EXAMPLES OF PARSIMONIOUS DECONVOLUTION

Jon F. Claerbout

Abstract

Two tests were made of the parsimonious deconvolution algorithm. The single trace synthetic data test worked very well even with included noise. A second trial done on a profile of 48 hydrophones recording a marine source worked very well for some choices of parameters but other parameters led to unstable behavior outside the signal frequency band.

Synthetic Data Test Case

A source waveform was defined by a sequence of numbers to resemble the explosion-implosion sequence of a marine bubble seismic source. The sequence of 23 numbers is

6,10,-1,-3,-4,-4,-3,-1,6,-1,-2,-2,-1,0,4,-1,-2,-1,0,2,0,-1,-1

A synthetic spike train is defined (on program listing bunch/pard0.f subroutine job) as

[zeros, 1.01, zeros, .5, -.5, zeros]

After convolution of the spike train with the bubble, then pseudo random noise is added. The result is defined to be the filter output y_t. The noise amplitude is one-half percent of the maximum signal value. The noise, however, is added at every time point but innovations occur only once in a while. On a root mean square basis, the signal-to-noise ratio is about thirty. Figure 1 shows the results of the parsimoneous deconvolution algorithm. Fourteen iterations were done. Before each iteration a display is made of the estimated input x_t and the estimated filter

b_t which when convolved together give the known synthetic data y_t. At the first iteration, we see the input x looks like a bubble function followed by a differentiated bubble function. The presumed filter b_t is a delta function. The first eight displays are minimization of $S_{5/2}$ and the final six displays are of the subsequent minimization of S_2. The numerical value of $S_{5/2}$ and S_2 is displayed along the left edge. Also displayed are the successive filter perturbations db_t, the unconstrained gradients g_t and constrained gradients dx_t.

We did not have a varimax program available for comparison. Furthermore, the varimax procedure as described by Wiggins and Larner is a multichannel process. And furthermore as it was described, it had no facility for imposing causality on the forward filter b_t. Nevertheless, we wanted to make some kind of comparison to evaluate the importance of the concept of using norms of low order. What was done was to take the pard0 program and reset $n = 4$ (by the program statement en = 4). The result is the minimization of

$$\min = S_4 = \frac{\| x \|_4 - \varepsilon}{\| x \|_4}$$

which is displayed in figure 2. The result is judged to be subjectively inferior to the results of minimizing S_2 in figure 1.

Finally, I have included figure 3 which has been taken from the Western Geophysical Company's pamphlet *Minimum Entropy Deconvolution (MED)* It shows a verimax result. I must most emphatically assert that I am <u>not</u> sure it is valid to compare this result to figures 1 and 2. The wavelet is different, the constraints are different, and the number of channels is different. Supposing that all calculations were done with single precision (10^{-6}) accuracy, it seems reasonable to expect $(10^{-6})^{1/4} = 3\%$ noise from the computation alone in figure 3. This is roughly what we see in figure 3. In figure 2 the noise by the same crude theoretical argument should be $(.005)^{1/4} = 26\%$, which is roughly what the figure shows. Another peculiarity of figure 2 is that the doublet on the

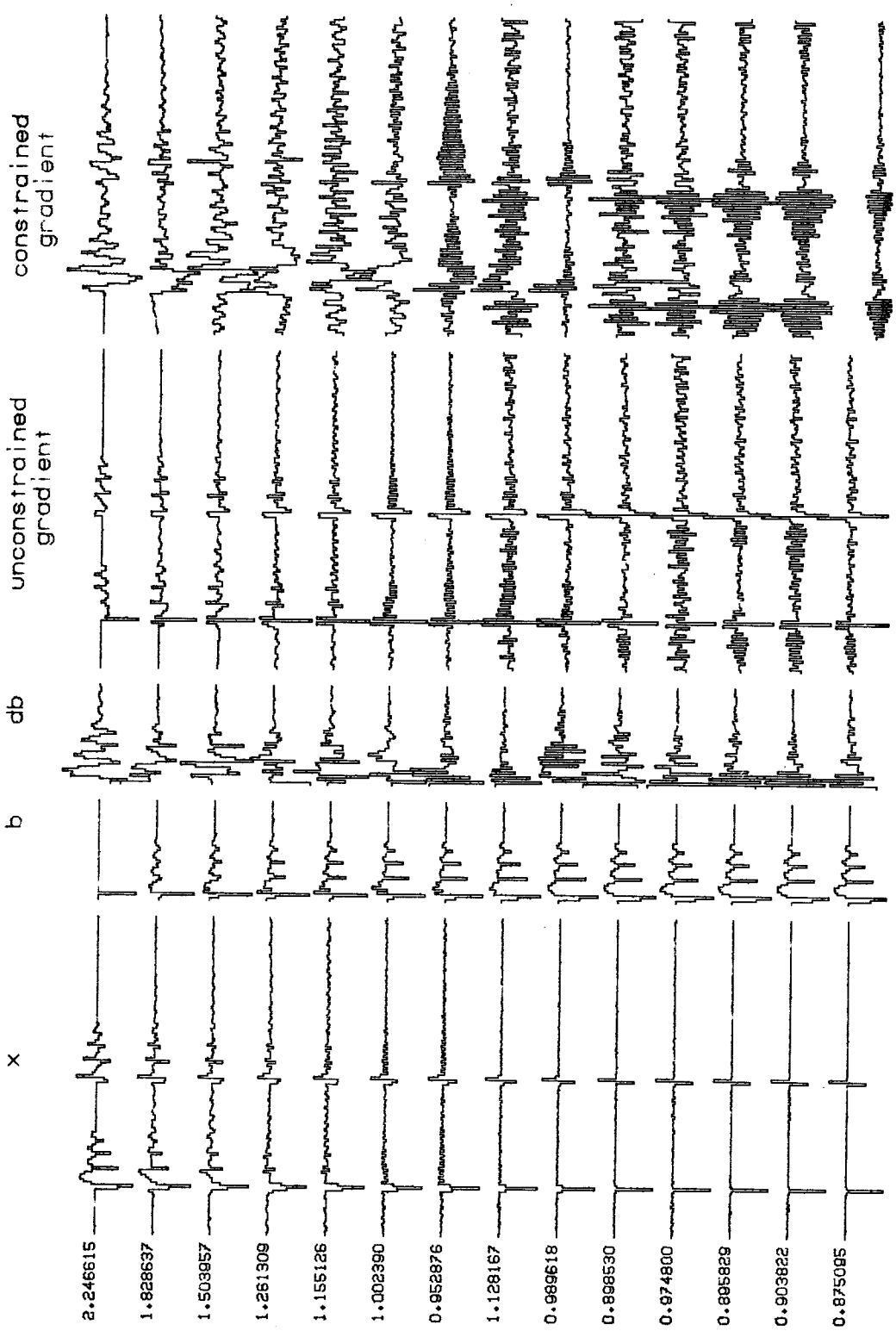

Figure 1.—Test of the parsimonious deconvolution program bunch/par[illegible].f. After 14 iterations, a subjectively favorable decomposition of the output y into the input x and filter b has been achieved.

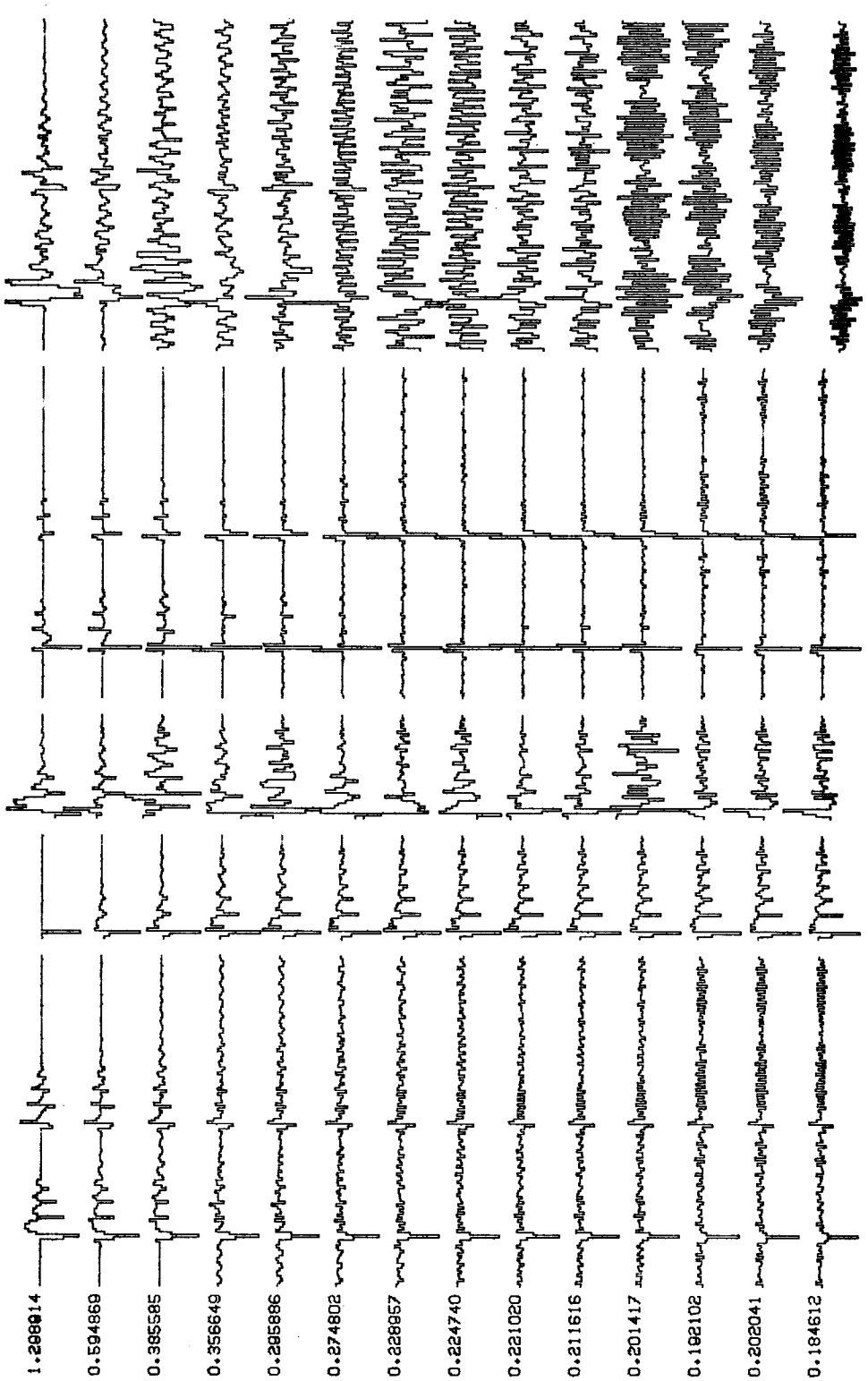

Figure 2.—In figure 1, the norm ratio $N_2^{2-\varepsilon}$ was minimized. In this figure, the norm ratio $N_4^{4-\varepsilon}$, which has some similarity to varimax, is minimized.

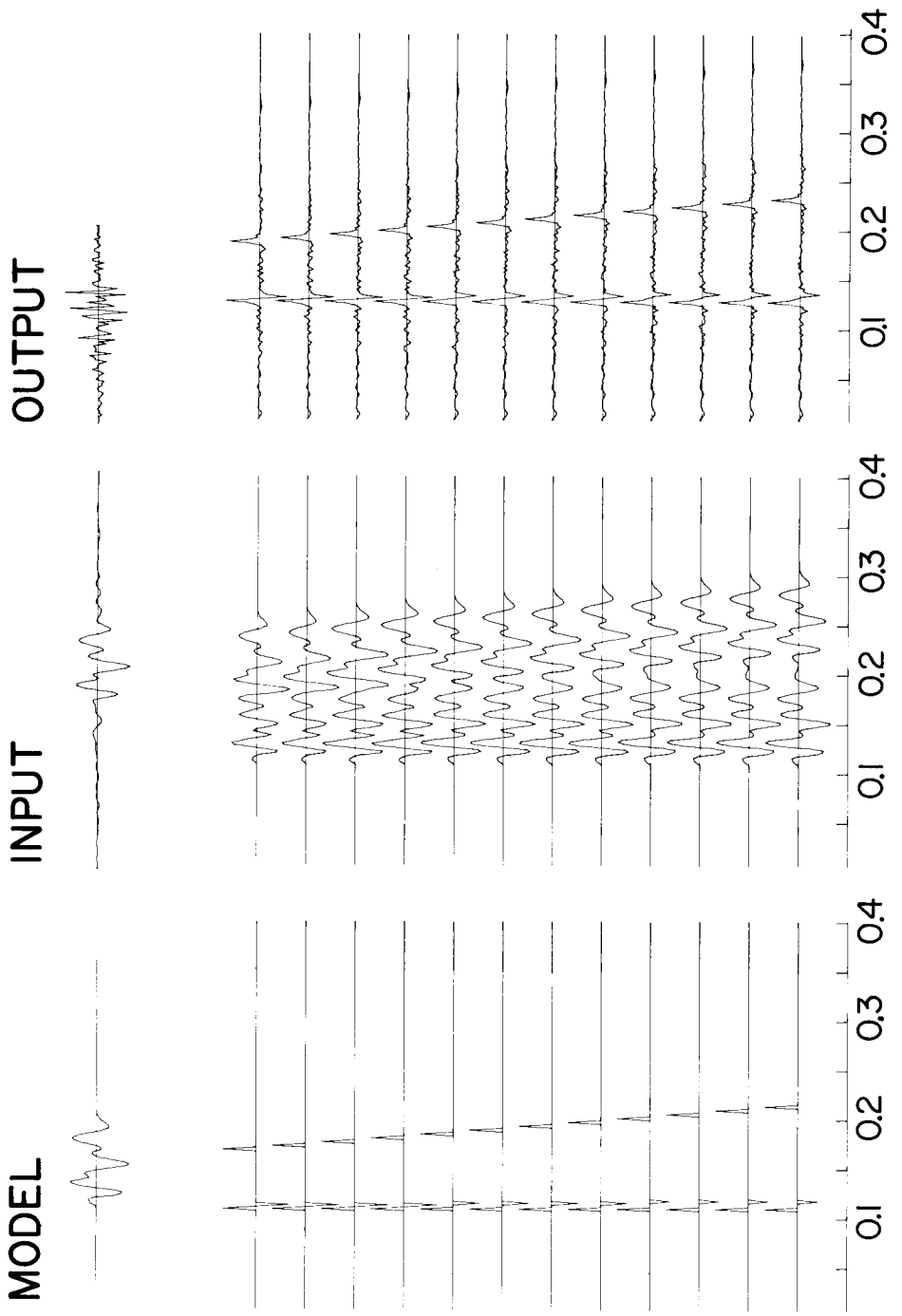

Figure 1. Synthetic data example
a. *A hypothetical CDP gather of twelve spike-series traces. The three spike "events" on each trace experience differential moveout from "near trace" to "far trace." The input wavelet is shown above.*
b. *Each trace is the convolution of the corresponding spike trace with the non-minimum-phase input wavelet. The effective wavelet resulting from MED processing is shown above.*
c. *Traces resulting from MED processing. The optimum MED operator is shown above.*

Figure 3.—Example of the Western Geophysical MED process.

deconvolved data looks smaller than it should be, even though the forward filter is constrained to be shorter than the interval between the pulse and the doublet.

Finally, I have included the computer program used to generate figure 1 and figure 2.

Field Data Test Case

Forty eight seismic traces from the same source (a common shot profile) were selected from a marine seismic section. The complete section is displayed in SEP 12 page 77. The shotpoint was very near to the left edge of that section. The common shot point profile is displayed in figure 4. After deconvolution, it was found that the frequency content of the data was significantly higher. This led to a data viewing problem because the stepout from trace to trace caused spatial aliasing. A more satisfactory presentation of the data is to perform time shifts which reduce the stepout. Hence, the traces were time shifted so that a theoretical first water bottom multiple would have no stepout. The raw field data with these time shifts and spherical divergence correction is displayed in figure 5. A prominent event labeled "bubble" is seen to be parallel to the sea floor arrival. Figure 6 shows the data after deconvolution by the waveform determined from the parsimonious deconvolution algorithm. The bubble event has virtually disappeared. The frequency content has changed too. In order to effect a more appropriate comparison, Larry Morley performed a conventional Wiener-Levinson deconvolution. It is displayed in figure 7. It happens to have about the same spectrum as figure 6. It is clear from these figures that the parsimonious deconvolution has done a much better job of bubble suppression than the conventional deconvolution. The most obvious case is the bubble event at small offsets. Bubble suppression is also notable on the refracted wave at wide offsets and to a lesser extent on the sea floor multiple reflection about 2/3 of the way out the cable. A single bubble function b_t was determined for the entire gather. It is displayed on figure 8 at the same time scale as the profile. The many replicas of this waveform in figure 8 are shown to exhibit the waveform at

both polarities and various gains. Because figures 4 through 7 have gain increasing with time according to spherical divergence, the bubble will appear more strongly on those figures than on 8.

Figure 9 shows a superposition of two different bubble waveform estimates. One was based on traces 2,3,4,5, and 6, that is to say, the far end of the streamer cable. The other is based on traces 2,6,10,14, 18,22,26,30,34,38,42, and 46, that is to say, a uniform distribution of 12 of the 48 traces. Not only are the estimated waveforms very similar but it is also possible to see clearly where $t = 0$ should be. The noise level can be estimated from the failure of the waveforms to vanish completely before $t = 0$.

Bugs, Instabilities

Unfortunately, the program sometimes exhibits resonant instabilities. A typical example of this is shown in figure 10. It might be a program bug. More likely, the algorithm needs some kind of explicit stabilization procedure. The Levinson algorithm, for example, often needs a bit of white noise. Here we have not yet devised the appropriate stabilizing procedure. Nevertheless, the results are *sufficiently good* that the computer program has been included. The next thing I'm going to try is to restrict the *relative* variation of the Fourier transform of the filter from one iteration to the next.

* * * * * * * * * * * * * * * * *

Einstein's Second Theory of Relativity:

Your chances of getting along with your relatives increases directly in proportion to the distance you keep away from them.

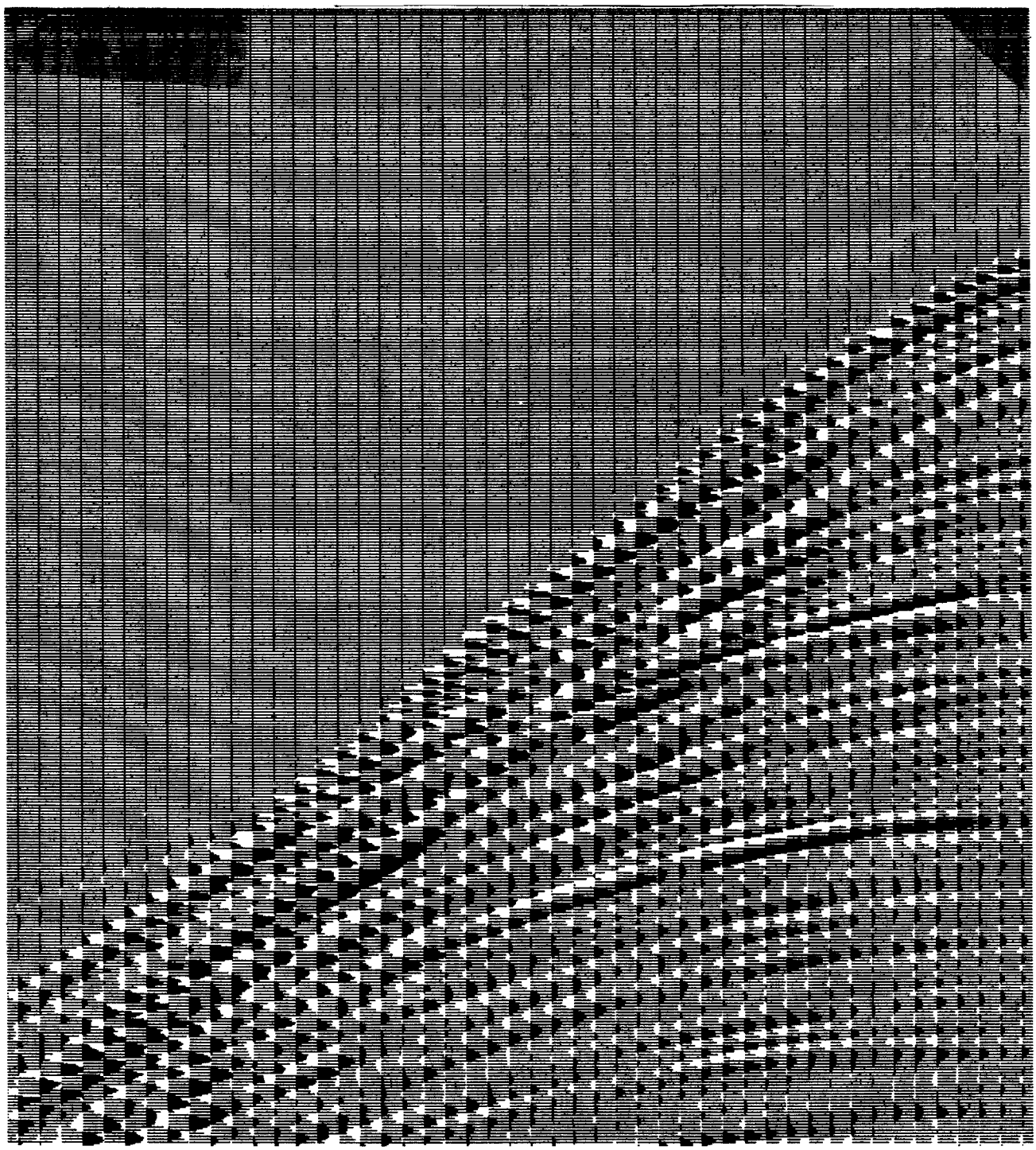

FIGURE 4.—The common shot gather which was used for the first field data test of parsimonious deconvolution. Displayed is the field tape with spherical divergence correction.

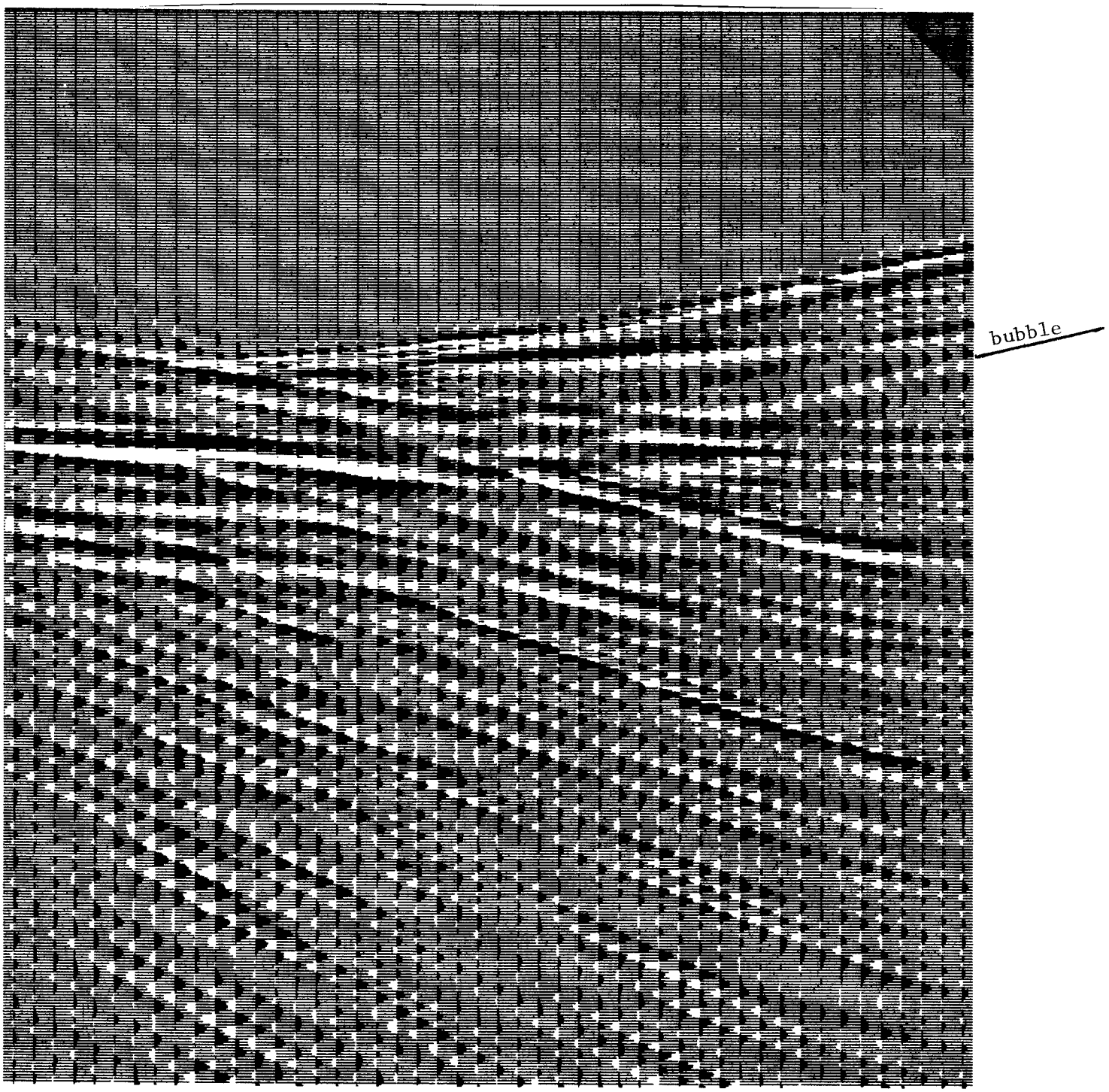

FIGURE 5.—The data of figure 4 were time shifted so as to flatten the first multiple reflection.

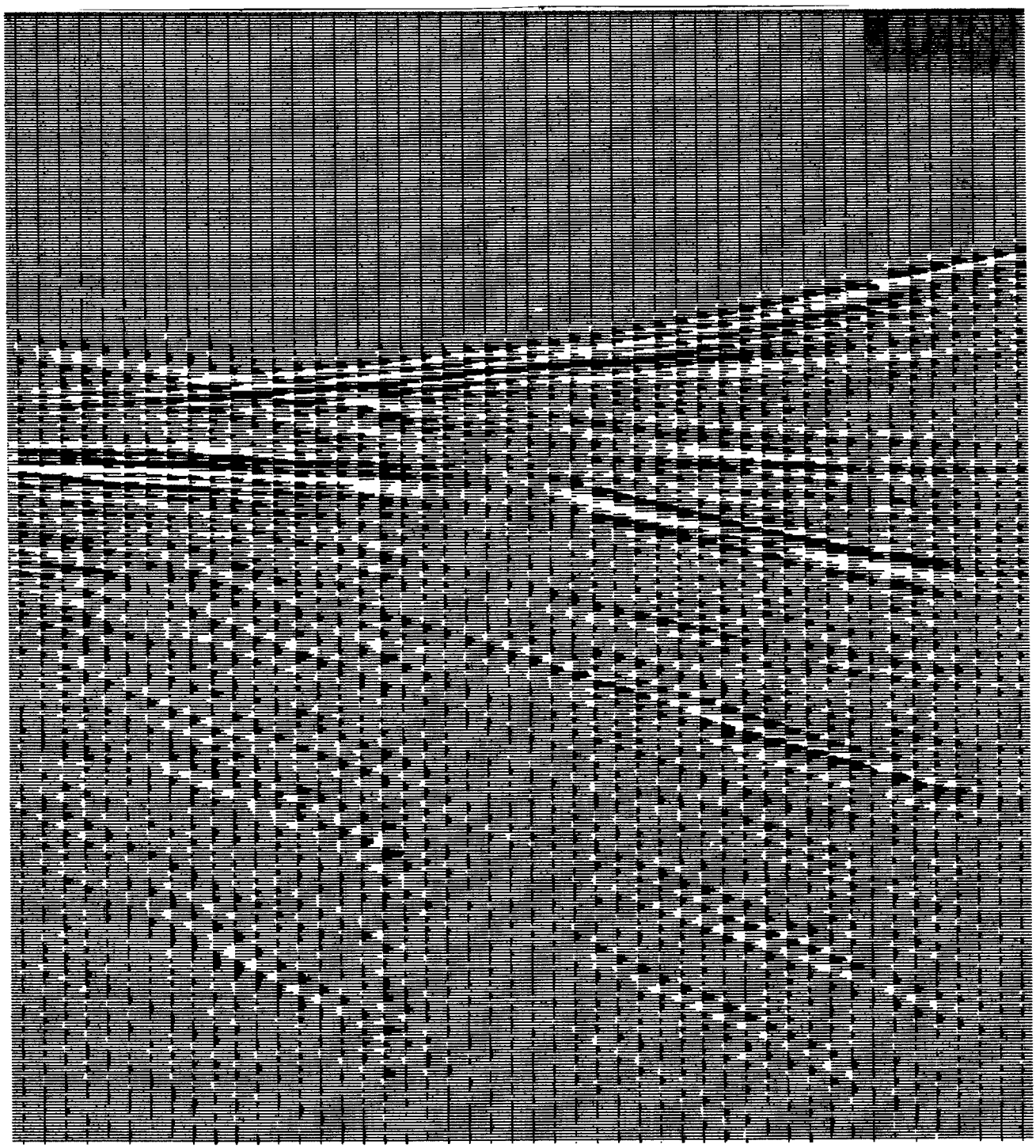

FIGURE 6.—The data of figure 5 after parsimonious deconvolution, spherical spreading, and trace equalization. Compare to figure 5 and 6 to note absence of bubble events.

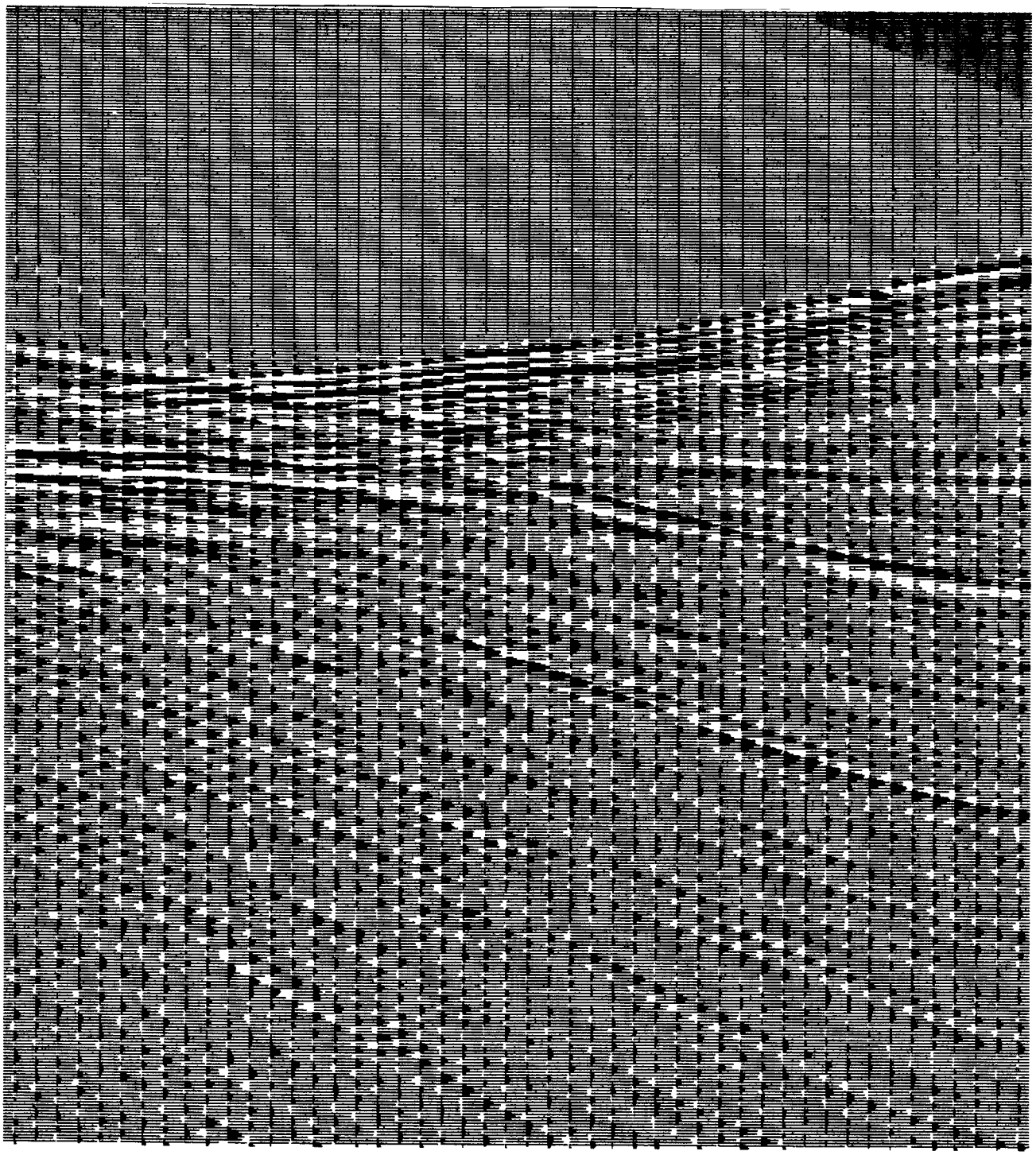

FIGURE 7.—The data of figure 5 after conventional deconvolution.

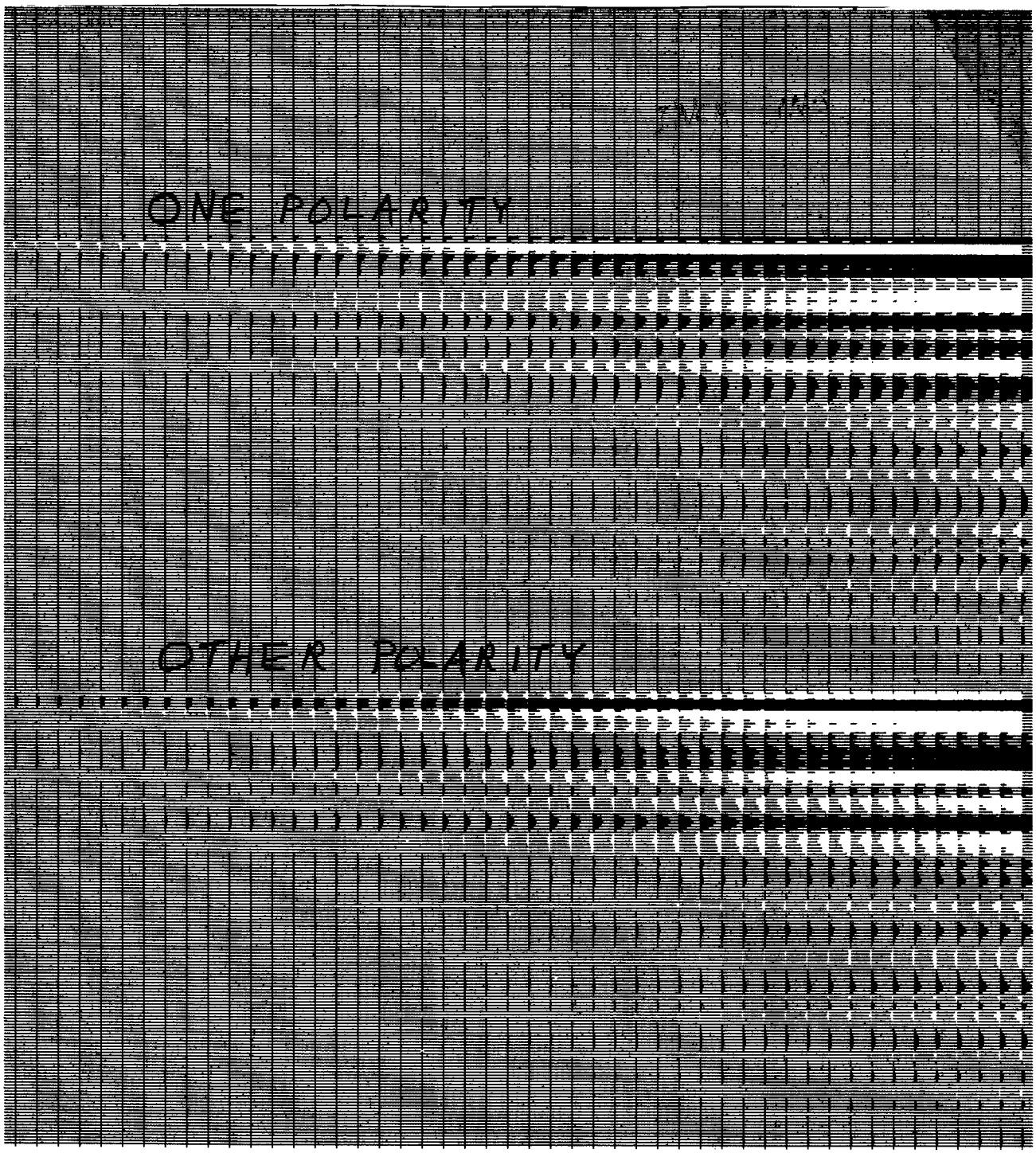

FIGURE 8.—Display of the bubble function which was determined by parsimonious deconvolution. Gain increases to the right. Both polarities displayed. Bubble tails will be larger on figures 4-7 than here because of spherical divergence.

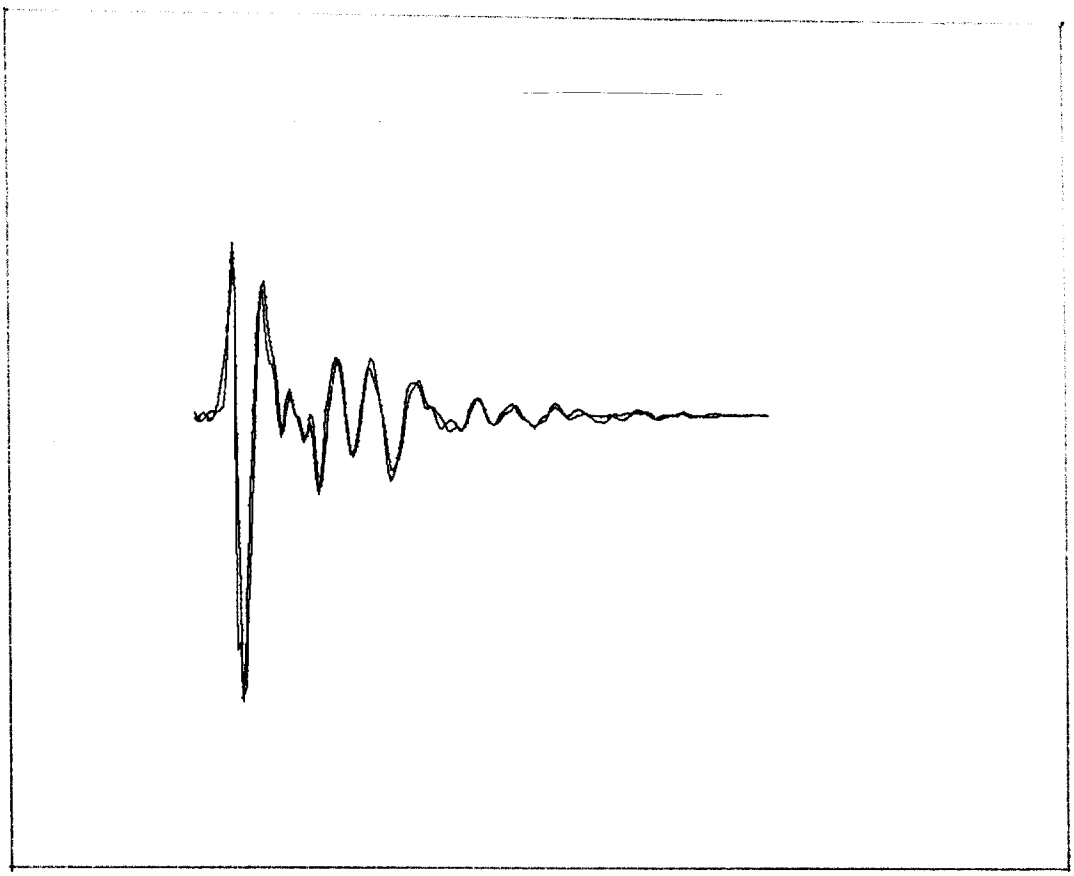

FIGURE 9.—A superposition of two different bubble waveform estimates. One was based on traces 2,3,4,5,6, that is to say, the far end of the streamer cable. The other is based on traces 2,6,10,14,18,22,26,30,34,38,42, and 46, that is to say, a uniform disbribution of 12 of the 48 traces. Not only are the waveforms very similar, but it is also possible to see clearly where $t = 0$ should be. The noise level can be estimated from the failure of the waveforms to vanish completely before $t = 0$.

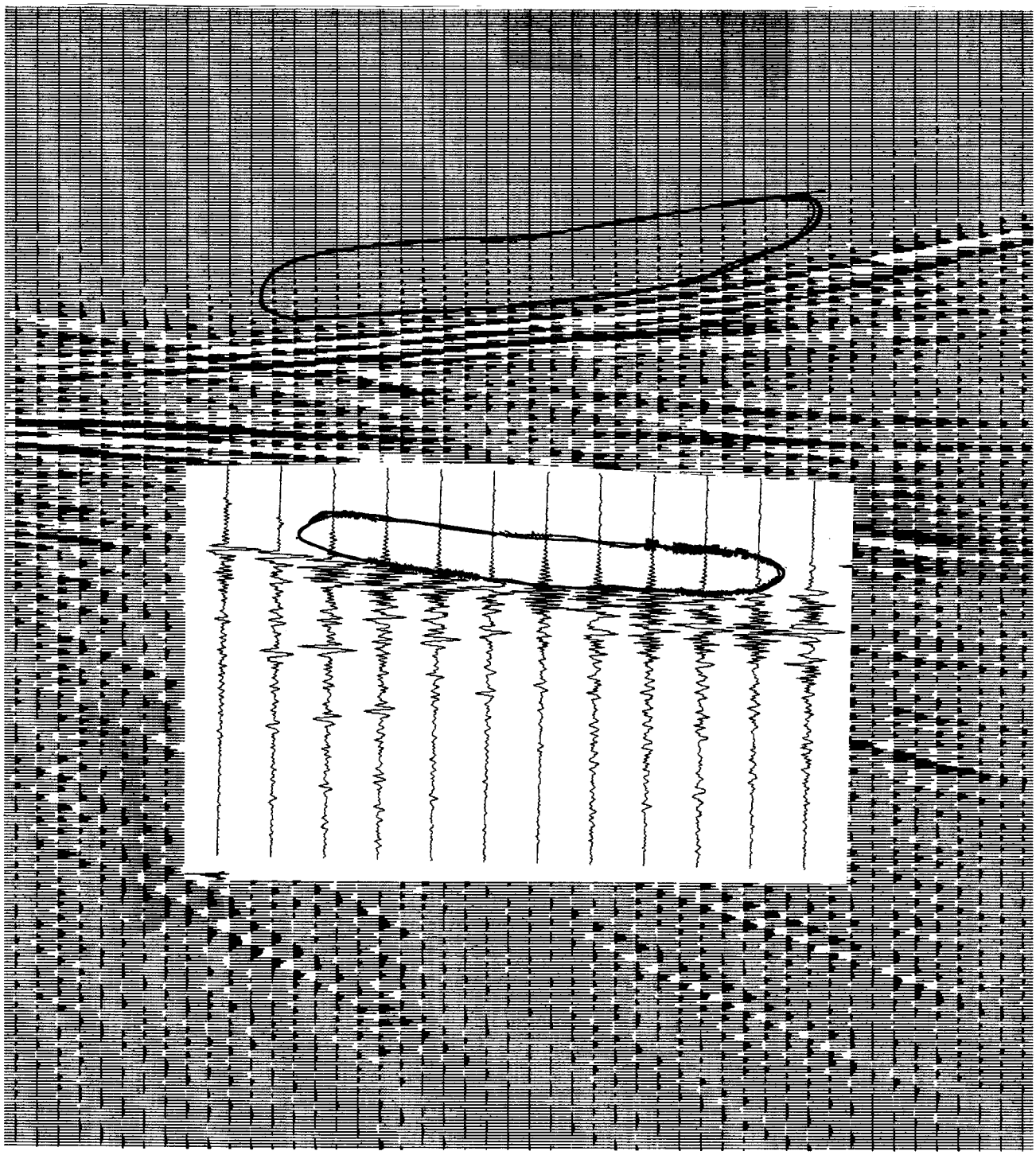

FIGURE 10.—A resonant instability which was not understood at the time of this report's printing deadline.

```
      real x(130),y(130),g(130),b(130),db(130),dx(130)
      complex cx(65),cy(65),cg(65),cb(65),cdb(65),cdx(65)
      equivalence(y,cy),(db,cdb),(dx,cdx)
      complex conjg
      call setmod(11,128) OUTPUT ON UNIT 11
      nt=128   trace length
      nth=nt/2
      nthp=nth+1
      nb=40    filter length
      amb=1./200.  half percent noise
      call job(nt,x,y,23,b,amb,rmssn) set up test case
      call fftr(nth,y,+1.)   fourier transform input data
      call move(nt,b,cb)
      call fftr(nth,cb,+1.)  fourier transform initial filter
      niter=14
      do 400 iter=1,niter    basic iterative loop
      alpha=.20*(niter-iter+.5)/niter   α decreases linearly
      en=2.5-.5*((iter-1)/((niter+1)/2))   n starts at 2.5 then becomes 2.0
      elog=grad(en,nt,x,g)      gradient computation
      call scale(-1.,nt,g)      scale grad / negative (for display)
      write(11,25)iter,alpha,elog
25    format(i4,f5.2,f12.8)
      call move(nt,g,cg)
      call fftr(nth,cg,+1.)      G ← g
      call move(nt,x,cx)
      call fftr(nth,cx,+1.)      X ← x
      do 50 i=1,nthp
50    cdb(i)=-cg(i)*cb(i)*conjg(cx(i))      dB = X̄GB
      call fftri(nth,cdb,-1.)
      do 75 i=nb,nt              restrict duration of filter   db ← dB
75    db(i)=0.
      do 100 i=1,nb
100   db(i)=db(i)*(nb-i)   taper filter perturbation db
      sc=dot(nb,b,db)/dot(nb,b,b)
      do 125 i=1,nb
125   db(i)=db(i)-sc*b(i)  remove projection of b from db
      nskip=1
      if(mod(iter-1,nskip).ne.0)go to 150
      idy=3
      ink=1+(iter-1)/nskip
      call plot(ink,idy,100,nt,x)
      call plot(ink,idy,120+idy*nt,nb,b)
      call plot(ink,idy,140+idy*(nt+nb),nb,db)
      call plot(ink,idy,160+idy*(nt+nb+nb),nt,g)
150   continue
      ix=(iter-1)*70/nskip+70
      iy=0
      call number(elog,'(f10.6)',ix,iy,2,1,9) annotate plot
      sc=bigest(nt,b)/bigest(nt,db)
      do 175 i=1,nb
175   b(i)=b(i)+db(i)*alpha*sc      b ← b + α b
      call move(nt,b,cb)
      call fftr(nth,cb,+1.)      B ← b
      sc=dot(nt+2,cb,cb)/nth
      do 350 i=1,nthp                          don't divide by zero
      den=(amb*amb+alpha*alpha)*sc+conjg(cb(i))*cb(i)
```

```
350     cdx(i)=conjg(cb(i))*(cy(i)-cx(i)*cb(i))/den
        call fftri(nth,cdx,-1.)                          dx ← DX
        call plot(ink,idy,180+idy*(2*nt+2*nb),nt,dx)
        do 375 i=1,nt
375     x(i)=x(i)+dx(i)                  x ← x + dx
400     continue
        stop
        end
        function grad(en,n,x,g)          computer equation 18
        dimension x(n),g(n),v(128)
        su=0.
        sulax=0.
        do 10 i=1,n
        xi=x(i)
        ax=abs(xi)+1.e-20
        alx=alog(ax)
        u=exp(en*alx)
        su=su+u
        sulax=sulax+alx*u
        deriv=(u*xi/ax)/ax
        g(i)=deriv
10      v(i)=alx*deriv
        scv=-en*en/su
        scg=en*en*sulax/(su*su)
        do 20 i=1,n
20      g(i)=g(i)*scg+v(i)*scv
        grad=alog(su)-en*sulax/su
        return
        end
        subroutine job(nt,x,y,nb,b,amb,rmssn)   setup test case
        dimension x(nt),b(nt),y(nt),temp(200)
        read(17,17)(b(i),i=1,nb)         read synthetic bubble
17      format(f5.1)
        do 10 i=1,nt
10      x(i)=0.                          set up spike series
        x(nt/6)=1.01                     zeros, 1, zeros, .5, -.5, zeros
        x(nt/2)=.5
        x(nt/2+1)=-.5
        rock=grad(2.0,nt,x,y)
        write(11,77) rock
77      format('preconceived best answer =',f20.8)
        call conv(nt,x,nb,b,temp)
        call move(nt,temp,y)
        call scale(1./bigest(nt,y),nt,y)
        rmssn=sqrt(dot(nt,y,y)/nt)/amb
        do 38 i=1,nt                     add noise
38      y(i)=y(i)+2.*(ran(i1,i2)-.5)*amb
        write(11,78)amb,rmssn
78      format('ambient=',f9.6,'       rms s/n=',f7.1)
        do 40 i=1,nt                     first filter guess
40      b(i)=0.
        b(3)=1.
        do 50 i=1,nt
50      x(i)=0.
        do 60 i=3,nt
60      x(i-2)=y(i)
```

```
      return
      end
      subroutine plot(iter,idy,ishift,n,p)
      dimension p(n)
      logical*1 m,d,e
      data m,d,e/'m','d','e'/
      call setmod(9,512)
      b=0.
      do 10 i=1,n
10    if(abs(p(i)).gt.b)b=abs(p(i))
      do 20 i=1,n
      iy=ishift+idy*i
      ix=iter*70+p(i)*45./b+.5
      if(i.eq.1)write(9)m,ix,iy
      write(9)d,ix,iy
      iy=iy+idy
      write(9)d,ix,iy
20    continue
      return
      end
```

move pen to (ix, iy)

draw line to (ix, iy)

```
      subroutine conv(nf,f,nx,x,y)
      dimension f(nf),x(nx),y(1)
      l=nf+nx-1
      do 10 i=1,l
10    y(i)=0.
      do 20 i=1,nf
      do 20 j=1,nx
      k=i+j-1
20    y(k)=y(k)+f(i)*x(j)
      return
      end
      real function bigest(n,x)
      dimension x(n)
      b=0.
      do 10 i=1,n
10    if(abs(x(i)).gt.b) b=abs(x(i))
      bigest=b
      return
      end
      function dot(n,x,y)
      dimension x(n),y(n)
      xy=0.
      do 10 i=1,n
10    xy=xy+x(i)*y(i)
      dot=xy
      return
      end
      subroutine cross(n,x,d,nf,g)
      dimension x(n),d(n),g(nf)
      do 10 j=1,nf
10    g(j)=g(j)+dot(n-j+1,x,d(j))
      return
      end
      subroutine zero(n,x)
      dimension x(n)
      do 10 i=1,n
10    x(i)=0.
      return
      end
      subroutine move(n,x,y)
      dimension x(n),y(n)
      do 10 i=1,n
10    y(i)=x(i)
      return
      end
      subroutine scale(sc,n,x)
      dimension x(n)
      do 10 i=1,n
10    x(i)=x(i)*sc
      return
      end
      subroutine fft(lx,cx,signi,scale)
c   Complex Fourier transform.                          (JFC 9/76)
c
c                          lx           signi*2*pi*i*(j-1)*(k-1)/lx
c   cx(k)  =  scale * sum cx(j) * e
```

```
c                       j=1              for k=1,2,...,lx=2**integer
c
        complex cx(lx),cmplx,cw,cdel,ctemp
        do 05 i=1,lx
 05     cx(i)=cx(i)*scale
        j=1
        do 30 i=1,lx
        if(i.gt.j) go to 10
        ctemp=cx(j)
        cx(j)=cx(i)
        cx(i)=ctemp
 10     m=lx/2
 20     if(j.le.m) go to 30
        j=j-m
        m=m/2
        if(m.ge.1) go to 20
 30     j=j+m
        l=1
 40     istep=2*l
        cw=1.
        arg=signi*3.14159265/l
        cdel=cmplx(cos(arg),sin(arg))
        do 60 m=1,l
        do 50 i=m,lx,istep
        ctemp=cw*cx(i+l)
        cx(i+l)=cx(i)-ctemp
 50     cx(i)=cx(i)+ctemp
 60     cw=cw*cdel
        l=istep
        if(l.lt.lx) go to 40
        return
        end
        subroutine fftr(lx,cx,signi)
c  Fourier transform of a real time function.
c  inputs-
c        lx=2**integer
c        cx=x(1)...x(2*lx) ,  dimensioned as x(2*lx+2)
c        signi = +1.  or -1.
c  output-
c        cx(1)...cx(lx+1) the spectrum on  0.le.omega.le.pi
        complex cx(lx),conjg,cmplx,cw,cdel,ca,cb
        call fft(lx,cx,-signi,.5)
        cx(lx+1)=cx(1)
        lxh=lx/2+1
        cw=(0.,1.)
        arg=signi*3.14159265/lx
        cdel=cmplx(cos(arg),sin(arg))
        do 10 j=1,lxh
        jr=lx-j+2
        ca=conjg(cx(j))+cx(jr)
        cb=(conjg(cx(j))-cx(jr))*cw
        cx(j)=ca+cb
        cx(jr)=conjg(ca-cb)
 10     cw=cw*cdel
        return
        end
```

```
      subroutine fftri(lx,cx,signi)
c Inverse Fourier transform to a real time function.
      complex cx(lx),cmplx,cw,cdel,conjg,ca,cb
      nh=lx/2+1
      cw=(0.,-1.)
      arg=signi*3.14159265/lx
      cdel=cmplx(cos(arg),sin(arg))
      do 10 j=1,nh
      jr=lx-j+2
      ca=cx(j)+conjg(cx(jr))
      cb=cx(j)-conjg(cx(jr))
      cb=cb*cw
      cx(j)=conjg(ca+cb)
      cx(jr)=ca-cb
10    cw=cw*cdel
      call fft(lx,cx,-signi,1./(2.*lx))
      return
      end

      subroutine fit(nf,xx,xg,f)
      dimension xx(nf),xg(nf),f(nf)
      dimension a(200),b(200)
      do 10 j=1,nf
      f(j)=0.
      a(j)=0.
10    b(j)=0.
      f(1)=xg(1)/xx(1)
      a(1)=1.
      b(1)=1.
      v=xx(1)
      do 85 j=2,nf
      e=0.
      do 20 i=1,j
20    e=e+xx(j-i+1)*a(i)
      c=e/v
      do 30 i=1,j
30    a(i)=a(i)-c*b(j-i+1)
      do 40 i=1,j
40    b(i)=a(i)
      v=v*(1.-c*c)
      e=0.
      do 50 i=1,j
50    e=e+xx(j-i+1)*f(i)
      c=(e-xg(j))/v
      do 60 i=1,j
60    f(i)=f(i)-c*a(j-i+1)
      e=0.
85    continue
      return
      end
      subroutine number(fltnum,fmt,x,y,size,orient,iunit)
      real fltnum
      logical*1 fmt(12),null,blnk,num(21),e
      integer x,y,size,orient
      data null/z00/,blnk/' '/,e/'e'/
      encode(20,fmt,num) fltnum
```

Levinson Shaper Program. Not actually used to generate Figs 1 & 2.

plot annotating program. Probably won't work on anything but PDP11 Fortran

```
        ist=1
10      if(num(ist).ne.blnk) go to 20
        ist=ist+1
        go to 10
20      iend=ist+1
30      if(num(iend).eq.blnk.or.iend.gt.20) go to 40
        if(num(iend).eq.e) iend=iend+1
        iend=iend+1
        go to 30
40      num(iend)=null
        call symbol(num(ist),x,y,size,orient,iunit)
        return
        end
        subroutine symbol(string,x,y,size,orient,iunit)
        logical*1 string(41),t,m,null
        integer x,y,size,orient,ikey
        data t/'t'/,m/'m'/,null/z00/
        ikey=size+32*orient
        n=1
10      if(string(n).eq.null) goto 20
        n=n+1
        go to 10
20      write(iunit) m,x,y,t,ikey,(string(i),i=1,n)
        return
        end
```

Oct 2 20:20 1977 field/disk.f Page 1

Like pard0 program but gets multichannel data from a disk

```
      real x(514),y(514),g(514),b(200),db(514),elog(48)
      complex cx(257),cy(257),cg(257),cb(257),cdb(257)
      equivalence(x,cx),(y,cy),(db,cdb)
      real space(514)
      equivalence(g,cg)
      complex conjg,cmplx
      logical*1 filcy(50)
      data filcy/'cy'/
      call setmod(11,128)
      call setmod(5,512)
      call setfil(61,'/scr/jon/wavelet',512+128)
```

job control language defining input unit 5 and output units 11 and 61

```
      nch=12
      nt=512
      nt2=nt+2
      nth=nt/2
      nt2h=nt2/2
      idy=1
      nb=200
      niter=21
      nskip=10
      amb=1./200.
      rmssn=1./amb
      call zero(nb,b)
      read(61) nbb,(b(i),i=1,nbb)
      call zero(nt2,cb)
      call move(nb,b,cb)
      call fftr(nth,cb,+1.)
      ifilcy=icreat(filcy,6*64+4*8+4)
      ncl=iclose(ifilcy)
      nop=iopen(filcy,2)
      do 60 ich=1,nch
      read(5)(y(i),i=1,nt)
      sc=bigest(nt,y)
      call scale(1./sc,nt,y)
      call taper(nt,y)
      call fftr(nth,cy,+1.)
      do 50 i=1,nt2h
50    cx(i)=cy(i)/cb(i)
      cy(1)=cmplx(real(cy(1)),real(cy(nt2h)))
      nwrt=iwrite(ifilcy,cy,4*nt)
      call fftri(nth,cx,-1.)
      elog(ich)=grad(2.,nt,x,g)
60    continue
      do 200 iter=1,niter
      alpha=.10*(niter-iter+.5)/niter
      en=2.5-.5*((iter-1)/((niter+1)/2))
      call zero(nt2,cdb)
      nseek=iseek(ifilcy,0,0)
      do 100 ich=1,nch
      nread=iread(ifilcy,cy,4*nt)
      cy(nt2h)=aimag(cy(1))
      cy(1)=real(cy(1))
      do 180 i=1,nt2h
180   cx(i)=cy(i)/cb(i)
      call fftri(nth,cx,-1.)
      elog(ich)=grad(en,nt,x,g)
```

create a disk file for cy

close it

open it again

initialization loop over channels

pack folding frequency

write FT of input data

iterative descent starts here

"rewind" unit containing FT of input data

loop over channels

read a channel

unpack folding freq

get input from filter and output

CB CY

```
      if(mod(iter-1,nskip).ne.0)go to 80
      ink=ich+nch*((iter-1)/nskip)
      call plot(ink,idy,100,nt,cx)
      call plot(ink,idy,160+idy*(nt+nb*2),nt,g)
      ix=70*ink
      iy=0
      call number(elog(ich),'(f10.6)',ix,iy,2,1,9)
80    continue
      call fftr(nth,cg,+1.)
      call fftr(nth,cx,+1.)
      do 90 i=1,nt2h
90    cdb(i)=cdb(i)-cg(i)*cb(i)*conjg(cx(i))
100   continue
      call fftri(nth,cdb,-1.)
      do 140 i=1,nb
140   db(i)=(nb-i)*db(i)
      sc=dot(nb,b,db)/dot(nb,b,b)
      do 150 i=1,nb
150   db(i)=db(i)-sc*b(i)
      sc=bigest(nb,b)/bigest(nb,db)
      do 160 i=1,nb
160   b(i)=b(i)-db(i)*alpha*sc
      call plot(ink,idy,120+idy*nt,nb,b)
      call plot(ink,idy,140+idy*(nt+nb),nb,db)
      write(11,170) (elog(i),i=1,nch)
170   format(16f5.2)
      rewind 61
      write(61) nb,(b(i),i=1,nb)
      call zero(nt,cb)
      call move(nb,b,cb)
      call fftr(nth,cb,+1.)
      sc=(alpha*alpha+amb*amb)*dot(nt2,cb,cb)/nth
      do 175 i=1,nt2h
175   cb(i)=(sc+conjg(cb(i))*cb(i))/conjg(cb(i))
200   continue
      stop
      end
```

$d\beta = \sum_{\text{channels}} \bar{X} G B$

taper db

remove projection on b

update wavelet

write wavelet

pad zeros

remove zeros from B so we can divide by it