CWP-196
December 1995

Seismic Data Compression

Tong Chen

— Master's Thesis —
Mathematical and Computer Sciences

Center for Wave Phenomena
Colorado School of Mines
Golden, Colorado 80401
303/273-3557

ABSTRACT

The compression of seismic data provides potential cost reduction in both storing and transmitting the huge amount of data acquired in exploration seismology, yet there have been limited studies on this issue. Here, I study seismic data compression from several different aspects. Specifically, I compare several algorithms that use different transformations, in terms of the amount of compression achievable under a specified amount of compression error. For a given error, the cosine-based transforms are able to achieve slightly more compression than the standard wavelet transform. As to the trade-off between the amount of compression and the amount of error in compression, I introduce the compression ratio as a function of the compression error. This function is then used to explain why the compression error grows more rapidly for unstacked data than for stacked data, an empirical result observed by some researchers. I also raise the issue of random accessing of each trace, a particular need in treating seismic data, and give several solutions that achieve compression while accommodating that need. In addition, I process some field data and evaluate the errors when a certain process is applied to the original data as well as the uncompressed data. It turns out that the error is reduced after migration and enlarged after deconvolution. Of course, besides the amount of compression, the cost of compression and uncompression is another important factor in possible applications of data compression. In general, the more available computing power and less available storage and transmission bandwidth, the more advantage can be taken of data compression.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I am grateful to my advisor, Dr. Ken Larner, whose persistent encouragement, technical discussions and valuable suggestions kept this work going.

I would like to thank Dr. Dave Hale who suggested this project and provided me with an enjoyable summer at Advance Geophysical.

Thanks also go to the members of my committee, Drs. Norm Bleistein, Jack Cohen, John Scales, and Ilya Tsvankin, for critiquing this thesis.

I would like to thank my fellow students for making my time at the Colorado School of Mines memorable. Special thanks to Zhenyue Liu, Lydia Deng, Tagir Galikeev and Timo Tjan for their support in the difficult times. Thanks to Tariq Alkhalifah for providing the data as well as the velocity model used in the test.

I would also like to thank the sponsors of the Consortium Project at the Center for Wave Phenomena for their financial support. The computing facilities necessary for this research were provided by the Center for Geoscience Computing, Colorado School of Mines.

Chapter 1

INTRODUCTION

Seismic data volumes, these days, are huge and growing. With the emergence of 3D technology, a typical data volume is particularly large ($> 10^{12}$ bytes are common in 3D surveys). Simply archiving these data requires a vast amount of storage. Moreover, as more data are processed and interpreted on workstations, more data transfer among the workstations through local area networks is required.

It is therefore desirable to compress the data, in order to reduce the costs of storage and transmission. The ability to compress the data by a factor of 10 means that 10 times as much data could be stored in the space required to store one uncompressed dataset. Moreover, transmitting the compressed data therefore requires only one-tenth the bandwidth that it takes to transmit the original data, thus easing the problem of network traffic. One might suspect that the need for data compression is obviated by wide-band technologies such as optical fiber communication networks and optical storage disks. While it is true in some examples that available bandwidth lessens the need for complex data compression, compression can still be important. A prime example is the advantage gained in transmitting compressed data from a multi-streamer ship to a land-based processing center in nearly real time (Donoho, et al., 1995; Stigant, et al., 1995). More generally, resources such as bandwidth obey a corollary of Parkinson's Law: Resource use will expand to meet the resources available. This has certainly been the experience in exploration seismology. Hence there will always be advantages to be gained from data compression in terms of efficient use of bandwidth and storage capacity, even though the costs of bandwidth and storage capacity continue to drop.

There are two broad categories of data compression techniques: *lossless* and *lossy*. Lossless compression means no information is lost during the cycle of compression and decompression, and the original signal can be perfectly reconstructed from the compressed one. Lossy compression, on the other hand, means some information is lost during compression. Cost aside, lossless compression is what every customer would like since it provides a flawless reproduction of the original. However, the compression achievable by lossless compression is limited to around only 2:1. On the other hand, when seismic data, which inevitably are contaminated by noise in one form or another, are sampled and recorded, some amount of error is already introduced. So, instead of trying to reproduce the original signal perfectly, it is realistic to make compromises that yield reproductions that are satisfactory for foreseeable purposes, while achieving a much higher compression ratio.

Even with this said, lossy compression of seismic data has not yet represented an acceptable technical route. The main reason is that the seismic industry has been focusing on improving data quality, and lossy compression will potentially degrade data quality. However, some recent studies (Hewlett and Hatton, 1995; Hall et al., 1995) show that the amount of degradation introduced by even 10:1 compression is comparable to the difference caused by

using slightly different parameters of some processing modules, such as the filter length in deconvolution and the velocity in stacking, and significantly smaller than the difference experienced by processing with different software packages that are designed to accomplish the same task. These studies indicate that lossy compression can be used during the processing and interpretation stages, even though the raw acquired data may need to be kept intact. Another factor limiting use of data compression in practice is the cost of compressing and decompressing data. However, as modern computers become more powerful, this cost will become less significant. Hence, seismic data compression might find more applications.

Much of the original work in lossy compression can be found in the area of speech and image processing (e.g. Bellamy, 1991; Wallace, 1991; Le Gall, 1991). For seismic data, Wood (1974) discussed compression by truncating the Walsh transform of each trace. Bordley (1983), on the other hand, used linear predictive coding (LPC) to compress marine seismic data. Spanias et al. (1991) compared LPC with some of the transform-based compression techniques, such as the Karhunen-Loeve transform (KLT), the Walsh-Hadamard transform (WHT) and the discrete cosine transform (DCT). More recently, Luo and Schuster (1992) applied the wavelet packet transform to the compression of seismic data by discarding the small coefficients of the transform. Bosman and Reiter (1993) studied how the errors in the wavelet transform-based compression propagate through some seismic processing modules. Reiter and Heller (1994) compared the compression errors for normal moveout (NMO)-corrected, common-midpoint (CMP) gathers and stacked sections, and found that stacking can reduce the compression errors.

In this thesis, I first introduce some general concepts to gain an understanding of data compression, in terms of what are the components in a practical data-compression algorithm, how each of these components works, and how the amount of compression is related to the amount of error introduced during compression. Then, using some data examples, I discuss some practical issues involved specifically in the compression of seismic data, such as what might be an appropriate way to distribute the compression error, how the different choices of transformation in the algorithm influence the amount of compression achievable, what are the different aspects of the data that determine the amount of compression achievable, and how the different error measures can be used to evaluate the compression result. After that, I study several approaches to dealing with the problem of randomly accessing each trace, a problem that is specific to seismic data compression. These approaches include the one-dimensional technique of compressing each trace independently, thus ignoring lateral (trace-to-trace) coherency; the two-dimensional technique of compressing small two-dimensional strips with standard two-dimensional compression techniques, and the mixed technique of applying lateral prediction first and then compressing the predicted residuals trace-by-trace using one-dimensional techniques. Finally, I process some real data, both the original data and the data that have been compressed and uncompressed, using two representative processing algorithms, migration and deconvolution, and compare the processed results to study the behavior of the compression error for the two different processes.

Chapter 2

BASICS OF COMPRESSION

In this chapter, I use the commonly used transform-based compression technique to introduce some of the basic concepts of compression.

A transform-based, lossy compression technique consists of three building blocks: transformation, quantization and coding (Figure 2.1). First, some transform is applied to the signal. This step is completely lossless and invertible. The idea is that after an appropriate transform (the discrete cosine transform, the discrete wavelet transform, etc.), the coefficients are much less correlated than are the original samples so that the information may be more "compact" in the sense of being concentrated into a relatively smaller region in the transformed domain than in the original data domain. Being more compact, intuitively, fewer numbers (coefficients) are required to convey the content of the data. Even though there is no theorem stating that uncorrelated samples can be more efficiently compressed, as a matter of practice and experience, the more uncorrelated or independent a set of variables, the simpler the compression technique that can be used. After the transform, the coefficients, which are represented by real numbers, are converted into a finite set of integer numbers through quantization. It is in this step that the "lossy" part of the compression process occurs. The advantage of quantization however, is that fewer bits are needed to approximate the coefficients. Finally, the integer numbers are encoded by some lossless technique to further reduce the number of bits required to represent the data.

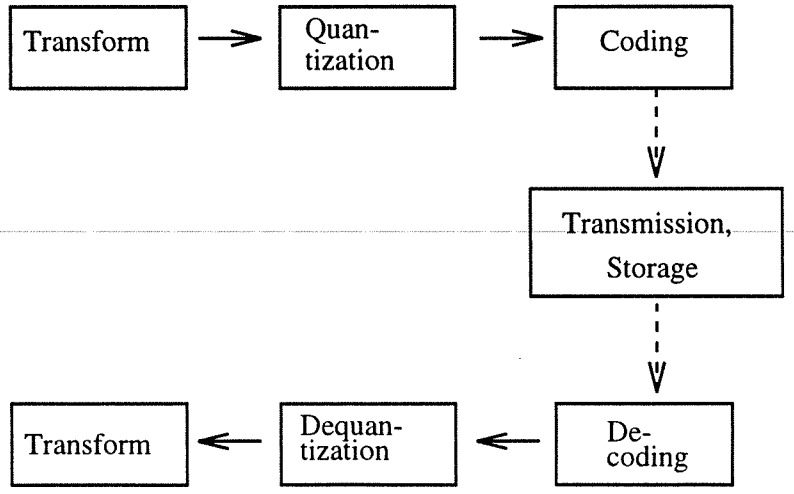

FIG. 2.1. Building blocks of a transform-based compression system.

In the following, I discuss the basic concepts used in quantization and coding. We shall

return to the topic of transforms in Chapter 3.

2.1 Quantization

Quantization is a process that maps an input signal, which can generally take on any value from a continuous range of possible amplitudes but is actually represented by floating-point numbers, into the output signal, which is uniquely specified by an integer in the set $1, 2, 3, ..., N$. In doing this, fewer bits are used and some resolution (or precision) is lost, resulting in some error.

Since quantization is the only step where approximations are made in representing the signal, how one designs a *quantizer* — an algorithm performing the quantization — will determine how the error is distributed in the approximation. This, in turn, will have direct influence on how the approximated signal looks, how the waveform in the approximation differs from that in the original, and how the approximation error propagates through different processing modules. Therefore, a good quantizer is one that is tuned for a specific type of signal and the processes that will be applied to it. For example, the quantizers that are best for speech signal compression are different from those used in image compression. Applying the quantizers (and the compression techniques using those quantizers) designed for one type of signal to another type is generally inappropriate.

In order to design quantizers that might be appropriate for seismic signals, it is necessary to understand the theory of quantization, a subject that is more complicated than it appears. In their book, Gersho and Gray (1992) discuss many different quantizers and therefore provide many options. However, after most transformations, *scalar quantizers* — where each sample is quantized independently (as opposed to *vector quantizers* where the samples are quantized as a group) — are often used for simplicity.

A scalar quantizer is an operator Q that maps real numbers x within a range (a, b) into a finite set of *output levels* $y_1, y_2, ..., y_N$ with the indexing chosen so that $y_1 < y_2 < \cdots < y_N$. The domain of the operator (a, b) consists of N *cells* $(x_{i-1}, x_i]$, each corresponding to an output level. In quantization, if an input number x falls in the ith cell $(x_{i-1}, x_i]$, then it can be uniquely approximated by an output level y_i and therefore represented by the index i. Since i can take on one of only N values $1, 2, ..., N$, $\log_2 N$ bits are needed to represent one quantized sample. The distance $\Delta_i = y_i - y_{i-1}$ is called the *stepsize*. The *maximum error* of a quantizer is defined as the maximum of the difference between the input and output, $\max|x - y_i|$, which is obviously less than $\max_i|x_{i-1} - x_i|$. The more commonly used error, however, is the mean-squared-error (MSE), which is the square of the root-mean-square (RMS) error. MSE is also called the *L^2-average distortion* (I will simply call it *average distortion* throughout the rest of the thesis). It is given by

$$D = \int [x - Q(x)]^2 f_X(x) dx, \tag{2.1}$$

where x is the input sample, $Q(x)$ is the output value, and $f_X(x)$ is the probability density function of random variable X.

Depending on how the y_i's are distributed, scalar quantizers are further categorized

as *uniform* and *nonuniform*. In a uniform quantizer, the output level y_i's are uniformly distributed; they are the midpoints of the equal-sized cells, $y_i = (x_{i-1} + x_i)/2$, as exemplified in Figure 2.2. Otherwise, the quantization is nonuniform, as shown in Figure 2.3.

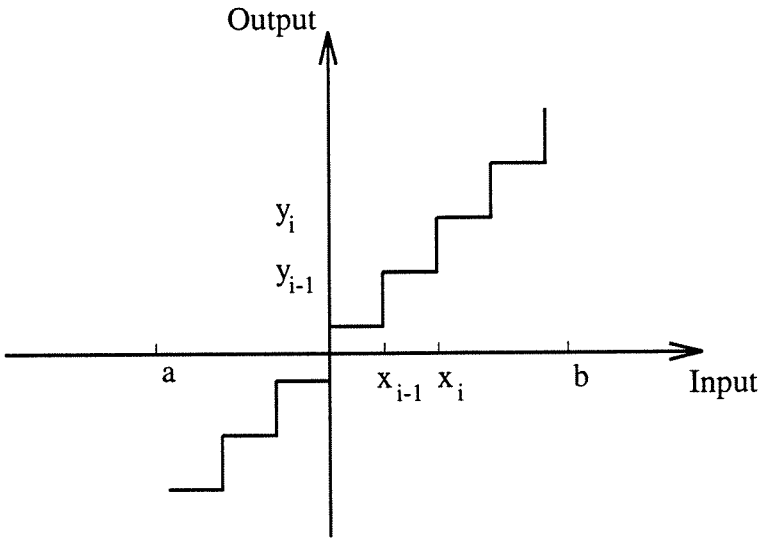

FIG. 2.2. A uniform quantizer wherein the output levels y_i are uniformly distributed and are the midpoints of the cells.

Given these definitions, following are some results important in designing a quantizer. The proofs for most of the observations can be found in Gersho and Gray (1992). Notice that these results are based on the so-called high-resolution assumption, meaning the number of output levels N is large and the average distortion is small. (A common borderline of high resolution is when the average distortion is less than 10 percent of the mean-squared amplitude of the signal.)

1. *For a given number of output levels N, the uniform quantizer minimizes the maximum error, and the maximum error for a uniform quantizer is $\Delta/2$, where Δ is the stepsize. (All the stepsizes are the same for the uniform quantizer.)*

   This observation shows that the uniform quantizer has, besides its simplicity, a useful robustness so that it maintains comparable performance for a wide variety of input signals.

2. *The average distortion of the uniform quantizer is*

$$D = \frac{\Delta^2}{12}, \tag{2.2}$$

   provided that $f_X(x)$ is smooth and the stepsize Δ is small relative to the RMS amplitude of the signal, so that the high-resolution assumption is satisfied.

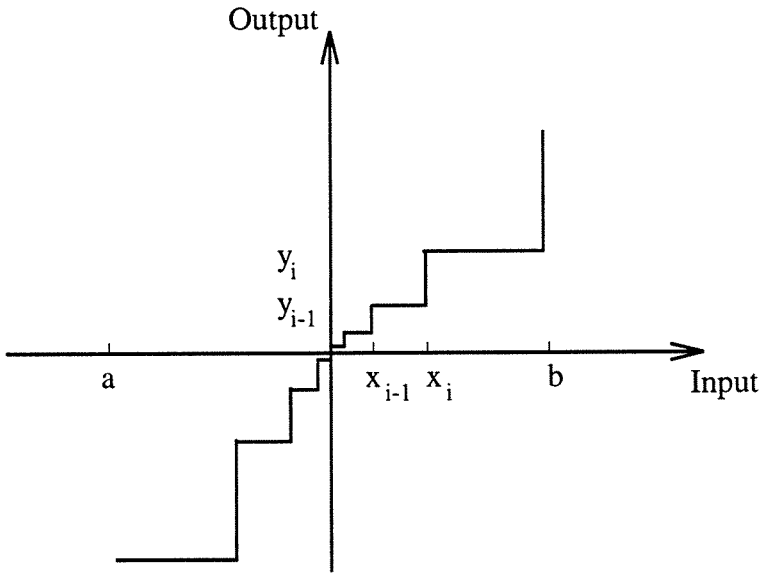

FIG. 2.3. A nonuniform quantizer. The output levels y_i are non-uniformly distributed so that the stepsizes for smaller input values are smaller than those for larger input values.

This observation relates the average distortion (or the MSE) to the stepsize in the uniform quantizer for small errors. This relation is important in designing a uniform quantizer given the MSE. Once an acceptable level D is specified, the stepsize is just

$$\Delta = \sqrt{12D}. \tag{2.3}$$

3. *For a given number of output levels N, a nonuniform quantizer that matches the input probability density function $f_X(x)$ minimizes the average distortion.*

In this observation, it is the average distortion that needs to be minimized. Intuitively from equation (2.1), a smaller error is desired for x values with larger $f_X(x)$. Therefore, the error distribution needs to depend on the probability density function $f_X(x)$. This in turn requires that the output levels of the quantizer depend on $f_X(x)$. To be more specific, we first define the *point density function* as follows. Suppose we have a family of nonuniform quantizers each with the same relative concentration of output levels but with a successively increasing number of levels, N. As N gets large, let $N(x)dx$ denote the number of quantization levels that lie between x and $x + dx$ and assume that as $N \to \infty$ we have a limiting density function (the point density function)

$$\lambda(x) = \lim_{N\to\infty} \frac{N(x)}{N}. \tag{2.4}$$

With this definition, the stepsize Δ_i can then be represented by the point density

function as

$$\Delta_i \approx \frac{1}{N\lambda(y_i)}, \tag{2.5}$$

where y_i is the corresponding output level. Gersho and Gray (1992) show that for the optimal nonuniform quantizer, the point density function satisfies

$$\lambda(x) = \frac{f_X(x)^{1/3}}{\int f_X(y)^{1/3} dy}. \tag{2.6}$$

In most cases, the variable x is not uniformly distributed and $f_X(x)$ is not a constant; therefore a nonuniform quantizer is generally necessary to minimize the average distortion.

The above observations are for the case of *fixed-length coding*, where each output sample is represented by a fixed number of bits, given by $\log_2 N$, and each output index i can take on one of the N possible values.

The next two observations are for the case of *variable-length coding*, where the number of bits for each sample can vary. The result for variable-length coding differs from that for fixed-length coding. Since the results here are related to *entropy*, we first introduce the concept of entropy. The Shannon's differential entropy $h(X)$ is defined for a continuous-alphabet random variable X, as

$$h(X) \equiv -\int f_X(y) \log f_X(y) dy, \tag{2.7}$$

where f_X is the probability density function of X. This $h(X)$ is a characteristic of the data themselves. It quantifies the amplitude distribution of the samples in the data, (solely in terms of how often a certain amplitude value occurs), and therefore in some sense the complexity of the data. Note that this is a different concept from coherency, which characterizes how the samples with similar amplitude values are aligned, or distributed geometrically (spatially). The entropy I discuss here and in the rest of the thesis is only the first-order entropy. A higher-order entropy could be used to characterize the sample-to-sample coherency in the data. However, after an appropriate transform, I shall presume that the samples are uncorrelated, so that entropy (first-order) can then be used to characterize the data. Generally after an appropriate transform, data originally with strong coherency map into data with lower differential entropy.

Though a fundamental characteristic of the data, the differential entropy is difficult to measure. More commonly used is the entropy for a discrete-alphabet random variable (i.e., a random variable that can take on a discrete number of values), defined as

$$H_Q \equiv -\sum_{i=1}^{N} P(i) \ \log_2 P(i), \tag{2.8}$$

where $P()$ is the *probability mass function*, the discrete analog of the probability density function, which can be approximated by the frequency of occurrence of the symbol i in a sequence. Throughout the rest of the thesis, I shall refer to H_Q simply as entropy.

Entropy is a measure that characterizes the information content in a signal. The larger the entropy, the more "complicated" and less predictable the signal, and therefore the more information contained. Figure 2.4 shows a very simple example consisting of two signals, the first having an amplitude value of unity for every sample and the second having an amplitude value of either plus or minus one for each sample. For these two signals, if one needs to "bet" on what the amplitude of a certain sample will be, he is sure that it is unity for the first signal. For the second signal, he will not be so sure, since there is 50% probability that the amplitude value will be plus one, and 50% probability that the amplitude value will be minus one. Using the definition in equation (2.8), it is not difficult to find that the first signal has an entropy value of 0 and the second one 1. The first signal is more predictable and less complicated and therefore has a smaller entropy value (in fact, in this case, because it is fully predictable, its entropy is zero).

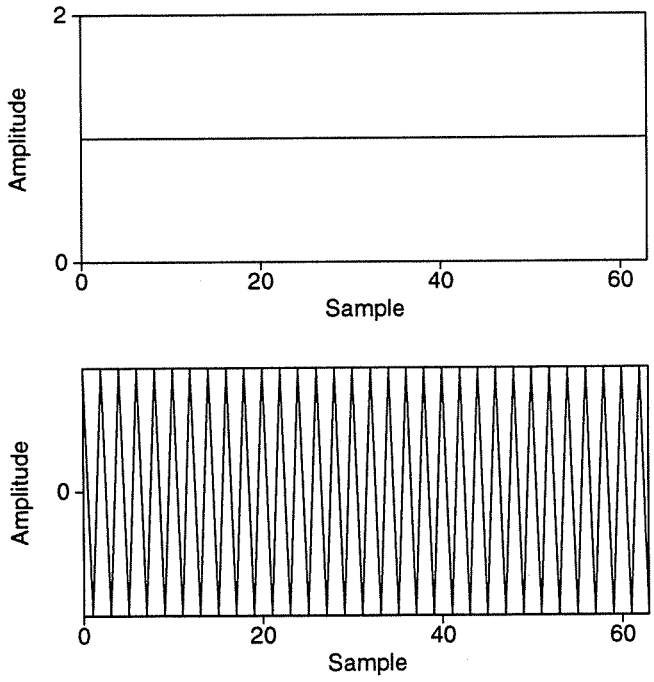

FIG. 2.4. A simple example to illustrate the concept of entropy. The first signal is more predictable and less complicated, and therefore has a smaller entropy value.

In terms of storage and transmission costs, the larger the entropy, the more bits needed to store the signal and the more bandwidth needed to transmit the signal. For data

compression, entropy is a quantity that determines the ideal average number of bits per sample needed to represent a signal. Therefore, the lower the entropy, (i.e., the lower the information content), the fewer bits needed to represent the signal, and the more compression that can be obtained.

4. *For a fixed entropy, the uniform quantizer minimizes the average distortion. Equivalently, for a fixed average distortion, the uniform quantizer achieves the minimum entropy.*

   Notice the difference between this observation and Observation 3. Observation 3 indicates that for a given number of total bits, if a fixed-length coding technique is employed, then it is a nonuniform quantizer that minimizes the average distortion. Or equivalently, for a given amount of error, if a fixed-length coding technique is used, then it is a nonuniform quantizer that minimizes the number of bits needed. Observation 4, on the other hand, gives the optimal result if a variable-length coding technique is used, and the optimal result in this case is the uniform quantizer.

5. *The minimum entropy for a fixed average distortion is given approximately by*

$$H_Q \approx h(X) - \frac{1}{2}\log_2 12D, \tag{2.9}$$

   provided that the average distortion D is small relative to the mean-squared amplitude of the signal so that the high-resolution assumption is satisfied.

   This observation, relating the entropy value H_Q to the average distortion D (or the MSE), is useful when designing a quantizer given the desired number of bits. Even though the desired number of bits can be approximated as entropy H_Q, we cannot obtain D directly from this equation since $h(X)$ is difficult to obtain in practice. A practical procedure could be as follows. First, use an initial value (this can be a random guess or something that is based on previous experience) for the average distortion D_1 allowed in compressing the data, design a quantizer with the stepsize determined from equation (2.3), and apply the entropy coding. Then, divide the number of bits after entropy coding by the original number of samples, and obtain the resulting average number of bits (which is approximately H_{Q_1}). With this D_1 and H_{Q_1}, combined with the desired number of bits, which can be approximated as another entropy value H_{Q_2}, we can then obtain an estimate of the average distortion D_2 in this case since, from equation (2.9), we have

$$H_{Q_1} - H_{Q_2} = -\frac{1}{2}\log_2 \frac{D_1}{D_2}, \tag{2.10}$$

   or equivalently,

$$D_2 = D_1 2^{2(H_{Q_1} - H_{Q_2})}. \tag{2.11}$$

   With this D_2, we can then determine the associated stepsize, again from equation (2.3).

I will use these observations in later chapters to design the quantizer for seismic data compression. Also, I will use some of the observations to explain phenomena observed by

Table 2.1. A simple example of Huffman coding.

i	$P(i)$	Natural Code	Huffman Code
0	1/2	000	0
1	1/4	001	10
2	1/8	010	110
3	1/16	011	1110
4	1/32	100	11110
5	1/64	101	111110
6	1/128	110	1111110
7	1/128	111	1111111

other researchers.

2.2 Coding

As suggested in the previous section, good compression can be achieved by coding the output of a uniform quantizer using a variable-length coding technique, called *entropy coding*. Entropy coding is a lossless compression step that attempts to compress the data so that the average number of bits per symbol is close to the entropy of a sequence of symbols, defined by equation (2.8). An algorithm that performs the entropy coding is called an entropy coder. The literature contains extensive study on entropy coding, and detailed accounts may be found in many books and papers, e.g., Gersho and Gray (1992).

Examples of the many forms of entropy coders include Huffman coders (Huffman, 1952), arithmetic coders (Witten et al., 1987) and dictionary-based coders (Welch, 1984). Here, I use a simple example to show how Huffman coders compress data.

Suppose we have a sequence of symbols, wherein each symbol belongs to the set of $\{0, 1, 2, ..., 7\}$. Their corresponding binary (natural) codes are shown in Table 2.1. The binary code requires 3 bits per symbol, no matter what the distribution of the symbols in a sequence. Now suppose each symbol i has the frequency of occurrence or probability $P(i)$ shown in the table. In Huffman coding, each symbol i is assigned a code according to its probability $P(i)$. The Huffman code length for symbol i approaches $-\log_2 P(i)$. (The code length needs to be an integer number. However, $-\log_2 P(i)$ might not be an integer number; therefore the code length only approaches $-\log_2 P(i)$. In this example, the Huffman code length for symbol i actually equals $-\log_2 P(i)$, since $-\log_2 P(i)$ is already an integer value.) Therefore, symbols occurring more frequently will have shorter code length, as shown by their Huffman codes in the above table. For this example, Huffman code requires

$$H_Q = -\sum_{i=1}^{N} P(i) \log_2 P(i)$$

$$
\begin{aligned}
&= -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{8}\log_2\frac{1}{8} - \frac{1}{16}\log_2\frac{1}{16} \\
&\quad -\frac{1}{32}\log_2\frac{1}{32} - \frac{1}{64}\log_2\frac{1}{64} - \frac{1}{128}\log_2\frac{1}{128} - \frac{1}{128}\log_2\frac{1}{128} \\
&= 1\times\frac{1}{2} + 2\times\frac{1}{4} + 3\times\frac{1}{8} + 4\times\frac{1}{16} \\
&\quad +5\times\frac{1}{32} + 6\times\frac{1}{64} + 7\times\frac{1}{128} + 7\times\frac{1}{128} \\
&\approx 1.98
\end{aligned}
$$

bits per sample on average. Therefore, for this example Huffman coding compresses the sequence by a factor of more than 3 : 2 relative to binary coding. Of course, this is just an idealized example. In practice, in order to recover the coded sequence, we need to store some side information, such as the table that translates the symbols into the Huffman codes (the so-called *codebook*), or other equivalent information such as the probability values. Spreading the amount of side information required over all the samples results in an increased average number of bits per sample needed. Therefore, due to this overhead, the average number of bits per sample after entropy coding will be larger than the entropy value.

Since, after entropy coding, the average number of bits per sample approaches the entropy, the lower the entropy of the data, the fewer bits required per sample of the representation and the more compression will be achievable. From the definition [equation (2.8)], it is not difficult to show that the more evenly distributed is $P(i)$, the higher the entropy. If in the previous example, all the symbols i have the same probability $P(i)$, the entropy will be 3 bits, and no compression can be achieved. This is made precise by the following theorem (Wickerhauser, 1994).

Theorem 1. *Suppose that p and q are two probability mass functions over an N-point sample space, with corresponding entropies as $H(p)$ and $H(q)$. The sequences $p(i) : i = 1, 2, ..., N$ and $q(i) : i = 1, 2, ..., N$ are sorted so that they are non-increasing sequences. Define the partial sum sequence $S[p]$ as $S[p](k) \equiv \sum_{i=1}^{k} p(i)$. If $S[p] \geq S[q]$, then we say p is more concentrated than q, or equivalently q is more evenly distributed than p.*

- *If $S[p] \geq S[q]$, then $H(p) \leq H(q)$.*
- *$0 \leq H(p) \leq \log_2 N$, where $H(p) = 0$ if and only if $p(1) = 1$ with $p(n) = 0$ for all $n > 1$, and $H(p) = \log_2 N$ if and only if $p(1) = p(2) = \cdots = p(N) = \frac{1}{N}$.*

2.3 Compression ratio

With the above concepts introduced, it is now easy to discuss the quantity *compression ratio*. The compression ratio r is defined as the ratio of the average number of bits per sample before and after compression. For some given data, the number of bits per sample generally is a fixed quantity b before compression, and it can ideally be the entropy H_Q after

compression, if entropy coders are used. Therefore, the ideal compression ratio is given by

$$r \equiv \frac{b}{H_Q}. \tag{2.12}$$

Since H_Q is related to the average distortion D according to equation (2.9), the compression ratio is therefore a function of D

$$r(D) \equiv \frac{b}{H_Q} = \frac{b}{h(X) - \frac{1}{2}\log_2 12D}. \tag{2.13}$$

This equation describes the trade-off between the quantization error, or the average distortion D, and the compression ratio r. The larger the average distortion D, or the error allowed, the larger the ideal compression ratio r.

Generally, it is difficult to estimate the absolute $h(X)$ and D from the data. However, we can get some idea about the data to be compressed by performing a simple experiment. Suppose we use a pre-defined value of the average distortion D_0, design the quantizer according to equation (2.3), and compress the data using the entropy coder. Of course, the resulting number of bits per sample will be larger than the true entropy value. In most cases, however, the overhead is not significant so we will use the measured average number of bits per sample as the true entropy value $H_{Q_0} = h(X) - \frac{1}{2}\log_2 12D_0$, and compute the compression ratio r_0 according to equation (2.12). Therefore, we obtain a pair of values D_0 and r_0. With these initial values, we can then predict the ideal compression ratio for other values of average distortion, according to the following equation

$$r(D) = \frac{b}{\frac{b}{r_0} - \frac{1}{2}\log_2 \frac{D}{D_0}}. \tag{2.14}$$

Expression (2.14) is obtained by eliminating $h(X)$ in equation (2.13) and the following equation

$$r_0 = r(D_0) = \frac{b}{h(X) - \frac{1}{2}\log_2 12D_0}. \tag{2.15}$$

Defining the relative error e as the MSE (which is just the average distortion D) divided by the mean-squared amplitude of the data E,

$$e \equiv \frac{D}{E}, \tag{2.16}$$

the compression ratio can alternatively be represented as a function of e

$$r(e) = \frac{b}{\frac{b}{r_0} - \frac{1}{2}\log_2 \frac{e}{e_0}}, \tag{2.17}$$

where $r_0 = r(e_0)$.

Equation (2.17) gives a way to predict how much compression one can expect for a given

amount of tolerated error e. Once the initial value r_0 is measured for a given value of e_0, the compression ratio r for any other error value e can be readily calculated from equation (2.17).

Figure 2.5 shows how the compression ratio $r(e)$ changes with the relative error e for two hypothetical data sets, one with $r_0 = 6$ and the other with $r_0 = 4$, when $e_0 = .01\%$ and $b = 32$ are assumed for both cases. It looks similar to the one empirically obtained by Reiter and Heller (1994), where they compared how the compression ratios change with the relative error for an NMO-corrected common-midpoint (CMP) gather and a stacked section. From comparison of the two curves, they concluded that the error increases with compression

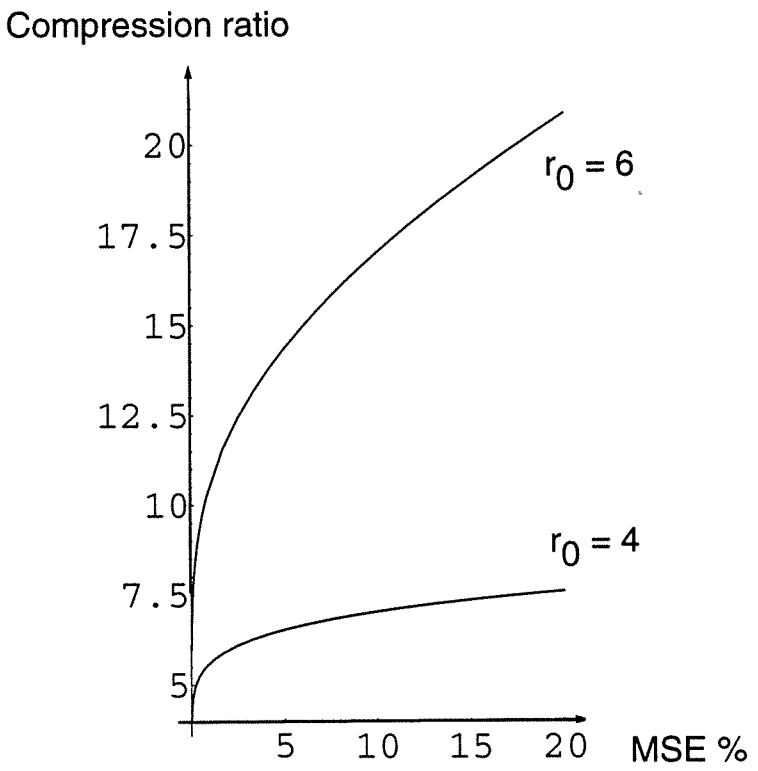

FIG. 2.5. Compression ratios as a function of compression error for hypothetical data sets. We might associate the two curves with unstacked data (lower curve) and CMP-stacked data (upper curve).

ratio more rapidly for CMP gathers than for stacked sections. Figure 2.5 gives a possible explanation for this phenomenon. Compared to CMP gathers, stacked sections often have higher signal-to-noise ratios and therefore more coherency. After some appropriate transform, they may therefore have a smaller differential entropy value $h(X)$. (I will illustrate this in later chapters.) Since

$$r_0 = \frac{b}{h(X) - \frac{1}{2}\log_2(12D_0)},$$

the stacked section will have a larger initial compression ratio r_0 for the same amount of

initial error than will the unstacked data. From equation (2.17), then, the error will increase more slowly for stacked sections, as illustrated by the curves shown in Figure 2.5.

Chapter 3

TRANSFORMATIONS

Any of a collection of transforms can be used in a transform-based lossy compression technique. Here, I briefly introduce two types of transforms that I study: the discrete cosine-type transform and the discrete wavelet-type transform. Detailed treatment can be found in Wickerhauser (1994) and Daubechies (1992). Comparison of the actions of these transforms in compression of seismic data will be left to later chapters.

3.1 Discrete Cosine Transform

The *discrete cosine transform* (DCT) has some of the features of a transformation to the frequency domain. It is equivalent to a discrete Fourier transform of a symmetricized extension of the input signal. For a given input signal $x(n)$, the transformed coefficients can be obtained as (e.g., Gersho and Gray, 1992)

$$y(m) = b_m \sum_{n=0}^{N-1} x(n) \cos\left[\frac{\pi}{N} m(n+\frac{1}{2})\right], \tag{3.1}$$

where

$$b_m = \begin{cases} \sqrt{\frac{2}{N}}, & \text{for } m = 1, 2, ..., N-1, \\ \sqrt{\frac{1}{N}}, & \text{for } m = 0. \end{cases}$$

This is a slight variation of the discrete Fourier transform of the sequence

$$z(n) = \begin{cases} x(n), & \text{for } n = 0, 1, ..., N-1, \\ x(2N-n-1), & \text{for } n = N, N+1, ..., 2N-1. \end{cases}$$

The reason that DCT is often used instead of the discrete Fourier transform is that it involves only real-number operations; no complex-number operations are needed.

In most applications, DCT is not performed on the entire input signal. Instead, the input signal is subdivided into segments and DCT is performed for each segment to better describe the local (in time) frequency characteristics of the input signal. In this sense, DCT is similar to the windowed Fourier transform, where the transformed coefficients carry local spectral information. Figure 3.1 depicts this subdivision. Clearly, the window used here is just a box function, and each segment is disjoint. Another way to look at this is that the basis functions here are truncated cosine functions. The DCT applies these truncated cosine functions to the original signal.

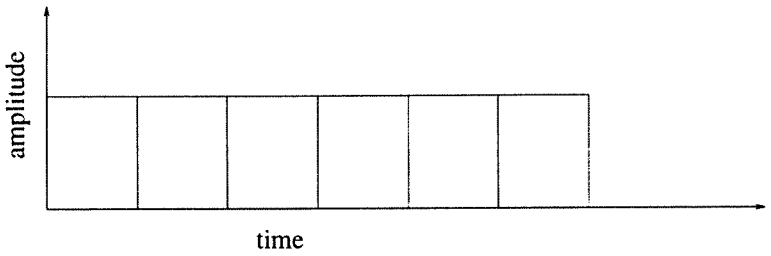

FIG. 3.1. Subdivision for the DCT.

3.2 Local Cosine Transform

Because the box-function window used in DCT is not smooth, applying this window function (multiplying data by the box function) will introduce abrupt changes from nonzero values to zero, distortions that are not generally in the original data. The *local cosine transform* (LCT) overcomes this problem by introducing a smooth orthogonal projection, instead of the box-shape window. Therefore, instead of using the truncated cosine functions as the bases in DCT, LCT uses the smoothly windowed cosine functions. In terms of time-frequency localization, the bases for DCT have perfect time localization (no overlap) but poor frequency localization (the Fourier transform for the box function is the slowly decaying and oscillatory *sinc* function), while the bases for LCT have both time and frequency localization.

In LCT (e.g., Wickerhauser, 1994), a smooth function $r(t)$ is used to provide a smooth-windowed cosine basis function. In order to have orthogonal projection, $r(t)$ needs to satisfy the following conditions:

$$|r(t)|^2 + |r(-t)|^2 = 1 \text{ for all } t \in \mathbf{R}; \quad r(t) = \begin{cases} 0, & \text{if } t \le -1, \\ 1, & \text{if } t \ge 1. \end{cases} \tag{3.2}$$

One example of the function $r(t)$ is the so-called iterated sine function, defined as

$$r_{sin}(t) = \begin{cases} 0, & \text{if } t \le -1, \\ \sin[\frac{\pi}{4}(1+t)], & \text{if } -1 < t < 1, \\ 1, & \text{if } t \ge 1. \end{cases} \tag{3.3}$$

and

$$r_{[0]}(t) \equiv r_{sin}(t); \quad r_{[n+1]}(t) \equiv r_{[n]}(\sin \frac{\pi}{2} t), \tag{3.4}$$

where $r_{[n]} \in C^n$; that is, it has n continuous derivatives. Figure 3.2 shows the function $r_{[0]}(t)$ and Figure 3.3 shows the function $r_{[4]}(t)$. Clearly, $r_{[4]}(t)$ goes to zero much more smoothly than does $r_{[0]}(t)$.

In LCT, instead of applying the box functions, we apply the smooth windows. Figure 3.4 depicts a subdivision for LCT. Clearly, the windows are no longer disjoint; they overlap. Therefore, the transform is no longer immediately orthogonal. One way to achieve an orthogonal transform (Wickerhauser, 1994) is to apply the *folding* operator, as follows.

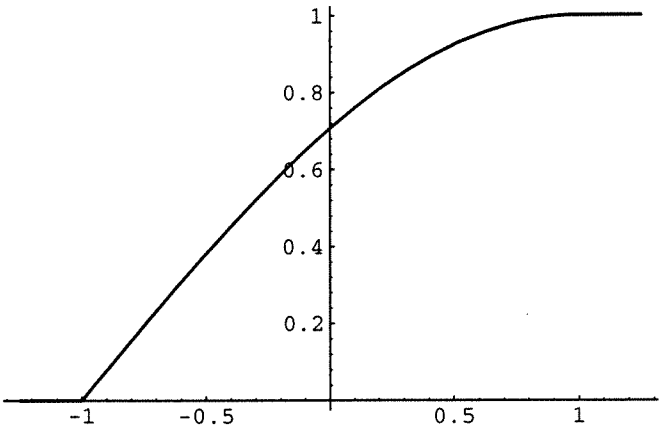

FIG. 3.2. The function $r_{[0]}(t)$.

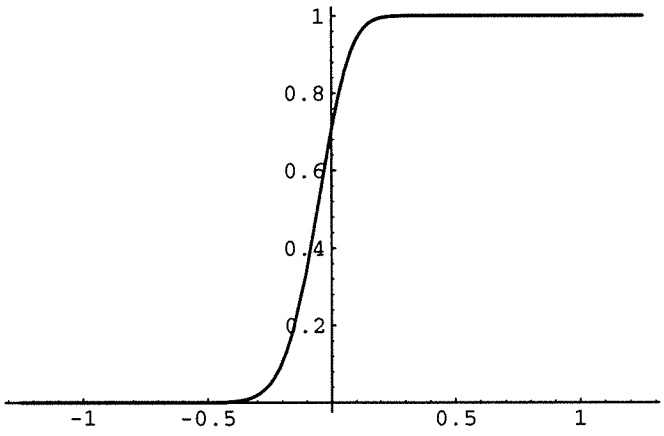

FIG. 3.3. The function $r_{[4]}(t)$.

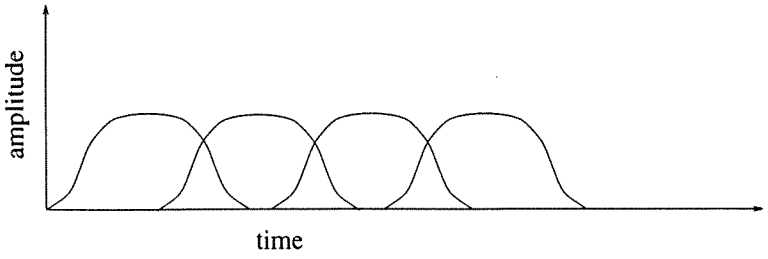

FIG. 3.4. Subdivision for the LCT.

The folding of $x(n)$ is defined as

$$Ux(n) = \begin{cases} r(n)x(n) + r(-n)x(-n), & \text{if } n > 0, \\ r(-n)x(n) - r(n)x(-n), & \text{if } n < 0. \end{cases} \tag{3.5}$$

Its adjoint operator U^*, the unfolding operator, is defined as

$$U^*x(n) = \begin{cases} r(n)x(n) - r(-n)x(-n), & \text{if } n > 0, \\ r(-n)x(n) + r(n)x(-n), & \text{if } n < 0. \end{cases} \tag{3.6}$$

Clearly, if $x(n)$ is supported on the right half-line, i.e. $x(n) = 0$, for $n < 0$, then the unfolding is equivalent to multiplying by the window as

$$U^*x(n) = \begin{cases} r(n)x(n), & \text{if } n > 0, \\ r(n)x(-n), & \text{if } n < 0. \end{cases} \tag{3.7}$$

The LCT of an input signal $x(n)$ then consists of two steps. First, $x(n)$ is cut into pieces and each piece is folded, using a smooth function $r(t)$. Then, the folded pieces are transformed by the DCT. Since $< Ux, \phi >=< x, U^*\phi >$, where $<,>$ denotes the inner-product, and ϕ denotes the truncated cosine function or the basis used in DCT, then folding the data followed by DCT is equivalent to transforming the data by an unfolded truncated cosine function, which is just a smoothly-windowed cosine basis function.

3.3 Wavelet Transform

The wavelet transform (Daubechies, 1992; Wickerhauser, 1994) is another orthogonal transform that can capture the local spectral information of a given signal. Different from the previously discussed cosine-type transforms where the window size is fixed, the wavelet transform employs windows whose widths are adapted to the frequency content, i.e., lower frequencies have longer windows and higher frequencies have shorter windows. The discrete wavelet transform employs two important filters [the so-called *conjugate quadrature filters* (CQFs) in signal processing literature]: the high-pass filter D and the low-pass filter A. A signal $x(n)$ is first decomposed by applying the two filters followed by decimation or subsampling — retaining only one sample in two. For notation purpose, I call the operation of high-pass filtering followed by subsampling, G, and the corresponding one for low-pass filtering, H. Then the output from H is further decomposed, and the process goes on until only one sample is left for the H operator, as illustrated in Figure 3.5.

In order for the filters to qualify for CQFs, the decomposition needs to be perfectly reconstructable. This imposes some conditions on the operators H and G, as follows (Wickerhauser, 1994):

- *Self-duality:* $HH^* + GG^* = I$;
- *Independence:* $GH^* = HG^* = 0$;
- *Exact reconstruction:* $H^*H + G^*G = I$;

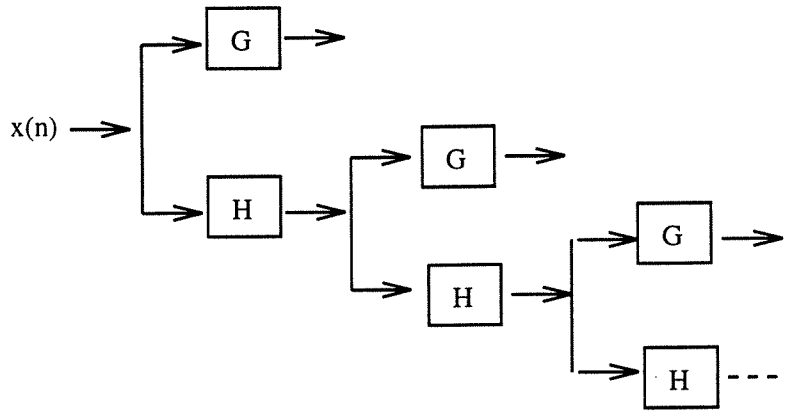

FIG. 3.5. Flow diagram for the discrete wavelet transform. The G and H operators correspond to first filtering by high-pass (for G) and low-pass (for H) filters and then subsampling the output by a factor of two.

- *Normalization:* $H\mathbf{1} = \sqrt{2}\mathbf{1}$ and $G\mathbf{1} = \mathbf{0}$.

  where H^* and G^* are the adjoints of H and G, I is the identity operator, $\mathbf{1} = \{..., 1, 1, 1, ...\}$ and $\mathbf{0} = \{..., 0, 0, 0, ...\}$.

If the operators H and G are formed respectively from the sequences h and g, the above conditions translate into the following equations:

$$\begin{aligned}
\sum_k h(k)h(k+2n) = \delta(n) = \sum_k g(k)g(k+2n);\\
\sum_k g(k)h(k+2n) = 0 = \sum_k h(k)g(k+2n);\\
\sum_k g(2k) = -\sum_k g(2k+1);\\
\sum_k h(2k) = \sum_k h(2k+1) = \frac{1}{\sqrt{2}};\\
\left|\sum_k g(2k)\right| = \left|\sum_k g(2k+1)\right| = \frac{1}{\sqrt{2}}.
\end{aligned} \tag{3.8}$$

Daubechies (1992) presents a detailed study on how to construct the filters. A list of filter coefficients for different types of filters can be found in Wickerhauser (1994).

The action of filters A and D is to split the frequency spectrum into two symmetrical parts, though the split is not perfect and there is some overlap. In the discrete wavelet transform, only the averages (the results after applying H) are further decomposed and the differences (the results after applying G) are kept as part of the output. Therefore, the first-level difference will occupy approximately the upper half in the frequency spectrum. The second-level difference will occupy approximately the upper half in the remaining frequency spectrum, and so on (Figure 3.6). Obviously, the first-level difference has the highest-

frequency component and the widest frequency window, which corresponds to the narrowest time window. Therefore, the window sizes are adapted: the higher the frequency content, the narrower the time window.

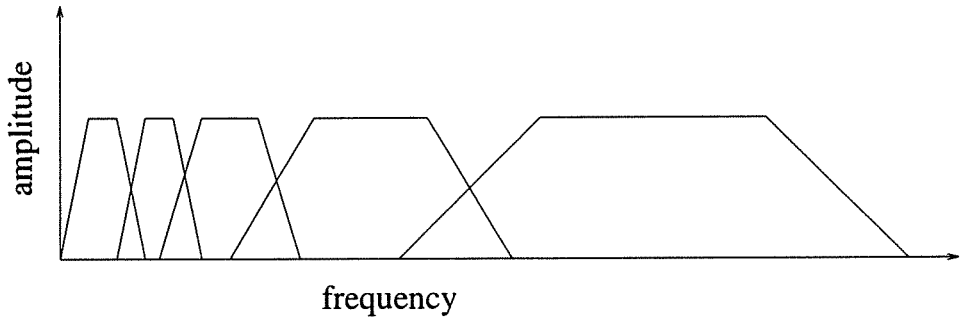

FIG. 3.6. The frequency division in the discrete wavelet transform. The higher frequencies have larger windows, and the lower frequencies have smaller windows.

3.4 Wavelet Packet Transform

In some applications, however, the large frequency windows (small time windows) for the high frequencies might be too crude. It is therefore desirable to further refine the high-frequency component as well as the low-frequency component during the decomposition. This leads to the discrete wavelet-packet transform.

In the discrete wavelet-packet transform, both the outputs from H and G are further decomposed, as shown in Figure 3.7. Consider the structure in this figure as a tree. For each node in the tree, there exists the choice of decomposing further or not. Therefore, there results a huge collection of possible valid decompositions, with the discrete wavelet transform being one of them. Another special decomposition is the so-called fixed-scale wavelet packets where all the nodes are decomposed to the same level. I will use fixed-scale wavelet packets and simply call it the discrete wavelet packet transform in the rest of the thesis.

As opposed to the case of the discrete wavelet transform, in the discrete wavelet packet transform (fixed-level), the frequency spectrum is divided with equal-sized windows, as shown in Figure 3.8.

Both the wavelet transform and the wavelet packet transform have their appropriate applications, as I will show in the next chapter. Because of the adaptive window used, the wavelet transform can zoom in on details (such as the edges in an image) and zoom out on the background (such as the smooth parts in an image). Therefore, it provides an efficient and effective representation for signals with dramatic changes in the local spectral information. With its fixed scale, the wavelet packet transform, on the other hand, can effectively represent bandlimited signals. This is because there is no dramatic change in the local spectral information in a bandlimited signal. The large frequency windows for the high-frequency components in the wavelet transform (Figure 3.6) might be too crude; the wavelet packet transform can provide the needed refinement (Figure 3.8).

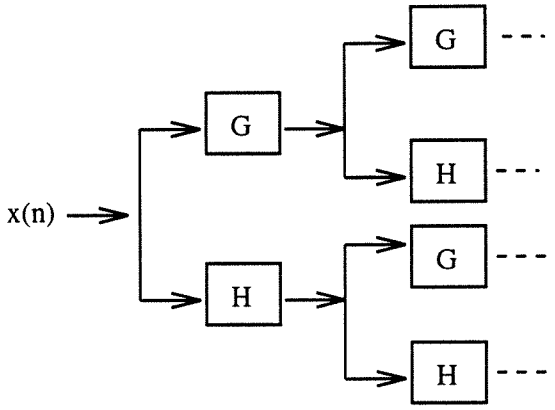

FIG. 3.7. Flow diagram for the discrete wavelet packet transform. The G and H operators correspond to first filtering by high-pass (for G) and low-pass (for H) filters and then subsampling the output by a factor of two.

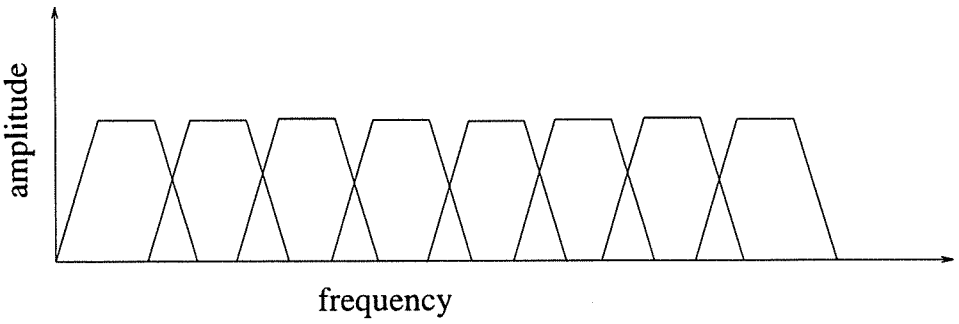

FIG. 3.8. The frequency division in the discrete wavelet packet transform (fixed-level). The window sizes are the same for high and low frequencies.

In practice, the DCT has found many applications in image compression, and current standards in the compression of still and motion pictures are based on the DCT (Wallace, 1991; Le Gall, 1991). Since its introduction, the DWT has been applied to image and seismic data compression (Zettler et al., 1990; Reiter and Heller, 1994). However, since it is a relatively new concept, its applicability requires further investigation. The DWPT, especially combined with the so-called "best-basis algorithm" (Coifman and Wickerhauser, 1992), has been reported to compress speech signals effectively. I will do some experiments in the next chapter to compare the performance of these transforms in the compression of seismic data.

Whichever transform to use depends largely on the character of the data since the idea is to transform the strong coherency in the original data into low entropy in the coefficients. In other words, under an appropriate transform, characteristics of the data that can potentially, but not yet directly, contribute to good compression are converted into a quantity that will directly give good compression. A good transform will take advantage of the coherency or redundancy in the data, resulting in a more compact representation of the data, thus lowering the differential entropy, so that more compression can be achieved than without transform.

Chapter 4

COMPRESSION IN PRACTICE

Having introduced the basic concepts, in this chapter I discuss some practical issues in the compression of seismic data in particular.

The amount of compression achievable for a given level of quality of reproduction after decompression depends on several factors, most important among which are the quantizer and the transform used as they relate to the nature of the data to be compressed, and the error measure.

4.1 Quantizer

In Chapter 2, I reviewed some results on optimal quantizer design. From the observations, different optimal quantizers are best suited for different purposes. For example, in digital telephone communication, where a fixed number of bits for each sample (*fixed-rate codes*) are used, Observation 3 indicates that a nonuniform quantizer that matches the statistics of speech signals is most desirable. Generally, there are more small-amplitude samples than large-amplitude samples in speech signals. Intuitively, to have as small an average distortion as possible, the nonuniform quantizer will allocate smaller errors for the small-amplitude values than for the large-amplitude values, as shown in Figure 2.3, because there are more of them. A nonuniform quantizer similar to this is what is used in the North American standard for digital telephony (CCITT G.711, e.g., Bellamy, 1991). It happens that, besides minimizing the average distortion, the nonuniform quantizer fits the purpose of telephony as well. This is because the human auditory system is not very sensitive to the volume of the sound. For a range of large-amplitude events, the content is already known, and the volume does not make too much difference (it might make some difference in expressing emotions though). In contrast, for the small-amplitude events (whispers) only small errors can be tolerated in order that the content be understandable.

This nonuniform quantizer, however, is not necessarily appropriate for seismic signals. This is because most seismic signals are noisy, at least to the extent of our currently limited understanding. When a nonuniform quantizer is used, more error is allocated to large-amplitude events, because they generally occur less frequently. However, in seismic data large-amplitude events (the stand-outs) are what we are often most interested in. Those are the events from which we often estimate various earth parameters. Keeping those events in position and their amplitudes as accurate as possible, intuitively, will help alleviate the possible exaggeration of the quantization errors in further processing. On the other hand, the small-amplitude events have a good chance of being random or interference noise. The nonuniform quantizer therefore might expend too much effort in approximating possible random noise.

While the observations introduced in Chapter 2 give some guidelines for designing a quantizer given an objective criterion, the situation will be different if some subjective criteria are used. For speech signals, as explained above, a nonuniform quantizer similar to the one shown in Figure 2.3 fits the purpose of telephony, perceptively. For image signals, however, different quantizers are generally used to yield desirable visual effects. Since human eyes are more sensitive to low-frequency features, in most image quantizers higher frequencies are quantized more roughly, i.e., with more error. If this type of quantizer is used for seismic data, it might cause not only amplitude errors, but phase errors as well.

It is highly desirable to design quantizers specially suited to seismic data. This is a difficult task, however, due to the multifacet nature of the applications of seismic data. In interpretation for example, the main concern might be the appearance of the data. Therefore, making the reconstruction look as close as possible to the original is important. For some of the processing modules, the main interest might be the low-frequency component of the data, while for others, it is the high-frequency component. Therefore, using a quantizer that favors one process might introduce large errors for others. One approach might be to design a quantizer for each step of the processing. However, before a final image is obtained, the data have to go through a sequence of processes, and once a "feature" is lost in the early steps, it will no longer exist for the rest of the processing flow, resulting in possibly large errors for processes that focus on this feature.

Until another quantizer is found to be more appropriate, the uniform quantizer might be a safe choice. From Observation 1, the uniform quantizer minimizes the maximum error for a given number of output levels N. Therefore, the uniform quantizer is robust in that good performance can be maintained for a wide variety of input signals. With the error allocated uniformly, the targeted features (large-amplitude events) are approximated accurately, while the small-amplitude events are treated with some care as well. The above reasoning remains valid for the transformed domain in a transform-based compression technique, as well as in the original data domain.

The uniform quantizer might be a safe choice for allocating the error, but will it provide enough compression for an allowed amount of error or average distortion? From Observation 4, the uniform quantizer minimizes the entropy. Since entropy coders can produce a compressed sequence whose average number of bits per sample approaches the entropy, they will provide the most compression for a given average distortion if the uniform quantizer is followed by an entropy coder.

Besides the quantization and coding approach, another approach discussed in some of the literature (e.g. Luo and Schuster, 1992) involves discarding the small transformed coefficients. This perhaps intuitively appealing approach can be considered as a special form of quantizer, where the small amplitudes are set to zero while the large amplitudes are kept intact. Therefore, large-amplitude events are treated with extreme care (with no approximation at all) while small-amplitude events are totally ignored. It is not difficult to compare the two approaches in terms of the amount of compression for a given error in the reconstructed data. For the stacked section shown in Figure 4.1, whose two-dimensional amplitude spectrum is shown in Figure 4.2, I compared the compression technique of quantization with coding, with the method of discarding small coefficients.

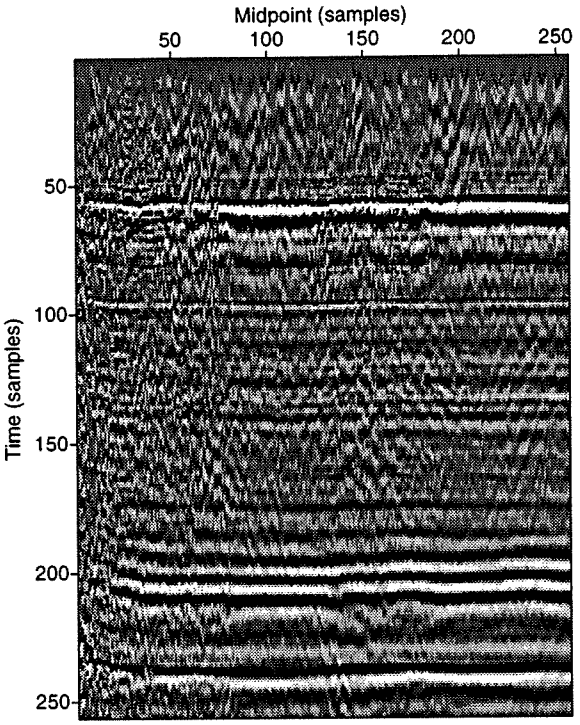

FIG. 4.1. Brute stack.

The transformations used are identical for both cases, the two-dimensional extension of the DWPT, with five levels of decomposition. To achieve an RMS error no larger than one percent, the quantization (with a uniform quantizer) and coding technique gives about 5.75:1 compression, with the RMS error defined as

$$RMS(e) = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}}, \tag{4.1}$$

where x_i are the sample values of the original signal and e_i are the sample values of the difference between the original and the reconstructed signals. To achieve this same amount of compression, the method of discarding small coefficients would require throwing away more than 80 percent of the smallest coefficients and also storing the indices of the remaining coefficients. This, however, gives an RMS error as large as 20 percent even though the coefficients discarded are smaller than two percent of the largest coefficient because, after transformation the coefficients become more compact, resulting in many small coefficients relative to the number of large ones. This result is just an example of Observation 4: the uniform quantizer minimizes the entropy for a given average distortion.

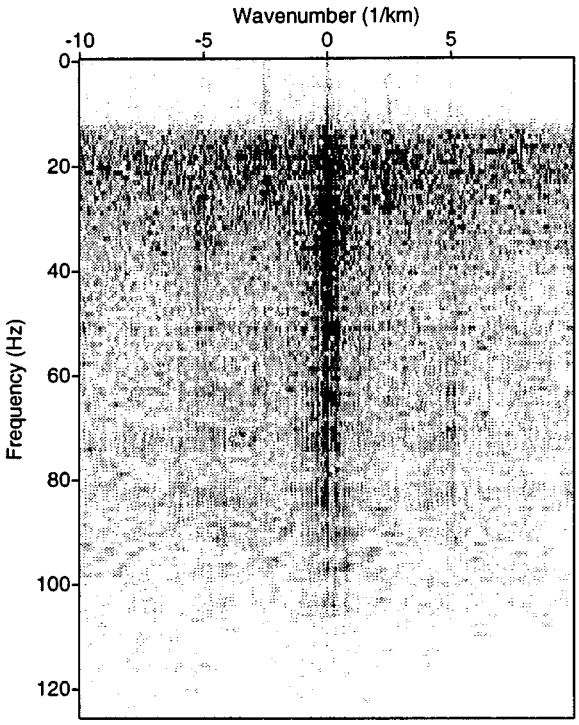

FIG. 4.2. Two-dimensional amplitude spectrum of the brute stack shown in Figure 4.1.

4.2 Transform

The choice of the transform is another factor that determines the amount of compression achievable. Since I will be adopting the approach of quantization plus coding, the amount of compression achievable can be quantified by the entropy value. In the following, I use the entropy value as the criterion in comparing some of the transforms.

Generally speaking, the transform that "best" characterizes the data can achieve the most compression. The wavelet transform, because of the adaptive windows used, is a good fit for the smooth-background-plus-local-discontinuity nature of images.

Figure 4.3 shows an example of what I call "image"-type data. It is a portion of the Marmousi velocity model (Versteeg and Grau, 1991). Compared to typical bandlimited seismic data, (see e.g., Figure 4.1), it is smooth in each block, with discontinuities at the boundaries between two blocks. Figure 4.4 shows one vertical slice of the section, i.e., the velocity function at the first midpoint. Clearly, it consists of smooth parts with isolated abrupt changes at the boundaries. For the section in Figure 4.3, I apply the wavelet transform, the wavelet packet transform (again, the fixed-level) and the discrete cosine transform, all along the vertical dimension, and compute the entropy values after the transforms. (All the transforms I discuss in this section are one-dimensional, along the vertical dimension, since

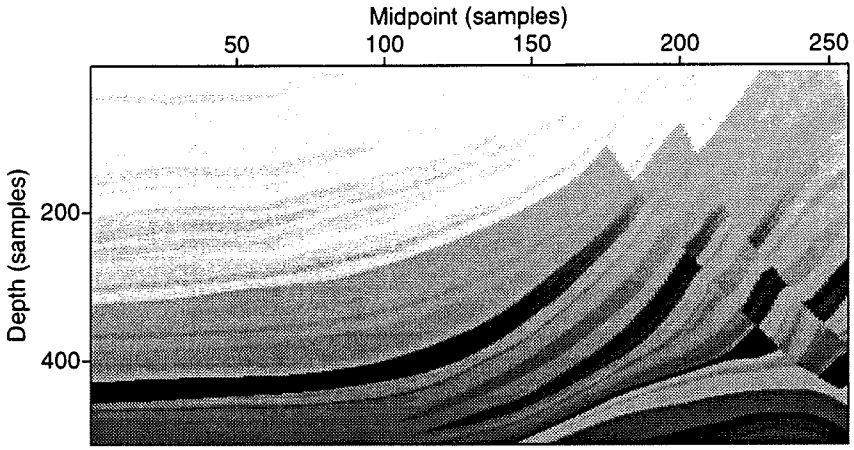

FIG. 4.3. Marmousi velocity model.

I want to do the comparison along this dimension.) In computing the entropy values, I first convert the transformed coefficients from real numbers into integer values under about one percent RMS error. (I will use one percent RMS error for all the entropy evaluations in this chapter.) After the wavelet transform, the entropy is 1.76 (bits). Both the discrete wavelet packet transform (DWPT) and the discrete cosine transform (DCT) have an extra parameter — the level of decomposition for the DWPT, and the window size for the DCT — that can be varied. Table 4.1 shows the entropy values varying with the level of decomposition in the DWPT. Table 4.2 shows the entropy values varying with window size (in samples) in the

Table 4.1. Entropy values after the DWPT.

level	1	2	3	4	5	6	7	8	9
entropy	3.53	2.63	2.23	2.16	2.27	2.45	2.67	2.84	2.89

Table 4.2. Entropy values after the DCT.

window size	8	16	32	64	128	256	512
entropy	2.16	1.96	1.90	1.92	1.96	2.10	2.26

DCT. Both of the tables show that the entropy values first decrease and then increase. Intu-

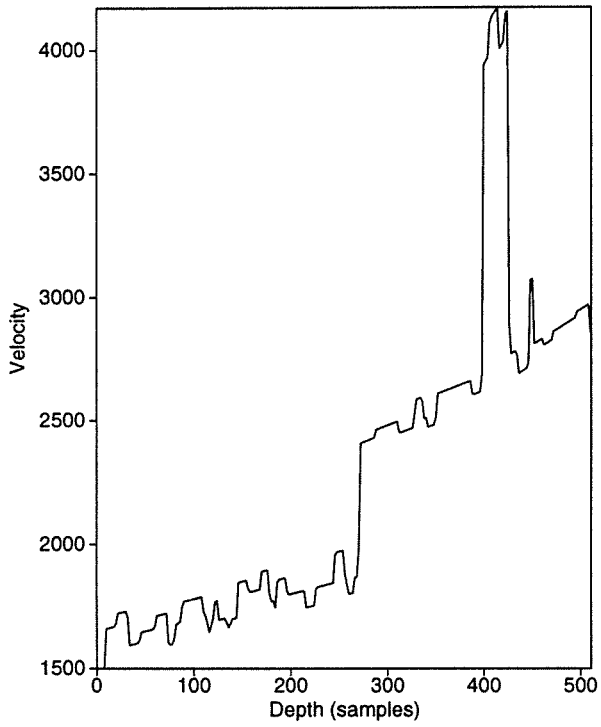

FIG. 4.4. Marmousi velocity model at the first midpoint.

itively, both the DWPT and the DCT employ fixed windows — in the frequency domain for the DWPT, and in the time domain for the DCT — to capture the local spectral information in the data. When using the window size that best matches the general characteristics of the data (perhaps related to the average thickness of the layers, which is about 12.4 samples for this example), the transformed coefficients will have the least entropy. That is why there is an optimal decomposition for both the DWPT and the DCT. Even with the optimal choices, however, the entropy values, 2.16 for the DWPT and 1.90 for the DCT, are still larger than 1.76, after the wavelet transform, where the window size is adapted to the local characteristics of the data — large time windows for the smooth part within each block and small time windows for the boundaries between two blocks.

For the velocity model, an example of the image-type of data, the wavelet transform results in the least entropy. Which transform will do best for typical bandlimited seismic data?

Figure 4.5 shows a shot gather from a land survey. Clearly, it has lots of events and, as opposed to the velocity model, is bandlimited in the time direction. For this gather, again I apply the wavelet transform, the wavelet packet transform and the discrete cosine transform, and compare resulting entropy values. The entropy after the wavelet transform is 6.21 (bits). Clearly, this is much larger than that of the velocity model (1.76 bits), indicating that seismic

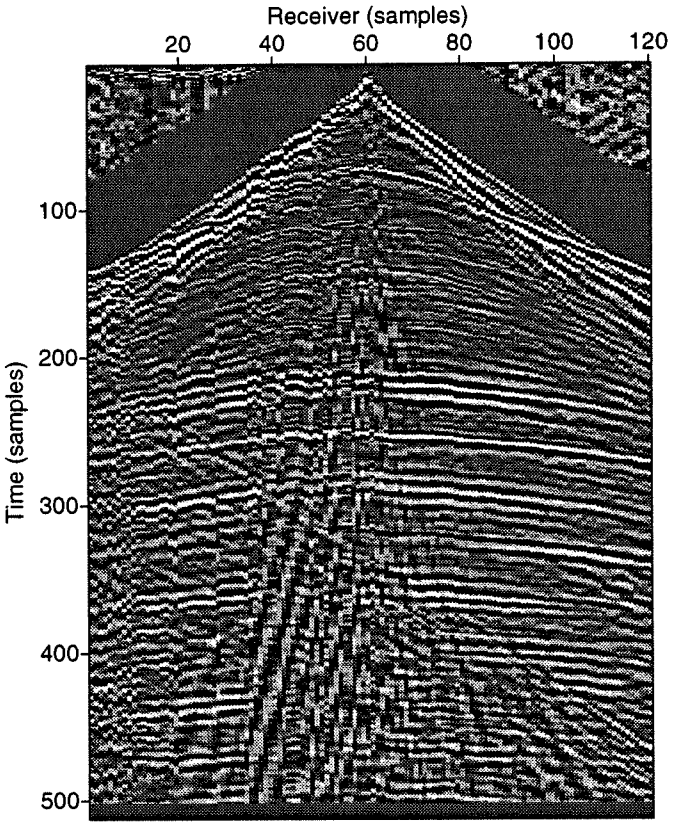

FIG. 4.5. AGC-corrected shot gather.

data are harder to compress than is the velocity model. (This is understandable since the velocity model can be described by only a slope, intercept, and top-of-layer, for each layer.) For the DWPT and the DCT, again I vary the level of decomposition and the window size. Table 4.3 shows the result for the DWPT and Table 4.4 shows the result for the DCT. Similar to the results in Table 4.1 and Table 4.2, there is an optimal choice of the level of decomposition for the DWPT and window size for the DCT. With the optimal choices, the DWPT results in smaller entropy, 6.07, than does the wavelet transform. This indicates that the large frequency window used in the wavelet transform for high-frequency components is too crude for seismic data. The DCT, on the other hand achieves even smaller entropy, 6.00, than does the DWPT. Moreover, the DCT achieves small entropy values for a wide range of window sizes (i.e., its quality is not highly sensitive to window size.). The reason why the DCT results in smaller entropy than does the DWPT is that seismic data are bandlimited and therefore highly oscillatory, as shown in Figure 4.6. The basis function in the DCT, the cosine function, matches the seismic data better than do many of the wavelets. For most types of wavelets, however, the longer the wavelet filter, the more oscillatory the wavelets

Table 4.3. Entropy values after the DWPT.

level	1	2	3	4	5	6	7	8	9
entropy	6.28	6.20	6.07	6.13	6.25	6.38	6.50	6.59	6.63

Table 4.4. Entropy values after the DCT.

window size	8	16	32	64	128	256	512
entropy	6.08	6.00	6.00	6.16	6.25	6.38	6.60

(i.e., the basis functions) become (Daubechies, 1992). A wavelet filter longer than the one used above might have a basis function that is oscillatory and thus might achieve results comparable to that of the DCT.

One might be dubious that the above results might be true only for a certain type of wavelet. Though the above experiments are performed using one of the "Coiflet" filters *Coifman 6* (Wickerhauser, 1994), the results are similar to those when other wavelet filters are used. To see how the choice of wavelet filters influences the result, I use two types of wavelet filters, the "Daubechies" filters and the "Coiflet" filters (Wickerhauser, 1994), with different filter lengths. Table 4.5 shows the entropy values for the shot gather shown in Figure 4.5 after the discrete wavelet transform (DWT) using Daubechies wavelet filters with different filter lengths. Clearly, the entropy value decreases as the filter length increases, slightly except for

Table 4.5. Entropy values after the DWT with "Daubechies" filters.

length	2	4	6	8	10	12	14	16	18	20
entropy	6.40	6.32	6.27	6.23	6.22	6.21	6.21	6.20	6.19	6.19

the first few filters. Intuitively this is because the longer the filter, the steeper the frequency response of the filter, the better the separation of the data in the frequency domain after filtering, and the less correlated the transformed data. The result of a similar experiment using the Coiflet filters is shown in Table 4.6. Again, the entropy value decreases slightly as the filter length increases, due to the same reason as stated above. Comparing the results for different filters with the same filter length, e.g., Daubechies 6 with Coiflet 6, the results are similar. This study shows that neither the type of the filter nor the filter length has much impact on the entropy values, or the compression achievable, if the filter is not too short. Between the two factors, the filter length is a more important choice than the filter type in governing the entropy value after the wavelet transform. Empirically, the longer the filter the

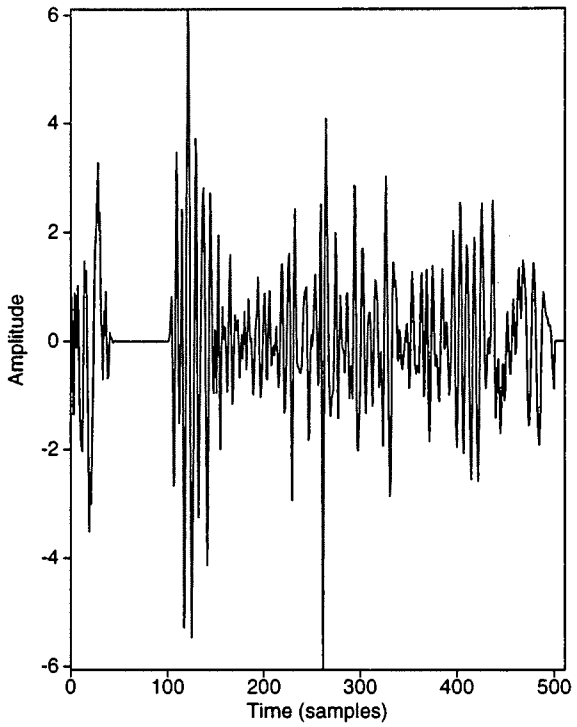

FIG. 4.6. One trace (receiver number one) from the shot gather shown in Figure 4.5.

better. However, the longer the filter, the higher the computational cost of the transform. Therefore, it is reasonable to leave the user to evaluate the trade-off and make the decision.

A similar comparison can be done for the wavelet packet transform. Table 4.7 shows entropy values varying with filter length for Daubechies filters and the level of decomposition in DWPT. Not surprisingly, for a fixed level, the entropy value decreases as the filter length increases, the same as for the wavelet transform. For a fixed filter, the entropy value reaches a minimum for a certain level, similar to the result shown before. Table 4.8 shows the result for the Coiflet filters. The result leads to the same conclusions as that for the Daubechies filters.

As seen from the previous studies, for seismic data, the discrete cosine transform res-

Table 4.6. Entropy values after the DWT with "Coiflet" filters.

length	6	12	18	24	30
entropy	6.29	6.21	6.16	6.14	6.13

Table 4.7. Entropy values after the DWPT with "Daubechies" filters.

length \ level	1	2	3	4	5	6	7	8	9
2	6.41	6.41	6.36	6.43	6.53	6.67	6.74	6.77	6.81
4	6.35	6.34	6.27	6.33	6.43	6.53	6.61	6.68	6.71
6	6.31	6.28	6.18	6.24	6.35	6.46	6.56	6.62	6.66
8	6.29	6.22	6.10	6.16	6.29	6.41	6.53	6.59	6.65
10	6.27	6.18	6.05	6.14	6.26	6.40	6.52	6.57	6.63
12	6.26	6.13	6.01	6.10	6.22	6.37	6.50	6.55	6.61
14	6.26	6.11	6.00	6.09	6.23	6.37	6.51	6.54	6.60
16	6.24	6.07	5.97	6.08	6.20	6.36	6.50	6.53	6.60
18	6.23	6.05	5.97	6.05	6.20	6.38	6.52	6.53	6.61
20	6.23	6.05	5.96	6.05	6.21	6.39	6.53	6.53	6.60

Table 4.8. Entropy values after the DWPT with "Coiflet" filters.

length \ level	1	2	3	4	5	6	7	8	9
6	6.33	6.30	6.21	6.27	6.39	6.51	6.60	6.66	6.70
12	6.28	6.20	6.07	6.13	6.25	6.38	6.50	6.59	6.63
18	6.25	6.09	5.96	6.04	6.16	6.31	6.46	6.56	6.61
24	6.23	6.03	5.90	5.98	6.10	6.27	6.43	6.54	6.60
30	6.22	5.99	5.88	5.96	6.09	6.27	6.43	6.53	6.60

ults in a smaller entropy value than do either the wavelet transform or the wavelet packet transform for many wavelet filters. As shown in Chapter 3, however, the local cosine transform (LCT) which uses a smoother window than does the box function to avoid the abrupt changes, is an alternative for the DCT. For the same shot gather shown in Figure 4.5, I applied the LCT with different window sizes and computed the entropy values. Table 4.9 shows the result. Comparison of Table 4.9 with Table 4.4 shows that the data after the LCT with

Table 4.9. Entropy values after the LCT.

window size	4	8	16	32	64	128	256	512
entropy	5.85	5.83	5.85	5.92	6.08	6.20	6.36	6.59

the best choice of window size have smaller entropy than does that after the DCT, and less than that after the DWPT with most wavelet filters.

In practice, for a representative set of data, one can try a set of window sizes in DCT or a set of levels of decomposition in DWPT, and choose the one that yields the best result. However, from my experience, the window size of 16 samples in DCT or LCT can generally produce reasonably good result for much seismic data. This might be because most seismic data are sampled with a sampling frequency of 500 Hz or 250 Hz, while the typical dominant frequency is about 40 Hz. Therefore, about seven to 13 samples can capture a dominant cycle, so the DCT with a window size of 16 samples can represent the local spectral information of the data.

The above tables show many entropy values, but how much difference is there if the entropy value is 5.8 instead of 6.0? From the numbers, 5.8 is about three percent improvement over 6.0. To have a better understanding of these numbers, consider some of the results shown in Chapter 2. From equation (2.5), the entropy decreases by one bit if a doubled RMS error is allowed in compression. Therefore, we need to specify the amount of error allowed to see the difference in performance of techniques A and B. For example, under one percent RMS error, one technique (A) results in an entropy value of 5.8 and the other (B) results in an entropy value of 6.0. Suppose the original data are represented in floating-point numbers, requiring 32 bits to represent each sample. Therefore, under one percent error, A results in a compression ratio of 5.5 and B results in a compression ratio of 5.3. In terms of compression ratio, A is only about four percent improvement over B. If a different amount of error, say 32 percent RMS error (which is just about 10 percent MSE), is allowed, then A results in an entropy value of 0.8 and B results in an entropy value of 1.0. In terms of compression ratio, A achieves 40:1 compression and B achieves 32:1 compression. In this case, A is 25 percent better than B in compression ratio. Therefore, the entropy value gives a useful indicator for comparison, but, to see the improvement of one technique over another, one needs to specify the amount of error allowed in the compression.

4.3 Data

In the previous sections, I studied how the choice of the compression technique, e.g., the quantizer and the transform, governs the ideal compression that can be achieved. In this section, I study how the different aspects of the data govern the ideal compression.

4.3.1 Frequency content

The frequency content of the data, or the bandwidth, is an important factor that determines the amount of compression achievable. Intuitively, the smaller the bandwidth relative to the Nyquist frequency, the more predictable the data and the more compression achievable. Figure 4.7 shows a trace and its amplitude spectrum from a stacked section that has been deconvolved. It has significant energy for most of the frequency components; that is the bandwidth is a large fraction of the Nyquist frequency. Figure 4.8 shows a trace from another stacked section where no deconvolution was applied, together with its amplitude spectrum. From its amplitude spectrum, it contains mainly low-frequency components and the bandwidth is significantly smaller than the Nyquist frequency. For these two traces, I apply the DCT and compute the entropy values after the transform. We find that the entropy for the trace in Figure 4.7 is 6.00 (bits), while that for the trace in Figure 4.8 is 5.46 (bits). This result supports the intuition that more compression can be achieved for data with smaller bandwidth.

Besides the range of frequency components with significant energy or the bandwidth, the shape of the amplitude spectrum within this range also plays a role in compression. Figure 4.9 shows a synthetic trace generated from random numbers followed by band-pass filtering, together with its frequency spectrum. Comparison of its frequency spectrum, as shown in Figure 4.9, with that shown in Figure 4.8, shows that both have energy up to about 65 Hz. The one shown in Figure 4.9, however, has a flatter spectrum than does the one in Figure 4.8. It turns out that the entropy of this noise trace after the DCT is 5.51 (bits), a little larger than that for the undeconvolved stacked trace, with an entropy of 5.46 (bits), indicating that the flatter the frequency spectrum (i.e., the broader the effective frequency band), the larger the entropy value after applying the DCT to the data, and therefore the less compression achievable.

When the Nyquist frequency is larger than the highest frequency for which there is significant signal energy, one might of course just re-sample the data using a larger sampling interval, to reduce the data storage, or alternatively, apply the DCT and save only the large coefficients. These are, of course, viable approaches to data compression. In compression, however, instead of running the risk either of causing some of the higher frequencies to be aliased or of totally zeroing out all the high-frequency components, we use a few bits to characterize those components. Of course, there are two ways to look at this risk. On the one hand, if one is sure that the high-frequency energy corresponds mainly to noise, then the resampling approach will result in a significant amount of compression without degrading the signal quality. On the other hand, when talking about compression, most of the measures such as the RMS error deal with the difference between the original and the reconstructed

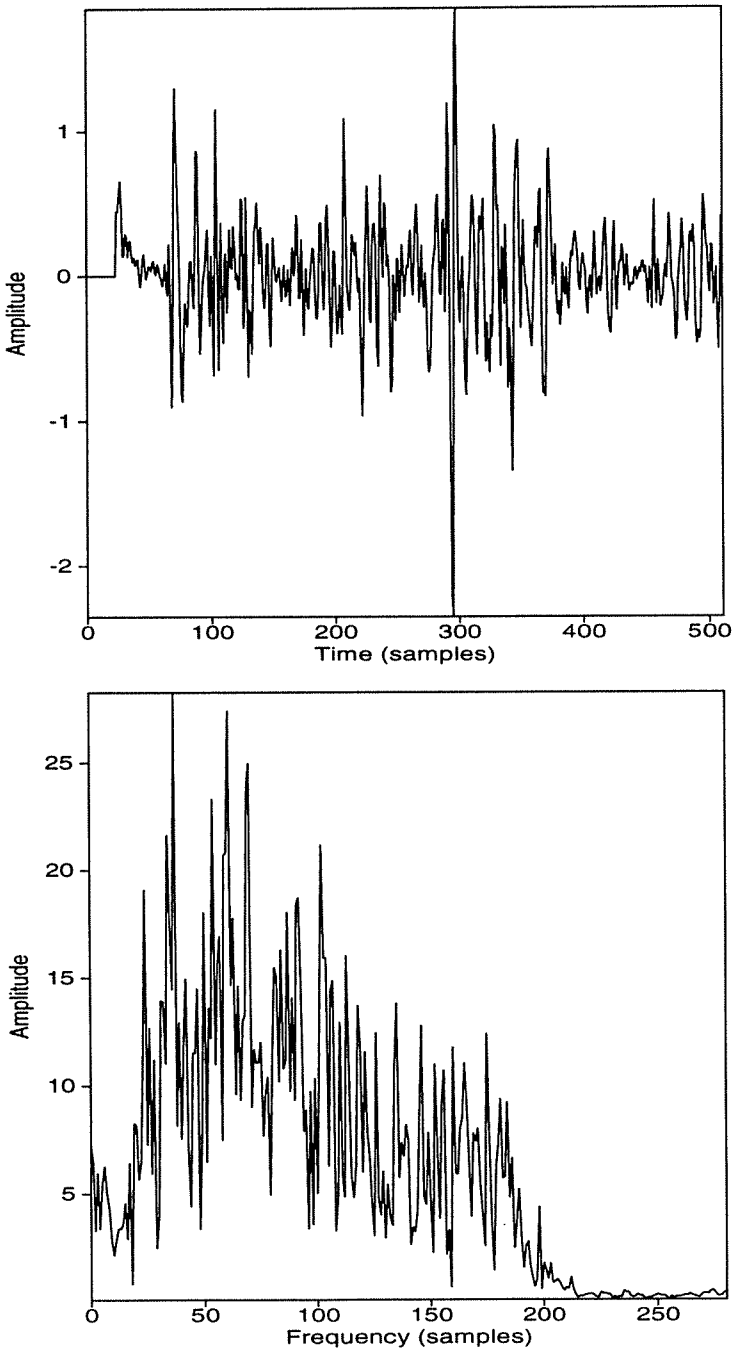

FIG. 4.7. Deconvolved trace.

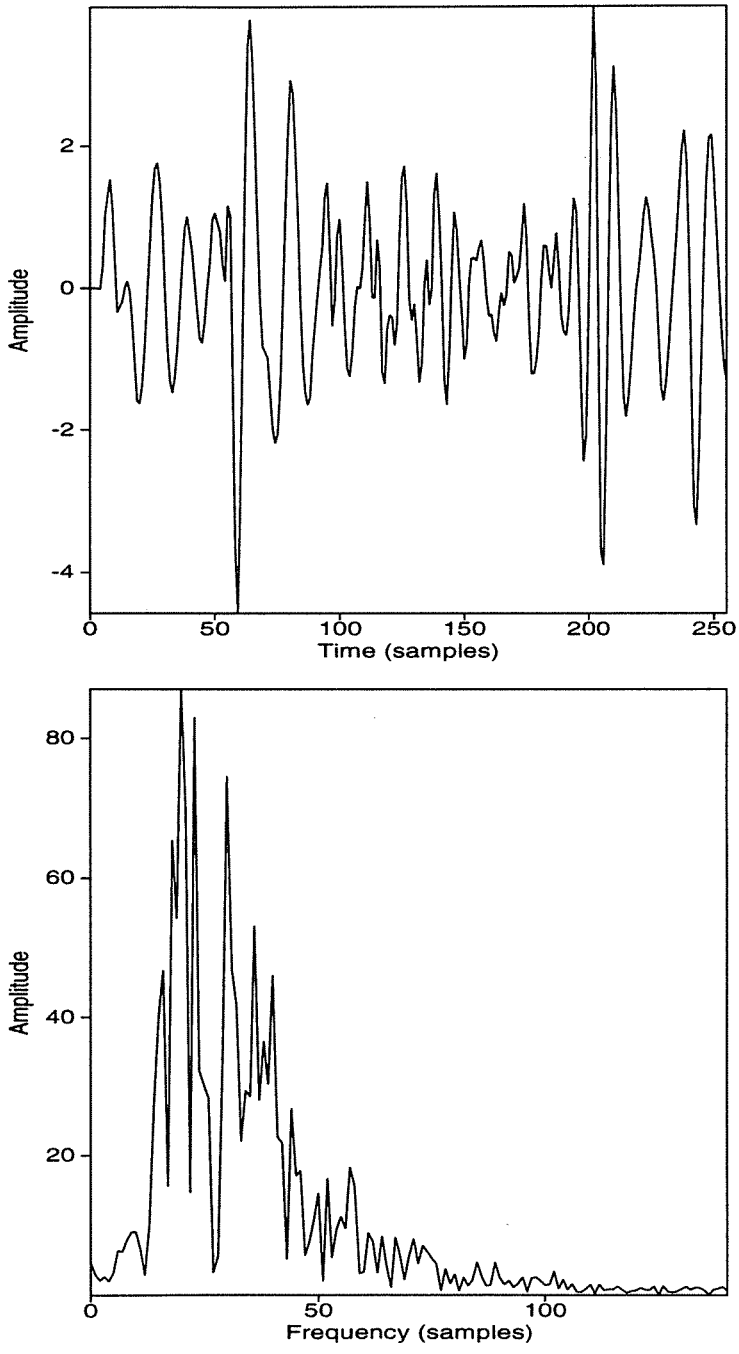

FIG. 4.8. Trace without deconvolution applied.

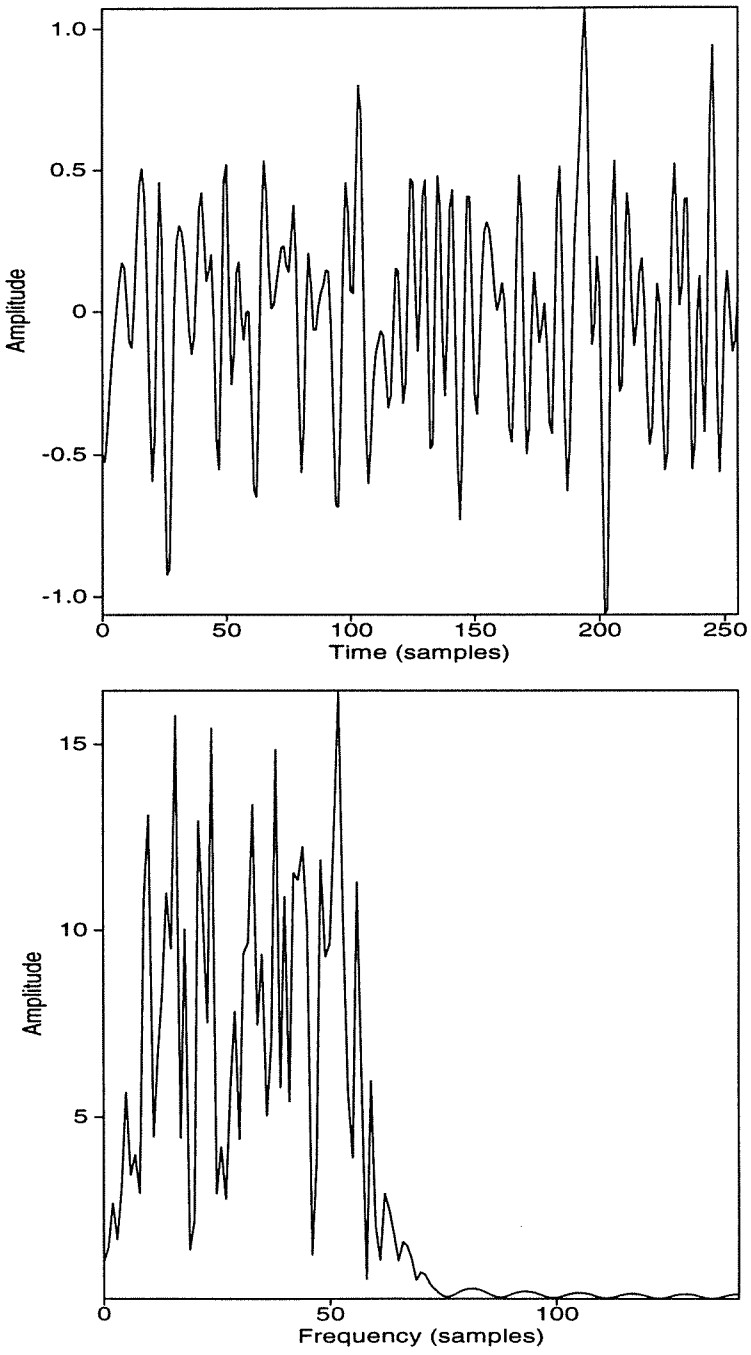

FIG. 4.9. Synthetic trace with bandlimited random noise.

data. When zeroing out the high-frequency components, the difference will generally be large, resulting in a large error as I showed in the example in Section 4.1.

4.3.2 Lateral coherency

For seismic data, which are multi-dimensional, lateral coherency (i.e., predictability) is also important in compression. In the discussion so far, I have talked about only one-dimensional transforms, along the vertical or time dimension, and have not touched on the lateral dimension. However, seismic data have lateral coherency. As a matter of fact, lateral coherency is the key in distinguishing between useful events and background noise in the interpretation of seismic data. The presence of lateral coherency, again, suggests that the data can be represented more compactly in some transform domain than in the original space-time domain.

To deal with the lateral coherency, two-dimensional transforms can be useful. The most straightforward way to generate a two-dimensional transform is to cascade two one-dimensional transforms, i.e., apply one-dimensional transforms successively along the two dimensions separately. To study the lateral coherency in the following, I compare entropy values after one-dimensional and two-dimensional transforms for several data sets.

Figure 4.10 shows a section after brute stack that clearly has a strong component with lateral coherency. For this section, I applied the one-dimensional wavelet transform to each trace along the vertical dimension, as well as the two-dimensional wavelet transform to the two-dimensional data set, and computed the entropy values of the resulting sections. The entropy value after the one-dimensional transform is 6.26, while the one after the two-dimensional transform is 5.77, suggesting that the lateral coherency is exploited by the extra transform along the lateral dimension.

To support this contention, Figure 4.11 shows a synthetic section consisting of a collection of traces generated from random numbers followed by band-pass filtering along the vertical direction. Therefore, the data are bandlimited along the vertical axis but not along the horizontal axis, and no lateral coherency is present in the data. For this section, I carry out the same experiments as for the previous section. The entropy values turn out to be 6.52 after the one-dimensional transform and 6.51 after the two-dimensional transform. The insignificant difference between the results for the one- and two-dimensional transforms results from the lack of lateral coherency in the data. Thus, here, the extra transform along the lateral dimension offers no gain.

Of course, lateral coherency is equivalent to bandlimiting along the horizontal direction. Since each trace in Figure 4.11 is generated independently, there is no lateral coherency in the data. If, however, the data are band-pass filtered along the lateral direction, as shown in Figure 4.12, some lateral coherency is present. For this laterally bandlimited section, I repeat the experiment. The entropy value after the one-dimensional transform is 6.54, and the one after the two-dimensional transform is 5.81.

Perfect lateral coherency, of course, would exist if the section consisted of a repeated trace, as shown in Figure 4.13. For this section, I repeat the experiment. The entropy value after the one-dimensional transform is 6.23, the same as the entropy value of a single trace

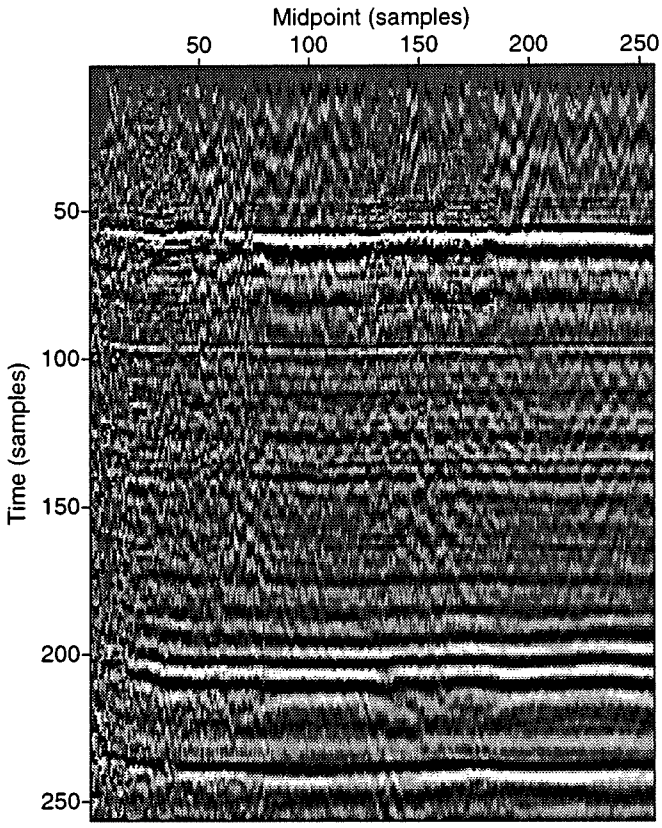

FIG. 4.10. Brute stack.

after the one-dimensional transform, because the two-dimensional section has exactly the same amplitude distribution as the single trace. The entropy value after the two-dimensional transform, however, is a mere 0.22. This is because the information of the entire section is already contained in one single trace and the rest of the traces are redundant.

4.3.3 Signal-to-noise ratio

Consider data contaminated by noise that is independent from one trace to another. Intuitively, the higher the signal-to-noise ratio (SNR), the more coherent the events in the data, and therefore the smaller the entropy value after the transform.

To perform the experiment, I start from a relatively "clean" data set with little background noise. Figure 4.14 shows such data, a stacked section from a marine survey. Though it contains reflections from complex structures, the level of random noise in the background is relatively low. To obtain noisy sections, I add random noise with different levels of SNR

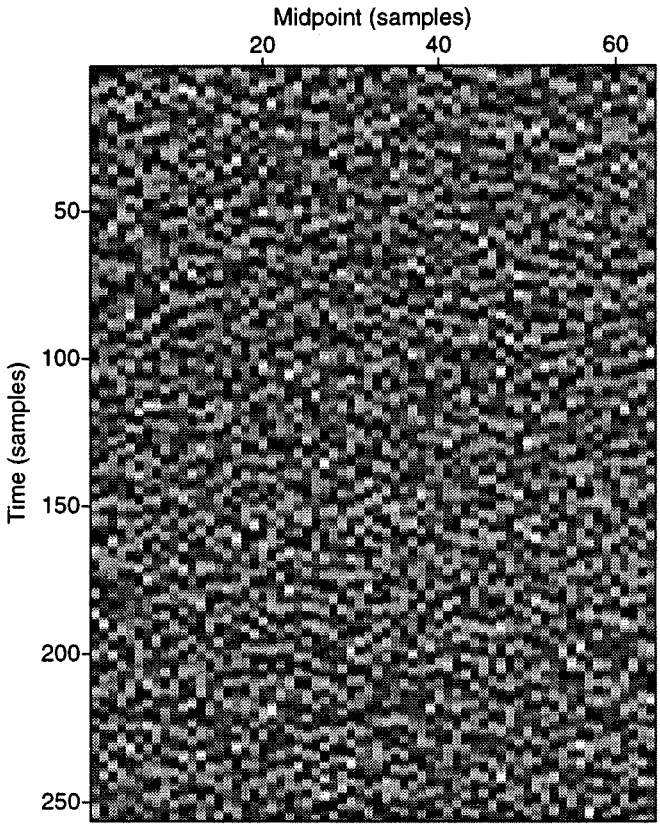

FIG. 4.11. Synthetic section with each trace being independent bandlimited random noise.

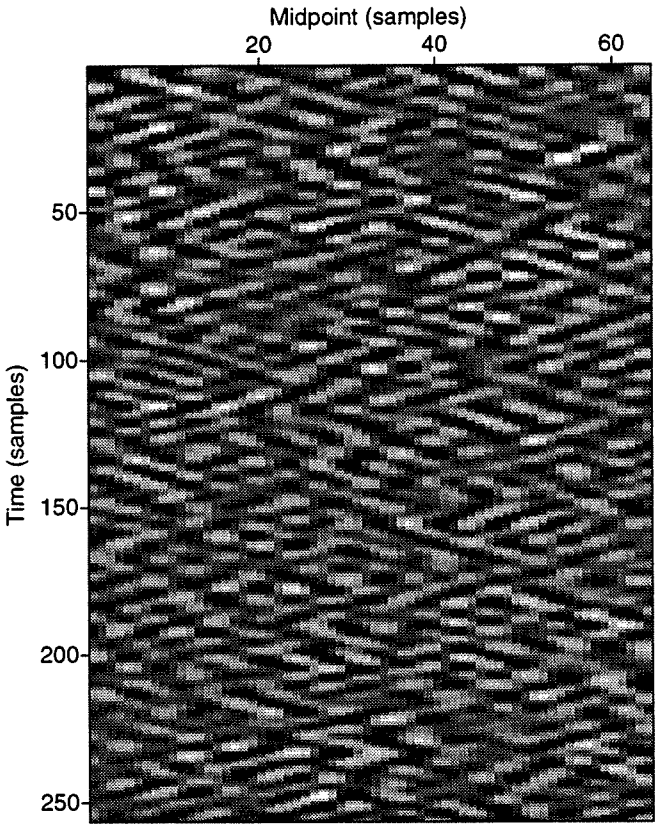

FIG. 4.12. The section shown in Figure 4.11, after applying band-pass filtering along the lateral direction.

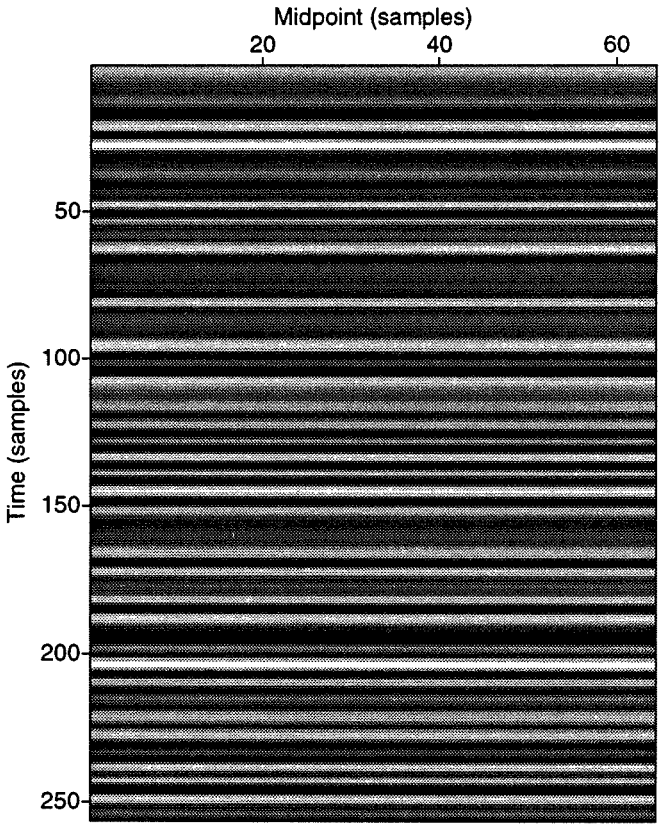

FIG. 4.13. Synthetic section generated by repeating a single trace.

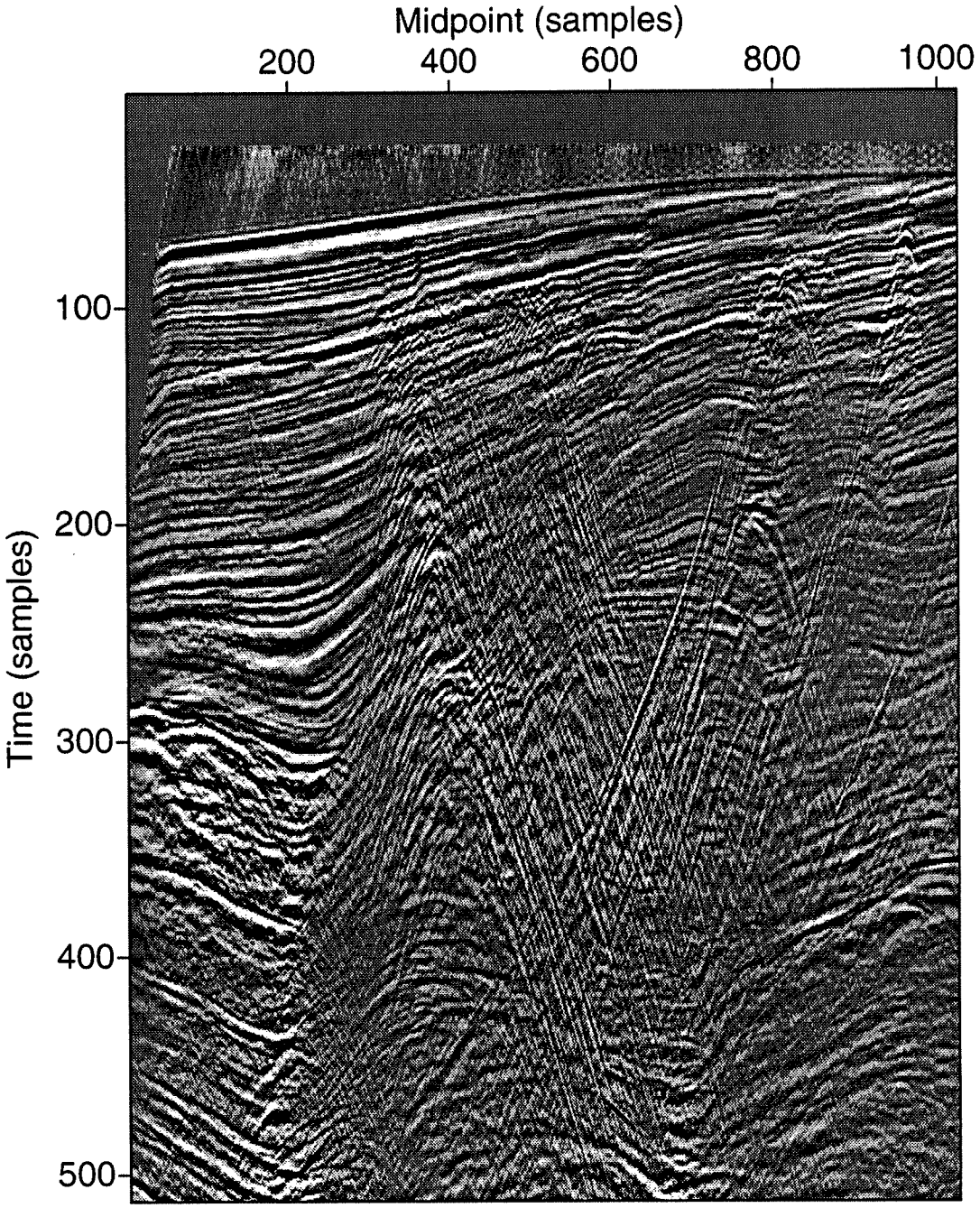

FIG. 4.14. "Clean" data set — stacked section from a marine survey.

to this section, with the SNR defined as

$$SNR = \frac{\max|\text{signal}|/\sqrt{2}}{\text{RMS(added noise)}}, \tag{4.2}$$

where the signal has already been gained. Notice that the added noise is not bandlimited in either the horizontal or vertical dimensions.

Figure 4.15 shows the section with $SNR = 4$. To see how compression relates to SNR, I perform both the one- and two-dimensional wavelet transforms and compute the entropy values for $SNR = 2, 4, 6, 8, 10, 12, 14, 16$ and ∞. $SNR = 2$ is admittedly poor for typical stacked marine data, but the range of SNR tested allows us to draw useful conclusions. Table 4.10 shows entropy values varying with SNR after both the 1D and the 2D transforms.

Table 4.10. Entropy values varying with SNR.

SNR	2	4	6	8	10	12	14	16	18	∞
$1D$	6.90	6.90	6.88	6.86	6.83	6.80	6.78	6.76	6.74	6.55
$2D$	6.89	6.86	6.79	6.71	6.62	6.54	6.47	6.40	6.35	5.70
difference	0.01	0.04	0.09	0.15	0.21	0.26	0.31	0.36	0.39	0.85

From this table, it is evident that after both the 1D and the 2D transforms, entropy decreases as SNR increases, which supports intuition, recalling that entropy is a measure of information content, i.e., unpredictability. Moreover, the difference of the entropy values between the 1D and the 2D transforms increases as the SNR increases. This indicates that the gain in performing the extra transform along the lateral dimension increases as the SNR increases. Conversely, the gain in performing the extra transform along the lateral dimension decreases as the SNR decreases. Therefore an increase in the amount of random noise in data will decrease the coherency in the data along both dimensions, vertical and lateral, assuming that the noise is not bandlimited along either direction.

The presence of random noise, in my opinion, is the major source of difficulty in compressing seismic data. Even though many coherent events can often be seen in data, background noise deteriorates the coherency and, therefore, the amount of compression, under a fixed amount of error. (The human visual system, on the other hand, is intelligent enough to ignore much incoherent noise and recognize coherent events.) Of course, the presence of large-amplitude noise might justify tolerating more error in the compression process, but in that case we must recognize that in doing so, signal is less-well honored than it could have been in a less noisy situation. Also, one should be cautious about tolerating increased compression error merely because the data are already contaminated by noise since much of that noise could be suppressed in subsequent processing, e.g., in CMP stacking.

The study here also gives support to the discussion on compression ratios between

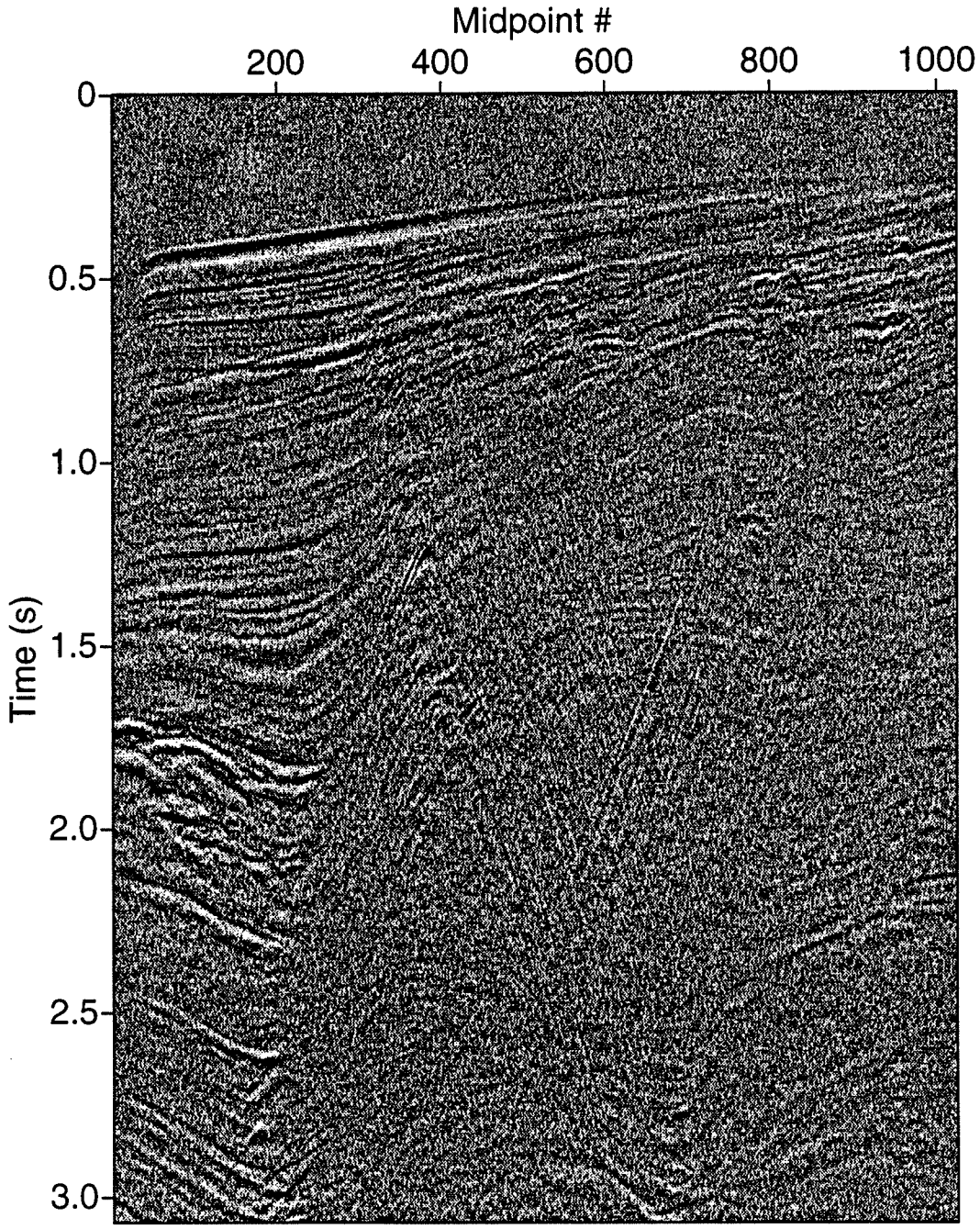

FIG. 4.15. Noisy section with random noise added to the section shown in Figure 4.14 and $SNR = 4$.

stacked sections and shot gathers in Chapter 2. Shot gathers generally have more noise than do the stacked sections. Therefore, there is stronger coherency in the stacked sections, resulting in smaller entropy values for the stacked sections than for shot gathers. This suggests that unstacked data cannot be compressed as effectively as can stacked data. Unfortunately, the far greater economic benefit in compression would be in compression of unstacked data.

4.4 Error measures

In most of the discussion so far, I used the RMS error to measure the difference between the original and the reconstructed data, since the RMS error can be easily related to the average distortion used in the theoretical results. Also, the RMS error is a commonly used quantity in other geophysical applications. In the entropy evaluations, I used one percent RMS error, since this amount of error is generally considered acceptable in seismic processing. As an example, take NMO correction, which is used in virtually all processing flows. In NMO correction, the traces need to be time-shifted. The time shift, possibly by an amount not an integral multiple of the sampling interval, can be represented as a convolution with a sinc-function, defined as

$$\text{sinc}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x}, & \text{if } x \neq 0, \\ 1, & \text{if } x = 0 \end{cases}$$

In most implementations, for practical reasons the sinc-function is truncated to, say, eight samples. This, of course, will introduce errors in the interpolation. The maximal error occurs when time-shifting by half a sample. Through tests, I find that the maximal error is about one percent RMS. Since time shifting is such a common building block in so many processes, the one percent RMS error can be considered generally unavoidable (unless, of course, one chooses to use a longer interpolation operator). Therefore, I used one percent RMS error as the tolerable error for data compression.

Of course, one might argue that one percent RMS error, or even the RMS error measure itself does not make much sense. In this section, I discuss some alternatives.

One alternative to the RMS error is some subjective other than objective measure, e.g., studying the reconstruction to see if there is any perceptible difference between the original and reconstructed data. Although this measure is qualitative, it is the closest to the way in which seismic data are ultimately assessed by the interpreter. A closer view of the error is obtained by subtracting the reconstructed from the original data to generate the difference. Any coherency in the difference section represents error that might be of concern to interpreters. As study of the human visual system shows, the smaller the RMS error in compression, the better "looking" will be a reconstructed image, if the same compression algorithm is used in the comparison.

To relate the amount of RMS error with the perceptible result, I compressed (two-dimensionally) the stacked section shown in Figure 4.14 for different levels of allowed RMS error, reconstructed the data from the compressed data and generated difference sections. Figure 4.16 shows the original, the reconstruction from compression allowing one percent

RMS error and the difference sections. The difference section and subsequent ones are gained up by a certain amount to the same amplitude scale as the original section to see details. For example, in Figure 4.16 the difference is gained up by a factor of 130 and represented by "(X130)". There is hardly any difference between the reconstruction and the original; moreover the weak difference looks random (i.e., there is no obvious pattern in it). Figure 4.17 shows a similar comparison with five percent RMS error. Again, the difference looks random. So are the cases with 10 percent RMS, as shown in Figure 4.18, and even 20 percent RMS, as shown Figure 4.19. Comparing the original and reconstructed data, there is hardly any difference discernible by looking at them, yet high compression ratios (12 and 19) are obtained. Note that as the RMS error level increases, the amplitudes in the difference section also increase, as seen in the reduced gain factor. Those amplitudes, however, remain smaller than the amplitudes of the weaker events in the original data.

When the error is allowed to become even larger, for the case of 30 percent RMS as shown in Figure 4.20, the error starts to show some pattern, with some dipping events appearing in the section, as shown by Figure 4.21 which is a detail of the difference section, even though the reconstruction still looks no different from the original. The comparison for the case of 35 percent RMS error is shown in Figure 4.22. Here, a large compression ratio of 59 is obtained. For this case, the error has stronger amplitudes and shows a stronger pattern, as seen in the detail of the difference section shown in Figure 4.23, and the reconstruction starts to show degradation, seen in the loss of detail from the original to the reconstructed section (Figure 4.24). This study indicates that if for the purpose of displaying and viewing, seismic data can tolerate a large amount of RMS error. What remains to be determined, below, is whether or not data that will subsequently be subjected to various seismic processing can tolerate such a large degree of compression.

Objective error measures other than the RMS error include the L_∞ and PSNR (Peak-Signal-to-Noise-Ratio), defined respectively as

$$L_\infty(e) = \frac{\max_{i=1}^n |e_i|}{\max_{i=1}^n |x_i|}, \tag{4.3}$$

$$PSNR = \frac{\max_{i=1}^n |x_i|}{\sqrt{\frac{1}{n}\sum_{i=1}^n e_i^2}}. \tag{4.4}$$

That is, the L_∞ is the ratio of the maximal absolute values of the difference to that of the original, and the PSNR is the ratio between the maximal absolute value of the original with the RMS of the difference. Different from the RMS error, the L_∞ error measures the maximal error instead of the average error, and therefore gives a good indication of the most severe distortion in the reconstruction. The PSNR, on the other hand, characterizes how well the signal stands out in the compression error. It is a commonly-used measure in image compression (Gersho and Gray, 1992). Here, to use an error measure rather than a signal measure, I define its reciprocal NPSR (Noise-to-Peak-Signal-Ratio)

$$NPSR = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^n e_i^2}}{\max_{i=1}^n |x_i|}. \tag{4.5}$$

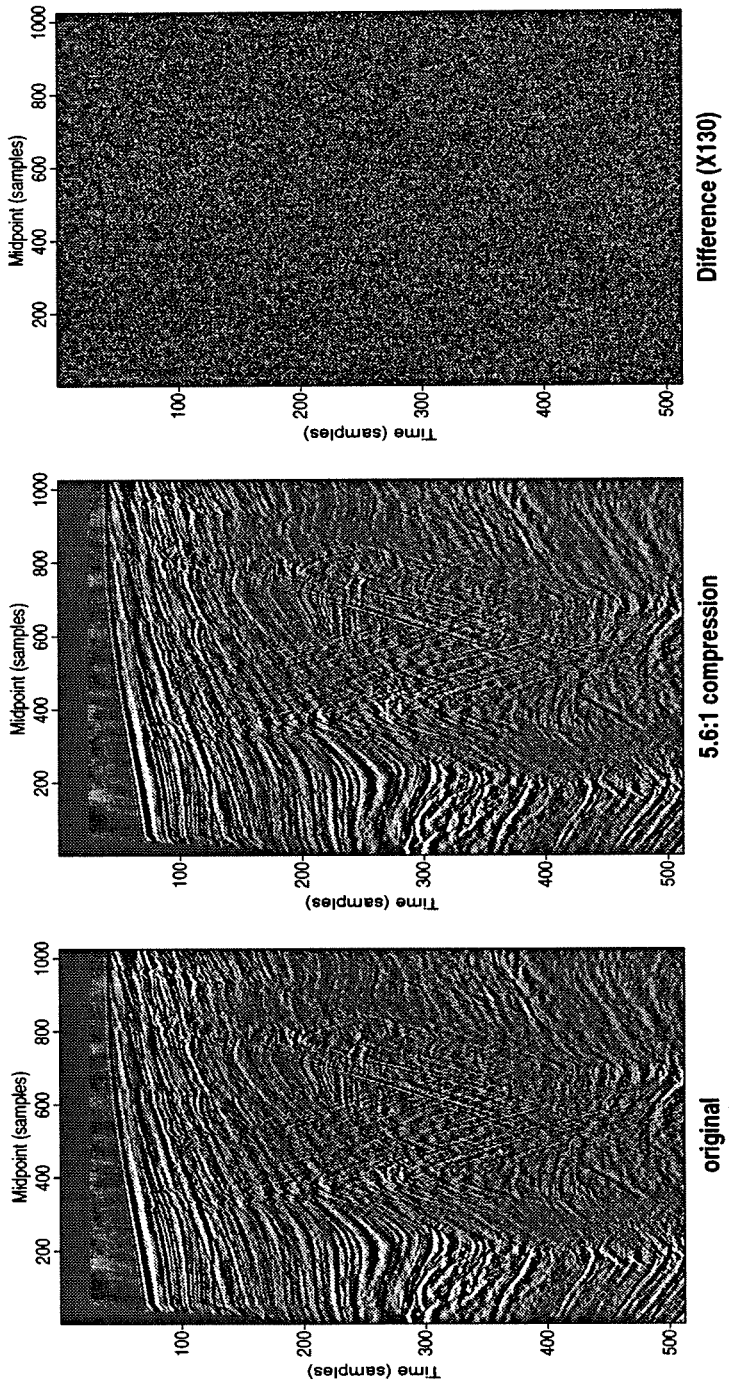

FIG. 4.16. Comparison for one percent RMS error.

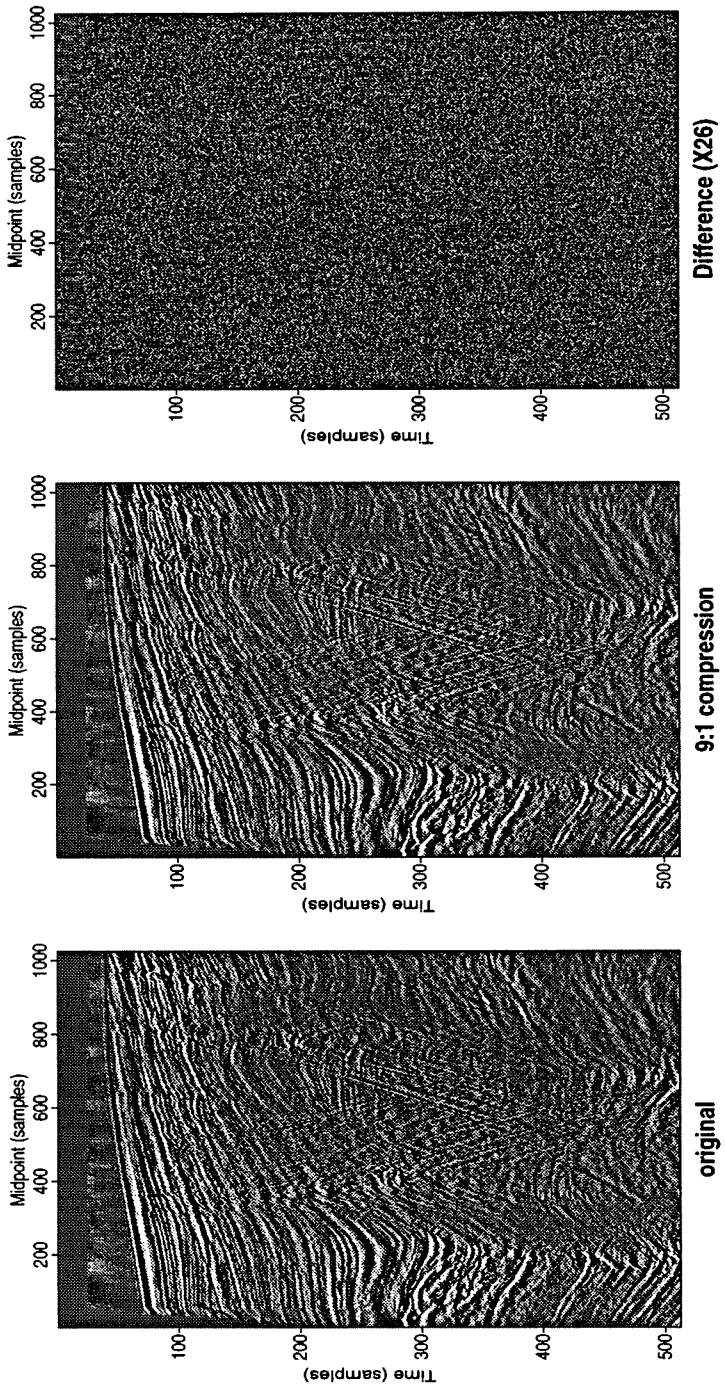

FIG. 4.17. Comparison for five percent RMS error.

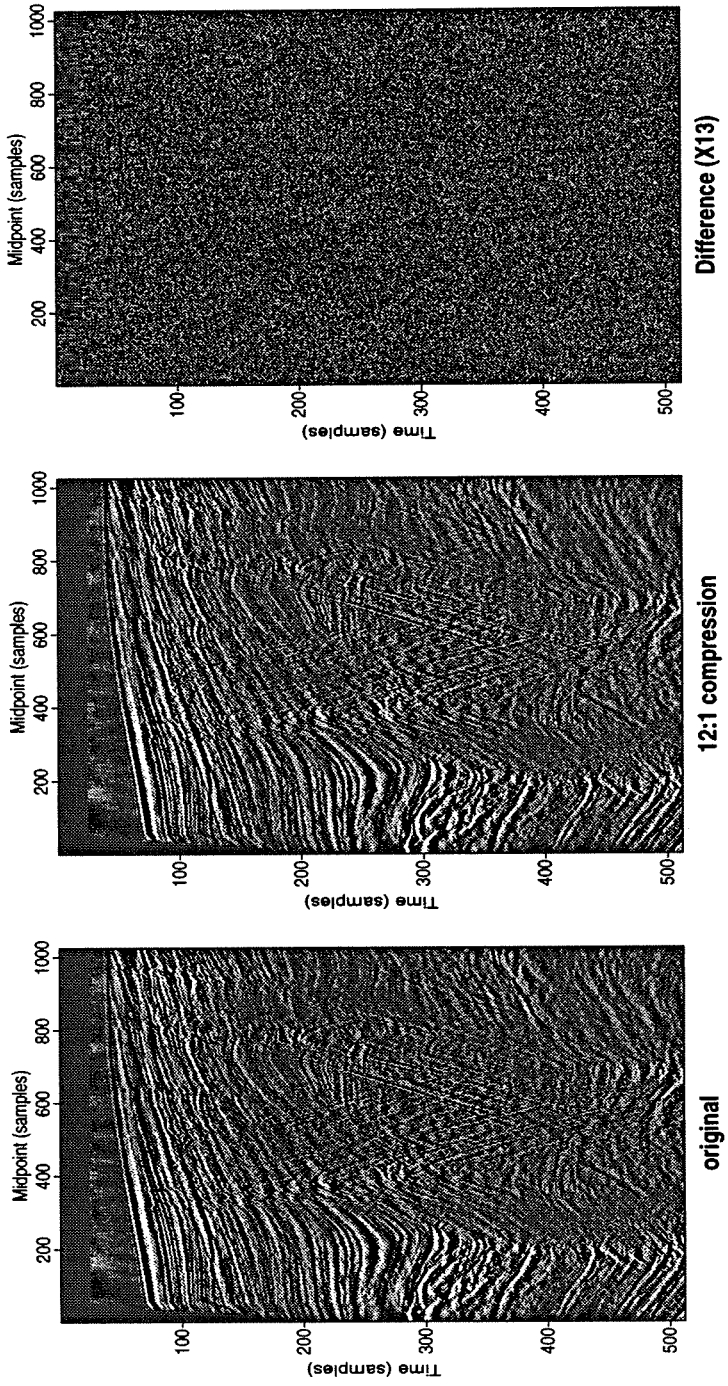

FIG. 4.18. Comparison for 10 percent RMS error.

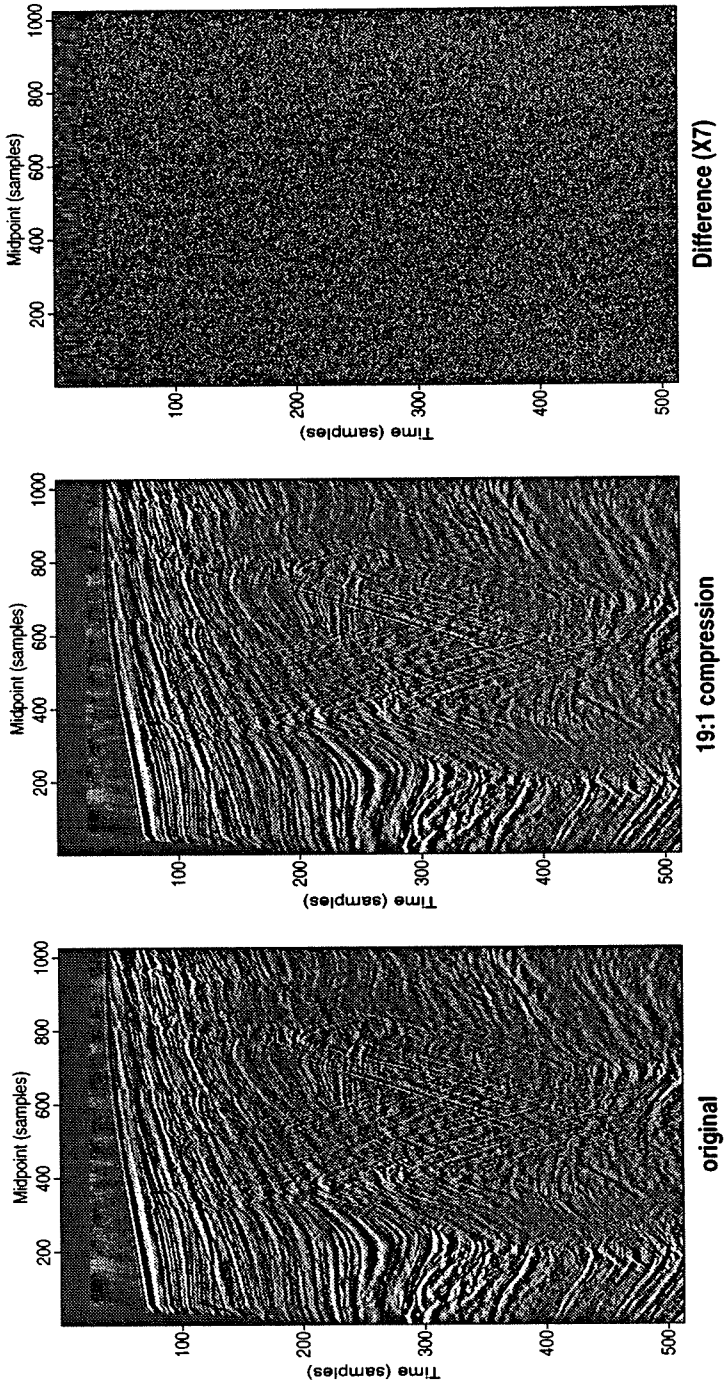

FIG. 4.19. Comparison for 20 percent RMS error.

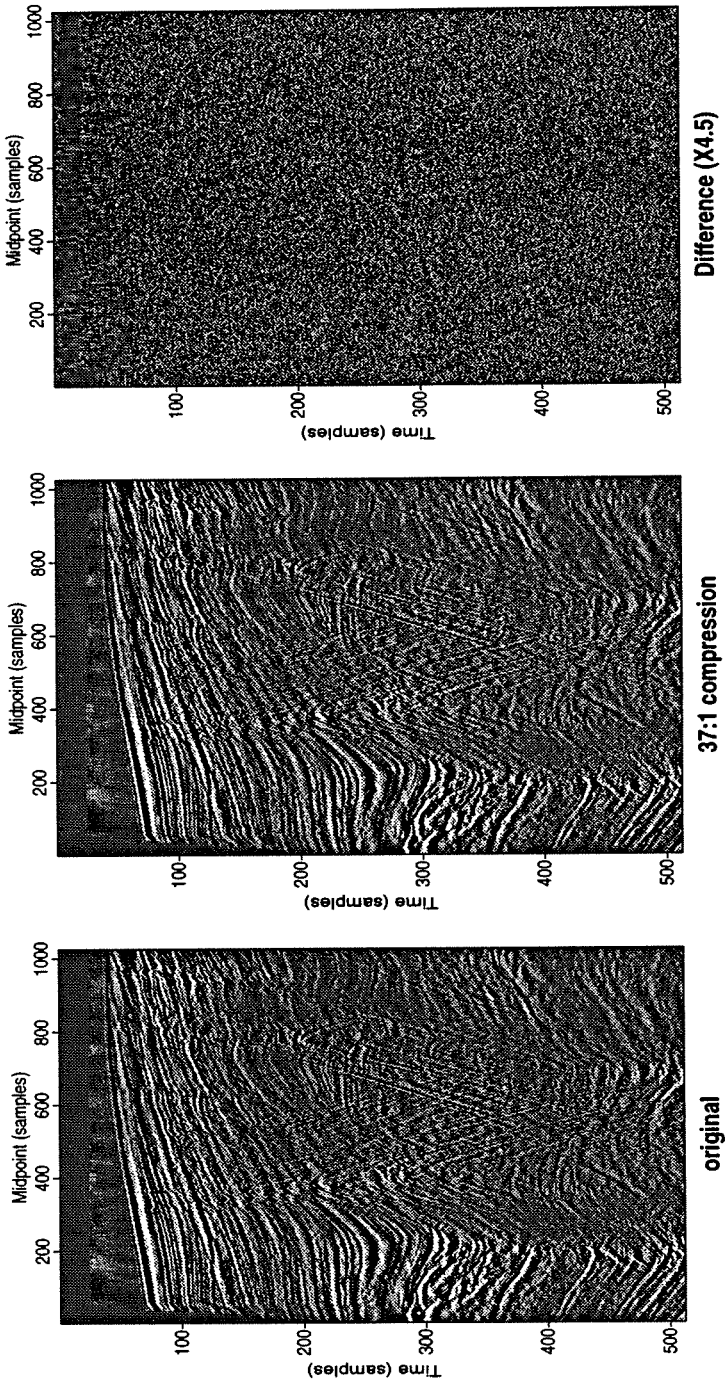

FIG. 4.20. Comparison for 30 percent RMS error.

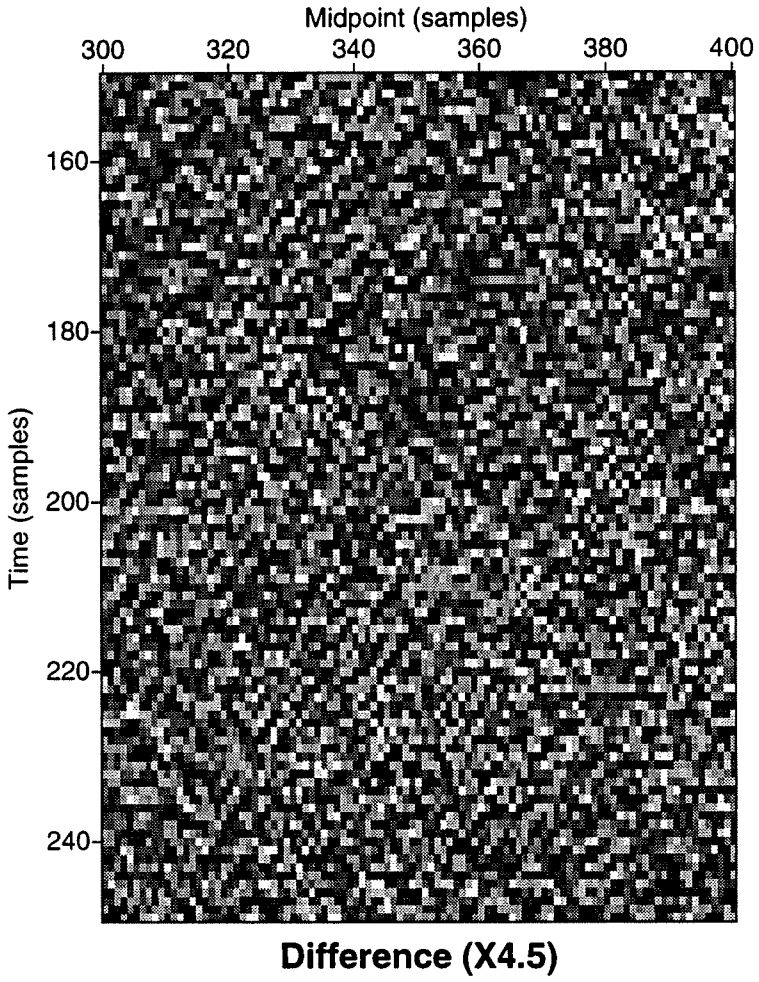

FIG. 4.21. Detail of the difference section with 37:1 compression.

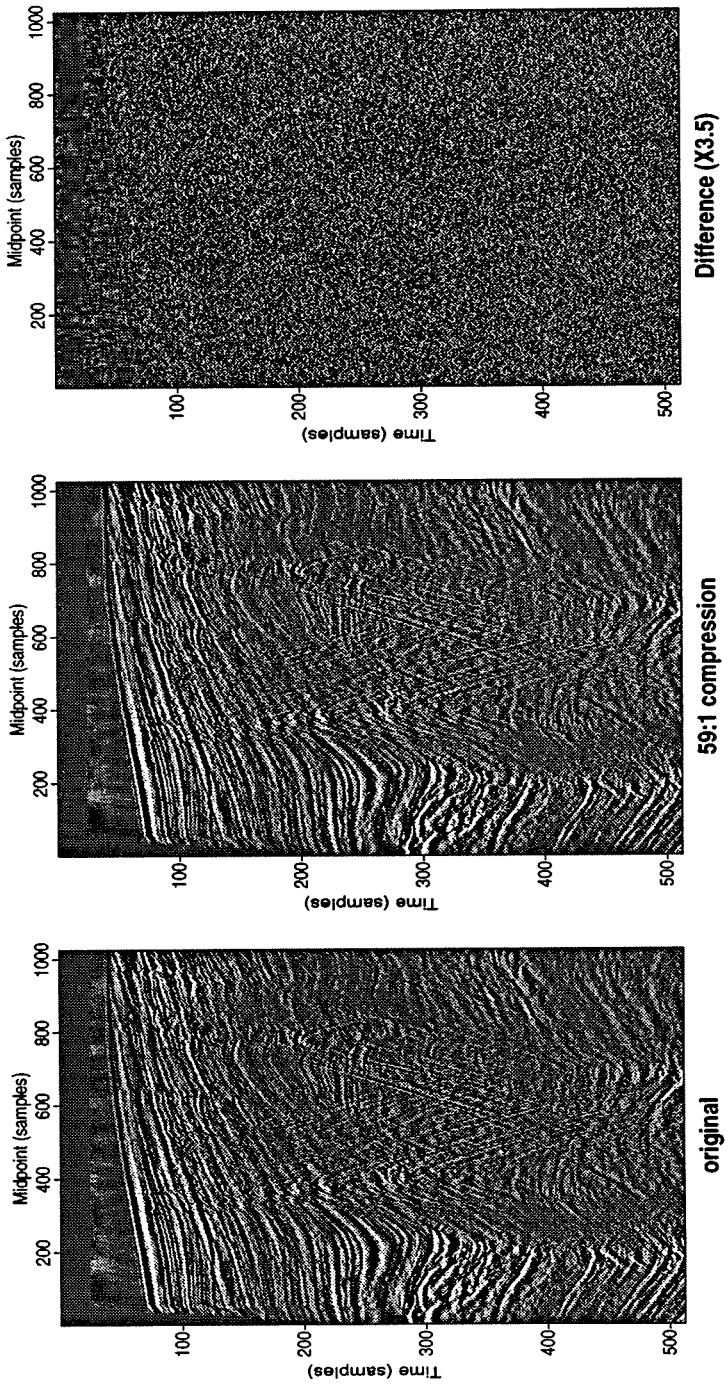

FIG. 4.22. Comparison for 35 percent RMS error.

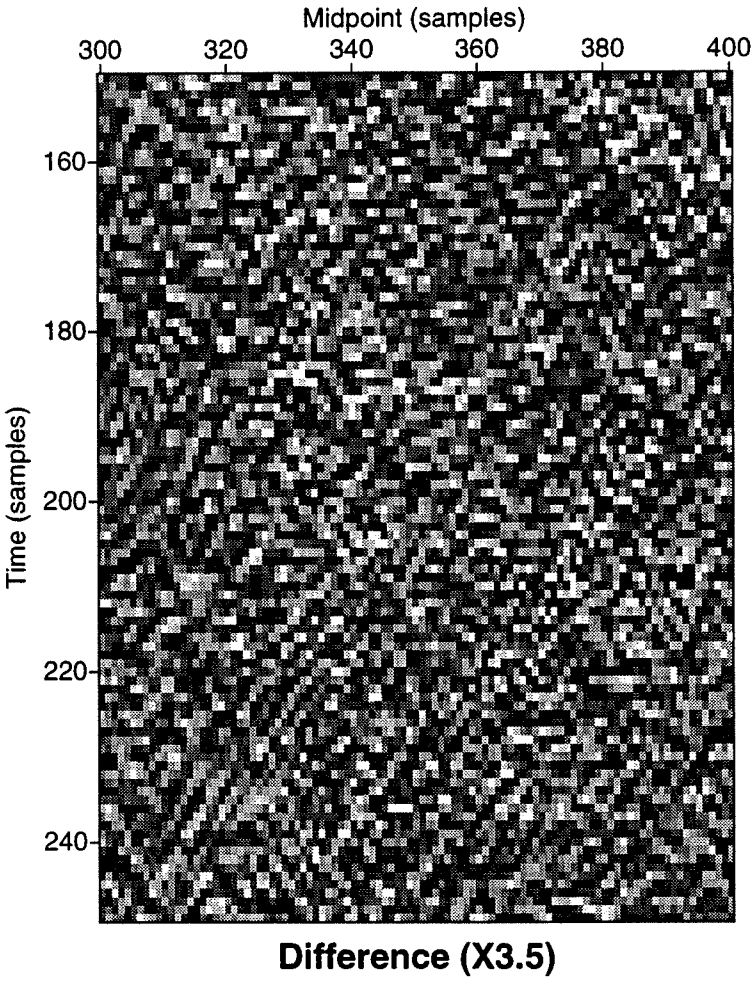

FIG. 4.23. Detail of the difference section with 59:1 compression.

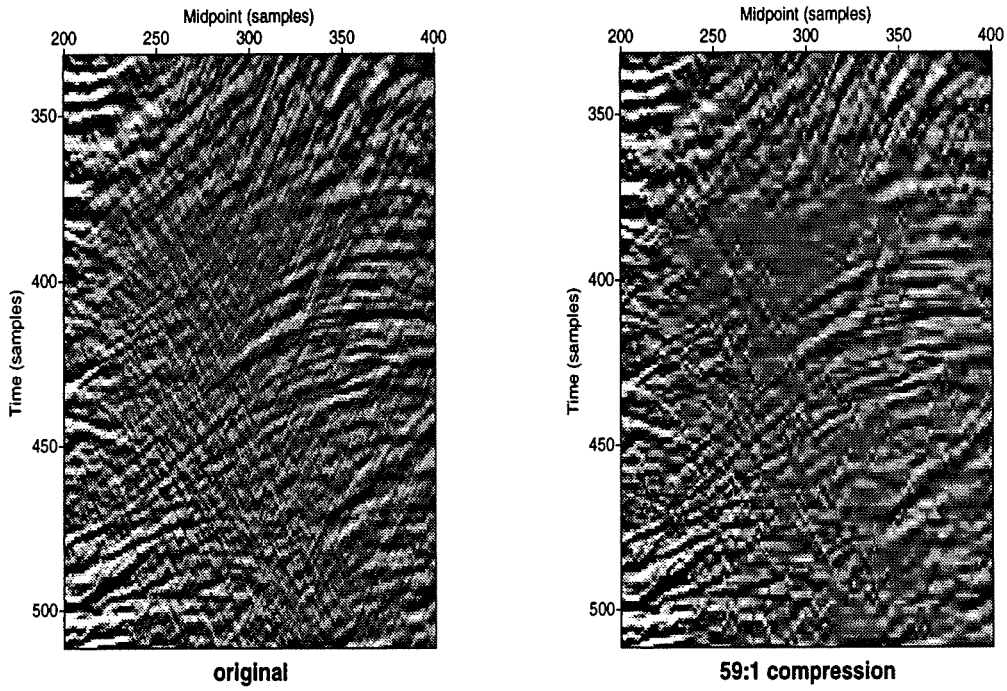

FIG. 4.24. A comparison of a detail of the original section and the reconstructed section from 59:1 compression.

For a comparison among these errors and the RMS error, I again compress the stacked section shown in Figure 4.14 when different RMS errors are allowed, reconstruct from the compressed data, generate the difference sections, and compute these errors. The result is shown in Table 4.11. As seen from the table, the L_∞ error is less than half the RMS error,

Table 4.11. Various error measures.

comp ratio	5.6	9	12	19	37	59
$RMS\%$	1	5	10	20	30	35
$L_\infty\%$	.4	2	4	7	13	17
$NPSR\%$	.08	.4	.8	1.5	2.5	3.1

for all the compression ratios tested. This phenomenon can be explained by the different distributions of the error and the data. The data have a relatively large number of very small amplitude samples, as shown in Figure 4.25, which looks like a Laplacian distribution. Therefore, the RMS amplitude is much smaller than the maximal amplitude for the data. For

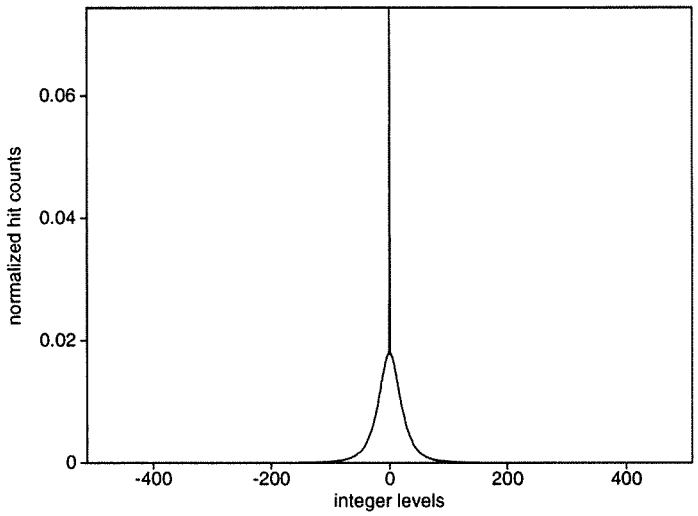

FIG. 4.25. Amplitude distribution of the data shown in Figure 4.14.

the example shown in Figure 4.25, the RMS amplitude of the signal is less than nine percent of the maximal amplitude of the signal. For the compression error or the difference, on the other hand, the sample values are distributed differently, somewhat close to each other, as shown in Figure 4.26, which looks like a Gaussian distribution. As a result, the RMS error is more than 20 percent of the maximal error, for all the compression ratios tested. Due to

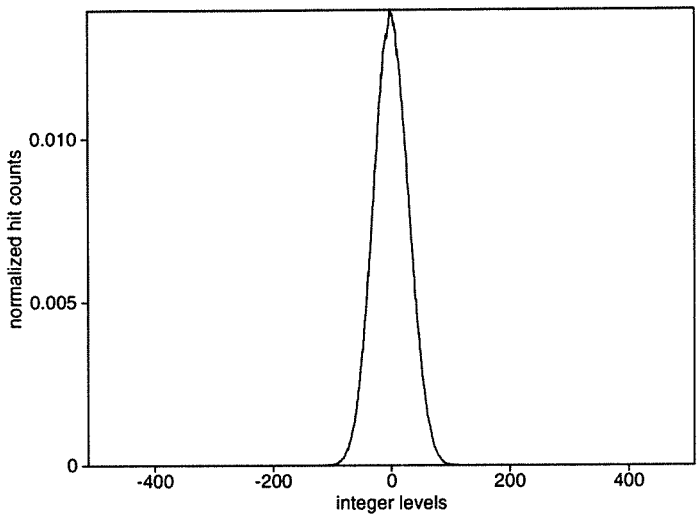

FIG. 4.26. Amplitude distribution of the difference shown in Figure 4.16

the different RMS-to-maximal ratios between the signal and the error, the relative RMS and the relative maximal errors differ. For the same reason that the RMS amplitude of the data is much smaller than the maximal amplitude of the data, the NPSR is much smaller than the RMS error, as shown in Table 4.11.

Chapter 5

RANDOM TRACE ACCESSING

In the previous chapters, I discussed several aspects of the compression of seismic data that mainly govern the amount of compression achievable. In this chapter, I discuss another aspect related to the ease in manipulating the compressed data.

In most applications of seismic data compression, to reduce either the storage cost or the network traffic, we might want to manipulate the compressed data directly. (For example, we might want to sort the compressed data directly into different gathers, without uncompressing them first.) Since many processes that seismic data undergo are trace-based, the random accessing of each trace in the compressed data is thus desirable. For example, suppose we have a 3D post-stack data volume. We can compress the data in any of several ways. One way is to compress each line separately using the 2D version of one of the transforms. In another way, we might compress the entire volume using the 3D version of one of the transforms. For the former, the compressed data consist of a collection of compressed seismic lines, while for the latter, the compressed data constitute just a single object, the compressed volume.

Now suppose we want to display one line of this 3D volume. If the data are compressed by a 2D compression technique, then we can select the line we want, decompress this line (only this line) and display it. If, on the other hand, the data are compressed by a 3D compression technique, then we have to first decompress the entire data volume and then select the line we want for display. Obviously, the former requires less computation than does the latter. If, instead of one line, we wished to display a cross-line section of this 3D volume, then for both of the methods, we have to decompress the entire data volume before we can select the cross-line section. Moreover, if we want to process a single trace, then we have to decompress one line containing this trace if the 2D compression technique is used, and the entire volume if the 3D compression technique is used. This simple example shows that the data after the 2D compression of each line or the 3D compression of the entire volume are not easy to manipulate for trace-based processes.

To have random trace accessing, the most straightforward technique is to compress each trace separately, i.e., to apply the 1D compression for each trace. This approach, however, fails to take advantage of the lateral (trace-to-trace) coherency, resulting in limitation of compression achievable, as indicated by the entropy values in Chapter 4. Moreover, in addition to this degradation in compression performance, there are some practical issues involved in trace-by-trace compression that deserve some discussion in the next section.

5.1 Single-trace compression

To compress each trace separately, we first apply a one-dimensional transform to each trace, quantize each transformed trace by, say, a uniform quantizer, and then entropy encode

each quantized trace. Generally, the average number of bits per sample after the entropy coder exceeds the entropy value of each trace. One cause of the difference is that some side information is required to decode the compressed data. In the Huffman coding for example, we need to store the table (or the so-called *codebook*) that translates the symbols into the Huffman codes required to decode the compressed data. Other entropy coding techniques, such as the adaptive or the dictionary-based techniques, though they do not explicitly require the codebook, generally suffer from some poorly-performing "start-up" period, only after which can optimal compression be achieved (e.g. Gersho and Gray, 1992). In most situations, where the sequence to be encoded is long (e.g., having 10000 samples), the overhead is spread out over all samples and is not significant. Unfortunately, this is not the case for most seismic traces. Typically, a seismic trace consists of about 1000 to 3000 samples. For so short a sequence, the overhead becomes significant. For example, for the trace shown in Figure 5.1, the entropy value after the DWPT is 6.10 under the criterion of one percent RMS error. Using the same quantization, the Huffman compressed sequence requires about 8.92 bits on the average for each sample, considerably larger than the ideal entropy, due to the coding overhead. As a matter of fact, the overhead is too large to make the Huffman coding

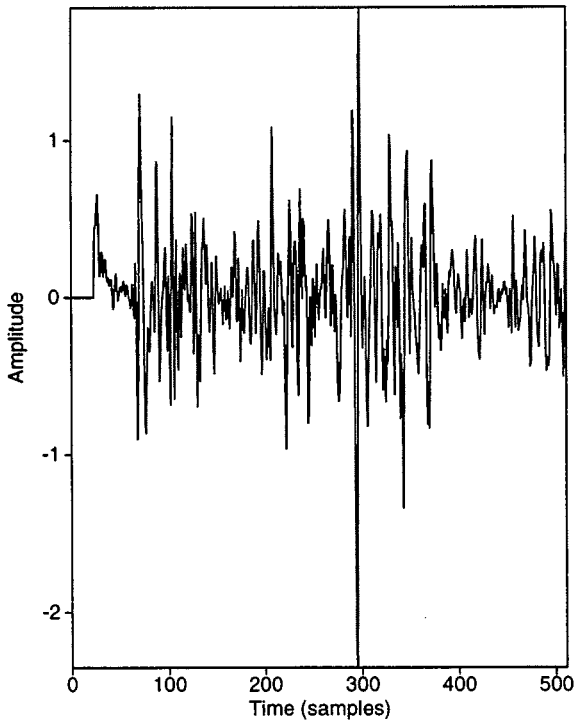

FIG. 5.1. A single trace containing 512 samples.

even worthwhile. For example, we could transform the trace and then simply quantize the

transformed values without any special encoding. For the same amount of error (one percent RMS error) as with the Huffman coding, this simplistic approach would result in about 8 bits. The overhead in Huffman coding, however, is much smaller for two-dimensional data. For the example shown in Figure 4.14, after the two-dimensional transform the entropy value is 5.11 while the Huffman compressed sequence requires about 5.29 bits per sample on the average.

To efficiently encode short sequences, I adopt an alternative variable-length coding, the variable-length integral coding technique (e.g., Wallace, 1991). As with Huffman encoding, in this technique, the integer numbers are represented using different numbers of bits according to some empirical statistics of the sequence, rather than using a fixed number of bits (thus the name variable-length integral coding). The difference is that Huffman coding is adapted to the input sequence, and therefore needs to store the statistics of the input sequence, while here this overhead is avoided, although since the coding is not adapted to the input sequence, it is sub-optimal. Generally speaking, for seismic data small numbers occur more often than do large numbers; therefore, small numbers should be represented using fewer bits than should large numbers to make the average number of bits small. In some sense, it is an entropy coder, that uses a pre-defined codebook instead of a codebook that perfectly matches the statistics of the input sequence. Table 5.1 shows an example of this coding technique. It is similar to the one used in the baseline implementation of the JPEG standard (Wallace, 1991). Basically, it works like this. Every number is coded as two parts, the size part and the amplitude part. The size indicates how many bits are needed to code the amplitude. The amplitude is just the number itself. Both of the two parts are coded and the codes are concatenated. As seen from the table, each code consists of two parts: the first two bits

Table 5.1. An example of variable-length integral coding.

size	amplitude	code
1	-1, 1	000, 001
2	-3, -2, 2, 3	0100, 0101, 0110, 0111
3	-7, ..., -4, 4, ..., 7	10000, ..., 10011, 10100, ..., 10111
4	-15, ..., -8, 8, ..., 15	110000, ..., 110111, 111000, ..., 111111

represent the size since we have four possible sizes here, and the rest of the bits represent the amplitude. In doing this, smaller numbers are coded using shorter codes. For example, the number -1 is coded using only three bits as 000, while the number -15 is coded using six bits as 110000. If no such technique is applied and all the numbers are coded using the fixed length codes, then both of the two numbers -1 and -15 will need five bits since overall there are 30 numbers from -15 to 15 (zero is not included here).

In single-trace compression, I use a similar but slightly different approach. Instead of one table, during the compression of each trace, I may use a set of tables where each table is a variable-length integral codebook. The tables differ from one another so as to provide some

flexibility in matching the statistics of the input sequence. Therefore, different code books can be used in compressing different traces as well as different parts of one trace, depending on the range of sample amplitudes within each part. In this way, some adaptiveness remains in the coding.

Table 5.2 and Table 5.3 are two example codebooks used in single-trace compression.

Table 5.2. One codebook for single-trace compression.

integer number	code
0	0
1	10
2	11

Table 5.3. Another codebook for single-trace compression.

integer number	code
0	00
1	01
2	10
3	110
4	111

Comparing the two tables, note that Table 5.2 can be used only when the sequence contains only three symbols $0, 1, 2$, whereas Table 5.3 can be used for sequences containing five symbols $0, 1, 2, 3, 4$. Obviously, Table 5.3 can also be used instead of Table 5.2 when a sequence contains only three symbols $0, 1, 2$. However, it is clear that the codebook in Table 5.2 will out-perform that in Table 5.3 in this case, since the number 0 is coded as 0 in Table 5.2 while it is coded as 00 in Table 5.3. This is why I use a set of codebooks with each one somewhat tuned to the symbols in a sequence. Of course, this approach requires first testing the symbols in a sequence. However, if care is taken during the quantization step, this testing will not be necessary.

For the example of the single seismic trace shown in Figure 5.1, only 6.11 bits per sample on the average (including overhead) is required with this coding technique. This result is very close to the entropy value of 6.10, indicating that the empirical codebook I use is close to optimal and that the compression is efficient.

5.2 Lateral prediction

By compressing trace-by-trace, we can access each trace in the compressed data at random. However, the amount of compression achieved would be less than that after the two-dimensional compression, as indicated by the entropy evaluations in Chapter 4. This is due to the fact that seismic data are multi-dimensional and have trace-to-trace coherency, which is not exploited in the trace-by-trace compression.

One way to utilize this lateral coherency while maintaining some flexibility in trace accessing is to divide the data into strips and apply some compression to each strip. In doing this, a chosen trace can be retrieved by decompressing only the strip that contains this trace. At the same time, by compressing the strips, the lateral coherency within each strip is taken into account.

To compress each strip, we can, of course, apply the two-dimensional compression technique. This is just what we discussed above before, but now for data sets that are smaller than the original one. Here, I present and analyze a new technique based on lateral prediction.

The motivation for this approach is the following. Most of the events in seismic data can be approximated as linear, at least locally. Therefore, the data within each strip consist mainly of linear events. After performing a linear prediction along the lateral direction, hopefully, the data can be decomposed into the laterally coherent (predicted) part and the non-coherent (unpredicted) part. If the prediction is good, lateral prediction therefore provides a separation of the coherent and non-coherent events. This, among other things, provides the possibility of designing a quantizer with more error allocated to the unpredicted part and less error to the predicted part. I will leave the comparison between this approach and the 2D transform approach (also on small strips) to later sections. First, I discuss how the prediction approach works.

5.2.1 Modeling

Since within a small region, most seismic events can be considered as approximately linear, a small block of seismic section can be approximated as consisting of several events with different slopes. In lateral prediction, we use one trace (or several traces) as a "pivot" that, coupled with an appropriately designed filter, generates all (or most) of the events in this block. The idea here is similar to one used in finding missing data in Claerbout (1992), but here it is applied to data compression.

First, let us model the linear events. Suppose the section is the recorded wave field. (Although this is not true if the section is a migrated section representing subsurface structure, the derivation stands.) For the simplest case where there is only one slope, the wave field contains just a single plane wave, i.e., $u(t,x) = u(t-px)$, which satisfies the following equation

$$\frac{\partial u}{\partial x} + p\frac{\partial u}{\partial t} = 0. \tag{5.1}$$

In the discrete form (i.e., recorded data are discretely sampled), this equation can be repres-

ented as

$$\frac{u(t,x+\Delta x)-u(t,x)}{\Delta x}+p\frac{u(t+\Delta t,x)-u(t,x)}{\Delta t}=0, \tag{5.2}$$

or equivalently as

$$u(t,x+\Delta x)-(1+q)u(t,x)+qu(t+\Delta t,x)=0, \tag{5.3}$$

where $q=p\frac{\Delta x}{\Delta t}$. This can be represented as a difference star shown in Table 5.4. If a higher-order approximation is applied on $\frac{\partial}{\partial t}$, then the filter can be represented as shown in Table 5.5, where the x's represent nonzero values and the .'s represent zeroes. Clearly, one column of

Table 5.4. First-order filter for a single slope.

	x	$x+\Delta x$
t	$-1-q$	1
$t+\Delta t$	q	0

Table 5.5. High-order filter for a single slope.

x	.
x	.
x	1
x	.
x	.

coefficients (the x's) is needed to predict a single slope.

If two, instead of one, slopes are present in the section, the wave field can then be written as $u(t,x)=u_1(t-p_1x)+u_2(t-p_2x)$. Since

$$\frac{\partial}{\partial x}=-p_1u_1'-p_2u_2',\ \frac{\partial}{\partial t}=u_1'+u_2', \tag{5.4}$$

then

$$(\frac{\partial}{\partial x}+p_2\frac{\partial}{\partial t})u(t,x)=(p_2-p_1)u_1'. \tag{5.5}$$

Therefore $u(t,x)$ satisfies

$$(\frac{\partial}{\partial x}+p_1\frac{\partial}{\partial t})(\frac{\partial}{\partial x}+p_2\frac{\partial}{\partial t})u(t,x)=0, \tag{5.6}$$

which is equivalent to

$$\left(\frac{\partial^2}{\partial x^2} + a\frac{\partial^2}{\partial x \partial t} + b\frac{\partial^2}{\partial t^2}\right) u(t,x) = 0, \tag{5.7}$$

where $a = p_1 + p_2$ and $b = p_1 p_2$. Again in the discrete form, this can be represented as

$$\begin{aligned}\frac{u(t,x+\Delta x) - 2u(t,x) + u(t,x-\Delta x)}{\Delta x^2} + a\frac{u(t,x) - u(t-\Delta t, x-\Delta x)}{\Delta x \Delta t} \\ +b\frac{u(t+\Delta t,x) - 2u(t,x) + u(t-\Delta t,x)}{\Delta t^2} = 0,\end{aligned} \tag{5.8}$$

which is equivalent to

$$\begin{aligned}u(t,x+\Delta x) + q_2 u(t-\Delta t,x) - (2+q_1+2q_2)u(t,x) \\ +q_2 u(t+\Delta t,x) + u(t,x-\Delta x) - q_1 u(t-\Delta t,x-\Delta x) = 0,\end{aligned} \tag{5.9}$$

where $q_1 = a\frac{\Delta x}{\Delta t}$ and $q_2 = b\frac{\Delta x^2}{\Delta t^2}$, or to the difference star as in Table 5.6. Its high-order analogue is shown in Table 5.7, where the x's represent nonzero values and the .'s represent

Table 5.6. First-order filter for two slopes.

	$x-\Delta x$	x	$x+\Delta x$
$t-\Delta t$	$-q_1$	q_2	0
t	1	$-2-q_1-2q_2$	1
$t+\Delta t$	0	q_2	0

Table 5.7. High-order filter for two slopes.

x	x	.
x	x	.
x	x	1
x	x	.
x	x	.

zeroes. Here, two columns of coefficients (the x's) are needed to predict two slopes.

Clearly from Table 5.5, if one slope is present, one pivot trace is needed to correlate with the filter and generate a prediction of the next trace, which then can be used to predict the trace after the next, and so on. Therefore, one pivot trace is needed for the prediction, if there is only one slope. Similarly from Table 5.7, if two slopes are present, two pivot traces

are needed to predict the next trace, and so on. This, of course, can easily be extended to cases with more than two slopes.

5.2.2 Formulation

With the above derived models, we can now formulate the problem of lateral prediction. Literally, the problem is to find a filter of the form similar to the ones shown in Table 5.5 and Table 5.7 (a lateral prediction filter) that can generate a block of data consisting of linear events, given some pivot traces. Suppose we have a block of data with x_i being the ith trace in the block, where $i = 0, 1, ..., N-1$. Suppose also that there are M slopes and therefore we need a filter of M columns of coefficients f_j, as shown in Table 5.7, where $j = 0, 1, ..., M-1$. The prediction then is to generate the next trace from the previous ones, as

$$\sum_{j=0}^{M-1} x_{i+j} * f_j = \hat{x}_{i+M}, \tag{5.10}$$

where $*$ denotes convolution, and $\hat{x}$ denotes the predicted trace. (Of course, cross-correlation can be used instead of the convolution here, with the coefficients reordered.)

To perform the prediction, we need to find the filter first. To do this, we seek the filter that performs the best prediction, in the sense that the filter minimizes the difference between the predicted and the original traces, as shown in equation (5.11)

$$min \left\| \begin{pmatrix} \sum_{j=0}^{M-1} x_j * f_j - x_M \\ \vdots \\ \sum_{j=0}^{M-1} x_{j+N-M-1} * f_j - x_{N-1} \end{pmatrix} \right\|, \tag{5.11}$$

where $\|.\|$ denotes the vector norm, for which I use the L^2-norm here. After finding the filter, the prediction involves just applying the filter and generating the residuals, defined as the difference between the original and the predicted data shown in equation (5.12).

$$r_{i+M} = x_{i+M} - \sum_{j=0}^{M-1} x_{i+j} * f_j, \tag{5.12}$$

where $i = 0, 1, ..., N-M-1$.

In more detail, we can expand the convolutions in equation (5.11). Suppose each trace within the block contains K samples and there are L coefficients in each column of the filter coefficients. Then the kth sample in the convolution $x_i * f_j$ represented as $(x_i * f_j)_k$ is

$$(x_i * f_j)_k = \sum_{l=0}^{L-1} x_{i,k-l} f_{j,l}. \tag{5.13}$$

Substituting this into equation (5.11), we then have

$$min \left\| \begin{pmatrix} \sum_{j=0}^{M-1} \sum_{l=0}^{L-1} x_{j,-l} f_{j,l} - x_{M,0} \\ \sum_{j=0}^{M-1} \sum_{l=0}^{L-1} x_{j,1-l} f_{j,l} - x_{M,1} \\ \vdots \\ \sum_{j=0}^{M-1} \sum_{l=0}^{L-1} x_{j+N+M-1,K-1-l} f_{j,l} - x_{N-1,K-1} \end{pmatrix} \right\|, \tag{5.14}$$

which is equivalent to

$$min \left\| A\bar{f} - \bar{b} \right\|, \tag{5.15}$$

where

$$A = \begin{pmatrix} x_{0,0} & x_{0,-1} & \cdots & x_{M-1,0} & \cdots & x_{M-1,1-L} \\ x_{0,1} & x_{0,0} & \cdots & x_{M-1,1} & \cdots & x_{M-1,2-L} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{N+M-1,K-1} & x_{N+M-1,K-2} & \cdots & x_{N-1,K-1} & \cdots & x_{N-1,K-L} \end{pmatrix}, \tag{5.16}$$

$$\bar{f} = \begin{pmatrix} f_{0,0} \\ f_{0,1} \\ \vdots \\ f_{M-1,L-1} \end{pmatrix}, \tag{5.17}$$

and

$$\bar{b} = \begin{pmatrix} x_{M,0} \\ x_{M,1} \\ \vdots \\ x_{N-1,K-1} \end{pmatrix}. \tag{5.18}$$

Formulated in the matrix form, it is clear that the problem is just a least-squares problem. For the L^2-norm case, the filter coefficients can easily be solved using the conjugate gradient method.

5.2.3 Examples

To see how lateral prediction performs, we now look at some examples. The left in Figure 5.2 shows a synthetic section consisting of a horizontal linear event. The middle one shows the residual after prediction. The first trace is used as the pivot trace, therefore it remains the same as in the original section. Not surprisingly, the prediction is perfect as indicated by the zero residuals, since the event is perfectly linear. For comparison, the section after the wavelet transform along the lateral direction (along only the lateral direction solely for comparison) is shown on the right. Again, all but one trace is zero. Since the original section contains only a horizontal event, in the wavenumber domain only the DC component is nonzero. Therefore, after the 1D wavelet transform along the lateral direction, only the average trace is nonzero, and all the difference traces are zero. This example shows that both the wavelet transform and the lateral prediction can compact the energy in the horizontal

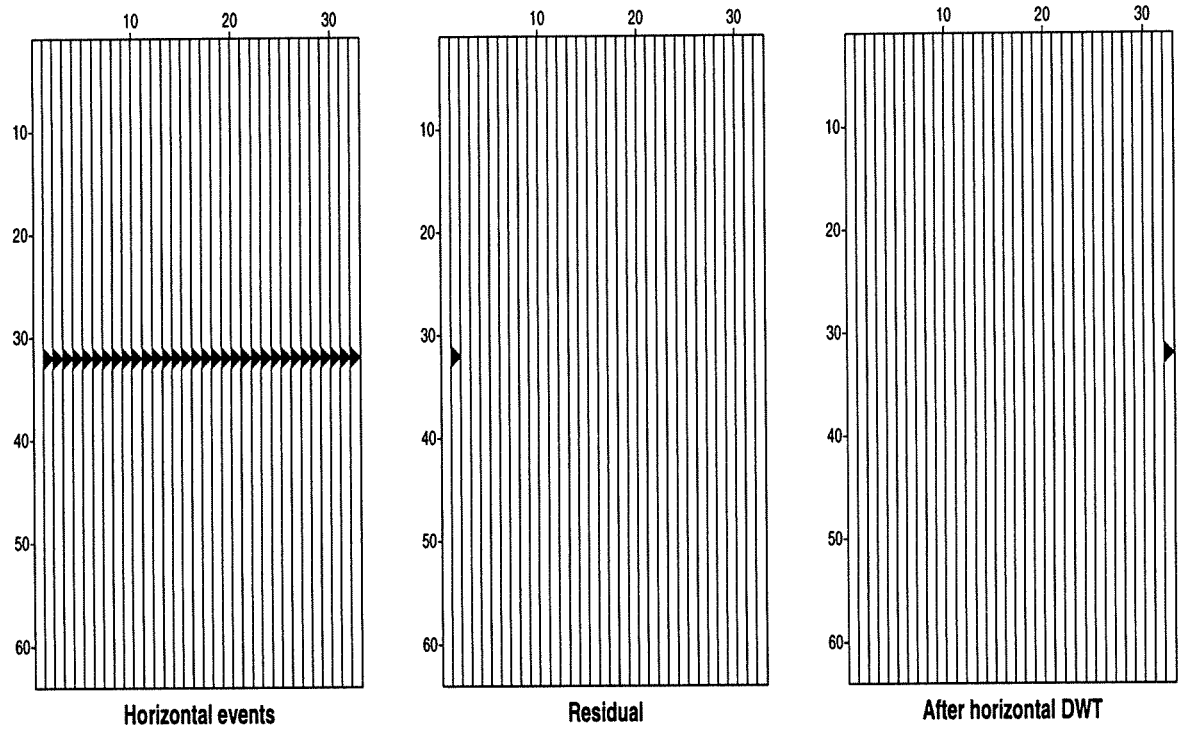

FIG. 5.2. Comparison of lateral prediction with wavelet transform for a horizontal linear event. The left shows the original section. The middle shows the pivot trace (on the left) and the residual traces after the lateral prediction. The right shows the section after the wavelet transform along the lateral direction.

linear events.

Figure 5.3 shows the same comparison on another synthetic section, this time with a slant linear event. The figure on the left again shows the original section, and the middle one

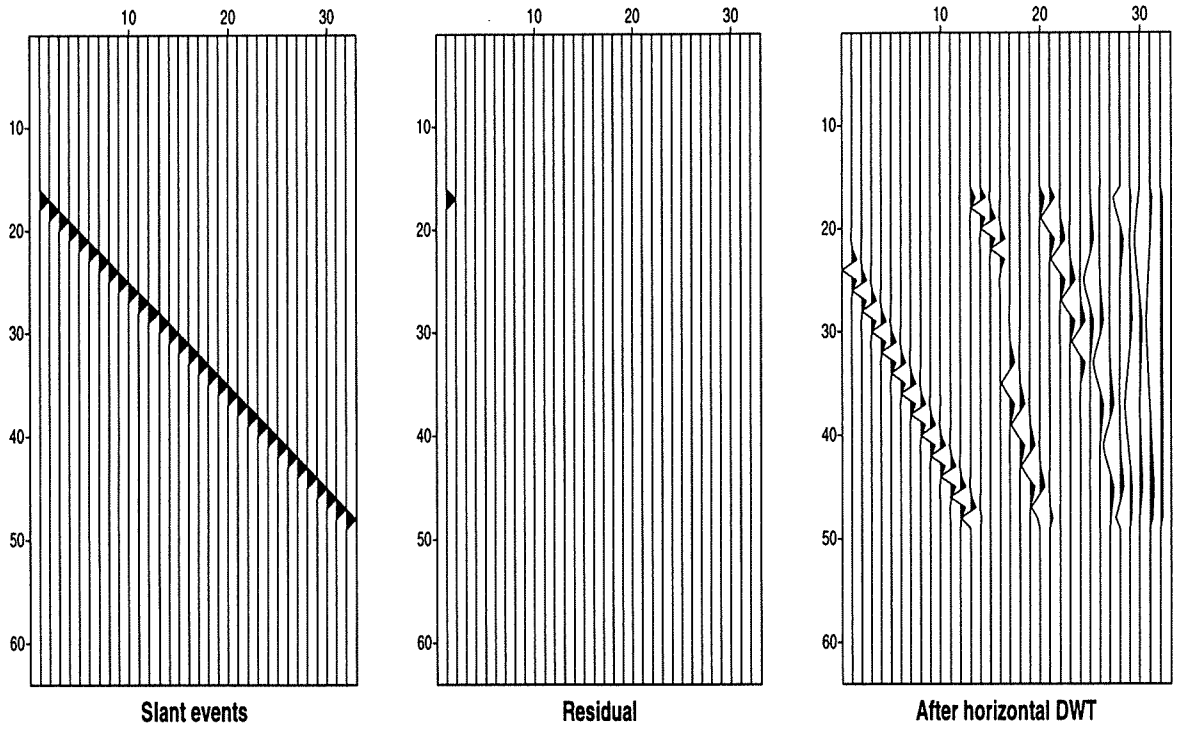

FIG. 5.3. Comparison of lateral prediction with wavelet transform for a slant linear event. The left shows the original section. The middle shows the pivot trace (on the left) and the residual traces after the lateral prediction. The right shows the section after the wavelet transform along the lateral direction.

shows the residual after prediction. Again, since the event is perfectly linear, the prediction is perfect, as indicated by the zero residuals for all but the first pivot trace. For this section, I also perform the 1D wavelet transform along the lateral direction, and the result is shown on the right. Comparing this figure with Figure 5.2, it is clear how differently the wavelet transform performs for horizontal and slant linear events. The wavelet transform compacts the energy in the horizontal events but not the slant ones. This distinction was also observed by Bosman and Reiter (1992) and motivated their approach of performing NMO-correction on the CMP gathers to align events before applying a compression technique based on wavelet transformation (Reiter and Heller, 1994).

If the 2D wavelet transform is applied, instead of the 1D transform along the lateral direction, the result does not change much, as shown in Figure 5.4. From this example, we can say that lateral prediction can compact the energy if the event is linear, no matter

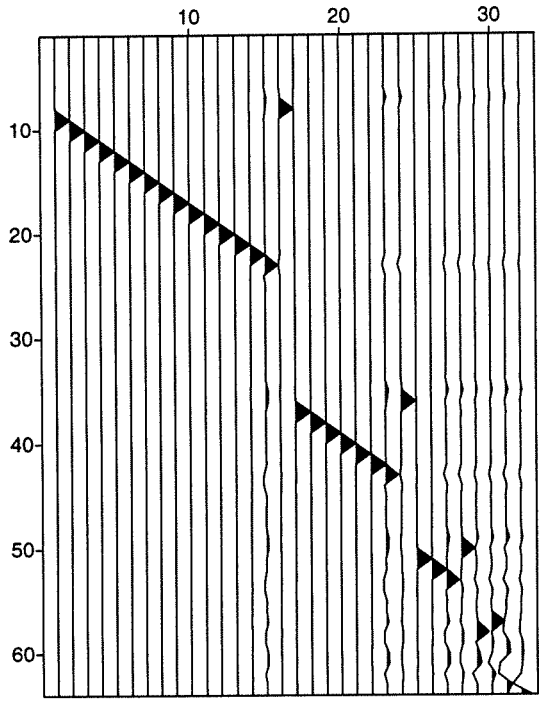

FIG. 5.4. The section in Figure 5.3 after the two-dimensional wavelet transform along the lateral and vertical directions.

whether it is horizontal or slant. The wavelet transform, on the other hand, can handle the horizontal events very well, but not the slant events.

This example seems to indicate that the approach of lateral prediction will be far superior to the 2D wavelet transform when used for data compression. For the slant event in Figure 5.3, the lateral prediction results in an entropy value of 0.0061 and the 2D wavelet transform results in an entropy value of 1.8, as compared with a computed entropy of 0.12 before transformation. As a matter of fact, this significant entropy reduction is the main motivation for the study of the lateral-prediction approach. Unfortunately, the result here does not readily extend to field data cases, as shown later in this chapter. One reason is that field data always contain noise, which significantly deteriorates the predictability of the data and therefore the efficiency of the lateral-prediction approach. Moreover, the events present in field data generally have smaller wavenumber bandwidth than does the example shown here. As shown in Chapter 4, the smaller the bandwidth, the more efficient the wavelet-transform approach becomes. The lateral-prediction approach, however, will not be more efficient for data with smaller wavenumber bandwidth: independent of the size of the slopes. lateral prediction will have similar performance. Therefore, the noise in field data, which significantly deteriorates the compression performance of the lateral-prediction approach, and the small

wavenumber in most field data which improves the compression performance of the wavelet-transform approach, result in a much less dramatic improvement of the lateral-prediction approach over the wavelet-transform approach for field data than for the simple example shown here.

If the event is not perfectly linear, as shown in the shot record section in Figure 5.5, which contains some hyperbolic events, we can still apply lateral prediction to a small piece of it. The left of Figure 5.6 shows one part of the data from receiver one through eight, and

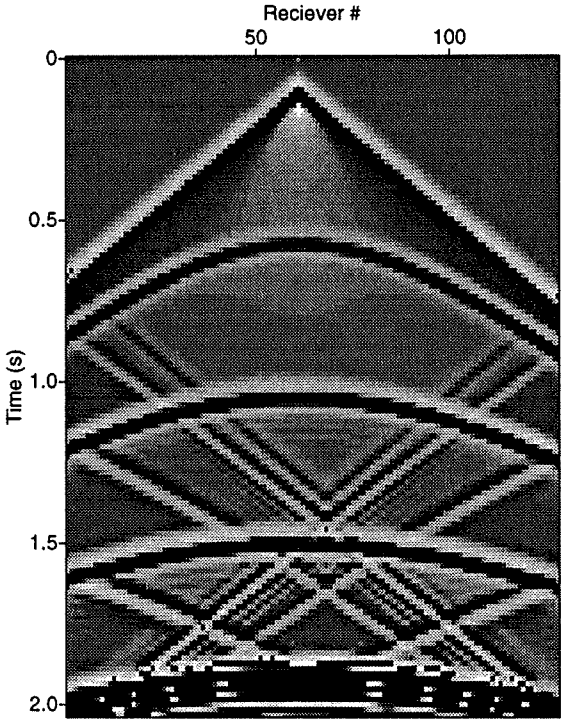

FIG. 5.5. Synthetic shot record containing hyperbolic events.

the right shows the pivot and residual traces after lateral prediction. The prediction in this case is not so perfect since the data deviate from the model used. However, the residuals are small compared to the original, indicating that the linear model is a good approximation to the hyperbolic events, locally. Just how good is this approximation depends on any advantage that can be gained in data compression. That issue is addressed below.

If the data become complicated, with crossing events, then we need to use a model better than the single-slope one. Figure 5.7 shows such an example. The figure on the left shows the original section with two conflicting slopes. The middle shows the pivot and residual traces after applying the single-slope model. The right shows the pivot and residual traces after applying the two-slope model. Clearly, the single-slope model is not able to predict the two

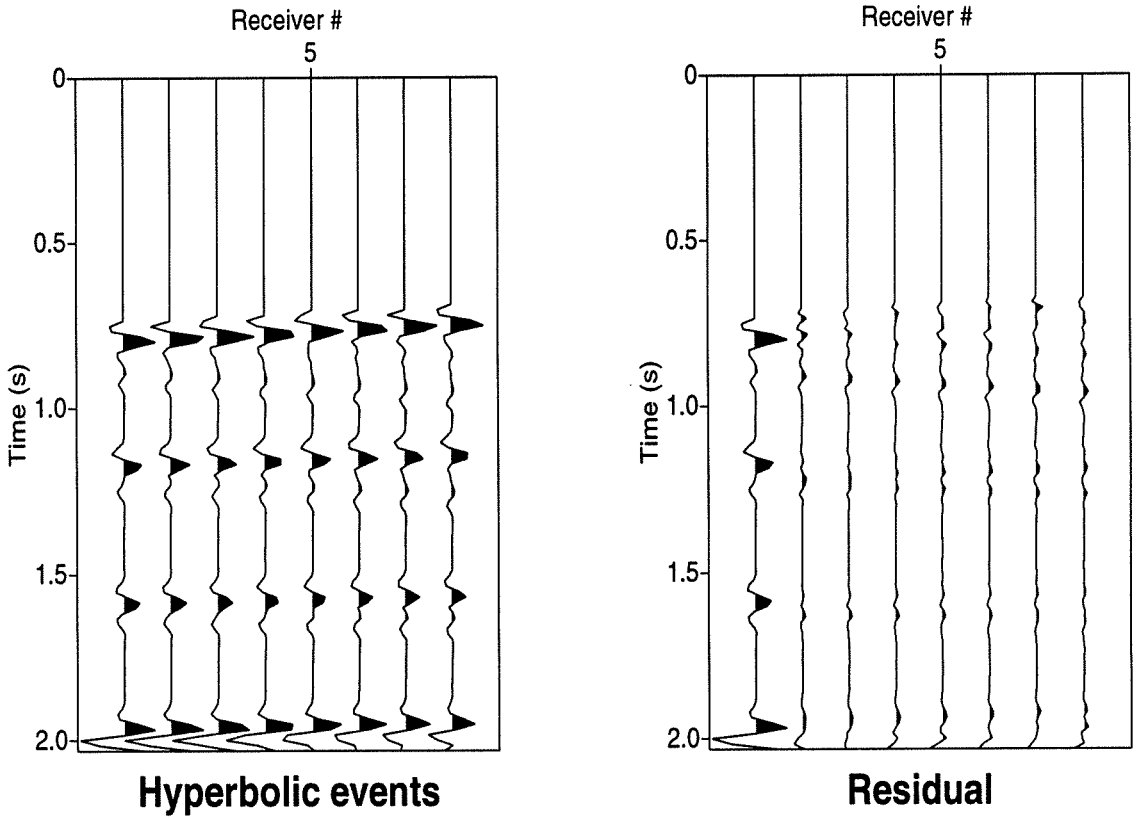

FIG. 5.6. One part of the section shown in Figure 5.5, from receiver one through eight (left), and the pivot and residual traces after lateral prediction (right).

slopes present in the data. The two-slope model, on the other hand, can predict very well, as indicated by the zero residuals.

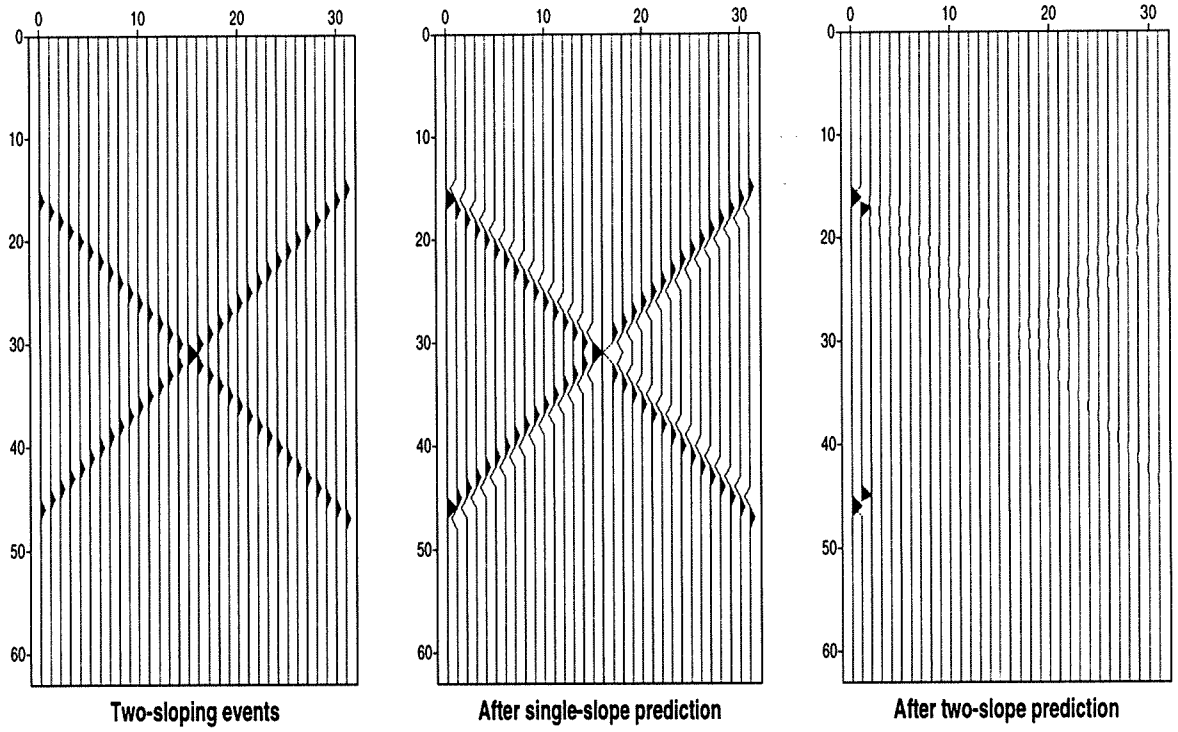

FIG. 5.7. Comparison of single- and two-slope models when the data contain two dipping events. The left shows the original section. The middle shows the pivot and residual traces after single-slope prediction. The right shows the pivot and residual traces after two-slope prediction.

Of course, field data with different types of noise present, will deviate more from the linear model than do the synthetic hyperbolic events. Figure 5.8 shows a field shot gather. The left figure in Figure 5.9 shows a small part of it, from receiver number 73 to 80. On this part, I apply the single-slope lateral prediction, and the residual is shown in the middle. Clearly, the prediction is not very good, with significant events in the residual. This is not surprising, since the one-slope linear model is obviously too simple for the data, as indicated by the several slopes that constitute the first arrivals shown in the original section. For this section, of course, we can use a better model, say, a two-slope model, and the result is shown on the right, plotted using the same amplitude scale as the one used in the middle. The prediction is clearly much improved so that the residual is much smaller than that using one-slope model; however, two pivot traces are needed in this case. I will leave the discussion of the cost-performance trade-off to the next section.

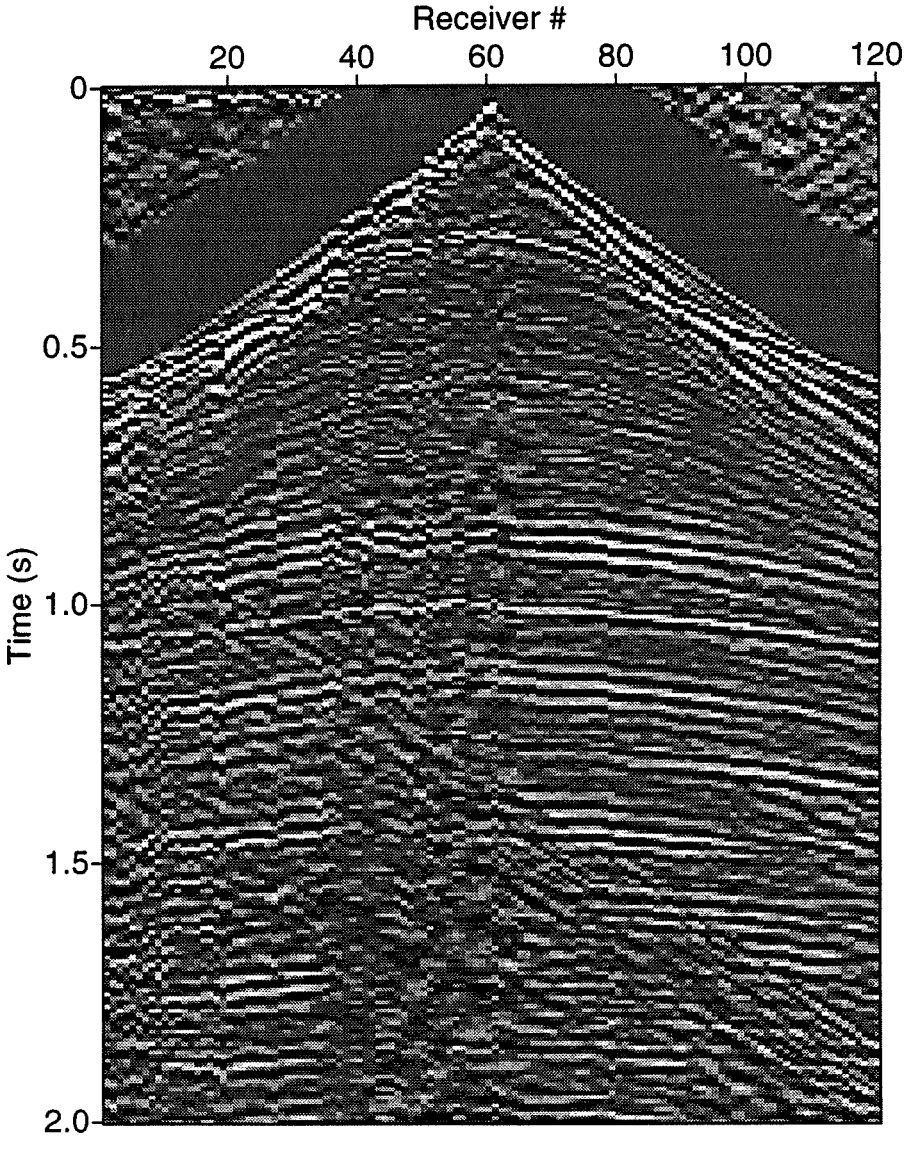

FIG. 5.8. Shot gather.

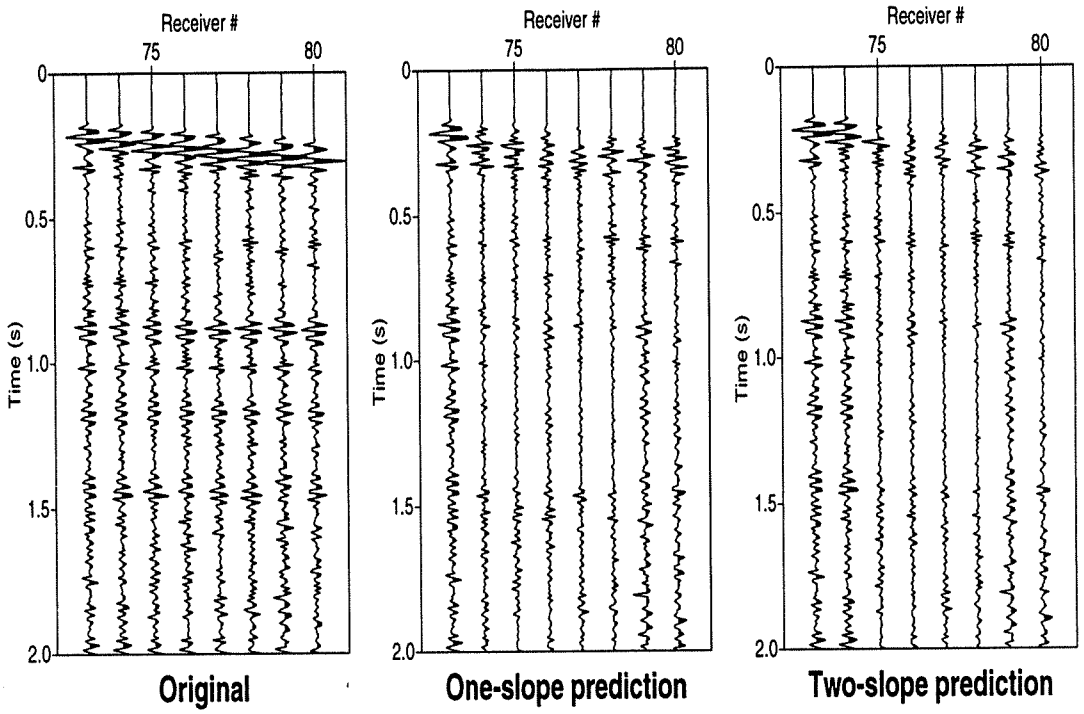

FIG. 5.9. Field data example on one part of the shot gather shown in Figure 5.8, from receiver number 73 through 80. The left shows the original section, and the middle shows the pivot and residual traces after single-slope prediction. The right shows the pivot and residual traces after two-slope prediction.

5.3 Putting it together

As shown in the previous examples, the lateral prediction can compact the energy so that the amplitudes in the residual traces after the prediction are smaller than the original trace amplitudes. However, this compaction is not equivalent to good compression performance. To see how lateral prediction works in compression, we need to evaluate the entropy values. Again, we need to first approximate the data using integer numbers. I will use one percent relative RMS error again, meaning the RMS error is one percent of the RMS amplitude of the original data. Under this criterion, the entropy value of the residual (shown in the middle of Figure 5.9) is 5.98. Compared to the entropy value of the original data, 6.56, the entropy is reduced, indicating that more compression can be achieved by applying the lateral prediction than without the prediction. If the two-slope model is used instead, the entropy value of the residual (shown in the right of Figure 5.9) becomes 5.84, a slight improvement over the entropy value of the residual after the one-slope prediction. However, this is not a large improvement compared to the improvement from no prediction to one-slope prediction. This is because after the two-slope prediction, though the residual for predicted traces becomes

much smaller than that after the one-slope prediction, there are now two pivot traces instead of one. The added entropy of this extra pivot trace partially offsets the improvement in the prediction. Therefore, using a more complicated model here does not help in compression.

So far, I have discussed applying only the lateral prediction to take into account the lateral coherency in the data. There is, of course, redundancy along the vertical direction, in both the original data and the prediction residual. Since the predicted traces are just linear combinations of the pivot traces (after linear filtering), the residual, which is the difference between the original and the predicted, is also a linear combination of the original traces and should have similar bandwidth to that of the original. This is shown in Figure 5.10. Therefore, an effective 1D compression technique for the original data should also work well

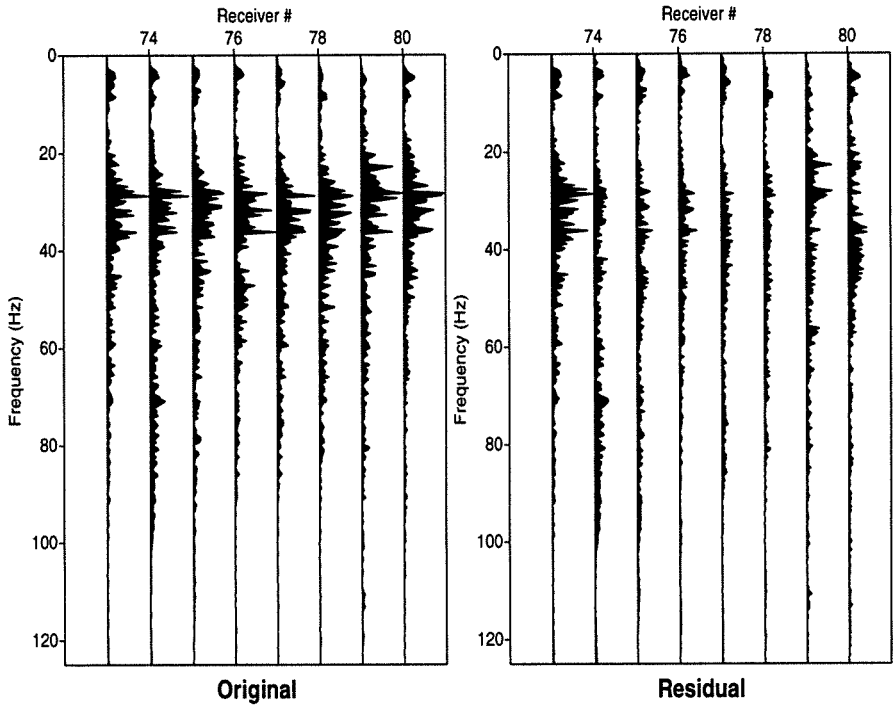

FIG. 5.10. A comparison of the amplitude spectra before and after the lateral prediction for the section shown in Figure 5.9.

on the residual. This naturally leads to a compression technique that combines characteristics of seismic data in the two directions: first apply the lateral prediction along the lateral direction and form the residuals; then apply the LCT, which is shown to be best for 1D compression of seismic traces, to the residual traces. I will call this technique PLCT, short for Prediction-and-LCT. Using PLCT, we can again evaluate the entropy values to see how much compression we can achieve. For the section in Figure 5.9, after applying PLCT (one-slope model) the entropy value becomes 5.58, as compared with the entropy value after applying only the lateral prediction, 5.98. This improvement is due to the contribution from

the LCT along the vertical axis. Also, compared with the entropy value after only applying the LCT along the vertical axis, 5.92, it is again an improvement because we have taken the lateral coherency into account in applying the lateral prediction.

5.4 Pros and cons

In this section, I perform comparisons between the compression technique PLCT and the other techniques discussed above.

Since we desire random trace accessing, we need to do the compression on small strips of data. I will use the small section shown in Figure 5.9 to do the comparison.

For this section, I apply different techniques, the 2D DCT, 2D DWT, 2D DWPT and the PLCT. Table 5.8 shows the entropy values under the criterion of one percent RMS error after applying these techniques. Clearly, the PLCT results in the least entropy.

Table 5.8. Entropy values after different techniques.

technique	2D DCT	2D DWT	2D DWPT	PLCT
entropy	5.87	5.99	6.06	5.58

Unfortunately, this does not tell the whole story. In the PLCT technique, the entropy is calculated for saved pilot trace and the prediction residual traces. However, in addition to saving the residual traces, we also need to store the filter coefficients in order to reconstruct the data. This overhead will offset the gain in the entropy reduction in the PLCT technique. For the lateral prediction I used to generate the above result, I use three coefficients. If these coefficients are also quantized, but using more bits, say 10 bits each rather than the eight bits used for the residuals, and thus introducing a smaller error, 30 bits will be needed to store the coefficients. Spreading this overhead out to all the samples in a 16×8 block, results in about a quarter of a bit overhead. This, combined with the entropy value, will result in about 5.82 bits, which is close to the entropy after the 2D DCT. Therefore, taking the overhead into consideration, the performance of the PLCT technique is similar to that of the 2D DCT. How to reduce this overhead further remains to be studied.

Besides the amount of compression, the computational cost of compressing and uncompressing data is another factor that needs to be considered in many applications. In terms of complexity, the DWT is $O(P)$, meaning the computational cost for the DWT grows linearly with the overall number P of samples. The DCT is $O(P\log(P))$, if the entire trace or section is transformed. However, in practice, DCT is applied to small blocks of data only. Suppose each block consists of N samples, where N is a small number (8 or 16 are often used), then the computational cost of DCT becomes proportional to $P\log(N)$, or simply $O(P)$. The DWPT, if the decomposition is applied all the way to the last level, also requires $O(P\log(P))$. Again, we do not generally go to the last level in most applications. In fact, doing that would even decrease of the amount of compression, as indicated by the entropy evaluations in Chapter 4.

If stopped at a small fixed level (e.g., third level), however, its computational cost can also be considered as $O(P)$.

These computational costs are valid for both the forward transform involved in the compression step and the inverse transform involved in the decompression step. For the PLCT technique, the most costly part is the filter-design step. Since the conjugate-gradient algorithm is used and the matrix used in the least-squares problem is $(N-M)K \times ML$, where M is the number of slopes in the model, L is the filter length, K is the number of samples in each block and N is the number of traces in each block, the computational cost will be proportional to $((N-M)K)^2ML$. Clearly, the more filter coefficients and the more complicated model used, the higher the computational cost in the PLCT technique. This, combined with the result shown before that complicated filters may not help much in improving the amount of compression, leads to a preference of using the one-slope model in practice. In terms of the total number of sample points P, the computational cost for the PLCT technique is also $O(P)$, since the size of each block is fixed.

Although all the algorithms can be considered as $O(P)$, the actual execution times of the algorithms do differ. Among them, the lateral-prediction approach is the most expensive, followed by the DCT algorithm, then by the DWPT algorithm, and the DWT is the most computationally efficient algorithm.

With the relative computational cost in mind, we can evaluate the trade-off between the computational cost and the compression ratio. Throughout the thesis, however, I have used the entropy value as the measure for compression achievable. Similar to the discussion in Chapter 4, to understand how the entropy values are related to the the compression ratio, we need to specify the amount of error allowed during the compression, and this error might depend on the application of the compression. Suppose in data processing, the allowed RMS error is one percent, and, for example, under one percent RMS error, one technique (A) results in an entropy value of 5.0 and another (B) results in an entropy value of 6.0. Then, if compression is applied in processing, A results in an ideal compression ratio of 6.4:1 and B results in an ideal compression ratio of 5.3:1; A is about 20 percent improvement over B. For data interpretation, however, the allowed RMS error is generally larger than one percent. Suppose 16-percent RMS error (which is just 2.5-percent MSE) is allowed. From equation (2.9), A might result in an entropy value of 1.0, and B in an entropy value of 2.0. In terms of compression ratio, A achieves 32:1 compression and B achieves 16:1 compression; here A is 100 percent improvement over B. Therefore, when comparing different compression techniques, it is important to know how the techniques will be applied and how much error will be allowed in those applications.

Chapter 6

INFLUENCE OF DATA PROCESSING

Our interest in seismic data arises from their ability to provide information about the earth's subsurface, and that information emerges only after the data have been subjected to a number of stages in data processing. Therefore, ultimately our assessment of the performance of compression must be done on data after they have been processed. Here, I compare errors on data that have been compressed and uncompressed before processing with those that remain after the uncompressed data have been processed. I concentrate on two representative processes: migration and deconvolution, migration representing a pure-phase process and deconvolution an inverse one. Therefore, we can expect compression error to propagate through these two processes differently.

The processing is performed using the freely available seismic data processing package "Seismic Unix" (SU), developed at the Center for Wave Phenomena of Colorado School of Mines.

6.1 Migration

The stacked section (Figure 6.1) used for the migration test is from a marine survey. The migration algorithm used in this experiment is the phase-shift migration *sumigps*, and the velocity used in the migration is obtained from stacking velocity analysis on the original data. To evaluate the compression performance, I apply the migration process to the original data as well to data uncompressed from different levels of compression, each one allowing a different amount of error in the compression. Figure 6.2 shows the migrated section from the original data, the migrated section from the uncompressed data with ten percent error in the compression (compression ratio equals to 12:1), and the difference section between these two migrated sections. The two migrated sections look almost identical and the difference section shows no coherency.

For a quantitative measure of this difference between the migrated sections, I calculate the relative RMS error after migration, defined as the ratio between the RMS amplitude of the difference of the migrated sections and the RMS amplitude of the migrated section from the original section, shown in eqation (6.1).

$$\mathrm{Err}_{\mathrm{mig}} = \frac{\mathrm{RMS[Mig(original) - Mig(after\ compression)]}}{\mathrm{RMS[Mig(original)]}}. \tag{6.1}$$

As a comparison, I also compute the relative RMS error before migration, defined as the ratio between the RMS amplitude of the difference (between the original section and the section from uncompressed data) and the RMS amplitude of the original section, shown in

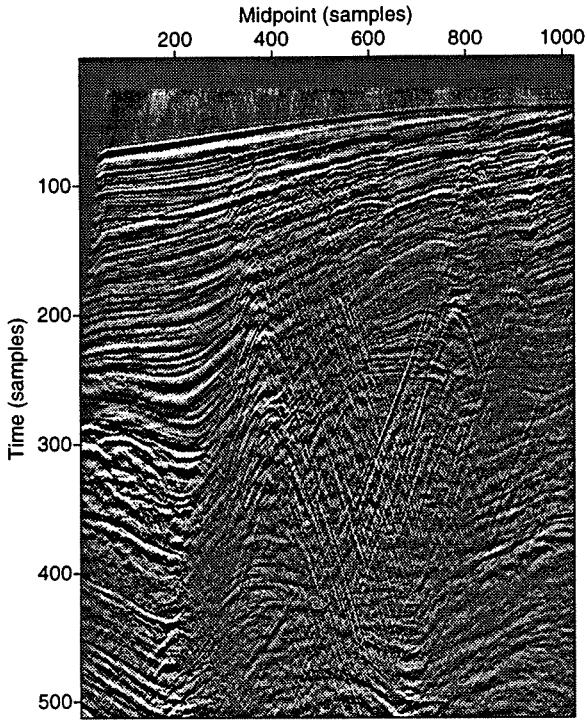

FIG. 6.1. Stack section.

equation (6.2).

$$\mathrm{Err}_{\mathrm{orig}} = \frac{\mathrm{RMS}(\mathrm{original} - \mathrm{after\ compression})}{\mathrm{RMS}(\mathrm{original})}. \tag{6.2}$$

Table 6.1 shows these relative RMS errors. From the table, the error between the migrated

Table 6.1. RMS error before and after migration.

$\mathrm{Err}_{\mathrm{orig}}$	1.0%	4.8%	9.4%	17.4%	28.6%	36.0%
$\mathrm{Err}_{\mathrm{mig}}$	.8%	4.0%	8.0%	15.0%	25.5%	32.9%
ratio	.80	.83	.85	.86	.89	.91

sections is smaller than that between the original sections for all levels of error. In other words, the migration process actually reduces the amount of error. This is possibly related to the fact that migration is a weighted summation process and random noise becomes somewhat weakened. Therefore, if the error during compression can be treated as some random noise added to the original data, it will be reduced after migration. Also from the table, the amount of error reduction after migration becomes smaller as the error gets larger. One possible

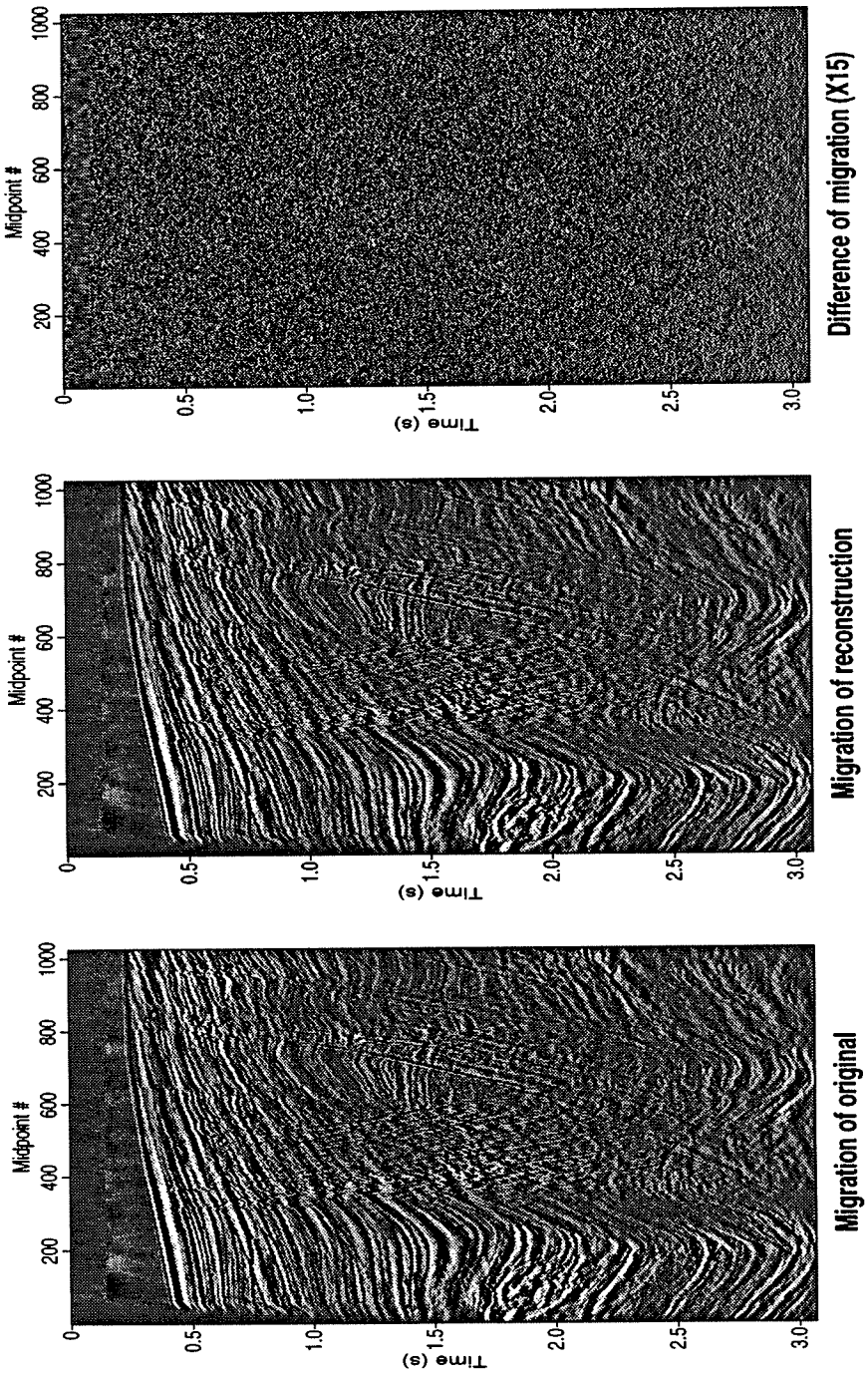

FIG. 6.2. Comparison of the migrated sections for the original data and data after compression and uncompression with ten percent error in the compression.

interpretation for this is that as we allow more error in compression, as shown before, the error becomes more coherent so the summation becomes less effective in reducing the amount of error.

6.2 Deconvolution

Similarly, I apply the deconvolution process to the original data as well as the uncompressed data. The data set used for this test is a shot gather, shown in Figure 6.3. The

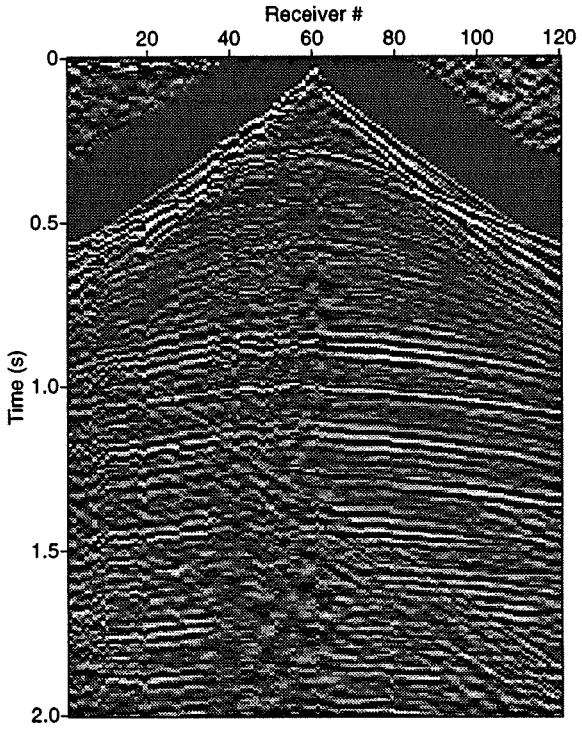

FIG. 6.3. Shot gather before deconvolution.

deconvolution algorithm used for this experiment is the Wiener prediction-error filtering *supef* with unit prediction distance (i.e., spiking deconvolution) and operator length 150 ms. The relative additive noise level is set to 0.01 percent. Following the deconvolution, band-pass filtering is then applied. Two experiments with passbands of $5 - 50$ Hz and $5 - 75$ Hz are performed. The sections in Figure 6.4 show the deconvolved original data, the deconvolved uncompressed data with ten-percent error in the compression, and the difference between these two deconvolved sections, when the filter with a passband of $5 - 50$ Hz is used. The two deconvolved sections look almost identical, and the difference section shows no coherency. Figure 6.5 shows a similar comparison for which the final filter has a passband of $5 - 75$ Hz. Comparing the result here with that shown in Figure 6.4, the deconvolved sections are similar.

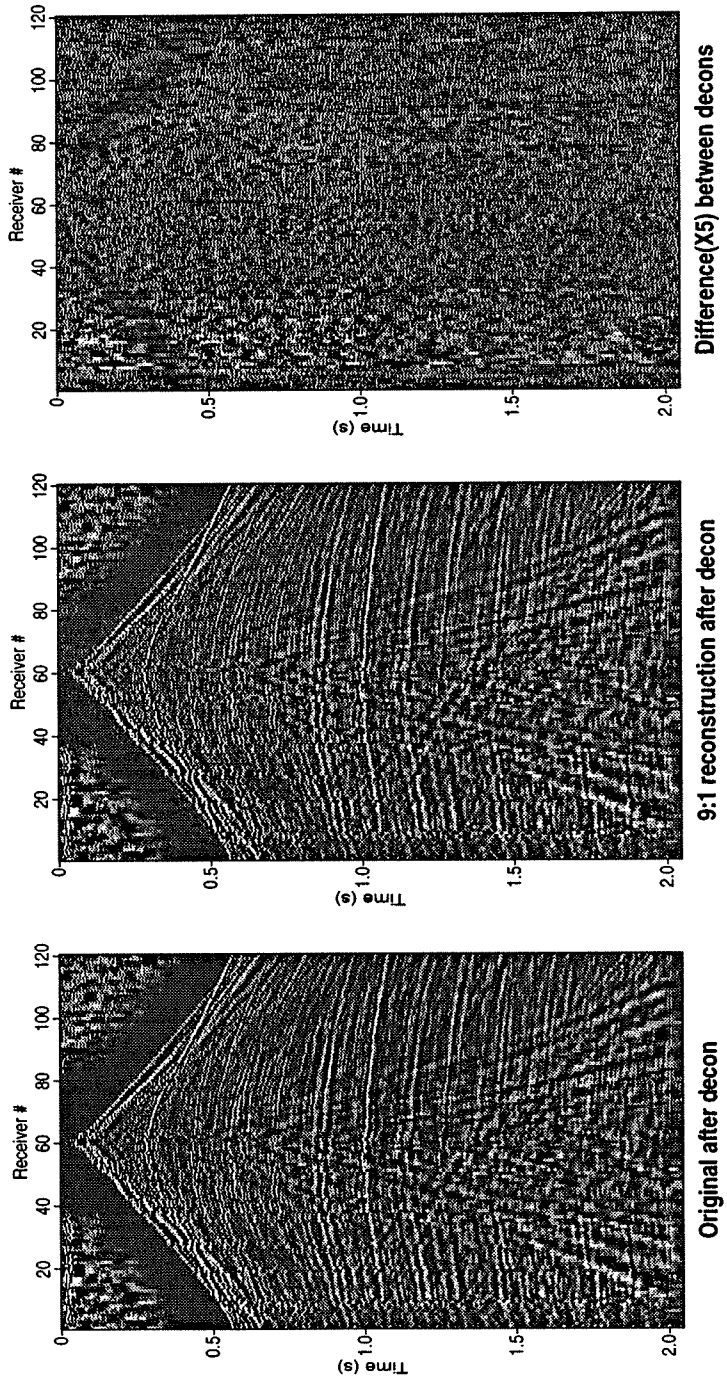

FIG. 6.4. Comparison of the deconvolved original data and deconvolved data after compression and uncompression. The deconvolution is followed by band-pass filtering with a passband of $5 - 50$ Hz.

However, the difference section here has higher noise level and shows coherency associated with some of the reflections This is understandable since a larger bandwidth is used here.

Again, for a quantitative measure of this difference between the deconvolved sections, I calculate the relative RMS errors before deconvolution, shown in equation (6.2), and after deconvolution, shown in equation (6.3).

$$\mathrm{Err}_{\mathrm{decon}} = \frac{\mathrm{RMS[Decon(original) - Decon(after\ compression)]}}{\mathrm{RMS[Decon(original)]}}. \tag{6.3}$$

Table 6.2 shows these relative RMS errors. From the table, the error between the deconvolved

Table 6.2. RMS error before and after deconvolution.

$\mathrm{Err}_{\mathrm{orig}}$	1.1%	5.3%	10.2%	14.9%	19.2%	23.3%
$\mathrm{Err}_{\mathrm{decon}_{5-50}}$	1.7%	8.5%	16.2%	23.9%	30.8%	35.9%
$\mathrm{Err}_{\mathrm{decon}_{5-75}}$	1.7%	8.6%	16.7%	24.7%	31.5%	37.0%

sections is larger than that between the original sections. In other words, the deconvolution process enlarges the amount of error. This is understandable since deconvolution is used to enhance the low- and high-frequency components, which are usually weaker than the dominant-frequency components in the signal. On the other hand, the error introduced during the compression is distributed more or less evenly. Considering the compression error as added noise, the signal-to-noise ratio for the low- and high-frequency components is smaller than that for the dominant-frequency components. Deconvolution boosts the weakly-represented frequencies, and therefore decreases the net signal-to-noise ratio, resulting in the amplification of compression error. Also from the table, the errors are slightly larger for the broader $5 - 75$ Hz bandwidth data, which contain more contribution from the noisier and less coherent high frequencies.

In Chapter 4, I showed that data before deconvolution can be better compressed than those after deconvolution, if the same amount of error is allowed in compression. We have just seen here that the error is enlarged during the deconvolution process. Suppose we allowed a larger error when compressing the deconvolved data than when compressing the undeconvolved data? How would the amount of compression for a process of compression and uncompression followed by deconvolution ("compression+deconvolution") differ from that for a process of deconvolution followed by compression and uncompression ("deconvolution+compression") under the same final error? Table 6.3 shows the compression ratios for the two processes, with the same final errors, where the final error is defined as the relative RMS error of the uncompressed deconvolved (or the deconvolved uncompressed) data with respect to the deconvolved data without compression applied. The passband of the band-pass filter here is $5 - 50$ Hz. As shown in the table, the "compression+deconvolution" process achieves higher compression ratio than does the "deconvolution+compression" process, for

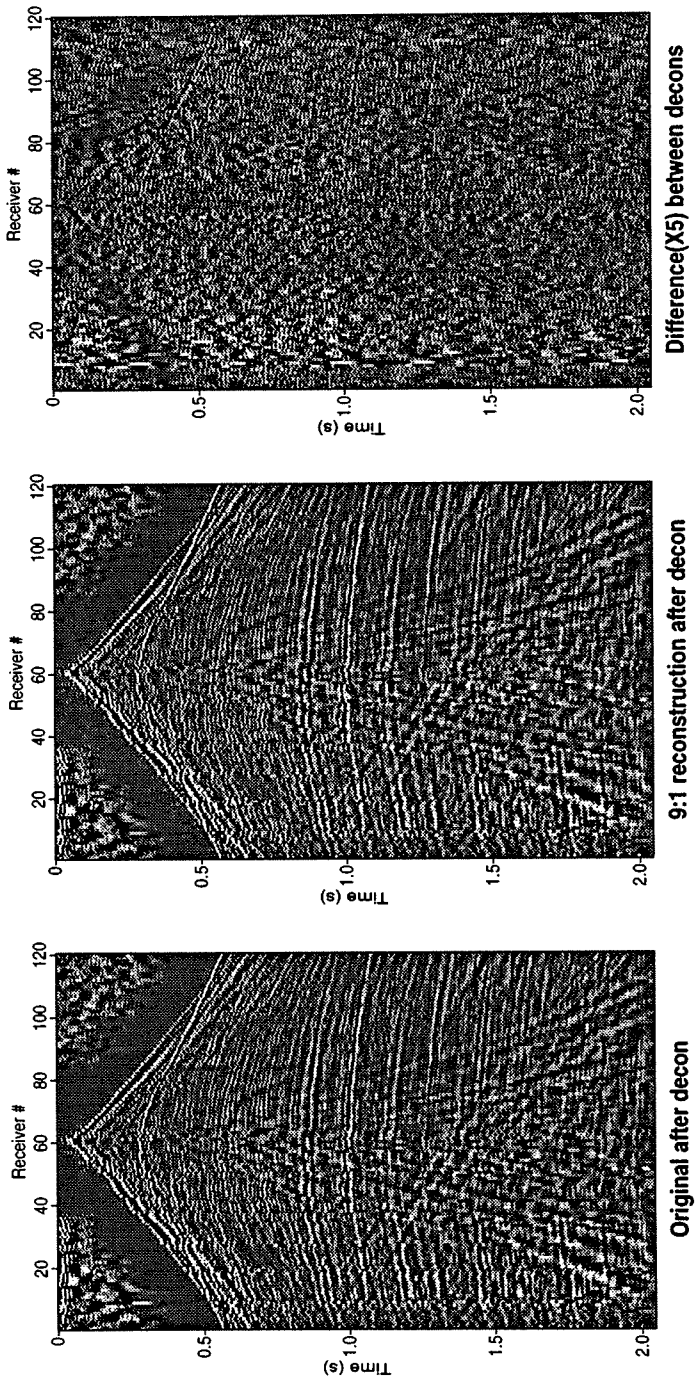

FIG. 6.5. Comparison of the deconvolved original data and deconvolved data after compression and uncompression. The deconvolution is followed by band-pass filtering with a passband of $5 - 75$ Hz.

Table 6.3. Compression ratios for the two processes for the $5 - 50$ Hz passband.

final err	.7%	3.5%	6.7%	9.9%	12.7%	14.9%
comp+decon	4.77	7.02	9.08	10.57	11.79	13.03
decon+comp	4.45	6.33	7.80	9.08	9.94	10.45

the deconvolution operator I use here. A similar test for the case of $5 - 75$ Hz is shown in Table 6.4. Again, as shown in the table, the "compression+deconvolution" process achieves

Table 6.4. Compression ratios for the two processes for the $5 - 75$ Hz passband.

final err	.8%	4.0%	7.8%	11.6%	14.8%	17.3%
comp+decon	4.77	7.02	9.08	10.57	11.79	13.03
decon+comp	4.55	6.58	8.23	9.56	10.29	10.78

higher compression ratio than does the "deconvolution+compression" process. Notice that the same compression ratios for the "compression+deconvolution" process were used for both passbands. Therefore, the errors in the compression and uncompression step are the same for the two passbands and the final error for the $5 - 75$ Hz passband is larger than that for the $5 - 50$ Hz passband, since it has a larger bandwidth. Because of this larger final error, the compression ratio for the "deconvolution+compression" process is larger for the $5 - 75$ Hz passband than for the $5 - 50$ Hz one.

Of course, one could study the relationship between data compression, error, and other seismic processes. Of the various processes other than deconvolution and migration, perhaps CMP stacking is the one that is most important. As Bosman and Reiter (1993) have shown, stacking can reduce the compression error, much as does migration. This result has been used as the rationale for allowing a larger error when compressing the prestack data than when compressing the poststack data.

Chapter 7

DISCUSSIONS

7.1 Summary

In Chapter 2, after reviewing some results on the quantization theory, I obtained the compression ratio as a function of compression error, thus quantitatively describing the trade-off between the compression ratio and compression error. This theoretical result matches well with the experimental one observed by Reiter and Heller (1994), and can be used as the guideline in helping to design compression algorithms.

In Chapter 4, using the entropy value to measure the maximum amount of compression achievable, I compared the performance of some transforms that have been studied in the literature, when applied to compressing seismic data. It turns out that for both 1D and 2D data, the local cosine transform performs marginally better than does the fixed-level wavelet packet transform for most wavelets. The standard wavelet transform, when applied to seismic data, does not perform as well as the fixed-level wavelet packet transform, although it is computationally less expensive. Besides the choice of transforms, other factors related to the data, such as the frequency bandwidth, lateral coherency, signal-to-noise ratio all play a role, in the amount of compression achievable. The smaller the frequency bandwidth, the stronger the lateral coherency, and the higher the signal-to-noise ratio, the more compression achievable.

In Chapter 5, I raise the issue of random trace accessing, an issue somewhat unique to seismic data. I study several approaches in dealing with this issue. Compressing each trace independently, though it achieves perfect random trace accessing, ignores the lateral (trace-to-trace) coherency, resulting in degradation of compression performance. In addition, single-trace compression requires a coding technique that introduces small coding overhead; this disqualifies some standard coding techniques. I introduce a form of variable-length integral coding technique that reduces the overhead.

To take into account the lateral coherency in seismic data, I compress small two-dimensional strips, though it sacrifices some random trace accessing. On the small strips, I compare two-dimensional techniques as well as a technique that employs lateral prediction, where I first apply the lateral prediction to compact the lateral coherency and then compress the residuals using one-dimensional local cosine transform. The compression performance of the prediction approach is little different from that of the two-dimensional techniques applied on small strips while the computational cost is significantly higher. Lateral prediction, however, somewhat compacts the energy into a smaller region (the pivot trace), and thus deserves further investigation since some sophisticated coding techniques might take advantage of this compaction and achieve better compression performance.

In Chapter 6, I process some field data that have undergone the compression and un-

compression cycle, and compare the result with that of the original data. It turns out that migration reduces the compression error while deconvolution enlarges this error, a result that is within expectation.

7.2 Conclusions

The main purposes of this research have been to gain some understanding of specific characteristic of seismic data as they relate to data compression.

Here is a brief summary of the understanding drawn from the study.

- Data can be characterized by a differential entropy governed by the amplitude probability density function $h(X)$.
- Although the entropy (first-order) I discuss here does not directly characterize the coherency in the data and higher-order entropy might be used to better describe the data, after an appropriate transform, it can be used to characterize the complexity of the data.
- With a well-chosen transformation, the transformed data can be made more compact; the changed probability density function thus results in a lowering of the entropy relative to that without the transform. The appropriate transformation is the one that best exploits characteristics of the data, such as the band-limited character in time and space.
- With suitable quantization and coding, along with a choice of allowable average distortion, the entropy H_Q can be further reduced, and, for the given allowed distortion, that entropy is the ideal minimum number of average bits required to represent the data in a compressed form.
- The storage requirements for the compressed data are dictated by H_Q plus overhead required by the particular encoding algorithm (some algorithms require less overhead than do others).
- That overhead is a burden that becomes relatively smaller the larger the data set (usually, the larger the dimensionality of the data).
- Additional (sometimes unacceptable) cost is required to compress and to uncompress the data.
- The cost-benefit ratio will be a moving target, but at present there seems to be benefit in compressing larger-volume, three- and four-dimensional data for telemetering from ship to processing center (Donoho, et al., 1995; Stigant, et al., 1995).
- An important part of the decision process as to cost-effectiveness is the perception of the geophysicists as to what level of compression noise is acceptable; the larger the tolerable noise level, the more cost-effective the compression effort. This is almost in

the realm of psychology since the largest breakthrough may be nothing other than a raising in the amount of noise that is judged tolerable. Also, the issue of subsequent processing that will be done on uncompressed data must be taken into account.

- Data access is an issue that may come into play in possible applications such as data retrieval.

7.3 Future studies

This research has produced a collection of software modules that can be used as a platform for further investigation into new compression techniques. It is also possible to extend the analysis to higher dimensions. Other tests, such as processing data with and without compression and uncompression for other processes, or applying compression after each step of processing, can also be performed. These tests are important in a scenario where the data volume is very large, such as the terrabytes of data in modern 3D surveys. Therefore, one could compress the data at the beginning of a processing flow, process the data, compress the result of subsequent processes and save these compressed results. Finally, much room exists for theoretical analyses of the propagation of compression error through various processes.

REFERENCES

Bellamy, J., 1991, Digital Telephony, 2nd edition, Wiley.

Bordley, T. E., 1983, Linear predictive coding of marine seismic data: IEEE Trans. on ASSP, **31**, 828-835.

Bosman, C. L., and Reiter, E. C., 1993, Seismic data compression using wavelet transforms: Expanded Abstracts, 63rd Annual International SEG Meeting, 1261-1264.

Claerbout, J. F., 1992, Earth Soundings Analysis, Blackwell Scientific Publications.

Daubechies, I., 1992, Ten Lectures on Wavelets, SIAM.

Donoho, P., Ergas, R., and Villasenor, J., 1995, High-performance seismic trace compression: Expanded Abstraces, 65th Annual International SEG Meeting, 160-163.

Le Gall, D., 1991, MPEG: A video compression standard for multimedia applications: Commun. ACM, **34**, 46-58.

Gersho, A., and Gray, R. M., 1992, Vector quantization and signal compression, Kluwer Academic Publishers.

Hall, M., Monk, D., and Reiter, E., 1995, An evaluation of seismic data compression on the interpretability of the final product: EAEG 57th Conference and Technical Exhibition, **B035**.

Hewlett, C. J. M., and Hatton, L., 1995, Seismic data compression with CD-ROM archiving: EAEG 57th Conference and Technical Exhibition, **B034**.

Huffman, D. A., 1952, A method for the construction of minimum-redundancy codes: Proceedings of the IRE, **40**, 1098-1101.

Luo, Y., and Schuster, G. T., 1992, Wave packet transform and data compression: Expanded Abstracts, 62nd Annual International SEG Meeting, 1187-1190.

Reiter, E. C., and Heller, P. N., 1994, Wavelet transform-based compression of NMO-corrected CDP gathers: Expanded Abstracts, 64th Annual International SEG Meeting, 731-734.

Spanias, A. S., Jonsson, S. B., and Stearns, S. D., 1991, Transform methods for seismic data compression: IEEE Trans. on Geoscience and Remote Sensing, **29**, 407-416.

Stigant, J., Ergas, R., Donoho, P., Minchella, A., and Galibert, P., 1995, Field trial of seismic compression for real-time compression: Expanded Abstraces, 65th Annual International SEG Meeting, 960-962.

Wallace, G. K., 1991, The JPEG still-picture compression standard: Commun. ACM, **34**, 30-44.

Welch, T. A., 1984, A Technique for high-performance data compression: Computer, June, 8-19.

Witten, I. H., Neal, R. M., and Cleary, J. G., 1987, Arithmetic coding for data compression: Commun. ACM, **30** 520-540.

Wood, L. C., 1974, Seismic data compression methods: Geophysics, **39**, 499-525.

Wickerhauser, M. V., 1994, Adapted Wavelet Analysis from Theory to Software, A K Peters.