Chapter 1

Antialiased hyperbolas

A most universal practical problem in geophysics is that we never have enough recordings. This leads to the danger of spatial aliasing of data. There is no universal cure for this problem (although there are some specialized techniques of limited validity). A related, but less severe problem arises with Kirchhoff type operators. This problem is called "operator-aliasing". It has a cure, which we investigate in this chapter.

Fourier and finite-difference methods of migration are immune to the operator-aliasing malady suffered by hyperbola summation (Kirchhoff) migration. Here we will see a way to overcome the operator-aliasing malady shared by all Kirchhoff-like operators and bring them up to the quality of phase-shift methods. The antialiasing methods we develop here also lead to natural ways of handling irregularly sampled data.

We like to imagine that our data is a continuum and that our sums are like integrals. For practical purposes, our data is adequately sampled in time, but often it is not adequately sampled in space. Sometimes the data is sampled adequately in space, but our operators, such as hyperbolic integrations, are not adequately represented by a summation ranging over the x-coordinate picking a value at the nearest time $t(x)$. First we could improve nearest-neighbor interpolation by using linear interpolation. Linear interpolation, however, is not enough. Trouble arises when we jump from one trace to the next, $x \to x + \Delta x$, and find that $t(x)$ jumps more than a single Δt. Then we need a bigger "footprint" on the time axis than the two neighboring points used by linear interpolation. See Figure 1.1. 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 1.1.

1.0.1 Amplitude pitfall

In geophysics we often discuss signal amplitude versus offset distance. It sounds easy, but there are some serious pitfalls. Such pitfalls are one reason why mathematicians often use

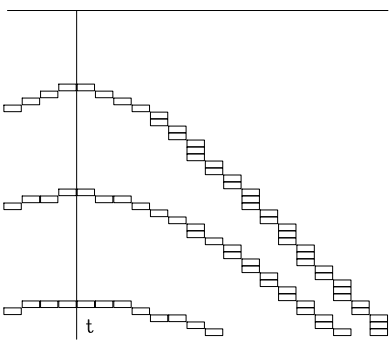

Figure 1.1: To integrate along hyperbolas without aliasing, you should include (at least) the points shown. trimo-nmotraj [ER]

nonintuitive weasel words. The best way for you to appreciate the pitfall is for me to push you into the pit.

Suppose we are writing a seismogram modeling program and we wish to model an impulsive plane wave of unit amplitude. Say the signal seen at x is $(\cdots,0,0,1,0,0,\cdots)$. At $x+\Delta x$ the plane wave is shifted in time so that the impulse lies half way between two points, say it is $(\cdots,0,0,a,a,0,0,\cdots)$. The question is, "what should be the value of a?" There are three contradictory points of view:

1. The amplitude a should be 1 so that the peak amplitude is constant with x.
2. The amplitude a should be $1/\sqrt{2}$ so that both seismic signals have the same energy.
3. The amplitude a should be $1/2$ so that both seismic signals have the same area.

Make your choice before reading further.

What is important in the signal is not the high frequencies especially those near the Nyquist. We cannot model the continuous universe with sampled data at frequencies above the Nyquist frequency nor can we do it well or easily at frequencies approaching the Nyquist. For example, at half the Nyquist frequency, a derivative is quite different from a finite difference. What we must try to handle correctly is the low frequencies (the adequately sampled signals). The above three points of view are contradictory at low frequencies. Examine only the zero frequency of each. Sum over time. Only by choosing equal areas $a = 1/2$ do the two signals have equal strength. The appropriate definition of amplitude on a sampled representation of the continuum is the *area per unit time.* Think of each signal value as representing the integral of the continuous amplitude from $t - \Delta t/2$ to $t + \Delta t/2$. Amplitude defined in this way cannot be confounded by functions oscillating between the sampled values.

Consider the task of abandoning data: We must reduce data sampled at a two millisecond rate to data sampled at a four millisecond rate. A method with aliasing is to abandon alternate points. A method with reasonably effective antialiasing is to convolve with the rectangle $(1,1)$ (add two neighboring values) and then abandon alternate values. Without the antialiasing, you could lose the impulse on the $(\cdots,0,0,1,0,0,\cdots)$ signal. A method with no aliasing is to multiply in the frequency domain by a rectangle function between $\pm$ Nyquist/2 (equivalent to convolving with a sinc function) and then abandoning alternate data points. This method perfectly preserves all frequencies up to the new Nyquist frequency (which is half the original).

1.1 MIMICING FIELD ARRAY ANTIALIASING

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 in a two-dimensional array of 13×13 recorders with 24 geophones added at each recorder. 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 1.2 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

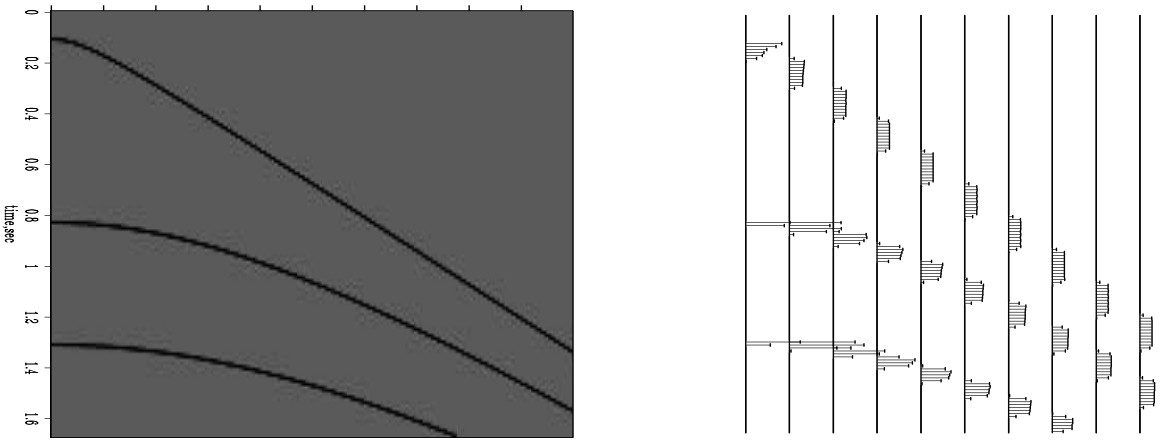

Figure 1.2: Quasicontinuous field (left) added in groups (right). trimo-oversamp [ER]

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 within the group. 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 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 sparse data we would recover the original characteristic that all these signals have the same amplitude.

Figure 1.3 shows a quasisinusoidal signal and compares subsampling to antialiasing via field arrays as in Figure 1.2. We see that aliased energy has been surpressed but not removed. Let us see how we can understand the result and how we could do better (but we won't). Suppose that the 500 channels had been individually recorded. The right panel in Figure 1.3 was computed simply by adding in groups of 25. A lengthier explanation of the calculation is that

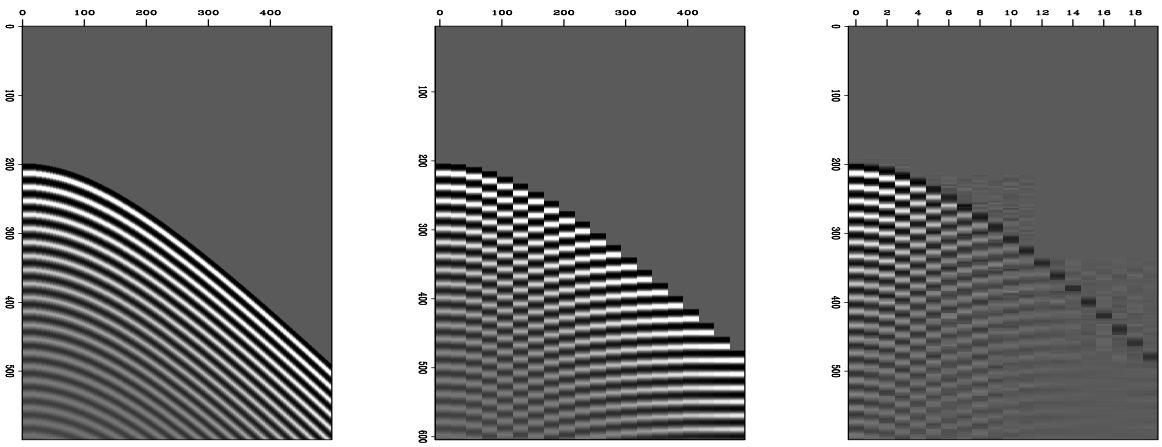

Figure 1.3: 500 channels (left), subsampled to 20 (middle), added in groups of 25(right). trimo-subsampvrsaa [ER]

the 500 channels were convolved along the horizontal x-axis with a 25 point long rectangle function. Then the 500 channel output was subsampled to 20 channels. This lengthier calculation gives the same result but has a simple Fourier explanation: Convolving with a rectangle function of x is the Fourier equivalent to multiplying by a sinc function $\sin(k_x \Delta x)/(k_x \Delta x)$ in the Fourier domain. We have convolved with a rectangle in the physical domain which amounts to multiplication by a sinc function in the Fourier domain. Theoretically we would prefer to have done it the other way around, convolved with a sinc in the physical domain, equivalently multiplying with a rectangle in the Fourier domain. The Fourier rectangle would drop to zero at half Nyquist and thus subsampling would not fold back any energy from above the half Nyquist to below it. Although Figure 1.3 shows that the aliased information is strongly suppressed, you can see that it has not been eliminated. Had we instead convolved with a sinc on the x-axis, the Fourier function would have been a rectangle. You would see the wavefronts in Figure 1.3 (right panel) vanishing where the dip reached a critical threshhold instead of seeing the wavefronts gradually tapering off and weak aliased events still being visible.

1.1.1 Adjoint of data acquisition

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 1.4 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.

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.

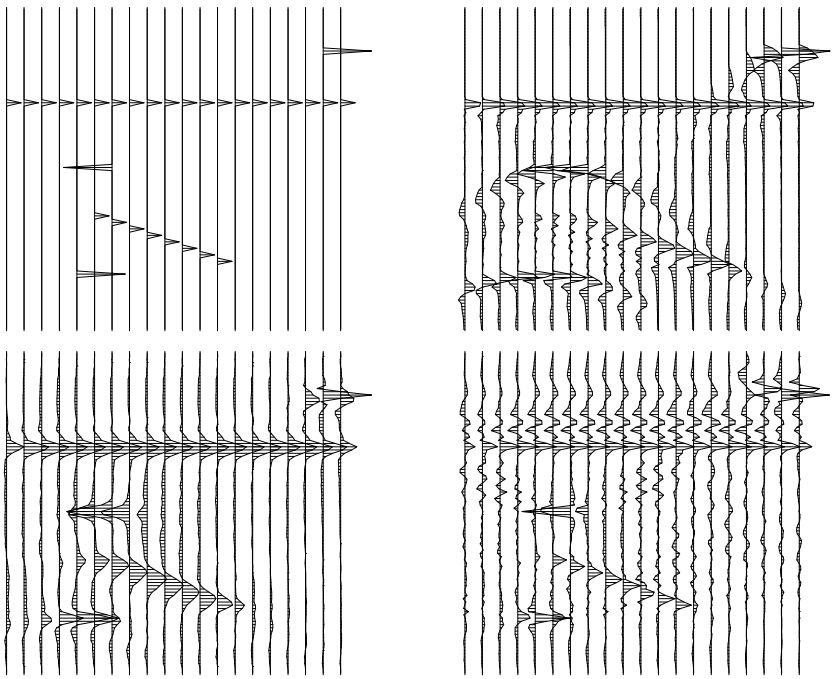

Figure 1.4: Top left is a synthetic image. Top right is synthetic data from the synthetic image. Bottom are migrations of the data with and without antialiasing. trimo-migalias [ER]

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.

A simple program designed for antialiasing gave the result in Figure 1.5. A zero-offset signal is input to adjoint 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 1.5 is a fussy and 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+n+1) - qq(it)` .

Figure 1.5 is not fully consistent with Figure 1.1. In Figure 1.5 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

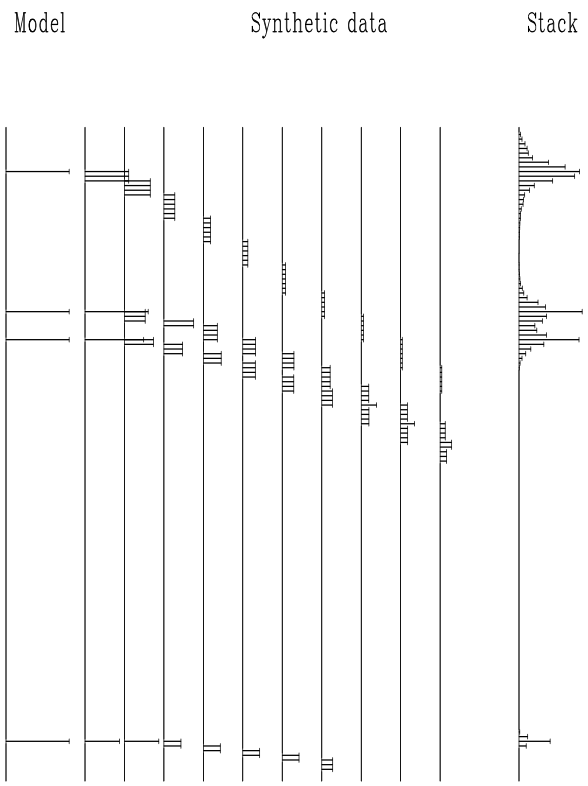

Figure 1.5: Rectangle smoothing during NMO and stacking. Notice that the end of one rectangle exactly coincides with the beginning of the rectangle at next larger offset. Thus, rectangle width increases with offset and decreases with time. (`antialias=1.`) trimo-boxmo1 [ER]

hyperbola would have *zero length* which is wholly unacceptable. Should we write a code to match Figure 1.1? This would be better, but far from perfect. Notice in Figure 1.1 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.

1.1.2 NMO and stack with a rectangle footprint

A subroutine for causal summation is subroutine `causint()` on page ??. Recall that the adjoint of causal integration is anticausal integration. For each reflector, data modeling proceeds by throwing out two pulses of opposite polarity. Then causal summation produces a rectangle between the pulses (sometimes called "box car"). Since the last step in the modeling operator is causal summation, the first step in the adjoint operator (which does NMO) is anticausal summation. 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 traveltime 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 boxmo( adj, add, t0,dt, dx, x, nt,slow,  antialias, zz,     tt   )
integer it,iz,itp,adj, 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 adjnull( adj, add,         zz,nt,  tt,nt)
if( adj != 0)                   call causint(     1,   0, nt, ss, tt)
do iz= 2, 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) * z/t / (itp - it)
        if ( itp < nt   ) {
                if( adj == 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( adj == 0)                   call causint(     0,  add, nt, ss, tt)
return; end

subroutine boxstack( adj,add,slow,antialias, t0,dt,x0,dx,nt,nx, stack, gather)
integer adj, add, ix, nx, nt
real x,                slow(nt),antialias, t0,dt,x0,dx, stack(nt), gather(nt,nx)
call adjnull( adj, add, stack, nt, gather, nt*nx)
do ix= 1, nx {  x =  x0 + dx * (ix-1)
        call boxmo( adj,1, t0,dt,dx,x,nt, slow,antialias, stack, gather(1,ix))
        }
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 $1/\sqrt{t}$; 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 1.6 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.

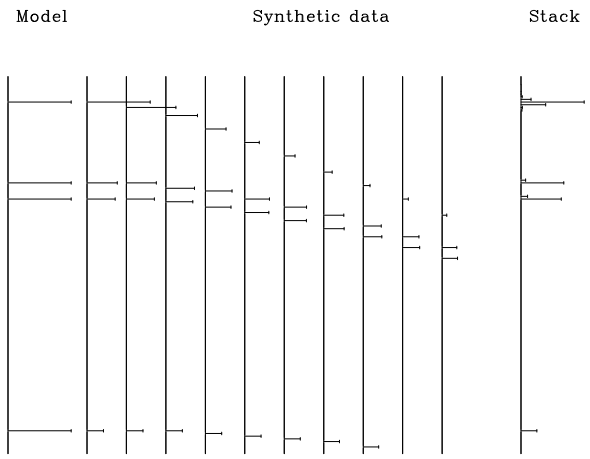

Figure 1.6: Rectangles shortened to one point duration. (`antialias=0.`) trimo-boxmo0 [ER]

1.1.3 Coding a triangle footprint

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 that 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()` on the current page.

```
# Double integration, first causal, then anticausal.
#
subroutine doubint( adj, add, n,              pp   , qq  )
integer             adj, add, n;       real  pp(n), qq(n)
temporary real tt(n)
call adjnull(       adj, add,                 pp,n,  qq,n)
if( adj == 0 ) {         call causint( 0,   0, n,pp,  tt  )
                         call causint( 1, add, n,qq,  tt  )
               }
else {                   call causint( 1,   0, n,tt,  qq  )
                         call causint( 0, add, n,tt,  pp  )
               }
return; end
```

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. Subroutine `lint1()` on page ?? does linear interpolation, but here we need weighted results as provided by `spotw()` on the current page.

```
#  Scaled linear interpolation.
#
subroutine spotw( adj, add, scale, nt,t0,dt, t, val,   vec   )
integer it,itc,   adj, add,        nt
```

```
real tc, fraction,          scale,    t0,dt, t, val,   vec(nt)
call adjnull(     adj, add,                      val,1, vec,nt)
tc = .5+ (t-t0) / dt;   itc = tc;        it = 1 + itc;  fraction = tc - itc
if( 1 <= it  &&  it < nt) {
        if( adj == 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 1.7 and 1.8.

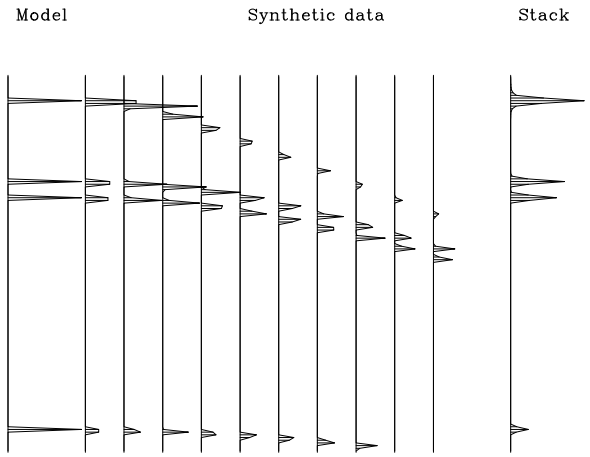

Figure 1.7: Triangle wavelets, accurately positioned, but aliased (`antialias=0.`) trimo-trimo0 [ER]

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

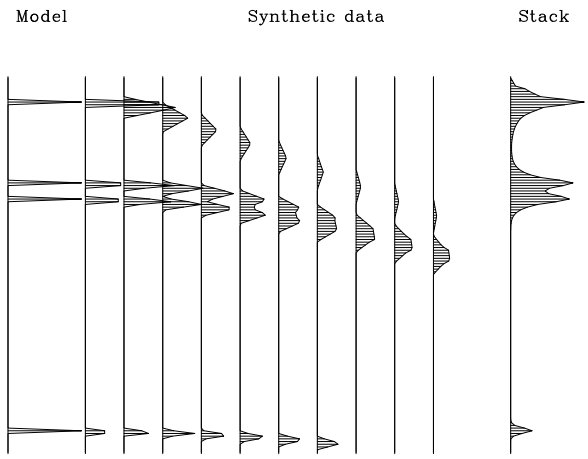

Figure 1.8: Antialiased triangle wavelets. (`antialias=1.`) Where ever triangle duration is more than about three points, the end of one triangle marks the apex of the next. trimo-trimo1 [ER]

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

From the stack reconstruction of the model in Figure 1.8 we see the reconstruction is more blured with antialiasing than it was without in Figure 1.7. The benefit of antialiasing will become clear next in more complicated examples where events cross.

1.2 MIGRATION WITH ANTIALIASING

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

```
# anti-aliased kirchhoff migration (adj=1) and modeling (adj=0)
```

```
#
subroutine aamig( adj,add, slow,antialias,t0,dt, dx, nt,nx, image, data  )
integer adj, add, ix, nx, nt, iy
real    h,            slow(nt),antialias,t0,dt, dx, image(nt,nx), data(nt,nx)
call adjnull(    adj, add,                          image,nt*nx,  data,nt*nx)
do ix= 1, nx {
do iy= 1, nx {
        h  = dx * (iy - ix)
        call trimo( adj, 1, t0,dt,dx, h, nt,slow, 0., 1., antialias, _
                                                        image(1,iy), data(1,ix))
        }}
return; end
```

Figure 1.9 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. The velocity is taken constant. Figure 1.10 shows

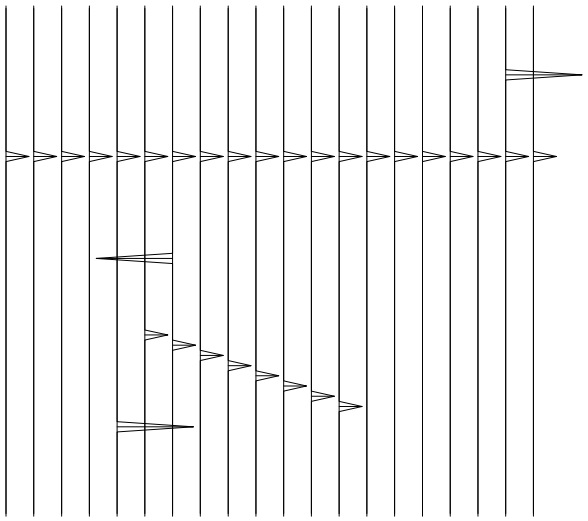

Figure 1.9: Model image for migration study. trimo-aamod [ER]

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 1.11 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.

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 1.11, 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 1.11 has a waveform that resembles a damped step function which is related to the Hankel tail we studied in chapter **??** where subroutine `halfdifa()` on

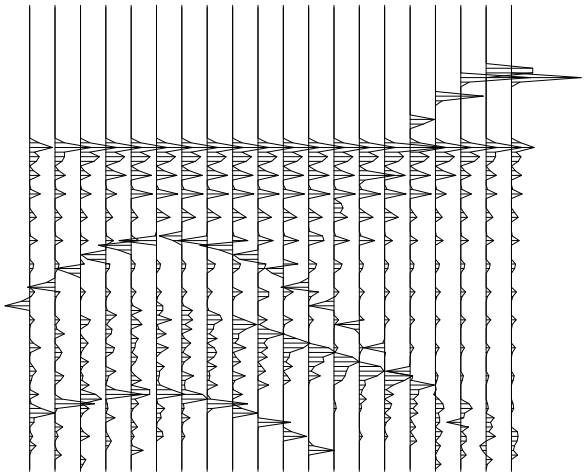

Figure 1.10: Synthetic data without regard for aliasing. Made from model image with `aamig()` taking `antialias=0.` trimo-aad0 [ER]

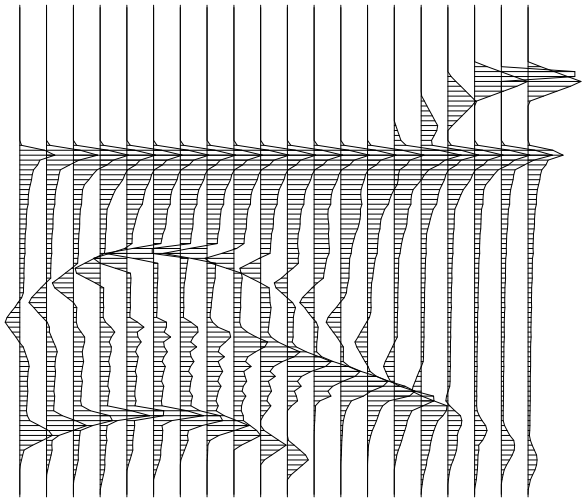

Figure 1.11: Synthetic data accounting for aliasing. Made from model image with `aamig()` taking `antialias=1.` trimo-aad1 [ER]

page ?? was introduced to provide the filter required to convert the waveform on the horizontal layer in Figure 1.11 back to an impulse. This was done in Figure 1.12. You can see the final flat-layer waveform is roughly the zero-phase shape we started with. Figure 1.13 shows my

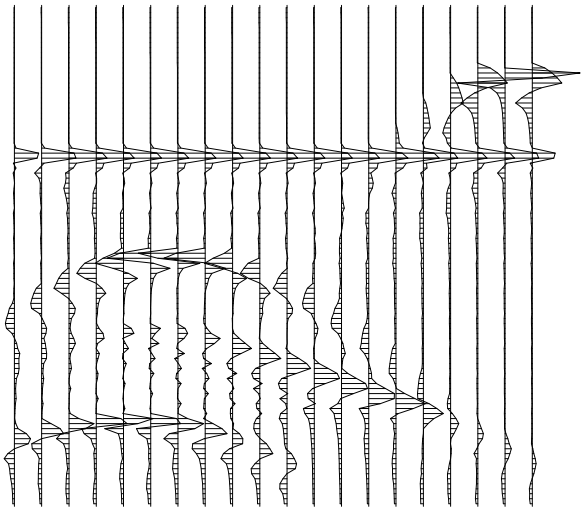

Figure 1.12: Best synthetic data. Made from model image using `aamig()` with `antialias=1` followed by a causal half-order time derivative. Lowpass temporal filtering would make wavelets more independent of location on a hyperbola. trimo-aad1h [ER]

best migration of my best synthetic data. All the features of the original model are apparent. 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 1.14 shows what happens when antialiasing is ignored in migration. Notice

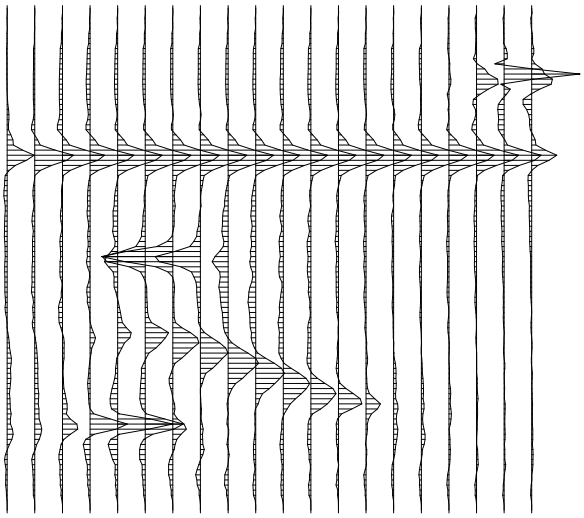

Figure 1.13: Best migration of best synthetic data. Uses `aamig()` with `antialias=2` followed by an anticausal half-order time derivative. trimo-aamig2 [ER]

many false layers above the given horizontal layer. Notice semicircles above the impulses. Notice apparent noise everywhere. But notice also that the dipping bed is sharper than the antialiased result in Figure 1.13.

1.2.1 Use of the 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 is where the

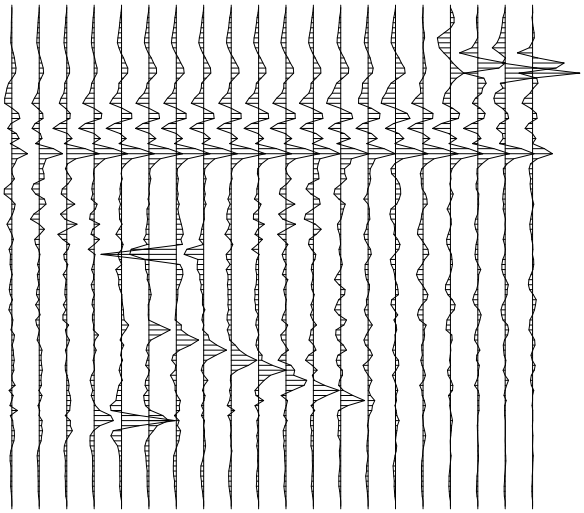

Figure 1.14: Migration of best synthetic data without regard for aliasing. Uses `aamig()` with `antialias=0.` (and an anticausal half-order time derivative) trimo-aamig0 [ER]

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 1.15 suggests that such hyperbolas require an antialias parameter greater than unity, say `antialias=2.`

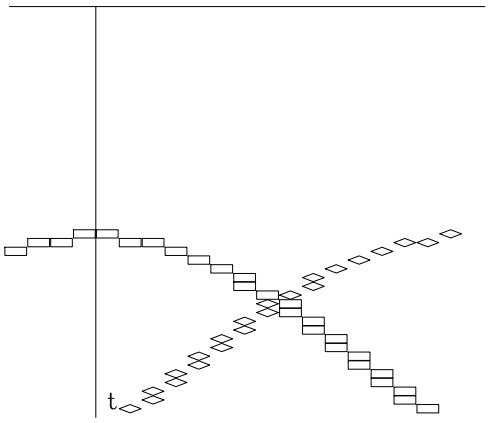

Figure 1.15: Crossing hyperbolas that do not touch. Thus the points shown are not enough to prevent spatial aliasing a line integral along one trajectory of signal on the other. trimo-croshyp [ER]

1.2.2 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.

1.3 ANTIALIASED OPERATIONS ON A 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.

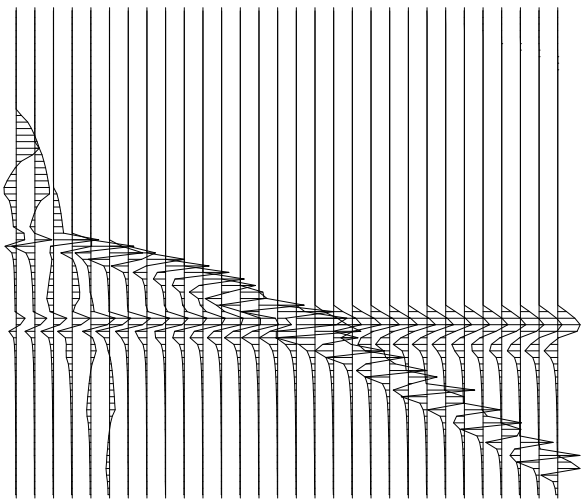

Figure 1.16: The air wave and fast wave are broadened increasingly with offset, but the water wave does not. This broadening enables crossing events to maintain their orthogonality. trimo-aacdp [ER]

1.3.1 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 adjoint 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 1.17.

Figure 1.17 shows that on a second pass, the velocity spectrum of the slow wave is much

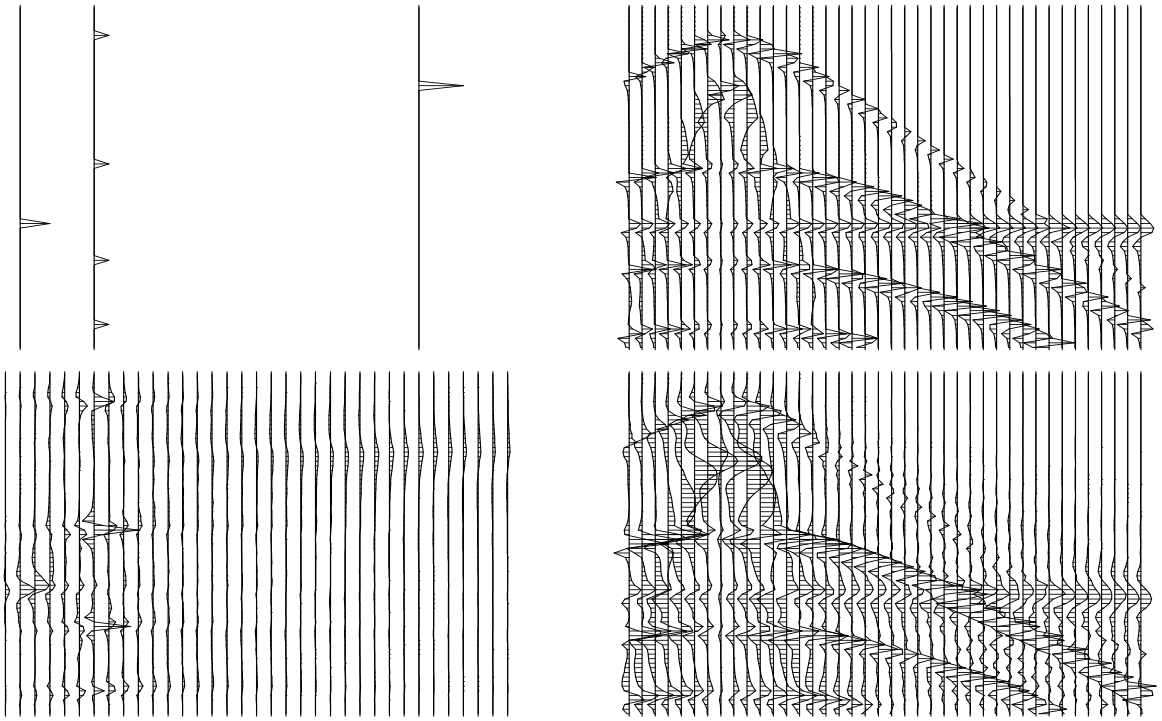

Figure 1.17: Top left: Slowness model. Top right: Data derived from it using the pseudounitary scale factor. Bottom left: the velocity spectrum of top right. Bottom right: data made from velocity spectrum. trimo-aavel1 [ER]

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 hyperbolas curving down.

Figure 1.18 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 1.18 or should we use an antialiased operator as that in Figure 1.17? Figure 1.19 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

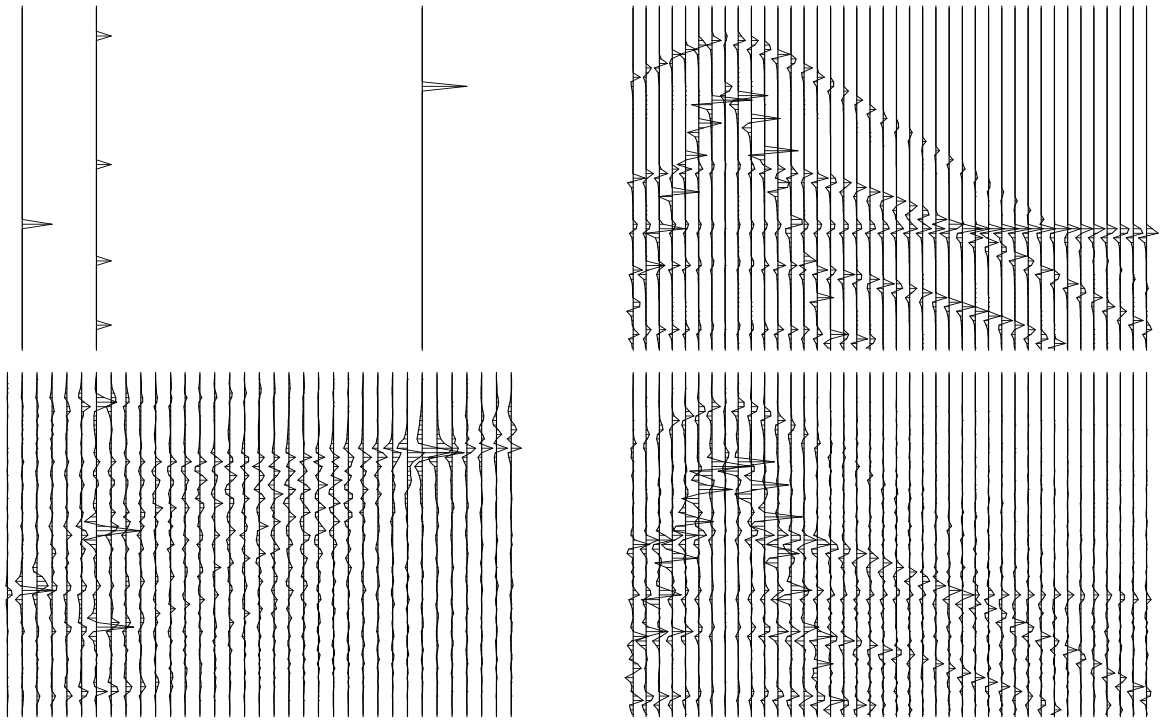

Figure 1.18: Like Figure 1.17 but with `antialias=0`. This synthetic data presumes no receiver groups in the field recording. trimo-aavel0 [ER]

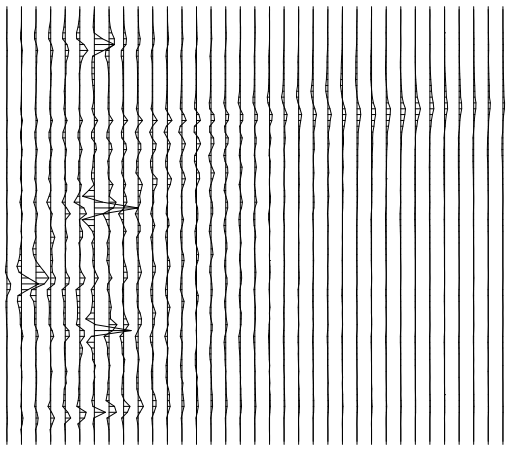

Figure 1.19: Aliased data analyzed with antialiased operator. Compare to the lower left of Figure 1.18. trimo-adataavel [ER]

velocity analysis. Failing to do so will give a result like in figure 1.20. 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 1.17 is that points in velocity space are orthogonal components like Fourier components whereas for nonunitary transforms like with Figure 1.20 the different points in velocity space are not orthogonal components.

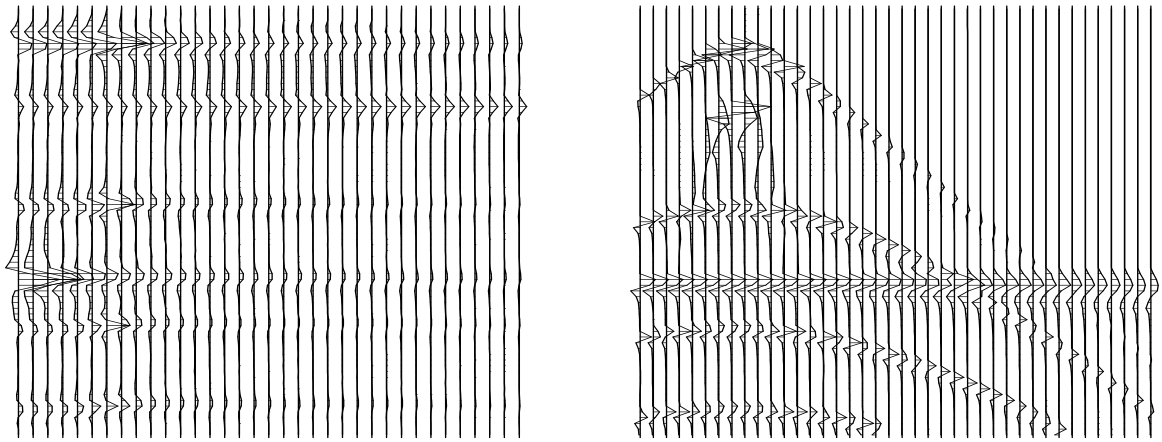

Figure 1.20: Like Figure 1.17 omitting pseudounitary scaling. `psun=0`. Right is synthetic data and left the analysis of it which is badly smeared. trimo-velsmear [ER]

Subroutine `veltran()` does the work.

```
# veltran --- velocity transform with anti-aliasing and sqrt(-i omega)
#
subroutine veltran(adj,add,psun,s02,anti,t0,dt,x0,dx,s0,ds,nt,nx,ns,model,data)
integer it,ix,is,  adj,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 adjnull(      adj,add,                                model,nt*ns, data,nt*nx)
if( adj != 0)   do ix = 1, nx
                        call halfdifa( adj,   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( adj, 1, t0,dt,dx, x, nt,slow, s02, _
                        wt , anti,  model(1,is),  half(1,ix))
        }}
if( adj == 0)   do ix = 1, nx
                        call halfdifa( adj, add, nt, half(1, ix), data(1, ix) )
return; end
```

EXERCISES:

1 What circumstances would suggest that the linear interpolation in subroutine `trimo()` on page 10 be replaced by nearest-neighbor interpolation?

2 Show how to adapt the programs of this chapter to variable trace spacing and missing data. Hint: Split `trimo()` into two parts, the first determining the location of the neighboring traces and the second using that information.

Index

aamig subroutine, 10
alias, 1
antialias, 13
    migration, 10
    stack, 6, 8
    velocity analysis, 15

boxmo subroutine, 6

doubint subroutine, 8

field array, 3

spotw subroutine, 8
stack, antialias, 6
subroutine
    `aamig`, antialias migration, 10
    `boxmo`, box footprint, 6
    `doubint`, double integration, 8
    `spotw`, weighted linear interp., 8
    `trimo`, triangle footprint, 10
    `tristack`, stack with triangle footprint, 9
    `veltran`, antialiased velocity transform, 18

triangle footprint, 8
trimo subroutine, 10
tristack subroutine, 9

veltran subroutine, 18