REDUCING THE PRESSURE ON DATA ACQUISITION AND PROCESSING: II - DATA-DRIVEN COMPRESSION USING CONIC CODING

LUC T. IKELLE and IOAN STURZU

CASP Project, Department of Geology and Geophysics, Texas A&M University, College Station, Texas 77843-3115, U.S.A.

(Received September 25, 2008; accepted December 13, 2008)

ABSTRACT

Ikelle, L.T. and Sturzu, I., 2009. Reducing the pressure on data acquisition and processing: II - Data-driven compression using conic coding. *Journal of Seismic Exploration*, 18: 119-133.

We here propose a compression method that we characterize as data-driven compression because it is based on transforms which are data-dependent instead of generic mathematical transforms like wavelet transforms. Furthermore, our data-driven compression can be used in cascaded form with existing arithmetic data-compression methods (that is, a data-driven compression followed by an arithmetic data-compression method) because our data-compression method is based on accuracy, whereas arithmetic data-compression methods are based on precision. Such a cascade application of compression techniques can increase the compression ratio to 100:1 or more.

KEY WORDS: data compression, matrix factorization, data acquisition, data processing, multiplicative update rules.

INTRODUCTION

The goal of data compression is to shrink the computer space needed to store a dataset. Modern seismic-data acquisition and processing techniques involve the use of datasets whose sizes can range into the terabytes. The storage and handling of these datasets are significant components of the cost of data acquisition and processing. Despite the rapid increase in available disk storage at ever-lower costs, manipulation of very large datasets on the order of terabytes remains a formidable problem in many E&P organizations. Hence it is important to develop compression methods which allow us to reduce the storage size of these datasets. Yet the literature on seismic-data-compression methods is quite limited. Here are the classical examples: Spanias et al. (1991), Luo and Schuster (1992), Bosman and Reiter (1993), Donoho et al. (1995), and Tage et al. (2004).

One of the reasons often cited for the limitations of investigations of seismic data compression is that early developments have produced disappointing compression ratios, generally less than 10:1*. Moreover, these developments are based on generic mathematical transforms such as wavelet transforms or cosine transforms, which do not take into account the specific characteristics of seismic data. Therefore, short of creating new transforms, very little progress can be made in data compression.

We here propose a compression method that we characterize as data-driven because it is based on transforms which are data-dependent instead of on generic mathematical transforms such as wavelet transforms. Furthermore, data-driven compression can be used in cascaded form with existing arithmetic data-compression methods (that is, a data-driven compression followed by an arithmetic data-compression method) because our data-compression method is based on accuracy, whereas the arithmetic data-compression methods are based on precision. Such a cascade application of compression techniques can increase the compression ratio to 100:1 or more.

Before we immerse ourselves in the construction of data-driven transforms (i.e., transforms which vary with the data under consideration) needed in the development of data-driven compression, let us look at how such transforms can help us in our compression process. Consider a dataset, $Y(\mathbf{x}_r,t,\mathbf{x}_s)$, where $\mathbf{x}_r$ and $\mathbf{x}_s$ describe the receiver points and shot points, respectively, and t describes the arrival times of seismic events. Suppose that we can describe this dataset as a linear superposition of some features that can be extracted from the same dataset. If we denote these features by the functions $\psi_i(t)$, then we can write our data as follows:

$$Y(\mathbf{x}_r,t,\mathbf{x}_s) = \sum_{i=1}^{M} a_i(\mathbf{x}_r,\mathbf{x}_s)\psi_i(t) \quad , \tag{1}$$

* The data-compression ratio is the ratio between the size of the uncompressed data and the size of the compressed data. Thus, for a 20-terabyte dataset compressed to 2 terabytes, we have a compression ratio of 10:1 (which is often written "ten to one").

where $a_i(\mathbf{x}_r,\mathbf{x}_s)$ are stochastic coefficients. When the M functions $\psi_i(t)$ are taken as a set, they constitute a dictionary. Individually, they are known as basis functions, atoms, or elements of the dictionary.

Suppose now that we have a dataset which has 1,000 source points,320 receiver points for each shot point, and 625 samples per receiver (i.e., each trace has 625 samples). The number of datapoints is $K = 625 \times 1000 \times 320$. If we assume that the number of basis functions needed to describe this dataset is $M = 625$, then we need $K = 625 \times 625 + (625 \times 1{,}000 \times 320)$ datapoints to describe the decomposed data, with 625×625 datapoints for the dictionary and $625 \times 1{,}000 \times 320$ for the stochastic coefficients. In other words, we need more storage for the decomposed data than for the original data. Fortunately, we need only 40 basis functions, on average, to describe a seismic trace (i.e., a source and receiver pair). The others are almost zero. Thus we need fewer than K' points [with $K' < 625\times 625 + (40 \times 1{,}000 \times 320)$] to describe the decomposed data. That means that we can achieve a compression rate of 15:1 or more. If we apply an additional arithmetic compression to store the coefficients $a_i(\mathbf{x}_r,\mathbf{x}_s)$, we can increase the compression rate even more.

DATA-DRIVEN TRANSFORM

The basic idea for constructing the basis functions $\psi_i(t)$ is to consider them as small parts of $Y(\mathbf{x}_r,t,\mathbf{x}_s)$ in such a way that the sum of their parts allows us to reconstruct any trace of $Y(\mathbf{x}_r,t,\mathbf{x}_s)$. To achieve the sum of their parts, we assume that the coefficients $a_i(\mathbf{x}_r,\mathbf{x}_s)$ are always nonnegative. This nonnegative constraint ensures that the decomposition in (1) is purely additive (no subtraction is allowed). Why do we need this constraint? As we see in the numerical example, this constraint allows us to increase the sparsity of coefficients $a_i(\mathbf{x}_r,\mathbf{x}_s)$ - that is, to increase the number of near-zero coefficients and thus increase the compression ratio.

The mathematics for computing $\psi_i(t)$ can be formulated as follows. We start by randomly extracting N traces from a number of gathers of $Y(\mathbf{x}_r,t,\mathbf{x}_s)$ that are representative of the characteristics of this dataset. For example, in a 300-gather dataset, we can take one in every six gathers to form a set of 50 gathers that is representative of the data. We can then randomly extract N traces from this set. The number N is generally very large - about 10,000 or more. We group these N traces into an $L \times N$ matrix that we denote $\mathbf{D}$. We also describe the basis functions $\psi_i(t)$ and $a_i(\mathbf{x}_r,\mathbf{x}_s)$ into matrix forms. The basis functions are described as an $L \times M$ matrix, which we denote $\boldsymbol{\Psi}$, and the coefficients $a_i(\mathbf{x}_r,\mathbf{x}_s)$ as an $M \times N$ matrix, which we denote $\mathbf{A}$. So in matrix form, (1) becomes

$$\mathbf{D} = \boldsymbol{\Psi}\mathbf{A} \ , \tag{2}$$

where **D** is known but $\mathbf{\Psi}$ and **A** are unknowns.

Our next task is to reconstruct $\mathbf{\Psi}$ and **A**. We propose to obtain these two matrices as a minimization with the following objective function:

$$E(\mathbf{A},\mathbf{\Psi}) = \|\mathbf{D} - \mathbf{\Psi}\mathbf{A}\|_2 = \sum_{l=1}^{L}\sum_{n=1}^{N}[D_{ln} - \sum_{m}^{M}\psi_{lm}A_{mn}]^2 \ , \tag{3}$$

with the nonnegative constraint that the elements of **A** are always nonnegative.The elements of the matrices of **D**, **A**, and $\mathbf{\Psi}$ are denoted here as D_{ln}, A_{mn} and ψ_{lm}, respectively. Note also that an additional constraint is needed in (3) because the objective function can always be decreased by simply scaling up **A** and correspondingly scaling down $\mathbf{\Psi}$. Basically, setting $\mathbf{\Psi} := \alpha\mathbf{\Psi}$ and **A** $:= (1/\alpha)\mathbf{A}$ (with $\alpha > 0$) does not alter (3). The choice for this additional constraint is to arbitrarily fix the norms of the columns of $\mathbf{\Psi}$ (e.g., $\|\psi_l\| = 1$ for all l, where ψ_l is the l-th column of $\mathbf{\Psi}$).

We can actually turn the optimization in (3) into a nonnegative matrix factorization problem (Lee and Seung, 1999) by rewriting the input matrix **D** as a nonnegative matrix. This reorganization consists of constructing two new matrices, $\mathbf{D}^+$ and $\mathbf{D}^-$, whose elements are only positive. The elements of $\mathbf{D}^+$ are defined as $D^+_{ln} = \max\{D_{ln},0\}$ for all l and n, while the elements of $\mathbf{D}^-$ are defined as $D^-_{ln} = \max\{-D_{ln},0\}$. The new input data matrix is

$$\tilde{\mathbf{D}} = [\mathbf{D}^+,\mathbf{D}^-]^T \ . \tag{4}$$

So we now have a data matrix of size (2L) × N instead of L × N. This implies that the size of the matrix containing the basis vectors will also increase in size from L × M to (2L) × M. We denote the new matrix of the basis vectors as $\tilde{\mathbf{\Psi}}$. It can also be written as

$$\tilde{\mathbf{\Psi}} = [\mathbf{\Psi}^+,\mathbf{\Psi}^-]^T \ , \tag{5}$$

and the matrix containing the actual basis vectors can be obtained as

$$\mathbf{\Psi} = \mathbf{\Psi}^+ - \mathbf{\Psi}^- \ . \tag{6}$$

Note that the size of **A** is unchanged. We can rewrite the minimization problem in (1) as follows:

$$E(\mathbf{A},\tilde{\mathbf{\Psi}}) = \|\tilde{\mathbf{D}} - \tilde{\mathbf{\Psi}}\mathbf{A}\|_2 \ , \tag{7}$$

subject to the constraints $\tilde{\Psi}_{lm} \geq 0$, $A_{mn} \geq 0$, and $\|\tilde{\psi}_l\| = 1$, where $\tilde{\psi}_l$ is the

l-th column of $\tilde{\mathbf{\Psi}}$. The minimization problem in (7) for $\mathbf{\Psi}$ and $\mathbf{A}$ can be solved by using, for example, an iterative gradient-descent method. Each iteration has essentially two sequential updates: (i) we update $\mathbf{A}$ while holding $\mathbf{\Psi}$ fixed:

$$A_{mn} \leftarrow A_{mn} + \eta_{mn}[(\tilde{\mathbf{\Psi}}^T\tilde{\mathbf{D}})_{mn} - (\tilde{\mathbf{\Psi}}^T\tilde{\mathbf{\Psi}}\mathbf{A})_{mn}] \quad , \tag{8}$$

and (ii) we update $\mathbf{\Psi}$ while holding $\mathbf{A}$ fixed:

$$\tilde{\Psi}_{lm} \leftarrow \tilde{\Psi}_{lm} + \mu_{lm}[(\tilde{\mathbf{D}}\mathbf{A}^T)_{lm} - (\tilde{\mathbf{\Psi}}^T\mathbf{A}\mathbf{A}^T)_{lm}] \quad , \tag{9}$$

where η_{mn} and μ_{lm} are stepsizes. The symbol $\leftarrow$ again means that the term on the right-hand side is computed and then substituted in A_{mn} for eq. (8) and in $\tilde{\Psi}_{lm}$ for eq. (9). In the gradient-descent updates, as in (8) and (9), the stepsizes are constant. Lee and Seung (1999) have proposed to select stepsizes which are not constants. They defined these stepsizes as follows:

$$\eta_{mn} = A_{mn}/(\tilde{\mathbf{\Psi}}^T\tilde{\mathbf{\Psi}}\mathbf{A})_{mn}\} \quad , \quad \text{and} \quad \mu_{lm} = \tilde{\Psi}_{lm}(\tilde{\mathbf{\Psi}}\mathbf{A}\mathbf{A}^T)_{lm} \quad . \tag{10}$$

By substituting these expressions of the stepsizes in (8) and (9), we obtain the following new update rules:

$$A_{mn} \leftarrow A_{mn}(\tilde{\mathbf{\Psi}}^T\tilde{\mathbf{D}})_{mn}/(\tilde{\mathbf{\Psi}}^T\tilde{\mathbf{\Psi}}\mathbf{A})_{mn} \quad ,$$

and (11)

$$\tilde{\Psi}_{lm} \leftarrow \tilde{\Psi}_{lm}(\tilde{\mathbf{D}}\mathbf{A}^T)_{lm}/(\tilde{\mathbf{\Psi}}\mathbf{A}\mathbf{A}^T)_{lm} \quad ,$$

which are generally known as the multiplicative rules. The multiplicative rules are more attractive for numerical implementation than the gradient rules in (8) and (9) because they are extremely simple to implement, as we can see in the Matlab code in Table 1a, and they do not require any stepsize input. Moreover, our experience suggests that the method with multiplicative-rule updates converges significantly quite faster than gradient methods.

EXAMPLES OF NONNEGATIVE CODING AND DECODING

Let us look at a numerical example. Using finite-difference modeling, we have created a 320-shot-gather dataset, with each shot having 320 receivers. Fig. 1 shows one of these shot gathers. The number of samples per trace is 625 (i.e., $L = 625$). We have selected a total of 50 shots as representative of this dataset by taking one in six shot gathers. From these 50 shot gathers, we have extracted $N = 10{,}000$ traces to form the data matrix $\tilde{\mathbf{D}}$ of size $1{,}250 \times 10{,}000$. We then use the Matlab code in Table 1a to obtain the dictionary described in Fig. 2, with $M = 625$. Fig. 2a shows the dictionary in the order it is output from the Matlab code. For clarity, we have displayed only one of four basis functions. We have even limited the number of basis functions to 10 in Fig. 2b to provide more insight into the wavelets which characterize basis functions. Notice that the basis functions are very sparse and repetitive, albeit with time shifts. Thus they can also be compressed if necessary.

In Fig. 2c we have sorted the basis functions in Fig. 2a with respect to the position of their main pulses. We can see a clear trend which covers the entire length of any given trace of our data. Our experience shows that this dictionary has enough information to describe any seismic trace of our datasets.

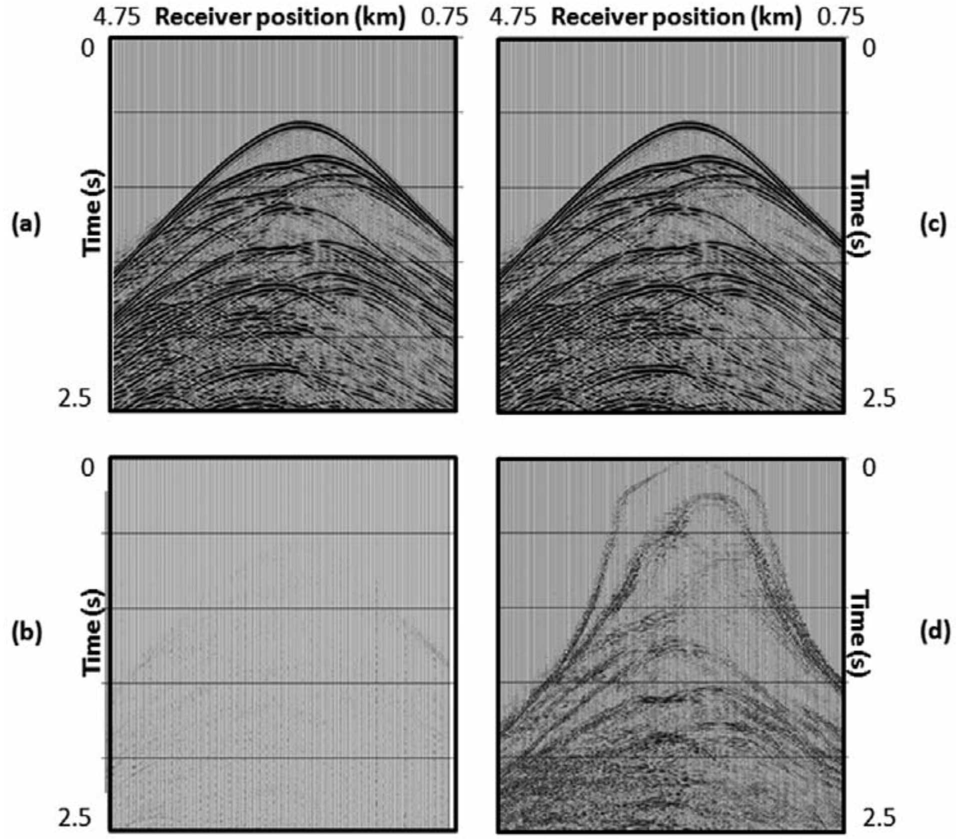
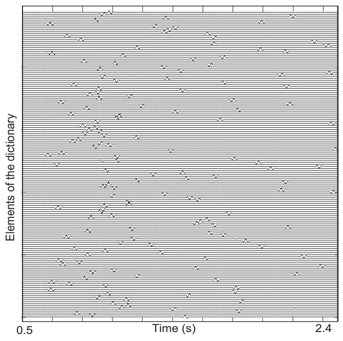
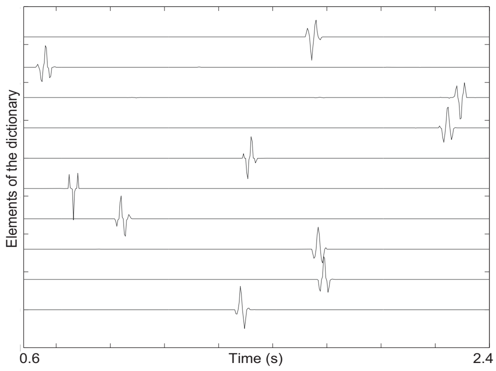


Fig. 1. (a) A shot gather of a 2D dataset before the compression process. Note that this shot gather is not included in the 50 shot gathers used in the construction of the dictionary in Fig. 2. (b) The reconstructed data using the dictionary described in Fig. 2. (c) The difference between Fig. 1a and Fig. 1b at the same scale as Fig. 1a. (d) The stochastic coefficients used in the reconstruction of Fig. 1b. The coefficients are in the same order as the dictionary in Fig. 2c.

(a)



(b)



(c)

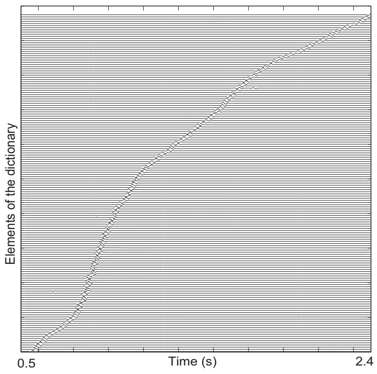


Fig. 2. Dictionary obtained by using 50 gathers of a 320-shot-gather dataset. We have taken one in six shot gathers. From these 50 shot gathers, we have extracted N = 10,000 traces. The other parameters used in the construction of the dictionary are L = M = 625. (a) The dictionary in the order it is output from the Matlab code. We have displayed only one of four basis functions. (b) The first 10 basis functions of the dictionary. (c) The dictionary sorted as a function of the position of their main pulses.

Table 1(a). A Matlab code for constructing the dictionary when working on a trace-by-trace basis.

```
function [A,S] = main(data,samples)
                    % INPUT
%data       seismic data
% samples    total number of traces to be used
                    % OUTPUT
%A        dictionary
%S        coefficients of the choosen traces

[nshots,nrecvs,nt]=size(data);
nx=1; rdim=nx*nt;

Z = sampdata( data, samples,nt,nshots);
X = [Z+abs(Z); abs(Z)-Z]/2;

niter = 150; sources = nt*nx;
[A,S] = sparc(X, sources, niter);

function X = sampdata(data,samples,nt,nsh);
% INPUT    nt  nr. of samples in a trace
%         nsh       nr. of shots
% OUTPUT   X        patches organized as
column vectors

nsmp = floor(samples/nsh);

% Initialize the matrix to hold the traces
X = zeros(nt,nsmp*nsh); smpn = 1;
for i=1:nsh
 % Load a single-shot image
 I = squeeze( data(i,:,:));

 sizex = size(I,2); sizey = size(I,1);
 posy = floor(rand(1,nsmp)*sizey)+1;
 for j=1:nsmp
   X(:,smpn) = (I(posy(1,j),:));    smpn=smpn+1;
 end
end
```

```
function  [A,S]= sparc(X,sources,niter);
% INPUT:
% X      non-negative patches as column vectors

[dims, samples] = size(X);
% Initializing
A = abs(randn(dims,sources));
A = A./(ones(dims,1)*sqrt(sum(A.^2)));
S = abs(randn(sources,samples));
S = S./(sqrt(sum(S.^2,2))*ones(1,samples));

% Loop
iter = 0; while iter < niter
obj = sum(sum((X-A*S).^2))/samples;

 % Update S and A with multiplicative steps
 S = S.*(A'*X)./(A'*A*S);
 A = A.*(X*S')./(A*S*S');

 % Normalize columns of A (and scale rows of S
correspondingly)
 scaling = sqrt(sum(A.^2));
 A = A./(ones(size(A,1),1)*scaling);
 S = S.*(scaling'*ones(1,size(S,2)));

 iter = iter+1;

end

snr = sum(sum((X).^2))/sum(sum((X-A*S).^2));
['final global SNR is  ',num2str(snr)]
```

Let us turn to the reconstruction of the gathers using the dictionary in Fig. 2. The Matlab code for this reconstruction is identical to the one in Table 1a. We simply have to eliminate the update rule over $\boldsymbol{\Psi}$, which is no longer needed, and add $\boldsymbol{\Psi}$ as input. Fig. 3 shows the results of reconstruction for a shot gather not included in the 50 shot gathers used in the construction of our dictionary. By looking at the difference between the original and reconstructed data, we can see that the reconstruction is quite accurate. Similar results with an offset gather are shown in Fig. 4.

Fig. 5 shows the contributions of various basis vectors to the reconstruction of the data in Figs. 1b and 3b. We can see that traces which require 100 elements of the dictionary are rare. We estimated that we need, on average, just 40 elements for the reconstruction in Figs. 1b and 3b.

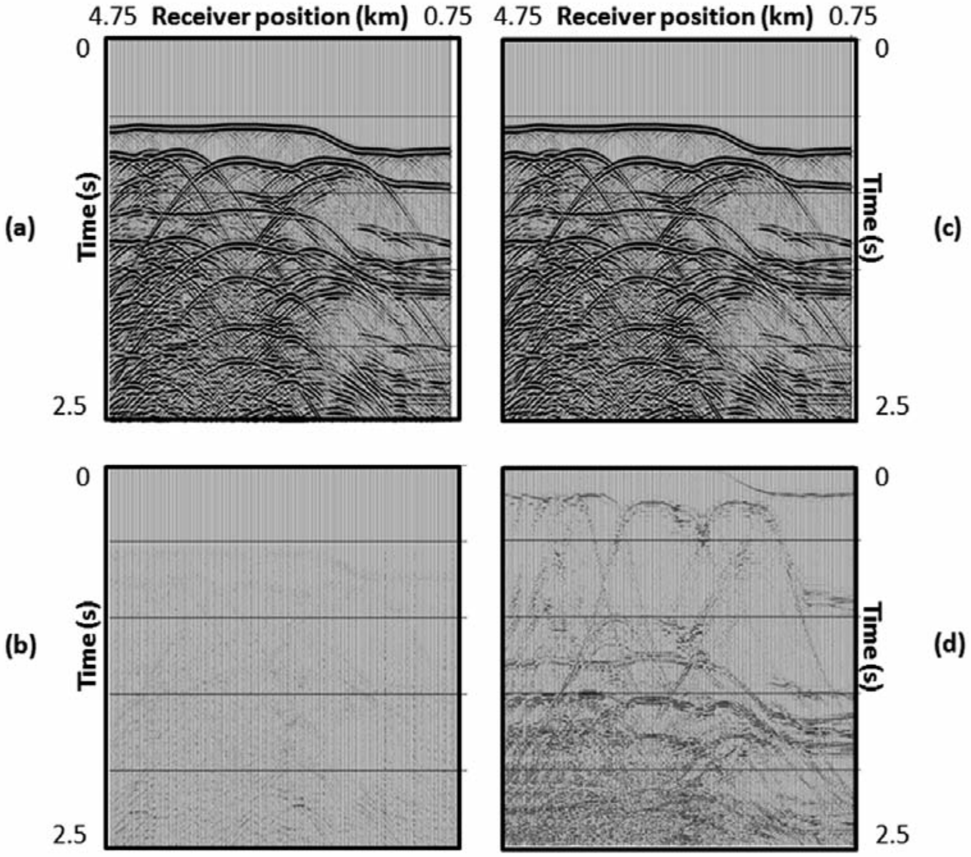


Fig. 3. (a) A zero-offset gather of a 2D dataset before the compression process. (b) The reconstructed data using the dictionary described in Fig. 2. (c) The difference between Fig. 3a and Fig. 3b at the same scale as Fig. 3a. (d) The stochastic coefficients used in the reconstruction of Fig. 3b. The coefficients are in the same order as the dictionary in Fig. 2c.

A DATA-DRIVEN TRANSFORM WITH ARBITRARY WINDOW SIDES

So far we have constructed our dictionary on a trace-by-trace basis. Actually, one can adapt the algorithm in Table 1b for any M_t traces $\times$ M_s data windows as long as the window size is small enough that we can extract a large number of windows from the data to construct an adequate dictionary of the data. However, we did not find this approach useful for data compression for reasons that we will point out later. However, we still describe this approach here because it may be useful for other seismic applications.

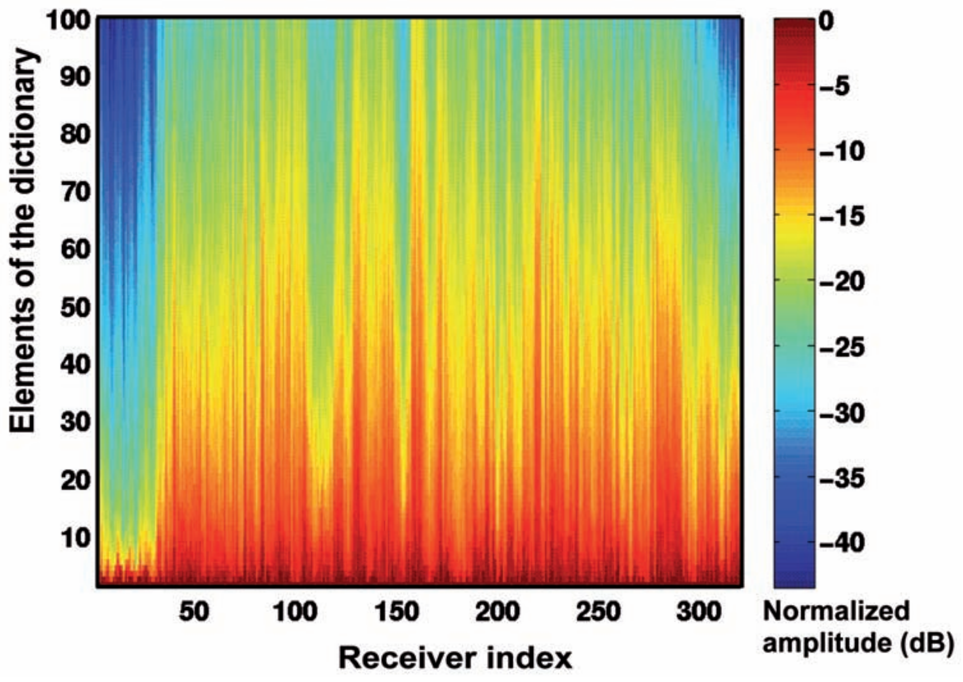


Fig. 4. The contributions of various basis vectors used in the reconstruction of the data in Fig. 1b. The horizontal axis of this plot describes the receiver points, and the vertical axis represents the index of the basis vector. The color describes the contributions of various basis vectors. For each receiver, only the first significant 100 basis vectors from the 625 basis vectors are shown.

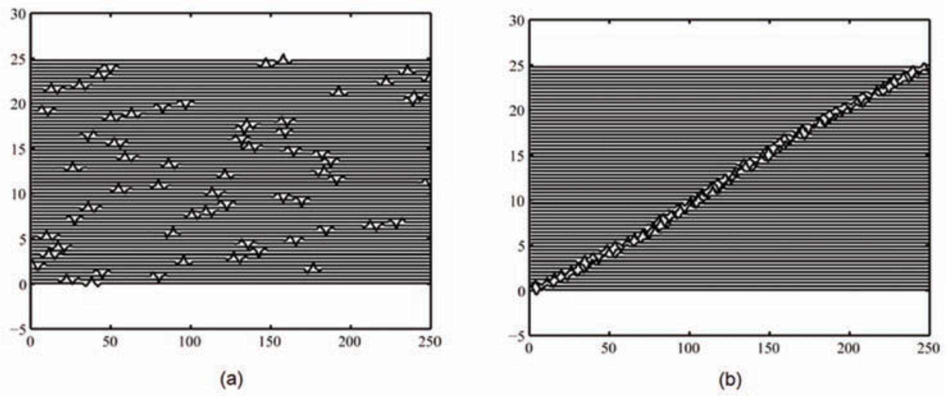


Fig. 5. A dictionary obtained using 10 gathers of a 320-shot-gather dataset. We have taken one of every six shot gathers from the first 60 shot gathers of this 2D dataset. From these 10 shot gathers, we have extracted N = 10,000 windows of one trace × 250 time samples. The other parameters used in the construction of the dictionary are L = M = 250. (a) The dictionary in the random order in which it is obtained from the Matlab code. (b) The dictionary sorted as a function of the position of the main pulses of the basis functions.

Table 1(b). A Matlab code for constructing the dictionary when working on a window-by-window basis.

```
function [A,S] = mainpatch(data,samples)
                % INPUT
%data           seismic data
% samples       total number of traces to be used
%nx, nt         dimensions of the window
                % OUTPUT
%A        dictionary
%S        coefficients of the choosen traces

[nshots,nrecvs,nt]=size(data);
rdim=nx*nt;

Z = sampdatapatch( data, samples,nt,nshots);
X = [Z+abs(Z); abs(Z)-Z]/2;

niter = 1000; sources = nt*nx;
delta=0.05
[A,S] = sparc(X, sources, niter, delta);

function X =
sampdataptch(data,samples,nx,nt,nsh);

                % INPUT
%nt             nr. of samples in a trace
% nsh           nr. of shots
                %  OUTPUT
% X             patches as column vectors
```

```
nsmp = floor(samples/nsh);

% Initialize the matrix to hold the traces
X = zeros(nt*nx,nsmp*nsh); smpn = 1;
for i=1:nsh
  % Load a single-shot image
  I = squeeze( data(i,:,:));

  sizex = size(I,2); sizey = size(I,1);

  posx = floor(rand(1,3*nsmp)*(sizex-nt-2))+1;
  posy = floor(rand(1,3*nsmp)*(sizey-nx-1))+1;
iter=1;
  for j=1:3*nsmp
    nor=norm(I(posy(1,j):posy(1,j)+nx-...
1,posx(1,j):posx(1,j)+nt-1));
    if nor>1
    X(:,smpn) = reshape((I(posy(1,j):posy(1,j)+nx-...
1,posx(1,j):posx(1,j)+nt-1)),nx*nt,1);
    smpn=smpn+1; iter=iter+1;
    end
    if iter>nsmp; break; end
  end
end

end
```

Computationally, the starting point is to select about 50 or so multishot gathers which are representative of the multishot dataset that we are dealing with. From these gathers, we randomly select a large number of M_t traces $\times$ M_s data windows. We will represent data in each window as an M-dimensional (with $M = M_t * M_s$) vector that we denote $\mathbf{d}_n = [d_{1n}, d_{2n}, \ldots, d_{mn}]^T$, where n indicates the window under consideration. The index n varies from 1 to N, with N being the number of M windows extracted from the 50 gathers. So we can also represent the basic functions as M-dimensional vectors. We denote these vectors as $\psi_p = [\psi_{1p}, \psi_{2p}, \ldots, \psi_{Mp}]^T$. Thus the dictionary can now be described as an M $\times$ M matrix:

$$\boldsymbol{\Psi} = [\psi_1 | \psi_2 | \ldots | \psi_M] \quad . \tag{12}$$

We can group the data vectors $\mathbf{d}_n$ into an M $\times$ N matrix that we will denote $\mathbf{D}$, and similarly, the stochastic coefficients can be grouped into an M $\times$ N matrix that we denote $\mathbf{A}$. We arrive again at eq. (2) by replacing L by M.

The Matlab code in Table 1b can also be used for reconstructing $\mathbf{\Psi}$ and $\mathbf{A}$. Let us now look at an example of data reconstruction using windows of one trace $\times$ 250 time samples. We considered 10 of the 50 shot gathers used in Fig. 1. We extracted 10,000 windows of the input shot gathers. The data in each window were thus represented by a 2 $\times$ 250-dimensional vector, as described earlier. The size of the resulting data matrix is 500 $\times$ 10,000. We then ran the Matlab code in Table 1b. The columns of the matrix $\mathbf{\Psi}$, which constitute our basis functions, are shown in Fig. 5. We can see that events in each window resemble those of the actual data, thus confirming that the nonnegative matrix factorization allows us to reconstruct parts of the original data.

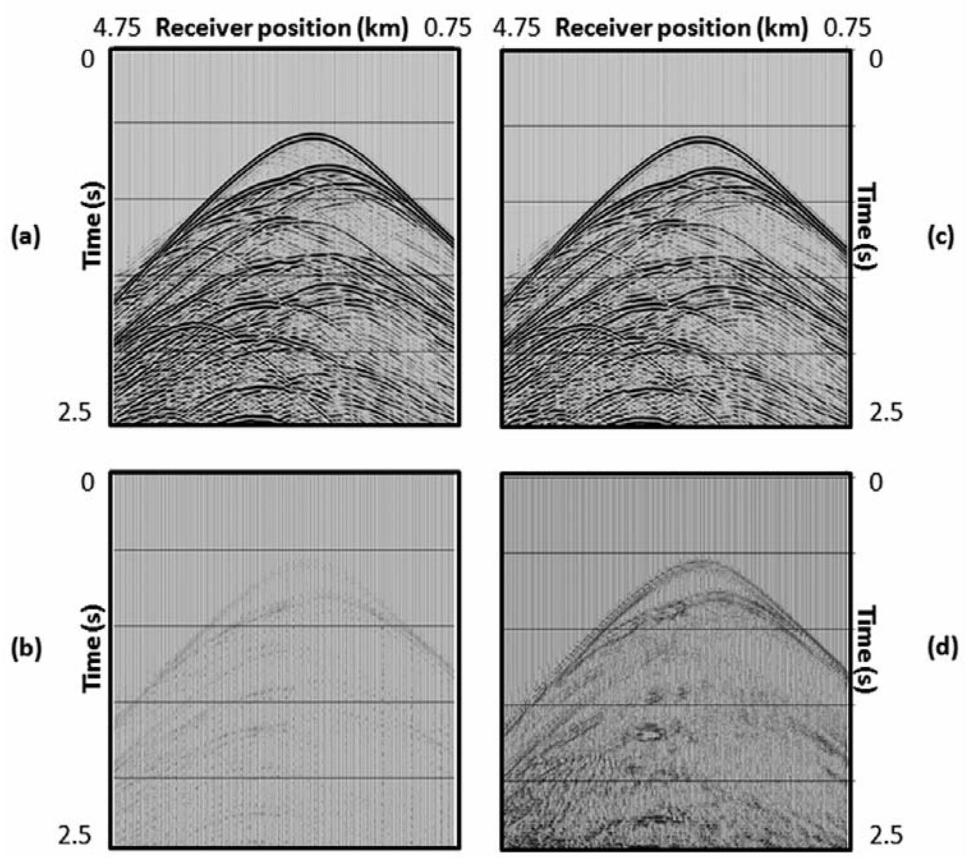


Fig. 6. (a) A shot gather of a 2D dataset before the compression process. (b) The reconstructed data using the dictionary described in Fig. 5.(c) The difference between Fig. 6a and Fig. 6b at the same scale as Fig. 6a. (d) The average of stochastic coefficients used in the reconstruction of Fig. 6b. The coefficients are in the same order as the dictionary in Fig. 5b.

Let us now turn to the reconstruction. We have displayed in Fig. 6 the shot gather that we wish to reconstruct. We started by decomposing this shot gather into 10,000 overlapping windows of one trace $\times$ 250 time samples. The overlapping between windows is necessary to avoid the numerical-edge effects associated with the fact that several windows are needed here to reconstruct a seismic trace. At the given data point, this overlapping allows us to average the reconstructed results associated with different windows. Hence, the edge effects are hardly visible in the reconstructed data because of these averages, as we can see in Fig. 6b. Clearly, the reconstructed results are satisfactory. It is difficult to tell reconstructed gathers from the original gathers with only the naked eye. The idea of working with the windows of the data instead of traces is not attractive for data compression because it requires a large number of overlapping windows to avoid edge effects, as a large number of windows implies a large number of coefficients for reconstructing the data. For example, when working on a trace-by-trace basis, we need at most a matrix of 625 $\times$ 320 (including the zero values) to store the stochastic coefficients for one shot gather. With a window of one trace $\times$ 250 time samples, we need a matrix of 250 $\times$ 10,000 (including the zero values) to store the stochastic coefficients of the same shot gather. In other words, the larger numbers of windows needed to store stochastic coefficients rendered the option of using the data-driven transform described here less attractive when working on a window-by-window basis instead of a trace-by-trace basis. Yet we have included the window option of this transform here because it may turn out to be useful in other seismic applications. Note that one can also construct windows with more than one trace. For example, we constructed a dictionary with square windows of 20 traces $\times$ 20 time samples, as illustrated in Fig. 7. Basically, we extracted 10,000 windows of the input shot gathers. The data in each window were thus represented by a 2 $\times$ 20 $\times$ 20-dimensional vector, as described earlier. The size of the resulting data matrix is 800 $\times$ 10,000. We then ran the Matlab code in Table 1b to obtain the dictionary in Fig. 7. Note that we obtained the results in Fig. 7 as follows. We first constructed $\tilde{\mathbf{\Psi}}$ using the code in Table 1. We then used eq. (6) to obtain the matrix $\mathbf{\Psi}$ of the basis vectors. Finally, we transformed each column of $\mathbf{\Psi}$ into square windows, which are displayed in Fig. 7. We can see that the events in each window resemble those of the actual data, thus confirming that the nonnegative matrix factorization allows us to reconstruct parts of the original data.

COMPRESSION RATIO

To increase the compression rate, we propose to combine several compression techniques, namely acquisition-driven, data-driven, and arithmetic compression. In Ikelle (2009) we described how the concept of multishooting can be used to collect compression. The basic idea of multishooting is that seismic waves can be generated from several locations simultaneously (or nearly simultaneously, by introducing small time delays between the shooting points) instead of one single-source location at a time, as is currently the case. Thus four shot gathers acquired simultaneously are equivalent to a compression ratio of 4:1. We will call the compression achieved by multishooting acquisition-driven compression. If we then apply the data-driven compression to multishot data, even with a compression ratio of, say, 10:1, we are already at 40:1. Moreover, the same dictionary can be used to compress other lines in a 3D survey which may be adjacent to the line used to create the dictionary. Using arithmetic compression for the ratio of, say, 3:1, we are already above 100:1.

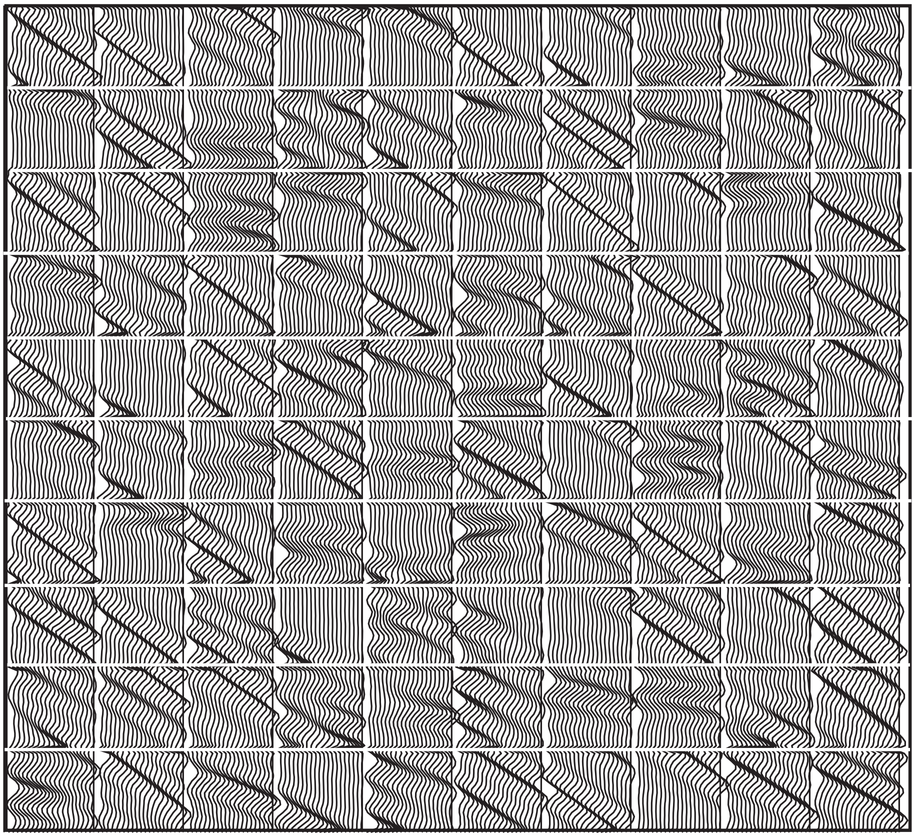


Fig. 7. A dictionary obtained using 10 gathers of a 320-shot-gather dataset. We have taken one of every six shot gathers from the first 60 shot gathers of this 2D dataset. From these 10 shot gathers, we have extracted N = 10,000 windows of 20 traces × 20 time samples. The other parameters used in the construction of the dictionary are L = M = 400.

CONCLUSIONS

We have described data-driven compression based on a data-driven transform. Besides the additional increase in compression ratio, the technique offers petroleum seismologists a new framework for investigating data compression. The dictionary of these new transforms is driven by seismic data rather than an arbitrary mathematical dictionary.

ACKNOWLEDGMENTS

We would like to thank the sponsors of the CASP Project for their comments and suggestions during the review process.

REFERENCES

Bosman, C. and Reiter, E., 1993. Seismic data compression using wavelet transforms. Expanded Abstr., 63th Ann. Internat. SEG Mtg., Washington D.C.: 1261-1264.

Donoho, P.L., Eagrs, R.A. and Villasenor, J.D., 1995. High performance seismic trace compression. Expanded Abstr., 65th Ann. Internat. SEG Mtg., Houston: 160-163.

Ikelle, L.T., 2009. Coding and Decoding: Seismic Data. Elsevier Science Publishers, Amsterdam (in press).

Lee, D.D. and Seung, H.S., 1999. Learning the parts of objects by non-negative matrix factorization. Nature, 401: 788-791.

Luo, Y. and Schuster, G.T., 1992. Wavelet packet transform and data compression. Expanded Abstr., 62th Ann. Internat. SEG Mtg., New Orleans: 1187-1190.

Spanias, A.S., Jonsson, S. and Stearns, S.D., 1991. Transform methods for seismic data compression. IEEE Transact. Geosci. Remote Sens., 29: 407-416.

Tage, R., Ramstad, T.A. and Amundsen, L., 2004. Optimization of sub-band coding method for seismic data compression. Geophys. Prosp., 52: 359-378.