offsets and the steep events around the first breaks also look better in the time domain result.

The bottom half of Figure 4.3 shows difference panels between the interpolation result and the original traces which were thrown out. The left is time domain, the right is frequency. Again, the known half of the traces are not shown, since they naturally have a difference of zero. Both differences have noticeable energy, but the time domain difference is mostly incoherent. The frequency domain difference shows some definite coherent events. In particular the near offset events, the first break events, and the mid-offset event at 3.0 s all show up in the frequency domain difference panel.

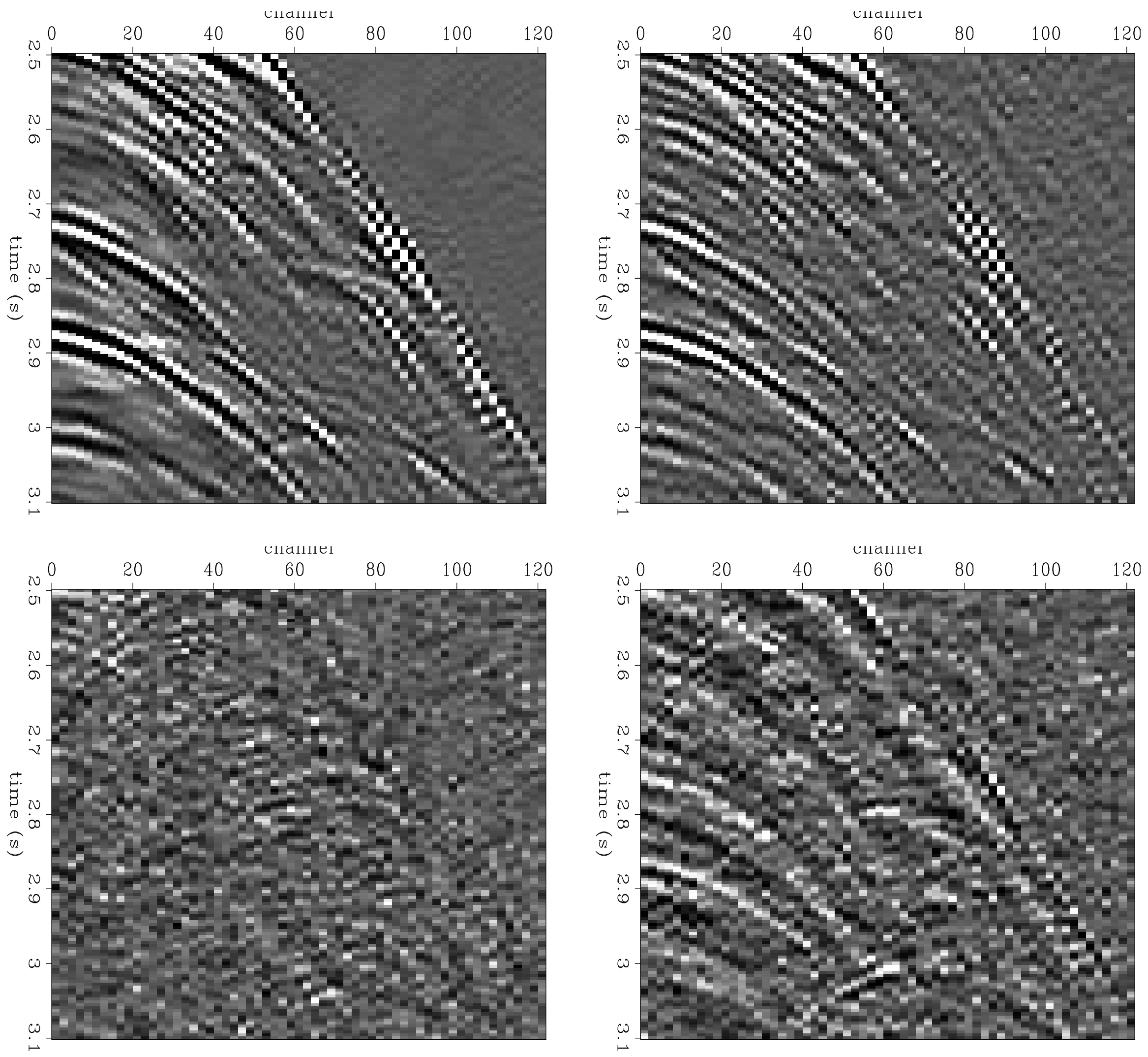


Figure 4.3: Interpolated traces and difference panels. The top left panel shows just the energy interpolated into the missing traces in the time domain. The top right shows just the energy interpolated into the missing traces in the frequency domain. The bottom panels show the differences between the interpolated traces in the top panels and the real traces. The time domain shows less noise energy interpolated above the first breaks, and less coherent event energy in the differences. $\boxed{\text{lnd-102.sn.all.cpd}}$ [NR]

LAND DATA

If you can interpolate noisy data, then it is natural to think about interpolating land data. Land data can be difficult and time-consuming to acquire, depending on the survey area, so naturally it is attractive to acquire less of it in the field. Unfortunately, land data is also much harder to interpolate, for a number of reasons. First, land geometries tend not to be well-behaved in the way that marine geometries are. Marine cables are all the same length, and are dragged through the water while recording, so that there is a restoring force working against sharp bends in the receiver lines, smoothing them out. Land geophones are static; they do not have anything smoothing out the kinks in the receiver lines. Sharp bends are common, as are receiver lines of different lengths, and large static shifts. Taken together, these things mean that land data does not have the same kind of lateral coherence and predictable acquisition that makes marine data ideal for interpolation. Nonetheless, because of the expense and effort of acquiring land data, it is worthwhile to try interpolating it.

Arabian data examples

The first land data examples are from a data set from Saudi Arabia, which has a mostly regular geometry, and is very clean for land data. The first is made up of all the positive offsets from a set of CMP gathers, shown in Figure 4.4. This data is divisible by inspection into two wedge-shaped regions. The nearer offset wedge is the noisier of the two; it contains the surface waves. The farther offset wedge is outside the cone of surface waves, and appears very clean. In the coherent wedge towards farther offsets, events are predictable both in offset and in midpoint. In the near offset wedge, events are predictable only over much shorter distances in offset, and are unpredictable in midpoint. Figure 4.5 shows portions of the original and interpolated data cubes. The data in Figure 4.5 is subsampled so that only missing traces are shown. The left panel shows some of the traces which were removed from the original data to make the input cube. The center panel shows those same traces taken from the interpolation result, and the right panel shows the difference. There are large differences, though they are mostly incoherent, and mostly restricted to the noisy inner offsets. Most of the somewhat coherent noise events visible in the original data are interpolated, though not fully. For instance, there are a number of nearly horizontal events near the bottom of the input traces, in the inner half of the offsets, each about five traces wide in offset, and one trace wide in midpoint. These events are visible in the output, but their amplitudes are lower than in the input. Over the whole input, about 72% of the variance is predicted.

The next figures zoom in on the near offset, short time portion of the data, where there is the strongest curvature. Figures 4.6 and 4.7 show input and output cubes where the data are just the near offsets and short times of some CMP gathers. In this case offsets are interpolated instead of shots; the data are CMP gathers, and the shot and receiver spacings are equal, so the fold is equal to half the number of receivers. Here we just double the fold. We do not begin by subsampling the known data, so there are not any obvious input/output comparisons. It is just presented as an exercise in interpolating strongly curved, somewhat aliased events. At any rate, the result is convincing, if subjective.

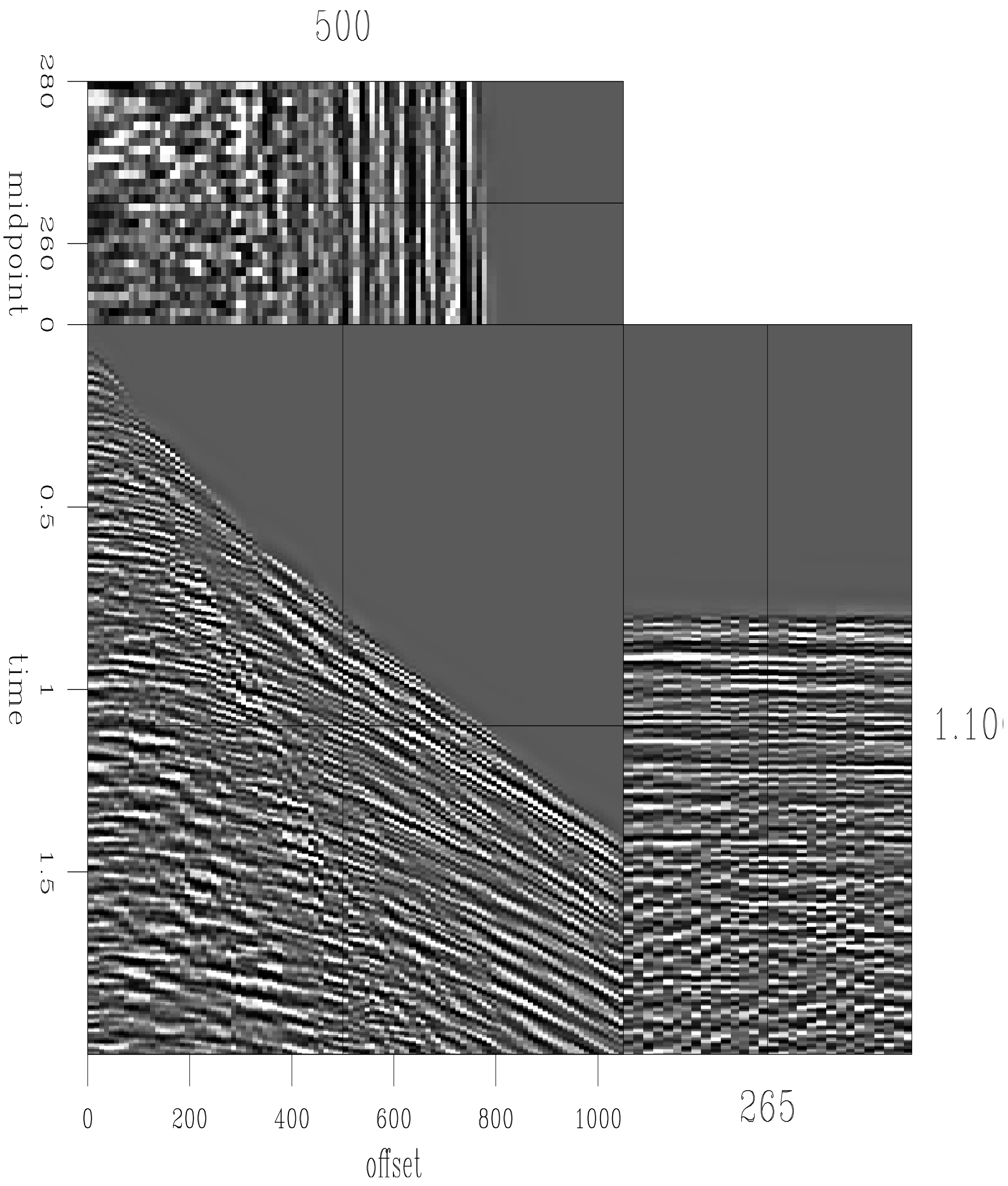


Figure 4.4: Land CMP gathers. Shot gathers were removed from the cube and interpolated back to produce Figure 4.5. lnd-sgyWin2.in [NR]

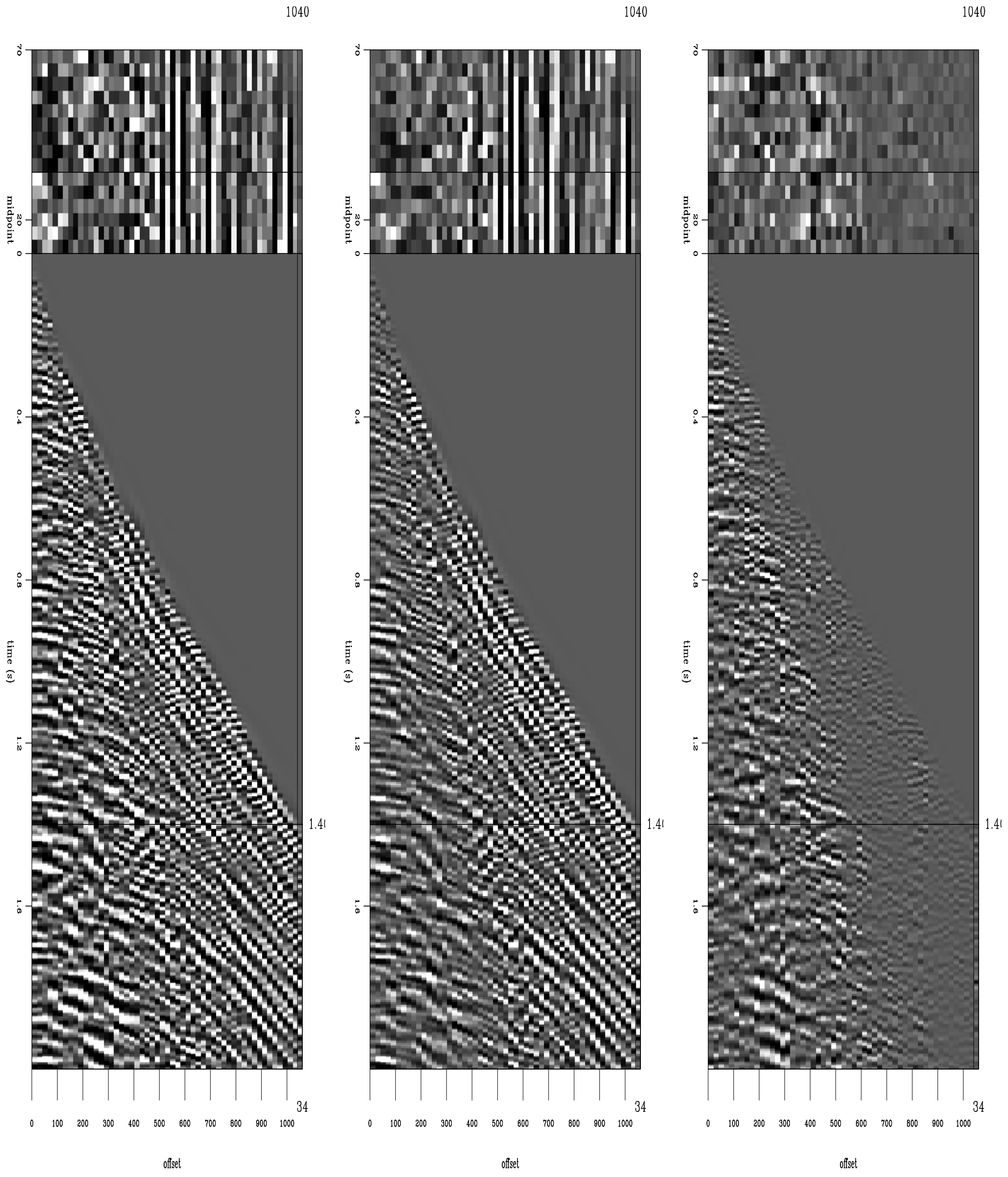


Figure 4.5: Interpolated traces. The left panel shows half of the traces which were removed from the original data (Figure 4.4) to make the input cube. The center panel shows those same traces taken from the interpolation result, and the right panel shows the difference. lnd-sgyWin2.out [NR]

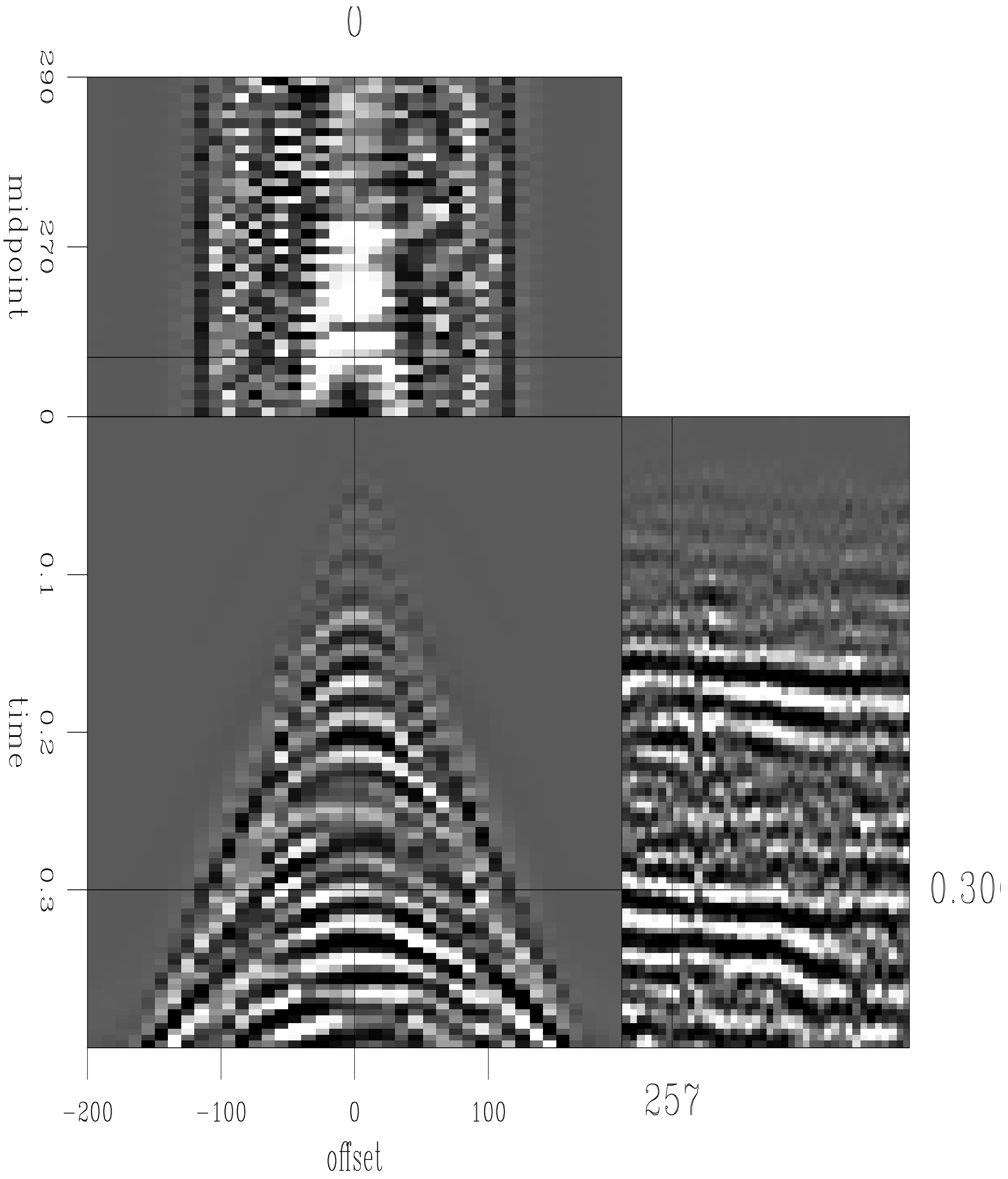


Figure 4.6: Land CMP gathers. This cube was interpolated on the offset axis to produce Figure 4.7. Events curve through a wide range of dips over just a few traces, and are somewhat aliased on the flanks. lnd-sgyWin1.in [NR]

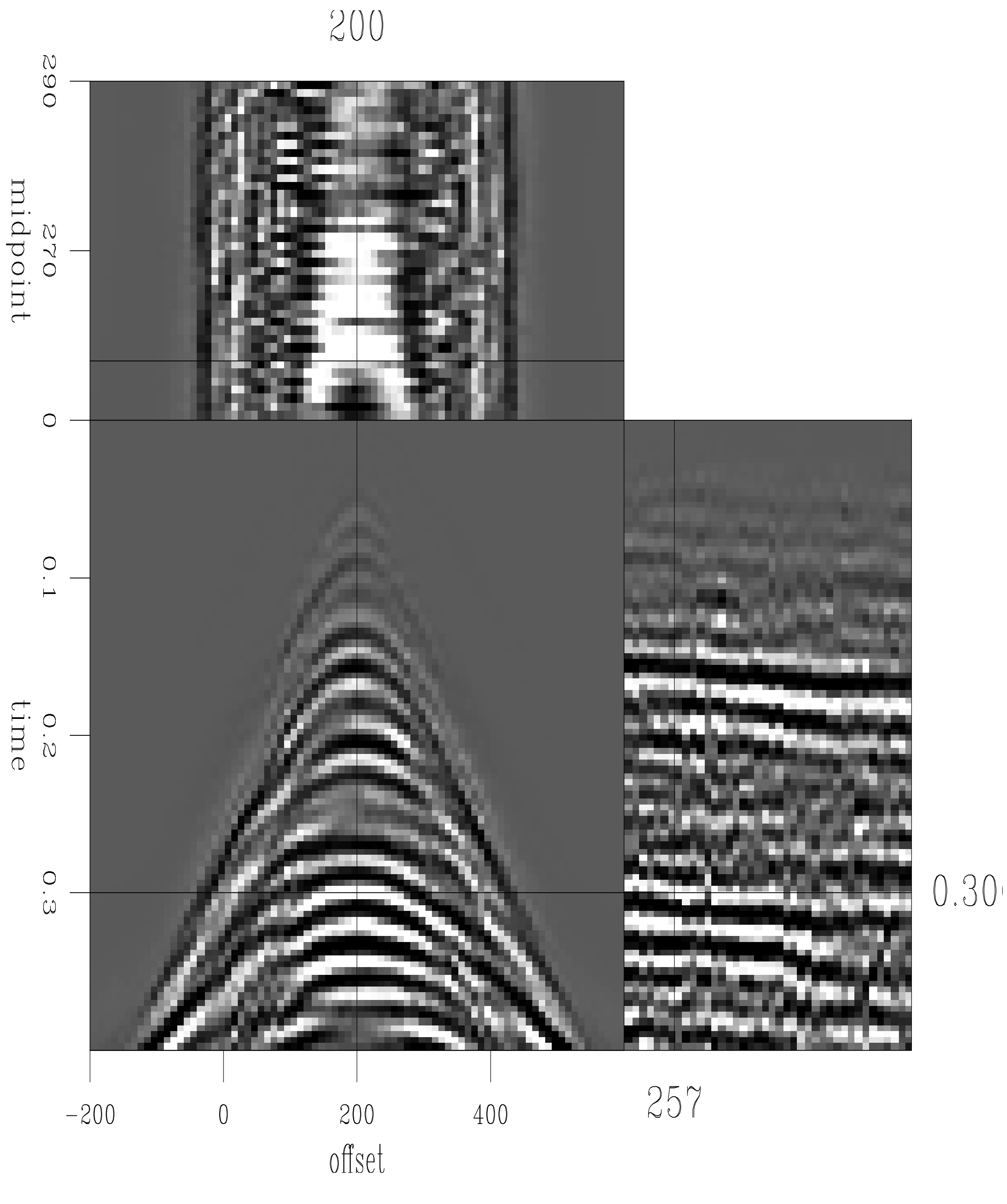


Figure 4.7: Land CMP gathers. This is the result of interpolating the data in Figure 4.6. The data have strong curvature over just a few traces, but are interpolated well. lnd-sgyWin1.out [NR]

Arizona data example

The last example uses an environmental data set acquired near a copper mine in Arizona. The data was acquired on rough terrain. There are large statics throughout the data, and the geometry is much less predictable than in a marine case, or in the Saudi data.

In this data set, receivers were zeroed in a checkerboard pattern and reinterpolated, similar to several earlier experiments on marine data. This survey used a roll switch to increment the active receiver range with each shot, so the geometry is similar to a marine survey in the sense that the offset range is the same on each shot gather. This subsampling can thus simulate half as many receiver locations or half as many shots with the input taken to be receiver gathers rather than shot gathers. Unlike the marine case, the ground surface where this data was recorded is rough, so that the geometry does not have the regularity of earlier examples. The rough surface and the relatively unpredictable geometry give rise to statics and scattered noise in the data, which make it difficult to predict. As a result, the interpolation is not perfect, with 75% of the variance predicted. Nevertheless, it is good, given the input.

Figure 4.8 shows a shot gather from the original input data. Figure 4.9 shows the same view of the output, after subsampling and interpolating. The most obvious difference is at the very closest offsets. Here events dip so steeply that they are beyond the angular range of the PEFs used to interpolate, so nothing is filled in. A more subtle difference is the static shifts between original traces and their interpolated counterparts. Interpolated traces may have the correct waveform, but not the correct static shift relative to surrounding traces. Figure 4.9 is a two-frame movie in the electronic version of this document. Flipping between the two frames you can see that several channels have statics in the original data which are not present in the interpolated trace. The possibility of introducing non-surface-consistent statics in this way is an interesting, unexplored topic.

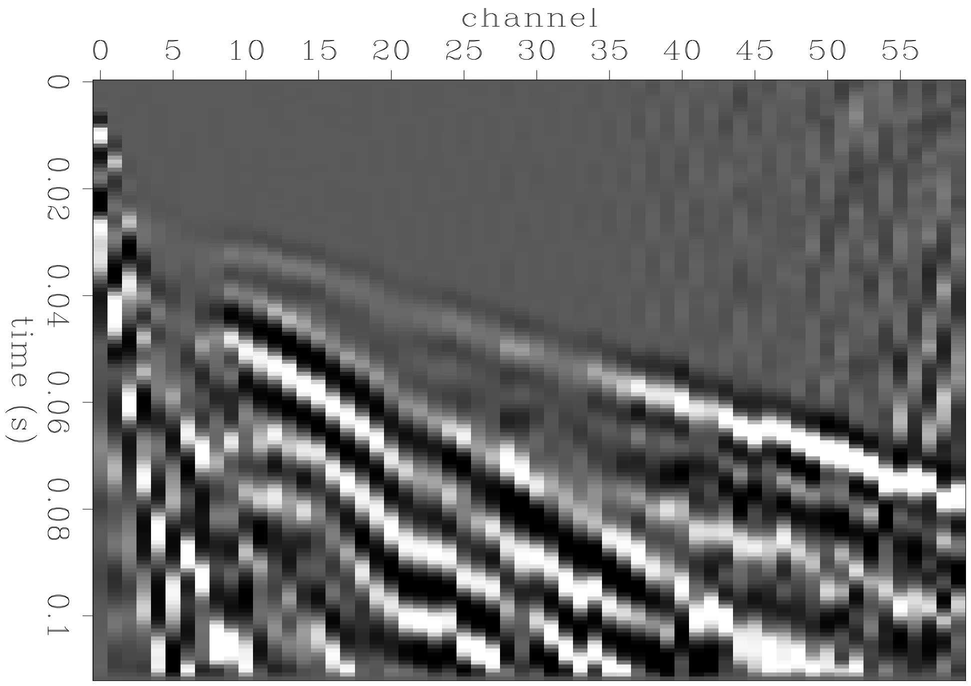


Figure 4.8: Land shot gather. This is one of the input shots that was subsampled and interpolated to make Figure 4.9. This data is noisy and has statics, making it difficult to interpolate. lnd-minetestin [NR]

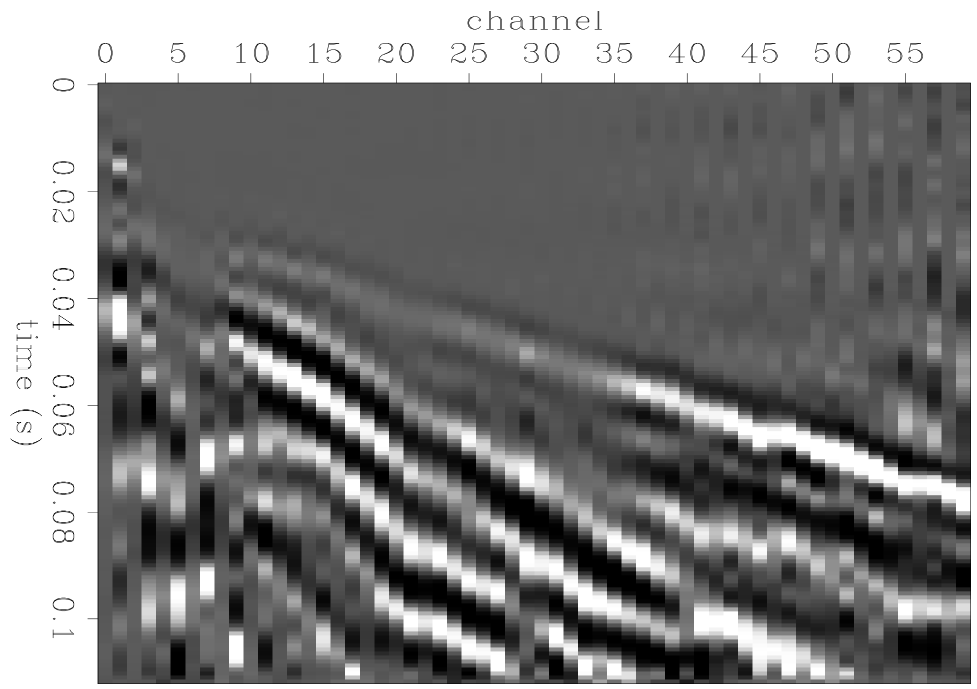


Figure 4.9: Land shot gather. This is the result of subsampling and interpolating the data in Figure 4.8. In the electronic version of this thesis, this figure is a two-frame movie comparing the original data and the interpolation result. lnd-minetestout [NR]

Chapter 5

Summary

In this thesis, I present a method of interpolating aliased seismic data using time- and space-domain prediction-error filters (PEFs). The method works in two steps. First, PEFs are calculated from the aliased input data. The spectrum of a PEF is the inverse of the input data spectrum. If the data are aliased, then the PEF also has an aliased spectrum. The aliased data spectrum contained in a PEF can be "unwrapped" easily by scaling and rescaling the PEF's axes. This dealiases the PEF spectrum. The second step is then to calculate new data, using the PEFs and the existing data. Where in the first step the PEF takes on the inverse of the aliased data spectrum, now the data takes on the inverse of the dealiased PEF spectrum. The result is a dealiased data set.

The theory for PEFs assumes that the data are wide-sense stationary. Seismic data are not stationary. The dips of events change in time and space. How to deal with nonstationarity is one of the most important details in interpolation. This thesis describes two strategies. The first strategy, well known in various filtering applications, is called patching. The input data are simply divided into patches (also known as analysis windows, design gates, etc.), and assumed to be stationary within the patch. Each patch gets a single PEF, and is interpolated as an independent problem. At the end, the patches are reassembled to form the interpolated data volume, with some overlap and normalization to hide the patch boundaries.

The second (and new) approach treats the data as a set of gradually varying dips rather than as independent dips. Seismic events are curvy in prestack data, and so have gradually changing dips. Instead of dividing the data into patches, we treat the data as a single nonstationary unit. We estimate many PEFs, as many as one per data sample, and the filter calculation problem becomes underdetermined. To control the null space we add a penalty function, which says that different PEFs which are calculated from adjacent portions of the data should be approximately equal. More specifically, we add directionality so that PEFs are approximately equal along radial lines extending from zero time and offset in CMP gathers.

On data with complicated dips, nonstationary filters work noticeably better than filters calculated in patches. Nonstationary filters also work noticeably better on data with noise and statics. Land data is an interesting interpolation problem because it so expensive to acquire. It can be much more difficult than marine data, because of the differences in acquisition geometry, and the relatively large amount of noise that tends to be in the data. Nonetheless, the interpolation results are promising.

Bibliography

Abma, R., 1995, Least-squares separation of signal and noise with multidimension al filters: Ph.D. thesis, Stanford University.

Beasley, C., and Mobley, E., 1998, Spatial dealiasing of 3-d dmo: The Leading Edge, **17**, no. 11, 1590, 1592–1594.

Berkhout, A. J., and Verschuur, D. J., 1997, Estimation of multiple scattering by iterative inversion, part i: Theoretical considerations: Geophysics, **62**, no. 05, 1586–1595.

Berryhill, J. R., and Kim, Y. C., 1986, Deep-water peglegs and multiples - emulation and suppression: Geophysics, **51**, no. 12, 2177–2184.

Brown, M., Clapp, R. G., and Marfurt, K., 1999, Predictive signal/noise separation of groundroll-contaminated data: SEP–**102**, 111–128.

Cabrera, J. J., and Levy, S., 1984, Stable plane-wave decomposition and spherical-wave reconstruction - applications to converted s-mode separation and trace interpolation: Geophysics, **49**, no. 11, 1915–1932.

Chemingui, N., and Biondi, B., 1996, Handling the irregular geometry in wide-azimuth surveys: 66th Ann. Internat. Meeting, Soc. Expl. Geophys., Expanded Abstracts, 32–35.

Claerbout, J. F., and Nichols, D., 1991, Interpolation beyond aliasing by (tau,x)-domain pefs: Interpolation beyond aliasing by (tau,x)-domain pefs:, 53rd Mtg. Eur. Assoc. Expl Geophys., Abstracts, 2–3.

Claerbout, J. F., 1992a, Anti aliasing: SEP–**73**, 371–390.

Claerbout, J. F., 1992b, Earth Soundings Analysis: Processing Versus Inversion: Blackwell Scientific Publications.

Claerbout, J. F., 1997, Geophysical exploration by example: Environmental soundings image enhancement: Stanford Exploration Project.

Claerbout, J. F., 1998, Multi-dimensional recursive filtering via the helix: Geophysics, **63**, no. 5, 1532–1541.

Clapp, R. G., and Brown, M., 1999, Applying sep's latest tricks to the multiple suppression problem: SEP–**102**, 91–100.

Hugonnet, P., and Canadas, G., 1997, Regridding of irregular data using 3-d radon decompositions: Regridding of irregular data using 3-d radon decompositions:, 67th Annual Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1111–1114.

Jain, A. K., 1989, Fundamentals of digital image processing: Prentice Hall.

Jakubowicz, H., 1994, Wavefield reconstruction: Wavefield reconstruction:, 64th Annual Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1557–1560.

Leon-Garcia, A., 1994, Probability and random processes for electrical engineering: Addison Wesley.

Lumley, D. E., Claerbout, J. E., and Bevc, D., 1994, Anti-aliased kirchhoff migration: Anti-aliased kirchhoff migration:, 56th Mtg. Eur. Assoc. Expl Geophys., Extended Abstracts, Session:H027.

Manin, M., and Spitz, S., 1995, Wavefield de-aliasing for acquisition configurations leading to coarse sampling: Wavefield de-aliasing for acquisition configurations leading to coarse sampling:, 65th Annual Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 930–932.

Marfurt, K. J., Schneider, R. V., and Mueller, M. C., 1996, Pitfalls of using conventional and discrete radon transforms on poorly sample data: Geophysics, **61**, no. 05, 1467–1482.

Nichols, D., 1992, Dealiasing band limited data using a spectral continuity constraint: Dealiasing band limited data using a spectral continuity constraint:, 62nd Annual Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1183–1186.

Novotny, M., 1990, Trace interpolation by slant-stack migration: Geophys. Prosp., **38**, no. 08, 833–852.

Regone, C., 1998, Suppression of coherent noise in 3-d seismology: The Leading Edge, **17**, no. 11, 1584–1589.

Ronen, J., 1987, Wave equation trace interpolation: Geophysics, **52**, no. 07, 973–984.

Ronen, S., 1990, Spatial dealiasing of 3-d data: Spatial dealiasing of 3-d data:, 60th Annual Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1637–1640.

Schoepp, A. R., and Margrave, G. F., 1998, Improving seismic resolution with nonstationary deconvolution: Improving seismic resolution with nonstationary deconvolution:, 68th Annual Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1096–1099.

Spitz, S., 1990, 3-d seismic interpolation in the f-xy domain: 3-d seismic interpolation in the f-xy domain:, 60th Annual Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1641–1643.

Spitz, S., 1991, Seismic trace interpolation in the f-x domain: Geophysics, **56**, no. 06, 785–794.

Thorson, J. R., 1984, Velocity stack and slant stack inversion methods: Ph.D. thesis, Stanford University.

Vermeer, G. J. O., 1990, Seismic wavefield sampling: Soc. Expl. Geophys.

Verschuur, D. J., and Berkhout, A. J., 1997, Estimation of multiple scattering by iterative inversion, part ii: Practical aspects and examples: Geophysics, **62**, no. 05, 1596–1611.