Stanford Exploration Project, Report 73, November 18, 1997, pages 395–415

Stanford Exploration Project, Report 73, November 18, 1997, pages 395–415

Anti aliasing

Jon F. Claerbout[1]

keywords: *velocity, aliasing, modeling, synthetic data, Fourier transform*

ABSTRACT

Field arrays filter spatially. On dipping arrivals this produces a temporal filtering effect. To match field arrays, Kirchhoff operators are designed to antialias with short rectangle and triangle smoothing functions whose duration depends on the local trace-to-trace moveout. Integrals under triangular shaped weighting windows are rapidly computed from the double integral of the trace. Each integral is a weighted sum of three values of the double integral, one from each corner of the triangle. Complete code is given along with examples.

FIELD ARRAYS

In geophysical data recording there is usually a local array whose elements are added locally before a single channel is recorded. For example, the SEP student group once laid out more than 4056 geophones which they grouped and added into 169 separate recorders. We may think of the local superposition as an integration over a small interval of space to create a sampled space function from a continuous one. With vibrator sources, it is also customary to vibrate on various nearby source locations and sum them into a single signal. Figure 1 is a caricature of what happens. On the left a data field appears to be a continuous function of space (it is actually 500 spatial locations) with various impulsive signals at different times and distances. For simplicity, all signals have unit amplitude. The 500 signals are segregated into 10 groups of 50 and each group of 50 is summed into a single channel. The various signals sum to functions that could be called "slump shouldered rectangles." If both x and t-meshes were refined further, the "slump shoulders" on the rectangles would diminish in importance and we would notice that the rectangles were still imperfect. This is because the rectangle approximation arises from the approximation that the hyperbola is a straight line over the recording array. In reality, there is curvature and the effect of curvature is strongest near the apex, so the rectangle approximation is poor at the apex.

Some of the rectangles are longer than others. The narrow ones are tall and the

[1]**email:** not available

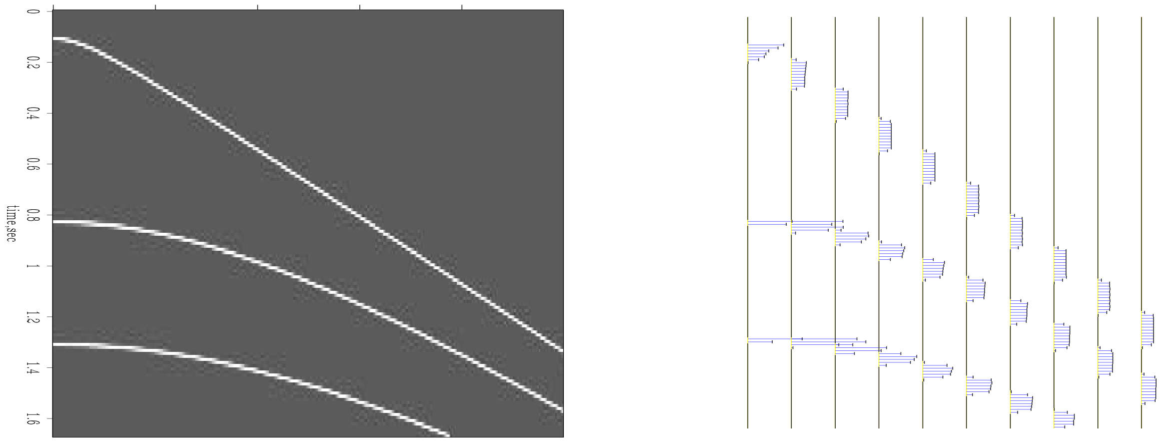

Figure 1: Quasicontinuous field (left) added in groups (right). jon3.trimo-oversamp [ER]

wide ones are short because the area of each rectangle must be 50 (being the sum of 50 channels each holding a 1). Since the rectangles all have the same area, were we to lowpass filter the recorded data we would recover the original characteristic that all these signals have the same amplitude.

If the 500 channels had been individually recorded we could convolve them along the horizontal axis with a 50 point long rectangle function. The output could be sampled once every 50 points to give the coarse-mesh data in Figure 1. Convolving with a rectangle function of x is dual to multiplying by a sinc function $\sin(k_x \Delta x)/(k_x \Delta x)$ in the Fourier domain. The sinc function has its first zero where one wavelength of a spatial sinusoid precisely fits inside the rectangle function. When 49 out of 50 channels are abandoned we have spatial aliasing (spectral folding) of higher wavelengths into lower ones. Mathematically, we would prefer to have a rectangle in the k_x-domain which would erase the energy that would otherwise be aliased to lower frequencies and would preserve the lower frequencies without scaling them. Thus Fourier transform theory implies we should use the (unruly) sinc function as a weighting function for the field operations. As you might imagine, the subject of antenna design is the subject of many patents that address the fundamental issues in various ways.

PROCESSING AS CONJUGATE TO FIELD OPERATION

Knowing how data is recorded, or how we would like it to be recorded, suggests various possibilities for data processing. Should we ignore the little rectangle functions, or should we include them in the data processing? Figure 2 shows a simple model and its implied data, along with migrations, with and without attention to aliasing the horizontal space axis. The figure shows that migration without attention to aliasing leads to systematic noise and (apparently) random noise.

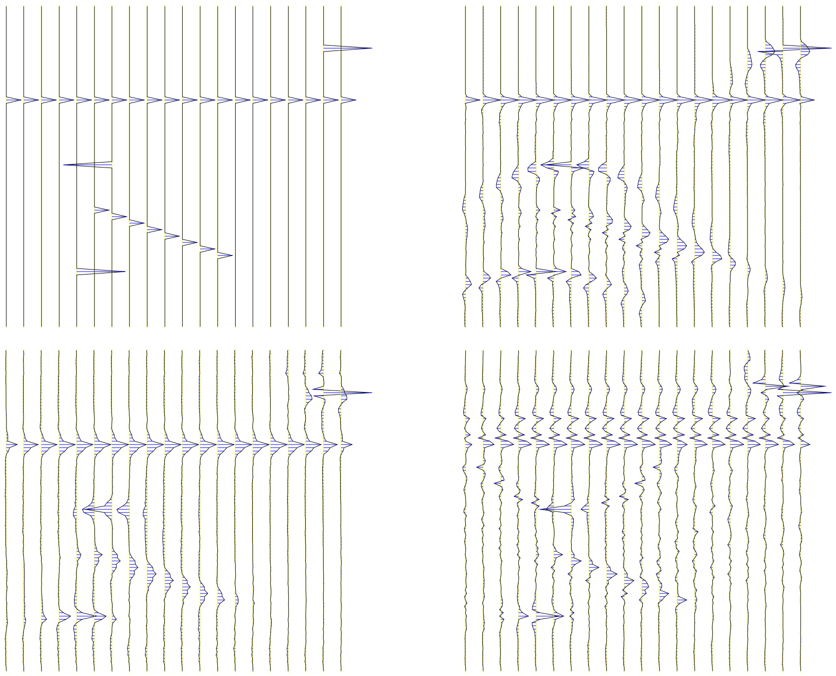

Figure 2: Top left is a synthetic image. Top right is synthetic data from the synthetic image. Bottom are migrations of the data with and without antialiasing. jon3.trimo-migalias [ER]

This figure is based on realistic parameters except that I compute and display the results on a very coarse mesh (20×100) to enable you to see clearly the phenomena of numerical analysis. No additional values were used between mesh points or off the edges of what is shown.

Fourier and finite-difference methods of migration seem to be immune to an aliasing malady suffered by hyperbola summation (Kirchhoff) migration. Here we limit our studies to Kirchhoff-type migrations. We will see a simple method to overcome this "operator-aliasing" malady. Operator aliasing should not be confused with data aliasing which is a deeper malady for which there is no direct cure.

The practical need to limit operator aliasing is often reduced by three indirect measures. First is temporal low pass filtering which has the unfortunate side effect of reducing the temporal bandwidth. Second is dip limiting (limiting the aperture of the hyperbola) which has the unfortunate side effect of limiting the dip bandwidth. Third is interlacing the data traces. Interpolating the data also interpolates the operator so if enough trace interpolation is done, the operator is no longer subsampled. A disadvantage of data interpolation is that the data becomes more bulky. Here we attack the operator aliasing problem directly.

Look at Figure 3. Note that in some places each value of x corresponds to several values of t, and other places it is the opposite where one value of t corresponds to several values of x. An aliasing problem arises when we approximate a line integral by a simple sum of points, one for each value on the x-axis instead of using the more complicated trajectory that you see in Figure 3.

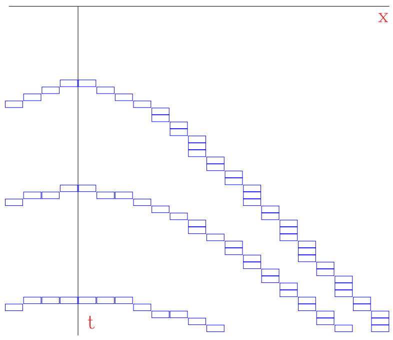

Figure 3: To integrate along hyperbolas without aliasing, you should include (at least) the points shown. jon3.trimo-nmotraj [ER]

A simple program designed for antialiasing gave the result in Figure 4. A zero-offset signal is input to conjugate NMO to make synthetic data which is then NMO'ed and stacked. Notice that the end of each rectangle is the beginning of the rectangle at the next offset. You might fear the coding that led up to Figure 4 is a fussy and

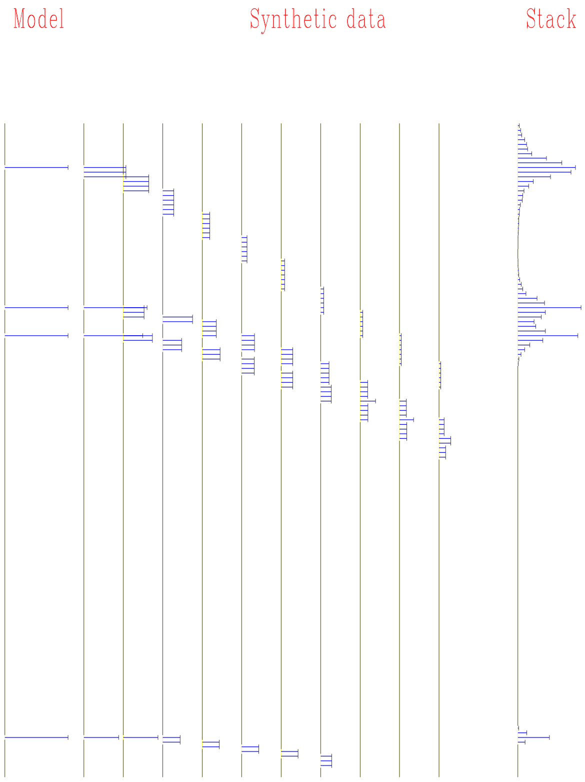

Figure 4: Rectangle smoothing during NMO and stacking. Notice that the end of one rectangle exactly coincides with the beginning of the rectangle at next larger offset. Thus, rectangle width increases with offset and decreases with time. (`antialias=1.`) jon3.trimo-boxmo1 [ER]

inefficient business because of all the short little summation loops. Luckily, there is a marvelous little formula that allows us to express the integral under any of the little rectangles, no matter how many points it contains, by a single subtraction. Integration is the key. It is only necessary to realize that the sums are, like a definite integral, representable by the difference of the indefinite integral at each end. In other words, to find the sum of all the values between `it` and `it+n` we begin with a recursive summation such as qq(it)=qq(it-1)+pp(it). Then, any sum of values like pp(it)+···+p(it+n) is simply qq(it)-qq(it+n+1).

Figure 4 is not fully consistent with Figure 3. In Figure 4 notice that the last point in each rectangular area overlaps the next rectangular area by one point. Overlap could be avoided by shortening each rectangle by one point, but then rectangles near the apex of the hyperbola would have *zero length* which is wholly unacceptable. Should we write a code to match Figure 3? This would be better, but far from perfect. Notice in Figure 3 that a horizontal sum of the number of boxes is not a smooth function of time. To achieve more smoothness, we later turn to triangles, but first we look at some implementation details for rectangles.

CODING NMO WITH RECTANGLES

A subroutine for causal summation is causint().

```
# causal integral
#
subroutine causint(  conj, add,        n,pp,  qq   )
integer i,        n, conj, add;   real pp(n), qq(n )
temporary real tt(n)
call conjnull(       conj, add,        pp,n,  qq,n )
if( conj == 0 ) {                       tt(1) = pp(1)
                                        do i= 2, n
                                                tt(i) = tt(i-1) + pp(i)
                                        do i= 1, n
                                                qq(i) = qq(i) + tt(i)
                                        }
else {          tt(n) = qq(n)
                do i= n, 2, -1
                        tt(i-1) = tt(i) + qq(i-1)
                do i= 1, n
                        pp(i) = pp(i) + tt(i)
                }
return; end
```

The conjugate of causal integration is anticausal integration. (See PVI (Claerbout, 1992) for fuller explanation of conjugates and this detail.) For each reflector, data modeling proceeds by throwing out two pulses of opposite polarity. Then causal summation produces a rectangle (also called "box car") between the pulses. When moving out data, anticausal summation is the first step. Thus each impulse in the data becomes a rectangle from the impulse to $t = 0$. Then subtracting values at rectangle ends gives the desired integral of data in the rectangle. The code is in subroutines boxmo() and boxstack(). The travel-time depth τ is denoted by z in the code. The inverse of the earth velocity $v(\tau)$, called the slowness $s(\tau)$, is denoted by slow(iz).

```
subroutine boxstack( conj,add,slow,antialias, t0,dt,x0,dx,nt,nx, stack, gather)
```

```
integer conj, add, ix, nx, nt
real x,              slow(nt),antialias, t0,dt,x0,dx, stack(nt), gather(nt,nx)
call conjnull( conj, add, stack, nt, gather, nt*nx)
do ix= 1, nx {  x =  x0 + dx * (ix-1)
        call boxmo(conj,1,t0,dt,dx,x,nt,slow,antialias, stack, gather(1,ix))
        }
return; end

subroutine boxmo( conj, add, t0,dt, dx, x, nt,slow,  antialias, zz,     tt   )
integer it,iz,itp,conj, add,               nt
real t, tp, z, amp,         t0,dt, dx, x, slow(nt), antialias, zz(nt), tt(nt)
temporary real ss(nt)
call null(     ss,nt);          call conjnull( conj, add,       zz,nt,  tt,nt)
if( conj != 0)                  call causint(     1,   0, nt, ss, tt)
do iz= 1, nt {  z = t0 + dt*(iz-1)
        t = sqrt( z**2 + (slow(iz)* abs(x)         )**2 ); it = 1.5 + (t -t0)/dt
        tp= sqrt( z**2 + (slow(iz)*(abs(x)+abs(dx)))**2 )
        tp = t + antialias * (tp - t) + dt;               itp= 1.5 + (tp-t0)/dt
        amp =  sqrt( nt*dt/(t+dt)) * (z+dt)/(t+dt) / (itp - it)
        if ( itp < nt   ) {
                if( conj == 0 ) {       ss(it ) = ss(it ) + amp * zz(iz)
                                        ss(itp) = ss(itp) - amp * zz(iz)
                                        }
                else            {       zz(iz) = zz(iz)  + amp * ss(it )
                                        zz(iz) = zz(iz)  - amp * ss(itp)
                                        }
                }
        }
if( conj == 0)          call causint(     0,  add, nt, ss, tt)
return; end
```

To find the end points of the rectangular intervals, given the vertical travel time, I get the time `t`, in the usual way. Likewise I get the time, `tp`, on the next further-out trace for the ending location of the rectangle wavelet. I introduce a parameter called `antialias` that can be used to increase or decrease the `tp-t` gap. Normally `antialias=1.`

Theoretical solutions to various problems lead to various expressions for amplitude along the hyperbola. I set the amplitude `amp` by a complicated expression that I do not defend or explain fully here but merely indicate that: a "divergence" correction is in the factor `sqrt( nt*dt/(t+dt))`; a cosine like "obliquity" scale is z/t; and the wavelet area must be conserved, so the height is inversely proportional to the pulse width `(itp - it)`. Wavelet area is conserved to assure that after low-pass filtering, the strength of a wave is independent of whether it straddled two mesh points as $(.5, .5)$ or lay wholly on one of them as $(1, 0)$.

To test a limiting case, I set the antialias parameter to zero and show the result in Figure 5 which is the same as the simple prescription to "sum over the x-axis." We notice that the final stack is not the perfect impulses that we began with. The explanation is: information can be expanded in time and then compressed with no loss, but here it is compressed first and then expanded, so the original location is smeared. Notice also that the full amplitude is not recovered on the latest event. The explanation is that a significant fraction of the angular aperture has been truncated at the widest offset.

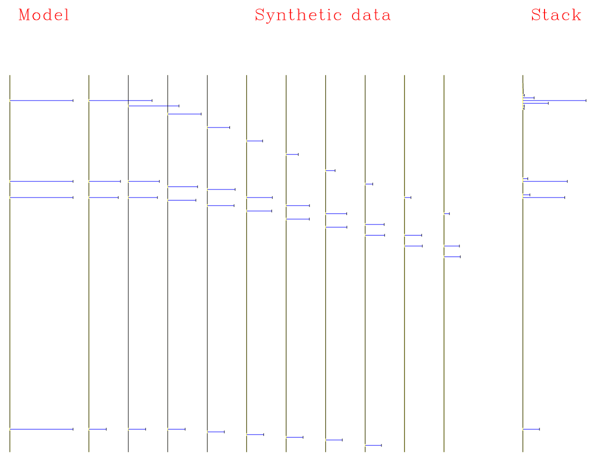

Figure 5: Rectangles shortened to one point duration. (`antialias=0.`) jon3.trimo-boxmo0 [ER]

CODING NMO WITH TRIANGLES

We should take some care with anti-aliasing in data processing. The anti-aliasing measures we take, however, need not match the field recording. If the field arrays were rectangles, we could use triangles or sincs in the data processing. It happens that triangles are an easy extension of the rectangle work we have been doing and triangles make a big step in the right direction.

For an input pulse, the output of integration is a step. The output of a second integration is a ramp. For an input triplet $(1, 0, 0, -2, 0, 0, 1)$ the output of two integrations is a short triangle. An easy way to assure time alignment of the triangle center with the triplet center is to integrate once causally and once anticausally as done in subroutine `doubint()`.

You can imagine placing the ends and apex of each triangle at a nearest neighbor mesh point as we did with the rectangles. Instead I place these ends more precisely on the mesh with linear interpolation using subroutine `spotw()` which adds or extracts a value to any location on a vector where the value may lie between mesh points.

```
#   Scaled linear interpolation.
#
subroutine spotw( conj, add, scale, nt,t0,dt, t, val,   vec   )
integer it,itc,   conj, add,        nt
real tc, fraction,          scale,    t0,dt, t, val,   vec(nt)
call conjnull(    conj, add,                    val,1, vec,nt)
tc = (t-t0) / dt;       itc = tc;        it = 1 + itc;  fraction = tc - itc
if( 1 <= it  &  it < nt) {
        if( conj == 0) {
                vec(it  ) = vec(it  ) + (1.-fraction) * val * scale
                vec(it+1) = vec(it+1) +   fraction    * val * scale
```

```
                }
        else
                val = val + ((1.-fraction) * vec(it)  +
                               fraction    * vec(it+1)  ) * scale
        }
else
        call erexit('spotw: at boundary')
return; end
```

Using these subroutines, I assembled the stacking subroutine `tristack()` and the NMO routine `trimo()` with triangle wavelets. The triangle routines are like those for rectangles except for some minor changes. Instead of computing the theoretical locations of impulses on nearer and further traces, I assumed a straight line tangent to the hyperbola $t^2 = \tau^2 + x^2/v^2$. Differentiating by x at constant τ gives the slope $dt/dx = x/(v^2 t)$. As before, the area of the the wavelets, now triangles must be preserved. The area of a triangle is proportional to the base times the height. Since the triangles are built from double integration ramp functions, the height is proportional to the base length. Thus to preserve areas, each wavelet is scaled by the inverse *squared* of the triangle's base length. Results are shown in Figures 6 and 7.

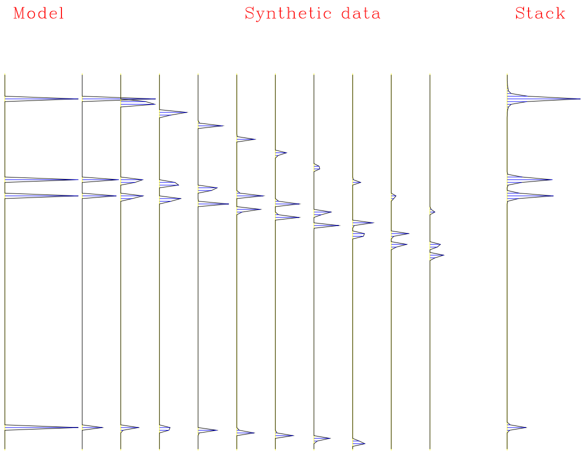

Figure 6: Triangle wavelets, accurately positioned, but aliased (`antialias=0.`) jon3.trimo-trimo0 [ER]

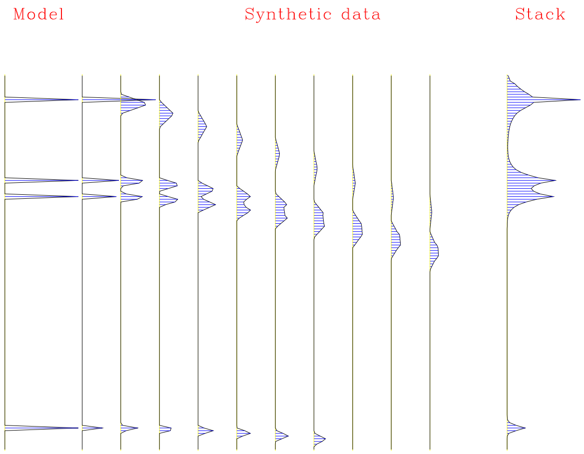

Figure 7: Antialiased triangle wavelets. (`antialias=1.`) Where ever triangle duration is more than about three points, the end of one triangle marks the apex of the next. jon3.trimo-trimo1 [ER]

```
# Modeling and stacking using triangle weighted moveout.
#
subroutine tristack( conj,add, slow,anti,t0,dt,x0,dx, nt,nx, stack, gather   )
integer ix,          conj,add,                        nt,nx
real x,                    slow(nt),anti,t0,dt,x0,dx, stack(nt), gather(nt,nx)
call conjnull(       conj, add,                       stack,nt,  gather,nt*nx)
do ix= 1, nx {  x =  x0 + dx * (ix-1)
        call trimo(conj,1,t0,dt,dx, x, nt,slow,0.,1.,anti,stack, gather(1,ix))
        }
return; end
```

```
# moveout with triangle shaped smoothing window.
#
subroutine trimo( conj, add, t0,dt, dx,x, nt, slow, s02, wt, anti, zz,    tt )
integer iz,itp,itm, conj, add,            nt
real                        t0,dt, dx,x, slow(nt), s02, wt, anti, zz(nt),tt(nt)
real z, t,tm,tp, amp, slope
temporary real ss(nt)
call null(     ss,nt);  call conjnull( conj, add,                 zz,nt, tt,nt)
if( conj != 0 )         call doubint( 1, 0, nt, ss, tt)
do iz= 1, nt {  z = t0 + dt * (iz-1)
        t = sqrt( z**2 + (slow(iz) * x)**2 )
        slope = anti * ( slow(iz) * slow(iz) - s02 ) * x / (t+dt)
        tp = t + abs(slope * dx) + dt;   itp = 1.5 + (tp-t0) / dt
        tm = t - abs(slope * dx) - dt;   itm = 1.5 + (tm-t0) / dt
        if( itm <= 1  )  next
        if( itp >= nt )  break
        amp = wt * sqrt( nt*dt/(t+dt)) * (z+dt)/(t+dt) * (dt/(dt+tp-tm)) ** 2
        call spotw( conj, 1,  -amp, nt,t0,dt,tm, zz(iz), ss)
        call spotw( conj, 1, 2*amp, nt,t0,dt,t , zz(iz), ss)
        call spotw( conj, 1,  -amp, nt,t0,dt,tp, zz(iz), ss)
        }
if( conj == 0)          call doubint( 0, add, nt, ss, tt)
return; end
```

MIGRATION

Subroutine `aamig()` below does migration and diffraction modeling using subroutine `trimo()` as the workhorse.

```
# anti-aliased kirchhoff migration (conj=1) and modeling (conj=0)
#
subroutine aamig( conj,add, slow,antialias,t0,dt, dx, nt,nx, image, data  )
integer conj, add, ix, nx, nt, iy
real    h,              slow(nt),antialias,t0,dt, dx, image(nt,nx), data(nt,nx)
call conjnull(    conj, add,                          image,nt*nx,  data,nt*nx)
do ix= 1, nx {
do iy= 1, nx {
        h  = dx * (iy - ix)
        call trimo( conj, 1, t0,dt,dx, h, nt,slow, 0., 1., antialias,
                                                    image(1,iy), data(1,ix))
```

```
        }}
return; end
```

Figure 8 shows the synthetic image that is used for testing. There is a horizontal layer, a dipping layer, and a few impulses. The impulses are chosen stronger than the layers because they will spread out in the synthetic data.

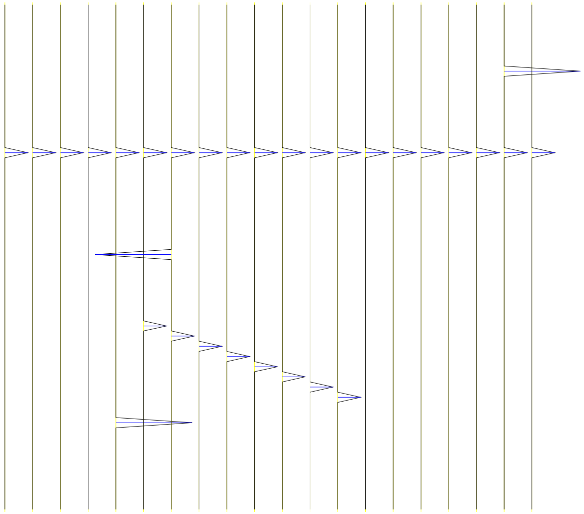

Figure 8: Model image for migration study. jon3.trimo-aamod [ER]

The velocity is taken constant. Figure 9 shows synthetic data made without regard for aliasing. The hyperbolas look fine—the way we expect. The horizontal layer, however, is followed by many pseudo layers. These pseudo layers are the result of modeling with an operator that is spatially aliased.

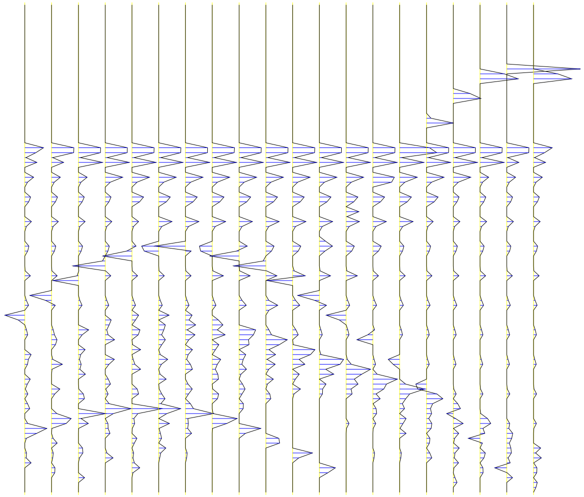

Figure 9: Synthetic data without regard for aliasing. Made from model image with `aamig()` taking `antialias=0`. jon3.trimo-aad0 [ER]

Figure 10 shows how the synthetic data improves dramatically when aliasing is taken into account. The layers look fine now. The hyperbolas, however, have a waveform that is rapidly changing with offset from the apex. This changing waveform is an inevitable consequence of the anti-aliasing. The apex has a huge amplitude because the temporal bandwidth is widest at the apex (because the dip is zero there, there is no filtering away of high spatial frequencies). Simple low-pass temporal filtering (not shown) will cause the wavelet to be largely independent of offset.

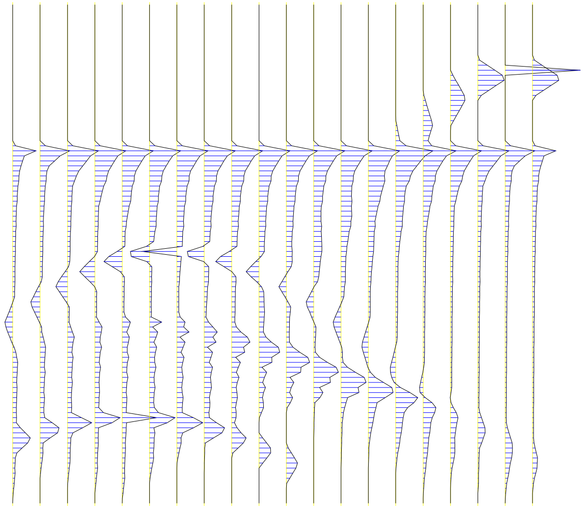

Figure 10: Synthetic data accounting for aliasing. Made from model image with `aamig()` taking `antialias=1.` jon3.trimo-aad1 [ER]

Do not confuse aliased data with synthetic data made by an aliased operator. To make aliased data, you would start from good data, such as Figure 10, and throw out alternate traces. More typically, the earth makes good data and we fail to record all the needed traces for the quality of our field arrays.

The horizontal layer in Figure 10 has a waveform that resembles a damped step function. It can be shown mathematically that: the decay is proportional to $1/\sqrt{t}$; this wavelet has an amplitude spectrum proportional to $1/\sqrt{\omega}$; and the phase is such that this waveform can be converted back to about the impulsive waveform in the original synthetic image by a causal half-order derivative operator. The half-order derivative operator is well known (Kjartansson, 1979), (Claerbout, 1985) and (1992). For Figure 11 I used the subroutine `halfdif()` given in PVI. You can see the final flat-layer waveform is roughly the zero-phase shape we started with.

Figure 12 shows my best migration of my best synthetic data. All the features of the original model are apparant. Naturally, high frequencies are lost, more on the dipping bed than the level one. Likewise the broadening of the deeper point scatterer compared to the shallow one is a well known aperture effect.

Figure 13 shows what happens when antialiasing is ignored in migration. Notice many false layers above the given horizontal layer. Notice semicircles above the impulses. Notice apparent noise everywhere.

USE OF ANTIALIASING PARAMETER

Migration requires antialiasing, even where the earth has zero dip. This is because the earth's horizontal layers cut across the migration hyperbola. An interesting extension

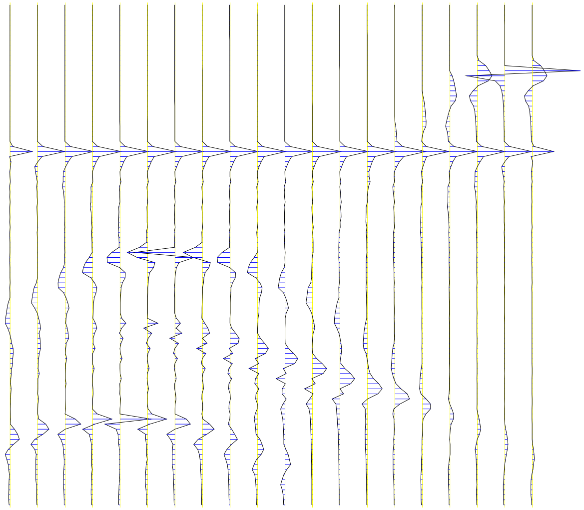

Figure 11: Best synthetic data. Made from model image using `aamig()` with `antialias=1` followed by a causal half-order time derivative. Lowpass temporal filtering would make wavelets more independent of location on a hyperbola. jon3.trimo-aad1h [ER]

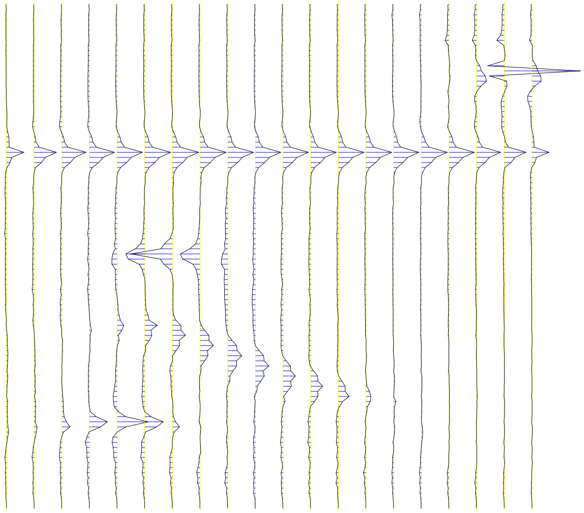

Figure 12: Best migration of best synthetic data. Uses `aamig()` with `antialias=2` followed by an anticausal half-order time derivative. jon3.trimo-aamig2 [ER]

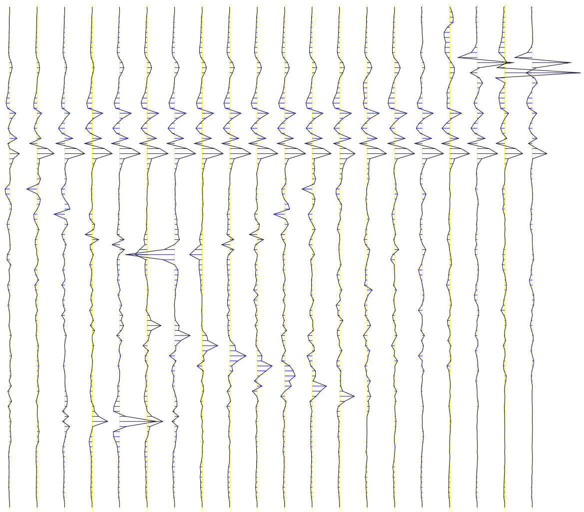

Figure 13: Migration of best synthetic data without regard for aliasing. Uses `aamig()` with `antialias=0.` (and an anticausal half-order time derivative) jon3.trimo-aamig0 [ER]

is where the earth has dipping layers. There the `slope` parameter could be biased to account for it.

Where the earth contains hyperbolas, they will cut steeply across our migration hyperbola. Figure 14 suggests that such hyperbolas require an antialias parameter greater than unity, say `antialias=2`.

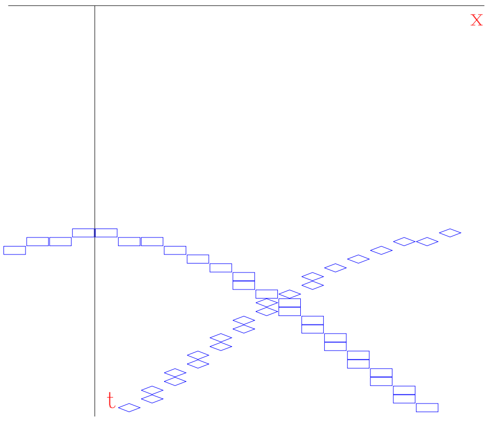

Figure 14: Crossing hyperbolas that do not touch. Thus the points shown are not enough to prevent spatial aliasing a line integral along one trajectory of signal on the other. jon3.trimo-croshyp [ER]

Orthogonality of crossing plane waves

Normally, waves do not contain zero frequency. Thus the time integral of a waveform normally vanishes. Likewise, for a dipping plane wave, the time integral vanishes. Likewise, a line integral across the (t, x)-plane along a straight line that crosses a plane wave or a dipping plane wave vanishes. Likewise, two plane waves with different slopes should be orthogonal if one of them has zero mean.

I suggest that spatial aliasing may be defined and analyzed with reference to plane waves rather than with reference to frequencies. Aliasing is when two planes that should be orthogonal, are not. This is like two different frequency sinusoids. They are orthogonal except perhaps if there is aliasing.

CMP GATHER

A common-midpoint gather holding data with only one velocity should stack OK without need for antialiasing. It is nice when antialiasing is not required because then high temporal frequencies need not be filtered away simply to avoid aliased spatial frequencies. When several velocities are simultaneously present on a CDP gather, we will find crossing waves. These waves will be curved, but aliasing concepts drawn from plane waves are still applicable. We designed the antialiasing of migration by expecting hyperbola flanks to be orthogonal to horizontal beds or dipping beds of some chosen dip. With a CDP gather we chose not a dip, but a slowness s_0. The slope of a wave of slowness s on a CDP gather is xs^2/t. The greater the contrast in dips, the more need for antialiasing. The slope of a wave with slowness s_0 is xs_0^2/t. The difference between this slope and that of another wave is $xs^2/t - xs_0^2/t$ or $(s^2 - s_0^2)x/t$ which in the program is the `slope` for the purpose of antialiasing. The choice of s_0 has yet to be determined according to the application. For illustration, I prepared a figure with three velocities, a very slow surface wave, a water wave, and a fast sediment wave. I chose s_0 to match the water wave. In practice s_0 might be the earth's slowness as a function of traveltime depth.

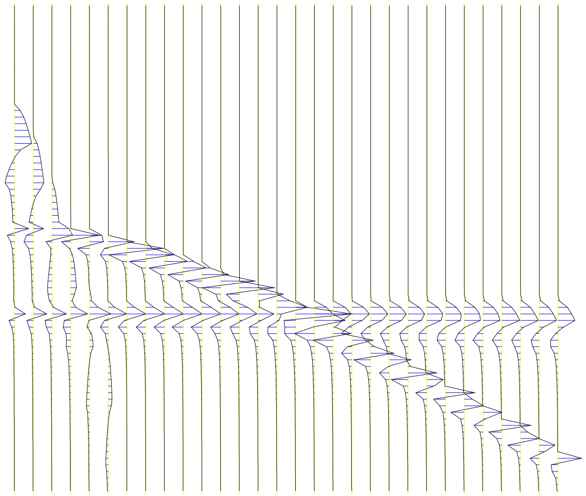

Figure 15: The air wave and fast wave are broadened increasingly with offset, but the water wave does not. This broadening enables crossing events to maintain their orthogonality. jon3.trimo-aacdp [ER]

Iterative velocity transform

After we use data to determine a velocity model (or slowness model) with an operator $\mathbf{A}$ we may wonder whether synthetic data made from that model with the conjugate operator $\mathbf{A}'$ resembles the original data. In other words, we may wonder how close the velocity transform $\mathbf{A}$ comes to being unitary. The first time I tried this, I discovered that large offsets and large slownesses were attenuated. With a bit of experimentation I found that the scale factor $\sqrt{sx}$ seems to make the velocity transform approximately a unitary one. Results are in Figure 16.

Figure 16 shows that on a second pass, the velocity spectrum of the slow wave is much smoothed. This suggests that it might be more efficient to parameterize the data with slowness *squared* rather than slowness itself. Another interesting thing about using slowness squared as an independent variable is that when slowness squared is negative (velocity imaginary) the data is matched by ellipses curving up instead of of hyperbolas curving down.

Figure 17 shows the effect of no antialiasing in either the field recording or the processing. The velocity spectrum is as sharp, if not sharper, but it is marred by a large amount of low-level noise.

Aliased data gives an interesting question. Should we use an aliased operator as in Figure 17 or should we use an antialiased operator as that in Figure 16? Figure 18

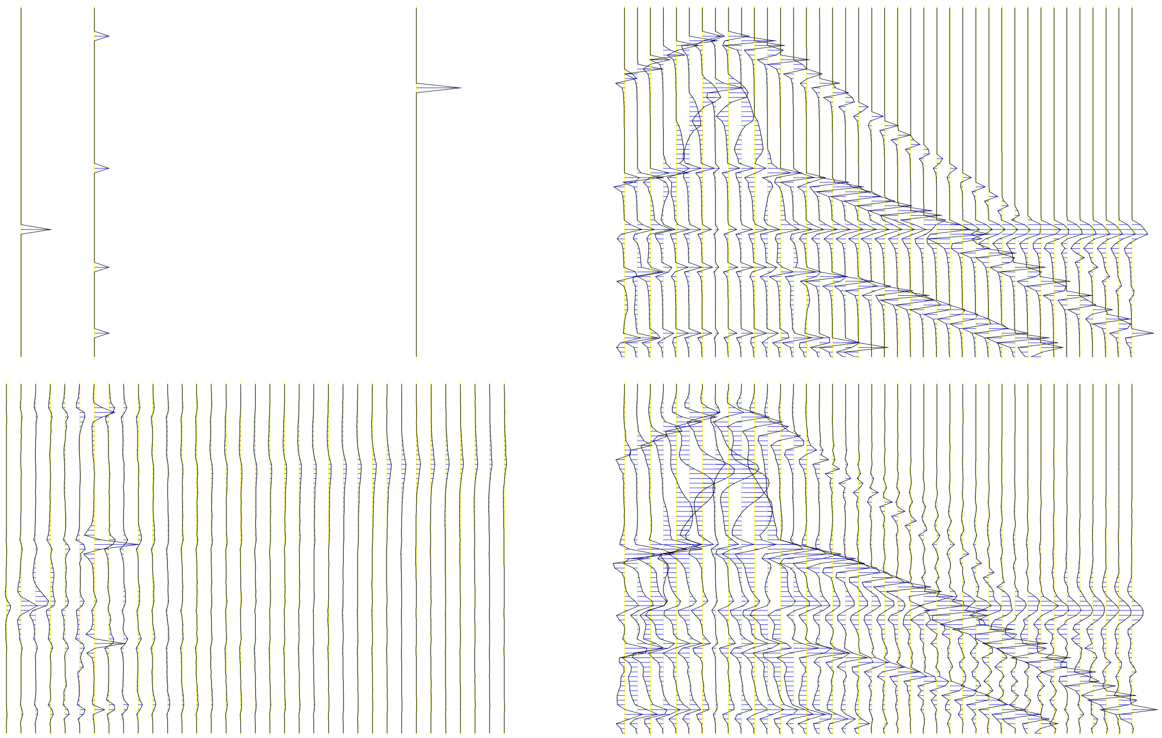

Figure 16: Top left: Slowness model. Top right: Data derived from it using the pseudounitary scale factor. Bottom left: the velocity spectrum of top right. Bottom right: data made from velocity spectrum. jon3.trimo-aavel1 [ER]

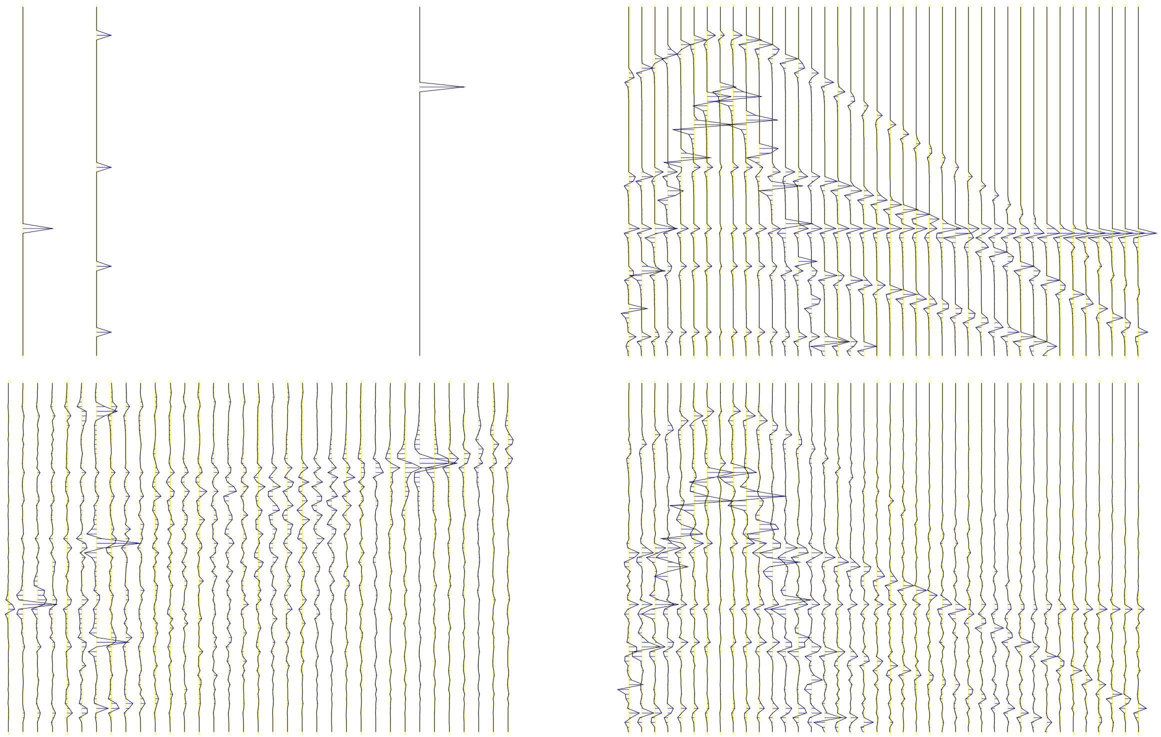

Figure 17: Like Figure 16 but with `antialias=0.` This synthetic data presumes no receiver groups in the field recording. jon3.trimo-aavel0 [ER]

shows the resulting velocity analysis. The antialiased operator seems well worth while, even when applied to aliased data.

In real life, the field arrays are not "dynamic" (able to respond with space and time variable s_0) but the data processing can be dynamic. Fourier and finite-difference methods of wave propagation and data processing are generally immune to aliasing difficulties. On the other hand, dynamic arrays in the data processing are a helpful feature of the ray approach whose counterparts seem unknown with Fourier and finite-difference techniques.

Since $\sqrt{sx}$ does not appear in physical modeling, people are sometimes hesitant to put it in the velocity analysis. If $\sqrt{sx}$ is omitted from the modeling, then $|sx|$ should be put in the velocity analysis. Failing to do so will give a result like in figure 19. The principal feature of such a velocity analysis is the velocity smearing. A reason for smearing is that the zero-offset signal is strong in all velocities. Multiplying by $\sqrt{sx}$ kills that signal (which is never recorded in the field anyway). The conceptual advantage of a pseudounitary transformation like Figure 16 is that points in velocity space are orthogonal components like Fourier components whereas for nonunitary transforms like with Figure 19 the different points in velocity space are not orthogonal components.

Subroutine `veltran()` does the work.

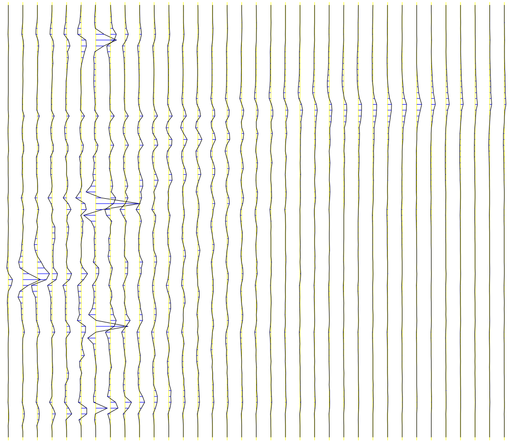

Figure 18: Aliased data analyzed with antialiased operator. Compare to the lower left of Figure 17. jon3.trimo-adataavel [ER]

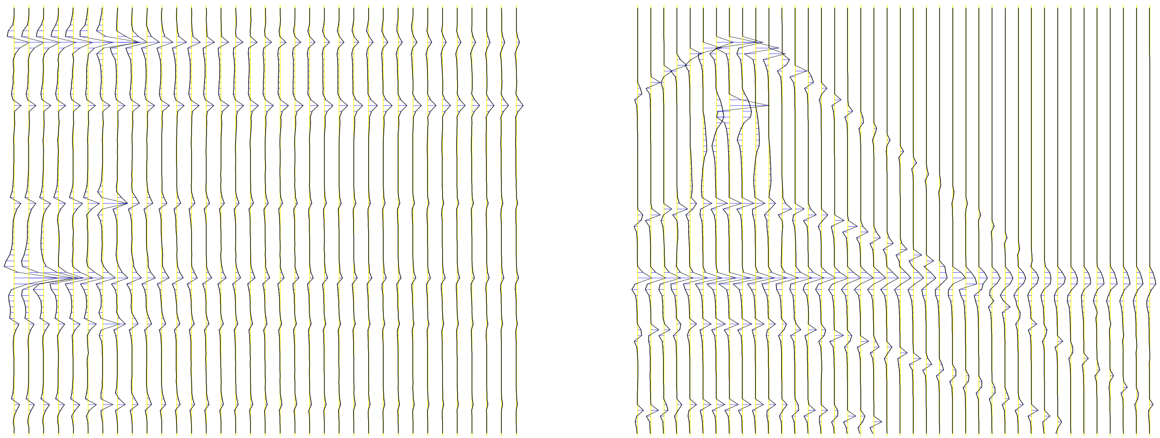

Figure 19: Like Figure 16 omitting pseudounitary scaling. `psun=0.` Right is synthetic data and left the analysis of it which is badly smeared. jon3.trimo-velsmear [ER]

```
# veltran --- velocity transform with anti-aliasing and sqrt(-i omega)
#
subroutine veltran(conj,add,psun,s02,anti,t0,dt,x0,dx,s0,ds,nt,nx,ns,model,data)
integer it,ix,is,  conj,add,psun,                          nt,nx,ns
real x, s, wt,              s02,anti,t0,dt,x0,dx,s0,ds,model(nt,ns),data(nt,nx)
temporary real slow(nt), half(nt,nx)
call null(              half,nt*nx)
call conjnull(     conj,add,                           model,nt*ns, data,nt*nx)
if( conj != 0)  do ix = 1, nx
                        call halfdifa( conj,   0, nt, half(1, ix), data(1, ix) )
do is= 1, ns {  s = s0 + (is-1) * ds;   do it=1,nt { slow(it) = s}
do ix= 1, nx {  x = x0 + (ix-1) * dx
        if     ( psun == 2 ) {  wt =       abs( s * x)  }       # vel tran
        else if( psun == 1 ) {  wt = sqrt( abs( s * x)) }       # pseudounitary
        else                 {  wt = 1.                 }       # modeling
        call trimo( conj, 1, t0,dt,dx, x, nt,slow, s02,
                        wt , anti,  model(1,is),  half(1,ix))
        }}
if( conj == 0)  do ix = 1, nx
                        call halfdifa( conj, add, nt, half(1, ix), data(1, ix) )
return; end
```

Fourier interpolation

Since these topics are near the heart of production seismology, considerable attention is often given to numerical analysis. Rectangle and triangle functions are instances of low-pass filters. The "ideal" low-pass filter is the sinc function. It is constant in the passband (so it disturbs nothing) and zero in the reject band (so it rejects perfectly). Given the slope dt/dx of the moveout curve, the cutoff frequency ω of the sinc function should be chosen in relation to the spatial Nyquist frequency k so that $dt/dx = k/\omega$. Dave Hale (electronic mail) says:

> The "production" approach to anti-aliasing Kirchhoff migration is

```
for each input trace {
        apply a suite of high-cut filters to input trace
        for each output trace {
                for each depth {
                        determine slope of migration operator;
                        use slope of operator to determine
                        which filtered version of input to use;
                        map (interpolate) input to output
                }
        }
}
```

> Because each input trace gets mapped to lots of output traces, the overhead of the initial high-cut filtering is negligible. Furthermore, most folks

> do this in the frequency domain and take the opportunity to supersample the trace (by padding with zeros in the frequency domain) so that the mapping in the innermost loop can be done with linear or even nearest neighbor interpolation. Obviously, this approach implies "semicircle superposition" instead of "hyperbola summation".

Hale might have the best method, but an argument against sinc interpolations is that the sharpness of the cutoff frequency loses relevance in view of arbitrariness of the three parameters that determine it, `antialias`, s_0, and regional dip parameter. An alternate approach is to interpolate the data first with a Fourier method (for accuracy) and then use the triangle method given here (for speed and simplicity).

EXTENSIONS AND FUTURE WORK

Velocity derivative

Changing one velocity to another moves the location of a triangle on the time axis. The derivative of a triangle function is a box doublet. Thus to calculate a derivative with respect to velocity, it seems the main programming change to `trimo()` is to change ramp functions to step functions, i.e. to replace the call to `doubint()` by a call to `causint()`. The amplitude also changes by the scale dt/dv. It would be fun to continue here.

Optimizing migration

I notice 2-3 dozen computational operations for each triangle wavelet. While these many operations are of no concern in demonstrations and experimental work, for production work it is desirable to eliminate them if possible. The midpoint axis is the key. With velocity transformations, the inner loop can then be a sum over midpoint. With migration, this speedup is only possible if the trace spacing and velocity are constant over the x-axis. Subroutine `aamig()` implements this convolution over midpoint outside the loop on t but the midpoint loop can be brought inside the inner loops in the `spotw()` subroutine. Then the outer overhead would be of little consequence and the migration should speed up by a factor of 2-3 dozen.

It would be fun to have movies of migration versus the antialiasing parameter to see if there is a practical importance to the distinction between `antialias=1` and `antialias=2`.

Given the ample computer power that we enjoy nowadays, it would also be interesting to adapt the program to variable trace spacing and missing data. Notice that the basic concept of wavelet overlap from one trace to the next is easily adaptable to drastically variable spacing or missing traces.

Most seismic data is recorded with low frequencies strongly filtered, however, I have seen recorded data with substantial zero frequency energy. This seems to be from pressure sensors that stay at a constant depth from the bottom as surface water waves roll by effectively at zero velocity. I conjecture the double integral of such data would be so strongly dominated by the low frequency that a precision problem would arise. One method of solving this problem is to prefilter the data. Another would be to use tapered rectangles and tapered triangles starting from something like $(1-(.99Z)^n)/(1-.99Z)$.

REFERENCES

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

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

Kjartansson, E., 1979, Attenuation of seismic waves in rocks and applications in energy exploration: SEP–**23**.