*Stanford Exploration Project, Report 73, November 18, 1997, pages 233–248*

*Stanford Exploration Project, Report 73, November 18, 1997, pages 233–248*

# Implementation of a simple trace interpolation scheme

*Dimitri Bevc*[1]

***keywords:*** *interpolation, Fourier transform, algorithm, cross-well*


## ABSTRACT

A trace interpolation scheme suggested by Muir (1991) is implemented and tested on real and synthetic data. The algorithm is conceptually simple: the assumption is made that there is some low-frequency portion of the data that is not spatially aliased and which can safely be interpolated. For each trace there is some relationship between the unaliased low-frequency data and the spatially aliased high-frequency data. This relationship can be found for each of the original traces and then used to reconstruct the high-frequency data on the interpolated traces. The nonstationarity of seismic data makes it necessary to perform the interpolation on small windows of data. The method works well for real and synthetic data when the original data are uniformly sampled and have an adequate low-frequency component.


## INTRODUCTION

Modern recording methods insure that seismic data are adequately sampled in time. However, economic and practical considerations dictate that data are gathered at surface locations which are often far enough apart that the data are spatially aliased. This aliasing can adversely effect the performance of migration algorithms as well as other processing and imaging techniques. One low cost method of increasing the performance of imaging methods is to interpolate the data in space. The effectiveness of this type of interpolation is demonstrated by Spitz (1991), who shows that using an $f-x$ prediction method to interpolate data results in an improved migrated section. To this end, Claerbout (1992) presents a $t-x$ prediction filter method of interpolation. Indeed, the subject is receiving so much attention at the Stanford Exploration Project that whole sections in this and the last report are devoted to interpolation.

I describe the implementation of the interpolation scheme suggested by Muir (1991) and show that it works for real and synthetic data. The nice thing about this method is that it is simple and easy to understand. Regularly sampled data sets

[1]**email:** not available

which require trace interpolation and have an adequate low-frequency portion of their spectra unaliased are candidates for this scheme.

First I describe the algorithm in detail, and then I present interpolations of synthetic and real data.

## THE ALGORITHM

I implement the algorithm in several straightforward one-dimensional modules. The data are partitioned by low-pass filtering and the low-frequency portion is interpolated. The high-frequency (HF) portion of the data is then reconstructed for the interpolated traces. The data are partitioned into small windows in which the interpolation is performed. The windows are then pieced back together to create the final interpolated section. The task of partitioning and re-assembling the data is performed by Claerbout's subroutine `patch()` (Claerbout, 1992b).

The whole interpolation scheme can be divided into five discrete modules:

- Each trace is low-pass filtered. By filtering the data in time, the spatially aliased portion of the spectrum is removed.
- The low-pass filtered data are interpolated in space by performing a one-dimensional Fourier transform, zero padding, and performing the inverse transform.
- The low-frequency (LF) data are then cubed in the time domain in order to broaden the spectrum. This is done to both the original and interpolated LF data.
- The cubed LF traces ($[f_{low}(t)]^3$) are then used to find a shaping filter. For each original trace, a filter $b(t)$ is found such that

  $$b(t) * [f_{low}(t)]^3 = f_{all}(t)$$

  where $f_{all}(t)$ is the wide-band data including high- and low- frequency. This is done using conjugate gradients with the subroutine `shp()` (Appendix).
- Once the shaping filter is found, it is used to reconstruct the HF data on the interpolated traces.

The cubing operation is used simply to enrichen the spectral content of the interpolated low-frequency data so that the shaping filter can be applied. It was chosen since it is a low order polynomial power which preserves polarity. In a more general formulation the fifth, seventh, etc. powers could be used. Even powers could also be used by forming something like $|f_{low}(t)| \times f_{low}(t)$ so that polarity is preserved.

## RESULTS AND DISCUSSION

First I interpolate a simple synthetic model consisting of two crossing linear events. This simple example will point out some of the pitfalls of this interpolation algorithm. Next, I apply the algorithm to a cross-well data set.

### Synthetic data

The synthetic data set used for this example is evenly sampled in space and time, and the higher temporal frequencies are aliased in space. There are 64 traces, each containing 512 time samples. The sampling interval is $4ms$. The linear events consist of low-pass ($60Hz$) Butterworth wavelets. Only first order interpolation is performed on this example. The low-frequency data are created by low-pass filtering the synthetic with a $30Hz$ Butterworth filter.

Figure **??** shows two crossing events and their amplitude spectra. The steeply dipping event is aliased in space. The shaping filter used to reconstruct the HF from the LF is shown in Figure **??**. This filter is convolved with the cubed LF data on the interpolated traces to create the final interpolated section (Figure **??**). The prominent alias visible in Figure **??** has been removed. However, the interpolation breaks down where the two events cross. The break-down is seen as oscillations and distortions near the crossing point in the time domain and as horizontal lines of noise in the amplitude spectrum.

This distortion is due to the fact that the shaping filter is aliased in space. Where the events are far apart, the shaping filter is very smooth. Near the point where the events cross, the shaping filter gets more complicated and varies rapidly in the spatial direction (Figure **??**). The filter is calculated where both the HF and LF are known and then interpolated to the trace locations where the HF is unknown. Since the filter is aliased near crossing events, distortions arise in the interpolation. In the previous examples nearest neighbor interpolation was used to interpolate the shaping filter.

The interpolation may be improved by median filtering the shaping filter. The result of this interpolation is presented in Figure **??**. The result is much more satisfying but the amplitude at the crossover may be a little too high and there is still some noise visible in the frequency domain.

### Cross-well data

The cross-well data donated by the Stanford Tomography Project (STP) is used to test the algorithm. The shot gather (Figure **??**) consists of 316 traces, each containing 512 time samples. The traces are 1 foot apart and are sampled at $0.2ms$. The shot was at 540 foot depth. In order to test the interpolation algorithm, the data set was

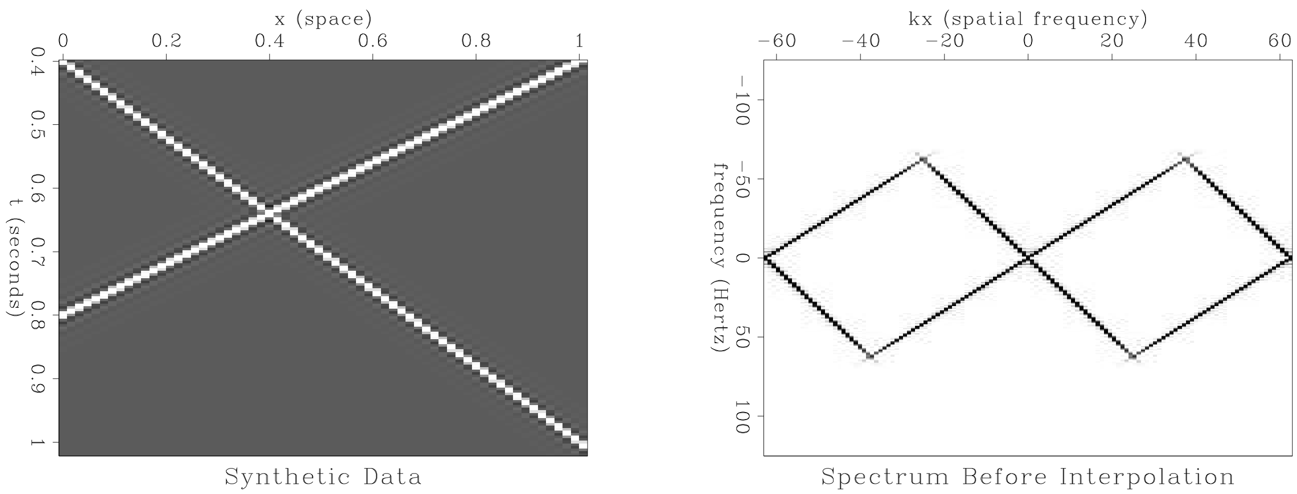


Figure 1: Synthetic data and amplitude spectrum for two crossing events. The amplitude spectrum is tilled between plus and minus twice Nyquist along the wavenumber axis to emphasize the aliased energy. dimitri1-cross [ER]

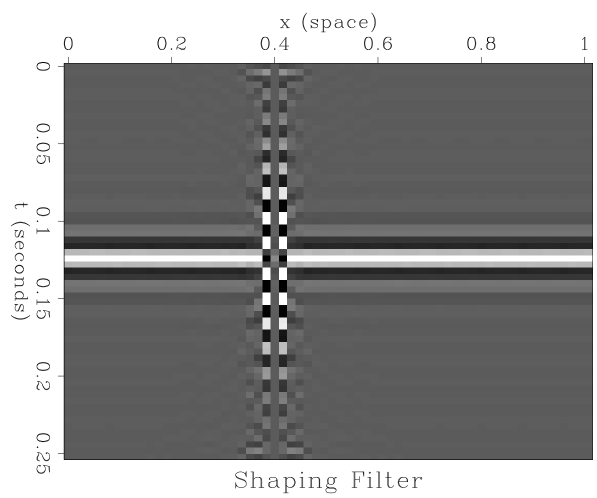


Figure 2: Shaping filter used to reconstruct the high-frequency data from the cubed low-frequency data. Notice the oscillations in the filter where the events cross. dimitri1-crossshape [ER]

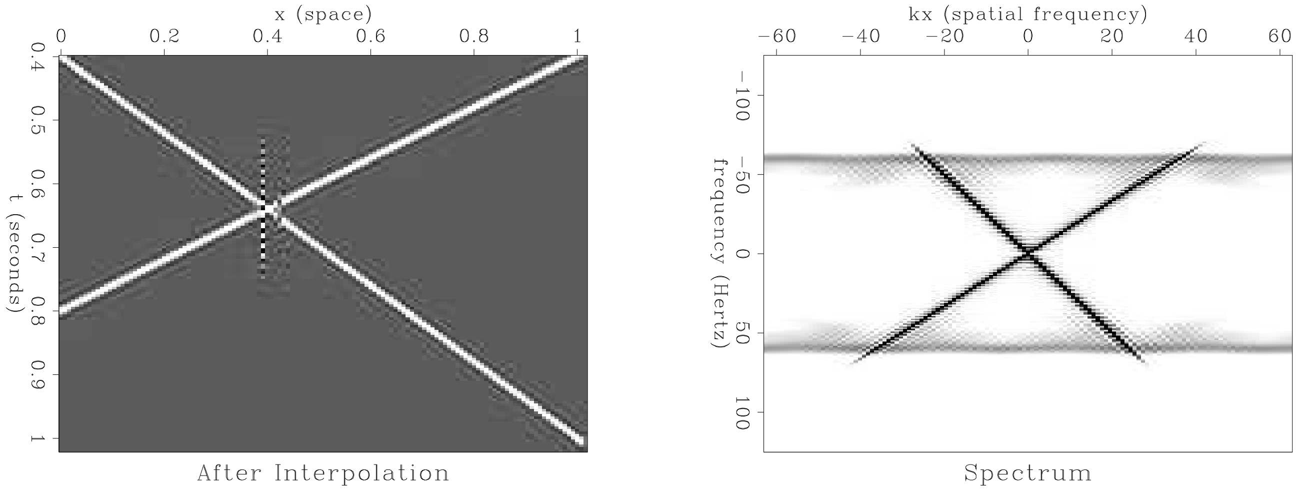


Figure 3: Interpolation result for the two crossing events. Notice the poor reconstruction where the two events cross and the noise in the spectrum. The error in $t-x$ space is asymmetric because the shaping filter is constructed by nearest-previous-neighbor interpolation. dimitri1-intcross [ER]

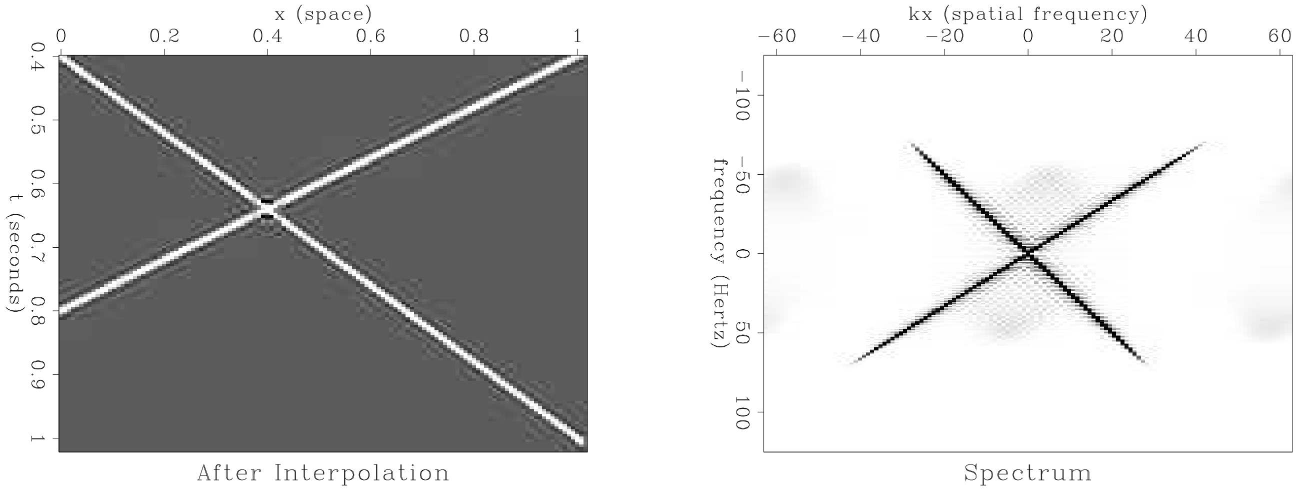


Figure 4: Improved interpolation result for the two crossing events. dimitri1-crossfix [ER]

decimated by removing every other trace (Figure **??**). This spatially aliases the data and gives it a rougher checkerboard appearance. A plot of the spectrum (Figure **??**) clearly shows the aliased energy.

This data set is nonstationary. In order to perform the interpolation I must apply the algorithm to small windows of data and then piece these back together. This is done by the program `Pstri()` (Appendix). The data set is split up into 2048 overlapping windows. The shaping filter is calculated and applied in each window. The interpolated windows are then patched back together with triangular weights on each window.

The result of the interpolation is presented in Figure **??**. This compares quite well to Figure **??**. The events are now much smoother and more continuos than in Figure **??**. The higher amplitude events are reconstructed better than the weaker events.

There are artifacts in the interpolation that I have not been able to removing yet. The most distracting artifacts are the oscillations at various points in the interpolated data (such as at $depth = 740', t = 0.11s$). These are due to the aliasing of the shaping filter. Median filtering the shaping filter is unsuccessful in removing these artifacts because the shaping filter varies too much in each window. Increasing the number of windows does reduce these artifacts. Some of the fainter steeply dipping tube wave arrivals visible late in the section at $depth = 700'$ to $depth = 800'$ in Figure **??** have been removed by the interpolation (Figure **??**). The tube waves arriving at earlier times have also been attenuated. The algorithm has difficulty reconstructing these arrivals because they are high frequency and relatively low amplitude. Although it is often desirable to remove tube waves from cross-well data, that is not the goal of interpolation.

The spectrum of the interpolated cross-well data (Figure **??**) shows that most of the aliased energy has been removed.

Windowing the data gives rise to a whole set of parameter choices which effect the final quality of the interpolation. In general, increasing the number of windows in which the interpolation is done increases the quality of the interpolation.

I plan to experiment with the shaping filter design in order to eliminate the artifacts due to filter aliasing. Since the effect of the filter aliasing is distinctive, it seems likely that it could be eliminated.

## CONCLUSION

This trace interpolation scheme works well on evenly sampled data sets which have a substantial low-frequency component. The nonstationarity of seismic data makes it necessary to perform the interlacing on small windows of data which are then patched back together. The algorithm has difficulty when there are many crossing events because this tends to alias the shaping filter. This problem is partially overcome by

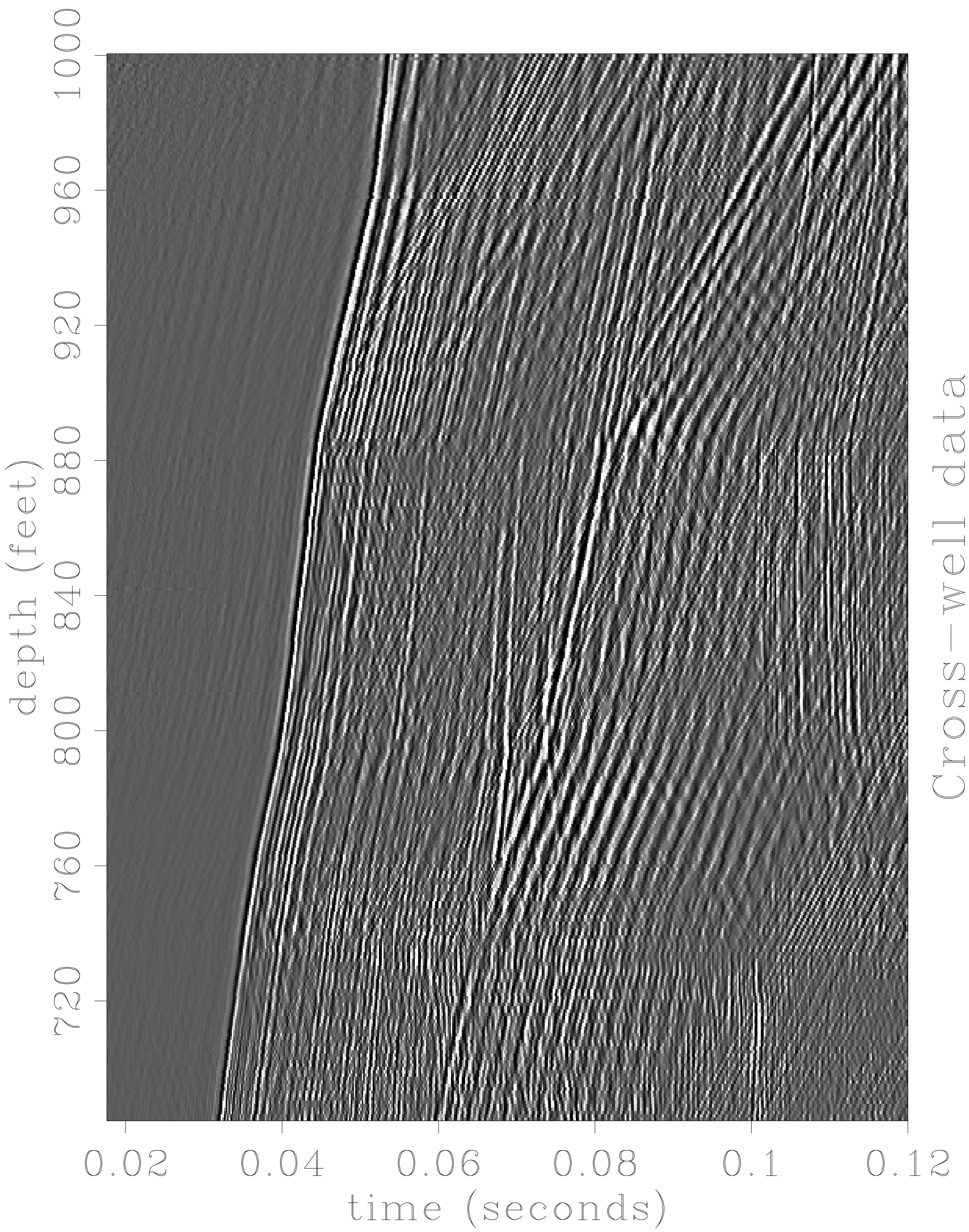


Figure 5: Cross-well data set before decimation. The trace separation is 1 foot. dimitri1-cwldat [ER]

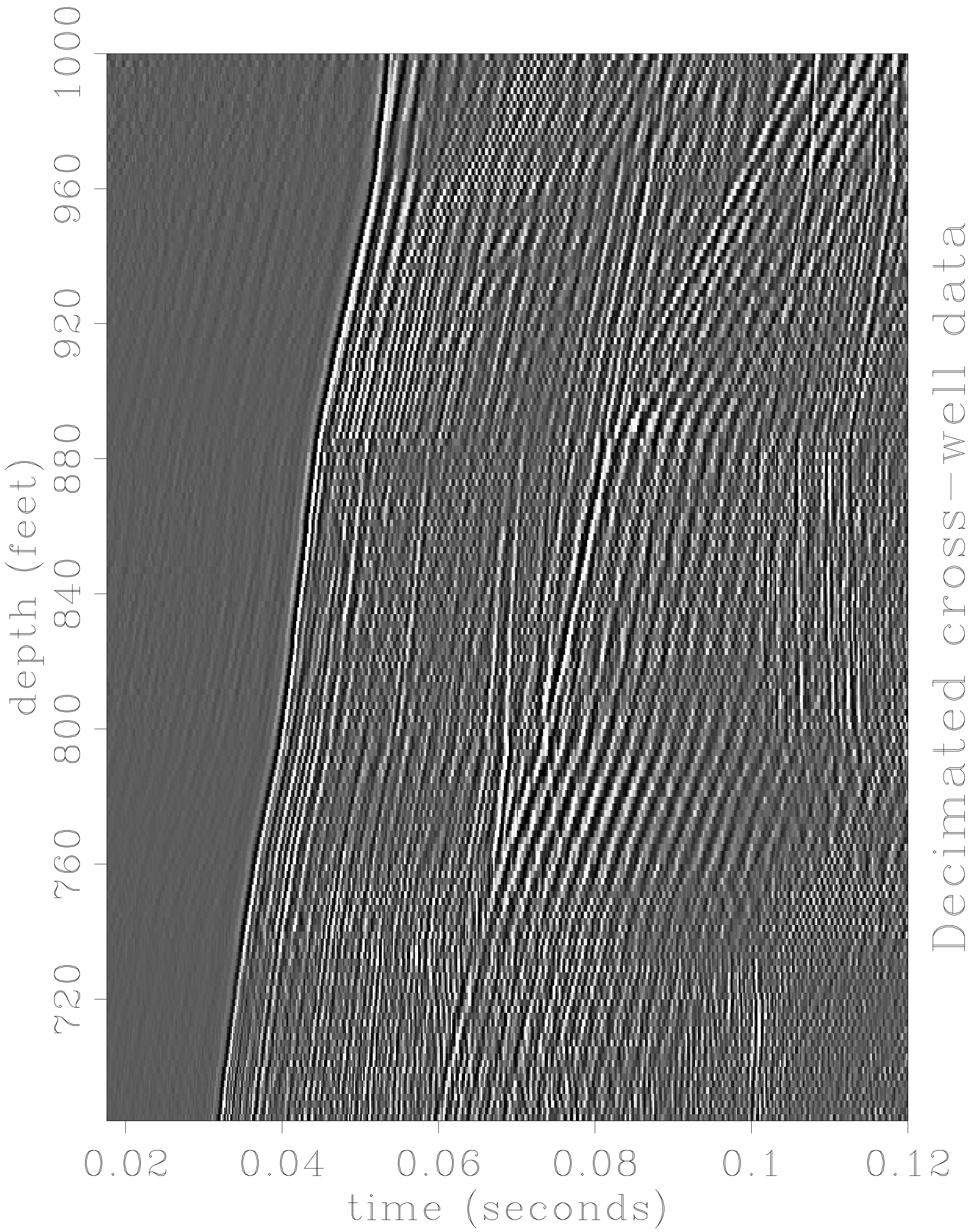


Figure 6: Decimated cross-well data set. Every other trace has been removed from the original data set so that the trace separation is 2 feet. dimitri1-cwldec [ER]

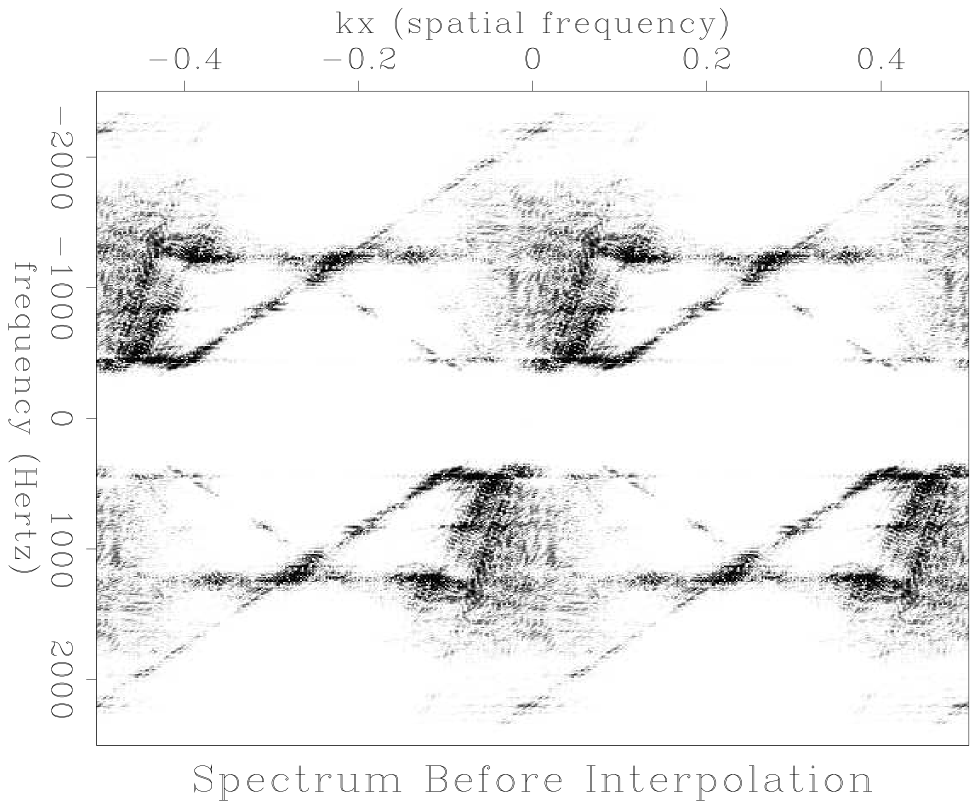


Figure 7: Decimated cross-well data set spectrum. The spectrum is tilled between plus and minus twice Nyquist along the wavenumber axis to emphasize the aliased energy. dimitri1-cwlalias [ER]

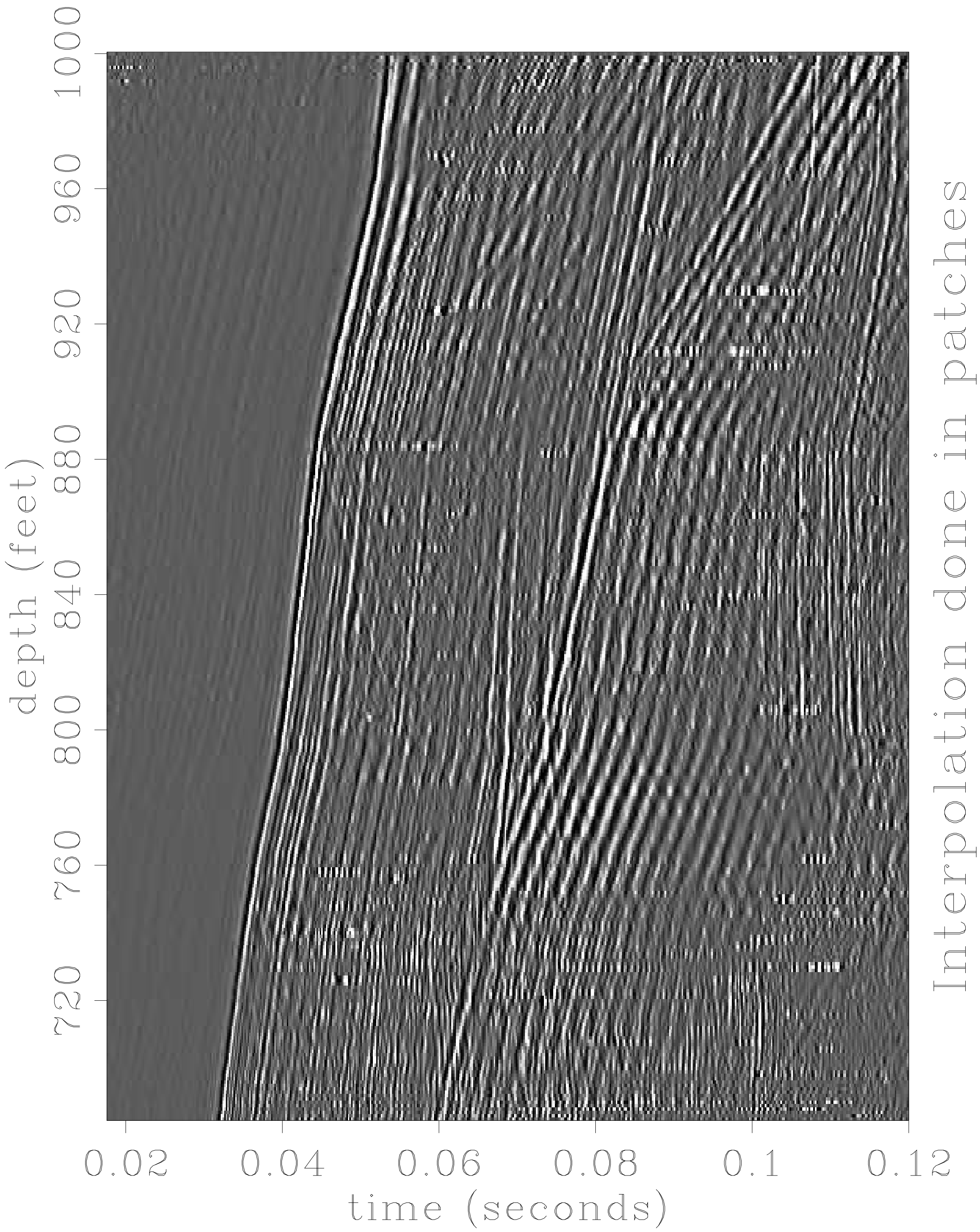


Figure 8: Interpolated cross-well data. Every other trace has been interpolated so that the trace separation is 1 foot. $\boxed{\text{dimitri1-intcwl}}$ [ER]

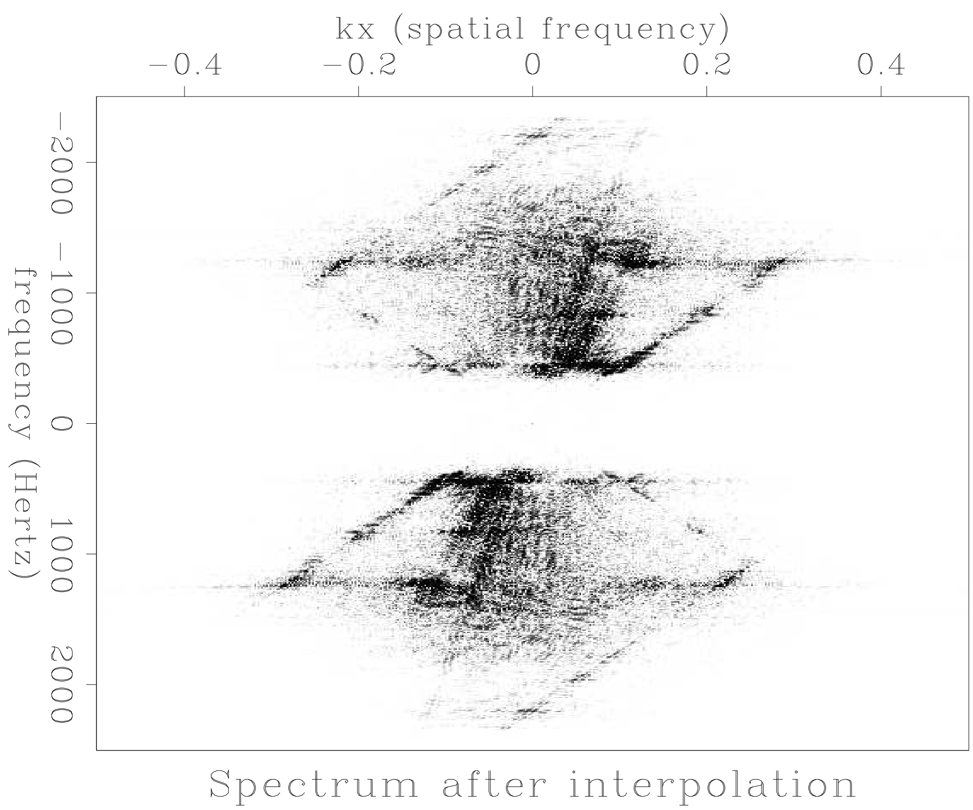


Figure 9: Spectrum of the interpolated cross-well data. dimitri1-intcwlfft [ER]

performing the interpolation in overlapping patches of data. For the cross-well data set, the presence of numerous events causes the shaping filter to be aliased. As a result the final interpolation has artifacts in it.

It may be possible to smooth the shaping filter to eliminate the artifacts but the method will eventually fail when there are too many crossing events. In general the method is better suited to relatively sparse and slowly varying data sets.

## ACKNOWLEDGMENTS

Francis Muir thought of this interpolation scheme and brought it to my attention. I thank him for encouraging me to work on it and for his abundant advice.

I would also like to thank the Stanford Tomography Project for providing the cross-well data set.

## REFERENCES

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

Claerbout, J. F., 1992b, Nonstationarity and conjugacy: Utilities for data patch work: SEP–**73**, 391–400.

Muir, F., 1991, A trace interpolation scheme: SEP–**72**, 73–76.

Spitz, S., 1991, Seismic trace interpolation in the f-x domain: Geophysics, **56**, no. 6, 785–794.

## APPENDIX

### Finding the shaping filter

The program `shp()` is used to find the shaping filter that reconstructs the HF from the LF for each interpolated trace. The routine is a modification of `shaper()` (Claerbout, 1992) with `contrunc()` replacing `contran()`.

```
# Shaping Filter (Modification of shaper [PVI])
# find the shaping filter for the cubed data in time domain
#
#  rr = yy - xx * ff
#%
subroutine  shp(n1,n2,nf,ny,yh,yl,x)
#
integer n1,n2,i1,i2,nf,ny,lag,iter,niter,i,nx
real yh(n1,n2),yl(n1,n2),x(nf,n2)
temporary real ff(nf),xx(n1),yy(ny),rr(ny)
temporary real res(ny,n2),df(nf),dr(ny),sf(nf),sr(ny)
#
lag=nf/2
niter=nf
nx=n1
do i2=1,n2 {
        call zero(n1,xx)
        call zero(ny,yy)
        call zero(ny,rr)
        call zero(nf,ff)
        do i1=1,n1 {
                xx(i1)=yl(i1,i2)**3
                yy(i1)=yh(i1,i2)
                }
do i=1,ny
        rr(i)=yy(i)
do iter= 0, niter {
        call contrunc( 1, 0,lag,nx,xx,  nf,df,  ny,rr)  # df=xx*rr
        call contrunc( 0, 0,lag,nx,xx,  nf,df,  ny,dr)  # dr=xx*df
        call cgstep( iter, nf,ff,df,sf,  ny,rr,dr,sr)   # rr=rr-dr; ff=ff+df
        }
#}
#do i2=1,n2 {
        do i1=1,nf {
                x(i1,i2)=ff(i1)
                }
}
call slice('filterxc.h',4,nf,1,ff)
call slice('filter.h',4,nf,n2,x)
return;end
```

### Interpolating in windows

The program `Pstri()` performs the interpolation on small windows of data which are then patched back together. The shaping filter is found by `shpsafe()` which is similar to `shp()`. The actual interpolation is performed by `sip()`. The decimated crosswell data set has dimensions `n1=512, n2=158`. The interpolation in this paper is performed by setting `w1=32, w2=16` and `k1=32, k2=64`. This makes 2048 windows. The shaping filter is 32 points long (same length as the windows).

```
# Pstri: calculate shaping filter in windows and patch back together.
#               n1, w1, k1 (data dimension, window dimension, # of windows)
#               n2, w2, j2
#% end of self doc.
integer n1,n2,nf,ntr, w1,w2, k1,k2
real d2
from history:   integer n1, n2
from history:   real d2
from par:       integer ntr=1
from par:       integer nf=32
from par:       integer w1=32 , w2=16
from par:       integer k1=64  , k2=128
allocate:       real all(n1,n2), low(n1,n2)
allocate:       real shapeout(k1*nf,k2*w2), shape(nf,w2)
allocate:       real tempall(w1,w2), templow(w1,w2)
allocate:       real temptest(w1,(ntr+1)*w2),test(n1,(ntr+1)*n2)
allocate:       real final(n1,(ntr+1)*n2)
allocate:       real windwt(w1,(ntr+1)*w2),wallwt(n1,(ntr+1)*n2)

subroutine Pstri(n1,n2,d2,nf,ntr,all,low,shapeout,tempall,templow, w1,w2, k1,k2,shape,test,temptest,windwt,wallwt,final)
integer nf,ny,ntr, i1,n1, i2,n2,  w1,j1,w2,j2, k1,k2, outfd,output, np
real all(n1,n2), low(n1,n2),d2
real shapeout(k1*nf,k2*w2), shape(nf,w2)
real tempall(w1,w2), templow(w1,w2)
real temptest(w1,(ntr+1)*w2),test(n1,(ntr+1)*n2)
real final(n1,(ntr+1)*n2)
real windwt(w1,(ntr+1)*w2),wallwt(n1,(ntr+1)*n2)
ny=w1+nf-1
np=(ntr+1)*w2
outfd=output()
call putch('n1','i',n1)
call putch('n2','i',(ntr+1)*n2)
call putch('d2','f',d2/(ntr+1))
call putch('title','s','Interpolation done in patches')
call hclose()

call sreed('in', all, 4*n1*n2 )
call sreed('inlow', low, 4*n1*n2 )

call cinloiwt(1,1,1,1,windwt,w1,np)
call mkwallwt(k1,k2, windwt, w1,np, wallwt,n1,(ntr+1)*n2)

do i1=1,k1{
do i2=1,k2{
```

```
        call patch( 0,0, i1,i2, k1,k2, all,n1,n2, tempall,w1,w2)
        call patch( 0,0, i1,i2, k1,k2, low,n1,n2, templow,w1,w2)
        call shpsafe(w1,w2,nf,ny,tempall,templow,shape)
        call patch( 1,1, i1,i2, k1,k2, shapeout,k1*nf,k2*w2, shape,nf,w2)
        call sip(w1,w2,tempall,templow,np,shape,ntr,nf,ny,temptest)
        call diag(0,0, windwt,w1*np,temptest,temptest)
        call patch( 1,1, i1,i2, k1,k2, test,n1,(ntr+1)*n2, temptest,w1,np)
        }}
call diag(0,1,wallwt,n1*(ntr+1)*n2,test,final)

call srite('out', final, 4*n1*(ntr+1)*n2)
return; end
```