Trace interpolation using a recursive dip filter

Jun Ji and Jon Claerbout [1]

ABSTRACT

A scheme for missing-trace interpolation of a common midpoint (CMP) gather is proposed. Spatial aliasing in a CMP gather can be overcome by normal-moveout (NMO) correction. After NMO correction, we fill in missing traces so as to minimize the output of high-pass, recursive dip filtering. For optimization using a conjugate gradient, we derive the conjugate operator of the recursive dip filter. The proposed scheme is applied to three kinds of missing data problems - interlaced missing traces, truncation at the end of survey and randomly positioned missing traces.

INTRODUCTION

Trace interpolation procedures are becoming an integral part of processing coarsely or irregularly sampled field data. Generally speaking, trace interpolation is used to improve the quality of multitrace processing, especially that of migration. All standard methods of migration yield poor results when sampling is too coarse, although each method causes different types of errors. In this paper, we will consider CMP gathers that are regularly sampled in space, but have some number of missing traces. The missing traces can occur at the end of the CMP gather, they can be the result of spatial aliasing (interlaced missing traces), or they can be missing traces located randomly throughout the gather.

First we will review the recursive dip filter, including its character, implementation and conjugate operator. We will then discuss the dealiasing effect of NMO correction and the spectral characteristics of various missing-trace patterns. The final section will demonstrate the validity of the method given in this paper using a synthetic CMP gather and real data.

[1] **email:** Author has no known email address.

REVIEW OF CLAERBOUT'S RECURSIVE DIP FILTER

The recursive dip filter, invented by Claerbout (1985), works in the time and space domain. It is more attractive for seismic data processing than a dip filter that operates in the frequency-wavenumber domain, because it can be temporally and spatially nonstationary. In addition to time and space variability, the recursive dip filter offers the advantage of a simple and economic recursive implementation.

Let P denote raw data and Q denote filtered data in the frequency-wavenumber domain. Dip filtering can be achieved by using a low-dip-pass filter:

$$Q(\omega, k) = \frac{\alpha}{\alpha + (\frac{k^2}{-i\omega})} P(\omega, k), \tag{1}$$

or a high-dip-pass filter:

$$Q(\omega, k) = \frac{(\frac{k^2}{-i\omega})}{\alpha + (\frac{k^2}{-i\omega})} P(\omega, k), \tag{2}$$

where ω is the temporal frequency, k is the spatial frequency, and α determines the cutoff dip (Claerbout, 1985).

To understand the recursive dip filter, we need to examine its amplitude spectrum. Figure 1 shows the amplitude spectrum of a high-dip-pass filter. The spectrum changes slowly from a pass zone to a reject zone. This is the only drawback of recursive filters, however, the transition zone need not be short in our application to trace interpolation.

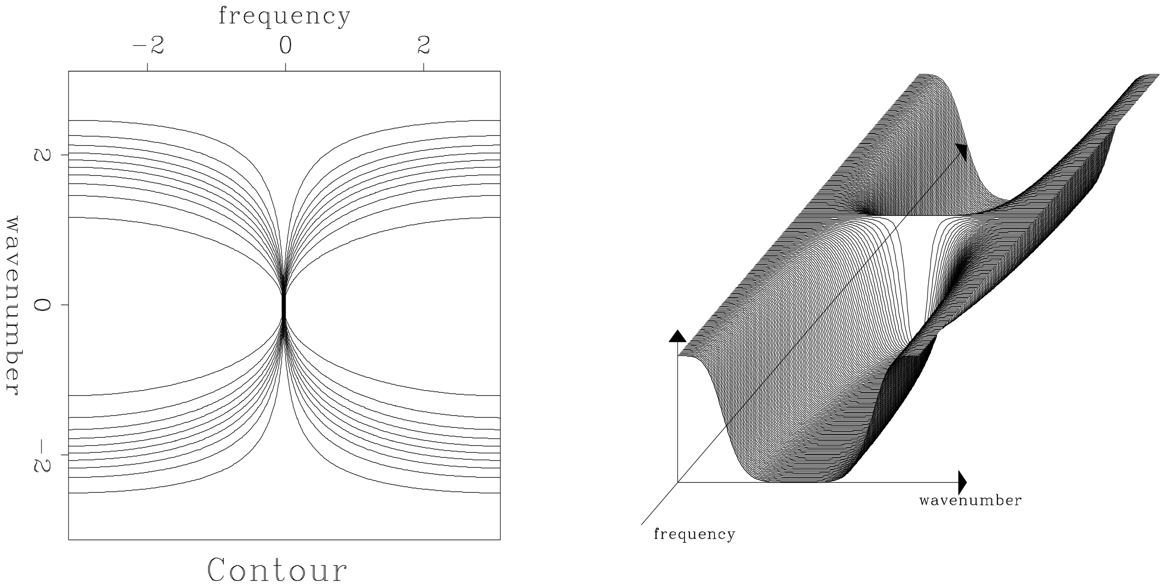

Figure 1: Left: contour plot of the amplitude spectrum of a high-pass recursive dip filter with $\alpha = 0.5$ Right: 3D plot of the amplitude spectrum of the same filter. $\boxed{\text{fig1}}$ [R]

The realization of equations (1) and (2) is obtained by replacing $-i\omega$ and $-k^2$ with $\frac{\partial}{\partial t}$ and $\frac{\partial^2}{\partial x^2}$, respectively. Clearing out all the fractions in equation (1) then leads t the following partial-differential equation:

$$\alpha(\frac{\partial q}{\partial t} - \frac{\partial p}{\partial t}) = \alpha\frac{\partial^2 q}{\partial x^2}. \tag{3}$$

Approximating the derivative by a difference operator gives the desired recursive relations (Claerbout, 1985). For example, the low-pass filter becomes

$$(\frac{\alpha}{\Delta t} + \frac{1}{2\Delta x^2}T)q_{t+1} = (\frac{\alpha}{\Delta t} - \frac{1}{2\Delta x^2}T)q_t + \frac{\alpha}{\Delta t}(p_{t+1} - p_t), \tag{4}$$

where T represents a tridiagonal matrix with (1, -2, 1) along the diagonal, and α determines the cutoff dip. For a stable and accurate implementation, we apply the Crank-Nicholson method and the 1/6 trick. The result is the following differencing star:

$(-1-\beta)q_t^{x-1} + p_t^{x-1}$	$(-\frac{1}{b} + 2 + 2\beta)q_t^x + (\frac{1}{b} - 2)p_t^x$	$(-1-\beta)q_t^{x+1} + p_t^{x+1}$
$(1-\beta)q_{t+1}^{x-1} - p_{t+1}^{x-1}$	$(\frac{1}{b} - 2 + 2\beta)q_{t+1}^x + (\frac{1}{b} - 2)p_{t+1}^x$	$(1-\beta)q_{t+1}^{x+1} - p_{t+1}^{x+1}$

where $\beta = \frac{\Delta t}{2\alpha b \Delta x^2}$ and $b = \frac{1}{6}$.

Implementation of the above differencing star is straightforward. However, in order to develop the forward and transpose operators needed for iterative application, we need to write these operators in a matrix form as shown in the following section.

CONJUGATE OPERATOR

The strategy for interpolation of missing traces is an inversion which determines the missing traces so as to minimize the high-dip-pass filtered output. For optimization, we use a conjugate-gradient algorithm. Conjugate-gradient inversion is an iterative method in which each iteration involves the application of a forward operator and its transpose.

The forward operator of a recursive dip filter can be represented as

$$Aq = Bp \quad or \quad q = A^{-1}Bp, \tag{5}$$

where A and B are matrices as follows:

$$
\left[\begin{array}{cccc|cccc|cccc}
b_1 & a_1 & . & . & . & . & . & . & . & . & . & . \\
a_1 & b_1 & a_1 & . & . & . & . & . & . & . & . & . \\
. & a_1 & b_1 & a_1 & . & . & . & . & . & . & . & . \\
. & . & a_1 & b_1 & . & . & . & . & . & . & . & . \\
\hline
b_2 & a_2 & . & . & b_1 & a_1 & . & . & . & . & . & . \\
a_2 & b_2 & a_2 & . & a_1 & b_1 & a_1 & . & . & . & . & . \\
. & a_2 & b_2 & a_2 & . & a_1 & b_1 & a_1 & . & . & . & . \\
. & . & a_2 & b_2 & . & . & a_1 & b_1 & . & . & . & . \\
\hline
. & . & . & . & b_2 & a_2 & . & . & b_1 & a_1 & . & . \\
. & . & . & . & a_2 & b_2 & a_2 & . & a_1 & b_1 & a_1 & . \\
. & . & . & . & . & a_2 & b_2 & a_2 & . & a_1 & b_1 & a_1 \\
. & . & . & . & . & . & a_2 & b_2 & . & . & a_1 & b_1
\end{array}\right]
\left[\begin{array}{c}
q_1^1 \\ q_1^2 \\ q_1^3 \\ q_1^4 \\ \hline q_2^1 \\ q_2^2 \\ q_2^3 \\ q_2^4 \\ \hline q_3^1 \\ q_3^2 \\ q_3^3 \\ q_3^4
\end{array}\right]
=
$$

$$
\left[\begin{array}{cccc|cccc|cccc}
d_1 & c_1 & . & . & . & . & . & . & . & . & . & . \\
c_1 & d_1 & c_1 & . & . & . & . & . & . & . & . & . \\
. & c_1 & d_1 & c_1 & . & . & . & . & . & . & . & . \\
. & . & c_1 & d_1 & . & . & . & . & . & . & . & . \\
\hline
d_2 & c_2 & . & . & d_1 & c_1 & . & . & . & . & . & . \\
c_2 & d_2 & c_2 & . & c_1 & d_1 & c_1 & . & . & . & . & . \\
. & c_2 & d_2 & c_2 & . & c_1 & d_1 & c_1 & . & . & . & . \\
. & . & c_2 & d_2 & . & . & c_1 & d_1 & . & . & . & . \\
\hline
. & . & . & . & d_2 & c_2 & . & . & d_1 & c_1 & . & . \\
. & . & . & . & c_2 & d_2 & c_2 & . & c_1 & d_1 & c_1 & . \\
. & . & . & . & . & c_2 & d_2 & c_2 & . & c_1 & d_1 & c_1 \\
. & . & . & . & . & . & c_2 & d_2 & . & . & c_1 & d_1
\end{array}\right]
\left[\begin{array}{c}
p_1^1 \\ p_1^2 \\ p_1^3 \\ p_1^4 \\ \hline p_2^1 \\ p_2^2 \\ p_2^3 \\ p_2^4 \\ \hline p_3^1 \\ p_3^2 \\ p_3^3 \\ p_3^4
\end{array}\right]
$$

where

$$
\begin{aligned}
a_1 &= \quad 1-\beta & b_1 &= \frac{1}{b} - 2 + 2\beta \\
a_2 &= \quad -1-\beta & b_2 &= -\frac{1}{b} + 2 + 2\beta \\
c_1 &= \quad 1 & d_1 &= -\frac{1}{b} + 2 \\
c_2 &= \quad -1 & d_2 &= -\frac{1}{b} + 2
\end{aligned}
$$

and the subscript and the superscript in p_j^i and q_j^i represent offset-axis and time-axis indices respectively.

The transpose operator of equation (5) is defined by

$$p = B^T (A^T)^{-1} q. \tag{6}$$

In actual calculations, instead of inverting the matrix, we can use a tridiagonal solver, as follows:

$$p = B^T r \quad where \quad A^T r = q.$$

In the appendix, we list the program that applies the operator and its conjugate operator, along with the result of the dot-product test for demonstrating conjugacy (Claerbout, 1991).

NMO CORRECTION AND SPECTRA OF MISSING TRACES

One of the most important considerations in interpolation is to avoid interpolating aliased events. It is very difficult to distinguish steeply dipping events from aliased data. NMO processing provides a good means of moving the steeply dipping events to the nonaliased dip range of the frequency-wavenumber domain. Figure 2 shows the spectrum of a CMP gather before and after NMO correction. Notice that the NMO correction has significantly reduced the steep dips.

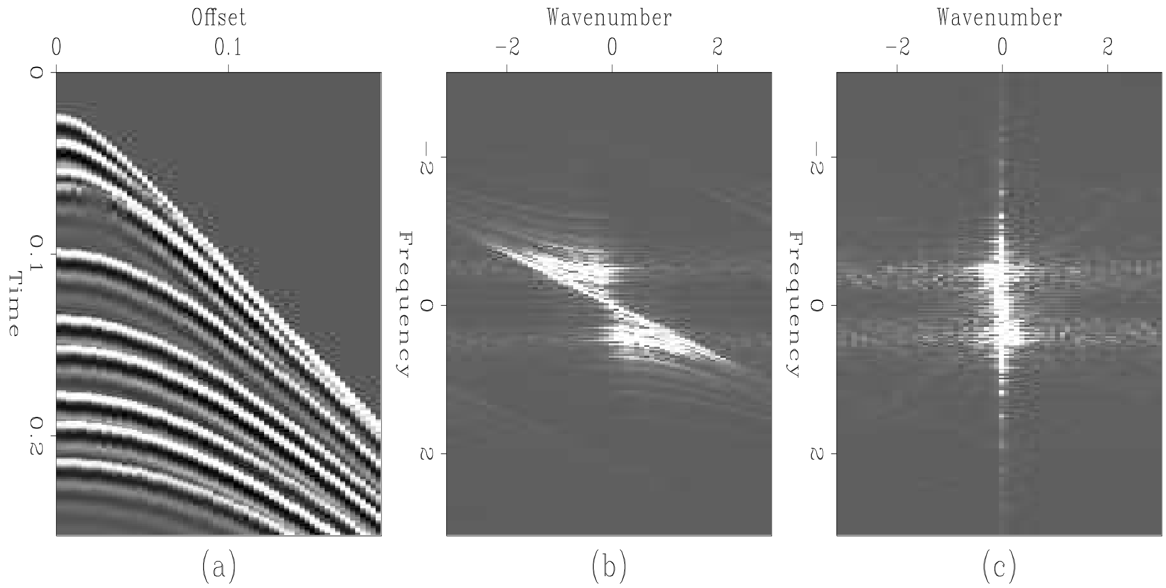

Figure 2: (a) Synthetic CMP gather, (b) amplitude spectrum of the CMP gather, and (c) amplitude spectrum of the NMO-corrected CMP gather. fig2 [R]

If there existed any missing trace, its spectrum would spread over a wide wavenumber range. Figure 3 shows three amplitude spectra, which correspond to the interlaced missing, truncated, and randomly missing CMP gathers, respectively. Among the three spectra, only the amplitude spectrum of the interlaced missing CMP gather clearly indicates that a conventional frequency-wavenumber domain interpolation method can be applied (i.e., Fourier transforming the data, applying a high-cut filter, and inverse Fourier transforming). The others, the amplitude spectra of truncated and randomly missing CMP gathers, are blurred by the missing traces, and it is difficult to find a clear-cut passband.

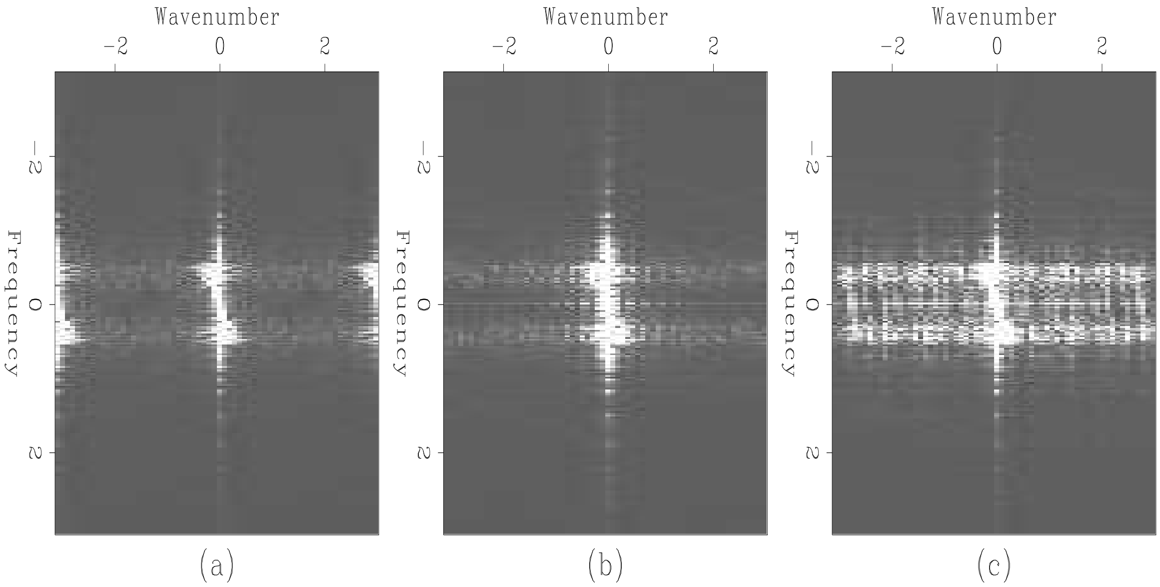

Figure 3: The amplitude spectrums of NMO corrected CMP gathers of (a) the interlaced missing, (b) the truncated missing and (c) the randomly missing traces. fig3 [R]

RESULTS

Interpolation of missing traces is carried out iteratively using a conjugate-gradient algorithm, which minimizes the high-dip-pass-filtered output of NMO-corrected CMP gathers. Finally, inverse NMO processing is applied with the same velocity as was used in the NMO correction. To preserve the amplitude of the original data, we use pseudounitary NMO processing (Claerbout, 1991).

In order to test the validity of the proposed method, it has been applied to several sets of synthetic and real data. Figure 4 shows a synthetic CMP gather with interlaced missing traces, the result of interpolation, and the difference between the original traces and the interpolated traces. Similarly, Figure 5 shows a synthetic CMP gather with truncation at the end, the result of interpolation, and the difference between the original traces and the interpolated traces. In Figure 6 a synthetic CMP gather with random missing traces is shown, along with the result of interpolation and the difference between the original traces and the interpolated traces.

Figures 7, 8 and 9 show the processing of a real dataset. The data set, wz27 (Yilmaz and Cumro, 1983), has been windowed to provide a small test dataset. Normal moveout was applied to the data using water velocity, making most events reasonably, but not perfectly, flat. Figure 7 shows the results of interpolating the dataset when odd-number traces were zapped to produce missing traces. The difference between the original traces and interpolated traces is also shown. Figure 8 shows the result of the interpolation and the interpolation error for a truncated CMP gather. Figure 9 shows the result of interpolation for random missing traces.

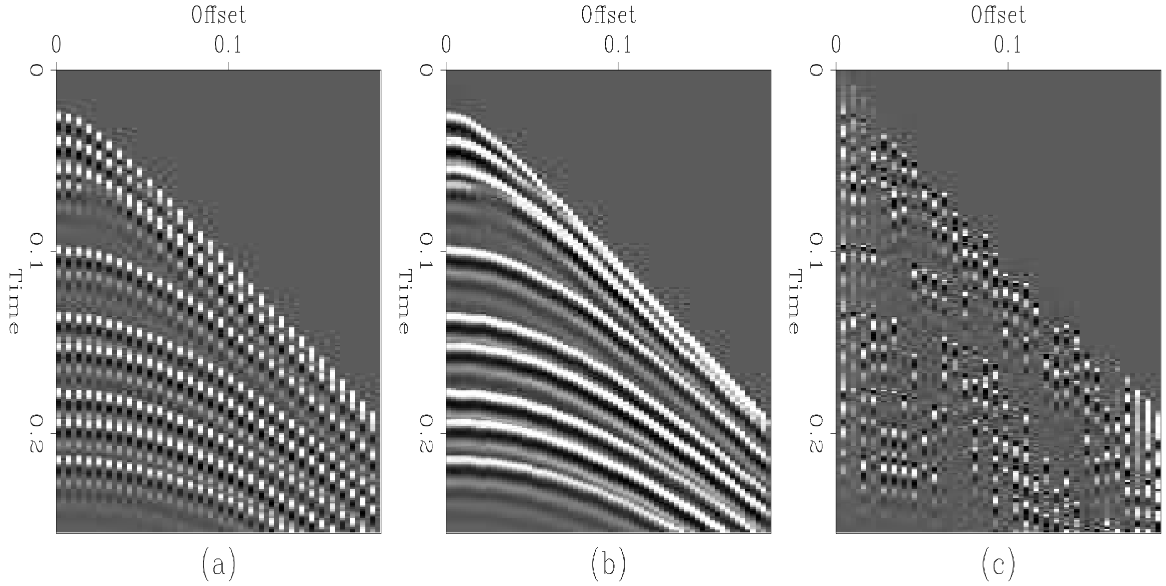

Figure 4: (a) Synthetic CMP gather with interlaced missing traces, (b) the result of interpolation, and (c) the difference between original and interpolated traces. fig4 [R]

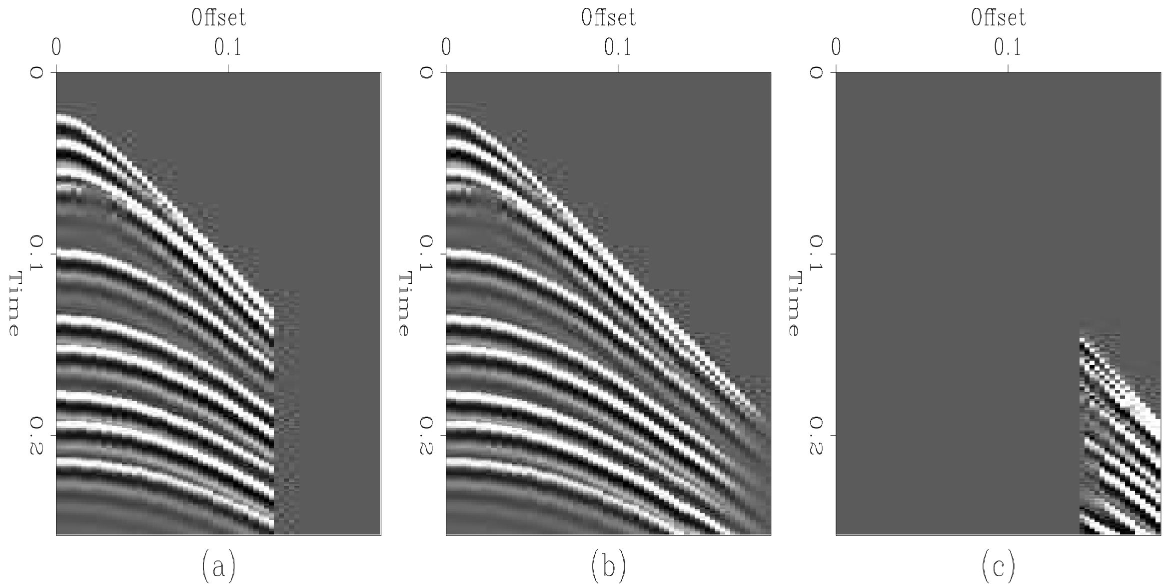

Figure 5: (a) Synthetic CMP gather with truncation at the end, (b) the result of interpolation, and (c) the difference between original and interpolated traces. fig5 [R]

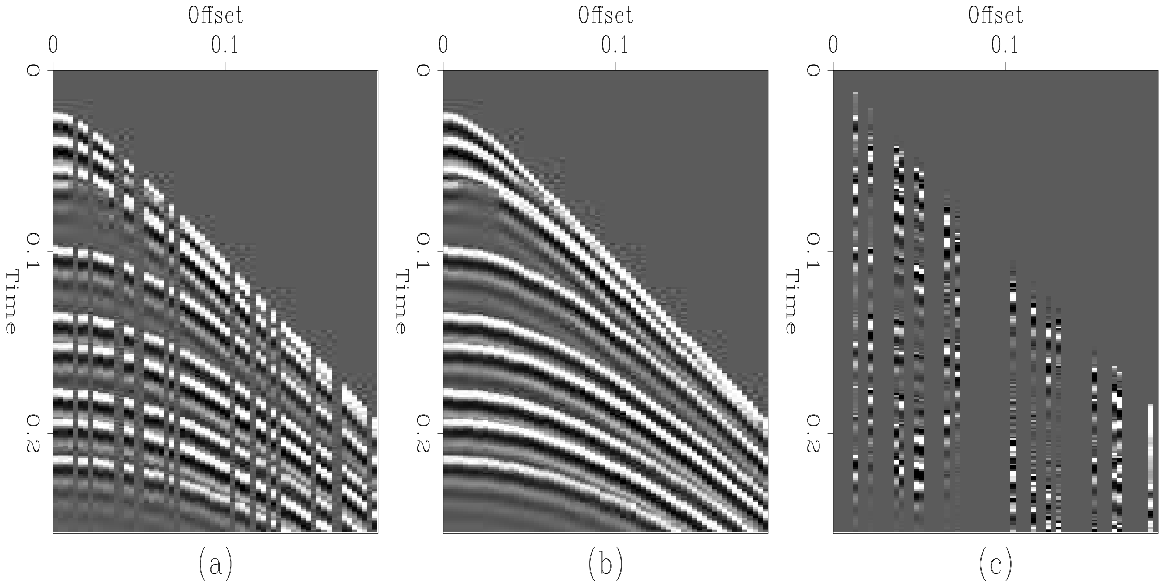

Figure 6: (a) Synthetic CMP gather with randomly missing traces, (b) the result of interpolation, and (c) the difference between original and interpolated traces. fig6 [R]

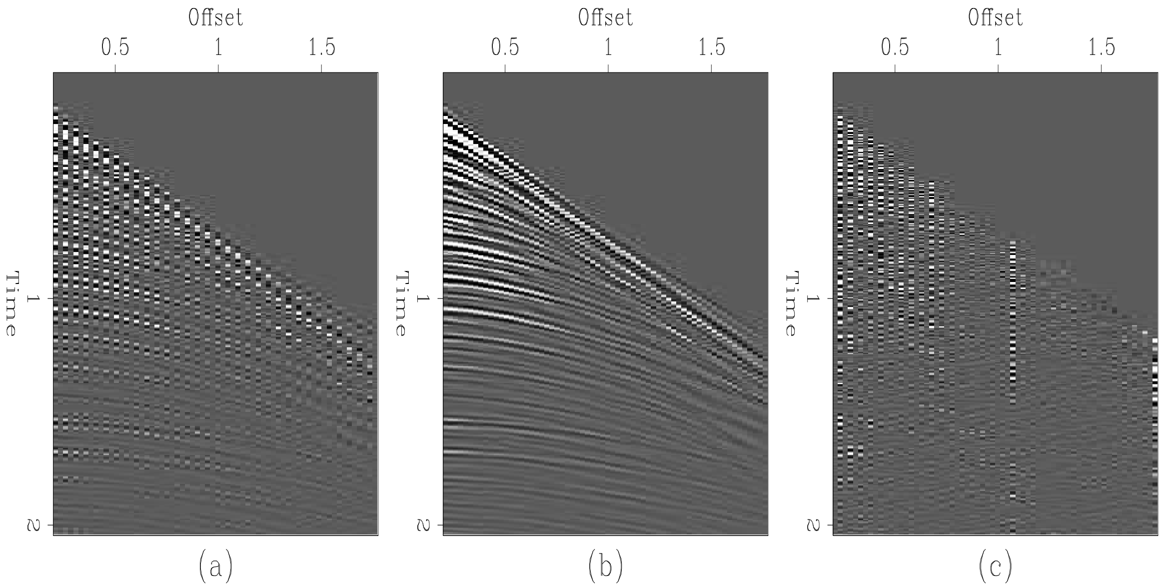

Figure 7: (a) Real CMP gather with interlaced missing traces, (b) the result of interpolation, and (c) the difference between original and interpolated traces. fig7 [R]

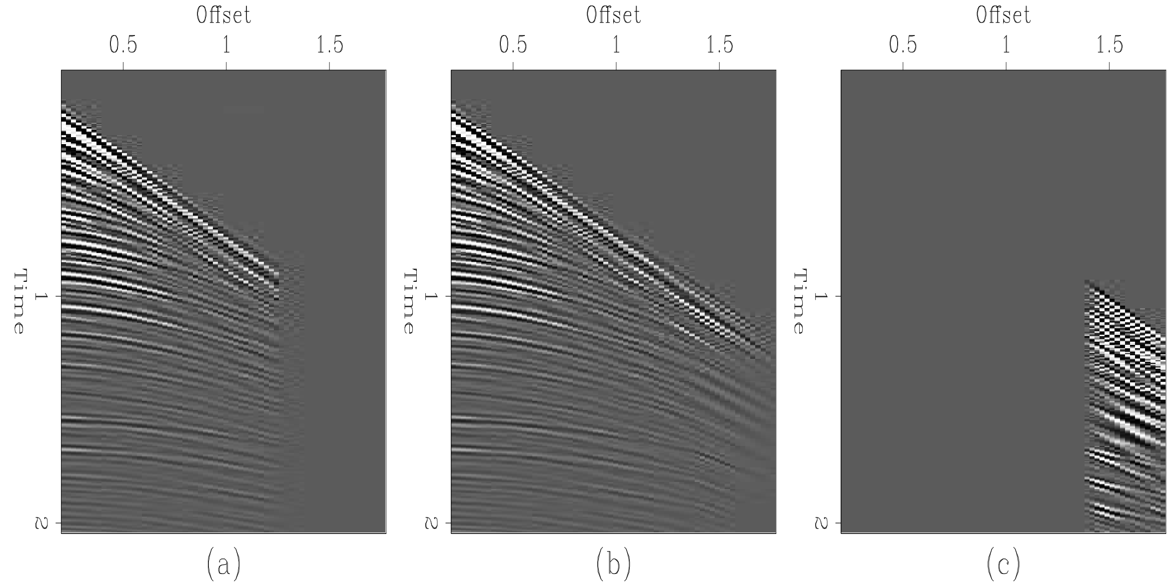

Figure 8: (a) Real CMP gather with truncation at the end, (b) the result of interpolation, and (c) the difference between original and interpolated traces. fig8 [R]

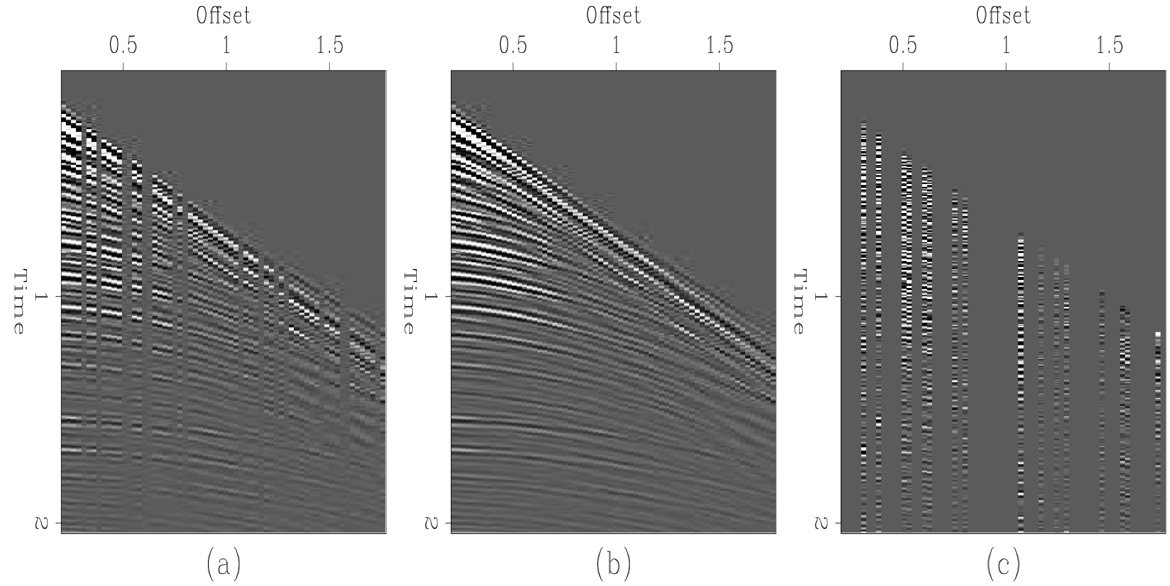

Figure 9: (a) Real CMP gather with randomly missing traces, (b) the result of interpolation, and (c) the difference between original and interpolated traces. fig9 [R]

Figure 10 shows residuals as a function of the iteration number for the three missing trace types. We see that all cases converge very fast, and that interlaced and random missing traces converge in just a few steps.

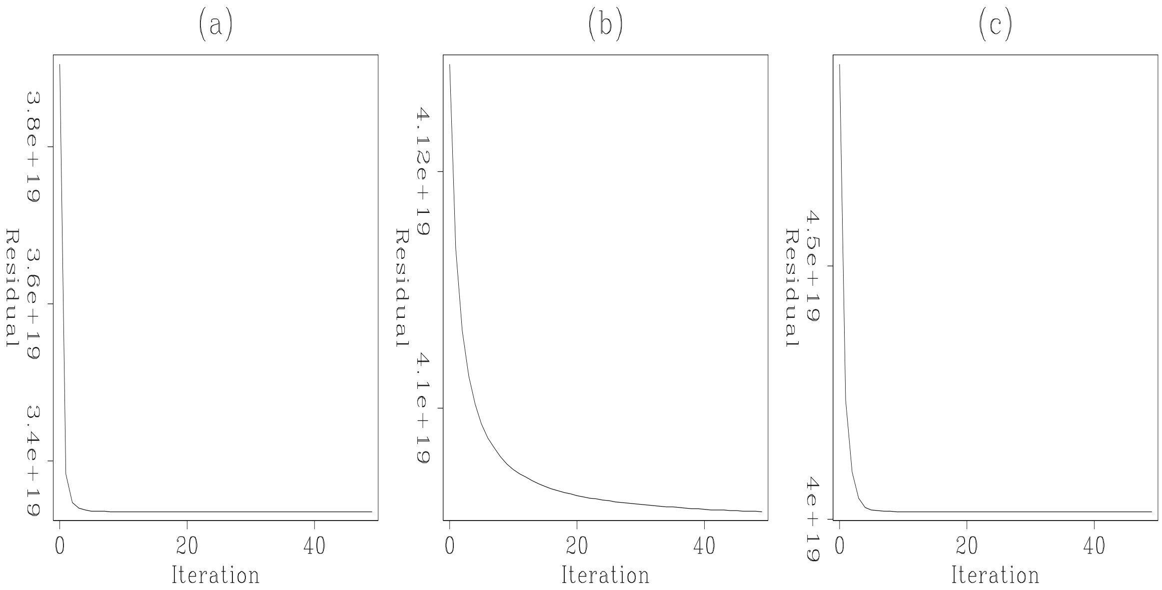

Figure 10: Residuals with respect to iteration for (a) interlaced missing traces, (b) truncation at the end, and (c) randomly missing traces. fig10 [NR]

CONCLUSION

A scheme for trace interpolation of CMP gathers was proposed. The interpolation was performed by finding the missing traces that minimize the output of applying a high-pass, recursive dip filter to the NMO-corrected gather. For optimization, we used a conjugate-gradient algorithm and found that it converged rapidly. Compared to sinc interpolation, which is equivalent to applying a high-cut filter in the frequency domain, the proposed method is superior for interpolating both truncated traces and random missing traces.

ACKNOWLEDGEMENTS

We would like to thank Steve Cole, who described his own work on recursive dip filter to us.

REFERENCES

Claerbout, J.F., 1985, Imaging the Earth's Interior: Blackwell Scientific Publications.

Claerbout, J.F., 1991, Earth Sounding Analysis: Processing versus Inversion: SEP–**71**.

Yilmaz, O., and Cumro, D., 1983, Worldwide assortment of field seismic records, released by Western Geophysical Company of America, Houston.

APPENDIX

```
subroutine recdfilt( conj, hilo, alpha, nt,nx, p, q)
#
#---------------------------------------------------------------------------
#       recursive dip filter
#---------------------------------------------------------------------------
#       conj    : forward operator [ = 0 ] conjugate operator [ = 1 ]
#       hilo    : low-dip pass filter [ = 0 ] high-dip pass filter [ = 1 ]
#       alpha   : cutoff dip
#       nt,nx   : dimension of array
#       p       : input data
#       q       : output data
#
#---------------------------------------------------------------------------
integer it,ix, nt,nx
real p(nt,nx), q(nt,nx)
real bl,br,endl,endr
real conj, hilo, alpha, beta
real p11,p12,p13,q11,q12,q13,p21,p22,p23,q21,q22,q23
temporary real r(nt,nx),px(nx),rhs(nx),sol(nx),e(nx),f(nx)

beta = 1./6.726

q11 =-1.-alpha      ;   q12 = -1./beta+2.+2.*alpha      ;q13 = -1.-alpha
p11 =-1.*(1.)       ;   p12 = -1.*(1./beta-2.)          ;p13 = -1.*(1.)
q21 = 1.-alpha      ;   q22 = 1./beta-2.+2.*alpha       ;q23 = 1.-alpha
p21 = -1.*(-1.)     ;   p22 = -1.*(-1./beta+2.)         ;p23 = -1.*(-1.)

# boundary condition
bl = 0.;        br = 0.
endl = q23*bl + q22;    endr = q21*br + q22

if( conj == 0. ) {
        do it= 1, nt {
        do ix= 1, nx {
                q(it,ix) = 0.; r(it,ix) = 0.;  rhs(ix)=0.
                }}
        call bicon(0.,bl,p11,p12,p13,p21,p22,p23,nt,nx,p,r)
        do it= 1, nt {
                if(it==1) { do ix= 1, nx { rhs(ix) = r(it,ix) } }
                else {  call tricon(bl,q11,q12,q13,nx,sol,rhs)
                        do ix= 1, nx { rhs(ix) = r(it,ix) - rhs(ix)}
                        }
                call rtris(nx,endl,q21,q22,q23,endr,rhs,sol,e,f)
                do ix = 1,nx{ q(it,ix) = sol(ix) }
                }
        if( hilo==0.)
        do it= 1, nt {  do ix= 1, nx { q(it,ix) = q(it,ix) }}
        else
        do it= 1, nt {  do ix= 1, nx { q(it,ix) = p(it,ix) - q(it,ix) }}
        }
else    {
        do it= 1, nt {
        do ix= 1, nx {
                p(it,ix) = 0.; r(it,ix) = 0.; rhs(ix)=0.
                }}
        do it= nt, 1, -1 {
                if(it==nt) { do ix= 1, nx { rhs(ix) = q(it,ix) } }
                else {  call tricon(bl,q11,q12,q13,nx,sol,rhs)
                        do ix= 1, nx { rhs(ix) = q(it,ix) - rhs(ix)}
                        }
                call rtris(nx,endl,q21,q22,q23,endr,rhs,sol,e,f)
                do ix = 1,nx{ r(it,ix) = sol(ix) }
                }
        call bicon(1.,bl,p11,p12,p13,p21,p22,p23,nt,nx,p,r)
        if( hilo==0.)
        do it= 1, nt {  do ix= 1, nx { p(it,ix) = p(it,ix) }}
        else
```

```
        do it= 1, nt {  do ix= 1, nx { p(it,ix) = q(it,ix) - p(it,ix) }}
        }
return; end


subroutine bicon(conj,bl,p11,p12,p13,p21,p22,p23,nt,nx,p,r)
integer it,ix,nt,nx
real p(nt,nx),r(nt,nx)
real conj,bl,p11,p12,p13,p21,p22,p23
temporary real px(nx),dia(nx),offdia(nx)

do ix= 1, nx { dia(ix) = 0.; offdia(ix) = 0.  }
if( conj==0. )
        do it= 1, nt {
                if( it==1 ) {   do ix= 1, nx { px(ix) = p(it,ix) }
                                call tricon(bl,p21,p22,p23,nx,px,dia)    }
                else {          do ix= 1, nx { px(ix) = p(it,ix) }
                                call tricon(bl,p21,p22,p23,nx,px,dia)
                                do ix= 1, nx { px(ix) = p(it-1,ix) }
                                call tricon(bl,p11,p12,p13,nx,px,offdia) }
                do ix= 1, nx { r(it,ix) = dia(ix) + offdia(ix) }
        }
else
        do it= nt, 1, -1{
                if( it==nt ) {  do ix= 1, nx { px(ix) = r(it,ix) }
                                call tricon(bl,p21,p22,p23,nx,px,dia)    }
                else {          do ix= 1, nx { px(ix) = r(it,ix) }
                                call tricon(bl,p21,p22,p23,nx,px,dia)
                                do ix= 1, nx { px(ix) = r(it+1,ix) }
                                call tricon(bl,p11,p12,p13,nx,px,offdia) }
                do ix= 1, nx { p(it,ix) = dia(ix) + offdia(ix) }
        }
return; end


subroutine tricon(bl,p1,p2,p3,nx,px,rhs)
integer ix,nx
real px(nx),rhs(nx)
real bl,p0,pnxp1,p1,p2,p3

p0 = bl*px(1);  pnxp1 = bl*px(nx)

                    rhs(1)  = p1*p0       + p2*px(1)  + p3*px(2)
do ix = 2, nx-1 {   rhs(ix) = p1*px(ix-1) + p2*px(ix) + p3*px(ix+1) }
                    rhs(nx) = p1*px(nx-1) + p2*px(nx) + p3*pnxp1
return;  end
```

DOT-PRODUCT TEST RESULT

```
<x~,x>= -9.36094284,  <y,y~>= -9.36094379
```