BASIC EARTH IMAGING (Version 2.4)

Jon F. Claerbout with James L. Black

Directory

Last Revision Date: November 17, 2005

Table of Contents

Themes

The main theme of this book is to take a good quality reflection seismic data set from the Gulf of Mexico and guide you through the basic geophysical data processing steps from raw data to the best-quality final image. Secondary themes are to introduce you (1) to cleaned up but real working Fortran code that does the job, (2) to the concept of “adjoint operator”, and (3) to the notion of electronic document.

What it does, what it means, and how it works

A central theme of this book is to merge the abstract with the concrete by linking mathematics to runnable computer codes. The codes are in a consistent style using nomenclature that resembles the accompanying mathematics so the two illuminate each other. The code shown is exactly that used to generate the illustrations. There is little or no mathematics or code that is not carried through with examples using both synthetic and real data. The code itself is in a dialect of Fortran more suitable for exposition than standard Fortran. (This "ratfor" dialect easily translates to standard Fortran). Some codes have been heavily tested while others have only been tested by the preparation of the illustrations.

Imaging with adjoint (conjugate-transpose) operators

A secondary theme of this book is to develop in the reader an understanding of a universal linkage beween forward modeling and data processing. Thus the codes here that incarnate linear operators are written in a style that also incarnates the adjoint (conjugate-transpose) operator thus enabling both modeling and data processing with the same code. This style of coding, besides being concise and avoiding

redundancy, ensures the consistency required for estimation by conjugate-gradient optimization as described in my other books.

Adjoint operators link the modeling activity to the model estimation activity. While this linkage is less sophisticated than formal estimation theory ("inversion"), it is robust, easily available, and does not put unrealistic demands on the data or imponderable demands on the interpreter.

Electronic document

A goal that we met with the 1992 CD-ROM version of this book was to give the user a full copy, not only of the book, but of all the software that built the book including not only the seismic data processing codes but also the word processing, the data, and the whole superstructure. Although we succeeded for a while having a book that ran on machines of all the major manufacturers, eventually we were beaten down by a host of incompatibilities. This struggle continues. With my colleagues, we are now working towards having books on the World Wide Web where you can grab parts of a book that generates illustrations and modify them to create your own illustrations.

Acknowledgements

I had the good fortune to be able to establish a summer 1992 collaboration with Jim Black of IBM in Dallas who, besides bringing fresh eyes to the whole undertaking, wrote the first version of chapter 8 on dip moveout, made significant contributions to the other chapters, and organized the raw data.

In this book, as in my previous (and later) books, I owe a great deal to the many students at the Stanford Exploration Project. The local computing environment from my previous book is still a benefit, and for this I thank Stew Levin, Dave Hale, and Richard Ottolini. In preparing this book I am specially indebted to Joe Dellinger for his development of the intermediate graphics language `vplot` that I used for all the figures. I am grateful to Kamal Al-Yahya for converting my thinking from the `troff` typesetting language to LaTeX. Bill Harlan offered helpful suggestions. Steve Cole adapted `vplot` to Postscript and X. Dave Nichols introduced our multivendor environment. Joel M. Schroeder and Matthias Schwab converted from cake to gmake. Bob Clapp expanded Ratfor for Fortran 90. Martin Karrenbach got us started with CD-ROMs. Sergey Fomel upgraded the Latex version to “2e” and he implemented the basic changes taking us from CD-ROM to the WWW, a process which continues to this day in year 2000.

Jon Claerbout
Stanford University
November 17, 2005

Chapter 1

Field recording geometry

The basic equipment for reflection seismic prospecting is a source for impulsive sound waves, a geophone (something like a microphone), and a multichannel wave-form display system. A survey line is defined along the earth's surface. It could

be the path for a ship, in which case the receiver is called a hydrophone. About every 25 meters the source is activated, and the echoes are recorded nearby. The sound source and receiver have almost no directional tuning capability because the frequencies that penetrate the earth have wavelengths longer than the ship. Consequently, echoes can arrive from several directions at the same time. It is the joint task of geophysicists and geologists to interpret the results. Geophysicists assume the quantitative, physical, and statistical tasks. Their main goals, and the goal to which this book is mainly directed, is to make good pictures of the earth's interior from the echoes.

1.1. RECORDING GEOMETRY

Along the horizontal x-axis we define two points, s, where the source (or shot or sender) is located, and g, where the geophone (or hydrophone or microphone) is located. Then, define the **midpoint** y between the shot and geophone, and define h to be half the horizontal **offset** between the shot and geophone:

$$y = \frac{g + s}{2} \tag{1.1}$$

$$h = \frac{g - s}{2} \tag{1.2}$$

The reason for using *half* the offset in the equations is to simplify and symmetrize many later equations. Offset is defined with $g - s$ rather than with $s - g$ so that positive offset means waves moving in the positive x direction. In the marine case, this means the ship is presumed to sail negatively along the x-axis. In reality the ship may go either way, and shot points may either increase or decrease as the survey proceeds. In some situations you can clarify matters by setting the field observer's shot-point numbers to negative values.

Data is defined experimentally in the space of (s, g). Equations (1.1) and (1.2) represent a change of coordinates to the space of (y, h). Midpoint-offset coordinates are especially useful for interpretation and data processing. Since the data is also a function of the travel time t, the full dataset lies in a volume. Because it is so difficult to make a satisfactory display of such a volume, what is customarily done is to display slices. The names of slices vary slightly from one company to the next. The following names seem to be well known and clearly understood:

$(y,\ h=0,\ t)$	zero-offset section
$(y,\ h=h_{\min},\ t)$	near-trace section
$(y,\ h=\text{const},\ t)$	constant-offset section
$(y,\ h=h_{\max},\ t)$	far-trace section
$(y=\text{const},\ h,\ t)$	common-midpoint gather
$(s=\text{const},\ g,\ t)$	field profile (or common-shot gather)
$(s,\ g=\text{const},\ t)$	common-geophone gather
$(s,\ g,\ t=\text{const})$	time slice
$(h,\ y,\ t=\text{const})$	time slice

A diagram of slice names is in Figure 1.1. Figure 1.2 shows three slices from the data volume. The first mode of display is "engineering drawing mode." The second mode of display is on the faces of a cube. But notice that although the data is displayed on the surface of a cube, the slices themselves are taken from the interior of the cube. The intersections of slices across one another are shown by dark lines.

A common-depth-point (CDP) gather is defined by the industry and by common usage to be the same thing as a common-midpoint (CMP) gather. But in this book a distinction will be made. A **CDP gather** is a **CMP gather** with its time axis

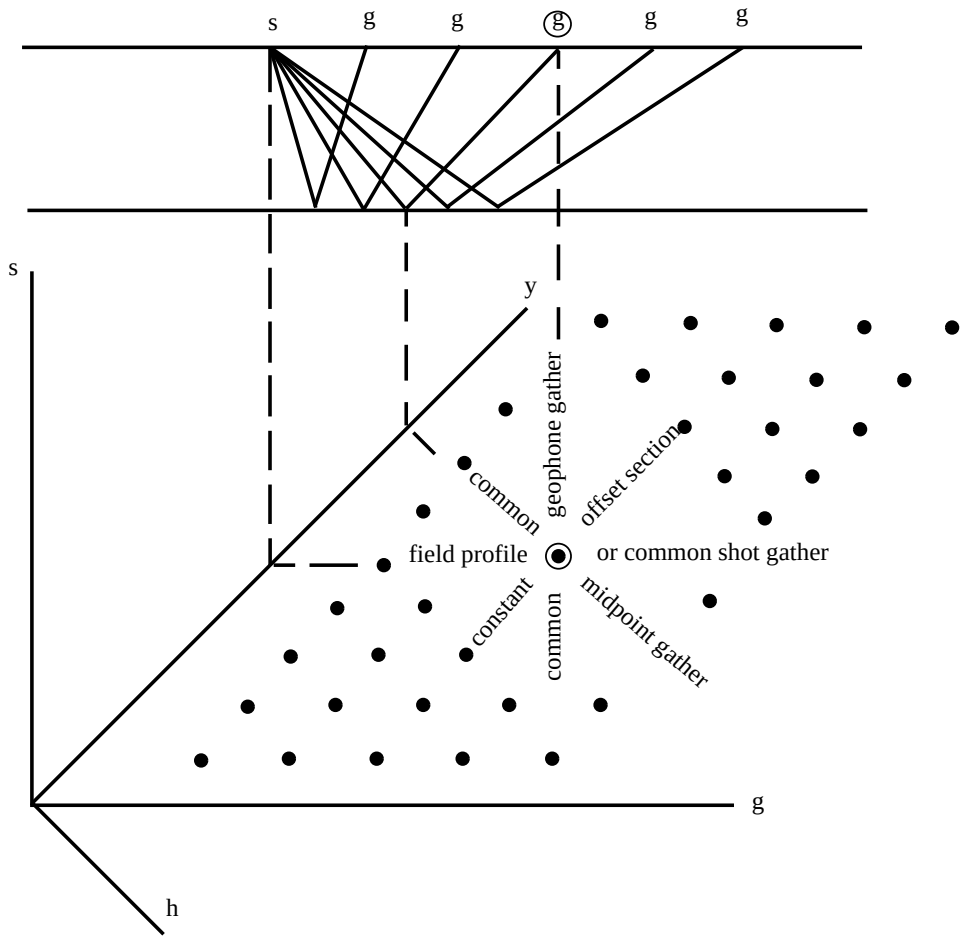


Figure 1.1: Top shows field recording of marine seismograms from a shot at location s to geophones at locations labeled g. There is a horizontal reflecting layer to aid interpretation. The lower diagram is called a **stacking diagram**. (It is *not* a perspective drawing). Each dot in this plane depicts a possible seismogram. Think of time running out from the plane. The center geophone above (circled) records the seismogram (circled dot) that may be found in various geophysical displays. Lines

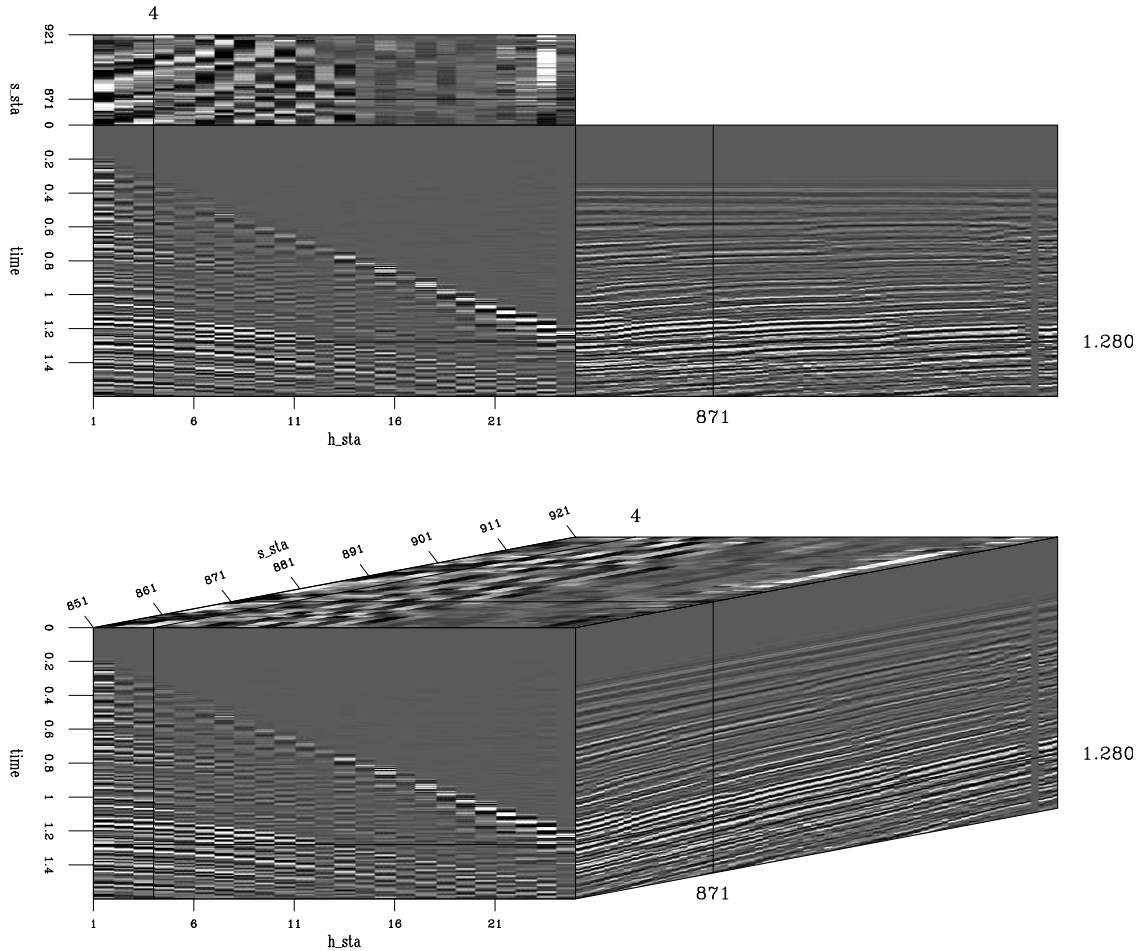


Figure 1.2: Slices from within a cube of data. Top: Slices displayed as a mechanical drawing. Bottom: Same slices shown on perspective of cube faces. fld-cube [ER]

stretched according to some velocity model, say,

$$(y = \text{const},\ h,\ \sqrt{t^2 - 4h^2/v^2}) \qquad \text{common-depth-point gather}$$

This offset-dependent stretching makes the time axis of the gather become more like a *depth* axis, thus providing the D in CDP. The stretching is called ***normal moveout*** *correction* (NMO). Notice that as the velocity goes to infinity, the amount of stretching goes to zero.

There are basically two ways to get two-dimensional information from three-dimensional information. The most obvious is to cut out the slices defined above. A second possibility is to remove a dimension by summing over it. In practice, the offset axis is the best candidate for summation. Each CDP gather is summed over offset. The resulting sum is a single trace. Such a trace can be constructed at each midpoint. The collection of such traces, a function of midpoint and time, is called a CDP stack. Roughly speaking, a CDP stack is like a zero-offset section, but it has a less noisy appearance.

The construction of a CDP stack requires that a numerical choice be made for the moveout-correction velocity. This choice is called the *stacking velocity.* The stacking velocity may be simply someone's guess of the earth's velocity. Or the

guess may be improved by stacking with some trial velocities to see which gives the strongest and least noisy CDP stack.

Figures 1.3 and 1.4 show typical marine and land **profile**s (common-shot gathers). The land data has geophones on both sides of the source. The arrangement shown is called an *uneven* ***split spread***. The energy source was a vibrator. The marine data happens to nicely illustrate two or three head waves. The marine energy source was an air gun. These field profiles were each recorded with about 120 geophones.

1.1.1. Fast ship versus slow ship

For marine seismic data, the spacing between shots Δs is a function of the speed of the ship and the time interval between shots. Naturally we like Δs small (which means more shots) but that means either the boat slows down, or one shot follows the next so soon that it covers up late arriving echos. The geophone spacing Δg is fixed when the marine **streamer** is designed. Modern streamers are designed for more powerful computers and they usually have smaller Δg. Much marine seismic data is recorded with $\Delta s = \Delta g$ and much is recorded with $\Delta s = \Delta g/2$.

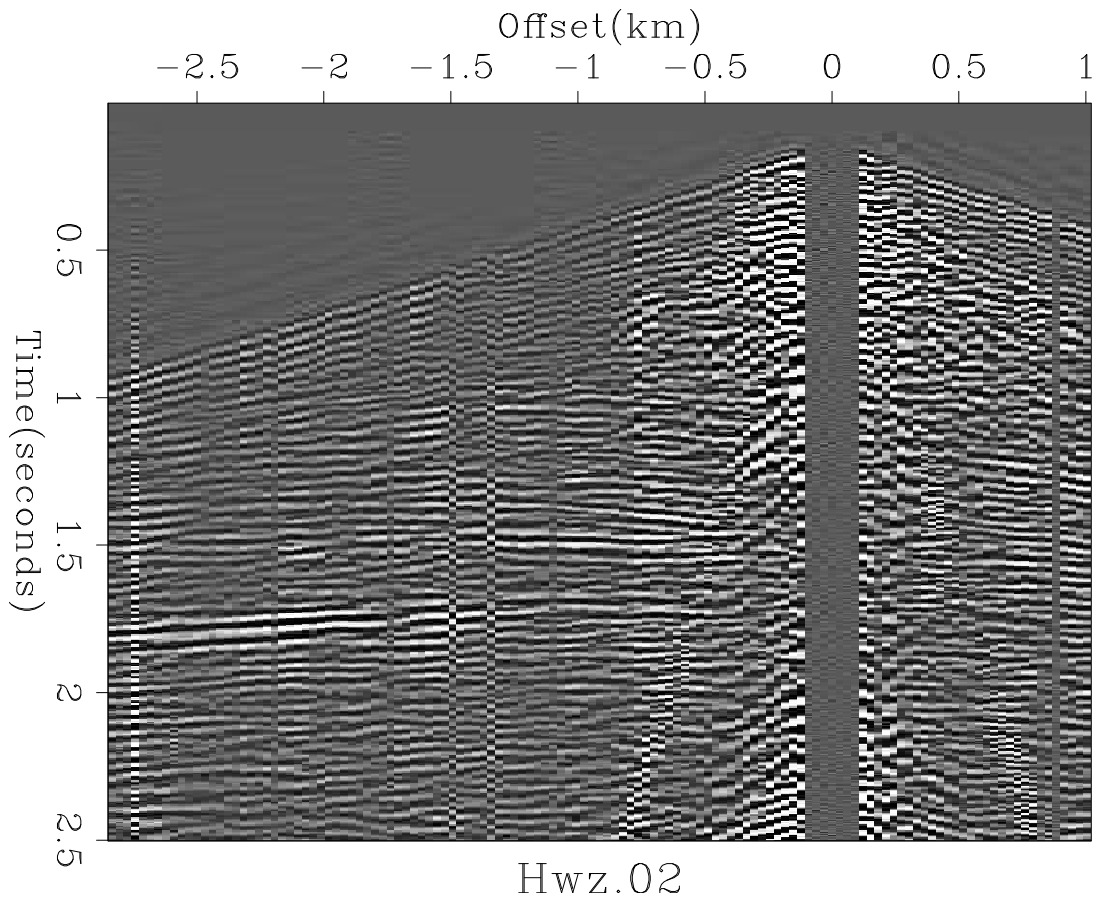


Figure 1.3: A seismic land profile. There is a gap where there are no receivers near the shot. You can see events of three different velocities. (Western Geophysical). fld-yc02 [ER]

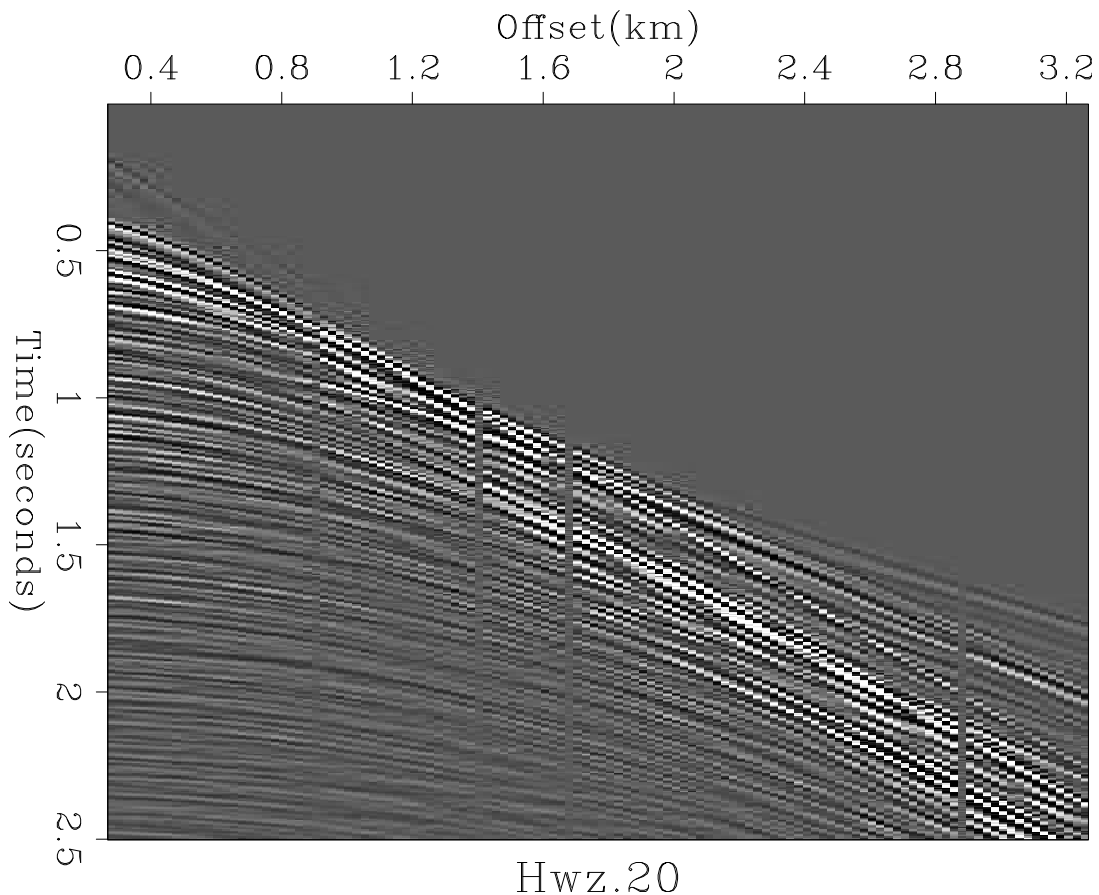


Figure 1.4: A marine profile off the Aleutian Islands. (Western Geophysical). fld-yc20 [ER]

There are unexpected differences in what happens in the processing. Figure 1.5 shows $\Delta s = \Delta g$, and Figure 1.6 shows $\Delta s = \Delta g/2$. When $\Delta s = \Delta g$ there are

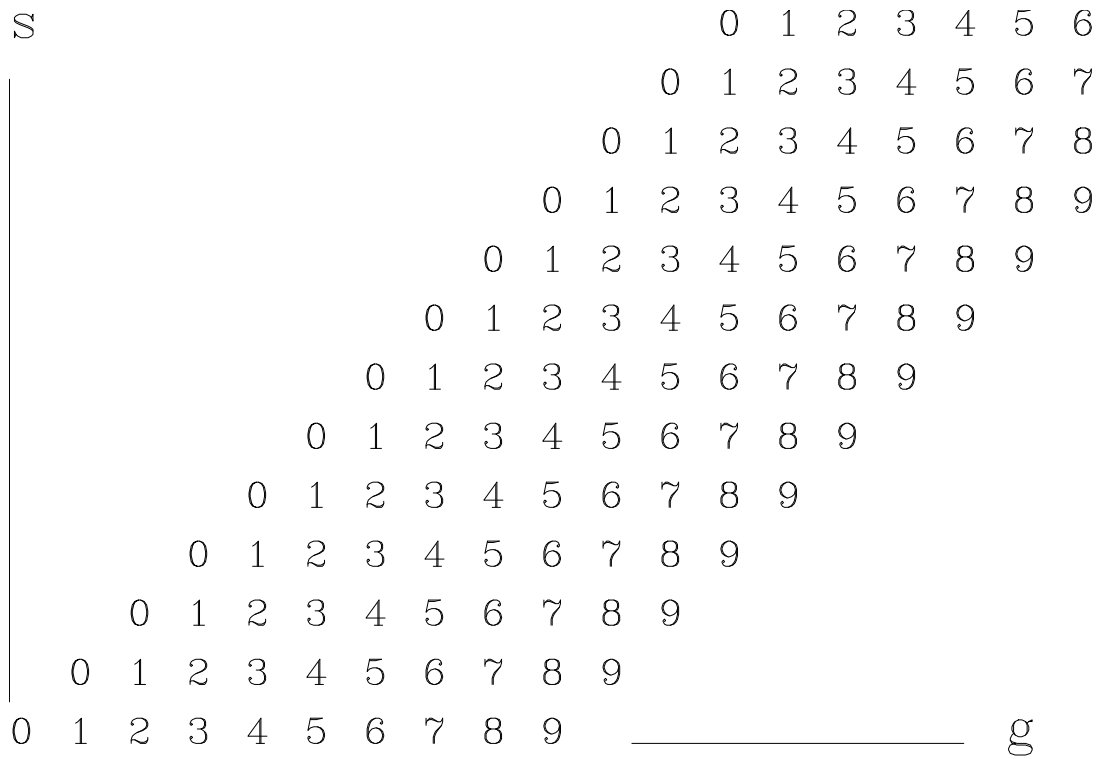


Figure 1.5: $\Delta g = \Delta s$. The zero-offset section lies under the zeros. Observe the common midpoint gathers. Notice that even numbered receivers have a different geometry than odd numbers. Thus there are two kinds of CMP gathers with different values of the **lead-in** x_0 = `x0` fld-geqs [ER]

some irritating complications that we do not have for $\Delta s = \Delta g/2$. When $\Delta s = \Delta g$, even-numbered traces have a different midpoint than odd-numbered traces. For a

common-midpoint analysis, the evens and odds require different processing. The words "**lead-in**" describe the distance (x_0 = `x0`) from the ship to the nearest trace. When $\Delta s = \Delta g$ the lead-in of a CMP gather depends on whether it is made from the even or the odd traces. In practice the lead-in is about $3\Delta s$. Theoretically we would prefer no lead in, but it is noisy near the ship, the tension on the cable pulls it out of the water near the ship, and the practical gains of a smaller lead-in are evidently not convincing.

1.2. TEXTURE

Gravity is a strong force for the stratification of rocks, and many places in the world rocks are laid down in horizontal beds. Yet even in the most ideal environment the bedding is not mirror smooth; it has some ***texture***. We begin with synthetic data that mimics the most ideal environment. Such an environment is almost certainly marine, where sedimentary deposition can be slow and uniform. The wave velocity will be taken to be constant, and all rays will reflect as from horizontally lying mirrors. Mathematically, *texture* is introduced by allowing the reflection coefficients of the beds to be laterally variable. The lateral variation is presumed to be a ran-

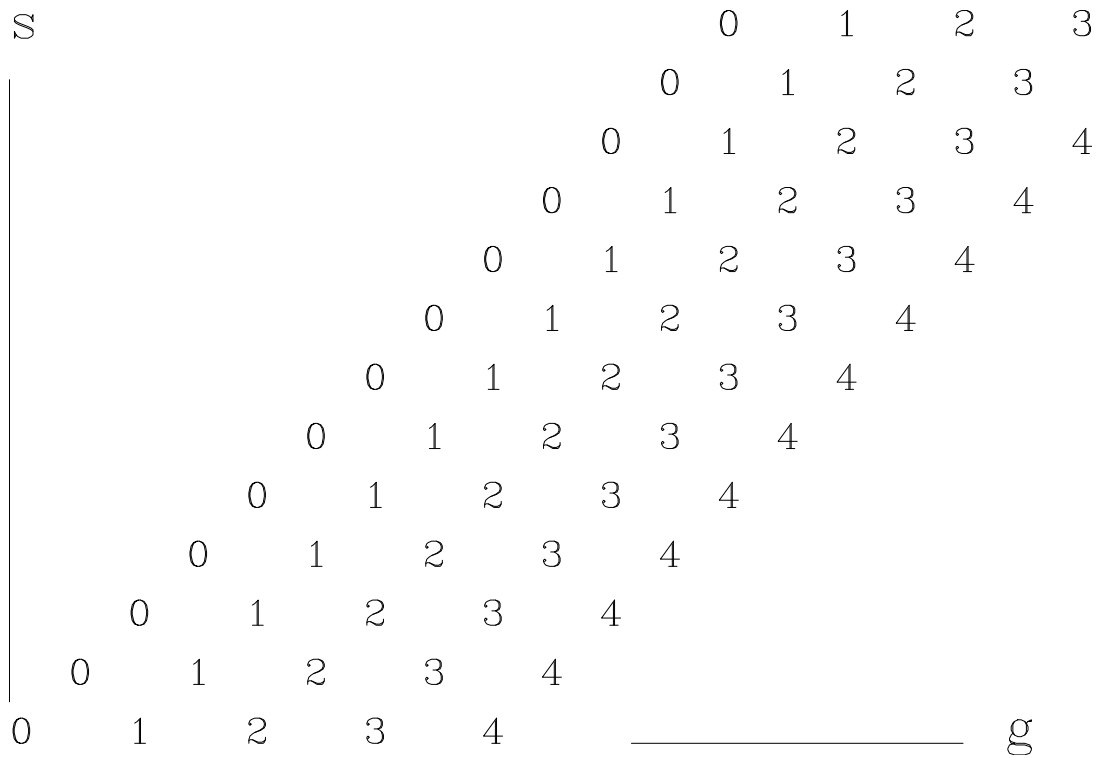


Figure 1.6: $\Delta g = 2\Delta s$. This is like Figure 1.5 with odd valued receivers omitted. Notice that each common-midpoint gather has the same geometry. fld-geq2s [ER]

dom function, though not necessarily with a white spectrum. Let us examine the appearance of the resulting field data.

1.2.1. Texture of horizontal bedding, marine data

Randomness is introduced into the earth with a random function of midpoint y and depth z. This randomness is impressed on some geological "layer cake" function of depth z. This is done in the first half of subroutine `synmarine()` /prog:synmarine.

synmarine The second half of subroutine `synmarine()` /prog:synmarine scans all shot and geophone locations and depths and finds the midpoint, and the reflection coefficient for that midpoint, and adds it into the data at the proper travel time.

There are two confusing aspects of subroutine `synmarine()` /prog:synmarine. First, refer to figure 1.1 and notice that since the ship drags the long cable containing the receivers, the ship must be moving to the left, so data is recorded for sequentially *decreasing* values of s. Second, to make a continuous **movie** from a small number of frames, it is necessary only to make the midpoint axis periodic, i.e. when a value of `iy` is computed beyond the end of the axis `ny`, then it must be moved back an integer multiple of `ny`.

```
subroutine synmarine ( data, nt,nh,ny, nz)
integer                      nt,nh,ny, nz,    it,ih,iy,is,iz, ns, iseed
          real          data( nt,nh,ny),       layer, rand01
temporary real                                refl(nz,ny), depth(nz)
iseed= 1992;    ns = ny
do iz= 1, nz {                                          # 0 < rand01() < 1
        depth( iz) =  nt * rand01(iseed)                # Reflector depth
        layer      =  2. * rand01(iseed) - 1.           # Reflector strength
        do iy= 1, ny {                                  # Impose texture on layer
                refl(iz,iy) = layer * (1. + rand01(iseed))
                }
        }
call null(              data, nt*nh*ny)                 # erase data space
do is= 1, ns    {                                       # shots
do ih= 1, nh    {                                       # down cable h = (g-s)/2
do iz= 1, nz    {                                       # Add hyperbola for each layer
        iy = (ns-is)+(ih-1)                             # y = midpoint
        iy = 1 + (iy-ny*(iy/ny))                        # periodic with midpoint
        it = 1 + sqrt( depth(iz)**2 + 25.*(ih-1)**2 )
        if( it <= nt)
                data(it,ih,is) = data(it,ih,is) + refl(iz,iy)
        }}}
return; end
```

Back

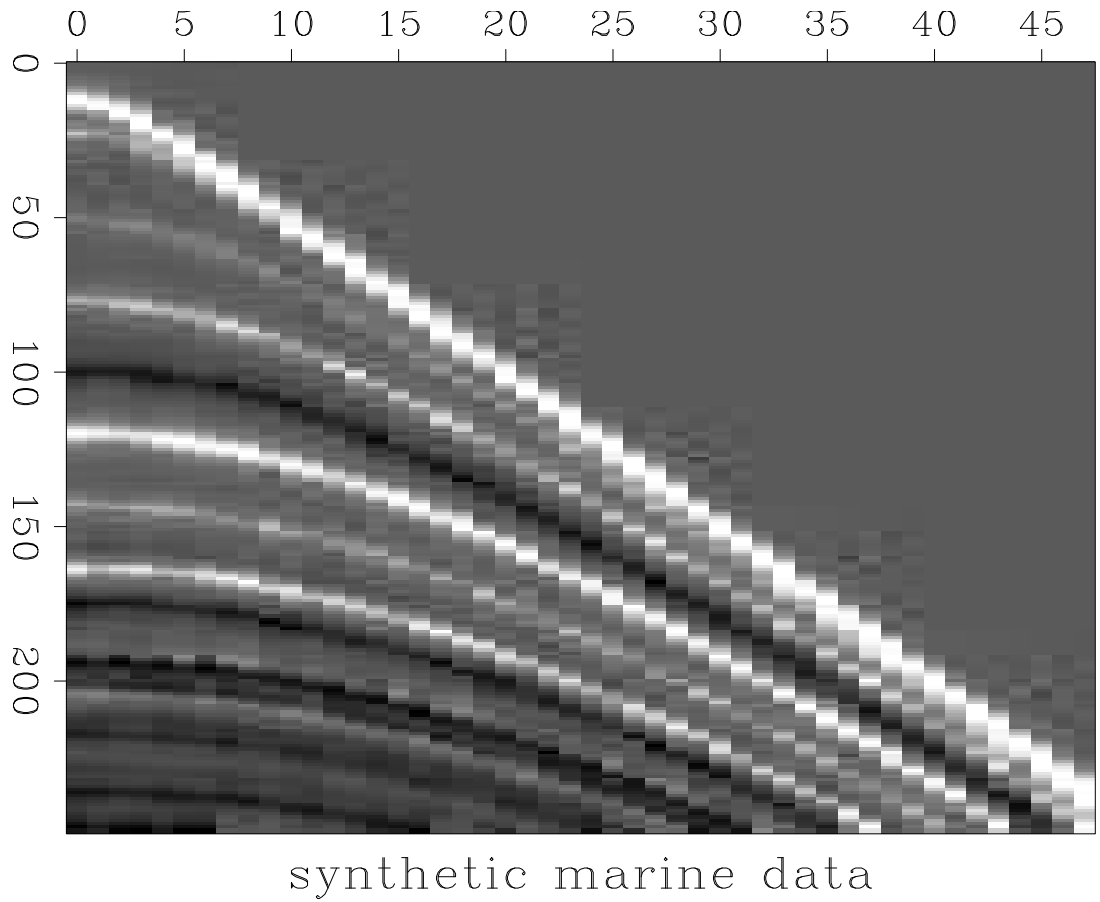


Figure 1.7: Output from `synmarine()` subroutine (with temporal filtering on the *t*-axis). fld-synmarine [ER,M]

What does the final data space look like? This question has little meaning until we decide how the three-dimensional data volume will be presented to the eye. Let us view the data much as it is recorded in the field. For each shot point we see a frame in which the vertical axis is the travel time and the horizontal axis is the distance from the ship down the towed hydrophone cable. The next shot point gives us another frame. Repetition gives us the accompanying program that produces a cube of data, hence a **movie**. This cube is synthetic data for the ideal marine environment. And what does the **movie** show?

A single frame shows hyperbolas with imposed texture. The **movie** shows the texture moving along each hyperbola to increasing offsets. (I find that no sequence of still pictures can give the impression that the **movie** gives). Really the ship is moving; the texture of the earth is remaining stationary under it. This is truly what most marine data looks like, and the computer program simulates it. Comparing the simulated data to real marine-data **movie**s, I am impressed by the large amount of random lateral variation required in the simulated data to achieve resemblance to field data. The randomness seems too great to represent lithologic variation. Apparently it is the result of something not modeled. Perhaps it results from our incomplete understanding of the mechanism of reflection from the quasi-random

earth. Or perhaps it is an effect of the partial focusing of waves sometime after they reflect from minor topographic irregularities. A full explanation awaits more research.

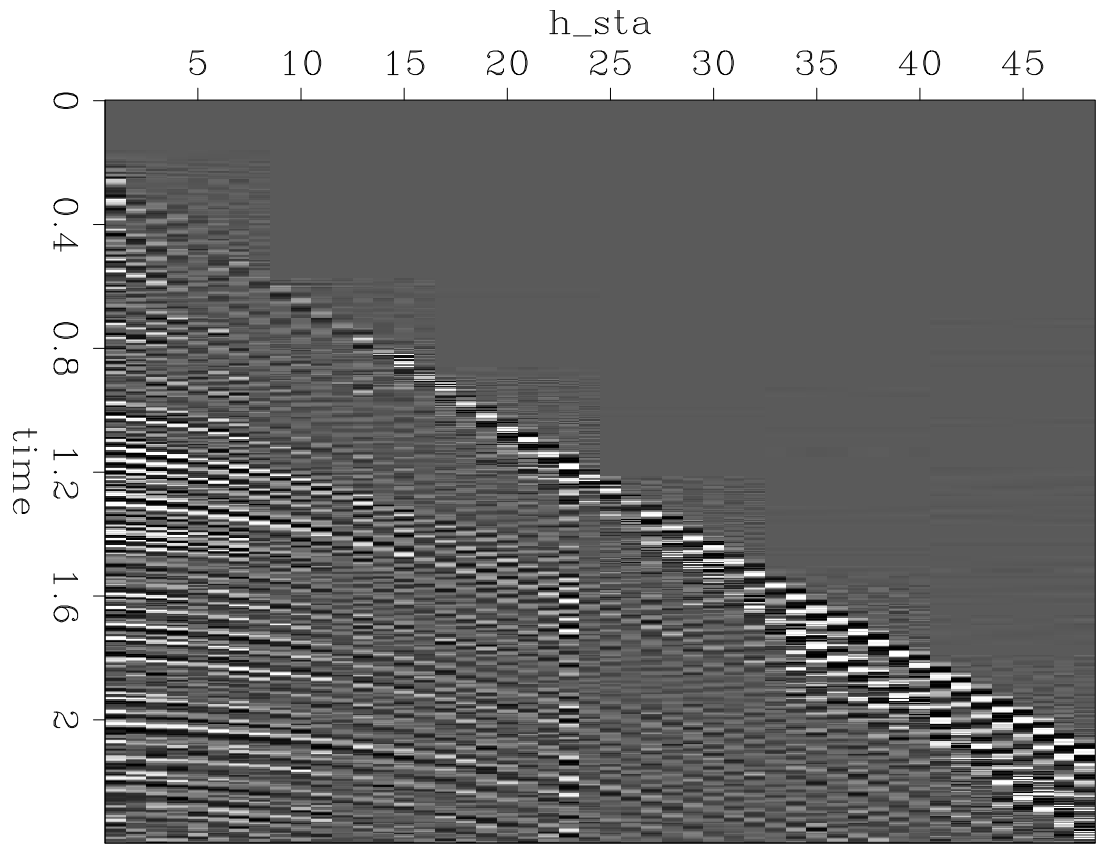


Figure 1.8: Press button for field data **movie**. fld-shotmovie [ER,M]

1.2.2. Texture of land data: near-surface problems

Reflection seismic data recorded on land frequently displays randomness because of the irregularity of the soil layer. Often it is so disruptive that the seismic energy sources are deeply buried (at much cost). The geophones are too many for burial. For most land reflection data, the texture caused by these near-surface irregularities exceeds the texture resulting from the reflecting layers.

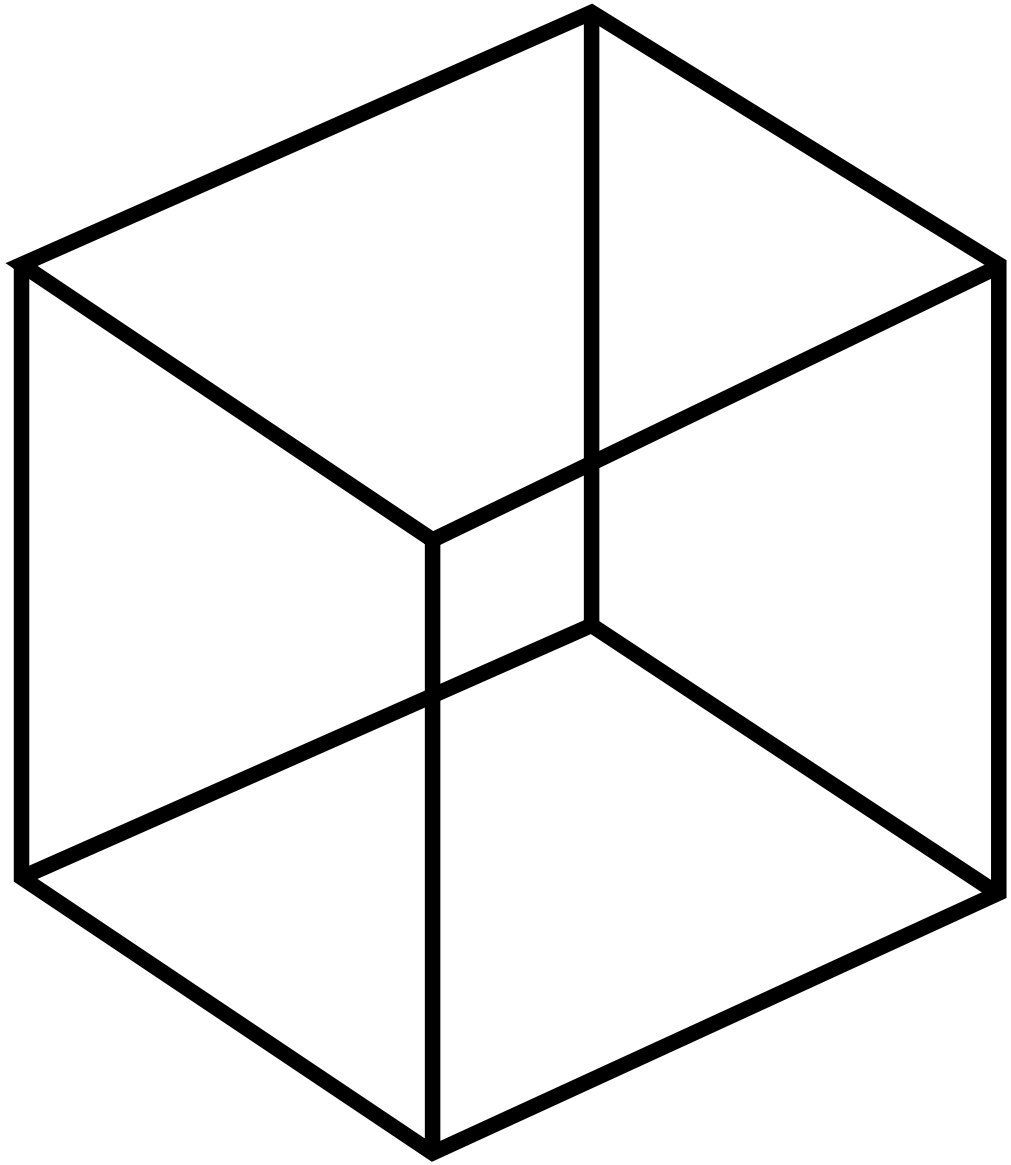
To clarify our thinking, an ideal mathematical model will be proposed. Let the reflecting layers be flat with no texture. Let the geophones suffer random time delays of several time points. Time delays of this type are called *statics*. Let the shots have random strengths. For this **movie**, let the data frames be *common-midpoint gathers*, that is, let each frame show data in (h,t) -space at a fixed midpoint y. Successive frames will show successive midpoints. The study of Figure 1.1 should convince you that the traveltime irregularities associated with the geophones should move leftward, while the amplitude irregularities associated with the shots should move rightward (or vice versa). In real life, both amplitude and time anomalies are associated with both shots and geophones.

EXERCISES:

1 Modify the program of Figure 1.2 to produce a movie of synthetic *midpoint* gathers with random shot amplitudes and random geophone time delays. Observing this **movie** you will note the perceptual problem of being able to see the left-ward motion along with the rightward motion. Try to adjust anomaly strengths so that both left-moving and right-moving patterns are visible. Your mind will often see only one, blocking out the other, similar to the way you perceive a 3-D cube, from a 2-D projection of its edges.

2 Define recursive dip filters to pass and reject the various textures of shot, geo-phone, and midpoint.

Figure 1.9: fld-wirecube [NR]



Chapter 2

Adjoint operators

A great many of the calculations we do in science and engineering are really matrix multiplication in disguise. The first goal of this chapter is to unmask the disguise by showing many examples. Second, we see how the **adjoint** operator (matrix

transpose) back-projects information from data to the underlying model.

Geophysical modeling calculations generally use linear operators that predict data from models. Our usual task is to find the inverse of these calculations; i.e., to find models (or make maps) from the data. Logically, the adjoint is the first step and a part of all subsequent steps in this **inversion** process. Surprisingly, in practice the adjoint sometimes does a better job than the inverse! This is because the adjoint operator tolerates imperfections in the data and does not demand that the data provide full information.

Using the methods of this chapter, you will find that once you grasp the relationship between operators in general and their adjoints, you can obtain the adjoint just as soon as you have learned how to code the modeling operator.

If you will permit me a poet's license with words, I will offer you the following table of **operator**s and their **adjoint**s:

matrix multiply	conjugate-transpose matrix multiply
convolve	crosscorrelate
truncate	zero pad
replicate, scatter, spray	sum or stack
spray into neighborhood	sum in bins

derivative (slope)	negative derivative
causal integration	anticausal integration
add functions	do integrals
assignment statements	added terms
plane-wave superposition	slant stack / beam form
superpose on a curve	sum along a curve
stretch	squeeze
upward continue	downward continue
hyperbolic modeling	normal moveout and CDP stack
diffraction modeling	imaging by migration
ray tracing	**tomography**

The left column above is often called "**modeling**," and the adjoint operators on the right are often used in "data **processing**."

The adjoint operator is sometimes called the "**back projection**" operator because information propagated in one direction (earth to data) is projected backward (data to earth model). For complex-valued operators, the transpose goes together with a complex conjugate. In **Fourier analysis**, taking the complex conjugate of

$\exp(i\omega t)$ reverses the sense of time. With more poetic license, I say that adjoint operators *undo* the time and phase shifts of modeling operators. The inverse operator does this too, but it also divides out the color. For example, when linear interpolation is done, then high frequencies are smoothed out, so inverse interpolation must restore them. You can imagine the possibilities for noise amplification. That is why adjoints are safer than inverses.

Later in this chapter we relate adjoint operators to inverse operators. Although inverse operators are more well known than adjoint operators, the inverse is built upon the adjoint so the adjoint is a logical place to start. Also, computing the inverse is a complicated process fraught with pitfalls whereas the computation of the adjoint is easy. It's a natural companion to the operator itself.

Much later in this chapter is a formal definition of adjoint operator. Throughout the chapter we handle an adjoint operator as a matrix transpose, but we hardly ever write down any matrices or their transposes. Instead, we always prepare two subroutines, one that performs $\mathbf{y} = \mathbf{A}\mathbf{x}$ and another that performs $\tilde{\mathbf{x}} = \mathbf{A}'\mathbf{y}$. So we need a test that the two subroutines really embody the essential aspects of matrix transposition. Although the test is an elegant and useful test and is itself a fundamental definition, curiously, that definition does not help construct adjoint operators, so we

postpone a formal definition of adjoint until after we have seen many examples.

2.1. FAMILIAR OPERATORS

The operation $y_i = \sum_j b_{ij} x_j$ is the multiplication of a matrix **B** by a vector **x**. The adjoint operation is $\tilde{x}_j = \sum_i b_{ij} y_i$. The operation adjoint to multiplication by a matrix is multiplication by the transposed matrix (unless the matrix has complex elements, in which case we need the complex-conjugated transpose). The following **pseudocode** does matrix multiplication $\mathbf{y} = \mathbf{B}\mathbf{x}$ and multiplication by the transpose $\tilde{\mathbf{x}} = \mathbf{B}'\mathbf{y}$:

```
if operator itself
        then erase y
if adjoint
        then erase x
do iy = 1, ny {
do ix = 1, nx {
        if operator itself
                y(iy) = y(iy) + b(iy,ix) × x(ix)
        if adjoint
                x(ix) = x(ix) + b(iy,ix) × y(iy)
}       }
```

Notice that the "bottom line" in the program is that *x* and *y* are simply interchanged. The above example is a prototype of many to follow, so observe carefully the similarities and differences between the operation and its adjoint.

A formal subroutine[1] for **matrix multiply** and its adjoint is found below. The first step is a subroutine, `adjnull()`, for optionally erasing the output. With the

[1] The programming language used in this book is Ratfor, a dialect of Fortran.

```
subroutine adjnull( adj, add, x, nx,  y, ny )
integer ix, iy,     adj, add,    nx,     ny
real                           x( nx), y( ny )
if( add == 0 )
        if( adj == 0 )
                do iy= 1, ny
                        y(iy) = 0.
        else
                do ix= 1, nx
                        x(ix) = 0.
return; end
```

Back

```
# matrix multiply and its adjoint
#
subroutine matmult( adj, add, bb,         x,nx,  y,ny)
integer ix, iy,     adj, add,               nx,    ny
real                           bb(ny,nx), x(nx), y(ny)
call adjnull(       adj, add,             x,nx,  y,ny)
do ix= 1, nx {
do iy= 1, ny {
        if( adj == 0 )
                        y(iy) = y(iy) + bb(iy,ix) * x(ix)
        else
                        x(ix) = x(ix) + bb(iy,ix) * y(iy)
        }}
return; end
```

Back

option `add=1`, results accumulate like `y=y+B*x`. adjnull The subroutine `matmult()` for matrix multiply and its adjoint exhibits the style that we will use repeatedly. matmult

Sometimes a matrix operator reduces to a simple row or a column.
A **row** is a summation operation.
A **column** is an impulse response.

If the inner loop of a matrix multiply ranges within a
row, the operator is called *sum* or *pull*.
column, the operator is called *spray* or *push*.

A basic aspect of adjointness is that the adjoint of a row matrix operator is a column matrix operator. For example, the row operator $[a, b]$

$$y \quad = \quad [\, a \;\; b \,] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad = \quad a x_1 + b x_2 \tag{2.1}$$

For more details, see Appendix A.

has an adjoint that is two assignments:

$$\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} y \tag{2.2}$$

The adjoint of a sum of N terms is a collection of N assignments.

2.1.1. Adjoint derivative

Given a sampled signal, its time **derivative** can be estimated by convolution with the filter $(1, -1)/\Delta t$, expressed as the matrix-multiply below:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} -1 & 1 & . & . & . & . \\ . & -1 & 1 & . & . & . \\ . & . & -1 & 1 & . & . \\ . & . & . & -1 & 1 & . \\ . & . & . & . & -1 & 1 \\ . & . & . & . & . & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \tag{2.3}$$

Technically the output should be `n-1` points long, but I appended a zero row, a small loss of logical purity, so that the size of the output vector will match that of the input. This is a convenience for plotting and for simplifying the assembly of other operators building on this one.

The **filter impulse response** is seen in any column in the middle of the matrix, namely $(1, -1)$. In the transposed matrix, the filter-impulse response is time-reversed to $(-1, 1)$. So, mathematically, we can say that the adjoint of the time derivative operation is the negative time derivative. This corresponds also to the fact that the complex conjugate of $-i\omega$ is $i\omega$. We can also speak of the adjoint of the boundary conditions: we might say that the adjoint of "no boundary condition" is a "specified value" boundary condition.

A complicated way to think about the adjoint of equation (2.3) is to note that it is the negative of the derivative and that something must be done about the ends. A simpler way to think about it is to apply the idea that the adjoint of a sum of N terms is a collection of N assignments. This is done in subroutine `igrad1()`, which implements equation (2.3) and its adjoint. igrad1
Notice that the do loop in the code covers all the outputs for the operator itself, and that in the adjoint operation it gathers all the inputs. This is natural because in

```
subroutine igrad1( adj, add,  xx,n,  yy   )
integer i,         adj, add,     n
real                            xx(n), yy(n)
call adjnull(      adj, add,  xx,n,  yy,n )
do i= 1, n-1 {
        if( adj == 0 )
                yy(i) = yy(i) + xx(i+1) - xx(i)
        else {
                xx(i+1) = xx(i+1) + yy(i)
                xx(i  ) = xx(i  ) - yy(i)
                }
        }
return; end
```

Back

switching from operator to adjoint, the outputs switch to inputs.

As you look at the code, think about matrix elements being $+1$ or -1 and think about the forward operator "pulling" a sum into `yy(i)`, and think about the adjoint operator "pushing" or "spraying" the impulse `yy(i)` back into `xx()`.

You might notice that you can simplify the program by merging the "erase output" activity with the calculation itself. We will not do this optimization however because in many applications we do not want to include the "erase output" activity. This often happens when we build complicated operators from simpler ones.

2.1.2. Zero padding is the transpose of truncation

Surrounding a dataset by zeros (**zero pad**ding) is adjoint to throwing away the extended data (**truncation**). Let us see why this is so. Set a signal in a vector $\mathbf{x}$, and then to make a longer vector $\mathbf{y}$, add some zeros at the end of $\mathbf{x}$. This zero padding can be regarded as the matrix multiplication

$$\mathbf{y} \quad = \quad \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} \mathbf{x} \tag{2.4}$$

The matrix is simply an identity matrix $\mathbf{I}$ above a zero matrix $\mathbf{0}$. To find the transpose to zero-padding, we now transpose the matrix and do another matrix multiply:

$$\tilde{\mathbf{x}} \quad = \quad \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \mathbf{y} \tag{2.5}$$

So the transpose operation to zero padding data is simply *truncating* the data back to its original length. Subroutine `zpad1()` below pads zeros on both ends of its input. Subroutines for two- and three-dimensional padding are in the library named `zpad2()` and `zpad3()`. zpad1

2.1.3. Adjoints of products are reverse-ordered products of adjoints

Here we examine an example of the general idea that adjoints of products are reverse-ordered products of adjoints. For this example we use the Fourier transformation. No details of **Fourier transformation** are given here and we merely use it as an example of a square matrix $\mathbf{F}$. We denote the complex-conjugate transpose (or **adjoint**) matrix with a prime, i.e., $\mathbf{F}'$. The adjoint arises naturally whenever we consider energy. The statement that Fourier transforms conserve energy is $\mathbf{y}'\mathbf{y} = \mathbf{x}'\mathbf{x}$

```
# Zero pad.  Surround data by zeros. 1-D
#
subroutine zpad1( adj,add,  data,nd,  padd,np )
integer           adj,add,    d, nd,    p, np
real                          data(nd), padd(np)
call adjnull(     adj,add,  data,nd,  padd,np)
do d= 1, nd {                  p = d + (np-nd)/2
        if( adj == 0 )
                              padd(p) = padd(p) + data(d)
        else
                              data(d) = data(d) + padd(p)
        }
return; end
```

Back

where $\mathbf{y} = \mathbf{F}\mathbf{x}$. Substituting gives $\mathbf{F}'\mathbf{F} = \mathbf{I}$, which shows that the inverse matrix to Fourier transform happens to be the complex conjugate of the transpose of $\mathbf{F}$.

With Fourier transforms, **zero pad**ding and **truncation** are especially prevalent. Most subroutines transform a dataset of length of 2^n, whereas dataset lengths are often of length $m \times 100$. The practical approach is therefore to pad given data with zeros. Padding followed by Fourier transformation $\mathbf{F}$ can be expressed in matrix algebra as

$$\text{Program} \quad = \quad \mathbf{F} \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} \tag{2.6}$$

According to matrix algebra, the transpose of a product, say $\mathbf{AB} = \mathbf{C}$, is the product $\mathbf{C}' = \mathbf{B}'\mathbf{A}'$ in reverse order. So the adjoint subroutine is given by

$$\text{Program}' \quad = \quad \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \mathbf{F}' \tag{2.7}$$

Thus the adjoint subroutine *truncates* the data *after* the inverse Fourier transform. This concrete example illustrates that common sense often represents the mathematical abstraction that adjoints of products are reverse-ordered products of adjoints. It is also nice to see a formal mathematical notation for a practical necessity. Making an approximation need not lead to collapse of all precise analysis.

2.1.4. Nearest-neighbor coordinates

In describing physical processes, we often either specify models as values given on a uniform mesh or we record data on a uniform mesh. Typically we have a function f of time t or depth z and we represent it by `f(iz)` corresponding to $f(z_i)$ for $i = 1, 2, 3, \ldots, n_z$ where $z_i = z_0 + (i-1)\Delta z$. We sometimes need to handle depth as an integer counting variable i and we sometimes need to handle it as a floating-point variable z. Conversion from the counting variable to the floating-point variable is exact and is often seen in a computer idiom such as either of

```
do iz= 1, nz {   z = z0 + (iz-1) * dz
do i3= 1, n3 {  x3 = o3 + (i3-1) * d3
```

The reverse conversion from the floating-point variable to the counting variable is inexact. The easiest thing is to place it at the nearest neighbor. This is done by solving for `iz`, then adding one half, and then rounding down to the nearest integer. The familiar computer idioms are:

```
iz =  .5 + 1 + ( z - z0) / dz
iz = 1.5 +     ( z - z0) / dz
i3 = 1.5 +     (x3 - o3) / d3
```

A small warning is in order: People generally use positive counting variables. If you also include negative ones, then to get the nearest integer, you should do your rounding with the Fortran function `NINT()`.

2.1.5. Data-push binning

Binning is putting data values in bins. Nearest-neighbor binning is an operator. There is both a forward operator and its adjoint. Normally the model consists of values given on a uniform mesh, and the data consists of pairs of numbers (ordinates at coordinates) sprinkled around in the continuum (although sometimes the data is uniformly spaced and the model is not).

In both the forward and the adjoint operation, each data coordinate is examined and the nearest mesh point (the bin) is found. For the forward operator, the value of the bin is added to that of the data. The adjoint is the reverse: we add the value of the data to that of the bin. Both are shown in two dimensions in subroutine `dpbin2()`.

dpbin2

The most typical application requires an additional step, inversion. In the inversion applications each bin contains a different number of data values. After the adjoint operation is performed, the inverse operator divides the bin value by the

```
# Data-push binning in 2-D.
#
subroutine dpbin2 ( adj, add, o1,d1,o2,d2, xy,        mm,m1,m2,  dd, nd)
integer      i1,i2, adj, add,                    id,     m1,m2,       nd
real                          o1,d1,o2,d2, xy(2,nd), mm(m1,m2), dd( nd)
call adjnull(       adj, add,                        mm,m1*m2,  dd, nd)
do id=1,nd {
        i1 = 1.5 + (xy(1,id)-o1)/d1
        i2 = 1.5 + (xy(2,id)-o2)/d2
        if( 1<=i1 && i1<=m1 &&
            1<=i2 && i2<=m2   )
                if( adj == 0)
                        dd( id)   = dd( id)   +  mm(i1,i2)
                else
                        mm(i1,i2) = mm(i1,i2) +  dd( id)
        }
return; end
```

Back

number of points in the bin. It is this inversion operator that is generally called binning. To find the number of data points in a bin, we can simply apply the adjoint of `dpbin2()` to pseudo data of all ones.

2.1.6. Linear interpolation

The **linear interpolation** operator is much like the binning operator but a little fancier. When we perform the forward operation, we take each data coordinate and see which two model mesh points bracket it. Then we pick up the two bracketing model values and weight each of them in proportion to their nearness to the data coordinate, and add them to get the data value (ordinate). The adjoint operation is adding a data value back into the model vector; using the same two weights, this operation distributes the ordinate value between the two nearest points in the model vector. For example, suppose we have a data point near each end of the model and a third data point exactly in the middle. Then for a model space 6 points long, as shown in Figure 2.1, we have the operator in (2.8).

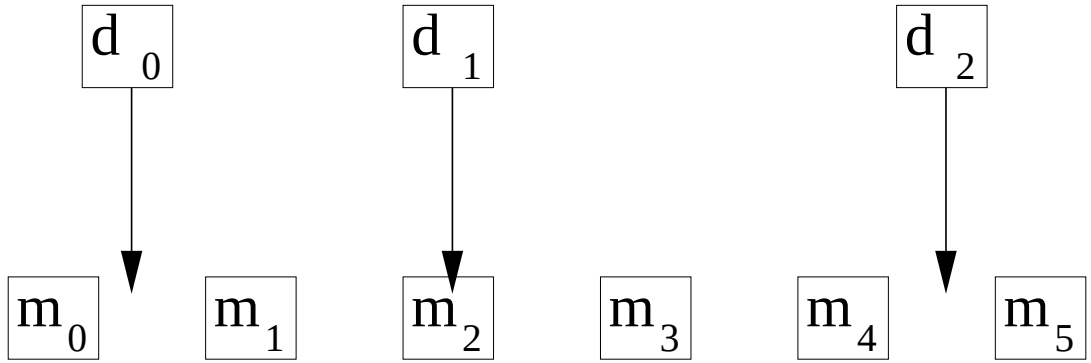


Figure 2.1: Uniformly sampled model space and irregularly sampled data space corresponding to (2.8). conj-helgerud [NR]

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} \approx \begin{bmatrix} .8 & .2 & . & . & . & . \\ . & . & 1 & . & . & . \\ . & . & . & . & .5 & .5 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{bmatrix} \tag{2.8}$$

The two weights in each row sum to unity. If a binning operator were used for the same data and model, the binning operator would contain a "1." in each row. In one dimension (as here), data coordinates are often sorted into sequence, so that the matrix is crudely a diagonal matrix like equation (2.8). If the data coordinates covered the model space uniformly, the adjoint would roughly be the inverse. Otherwise,

when data values pile up in some places and gaps remain elsewhere, the adjoint would be far from the inverse.

Subroutine `lint1()` does linear interpolation and its adjoint. lint1

2.1.7. Causal integration

Causal integration is defined as

$$y(t) \quad = \quad \int_{-\infty}^{t} x(t)\, dt \tag{2.9}$$

```
# Linear interpolation 1-D, uniform model mesh to data coordinates and values.
#
subroutine lint1( adj, add, o1,d1,coordinate,     mm,m1,  dd, nd)
integer i, im,    adj, add,                   id,     m1,      nd
real    f, fx,gx,          o1,d1,coordinate(nd), mm(m1), dd( nd)
call adjnull(     adj, add,                        mm,m1,  dd, nd)
do id= 1, nd {
        f = (coordinate(id)-o1)/d1;     i=f;     im= 1+i
        if( 1<=im && im<m1) {          fx=f-i;   gx= 1.-fx
                if( adj == 0)
                        dd(id) = dd(id) + gx * mm(im) + fx * mm(im+1)
                else {
                        mm(im  ) = mm(im  ) + gx * dd(id)
                        mm(im+1) = mm(im+1) + fx * dd(id)
                        }
                }
        }
return; end
```

Back

Sampling the time axis gives a matrix equation which we should call causal summation, but we often call it causal integration.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix} \tag{2.10}$$

(In some applications the 1 on the diagonal is replaced by 1/2.) Causal integration is the simplest prototype of a recursive operator. The coding is trickier than operators we considered earlier. Notice when you compute y_5 that it is the sum of 6 terms, but that this sum is more quickly computed as $y_5 = y_4 + x_5$. Thus equation (2.10)

is more efficiently thought of as the recursion

$$y_t \quad = \quad y_{t-1} + x_t \qquad \text{for increasing } t \tag{2.11}$$

(which may also be regarded as a numerical representation of the **differential equation** $dy/dt = x$.)

When it comes time to think about the adjoint, however, it is easier to think of equation (2.10) than of (2.11). Let the matrix of equation (2.10) be called $\mathbf{C}$. Transposing to get $\mathbf{C}'$ and applying it to $\mathbf{y}$ gives us something back in the space of $\mathbf{x}$, namely $\tilde{\mathbf{x}} = \mathbf{C}'\mathbf{y}$. From it we see that the adjoint calculation, if done recursively, needs to be done backwards like

$$\tilde{x}_{t-1} \quad = \quad \tilde{x}_t + y_{t-1} \qquad \text{for decreasing } t \tag{2.12}$$

We can sum up by saying that the adjoint of causal integration is anticausal integration.

A subroutine to do these jobs is `causint()` /prog:causint. The code for anticausal integration is not obvious from the code for integration and the adjoint coding tricks we learned earlier. To understand the adjoint, you need to inspect the detailed form of the expression $\tilde{\mathbf{x}} = \mathbf{C}'\mathbf{y}$ and take care to get the ends correct. causint

```
# causal integration  (1's on diagonal)
#
subroutine causint( adj, add,         n,xx,  yy    )
integer i,        n, adj, add;   real xx(n), yy(n )
temporary real tt( n)
call adjnull(        adj, add,         xx,n,  yy,n )
if( adj == 0){                                    tt(1) = xx(1)
                                             do i= 2, n
                                                  tt(i) = tt(i-1) + xx(i)
        do i= 1, n
                yy(i) = yy(i) + tt(i)
        }
else {                                            tt(n) = yy(n)
                                             do i= n, 2, -1
                                                  tt(i-1) = tt(i) + yy(i-1)
        do i= 1, n
                xx(i) = xx(i) + tt(i)
        }
return; end
```

Back

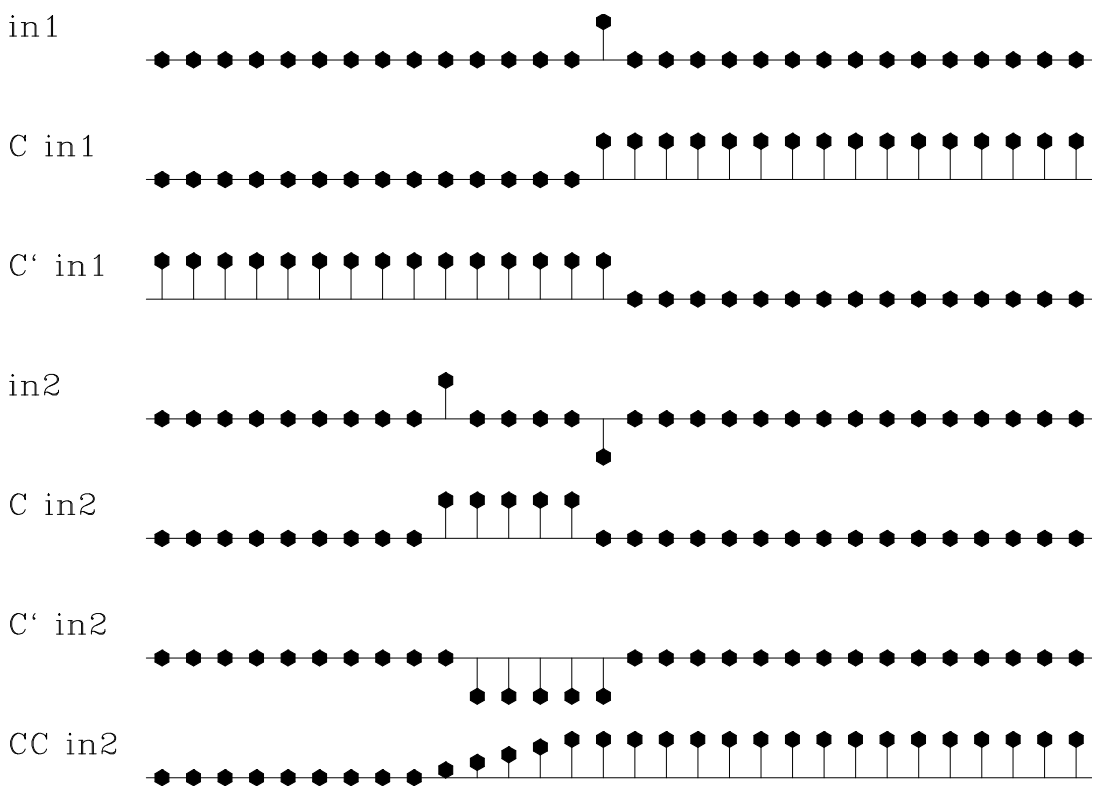


Figure 2.2: `in1` is an input pulse. `C in1` is its causal integral. `C' in1` is the anticausal integral of the pulse. `in2` is a separated doublet. Its causal integration is a box and its anticausal integration is the negative. `CC in2` is the double causal integral of `in2`. How can an equilateral triangle be built? conj-causint [ER]

Later we will consider equations to march wavefields up towards the earth's surface, a layer at a time, an operator for each layer. Then the adjoint will start from the earth's surface and march down, a layer at a time, into the earth.

EXERCISES:

1 Modify the calculation in Figure 2.2 to make a triangle waveform on the bottom row.

2.2. ADJOINTS AND INVERSES

Consider a model $\mathbf{m}$ and an operator $\mathbf{F}$ which creates some theoretical data $\mathbf{d}_{\text{theor}}$.

$$\mathbf{d}_{\text{theor}} \quad = \quad \mathbf{F}\mathbf{m} \tag{2.13}$$

The general task of geophysicists is to begin from observed data $\mathbf{d}_{\text{obs}}$ and find an estimated model $\mathbf{m}_{\text{est}}$ that satisfies the simultaneous equations

$$\mathbf{d}_{\text{obs}} \quad = \quad \mathbf{F}\mathbf{m}_{\text{est}} \tag{2.14}$$

This is the topic of a large discipline variously called "inversion" or "estimation". Basically, it defines a residual $\mathbf{r} = \mathbf{d}_{\text{obs}} - \mathbf{d}_{\text{theor}}$ and then minimizes its length $\mathbf{r} \cdot \mathbf{r}$. Finding $\mathbf{m}_{\text{est}}$ this way is called the **least squares** method. The basic result (not proven here) is that

$$\mathbf{m}_{\text{est}} = (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{d}_{\text{obs}} \tag{2.15}$$

In many cases including all seismic imaging cases, the matrix $\mathbf{F}'\mathbf{F}$ is far too large to be invertible. People generally proceed by a rough guess at an approximation for $(\mathbf{F}'\mathbf{F})^{-1}$. The usual first approximation is the optimistic one that $(\mathbf{F}'\mathbf{F})^{-1} = \mathbf{I}$. To this happy approximation, the inverse $\mathbf{F}^{-1}$ is the adjoint $\mathbf{F}'$.

In this book we'll see examples where $\mathbf{F}'\mathbf{F} \approx \mathbf{I}$ is a good approximation and other examples where it isn't. We can tell how good the approximation is. We take some hypothetical data and convert it to a model, and use that model to make some reconstructed data $\mathbf{d}_{\text{recon}} = \mathbf{F}\mathbf{F}'\mathbf{d}_{\text{hypo}}$. Likewise we could go from a hypothetical model to some data and then to a reconstructed model $\mathbf{m}_{\text{recon}} = \mathbf{F}'\mathbf{F}\mathbf{m}_{\text{hypo}}$. Luckily, it often happens that the reconstructed differs from the hypothetical in some trivial way, like by a scaling factor, or by a scaling factor that is a function of physical location or time, or a scaling factor that is a function of frequency. It isn't always simply a matter of a scaling factor, but it often is, and when it is, we often simply

redefine the operator to include the scaling factor. Observe that there are two places for scaling functions (or filters), one in model space, the other in data space.

We could do better than the adjoint by iterative modeling methods (conjugate gradients) that are also described elsewhere. These methods generally demand that the adjoint be computed correctly. As a result, we'll be a little careful about adjoints in this book to compute them correctly even though this book does not require them to be exactly correct.

2.2.1. Dot product test

We define an adjoint when we write a program that computes one. In an abstract logical mathematical sense, however, every adjoint is defined by a **dot product test**. This abstract definition gives us no clues how to code our program. After we have finished coding, however, this abstract definition (which is actually a test) has considerable value to us.

Conceptually, the idea of matrix transposition is simply $a'_{ij} = a_{ji}$. In practice, however, we often encounter matrices far too large to fit in the memory of any computer. Sometimes it is also not obvious how to formulate the process at hand

as a matrix multiplication. (Examples are differential equations and fast Fourier transforms.) What we find in practice is that an application and its adjoint amounts to two subroutines. The first subroutine amounts to the matrix multiplication $\mathbf{Fx}$. The adjoint subroutine computes $\mathbf{F}'\mathbf{y}$, where $\mathbf{F}'$ is the conjugate-transpose matrix. Most methods of solving inverse problems will fail if the programmer provides an inconsistent pair of subroutines for $\mathbf{F}$ and $\mathbf{F}'$. The dot product test described next is a simple test for verifying that the two subroutines really are adjoint to each other.

The matrix expression $\mathbf{y}'\mathbf{Fx}$ may be written with parentheses as either $(\mathbf{y}'\mathbf{F})\mathbf{x}$ or $\mathbf{y}'(\mathbf{Fx})$. Mathematicians call this the "associative" property. If you write matrix multiplication using summation symbols, you will notice that putting parentheses around matrices simply amounts to reordering the sequence of computations. But we soon get a very useful result. Programs for some linear operators are far from obvious, for example `causint()` /prog:causint. Now we build a useful test for it.

$$\mathbf{y}'(\mathbf{Fx}) \quad = \quad (\mathbf{y}'\mathbf{F})\mathbf{x} \tag{2.16}$$

$$\mathbf{y}'(\mathbf{Fx}) \quad = \quad (\mathbf{F}'\mathbf{y})'\mathbf{x} \tag{2.17}$$

For the dot-product test, load the vectors $\mathbf{x}$ and $\mathbf{y}$ with random numbers. Compute the vector $\tilde{\mathbf{y}} = \mathbf{Fx}$ using your program for $\mathbf{F}$, and compute $\tilde{\mathbf{x}} = \mathbf{F}'\mathbf{y}$ using your pro-

gram for $\mathbf{F}'$. Inserting these into equation (2.17) gives you two scalars that should be equal.

$$\mathbf{y}'(\mathbf{F}\mathbf{x}) \quad = \quad \mathbf{y}'\tilde{\mathbf{y}} \quad = \quad \tilde{\mathbf{x}}'\mathbf{x} \quad = \quad (\mathbf{F}'\mathbf{y})'\mathbf{x} \tag{2.18}$$

The left and right sides of this equation will be computationally equal only if the program doing $\mathbf{F}'$ is indeed adjoint to the program doing $\mathbf{F}$ (unless the random numbers do something miraculous). Note that the vectors $\mathbf{x}$ and $\mathbf{y}$ are generally of different lengths.

Of course passing the dot product test does not prove that a computer code is correct, but if the test fails we know the code is incorrect. More information about adjoint operators, and much more information about inverse operators is found in my other books, Earth Soundings Analysis: Processing versus inversion (PVI) and Geophysical Estimation by Example (GEE).

Chapter 3

Waves in strata

The waves of practical interest in reflection seismology are usually complicated because the propagation velocities are generally complex. In this book, we have chosen to build up the complexity of the waves we consider, chapter by chapter. The

simplest waves to understand are simple plane waves and spherical waves propagating through a constant-velocity medium. In seismology however, the earth's velocity is almost never well approximated by a constant. A good first approximation is to assume that the earth's velocity increases with depth. In this situation, the simple planar and circular wavefronts are modified by the effects of $v(z)$. In this chapter we study the basic equations describing plane-like and spherical-like waves propagating in media where the velocity $v(z)$ is a function only of depth. This is a reasonable starting point, even though it neglects the even more complicated distortions that occur when there are lateral velocity variations. We will also examine data that shows plane-like waves and spherical-like waves resulting when waves from a point source bounce back from a planar reflector.

3.1. TRAVEL-TIME DEPTH

Echo soundings give us a picture of the earth. A zero-offest section, for example, is a planar display of traces where the horizontal axis runs along the earth's surface and the vertical axis, running down, seems to measure depth, but actually measures the two-way echo delay time. Thus, in practice the vertical axis is almost never depth

z; it is the *vertical travel time* τ. In a constant-velocity earth the time and the depth are related by a simple scale factor, the speed of sound. This is analogous to the way that astronomers measure distances in light-years, always referencing the speed of light. The meaning of the scale factor in seismic imaging is that the (x, τ)-plane has a vertical exaggeration compared to the (x, z)-plane. In reconnaissance work, the vertical is often exaggerated by about a factor of five. By the time prospects have been sufficiently narrowed for a drill site to be selected, the vertical exaggeration factor in use is likely to be about unity (no exaggeration).

In seismic reflection imaging, the waves go down and then up, so the **traveltime depth** τ is defined as two-way vertical travel time.

$$\tau = \frac{2z}{v} \quad . \tag{3.1}$$

This is the convention that I have chosen to use throughout this book.

3.1.1. Vertical exaggeration

The first task in interpretation of seismic data is to figure out the approximate numerical value of the **vertical exaggeration**. The vertical exaggeration is $2/v$ be-

cause it is the ratio of the apparent slope $\Delta\tau/\Delta x$ to the actual slope $\Delta z/\Delta x$ where $\Delta\tau = 2\ \Delta z/v$. Since the velocity generally *increases* with depth, the **vertical exaggeration** generally *decreases* with depth.

For velocity-stratified media, the time-to-depth conversion formula is

$$\tau(z) \;=\; \int_0^z \frac{2\,dz}{v(z)} \qquad \text{or} \qquad \frac{d\tau}{dz} \;=\; \frac{2}{v} \tag{3.2}$$

3.2. HORIZONTALLY MOVING WAVES

In practice, horizontally going waves are easy to recognize because their travel time is a linear function of the offset distance between shot and receiver. There are two kinds of horizontally going waves, one where the traveltime line goes through the origin, and the other where it does not. When the line goes through the origin, it means the ray path is always near the earth's surface where the sound source and the receivers are located. (Such waves are called "**ground roll**" on land or "**guided waves**" at sea; sometimes they are just called "**direct arrivals**".)

When the traveltime line does not pass through the origin it means parts of the ray path plunge into the earth. This is usually explained by the unlikely looking

rays shown in Figure 3.1 which frequently occur in practice. Later in this chapter

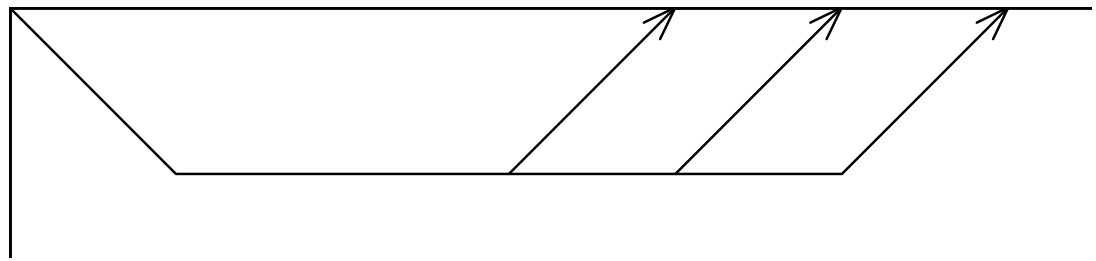


Figure 3.1: Rays associated with **head waves**. wvs-headray [ER]

we will see that Snell's law predicts these rays in a model of the earth with two layers, where the deeper layer is faster and the ray bottom is along the interface between the slow medium and the fast medium. Meanwhile, however, notice that these ray paths imply data with a linear travel time versus distance corresponding to increasing ray length along the ray bottom. Where the ray is horizontal in the lower medium, its wavefronts are vertical. These waves are called "**head waves**," perhaps because they are typically fast and arrive *ahead* of other waves.

3.2.1. Amplitudes

The nearly vertically-propagating waves (reflections) spread out essentially in three dimensions, whereas the nearly horizontally-going waves never get deep into the

earth because, as we will see, they are deflected back upward by the velocity gradient. Thus horizontal waves spread out in essentially two dimensions, so that energy conservation suggests that their amplitudes should dominate the amplitudes of reflections on raw data. This is often true for **ground roll**. Head waves, on the other hand, are often much weaker, often being visible only because they often arrive before more energetic waves. The weakness of **head wave**s is explained by the small percentage of solid angle occupied by the waves leaving a source that eventually happen to match up with layer boundaries and propagate as head waves. I selected the examples below because of the strong headwaves. They are nearly as strong as the guided waves. To compensate for diminishing energy with distance, I scaled data displays by multiplying by the offset distance between the shot and the receiver.

In data display, the slowness (slope of the time-distance curve) is often called the **stepout** p. Other commonly-used names for this slope are **time dip** and **reflection slope**. The best way to view waves with **linear moveout** is after time shifting to remove a standard linear moveout such as that of water. An equation for the shifted time is

$$\tau \quad = \quad t - px \tag{3.3}$$

where p is often chosen to be the inverse of the velocity of water, namely, about 1.5

km/s, or $p = .66\text{s/km}$ and $x = 2h$ is the horizontal separation between the sound source and receiver, usually referred to as the **offset**.

Ground roll and **guided wave**s are typically slow because materials near the earth's surface typically are slow. Slow waves are steeply sloped on a time-versus-offset display. It is not surprising that marine guided waves typically have speeds comparable to water waves (near 1.47 km/s approximately 1.5 km/s). It is perhaps surprising that **ground roll** also often has the speed of sound in water. Indeed, the depth to underground water is often determined by seismology before drilling for water. Ground roll also often has a speed comparable to the speed of sound in air, 0.3 km/sec, though, much to my annoyance I could not find a good example of it today. Figure 3.2 is an example of energetic **ground roll** (land) that happens to have a speed close to that of water.

The speed of a ray traveling along a layer interface is the rock speed in the faster layer (nearly always the lower layer). It is not an average of the layer above and the layer below.

Figures 3.3 and 3.4 are examples of energetic marine guided waves. In Figure 3.3 at $\tau = 0$ (designated `t-t_water`) at small offset is the wave that travels directly from the shot to the receivers. This wave dies out rapidly with offset (be-

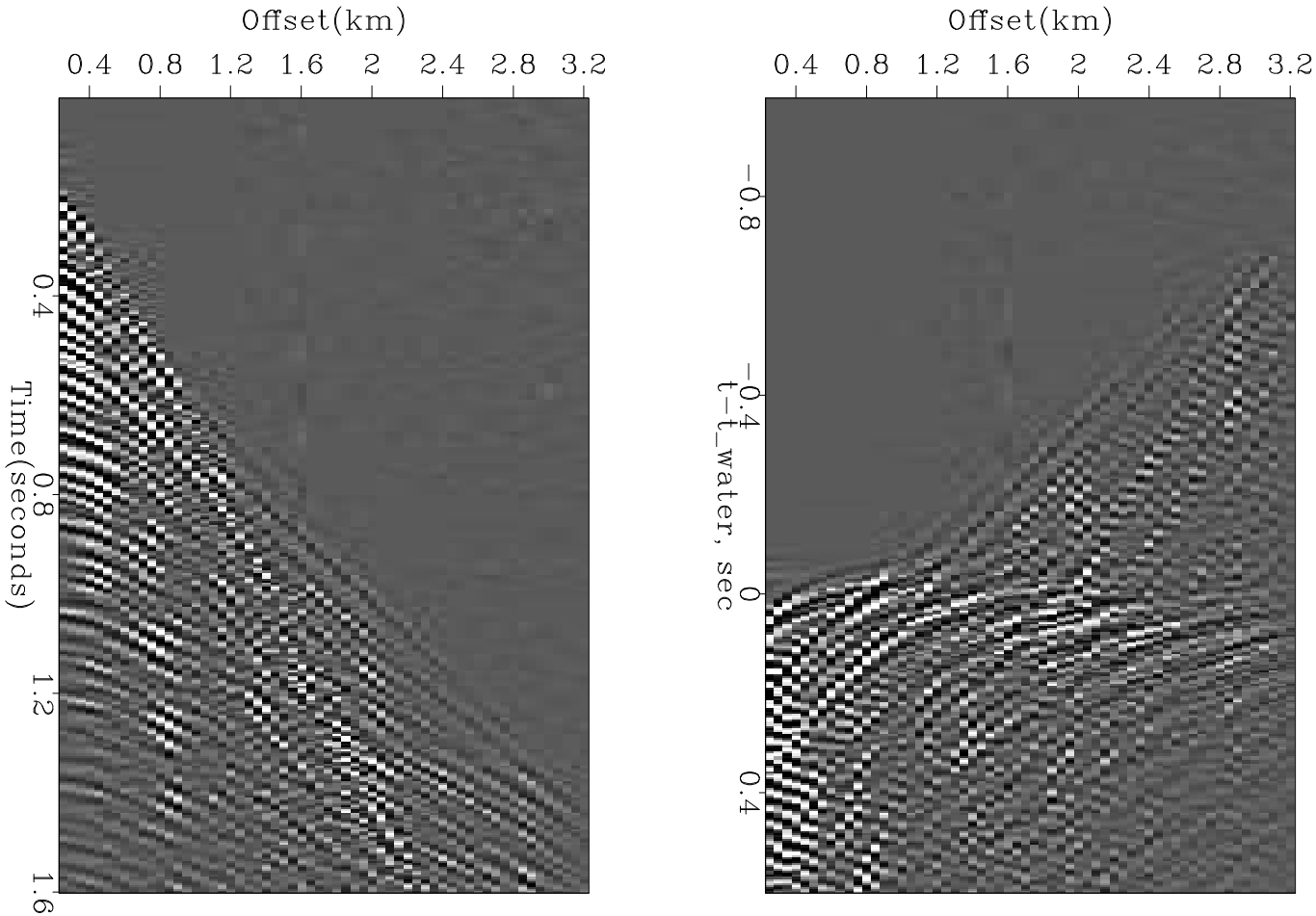


Figure 3.2: Land shot profile (Yilmaz and Cumro) #39 from the Middle East before (left) and after (right) linear moveout at water velocity. wvs-wzl.34 [ER]

cause it interferes with a wave of opposite polarity reflected from the water surface). At near offset slightly later than $\tau = 0$ is the water bottom reflection. At wide offset, the water bottom reflection is quickly followed by multiple reflections from the bottom. Critical angle reflection is defined as where the **head wave** comes tangent to the reflected wave. Before (above) $\tau = 0$ are the **head wave**s. There are two obvious slopes, hence two obvious layer interfaces. Figure 3.4 is much like Figure 3.3 but the water bottom is shallower.

Figure 3.5 shows data where the first arriving energy is not along a few straight line segments, but is along a curve. This means the velocity increases smoothly with depth as soft sediments compress.

3.2.2. LMO by nearest-neighbor interpolation

To do **linear moveout** (**LMO**) correction, we need to time-shift data. Shifting data requires us to interpolate it. The easiest interpolation method is the nearest-neighbor method. We begin with a signal given at times `t = t0+dt*(it-1)` where `it` is an integer. Then we can use equation (3.3), namely $\tau = t - px$. Given the location `tau` of the desired value we backsolve for an integer, say `itau`. In Fortran, conversion

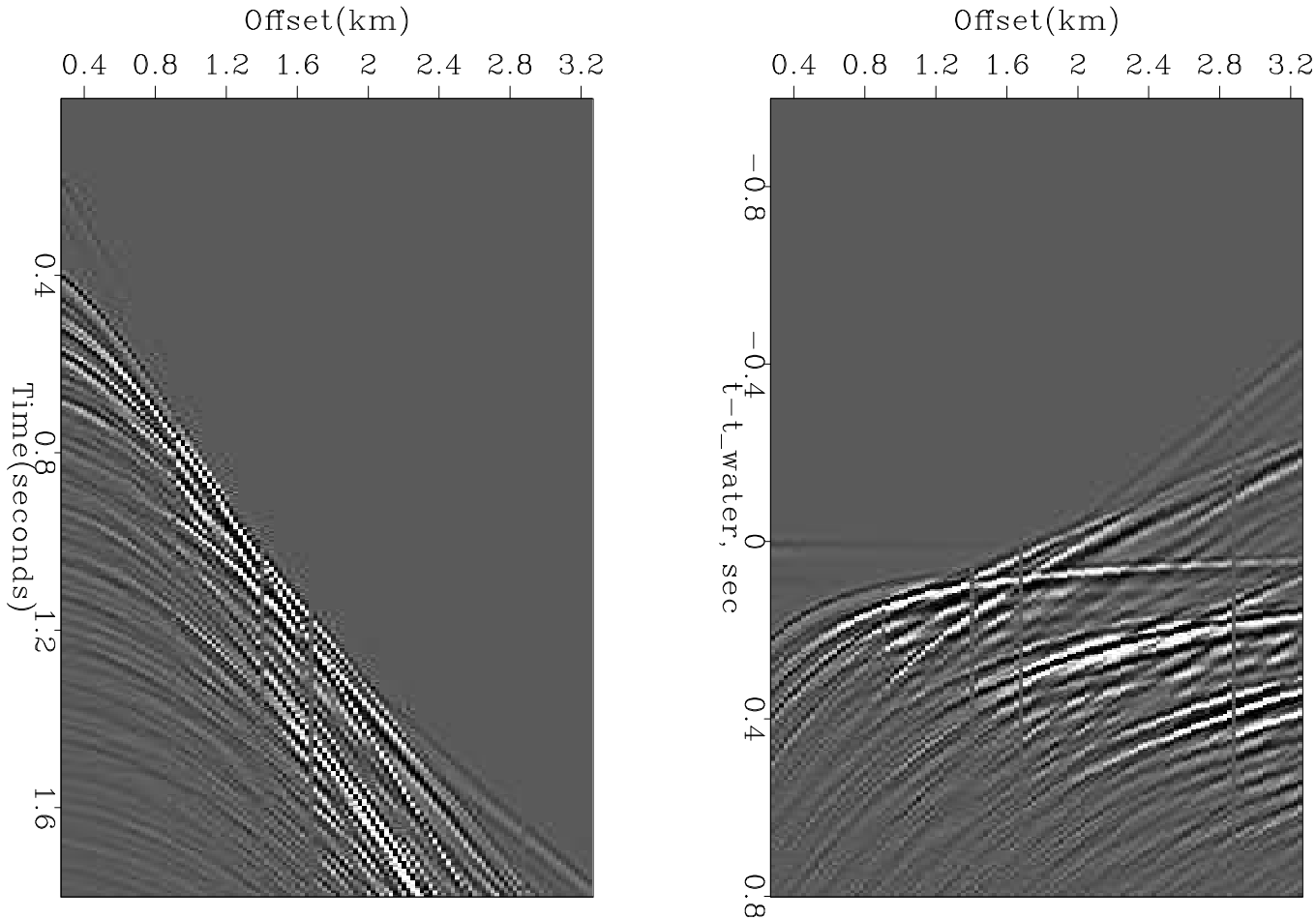


Figure 3.3: Marine shot profile (Yilmaz and Cumro) #20 from the Aleutian Islands. wvs-wzl.20 [ER]

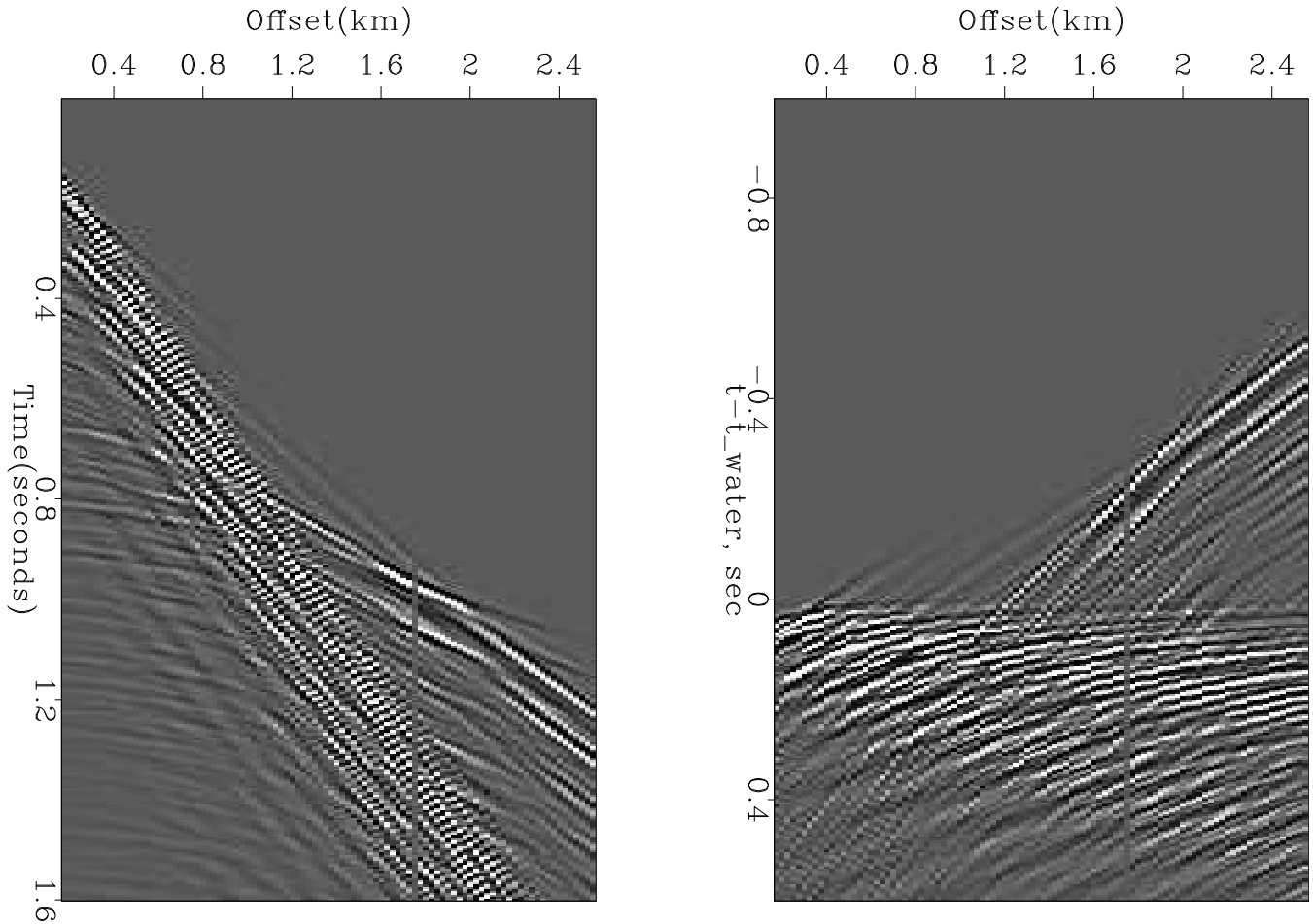


Figure 3.4: Marine shot profile (Yilmaz and Cumro) #32 from the North Sea. wvs-wzl.32 [ER]

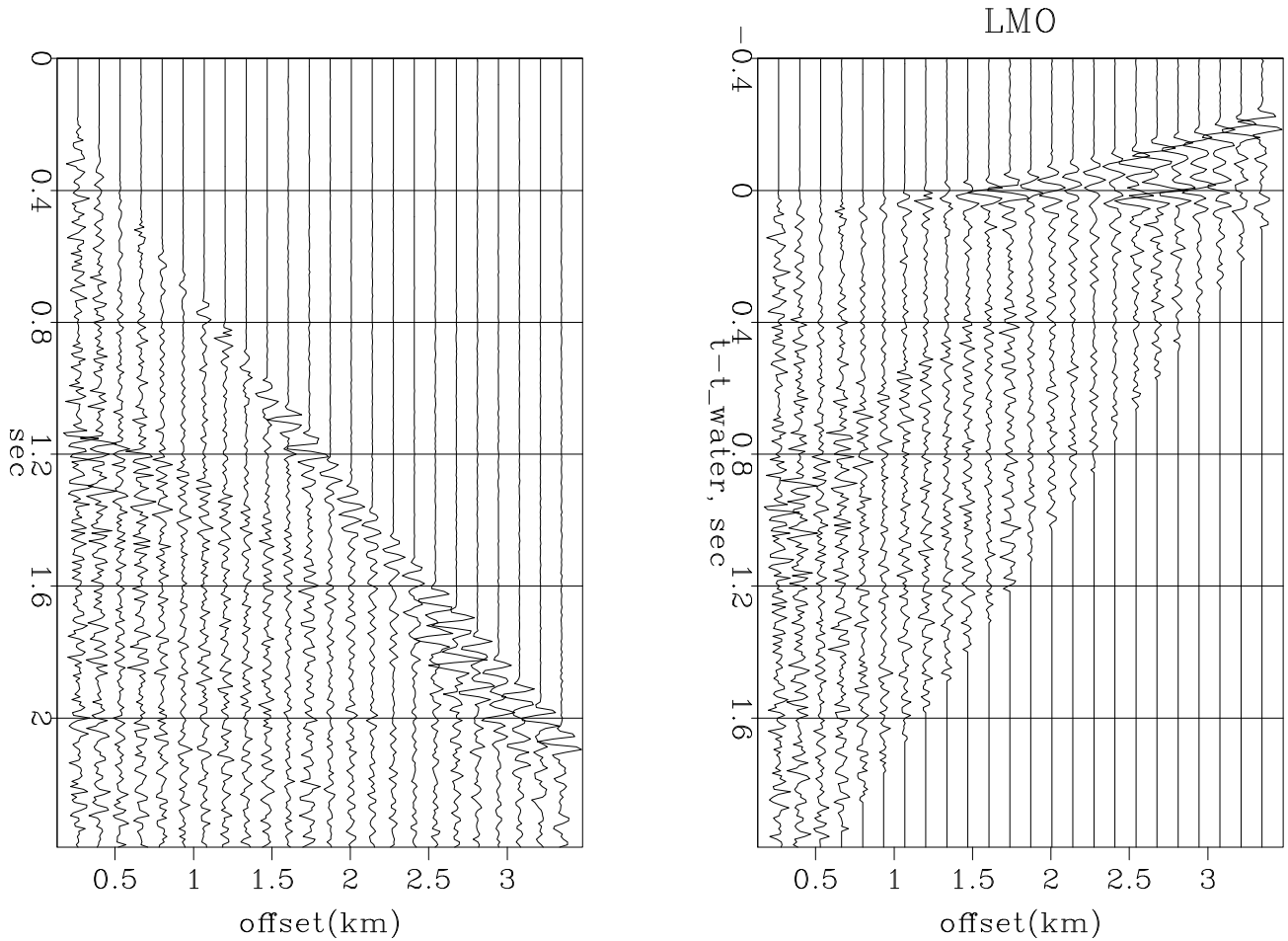


Figure 3.5: A common midpoint gather from the Gulf of Mexico before (left) and after (right) linear moveout at water velocity. Later I hope to estimate velocity with depth in shallow strata. Press button for **movie** over midpoint. wvs-wglmo [ER,M]

```
# linear moveout
#
subroutine lmo( adj,add, slow, tau0, t0,dt, x0,dx, modl,nt,nx,  data       )
integer         adj,add,                           nt,nx,       it,ix,iu
real t, x, tau,          slow, tau0, t0,dt, x0,dx, modl(nt,nx), data(nt,nx)
call adjnull(   adj,add,                           modl,nt*nx,  data,nt*nx)
do ix= 1, nx {  x= x0 + dx * (ix-1)
do it= 1, nt {  t= t0 + dt * (it-1)
                tau =  t - x * slow
                 iu = 1.5001 + (tau-tau0)/dt
                if( 0 < iu  &&  iu <= nt)
                        if( adj == 0 )
                                data(it,ix) = data(it,ix) + modl(iu,ix)
                        else
                                modl(iu,ix) = modl(iu,ix) + data(it,ix)
                }}
return; end
```

Back

of a real value to an integer is done by truncating the fractional part of the real value. To get rounding up as well as down, we add `0.5` before conversion to an integer, namely `itau=int(1.5+(tau-tau0)/dt)`. This gives the nearest neighbor. The way the program works is to identify two points, one in (t,x)-space and one in (τ,x)-space. Then the data value at one point in one space is carried to the other. The adjoint operation copies τ space back to t space. The subroutine used in the illustrations above is `lmo()` /prog:lmo with `adj=1`. lmo

Nearest neighbor rounding is crude but ordinarily very reliable. I discovered a very rare numerical roundoff problem peculiar to signal time-shifting, a problem which arises in the linear moveout application when the water velocity, about 1.48km/sec is approximated by 1.5=3/2. The problem arises only where the amount of the time shift is a numerical value (like 12.5000001 or 12.499999) and the fractional part should be exactly 1/2 but numerical rounding pushes it randomly in either direction. We would not care if an entire signal was shifted by either 12 units or by 13 units. What is troublesome, however, is if some random portion of the signal shifts 12 units while the rest of it shifts 13 units. Then the output signal has places which are empty while adjacent places contain the sum of two values. Linear moveout is the only application where I have ever encountered this difficulty. A

simple fix here was to modify the `lmo()` /prog:lmo subroutine changing the "1.5" to "1.5001". The problem disappears if we use a more accurate sound velocity or if we switch from nearest-neighbor interpolation to linear interpolation.

3.2.3. Muting

Surface waves are a mathematician's delight because they exhibit many complex phenomena. Since these waves are often extremely strong, and since the information they contain about the earth refers only to the shallowest layers, typically, considerable effort is applied to array design in field recording to suppress these waves. Nevertheless, in many areas of the earth, these pesky waves may totally dominate the data.

A simple method to suppress **ground roll** in data processing is to multiply a strip of data by a near-zero weight (the mute). To reduce truncation artifacts, the mute should taper smoothly to zero (or some small value). Because of the extreme variability from place to place on the earth's surface, there are many different philosophies about designing mutes. Some mute programs use a data dependent weighting function (such as automatic gain control). Subroutine `mutter()`

```
#  Data is weighted by sine squared inside a mute zone.
#  The weight is zero when      t <       x * slope0
#  The weight is one  when      t >  tp + x * slopep
#  Suggested defaults: slopep = slope0= 1./1.45 sec/km;  tp=.150 sec
#
subroutine mutter( tp, slope0,slopep, dt,dx, t0,x0, data,nt,nx)
integer it,ix,                                        nt,nx
real t,x, wt,      tp, slope0,slopep, dt,dx, t0,x0, data(nt,nx)
do ix=1,nx {    x= x0+(ix-1)*dx;        x = abs( x)
do it=1,nt {    t= t0+(it-1)*dt;
        if     ( t <      x * slope0)   wt = 0
        else if( t > tp + x * slopep)   wt = 1.
        else                            wt = sin(
                0.5 * 3.14159265 * (t-x*slope0)/(tp+x*(slopep-slope0))) ** 2
        data(it,ix) = data(it,ix) * wt
        }}
return; end
```

Back

/prog:mutter, however, operates on a simpler idea: the user supplies trajectories defining the mute zone. mutter

Figure 3.6 shows an example of use of the routine `mutter()` /prog:mutter on the shallow water data shown in Figure 3.5.

3.3. DIPPING WAVES

Above we considered waves going vertically and waves going horizontally. Now let us consider waves propagating at the intermediate angles. For the sake of definiteness, I have chosen to consider only downgoing waves in this section. We will later use the concepts developed here to handle both downgoing and upcoming waves.

3.3.1. Rays and fronts

It is natural to begin studies of waves with equations that describe plane waves in a medium of constant velocity. Figure 3.7 depicts a ray moving down into the earth at an angle θ from the vertical. Perpendicular to the ray is a wavefront. By elementary geometry the angle between the wave**front** and the earth's surface is

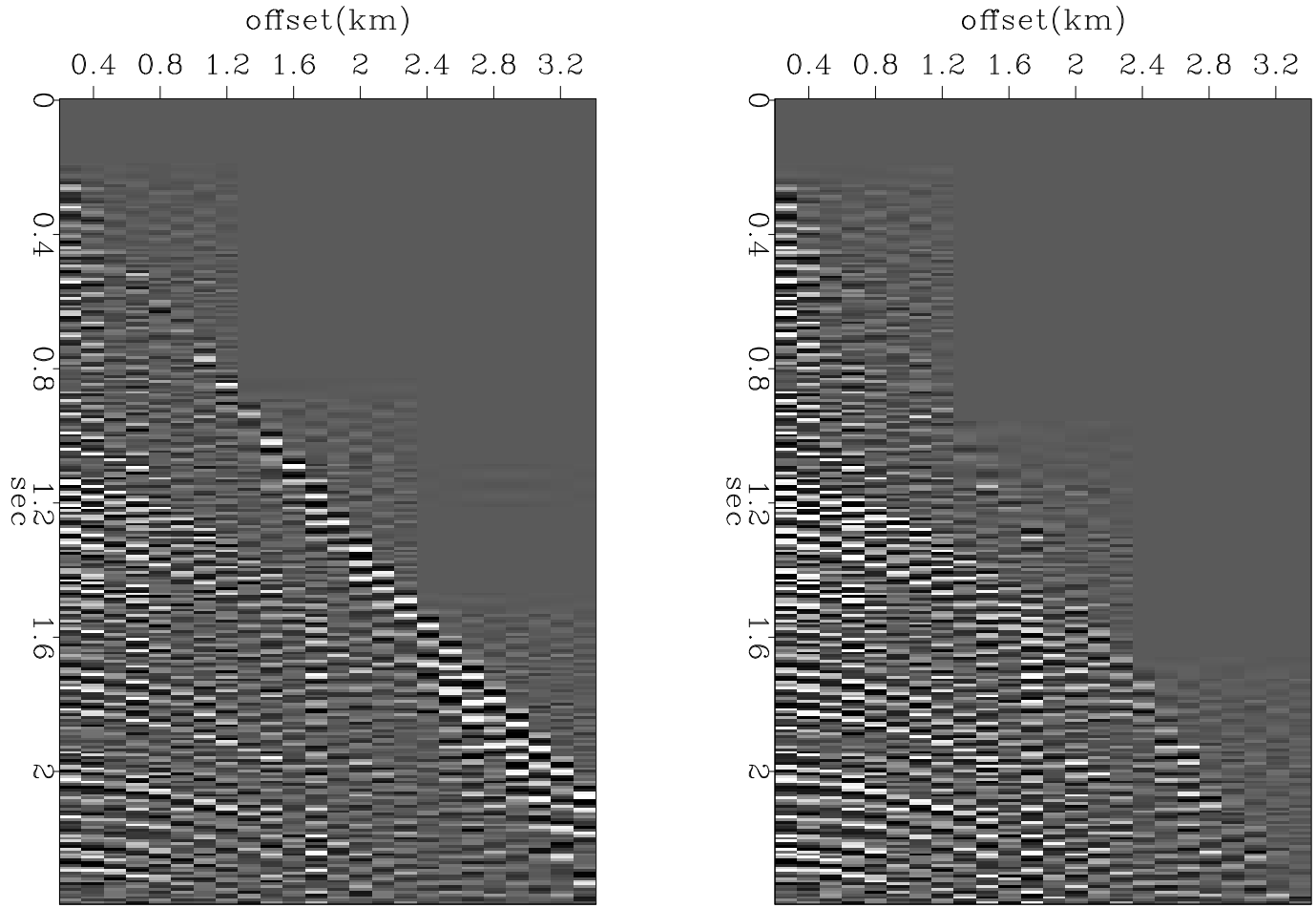


Figure 3.6: Jim's first gather before and after muting. wvs-mutter [ER]

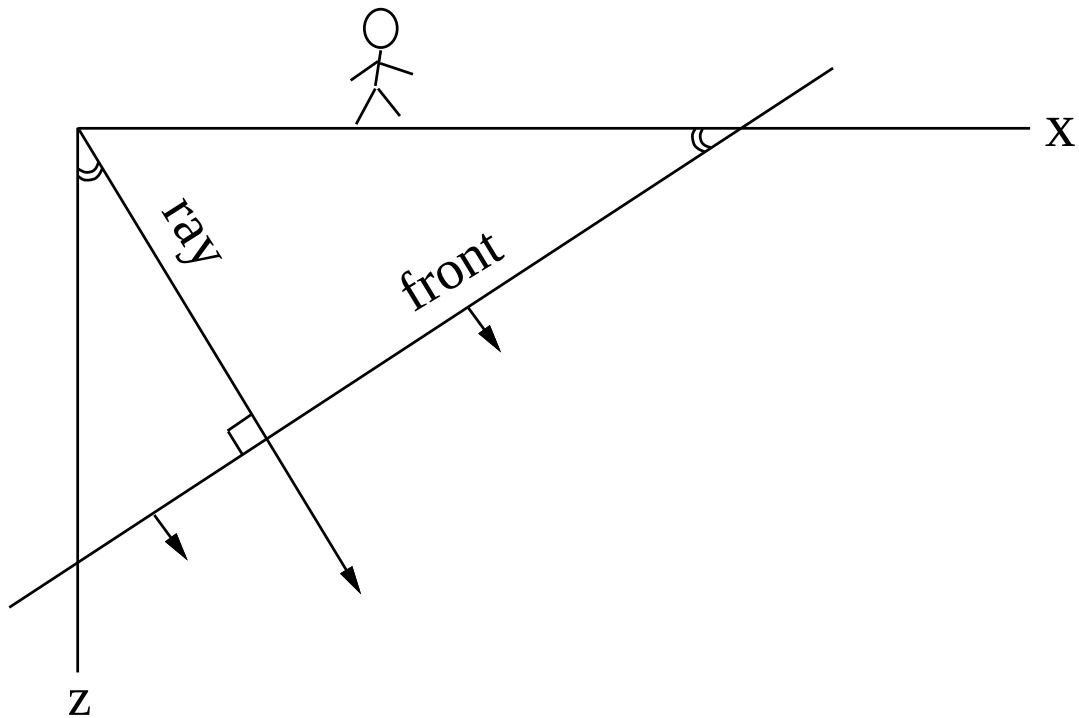


Figure 3.7: Downgoing ray and wavefront. wvs-front [NR]

also θ. The **ray** increases its length at a speed v. The speed that is observable on the earth's surface is the intercept of the wavefront with the earth's surface. This speed, namely $v/\sin\theta$, is faster than v. Likewise, the speed of the intercept of the wavefront and the vertical axis is $v/\cos\theta$. A mathematical expression for a straight line like that shown to be the wavefront in Figure 3.7 is

$$z \;=\; z_0 \;-\; x\;\tan\theta \tag{3.4}$$

In this expression z_0 is the intercept between the wavefront and the vertical axis. To make the intercept move downward, replace it by the appropriate velocity times time:

$$z \;=\; \frac{v\,t}{\cos\theta} \;-\; x\;\tan\theta \tag{3.5}$$

Solving for time gives

$$t(x,z) \;=\; \frac{z}{v}\;\cos\theta \;+\; \frac{x}{v}\;\sin\theta \tag{3.6}$$

Equation (3.6) tells the time that the wavefront will pass any particular location (x,z). The expression for a shifted waveform of arbitrary shape is $f(t-t_0)$. Using (3.6) to define the time shift t_0 gives an expression for a wavefield that is some

waveform moving on a **ray**.

$$\text{moving wavefield} \;=\; f\left(\, t \;-\; \frac{x}{v}\,\sin\theta \;-\; \frac{z}{v}\,\cos\theta \,\right) \tag{3.7}$$

3.3.2. Snell waves

In reflection seismic surveys the velocity contrast between shallowest and deepest reflectors ordinarily exceeds a factor of two. Thus depth variation of velocity is almost always included in the analysis of field data. Seismological theory needs to consider waves that are just like plane waves except that they bend to accommodate the velocity stratification $v(z)$. Figure 3.8 shows this in an idealized geometry: waves radiated from the horizontal flight of a supersonic airplane. The airplane passes location x at time $t_0(x)$ flying horizontally at a constant speed. Imagine an earth of horizontal plane layers. In this model there is nothing to distinguish any point on the x-axis from any other point on the x-axis. But the seismic velocity varies from layer to layer. There may be reflections, head waves, shear waves, converted waves, anisotropy, and multiple reflections. Whatever the picture is, it moves along with the airplane. A picture of the wavefronts near the airplane moves along with the airplane. The top of the picture and the bottom of the picture both

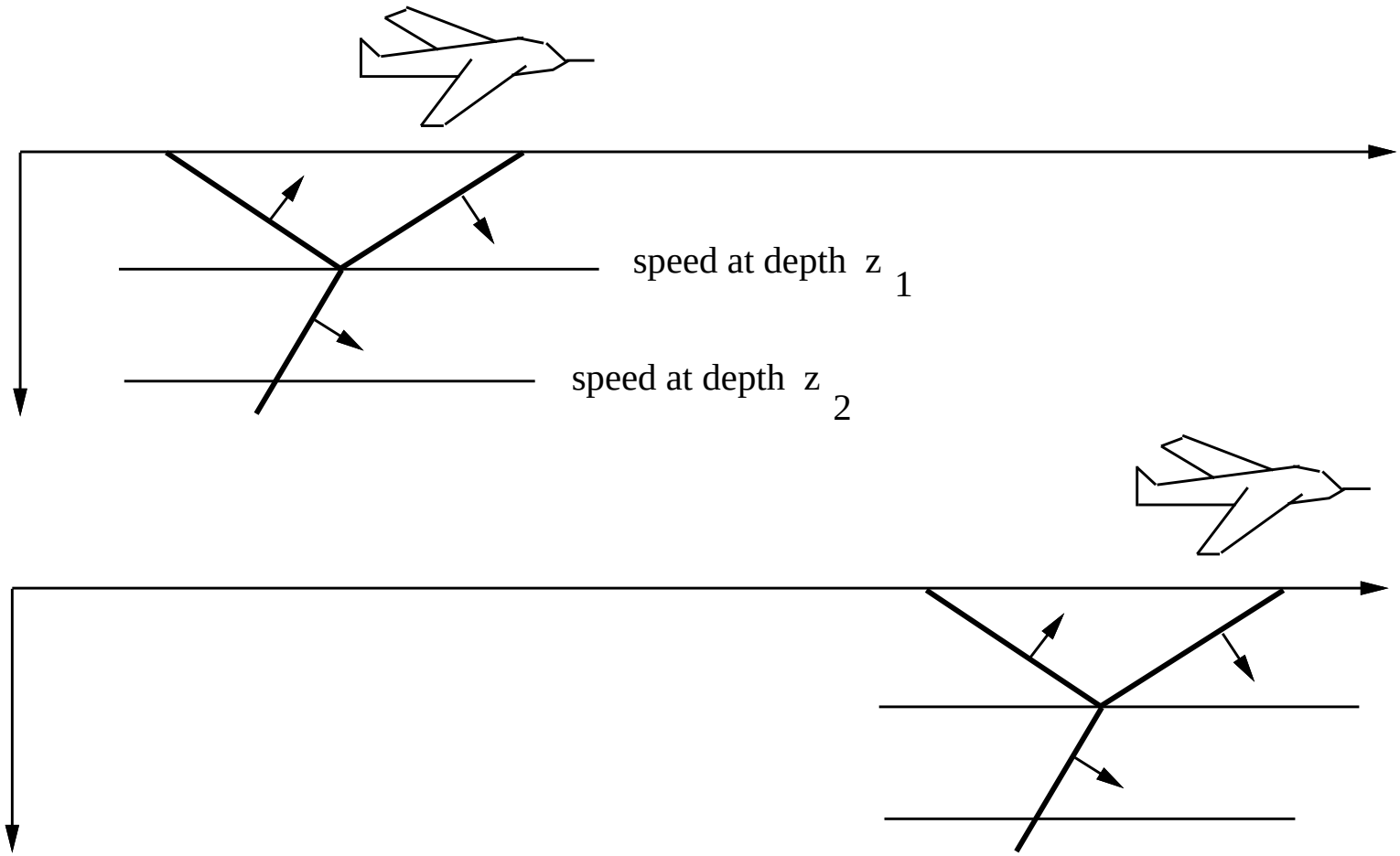


Figure 3.8: Fast airplane radiating a sound wave into the earth. From the figure you can deduce that the horizontal speed of the wavefront is the same at depth z_1 as it is at depth z_2. This leads (in isotropic media) to Snell's law. wvs-airplane [NR]

move laterally at the same speed even if the earth velocity increases with depth. If the top and bottom didn't go at the same speed, the picture would become distorted, contradicting the presumed symmetry of translation. This horizontal speed, or rather its inverse $\partial t_0/\partial x$, has several names. In practical work it is called the ***stepout***. In theoretical work it is called the ***ray parameter***. It is very important to note that $\partial t_0/\partial x$ does not change with depth, even though the seismic velocity does change with depth. In a constant-velocity medium, the angle of a wave does not change with depth. In a stratified medium, $\partial t_0/\partial x$ does not change with depth.

Figure 3.9 illustrates the differential geometry of the wave. Notice that triangles have their hypotenuse on the x-axis and the z-axis but not along the ray. That's because this figure refers to wave fronts. (If you were thinking the hypotenuse would measure $v\Delta t$, it could be you were thinking of the tip of a ray and its projection onto the x and z axes.) The diagram shows that

$$\frac{\partial t_0}{\partial x} = \frac{\sin\theta}{v} \tag{3.8}$$

$$\frac{\partial t_0}{\partial z} = \frac{\cos\theta}{v} \tag{3.9}$$

These two equations define two (inverse) speeds. The first is a horizontal speed,

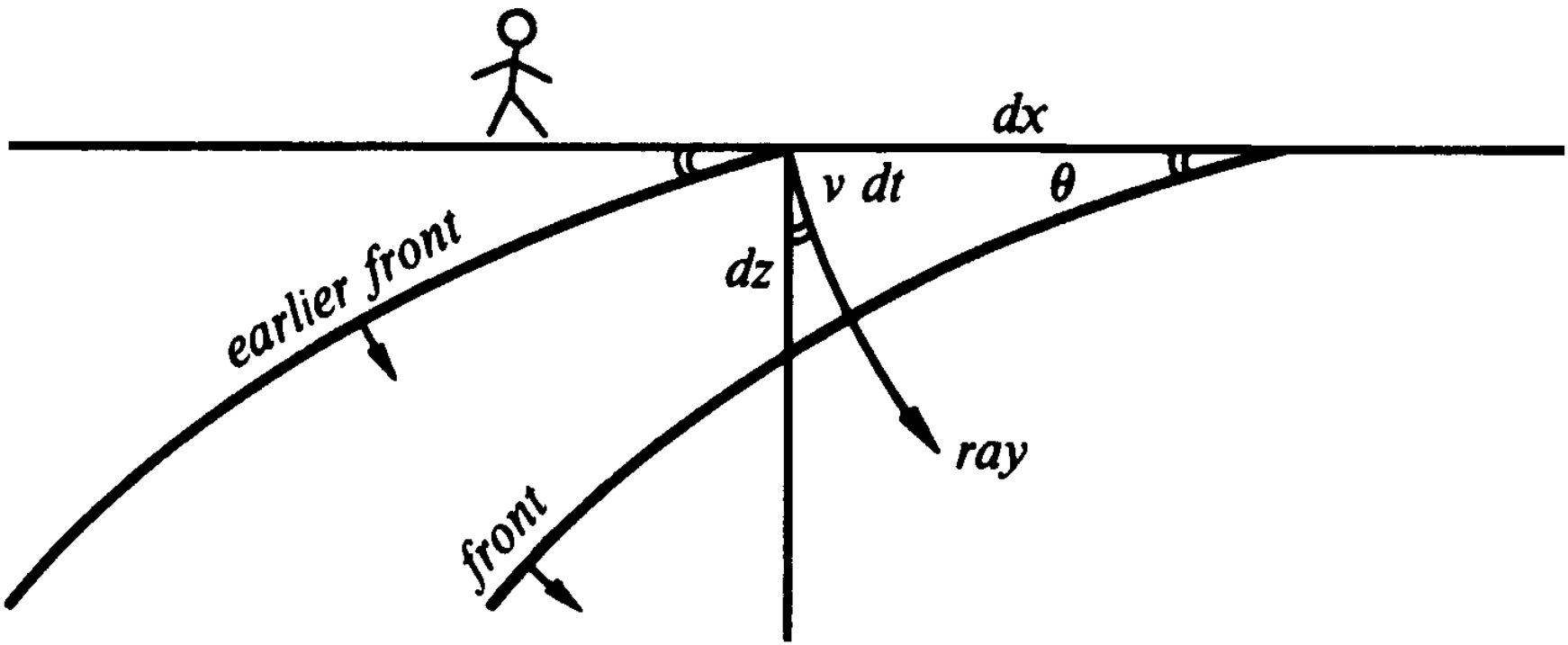


Figure 3.9: Downgoing fronts and rays in stratified medium $v(z)$. The wavefronts are horizontal translations of one another. wvs-frontz [NR]

measured along the earth's surface, called the *horizontal* ***phase velocity***. The second is a vertical speed, measurable in a borehole, called the *vertical phase velocity*. Notice that both these speeds *exceed* the velocity v of wave propagation in the medium. Projection of wave *fronts* onto coordinate axes gives speeds larger than v, whereas projection of *rays* onto coordinate axes gives speeds smaller than v. The inverse of the phase velocities is called the ***stepout*** or the ***slowness***.

Snell's law relates the angle of a wave in one layer with the angle in another. The constancy of equation (3.8) in depth is really just the statement of Snell's law. Indeed, we have just derived Snell's law. All waves in seismology propagate in a velocity-stratified medium. So they cannot be called plane waves. But we need a name for waves that are near to plane waves. A ***Snell wave*** will be defined to be the generalization of a plane wave to a stratified medium $v(z)$. A plane wave that happens to enter a medium of depth-variable velocity $v(z)$ gets changed into a Snell wave. While a plane wave has an angle of propagation, a Snell wave has instead a ***Snell parameter*** $p = \partial t_0 / \partial x$.

It is noteworthy that Snell's parameter $p = \partial t_0 / \partial x$ is directly observable at the surface, whereas neither v nor θ is directly observable. Since $p = \partial t_0 / \partial x$ is not only observable, but constant in depth, it is customary to use it to eliminate θ from

equations (3.8) and (3.9):

$$\frac{\partial t_0}{\partial x} = \frac{\sin\theta}{v} = p \tag{3.10}$$

$$\frac{\partial t_0}{\partial z} = \frac{\cos\theta}{v} = \sqrt{\frac{1}{v(z)^2} - p^2} \tag{3.11}$$

3.3.3. Evanescent waves

Suppose the velocity increases to infinity at infinite depth. Then equation (3.11) tells us that something strange happens when we reach the depth for which p^2 equals $1/v(z)^2$. That is the depth at which the ray turns horizontal. We will see in a later chapter that below this critical depth the seismic wavefield damps exponentially with increasing depth. Such waves are called **evanescent**. For a physical example of an evanescent wave, forget the airplane and think about a moving bicycle. For a bicyclist, the slowness p is so large that it dominates $1/v(z)^2$ for all earth materials. The bicyclist does not radiate a wave, but produces a ground deformation that decreases exponentially into the earth. To radiate a wave, a source must move faster than the material velocity.

3.3.4. Solution to kinematic equations

The above differential equations will often reoccur in later analysis, so they are very important. Interestingly, these differential equations have a simple solution. Taking the Snell wave to go through the origin at time zero, an expression for the arrival time of the Snell wave at any other location is given by

$$t_0(x,z) = \frac{\sin\theta}{v}\,x + \int_0^z \frac{\cos\theta}{v}\,dz \tag{3.12}$$

$$t_0(x,z) = p\,x + \int_0^z \sqrt{\frac{1}{v(z)^2} - p^2}\;dz \tag{3.13}$$

The validity of equations (3.12) and (3.13) is readily checked by computing $\partial t_0/\partial x$ and $\partial t_0/\partial z$, then comparing with (3.10) and (3.11).

An arbitrary waveform $f(t)$ may be carried by the Snell wave. Use (3.12) and (3.13) to *define* the time t_0 for a delayed wave $f[t - t_0(x,z)]$ at the location (x,z).

$$\text{SnellWave}(t,x,z) = f\left(t - p\,x - \int_0^z \sqrt{\frac{1}{v(z)^2} - p^2}\;dz\right) \tag{3.14}$$

Equation (3.14) carries an arbitrary signal throughout the whole medium. Interestingly, it does not agree with wave propagation theory or real life because equation (3.14) does not correctly account for amplitude changes that result from velocity changes and reflections. Thus it is said that Equation (3.14) is "kinematically" correct but "dynamically" incorrect. It happens that most industrial data processing only requires things to be kinematically correct, so this expression is a usable one.

3.4. CURVED WAVEFRONTS

The simplest waves are expanding circles. An equation for a circle expanding with velocity v is

$$v^2 t^2 \quad = \quad x^2 \; + \; z^2 \tag{3.15}$$

Considering t to be a constant, i.e. taking a snapshot, equation (3.15) is that of a circle. Considering z to be a constant, it is an equation in the (x,t)-plane for a hyperbola. Considered in the (t,x,z)-volume, equation (3.15) is that of a cone. Slices at various values of t show circles of various sizes. Slices of various values of z show various hyperbolas.

Converting equation (3.15) to traveltime depth τ we get

$$v^2 t^2 = z^2 + x^2 \tag{3.16}$$

$$t^2 = \tau^2 + \frac{x^2}{v^2} \tag{3.17}$$

The earth's velocity typically increases by more than a factor of two between the earth's surface, and reflectors of interest. Thus we might expect that equation (3.17) would have little practical use. Luckily, this simple equation will solve many problems for us if we know how to interpret the velocity as an average velocity.

3.4.1. Root-mean-square velocity

When a ray travels in a depth-stratified medium, Snell's parameter $p = v^{-1} \sin\theta$ is constant along the ray. If the ray emerges at the surface, we can measure the distance x that it has traveled, the time t it took, and its apparent speed $dx/dt = 1/p$. A well-known estimate $\hat{v}$ for the earth velocity contains this apparent speed.

$$\hat{v} = \sqrt{\frac{x}{t}\frac{dx}{dt}} \tag{3.18}$$

To see where this velocity estimate comes from, first notice that the stratified velocity $v(z)$ can be parameterized as a function of time and take-off angle of a ray from the surface.

$$v(z) \quad = \quad v(x,z) \quad = \quad v'(p,t) \tag{3.19}$$

The x coordinate of the tip of a ray with Snell parameter p is the horizontal component of velocity integrated over time.

$$x(p,t) \quad = \quad \int_0^t v'(p,t)\, \sin\theta(p,t)\, dt \quad = \quad p \int_0^t v'(p,t)^2\, dt \tag{3.20}$$

Inserting this into equation (3.18) and canceling $p = dt/dx$ we have

$$\hat{v} \quad = \quad v_{\text{RMS}} \quad = \quad \sqrt{\frac{1}{t} \int_0^t v'(p,t)^2\, dt} \tag{3.21}$$

which shows that the observed velocity is the "root-mean-square" velocity.

When velocity varies with depth, the traveltime curve is only roughly a hyperbola. If we break the event into many short line segments where the i-th segment has a slope p_i and a midpoint (t_i, x_i) each segment gives a different $\hat{v}(p_i, t_i)$ and we have the unwelcome chore of assembling the best model. Instead, we can fit the

observational data to the best fitting hyperbola using a different velocity hyperbola for each apex, in other words, find $V(\tau)$ so this equation will best flatten the data in (τ, x)-space.

$$t^2 \quad = \quad \tau^2 + x^2/V(\tau)^2 \tag{3.22}$$

Differentiate with respect to x at constant τ getting

$$2t\,dt/dx \quad = \quad 2x/V(\tau)^2 \tag{3.23}$$

which confirms that the observed velocity $\hat{v}$ in equation (3.18), is the same as $V(\tau)$ no matter where you measure $\hat{v}$ on a hyperbola.

3.4.2. Layered media

From the assumption that experimental data can be fit to hyperbolas (each with a different velocity and each with a different apex τ) let us next see how we can fit an earth model of layers, each with a constant velocity. Consider the horizontal reflector overlain by a stratified **interval velocity** $v(z)$ shown in Figure 3.10. The separation between the source and geophone, also called the offset, is $2h$ and the total travel time is t. Travel times are not be precisely hyperbolic, but it is common

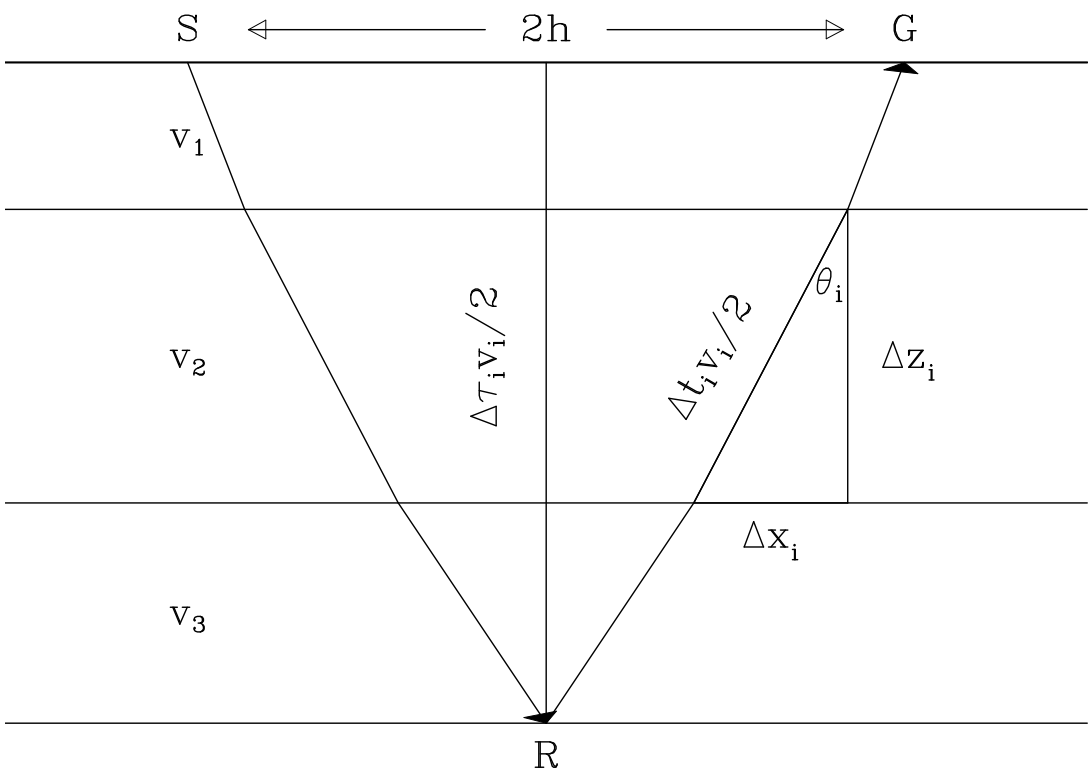


Figure 3.10: Raypath diagram for normal moveout in a stratified earth. wvs-stratrms [ER]

practice to find the best fitting hyperbolas, thus finding the function $V^2(\tau)$.

$$t^2 \quad = \quad \tau^2 + \frac{4h^2}{V^2(\tau)} \tag{3.24}$$

where τ is the zero-offset two-way traveltime.

An example of using equation (3.24) to stretch t into τ is shown in Figure 3.11. (The programs that find the required $V(\tau)$ and do the stretching are coming up in chapter 4.)

Equation (3.21) shows that $V(\tau)$ is the "root-mean-square" or "RMS" velocity defined by an average of v^2 over the layers. Expressing it for a small number of layers we get

$$V^2(\tau) \quad = \quad \frac{1}{\tau} \sum_i v_i^2 \Delta\tau_i \tag{3.25}$$

where the zero-offset traveltime τ is a sum over the layers:

$$\tau \quad = \quad \sum_i \Delta\tau_i \tag{3.26}$$

The two-way vertical travel time τ_i in the ith layer is related to the thickness Δz_i

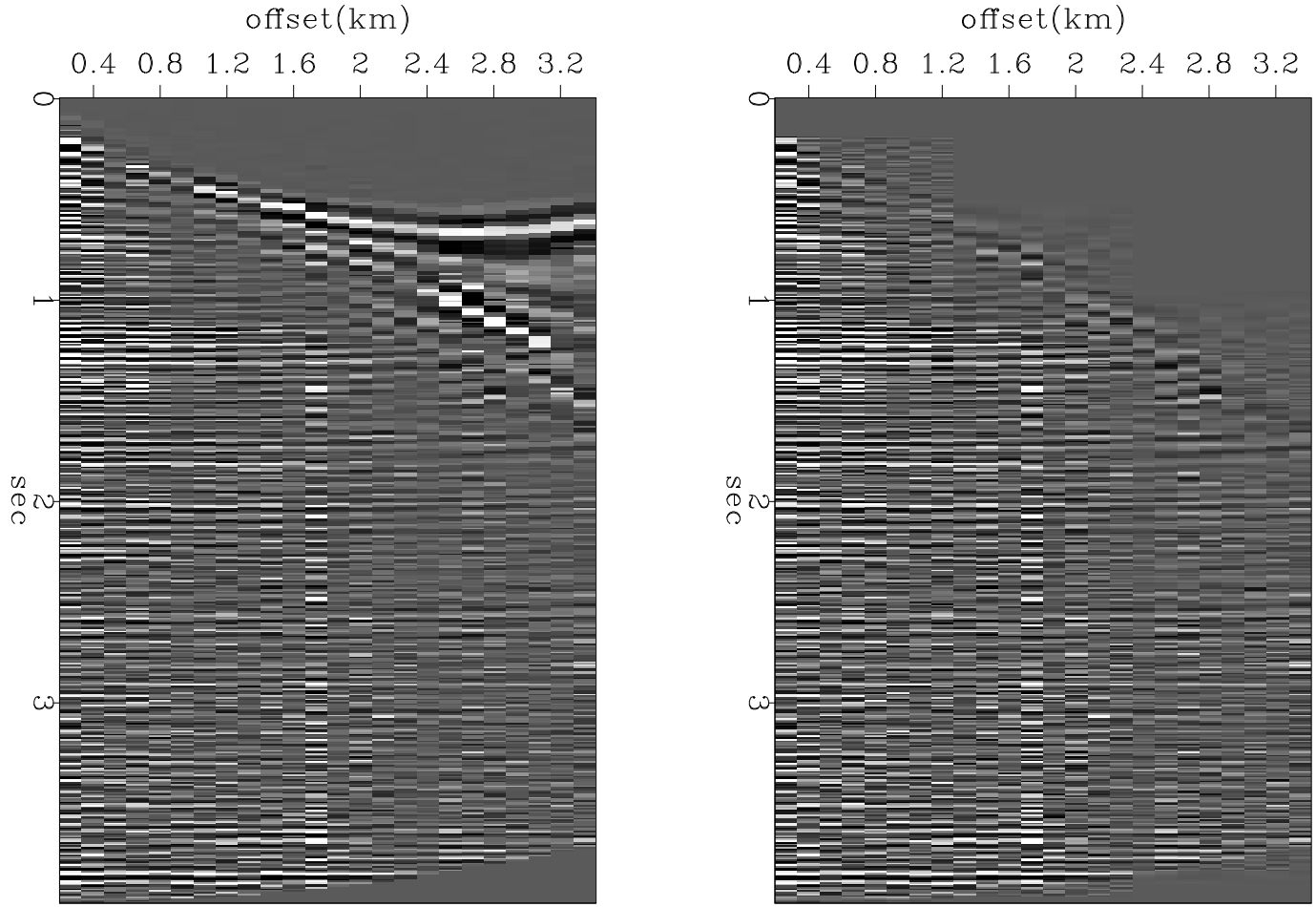


Figure 3.11: If you are lucky and get a good velocity, when you do NMO, everything turns out flat. Shown with and without mute. wvs-nmogath [ER]

and the velocity v_i by

$$\Delta\tau_i \quad = \quad \frac{2\ \Delta z_i}{v_i}\ . \tag{3.27}$$

Next we examine an important practical calculation, getting interval velocities from measured RMS velocities: Define in layer i, the interval velocity v_i and the two-way vertical travel time $\Delta\tau_i$. Define the RMS velocity of a reflection from the bottom of the i-th layer to be V_i. Equation (3.25) tells us that for reflections from the bottom of the first, second, and third layers we have

$$V_1^2 \quad = \quad \frac{v_1^2\Delta\tau_1}{\Delta\tau_1} \tag{3.28}$$

$$V_2^2 \quad = \quad \frac{v_1^2\Delta\tau_1 + v_2^2\Delta\tau_2}{\Delta\tau_1 + \Delta\tau_2} \tag{3.29}$$

$$V_3^2 \quad = \quad \frac{v_1^2\Delta\tau_1 + v_2^2\Delta\tau_2 + v_3^2\Delta\tau_3}{\Delta\tau_1 + \Delta\tau_2 + \Delta\tau_3} \tag{3.30}$$

Normally it is easy to measure the times of the three hyperbola tops, $\Delta\tau_1$, $\Delta\tau_1 + \Delta\tau_2$ and $\Delta\tau_1 + \Delta\tau_2 + \Delta\tau_3$. Using methods in chapter 4 we can measure the RMS

velocities V_2 and V_3. With these we can solve for the interval velocity v_3 in the third layer. Rearrange (3.30) and (3.29) to get

$$(\Delta\tau_1 + \Delta\tau_2 + \Delta\tau_3)V_3^2 = v_1^2\Delta\tau_1 + v_2^2\Delta\tau_2 + v_3^2\Delta\tau_3 \quad (3.31)$$

$$(\Delta\tau_1 + \Delta\tau_2)V_2^2 = v_1^2\Delta\tau_1 + v_2^2\Delta\tau_2 \quad (3.32)$$

and subtract getting the squared interval velocity v_3^2

$$v_3^2 = \frac{(\Delta\tau_1 + \Delta\tau_2 + \Delta\tau_3)V_3^2 - (\Delta\tau_1 + \Delta\tau_2)V_2^2}{\Delta\tau_3} \quad (3.33)$$

For any real earth model we would not like an imaginary velocity which is what could happen if the squared velocity in (3.33) happened to be negative. You see that this means that the RMS velocity we estimate for the third layer cannot be too much smaller than the one we estimate for the second layer.

3.4.3. Nonhyperbolic curves

Occasionally data does not fit a hyperbolic curve very well. Two other simple fitting functions are

$$t^2 = \tau^2 + \frac{x^2}{v^2} + x^4 \times \text{parameter} \tag{3.34}$$

$$(t - t_0)^2 = (\tau - t_0)^2 + \frac{x^2}{v^2} \tag{3.35}$$

Equation (3.34) has an extra adjustable parameter of no simple interpretation other than the beginning of a power series in x^2. I prefer Equation (3.35) where the extra adjustable parameter is a time shift t_0 which has a simple interpretation, namely, a time shift such as would result from a near-surface low velocity layer. In other words, a datum correction.

3.4.4. Velocity increasing linearly with depth

Theoreticians are delighted by velocity increasing linearly with depth because it happens that many equations work out in closed form. For example, rays travel in

circles. We will need convenient expressions for velocity as a function of traveltime depth and RMS velocity as a function of traveltime depth. Let us get them. We take the **interval velocity** $v(z)$ increasing linearly with depth:

$$v(z) \quad = \quad v_0 + \alpha z \tag{3.36}$$

This presumption can also be written as a differential equation:

$$\frac{dv}{dz} \quad = \quad \alpha. \tag{3.37}$$

The relationship between z and vertical two-way traveltime $\tau(z)$ (see equation (3.27)) is also given by a differential equation:

$$\frac{d\tau}{dz} \quad = \quad \frac{2}{v(z)}. \tag{3.38}$$

Letting $v(\tau) = v(z(\tau))$ and applying the chain rule gives the differential equation for $v(\tau)$:

$$\frac{dv}{dz}\frac{dz}{d\tau} \quad = \quad \frac{dv}{d\tau} \quad = \quad \frac{v\alpha}{2}, \tag{3.39}$$

whose solution gives us the desired expression for **interval velocity** as a function of traveltime depth.

$$v(\tau) \quad = \quad v_0 \, e^{\alpha\tau/2}. \tag{3.40}$$

3.4.5. Prior RMS velocity

Substituting the theoretical interval velocity $v(\tau)$ from equation (3.40) into the definition of RMS velocity $V(\tau)$ (equation (3.25)) yields:

$$\tau \; V^2(\tau) \quad = \quad \int_0^\tau v^2(\tau') \, d\tau' \tag{3.41}$$

$$= \quad v_0^2 \; \frac{e^{\alpha\tau} - 1}{\alpha}. \tag{3.42}$$

Thus the desired expression for RMS velocity as a function of traveltime depth is:

$$V(\tau) \quad = \quad v_0 \sqrt{\frac{e^{\alpha\tau} - 1}{\alpha\tau}} \tag{3.43}$$

For small values of $\alpha\tau$, this can be approximated as

$$V(\tau) \quad \approx \quad v_0 \sqrt{1+\alpha\tau/2}. \tag{3.44}$$

Chapter 4

Moveout, velocity, and stacking

In this chapter we handle data as though the earth had no dipping reflectors. The earth model is one of stratified layers with velocity a (generally increasing) function of depth. We consider reflections from layers, which we process by normal moveout

correction (NMO). The NMO operation is an interesting example of many general principles of linear operators and numerical analysis. Finally, using NMO, we estimate the earth's velocity with depth and we stack some data, getting a picture of an earth with dipping layers. This irony, that techniques developed for a stratified earth can give reasonable images of non-stratified reflectors, is one of the "lucky breaks" of seismic processing. We will explore the limitations of this phenomenon in the chapter on dip-moveout.

First, a few words about informal language. The inverse to velocity arises more frequently in seismology than the velocity itself. This inverse is called the "slowness." In common speech, however, the word "velocity" is a catch-all, so what is called a "velocity analysis" might actually be a plane of slowness versus time.

4.1. INTERPOLATION AS A MATRIX

Here we see how general principles of linear operators are exemplified by linear interpolation. Because the subject matter is so simple and intuitive, it is ideal to exemplify abstract mathematical concepts that apply to all linear operators.

Let an integer k range along a survey line, and let data values x_k be packed into

a vector $\mathbf{x}$. (Each data point x_k could also be a seismogram.) Next we resample the data more densely, say from 4 to 6 points. For illustration, I follow a crude **nearest-neighbor interpolation** scheme by sprinkling ones along the diagonal of a rectangular matrix that is

$$\mathbf{y} \quad = \quad \mathbf{B}\mathbf{x} \tag{4.1}$$

where

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{4.2}$$

The interpolated data is simply $\mathbf{y} = (x_1, x_2, x_2, x_3, x_4, x_4)$. The matrix multiplication (4.2) would not be done in practice. Instead there would be a loop running over the space of the outputs $\mathbf{y}$ that picked up values from the input.

4.1.1. Looping over input space

The obvious way to program a deformation is to take each point from the *input* space and find where it goes on the output space. Naturally, many points could land in the same place, and then only the last would be seen. Alternately, we could first erase the output space, then add in points, and finally divide by the number of points that ended up in each place. The biggest aggravation is that some places could end up with no points. This happens where the transformation **stretch**es. There we need to decide whether to interpolate the missing points, or simply low-pass filter the output.

4.1.2. Looping over output space

The alternate method that is usually preferable to looping over input space is that our program have a loop over the space of the *outputs*, and that each output find its input. The matrix multiply of (4.2) can be interpreted this way. Where the transformation **shrink**s is a small problem. In that area many points in the input space are ignored, where perhaps they should somehow be averaged with their neighbors. This is not a serious problem unless we are contemplating iterative transformations back and

forth between the spaces.

We will now address interesting questions about the reversibility of these deformation transforms.

4.1.3. Formal inversion

We have thought of equation (4.1) as a formula for finding **y** from **x**. Now consider the opposite problem, finding **x** from **y**. Begin by multiplying equation (4.2) by the **transpose matrix** to define a new quantity $\tilde{\mathbf{x}}$:

$$\begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \\ \tilde{x}_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} \tag{4.3}$$

$\tilde{\mathbf{x}}$ is not the same as **x**, but these two vectors have the same dimensionality and in many applications it may happen that $\tilde{\mathbf{x}}$ is a good approximation to **x**. In general, $\tilde{\mathbf{x}}$

may be called an "image" of $\mathbf{x}$. Finding the image is the first step of finding $\mathbf{x}$ itself. Formally, the problem is

$$\mathbf{y} \quad = \quad \mathbf{B}\mathbf{x} \tag{4.4}$$

And the formal solution to the problem is

$$\mathbf{x} \quad = \quad (\mathbf{B}'\mathbf{B})^{-1}\mathbf{B}'\mathbf{y} \tag{4.5}$$

Formally, we verify this solution by substituting (4.4) into (4.5).

$$\mathbf{x} \quad = \quad (\mathbf{B}'\mathbf{B})^{-1}(\mathbf{B}'\mathbf{B})\mathbf{x} \quad = \quad \mathbf{I}\mathbf{x} \quad = \quad \mathbf{x} \tag{4.6}$$

In applications, the possible nonexistence of an inverse for the matrix $(\mathbf{B}'\mathbf{B})$ is always a topic for discussion. For now we simply examine this matrix for the inter-

polation problem. We see that it is diagonal:

$$\mathbf{B}'\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \tag{4.7}$$

So, $\tilde{\mathbf{x}}_1 = \mathbf{x}_1$; but $\tilde{\mathbf{x}}_2 = 2\mathbf{x}_2$. To recover the original data, we need to divide $\tilde{\mathbf{x}}$ by the diagonal matrix $\mathbf{B}'\mathbf{B}$. Thus, matrix inversion is easy here.

Equation (4.5) has an illustrious reputation, which arises in the context of "least squares." **Least squares** is a general method for solving sets of equations that have more equations than unknowns.

Recovering $\mathbf{x}$ from $\mathbf{y}$ using equation (4.5) presumes the existence of the inverse of $\mathbf{B}'\mathbf{B}$. As you might expect, this matrix is nonsingular when $\mathbf{B}$ *stretches* the data, because then a few data values are distributed among a greater number of locations. Where the transformation *squeezes* the data, $\mathbf{B}'\mathbf{B}$ must become singular, since returning uniquely to the uncompressed condition is impossible.

We can now understand why an adjoint operator is often an approximate inverse. This equivalency happens in proportion to the nearness of the matrix $\mathbf{B}'\mathbf{B}$ to an identity matrix. The interpolation example we have just examined is one in which $\mathbf{B}'\mathbf{B}$ differs from an identity matrix merely by a scaling.

4.2. THE NORMAL MOVEOUT MAPPING

Recall the traveltime equation (3.17).

$$v^2 t^2 = z^2 + x^2 \tag{4.8}$$

$$t^2 = \tau^2 + \frac{x^2}{v^2} \tag{4.9}$$

where τ is traveltime depth. This equation gives either time from a surface source to a receiver at depth τ, or it gives time to a surface receiver from an image source at depth τ.

A seismic **trace** is a signal $d(t)$ recorded at some constant x. We can convert the trace to a "vertical propagation" signal $m(\tau) = d(t)$ by stretching t to τ. This process is called "**normal moveout** correction" (NMO). Typically we have many traces at

different x distances each of which theoretically produces the same hypothetical zero-offset trace. Figure 4.1 shows a marine shot profile before and after NMO correction at the water velocity. You can notice that the wave packet reflected from the ocean bottom is approximately a constant width on the raw data. After NMO, however, this waveform broadens considerably—a phenomenon known as “NMO stretch."

The NMO transformation **N** is representable as a square matrix. The matrix **N** is a (τ, t)-plane containing all zeros except an interpolation operator centered along the hyperbola. The dots in the matrix below are zeros. The input signal d_t is put into the vector **d**. The output vector **m**—i.e., the NMO’ed signal—is simply $(d_6, d_6, d_6, d_7, d_7, d_8, d_8, d_9, d_{10}, 0)$. In real life examples such as Figure 4.1 the

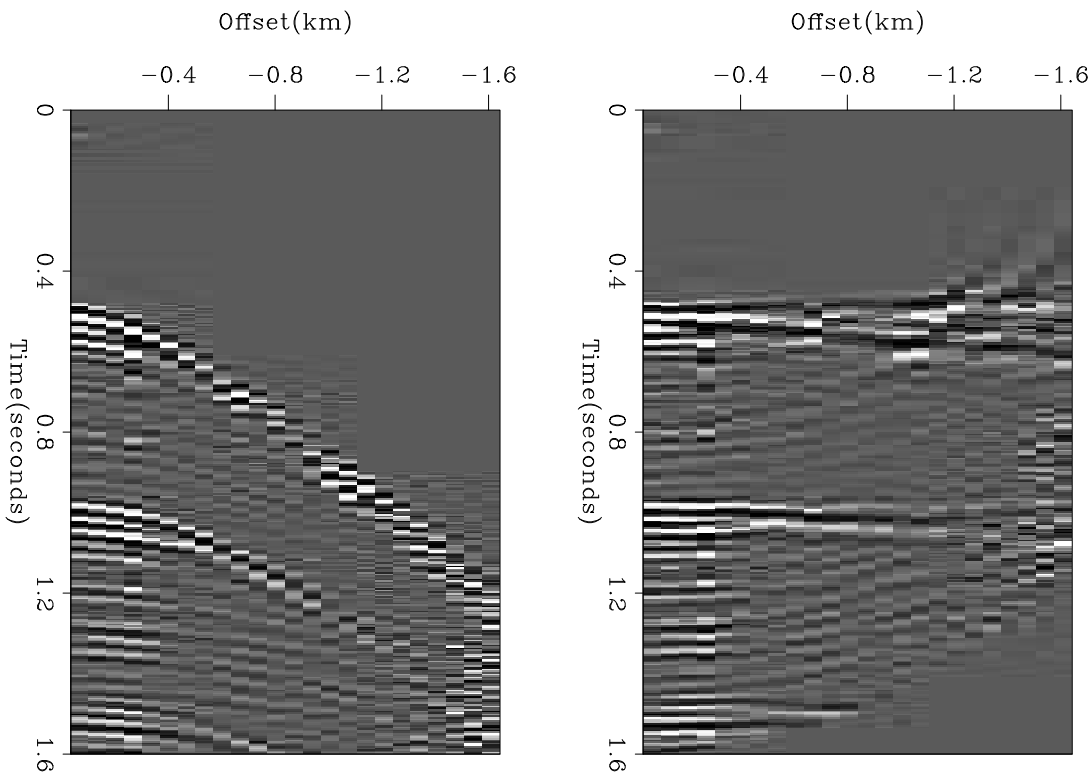


Figure 4.1: Marine data moved out with water velocity. Input on the left, output on the right. Press button for **movie** sweeping through velocity (actually through slowness squared). vela-stretch [ER,M]

subscript goes up to about one thousand instead of merely to ten.

$$\mathbf{m} \quad = \quad \mathbf{Nd} \quad = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \\ m_9 \\ m_{10} \end{bmatrix} = \begin{bmatrix} . & . & . & . & . & 1 & . & . & . & . \\ . & . & . & . & . & 1 & . & . & . & . \\ . & . & . & . & . & 1 & . & . & . & . \\ . & . & . & . & . & . & 1 & . & . & . \\ . & . & . & . & . & . & 1 & . & . & . \\ . & . & . & . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & . & . & 1 & . \\ . & . & . & . & . & . & . & . & . & 1 \\ . & . & . & . & . & . & . & . & . & . \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \\ d_8 \\ d_9 \\ d_{10} \end{bmatrix} \tag{4.10}$$

You can think of the matrix as having a horizontal t-axis and a vertical τ-axis. The 1's in the matrix are arranged on the hyperbola $t^2 = \tau^2 + x_0^2/v^2$. The transpose matrix defining some $\tilde{\mathbf{d}}$ from $\mathbf{m}$ gives synthetic data $\tilde{\mathbf{d}}$ from the zero-offset (or stack)

model **m**, namely,

$$\tilde{\mathbf{d}} \quad = \quad \mathbf{N}'\mathbf{m} \quad = \quad \begin{bmatrix} \tilde{d}_1 \\ \tilde{d}_2 \\ \tilde{d}_3 \\ \tilde{d}_4 \\ \tilde{d}_5 \\ \tilde{d}_6 \\ \tilde{d}_7 \\ \tilde{d}_8 \\ \tilde{d}_9 \\ \tilde{d}_{10} \end{bmatrix} \quad = \quad \begin{bmatrix} . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ 1 & 1 & 1 & . & . & . & . & . & . & . \\ . & . & . & 1 & 1 & . & . & . & . & . \\ . & . & . & . & . & 1 & 1 & . & . & . \\ . & . & . & . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & . & . & 1 & . \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \\ m_9 \\ m_{10} \end{bmatrix} \tag{4.11}$$

A program for **nearest-neighbor normal moveout** as defined by equations (4.10) and (4.11) is `nmo0()`. Because of the limited alphabet of programming languages, I used the keystroke `z` to denote τ. nmo0 A program is a "pull" program if the loop creating the output covers each location in the output and gathers the input from wherever it may be. A program is a "push" program if it takes each input and

```
subroutine nmo0( adj, add, slow,    x, t0, dt, n,zz,  tt )
integer  it, iz, adj, add,                        n
real  xs, t , z,           slow(n), x, t0, dt, zz(n), tt(n)
call adjnull(    adj, add,                        zz,n,  tt,n)
do iz= 1, n {   z = t0 + dt*(iz-1)                # Travel-time depth
                xs= x * slow(iz)
                t = sqrt ( z * z + xs * xs)
                it= 1 + .5 + (t - t0) / dt        # Round to nearest neighbor.
                if( it <= n )
                        if( adj == 0 )
                                tt(it) = tt(it) + zz(iz)
                        else
                                zz(iz) = zz(iz) + tt(it)
        }
return; end
```

Back

pushes it to wherever it belongs. Thus this NMO program is a "pull" program for doing the model building (data processing), and it is a "push" program for the data building. You could write a program that worked the other way around, namely, a loop over t with z found by calculation $z = \sqrt{t^2/v^2 - x^2}$. What is annoying is that if you want a push program going both ways, those two ways cannot be adjoint to one another.

Normal moveout is a linear operation. This means that data can be decomposed into any two parts, early and late, high frequency and low, smooth and rough, steep and shallow dip, etc.; and whether the two parts are NMO'ed either separately or together, the result is the same. The reason normal moveout is a linear operation is that we have shown it is effectively a matrix multiply operation and that operation fulfills $\mathbf{N}(\mathbf{d_1} + \mathbf{d_2}) = \mathbf{N}\mathbf{d_1} + \mathbf{N}\mathbf{d_2}$.

4.3. COMMON-MIDPOINT STACKING

Typically, many receivers record every shot, and there are many shots over the reflectors of interest. It is common practice to define the midpoint $y = (x_s + x_g)/2$ and then to sort the seismic traces into "**common-midpoint** gathers". After sorting,

each trace on a common-midpoint gather can be transformed by NMO into an equivalent zero-offset trace and the traces in the gather can all be added together. This is often called "common-depth-point (**CDP**) stacking" or, more correctly, "**common-midpoint stack**ing".

The adjoint to this operation is to begin from a model that is identical to the zero-offset trace and spray this trace to all offsets. There is no "official" definition of which operator of an operator pair is the operator itself and which is the adjoint. On the one hand, I like to think of the modeling operation itself as *the* operator. On the other hand, the industry machinery keeps churning away at many processes that have well-known names, so people often think of one of them as *the* operator. Industrial data-processing operators are typically **adjoint**s to modeling operators.

Figure 4.2 illustrates the operator pair, consisting of spraying out a zero-offset trace (the model) to all offsets and the adjoint of the spraying, which is **stack**ing. The moveout and stack operations are in subroutine `stack0()`. stack0 Let $\mathbf{S}'$ denote NMO, and let the stack be defined by invoking `stack0()` with the `adj=1` argument. Then **S** is the modeling operation defined by invoking `stack0()` with the `adj=0` argument. Figure 4.2 illustrates both. Notice the roughness on the waveforms caused by different numbers of points landing in one place. Notice also the increase

```
subroutine stack0( adj, add, slow,     t0,dt, x0,dx, nt,nx, stack, gather)
integer ix,        adj, add,                         nt,nx
real     x,                   slow(nt), 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 nmo0( adj, 1, slow, x, t0,dt, nt, stack, gather(1,ix))
        }
return; end
```

Back

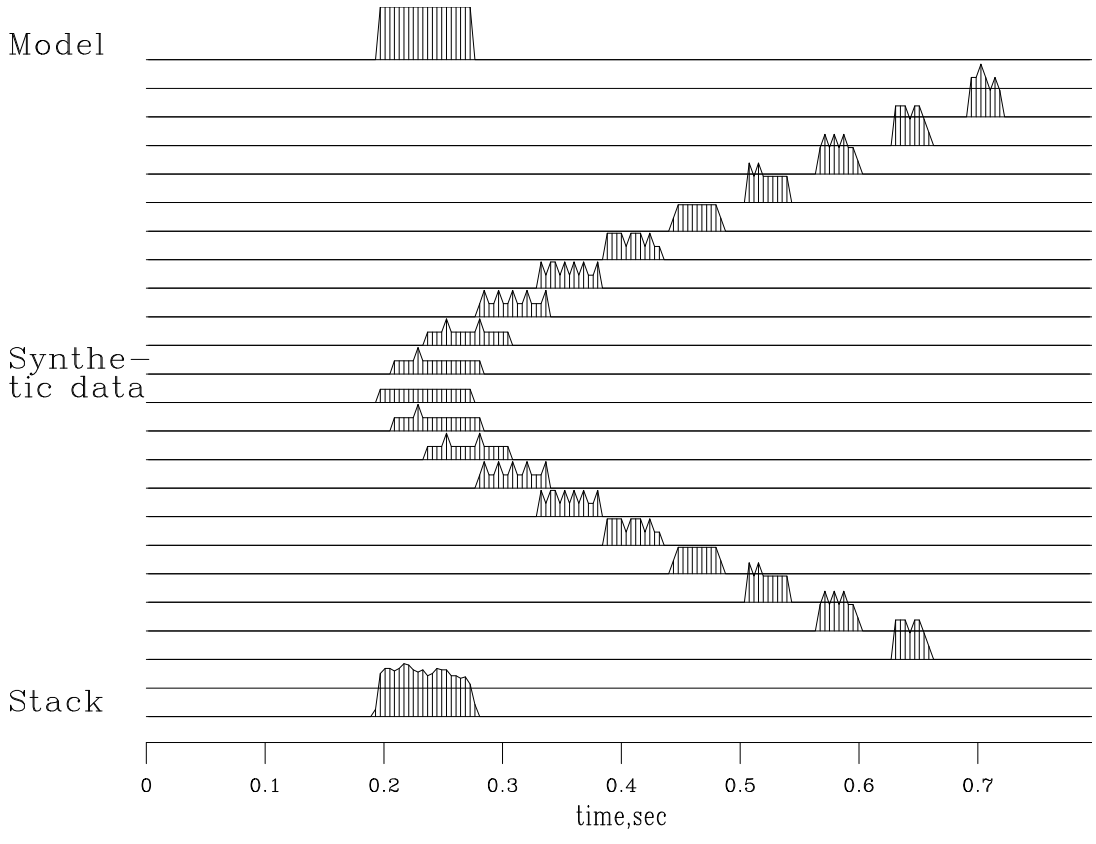


Figure 4.2: Top is a model trace **m**. Center shows the spraying to synthetic traces, $\mathbf{S}'\mathbf{m}$. Bottom is the stack of the synthetic data, $\mathbf{S}\mathbf{S}'\mathbf{m}$. vela-stack [ER]

of **AVO** (**amplitude** versus **offset**) as the waveform gets compressed into a smaller space. Finally, notice that the stack is a little rough, but the energy is all in the desired time window.

We notice a contradiction of aspirations. On the one hand, an operator has smooth outputs if it "loops over output space" and finds its input where ever it may. On the other hand, it is nice to have modeling and processing be exact adjoints of each other. Unfortunately, we cannot have both. If you loop over the output space of an operator, then the adjoint operator has a loop over input space and a consequent roughness of its output.

4.3.1. Crossing traveltime curves

Since velocity increases with depth, at wide enough offset a deep enough path will arrive sooner than a shallow path. In other words, traveltime curves for shallow events must cut across the curves of deeper events. Where traveltime curves cross, NMO is no longer a one-to-one transformation. To see what happens to the **stack**ing process I prepared Figures 4.3-4.5 using a typical marine recording geometry (although for clarity I used larger $(\Delta t, \Delta x)$) and we will use a typical Texas gulf coast

average velocity, $v(z) = 1.5 + \alpha z$ where $\alpha = .5$. First we repeat the calculation of Figure 4.2 with constant velocity $\alpha = 0$ and more reflectors. We see in Figure 4.3 that the **stack** reconstructs the model except for two details: (1) the **amplitude** diminishes with time, and (2) the early waveforms have become rounded. Then we

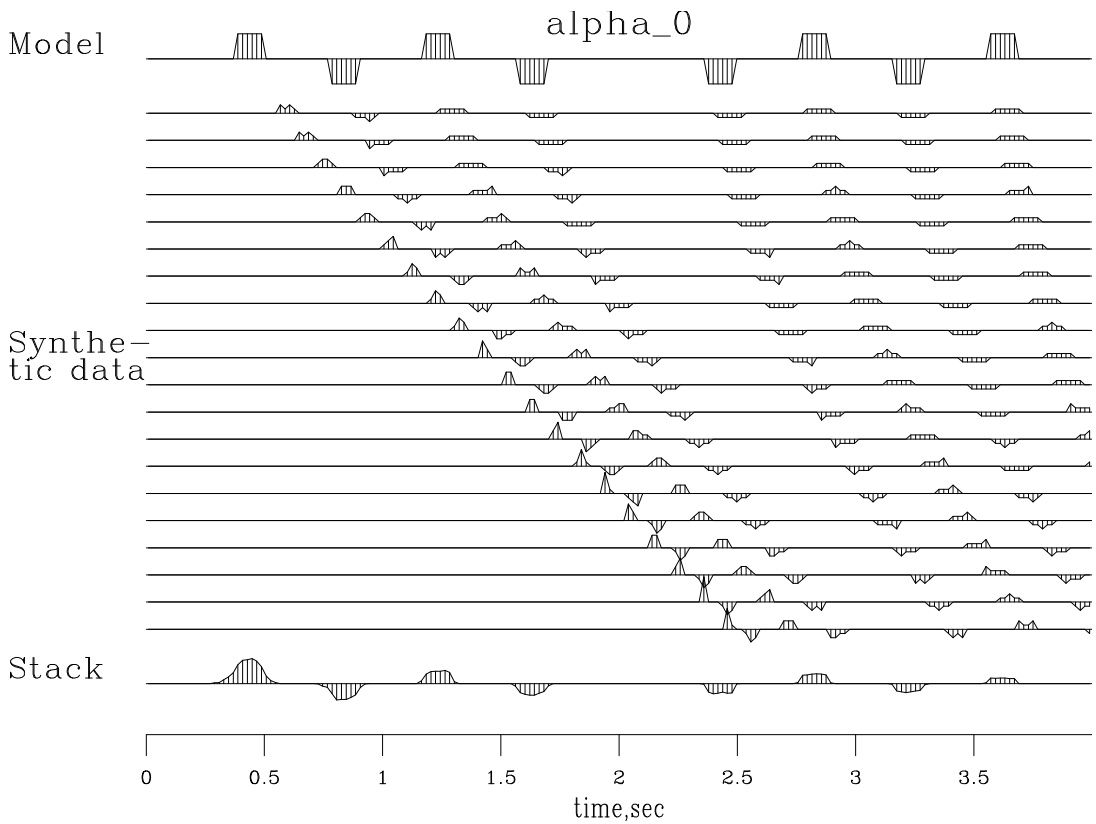


Figure 4.3: Synthetic CMP gather for constant velocity earth and reconstruction. vela-nmo0alfa0 [ER]

repeat the calculation with the Gulf coast typical velocity gradient $\alpha = 1/2$. The polarity reversal on the first arrival of the wide offset trace in Figure 4.4 is evidence that in practice traveltime curves do cross. (As was plainly evident in Figures 3.2, 3.3 and 3.4 crossing traveltime curves are even more significant elsewhere in the world.) Comparing Figure 4.3 to Figure 4.4 we see that an effect of the velocity gradient is to degrade the **stack**'s reconstruction of the model. Velocity gradient has ruined the waveform on the shallowest event, at about 400ms. If the plot were made on a finer mesh with higher frequencies, we could expect ruined waveforms a little deeper too.

Our NMO and stack subroutines can be used for modeling or for data processing. In designing these programs we gave no thought to signal **amplitude**s (although results showed an interesting **AVO** effect in Figure 4.2.) We could redesign the programs so that the modeling operator has the most realistic **amplitude** that we can devise. Alternately, we could design the **amplitude**s to get the best approximation to $\mathbf{S}'\mathbf{S} \approx \mathbf{I}$ which should result in "`Stack`" being a good approximation to "`Model`." I experimented with various weighting functions until I came up with subroutines `nmo1()` /prog:nmo1 and `stack1()` (like `stack0()` /prog:stack0) which embodies the **weighting function** $(\tau/t)(1/\sqrt{t})$ and which produces the result

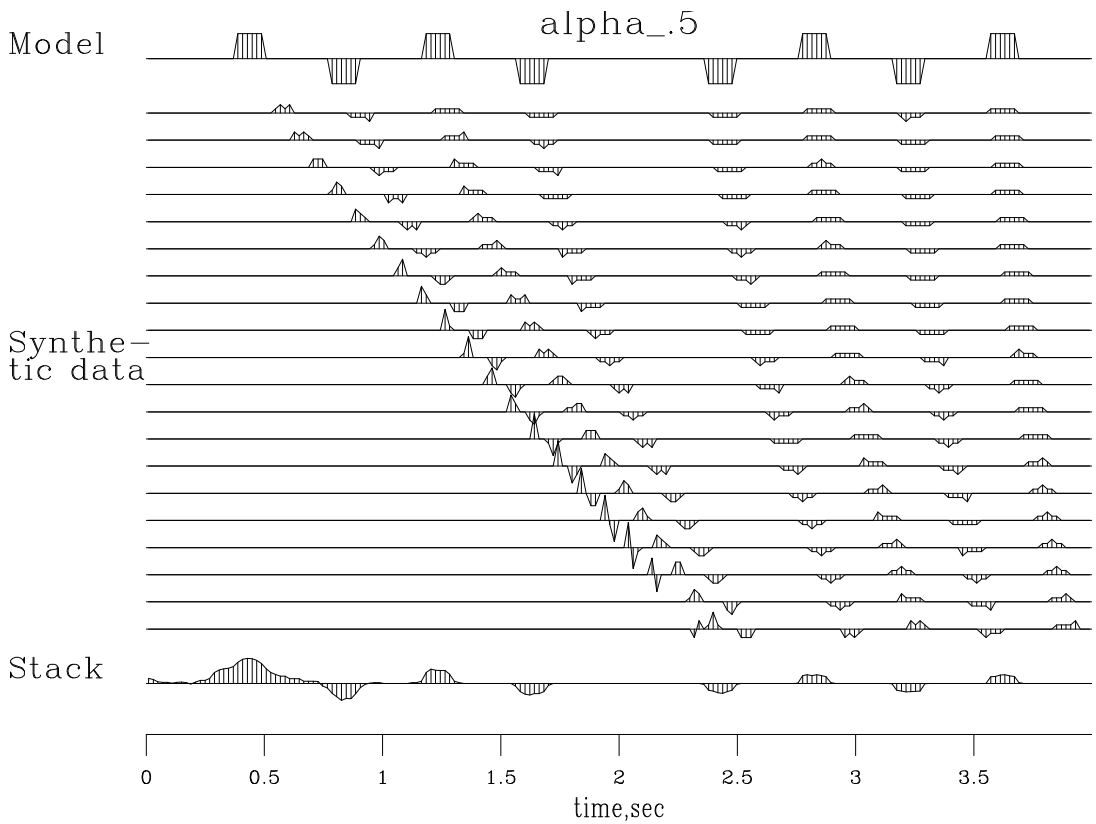


Figure 4.4: Synthetic CMP gather for velocity linearly increasing with depth (typical of Gulf of Mexico) and reconstruction. vela-nmo0alfa.5 [ER]

in Figure 4.5. The result in Figure 4.5 is very pleasing. Not only is the **amplitude** as

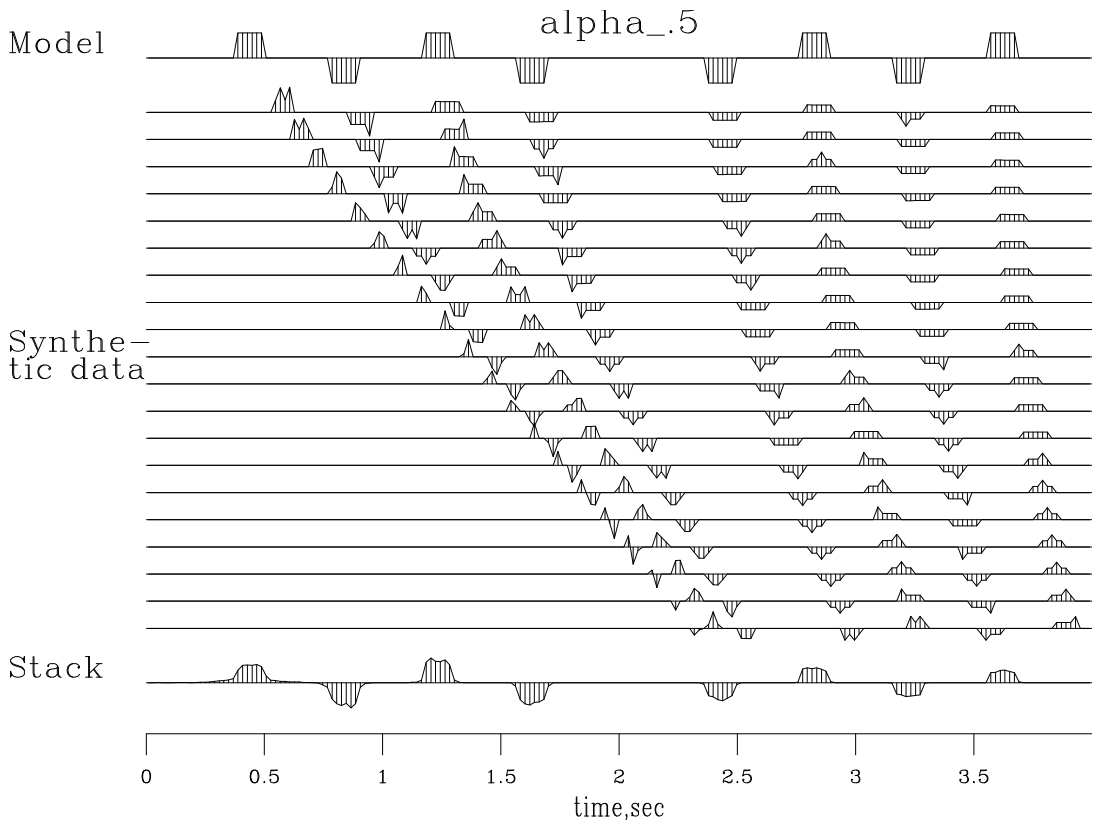


Figure 4.5: Synthetic CMP gather for velocity linearly increasing with depth and reconstruction with weighting functions in subroutine `nmo1()`. Lots of adjustable parameters here. vela-nmo1alfa.5 [ER]

a function of time better preserved, more importantly, the shallow wavelets are less smeared and have recovered their rectangular shape. The reason the reconstruction

```
subroutine nmo1( adj, add, slow,    x, t0, dt, n,zz,  tt )
integer  it, iz, adj, add,                        n
real  xs, t , z,           slow(n), x, t0, dt, zz(n), tt(n), wt
call adjnull(    adj, add,                        zz,n,  tt,n)
do iz= 1, n {   z = t0 + dt*(iz-1)
                xs = x * slow(iz)
                t = sqrt ( z * z + xs * xs) + 1.e-20
                wt = z/t * (1./sqrt(t))                  #  weighting function
                it = 1 + .5 + (t - t0) / dt
                if( it <= n )
                        if( adj == 0 )
                                tt(it) = tt(it) + zz(iz) * wt
                        else
                                zz(iz) = zz(iz) + tt(it) * wt
            }
return; end
```

Back

is much better is the cosine weighting implicit in τ/t. It has muted away much of the energy in the shallow asymptote. I think this energy near the asymptote is harmful because the waveform stretch is so large there. Perhaps a similar good result could be found by experimenting with muting programs such as `mutter()` /prog:mutter.

However, subroutine `nmo1()` /prog:nmo1 differs from `mutter()` in two significant respects: (1) `nmo1()` is based on a theoretical concept whereas `mutter()` requires observational parameters and (2) `mutter()` applies a weighting in the coordinates of the (t,x) input space, while `nmo1()` does that but also includes the coordinate τ of the the output space. With `nmo1()` events from different τ depths see different mutes which is good where a shallow event asymptote crosses a deeper event far from its own asymptote. In practice the problem of crossing traveltime curves is severe, as evidenced by Figures 3.2-3.4 and both weighting during NMO and muting should be used. nmo1 It is important to realize that the most accurate possible physical **amplitude**s are not necessarily those for which $\mathbf{S}'\mathbf{S} \approx \mathbf{I}$. Physically accurate amplitudes involve many theoretical issues not covered here. It is easy to include some effects (spherical divergence based on velocity depth variation) and harder to include others (surface ghosts and arrays). We omit detailed modeling here because it is the topic of so many other studies.

4.3.2. Ideal weighting functions for stacking

The difference between **stack**ing as defined by `nmo0()` /prog:nmo0 and by `nmo1()` /prog:nmo1 is in the weighting function $(\tau/t)(1/\sqrt{t})$. This weight made a big difference in the resolution of the stacks but I cannot explain whether this weighting function is the best possible one, or what systematic procedure leads to the best weighting function in general. To understand this better, notice that $(\tau/t)(1/\sqrt{t})$ can be factored into two weights, τ and $t^{-3/2}$. One weight could be applied before NMO and the other after. That would also be more efficient than weighting inside NMO, as does `nmo1()`. Additionally, it is likely that these weighting functions should take into account data truncation at the cable's end. Stacking is the most important operator in seismology. Perhaps some objective measure of quality can be defined and arbitrary powers of t, x, and τ can be adjusted until the optimum stack is defined. Likewise, we should consider weighting functions in the spectral domain. As the weights τ and $t^{-3/2}$ tend to cancel one another, perhaps we should filter with opposing filters before and after **NMO** and stack.

4.3.3. Gulf of Mexico stack and AGC

Next we create a "CDP stack" of our the Gulf of Mexico data set. Recall the moved out common-midpoint (CMP) gather Figure 3.11. At each midpoint there is one of these CMP gathers. Each gather is summed over its offset axis. Figure 4.6 shows the result of stacking over offset, at each midpoint. The result is an image of a cross section of the earth. In Figure 4.6 the early signals are too weak to see. This results from the small number of traces at early times because of the mute function. (Notice missing information at wide offset and early time on Figure 3.11.) To make the stack properly, we should divide by the number of nonzero traces. The fact that the mute function is tapered rather than cut off abruptly complicates the decision of what is a nonzero trace. In general we might like to apply a weighting function of offset. How then should the stack be weighted with time to preserve something like the proper signal strength? A solution is to make constant synthetic data (zero frequency). Stacking this synthetic data gives a weight that can be used as a divisor when stacking field data. I prepared code for such weighted stacking, but it cluttered the NMO and stack program and required two additional new subroutines, so I chose to leave the clutter in the electronic book and not to display it here. Instead, I chose to solve the signal strength problem by an old standby method, Automatic Gain

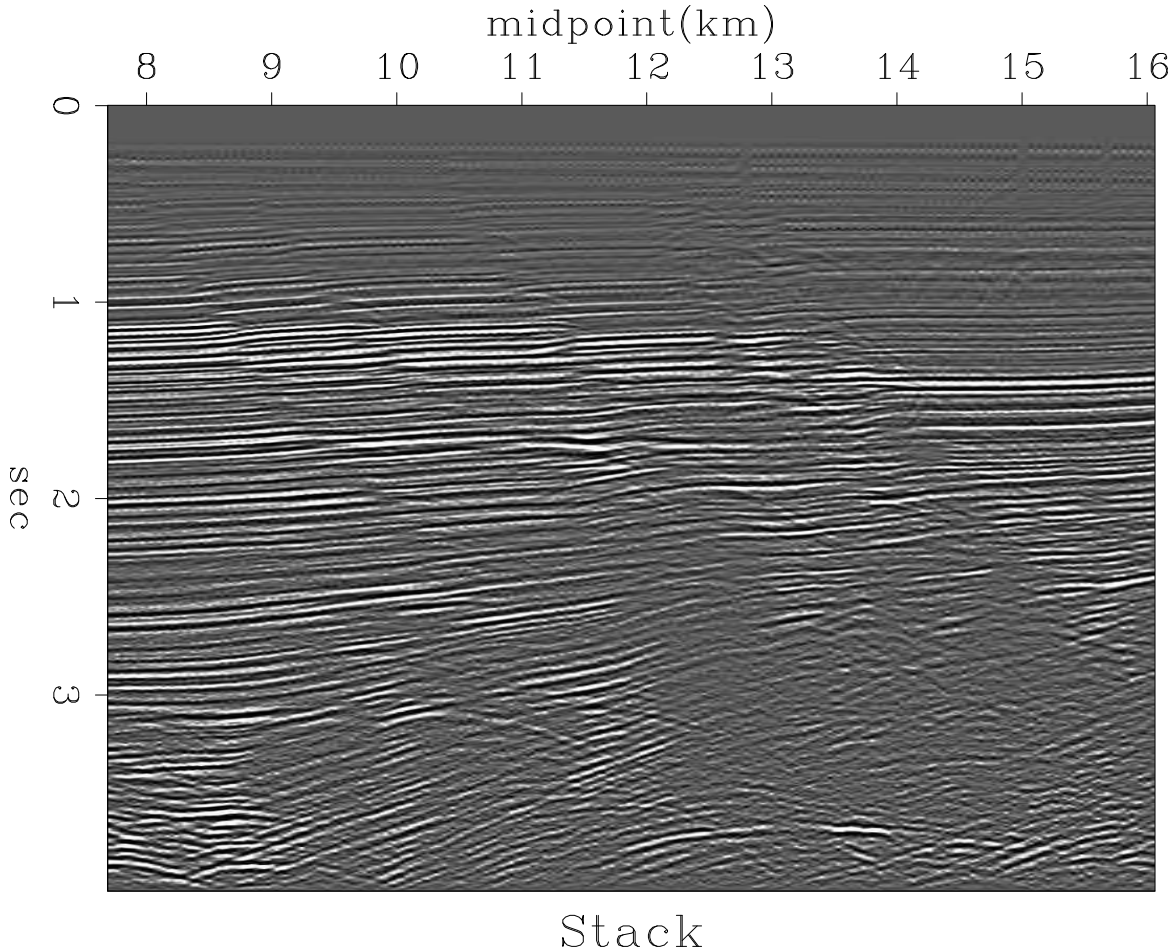


Figure 4.6: Stack done with a given velocity profile for all midpoints. vela-wgstack [ER]

Control (AGC). A divisor for the data is found by smoothing the absolute values of the data over a moving window. To make Figure 4.7 I made the divisor by smoothing in triangle shaped windows about a half second long. To do this, I used subroutine `triangle()` /prog:triangle.

4.4. VELOCITY SPECTRA

An important transformation in exploration geophysics takes data as a function of shot-receiver offset and transforms it to data as a function of apparent velocity. Data is summed along hyperbolas of many velocities. This important industrial process is adjoint to another that may be easier to grasp: data is synthesized by a superposition of many hyperbolas. The hyperbolas have various asymptotes (velocities) and various tops (apexes). Pseudocode for these transformations is

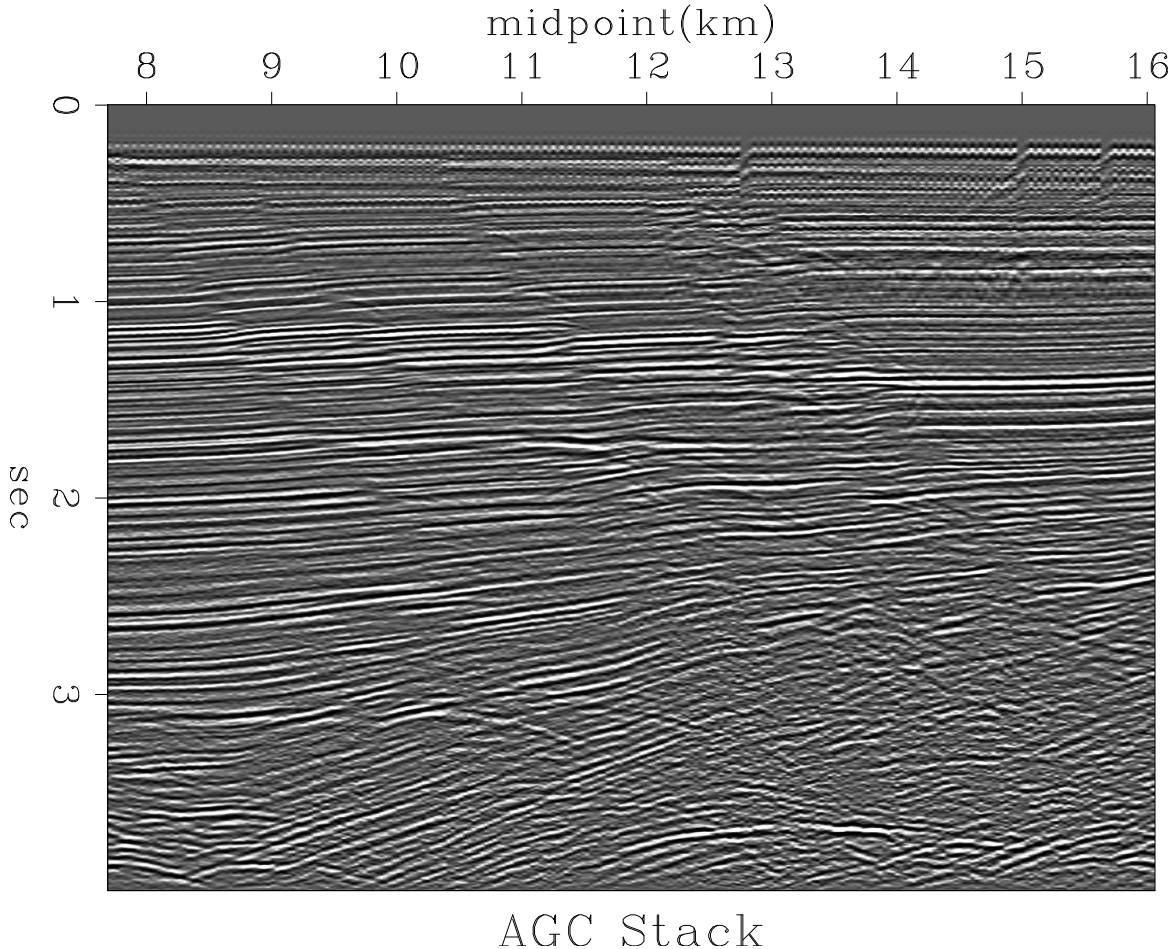


Figure 4.7: Stack of Figure 4.6 after AGC. vela-agcstack [ER,M]

```
do v {
do τ {
do x {
        t = sqrt(τ^2 + x^2/v^2)
        if hyperbola superposition
                data(t,x) = data(t,x) + vspace(τ,v)
        else if velocity analysis
                vspace(τ,v) = vspace(τ,v) + data(t,x)
}}}
```

We can ask the question, if we transform data to velocity space, and then return to data space, will we get the original data? Likewise we could begin from the velocity space, synthesize some data, and return to velocity space. Would we come back to where we started? The answer is yes, in some degree. Mathematically, the question amounts to this: Given the operator $\mathbf{A}$, is $\mathbf{A}'\mathbf{A}$ approximately an identity operator, i.e. is $\mathbf{A}$ nearly a unitary operator? It happens that $\mathbf{A}'\mathbf{A}$ defined by the pseudocode above is rather far from an identity transformation, but we can bring it much closer by including some simple scaling factors. It would be a lengthy digression here to

derive all these weighting factors but let us briefly see the motivation for them. One weight arises because waves lose **amplitude** as they spread out. Another weight arises because some angle-dependent effects should be taken into account. A third weight arises because in creating a velocity space, the near offsets are less important than the wide offsets and we do not even need the zero-offset data. A fourth weight is a frequency dependent one which is explained in chapter 6. Basically, the summations in the velocity transformation are like integrations, thus they tend to boost low frequencies. This could be compensated by scaling in the frequency domain with frequency as $\sqrt{-i\omega}$ with subroutine `halfdifa()` /prog:halfdifa.

The weighting issue will be examined in more detail later. Meanwhile, we can see nice quality examples from very simple programs if we include the weights in the physical domain, $w = \sqrt{1/t}\,\sqrt{x/v}\,\tau/t$. (Typographical note: Do not confuse the weight w (double you) with omega ω.) To avoid the coding clutter of the frequency domain weighting $\sqrt{-i\omega}$ I omit that, thus getting smoother results than theoretically preferable. Figure 4.8 illustrates this smoothing by starting from points in velocity space, transforming to offset, and then back and forth again.

There is one final complication relating to weighting. The most symmetrical approach is to put w into both $\mathbf{A}$ and $\mathbf{A}'$. This is what subroutine `velsimp()`

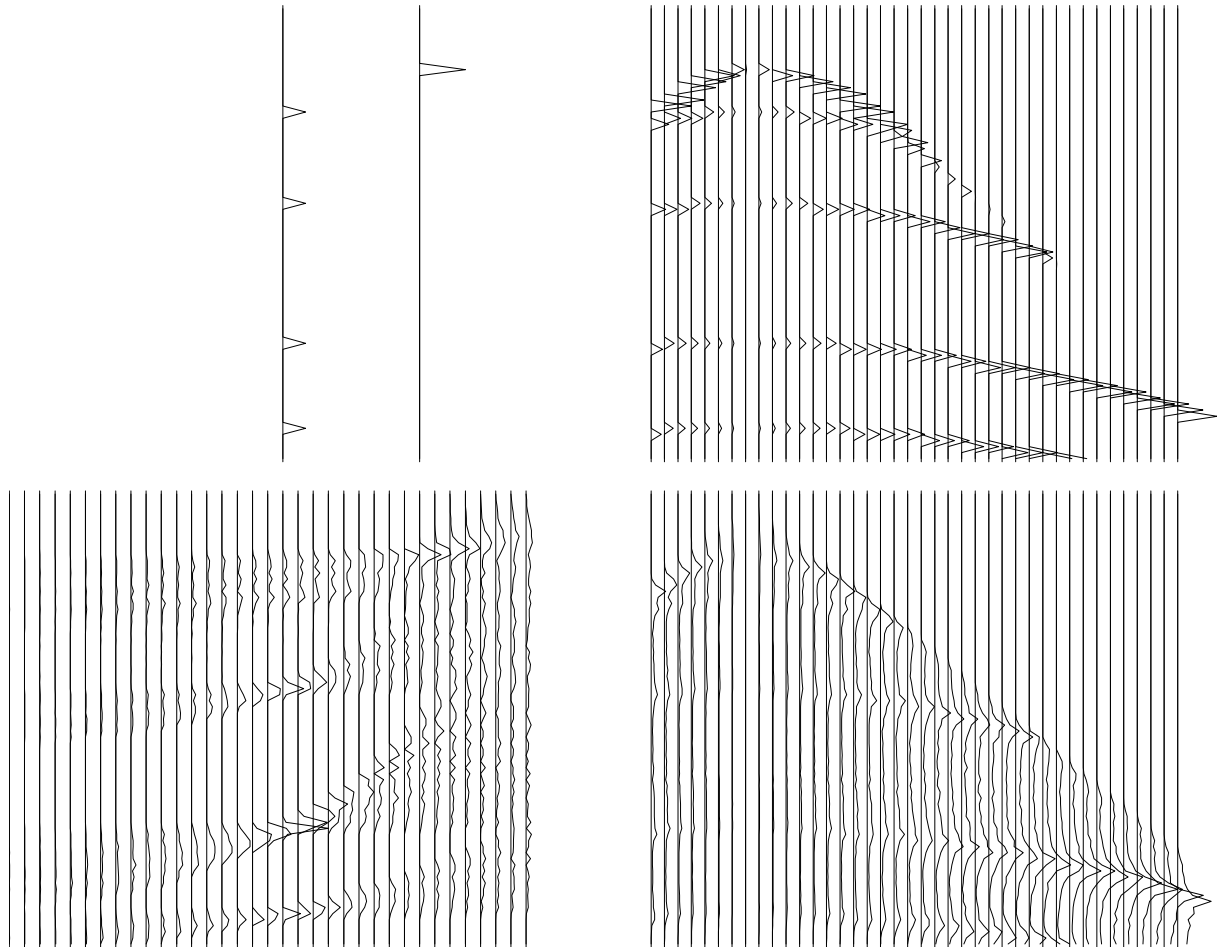


Figure 4.8: Iteration between spaces. Left are model spaces. Right are data spaces. Right derived from left. Lower model space derived from upper data space. vela-velvel [ER]

```
# velsimp --- simple velocity transform
#
subroutine velsimp( adj,add, t0,dt,x0,dx,s0,ds, nt,nx,ns, modl, data)
integer it,ix,is,   adj,add,                     nt,nx,ns,              iz,nz
real x,s,sx, t,z, z0,dz,wt,   t0,dt,x0,dx,s0,ds, modl(nt,ns),data(nt,nx)
call adjnull(       adj,add,                      modl,nt*ns, data,nt*nx)
nz= nt;  z0=t0; dz= dt;                 # z is travel time depth
do is= 1, ns {  s = s0 + (is-1) * ds
do ix= 1, nx {  x = x0 + (ix-1) * dx
do iz= 2, nz {  z = z0 + (iz-1) * dz
        sx =  abs( s * x)
        t  = sqrt( z * z + sx * sx)
        it = 1.5 + (t - t0) / dt
        if ( it <= nt) {                                wt= (z/t) / sqrt( t)
                if( adj == 0 )
                        data(it,ix) = data(it,ix) + modl(iz,is) * sx * wt
                else
                        modl(iz,is) = modl(iz,is) + data(it,ix) * sx * wt
                }
        }}}
return; end
```

Back

/prog:velsimp does. Thus, because of the weighting by $\sqrt{x}$, the synthetic data in Figure 4.8 is nonphysical. An alternate view is to *define* **A** (by the pseudo code above, or by some modeling theory) and then for reverse transformation use $w^2\mathbf{A}'$.
velsimp

An example of applying subroutine `velsimp()` /prog:velsimp to field data is shown in Figure 4.9.

4.4.1. Velocity picking

For many kinds of data analysis, we need to know the velocity of the earth as a function of depth. To derive such information we begin from Figure 4.9 and draw a line through the maxima. In practice this is often a tedious manual process, and it needs to be done everywhere we go. There is no universally accepted way to automate this procedure, but we will consider one that is simple enough that it can be fully described here, and which works well enough for these demonstrations. (I plan to do a better job later.)

Theoretically we can define the velocity or slowness as a function of traveltime depth by the moment function. Take the absolute value of the data scans and smooth

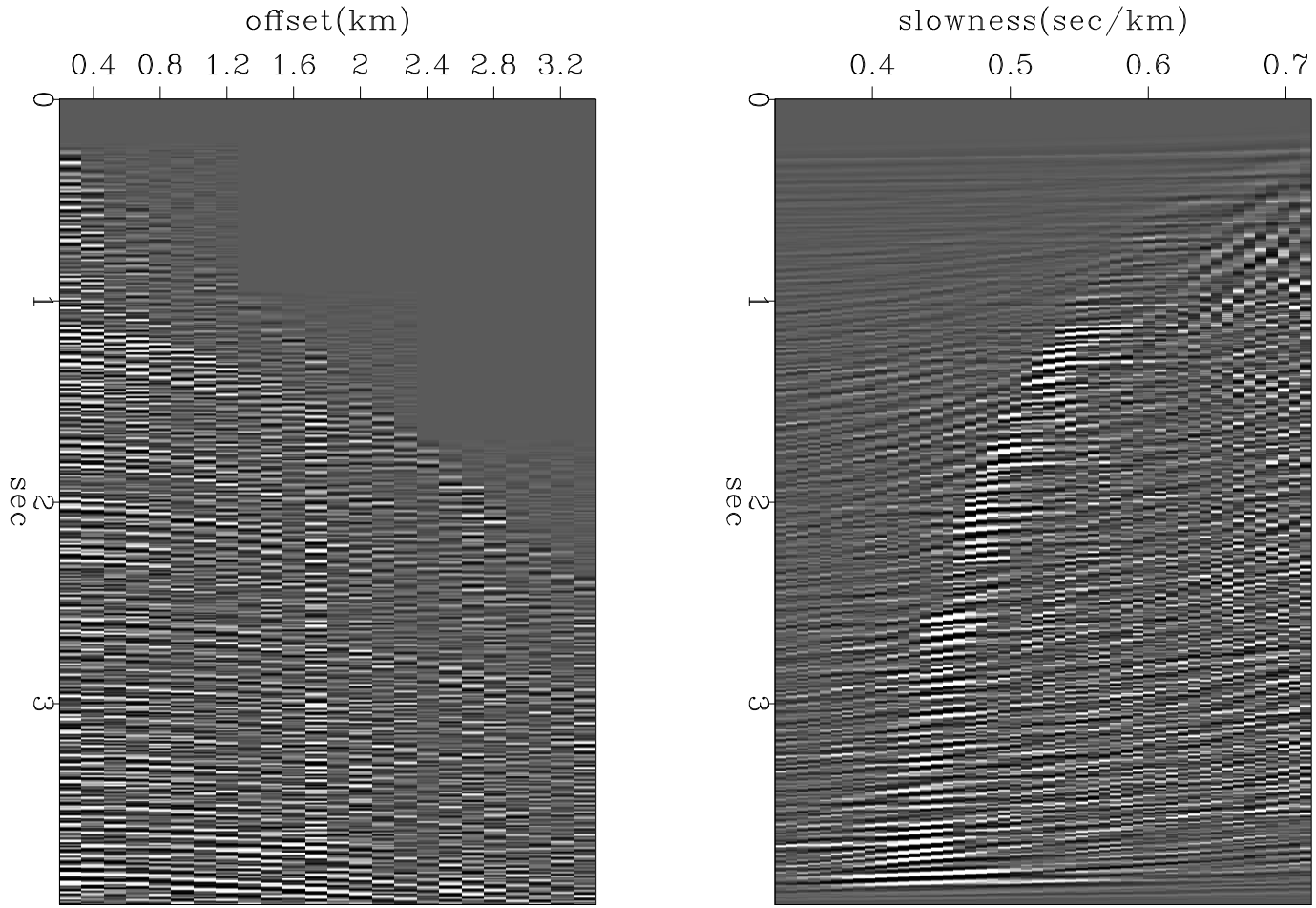


Figure 4.9: Transformation of data as a function of offset (left) to data as a function of slowness (velocity scans) on the right using subroutine `velsimp()`. vela-mutvel [ER]

them a little on the time axis to make something like an unnormalized probability function, say $p(\tau,s) > 0$. Then the slowness $s(\tau)$ could be defined by the moment function, i.e.,

$$s(\tau) \quad = \quad \frac{\sum_s \; s \; p(\tau,s)}{\sum_s \; p(\tau,s)} \tag{4.12}$$

The problem with defining slowness $s(\tau)$ by the moment is that it is strongly influenced by noises away from the peaks, particularly water velocity noises. Thus, better results can be obtained if the sums in equation (4.12) are limited to a range about the likely solution. To begin with, we can take the likely solution to be defined by universal or regional experience. It is sensible to begin from a one-parameter equation for velocity increasing with depth where the form of the equation allows a ray tracing solution such as equation (3.43). Experience with Gulf of Mexico data shows that $\alpha \approx 1/2$ sec^{-1} is reasonable there for equation (3.43), and that is the smooth curve in Figure 4.10.

Experience with moments, equation (4.12), shows they are reasonable when the desired result is near the guessed center of the range. Otherwise, the moment is biased towards the initial guess. This bias can be reduced in stages. At each stage we shrink the width of the zone used to compute the moment. This procedure is used in

```
subroutine slowfit( vsurface, alpha, t0,dt, s0,ds, scan,nt,ns, reg,    slow)
integer irange, it,is,                                  nt,ns
real num,den, t,s,  vsurface, alpha, t0,dt, s0,ds, scan(nt,ns),reg(nt),slow(nt)
do it= 1, nt {  t= t0 + dt*(it-1) + dt
        reg(it)  = 1./( vsurface * sqrt( (exp(alpha*t) - 1.)/(alpha*t) ))
        slow(it) = reg(it)
        }
do irange= ns/4, 5, -1 {                                # shrink the fairway
do it=     1,   nt     {        t= t0 + dt*(it-1)
        do is= 1, ns   {        s= s0 + ds*(is-1)
                if( s > slow(it) + irange*ds)   scan(it,is) = 0.
                if( s < slow(it) - irange*ds)   scan(it,is) = 0.
                if( s > 1./1.6              )   scan(it,is) = 0.        # water
                }
        den= 0.0;       num= 0.0
        do is= 1, ns {  s= s0 + ds*(is-1)
                num = num + scan(it,is) * s
                den = den + scan(it,is)
                }
        slow(it) = num / ( den + 1.e-20)
        if( slow(it) == 0.)   slow(it) = 1./vsurface
        }}
return; end
```

Back

subroutine `slowfit()` /prog:slowfit which after smoothing to be described, gives the oscillatory curve you see in Figure 4.10. slowfit A more customary way to view velocity space is to square the velocity scans and normalize them by the sum of the squares of the signals. This has the advantage that the remaining information represents velocity spectra and removes variation due to seismic **amplitude**s. Since in practice, reliability seems somehow proportional to **amplitude** the disadvantage of normalization is that reliability becomes more veiled.

An appealing visualization of velocity is shown in the right side of Figure 4.10. This was prepared from the absolute value of left side, followed by filtering spatially with an antisymmetric leaky integral function. (See PVI page 57). An example is shown on the right side of Figure 4.10.

4.4.2. Stabilizing RMS velocity

With velocity analysis, we estimate the RMS velocity. Later we will need both the RMS velocity and the **interval velocity**. (The word “interval” designates an interval

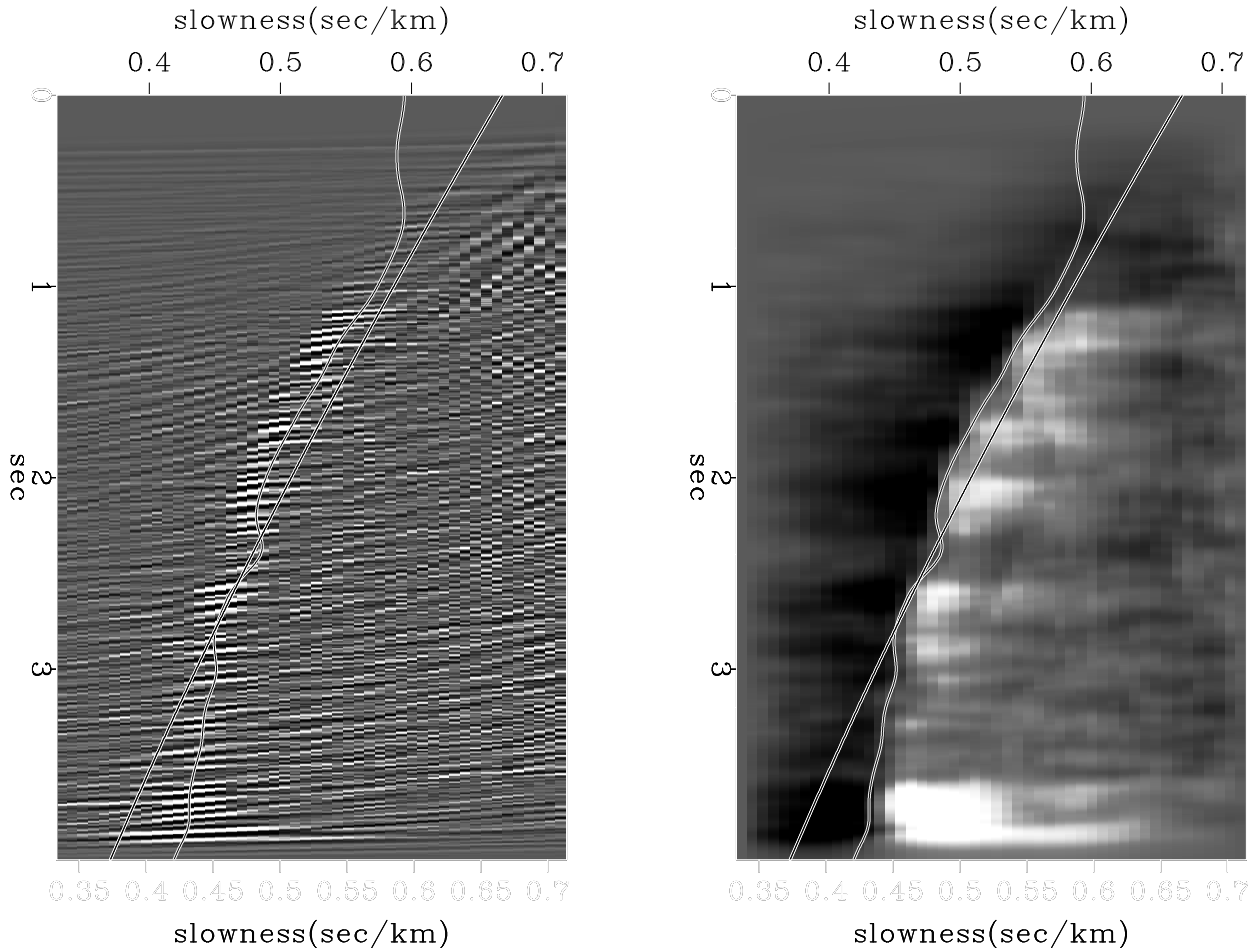


Figure 4.10: Left is the slowness scans. Right is the slowness scans after absolute value, smoothing a little in time, and antisymmetric leaky integration over slowness. Overlaying both is the line of slowness picks. vela-slowfit [ER]

between two reflectors.) Recall from chapter 3 equation (3.24)

$$t^2 \quad = \quad \tau^2 + \frac{4h^2}{V^2(\tau)}$$

Routine `vint2rms()` /prog:vint2rms converts from interval velocity to RMS velocity and vice versa. vint2rms The forward conversion follows in straightforward steps: square, integrate, square root. The inverse conversion, like an adjoint, retraces the steps of the forward transform but it does the inverse at every stage. There is however, a messy problem with nearly all field data that must be handled along the inverse route. The problem is that the observed RMS velocity function is generally a rough function, and it is generally unreliable over a significant portion of its range. To make matters worse, deriving an **interval velocity** begins as does a derivative, roughening the function further. We soon find ourselves taking square roots of negative numbers, which requires judgement to proceed. The technique used in `vint2rms()` /prog:vint2rms is to average the squared interval velocity in ever expanding neighborhoods until there are no longer any negative squared interval velocities. As long as we are restricting v^2 from being negative, it is easy to restrict it to be above some allowable velocity, say `vminallow`. Figures 4.11 and 4.12

```
# Invertible transform from interval velocity to RMS.
#
subroutine vint2rms(  inverse, vminallow, dt, vint, nt,  vrms )
integer it, wide,     inverse,                      nt
real    vmin,                   vminallow, dt, vint( nt), vrms( nt)
temporary real                                     vis( nt),  sum( nt)
if( inverse == 0) {              do it= 1, nt
                                          vis(it) = vint(it) ** 2
        sum(1) = 0.;             do it= 2, nt
                                          sum(it) = sum(it-1) + vis(it) * dt
        vrms(1) = vint(1);       do it= 2, nt
                                          vrms(it) = sqrt( sum(it) / ((it-1)*dt) )
        }
else {  do it= 1, nt
                 sum(it)= ((it-1)*dt) * amax1( vrms(it)**2, vminallow**2 )
        vis(1) = vrms(1) ** 2
        do it= 2, nt
                 vis(it) = ( sum(it) - sum(it-1) )/ dt
        wide= 2;  repeat {
                 vmin = vis(1);  do it=1,nt { if( vis(it)<vmin)  vmin = vis(it) }
                 if( vmin > vminallow**2 )   break
                 call triangle( wide, 1, nt, vis, vis)           # smooth vis()
                 wide = wide + 1
                 if( wide >= nt/3) call erexit('Velocity less than allowable.')
                 }
        do it= 1, nt
                 vint(it) = sqrt( vis(it))
        }
return; end
```

Back

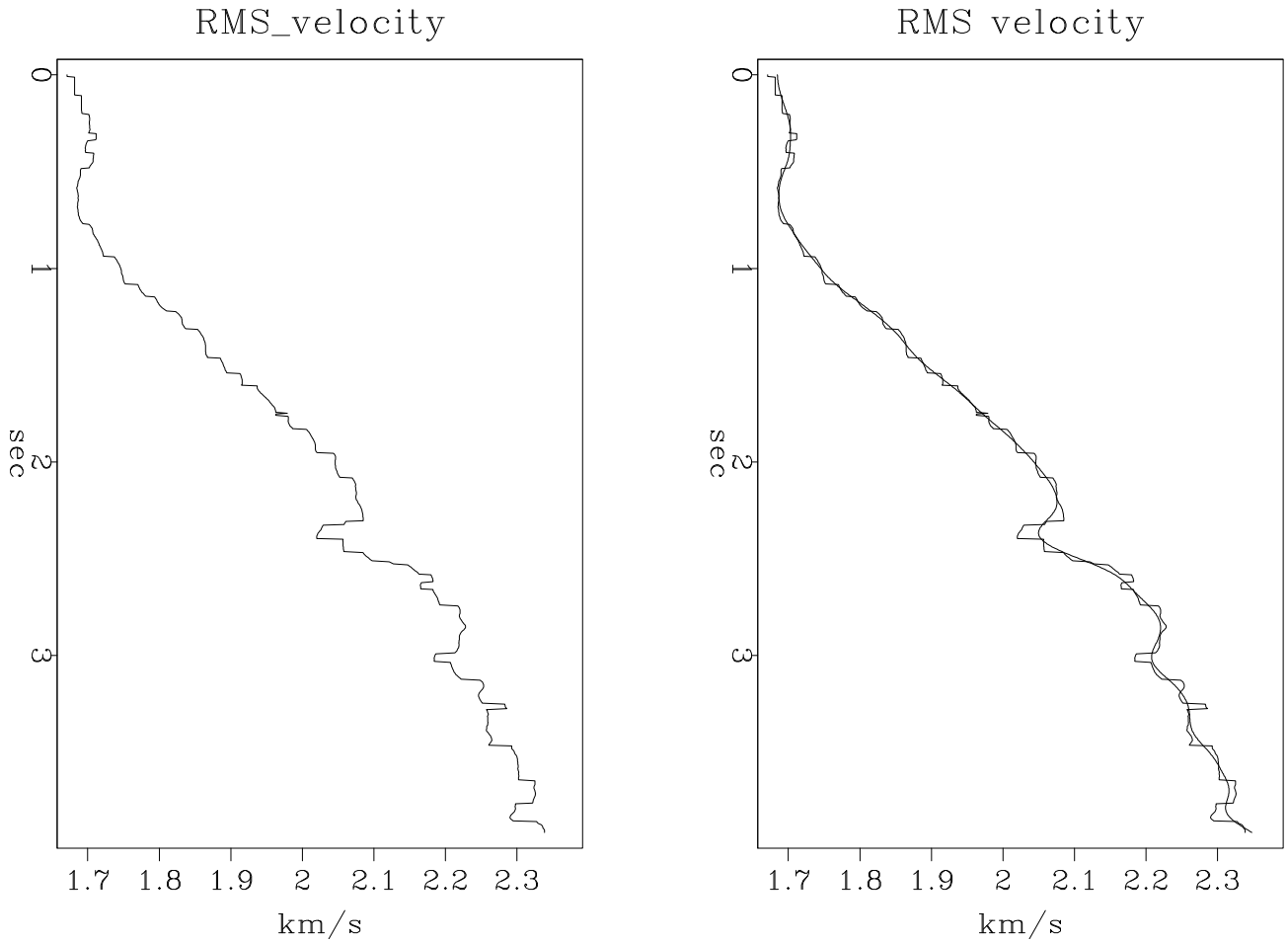


Figure 4.11: Left is the raw RMS velocity. Right is a superposition of RMS velocities, the raw one, and one constrained to have realistic interval velocities. vela-rufsmo [ER]

were derived from the velocity scans in Figure 4.10. Figure 4.11 shows the RMS velocity before and after a trip backward and forward through routine `vint2rms()` /prog:vint2rms. The interval velocity associated with the smoothed velocity is in figure 4.12.

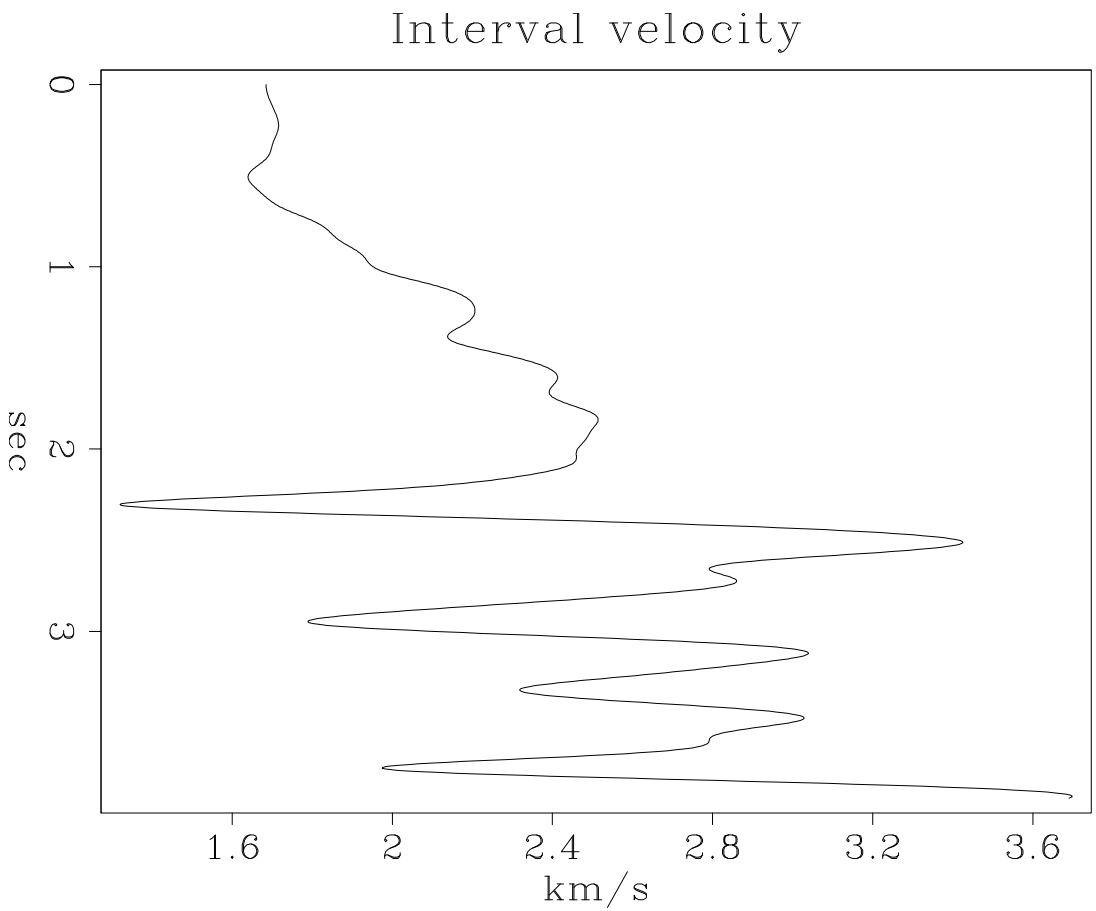


Figure 4.12: Interval velocity associated with the smoothed RMS velocity of Figure 4.11. Pushbutton allows experimentation with `vminallow`. vela-vrmsint [ER]

Chapter 5

Zero-offset migration

In chapter 4 we discussed methods of imaging horizontal reflectors and of estimating velocity $v(z)$ from the offset dependence of seismic recordings. In this chapter, we turn our attention to imaging methods for *dipping* reflectors. These imaging

methods are usually referred to as “migration” techniques.

Offset is a geometrical nuisance when reflectors have dip. For this reason, we develop migration methods here and in the next chapter for forming images from hypothetical *zero-offset* seismic experiments. Although there is usually ample data recorded near zero-offset, we never record purely zero-offset seismic data. However, when we consider offset and dip together in chapter 8 we will encounter a widely-used technique (dip-moveout) that often converts finite-offset data into a useful estimate of the equivalent zero-offset data. For this reason, **zero-offset migration** methods are widely used today in industrial practice. Furthermore the concepts of zero-offset migration are the simplest starting point for approaching the complications of finite-offset migration.

5.1. MIGRATION DEFINED

The term “migration” probably got its name from some association with movement. A casual inspection of migrated and unmigrated sections shows that migration causes many reflection events to shift their positions. These shifts are necessary because the *apparent* positions of reflection events on unmigrated sections are gen-

erally not the *true* positions of the reflectors in the earth. It is not difficult to visualize why such "acoustic illusions" occur. An analysis of a zero-offset section shot above a dipping reflector illustrates most of the key concepts.

5.1.1. A dipping reflector

Consider the zero-offset seismic survey shown in Figure 5.1. This survey uses one source-receiver pair, and the receiver is always at the same location as the source. At each position, denoted by $S_1, S_2,$ and S_3 in the figure, the source emits waves and the receiver records the echoes as a single seismic trace. After each trace is recorded, the source-receiver pair is moved a small distance and the experiment is repeated.

As shown in the figure, the source at S_2 emits a spherically-spreading wave that bounces off the reflector and then returns to the receiver at S_2. The raypaths drawn between S_i and R_i are orthogonal to the reflector and hence are called *normal rays*. These rays reveal how the zero-offset section misrepresents the truth. For example, the trace recorded at S_2 is dominated by the reflectivity near reflection point R_2, where the normal ray from S_2 hits the reflector. If the zero-offset section corresponding to Figure 5.1 is displayed, the reflectivity at R_2 will be falsely displayed

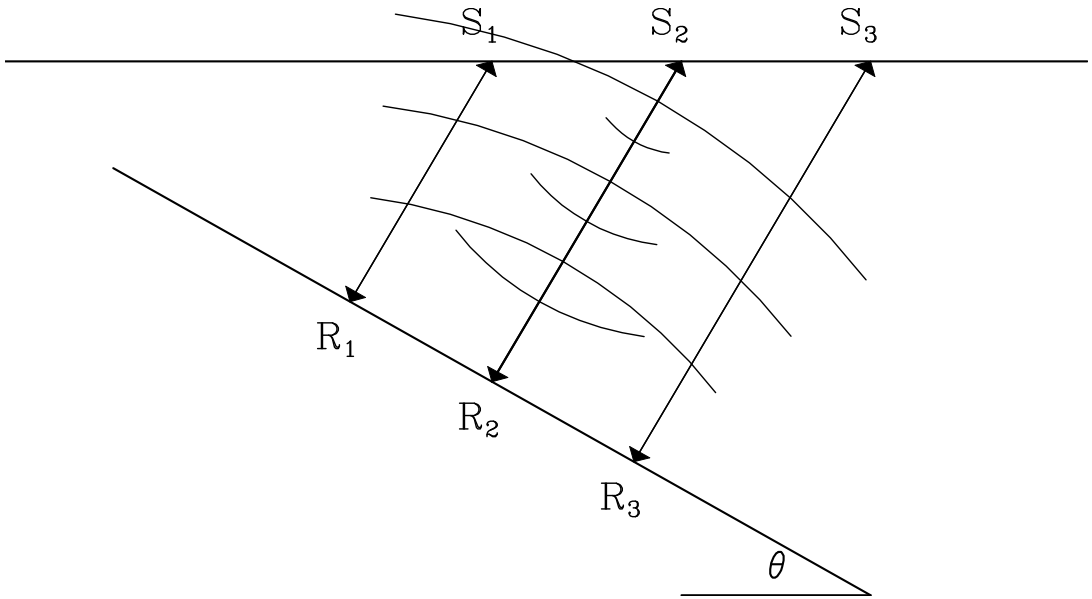


Figure 5.1: Raypaths and wavefronts for a zero-offset seismic line shot above a dipping reflector. The earth's propagation velocity is constant. krch-reflexpt [ER]

as though it were directly beneath S_2, which it certainly is not. This lateral mispositioning is the first part of the illusion. The second part is vertical: if converted to depth, the zero-offset section will show R_2 to be deeper than it really is. The reason is that the slant path of the normal ray is longer than a vertical shaft drilled from the surface down to R_2.

5.1.2. Dipping-reflector shifts

A little geometry gives simple expressions for the horizontal and vertical position errors on the zero-offset section, which are to be corrected by migration. Figure 5.2 defines the required quantities for a reflection event recorded at S corresponding to the reflectivity at R. The two-way travel time for the event is related to the length d

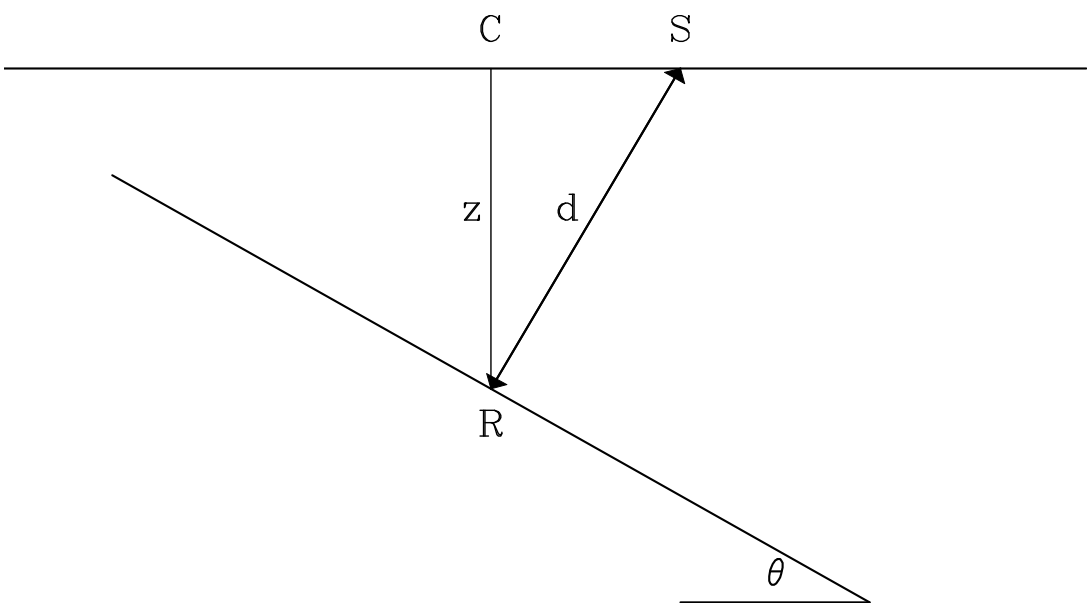


Figure 5.2: Geometry of the normal ray of length d and the vertical "shaft" of length z for a zero-offset experiment above a dipping reflector. krch-reflkine [ER]

of the normal ray by

$$t \quad = \quad \frac{2d}{v} \ , \tag{5.1}$$

where v is the constant propagation velocity. Geometry of the triangle CRS shows that the true depth of the reflector at R is given by

$$z \quad = \quad d \ \cos\theta \ , \tag{5.2}$$

and the lateral shift between true position C and false position S is given by

$$\Delta x \quad = \quad d \ \sin\theta \quad = \quad \frac{v\,t}{2} \ \sin\theta \ . \tag{5.3}$$

It is conventional to rewrite equation (5.2) in terms of two-way *vertical* traveltime τ:

$$\tau \quad = \quad \frac{2z}{v} \quad = \quad t \ \cos\theta \ . \tag{5.4}$$

Thus both the vertical shift $t - \tau$ and the horizontal shift Δx are seen to vanish when the dip angle θ is zero.

5.1.3. Hand migration

Geophysicists recognized the need to correct these positioning errors on zero-offset sections long before it was practical to use computers to make the corrections. Thus a number of hand-migration techniques arose. It is instructive to see how one such scheme works. Equations (5.3) and (5.4) require knowledge of three quantities: t, v, and θ. Of these, the event time t is readily measured on the zero-offset section. The velocity v is usually *not* measurable on the zero offset section and must be estimated from finite-offset data, as was shown in chapter 4. That leaves the dip angle θ. This can be related to the reflection slope p of the observed event, which is measurable on the zero-offset section:

$$p_0 \quad = \quad \frac{\partial t}{\partial y} \ , \tag{5.5}$$

where y (the midpoint coordinate) is the location of the source-receiver pair. The slope p_0 is sometimes called the "*time-dip of the event*" or more loosely as the "*dip of the event*". It is obviously closely related to Snell's parameter, which we discussed in chapter 3. The relationship between the measurable time-dip p_0 and

the dip angle θ is called "**Tuchel's law**":

$$\sin\theta \quad = \quad \frac{v\,p_0}{2} \quad . \tag{5.6}$$

This equation is clearly just another version of equation (3.8), in which a factor of 2 has been inserted to account for the two-way traveltime of the zero-offset section.

Rewriting the migration shift equations in terms of the measurable quantities t and p yields usable "hand-migration" formulas:

$$\Delta x \quad = \quad \frac{v^2\ p\ t}{4} \tag{5.7}$$

$$\tau \quad = \quad t\sqrt{1\ -\ \frac{v^2 p^2}{4}} \quad . \tag{5.8}$$

Hand migration divides each observed reflection event into a set of small segments for which p has been measured. This is necessary because p is generally not constant along real seismic events. But we can consider more general events to be the union of a large number of very small dipping reflectors. Each such segment is then mapped from its unmigrated (y,t) location to its migrated (y,τ) location based on

the equations above. Such a procedure is sometimes also known as “map migration.”

Equations (5.7) and (5.8) are useful for giving an idea of what goes on in zero-offset migration. But using these equations directly for practical seismic migration can be tedious and error-prone because of the need to provide the time dip p as a separate set of input data values as a function of y and t. One nasty complication is that it is quite common to see *crossing events* on zero-offset sections. This happens whenever reflection energy coming from two different reflectors arrives at a receiver at the same time. When this happens the time dip p becomes a *multi-valued* function of the (y,t) coordinates. Furthermore, the recorded wavefield is now the sum of two different events. It is then difficult to figure out which part of summed amplitude to move in one direction and which part to move in the other direction.

For the above reasons, the seismic industry has generally turned away from hand-migration techniques in favor of more automatic methods. These methods require as inputs nothing more than

- The zero-offset section
- The velocity v .

There is no need to separately estimate a $p(y,t)$ field. The automatic migration program somehow "figures out" which way to move the events, even if they cross one another. Such automatic methods are generally referred to as *"wave-equation migration"* techniques, and are the subject of the remainder of this chapter. But before we introduce the automatic migration methods, we need to introduce one additional concept that greatly simplifies the migration of zero-offset sections.

5.1.4. A powerful analogy

Figure 5.3 shows two wave-propagation situations. The first is realistic field sounding. The second is a thought experiment in which the reflectors in the earth suddenly explode. Waves from the hypothetical explosion propagate up to the earth's surface where they are observed by a hypothetical string of geophones.

Notice in the figure that the ray paths in the field-recording case seem to be the same as those in the **exploding-reflector** case. It is a great conceptual advantage to imagine that the two wavefields, the observed and the hypothetical, are indeed the same. If they are the same, the many thousands of experiments that have really been done can be ignored, and attention can be focused on the one hypothetical experi-

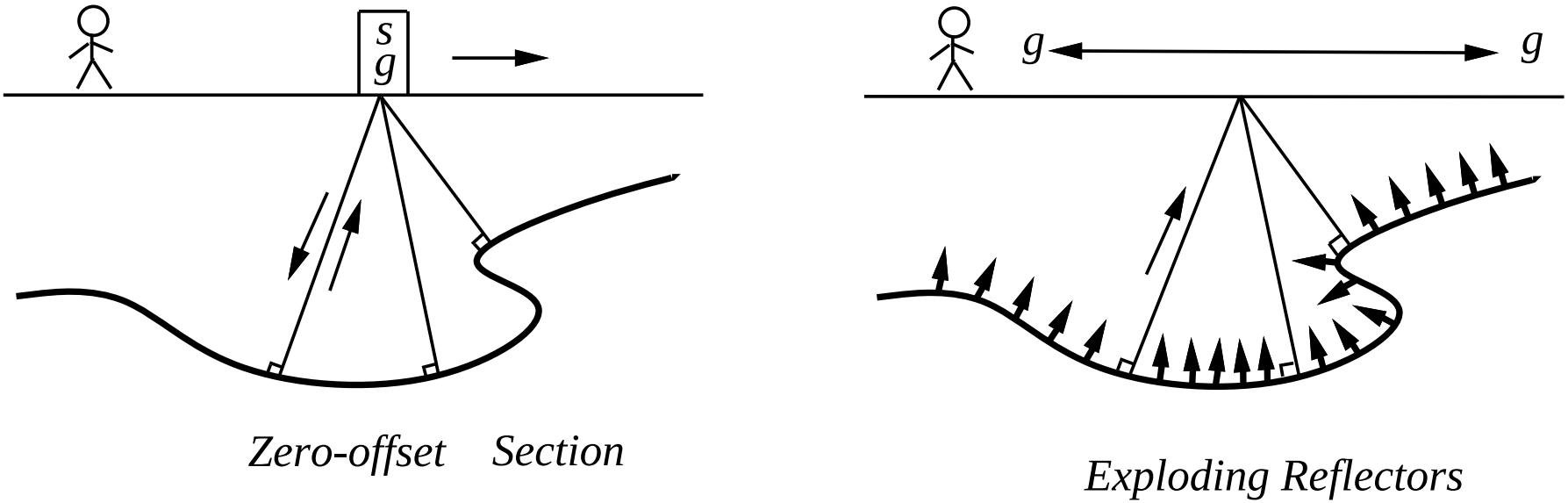


Figure 5.3: Echoes collected with a source-receiver pair moved to all points on the earth's surface (left) and the "exploding-reflectors" conceptual model (right). krch-expref [NR]

ment. One obvious difference between the two cases is that in the field geometry waves must first go down and then return upward along the same path, whereas in the hypothetical experiment they just go up. Travel time in field experiments could be divided by two. In practice, the data of the field experiments (two-way time) is analyzed assuming the sound velocity to be half its true value.

5.1.5. Limitations of the exploding-reflector concept

The exploding-reflector concept is a powerful and fortunate analogy. It enables us to think of the data of many experiments as though it were a single experiment. Unfortunately, the exploding-reflector concept has a serious shortcoming. No one has yet figured out how to extend the concept to apply to data recorded at nonzero offset. Furthermore, most data is recorded at rather large offsets. In a modern marine prospecting survey, there is not one hydrophone, but hundreds, which are strung out in a cable towed behind the ship. The recording cable is typically 2-3 kilometers long. Drilling may be about 3 kilometers deep. So in practice the angles are big. Therein lie both new problems and new opportunities, none of which will be considered until chapter 8.

Furthermore, even at zero offset, the exploding-reflector concept is not quantitatively correct. For the moment, note three obvious failings: First, Figure 5.4 shows rays that are not predicted by the exploding-reflector model. These rays will

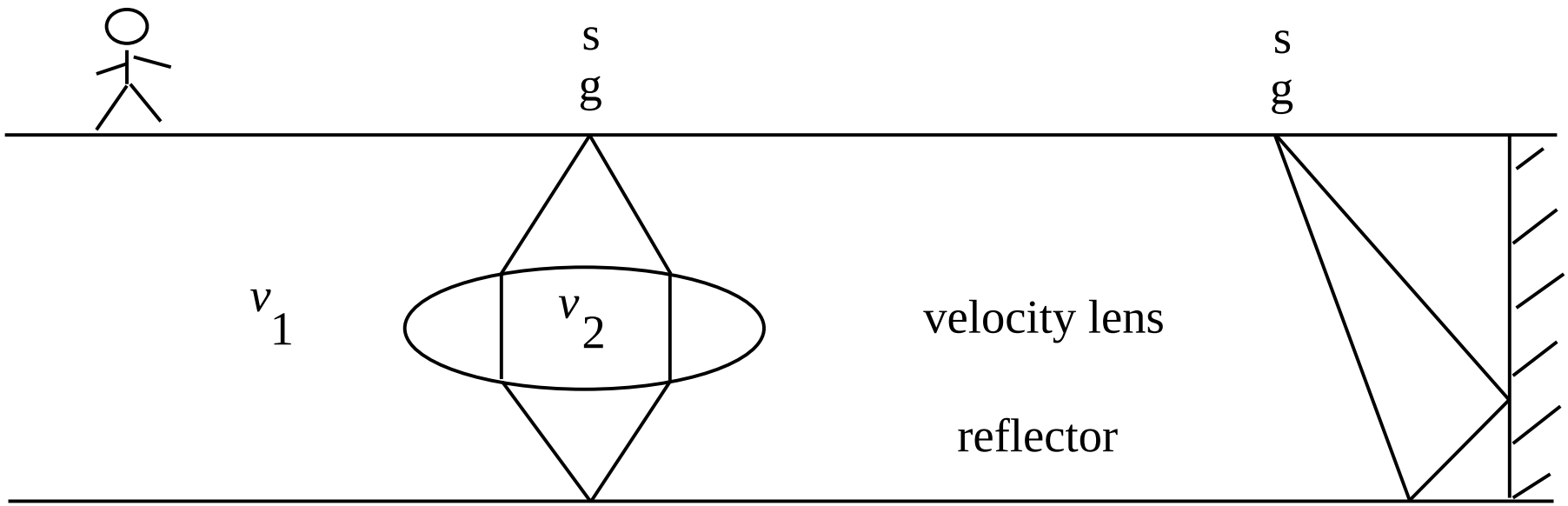


Figure 5.4: Two rays, not predicted by the exploding-reflector model, that would nevertheless be found on a zero-offset section. krch-fail [NR]

be present in a zero-offset section. Lateral velocity variation is required for this

situation to exist.

Second, the exploding-reflector concept fails with **multiple reflection**s. For a flat sea floor with a two-way travel time t_1, multiple reflections are predicted at times $2t_1$, $3t_1$, $4t_1$, etc. In the exploding-reflector geometry the first multiple goes from reflector to surface, then from surface to reflector, then from reflector to surface, for a total time $3t_1$. Subsequent multiples occur at times $5t_1$, $7t_1$, etc. Clearly the multiple reflections generated on the zero-offset section differ from those of the exploding-reflector model.

The third failing of the exploding-reflector model is where we are able to see waves bounced from both sides of an interface. The exploding-reflector model predicts the waves emitted by both sides have the same polarity. The physics of reflection coefficients says reflections from opposite sides have opposite polarities.

5.2. HYPERBOLA PROGRAMMING

Consider an exploding reflector at the point (z_0, x_0). The location of a circular wave front at time t is $v^2t^2 = (x - x_0)^2 + (z - z_0)^2$. At the surface, $z = 0$, we have the equation of the hyperbola where and when the impulse arrives on the surface data

plane (t, x). We can make a "synthetic data plane" by copying the explosive source amplitude to the hyperbolic locations in the (t, x) data plane. (We postpone including the amplitude reduction caused by the spherical expansion of the wavefront.) Forward modeling amounts to taking every point from the (z, x)-plane and adding it into the appropriate hyperbolic locations in the (t, x) data plane. Hyperbolas get added on top of hyperbolas.

Now let us think backwards. Suppose we survey all day long and record no echos except for one echo at time t_0 that we can record only at location x_0. Our data plane is thus filled with zero values except the one nonzero value at (t_0, x_0). What earth model could possibly produce such data?

An earth model that is a spherical mirror with bottom at (z_0, x_0) will produce a reflection at only one point in data space. Only when the source is at the center of the circle will all the reflected waves return to the source. For any other source location, the reflected waves will not return to the source. The situation is summarized in Figure 5.5.

Above explains how an impulse at a point in image space can transform to a hyperbola in data space, likewise, on return, an impulse in data space can transform to a semicircle in image space. We can simulate a straight line in either space by

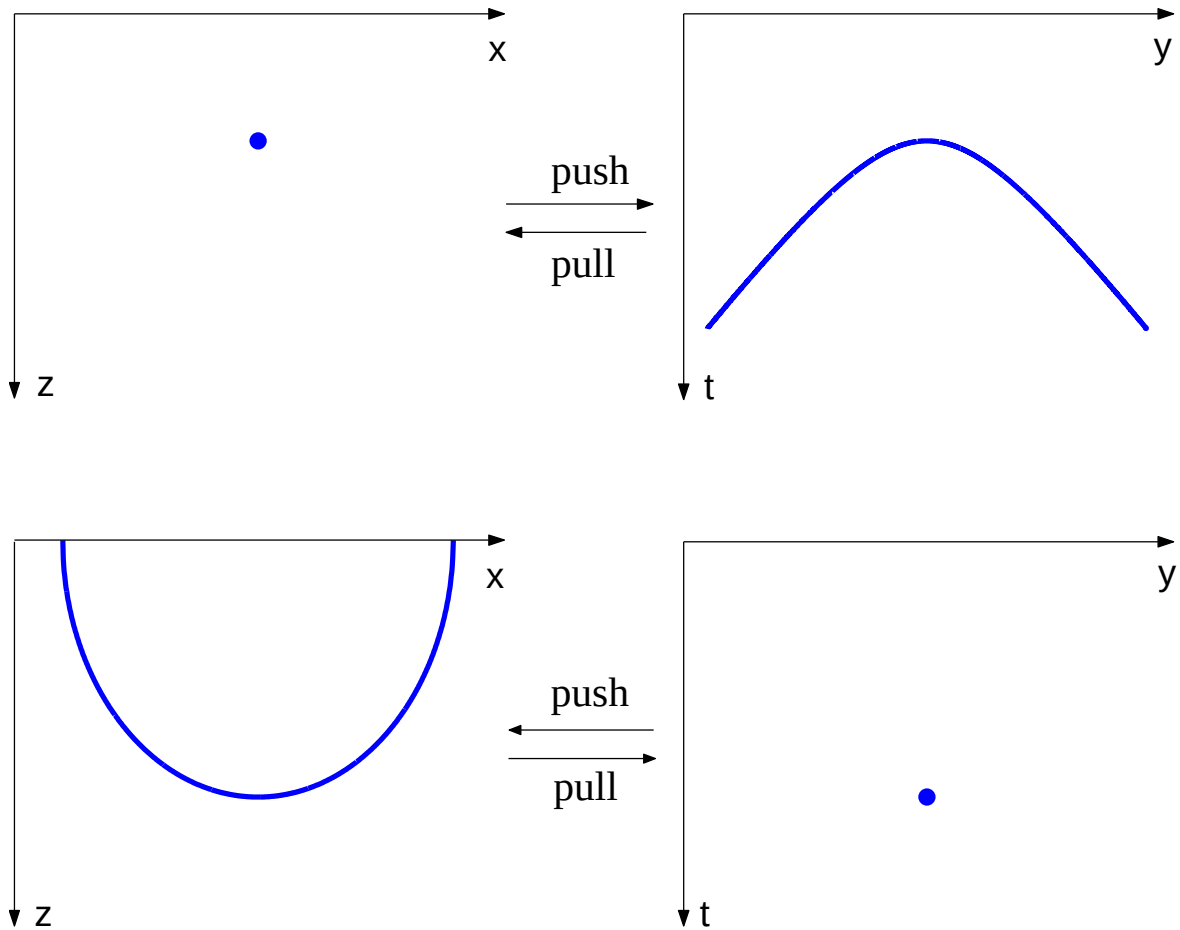


Figure 5.5: Point response model to data and converse. krch-yaxun [NR]

superposing points along a line. Figure 5.6 shows how points making up a line reflector diffract to a line reflection, and how points making up a line reflection migrate to a line reflector. First we will look at the simplest, most tutorial migration subroutine I could devise. Then we will write an improved version and look at some results.

5.2.1. Tutorial Kirchhoff code

Subroutine `kirchslow()` below is the best tutorial **Kirchhoff migration-**modeling program I could devise. A nice feature of this program is that it works OK while the edge complications do not clutter it. The program copies information from data space `data(it,iy)` to model space `modl(iz,ix)` or vice versa. Notice that of these four axes, three are independent (stated by loops) and the fourth is derived by the circle-hyperbola relation $t^2 = \tau^2 + x^2/v^2$. Subroutine `kirchslow()` for `adj=0` copies information from model space to data space, i.e. from the hyperbola top to its flanks. For `adj=1`, data summed over the hyperbola flanks is put at the hyperbola top. kirchslow Notice how this program has the ability to create a hyperbola given an input impulse in (x,z)-space, and a circle given an input impulse in (x,t)-space.

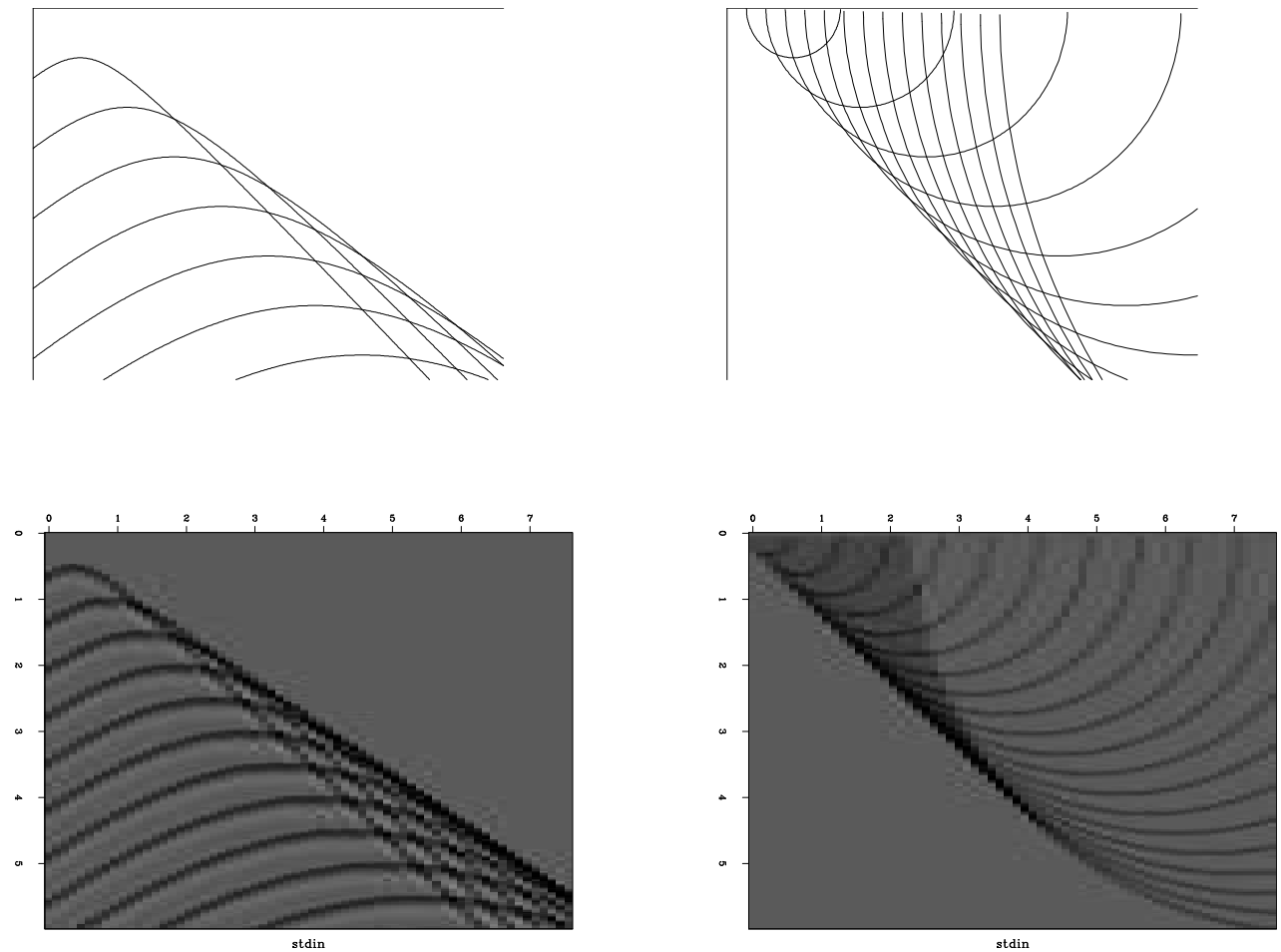


Figure 5.6: Left is a superposition of many hyperbolas. The top of each hyperbola lies along a straight line. That line is like a reflector, but instead of using a continuous line, it is a sequence of points. Constructive interference gives an apparent reflection off to the side. Right shows a superposition of semicircles. The bottom of each semicircle lies along a line that could be the line of an observed plane wave. Instead the plane wave is broken into point arrivals, each being interpreted as com-

```
# Kirchhoff migration and diffraction.  (tutorial, slow)
#
subroutine kirchslow(   adj, add,  velhalf, t0,dt,dx, modl,nt,nx,  data)
integer ix,iy,it,iz,nz, adj, add,                          nt,nx
real x0,y0,dy,z0,dz,t,x,y,z,hs,    velhalf, t0,dt,dx, modl(nt,nx), data(nt,nx)
call adjnull(           adj, add,                       modl,nt*nx,  data,nt*nx)
x0=0.;  y0=0;  dy=dx;  z0=t0;  dz=dt; nz=nt
do ix= 1, nx {  x = x0 + dx * (ix-1)
do iy= 1, nx {  y = y0 + dy * (iy-1)
do iz= 1, nz {  z = z0 + dz * (iz-1)            # z = travel-time depth
        hs=      (x-y) / velhalf
        t = sqrt( z * z  +  hs * hs )
        it = 1.5 + (t-t0) / dt
        if( it <= nt )
                if( adj == 0 )
                        data(it,iy) = data(it,iy) + modl(iz,ix)
                else
                        modl(iz,ix) = modl(iz,ix) + data(it,iy)

        }}}
return; end
```

Back

The three loops in subroutine `kirchslow()` may be interchanged at will without changing the result. To emphasize this flexibility, the loops are set at the same indentation level. We tend to think of fixed values of the outer two loops and then describe what happens on the inner loop. For example, if the outer two loops are those of the model space `modl(iz,ix)`, then for `adj=1` the program sums data along the hyperbola into the "fixed" point of model space. When loops are reordered, we think differently and opportunities arise for speed improvements.

5.2.2. Fast Kirchhoff code

Subroutine `kirchslow()` can easily be speeded by a factor that is commonly more than 30. The philosopy of this book is to avoid minor optimizations, but a factor of 30 really is significant, and the analysis required for the speed up is also interesting. Much of the inefficiency of `kirchslow()` arises when $x_{\max} \gg v t_{\max}$ because then many values of t are computed beyond $t_{\max}$. To avoid this, we notice that for fixed offset (`ix-iy`) and variable depth `iz`, as depth increases, time `it` eventually goes beyond the bottom of the mesh and, as soon as this happens, it will continue to happen for all larger values of `iz`. Thus we can `break` out of the `iz` loop the first

```
# Kirchhoff migration and diffraction.  (greased lightning)
#
subroutine kirchfast( adj, add, vrms,     t0,dt,dx, modl,nt,nx, data)
integer ix,iz,it,ib,  adj, add,                          nt,nx
real    amp,t,z,b,                vrms(nt), t0,dt,dx, modl(nt,nx),data(nt,nx)
call adjnull(           adj, add,                       modl,nt*nx, data,nt*nx)
do ib= -nx, nx {          b = dx * ib                       # b = offset
        do iz= 2, nt {    z = t0 + dt * (iz-1)              # z = travel-time depth
                          t = sqrt( z**2  +  (b*2/vrms(iz))**2 )
                          it = 1.5 + (t - t0) / dt
             if( it > nt )  break
                amp = (z / t)  *  sqrt( nt*dt / t )
                do ix= max0(1, 1-ib),  min0(nx, nx-ib)
                          if( adj == 0 )
                                  data(it,ix+ib)=data(it,ix+ib)+modl(iz,ix   )*amp
                          else
                                  modl(iz,ix   )=modl(iz,ix   )+data(it,ix+ib)*amp
                }
        }
return; end
```

Back

time we go off the mesh to avoid computing anything beyond as shown in subroutine `kirchfast()`. (Some quality compromises, limiting the aperture or the dip, also yield speedup, but we avoid those.) Another big speedup arises from reusing square roots. Since the square root depends only on offset and depth, once computed it can be used for all `ix`. Finally, these changes of variables have left us with more complicated side boundaries, but once we work these out, the inner loops can be devoid of tests and in `kirchfast()` they are in a form that is highly optimizable by many compilers. kirchfast

Originally the two Kirchhoff programs produced identical output, but finally I could not resist adding an important feature to the fast program, scale factors $z/t = \cos\theta$ and $1/\sqrt{t}$ that are described elsewhere. The fast program allows for velocity variation with depth. When velocity varies laterally the story becomes much more complicated.

Figure 5.7 shows an example. The model includes dipping beds, syncline, anticline, fault, unconformity, and buried focus. The result is as expected with a "bow tie" at the buried focus. On a video screen, I can see hyperbolic events originating from the unconformity and the fault. At the right edge are a few faint edge artifacts. We could have reduced or eliminated these edge artifacts if we had extended the

model to the sides with some empty space.

5.2.3. Kirchhoff artifacts

Reconstructing the earth model with the adjoint option in `kirchfast()` /prog:kirchfast yields the result in Figure 5.8. The reconstruction generally succeeds but is imperfect in a number of interesting ways. Near the bottom and right side, the reconstruction fades away, especially where the dips are steeper. Bottom fading results because in modeling the data we abandoned arrivals after a certain maximum time. Thus energy needed to reconstruct dipping beds near the bottom was abandoned. Likewise along the side we abandoned rays shooting off the frame.

Difficult migrations are well known for producing semicircular reflectors. Here we have controlled everything fairly well so none are obvious, but on a video screen I see some semicircles.

Next is the problem of the spectrum. Notice in Figure 5.8 that the reconstruction lacks the sharp crispness of the original. It is shown in chapter 6 that the spectrum of our reconstruction loses high frequencies by a scale of $1/|\omega|$. Philosophically, we can think of the hyperbola summation as integration, and integration

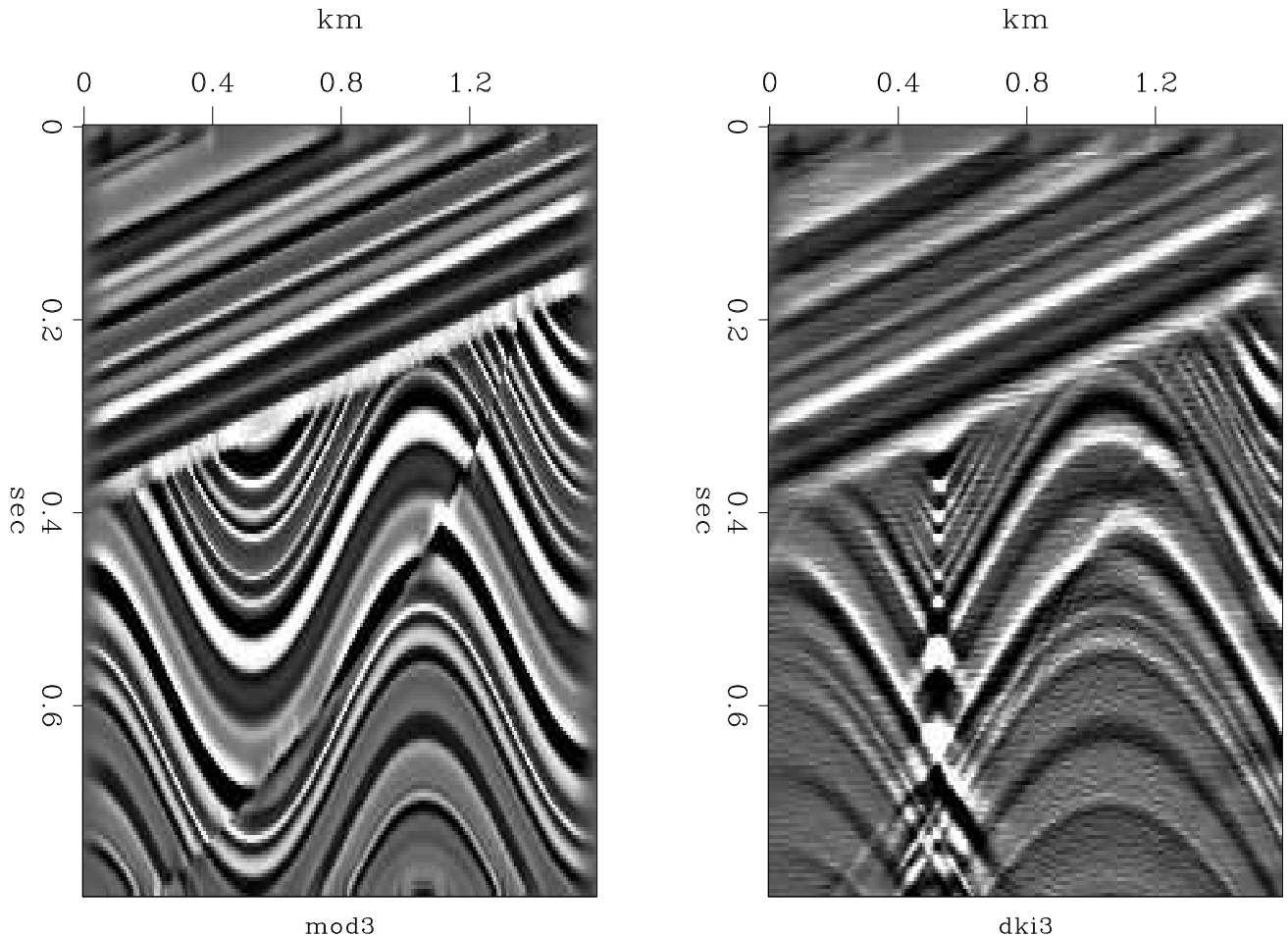


Figure 5.7: Left is the model. Right is diffraction to synthetic data. krch-kfgood [ER,M]

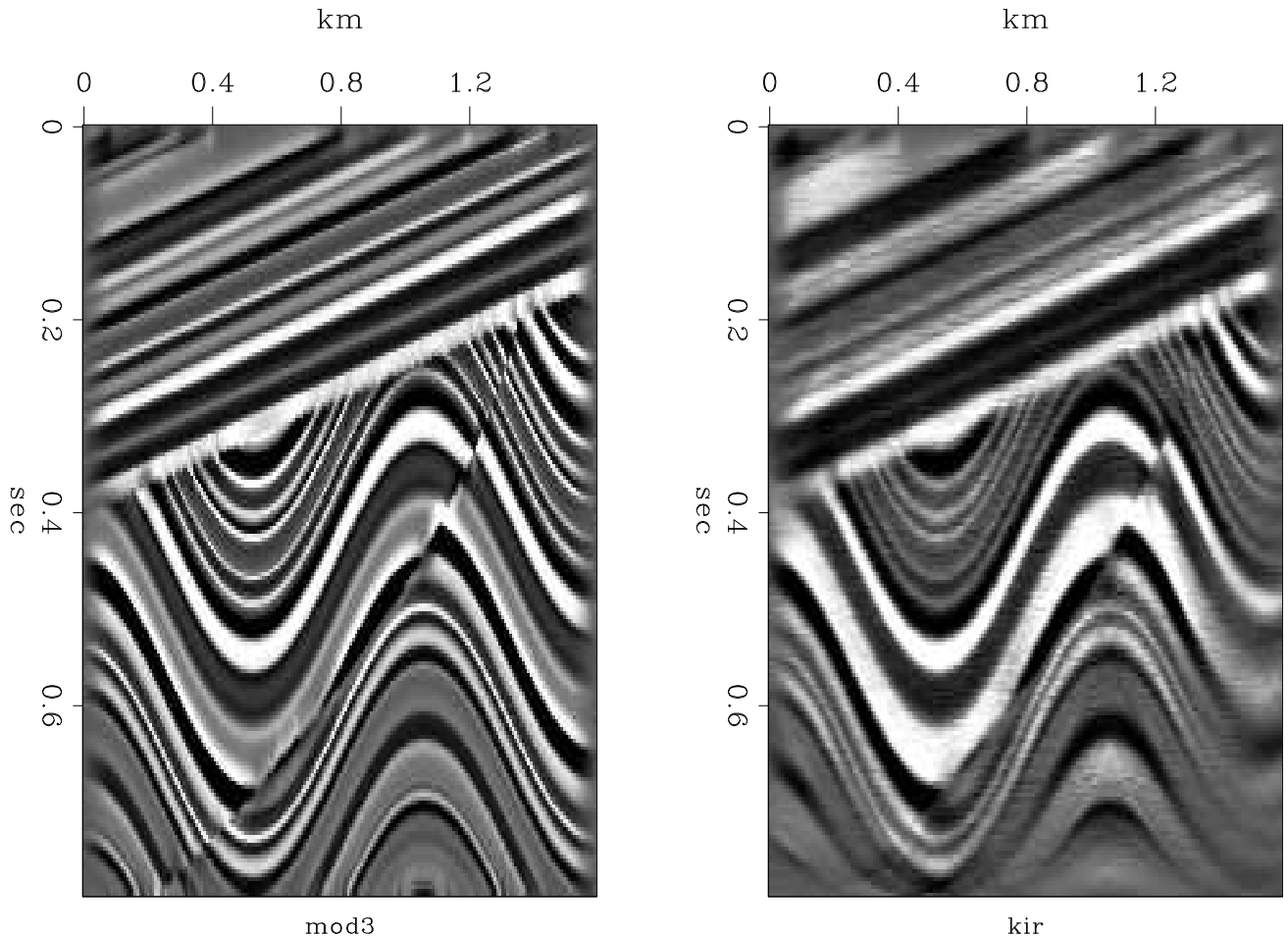


Figure 5.8: Left is the original model. Right is the reconstruction. krch-skmig [ER,M]

boosts low frequencies. Figure 5.9 shows the average over x of the relevant spectra. First, notice the high frequencies are weak because there is little high frequency en-

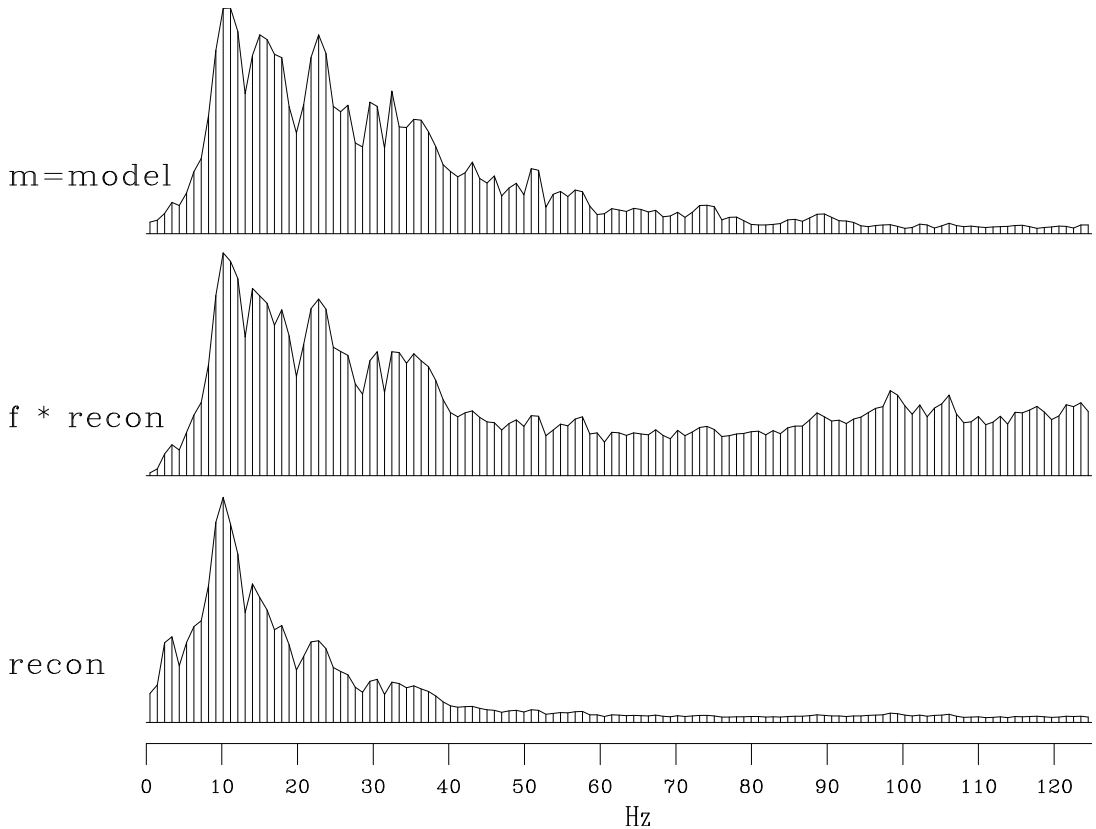


Figure 5.9: Top is the spectrum of the the model, i.e. the left side of Figure 5.8. Bottom is the spectrum of the the reconstruction, i.e. the right side of Figure 5.8. Middle is the reconstruction times frequency f. krch-kirspec [ER]

ergy in the original model. Then notice that our cavalier approach to interpolation

created more high frequency energy. Finally, notice that multiplying the spectrum of our migrated model by frequency, f, brought the important part of the spectral bands into agreement. This suggests applying an $|\omega|$ filter to our reconstruction, or $\sqrt{-i\omega}$ operator to both the modeling and the reconstruction, an idea implemented in subroutine `halfdifa()` /prog:halfdifa.

Neither of these Kirchhoff codes addresses the issue of spatial **alias**ing. Spatial aliasing is a vexing issue of numerical analysis. The Kirchhoff codes shown here do not work as expected unless the space mesh size is suitably more refined than the time mesh. Figure 5.10 shows an example of forward modeling with an x mesh of 50 and 100 points. (Previous figures used 200 points on space. All use 200 mesh points on the time.) Subroutine `kirchfast()` /prog:kirchfast does interpolation by moving values to the nearest neighbor of the theoretical location. Had we taken the trouble to interpolate the two nearest points, our results would have been a little better, but the basic problem (resolved in chapter 10) would remain.

5.2.4. Sampling and aliasing

*Spatial **alias**ing* means insufficient sampling of the data along the space axis. This

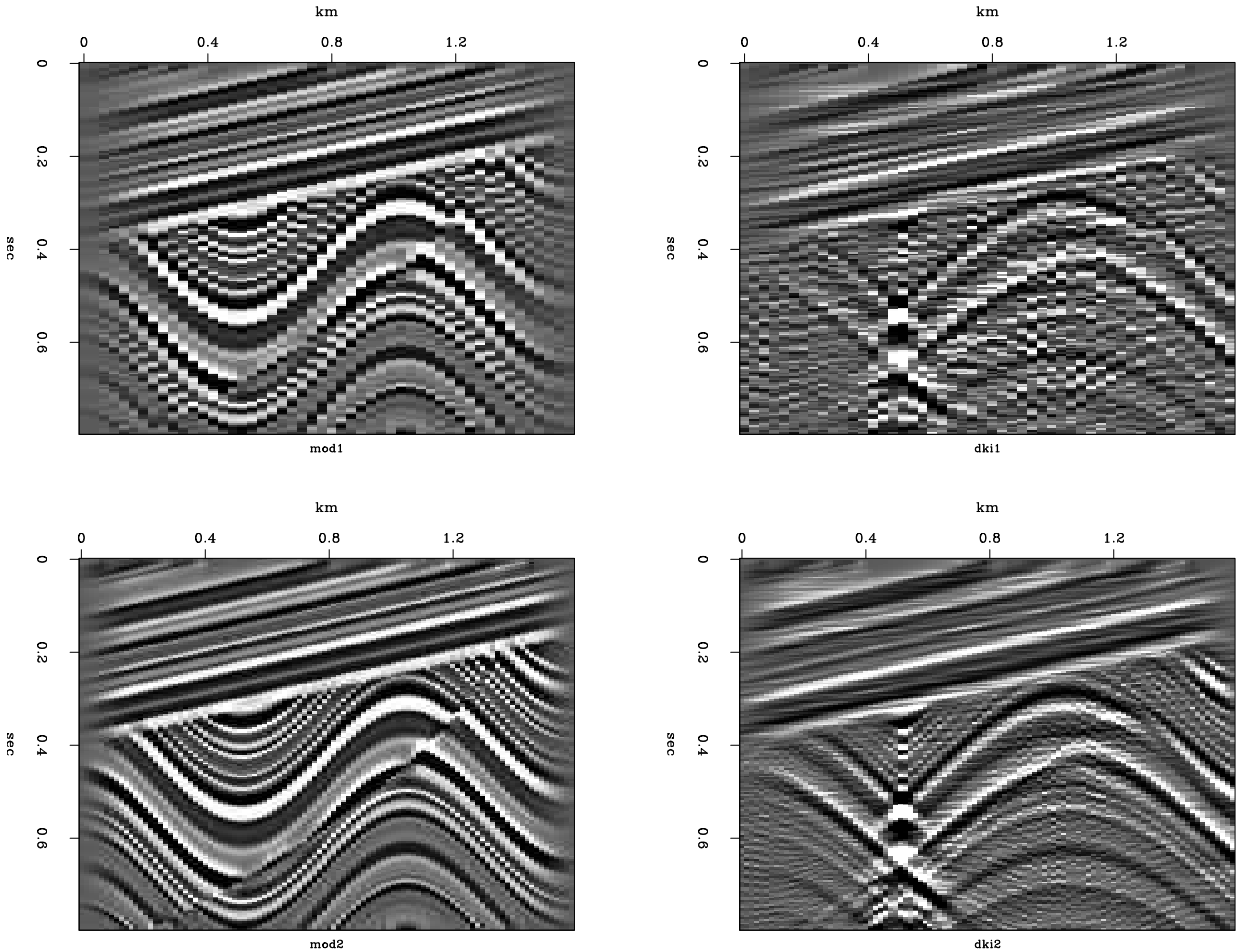


Figure 5.10: Left is model. Right is synthetic data from the model. Top has 50 points on the x-axis, bottom has 100. krch-skmod [ER]

difficulty is so universal, that all migration methods must consider it.

Data should be sampled at more than two points per wavelength. Otherwise the wave arrival direction becomes ambiguous. Figure 5.11 shows synthetic data that is sampled with insufficient density along the x-axis. You can see that the problem becomes more acute at high frequencies and steep dips.

There is no generally-accepted, automatic method for migrating spatially aliased data. In such cases, human beings may do better than machines, because of their skill in recognizing true slopes. When the data is adequately sampled however, computer migrations give better results than manual methods.

5.2.5. Kirchhoff migration of field data

Figure 5.12 shows migrated field data.

The on-line movie behind the figure shows the migration before and after amplitude gain with time. You can get a bad result if you gain up the data, say with automatic gain or with t^2, for display before doing the migration. What happens is that the hyperbola flanks are then included incorrectly with too much strength.

The proper approach is to gain it first with $\sqrt{t}$ which converts it from 3-D

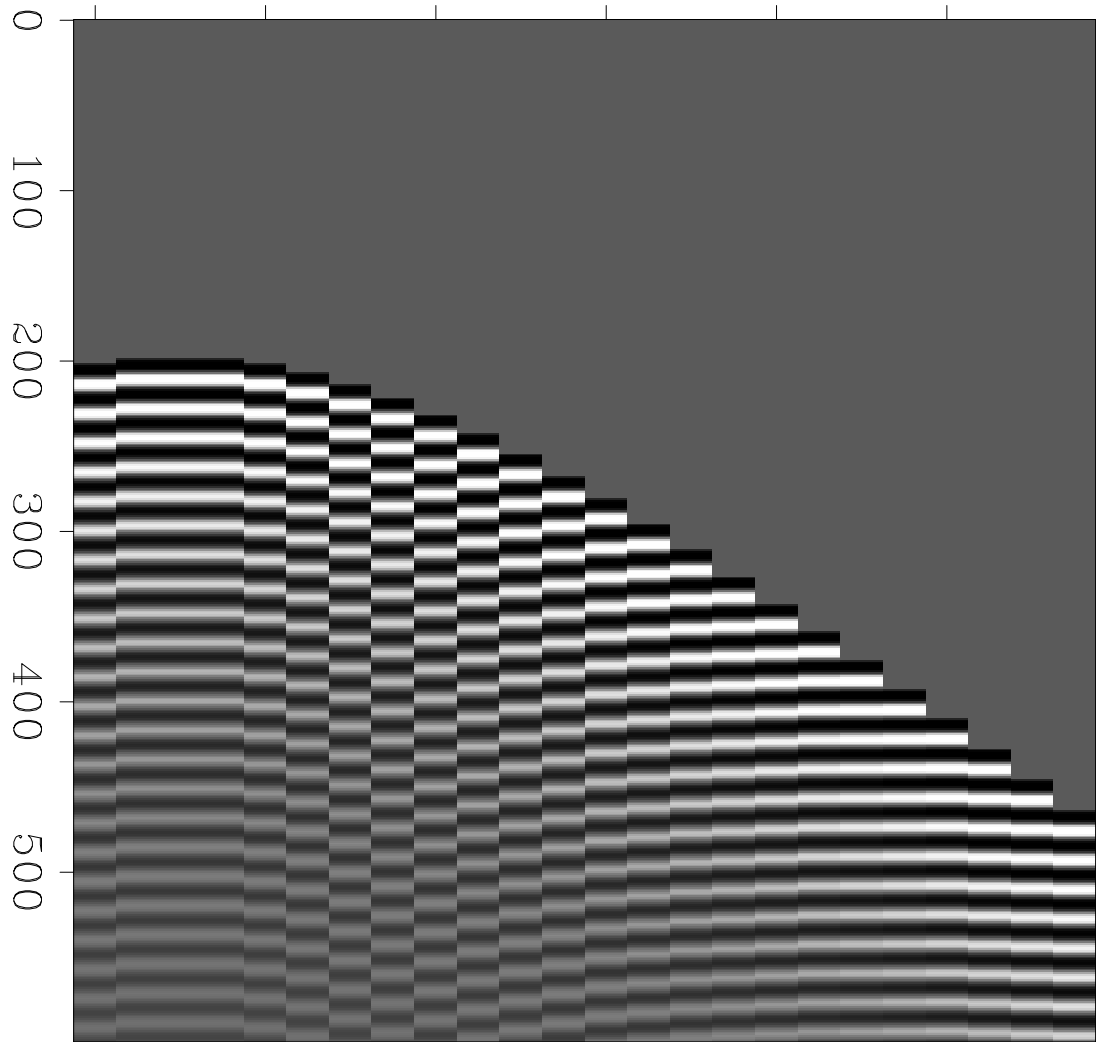


Figure 5.11: Insufficient spatial sampling of synthetic data. To better perceive the ambiguity of arrival angle, view the figures at a grazing angle from the side. krch-alias [ER]

wavefields to 2-D. Then migrate it with a 2-D migration like `kirchfast()`, and finally gain it further for display (because deep reflectors are usually weaker).

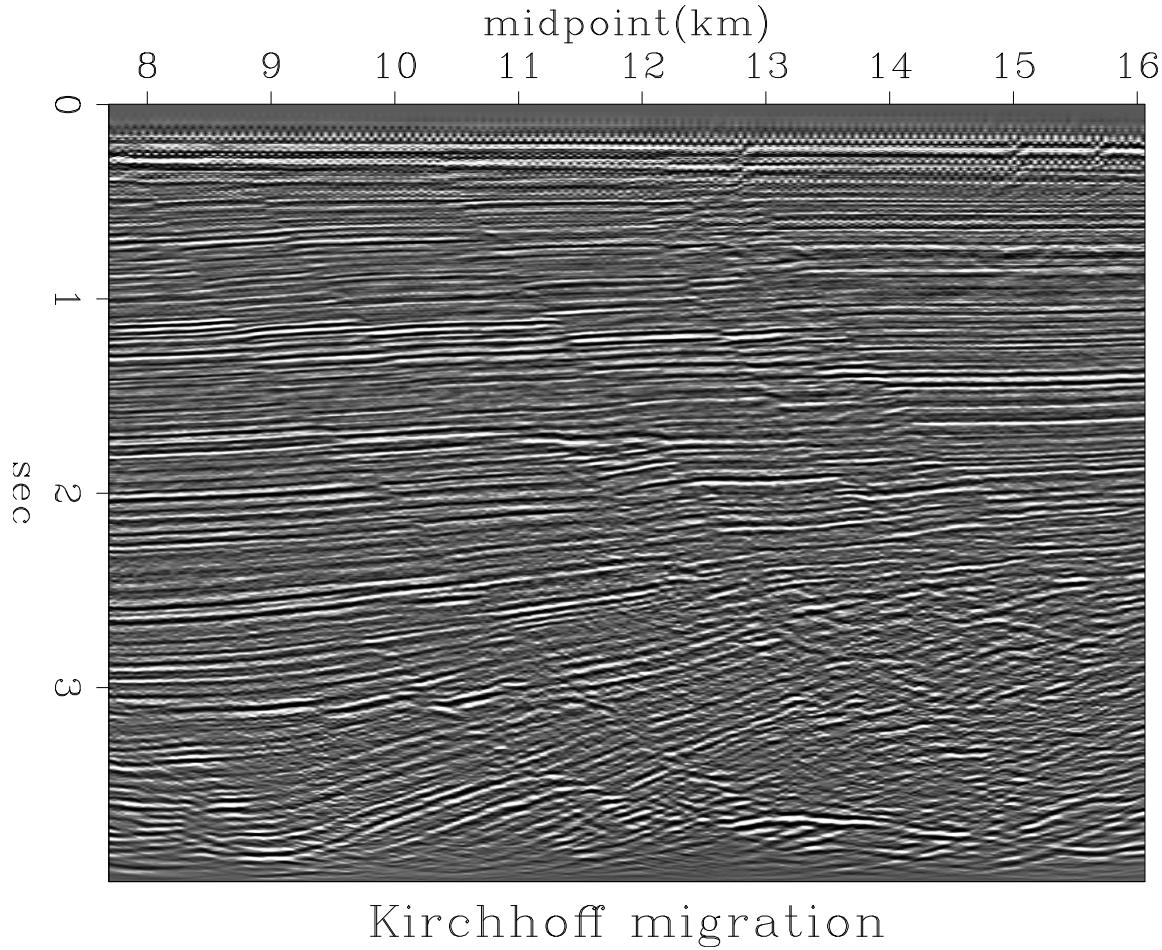


Figure 5.12: Kirchhoff migration of Figure 4.7. Press button for movie comparing stack to migrated stack. krch-wgkirch [ER,M]

Chapter 6

Waves and Fourier sums

An important concept in wave imaging is the extrapolation of a wavefield from one depth z to another. Fourier transforms are an essential basic tool. There are many books and chapters of books on the *theory* of Fourier transformation.

The first half of this chapter is an introduction to *practice* with Fourier sums. It assumes you already know something of the theory and takes you through the theory rather quickly emphasizing practice by examining examples, and by performing two-dimensional Fourier transformation of data and interpreting the result. For a somewhat more theoretical background, I suggest my previous book PVI at http://sepwww.stanford.edu/sep/prof/.

The second half of this chapter uses Fourier transformation to explain the Hankel waveform we observed in chapter 4 and chapter 5. Interestingly, it is the Fourier transform of $\sqrt{-i\omega}$, which is half the derivative operator.

6.1. FOURIER TRANSFORM

We first examine the two ways to visualize polynomial multiplication. The two ways lead us to the most basic principle of Fourier analysis that

> A product in the Fourier domain is a convolution in the physical domain

Look what happens to the coefficients when we multiply polynomials.

$$X(Z)\,B(Z) \quad = \quad Y(Z) \tag{6.1}$$

$$(x_0 + x_1 Z + x_2 Z^2 + \cdots)(b_0 + b_1 Z + b_2 Z^2) \quad = \quad y_0 + y_1 Z + y_2 Z^2 + \cdots \tag{6.2}$$

Identifying coefficients of successive powers of Z, we get

$$\begin{aligned} y_0 &= x_0 b_0 \\ y_1 &= x_1 b_0 + x_0 b_1 \\ y_2 &= x_2 b_0 + x_1 b_1 + x_0 b_2 \\ y_3 &= x_3 b_0 + x_2 b_1 + x_1 b_2 \\ y_4 &= x_4 b_0 + x_3 b_1 + x_2 b_2 \\ &= \cdots\cdots\cdots\cdots\cdots \end{aligned} \tag{6.3}$$

In matrix form this looks like

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} x_0 & 0 & 0 \\ x_1 & x_0 & 0 \\ x_2 & x_1 & x_0 \\ x_3 & x_2 & x_1 \\ x_4 & x_3 & x_2 \\ 0 & x_4 & x_3 \\ 0 & 0 & x_4 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} \tag{6.4}$$

The following equation, called the "convolution equation," carries the spirit of the group shown in (6.3)

$$y_k = \sum_{i=0} x_{k-i} b_i \tag{6.5}$$

The second way to visualize polynomial multiplication is simpler. Above we did not think of Z as a numerical value. Instead we thought of it as "a unit delay operator". Now we think of the product $X(Z)B(Z) = Y(Z)$ numerically. For all possible numerical values of Z, each value Y is determined from the product of the two numbers X and B. Instead of considering all possible numerical values we limit

ourselves to all values of unit magnitude $Z = e^{i\omega}$ for all real values of ω. This is Fourier analysis, a topic we consider next.

6.1.1. FT as an invertible matrix

A **Fourier sum** may be written

$$B(\omega) \quad = \quad \sum_t b_t \; e^{i\omega t} \quad = \quad \sum_t b_t \; Z^t \tag{6.6}$$

where the complex value Z is related to the real frequency ω by $Z = e^{i\omega}$. This Fourier sum is a way of building a continuous function of ω from discrete signal values b_t in the time domain. Here we specify both time and frequency domains by a set of points. Begin with an example of a signal that is nonzero at four successive instants, (b_0, b_1, b_2, b_3). The transform is

$$B(\omega) \quad = \quad b_0 + b_1 Z + b_2 Z^2 + b_3 Z^3 \tag{6.7}$$

The evaluation of this polynomial can be organized as a matrix times a vector, such as

$$\begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 \\ 1 & W^2 & W^4 & W^6 \\ 1 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{6.8}$$

Observe that the top row of the matrix evaluates the polynomial at $Z = 1$, a point where also $\omega = 0$. The second row evaluates $B_1 = B(Z = W = e^{i\omega_0})$, where ω_0 is some base frequency. The third row evaluates the Fourier transform for $2\omega_0$, and the bottom row for $3\omega_0$. The matrix could have more than four rows for more frequencies and more columns for more time points. I have made the matrix square in order to show you next how we can find the inverse matrix. The size of the matrix in (6.8) is $N = 4$. If we choose the base frequency ω_0 and hence W correctly, the inverse matrix will be

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = 1/N \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/W & 1/W^2 & 1/W^3 \\ 1 & 1/W^2 & 1/W^4 & 1/W^6 \\ 1 & 1/W^3 & 1/W^6 & 1/W^9 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix} \tag{6.9}$$

Multiplying the matrix of (6.9) with that of (6.8), we first see that the diagonals are +1 as desired. To have the off diagonals vanish, we need various sums, such as $1+W+W^2+W^3$ and $1+W^2+W^4+W^6$, to vanish. Every element (W^6, for example, or $1/W^9$) is a unit vector in the complex plane. In order for the sums of the unit vectors to vanish, we must ensure that the vectors pull symmetrically away from the origin. A uniform distribution of directions meets this requirement. In other words, W should be the N-th root of unity, i.e.,

$$W \quad = \quad \sqrt[N]{1} \quad = \quad e^{2\pi i/N} \tag{6.10}$$

The lowest frequency is zero, corresponding to the top row of (6.8). The next-to-the-lowest frequency we find by setting W in (6.10) to $Z = e^{i\omega_0}$. So $\omega_0 = 2\pi/N$; and for (6.9) to be inverse to (6.8), the frequencies required are

$$\omega_k \quad = \quad \frac{(0,1,2,\ldots,N-1)2\pi}{N} \tag{6.11}$$

6.1.2. The Nyquist frequency

The highest frequency in equation (6.11), $\omega = 2\pi(N-1)/N$, is almost 2π. This frequency is twice as high as the Nyquist frequency $\omega = \pi$. The **Nyquist frequency**

is normally thought of as the "highest possible" frequency, because $e^{i\pi t}$, for integer t, plots as $(\cdots, 1, -1, 1, -1, 1, -1, \cdots)$. The double Nyquist frequency function, $e^{i2\pi t}$, for integer t, plots as $(\cdots, 1, 1, 1, 1, 1, \cdots)$. So this frequency above the highest frequency is really zero frequency! We need to recall that $B(\omega) = B(\omega - 2\pi)$. Thus, all the frequencies near the upper end of the range equation (6.11) are really small negative frequencies. Negative frequencies on the interval $(-\pi, 0)$ were moved to interval $(\pi, 2\pi)$ by the matrix form of Fourier summation.

A picture of the Fourier transform matrix is shown in Figure 6.1. Notice the Nyquist frequency is the center row and center column of each matrix.

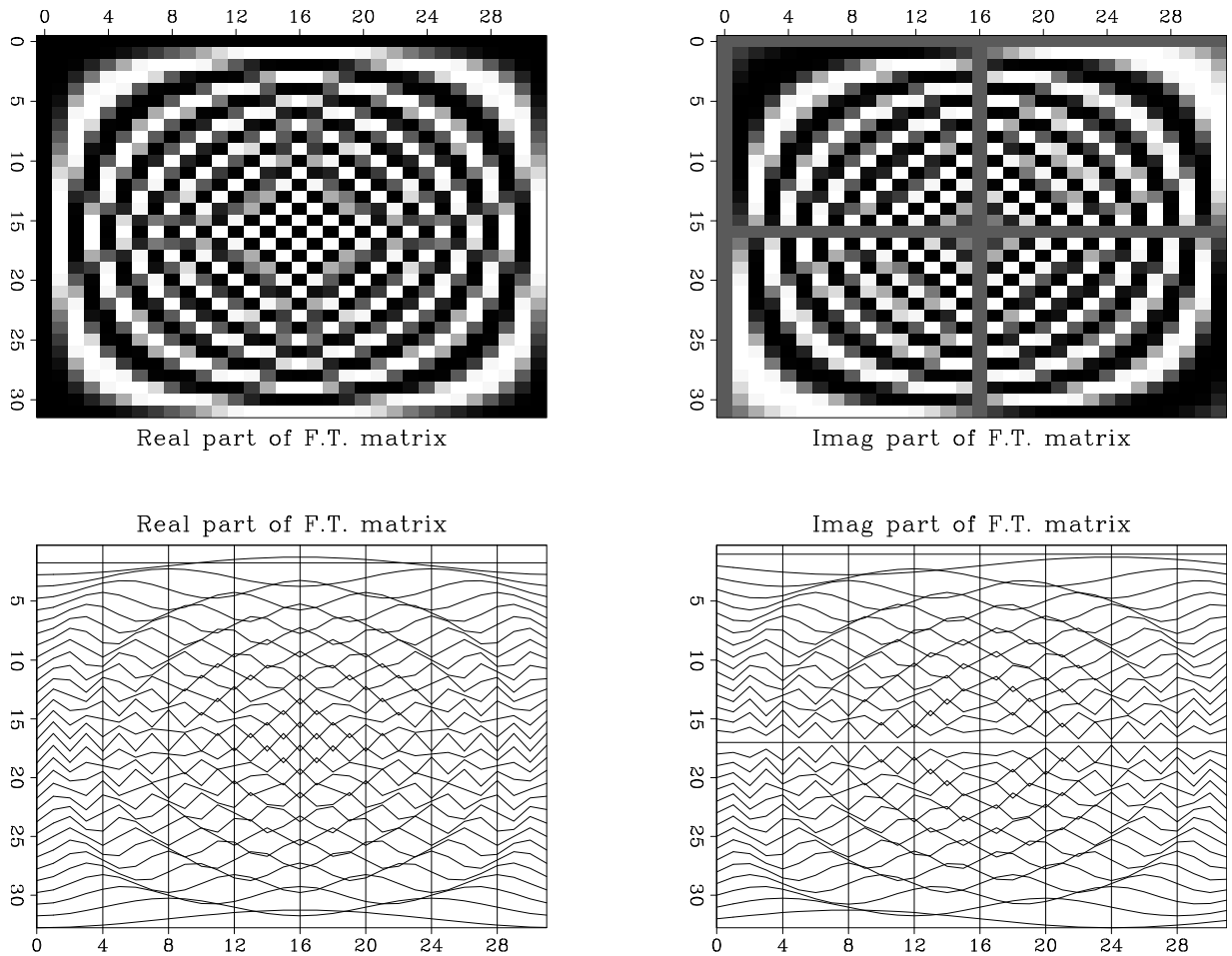


Figure 6.1: Two different graphical means of showing the real and imaginary parts of the Fourier transform matrix of size 32×32. ft1-matrix [ER]

6.1.3. Laying out a mesh

In theoretical work and in programs, the unit delay operator definition $Z = e^{i\omega\Delta t}$ is often simplified to $\Delta t = 1$, leaving us with $Z = e^{i\omega}$. How do we know whether ω is given in radians per second or radians per sample? We may not invoke a cosine or an exponential unless the argument has no physical dimensions. So where we see ω without Δt, we know it is in units of radians per sample.

In practical work, frequency is typically given in cycles/sec or **Hertz**, f, rather than radians, ω (where $\omega = 2\pi f$). Here we will now switch to f. We will design a computer **mesh** on a physical object (such as a waveform or a function of space). We often take the mesh to begin at $t = 0$, and continue till the end $t_{\max}$ of the object, so the time range $t_{\text{range}} = t_{\max}$. Then we decide how many points we want to use. This will be the N used in the discrete Fourier-transform program. Dividing the range by the number gives a mesh interval Δt.

Now let us see what this choice implies in the frequency domain. We customarily take the maximum frequency to be the Nyquist, either $f_{\max} = .5/\Delta t$ Hz or $\omega_{\max} = \pi/\Delta t$ radians/sec. The frequency range f_{range} goes from $-.5/\Delta t$ to $.5/\Delta t$. In summary:

- $\Delta t \quad = \quad t_{\text{range}}/N \quad$ is time **resolution**.

- $f_{\text{range}} = 1/\Delta t = N/t_{\text{range}}$ is frequency range.
- $\Delta f = f_{\text{range}}/N = 1/t_{\text{range}}$ is frequency **resolution**.

In principle, we can always increase N to refine the calculation. Notice that increasing N sharpens the time resolution (makes Δt smaller) but does not sharpen the frequency resolution Δf, which remains fixed. Increasing N increases the frequency *range*, but not the frequency *resolution*.

What if we want to increase the frequency resolution? Then we need to choose t_{range} larger than required to cover our object of interest. Thus we either record data over a larger range, or we assert that such measurements would be zero. Three equations summarize the facts:

$$\Delta t \; f_{\text{range}} = 1 \tag{6.12}$$

$$\Delta f \; t_{\text{range}} = 1 \tag{6.13}$$

$$\Delta f \; \Delta t = \frac{1}{N} \tag{6.14}$$

Increasing *range* in the time domain increases *resolution* in the frequency domain and vice versa. Increasing **resolution** in one domain does not increase **resolution** in the other.

6.2. INVERTIBLE SLOW FT PROGRAM

Typically, signals are real valued. But the programs in this chapter are for complex-valued signals. In order to use these programs, copy the real-valued signal into a complex array, where the signal goes into the real part of the complex numbers; the imaginary parts are then automatically set to zero.

There is no universally correct choice of **scale factor** in Fourier transform: choice of scale is a matter of convenience. Equations (6.8) and (6.9) mimic the Z-transform, so their scaling factors are convenient for the convolution theorem—that a product in the frequency domain is a convolution in the time domain. Obviously, the scaling factors of equations (6.8) and (6.9) will need to be interchanged for the complementary theorem that a convolution in the frequency domain is a product in the time domain. I like to use a scale factor that keeps the sums of squares the same

```
subroutine scale( factor, n, data)
integer i,                n
real factor, data(n)
do i= 1, n
        data(i) = factor * data(i)
return; end
```

Back

in the time domain as in the frequency domain. Since I almost never need the scale factor, it simplifies life to omit it from the subroutine argument list. When a scaling program is desired, we can use a simple one like `scale()` /prog:scale. Complex-valued data can be scaled with `scale()` merely by doubling the value of `n`. scale

6.2.1. The simple FT code

Subroutine `simpleft()` /prog:simpleft exhibits features found in many physics and engineering programs. For example, the time-domain signal (which is denoted "`tt()`"), has `nt` values subscripted, from `tt(1)` to `tt(nt)`. The first value of this signal `tt(1)` is located in real physical time at `t0`. The time interval between values is `dt`. The value of `tt(it)` is at time `t0+(it-1)*dt`. We do not use "`if`" as a pointer on the frequency axis because `if` is a keyword in most programming languages. Instead, we count along the frequency axis with a variable named `ie`. simpleft

```
subroutine simpleft( adj, add, t0,dt,tt,nt,  f0,df,  ff,nf)
integer it,ie,       adj, add,          nt,          nf
complex cexp, cmplx,                 tt(nt),         ff(nf)
real pi2, freq, time, scale,   t0,dt,         f0,df
call adjnull(        adj, add,       tt,nt*2,        ff,nf*2 )
                        pi2= 2. * 3.14159265;          scale = 1./sqrt( 1.*nt)
                        df = (1./dt) / nf
                        f0 =   - .5/dt
do ie = 1, nf {   freq= f0 + df*(ie-1)
do it = 1, nt {   time= t0 + dt*(it-1)
        if( adj == 0 )
                ff(ie)= ff(ie) + tt(it) * cexp(cmplx(0., pi2*freq*time)) * scale
        else
                tt(it)= tt(it) + ff(ie) * cexp(cmplx(0.,-pi2*freq*time)) * scale
        }}
return; end
```

Back

The total frequency band is 2π radians per sample unit or $1/\Delta t$ Hz. Dividing the total interval by the number of points `nf` gives Δf. We could choose the frequencies to run from 0 to 2π radians/sample. That would work well for many applications, but it would be a nuisance for applications such as differentiation in the frequency domain, which require multiplication by $-i\omega$ including the **negative frequencies** as well as the positive. So it seems more natural to begin at the most negative frequency and step forward to the most positive frequency.

6.3. CORRELATION AND SPECTRA

The spectrum of a signal is a positive function of frequency that says how much of each tone is present. The Fourier transform of a spectrum yields an interesting function called an "**autocorrelation**," which measures the similarity of a signal to itself shifted.

6.3.1. Spectra in terms of Z-transforms

Let us look at spectra in terms of Z-transforms. Let a **spectrum** be denoted $S(\omega)$, where

$$S(\omega) \quad = \quad |B(\omega)|^2 \quad = \quad \overline{B(\omega)}B(\omega) \tag{6.15}$$

Expressing this in terms of a three-point Z-transform, we have

$$S(\omega) \quad = \quad (\bar{b}_0 + \bar{b}_1 e^{-i\omega} + \bar{b}_2 e^{-i2\omega})(b_0 + b_1 e^{i\omega} + b_2 e^{i2\omega}) \tag{6.16}$$

$$S(Z) \quad = \quad \left(\bar{b}_0 + \frac{\bar{b}_1}{Z} + \frac{\bar{b}_2}{Z^2}\right)(b_0 + b_1 Z + b_2 Z^2) \tag{6.17}$$

$$S(Z) \quad = \quad \bar{B}\left(\frac{1}{Z}\right) B(Z) \tag{6.18}$$

It is interesting to multiply out the polynomial $\bar{B}(1/Z)$ with $B(Z)$ in order to examine the coefficients of $S(Z)$:

$$S(Z) \quad = \quad \frac{\bar{b}_2 b_0}{Z^2} + \frac{(\bar{b}_1 b_0 + \bar{b}_2 b_1)}{Z} + (\bar{b}_0 b_0 + \bar{b}_1 b_1 + \bar{b}_2 b_2) + (\bar{b}_0 b_1 + \bar{b}_1 b_2)Z + \bar{b}_0 b_2 Z^2$$

$$S(Z) \quad = \quad \frac{s_{-2}}{Z^2} + \frac{s_{-1}}{Z} + s_0 + s_1 Z + s_2 Z^2 \tag{6.19}$$

The coefficient s_k of Z^k is given by

$$s_k = \sum_i \bar{b_i} b_{i+k} \tag{6.20}$$

Equation (6.20) is the **autocorrelation** formula. The autocorrelation value s_k at lag 10 is s_{10}. It is a measure of the similarity of b_i with itself shifted 10 units in time. In the most frequently occurring case, b_i is real; then, by inspection of (6.20), we see that the autocorrelation coefficients are real, and $s_k = s_{-k}$.

Specializing to a real time series gives

$$S(Z) = s_0 + s_1\left(Z + \frac{1}{Z}\right) + s_2\left(Z^2 + \frac{1}{Z^2}\right) \tag{6.21}$$

$$S(Z(\omega)) = s_0 + s_1(e^{i\omega} + e^{-i\omega}) + s_2(e^{i2\omega} + e^{-i2\omega}) \tag{6.22}$$

$$S(\omega) = s_0 + 2s_1\cos\omega + 2s_2\cos 2\omega \tag{6.23}$$

$$S(\omega) = \sum_k s_k \cos k\omega \tag{6.24}$$

$$S(\omega) = \text{cosine transform of } \; s_k \tag{6.25}$$

This proves a classic theorem that for real-valued signals can be simply stated as follows:

> For any real signal, the cosine transform of the **autocorrelation** equals the magnitude squared of the Fourier transform.

6.3.2. Two ways to compute a spectrum

There are two computationally distinct methods by which we can compute a spectrum: (1) compute all the s_k coefficients from (6.20) and then form the cosine sum (6.24) for each ω; and alternately, (2) evaluate $B(Z)$ for some value of Z on the unit circle, and multiply the resulting number by its complex conjugate. Repeat for many values of Z on the unit circle. When there are more than about twenty lags, method (2) is cheaper, because the fast Fourier transform (coming up soon) can be used.

6.3.3. Common signals

Figure 6.2 shows some common signals and their **autocorrelation**s. Figure 6.3 shows the cosine transforms of the autocorrelations. Cosine transform takes us from time to frequency and it also takes us from frequency to time. Thus, transform pairs in Figure 6.3 are sometimes more comprehensible if you interchange time and frequency. The various signals are given names in the figures, and a description of each follows:

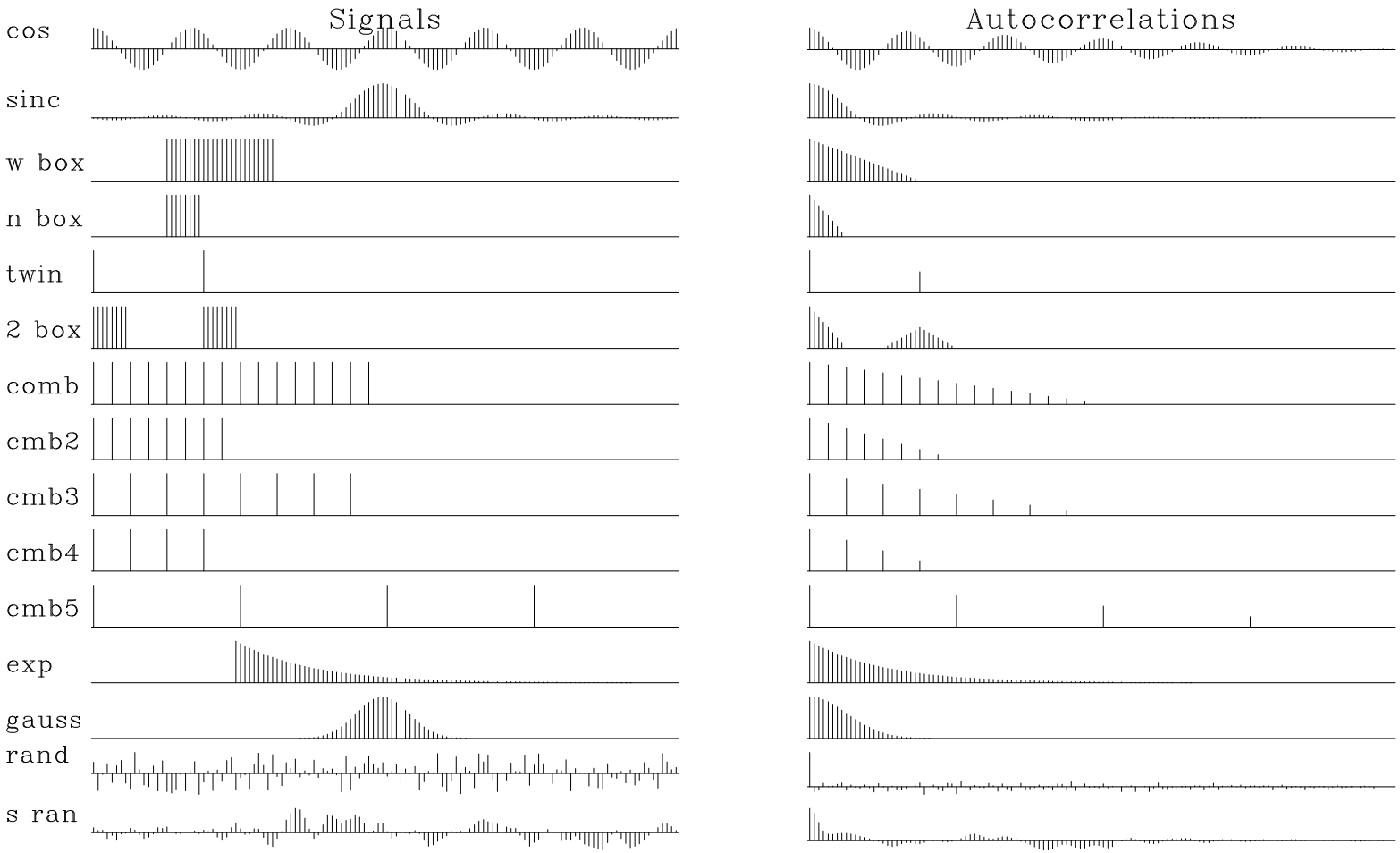


Figure 6.2: Common signals and one side of their autocorrelations. ft1-autocor [ER]

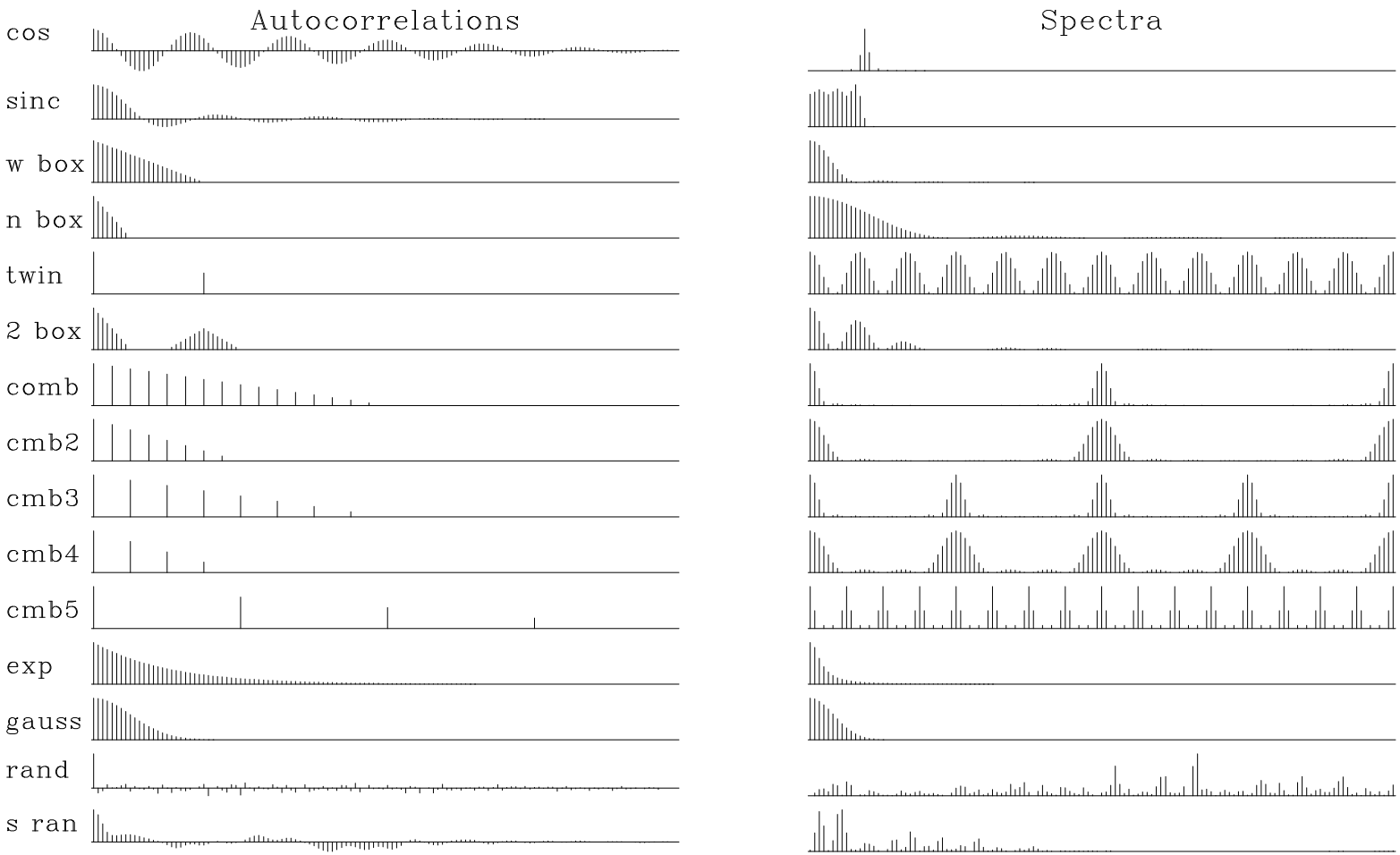


Figure 6.3: Autocorrelations and their cosine transforms, i.e., the (energy) spectra of the common signals. ft1-spectra [ER]

cos The theoretical spectrum of a sinusoid is an impulse, but the sinusoid was truncated (multiplied by a rectangle function). The autocorrelation is a sinusoid under a triangle, and its spectrum is a broadened impulse (which can be shown to be a narrow sinc-squared function).

sinc The **sinc** function is $\sin(\omega_0 t)/(\omega_0 t)$. Its autocorrelation is another sinc function, and its spectrum is a rectangle function. Here the rectangle is corrupted slightly by "**Gibbs sidelobes**," which result from the time truncation of the original sinc.

wide box A wide **rectangle function** has a wide triangle function for an autocorrelation and a narrow sinc-squared spectrum.

narrow box A narrow rectangle has a wide sinc-squared spectrum.

twin Two pulses.

2 boxes Two separated narrow boxes have the spectrum of one of them, but this spectrum is modulated (multiplied) by a sinusoidal function of frequency, where the modulation frequency measures the time separation of the narrow

boxes. (An oscillation seen in the frequency domain is sometimes called a "**quefrency**.")

comb Fine-toothed-**comb** functions are like rectangle functions with a lower Nyquist frequency. Coarse-toothed-comb functions have a spectrum which is a fine-toothed comb.

exponential The autocorrelation of a transient **exponential** function is a **double-sided exponential** function. The spectrum (energy) is a Cauchy function, $1/(\omega^2 + \omega_0^2)$. The curious thing about the **Cauchy function** is that the amplitude spectrum diminishes inversely with frequency to the *first* power; hence, over an infinite frequency axis, the function has infinite integral. The sharp edge at the onset of the transient exponential has much high-frequency energy.

Gauss The autocorrelation of a **Gaussian** function is another Gaussian, and the spectrum is also a Gaussian.

random **Random** numbers have an autocorrelation that is an impulse surrounded by some short grass. The spectrum is positive random numbers.

smoothed random Smoothed random numbers are much the same as random numbers, but their spectral bandwidth is limited.

6.4. SETTING UP THE FAST FOURIER TRANSFORM

Typically we Fourier transform seismograms about a thousand points long. Under these conditions another Fourier summation method works about a hundred times faster than those already given. Unfortunately, the faster Fourier transform program is not so transparently clear as the programs given earlier. Also, it is slightly less flexible. The speedup is so overwhelming, however, that the fast program is always used in routine work.

Flexibility may be lost because the basic fast program works with complex-valued signals, so we ordinarily convert our real signals to complex ones (by adding a zero imaginary part). More flexibility is lost because typical fast FT programs require the data length to be an integral power of 2. Thus geophysical datasets often have zeros appended (a process called "**zero pad**ding") until the data length is a

```
integer function pad2( n )
integer n
pad2 = 1
while( pad2 < n )
        pad2 = pad2 * 2
return; end
```

Back

power of 2. From time to time I notice clumsy computer code written to deduce a number that is a power of 2 and is larger than the length of a dataset. An answer is found by rounding up the logarithm to base 2. The more obvious and the quicker way to get the desired value, however, is with the simple Fortran function `pad2()`. pad2

How fast is the fast Fourier transform method? The answer depends on the size of the data. The matrix times vector operation in (6.8) requires N^2 multiplications and additions. That determines the speed of the slow transform. For the fast method the number of adds and multiplies is proportional to $N \log_2 N$. Since $2^{10} = 1024$, the speed ratio is typically 1024/10 or about 100. In reality, the fast method is not quite that fast, depending on certain details of overhead and implementation.

Below is `ftu()`, a version of the **fast Fourier transform** program. There are many versions of the program—I have chosen this one for its simplicity. Considering the complexity of the task, it is remarkable that no auxiliary memory vectors are required; indeed, the output vector lies on top of the input vector. To run this program, your first step might be to copy your real-valued signal into a complex-valued array. Then append enough zeros to fill in the remaining space. ftu

The following two lines serve to Fourier transform a vector of 1024 complex-

```
subroutine ftu( signi, nx, cx )
#   complex fourier transform with unitary scaling
#
#               1         nx          signi*2*pi*i*(j-1)*(k-1)/nx
#   cx(k)  =  -------- * sum cx(j) * e
#             sqrt(nx)   j=1             for k=1,2,...,nx=2**integer
#
integer nx, i, j, k, m, istep, pad2
real    signi, scale, arg
complex cx(nx), cmplx, cw, cdel, ct
if( nx != pad2(nx) )    call erexit('ftu: nx not a power of 2')
scale = 1. / sqrt( 1.*nx)
do i= 1, nx
        cx(i) = cx(i) * scale
j = 1;  k = 1
do i= 1, nx {
        if (i<=j) { ct = cx(j); cx(j) = cx(i); cx(i) = ct }
        m = nx/2
        ! "&&" means .AND.
        while (j>m && m>1) { j = j-m; m = m/2 }
        j = j+m
        }
repeat {
        istep = 2*k;   cw = 1.;   arg = signi*3.14159265/k
        cdel = cmplx( cos(arg), sin(arg))
        do m= 1, k {
                do i= m, nx, istep
                        { ct=cw*cx(i+k);  cx(i+k)=cx(i)-ct;  cx(i)=cx(i)+ct }
                cw = cw * cdel
                }
        k = istep
        if(k>=nx) break
        }
return; end
```

Back

valued points, and then to **inverse Fourier transform** them back to the original data:

```
call ftu(  1., 1024, cx)
call ftu( -1., 1024, cx)
```

A reference given at the end of this chapter contains many other versions of the FFT program. One version transforms real-valued signals to complex-valued frequency functions in the interval $0 \leq \omega < \pi$. Others that do not transform data on top of itself may be faster with specialized computer architectures.

6.4.1. Shifted spectrum

Subroutine `simpleft()` /prog:simpleft sets things up in a convenient manner: The frequency range runs from minus Nyquist up to (but not including) plus Nyquist. Thus there is no problem with the many (but not all) user programs that have trouble with aliased frequencies. Subroutine `ftu()` /prog:ftu, however has a frequency range from zero to double the Nyquist. Let us therefore define a friendlier "front end" to `ftu()` which looks more like `simpleft()`.

Recall that a time shift of t_0 can be implemented in the Fourier domain by multiplication by $e^{-i\omega t_0}$. Likewise, in the Fourier domain, the frequency interval used by subroutine `ftu()` /prog:ftu, namely, $0 \le \omega < 2\pi$, can be shifted to the friendlier interval $-\pi \le \omega < \pi$ by a weighting function in the time domain. That weighting function is $e^{-i\omega_0 t}$ where ω_0 happens to be the Nyquist frequency, i.e. alternate points on the time axis are to be multiplied by -1. A subroutine for this purpose is `fth()`. fth

```
# FT a vector in a matrix, with first omega = - pi
#
subroutine  fth( adj,sign, m1, n12, cx)
integer i,       adj,      m1, n12
real sign
complex cx(m1,n12)
temporary complex temp(n12)
do i= 1, n12
        temp(i) = cx(1,i)
if( adj == 0)   { do i= 2, n12, 2
                        temp(i) =  -temp(i)
                  call ftu(  sign, n12, temp)
                }
else            { call ftu( -sign, n12, temp)
                  do i= 2, n12, 2
                        temp(i) =  -temp(i)
                }
do i= 1, n12
        cx(1,i) = temp(i)
return; end
```

Back

To Fourier transform a 1024-point complex vector `cx(1024)` and then inverse transform it, we would write

```
call fth( 0, 1., 1, 1024, cx)
call fth( 1, 1., 1, 1024, cx)
```

You might wonder about the apparent redundancy of using both the argument `adj` and the argument `sign`. Having two arguments instead of one allows us to define the *forward* transform for a *time* axis with the opposite sign as the forward transform for a *space* axis.

The subroutine `fth()` is somewhat cluttered by the inclusion of a frequently needed practical feature—namely, the facility to extract vectors from a matrix, transform the vectors, and then restore them into the matrix.

6.5. SETTING UP 2-D FT

The program `fth()` is set up so that the vectors transformed can be either rows or columns of a two-dimensional array. In any computer language there is a way to extract a vector (column or row) from a matrix. In some languages the vector can

```
# 1D Fourier transform on a 2D data set along the 1-axis
#
subroutine  ft1axis( adj, sign1, n1,n2, cx)
integer i2,          adj,        n1,n2
complex cx(n1,n2)
real sign1
do i2= 1, n2
                call fth( adj, sign1, 1,n1, cx(1,i2))
return; end
```

Back

```
# 1D Fourier transform on a 2D data set along the 2-axis
#
subroutine  ft2axis( adj, sign2, n1,n2, cx)
integer i1,          adj,        n1,n2
complex cx(n1,n2)
real    sign2
do i1= 1, n1
                call fth( adj, sign2, n1,n2, cx(i1,1))
return; end
```

Back

be processed directly without extraction. To see how this works in **Fortran**, recall a matrix allocated as `(n1,n2)` can be subscripted as a matrix `(i1,i2)` or as a long vector `(i1 + n1*(i2-1),1)`, and `call sub(x(i1,i2))` passes the subroutine a pointer to the `(i1,i2)` element. To transform an entire axis, the subroutines `ft1axis()` and `ft2axis()` are given. For a two-dimensional FT, we simply call both `ft1axis()` and `ft2axis()` in either order. ft1axis ft2axis

6.5.1. Basics of two-dimensional Fourier transform

Let us review some basic facts about **two-dimensional Fourier transform**. A two-dimensional function is represented in a computer as numerical values in a matrix, whereas a one-dimensional Fourier transform in a computer is an operation on a vector. A 2-D Fourier transform can be computed by a sequence of 1-D Fourier transforms. We can first transform each column vector of the matrix and then each row vector of the matrix. Alternately, we can first do the rows and later do the

columns. This is diagrammed as follows:

$$
\begin{array}{ccc}
p(t,\ x) & \longleftrightarrow & P(t,\ k_x) \\
\updownarrow & & \updownarrow \\
P(\omega,\ x) & \longleftrightarrow & P(\omega,\ k_x)
\end{array}
$$

The diagram has the notational problem that we cannot maintain the usual convention of using a lower-case letter for the domain of physical space and an upper-case letter for the Fourier domain, because that convention cannot include the mixed objects $P(t,k_x)$ and $P(\omega,x)$. Rather than invent some new notation, it seems best to let the reader rely on the context: the arguments of the function must help name the function.

An example of **two-dimensional Fourier transform**s on typical deep-ocean data is shown in Figure 6.4.

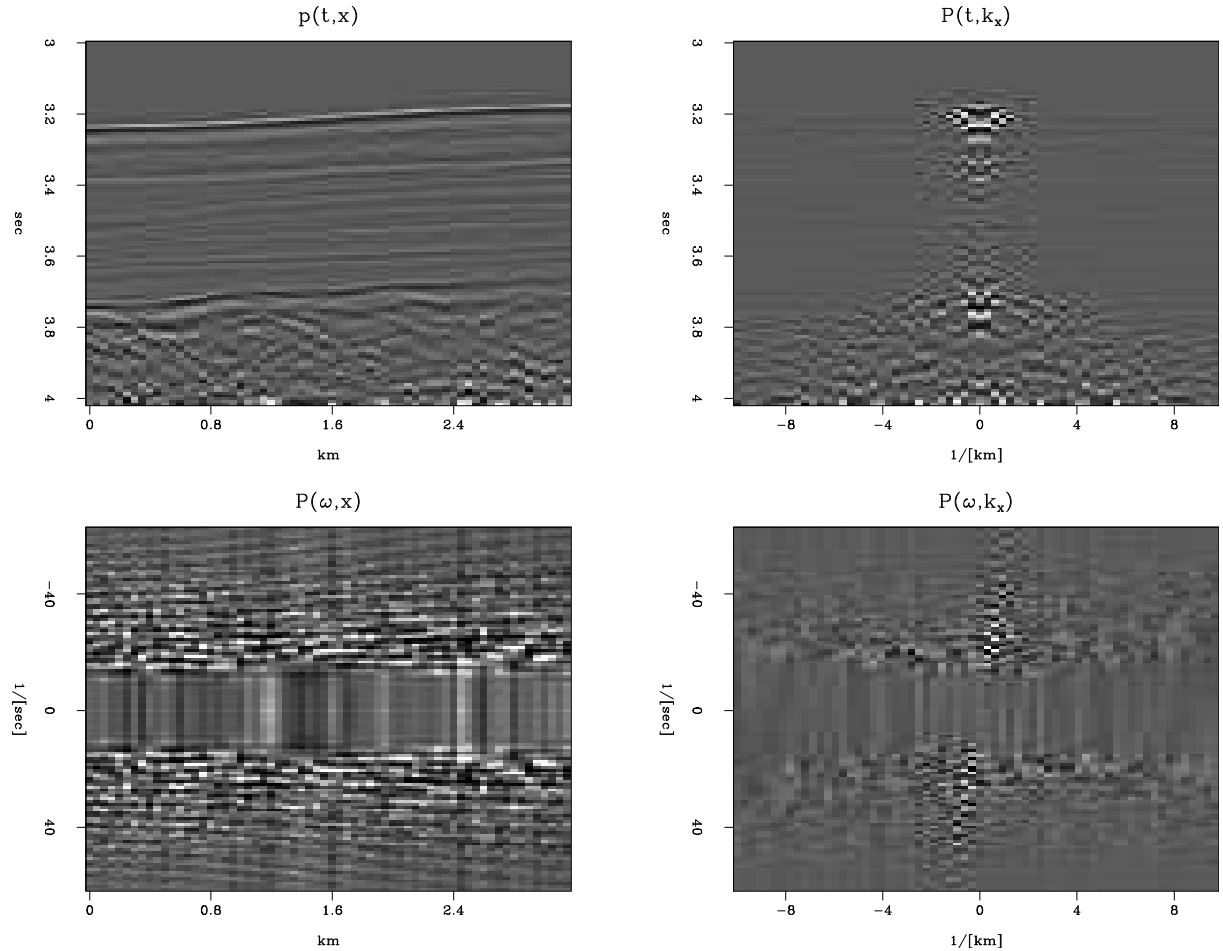


Figure 6.4: A deep-marine dataset $p(t,x)$ from Alaska (U.S. Geological Survey) and the *real* part of various Fourier transforms of it. Because of the long traveltime through the water, the time axis does not begin at $t = 0$. ft1-plane4 [ER]

In the deep ocean, sediments are fine-grained and deposit slowly in flat, regular, horizontal beds. The lack of permeable rocks such as sandstone severely reduces the potential for petroleum production from the deep ocean. The fine-grained shales overlay irregular, igneous, **basement rock**s. In the plot of $P(t,k_x)$, the lateral continuity of the sediments is shown by the strong spectrum at low k_x. The igneous rocks show a k_x spectrum extending to such large k_x that the deep data may be somewhat **spatially aliased** (sampled too coarsely). The plot of $P(\omega,x)$ shows that the data contains no low-frequency energy. The dip of the sea floor shows up in (ω,k_x)-space as the energy crossing the origin at an angle.

Altogether, the **two-dimensional Fourier transform** of a collection of seismograms involves only twice as much computation as the one-dimensional Fourier transform of each seismogram. This is lucky. Let us write some equations to establish that the asserted procedure does indeed do a 2-D Fourier transform. Say first that any function of x and t may be expressed as a superposition of sinusoidal functions:

$$p(t,x) \;=\; \int\int e^{-i\omega t + ik_x x}\; P(\omega,k_x)\, d\omega\, dk_x \tag{6.26}$$

The double integration can be nested to show that the temporal transforms are done

first (inside):

$$\begin{aligned} p(t,x) &= \int e^{i\,k_x x} \left[\int e^{-i\omega t}\; P(\omega,k_x)\, d\omega \right] dk_x \\ &= \int e^{i\,k_x x}\; P(t,k_x)\, dk_x \end{aligned}$$

The quantity in brackets is a Fourier transform over ω done for each and every k_x. Alternately, the nesting could be done with the k_x-integral on the inside. That would imply rows first instead of columns (or vice versa). It is the separability of $\exp(-i\omega t + i\,k_x x)$ into a product of exponentials that makes the computation easy and cheap.

6.5.2. Signs in Fourier transforms

In Fourier transforming t-, x-, and z-coordinates, we must choose a sign convention for each coordinate. Of the two alternative **sign convention**s, electrical engineers have chosen one and physicists another. While both have good reasons for their choices, our circumstances more closely resemble those of physicists, so we will

use their convention. For the *inverse* Fourier transform, our choice is

$$p(t,x,z) \quad = \quad \int\int\int e^{-i\omega t + ik_x x + ik_z z}\, P(\omega,k_x,k_z)\, d\omega\, dk_x\, dk_z \qquad (6.27)$$

For the *forward* Fourier transform, the space variables carry a *negative* sign, and time carries a *positive* sign.

Let us see the reasons why electrical engineers have made the opposite choice, and why we go with the physicists. Essentially, engineers transform only the time axis, whereas physicists transform both time and space axes. Both are simplifying their lives by their choice of sign convention, but physicists complicate their time axis in order to simplify their many space axes. The engineering choice minimizes the number of minus signs associated with the time axis, because for engineers, d/dt is associated with $i\omega$ instead of, as is the case for us and for physicists, with $-i\omega$. We confirm this with equation (6.27). Physicists and geophysicists deal with many more independent variables than time. Besides the obvious three space axes are their mutual combinations, such as midpoint and offset.

You might ask, why not make *all* the signs positive in equation (6.27)? The reason is that in that case waves would not move in a positive direction along the space axes. This would be especially unnatural when the space axis was a radius.

Atoms, like geophysical sources, always radiate from a point to infinity, not the other way around. Thus, in equation (6.27) the sign of the spatial frequencies must be opposite that of the temporal frequency.

The only good reason I know to choose the engineering convention is that we might compute with an array processor built and microcoded by engineers. Conflict of sign convention is not a problem for the programs that transform complex-valued time functions to complex-valued frequency functions, because there the sign convention is under the user's control. But sign conflict does make a difference when we use any program that converts real-time functions to complex frequency functions. The way to live in both worlds is to imagine that the frequencies produced by such a program do not range from 0 to $+\pi$ as the program description says, but from 0 to $-\pi$. Alternately, we could always take the complex conjugate of the transform, which would swap the sign of the ω-axis.

6.5.3. Simple examples of 2-D FT

An example of a **two-dimensional Fourier transform** of a pulse is shown in Figure 6.5.

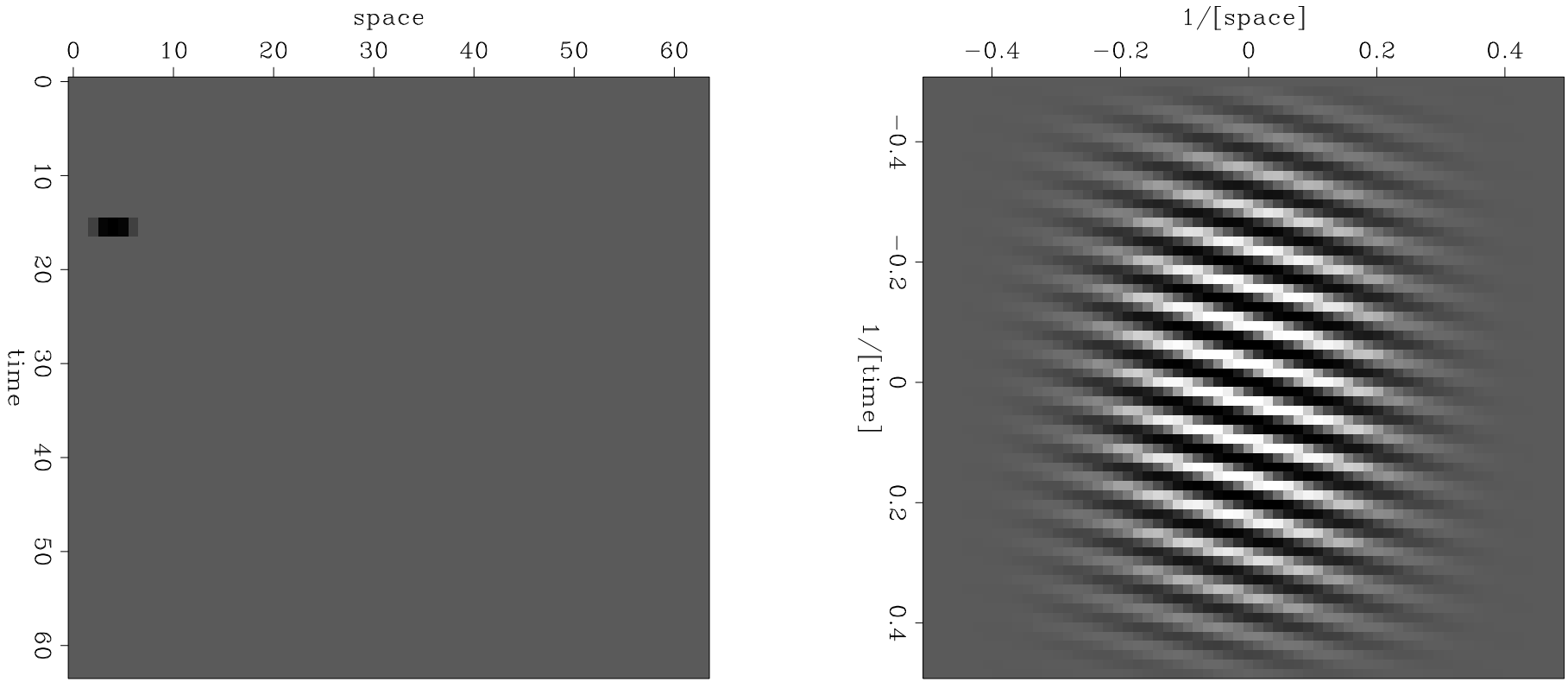


Figure 6.5: A broadened pulse (left) and the real part of its FT (right). ft1-ft2dofpulse [ER]

Notice the location of the pulse. It is closer to the time axis than the space axis. This will affect the real part of the FT in a certain way (see exercises). Notice the broadening of the pulse. It was an impulse smoothed over time (vertically) by convolution with (1,1) and over space (horizontally) with (1,4,6,4,1). This will affect the real part of the FT in another way.

Another example of a two-dimensional Fourier transform is given in Figure 6.6. This example simulates an impulsive air wave originating at a point on the x-axis. We see a wave propagating in each direction from the location of the source of the wave. In Fourier space there are also two lines, one for each wave. Notice that there are other lines which do not go through the origin; these lines are called "**spatial alias**es." Each actually goes through the origin of another square plane that is not shown, but which we can imagine alongside the one shown. These other planes are periodic replicas of the one shown.

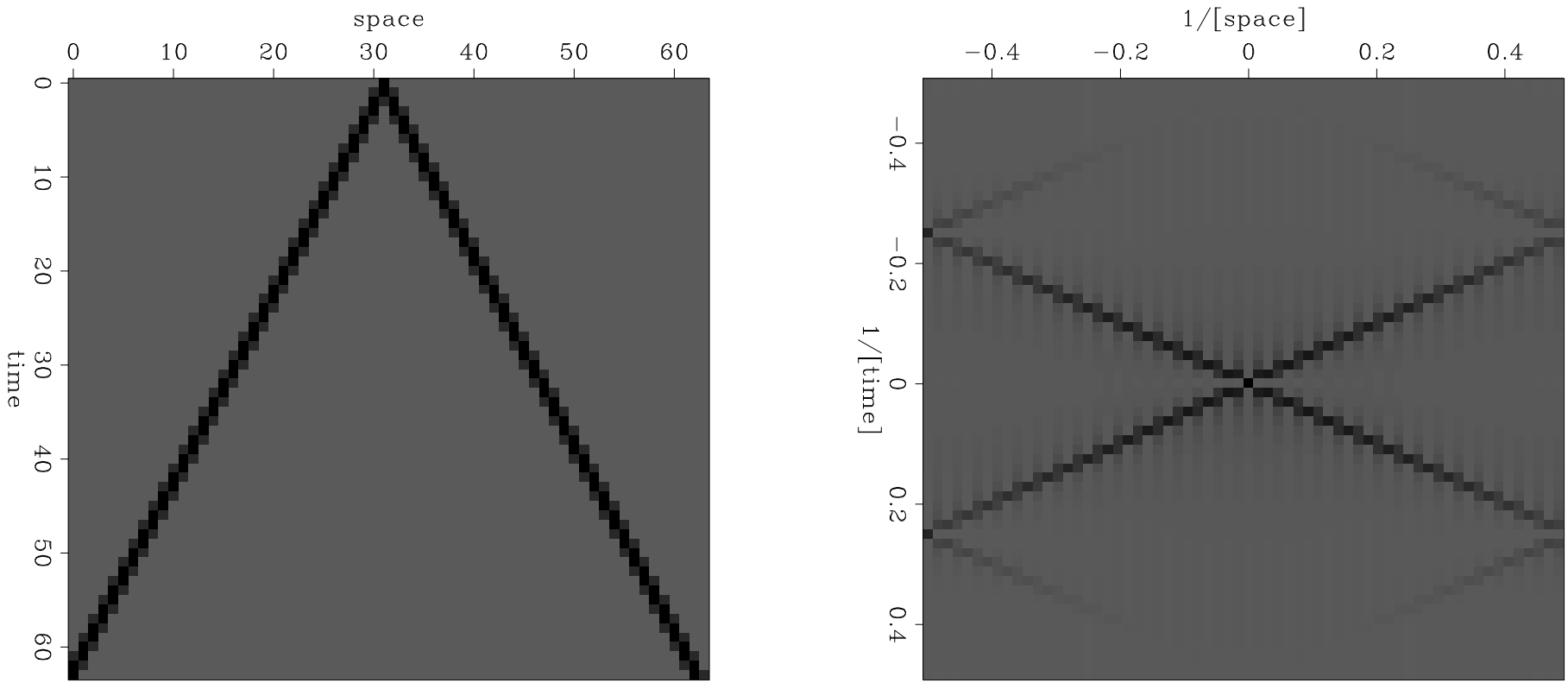


Figure 6.6: A simulated air wave (left) and the amplitude of its FT (right). ft1-airwave [ER]

EXERCISES:

1 Most time functions are real. Their imaginary part is zero. Show that this means that $F(\omega,k)$ can be determined from $F(-\omega,-k)$.

2 What would change in Figure 6.5 if the pulse were moved (a) earlier on the t-axis, and (b) further on the x-axis? What would change in Figure 6.5 if instead the time axis were smoothed with (1,4,6,4,1) and the space axis with (1,1)?

3 What would Figure 6.6 look like on an earth with half the earth velocity?

4 Numerically (or theoretically) compute the two-dimensional spectrum of a plane wave $[\delta(t-px)]$, where the plane wave has a randomly fluctuating amplitude: say, rand(x) is a random number between ± 1, and the randomly modulated plane wave is $[(1+.2\,\text{rand}(x))\,\delta(t-px)]$.

5 Explain the horizontal "layering" in Figure 6.4 in the plot of $P(\omega,x)$. What determines the "layer" separation? What determines the "layer" slope?

6.5.4. Magic with 2-D Fourier transforms

We have struggled through some technical details to learn how to perform a 2-D Fourier transformation. An immediate reward next is a few "magical" results on data.

In this book waves go down into the earth; they reflect; they come back up; and then they disappear. In reality after they come back up they reflect from the earth surface and go back down for another episode. Such waves, called multiple reflections, in real life are in some places negligible while in other places they overwhelm. Some places these multiply reflected waves can be suppressed because their RMS velocity tends to be slower because they spend more time in shallower regions. In other places this is not so. We can always think of making an earth model, using it to predict the multiply reflected waveforms, and subtracting the multiples from the data. But a serious pitfall is that we would need to have the earth model in order to find the earth model.

Fortunately, a little Fourier transform magic goes a long way towards solving the problem. Take a shot profile $d(t,x)$. Fourier transform it to $D(\omega,k_x)$. For every ω and k_x, square this value $D(\omega,k_x)^2$. Inverse Fourier transform. In Figure 6.7 we inspect the result. For the squared part the x-axis is reversed to facilitate comparison

at zero offset. A great many reflections on the raw data (right) carry over into the predicted multiples (left). If not, they are almost certainly primary reflections. This data shows more multiples than primaries.

Why does this work? Why does squaring the Fourier Transform of the raw data give us this good looking estimate of the multiple reflections? Recall Z-transforms $Z = e^{i\omega\Delta t}$. A Z-transform is really a Fourier transform. Take a signal that is an impulse of amplitude r at time $t = 100\Delta t$. Its Z-transform is $r\,Z^{100}$. The square of this Z-transform is $r^2 Z^{200}$, just what we expect of a multiple reflection — squared amplitude and twice the travel time. That explains vertically propagating waves. When a ray has a horizontal component, an additional copy of the ray doubles the horizontal distance traveled. Remember what squaring a Fourier transformation does – a convolution. Here the convolution is over both t and x. Every bit of the echo upon reaching the earth surface turns around and pretends it is a new little shot. Mathematically, every point in the upcoming wave $d(t,x)$ launches a replica of $d(t,x)$ shifted in both time and space – an autoconvolution.

In reality, multiple reflections offer a considerable number of challenges that I'm not mentioning. The point here is just that FT is a good tool to have.

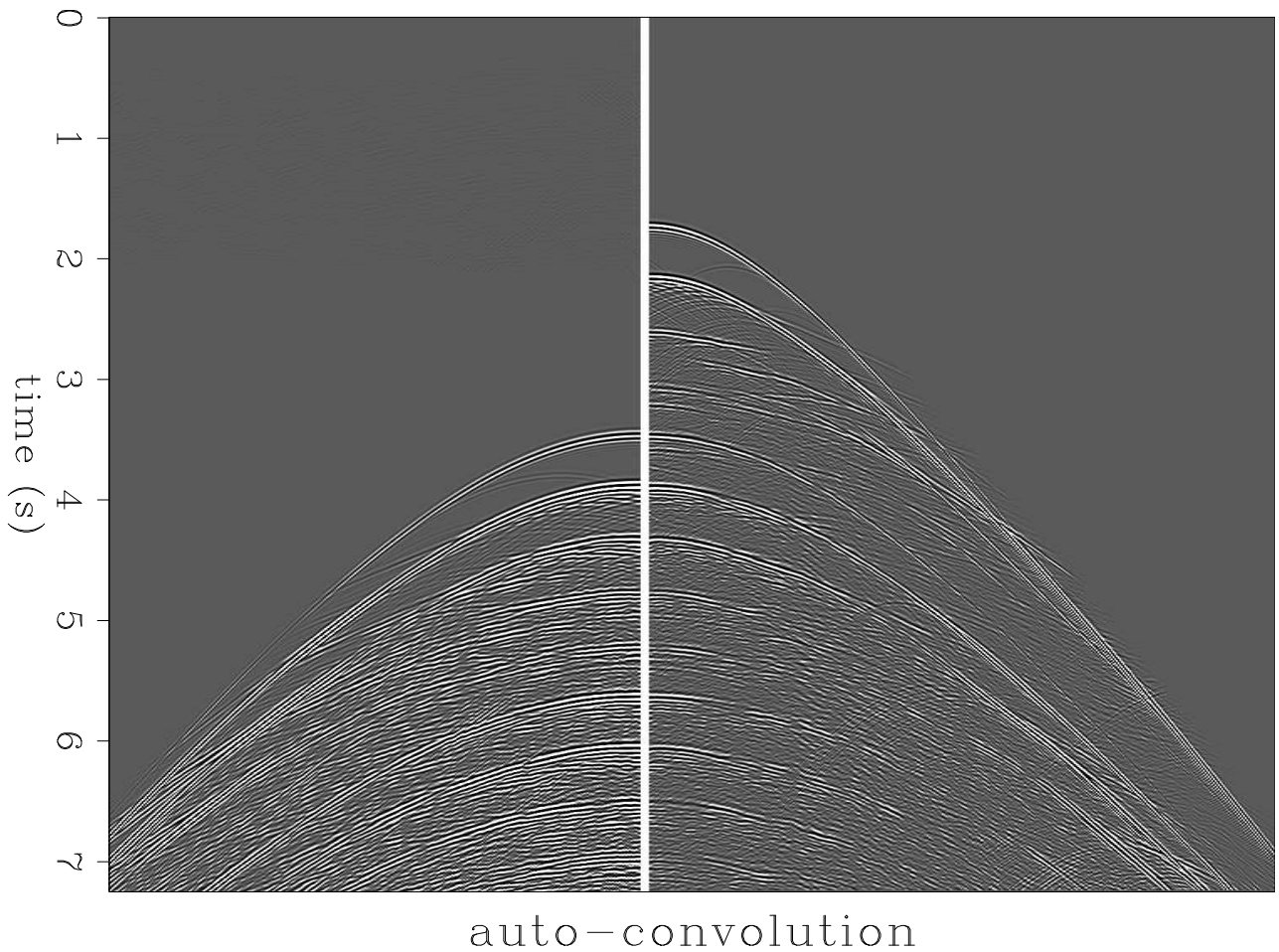


Figure 6.7: Data (right) with its FT squared (left). ft1-brad1 [ER]

6.5.5. Passive seismology

Signals go on and on, practically forever. Sometimes we like to limit our attention to something more limited such as their spectrum, or equivalently, their autocorrelation. We can compute the autocorrelation in the Fourier domain. We multiply the FT times its complex conjugate $D(\omega, k_x)\overline{D(\omega, k_x)}$. Transforming back to the physical domain we see Figure 6.8. We expect a giant burst at zero offset (upper right corner). We do not see it because it is "clipped", i.e. plot values above some threshhold are plotted at that threshhold. I could scale the plot to see the zero-offset burst, but then the interesting signals shown here would be too weak to be seen.

Figure 6.8 shows us that the 2-D autocorrelation of a shot profile shares a lot in common with the shot profile itself. This is interesting news. If we had a better understanding of this we might find some productive applications. We might find a situation where we do not have (or do not want) the data itself but we do wish to build an earth model. For example, suppose we have permanently emplaced geophones. The earth is constantly excited by seismic noise. Some of it is man made; some results from earthquakes elsewhere in the world; most probably results from natural sources such as ocean waves, wind in trees, etc. Recall every bit of acoustic energy that arrives at the surface from below becomes a little bit of a source for a

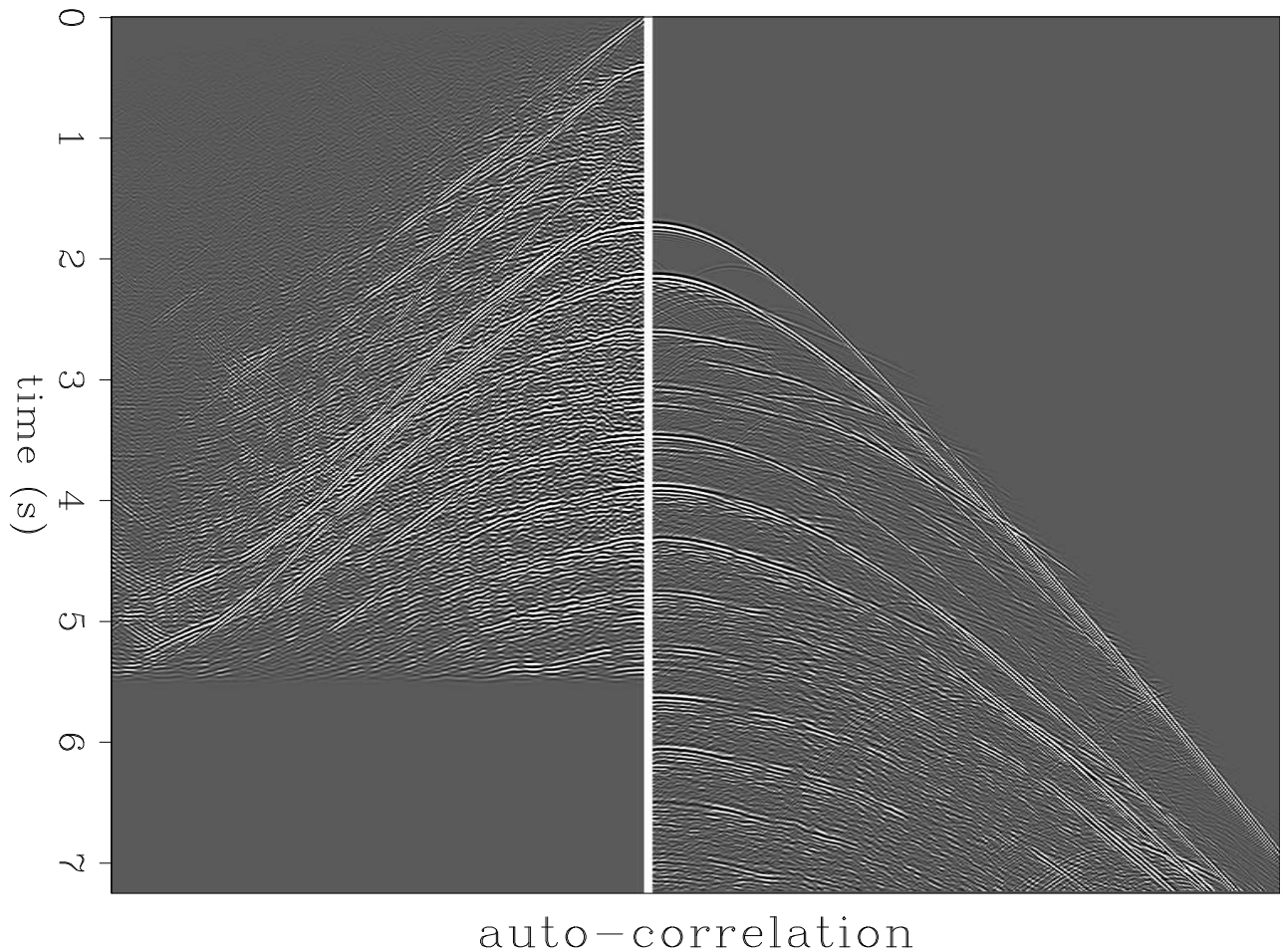


Figure 6.8: The 2-D autocorrelation of a shot profile resembles itself. ft1-brad2 [ER]

second reflection seismic experiment. So, by autocorrelating the data of hours and days duration we convert the chaos of continuing microseismic noise to something that might be the impulse response of the earth, or something like it. Autocorrelation converts a time axis of length of days to one of seconds. From the autocorrelation we might be able to draw conclusions in usual ways, alternately, we might learn how to make earth models from autocorrelations.

Notice from Figure 6.8 that since the first two seconds of the signal vanishes (travel time to ocean bottom), the last two seconds of the autocorrelation must vanish (longest nonzero lag on the data).

There are many issues on Figure 6.8 to intrigue an interpreter (starting with signal polarity). We also notice that the multiples on the autocorrelation die off rapidly with increasing offset and wonder why, and whether the same is true of primaries. But today is not the day to start down these paths.

In principal an autocorrelation is not comparable to the raw data or to the ideal shot profile because forming a spectrum squares amplitudes. We can overcome this difficulty by use of multidimensional spectral factorization — but that's an advanced mathematical concept not defined in this book. See my other book, Image Estimation.

6.6. THE HALF-ORDER DERIVATIVE WAVEFORM

Causal integration is represented in the time domain by convolution with a step function. In the frequency domain this amounts to multiplication by $1/(-i\omega)$. (There is also delta function behavior at $\omega = 0$ which may be ignored in practice and since at $\omega = 0$, wave theory reduces to potential theory). Integrating twice amounts to convolution by a ramp function, $t\,\mathrm{step}(t)$, which in the Fourier domain is multiplication by $1/(-i\omega)^2$. Integrating a third time is convolution with $t^2\,\mathrm{step}(t)$ which in the Fourier domain is multiplication by $1/(-i\omega)^3$. In general

$$t^{n-1}\,\mathrm{step}(t) \quad = \quad \mathrm{FT}\left(\frac{1}{(-i\omega)^n}\right) \tag{6.28}$$

Proof of the validity of equation (6.28) for integer values of n is by repeated indefinite integration which also indicates the need of an $n!$ scaling factor. Proof of the validity of equation (6.28) for fractional values of n would take us far afield mathematically. Fractional values of n, however, are exactly what we need to interpret Huygen's secondary wave sources in 2-D. The factorial function of n in the

scaling factor becomes a gamma function. The poles suggest that a more thorough mathematical study of convergence is warranted, but this is not the place for it.

The principal artifact of the hyperbola-sum method of 2-D migration is the waveform represented by equation (6.28) when $n = 1/2$. For $n = 1/2$, ignoring the scale factor, equation (6.28) becomes

$$\frac{1}{\sqrt{t}} \text{ step}(t) \quad = \quad \text{FT} \left(\frac{1}{\sqrt{-i\omega}} \right) \tag{6.29}$$

A waveform that should come out to be an impulse actually comes out to be equation (6.29) because Kirchhoff migration needs a little more than summing or spreading on a hyperbola. To compensate for the erroneous filter response of equation (6.29) we need its inverse filter. We need $\sqrt{-i\omega}$. To see what $\sqrt{-i\omega}$ is in the time domain, we first recall that

$$\frac{d}{dt} \quad = \quad \text{FT } (-i\omega) \tag{6.30}$$

A product in the frequency domain corresponds to a convolution in the time domain. A time derivative is like convolution with a doublet $(1, -1)/\Delta t$. Thus, from

equation (6.29) and equation (6.30) we obtain

$$\frac{d}{dt}\,\frac{1}{\sqrt{t}}\ \text{step}(t) \quad = \quad \text{FT}\left(\sqrt{-i\omega}\right) \tag{6.31}$$

Thus, we will see the way to overcome the principal artifact of hyperbola summation is to apply the filter of equation (6.31). In chapter 7 we will learn more exact methods of migration. There we will observe that an impulse in the earth creates not a hyperbola with an impulsive waveform but in two dimensions, a hyperbola with the waveform of equation (6.31), and in three dimensions, a hyperbola of revolution (umbrella?) carrying a time-derivative waveform.

6.6.1. Hankel tail

The waveform in equation (6.31) often arises in practice (as the 2-D Huygens wavelet). Because of the discontinuities on the left side of equation (6.31), it is not easy to visualize. Thinking again of the time derivative as a convolution with the doublet $(1, -1)/\Delta t$, we imagine the 2-D Huygen's wavelet as a positive impulse followed by negative signal decaying as $-t^{-3/2}$. This decaying signal is sometimes

called the "**Hankel tail**." In the frequency domain $-i\omega = |\omega| e^{-i90^\circ}$ has a 90 degree phase angle and $\sqrt{-i\omega} = |\omega|^{1/2} e^{-i45^\circ}$ has a **45 degree phase angle**. halfdifa

```
# Half order causal derivative.  OK to equiv(xx,yy)
#
subroutine halfdifa( adj, add, n, xx,    yy  )
integer n2, i,       adj, add, n
real omega,                       xx(n), yy(n)
complex cz, cv(4096)
n2=1; while(n2<n) n2=2*n2;          if( n2 > 4096) call erexit('halfdif memory')
do i= 1, n2 {             cv(i) = 0.}
do i= 1, n
        if( adj == 0)   { cv(i) = xx(i)}
        else            { cv(i) = yy(i)}
call adjnull(       adj, add,    xx,n,  yy,n)
call ftu( +1., n2, cv)
                        do i= 1, n2 {
                                omega = (i-1.) * 2.*3.14159265 / n2
                                cz = csqrt( 1. - cexp( cmplx( 0., omega)))
                                if( adj != 0)   cz = conjg( cz)
                                cv(i) = cv(i) * cz
                                }
call ftu( -1., n2, cv)
do i= 1, n
        if( adj == 0)   { yy(i) = yy(i) + cv(i)}
        else            { xx(i) = xx(i) + cv(i)}
return; end
```

Back

In practice, it is easiest to represent and to apply the 2-D Huygen's wavelet in the frequency domain. Subroutine `halfdifa()` /prog:halfdifa is provided for that purpose. Instead of using $\sqrt{-i\omega}$ which has a discontinuity at the Nyquist frequency and a noncausal time function, I use the square root of a causal representation of a finite difference, i.e. $\sqrt{1-Z}$, which is well behaved at the Nyquist frequency and has the advantage that the modeling operator is causal (vanishes when $t < t_0$). Fourier transform is done using subroutine `ftu()` /prog:ftu. Passing an impulse function into subroutine `halfdifa()` gives the response seen in Figure 6.9.

6.7. References

Special issue on fast Fourier transform, June 1969: IEEE Trans. on Audio and Electroacoustics (now known as IEEE Trans. on Acoustics, Speech, and Signal Processing), **AU-17**, entire issue (66-172).

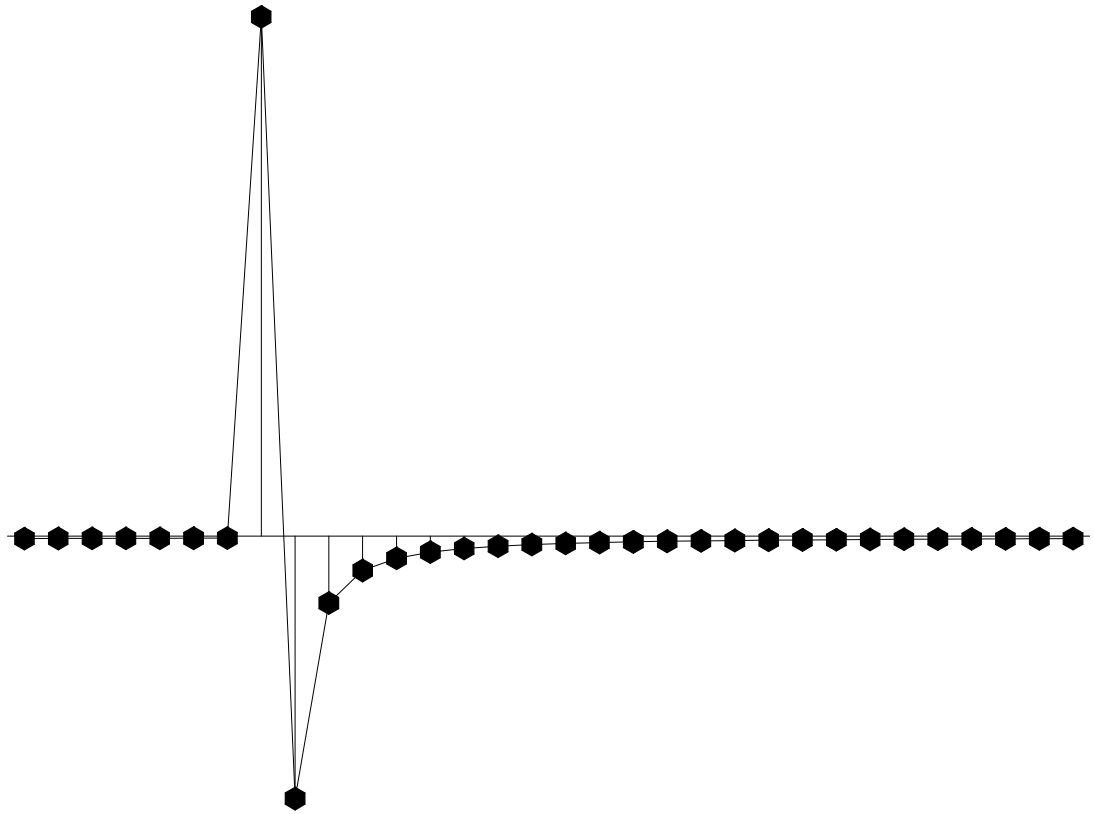


Figure 6.9: Impulse response (delayed) of finite difference operator of half order. Twice applying this filter is equivalent to once applying $(1, -1)$. ft1-hankel [ER]

Chapter 7

Downward continuation

7.1. MIGRATION BY DOWNWARD CONTINUATION

Given waves observed along the earth's surface, some well-known mathematical techniques that are introduced here enable us to extrapolate (**downward continue**)

these waves down into the earth. Migration is a simple consequence of this extrapolation.

7.1.1. Huygens secondary point source

Waves on the ocean have wavelengths comparable to those of waves in seismic prospecting (15-500 meters), but ocean waves move slowly enough to be seen. Imagine a long harbor barrier parallel to the beach with a small entrance in the barrier for the passage of ships. This is shown in Figure 7.1.

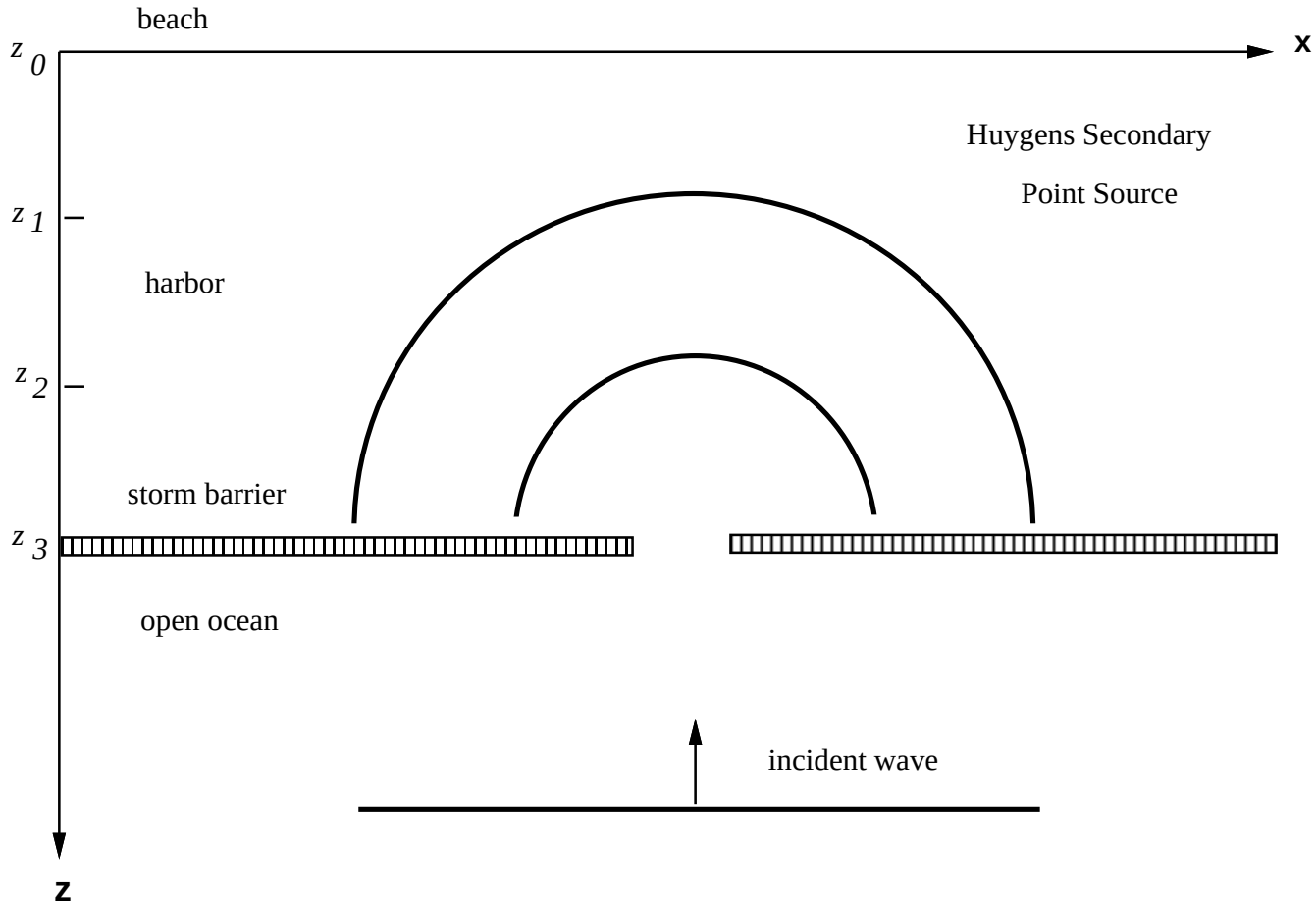


Figure 7.1: Waves going through a gap in a barrier have semicircular wavefronts (if the wavelength is long compared to the gap size). dwnc-storm [NR]

A plane wave incident on the barrier from the open ocean will send a wave through the gap in the barrier. It is an observed fact that the wavefront in the harbor becomes a circle with the gap as its center. The difference between this beam of water waves and a light beam through a window is in the ratio of wavelength to hole size.

Linearity is a property of all low-amplitude waves (not those foamy, breaking waves near the shore). This means that two gaps in the harbor barrier make two semicircular wavefronts. Where the circles cross, the wave heights combine by simple linear addition. It is interesting to think of a barrier with many holes. In the limiting case of very many holes, the barrier disappears, being nothing but one gap alongside another. Semicircular wavefronts combine to make only the incident plane wave. Hyperbolas do the same. Figure 7.2 shows hyperbolas increasing in density from left to right.

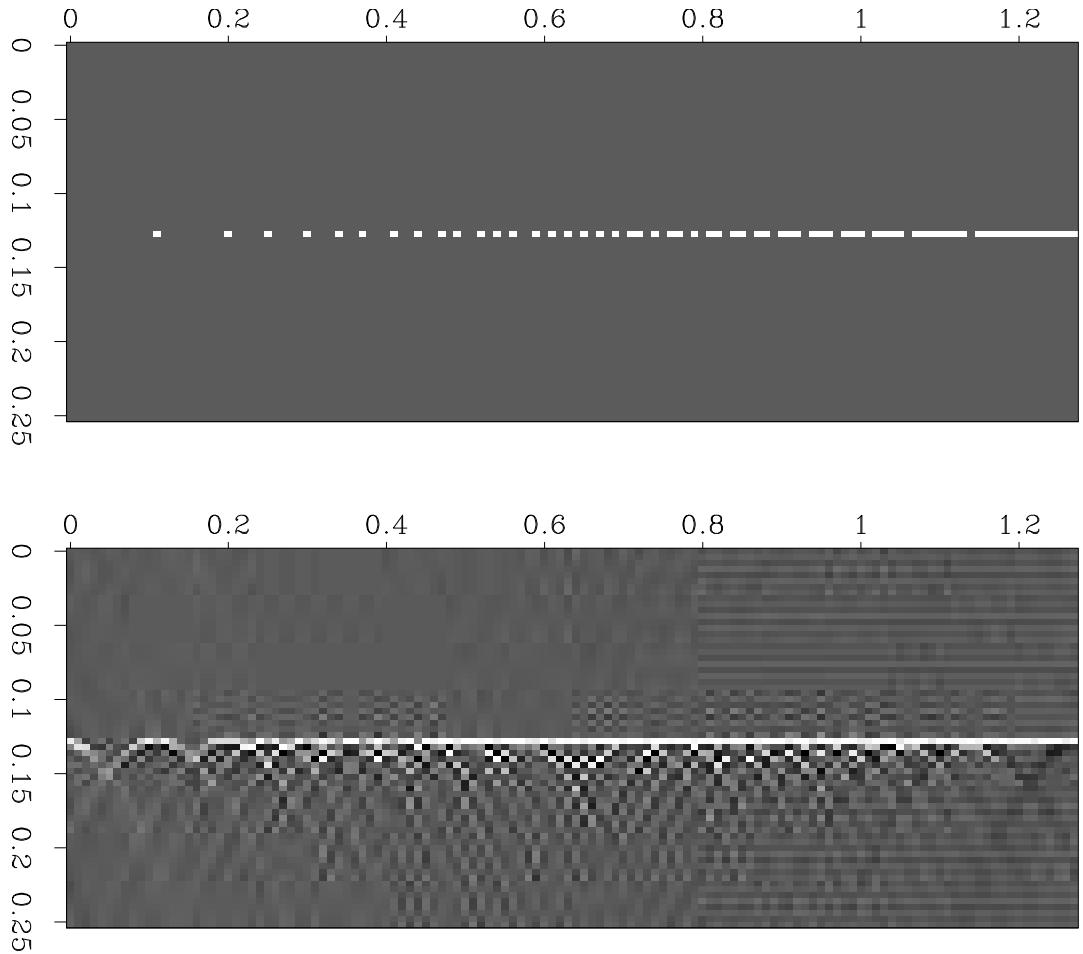


Figure 7.2: A barrier with many holes (top). Waves, (x,t)-space, seen beyond the barrier (bottom). dwnc-stormhole [ER]

All those waves at nonvertical angles must somehow combine with one another to extinguish all evidence of anything but the plane wave.

A Cartesian coordinate system has been superimposed on the ocean surface with x going along the beach and z measuring the distance from shore. For the analogy with reflection seismology, people are confined to the beach (the earth's surface) where they make measurements of wave height as a function of x and t. From this data they can make inferences about the existence of gaps in the barrier out in the (x,z)-plane. The first frame of Figure 7.3 shows the arrival time at the beach of a wave from the ocean through a gap.

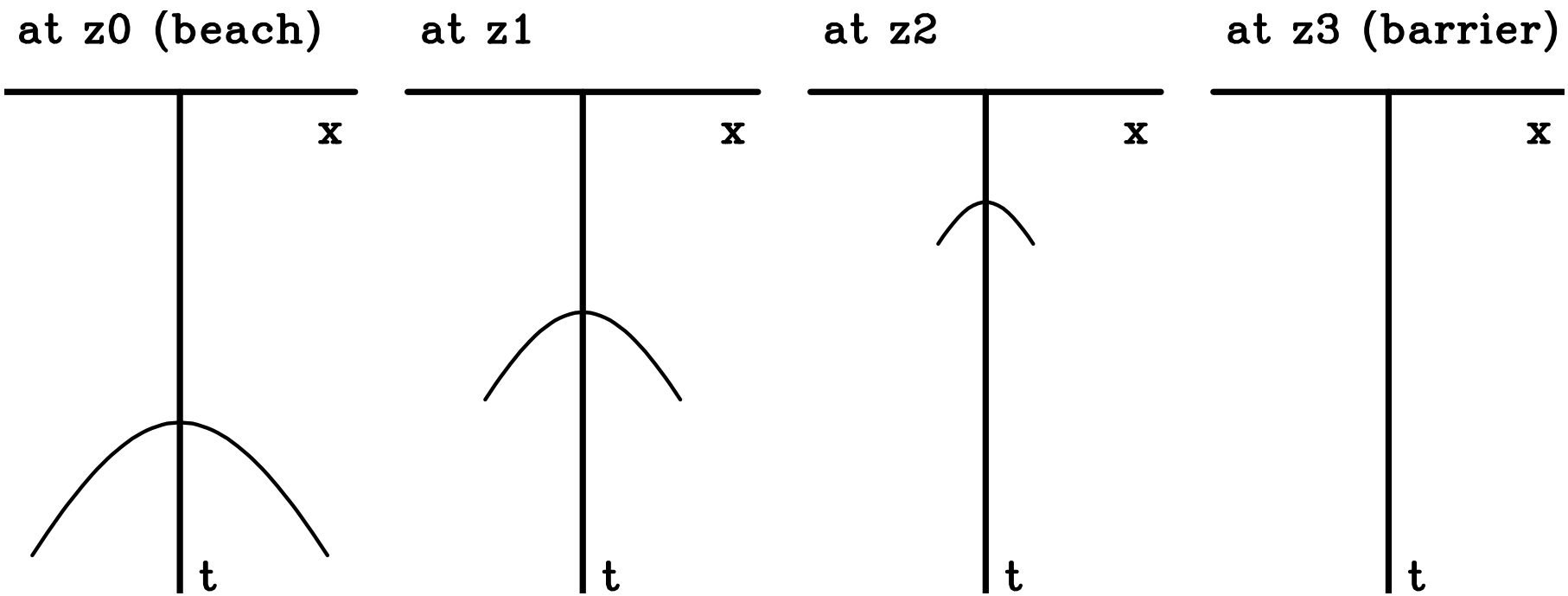


Figure 7.3: The left frame shows the hyperbolic wave arrival time seen at the beach. Frames to the right show arrivals at increasing distances out in the water. The x-axis is compressed from Figure 7.1. dwnc-dc [ER]

The earliest arrival occurs nearest the gap. What mathematical expression determines the shape of the arrival curve seen in the (x,t)-plane?

The waves are expanding circles. An equation for a circle expanding with velocity v about a point (x_3,z_3) is

$$(x-x_3)^2 \;+\; (z-z_3)^2 \;=\; v^2\,t^2 \tag{7.1}$$

Considering t to be a constant, i.e. taking a snapshot, equation (7.1) is that of a circle. Considering z to be a constant, it is an equation in the (x,t)-plane for a hyperbola. Considered in the (t,x,z)-volume, equation (7.1) is that of a cone. Slices at various values of t show circles of various sizes. Slices of various values of z show various hyperbolas. Figure 7.3 shows four hyperbolas. The first is the observation made at the beach $z_0 = 0$. The second is a hypothetical set of observations at some distance z_1 out in the water. The third set of observations is at z_2, an even greater distance from the beach. The fourth set of observations is at z_3, nearly all the way out to the barrier, where the hyperbola has degenerated to a point. All these hyperbolas are from a family of hyperbolas, each with the same asymptote. The asymptote refers to a wave that turns nearly 90° at the gap and is found moving nearly parallel to the shore at the speed dx/dt of a water wave. (For this water wave analogy it is

presumed—incorrectly—that the speed of water waves is a constant independent of water depth).

If the original incident wave was a positive pulse, the Huygens secondary source must consist of both positive and negative polarities to enable the destructive interference of all but the plane wave. So the Huygens waveform has a phase shift. In the next section, mathematical expressions will be found for the Huygens secondary source. Another phenomenon, well known to boaters, is that the largest amplitude of the Huygens semicircle is in the direction pointing straight toward shore. The amplitude drops to zero for waves moving parallel to the shore. In optics this amplitude drop-off with angle is called the *obliquity factor.*

7.1.2. Migration derived from downward continuation

A dictionary gives many definitions for the word *run.* They are related, but they are distinct. Similarly, the word *migration* in geophysical prospecting has about four related but distinct meanings. The simplest is like the meaning of the word *move.* When an object at some location in the (x,z)-plane is found at a different location at a later time t, then we say it *moves.* Analogously, when a wave arrival (often called

an *event*) at some location in the (x,t)-space of geophysical observations is found at a different position for a different survey line at a greater depth z, then we say it *migrates*.

To see this more clearly, imagine the four frames of Figure 7.3 being taken from a movie. During the movie, the depth z changes beginning at the beach (the earth's surface) and going out to the storm barrier. The frames are superimposed in Figure 7.4(left).

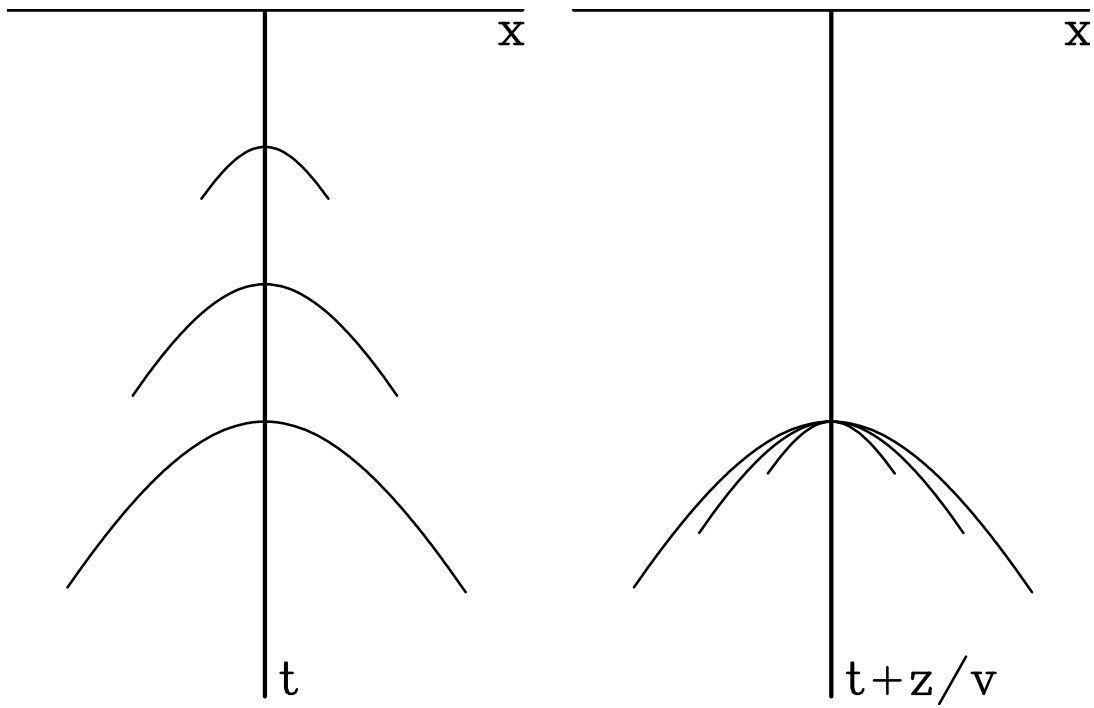


Figure 7.4: Left shows a superposition of the hyperbolas of Figure 7.3. At the right the superposition incorporates a shift, called retardation $t' = t + z/v$, to keep the hyperbola tops together. dwnc-dcretard [ER]

Mainly what happens in the movie is that the event migrates upward toward $t = 0$. To remove this dominating effect of vertical translation we make another superposition, keeping the hyperbola tops all in the same place. Mathematically, the time t axis is replaced by a so-called *retarded* time axis $t' = t + z/v$, shown in Figure 7.4(right). The second, more precise definition of *migration* is the motion of an event in (x, t')-space as z changes. After removing the vertical shift, the residual motion is mainly a shape change. By this definition, hyperbola tops, or horizontal layers, do not migrate.

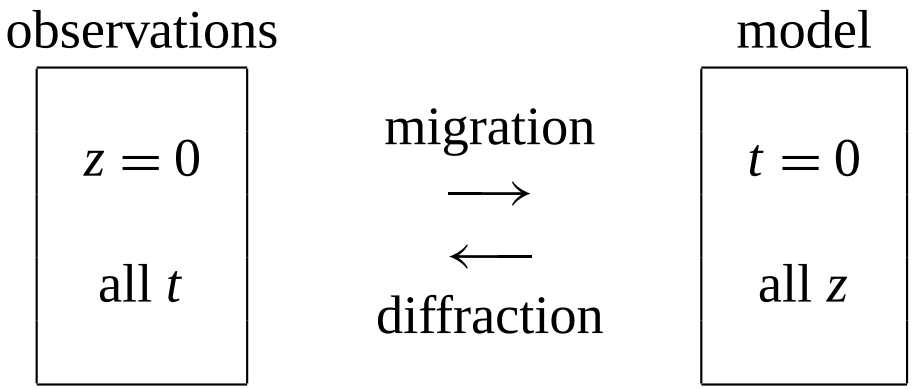
The hyperbolas in Figure 7.4 really extend to infinity, but the drawing cuts each one off at a time equal $\sqrt{2}$ times its earliest arrival. Thus the hyperbolas shown depict only rays moving within 45° of the vertical. It is good to remember this, that the ratio of first arrival time on a hyperbola to any other arrival time gives the cosine of the angle of propagation. The cutoff on each hyperbola is a ray at 45°. Notice that the end points of the hyperbolas on the drawing can be connected by a straight line. Also, the slope at the end of each hyperbola is the same. In physical space, the angle of any ray is $\tan\theta = dx/dz$. For any plane wave (or seismic event that is near a plane wave), the slope $v\,dt/dx$ is $\sin\theta$, as you can see by considering a wavefront intercepting the earth's surface at angle θ. So, energy

moving on a straight line in physical (x,z)-space migrates along a straight line in data (x,t)-space. As z increases, the energy of all angles comes together to a focus. The focus is the exploding reflector. It is the gap in the barrier. This third definition of migration is that it is the process that somehow pushes observational data—wave height as a function of x and t —from the beach to the barrier. The third definition stresses not so much the motion itself, but the transformation from the beginning point to the ending point.

To go further, a more general example is needed than the storm barrier example. The barrier example is confined to making Huygens sources only at some particular z. Sources are needed at other depths as well. Then, given a wave-extrapolation process to move data to increasing z values, exploding-reflector images are constructed with

$$\text{Image}\,(x,z) \quad = \quad \text{Wave}\,(t=0,x,z) \tag{7.2}$$

The fourth definition of migration also incorporates the definition of *diffraction* as the opposite of migration.



Diffraction is sometimes regarded as the natural process that creates and enlarges hyperboloids. *Migration* is the computer process that does the reverse.

Another aspect of the use of the word *migration* arises where the horizontal coordinate can be either shot-to-geophone midpoint y, or offset h. Hyperboloids can be downward continued in both the (y,t)- and the (h,t)-plane. In the (y,t)-plane this is called *migration* or *imaging*, and in the (h,t)-plane it is called ***focus**ing* or *velocity analysis*.

7.2. DOWNWARD CONTINUATION

Given a vertically upcoming plane wave at the earth's surface, say $u(t,x,z=0) = u(t)\text{const}(x)$, and an assumption that the earth's velocity is vertically stratified, i.e. $v = v(z)$, we can presume that the upcoming wave down in the earth is simply time-shifted from what we see on the surface. (This assumes no multiple reflections.) Time shifting can be represented as a linear operator in the time domain by representing it as convolution with an impulse function. In the frequency domain, time shifting is simply multiplying by a complex exponential. This is expressed as

$$u(t,z) = u(t,z=0) * \delta(t+z/v) \tag{7.3}$$

$$U(\omega,z) = U(\omega,z=0)\, e^{-i\omega z/v} \tag{7.4}$$

Sign conventions must be attended to, and that is explained more fully in chapter 6.

7.2.1. Continuation of a dipping plane wave.

Next consider a plane wave **dip**ping at some angle θ. It is natural to imagine continuing such a wave back along a ray. Instead, we will continue the wave straight down. This requires the assumption that the plane wave is a perfect one, namely that

the same waveform is observed at all x. Imagine two sensors in a vertical well bore. They should record the same signal except for a time shift that depends on the angle of the wave. Notice that the arrival time difference between sensors at two different depths is greatest for vertically propagating waves, and the time difference drops to zero for horizontally propagating waves. So the time shift Δt is $v^{-1}\cos\theta\,\Delta z$ where θ is the angle between the wavefront and the earth's surface (or the angle between the well bore and the ray). Thus an equation to downward continue the wave is

$$U(\omega,\theta,z+\Delta z) \quad = \quad U(\omega,\theta,z)\ \exp(-i\omega\,\Delta t) \tag{7.5}$$

$$U(\omega,\theta,z+\Delta z) \quad = \quad U(\omega,\theta,z)\ \exp\left(-i\omega\,\frac{\Delta z\cos\theta}{v}\right) \tag{7.6}$$

Equation (7.6) is a downward continuation formula for any angle θ. Following methods of chapter 3 we can generalize the method to media where the velocity is a function of depth. Evidently we can apply equation (7.6) for each layer of thickness Δz, and allow the velocity vary with z. This is a well known approximation that handles the timing correctly but keeps the amplitude constant (since $|e^{i\phi}|=1$) when in real life, the amplitude should vary because of reflection and transmission coefficients. Suffice it to say that in practical earth imaging, this approximation is

almost universally satisfactory.

In a stratified earth, it is customary to eliminate the angle θ which is depth variable, and change it to the Snell's parameter p which is constant for all depths. Thus the downward continuation equation for any Snell's parameter is

$$U(\omega, p, z+\Delta z) \quad = \quad U(\omega, p, z)\, \exp\left(-\frac{i\omega\Delta z}{v(z)} \sqrt{1-p^2 v(z)^2} \right) \tag{7.7}$$

It is natural to wonder where in real life we would encounter a **Snell wave** that we could downward continue with equation (7.7). The answer is that any wave from real life can be regarded as a sum of waves propagating in all angles. Thus a field data set should first be decomposed into Snell waves of all values of p, and then equation (7.7) can be used to downward continue each p, and finally the components for each p could be added. This process akin to Fourier analysis. We now turn to Fourier analysis as a method of downward continuation which is the same idea but the task of decomposing data into Snell waves becomes the task of decomposing data into sinusoids along the x-axis.

7.2.2. Downward continuation with Fourier transform

One of the main ideas in Fourier analysis is that an impulse function (a delta function) can be constructed by the superposition of sinusoids (or complex exponentials). In the study of time series this construction is used for the *impulse response* of a filter. In the study of functions of space, it is used to make a physical point source that can manufacture the downgoing waves that initialize the reflection seismic experiment. Likewise observed upcoming waves can be Fourier transformed over t and x.

Recall in chapter 3, a plane wave carrying an arbitrary waveform, specified by equation (3.7). Specializing the arbitrary function to be the real part of the function $\exp[-i\omega(t-t_0)]$ gives

$$\text{moving cosine wave} \;=\; \cos\left[\; \omega\left(\frac{x}{v}\,\sin\theta \;+\; \frac{z}{v}\,\cos\theta \;-\; t\;\right)\;\right] \tag{7.8}$$

Using Fourier integrals on time functions we encounter the *Fourier kernel* $\exp(-i\omega t)$. To use Fourier integrals on the space-axis x the spatial angular frequency must be defined. Since we will ultimately encounter many space axes (three for shot, three for geophone, also the midpoint and offset), the convention will be to use a subscript on the letter k to denote the axis being Fourier transformed. So k_x is the angular

spatial frequency on the x-axis and $\exp(ik_x x)$ is its Fourier kernel. For each axis and Fourier kernel there is the question of the sign before the i. The sign convention used here is the one used in most physics books, namely, the one that agrees with equation (7.8). Reasons for the choice are given in chapter 6. With this convention, a wave moves in the *positive* direction along the space axes. Thus the Fourier kernel for (x, z, t)-space will be taken to be

$$\text{Fourier kernel} = e^{i k_x x}\, e^{i k_z z}\, e^{-i\omega t} = \exp[i(k_x x + k_z z - \omega t)] \tag{7.9}$$

Now for the whistles, bells, and trumpets. Equating (7.8) to the real part of (7.9), physical angles and velocity are related to Fourier components. The Fourier kernel has the form of a plane wave. These relations should be memorized!

Angles and Fourier Components	
$\sin\theta = \dfrac{v\, k_x}{\omega}$	$\cos\theta = \dfrac{v\, k_z}{\omega}$

(7.10)

A point in (ω, k_x, k_z)-space is a plane wave. The one-dimensional Fourier kernel extracts frequencies. The multi-dimensional Fourier kernel extracts (monochromatic) plane waves.

Equally important is what comes next. Insert the angle definitions into the familiar relation $\sin^2\theta + \cos^2\theta = 1$. This gives a most important relationship:

$$k_x^2 + k_z^2 = \frac{\omega^2}{v^2} \tag{7.11}$$

The importance of (7.11) is that it enables us to make the distinction between an arbitrary function and a chaotic function that actually is a wavefield. Imagine any function $u(t,x,z)$. Fourier transform it to $U(\omega, k_x, k_z)$. Look in the (ω, k_x, k_z)-volume for any nonvanishing values of U. You will have a wavefield if and only if all nonvanishing U have coordinates that satisfy (7.11). Even better, in practice the (t,x)-dependence at $z = 0$ is usually known, but the z-dependence is not. Then the z-dependence is found by assuming U is a wavefield, so the z-dependence is inferred from (7.11).

Equation (7.11) also achieves fame as the "dispersion relation of the scalar **wave equation**," a topic developed more fully in IEI.

Given any $f(t)$ and its Fourier transform $F(\omega)$ we can shift $f(t)$ by t_0 if we

multiply $F(\omega)$ by $e^{i\omega t_0}$. This also works on the z-axis. If we were given $F(k_z)$ we could shift it from the earth surface $z = 0$ down to any z_0 by multiplying by $e^{ik_z z_0}$. Nobody ever gives us $F(k_z)$, but from measurements on the earth surface $z = 0$ and double Fourier transform, we can compute $F(\omega, k_x)$. If we assert/assume that we have measured a wavefield, then we have $k_z^2 = \omega^2/v^2 - k_x^2$, so knowing $F(\omega, k_x)$ means we know $F(k_z)$. Actually, we know $F(k_z, k_x)$. Technically, we also know $F(k_z, \omega)$, but we are not going to use it in this book.

We are almost ready to extrapolate waves from the surface into the earth but we need to know one more thing — which square root do we take for k_z? That choice amounts to the assumption/assertion of upcoming or downgoing waves. With the exploding reflector model we have no downgoing waves. A more correct analysis has two downgoing waves to think about: First is the spherical wave expanding about the shot. Second arises when upcoming waves hit the surface and reflect back down. The study of multiple reflections requires these waves.

7.2.3. Linking Snell waves to Fourier transforms

To link **Snell wave**s to Fourier transforms we merge equations (3.8) and (3.9) with equations (7.10)

$$\frac{k_x}{\omega} \quad = \quad \frac{\partial t_0}{\partial x} \quad = \quad \frac{\sin\theta}{v} \quad = \quad p \tag{7.12}$$

$$\frac{k_z}{\omega} \quad = \quad \frac{\partial t_0}{\partial z} \quad = \quad \frac{\cos\theta}{v} \quad = \quad \frac{\sqrt{1-p^2v^2}}{v} \tag{7.13}$$

The basic downward continuation equation for upcoming waves in Fourier space follows from equation (7.7) by eliminating p by using equation (7.12). For analysis of real seismic data we introduce a minus sign because equation (7.13) refers to downgoing waves and observed data is made from up-coming waves.

$$U(\omega,k_x,z+\Delta z) \quad = \quad U(\omega,k_x,z)\ \exp\left(-\frac{i\omega\Delta z}{v}\sqrt{1-\frac{v^2k_x^2}{\omega^2}}\right) \tag{7.14}$$

In Fourier space we delay signals by multiplying by $e^{i\omega\Delta t}$, analogously, equation (7.14) says we downward continue signals into the earth by multiplying by

$e^{ik_z \Delta z}$. Multiplication in the Fourier domain means convolution in time which can be depicted by the engineering diagram in Figure 7.5.

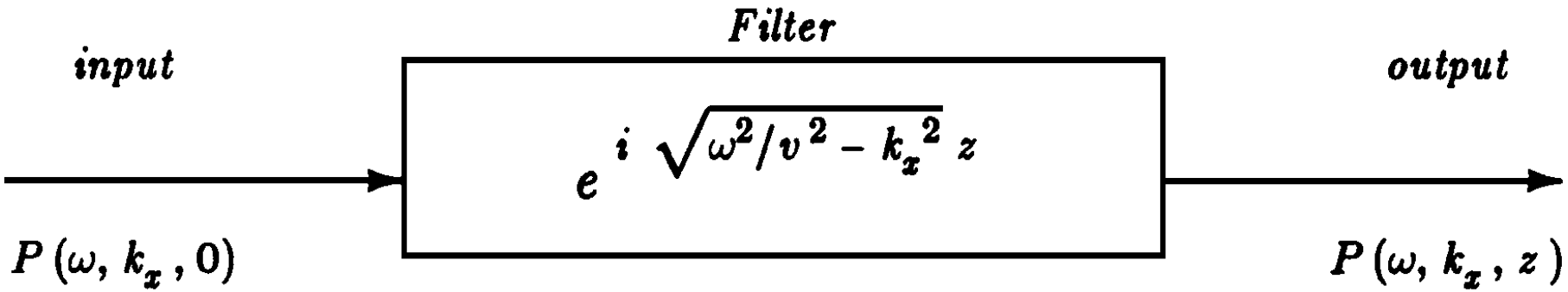


Figure 7.5: Downward continuation of a downgoing wavefield. dwnc-inout [NR]

Downward continuation is a product relationship in both the ω-domain and the k_x-domain. Thus it is a convolution in both time and x. What does the filter look like in the time and space domain? It turns out like a cone, that is, it is roughly an impulse function of $x^2 + z^2 - v^2t^2$. More precisely, it is the Huygens secondary wave source that was exemplified by ocean waves entering a gap through a storm barrier. Adding up the response of multiple gaps in the barrier would be convolution over x.

A nuisance of using Fourier transforms in migration and modeling is that spaces become periodic. This is demonstrated in Figure 7.6. Anywhere an event exits the frame at a side, top, or bottom boundary, the event immediately emerges on the opposite side. In practice, the unwelcome effect of periodicity is generally ameliorated by padding zeros around the data and the model.

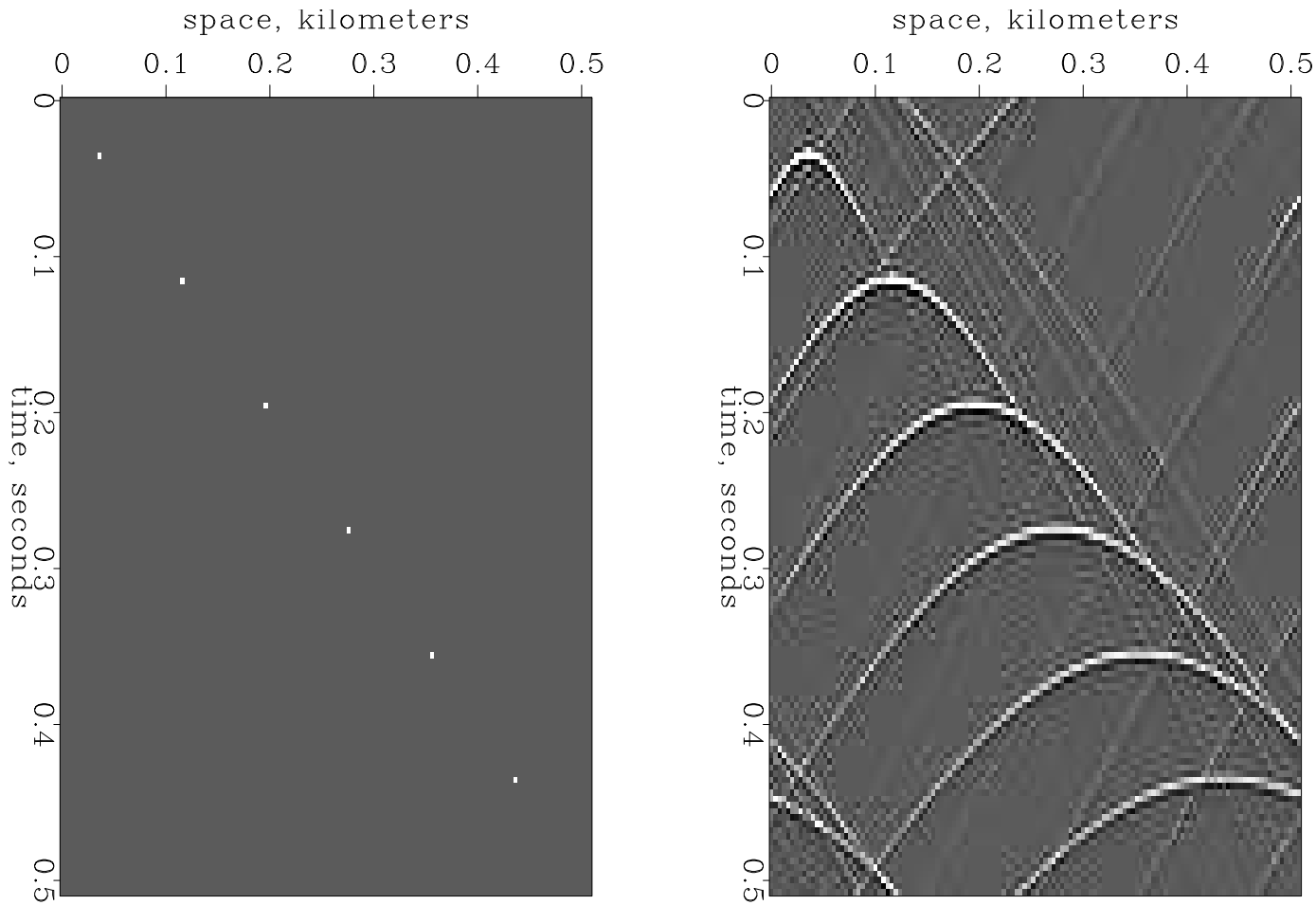


Figure 7.6: A reflectivity model on the left and synthetic data using a Fourier method on the right. dwnc-diag [ER]

7.3. PHASE-SHIFT MIGRATION

The phase-shift method of migration begins with a two-dimensional Fourier transform (2D-FT) of the dataset. (See chapter 6.) This transformed data is downward continued with $\exp(ik_z z)$ and subsequently evaluated at $t = 0$ (where the reflectors explode). Of all migration methods, the phase-shift method most easily incorporates depth variation in velocity. The phase angle and obliquity function are correctly included, automatically. Unlike Kirchhoff methods, with the phase-shift method there is no danger of aliasing the operator. (Aliasing the data, however, remains a danger.)

Equation (7.14) referred to upcoming waves. However in the reflection experiment, we also need to think about downgoing waves. With the exploding-reflector concept of a zero-offset section, the downgoing ray goes along the same path as the upgoing ray, so both suffer the same delay. The most straightforward way of converting one-way propagation to two-way propagation is to multiply time everywhere by two. Instead, it is customary to divide velocity everywhere by two. Thus the Fourier transformed data values, are downward continued to a depth Δz by mul-

tiplying by

$$e^{i\,k_z\,\Delta z} \quad = \quad \exp\left(-\,i\,\frac{2\,\omega}{v}\,\sqrt{1\,-\,\frac{v^2\,k_x^2}{4\,\omega^2}}\;\Delta z \right) \tag{7.15}$$

Ordinarily the time-sample interval $\Delta\tau$ for the output-migrated section is chosen equal to the time-sample rate of the input data (often 4 milliseconds). Thus, choosing the depth $\Delta z = (v/2)\Delta\tau$, the downward-extrapolation operator for a single time step $\Delta\tau$ is

$$C \quad = \quad \exp\left(-i\,\omega\;\Delta\tau\,\sqrt{1\,-\,\frac{v^2\,k_x^2}{4\,\omega^2}}\;\right) \tag{7.16}$$

Data will be multiplied many times by C, thereby downward continuing it by many steps of $\Delta\tau$.

7.3.1. Pseudocode to working code

Next is the task of imaging. Conceptually, at each depth an inverse Fourier transform is followed by selection of its value at $t = 0$. (Reflectors explode at $t = 0$).

Since only the Fourier transform at one point, $t = 0$, is needed, other times need not be be computed. We know the $\omega = 0$ Fourier component is found by the sum over all time, analogously, the $t = 0$ component is found as the sum over all ω. (This is easily shown by substituting $t = 0$ into the inverse Fourier integral.) Finally, inverse Fourier transform k_x to x. The migration process, computing the image from the upcoming wave u, may be summarized in the following pseudo code:

$U(\omega, k_x, \tau = 0) = FT[u(t,x)]$
For $\tau = \Delta\tau, 2\Delta\tau, \ldots,$ end of time axis on seismogram
    For all k_x
        For all ω
            $C = \exp(-i\omega\Delta\tau\sqrt{1 - v^2k_x^2/4\omega^2})$
            $U(\omega, k_x, \tau) = U(\omega, k_x, \tau - \Delta\tau) * C$
    For all k_x
        $\text{Image}(k_x, \tau) = 0.$
        For all ω
            $\text{Image}(k_x, \tau) = \text{Image}(k_x, \tau) + U(\omega, k_x, \tau)$
    $\text{image}(x, \tau) = FT[\text{Image}(k_x, \tau)]$

This pseudo code Fourier transforms a wavefield observed at the earth's surface $\tau = 0$, and then it marches that wavefield down into the earth ($\tau > 0$) filling up a three-dimensional function, $U(\omega, k_x, \tau)$. Then it selects $t = 0$, the time of the exploding reflectors by summing over all frequencies ω. (Mathematically, this is

like finding the signal at $\omega = 0$ by summing over all t).

Turning from pseudocode to real code, an important practical reality is that computer memories are not big enough for the three-dimensional function $U(\omega, k_x, \tau)$. But it is easy to intertwine the downward continuation with the summation over ω so a three-dimensional function need not be kept in memory. This is done in the real code in subroutine `phasemig()`. phasemig

Conjugate migration (modeling) proceeds in much the same way. Beginning from an upcoming wave that is zero at great depth, the wave is marched upward in steps by multiplication with $\exp(ik_z \Delta z)$. As each level in the earth is passed, exploding reflectors from that level are added into the upcoming wave. Pseudo code for modeling the upcoming wave u is

```
subroutine phasemig( up, nt, nx, dt, dx, image, ntau, dtau, vel)
integer                  nt, nx,                 ntau,          iw,nw,ikx,itau
real    dt,dx, w,w0,dw, kx,kx0,dkx,dtau, vel, sig1,sig2,pi, w2, vkx2
complex up(nt,nx), image(ntau,nx), cc
                        pi = 3.14159265;        sig1 = +1.;    sig2 = -1.
call ft1axis( 0, sig1, nt, nx, up)
call ft2axis( 0, sig2, nt, nx, up)
        nw = nt;              w0  = -pi/dt;      dw = 2.*pi/(nt*dt)
                              kx0 = -pi/dx;      dkx= 2.*pi/(nx*dx)
call null( image, ntau*nx*2)
do iw   = 2, nw  {      w  = w0  + (iw -1) * dw
do ikx  = 2, nx  {      kx = kx0 + (ikx-1) * dkx
        w2   = w * w
        vkx2 = vel*vel * kx*kx / 4.
        if( w2 > vkx2 ) {
                cc = cexp( cmplx( 0., - w * dtau * sqrt( 1. - vkx2/w2 ) ) )
                do itau = 1, ntau {
                        up(iw,ikx) = up(iw,ikx) * cc
                        image(itau,ikx) = image(itau,ikx) + up(iw,ikx)
                        }
                }
        }}
call ft2axis( 1, sig2, ntau, nx, image)
return; end
```

Back

$$
\begin{aligned}
&\text{Image}(k_x, z) = FT[\text{image}(x, z)] \\
&\text{For all } \omega \text{ and all } k_x \\
&\qquad U(\omega, k_x) = 0. \\
&\text{For all } \omega \ \{ \\
&\text{For all } k_x \ \{ \\
&\text{For } z = z_{\max}, z_{\max} - \Delta z, z_{\max} - 2\Delta z, \ldots, 0 \ \{ \\
&\qquad C = \exp(+i\,\Delta z \omega \sqrt{v^{-2} - {k_x}^2/\omega^2}) \\
&\qquad U(\omega, k_x) = U(\omega, k_x) * C \\
&\qquad U(\omega, k_x) = U(\omega, k_x) + \text{Image}(k_x, z) \\
&\qquad \} \ \} \ \} \\
&u(t, x) = FT[U(\omega, k_x)]
\end{aligned}
$$

Some real code for this job is in subroutine `phasemod()`. phasemod

The positive sign in the complex exponential is a combination of two negatives, the *up* coming wave and the *up*ward extrapolation. In principle, the three loops on ω, k_x, and z are interchangeable, however, since this tutorial program uses a velocity v that is a constant function of depth, I speeded it by a large factor by putting the

```
subroutine phasemod( image, nz, nx, dz, dx, up, nt, dt, vel)
integer                     nz, nx,           nt,       iw,nw,ikx,iz
real    dt,dx,dz, w,w0,dw, kx,kx0,dkx, vel, sig1,sig2,pi, w2, vkx2
complex up(nt,nx), image(nz,nx), cc
                pi = 3.14159265;        sig1 = +1.;     sig2 = -1.
call ft2axis( 0, sig2, nz, nx, image)
        nw = nt;         w0  = -pi/dt;   dw = 2.*pi/(nt*dt)
                         kx0 = -pi/dx;   dkx= 2.*pi/(nx*dx)
call null( up, nw*nx*2)
do iw  = 2, nw  {  w  = w0  + (iw-1)  * dw
do ikx = 2, nx  {  kx = kx0 + (ikx-1) * dkx
        w2   = w * w
        vkx2 = vel*vel * kx*kx / 4.
        if( w2 > vkx2 ) {
                cc = cexp( cmplx( 0., w * dz * sqrt(1. - vkx2/w2) ))
                do iz = nz, 1, -1
                        up(iw,ikx) = up(iw,ikx) * cc  + image(iz,ikx)
                }
        }}
call ft1axis( 1, sig1, nt, nx, up)
call ft2axis( 1, sig2, nt, nx, up)
return; end
```

Back

z-loop on the inside and pulling the complex exponential out of the inner loop. Figure 7.2 was made with subroutine `phasemod()` /prog:phasemod.

7.3.2. Kirchhoff versus phase-shift migration

In chapter 5, we were introduced to the Kirchhoff migration and modeling method by means of subroutines `kirchslow()` /prog:kirchslow and `kirchfast()` /prog:kirchfas From chapter 6 we know that these routines should be supplemented by a $\sqrt{-i\omega}$ filter such as subroutine `halfdifa()` /prog:halfdifa. Here, however, we will compare results of the unadorned subroutine `kirchfast()` /prog:kirchfast with our new programs, `phasemig()` /prog:phasemig and `phasemod()` /prog:phasemod. Figure 7.7 shows the result of modeling data and then migrating it. Kirchhoff and phase-shift migration methods both work well. As expected, the Kirchhoff method lacks some of the higher frequencies that could be restored by $\sqrt{-i\omega}$. Another problem is the irregularity of the shallow bedding. This is an operator aliasing problem addressed in chapter 10.

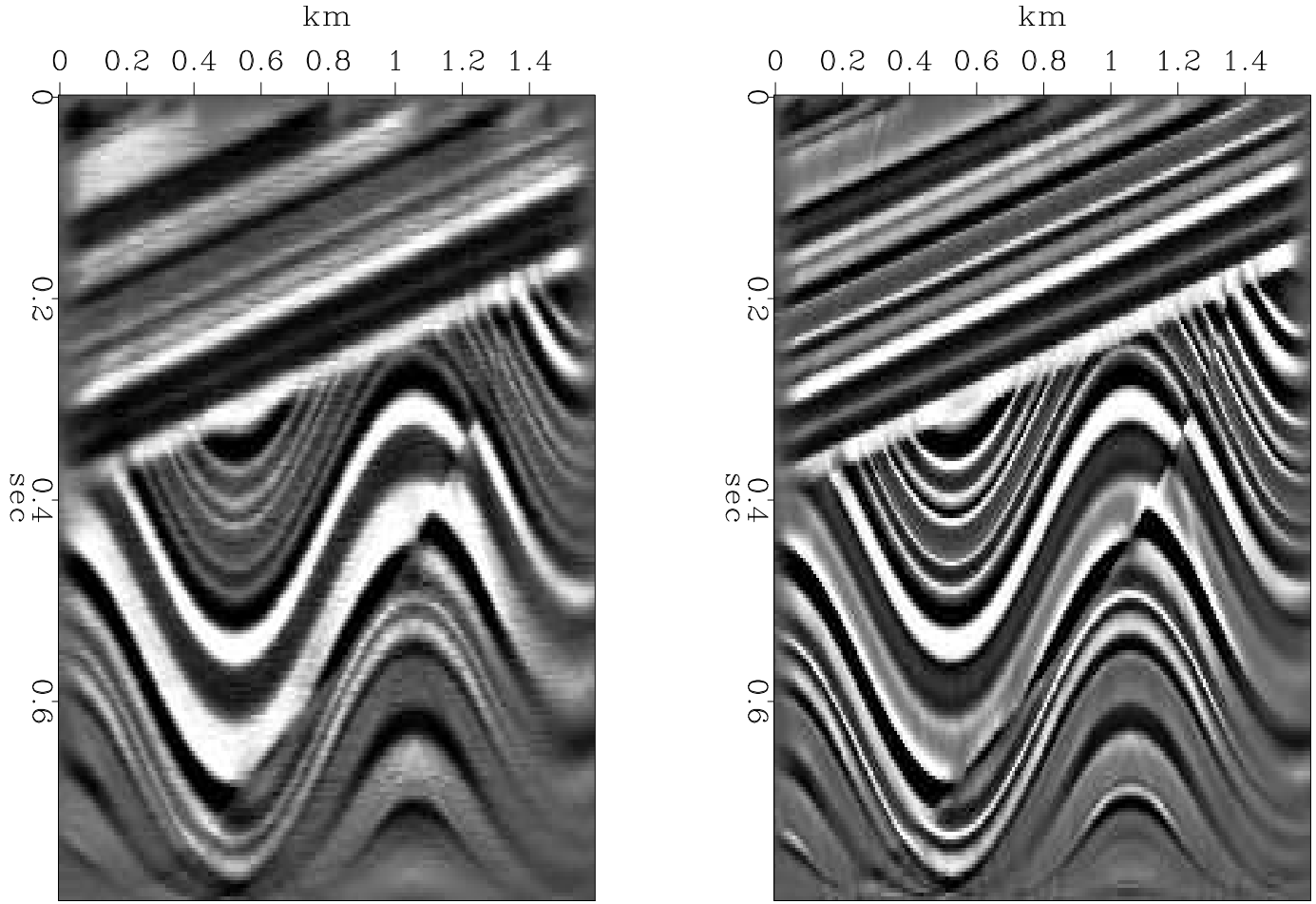


Figure 7.7: Reconstruction after modeling. Left is by the nearest-neighbor Kirchhoff method. Right is by the phase shift method. dwnc-comrecon [ER,M]

Figure 7.8 shows the temporal spectrum of the original sigmoid model, along with the spectrum of the reconstruction via phase-shift methods. We see the spectra are essentially identical with little growth of high frequencies as we noticed with the Kirchhoff method in Figure 5.9.

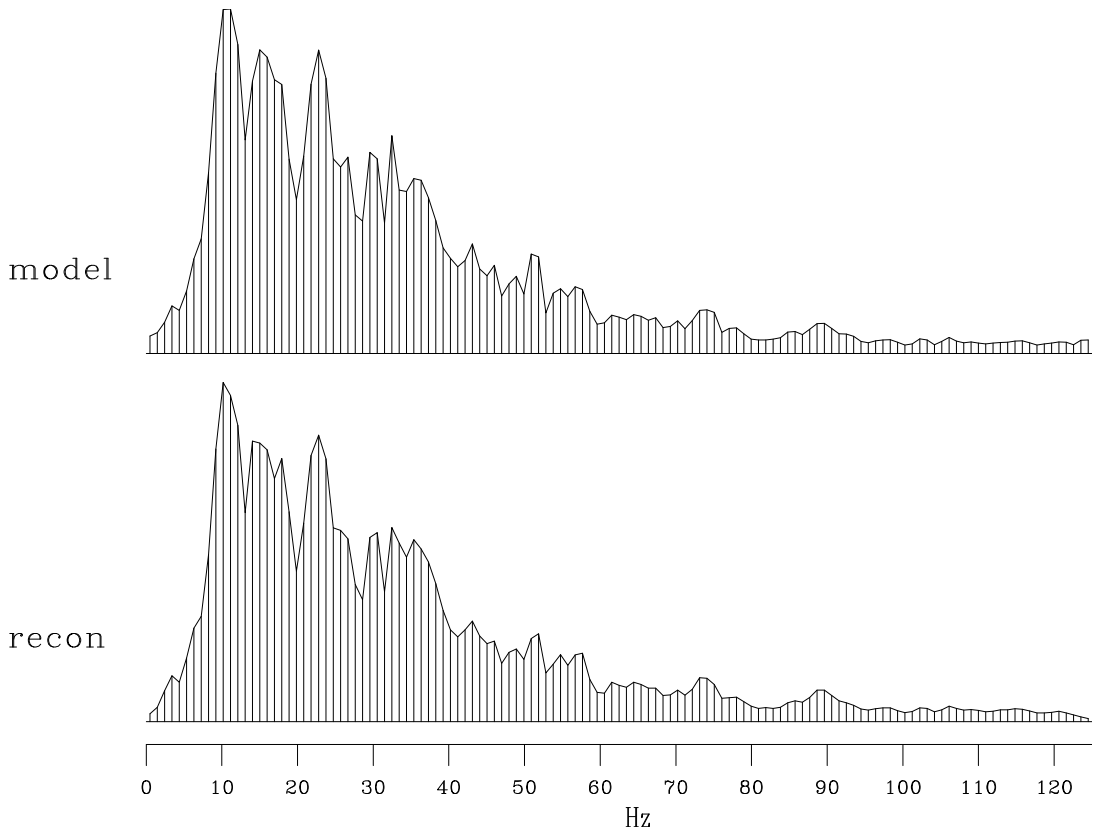


Figure 7.8: Top is the temporal spectrum of the model. Bottom is the spectrum of the reconstructed model. dwnc-phaspec [ER]

Figure 7.9 shows the effect of coarsening the space axis. Synthetic data is generated from an increasingly subsampled model. Again we notice that the phase-shift method of this chapter produces more plausible results than the simple Kirchhoff programs of chapter 5.

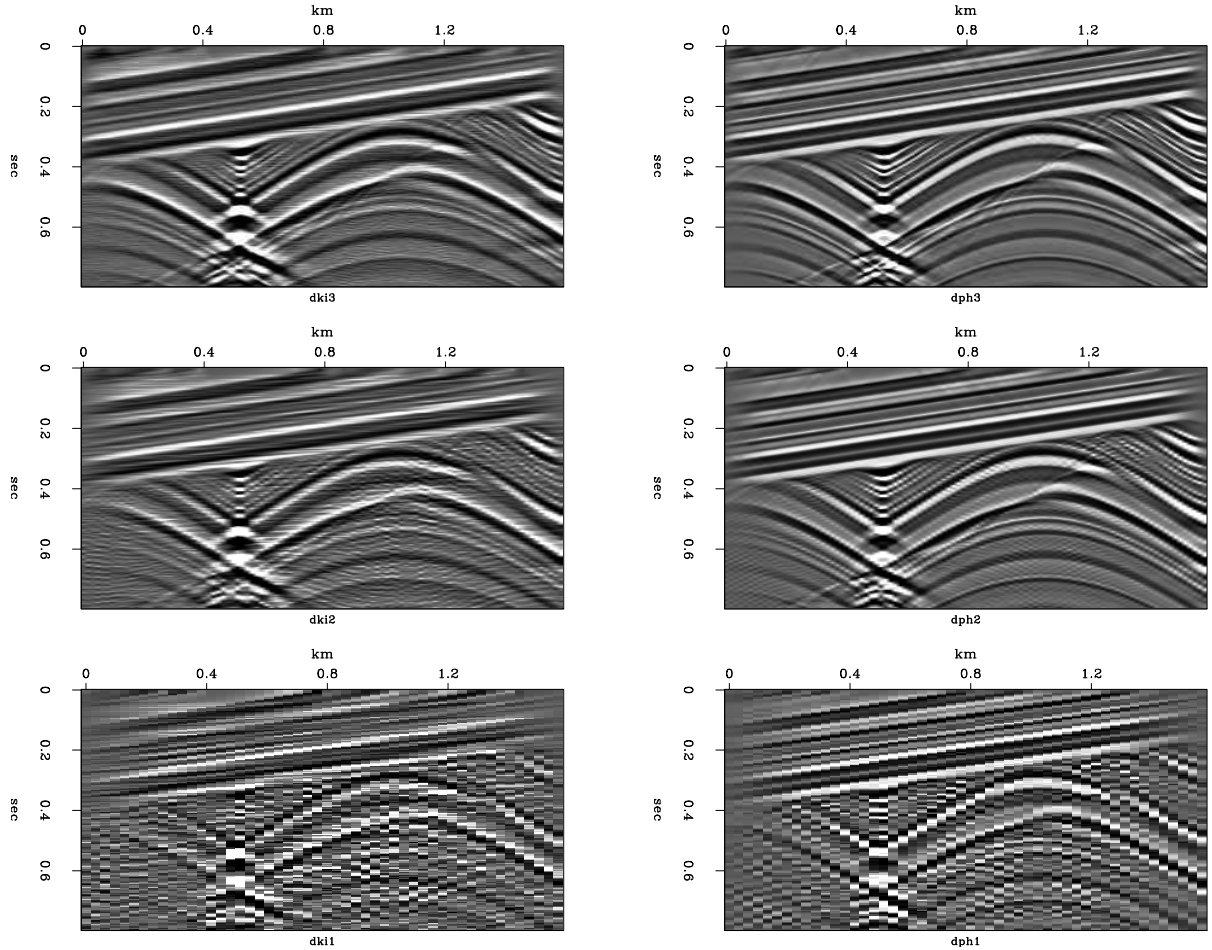


Figure 7.9: Modeling with increasing amounts of lateral subsampling. Left is the nearest-neighbor Kirchhoff method. Right is the phase-shift method. Top has 200 channels, middle has 100 channels, and bottom has 50 channels. dwnc-commod [ER]

7.3.3. Damped square root

The definition of k_z as $k_z = \sqrt{\omega^2/v^2 - k_x^2}$ obscures two aspects of k_z. First, which of the two square roots is intended, and second, what happens when $k_x^2 > \omega^2/v^2$. For both coding and theoretical work we need a definition of ik_z that is valid for both positive and negative values of ω and for all k_x. Define a function $R = ik_z(\omega, k_z)$ by

$$R \quad = \quad ik_z \quad = \quad \sqrt{(-i\omega + \epsilon)^2 + k_x^2} \tag{7.17}$$

It is important to know that for any $\epsilon > 0$, and any real ω and real k_x that the real part $\Re R > 0$ is positive. This means we can extrapolate waves safely with e^{-Rz} for increasing z or with e^{+Rz} for decreasing z. To switch from downgoing to upcoming we use the complex conjugate $\overline{R}$. Thus we have disentangled the damping from the direction of propagation.

Let us see why $\Re R > 0$ is positive for all real values of ω and k_x. Recall that for ω ranging between $\pm\infty$, $e^{i\omega\Delta t}$ rotates around the unit circle in the complex plane. Examine Figure 7.10 which shows the complex functions:

1. $f(\omega) = \epsilon - i\omega$,
2. $-i\hat{\omega} = (1+\epsilon) - e^{i\omega\Delta t}$,
3. $(-i\hat{\omega})^2$,
4. $(ik_z)^2 = (-i\hat{\omega})^2 + k_x^2$, and
5. $ik_z = [(-i\hat{\omega})^2 + k_x^2]^{1/2}$

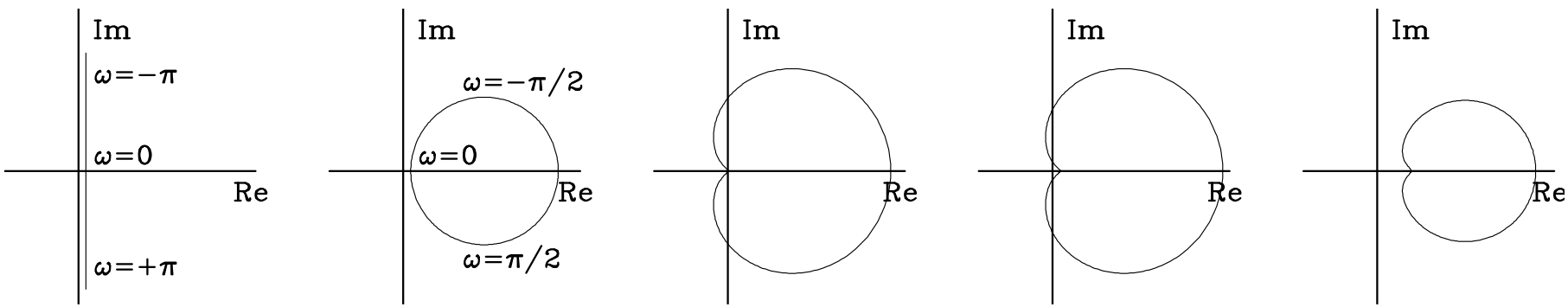


Figure 7.10: Some functions in the complex plane. dwnc-francis [ER]

The first two panels are explained by the first two functions. The first two functions and the first two panels look different but they become the same in the practical limit of $\epsilon \to 0$ and $\Delta t \to 0$. The left panel represents a time derivative in continuous time, and the second panel likewise in sampled time is for a "causal finite-difference operator" representing a time derivative. Notice that the graphs look the same near $\omega = 0$. As we sample seismic data with increasing density, $\Delta t \to 0$, the frequency content shifts further away from the Nyquist frequency. Measuring ω in radians/sample, in the limit $\Delta t \to 0$, the physical energy is all near $\omega = 0$.

The third panel in Figure 7.10 shows $(-i\hat{\omega})^2$ which is a cardioid that wraps itself close up to the negative imaginary axis without touching it. (To understand the shape near the origin, think about the square of the leftmost plane. You may have seen examples of the negative imaginary axis being a branch cut.) In the fourth panel a small positive quantity k_x^2 is added which shifts the cardioid to the right a bit. Taking the square root gives the last panel which shows the curve in the right half plane thus proving the important result we need, that $\Re\, ik_z(\omega, k_z) > 0$ for all real ω. It is also positive for all real k_x because any $k_x^2 > 0$ shifts the cardioid to the right. The additional issue of time causality in forward modeling is covered in IEI.

```
complex function eiktau( dt, w, vkx, qi )
real                     dt, w, vkx, qi
eiktau = cexp( - dt * csqrt( cmplx( qi, -w) ** 2   +   vkx * vkx /4. ) )
return; end
```

Back

Luckily the Fortran `csqrt()` function assumes the phase of the argument is between $\pm 180^\circ$ exactly as we need here. Thus the square root itself will have a phase between $\pm 90^\circ$ as we require. In applications, ϵ would typically be chosen proportional to the maximum time on the data. Thus the mathematical expression $-i\omega + \epsilon$ might be rendered in Fortran as `cmplx(qi,-omega)` where `qi=1./tmax` and the whole concept implemented as in function `eiktau()` /prog:eiktau. Do not set `qi=0` because then the `csqrt()` function cannot decipher positive from negative frequencies. eiktau

Finally, you might ask, why bother with all this careful theory connected with the damped square root. Why not simply abandon the evanescent waves as done by the "`if`" statement in subroutines `phasemig()` and `phasemod()`? There are several reasons:

1. The exploding reflector concept fails for evanescent waves (when $\omega^2 < v^2 k_x^2$). Realistic modeling would have them damping with depth. Rather than trying to handle them correctly we will make a choice, either (1) to abandon evanescent waves effectively setting them to zero, or (2) we will take them to be damping. (You might notice that when we switch from downgoing to upgoing, a damping exponential switches to a growing expo-

nential, but when we consider the adjoint of applying a damped exponential, that adjoint is also a damped exponential.)

I'm not sure if there is a practical difference between choosing to damp evanescent waves or simply to set them to zero, but there should be a noticable difference on synthetic data: When a Fourier-domain amplitude drops abruptly from unity to zero, we can expect a time-domain signal that spreads widely on the time axis, perhaps dropping off slowly as $1/t$. We can expect a more concentrated pulse if we include the evanescent energy, even though it is small. I predict the following behavior: Take an impulse; diffract it and then migrate it. When evanescent waves have been truncated, I predict the impulse is turned into a "butterfly" whose wings are at the hyperbola asymptote. Damping the evanescent waves, I predict, gives us more of a "rounded" impulse.

2. In a later chapter we will handle the x-axis by finite differencing (so that we can handle $v(x)$. There a stability problem will develop unless we begin from careful definitions as we are doing here.

3. Seismic theory includes an abstract mathematical concept known as branch-

line integrals. Such theory is most easily understood beginning from here.

7.3.4. Adjointness and ordinary differential equations

It is straightforward to adapt the simple programs `phasemig()` /prog:phasemig and `phasemod()` /prog:phasemod to depth variable velocity. As you might suspect, the two processes are adjoint to each other, and for reasons given at the end of chapter 2 it is desirable to code them to be so. With differential equations and their boundary conditions, the concept of adjoint is more subtle than previous examples. Thus, I postponed till here the development of adjoint code for phase-shift migration. This coding is a little strenuous, but it affords a review of many basic concepts, so we do so here. (Feel free to skip this section.) It is nice to have a high quality code for this fundamental operation.

Many situations in physics are expressed by the differential equation

$$\frac{du}{dz} - i\alpha u \quad = \quad s(z) \tag{7.18}$$

In the migration application, $u(z)$ is the up-coming wave, $\alpha = -\sqrt{\omega^2/v^2 - k_x^2}$, $s(z)$

is the exploding-reflector source. We take the medium to be layered (v constant in layers) so that α is constant in a layer, and we put the sources at the layer boundaries. Thus within a layer we have $du/dz - i\alpha u = 0$ which has the solution

$$u(z_k + \Delta z) \quad = \quad u(z_k)\, e^{i\alpha \Delta z} \tag{7.19}$$

For convenience, we use the "**delay operator**" in the k-th layer $Z_k = e^{-i\alpha\Delta z}$ so the delay of upward propagation is expressed by $u(z_k) = Z_k\, u(z_k + \Delta z)$. (Since α is negative for upcoming waves, $Z_k = e^{-i\alpha\Delta z}$ has a positive exponent which represents delay.) Besides crossing layers, we must cross layer boundaries where the (reflection) sources add to the upcoming wave. Thus we have

$$u_{k-1} \quad = \quad Z_{k-1} u_k + s_{k-1} \tag{7.20}$$

Recursive use of equation (7.20) across a medium of three layers is expressed in matrix form as

$$\mathbf{Mu} \quad = \quad \begin{bmatrix} 1 & -Z_0 & . & . \\ . & 1 & -Z_1 & . \\ . & . & 1 & -Z_2 \\ . & . & . & 1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad = \quad \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \quad = \quad \mathbf{s} \tag{7.21}$$

A recursive solution begins at the bottom with $u_3 = s_3$ and propagates upward.

The adjoint (complex conjugate) of the delay operator Z is the time advance operator $\bar{Z}$. The adjoint of equation (7.21) is given by

$$\mathbf{M}'\tilde{\mathbf{s}} \quad = \quad \begin{bmatrix} 1 & . & . & . \\ -\bar{Z}_0 & 1 & . & . \\ . & -\bar{Z}_1 & 1 & . \\ . & . & -\bar{Z}_2 & 1 \end{bmatrix} \begin{bmatrix} \tilde{s}_0 \\ \tilde{s}_1 \\ \tilde{s}_2 \\ \tilde{s}_3 \end{bmatrix} \quad = \quad \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad = \quad \mathbf{u} \tag{7.22}$$

where $\tilde{s}(z)$ (summed over frequency) is the migrated image. The adjointness of equation (7.21) and (7.22) seems obvious, but it is not the elementary form we are familiar with because the matrix multiplies the *output* (instead of multiplying the usual *input*). To prove the adjointness, notice that equation (7.21) is equivalent to $\mathbf{u} = \mathbf{M}^{-1}\mathbf{s}$ whose adjoint, by definition, is $\tilde{\mathbf{s}} = (\mathbf{M}^{-1})'\mathbf{u}$ which is $\tilde{\mathbf{s}} = (\mathbf{M}')^{-1}\mathbf{u}$ (because of the basic mathematical fact that the adjoint of an inverse is the inverse of the adjoint) which gives $\mathbf{M}'\tilde{\mathbf{s}} = \mathbf{u}$ which is equation (7.22).

We observe the wavefield only on the surface $z = 0$, so the adjointness of equations (7.21) and (7.22) is academic because it relates the wavefield at all depths with the source at all depths. We need to truncate $\mathbf{u}$ to its first coefficient u_0 since

the upcoming wave is known only at the surface. This truncation changes the adjoint in a curious way. We rewrite equation (7.21) using a truncation operator $\mathbf{T}$ that is the row matrix $\mathbf{T} = [1, 0, 0, \cdots]$ getting $\mathbf{u}_0 = \mathbf{T}\mathbf{u} = \mathbf{T}\mathbf{M}^{-1}\mathbf{s}$. Its adjoint is $\hat{\mathbf{s}} = (\mathbf{M}^{-1})'\mathbf{T}'\mathbf{u}_0' = (\mathbf{M}')^{-1}\mathbf{T}'\mathbf{u}_0'$ or $\mathbf{M}'\hat{\mathbf{s}} = \mathbf{T}'\mathbf{u}_0$ which looks like

$$\mathbf{M}'\tilde{\mathbf{s}} \quad = \quad \begin{bmatrix} 1 & . & . & . \\ -\bar{Z}_0 & 1 & . & . \\ . & -\bar{Z}_1 & 1 & . \\ . & . & -\bar{Z}_2 & 1 \end{bmatrix} \begin{bmatrix} \tilde{s}_0 \\ \tilde{s}_1 \\ \tilde{s}_2 \\ \tilde{s}_3 \end{bmatrix} \quad = \quad \begin{bmatrix} u_0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.23}$$

The operator 7.23 is a recursion beginning from $\tilde{s}_0 = u_0$ and continuing downward with

$$\tilde{s}_k \quad = \quad \bar{Z}_{k-1}\, \tilde{s}_{k-1} \tag{7.24}$$

A final feature of the migration application is that the image is formed from $\tilde{\mathbf{s}}$ by summing over all frequencies. Although I believe the mathematics above and the code in subroutine `gazadj()` /prog:gazadj, I ran the dot product test to be sure!

gazadj Finally, a few small details about the code. The loop on spatial frequency `ikx` begins at `ikx=2`. The reason for the 2, instead of a 1, is to omit the Nyquist frequency. If the Nyquist frequency were to be included, it should be divided into

```
# Phase shift modeling and migration.   (Warning: destroys its input!)
#
subroutine gazadj( adj, dt,dx, v,nt,nx, modl,          data )
integer             adj,          nt,nx,                 iw, ikx, iz,nz
complex eiktau, cup,                      modl(nt,nx),  data(nt,nx)
real                        dt,dx, v(nt),              pi, w,w0,dw, kx,kx0,dkx,qi
call adjnull(     adj,  0,                modl,nt*nx*2, data,nt*nx*2 )
pi = 4.*atan(1.);       w0  = -pi/dt;   dw = 2.*pi/(nt*dt);    qi=.5/(nt*dt)
nz = nt;                kx0 = -pi/dx;   dkx= 2.*pi/(nx*dx)
if( adj == 0)   call ft2axis( 0, -1., nz, nx, modl)
else    {       call ft2axis( 0, -1., nt, nx, data)
                call ft1axis( 0,  1., nt, nx, data)
                }
do ikx  = 2, nx     {       kx = kx0 + (ikx-1) * dkx
do iw   = 2, 1+nt/2 {       w  = w0  + (iw -1) * dw
        if( adj== 0) { data(iw,ikx) = modl(nz,ikx)
                do iz = nz-1, 1, -1
                        data(iw,ikx) = data(iw,ikx) * eiktau(dt,w,v(iz)*kx,qi) +
                                       modl(iz,ikx)
                }
        else {          cup          = data(iw,ikx)
                do iz = 1, nz {
                        modl(iz,ikx) = modl(iz,ikx)  +  cup
                        cup          = cup *  conjg(  eiktau(dt,w,v(iz)*kx,qi))
                        }
                }
        }}
if( adj == 0)  {  call ft1axis( 1,  1., nt, nx, data)
                  call ft2axis( 1, -1., nt, nx, data) }
else            { call ft2axis( 1, -1., nz, nx, modl) }
return; end
```

Back

one half at positive Nyquist and one half at negative Nyquist, which would clutter the code without adding practical value. Another small detail is that the loop on temporal frequency `iw` begins at `iw=1+nt/2` which effectly omits negative frequencies. This is purely an economy measure. Including the negative frequencies would assure that the final image be real, no imaginary part. Omitting negative frequencies simply gives an imaginary part that can be thrown away, and gives the same real image, scaled by a half. The factor of two speed up makes these tiny compromises well worthwhile.

7.3.5. Vertical exaggeration example

To examine questions of **vertical exaggeration** and spatial **resolution** we consider a line of point scatters along a 45° dipping line in (x,z)-space. We impose a linear velocity gradient such as that typically found in the Gulf of Mexico, i.e. $v(z) = v_0 + \alpha z$ with $\alpha = 1/2s^{-1}$. Viewing our point scatterers as a function of traveltime depth, $\tau = 2\int_0^z dz/v(z)$ in Figure 7.11 we see, as expected, that the points, although separated by equal intervals in x, are separated by shorter time intervals with increasing depth. The points are uniformly separated along a straight line in (x,z)-space, but

they are nonuniformly separated along a *curved* line in (x, τ)-space. The curve is steeper near the earth's surface where $v(z)$ yields the greatest vertical exaggeration. Here the vertical exaggeration is about unity (no exageration) but deeper the vertical exaggeration is less than unity (horizontal exaggeration).

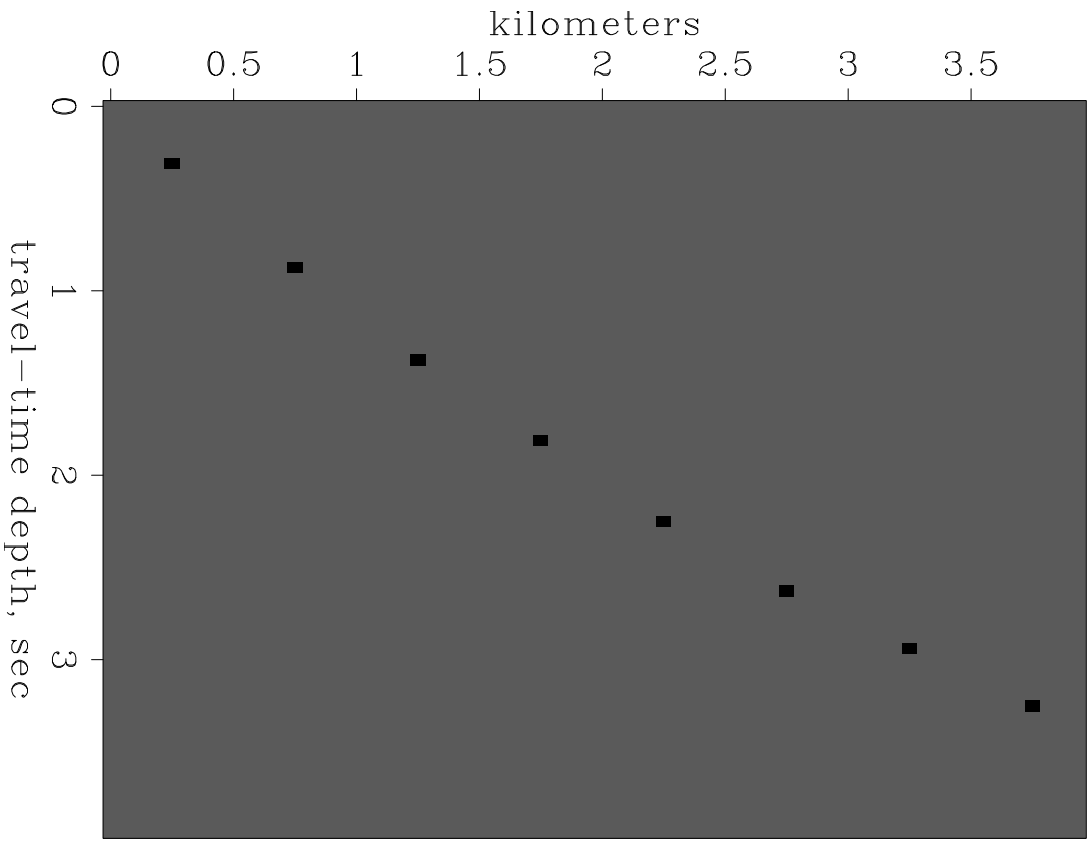


Figure 7.11: Points along a 45 degree slope as seen as a function of traveltime depth. dwnc-sagmod [ER]

Applying subroutine `gazadj()` /prog:gazadj the points spray out into hyperboloids (like hyperbolas, but not exactly) shown in Figure 7.12. The obvious feature of this synthetic data is that the hyperboloids appear to have different asymptotes.

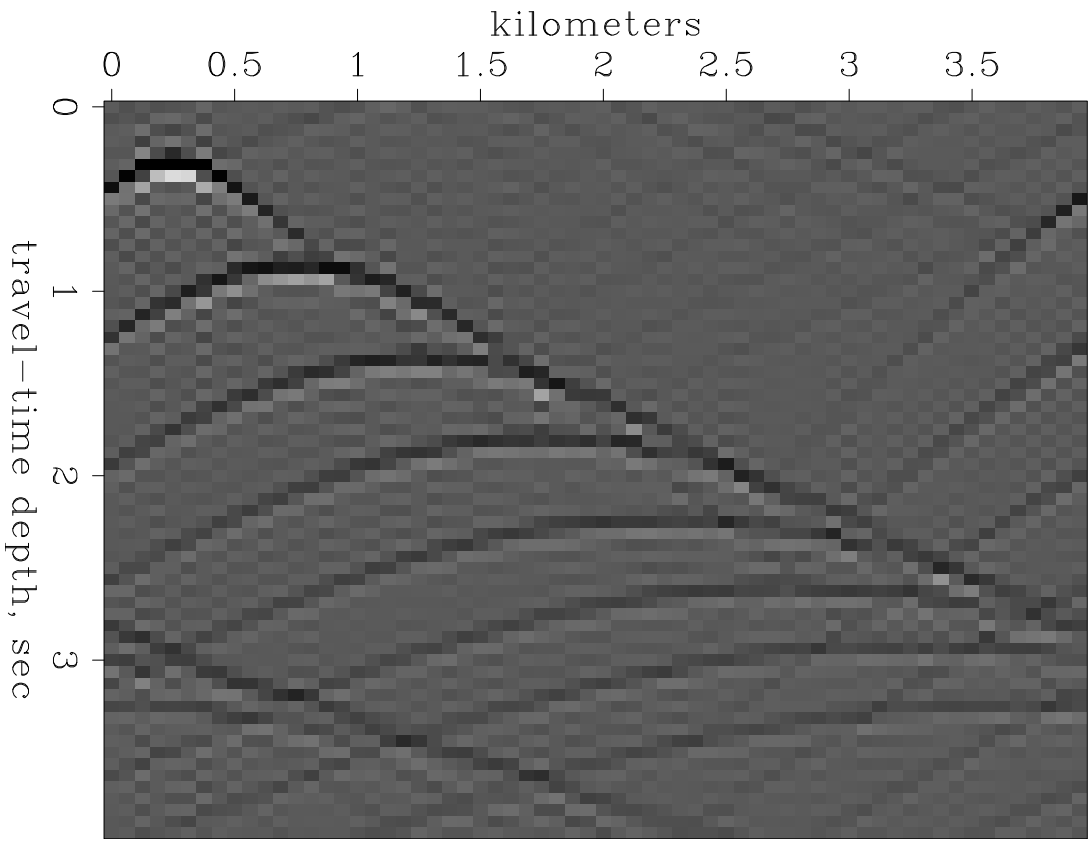


Figure 7.12: The points of Figure 7.11 diffracted into hyperboloids. dwnc-sagdat [ER]

In fact, there are no asymptotes because an asymptote is a ray going horizontal at a more-or-less constant depth, which will not happen in this model because the velocity increases steadily with depth.

(I should get energetic and overlay these hyperboloids on top of the exact hyperbolas of the Kirchhoff method, to see if there are perceptible traveltime differences.)

7.3.6. Vertical and horizontal resolution

In principle, migration converts hyperbolas to points. In practice, a hyperbola does not collapse to a point, it collapses to a *focus*. A focus has measurable dimensions. Vertical resolution is easily understood. For a given frequency, higher velocity gives longer vertical wavelength and thus less resolution. When the result of migration is plotted as a function of traveltime depth τ instead of true depth z, however, enlargement of focus with depth is not visible.

Horizontal resolution works a little differently. Migration is said to be "good" because it increases spatial **resolution**. It squeezes a large hyperbola down to a tiny focus. Study the focal widths in Figure 7.13. Notice the water-velocity focuses hardly broaden with depth. We expect some broadening with depth because the late

hyperbolas are cut off at their sides and bottom (an aperture effect), but there is no broadening here because the periodicity of the Fourier domain means that events are not truncated but wrapped around.

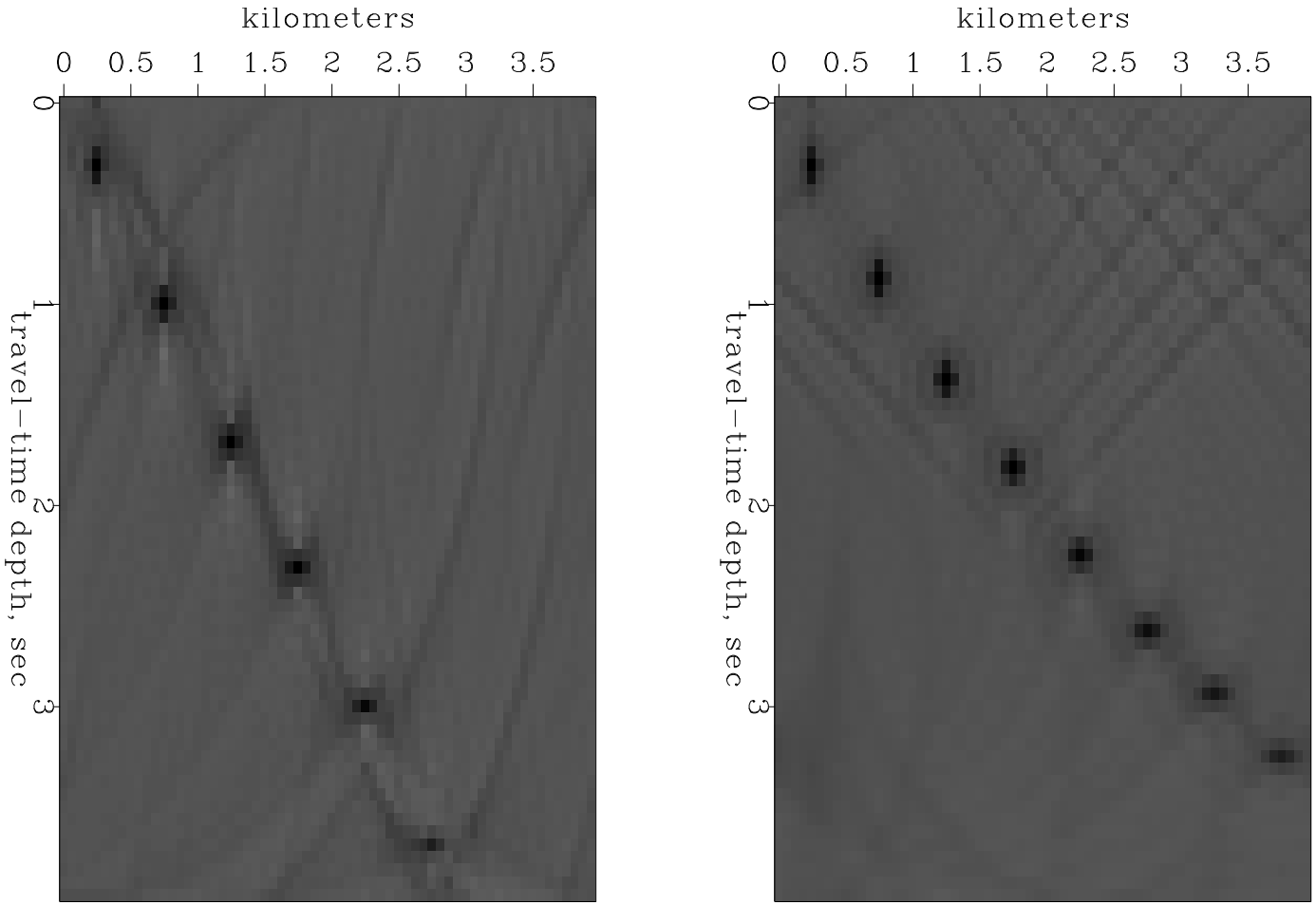


Figure 7.13: Left is migration back to a focus with a constant, water-velocity model. Right is the same, but with a Gulf of Mexico velocity, i.e. the hyperboloids of Figure 7.12 migrated back to focuses. Observe focus broadening with depth. dwnc-sagres [ER]

When the velocity increases with depth, wider focuses are found at increasing depth. Why is this? Consider each hyperbola to be made of many short plane wave segments. Migration moves all the little segments on top of each other. The sharpness of a focus cannot be narrower than the width of each of the many plane-wave segments that superpose to make the focus. The narrowest of these plane-wave segments is at the steepest part of a hyperbola asymptote. Deeper reflectors (which have later tops) have less steep asymptotes because of the increasing velocity. Thus deeper reflectors with faster RMS velocities have wider focuses so the deeper part of the image is more blurred. A second way to understand increased blurring with depth is from equation (7.12), that the horizontal frequency $k_x = \omega p = \omega v^{-1} \sin\theta$ is independent of depth. The steepest possible angle occurs when $|\sin\theta| = 1$. Thus, considering all possible angles, the largest $|k_x|$ is $|k_x| = |\omega|/v(z)$. Larger values of horizontal frequency $|k_x|$ could help us get narrower focuses, but the deeper we go (faster velocity we encounter), the more these high frequencies are lost because of the evanescent limit $|k_x| \leq |\omega/v(z)|$. The limit is where the ray goes no deeper but bends around and comes back up again without ever reflecting. Any ray that does this many times is said to be a surface-trapped wave. It cannot sharpen a deep focus.

7.3.7. Field data migration

Application of subroutine `gazadj()` /prog:gazadj to the Gulf of Mexico data set processed in earlier chapters yields the result in Figure 7.14.

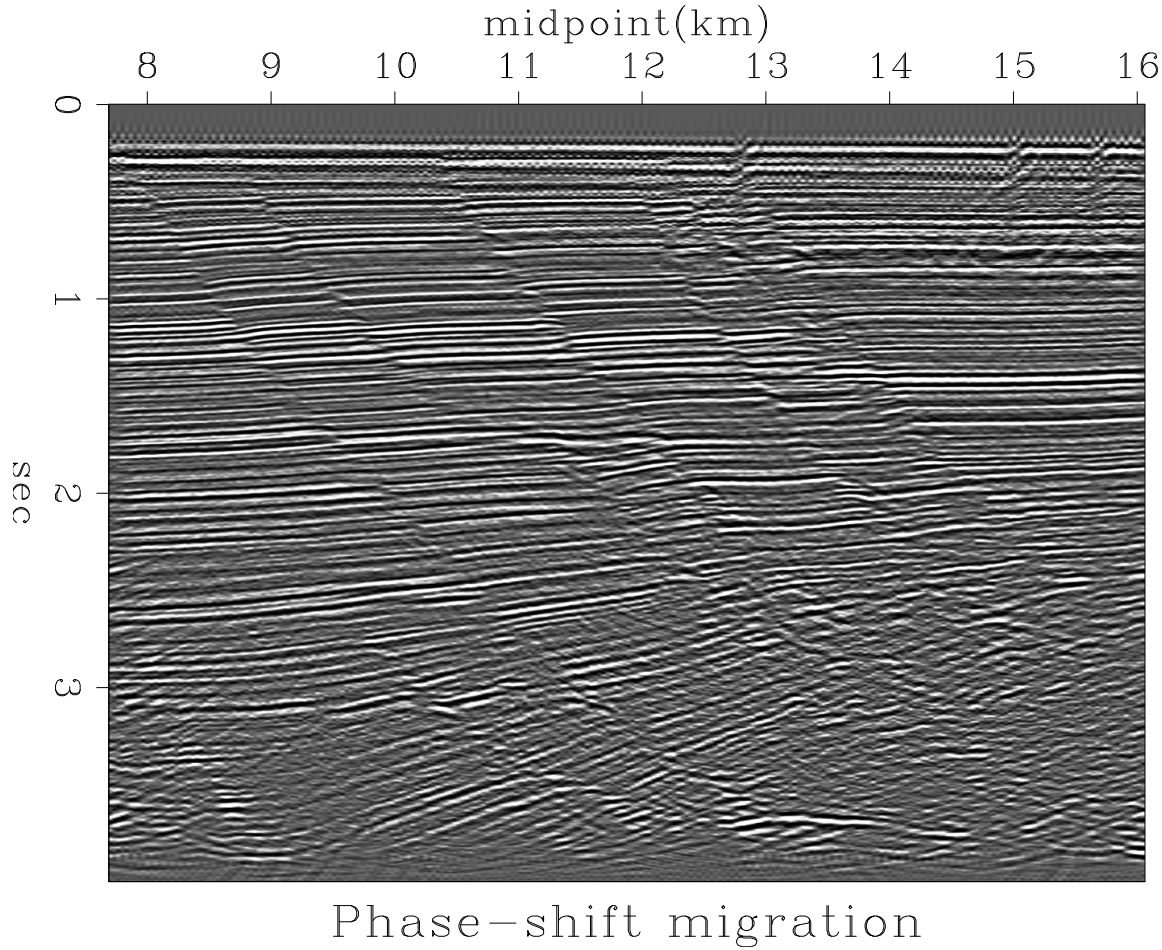


Figure 7.14: Phase shift migration of Figure 4.7. Press button for movie to compare to stack and Kirchhoff migration of Figure 4.6. dwnc-wgphase [ER,M]

EXERCISES:

1 Devise a mathematical expression for a plane wave that is an impulse function of time with a propagation angle of 15° from the vertical z-axis in the plus z direction. Express the result in the domain of

(a) (t, x, z)

(b) (ω, x, z)

(c) (ω, k_x, z)

(d) (ω, p, z)

(e) (ω, k_x, k_z)

(f) (t, k_x, k_z)

Chapter 8

Dip and offset together

[1]When dip and offset are combined, some serious complications arise. For many years it was common industry practice to ignore these complications and to handle

[1] Matt Schwab prepared a draft of the Gardner DMO derivation. Shuki Ronen gave me the “law of cosines” proof.

dip and offset separately. Thus offset was handled by velocity analysis, normal moveout and stack (chapter 4). And dip was handled by zero-offset migration after stack (chapters 5 and 7). This practice is a good approximation only when the dips on the section are small. We need to handle large offset angles at the same time we handle large dip angles at the same time we are estimating rock velocity. It is confusing! Here we see the important steps of bootstrapping yourself towards both the velocity and the image.

8.1. PRESTACK MIGRATION

Prestack migration creates an image of the earth's reflectivity directly from prestack data. It is an alternative to the "**exploding reflector**" concept that proved so useful in zero-offset migration. In **prestack migration**, we consider both downgoing and upcoming waves.

A good starting point for discussing prestack migration is a reflecting point within the earth. A wave incident on the point from any direction reflects waves in all directions. This geometry is particularly important because any model is a superposition of such point scatterers. The point-scatterer geometry for a point

located at (x, z) is shown in Figure 8.1. The equation for travel time t is the sum of

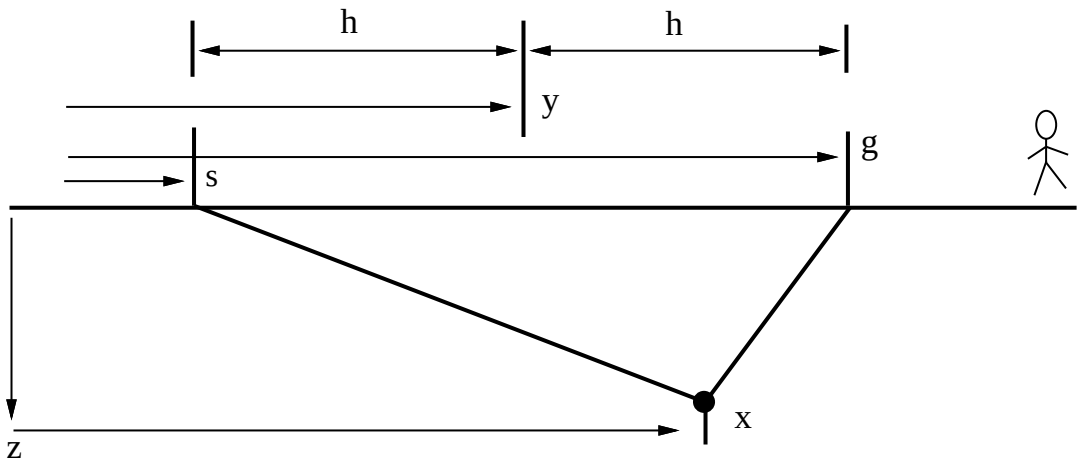


Figure 8.1: Geometry of a point scatterer. dpmv-pgeometry [NR]

the two travel paths is

$$t\,v \quad = \quad \sqrt{z^2 + (s - x)^2} + \sqrt{z^2 + (g - x)^2} \tag{8.1}$$

We could model field data with equation (8.1) by copying reflections from any point in (x, z)-space into (s, g, t)-space. The adjoint program would form an image stacked over all offsets. This process would be called prestack migration. The problem here is that the real problem is estimating velocity. In this chapter we will see that it

is not satisfactory to use a horizontal layer approximation to estimate velocity, and then use equation (8.1) to do migration. Migration becomes sensitive to velocity when wide angles are involved. Errors in the velocity would spoil whatever benefit could accrue from prestack (instead of poststack) migration.

8.1.1. Cheops' pyramid

Because of the importance of the point-scatterer model, we will go to considerable lengths to visualize the functional dependence among t, z, x, s, and g in equation (8.1). This picture is more difficult—by one dimension—than is the conic section of the exploding-reflector geometry.

To begin with, suppose that the first square root in (8.1) is constant because everything in it is held constant. This leaves the familiar hyperbola in (g,t)-space, except that a constant has been added to the time. Suppose instead that the other square root is constant. This likewise leaves a hyperbola in (s,t)-space. In (s,g)-space, travel time is a function of s plus a function of g. I think of this as one coat hanger, which is parallel to the s-axis, being hung from another coat hanger, which is parallel to the g-axis.

A view of the traveltime pyramid on the (s, g)-plane or the (y, h)-plane is shown in Figure 8.2a.

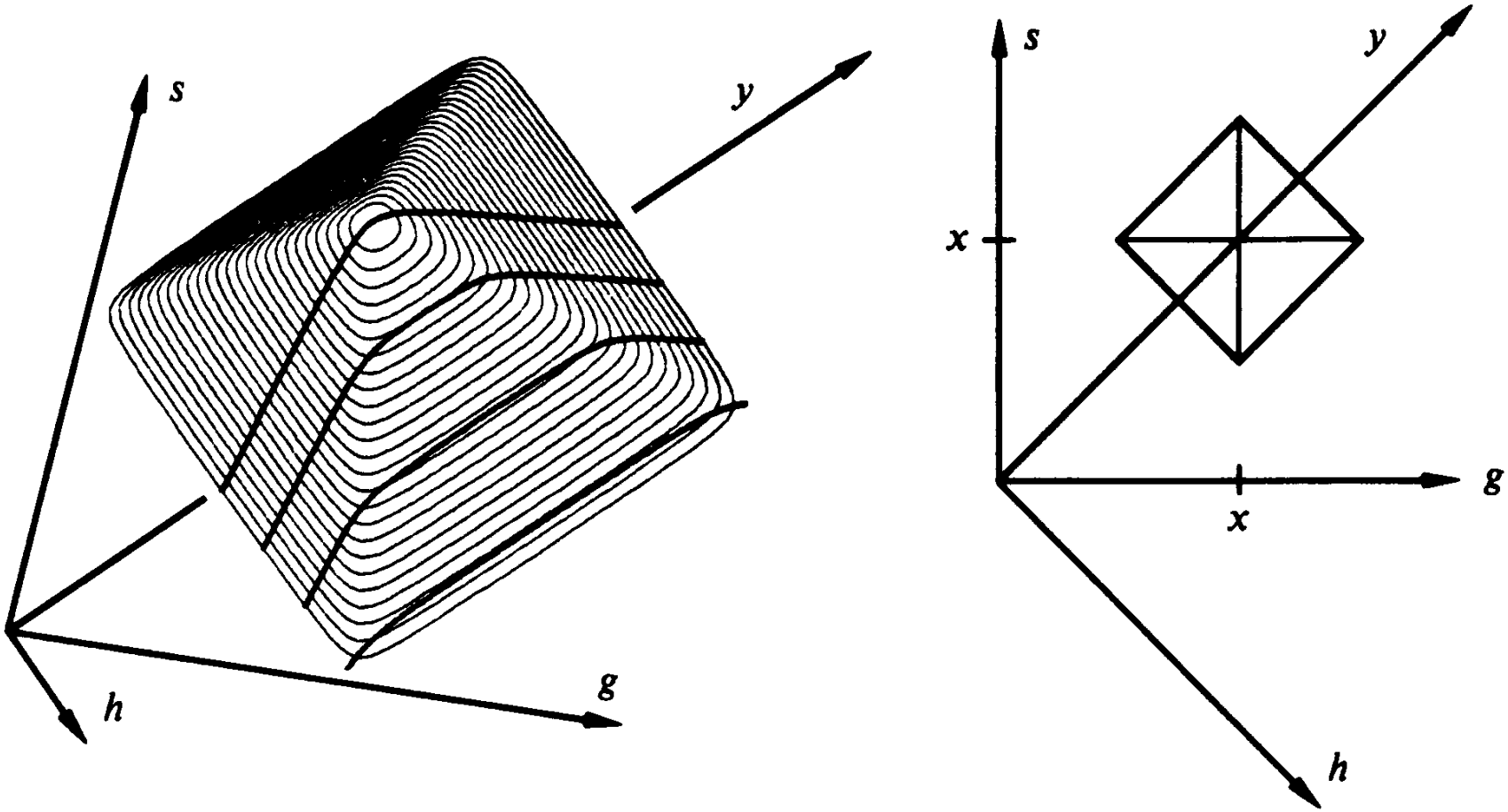


Figure 8.2: Left is a picture of the traveltime pyramid of equation ((8.1)) for fixed x and z. The darkened lines are constant-offset sections. Right is a cross section through the pyramid for large t (or small z). (Ottolini) dpmv-cheop [NR]

Notice that a cut through the pyramid at large t is a square, the corners of which have been smoothed. At very large t, a constant value of t is the square contoured in (s,g)-space, as in Figure 8.2b. Algebraically, the squareness becomes evident for a point reflector near the surface, say, $z \to 0$. Then (8.1) becomes

$$v\,t \quad = \quad |s - x| + |g - x| \tag{8.2}$$

The center of the square is located at $(s,g) = (x,x)$. Taking travel time t to increase downward from the horizontal plane of (s,g)-space, the square contour is like a horizontal slice through the Egyptian pyramid of Cheops. To walk around the pyramid at a constant altitude is to walk around a square. Alternately, the altitude change of a traverse over g (or s) at constant s (or g) is simply a constant plus an absolute-value function.

More interesting and less obvious are the curves on common-midpoint gathers and constant-offset sections. Recall the definition that the midpoint between the shot and geophone is y. Also recall that h is half the horizontal offset from the shot to the geophone.

$$y \quad = \quad \frac{g + s}{2} \tag{8.3}$$

$$h \quad = \quad \frac{g - s}{2} \tag{8.4}$$

A traverse of y at constant h is shown in Figure 8.2. At the highest elevation on the traverse, you are walking along a flat horizontal step like the flat-topped hyperboloids of Figure 8.8. Some erosion to smooth the top and edges of the pyramid gives a model for nonzero reflector depth.

For rays that are near the vertical, the traveltime curves are far from the hyperbola asymptotes. Then the square roots in (8.1) may be expanded in Taylor series, giving a parabola of revolution. This describes the eroded peak of the pyramid.

8.1.2. Prestack migration ellipse

Denoting the horizontal coordinate x of the scattering point by y_0 Equation (8.1) in (y,h)-space is

$$t\,v \quad = \quad \sqrt{z^2 + (y - y_0 - h)^2} + \sqrt{z^2 + (y - y_0 + h)^2} \tag{8.5}$$

A basic insight into equation (8.1) is to notice that at constant-offset h and constant travel time t the locus of possible reflectors is an ellipse in the (y,z)-plane centered

at y_0. The reason it is an **ellipse** follows from the geometric definition of an ellipse. To draw an ellipse, place a nail or tack into s on Figure 8.1 and another into g. Connect the tacks by a string that is exactly long enough to go through (y_0, z). An ellipse going through (y_0, z) may be constructed by sliding a pencil along the string, keeping the string tight. The string keeps the total distance tv constant as is shown in Figure 8.3

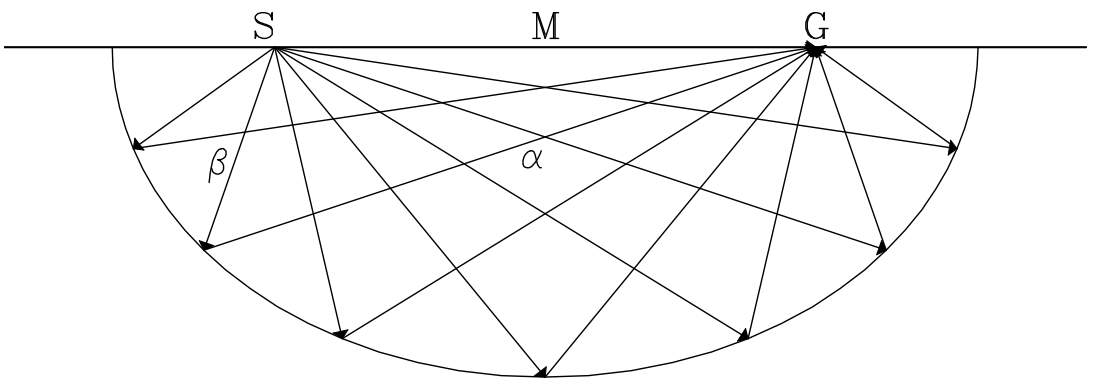


Figure 8.3: Prestack migration ellipse, the locus of all scatterers with constant traveltime for source S and receiver G. dpmv-ellipse1 [ER,M]

Replacing depth z in equation (8.5) by the vertical traveltime depth $\tau = 2z/v = z/v_{\text{half}}$ we get

$$t = \frac{1}{2}\left(\sqrt{\tau^2 + [(y-y_0)-h]^2/v_{\text{half}}^2} + \sqrt{\tau^2 + [(y-y_0)+h]^2/v_{\text{half}}^2}\right) \tag{8.6}$$

8.1.3. Constant offset migration

Considering h in equation (8.6) to be a constant, enables us to write a subroutine for migrating constant-offset sections. Subroutine `flathyp()` /prog:flathyp is easily prepared from subroutine `kirchfast()` /prog:kirchfast by replacing its hyperbola equation with equation (8.6). flathyp The amplitude in subroutine `flathyp()` should be improved when we have time to do so. Forward and backward responses to impulses of subroutine `flathyp()` are found in Figures 8.4 and 8.5.

```
# Flat topped hyperbolas and constant-offset section migration
#
subroutine flathyp(   adj, add, vel    , h, t0,dt,dx, modl,nt,nx, data)
integer ix,iz,it,ib,  adj, add,                          nt,nx
real    t, amp, z,b,             vel(nt), h, t0,dt,dx, modl(nt,nx),data(nt,nx)
call adjnull(        adj, add,                          modl,nt*nx, data,nt*nx)
do ib= -nx, nx {        b = dx * ib              # b = midpt separation y-y0
    do iz= 2, nt {      z = t0 + dt * (iz-1)     # z = zero-offset time
        t = .5 * ( sqrt( z**2 +((b-h)*2/vel(iz))**2) +
                   sqrt( z**2 +((b+h)*2/vel(iz))**2)   )
        it = 1.5 + (t - t0) / dt
     if( it > nt )                       break
        amp = (z/t)/ sqrt(t)
        do ix= max0(1, 1-ib),  min0(nx, nx-ib)
                 if( adj == 0 )
                        data(it,ix+ib)= data(it,ix+ib) + modl(iz,ix   ) * amp
                 else
                        modl(iz,ix   )= modl(iz,ix   ) + data(it,ix+ib) * amp
        }
    }
return; end
```

Back

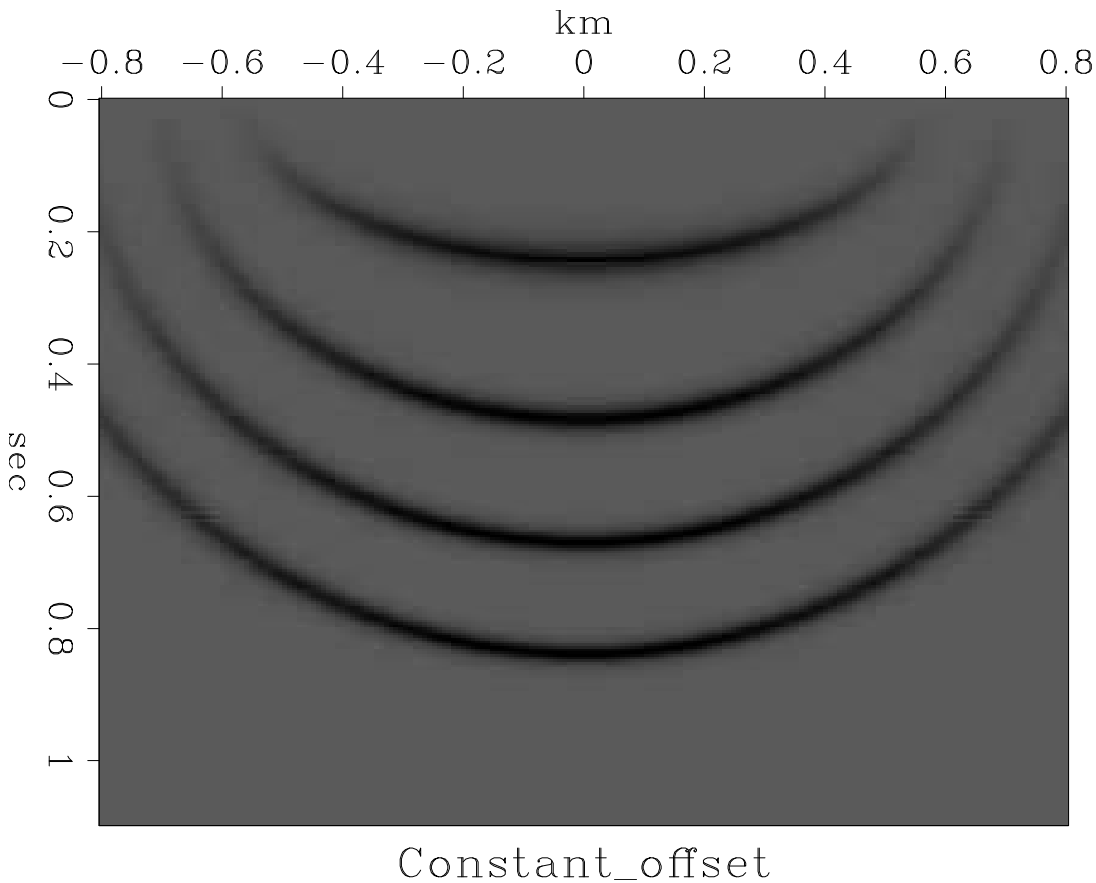


Figure 8.4: Migrating impulses on a constant-offset section with subroutine `flathyp()`. Notice that shallow impulses (shallow compared to h) appear ellipsoidal while deep ones appear circular. dpmv-Cos.1 [ER]

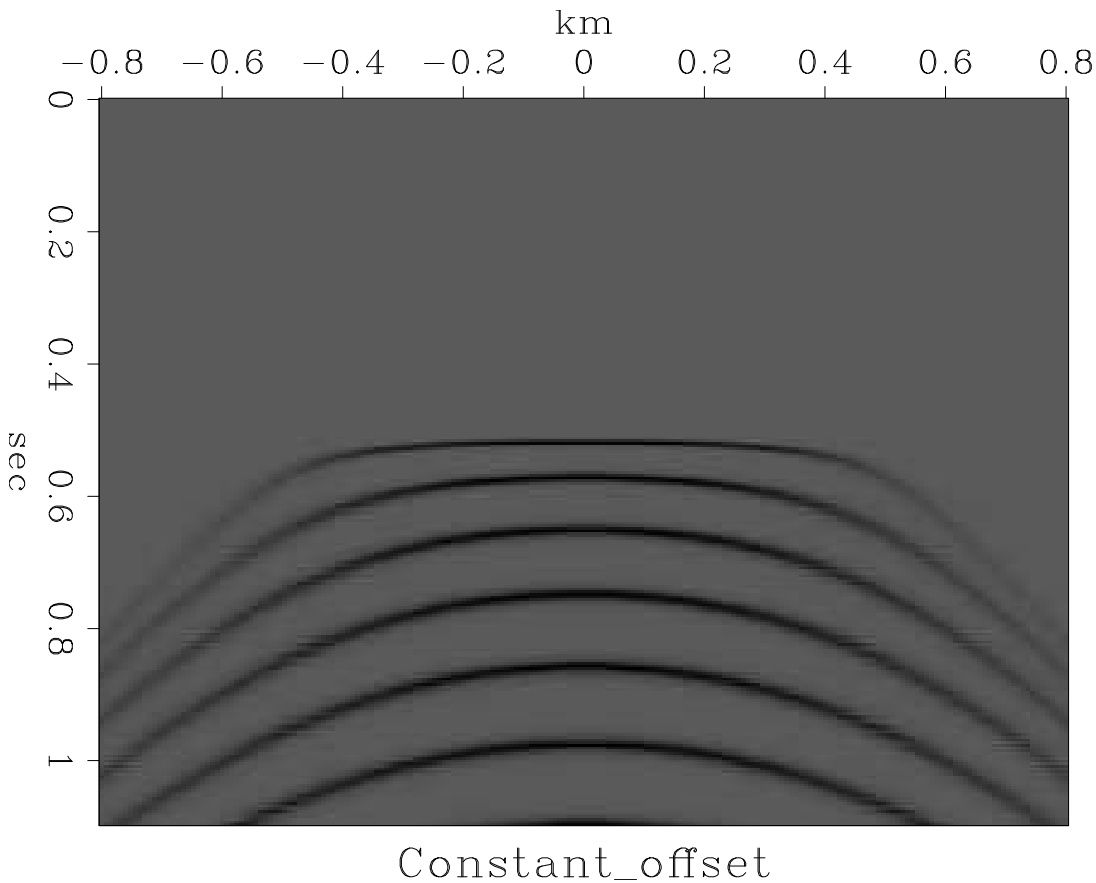


Figure 8.5: Forward modeling from an earth impulse with subroutine `flathyp()`. dpmv-Cos.0 [ER]

It is not easy to show that equation (8.5) can be cast in the standard mathematical form of an ellipse, namely, a stretched circle. But the result is a simple one, and an important one for later analysis. Feel free to skip forward over the following verification of this ancient wisdom. To help reduce algebraic verbosity, define a new y equal to the old one shifted by y_0. Also make the definitions

$$\begin{aligned} t\,v &= 2\,A \\ \alpha &= z^2 + (y+h)^2 \\ \beta &= z^2 + (y-h)^2 \\ \alpha - \beta &= 4\,y\,h \end{aligned} \tag{8.7}$$

With these definitions, (8.5) becomes

$$2\,A = \sqrt{\alpha} + \sqrt{\beta}$$

Square to get a new equation with only one square root.

$$4\,A^2 - (\alpha + \beta) = 2\sqrt{\alpha\beta}$$

Square again to eliminate the square root.

$$16\,A^4 - 8\,A^2(\alpha+\beta) + (\alpha+\beta)^2 = 4\,\alpha\,\beta$$

$$16\,A^4 \;-\; 8\,A^2(\alpha + \beta) \;+\; (\alpha - \beta)^2 \;=\; 0$$

Introduce definitions of α and β.

$$16\,A^4 \;-\; 8\,A^2\,[2z^2 + 2y^2 + 2h^2] \;+\; 16\,y^2h^2 \;=\; 0$$

Bring y and z to the right.

$$\begin{aligned} A^4 \;-\; A^2h^2 &= A^2(z^2 + y^2) \;-\; y^2h^2 \\ A^2(A^2 - h^2) &= A^2z^2 + (A^2 - h^2)y^2 \\ A^2 &= \frac{z^2}{1 - \frac{h^2}{A^2}} + y^2 \end{aligned} \tag{8.8}$$

Finally, recalling all earlier definitions and replacing y by $y - y_0$, we obtain the canonical form of an ellipse with semi-major axis A and semi-minor axis B:

$$\frac{(y - y_0)^2}{A^2} + \frac{z^2}{B^2} \;=\; 1\;, \tag{8.9}$$

where

$$A \;=\; \frac{v\,t}{2} \tag{8.10}$$

$$B \quad = \quad \sqrt{A^2 - h^2} \tag{8.11}$$

Fixing t, equation (8.9) is the equation for a circle with a stretched z-axis. The above algebra confirms that the "string and tack" definition of an **ellipse** matches the "stretched circle" definition. An **ellipse** in earth model space corresponds to an impulse on a constant-offset section.

8.2. INTRODUCTION TO DIP

We can consider a data space to be a superposition of points and then analyze the point response, or we can consider data space or model space to be a superposition of planes and then do an analysis of planes. Analysis of points is often easier than planes, but planes, particularly local planes, are more like observational data and earth models.

The simplest environment for reflection data is a single horizontal reflection interface, which is shown in Figure 8.6.

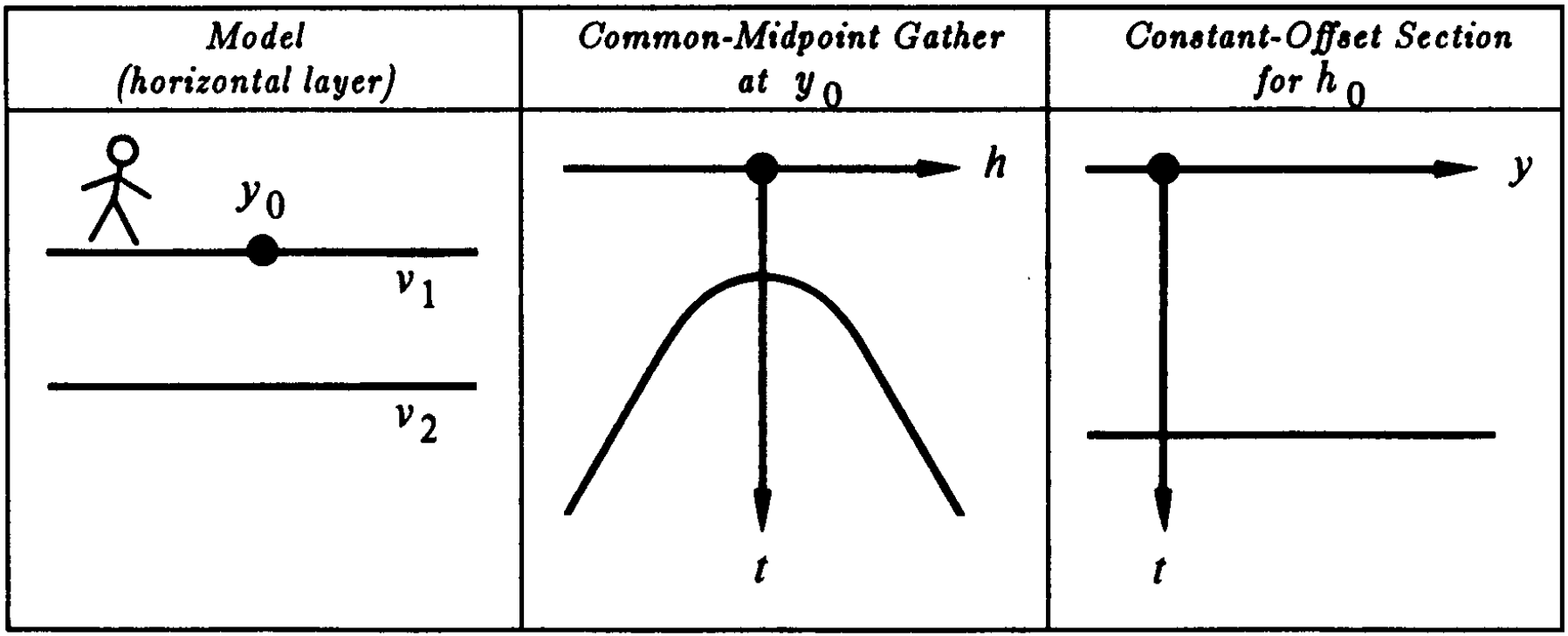


Figure 8.6: Simplest earth model. dpmv-simple [NR]

As expected, the zero-offset section mimics the earth model. The common-midpoint gather is a hyperbola whose asymptotes are straight lines with slopes of the inverse of the velocity v_1. The most basic data processing is called ***common-depth-point stack*** or **CDP stack**. In it, all the traces on the common-midpoint (CMP) gather are time shifted into alignment and then added together. The result mimics a zero-offset trace. The collection of all such traces is called the *CDP-stacked section.* In practice the CDP-stacked section is often interpreted and migrated as though it were a zero-offset section. In this chapter we will learn to avoid this popular, oversimplified assumption.

The next simplest environment is to have a planar reflector that is oriented vertically rather than horizontally. This might not seem typical, but the essential feature (that the rays run horizontally) really is common in practice (see for example Figure 8.9.) Also, the effect of dip, while generally complicated, becomes particularly simple in the extreme case. If you wish to avoid thinking of wave propagation along the air-earth interface you can take the reflector to be inclined a slight angle from the vertical, as in Figure 8.7.

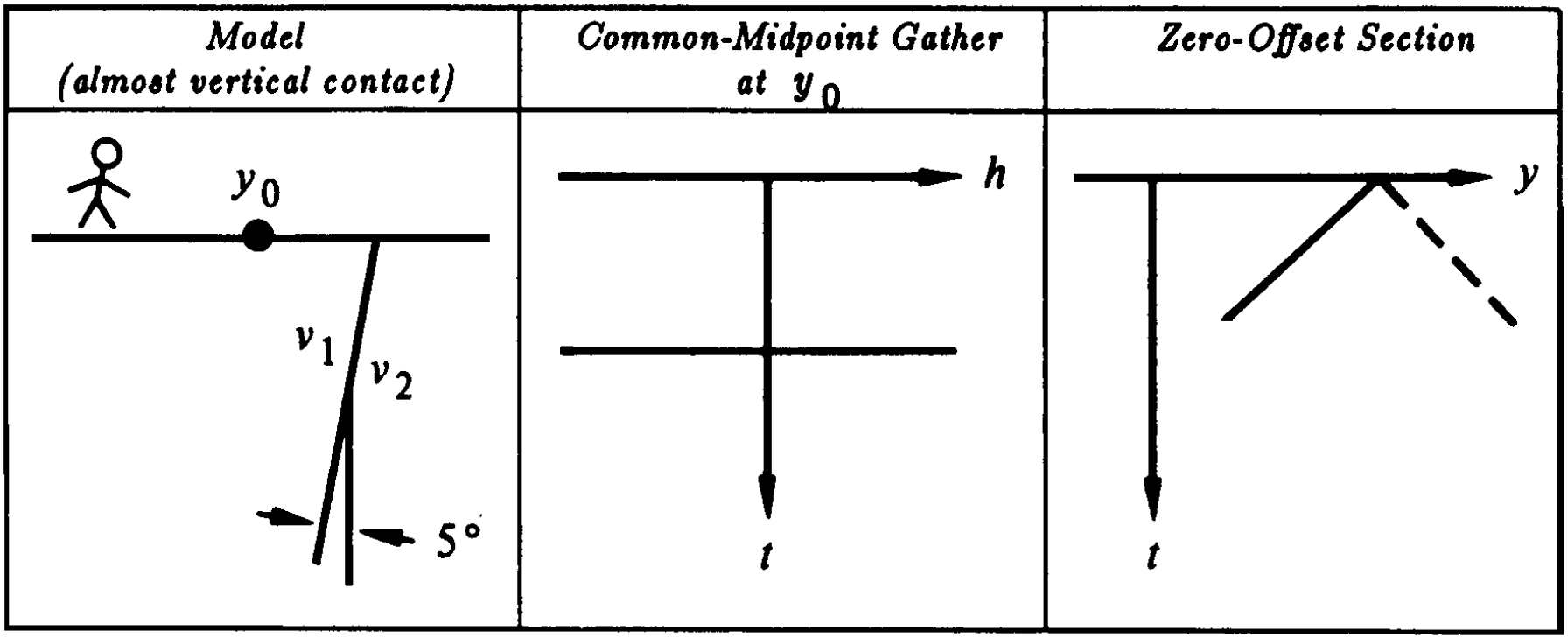


Figure 8.7: Near-vertical reflector, a gather, and a section. dpmv-vertlay [NR]

Figure 8.7 shows that the travel time does not change as the offset changes. It may seem paradoxical that the travel time does not increase as the shot and geophone get further apart. The key to the paradox is that midpoint is held constant, not shotpoint. As offset increases, the shot gets further from the reflector while the geophone gets closer. Time lost on one path is gained on the other.

A planar reflector may have any dip between horizontal and vertical. Then the common-midpoint gather lies between the common-midpoint gather of Figure 8.6 and that of Figure 8.7. The zero-offset section in Figure 8.7 is a straight line, which turns out to be the asymptote of a family of hyperbolas. The slope of the asymptote is the inverse of the velocity v_1.

It is interesting to notice that at small dips, information about the earth velocity is essentially carried on the offset axis whereas at large dips, the velocity information is essentially on the midpoint axis.

8.2.1. The response of two points

Another simple geometry is a reflecting point within the earth. A wave incident on the point from any direction reflects waves in all directions. This geometry is

particularly important because any model is a superposition of such point scatterers. Figure 8.8 shows an example.

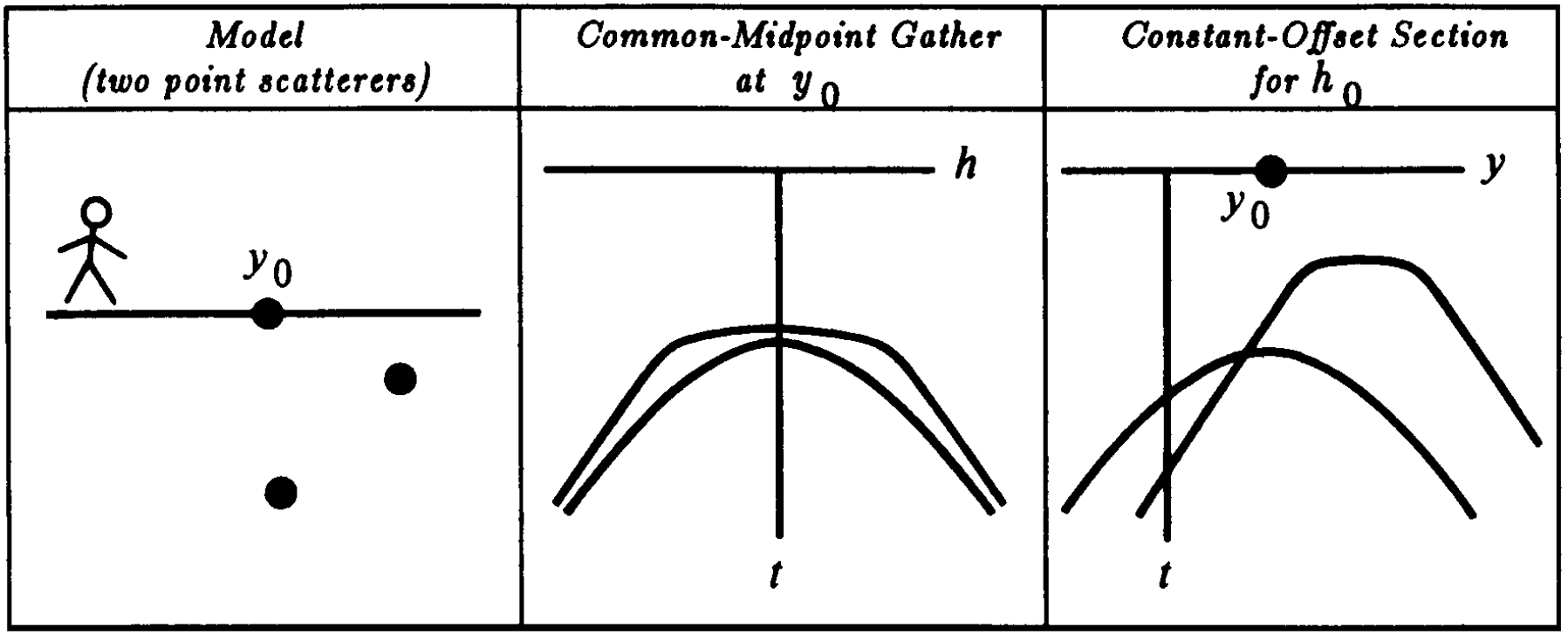


Figure 8.8: Response of two point scatterers. Note the flat spots. dpmv-twopoint [NR]

The curves in Figure 8.8 include flat spots for the same reasons that some of the curves in Figures 8.6 and 8.7 were straight lines.

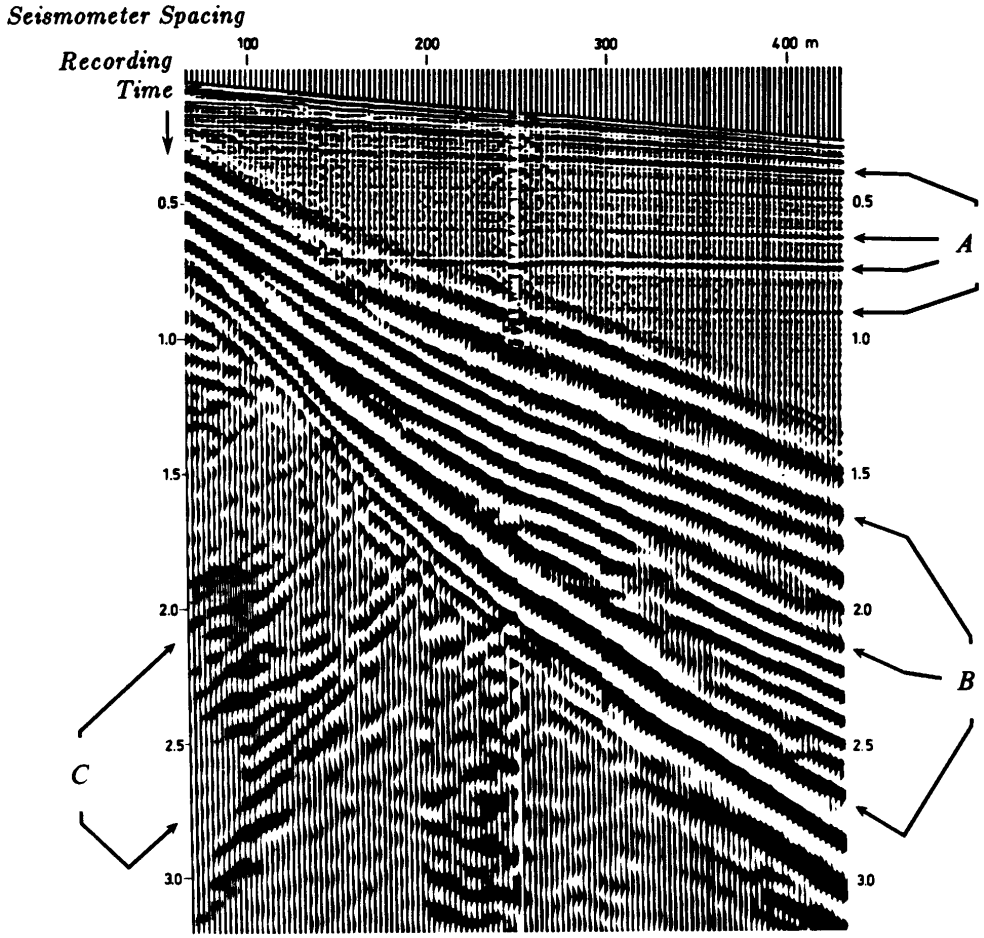


Figure 8.9: Undocumented data from a recruitment brochure. This data may be assumed to be of textbook quality. The speed of sound in water is about 1500 m/sec. Identify the events at A, B, and C. Is this a common-shotpoint gather or a common-midpoint gather? (Shell Oil Company) dpmv-shell [NR]

8.2.2. The dipping bed

While the traveltime curves resulting from a dipping bed are simple, they are not simple to derive. Before the derivation, the result will be stated: for a bed dipping at angle α from the horizontal, the traveltime curve is

$$t^2 v^2 \quad = \quad 4(y-y_0)^2 \sin^2\alpha \; + \; 4h^2 \cos^2\alpha \tag{8.12}$$

For $\alpha = 45°$, equation (8.12) is the familiar Pythagoras cone—it is just like $t^2 = z^2 + x^2$. For other values of α, the equation is still a cone, but a less familiar one because of the stretched axes.

For a common-midpoint gather at $y = y_1$ in (h,t)-space, equation (8.12) looks like $t^2 = t_0^2 + 4h^2/v^2_{\text{apparent}}$. Thus the common-midpoint gather contains an *exact* hyperbola, regardless of the earth dip angle α. The effect of dip is to change the asymptote of the hyperbola, thus changing the apparent velocity. The result has great significance in applied work and is known as Levin's dip correction [1971]:

$$v_{\text{apparent}} \quad = \quad \frac{v_{\text{earth}}}{\cos(\alpha)} \tag{8.13}$$

(See also Slotnick [1959]). In summary, dip increases the stacking velocity.

Figure 8.10 depicts some rays from a common-midpoint gather. Notice that

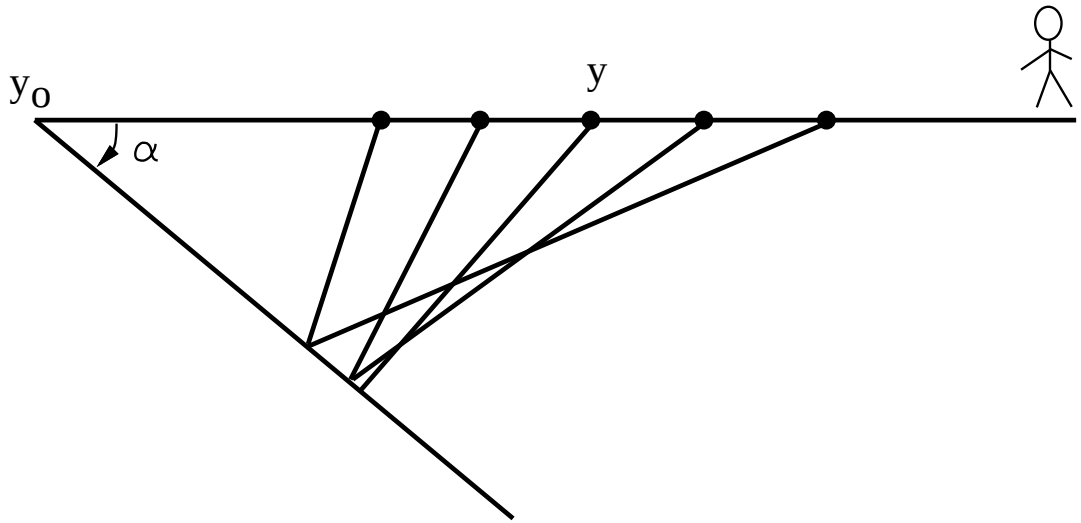


Figure 8.10: Rays from a common-midpoint gather. dpmv-dipray [NR]

each ray strikes the dipping bed at a different place. So a common-*midpoint* gather is not a common-*depth-point* gather. To realize why the reflection point moves on the reflector, recall the basic geometrical fact that an angle bisector in a triangle generally doesn't bisect the opposite side. The reflection point moves *up* dip with increasing offset.

Finally, equation (8.12) will be proved. Figure 8.11 shows the basic geometry along with an "image" source on another reflector of twice the dip.

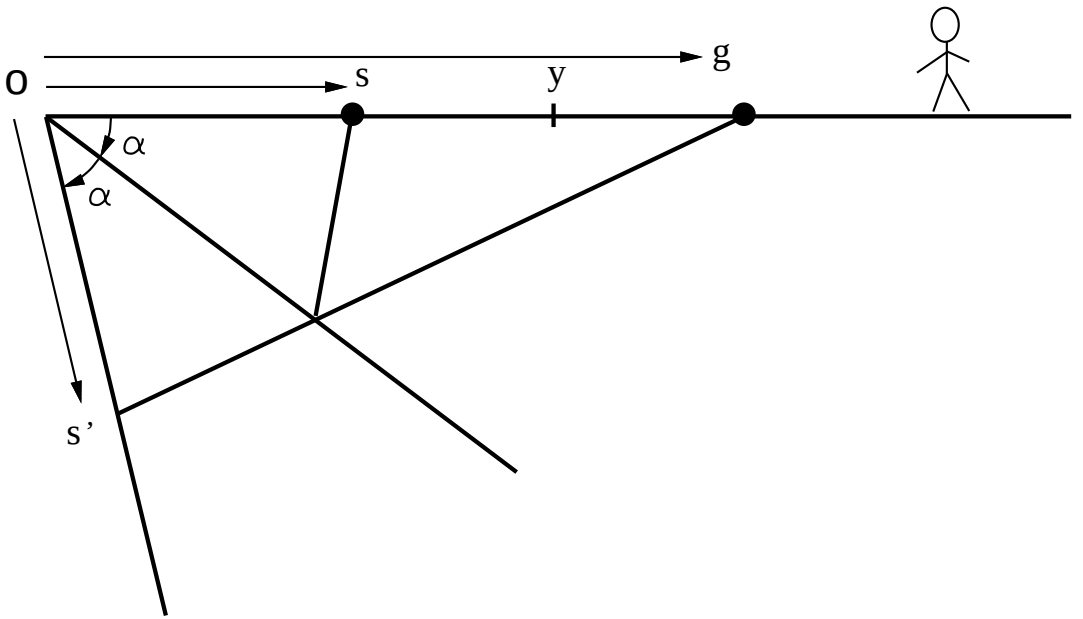


Figure 8.11: Travel time from image source at s' to g may be expressed by the law of cosines. dpmv-lawcos [NR]

For convenience, the bed intercepts the surface at $y_0 = 0$. The length of the line $s'g$ in Figure 8.11 is determined by the trigonometric Law of Cosines to be

$$\begin{aligned} t^2 v^2 &= s^2 + g^2 - 2sg\cos 2\alpha \\ t^2 v^2 &= (y-h)^2 + (y+h)^2 - 2(y-h)(y+h)\cos 2\alpha \\ t^2 v^2 &= 2(y^2+h^2) - 2(y^2-h^2)(\cos^2\alpha - \sin^2\alpha) \\ t^2 v^2 &= 4y^2\sin^2\alpha + 4h^2\cos^2\alpha \end{aligned}$$

which is equation (8.12).

Another facet of equation (8.12) is that it describes the constant-offset section. Surprisingly, the travel time of a dipping planar bed becomes curved at nonzero offset—it too becomes hyperbolic.

8.2.3. Randomly dipping layers

On a horizontally layered earth, a common shotpoint gather looks like a common midpoint gather. For an earth model of random dipping planes the two kinds of gathers have quite different traveltime curves as we see in Figure 8.12.

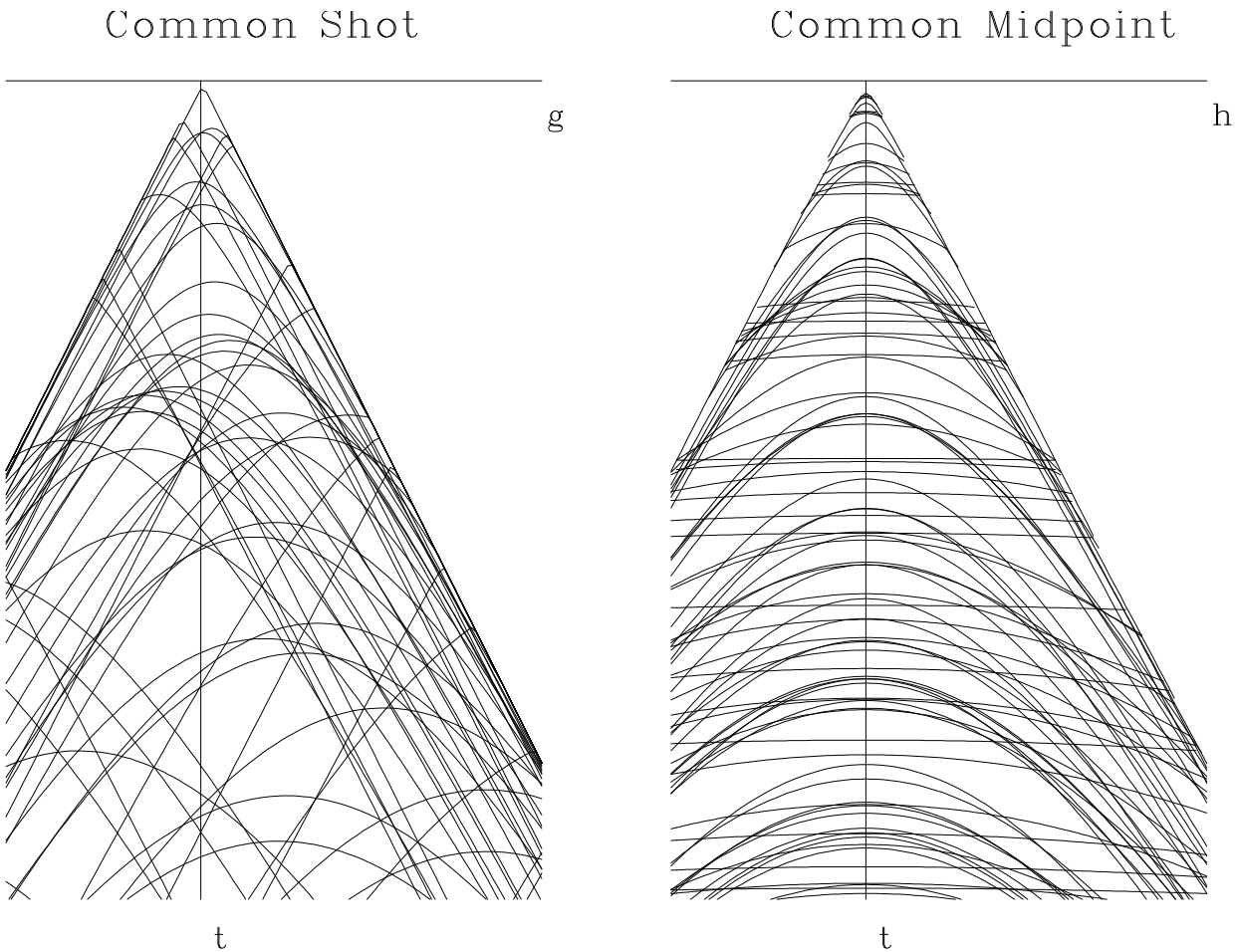


Figure 8.12: Seismic arrival times on an earth of random dipping planes. Left is for CSP. Right is for CMP. dpmv-randip [ER]

The common-shot gather is more easily understood. Although a reflector is dipping, a spherical wave incident remains a spherical wave after reflection. The center of the reflected wave sphere is called the image point. The traveltime equation is again a cone centered at the image point. The traveltime curves are simply hyperbolas topped above the image point having the usual asymptotic slope. The new feature introduced by dip is that the hyperbola is laterally shifted which implies arrivals before the fastest possible straight-line arrivals at $vt = |g|$. Such arrivals cannot happen. These hyperbolas must be truncated where $vt = |g|$. This discontinuity has the physical meaning of a dipping bed hitting the surface at geophone location $|g| = vt$. Beyond the truncation, either the shot or the receiver has gone beyond the intersection. Eventually both are beyond. When either is beyond the intersection, there are no reflections.

On the common-midpoint gather we see hyperbolas all topping at zero offset, but with asymptotic velocities higher (by the Levin cosine of dip) than the earth velocity. Hyperbolas truncate, now at $|h| = tv/2$, again where a dipping bed hits the surface at a geophone.

On a CMP gather, some hyperbolas may seem high velocity, but it is the dip, not the earth velocity itself that causes it. Imagine Figure 8.12 with all layers at

90° dip (abandon curves and keep straight lines). Such dip is like the backscattered groundroll seen on the common-shot gather of Figure 8.9. The backscattered groundroll becomes a “flat top” on the CMP gather in Figure 8.12.

Such strong horizontal events near zero offset will match any velocity, particularly higher velocities such as primaries. Unfortunately such noise events thus make a strong contribution to a CMP stack. Let us see how these flat-tops in offset create the diagonal streaks you see in midpoint in Figure 8.13.

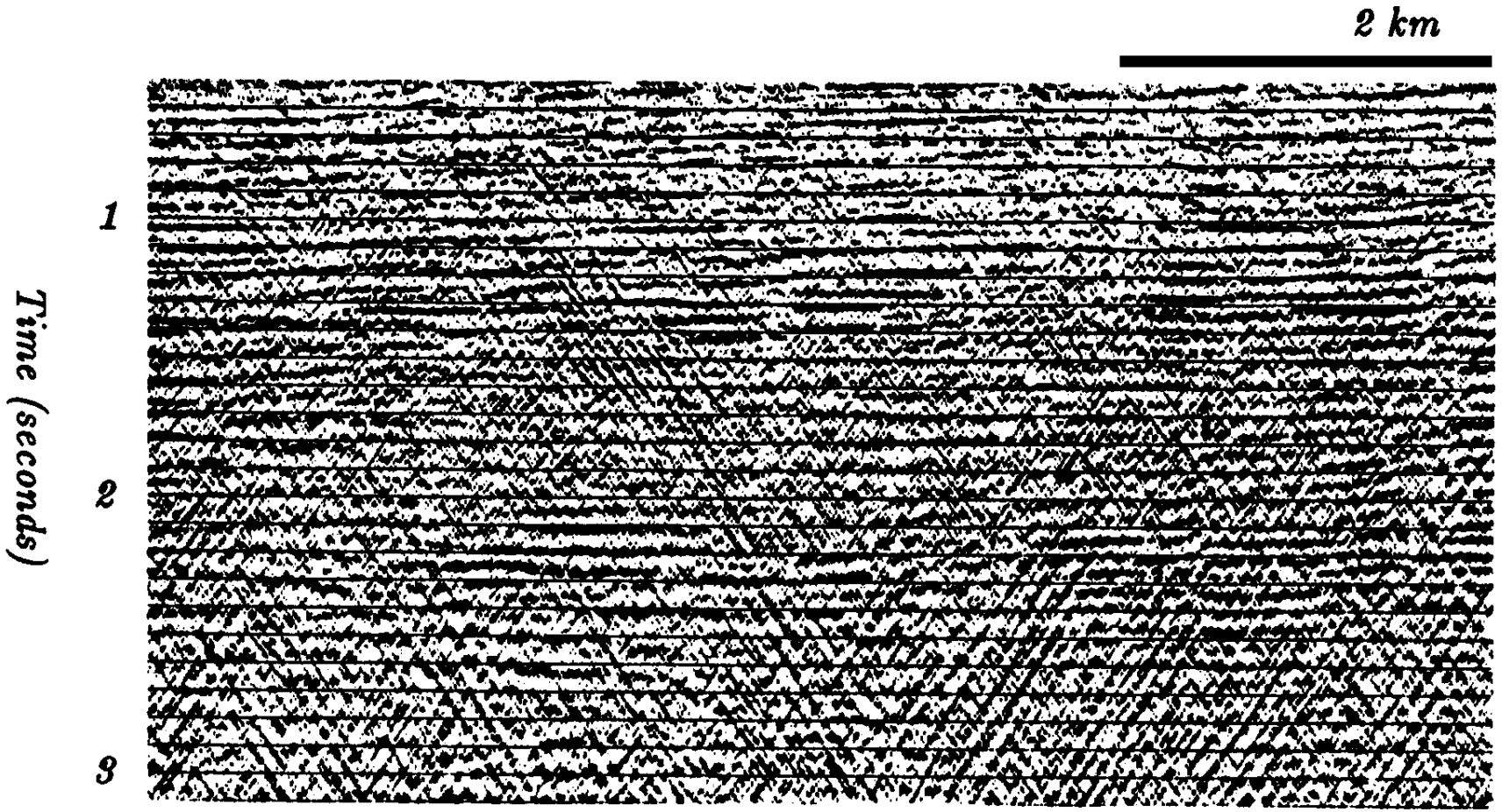


Figure 8.13: CDP stack with water noise from the Shelikof Strait, Alaska. (by permission from *Geophysics*, Larner et al.[1983]) dpmv-shelikof [NR]

Consider 360 rocks of random sizes scattered in an exact circle of 2 km diameter on the ocean floor. The rocks are distributed along one degree intervals. Our survey ship sails from south to north towing a streamer across the exact center of the circle, coincidentally crossing directly over rock number 180 and number 0. Let us consider the common midpoint gather corresponding to the midpoint in the center of the circle. Rocks 0 and 180 produce flat-top hyperbolas. The top is flat for $0 < |h| < 1$ km. Rocks 90 and 270 are 90° out of the plane of the survey. Rays to those rocks propagate entirely within the water layer. Since this is a homogeneous media, the travel time expression of these rocks is a simple hyperbola of water velocity. Now our CMP gather at the circle center has a "flat top" and a simple hyperbola both going through zero offset at time $t = 2/v$ (diameter 2 km, water velocity). Both curves have the same water velocity asymptote and of course the curves are tangent at zero offset.

Now consider all the other rocks. They give curves inbetween the simple water hyperbola and the flat top. Near zero offset, these curves range in apparent velocity between water velocity and infinity. One of these curves will have an apparent velocity that matches that of sediment velocity. This rock (and all those near the same azimuth) will have velocities that are near the sediment velocity. This noise

will stack very well. The CDP stack at sediment velocity will stack in a lot of water borne noise. This noise is propagating somewhat off the survey line but not very far off it.

Now let us think about the appearance of the CDP stack. We turn attention from offset to midpoint. The easiest way to imagine the CDP stack is to imagine the zero-offset section. Every rock has a water velocity asymptote. These asymptotes are evident on the CDP stack in Figure 8.13. This result was first recognized by Ken Larner.

Thus, backscattered low-velocity noises have a way of showing up on higher-velocity stacked data. There are two approaches to suppressing this noise: (1) mute the inner traces, and as we will see, (2) dip moveout processing.

8.3. TROUBLE WITH DIPPING REFLECTORS

The "standard process" is NMO, stack, and zero-offset migration. Its major shortcoming is the failure of NMO and stack to produce a section that resembles the true zero-offset section. In chapter 4 we derived the NMO equations for a stratified earth, but then applied them to seismic field data that was not really stratified. That

this works at all is a little surprising, but it turns out that NMO hyperbolas apply to dipping reflectors as well as horizontal ones. When people try to put this result into practice, however, they run into a nasty conflict: reflectors generally require a *dip-dependent* NMO velocity in order to produce a "good" stack. Which NMO velocity are we to apply when a dipping event is near (or even crosses) a horizontal event? Using conventional NMO/stack techniques generally forces velocity analysts to choose which events they wish to preserve on the stack. This inability to simultaneously produce a good stack for events with *all* dips is a serious shortcoming, which we now wish to understand more quantitatively.

8.3.1. Gulf of Mexico example

Recall the Gulf of Mexico dataset presented in chapter 4. We did a reasonably careful job of NMO velocity analysis in order to produce the stack shown in Figure 4.7. But is this the best possible stack? To begin to answer this question, Figure 8.14 shows some constant-velocity stacks of this dataset done with subroutine `velsimp()` /prog:velsimp. This figure clearly shows that there are some very steeply-dipping reflections that are missing in Figure 4.7. These steep reflections appear only when

the NMO velocity is quite high compared with the velocity that does a good job on the horizontal reflectors. This phenomenon is consistent with the predictions of equation (8.12), which says that dipping events require a *higher* NMO velocity than nearby horizontal events.

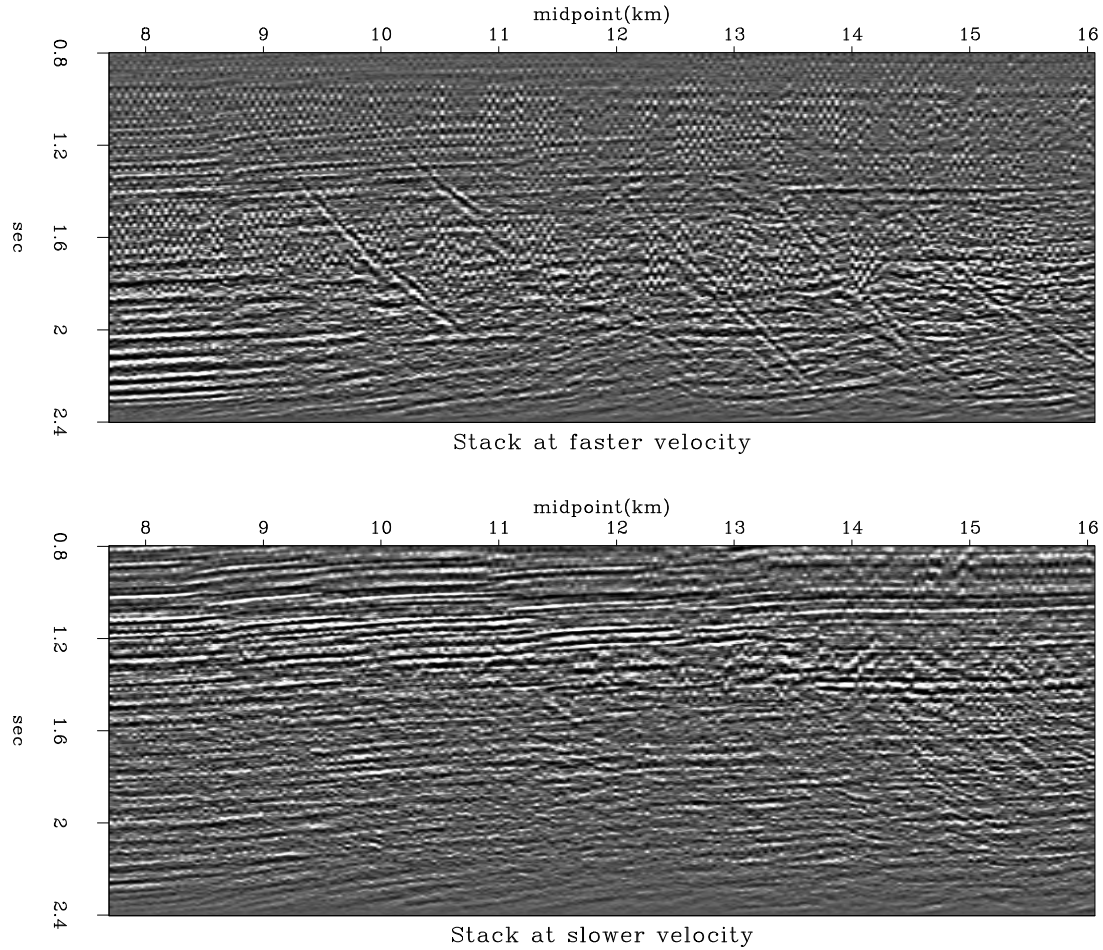


Figure 8.14: Stacks of Gulf of Mexico data with two different constant NMO velocities. Press button to see a **movie** in which each frame is a stack with a different constant velocity. dpmv-cvstacks [ER,M]

Another way of seeing the same conflict in the data is to look at a velocity-analysis panel at a single common-midpoint location such as the panel shown in Figure 8.15 made by subroutine `velsimp()` /prog:velsimp. In this figure it is easy to see that the velocity which is good for the dipping event at 1.5 sec is too high for the horizontal events in its vicinity.

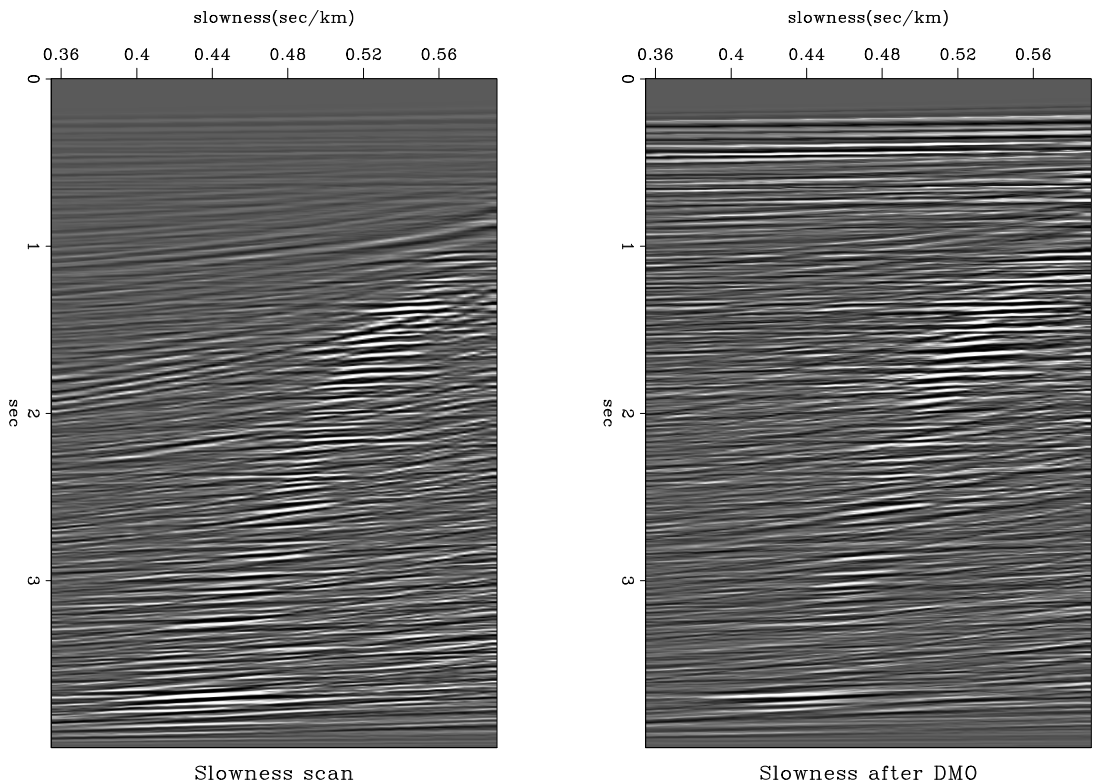


Figure 8.15: Velocity analysis panel of one of the panels in Figure 8.14 before (left) and after (right) DMO. Notice two velocities at the same time before DMO. dpmv-velscan [ER,M]

8.4. SHERWOOD'S DEVILISH

The migration process should be thought of as being interwoven with the velocity estimation process. J.W.C. **Sherwood** [1976] indicated how the two processes, migration and velocity estimation, should be interwoven. The moveout correction should be considered in two parts, one depending on offset, the NMO, and the other depending on dip. This latter process was conceptually new. Sherwood described the process as a kind of filtering, but he did not provide implementation details. He called his process *Devilish*, an acronym for "dipping-event velocity inequalities licked." The process was later described more functionally by **Yilmaz** as ***prestack partial migration***, and now the process is usually called *dip moveout* (**DMO**) although some call it MZO, migration to zero offset. We will first see Sherwood's results, then Rocca's conceptual model of the **DMO** process, and finally two conceptually distinct, quantitative specifications of the process.

Figure 8.16 contains a panel from a stacked section.

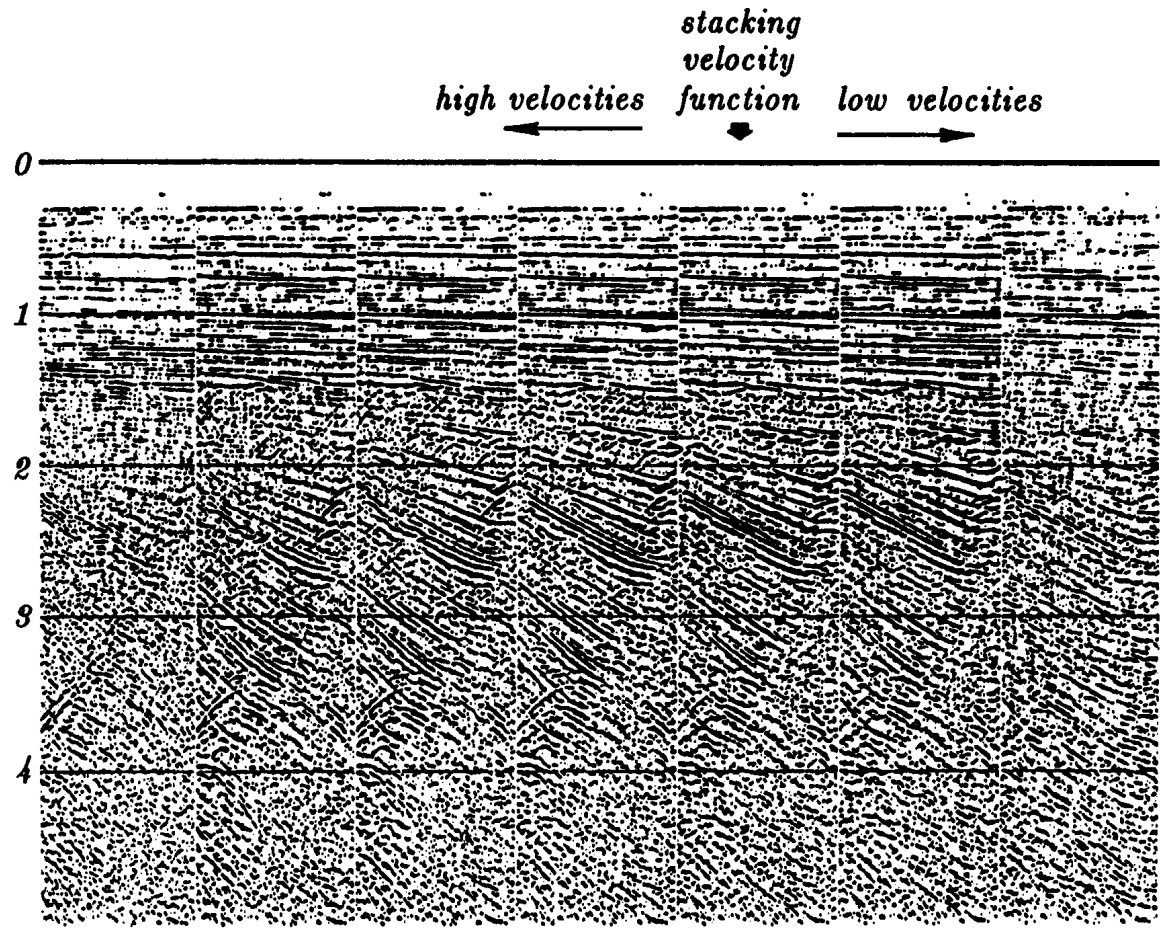


Figure 8.16: Conventional stacks with varying velocity. (distributed by Digicon, Inc.) dpmv-digicon [NR]

The panel is shown several times; each time the stacking velocity is different. It should be noted that at the low velocities, the horizontal events dominate, whereas at the high velocities, the steeply dipping events dominate. After the *Devilish* correction was applied, the data was restacked as before. Figure 8.17 shows that the stacking velocity no longer depends on the dip.

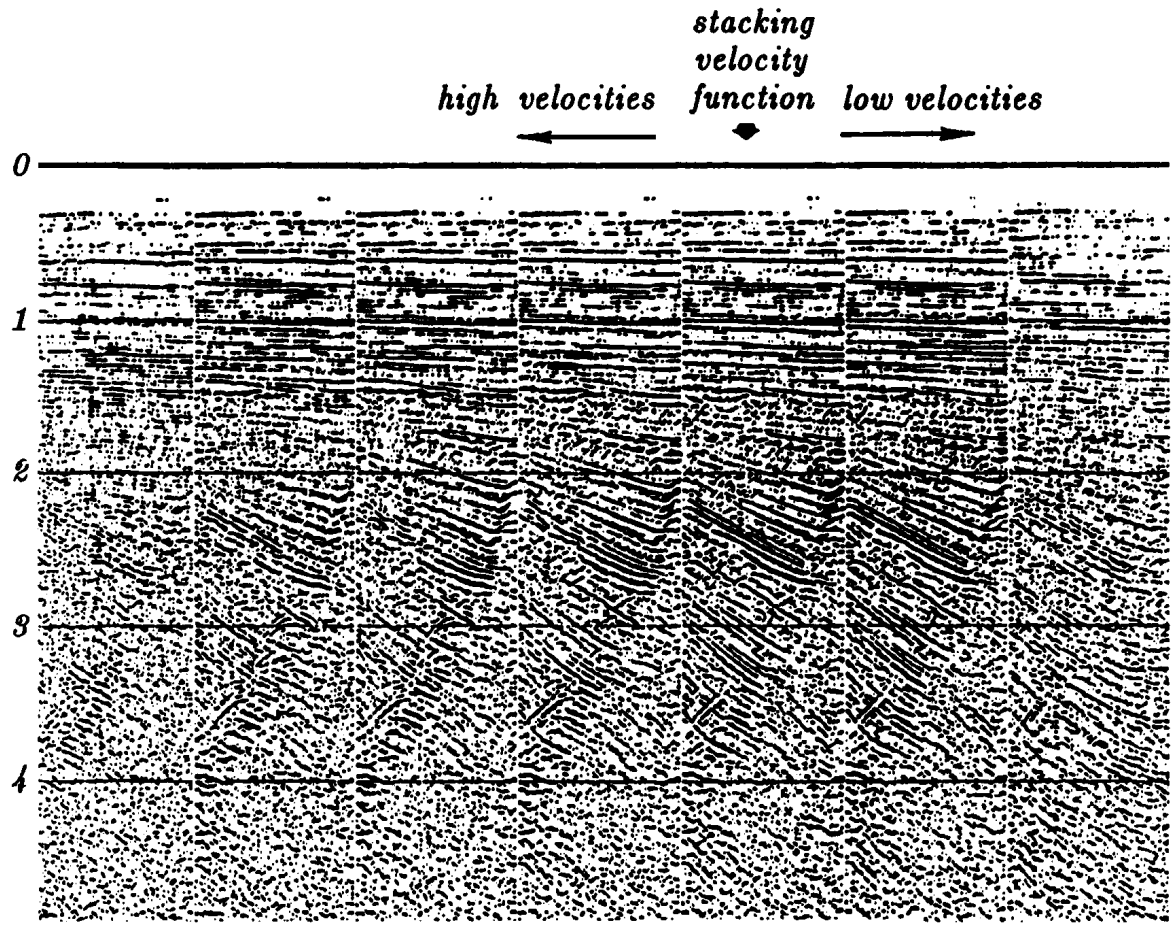


Figure 8.17: *Devilish* stacks with varying velocity. (distributed by Digicon, Inc.) dpmv-devlish [NR]

This means that after *Devilish*, the velocity may be determined without regard to dip. In other words, events with all dips contribute to the same consistent velocity rather than each dipping event predicting a different velocity. So the *Devilish* process should provide better velocities for data with conflicting dips. And we can expect a better final stack as well.

8.5. ROCCA'S SMEAR OPERATOR

Fabio **Rocca** developed a clear conceptual model for Sherwood's dip corrections. Start with an impulse on a common offset section, and migrate it getting ellipses like in Figure 8.4. We did this with subroutine `flathyp()` /prog:flathyp using some constant-offset `h`. Although the result is an ellipsoidal curve, think of it as a row of many points along an ellipsoidal curve. Then diffract the image thus turning each of the many points into a hyperbola. We do this with the return path of the same subroutine `flathyp()`, however the path back is taken with `h=0`. The result is shown in Figure 8.18. To enhance the appearance of the figure, I injected an intermediate step of converting the ellipsoid curve into a trajectory of dots on the ellipse. Notice that the hyperbola tops are not on the strong smear function that results from the

superposition.

The strong smear function that you see in Figure 8.18 is Rocca's **DMO**+NMO operator, the operator that converts a point on a constant-offset section to a zero-offset section. The important feature of this operator is that the bulk of the energy is in a much narrower region than the big ellipse of migration. The narrowness of the Rocca operator is important since it means that energies will not move far, so the operator will not have a drastic effect and be unduly affected by remote data. (Being a small operator also makes it cheaper to apply). The little signals you see away from the central burst in Figure 8.18 result mainly from my modulating the ellipse curve into a sequence of dots. However, noises from sampling and nearest-neighbor interpolation also yield a figure much like Figure 8.18. This warrants a more careful theoretical study to see how to represent the Rocca operator directly (rather than as a sequence of two nearly opposite operators).

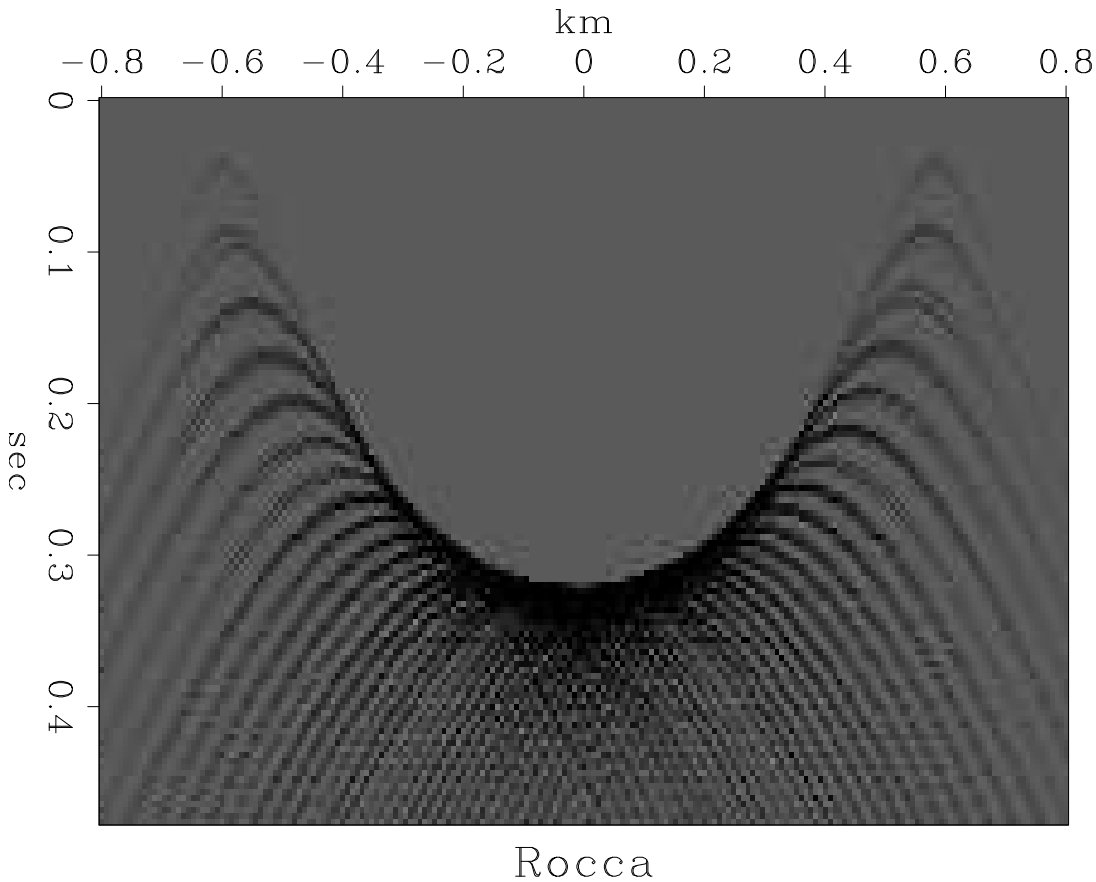


Figure 8.18: Rocca's prestack partial-migration operator is a superposition of hyperbolas, each with its top on an ellipse. dpmv-frocca [ER]

To get a sharper, more theoretical view of the Rocca operator, Figure 8.19 shows line drawings of the curves in a Rocca construction. It happens, and we will later show, that the Rocca operator lies along an ellipse that passes through $\pm h$ (and hence is independent of velocity!) Curiously, we see something we could not see on Figure 8.18, that the Rocca curve ends part way up the ellipse and it does not reach the surface. The place where the Rocca operator ends and the velocity independent ellipse continues is, however, velocity dependent as we will see. The Rocca operator is along the curve of osculation in Figure 8.19, i.e., the smile-shaped curve where the hyperbolas reinforce one another.

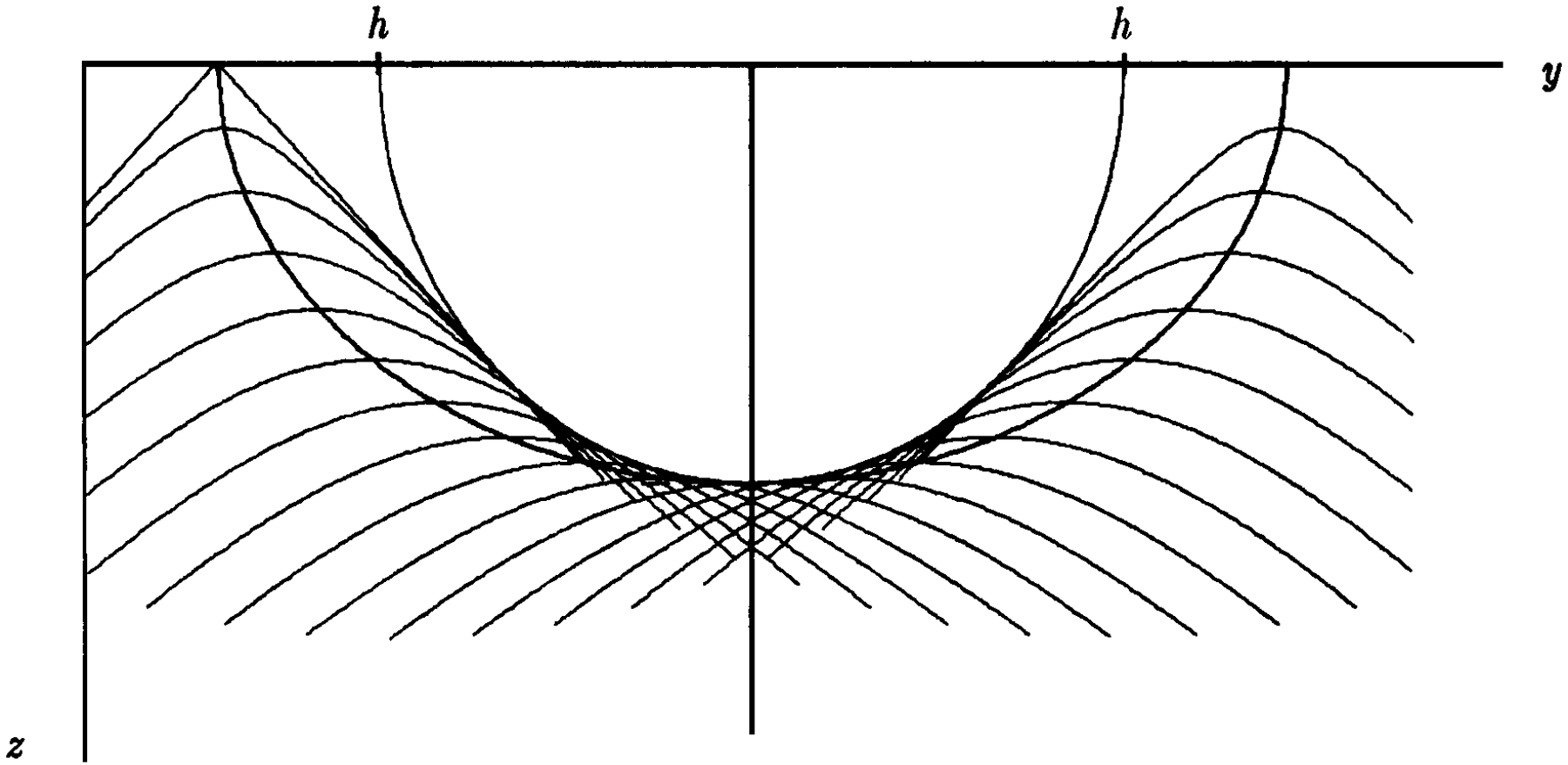


Figure 8.19: Rocca's smile. (Ronen) dpmv-rocca2 [NR]

8.5.1. Push and pull

Migration and diffraction operators can be conceived and programmed in two different ways. Let $\vec{\mathbf{t}}$ denote data and $\vec{\mathbf{z}}$ denote the depth image. We have

$$\vec{\mathbf{z}} = \mathbf{C}_h \, \vec{\mathbf{t}} \qquad \text{spray or push an ellipse into the output} \tag{8.14}$$

$$\vec{\mathbf{t}} = \mathbf{H}_h \, \vec{\mathbf{z}} \qquad \text{spray or push a flattened hyperbola into the output} \tag{8.15}$$

where h is half the shot-geophone offset. The adjoints are

$$\vec{\mathbf{t}} = \mathbf{C}_h' \, \vec{\mathbf{z}} \qquad \text{sum or pull a semiCircle from the input} \tag{8.16}$$

$$\vec{\mathbf{z}} = \mathbf{H}_h' \, \vec{\mathbf{t}} \qquad \text{sum or pull a flattened Hyperbola from the input} \tag{8.17}$$

In practice we can choose either of $\mathbf{C} \approx \mathbf{H}'$. A natural question is which is more correct or better. The question of "more correct" applies to modeling and is best answered by theoreticians (who will find more than simply a hyperbola; they will find its waveform including its amplitude and phase as a function of frequency). The question of "better" is something else. An important practical issue is that the transformation should not leave miscellaneous holes in the output. It is typically desirable to write programs that loop over all positions in the *output* space, "pulling" in whatever inputs are required. It is usually less desirable to loop over all positions

in the *input* space, "pushing" or "spraying" each input value to the appropriate location in the output space. Programs that push the input data to the output space might leave the output too sparsely distributed. Also, because of gridding, the output data might be irregularly positioned. Thus, to produce smooth outputs, we usually *prefer the summation operators* $\mathbf{H}'$ for migration and $\mathbf{C}'$ for diffraction modeling. Since one could always force smooth outputs by lowpass filtering, what we really seek is the highest possible resolution.

Given a nonzero-offset section, we seek to convert it to a zero-offset section. Rocca's concept is to first migrate the constant offset data with an ellipsoid push operator $\mathbf{C}_h$ and then take each point on the ellipsoid and diffract it out to a zero-offset hyperbola with a push operator $\mathbf{H}_0$. The product of push operators $\mathbf{R} = \mathbf{H}_0\mathbf{C}_h$ is known as Rocca's smile. This smile operator includes both normal moveout and dip moveout. (We could say that dip moveout is defined by Rocca's smile after restoring the normal moveout.)

Because of the approximation $\mathbf{H} \approx \mathbf{C}'$, we have four different ways to express the Rocca smile:

$$\mathbf{R} \quad = \quad \mathbf{H}_0\mathbf{C}_h \quad \approx \quad \mathbf{H}_0\mathbf{H}'_h \quad \approx \quad \mathbf{C}'_0\mathbf{H}'_h \quad \approx \quad \mathbf{C}'_0\mathbf{C}_h \tag{8.18}$$

$\mathbf{H}_0\mathbf{H}_h'$ says sum over a flat-top and then spray a regular hyperbola.

The operator $\mathbf{C}_0'\mathbf{H}_h'$, having two pull operators should have smoothest output. Sergey Fomel suggests an interesting illustration of it: Its adjoint is two push operators, $\mathbf{R}' = \mathbf{H}_h\mathbf{C}_0$. $\mathbf{R}'$ takes us from zero offset to nonzero offset first by pushing a data point to a semicircle and then by pushing points on the semicircle to flat-topped hyperbolas. As before, to make the hyperbolas more distinct, I broke the circle into dots along the circle and show the result in Figure 8.20. The whole truth is a little more complicated. Subroutine `flathyp()` /prog:flathyp implements $\mathbf{H}$ and $\mathbf{H}'$. Since I had no subroutine for $\mathbf{C}$, figures 8.18 and 8.20 were actually made with only $\mathbf{H}$ and $\mathbf{H}'$.

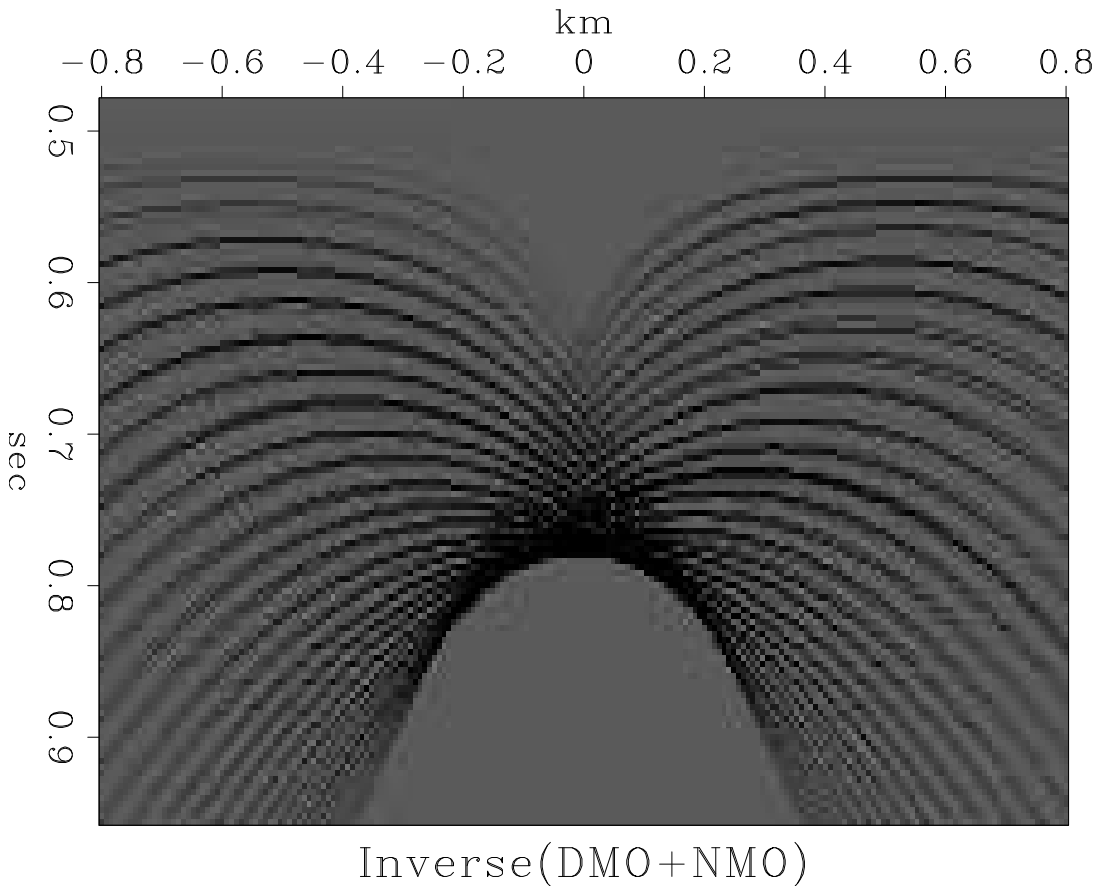


Figure 8.20: The adjoint of Rocca's smile is a superposition of flattened hyperbolas, each with its top on a circle. dpmv-sergey [ER]

We discuss the $\mathbf{C}_0'\mathbf{C}_h$ representation of $\mathbf{R}$ in the next section.

8.5.2. Dip moveout with $v(z)$

It is worth noticing that the concepts in this section are not limited to constant velocity but apply as well to $v(z)$. However, the circle operator $\mathbf{C}$ presents some difficulties. Let us see why. Starting from the Dix moveout approximation, $t^2 = \tau^2 + x^2/v(\tau)^2$, we can directly solve for $t(\tau, x)$ but finding $\tau(t, x)$ is an iterative process at best. Even worse, at wide offsets, hyperbolas cross one another which means that $\tau(t, x)$ is multivalued. The spray (push) operators $\mathbf{C}$ and $\mathbf{H}$ loop over inputs and compute the location of their outputs. Thus $\vec{\mathbf{z}} = \mathbf{C}_h\ \vec{\mathbf{t}}$ requires we compute τ from t so it is one of the troublesome cases. Likewise, the sum (pull) operators $\mathbf{C}'$ and $\mathbf{H}'$ loop over outputs. Thus $\vec{\mathbf{t}} = \mathbf{C}'_h\ \vec{\mathbf{z}}$ causes us the same trouble. In both cases, the circle operator turns out to be the troublesome one. As a consequence, most practical work is done with the hyperbola operator.

A summary of the meaning of the Rocca smile and its adjoint is found in Figures 8.21 and 8.22, which were computed using subroutine `flathyp()` /prog:flathyp.

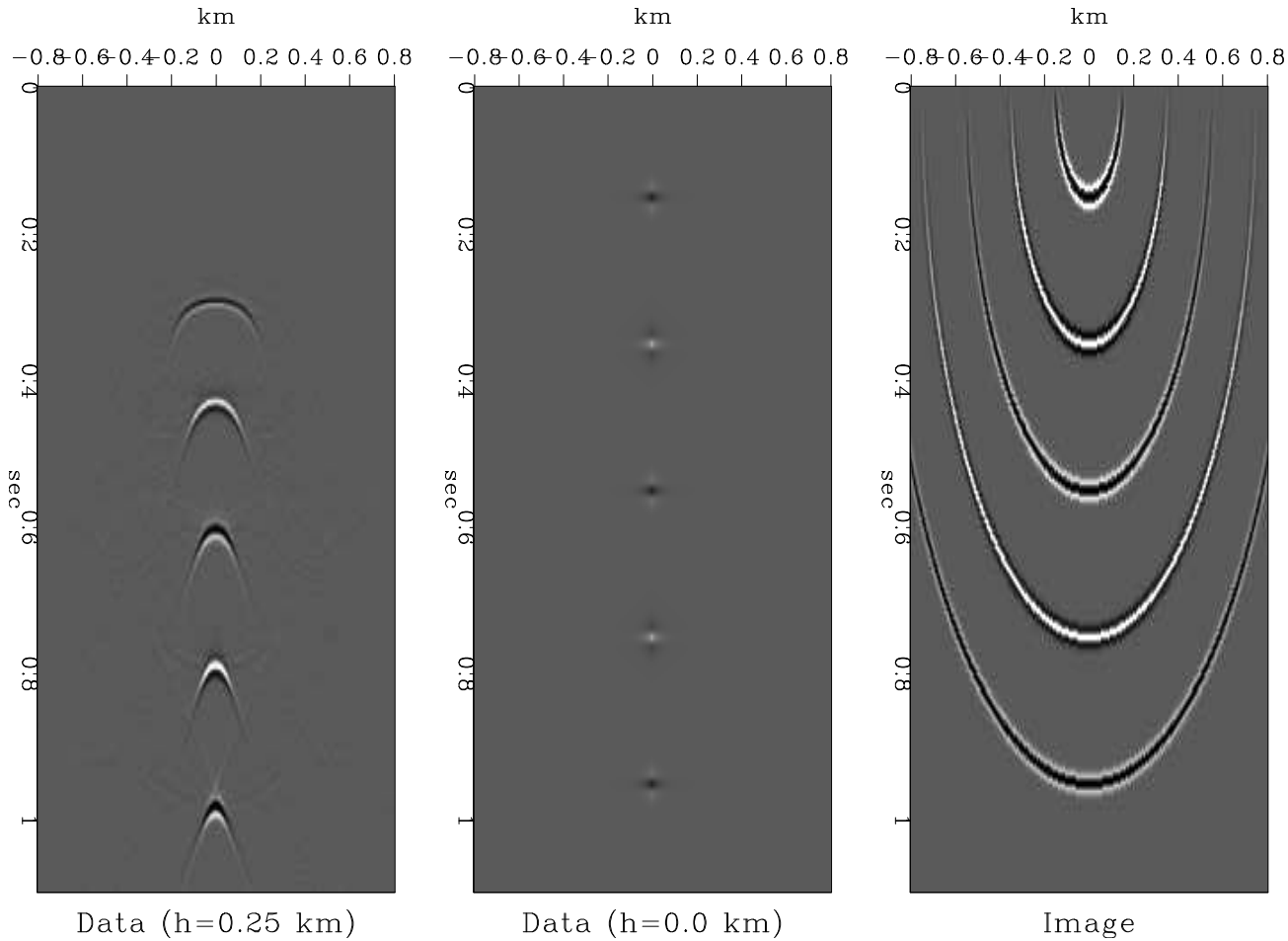


Figure 8.21: Impulses on a zero-offset section migrate to semicircles. The corresponding constant-offset section contains the adjoint of the Rocca smile. dpmv-yalei2 [ER]

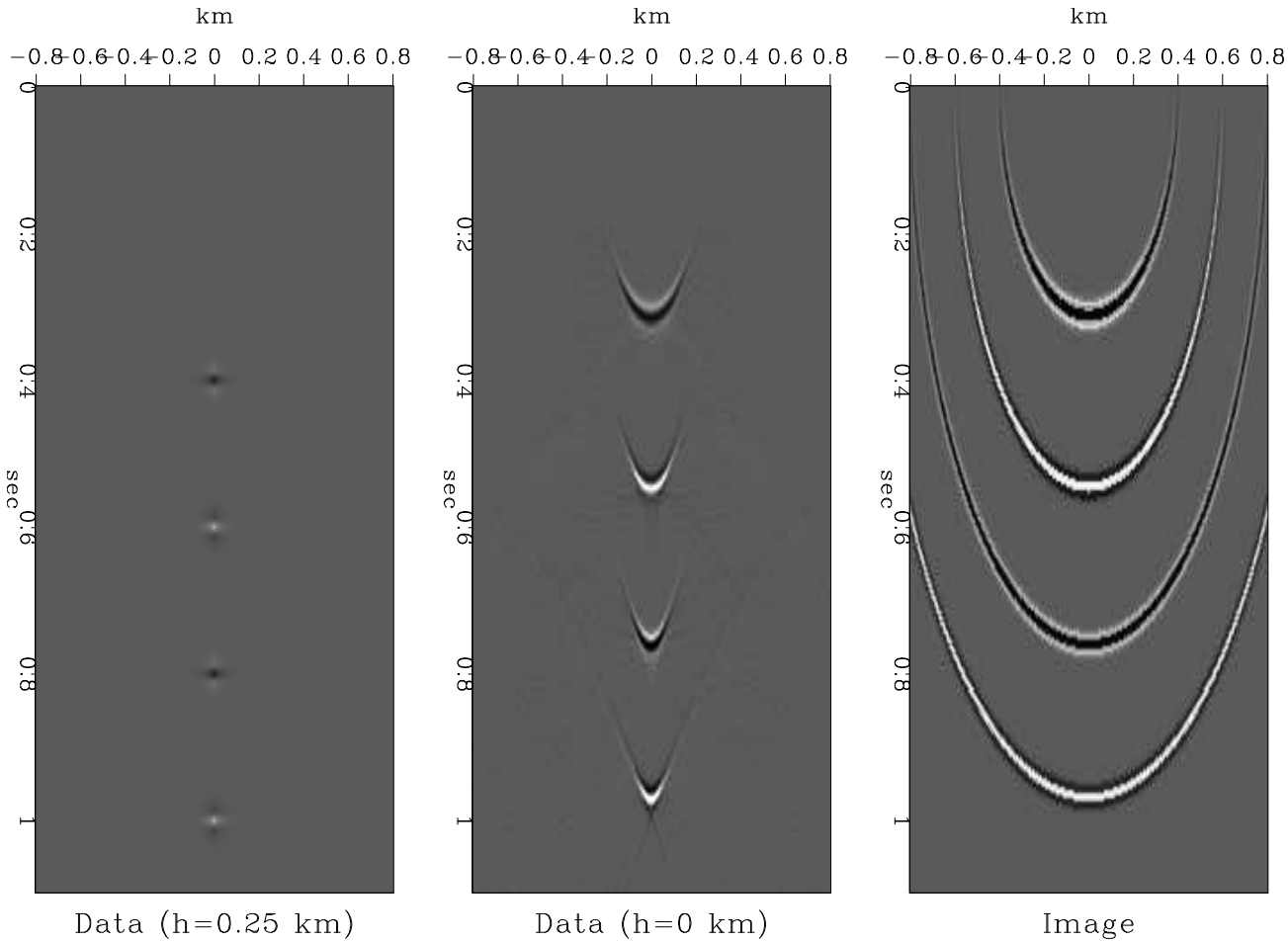


Figure 8.22: Impulses on a constant-offset section become ellipses in depth and Rocca smiles on the zero-offset section. dpmv-yalei1 [ER]

8.6. GARDNER'S SMEAR OPERATOR

A task, even in constant velocity media, is to find analytic expressions for the travel time in the Rocca operator. This we do now.

The Rocca operator $\mathbf{R} = \mathbf{C}_0' \mathbf{C}_h$ says to spray out an ellipse and then sum over a circle. This approach, associated with Gerry **Gardner**, says that we are interested in all circles that are inside and tangent to an ellipse, since only the ones that are tangent will have a constructive interference.

The Gardner formulation answers this question: Given a single nonzero offset impulse, which events on the zero-offset section will result in the same migrated subsurface picture? Since we know the migration response of a zero and nonzero offset impulses (circle and ellipse) we can rephrase this question: Given an ellipse corresponding to a nonzero offset impulse, what are the circles tangent to it that have their centers at the earth's surface? These circles if superposed will yield the ellipse. Furthermore, each of these circles corresponds to an impulse on the zero-offset section. The set of these impulses in the zero offset section is the **DMO**+NMO impulse response for a given nonzero offset event.

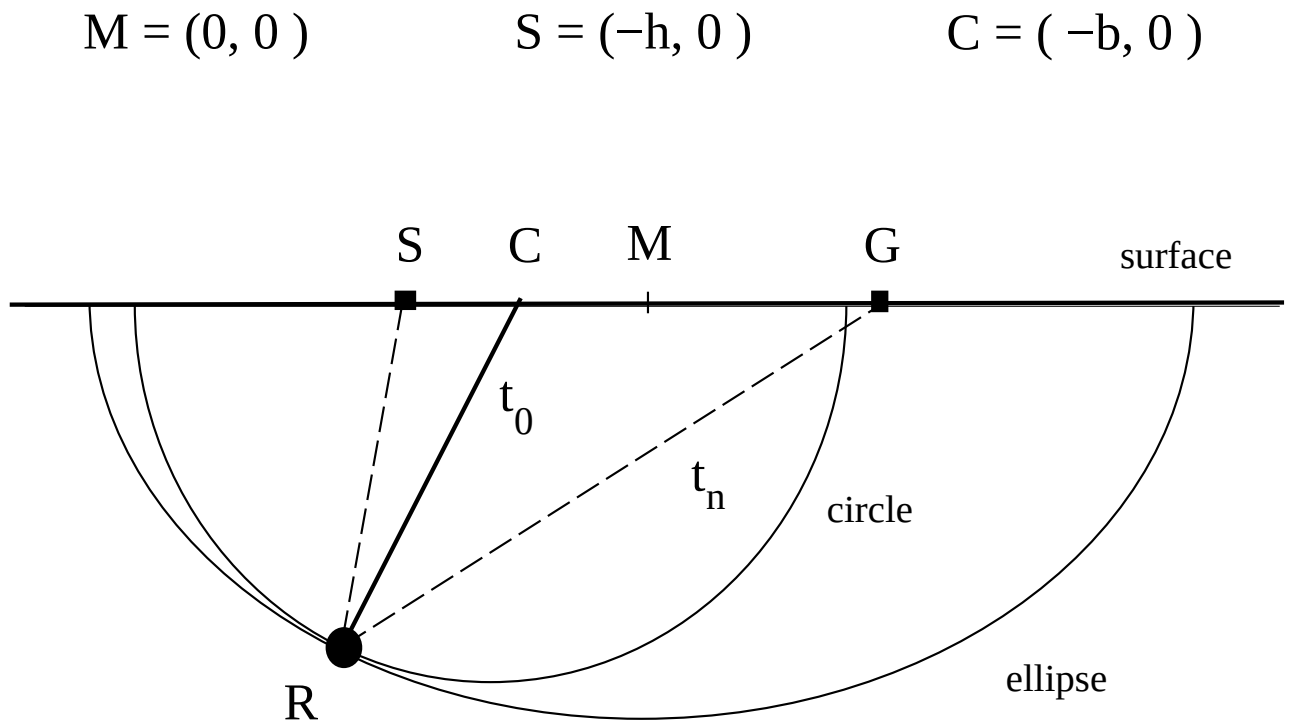


Figure 8.23: The nonzero offset migration impulse response is an ellipse. This ellipse can be mapped as a superposition of tangential circles with centers along the survey line. These circles correspond to zero offset migration impulse responses. dpmv-ell [NR]

8.6.1. Restatement of ellipse equations

Recall equation (8.9) for an ellipse centered at the origin.

$$0 \quad = \quad \frac{y^2}{A^2} + \frac{z^2}{B^2} - 1. \tag{8.19}$$

where

$$A \quad = \quad v_{\text{half}}\, t_h, \tag{8.20}$$

$$B^2 \quad = \quad A^2 - h^2. \tag{8.21}$$

The ray goes from the shot at one focus of the ellipse to anywhere on the ellipse, and then to the receiver in traveltime t_h. The equation for a circle of radius $R = t_0\, v_{\text{half}}$ with center on the surface at the source-receiver pair coordinate $x = b$ is

$$R^2 \quad = \quad (y-b)^2 + z^2, \tag{8.22}$$

where

$$R \quad = \quad t_0\, v_{\text{half}}. \tag{8.23}$$

To get the circle and ellipse tangent to each other, their slopes must match. Implicit differentiation of equation (8.19) and (8.22) with respect to y yields:

$$0 = \frac{y}{A^2} + \frac{z}{B^2}\frac{dz}{dy} \tag{8.24}$$

$$0 = (y-b) + z\frac{dz}{dy} \tag{8.25}$$

Eliminating dz/dy from equations (8.24) and (8.25) yields:

$$y = \frac{b}{1-\frac{B^2}{A^2}}. \tag{8.26}$$

At the point of tangency the circle and the ellipse should coincide. Thus we need to combine equations to eliminate x and z. We eliminate z from equation (8.19) and (8.22) to get an equation only dependent on the y variable. This y variable can be eliminated by inserting equation (8.26).

$$R^2 = B^2\left(\frac{A^2-B^2-b^2}{A^2-B^2}\right). \tag{8.27}$$

Substituting the definitions (8.20), (8.21), (8.23) of various parameter gives the relation between zero-offset traveltime t_0 and nonzero traveltime t_h:

$$t_0^2 \quad = \quad \left(t_h^2 - \frac{h^2}{v_{\text{half}}^2} \right) \left(1 - \frac{b^2}{h^2} \right). \tag{8.28}$$

As with the Rocca operator, equation (8.28) includes both dip moveout **DMO** and NMO.

8.7. DMO IN THE PROCESSING FLOW

Instead of implementing equation (8.28) in one step we can split it into two steps. The first step converts raw data at time t_h to NMOed data at time t_n.

$$t_n^2 \quad = \quad t_h^2 - \frac{h^2}{v_{\text{half}}^2} \tag{8.29}$$

The second step is the **DMO** step which like Kirchhoff migration itself is a convolution over the x-axis (or b-axis) with

$$t_0^2 \quad = \quad t_n^2 \left(1 - \frac{b^2}{h^2} \right) \tag{8.30}$$

and it converts time t_n to time t_0. Substituting (8.29) into (8.30) leads back to (8.28). As equation (8.30) clearly states, the **DMO** step itself is essentially velocity independent, but the NMO step naturally is not. Now the program. Backsolving equation (8.30) for t_n gives

$$t_n^2 \quad = \quad \frac{t_0^2}{1 - b^2/h^2}. \tag{8.31}$$

Like subroutine `flathyp()` /prog:flathyp, our **DMO** subroutine `dmokirch()` /prog:dmo is based on subroutine `kirchfast()` /prog:kirchfast. It is just the same, except where `kirchfast()` has a hyperbola we put equation (8.31). In the program, the variable t_0 is called `z` and the variable t_n is called `t`. Note, that the velocity `velhalf` does exclusively occur in the break condition (which we have failed to derive, but which is where the circle and ellipse touch at $z = 0$). dmokirch Figures 8.24 and

```
subroutine dmokirch( adj, add, velhalf, h, t0,dt,dx, modl,nt,nx, data)
integer ix,iz,it,ib, adj, add,                          nt,nx
real    amp,t,z,b,              velhalf, h, t0,dt,dx, modl(nt,nx),data(nt,nx)
call adjnull(         adj, add,                       modl,nt*nx, data,nt*nx)
if( h == 0)     call erexit('h=0')
do ib= -nx, nx {        b = dx * ib             # b = midpt separation
    do iz= 2, nt {      z = t0 + dt * (iz-1)    # z = zero-offset time

        if( h**2 <= b**2 )                  next
        t= sqrt(  z**2 / (1-b**2/h**2) )
        amp= sqrt(t) * dx/h
        if( velhalf*abs(b) * t*t > h**2*z)  break
        it = 1.5 + (t - t0) / dt
        if( it > nt )                       break
        do ix= max0(1, 1-ib),  min0(nx, nx-ib)
                if( adj == 0 )
                        data(it,ix+ib) = data(it,ix+ib) + modl(iz,ix   ) * amp
                else
                        modl(iz,ix   ) = modl(iz,ix   ) + data(it,ix+ib) * amp
        }
   }
return; end
```

Back

8.25 were made with subroutine `dmokirch()` /prog:dmokirch. Notice the big noise reduction over Figure 8.18.

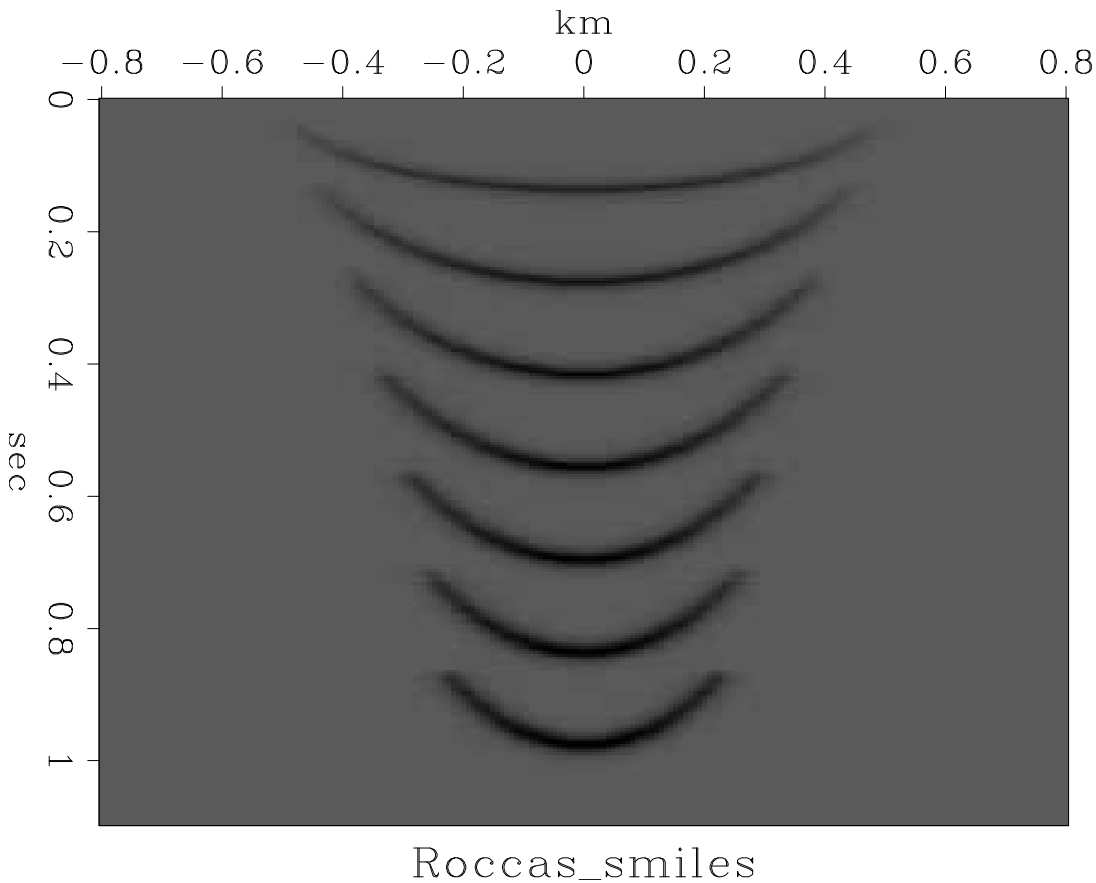


Figure 8.24: Impulse response of DMO and NMO dpmv-dmatt [ER]

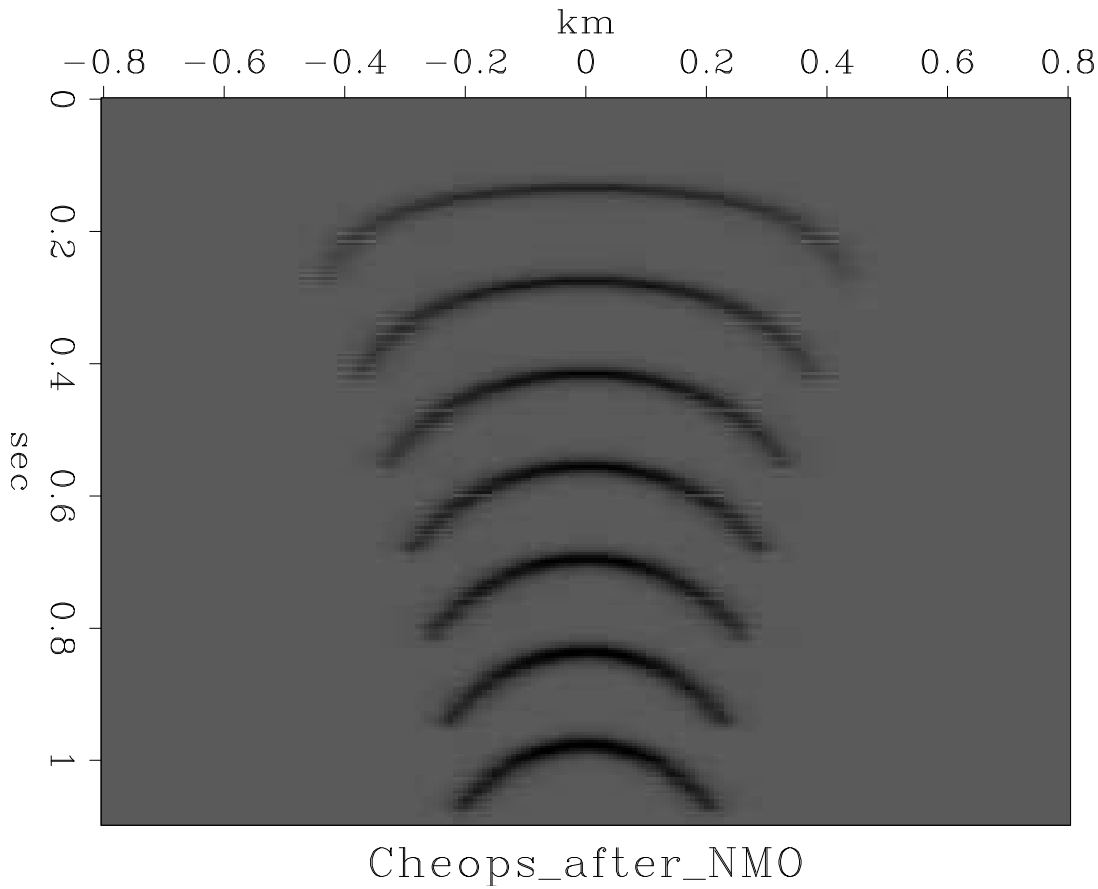


Figure 8.25: Synthetic Cheop's pyramid dpmv-coffs [ER]

8.7.1. Residual NMO

Unfortunately, the theory above shows that **DMO** should be performed *after* NMO. **DMO** is a convolutional operator, and significantly more costly than NMO. This is an annoyance because it would be much nicer if it could be done once and for all, and not need to be redone for each new NMO velocity.

Much practical work is done with using constant velocity for the DMO process. This is roughly valid since DMO, unlike NMO, does little to the data so the error of using the wrong velocity is much less.

It is not easy to find a theoretical impulse response for the DMO operator in $v(z)$ media, but you can easily compute the impulse response in $v(z)$ by using $\mathbf{R} = \mathbf{H}_0\mathbf{H}_h'$ from equation (8.18).

8.7.2. Results of our DMO program

We now return to the field data from the Gulf of Mexico, which we have processed earlier in this chapter and in chapter 4.

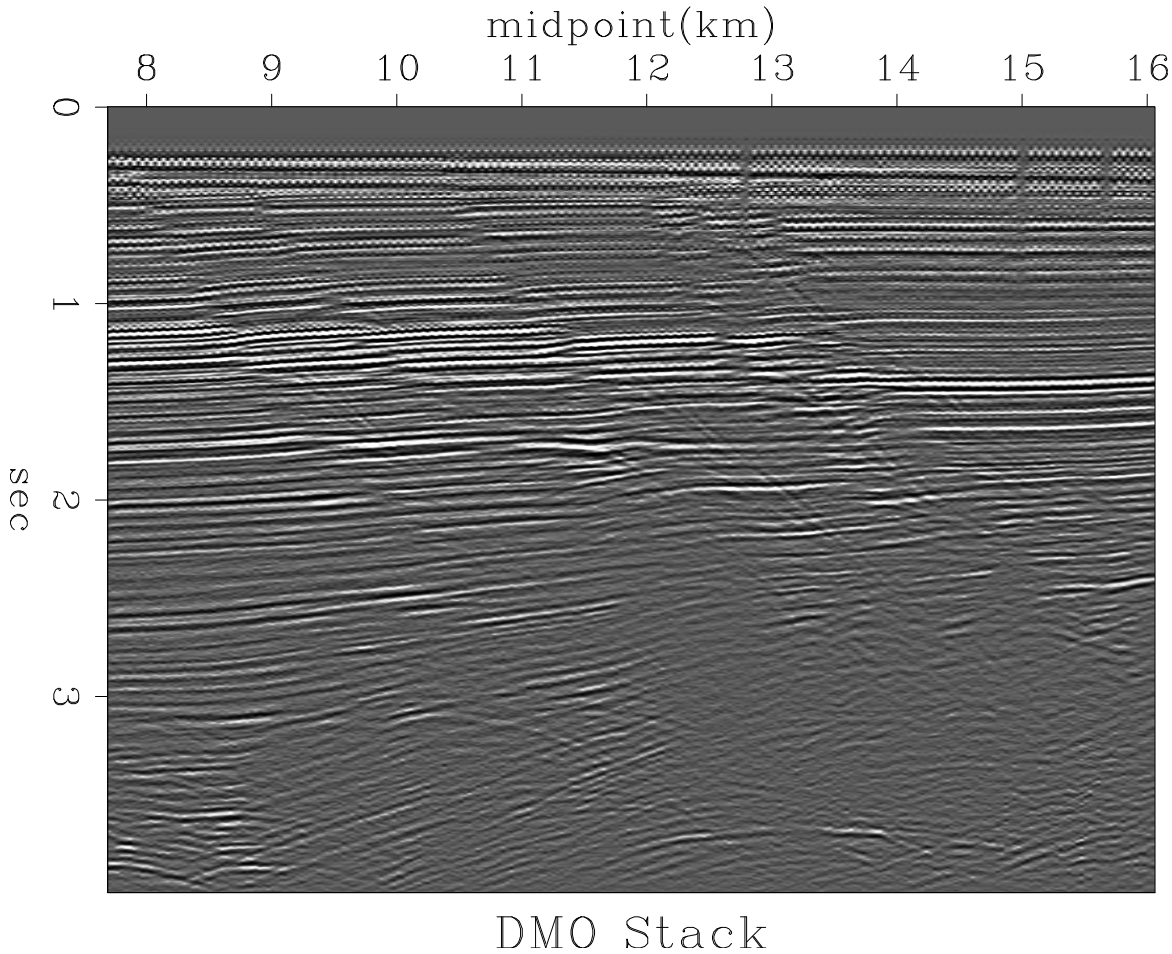


Figure 8.26: Stack after the dip-moveout correction. Compare this result with Figure 4.7. This one has fault plane reflections to the right of the faults. dpmv-wgdmostk [ER,M]

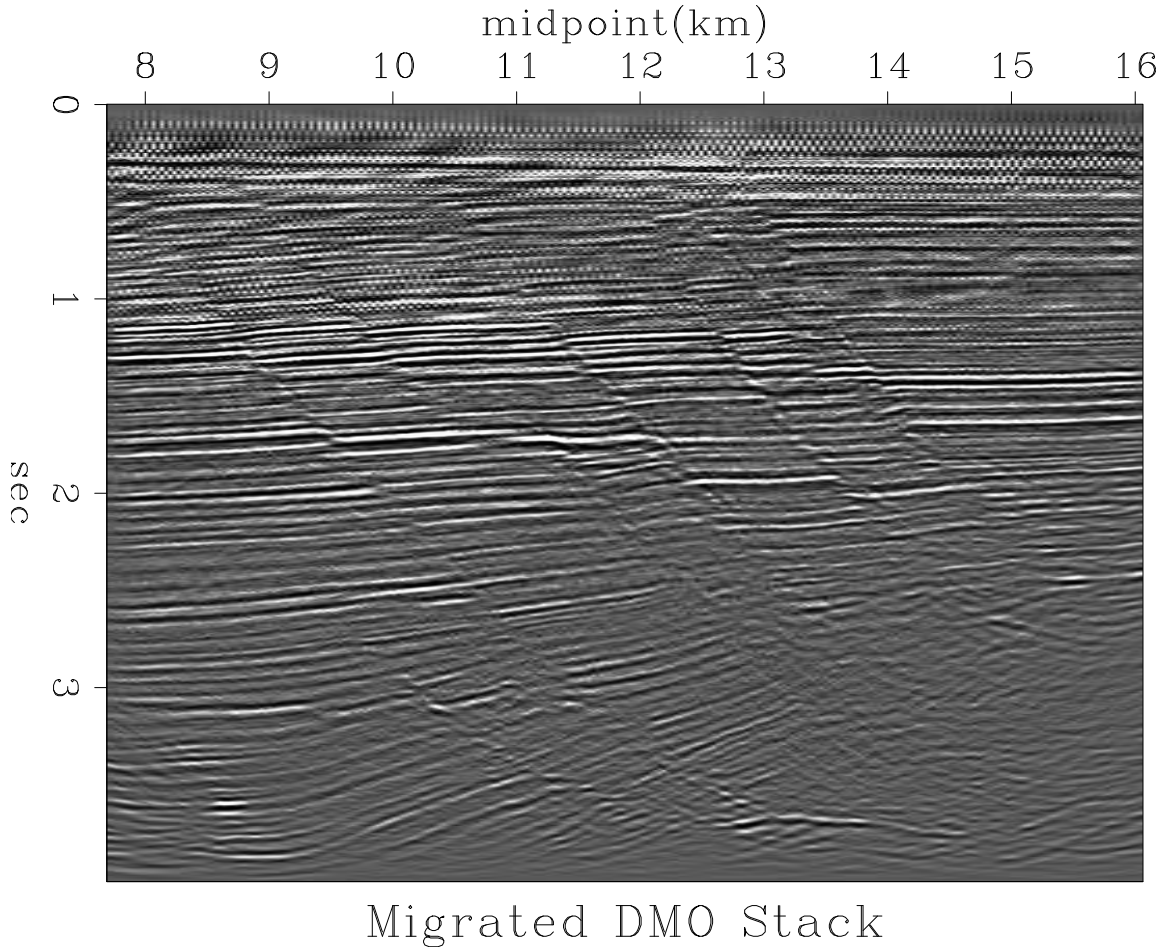


Figure 8.27: Kirchhoff migration of the previous figure. Now the fault plane reflections jump to the fault. dpmv-wgdmomig [ER,M]

Chapter 9

Finite-difference migration

This chapter is a condensation of wave extrapolation and finite-difference basics from IEI which is now out of print. On the good side, this new organization is more compact and several errors have been corrected. On the bad side, to follow up on

many many interesting details you will need to find a copy of IEI (http://sepwww.stanford.

In chapter 7 we learned how to extrapolate wavefields down into the earth. The process proceeded simply, since it is just a multiplication in the frequency domain by $\exp[ik_z(\omega, k_x)z]$. In this chapter instead of multiplying a wavefield by a function of k_x to downward continue waves, we will convolve them along the x-axis with a small filter that represents a differential equation. On space axes, a concern is the seismic velocity v. With **lateral velocity variation**, say $v(x)$, then the operation of extrapolating wavefields upward and downward can no longer be expressed as a product in the k_x-domain. (Wave-extrapolation procedures in the spatial frequency domain are no longer multiplication, but convolution.) The alternative we choose here is to go to finite differences which are convolution in the physical x domain. This is what the wave equation itself does.

9.1. THE PARABOLIC EQUATION

Here we derive the most basic migration equation via the dispersion relation, equation (7.11). Recall this equation basically says $\cos\theta = \sqrt{1-\sin^2\theta}$.

$$k_z \quad = \quad \frac{\omega}{v}\sqrt{1 \; - \; \frac{v^2 k_x^2}{\omega^2}} \tag{9.1}$$

The dispersion relation above is the foundation for downward continuing wavefields by Fourier methods in chapter 7. Recall that nature extrapolates forward in time from $t = 0$ whereas a geophysicist extrapolates information in depth from $z = 0$. We get ideas for our task, and then we hope to show that our ideas are consistent with nature. Suppose we substitute $ik_z = \partial/\partial z$ into equation (9.1), multiply by P, and interpret velocity as depth variable.

$$\frac{\partial P}{\partial z} \quad = \quad \frac{i\,\omega}{v(z)}\sqrt{1 \; - \; \frac{v(z)^2\,k_x^2}{\omega^2}}\;\; P \tag{9.2}$$

Since the above steps are unorthodox, we need to enquire about their validity. Suppose that equation (9.2) were valid. Then we could restrict it to constant velocity

and take a trial solution $P = P_0 \exp(-ik_z z)$ and we would immediately have equation (9.1). Why do we believe the introduction of $v(z)$ in equation (9.2) has any validity? We can think about the phase shift migration method in chapter 7. It handled $v(z)$ by having the earth velocity being a staircase function of depth. Inside a layer we had the solution to equation (9.2). To cross a layer boundary, we simply asserted that the wavefield at the bottom of one layer would be the same as the wavefield at the top of the next which is also the solution to equation (9.2). (Let $\Delta z \to 0$ be the transition from one layer to the next. Then $\Delta P = 0$ since $\partial P / \partial z$ is finite.) Although equation (9.2) is consistent with chapter 7, it is an approximation of limited validity. It assumes there is no reflection at a layer boundary. Reflection would change part of a downgoing wave to an upcoming wave and the wave that continued downward would have reduced amplitude because of lost energy. Thus, by our strong desire to downward continue wavefields (extrapolate in z) whereas nature extrapolates in t, we have chosen to ignore reflection and transmission coefficients. Perhaps we can recover them, but now we have bigger fish to fry. We want to be able to handle $v(x,z)$, **lateral velocity variation**. This requires us to get rid of

the square root in equation (9.2). Make a power series for it and drop higher terms.

$$\frac{\partial P}{\partial z} \quad = \quad \frac{i\,\omega}{v(z)} \left(1 \; - \; \frac{v(z)^2\, k_x^2}{2\,\omega^2} \right) P \; + \cdots \tag{9.3}$$

The first dropped term is $\sin^4\theta$ where θ is the dip angle of a wavefront. The dropped terms increase slowly with angle, but they do increase, and dropping them will limit the range of angles that we can handle with this equation. This is a bitter price to pay for the benefit of handling $v(x,z)$, and we really will return to patch it up (unlike the transmission coefficient problem). There are many minus signs cropping up, so I give you another equation to straighten them out.

$$\frac{\partial P}{\partial z} \quad = \quad \left(\frac{i\,\omega}{v(z)} \quad - \quad \frac{v(z)\,k_x^2}{-\,i\,\omega\,2} \right) P \tag{9.4}$$

Now we are prepared to leap to our final result, an equation for downward continuing waves in the presence of depth and **lateral velocity variation** $v(x,z)$. Substitute $\partial_{xx} = -k_x^2$ into equation (9.4) and revise interpretation of P from $P(\omega, k_x, z)$ to $P(\omega, x, z)$.

$$\frac{\partial P}{\partial z} \quad = \quad \frac{i\,\omega}{v(x,z)}\ P \ + \ \frac{v(x,z)}{-i\,\omega 2}\ \frac{\partial^2 P}{\partial x^2} \tag{9.5}$$

As with $v(z)$, there is a loss of lateral transmission and reflection coefficients. We plan to forget this minor problem. It is the price of being a data handler instead of a modeler. Equation (9.5) is the basis for our first program and examples.

9.2. SPLITTING AND SEPARATION

Two processes, **A** and **B**, which ordinarily act simultaneously, may or may not be interconnected. The case where they are independent is called ***full separation***. In this case it is often useful, for thought and for computation, to imagine process **A** going to completion before process **B** is begun. Where the processes are interconnected it is possible to allow **A** to run for a short while, then switch to **B**, and continue in alternation. This alternation approach is called *splitting*.

9.2.1. The heat-flow equation

We wish to solve equation (9.5) by a method involving **splitting**. Since equation (9.5) is an unfamiliar one, we turn to the **heat-flow equation** which besides being familiar, has no complex numbers. A two-sentence derivation of the **heat-flow equation** follows. (1) The heat flow H_x in the x-direction equals the negative of the gradient $-\partial/\partial x$ of temperature T times the heat conductivity σ. (2) The decrease of temperature $-\partial T/\partial t$ is proportional to the divergence of the heat flow $\partial H_x/\partial x$ divided by the heat storage capacity C of the material. Combining these, extending from one dimension to two, taking σ constant and $C = 1$, gives the equation

$$\frac{\partial T}{\partial t} = \left(\sigma \frac{\partial^2}{\partial x^2} + \sigma \frac{\partial^2}{\partial y^2}\right) T \tag{9.6}$$

9.2.2. Splitting

The **splitting** method for numerically solving the **heat-flow equation** is to replace the two-dimensional heat-flow equation by two one-dimensional equations, each of

which is used on alternate time steps:

$$\frac{\partial T}{\partial t} = 2\sigma \frac{\partial^2 T}{\partial x^2} \quad \text{(all } y\text{)} \tag{9.7}$$

$$\frac{\partial T}{\partial t} = 2\sigma \frac{\partial^2 T}{\partial y^2} \quad \text{(all } x\text{)} \tag{9.8}$$

In equation (9.7) the heat conductivity σ has been doubled for flow in the x-direction and zeroed for flow in the y-direction. The reverse applies in equation (9.8). At odd moments in time heat flows according to (9.7) and at even moments in time it flows according to (9.8). This solution by alternation between (9.7) and (9.8) can be proved mathematically to converge to the solution to (9.6) with errors of the order of Δt. Hence the error goes to zero as Δt goes to zero.

9.2.3. Full separation

Splitting can turn out to be much more accurate than might be imagined. In many cases there is *no* loss of accuracy. Then the method can be taken to an extreme limit. Think about a radical approach to equations (9.7) and (9.8) in which, instead

of alternating back and forth between them at alternate time steps, what is done is to march (9.7) through all time steps. Then this intermediate result is used as an initial condition for (9.8), which is marched through all time steps to produce a final result. It might seem surprising that this radical method can produce the correct solution to equation (9.6). But if σ is a constant function of x and y, it does. The process is depicted in Figure 9.1 for an impulsive initial disturbance. A differential equation like (9.6) is said to be *fully separable* when the correct solution is obtainable by the radical method. It should not be too surprising that **full separation** works when σ is a constant, because then Fourier transformation may be used, and the two-dimensional solution $\exp[-\sigma\,(k_x^2 + k_y^2)t]$ equals the succession of one-dimensional solutions $\exp(-\sigma\,k_x^2 t)\,\exp(-\sigma\,k_y^2 t)$. It turns out, and will later be shown, that the condition required for applicability of **full separation** is that $\sigma\,\partial^2/\partial x^2$ should commute with $\sigma\,\partial^2/\partial y^2$, that is, the order of differentiation should be irrelevant. Technically there is also a boundary-condition requirement, but it creates no difficulty when the disturbance dies out before reaching a boundary.

There are circumstances which dictate a middle road between **splitting** and **full separation**, for example if σ were a slowly variable function of x or y. Then you might find that although $\sigma\,\partial^2/\partial x^2$ does not strictly commute with $\sigma\,\partial^2/\partial y^2$,

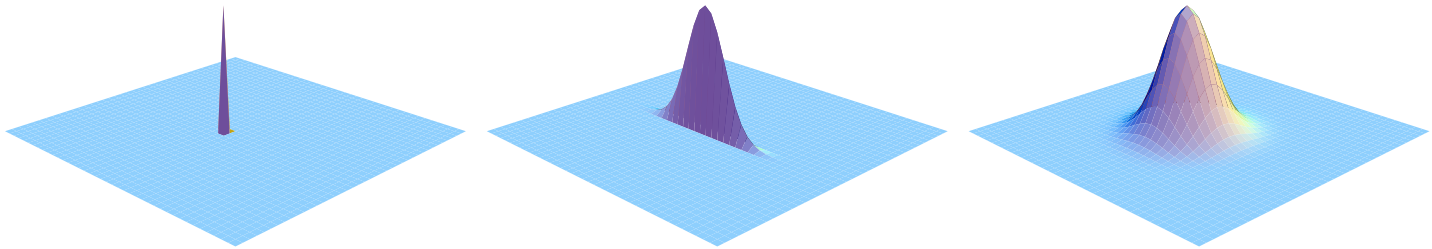


Figure 9.1: Temperature distribution in the (x, y)-plane beginning from a delta function (left). After heat is allowed to flow in the x-direction but not in the y-direction the heat is located in a "wall" (center). Finally allowing heat to flow for the same amount of time in the y-direction but not the x-direction gives the same symmetrical Gaussian result that would have been found if the heat had moved in x- and y-directions simultaneously (right). fdm-temperature [CR]

it comes close enough that a number of time steps may be made with (9.7) before you transpose the data and switch over to (9.8). Circumstances like this one but with more geophysical interest arise with the wave-extrapolation equation that is considered next.

9.2.4. Splitting the parabolic equation

In discussing and solving the **parabolic wave equation** it is convenient to rearrange it to recognize the role of an averaged stratified medium of velocity of $\overline{v}(z)$ and departures from it.

$$\begin{aligned} \frac{\partial P}{\partial z} &= i\,\omega \left(\frac{1}{\overline{v}(z)} \right) P + i\,\omega \left(\frac{1}{v(x,z)} - \frac{1}{\overline{v}(z)} \right) P + \left(\frac{v(x,z)}{-i\,\omega 2} \frac{\partial^2}{\partial x^2} \right) P \qquad (9.9) \\ &= A\,P + B\,P + C\,P \\ &= \text{shift} + \text{thin lens} + \text{diffraction} \end{aligned}$$

The shift term in (9.9) commutes with the thin-**lens term**, that is, $ABP = BAP$. the shift term also commutes with the diffraction term because $ACP = CAP$. But the thin-lens term and the diffraction term do not commute with one another because

$(BC-CB)P \neq 0$, because

$$0 \quad \neq \quad (BC-CB)P \quad = \quad v(x,z)\left[\left(-2\frac{d^2}{dx^2}\frac{1}{v(x,z)}\right) + \frac{1}{v(x,z)^2}\frac{dv(x,z)}{dx}\frac{\partial}{\partial x}\right]P \tag{9.10}$$

Mathematicians look at the problem this way: Consider any fixed wave propagation angle so vk_x/ω is a constant. Now let frequency ω (and hence k_x) tend together to infinity. The terms in BCP and CBP grow in proportion to the second power of frequency, whereas those in $(BC-CB)P$ grow as lower powers. There is however, a catch. The material properties have a "wavelength" too. We can think of $(dv/dx)/v$ as a spatial wavenumber for the material just as k_x is a spatial wavenumber for the wave. If the material contains a step function change in its properties, that is an infinite spatial frequency $(dv/dx)/v$ for the material. Then the $(BC-CB)P$ terms dominate near the place where one material changes to another. If we drop the $(BC-CB)P$ terms, we'll get the transmission coefficient incorrect, although everything would be quite fine everywhere else except at the boundary.

A question is, to what degree do the terms commute? The problem is just that of focusing a slide projector. Adjusting the focus knob amounts to repositioning the thin-lens term in comparison to the free-space diffraction term. There is a

small range of knob positions over which no one can notice any difference, and a larger range over which the people in the back row are not disturbed by misfocus. Much geophysical data processing amounts to downward extrapolation of data. The **lateral velocity variation** occurring in the **lens term** is known only to a limited accuracy and we often wish to determine $v(x)$ by the extrapolation procedure.

In practice it seems best to forget the $(BC - CB)P$ terms because we hardly ever know the material properties very well anyway. Then we split, doing the shift and the thin-lens part analytically while doing the diffraction part by a numerical method.

9.2.5. Validity of the splitting and full-separation concepts

Feel free to skip forward over this subsection which is merely a mathematical proof.

When Fourier transformation is possible, extrapolation operators are complex numbers like $e^{ik_z z}$. With complex numbers a and b there is never any question that $ab = ba$. Then both **splitting** and **full separation** are always valid.

Suppose Fourier transformation has not been done, or could not be done because of some spatial variation of material properties. Then extrapolation opera-

tors are built up by combinations of differential operators or their finite-difference representations. Let **A** and **B** denote two such operators. For example, **A** could be a matrix containing the second x differencing operator. Seen as matrices, the **boundary condition**s of a differential operator are incorporated in the corners of the matrix. The bottom line is whether $\mathbf{AB} = \mathbf{BA}$, so the question clearly involves the **boundary** conditions as well as the differential operators.

Extrapolation downward a short distance can be done with the operator $(\mathbf{I} + \mathbf{A}\,\Delta z)$. Let **p** denote a vector where components of the vector designate the wavefield at various locations on the x-axis. Numerical analysis gives us a matrix operator, say **A**, which enables us to project forward, say,

$$\mathbf{p}(z + \Delta z) \quad = \quad \mathbf{A}_1\ \mathbf{p}(z) \tag{9.11}$$

The subscript on **A** denotes the fact that the operator may change with z. To get a step further the operator is applied again, say,

$$\mathbf{p}(z + 2\,\Delta z) \quad = \quad \mathbf{A}_2\ [\mathbf{A}_1\ \mathbf{p}(z)] \tag{9.12}$$

From an operational point of view the matrix **A** is never squared, but from an ana-

lytical point of view, it really is squared.

$$\mathbf{A}_2\,[\mathbf{A}_1\,\mathbf{p}(z)] \quad = \quad (\mathbf{A}_2\,\mathbf{A}_1)\,\mathbf{p}(z) \tag{9.13}$$

To march some distance down the z-axis we apply the operator many times. Take an interval $z_1 - z_0$, to be divided into N subintervals. Since there are N intervals, an error proportional to $1/N$ in each subinterval would accumulate to an unacceptable level by the time z_1 was reached. On the other hand, an error proportional to $1/N^2$ could only accumulate to a total error proportional to $1/N$. Such an error would disappear as the number of subintervals increased.

To prove the validity of **splitting**, we take $\Delta z = (z_1 - z_0)/N$. Observe that the operator $\mathbf{I} + (\mathbf{A} + \mathbf{B})\Delta z$ differs from the operator $(\mathbf{I} + \mathbf{A}\,\Delta z)(\mathbf{I} + \mathbf{B}\,\Delta z)$ by something in proportion to Δz^2 or $1/N^2$. So in the limit of a very large number of subintervals, the error disappears.

It is much easier to establish the validity of the full-separation concept. Commutativity is whether or not $\mathbf{AB} = \mathbf{BA}$. Commutativity is always true for scalars. With finite differencing the question is whether the two matrices commute. Taking **A** and **B** to be differential operators, commutativity is defined with the help of the family of all possible wavefields P. Then **A** and **B** are commutative if

$\mathbf{A}\mathbf{B}\,P = \mathbf{B}\mathbf{A}\,P$.

The operator representing $\partial P/\partial z$ will be taken to be $\mathbf{A}+\mathbf{B}$. The simplest numerical integration scheme using the **splitting** method is

$$P(z_0 + \Delta z) \quad = \quad (\mathbf{I} + \mathbf{A}\,\Delta z)\,(\mathbf{I} + \mathbf{B}\,\Delta z)\,P(z_0) \tag{9.14}$$

Applying (9.14) in many stages gives a product of many operators. The operators $\mathbf{A}$ and $\mathbf{B}$ are subscripted with j to denote the possibility that they change with z.

$$P(z_1) \quad = \quad \prod_{j=1}^{N} [(\mathbf{I} + \mathbf{A}_j\,\Delta z)(\mathbf{I} + \mathbf{B}_j\,\Delta z)]\,P(z_0) \tag{9.15}$$

As soon as $\mathbf{A}$ and $\mathbf{B}$ are assumed to be commutative, the factors in (9.15) may be rearranged at will. For example, the $\mathbf{A}$ operator could be applied in its entirety before the $\mathbf{B}$ operator is applied:

$$P(z_1) \quad = \quad \left[\prod_{j=1}^{N} (\mathbf{I} + \mathbf{B}_j\,\Delta z)\right] \left[\prod_{j=1}^{N} (\mathbf{I} + \mathbf{A}_j\,\Delta z)\right] P(z_0) \tag{9.16}$$

Thus the **full-separation** concept is seen to depend on the commutativity of operators.

9.3. FINITE DIFFERENCING IN (omega,x)-SPACE

The basic method for solving differential equations in a computer is *finite differencing.* The nicest feature of the method is that it allows analysis of objects of almost any shape, such as earth topography or geological structure. Ordinarily, finite differencing is a straightforward task. The main pitfall is instability. It often happens that a seemingly reasonable approach to a reasonable physical problem leads to wildly oscillatory, divergent calculations. Luckily, a few easily learned tricks go a long way, and we will be covering them here.

9.3.1. The lens equation

The parabolic wave-equation operator can be split into two parts, a complicated part called the *diffraction* or *migration* part, and an easy part called the *lens* part. The **lens equation** applies a time shift that is a function of x. The **lens equation**

acquires its name because it acts just like a thin optical lens when a light beam enters on-axis (vertically). Corrections for nonvertical incidence are buried somehow in the diffraction part. The **lens equation** has an analytical solution, namely, $\exp[i\omega t_0(x)]$. It is better to use this analytical solution than to use a finite-difference solution because there are no approximations in it to go bad. The only reason the **lens equation** is mentioned at all in a chapter on finite differencing is that the companion diffraction equation must be marched forward along with the **lens equation**, so the analytic solutions are marched along in small steps.

9.3.2. First derivatives, explicit method

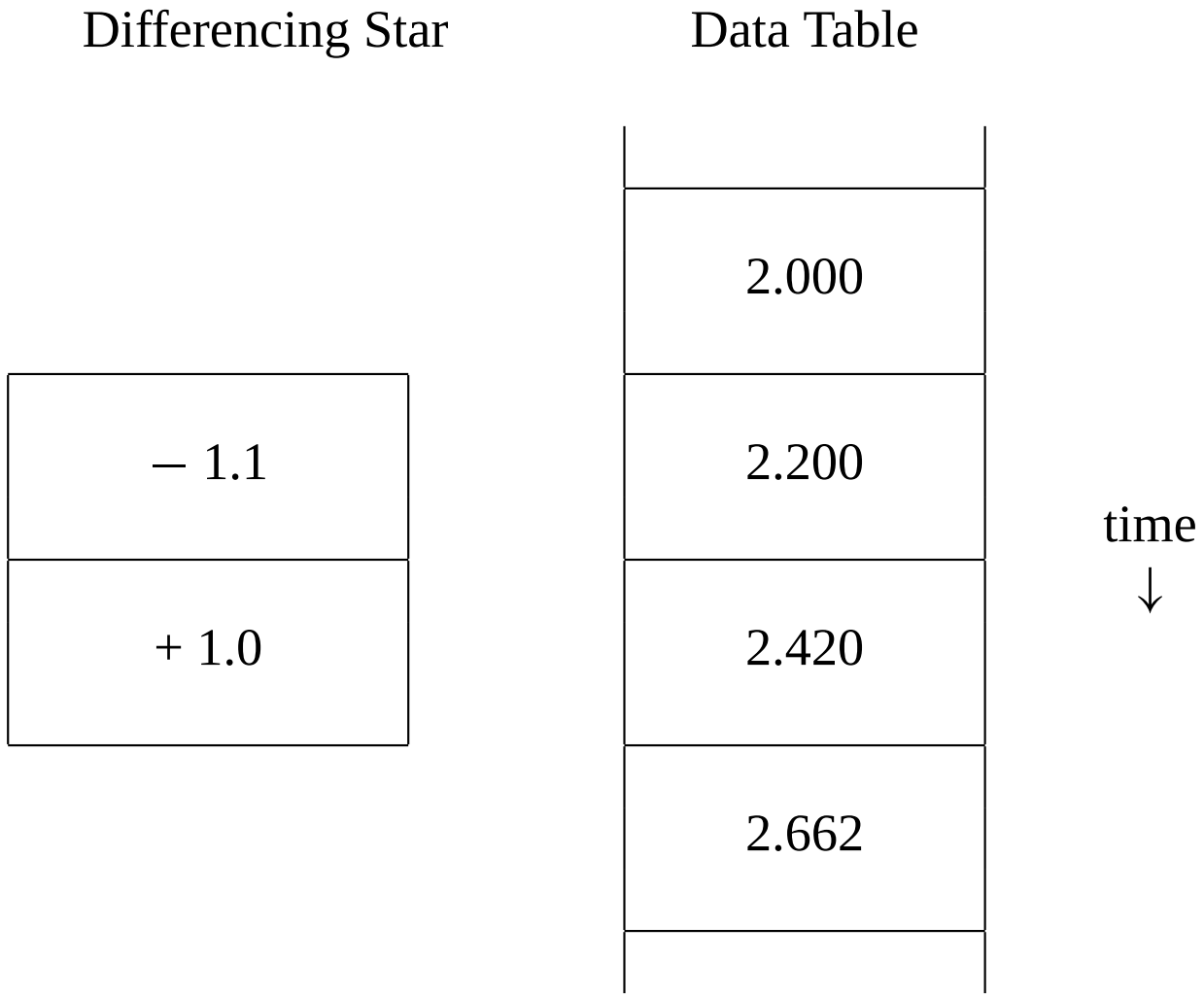
The inflation of money q at a 10% rate can be described by the difference equation

$$q_{t+1} - q_t = .10\, q_t \tag{9.17}$$

$$(1.0)\, q_{t+1} + (-1.1)\, q_t = 0 \tag{9.18}$$

This one-dimensional calculation can be reexpressed as a differencing star and a data table. As such it provides a prototype for the organization of calculations with

two-dimensional partial-differential equations. Consider



Since the data in the data table satisfy the difference equations (9.17) and (9.18), the differencing star may be laid anywhere on top of the data table, the numbers in the star may be multiplied by those in the underlying table, and the resulting cross products will sum to zero. On the other hand, if all but one number (the initial condition) in the data table were missing then the rest of the numbers could be filled in, one at a time, by sliding the star along, taking the difference equations to be true, and solving for the unknown data value at each stage.

Less trivial examples utilizing the same differencing star arise when the numerical constant .10 is replaced by a complex number. Such examples exhibit oscillation as well as growth and decay.

9.3.3. First derivatives, implicit method

Let us solve the equation

$$\frac{dq}{dt} = 2\,r\,q \tag{9.19}$$

by numerical methods. The most obvious (but not the only) approach is the basic definition of elementary calculus. For the time derivative, this is

$$\frac{dq}{dt} \approx \frac{q(t+\Delta t) - q(t)}{\Delta t} \tag{9.20}$$

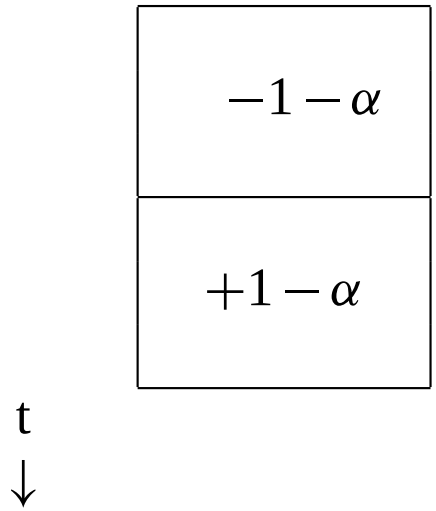
Using this in equation (9.19) yields the the inflation-of-money equations (9.17) and (9.18), where $2r = .1$. Thus in the inflation-of-money equation the expression of dq/dt is centered at $t + \Delta t/2$, whereas the expression of q by itself is at time t. There is no reason the q on the right side of equation (9.19) cannot be averaged at time t with time $t + \Delta t$, thus centering the whole equation at $t + \Delta t/2$. When writing difference equations, it is customary to write $q(t + \Delta t)$ more simply as q_{t+1}. (Formally one should say $t = n\Delta t$ and write q_{n+1} instead of q_{t+1}, but helpful mnemonic information is carried by using t as the subscript instead of some integer like n.) Thus, a centered approximation of (9.19) is

$$q_{t+1} - q_t = 2r\,\Delta t\,\frac{q_{t+1} + q_t}{2} \tag{9.21}$$

Letting $\alpha = r\,\Delta t$, this becomes

$$(1-\alpha)\,q_{t+1} - (1+\alpha)\,q_t = 0 \tag{9.22}$$

which is representable as the difference star



For a fixed Δt this star gives a more accurate solution to the differential equation (9.19) than does the star for the inflation of money. The reasons for the names "**explicit method**" and "**implicit method**" above will become clear only after we study a more complicated equation such as the **heat-flow equation**.

9.3.4. Explicit heat-flow equation

The **heat-flow equation** (9.6) is a prototype for migration. Let us recopy the heat-flow equation letting q denote the temperature.

$$\frac{\partial q}{\partial t} \quad = \quad \frac{\sigma}{C}\,\frac{\partial^2 q}{\partial x^2} \tag{9.23}$$

Implementing (9.23) in a computer requires some difference approximations for the partial differentials. As before we use a subscript notation that allows (9.20) to be compacted into

$$\frac{\partial q}{\partial t} \quad \approx \quad \frac{q_{t+1} - q_t}{\Delta t} \tag{9.24}$$

where $t + \Delta t$ is denoted by $t + 1$. The second-derivative formula may be obtained by doing the first derivative twice. This leads to $q_{t+2} - 2\,q_{t+1} + q_t$. The formula is usually treated more symmetrically by shifting it to $q_{t+1} - 2\,q_t + q_{t-1}$. These two versions are equivalent as Δt tends to zero, but the more symmetrical arrangement will be more accurate when Δt is not zero. Using superscripts to describe x-

dependence gives a finite-difference approximation to the second space derivative:

$$\frac{\partial^2 q}{\partial x^2} \quad \approx \quad \frac{q^{x+1} - 2q^x + q^{x-1}}{\Delta x^2} \tag{9.25}$$

Inserting the last two equations into the **heat-flow equation** (and using = to denote $\approx$) gives

$$\frac{q_{t+1}^x - q_t^x}{\Delta t} \quad = \quad \frac{\sigma}{C} \, \frac{q_t^{x+1} - 2q_t^x + q_t^{x-1}}{\Delta x^2} \tag{9.26}$$

(Of course it is not justified to use = to denote $\approx$, but the study of errors must be deferred until the concepts have been laid out. Errors are studied in IEI chapter 4. Letting $\alpha = \sigma \, \Delta t/(C \, \Delta x^2)$, equation (9.26) becomes

$$q_{t+1}^x - q_t^x - \alpha(q_t^{x+1} - 2q_t^x + q_t^{x-1}) \quad = \quad 0 \tag{9.27}$$

Equation (9.27) can be *explicitly* solved for q for any x at the particular time $t+1$ given q at all x for the particular time t and hence the name ***explicit method***.

Equation (9.27) can be interpreted geometrically as a computational star in the (x,t)-plane, as depicted in Table 9.1. By moving the star around in the data table you will note that it can be positioned so that only one number at a time (the 1) lies over

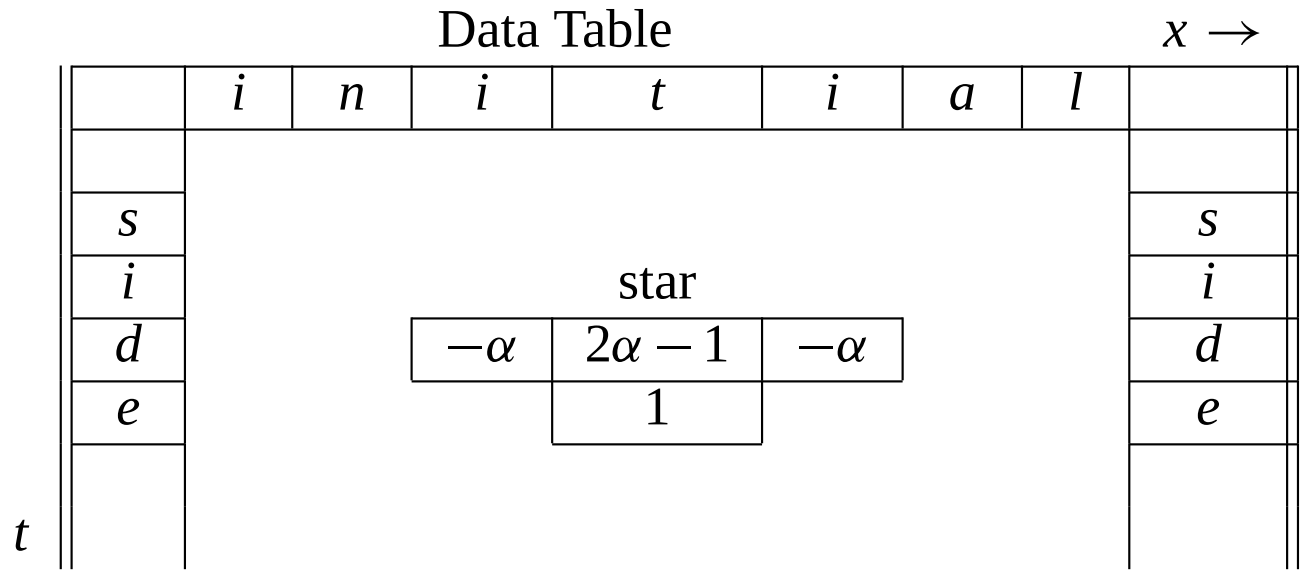


Table 9.1: Differencing star and table for one-dimensional **heat-flow equation**.

an unknown element in the data table. This enables the computation of subsequent rows beginning from the top. By doing this you are solving the partial-differential equation by the finite-difference method. There are many possible arrangements of initial and side conditions, such as zero-value side conditions. Next is a computer program for the job and its result.

```
# Explicit heat-flow equation
real q(12), qp(12)
nx = 12
do ia= 1, 2 {           # stable and unstable cases
        alpha = ia*.3333;       write(6,'(/"alpha =",f5.2)') alpha
        do ix= 1,6  { q(ix) = 0.}       # Initial temperature step
        do ix= 7,12 { q(ix) = 1.}
        do it= 1, 6 {
                write(6,'(20f6.2)') (q(ix),ix=1,nx)
                do ix= 2, nx-1
                        qp(ix) = q(ix) + alpha*(q(ix-1)-2.*q(ix)+q(ix+1))
                qp(1) = qp(2);  qp(nx) = qp(nx-1)
                do ix= 1, nx
                        q(ix) = qp(ix)
                }
        }
call exit(0);   end


alpha = 0.33
  0.00  0.00  0.00  0.00  0.00  0.00  1.00  1.00  1.00  1.00  1.00  1.00
  0.00  0.00  0.00  0.00  0.00  0.33  0.67  1.00  1.00  1.00  1.00  1.00
  0.00  0.00  0.00  0.00  0.11  0.33  0.67  0.89  1.00  1.00  1.00  1.00
```

```
  0.00  0.00  0.00  0.04  0.15  0.37  0.63  0.85  0.96  1.00  1.00  1.00
  0.00  0.00  0.01  0.06  0.19  0.38  0.62  0.81  0.94  0.99  1.00  1.00
  0.00  0.00  0.02  0.09  0.21  0.40  0.60  0.79  0.91  0.98  1.00  1.00

alpha = 0.67
  0.00  0.00  0.00  0.00  0.00  0.00  1.00  1.00  1.00  1.00  1.00  1.00
  0.00  0.00  0.00  0.00  0.00  0.67  0.33  1.00  1.00  1.00  1.00  1.00
  0.00  0.00  0.00  0.00  0.44  0.00  1.00  0.56  1.00  1.00  1.00  1.00
  0.00  0.00  0.00  0.30 -0.15  0.96  0.04  1.15  0.70  1.00  1.00  1.00
  0.00  0.00  0.20 -0.20  0.89 -0.39  1.39  0.11  1.20  0.80  1.00  1.00
  0.13  0.13 -0.20  0.79 -0.69  1.65 -0.65  1.69  0.21  1.20  0.87  0.87
```

9.3.5. The leapfrog method

A difficulty with the given program is that it doesn't work for all possible numerical values of α. You can see that when α is too large (when Δx is too small) the solution in the interior region of the data table contains growing oscillations. What is happening is that the low-frequency part of the solution is OK (for a while), but the high-frequency part is diverging. The mathematical reason the divergence occurs is the subject of mathematical analysis found in IEI section 2.8. Intuitively, at wavelengths long compared to Δx or Δt, we expect the difference approximation to agree with the true **heat-flow equation**, smoothing out irregularities in temperature. At short wavelengths the wild oscillation shows that the difference equation can

behave in a way almost opposite to the way the differential equation behaves. The short wavelength discrepancy arises because difference operators become equal to differential operators only at long wavelengths. The divergence of the solution is a fatal problem because the subsequent round-off error will eventually destroy the low frequencies too.

By supposing that the instability arises because the time derivative is centered at a slightly different time $t + 1/2$ than the second x-derivative at time t, we are led to the so-called ***leapfrog method***, in which the time derivative is taken as a difference between $t - 1$ and $t + 1$:

$$\frac{\partial q}{\partial t} \quad \approx \quad \frac{q_{t+1} - q_{t-1}}{2\ \Delta t} \tag{9.28}$$

Here the result is even worse. An analysis found in IEI shows that the solution is now divergent for *all* real numerical values of α. Although it was a good idea to center both derivatives in the same place, it turns out that it was a bad idea to express a first derivative over a span of more mesh points. The enlarged operator has two solutions in time instead of just the familiar one. The numerical solution is the sum of the two theoretical solutions, one of which, unfortunately (in this case), grows and oscillates for all real values of α.

To avoid all these problems (and get more accurate answers as well), we now turn to some slightly more complicated solution methods known as ***implicit methods***.

9.3.6. The Crank-Nicolson method

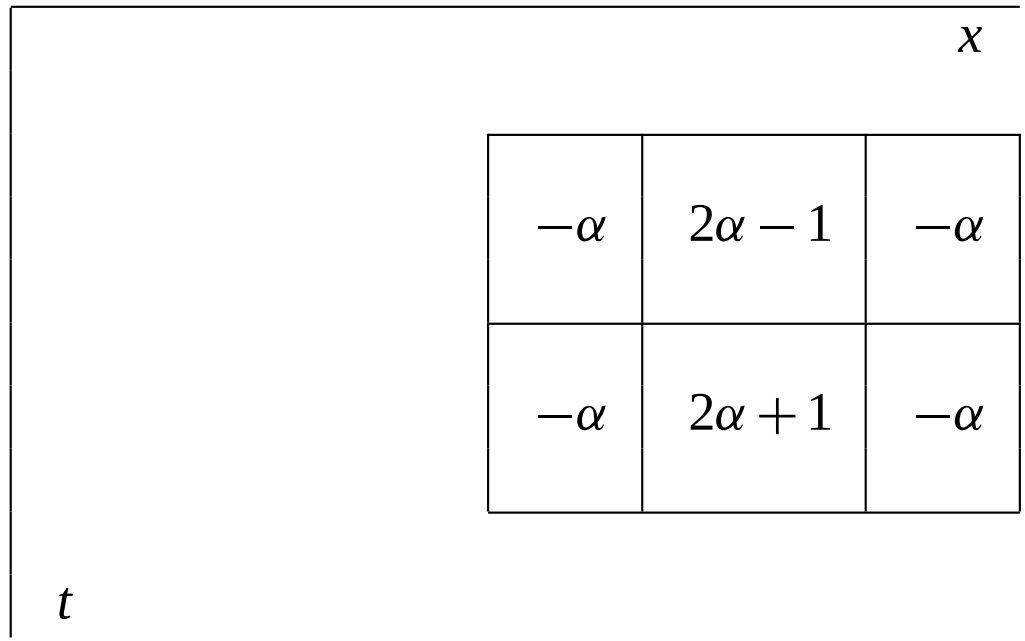
The **Crank-Nicolson method** solves both the accuracy and the stability problem. Recall the difference representation of the **heat-flow equation** (9.27).

$$q_{t+1}^{x} - q_t^x = a\left(q_t^{x+1} - 2q_t^x + q_t^{x-1}\right) \tag{9.29}$$

Now, instead of expressing the right-hand side entirely at time t, it will be averaged at t and $t+1$, giving

$$q_{t+1}^{x} - q_t^x = \frac{a}{2}\left[\left(q_t^{x+1} - 2q_t^x + q_t^{x-1}\right) + \left(q_{t+1}^{x+1} - 2q_{t+1}^x + q_{t+1}^{x-1}\right)\right] \tag{9.30}$$

This is called the **Crank-Nicolson method**. Defining a new parameter $\alpha = a/2$, the difference star is



$-\alpha$	$2\alpha - 1$	$-\alpha$
$-\alpha$	$2\alpha + 1$	$-\alpha$

(9.31)

When placing this star over the data table, note that, typically, three elements at a time cover unknowns. To say the same thing with equations, move all the $t+1$ terms in (9.30) to the left and the t terms to the right, obtaining

$$-\alpha q_{t+1}^{x+1} + (1+2\alpha)q_{t+1}^{x} - \alpha q_{t+1}^{x-1} \quad = \quad \alpha q_{t}^{x+1} + (1-2\alpha)q_{t}^{x} + \alpha q_{t}^{x-1} \tag{9.32}$$

Now think of the left side of equation (9.32) as containing all the *unknown* quantities and the right side as containing all *known* quantities. Everything on the right can be combined into a single known quantity, say, d_t^x. Now we can rewrite equation (9.32) as a set of simultaneous equations. For definiteness, take the x-axis to be limited to five points. Then these equations are:

$$\begin{bmatrix} e_{\text{left}} & -\alpha & 0 & 0 & 0 \\ -\alpha & 1+2\alpha & -\alpha & 0 & 0 \\ 0 & -\alpha & 1+2\alpha & -\alpha & 0 \\ 0 & 0 & -\alpha & 1+2\alpha & -\alpha \\ 0 & 0 & 0 & -\alpha & e_{\text{right}} \end{bmatrix} \begin{bmatrix} q_{t+1}^1 \\ q_{t+1}^2 \\ q_{t+1}^3 \\ q_{t+1}^4 \\ q_{t+1}^5 \end{bmatrix} = \begin{bmatrix} d_t^1 \\ d_t^2 \\ d_t^3 \\ d_t^4 \\ d_t^5 \end{bmatrix} \tag{9.33}$$

Equation (9.32) does not give us each q_{t+1}^x *explicitly*, but equation (9.33) gives them *implicitly* by the solution of simultaneous equations.

The values e_{left} and e_{right} are adjustable and have to do with the side **boundary conditions**. The important thing to notice is that the matrix is **tridiagonal**, that is, except for three central diagonals all the elements of the matrix in (9.33) are zero. The solution to such a set of simultaneous equations may be economically obtained. It turns out that the cost is only about twice that of the **explicit method** given by

(9.27). In fact, this **implicit method** turns out to be cheaper, since the increased accuracy of (9.32) over (9.27) allows the use of a much larger numerical choice of Δt. A program that demonstrates the stability of the method, even for large Δt, is given next.

A **tridiagonal** simultaneous equation solving subroutine `rtris()` explained in the next section. The results are stable, as you can see.

```
a = 8.00
  0.00  0.00  0.00  0.00  0.00  0.00  1.00  1.00  1.00  1.00  1.00  1.00
  0.17  0.17  0.21  0.30  0.47  0.76  0.24  0.53  0.70  0.79  0.83  0.83
  0.40  0.40  0.42  0.43  0.40  0.24  0.76  0.60  0.57  0.58  0.60  0.60
  0.44  0.44  0.44  0.44  0.48  0.68  0.32  0.52  0.56  0.56  0.56  0.56
```

```
# Implicit heat-flow equation
real q(12),d(12)
nx=12; a = 8.;  write(6,'(/"a =",f5.2)') a;     alpha = .5*a
do ix= 1,6  {   q(ix) = 0.}     # Initial temperature step
do ix= 7,12 {   q(ix) = 1.}
do it= 1,4 {
        write(6,'(20f6.2)') (q(ix),ix=1,nx)
        d(1) = 0.;      d(nx) = 0.
        do ix= 2, nx-1
                d(ix) = q(ix) + alpha*(q(ix-1)-2.*q(ix)+q(ix+1))
        call rtris( nx, alpha, -alpha, (1.+2.*alpha), -alpha, alpha, d, q)
        }
call exit(0);   end
```

9.3.7. Solving tridiagonal simultaneous equations

Much of the world's scientific computing power gets used up solving **tridiagonal** simultaneous equations. For reference and completeness the algorithm is included here.

Let the simultaneous equations be written as a difference equation

$$a_j\, q_{j+1} \;+\; b_j\, q_j \;+\; c_j\, q_{j-1} \quad = \quad d_j \tag{9.34}$$

Introduce new unknowns e_j and f_j, along with an equation

$$q_j \quad = \quad e_j\, q_{j+1} \;+\; f_j \tag{9.35}$$

Write (9.35) with shifted index:

$$q_{j-1} \quad = \quad e_{j-1}\, q_j \;+\; f_{j-1} \tag{9.36}$$

Insert (9.36) into (9.34)

$$a_j\, q_{j+1} \;+\; b_j\, q_j \;+\; c_j\, (e_{j-1}\, q_j \;+\; f_{j-1}) \quad = \quad d_j \tag{9.37}$$

Now rearrange (9.37) to resemble (9.35)

$$q_j \quad = \quad \frac{-\,a_j}{b_j \;+\; c_j\, e_{j-1}}\; q_{j+1} \;+\; \frac{d_j \;-\; c_j\; f_{j-1}}{b_j \;+\; c_j\, e_{j-1}} \tag{9.38}$$

Compare (9.38) to (9.35) to see recursions for the new unknowns e_j and f_j:

$$e_j = \frac{-a_j}{b_j + c_j e_{j-1}} \tag{9.39}$$

$$f_j = \frac{d_j - c_j f_{j-1}}{b_j + c_j e_{j-1}} \tag{9.40}$$

First a **boundary condition** for the left-hand side must be given. This may involve one or two points. The most general possible end condition is a linear relation like equation (9.35) at $j = 0$, namely, $q_0 = e_0 q_1 + f_0$. Thus, the **boundary condition** must give us both e_0 and f_0. With e_0 and all the a_j, b_j, c_j, we can use (9.39) to compute all the e_j.

On the right-hand boundary we need a **boundary condition**. The general two-point **boundary condition** is

$$c_{n-1} q_{n-1} + e_{\text{right}} q_n = d_n \tag{9.41}$$

Equation (9.41) includes as special cases the zero-value and zero-slope **boundary conditions**. Equation (9.41) can be compared to equation (9.36) at its end.

$$q_{n-1} = e_{n-1} q_n + f_{n-1} \tag{9.42}$$

```
# real tridiagonal equation solver
subroutine rtris( n, endl, a, b, c, endr, d, q)
integer i,        n
real q(n), d(n), a, b, c, den, endl, endr
temporary real f(n), e(n)
e(1) = -a/endl;   f(1) = d(1)/endl
do i= 2, n-1 {
        den = b+c*e(i-1);   e(i) = -a/den;   f(i) = (d(i)-c*f(i-1))/den
        }
q(n) = (d(n)-c*f(n-1)) / (endr+c*e(n-1))
do i= n-1, 1, -1
        q(i) = e(i) * q(i+1) + f(i)
return;         end
```

Back

Both q_n and q_{n-1} are unknown, but in equations (9.41) and (9.42) we have two equations, so the solution is easy. The final step is to take the value of q_n and use it in (9.36) to compute q_{n-1}, q_{n-2}, q_{n-3}, etc. The subroutine `rtris()` solves equation (9.33) for q where `n=5`, `endl`$= e_{\text{left}}$, `endr`$= e_{\text{right}}$, `a=c`$= -\alpha$, and `b`$= 1-2\alpha$.

rtris

If you wish to squeeze every last ounce of power from your computer, note some facts about this algorithm. (1) The calculation of e_j depends on the *medium* through a_j, b_j, c_j, but it does not depend on the *solution* q_j (even through d_j). This means that it may be possible to save and reuse e_j. (2) In many computers, division is much slower than multiplication. Thus, the divisor in (9.39) or (9.40) can be inverted once (and perhaps stored for reuse).

9.3.8. Finite-differencing in the time domain

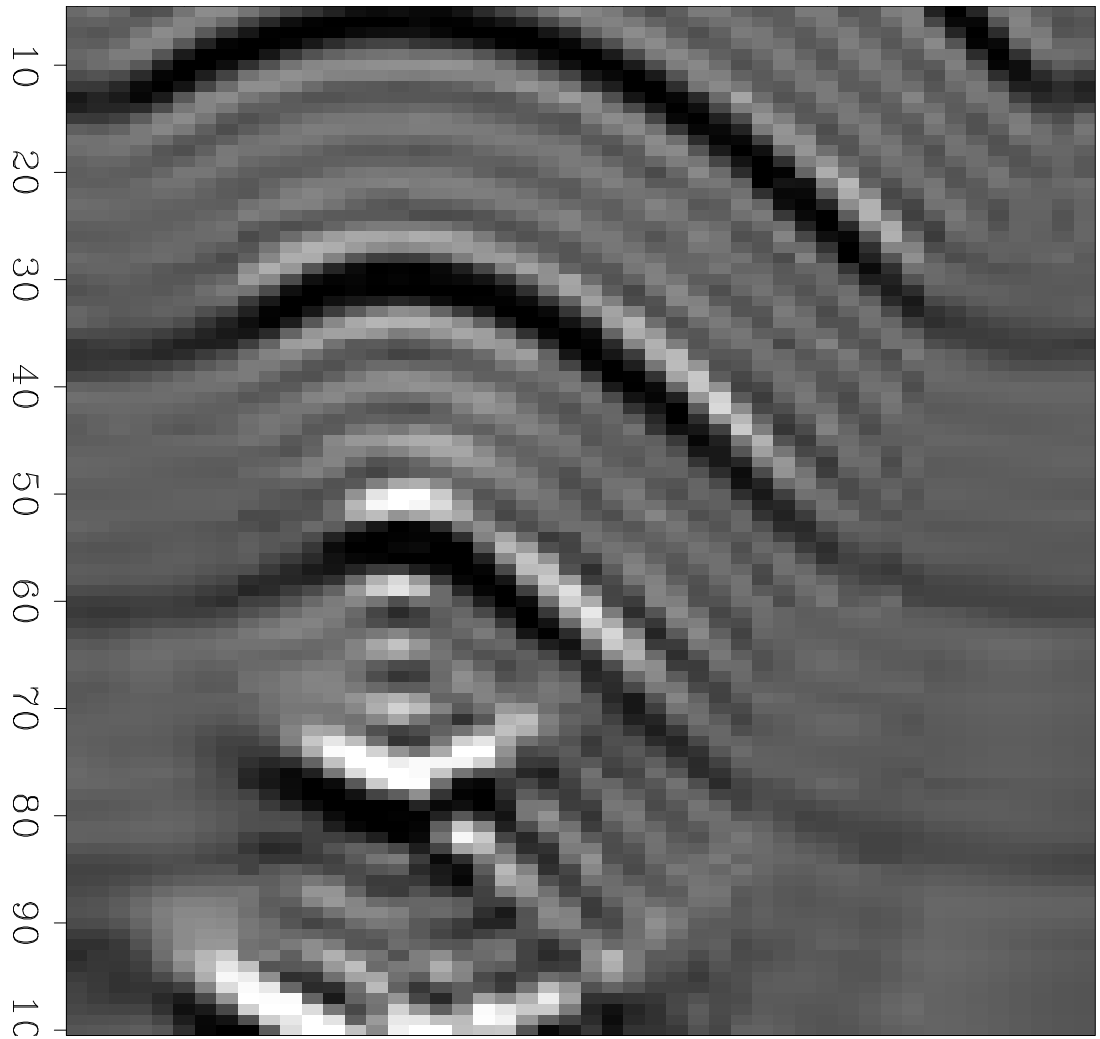
IEI develops time-domain finite differencing methods. Since the earth velocity is unvarying in time, a "basics only" book such as this omits this topic since you can, in principle, accomplish the same goals in the ω-domain. There are some applications, however, that give rise to time-variable coefficients in their partial differential

equations. Recursive dip filtering is one application. Residual migration is another. Some formulations of DMO are another.

9.4. WAVEMOVIE PROGRAM

Here we see solutions to exercises stated in figure captions. The problems and solutions were worked over by former teaching assistants. (Lynn, Gonzalez, JFC, Hale, Li, Karrenbach, Fomel). The various figures are all variations of the computer subroutine `wavemovie()`. It makes a **movie** of a sum of monochromatic waves. As it stands it will produce a **movie** (three-dimensional matrix) of waves propagating through a focus. The whole process from compilation through computation to finally viewing the film loop takes a few seconds. A sample frame of the **movie** is in Figure 9.2. It shows a snapshot of the (x,z)-plane. Collapsing spherical waves enter from the top, go through a focus and then expand again. Notice that the wavefield is small but not zero in the region of geometrical shadow. In the shadow region you see waves that appear to be circles emanating from point sources at the top corners. Notice that the amplitudes of expanding spherical waves drop off with distance and collapsing spherical waves grow towards the focus. We will study the program that

Figure 9.2: First frame of movie generated by `wave-movie()`. (Press button for movie.) fdm-Mfocus1590 [ER,M]



made this figure and see many features of waves and mathematics.

9.4.1. Earth surface boundary condition

The program that created Figure 9.2 begins with an initial condition along the top boundary, and then this initial wavefield is extrapolated downward. So, the first question is: what is the mathematical function of x that describes a collapsing spherical (actually cylindrical) wave? An expanding spherical wave has an equation $\exp[-i\omega(t-r/v)]$, where the radial distance is $r=\sqrt{(x-x_0)^2+(z-z_0)^2}$ from the source. For a collapsing spherical wave we need $\exp[-i\omega(t+r/v)]$. Parenthetically, I'll add that the theoretical solutions are not really these, but something more like these divided by $\sqrt{r}$; actually they should be a **Hankel function**s, but the picture is hardly different when the exact initial condition is used. If you have been following this analysis, you should have little difficulty changing the initial conditions in the program to create the downgoing plane wave shown in Figure 9.3. Notice the weakened waves in the zone of theoretical shadow that appear to arise from a point source on the top corner of the plot. You have probably learned in physics classes of "standing waves". This is what you will see near the reflecting side boundary if

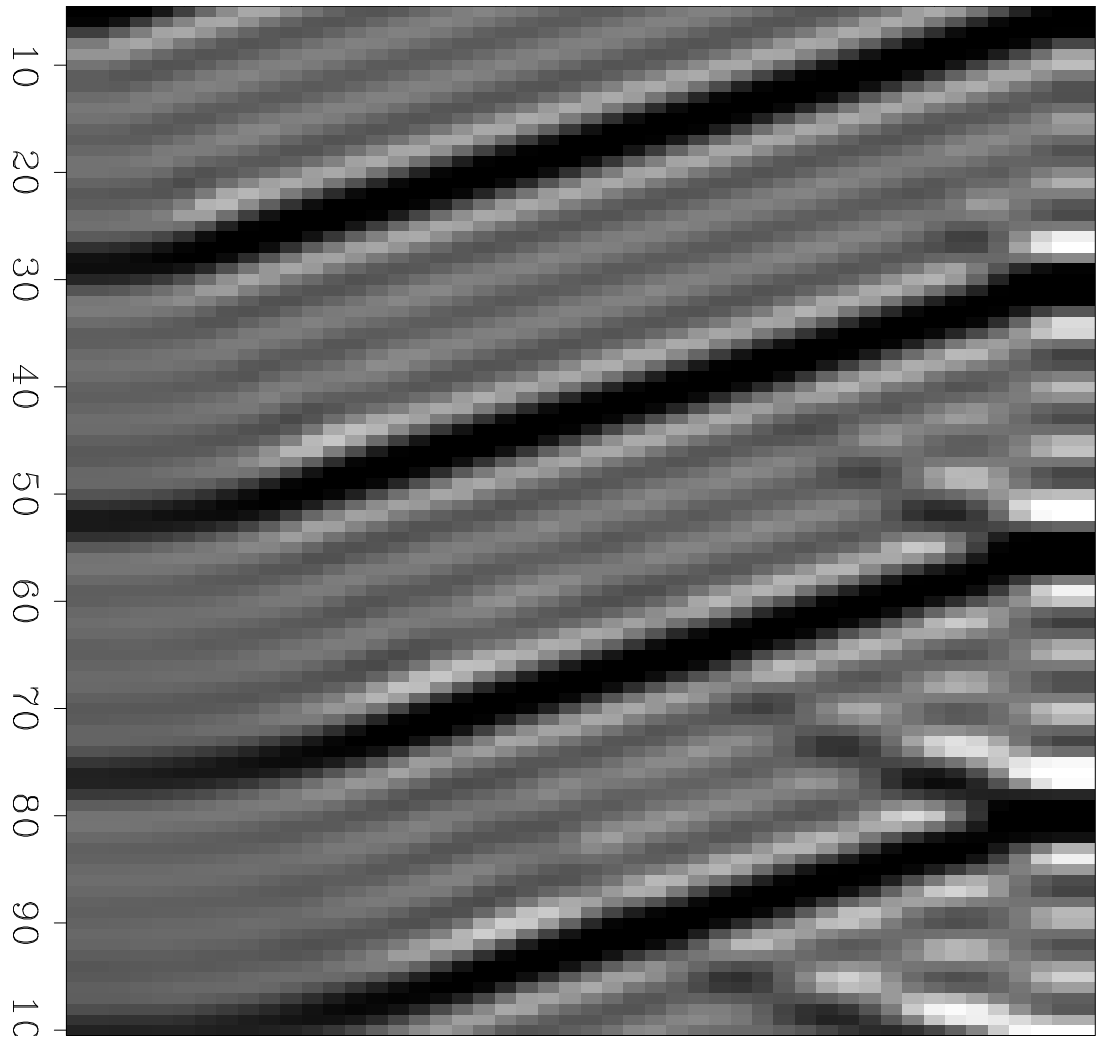


Figure 9.3: Specify program changes that give an initial plane wave propagating downward at an angle of 15° to the right of vertical. (Movie) fdm-Mdipplane90 [ER,M]

you recompute the plot with a single frequency `nw=1`. Then the plot will acquire a "checkerboard" appearance near the reflecting boundary. Even this figure with `nw=4` shows the tendency.

9.4.2. Frames changing with time

For a film loop to make sense to a viewer, the subject of the **movie** must be periodic, and organized so that the last frame leads naturally into the first. In the **movie** created by `wavemovie()` there is a parameter `lambda` that controls the basic repetition rate of wave pulses fired onto the screen from the top. When a wavelet travels one-quarter of the way down the frame, another is sent in. This is defined by the line

$$\texttt{lambda = nz * dz / 4} \quad = \quad \frac{N_z\,\Delta z}{4}$$

Take any point in (x,z)-space. The signal there will be a superposition of sinusoids of various frequencies, ω_j. We can choose what frequencies we will use in the calculation and what amplitudes and phases we will attach to the initial conditions at those frequencies. Here we will simply take uniformly spaced sinusoids of unit amplitude and no phase. The `nw` frequencies are $\omega_j = \Delta\omega,\ 2\,\Delta\omega,\ \ldots,\ \texttt{nw}\,\Delta\omega$. The

lowest frequency `dw` = $\Delta\omega$ must be inversely proportional to the wavelength `lambda` = λ

$$\texttt{dw = v * pi2 / lambda} = \frac{2\pi v}{\lambda}$$

Finally, the time duration of the film loop must equal the period of the lowest-frequency sinusoid

$$N_t \Delta t = \frac{2\pi}{\Delta\omega}$$

This latter equation defines the time interval on the line

```
dt = pi2 / ( nt * dw )
```

If you use more frequencies, you might like the result better because the wave pulses will be shorter, and the number of wavelengths between the pulses will increase. Thus the quiet zones between the pulses will get quieter. The frequency components can be weighted differently—but this becomes a digression into simple Fourier analysis. wavemovie

```
# from par:  integer n3:nt=12, n2:nx=48, n1:nz=96,  nw=2, nlam=4
# from par:  real dx=2, dz=1, v=1
#
subroutine wavemovie( nt, nx, nz, nw, nlam, dx,dz,v, p, cd, q)
integer it,nt,ix,nx,iz,nz,iw,nw, nlam
real dx,dz,v, phase,pi2,z0,x0,dt,dw,lambda,w,wov,x, p(nz,nx,nt)
complex aa,a,b,c,cshift, cd(nx),q(nx)
lambda=nz*dz/nlam;  pi2=2.*3.141592; dw=v*pi2/lambda;  dt=pi2/(nt*dw)
x0 = nx*dx/3; z0 = nz*dz/3
call null( p, nz*nx*nt)
do iw = 1,nw {                                  # superimpose nw frequencies
        w =  iw*dw;   wov = w/v                 # frequency / velocity
        do ix = 1,nx {                          # initial conditions for a
                x = ix*dx-x0;                   # collapsing spherical wave
                phase = -wov*sqrt( z0**2+x**2)
                q(ix) = cexp( cmplx( 0.,phase))
                }
        aa = (0.,1.)*dz/(4.*dx**2*wov)          # tridiagonal matrix coefficients
        a = -aa;      b = 1.+2.*aa;      c = -aa
        do iz = 1,nz {                          # extrapolation in depth
                do ix = 2,nx-1                  # diffraction term
                        cd(ix) = aa*q(ix+1) + (1.-2.*aa)*q(ix) + aa*q(ix-1)
                cd(1) = 0.;      cd(nx) = 0.
                call ctris( nx,-a,a,b,c,-c,cd,q)
                        # Solves complex tridiagonal equations
                cshift = cexp( cmplx( 0.,wov*dz))
                do ix = 1,nx                    # shifting term
                        q(ix) = q(ix) * cshift
                do it=1,nt {                    # evolution in time
                        cshift = cexp( cmplx( 0.,-w*it*dt))
                        do ix = 1,nx
                                p(iz,ix,it) = p(iz,ix,it) + q(ix)*cshift
                        }
                }
        }
return;         end
```

Back

9.4.3. Internals of the film-loop program

The differential equation solved by the program is equation (9.5), copied here as

$$\frac{\partial P}{\partial z} \quad = \quad \frac{i\,\omega}{v(x,z)}\;P \;+\; \frac{v}{-i\,\omega 2}\;\frac{\partial^2 P}{\partial x^2} \tag{9.43}$$

For each Δz-step the calculation is done in two stages. The first stage is to solve

$$\frac{\partial P}{\partial z} \quad = \quad \frac{v}{-i\,\omega 2}\;\frac{\partial^2 P}{\partial x^2} \tag{9.44}$$

Using the **Crank-Nicolson** differencing method this becomes

$$\frac{p_{z+1}^{x} - p_{z}^{x}}{\Delta z} \quad = \quad \frac{v}{-i\omega 2}\left(\frac{p_{z}^{x+1} - 2p_{z}^{x} + p_{z}^{x-1}}{2\,\Delta x^2} + \frac{p_{z+1}^{x+1} - 2p_{z+1}^{x} + p_{z+1}^{x-1}}{2\,\Delta x^2}\right) \tag{9.45}$$

Absorb all the constants into one and define

$$\alpha \quad = \quad \frac{v\,\Delta z}{-i\,\omega 4\,\Delta x^2} \tag{9.46}$$

getting

$$p_{z+1}^{x} - p_{z}^{x} \quad = \quad \alpha \left[(p_{z}^{x+1} - 2\,p_{z}^{x} + p_{z}^{x-1}) + (p_{z+1}^{x+1} - 2\,p_{z+1}^{x} + p_{z+1}^{x-1}) \right] \tag{9.47}$$

Bring the unknowns to the left:

$$-\alpha p_{z+1}^{x+1} + (1+2\alpha) p_{z+1}^{x} - \alpha p_{z+1}^{x-1} \quad = \quad \alpha p_{z}^{x+1} + (1-2\alpha) p_{z}^{x} + \alpha p_{z}^{x-1} \tag{9.48}$$

We will solve this as we solved equations (9.32) and (9.33). The second stage is to solve the equation

$$\frac{\partial P}{\partial z} \quad = \quad \frac{i\,\omega}{v}\ P \tag{9.49}$$

analytically by

$$P(z+\Delta z) \quad = \quad P(z)\, e^{i\,\Delta z\ \omega/v} \tag{9.50}$$

By alternating between (9.48) and (9.50), which are derived from (9.44) and (9.49), the program solves (9.43) by a **splitting** method. The program uses the **tridiagonal** solver discussed earlier, like subroutine `rtris()` /prog:rtris except that version needed here, `ctris()`, has all the real variables declared complex.

Figure 9.4 shows a change of initial conditions where the incoming wave on the top frame is defined to be an impulse, namely, $p(x, z=0) = (\cdots, 0, 0, 1, 0, 0, \cdots)$.

The result is alarmingly noisy. What is happening is that for any frequencies anywhere near the Nyquist frequency, the *difference* equation departs from the *differential* equation that it should mimic. This problem is addressed, analyzed, and ameliorated in IEI. For now, the best thing to do is to avoid sharp corners in the initial wave field.

9.4.4. Side-boundary analysis

In geophysics, we usually wish the side-boundary question did not arise. The only real reason for side boundaries is that either our survey or our processing activity is necessarily limited in extent. Given that side boundaries are inevitable, we must think about them. The subroutine `wavemovie()` included zero-slope boundary conditions. This type of boundary treatment resulted from taking

```
d(1) = 0.  ; d(nx) = 0.
```

and in the call to `ctris` taking

```
endl = - a ; endr = - c
```

A quick way to get zero-value side-boundary conditions is to take

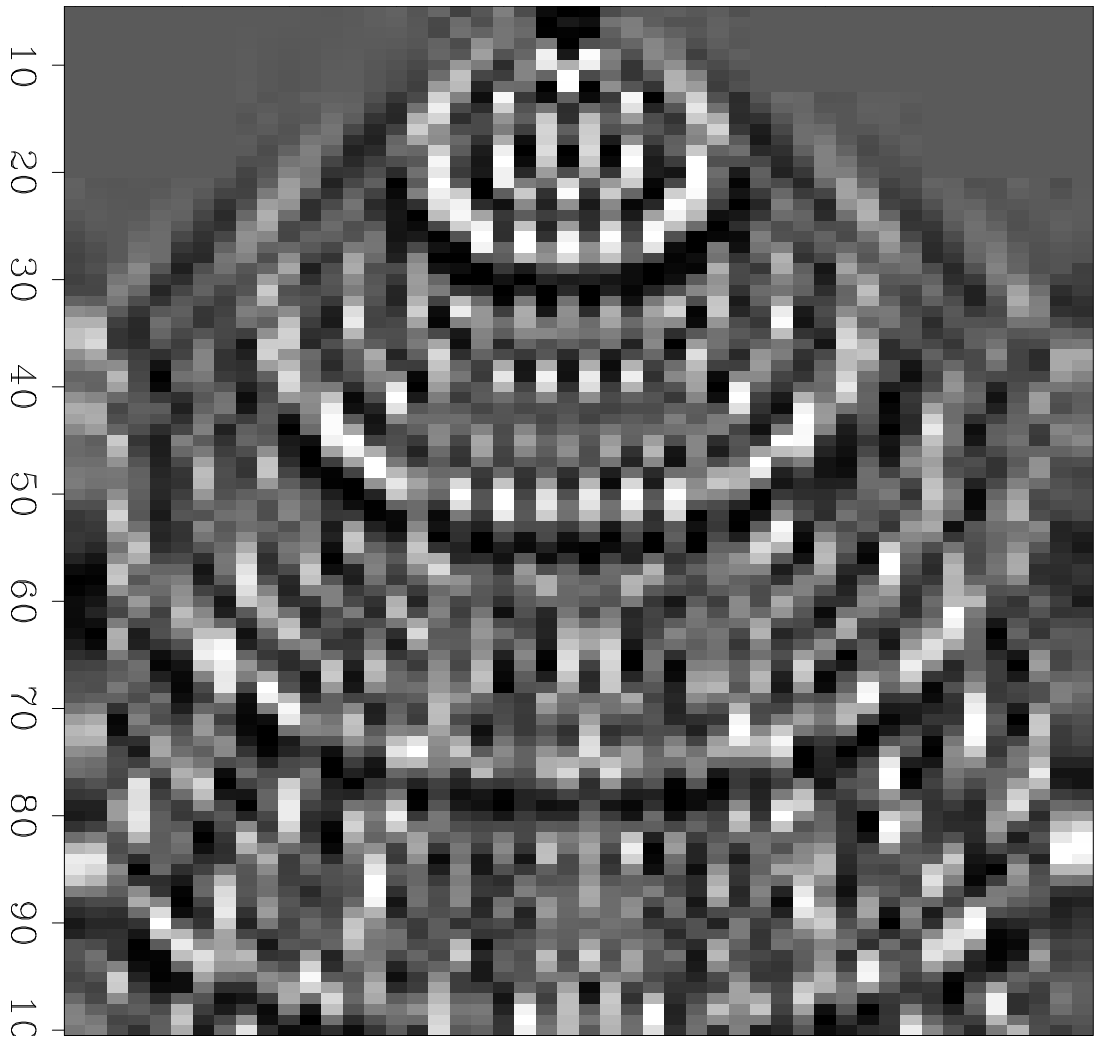


Figure 9.4: Observe and describe various computational artifacts by testing the program using a point source at (x,z) = `(xmax/2,0)`. Such a source is rich in the high spatial frequencies for which difference equations do not mimic their differential counterparts. (Movie) fdm-Mcompart90 [ER,M]

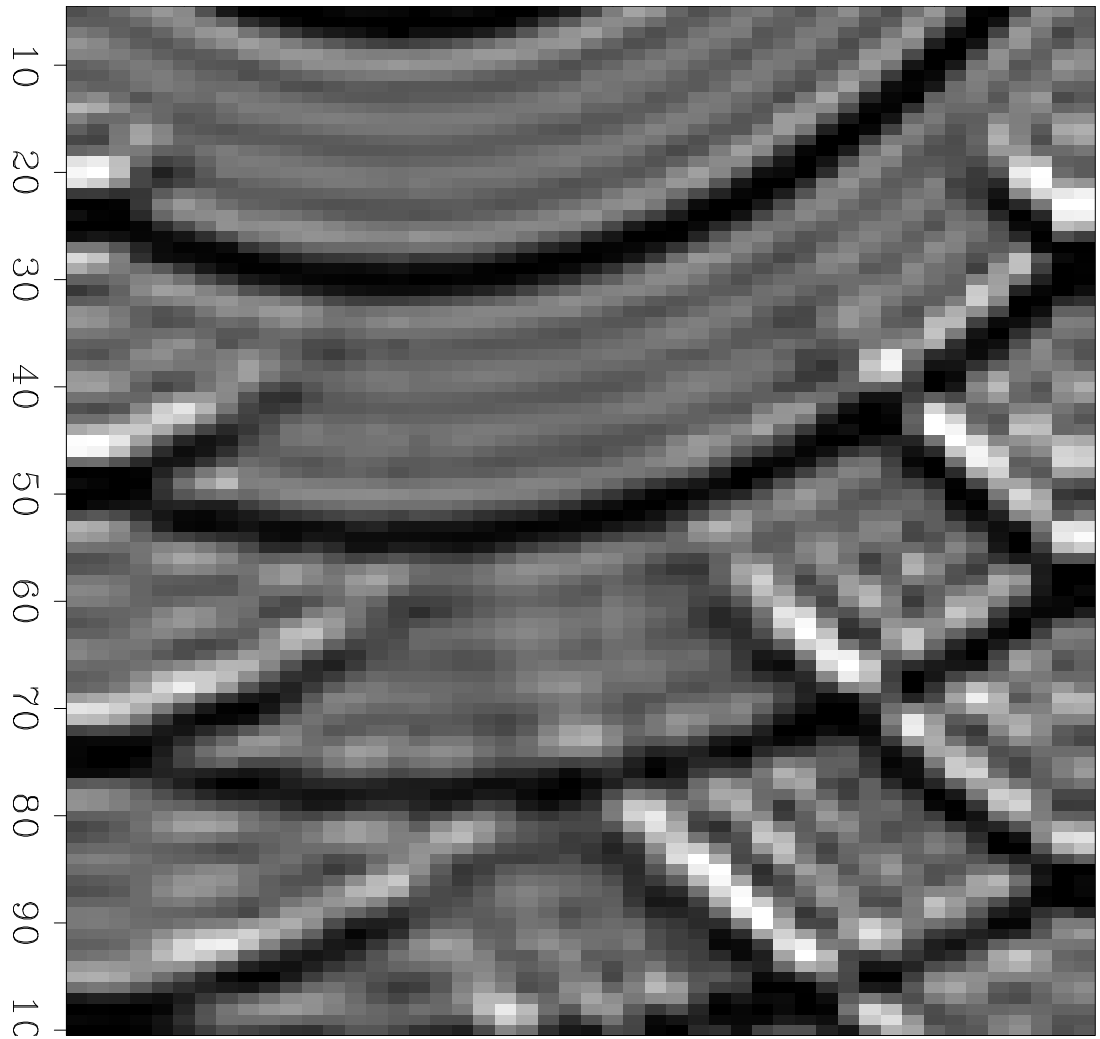


Figure 9.5: Given that the domain of computation is $0 \leq x \leq$ `xmax` and $0 \leq z \leq$`zmax`, how would you modify the initial conditions at $z = 0$ to simulate a point source at $(x,z) =$ `(xmax/3, -zmax/2)`? (Movie) fdm-Mexpandsphere90 [ER,M]

$$\texttt{endl = endr =} \quad 10^{30} \quad \approx \quad \infty$$

Compare the side-boundary behavior of Figures 9.5 and 9.6.

The zero slope boundary condition is explicitly visible as identical signal on the two end columns. Likewise, the zero-value side boundary condition has a column of zero-valued signal on each side.

9.4.5. Lateral velocity variation

Lateral velocity variation $v = v(x)$ has not been included in the program, but it is not difficult to install. It enters in two places. It enters first in equation (9.50). If the wavefield is such that k_x is small enough, then equation (9.50) is the only place it is needed. Second, it enters in the **tridiagonal** coefficients through the v in equation (9.46). The so-called thin-lens approximation of optics seems to amount to including the equation (9.50) part only. An example of **lateral velocity variation** is in Figure 9.7.

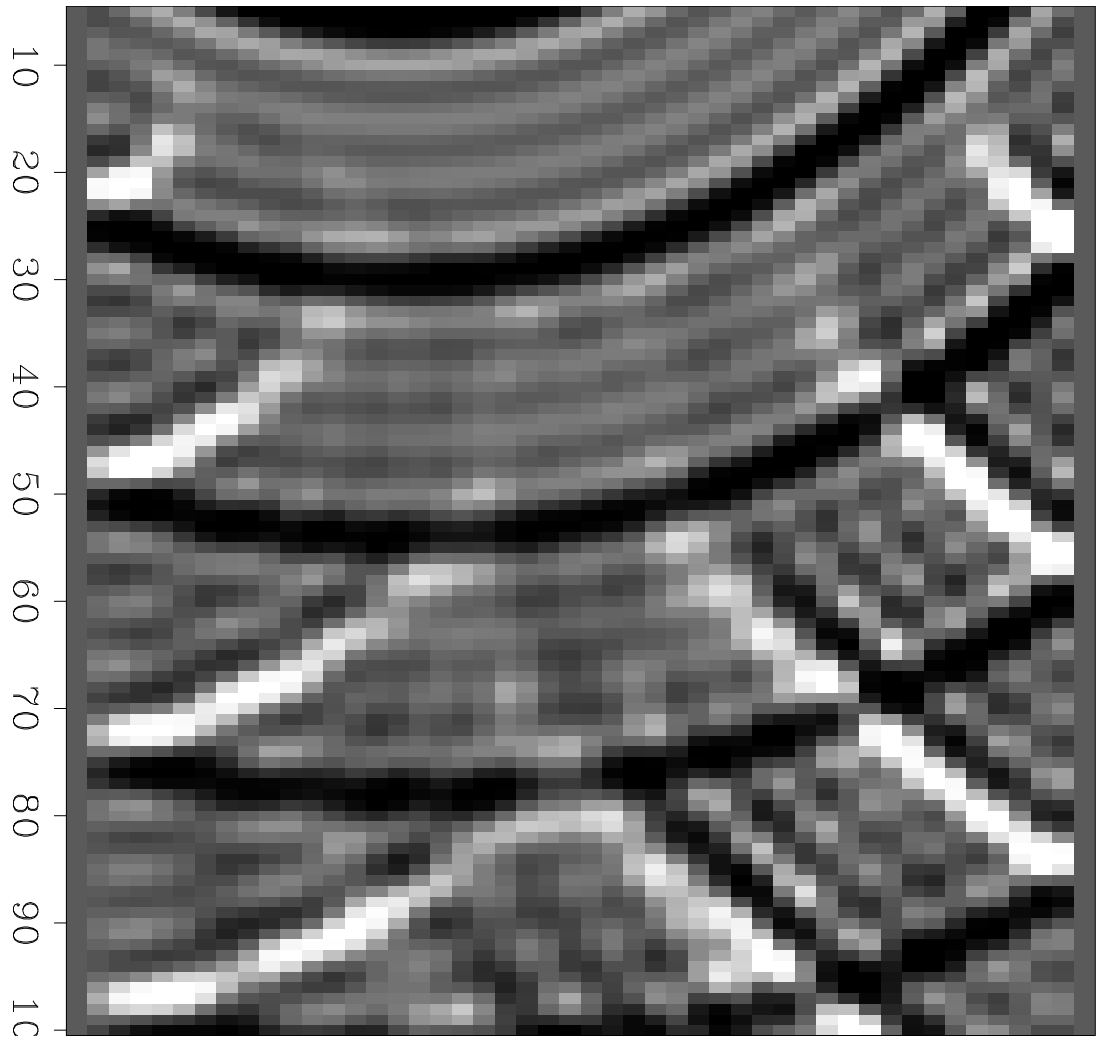


Figure 9.6: Modify the program so that zero-slope side boundaries are replaced by zero-value side boundaries. (Movie) fdm-Mzeroslope90 [ER,M]

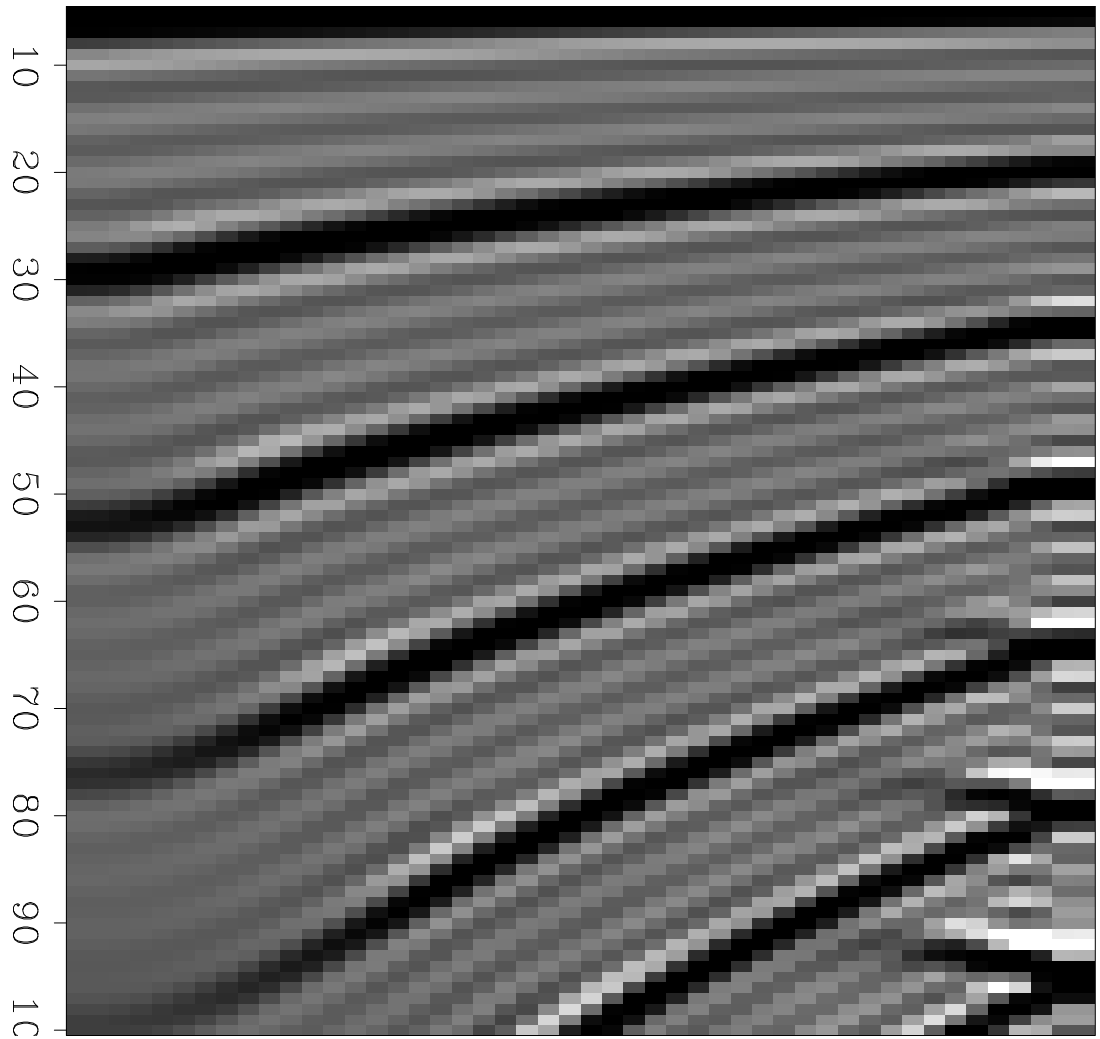


Figure 9.7: Make changes to the program to include a **thin-lens term** with a lateral velocity change of 40% across the frame produced by a constant slowness gradient. Identify other parts of the program which are affected by lateral velocity variation. You need not make these other changes. Why are they expected to be small? (Movie) fdm-Mlateralvel90 [ER,M]

9.4.6. Migration in (omega,x)-space

The migration program is similar to the film loop program. But there are some differences. The film loop program has "do loops" nested four deep. It produces results for many values of t. Migration requires a value only at $t = 0$. So one loop is saved, which means that for the same amount of computer time, the space volume can be increased. Unfortunately, loss of a loop seems also to mean loss of a **movie**. With ω-domain migration, it seems that the only interesting thing to view is the input and the output.

The input for this process will probably be field data, unlike for the film loop **movie**, so there will not be an analytic representation in the ω-domain. The input will be in the time domain and will have to be Fourier transformed. The beginning of the program defines some pulses to simulate field data. The pulses are broadened impulses and should migrate to approximate semicircles. Exact impulses were not used because the departure of difference operators from differential operators would make a noisy mess.

Next the program Fourier transforms the pseudodata from the time domain into the ω-frequency domain.

Then comes the downward continuation of each frequency. This is a loop on

depth z and on frequency ω. Either of these loops may be on the inside. The choice can be made for machine-dependent efficiency.

For migration an equation for upcoming waves is required, unlike the downgoing wave equation required for the film loop program. Change the sign of the z-axis in equation (9.43). This affects the sign of `aa` and the sign of the phase of `cshift`.

Another difference with the film loop program is that the input now has a time axis whereas the output is still a depth axis. It is customary and convenient to reorganize the calculation to plot traveltime depth, instead of depth, making the vertical axes on both input and output the same. Using $\tau = z/v$, equivalently $d\tau/dz = 1/v$, the chain rule gives

$$\frac{\partial}{\partial z} \quad = \quad \frac{\partial \tau}{\partial z}\,\frac{\partial}{\partial \tau} \quad = \quad \frac{1}{v}\,\frac{\partial}{\partial \tau} \tag{9.51}$$

Substitution into (9.43) gives

$$\frac{\partial P}{\partial \tau} \quad = \quad -i\,\omega\,P \;-\; \frac{v^2}{-i\,\omega 2}\,\frac{\partial^2 P}{\partial x^2} \tag{9.52}$$

In the program, the time sample size `dt` $= \Delta t$ and the traveltime depth sample `dtau` $= \Delta\tau$ are taken to be unity, so the maximum frequency is the Nyquist. Notice

that the frequency loop covers only the negative frequency axis. The positive frequencies serve only to keep the time function real, a task that is more quickly done by simply taking the real part. A program listing follows

```
#%  Migration in the (omega,x,z)-domain
program kjartjac{
real p(48,64), pi, alpha, dt, dtau, dw, w0, omega
complex cp(48,64), cd(48), ce(48), cf(48), aa, a, b, c, cshift
integer ix, nx, iz, nz, iw, nw, it, nt, esize
nt= 64;   nz= nt;    nx= 48;      pi= 3.141592
dt= 1.;   dtau= 1.;  w0=-pi/dt;   dw= 2*pi/(dt*nt);    nw= nt/2;
alpha = .25                                  #  alpha = v*v*dtau/(4*dx*dx)



do iz= 1, nz { do ix=1,nx { p(ix,iz) = 0.;    cp(ix,iz)=0. }}
do it= nt/3, nt, nt/4{
# Broadened impulse source
        do ix= 1, 4                            { cp(ix,it) = (5.-ix);     cp(ix,it+1)
ix) }}
call ft2axis( 0, 1., nx,nt, cp)
do iz= 1, nz {
do iw= 2, nw {    omega = w0 + dw*(iw-1)
        aa = - alpha /( (0.,-1.)*omega )
        a = -aa;      b = 1.+2.*aa;      c = -aa
        do ix= 2, nx-1
                cd(ix) = aa*cp(ix+1,iw) + (1.-2.*aa)*cp(ix,iw) + aa*cp(ix-1,iw)
        cd(1) = 0.;      cd(nx) = 0.
        call ctris( nx, -a, a, b, c, -c, cd, cp(1,iw))
        cshift = cexp( cmplx( 0.,-omega*dtau))
```

```
        do ix= 1, nx
                cp(ix,iw) = cp(ix,iw) * cshift
        do ix= 1, nx
                p(ix,iz) = p(ix,iz)+cp(ix,iw)   # p(t=0) = Sum P(omega)
        }}
esize=4
to history:     integer n1:nx, n2:nz, esize
call srite( 'out', p, nx*nz*4 )
call hclose()
}
```

The output of the program is shown in Figure 9.8. Mainly, you see semicircle approximations. There are also some artifacts at late time that may be ω-domain wraparounds. The input pulses were apparently sufficiently broad-banded in dip that the figure provides a preview of the fact, to be proved later, that the actual semicircle approximation is an ellipse going through the origin.

Notice that the waveform of the original pulses was a symmetric function of time, whereas the semicircles exhibit a waveform that is neither symmetric nor antisymmetric, but is a 45° phase-shifted pulse. Waves from a point in a three-dimensional world would have a phase shift of 90°. Waves from a two-dimensional exploding reflector in a three-dimensional world have the 45° phase shift.

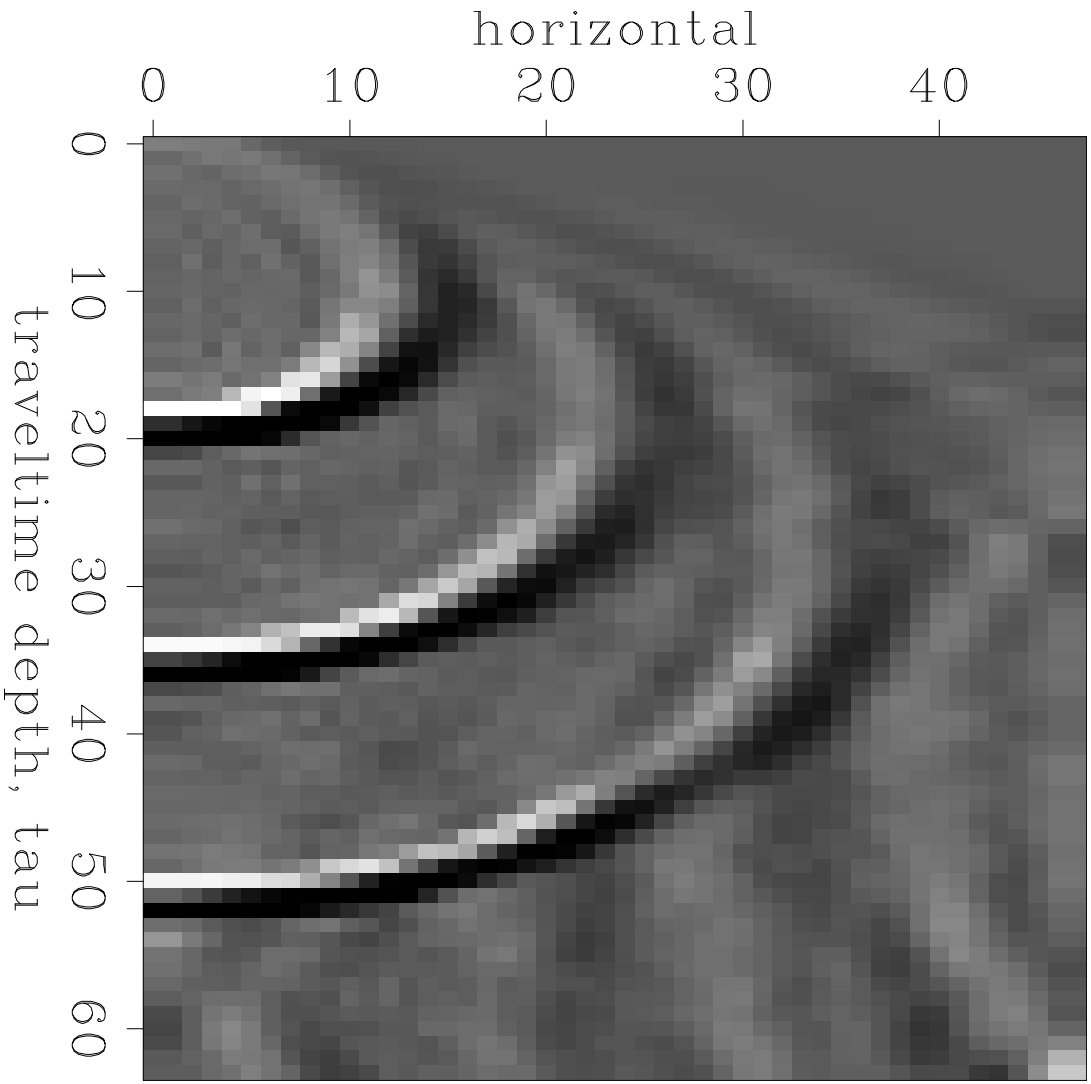


Figure 9.8: Output of the program `kjartjac`: semicircle approximations. fdm-kjartjac [ER]

9.5. HIGHER ANGLE ACCURACY

A wave-extrapolation equation is an expression for the derivative of a wavefield (usually in the depth z direction). When the wavefield and its derivative are known, extrapolation can proceed by various numerical representations of

$$P(z+\Delta z) \quad = \quad P(z) + \Delta z \, \frac{dP}{dz} \tag{9.53}$$

Extrapolation is moving information from z to $z+\Delta z$ and what we need to do it is a way to find dP/dz. Two theoretical methods for finding dP/dz are the original *transformation* method and the newer *dispersion-relation* method.

9.5.1. Another way to the parabolic wave equation

Here we review the historic "transformation method" of deriving the parabolic wave equation.

A vertically downgoing plane wave is represented mathematically by the equation

$$P(t,x,z) \quad = \quad P_0 \; e^{-i\omega(t-z/v)} \tag{9.54}$$

In this expression, P_0 is absolutely constant. A small departure from vertical incidence can be modeled by replacing the constant P_0 with something, say, $Q(x,z)$, which is not strictly constant but varies slowly.

$$P(t,x,z) \quad = \quad Q(x,z)\ e^{-i\omega(t-z/v)} \tag{9.55}$$

Inserting (9.55) into the scalar wave equation $P_{xx} + P_{zz} = P_{tt}/v^2$ yields

$$\begin{aligned} \frac{\partial^2}{\partial x^2}\ Q\ + \left(\frac{i\omega}{v} + \frac{\partial}{\partial z}\right)^2 Q \quad &= \quad -\frac{\omega^2}{v^2}\ Q \\ \frac{\partial^2 Q}{\partial x^2}\ + \ \frac{2i\omega}{v}\ \frac{\partial Q}{\partial z}\ + \ \frac{\partial^2 Q}{\partial z^2} \quad &= \quad 0 \end{aligned} \tag{9.56}$$

The wave equation has been reexpressed in terms of $Q(x,z)$. So far no approximations have been made. To require the wavefield to be near to a plane wave, $Q(x,z)$ must be near to a constant. The appropriate means (which caused some controversy when it was first introduced) is to drop the highest depth derivative of Q, namely, Q_{zz}. This leaves us with the ***parabolic wave equation***

$$\frac{\partial Q}{\partial z} \quad = \quad \frac{v}{-2i\omega}\ \frac{\partial^2 Q}{\partial x^2} \tag{9.57}$$

I called equation (9.57) the 15° equation. After using it for about a year I discovered a way to improve on it by estimating the dropped ∂_{zz} term. Differentiate equation (9.57) with respect to z and substitute the result back into equation (9.56) getting

$$\frac{\partial^2 Q}{\partial x^2} + \frac{2i\omega}{v}\frac{\partial Q}{\partial z} + \frac{v}{-2i\omega}\frac{\partial^3 Q}{\partial z \partial x^2} = 0 \qquad (9.58)$$

I named equation (9.58) the 45° migration equation. It is first order in ∂_z, so it requires only a single surface boundary condition, however, downward continuation will require something more complicated than equation (9.53).

The above approach, the transformation approach, was and is very useful. But people were confused by the dropping and estimating of the ∂_{zz} derivative, and a philosophically more pleasing approach was invented by Francis **Muir**, a way of getting equations to extrapolate waves at wider angles by fitting the dispersion relation of a semicircle by polynomial ratios.

9.5.2. Muir square-root expansion

Muir's method of finding wave extrapolators seeks polynomial ratio approximations to a square-root dispersion relation. Then fractions are cleared and the approximate dispersion relation is inverse transformed into a differential equation. Recall equation (9.1)

$$k_z \quad = \quad \frac{\omega}{v} \sqrt{1 - \frac{v^2 \, k_x^2}{\omega^2}} \tag{9.59}$$

To inverse transform the z-axis we only need to recognize that ik_z corresponds to $\partial/\partial z$. Getting into the x-domain, however, is not simply a matter of substituting a second x derivative for k_x^2. The problem is the meaning of the **square root** of a differential operator. The square root of a differential operator is not defined in undergraduate calculus courses and there is no straightforward finite difference representation. The square root becomes meaningful only when it is regarded as some kind of truncated series expansion. It is shown in IEI that the Taylor series is a poor choice. Francis **Muir** showed that my original 15° and 45° methods were just

truncations of a continued fraction expansion. To see this, define

$$X = \frac{vk_x}{\omega} \quad \text{and} \quad R = \frac{vk_z}{\omega} \tag{9.60}$$

With the definitions (9.60) equation (9.59) is more compactly written as

$$R = \sqrt{1 - X^2} \tag{9.61}$$

which you recognize as meaning that cosine is the square root of one minus sine squared. The desired polynomial ratio of order n will be denoted R_n, and it will be determined by the recurrence

$$R_{n+1} = 1 - \frac{X^2}{1 + R_n} \tag{9.62}$$

The recurrence is a guess that we verify by seeing what it converges to (if it converges). Set $n = \infty$ in (9.62) and solve

$$\begin{aligned} R_\infty &= 1 - \frac{X^2}{1 + R_\infty} \\ R_\infty(1 + R_\infty) &= 1 + R_\infty - X^2 \end{aligned}$$

$$R^2 \quad = \quad 1 - X^2 \tag{9.63}$$

The square root of (9.63) gives the required expression (9.61). Geometrically, (9.63) says that the cosine squared of the incident angle equals one minus the sine squared and truncating the expansion leads to angle errors. **Muir** said, and you can verify, that his recurrence relationship formalizes what I was doing by re-estimating the ∂_{zz} term. Although it is pleasing to think of large values of n, in real life only the low-order terms in the expansion are used. The first four truncations of **Muir**'s continued fraction expansion beginning from $R_0 = 1$ are

$$\begin{aligned} 5^\circ: \quad & R_0 = 1 \\ 15^\circ: \quad & R_1 = 1 - \frac{X^2}{2} \\ 45^\circ: \quad & R_2 = 1 - \frac{X^2}{2 - \frac{X^2}{2}} \end{aligned} \tag{9.64}$$

$$60^\circ: \qquad R_3 \quad = \quad 1 - \cfrac{X^2}{2 - \cfrac{X^2}{2 - \cfrac{X^2}{2}}}$$

For various historical reasons, the equations in the above equations are often referred to as the 5°, 15°, and 45° equations, respectively, the names giving a reasonable qualitative (but poor quantitative) guide to the range of angles that are adequately handled. A trade-off between complexity and accuracy frequently dictates choice of the 45° equation. It then turns out that a slightly wider range of angles can be accommodated if the recurrence is begun with something like $R_0 = \cos 45^\circ$. Figure 9.9 shows some plots.

9.5.3. Dispersion relations

Substituting the definitions (9.60) into equation (9.65) et. seq. gives dispersion relationships for comparison to the exact expression (9.59).

$$5^\circ: \qquad k_z \quad = \quad \frac{\omega}{v} \tag{9.65}$$

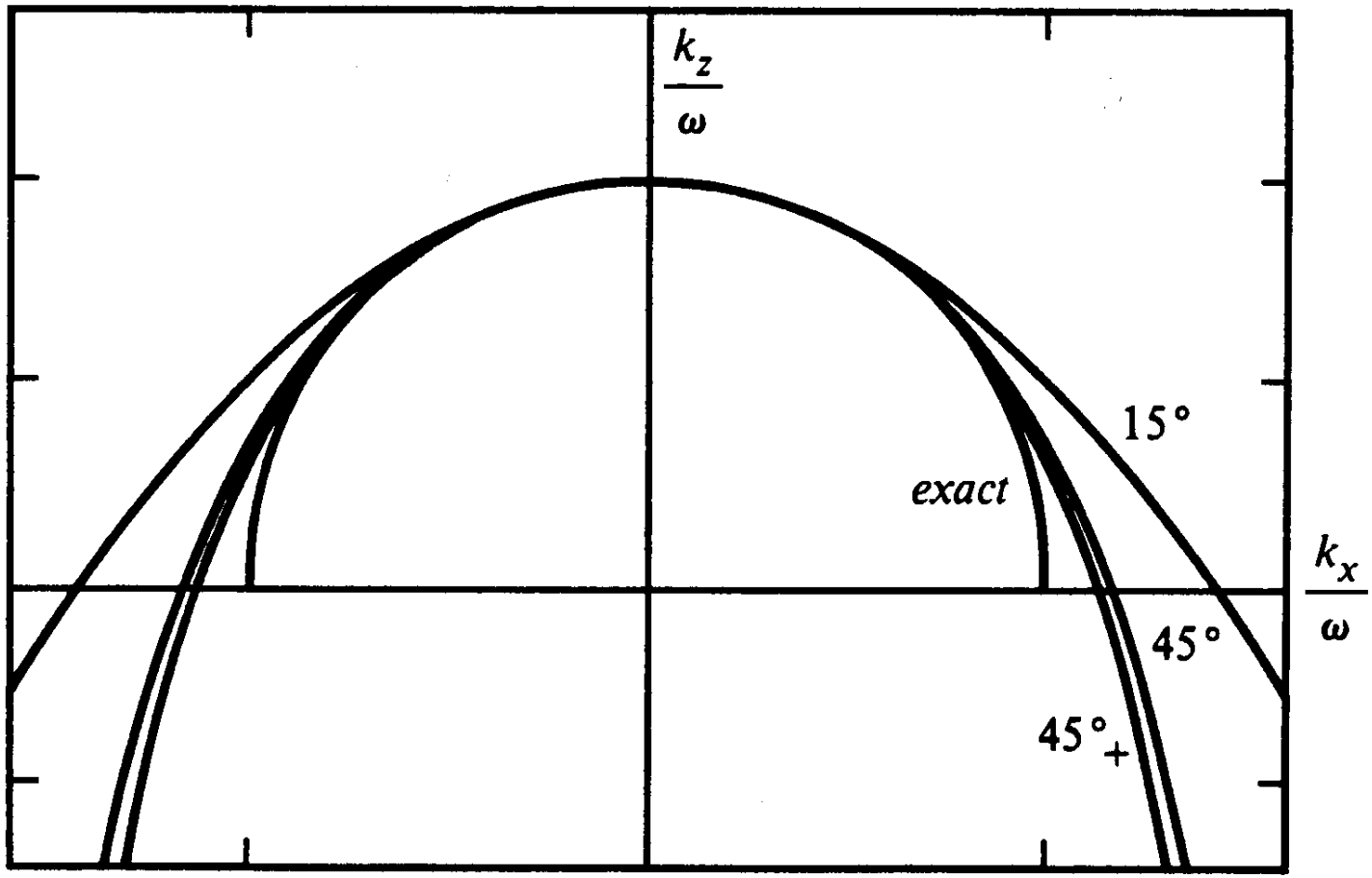


Figure 9.9: Dispersion relation of equation (9.65). The curve labeled 45°_+ was constructed with $R_0 = \cos 45^\circ$. It fits exactly at 0° and 45°. fdm-disper [NR]

$$15^\circ: \qquad k_z = \frac{\omega}{v} - \frac{v k_x^2}{2\omega}$$

$$45^\circ: \qquad k_z = \frac{\omega}{v} - \frac{k_x^2}{2\dfrac{\omega}{v} - \dfrac{v k_x^2}{2\omega}}$$

Identification of ik_z with $\partial/\partial z$ converts the dispersion relations (9.65) into the differential equations

$$5^\circ: \qquad \frac{\partial P}{\partial z} = i\left(\frac{\omega}{v}\right) P \tag{9.66}$$

$$15^\circ: \qquad \frac{\partial P}{\partial z} = i\left(\frac{\omega}{v} - \frac{v k_x^2}{2\omega}\right) P$$

$$45^\circ: \qquad \frac{\partial P}{\partial z} = i\left(\frac{\omega}{v} - \frac{k_x^2}{2\dfrac{\omega}{v} - \dfrac{v k_x^2}{2\omega}}\right) P$$

which are extrapolation equations for when velocity depends only on depth.

The differential equations above in Table 9.4 were based on a dispersion relation that in turn was based on an assumption of constant velocity. Surprisingly, these equations also have validity and great utility when the velocity is depth-variable, $v = v(z)$. The limitation is that the velocity be constant over each depth "slab" of width Δz over which the downward-continuation is carried out.

9.5.4. The xxz derivative

The 45° diffraction equation differs from the 15° equation by the inclusion of a $\partial^3/\partial x^2 \partial z$ -derivative. Luckily this derivative fits on the six-point differencing star

$$\frac{1}{\Delta x^2 \, \Delta z}$$

−1	2	−1
1	−2	1

So other than modifying the six coefficients on the star, it adds nothing to the computational cost. Using this extra term allows in programs like subroutine `wavemovie()` /prog:wavemovie yields wider angles.

Theory predicts that in two dimensions, waves going through a focus suffer a 90° phase shift. You should be able to notice that a symmetrical waveform is incident on the focus, but an antisymmetrical waveform emerges. This is easily seen in Figure 9.11.

In migrations, waves go just *to* a focus, not *through* it. So the migration impulse response in two dimensions carries a 45° phase shift. Even though real life is three dimensional, the two-dimensional response is appropriate for migrating seismic lines where focusing is presumed to arise from cylindrical, not spherical, reflectors.

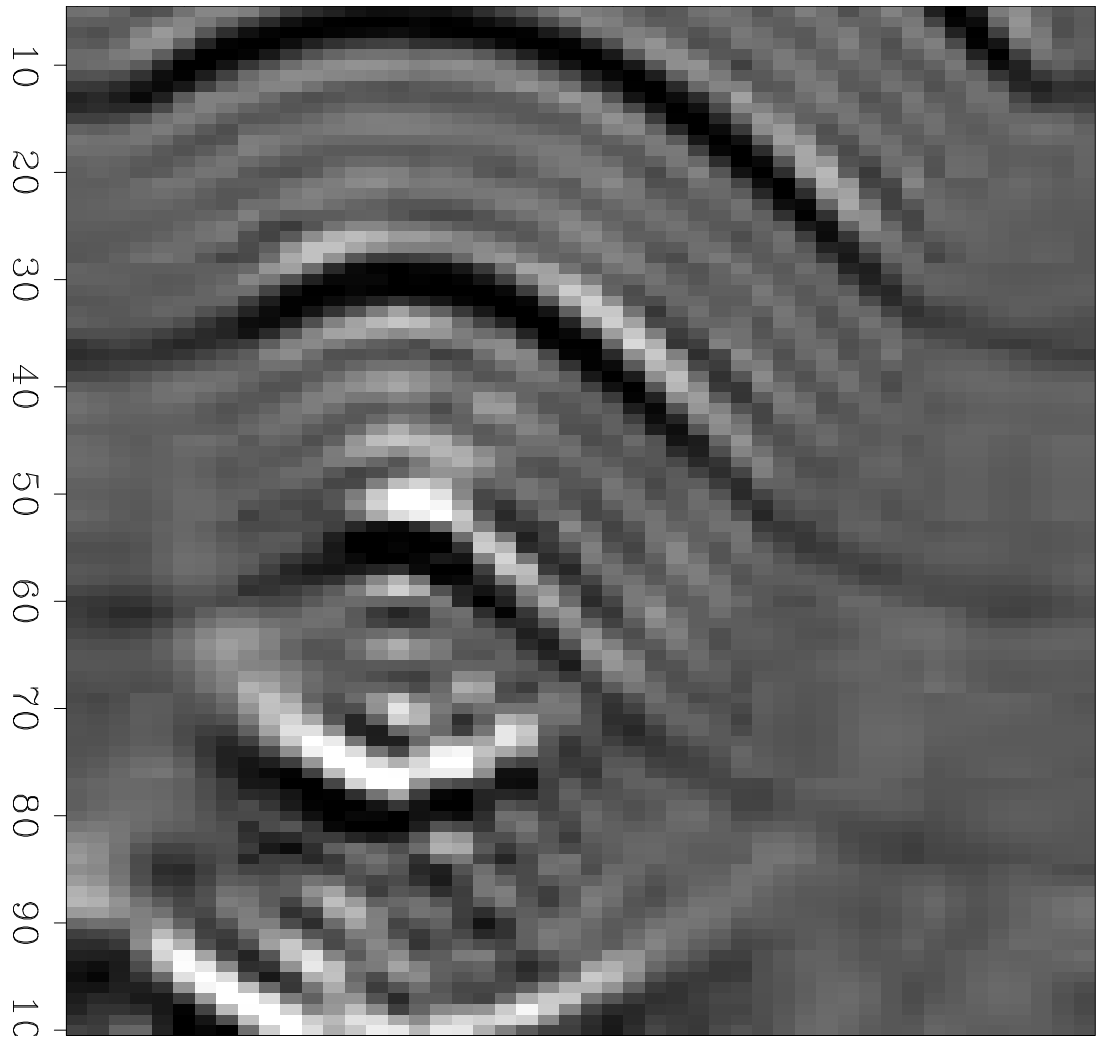


Figure 9.10: Figure 9.2 including the 45° term, ∂_{xxz}, for the collapsing spherical wave. What changes must be made to subroutine `wavemovie()` to get this result? Mark an *X* at the theoretical focus location. fdm-Mfortyfive90 [ER,M]

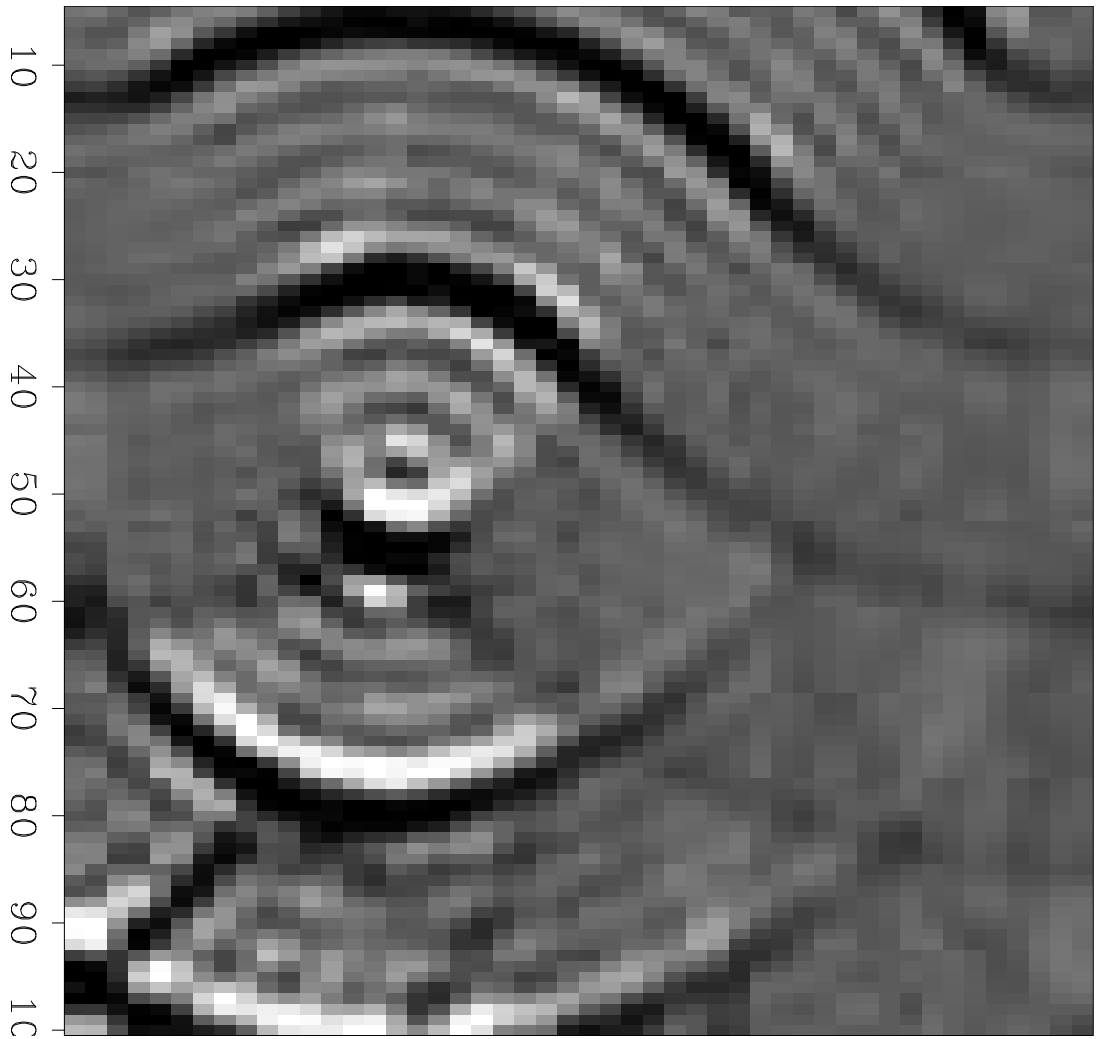


Figure 9.11: The accuracy of the x-derivative may be improved by a technique that is analyzed in IEI p 262-265. Briefly, instead of representing $k_x^2\,\Delta x^2$ by the **tridiagonal** matrix **T** with $(-1, 2, -1)$ on the main diagonal, you use $\mathbf{T}/(\mathbf{I} - \mathbf{T}/6)$. Modify the extrapolation analysis by multiplying through by the denominator. Make the necessary changes to the 45° collapsing wave program. Left without 1/6 trick; right, with 1/6 trick. fdm-Mhi45b90 [ER,M]

9.5.5. Time-domain parabolic equation

The parabolic wave extrapolation equation (9.57) is readily expressed in the time domain (instead of the ω-domain). Simply replace $-i\omega$ by a time derivative.

$$\frac{\partial^2 q}{\partial z\, \partial t} \quad = \quad \frac{v}{2}\, \frac{\partial^2 q}{\partial x^2} \tag{9.67}$$

In principal we never need the time domain because the earth velocity is a constant function of time. In practice, processes (like DMO) might involve time-dependent coefficients. In the time domain, a more complicated numerical procedure is required (details in my earlier book FGDP). An advantage of the time domain is that there is absolutely zero noise preceding a first arrival — no time-domain wraparound. Another advantage is that all signals are real valued — no complex arithmetic. A disadvantage arises when the t-axis is not sampled densely enough — the propagation velocity becomes frequency dispersive.

9.5.6. Wavefront healing

When a planar (or spherical) wavefront encounters an inhomogeneity it can be said to be "damaged". If it continues to propagate for a long time, it might be said to "heal". Here we construct an example of this phenomena and see that while there is some healing on the front edge, the overall destruction continues. The original simplicity of the wavefield is being destroyed by the continued propagation.

We begin with a plane wave. Then we deform it as though it had propagated through a slow lens of thickness $h(x) = \sin x$. This is shown in the first frame of Figure 9.12. In subsequent frames the wavefront has been extrapolated in z using equation (9.67).

In the second frame we notice convex portions of the wavefront weakening by something like spherical divergence while concave portions of the wavefront strengthen by focusing.

In the third frame we have moved beyond the focus and we see something like a parabolic wavefront emerge from each focus. Now we notice that the original waveform was a doublet whereas the parabolic wavefronts all have a single polarity. Focusing in 2-D has turned an asymmetrical wavelet into a symmetrical one.

In the fourth frame we see the paraboloids enlarging and crossing over one

Figure 9.12: Snapshots of a wavefront propagating to the right. The picture frame moves along with the wavefront. (Press button for movie.) fdm-heal [ER,M]

another. Inspect the top or the bottom edges of the 4th and 5th frames. You'll notice that the intersections of the wavefronts on these side boundaries are moving forward — towards the initial onset. This is peculiar. The phase fronts are moving forward while the energy is falling further behind the original onset.

Finally, in the last frame, we notice the that the front edge of the wave packet has "healed" into a plane wave — a plane wave like before encounting the original $\sin(x)$ velocity lens. I felt some delight on first viewing this picture. I had spent a couple years of my life looking at seismograms of earthquakes and nuclear explosions. For each event I had a seismic trace at each of about a dozen locations. Each trace would have about a hundred wiggles. Nothing would be consistent from trace to trace except for maybe the half wavelength of the first arrivals. Quite often these all would luckily begin with the same polarity but then become rapidly incoherent. Take a dozen random locations on the (vertical) x-axis of the last frame in Figure 9.12. You'll find the dozen time signals agree on the first arrival but are randomly related at later times just as usually seen with nuclear explosion data.

Perhaps if we had very dense recordings of earthquakes we could extrapolate the wavefield back towards its source and watch the waveform get simpler as we proceeded backward. Often throughout my career I've wondered how I might ap-

proach this goal. As we step back in z we wish, at each step, that we could find the best lens(x). My next book (GEE) has some clues, but nothing yet concrete enough to begin. We need to optimize some (yet unknown) expression of simplicity of the wavefield(t, x) at the next z as a function of the lens between here and there.

Chapter 10

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 10.1. 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 10.1.

10.0.1. 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 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:

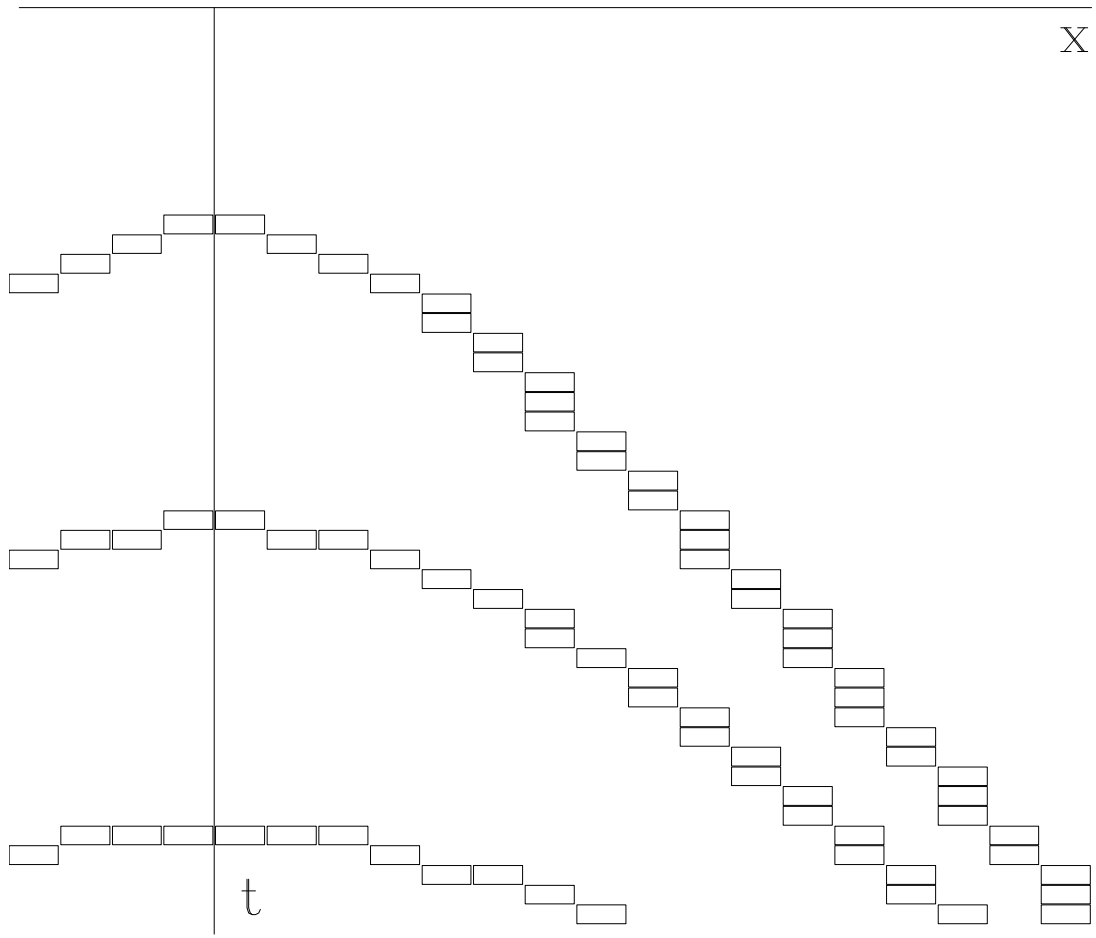


Figure 10.1: To integrate along hyperbolas without aliasing, you should include (at least) the points shown. trimo-nmotraj [ER]

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).

10.1. 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 10.2 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 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 rectan-

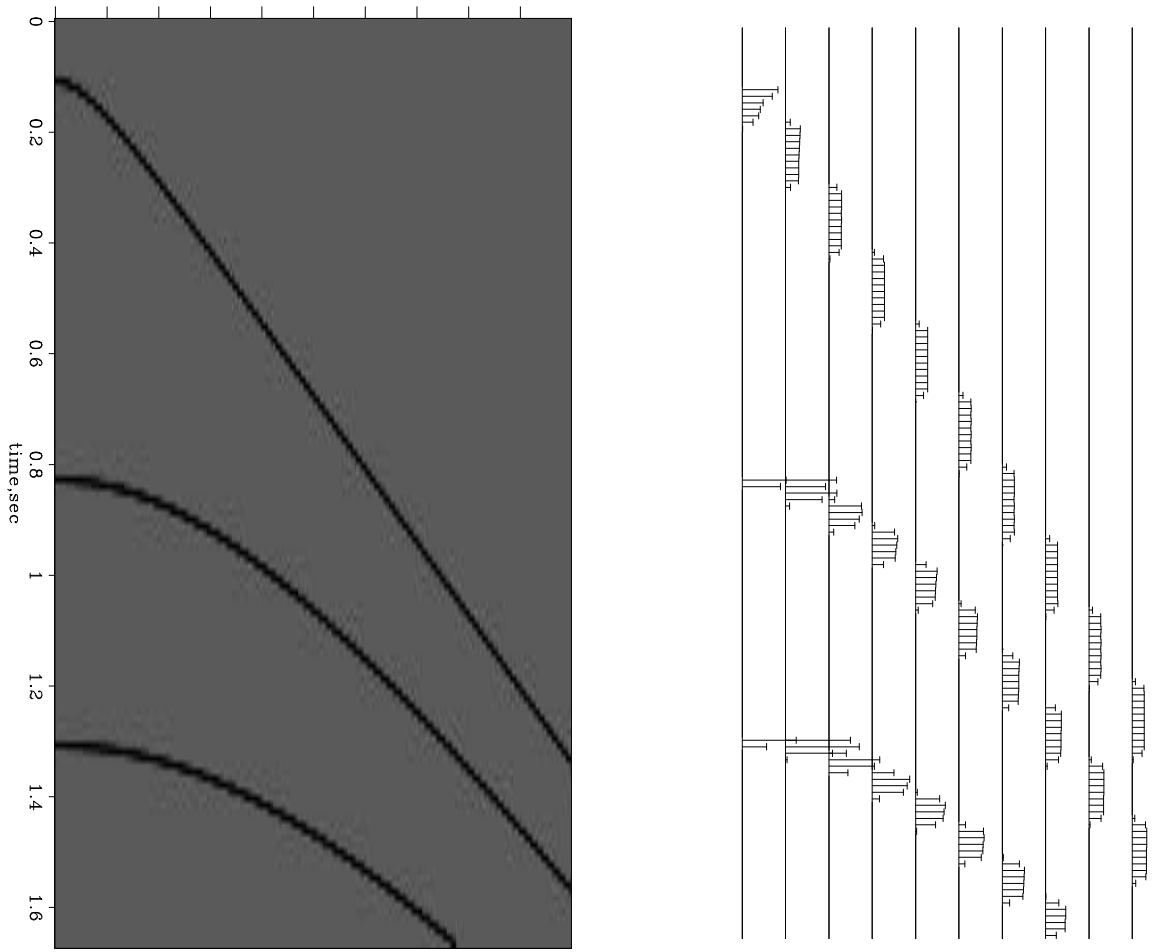


Figure 10.2: Quasicontinuous field (left) added in groups (right). trimo-oversamp [ER]

gle 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 10.3 shows a quasisinusoidal signal and compares subsampling to antialiasing via field arrays as in Figure 10.2. 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 10.3 was computed simply by adding in groups of 25. A lengthier explanation of the calculation is that 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

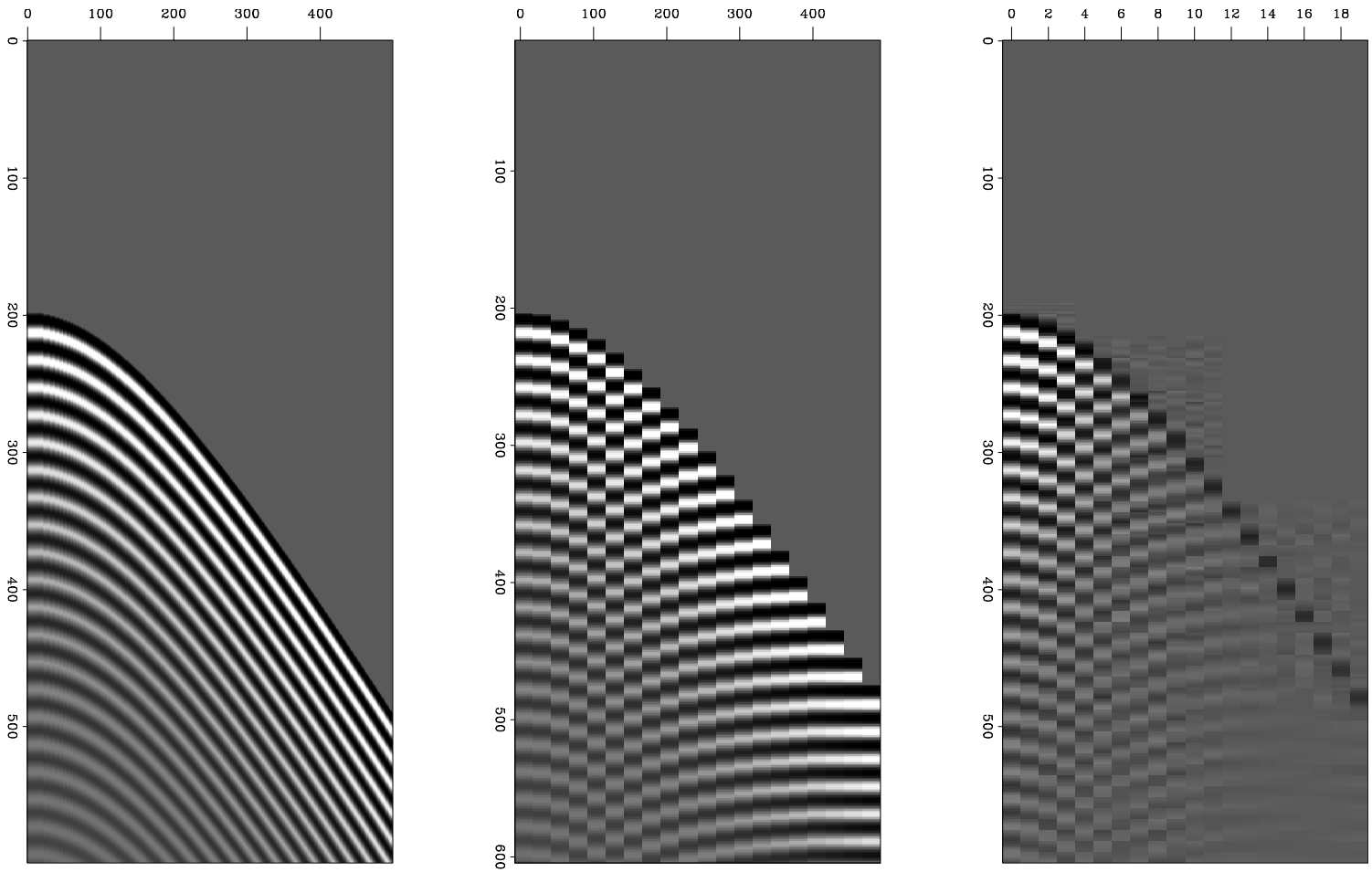


Figure 10.3: 500 channels (left), subsampled to 20 (middle), added in groups of 25(right). trimo-subsampvrsaa [ER]

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 10.3 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 10.3 (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.

10.1.1. 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 10.4 shows a sim-

ple 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.

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 10.5. 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

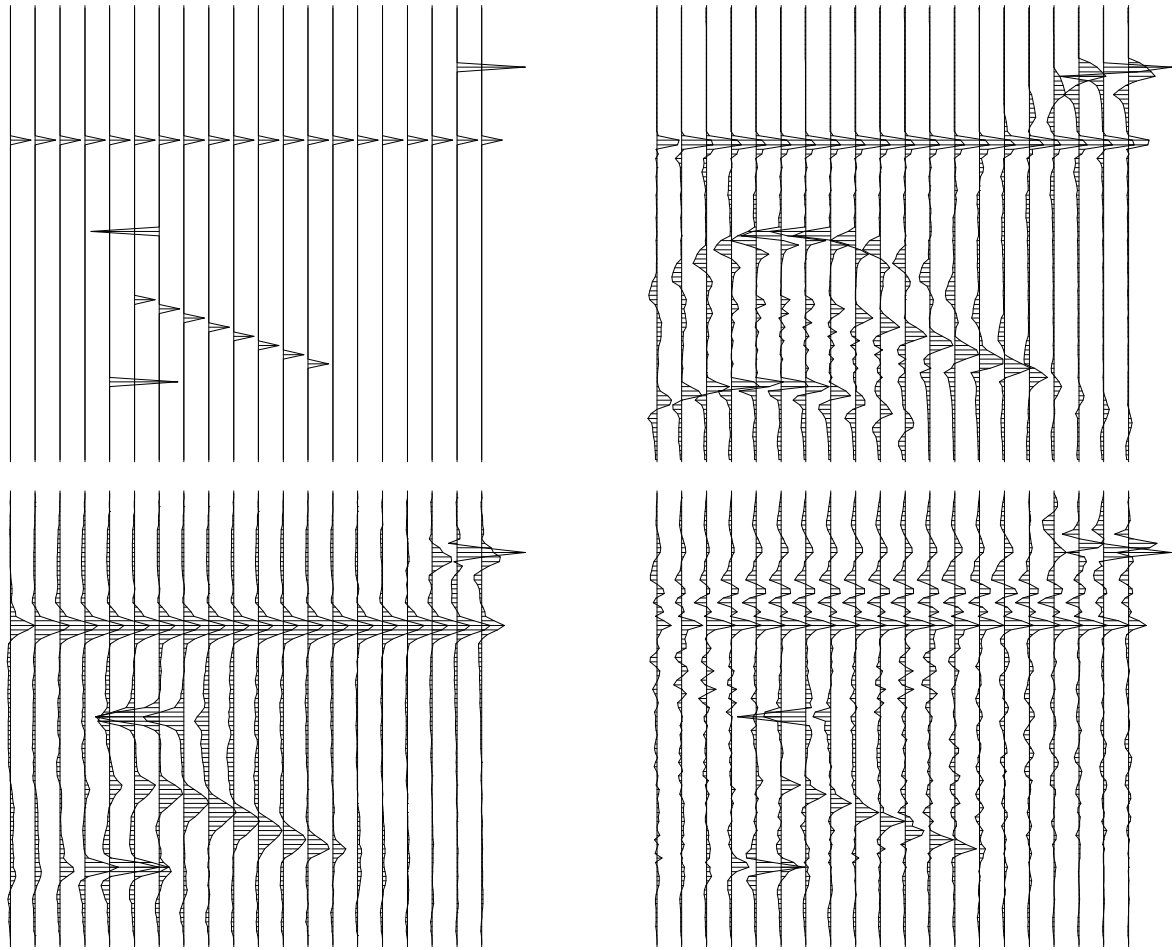


Figure 10.4: Top left is a synthetic image. Top right is synthetic data from the synthetic image. Bottom are migrations of the data with and without antialiasing. trimo-migalias [ER]

rectangle at the next offset. You might fear the coding that led up to Figure 10.5

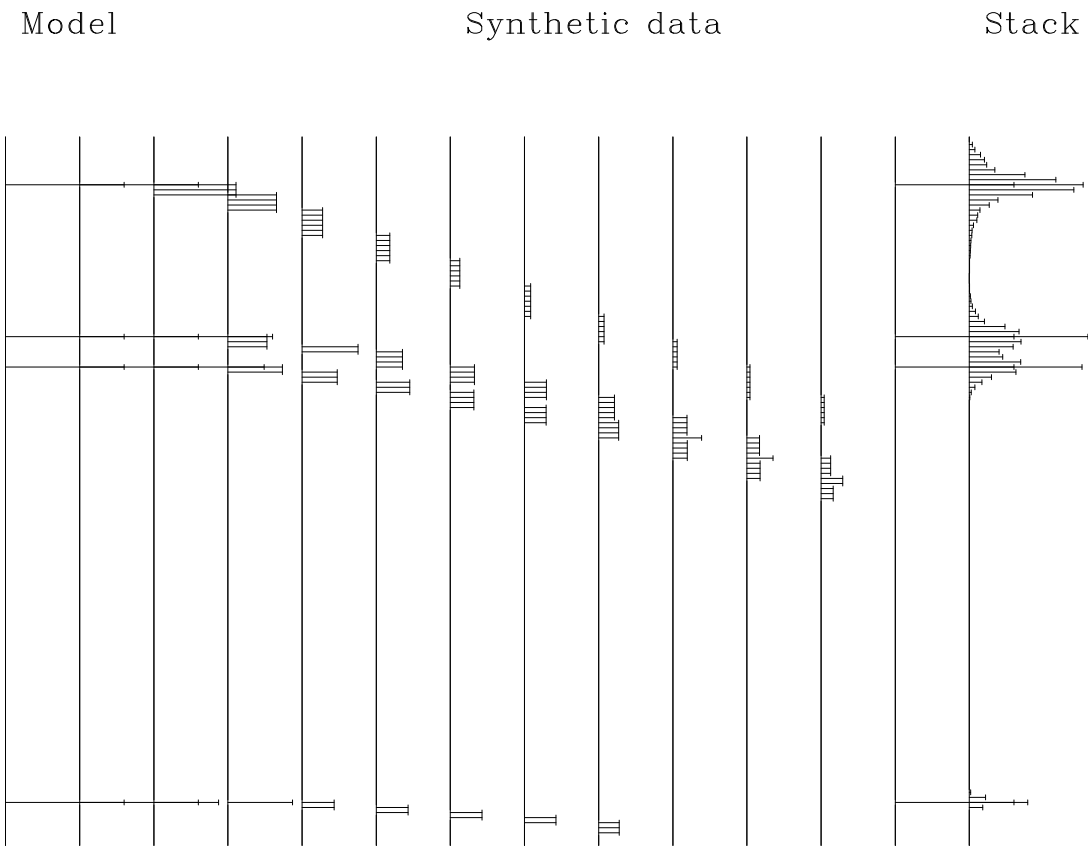


Figure 10.5: Rectangle smoothing during NMO and stacking. Notice that the end of one rectangle exactly coincides with the beginning of the rectangle at next larger offset. Thus, rectangle width increases with offset and decreases with time. (`antialias=1.`) trimo-boxmo1 [ER]

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 10.5 is not fully consistent with Figure 10.1. In Figure 10.5 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 hyperbola would have *zero length* which is wholly unacceptable. Should we write a code to match Figure 10.1? This would be better, but far from perfect. Notice in Figure 10.1 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.

```
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
```

Back

10.1.2. NMO and stack with a rectangle footprint

A subroutine for causal summation is subroutine `causint()` /prog:causint. 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)`. boxmo

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 ampli-

tude 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 10.6 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.

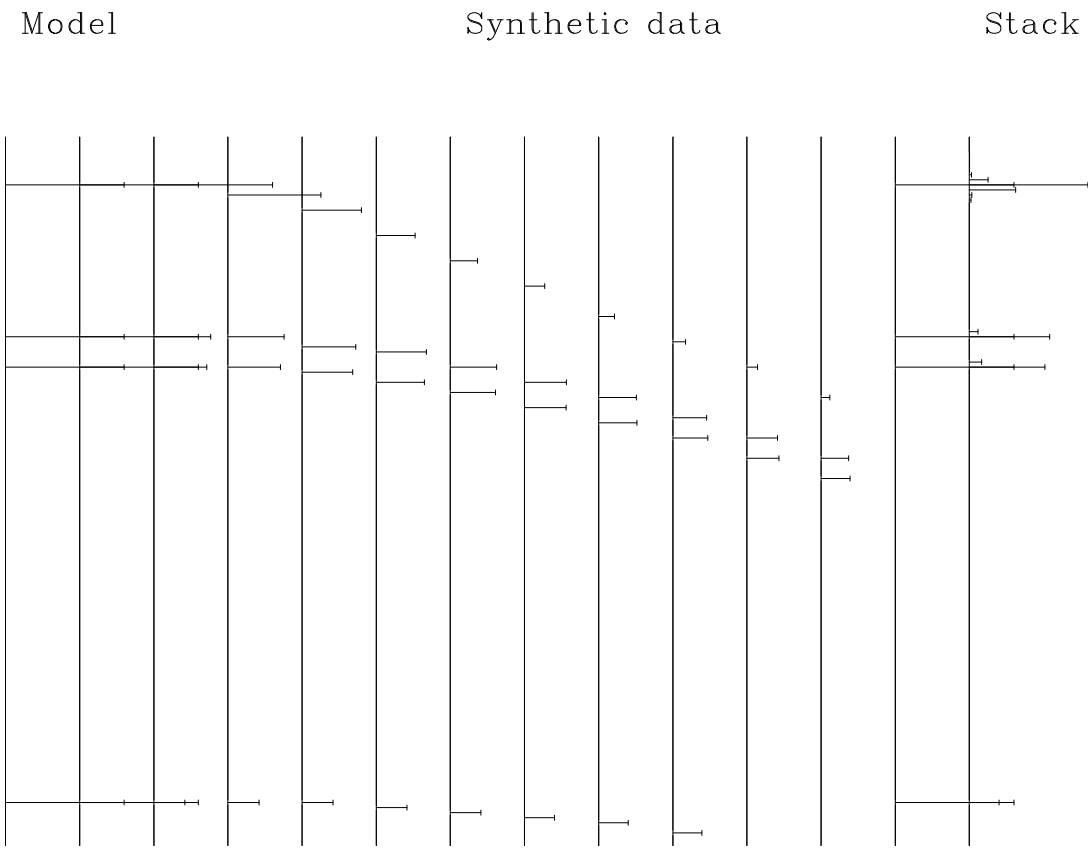


Figure 10.6: Rectangles shortened to one point duration. (`antialias=0.`) trimo-boxmo0 [ER]

10.1.3. 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()` /prog:doubint . 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. Subroutine `lint1()` /prog:lint1 does linear interpolation, but here we need weighted results as provided by `spotw()` /prog:spotw . spotw

Using these subroutines, I assembled the stacking subroutine `tristack()` and the NMO routine `trimo()` with triangle wavelets. The triangle routines are like

```
# 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
```

Back

```
#  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
```

Back

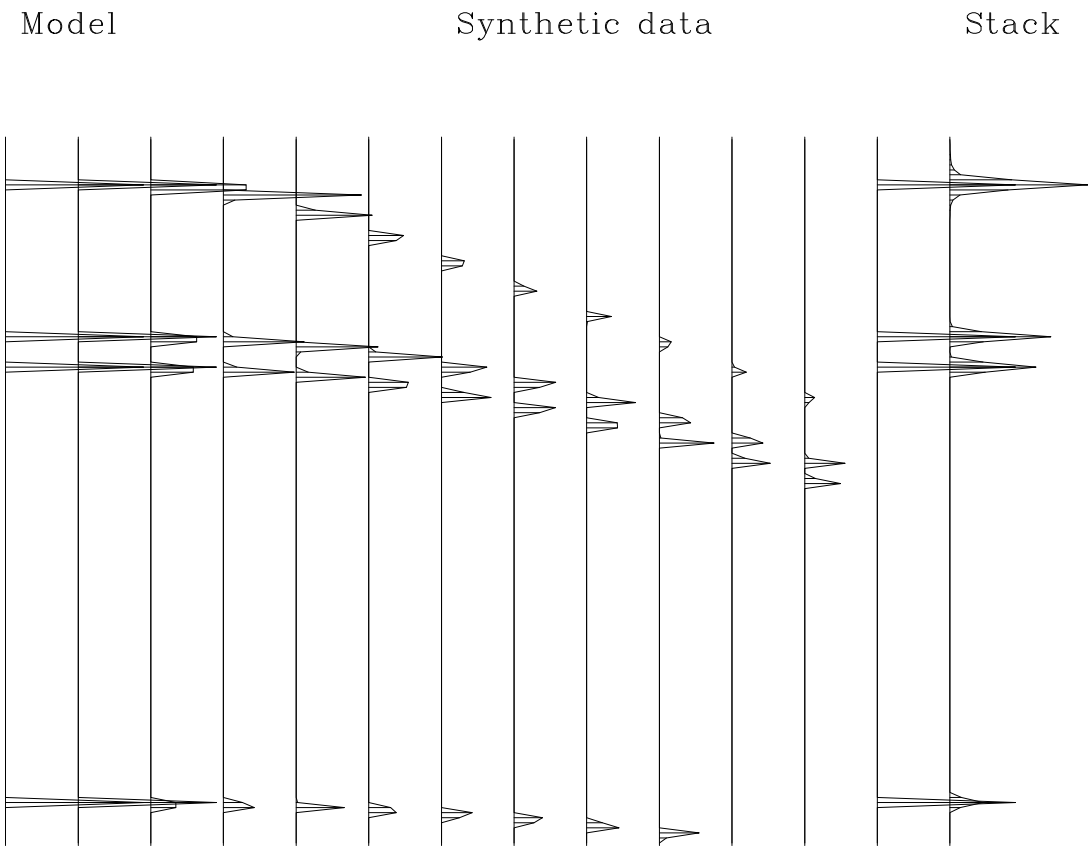


Figure 10.7: Triangle wavelets, accurately positioned, but aliased (`antialias=0.`) trimo-trimo0 [ER]

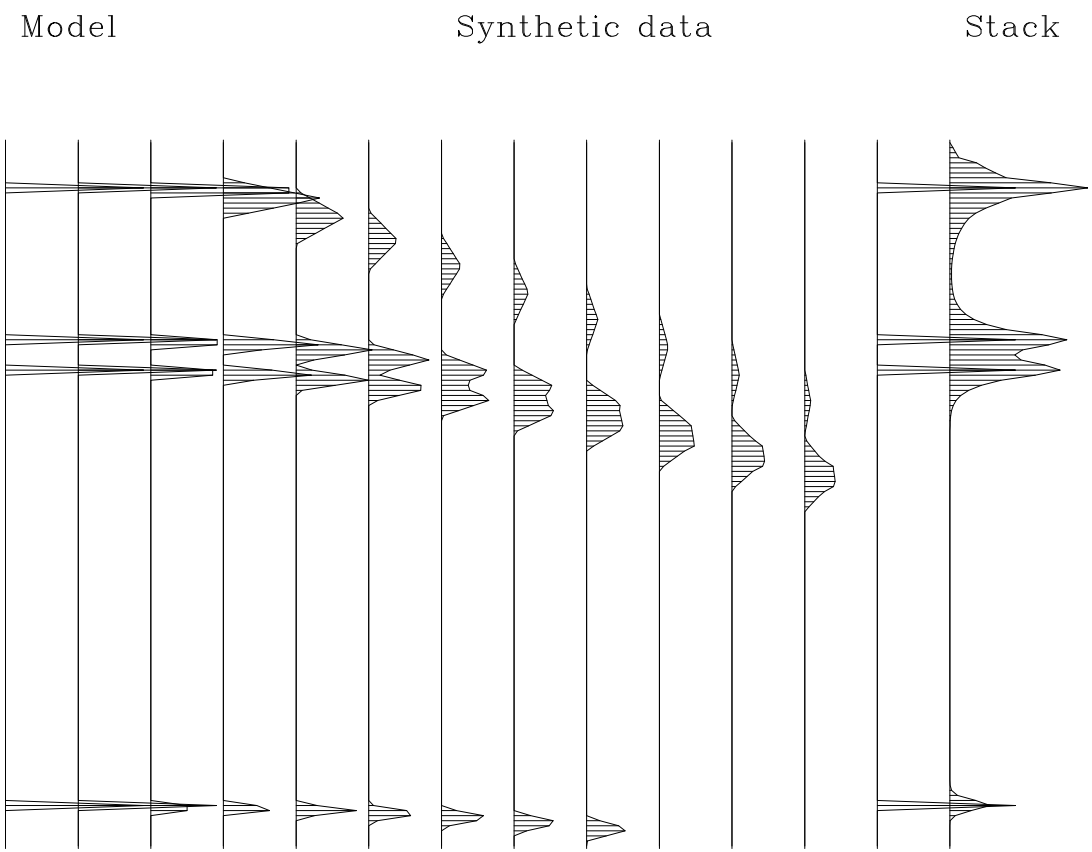


Figure 10.8: Antialiased triangle wavelets. (`antialias=1.`) Where ever triangle duration is more than about three points, the end of one triangle marks the apex of the next. trimo-trimo1 [ER]

```
# 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
```

Back

```
# 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
        tm = t - abs(slope * dx) - dt;   itm = 1.5 + (tm-t0) / dt
        if( itm <= 1  )  next
        tp = t + abs(slope * dx) + dt;   itp = 1.5 + (tp-t0) / dt
        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
```

Back

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 10.7 and 10.8. tristack trimo From the stack reconstruction of the model in Figure 10.8 we see the reconstruction is more blured with antialiasing than it was without in Figure 10.7. The benefit of antialiasing will become clear next in more complicated examples where events cross.

10.2. MIGRATION WITH ANTIALIASING

Subroutine `aamig()` below does migration and diffraction modeling using subroutine `trimo()` as the workhorse. aamig Figure 10.9 shows the synthetic image that is used for testing. There is a horizontal layer, a dipping layer, and a few im-

```
# 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
```

Back

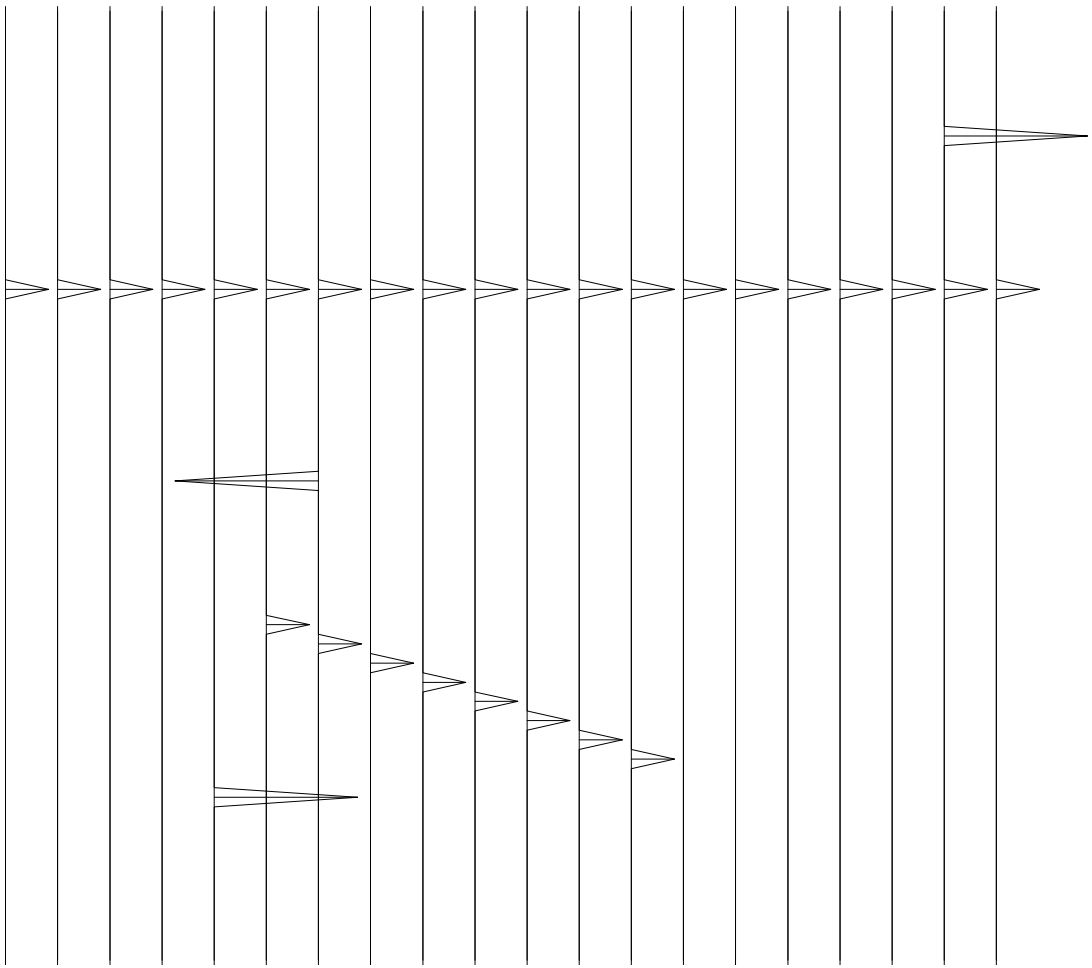


Figure 10.9: Model image for migration study. trimo-aamod [ER]

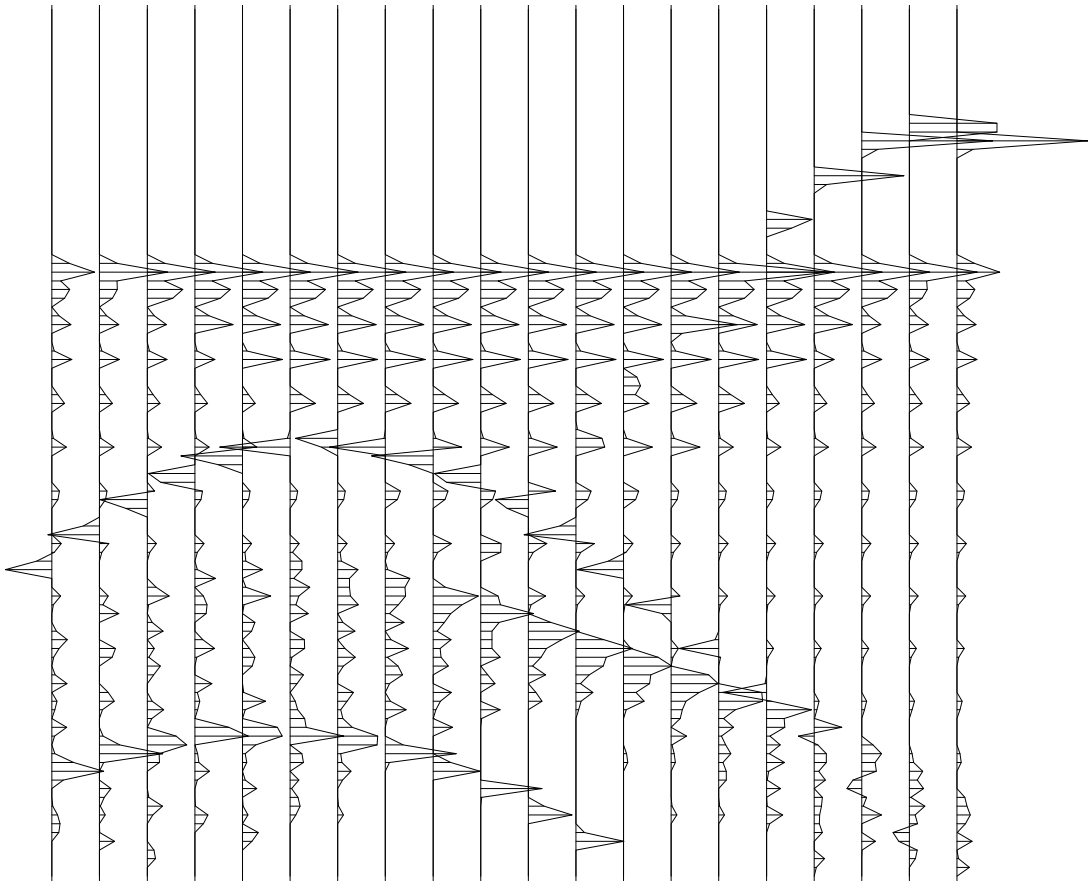


Figure 10.10: Synthetic data without regard for aliasing. Made from model image with `aamig()` taking `antialias=0`. trimo-aad0 [ER]

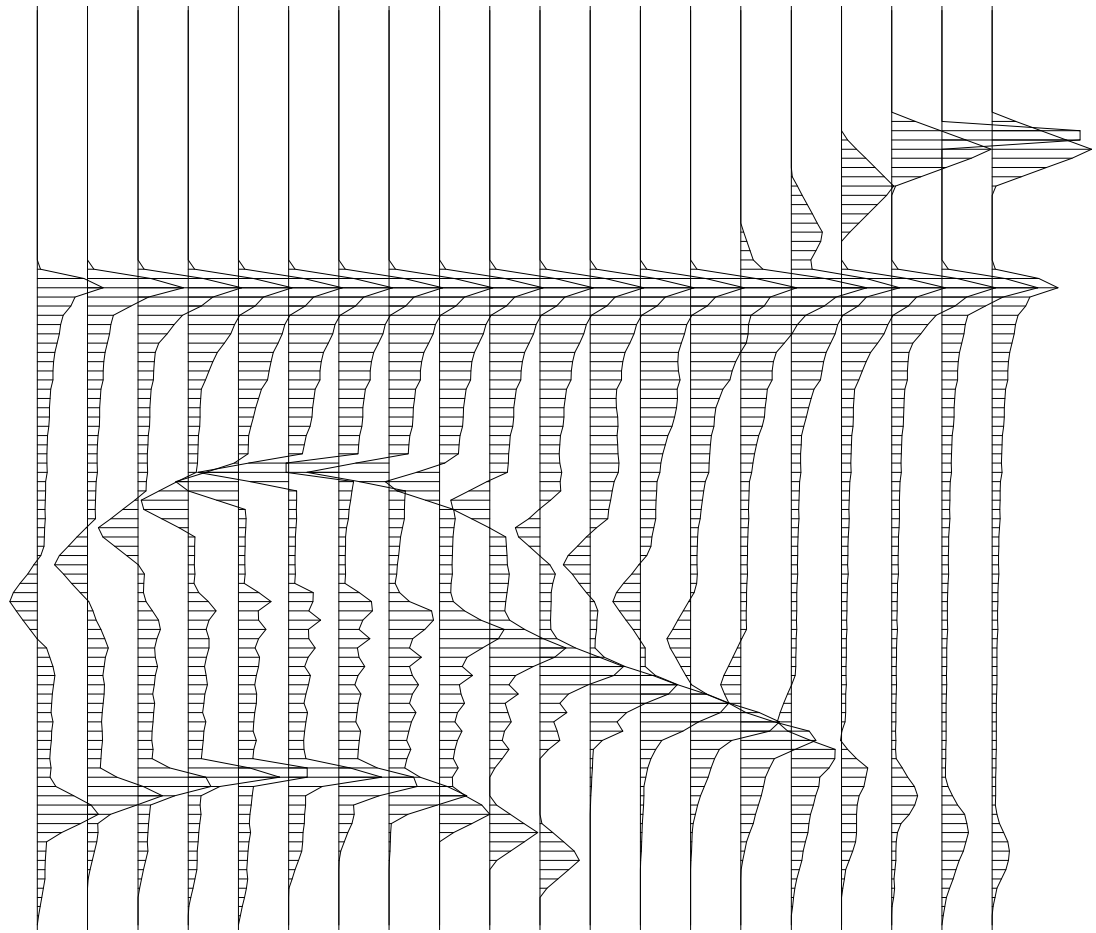


Figure 10.11: Synthetic data accounting for aliasing. Made from model image with `aamig()` taking `antialias=1.` trimo-aad1 [ER]

pulses. The impulses are chosen stronger than the layers because they will spread out in the synthetic data. The velocity is taken constant. Figure 10.10 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 10.11 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 10.11, 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 10.11 has a waveform that resembles a damped

step function which is related to the Hankel tail we studied in chapter 6 where subroutine `halfdifa()` /prog:halfdifa was introduced to provide the filter required to convert the waveform on the horizontal layer in Figure 10.11 back to an impulse. This was done in Figure 10.12. You can see the final flat-layer waveform is roughly the zero-phase shape we started with. Figure 10.13 shows my 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 10.14 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. But notice also that the dipping bed is sharper than the antialiased result in Figure 10.13.

10.2.1. 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 earth has dipping layers. There the `slope` parameter could be

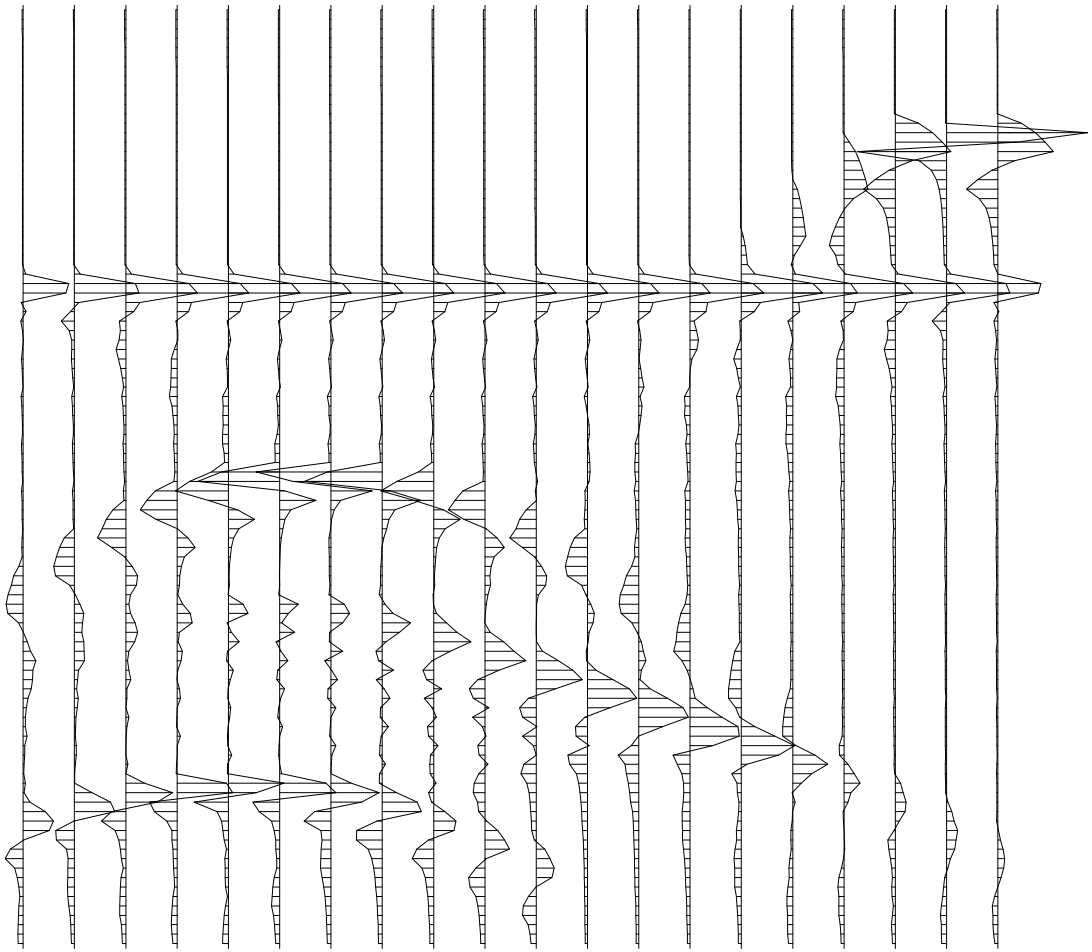


Figure 10.12: Best synthetic data. Made from model image using `aamig()` with `antialias=1` followed by a causal half-order time derivative. Lowpass temporal filtering would make wavelets more independent of location on a hyperbola. trimo-aad1h [ER]

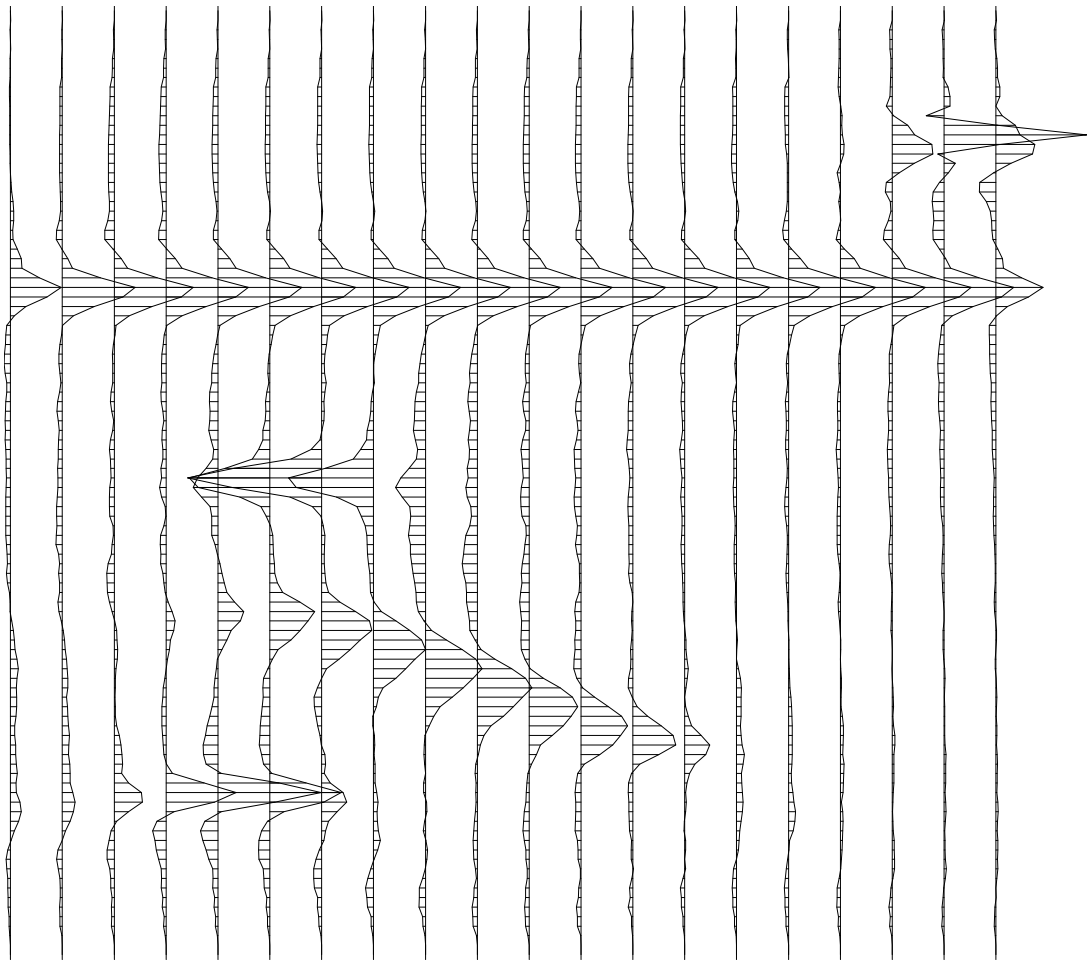


Figure 10.13: Best migration of best synthetic data. Uses `aamig()` with `antialias=2` followed by an anticausal half-order time derivative. trimo-aamig2 [ER]

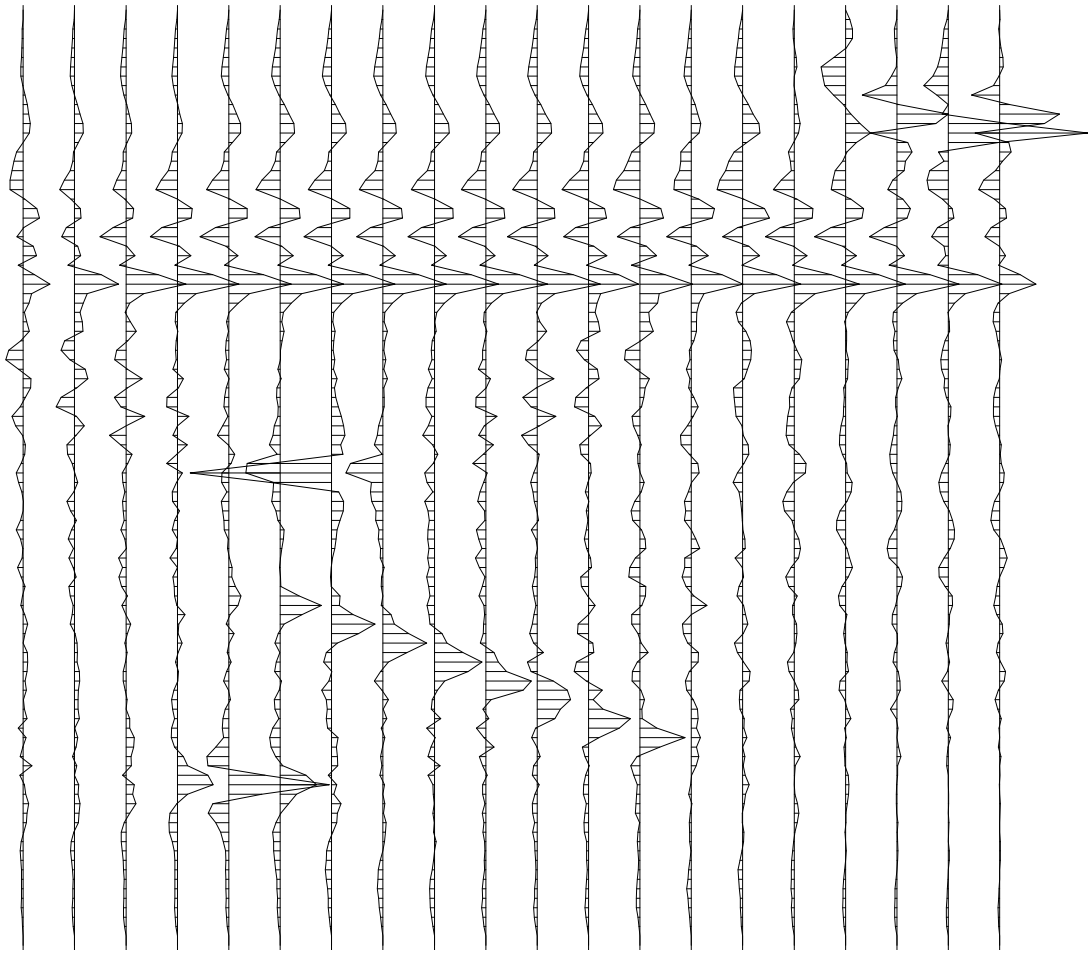


Figure 10.14: Migration of best synthetic data without regard for aliasing. Uses `aamig()` with `antialias=0.` (and an anti-causal half-order time derivative) trimo-aamig0 [ER]

biased to account for it.

Where the earth contains hyperbolas, they will cut steeply across our migration hyperbola. Figure 10.15 suggests that such hyperbolas require an antialias parameter greater than unity, say `antialias=2`.

10.2.2. 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.

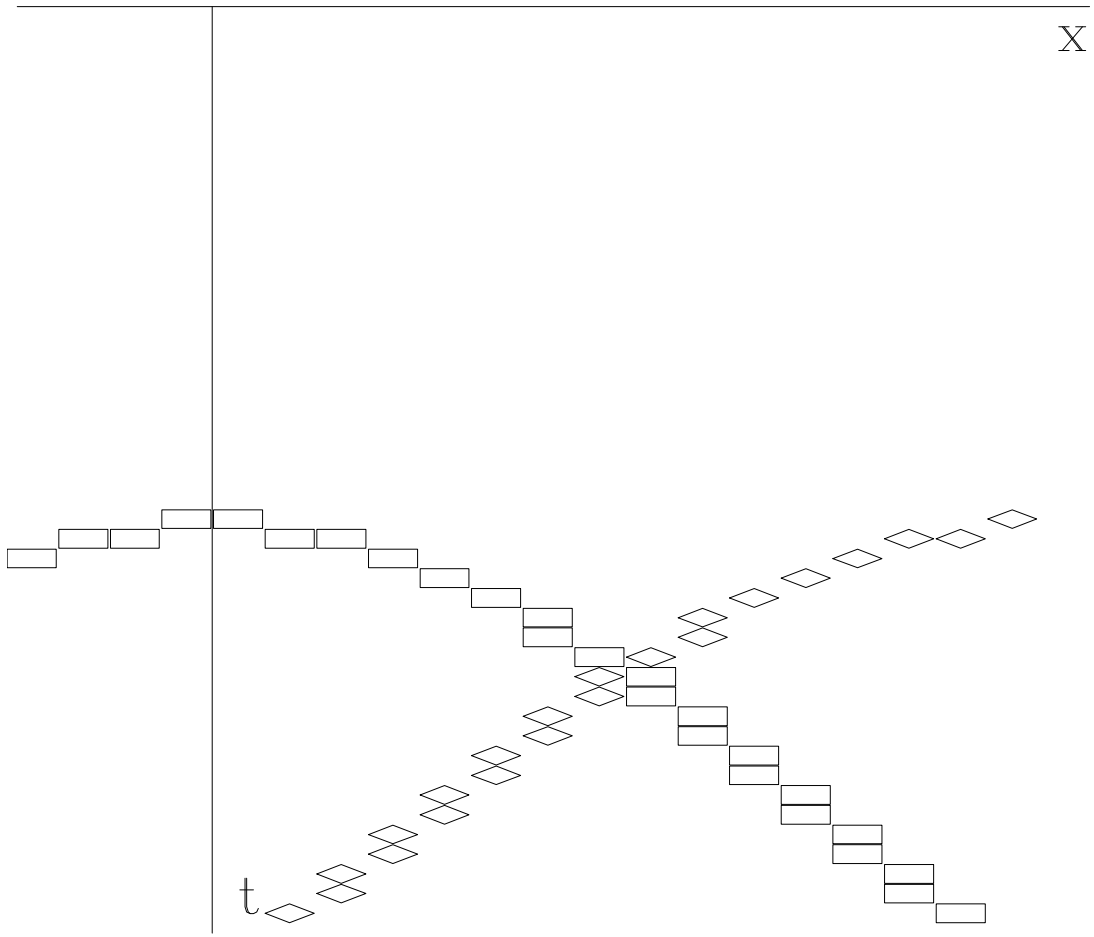


Figure 10.15: Crossing hyperbolas that do not touch. Thus the points shown are not enough to prevent spatial aliasing a line integral along one trajectory of signal on the other. trimo-croshyp [ER]

10.3. 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.

10.3.1. Iterative velocity transform

After we use data to determine a velocity model (or slowness model) with an operator **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 **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 10.17. Figure 10.17 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 hyperbolas curving down.

Figure 10.18 shows the effect of no antialiasing in either the field recording or

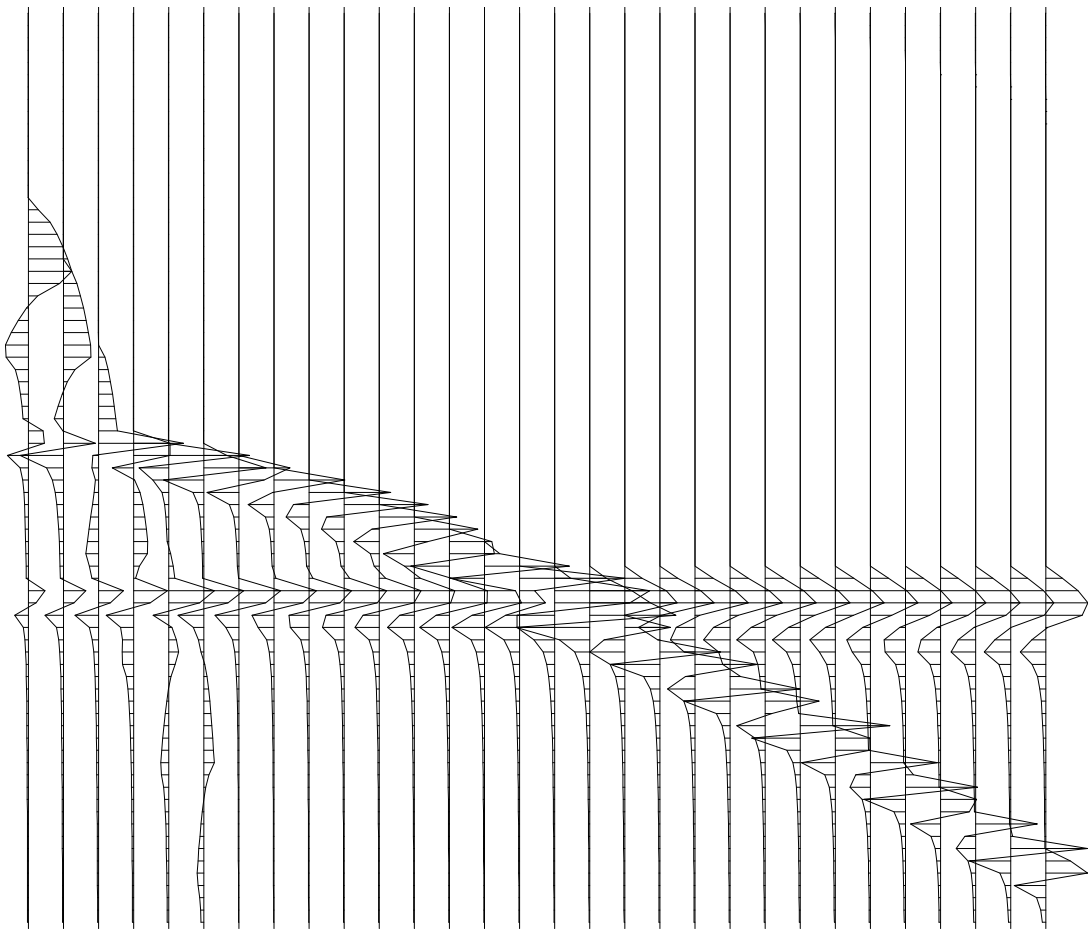


Figure 10.16: The air wave and fast wave are broadened increasingly with offset, but the water wave does not. This broadening enables crossing events to maintain their orthogonality. trimo-aacdp [ER]

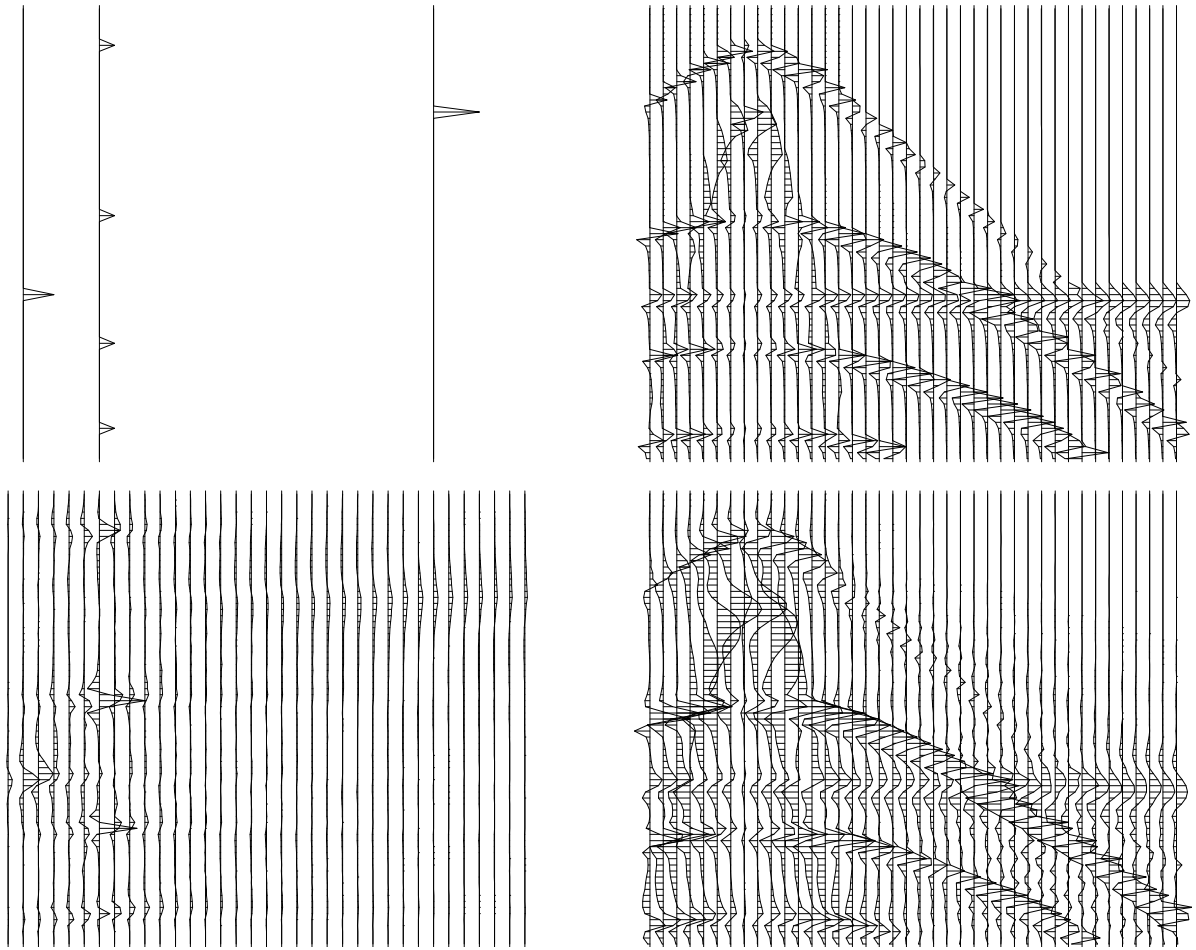


Figure 10.17: Top left: Slowness model. Top right: Data derived from it using the pseudounitary scale factor. Bottom left: the velocity spectrum of top right. Bottom right: data made from velocity spectrum. trimo-aavel1 [ER]

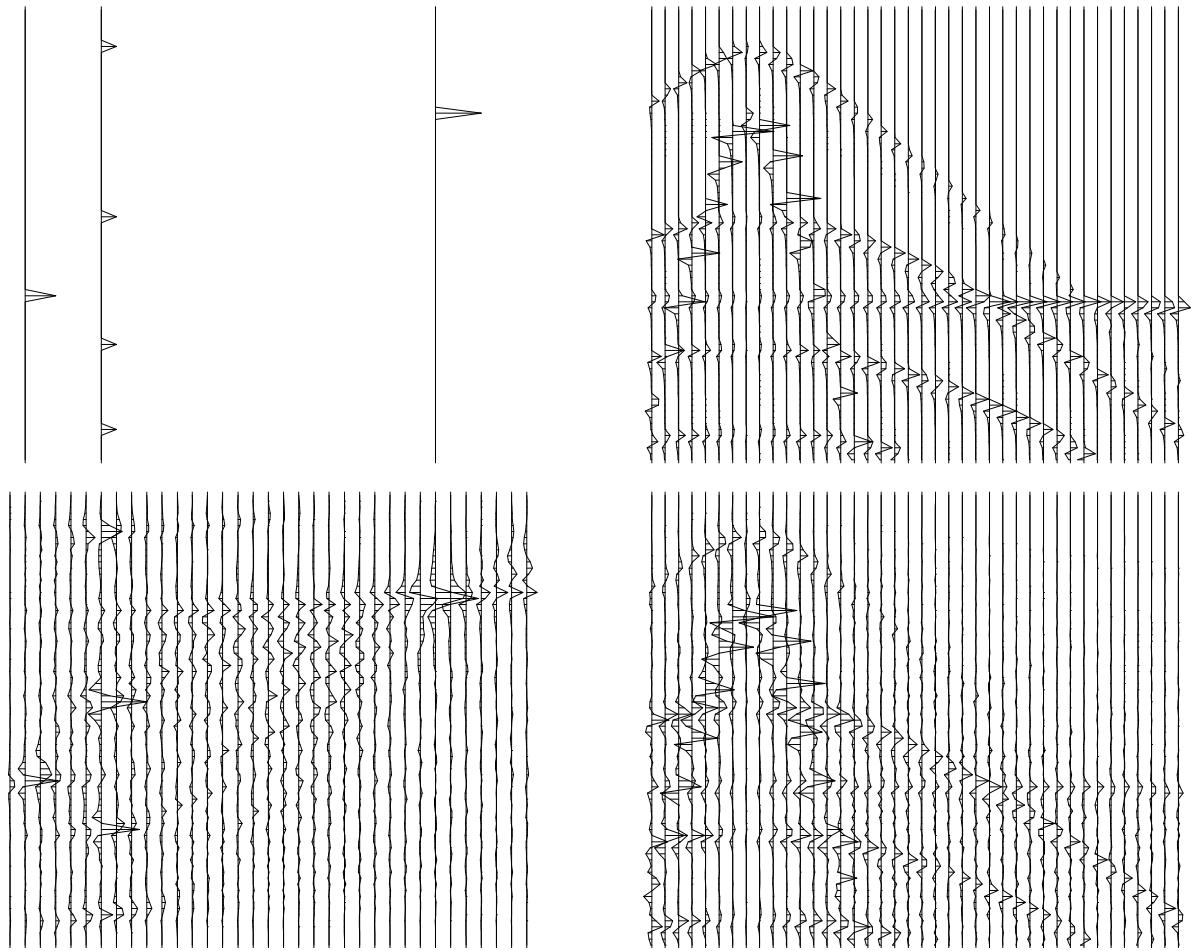


Figure 10.18: Like Figure 10.17 but with `antialias=0`. This synthetic data presumes no receiver groups in the field recording. trimo-aavel0 [ER]

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 10.18 or should we use an antialiased operator as that in Figure 10.17? Figure 10.19 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 velocity analysis. Failing to do so will give a result like in figure 10.20. 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).

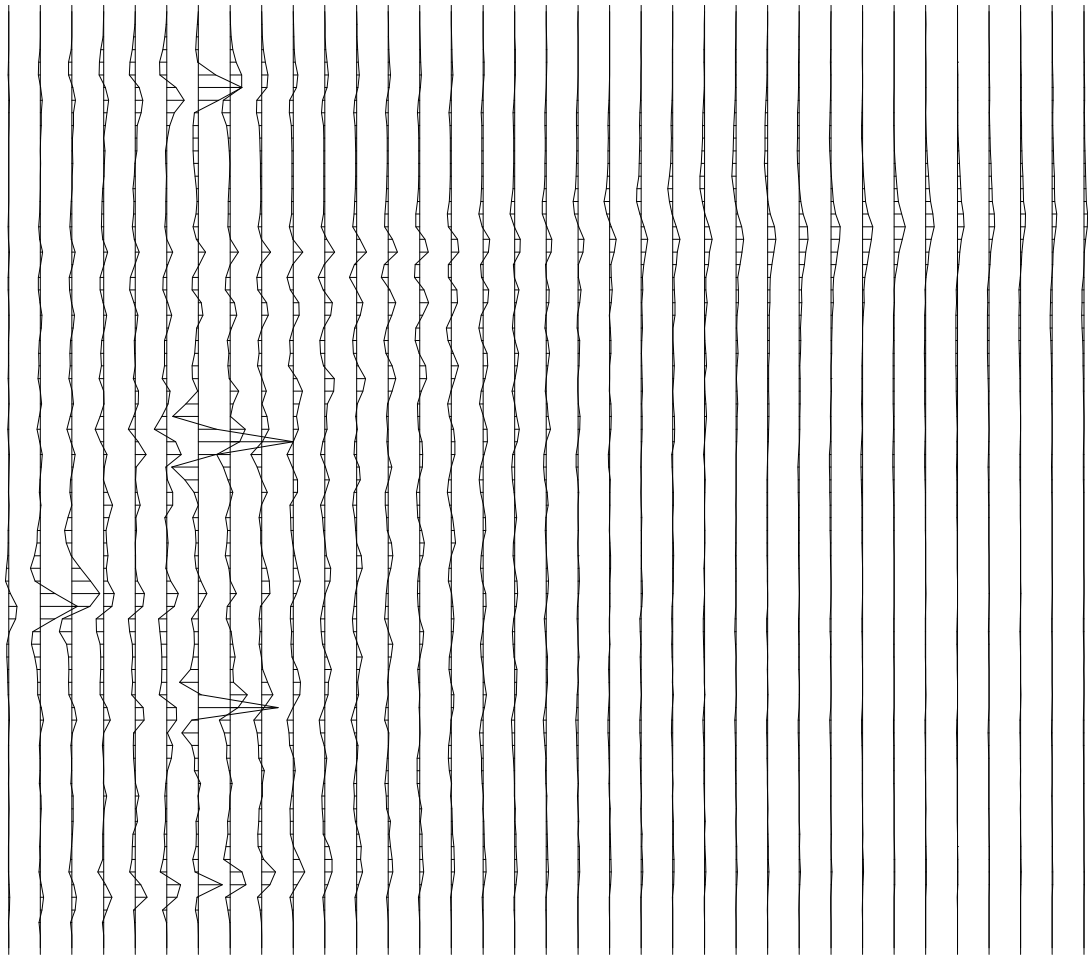


Figure 10.19: Aliased data analyzed with antialiased operator. Compare to the lower left of Figure 10.18. trimo-adataavel [ER]

The conceptual advantage of a pseudounitary transformation like Figure 10.17 is that points in velocity space are orthogonal components like Fourier components whereas for nonunitary transforms like with Figure 10.20 the different points in velocity space are not orthogonal components.

Subroutine `veltran()` does the work. veltran

EXERCISES:

1 What circumstances would suggest that the linear interpolation in subroutine `trimo()` /prog:trimo 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.

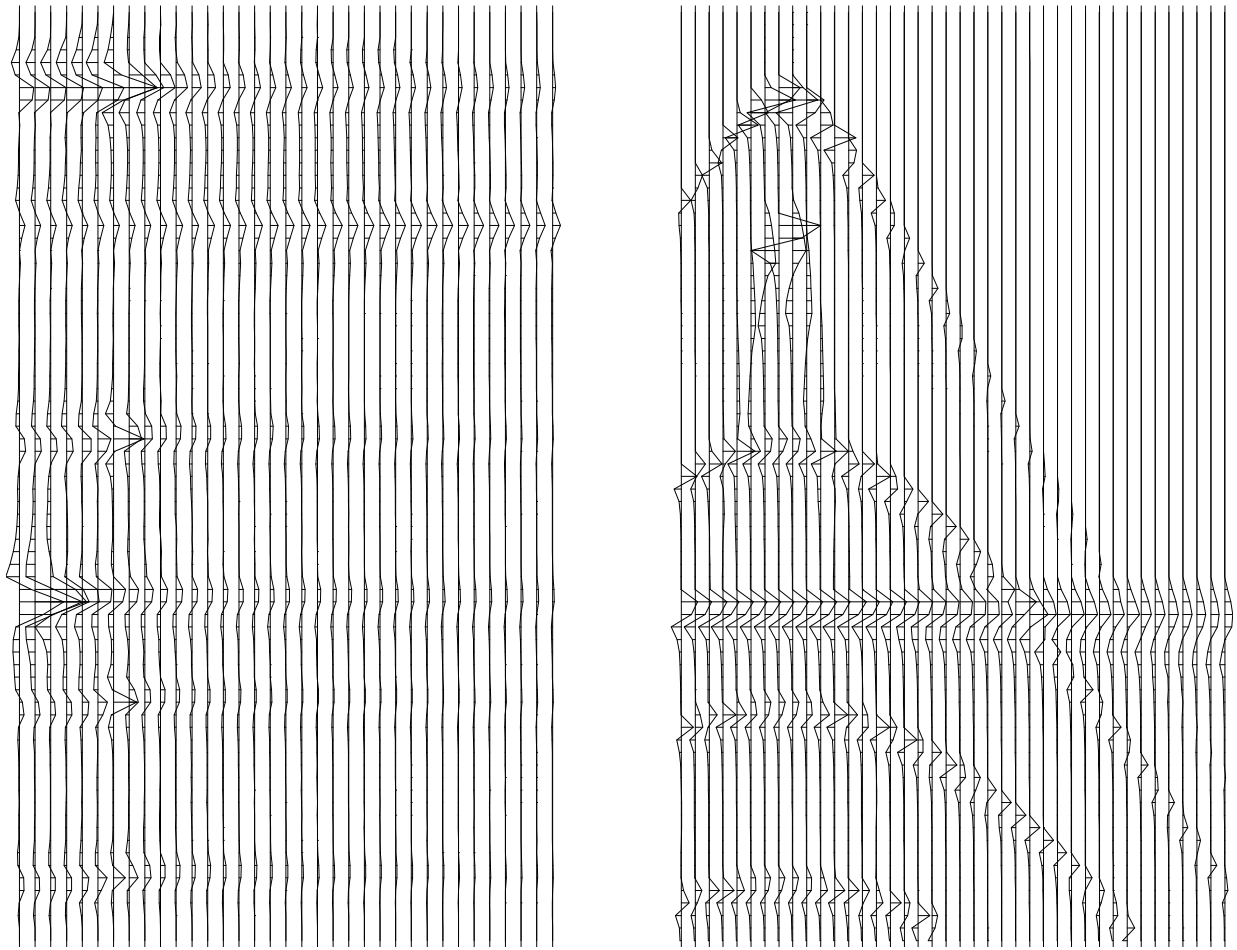


Figure 10.20: Like Figure 10.17 omitting pseudounitary scaling. `psun=0`. Right is synthetic data and left the analysis of it which is badly smeared. trimo-velsmear [ER]

```
# 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
```

Back

Chapter 11

Imaging in shot-geophone space

Till now, we have limited our data processing to midpoint-offset space. We have not analyzed reflection data directly in shot-geophone space. In practice this is often satisfactory. Sometimes it is not. The principal factor that drives us away

from (y,h)-space into (s,g)-space is lateral velocity variation $v(x,z) \neq v(z)$. In this chapter, we will see how migration can be performed in the presence of $v(x,z)$ by going to (s,g)-space.

Unfortunately, this chapter has no prescription for finding $v(x,z)$, although we will see how the problem manifests itself even in apparently stratified regions. We will also see why, in practice, amplitudes are dangerous.

11.1. TOMOGRAPY OF REFLECTION DATA

Sometimes the earth strata lie horizontally with little irregularity. There we may hope to ignore the effects of migration. Seismic rays should fit a simple model with large reflection angles occurring at wide offsets. Such data should be ideal for the measurement of reflection coefficient as a function of angle, or for the measurement of the earth acoustic absorptivity $1/Q$. In his doctoral dissertation, Einar **Kjartansson** reported such a study. The results were so instructive that the study will be thoroughly reviewed here. I don't know to what extent the Grand Isle gas field typifies the rest of the earth, but it is an excellent place to begin learning about the meaning of shot-geophone offset.

11.1.1. The grand isle gas field: a classic bright spot

The dataset **Kjartansson** studied was a seismic line across the Grand Isle gas field, off the shore of Louisiana. The data contain several classic "bright spots" (strong reflections) on some rather flat undisturbed bedding. Of interest are the lateral variations in amplitude on reflections at a time depth of about 2.3 seconds on Figure 11.3. It is widely believed that such bright spots arise from gas-bearing sands.

Theory predicts that reflection coefficient should be a function of angle. For an anomalous physical situation like gas-saturated sands, the function should be distinctive. Evidence should be found on common-midpoint gathers like those shown in Figure 11.1. Looking at any one of these gathers you will note that the reflection strength versus offset seems to be a smooth, sensibly behaved function, apparently quite measurable. Using layered media theory, however, it was determined that only the most improbably bizarre medium could exhibit such strong variation of reflection coefficient with angle, particularly at small angles of incidence. (The reflection angle of the energy arriving at wide offset at time 2.5 seconds is not a large angle. Assuming constant velocity, $\arccos(2.3/2.6) = 28°$). Compounding the puzzle, each common-midpoint gather shows a *different* smooth, sensibly behaved, measurable function. Furthermore, these midpoints are near one another, ten

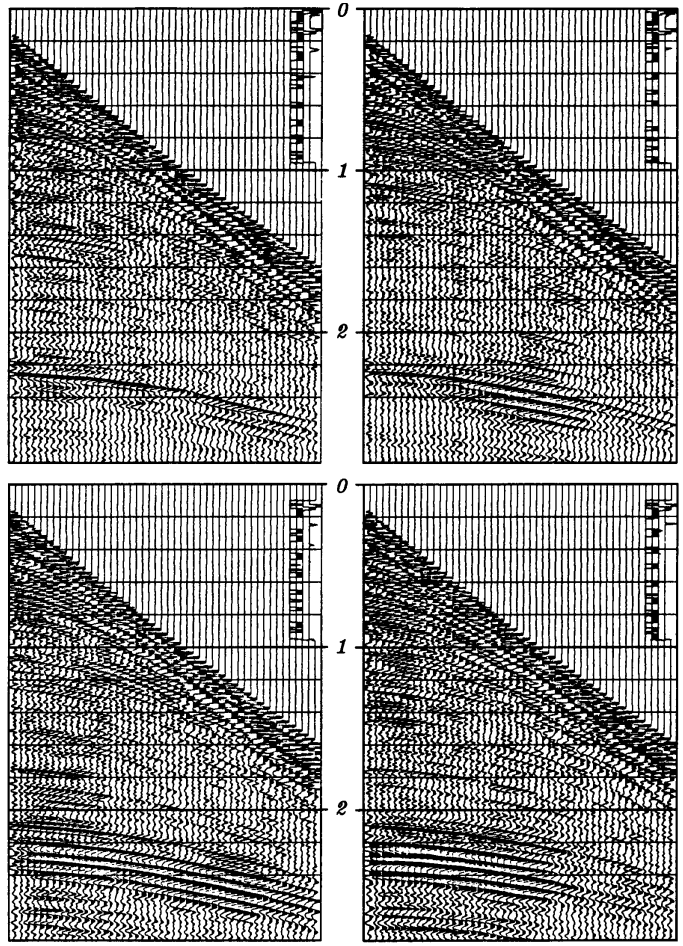


Figure 11.1: Top left is shot point 210; top right is shot point 220. No processing has been applied to the data except for a display gain proportional to time. Bottom shows shot points 305 and 315. (Kjartansson) sg-kjcmg [NR]

shot points spanning a horizontal distance of 820 feet.

11.1.2. Kjartansson's model for lateral variation in amplitude

The Grand Isle data is incomprehensible in terms of the model based on layered media theory. Kjartansson proposed an alternative model. Figure 11.2 illustrates a geometry in which rays travel in straight lines from any source to a flat horizontal reflector, and thence to the receivers. The only complications are "pods" of some material that is presumed to disturb seismic rays in some anomalous way. Initially you may imagine that the pods absorb wave energy. (In the end it will be unclear whether the disturbance results from energy focusing or absorbing).

Pod A is near the surface. The seismic survey is affected by it twice—once when the pod is traversed by the shot and once when it is traversed by the geophone. Pod C is near the reflector and encompasses a small area of it. Pod C is seen at all offsets h but only at one midpoint, y_0. The raypath depicted on the top of Figure 11.2 is one that is affected by all pods. It is at midpoint y_0 and at the widest offset $h_{\max}$. Find the raypath on the lower diagram in Figure 11.2.

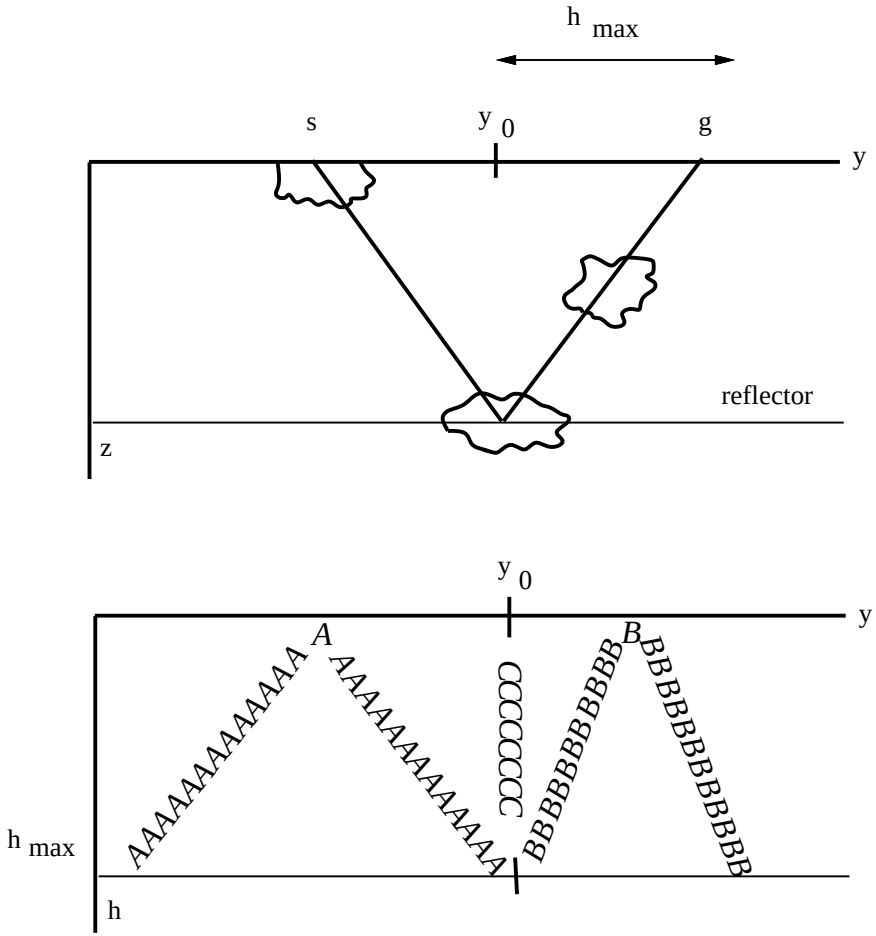


Figure 11.2: Kjartansson's model. The model on the top produces the disturbed data space sketched below it. Anomalous material in pods A, B, and C may be detected by its effect on reflections from a deeper layer. sg-kjidea [NR]

Pod B is part way between A and C. The slope of affected points in the (y,h)-plane is part way between the slope of A and the slope of C.

Figure 11.3 shows a common-offset section across the gas field. The offset shown is the fifth trace from the near offset, 1070 feet from the shot point. Don't be tricked into thinking the water was deep. The first break at about .33 seconds is wide-angle propagation.

The power in each seismogram was computed in the interval from 1.5 to 3 seconds. The logarithm of the power is plotted in Figure 11.4a as a function of midpoint and offset. Notice streaks of energy slicing across the (y,h)-plane at about a 45° angle. The strongest streak crosses at exactly 45° through the near offset at shot point 170. This is a missing shot, as is clearly visible in Figure 11.3. Next, think about the gas sand described as pod C in the model. Any gas-sand effect in the data should show up as a streak across all offsets at the midpoint of the gas sand—that is, horizontally across the page. I don't see such streaks in Figure 11.4a. Careful study of the figure shows that the rest of the many clearly visible streaks cut the plane at an angle noticeably *less* than $\pm 45°$. The explanation for the angle of the streaks in the figure is that they are like pod B. They are part way between the surface and the reflector. The angle determines the depth. Being closer to 45° than

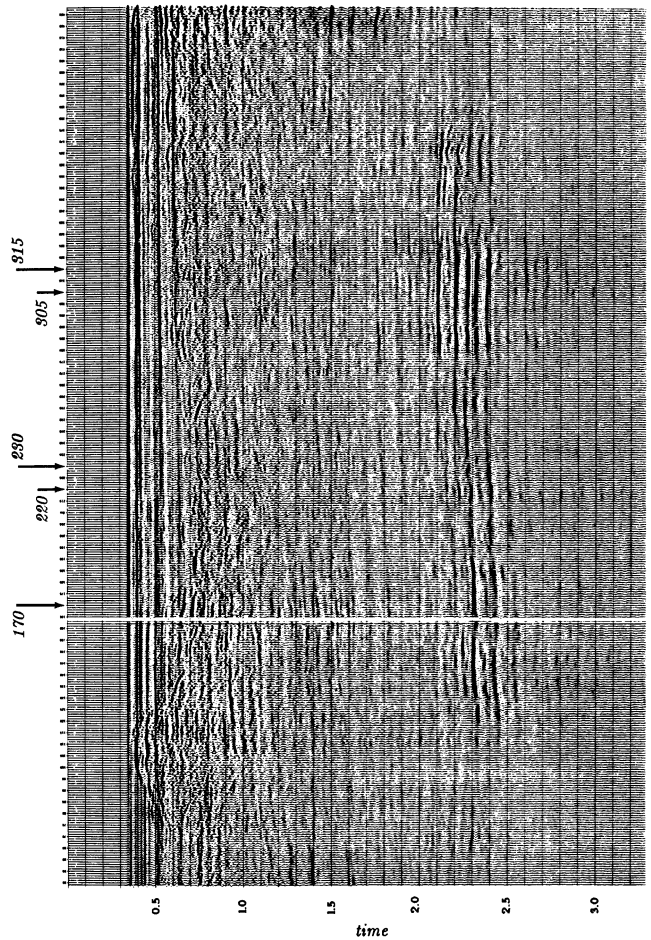


Figure 11.3: A constant-offset section across the Grand Isle gas field. The offset shown is the fifth from the near trace. (Kjartansson, Gulf) sg-kjcos [NR]

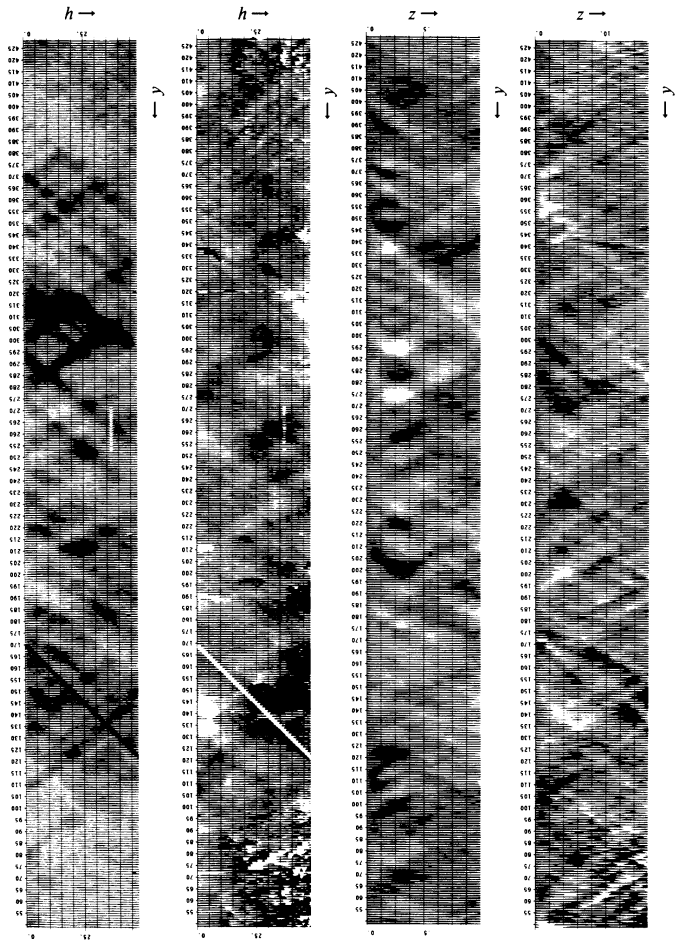


Figure 11.4: (a) amplitude (h,y), (b) timing (h,y) (c) amplitude (z,y), (d) timing (d,y) sg-kja [NR]

to 0°, the pods are closer to the surface than to the reflector.

Figure 11.4b shows timing information in the same form that Figure 11.4a shows amplitude. A CDP stack was computed, and each field seismogram was compared to it. A residual time shift for each trace was determined and plotted in Figure 11.4b. The timing residuals on one of the common-midpoint gathers is shown in Figure 11.5.

The results resemble the amplitudes, except that the results become noisy when the amplitude is low or where a "leg jump" has confounded the measurement. Figure 11.4b clearly shows that the disturbing influence on timing occurs at the same depth as that which disturbs amplitudes.

The process of *inverse* ***slant stack*** (not described in this book) enables one to determine the depth distribution of the pods. This distribution is displayed in figures 11.4c and 11.4d.

11.1.3. Rotten alligators

The sediments carried by the Mississippi River are dropped at the delta. There are sand bars, point bars, old river bows now silted in, a crow's foot of sandy distribu-

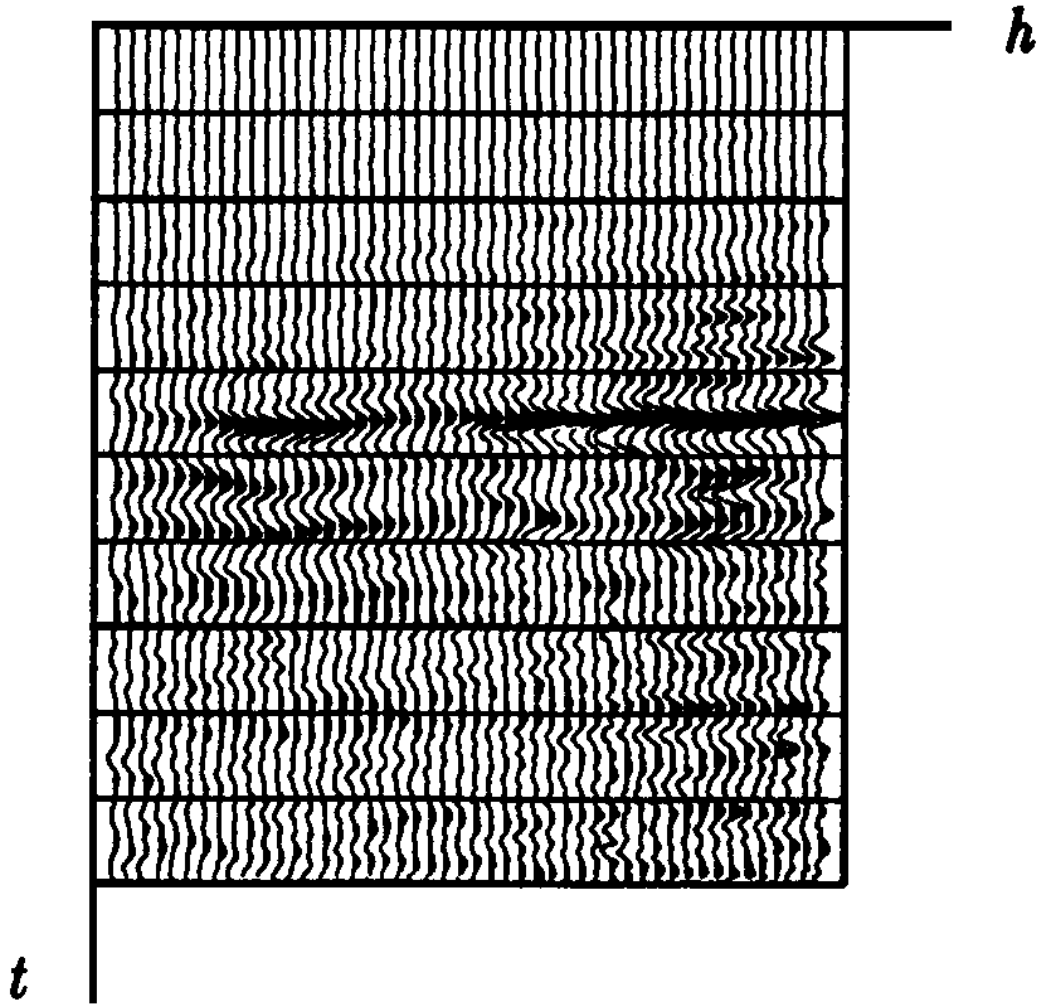


Figure 11.5: Midpoint gather 220 (same as timing of (h,y) in Figure 11.4b) after moveout. Shown is a one-second window centered at 2.3 seconds, time shifted according to an NMO for an event at 2.3 seconds, using a velocity of 7000 feet/sec. (Kjartansson) sg-kjmid [NR]

tary channels, and between channels, swampy flood plains are filled with decaying organic material. The landscape is clearly laterally variable, and eventually it will all sink of its own weight, aided by growth faults and the weight of later sedimentation. After it is buried and out of sight the lateral variations will remain as pods that will be observable by the seismologists of the future. These seismologists may see something like Figure 11.6. Figure 11.6 shows a *three-dimensional* seismic survey, that is, the ship sails many parallel lines about 70 meters apart. The top plane, a slice at constant time, shows buried river meanders.

11.1.4. Focusing or absorption?

Highly absorptive rocks usually have low velocity. Behind a low velocity pod, waves should be weakened by absorption. They should also be strengthened by focusing. Which effect dominates? How does the phenomenon depend on spatial wavelength? Maybe you can figure it out knowing that black on Figure 11.4c denotes low amplitude or high absorption, and black on Figure 11.4d denotes low velocities.

I'm inclined to believe the issue is focusing, not absorption. Even with that

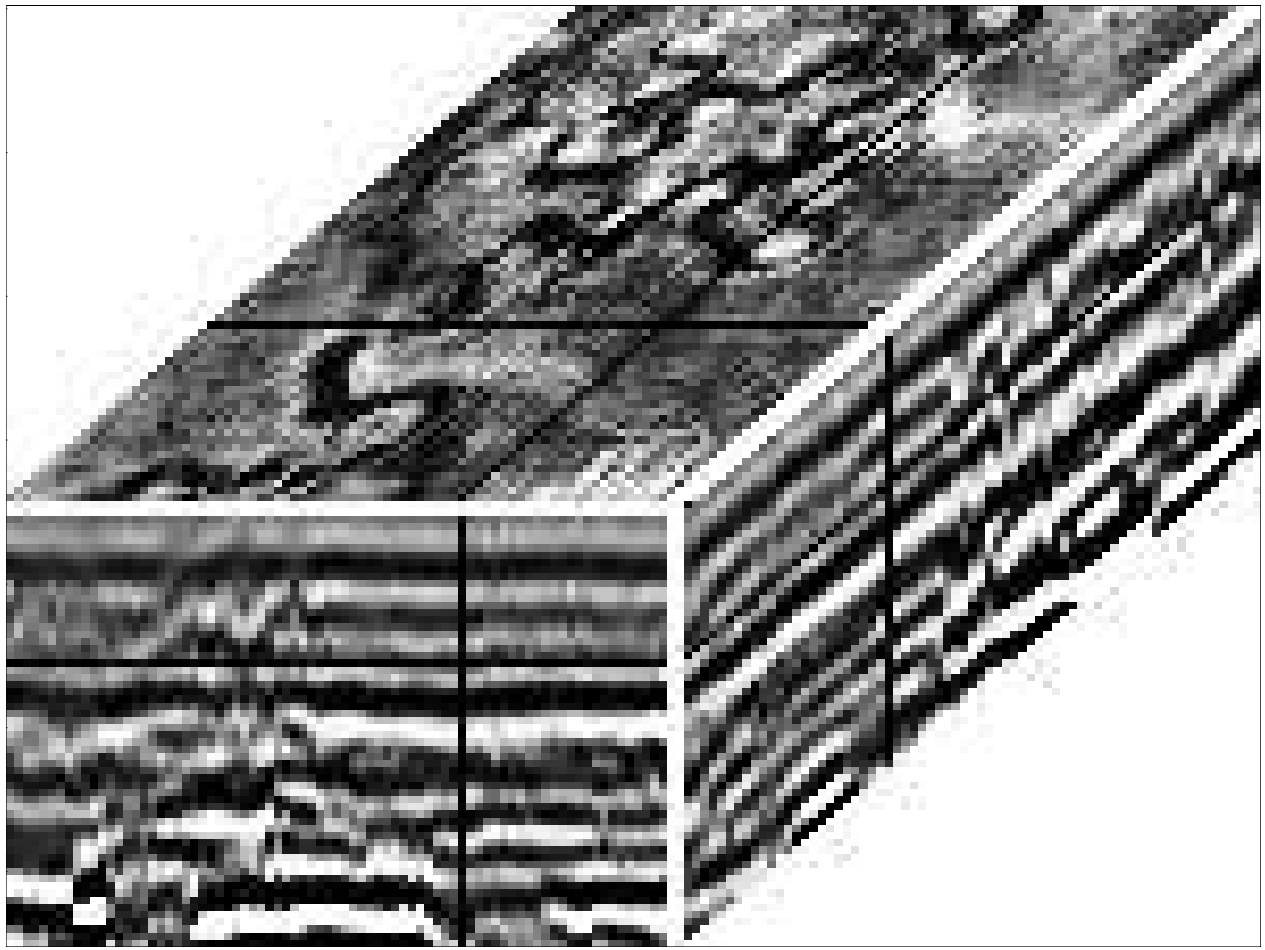


Figure 11.6: Three-dimensional seismic data from the Gulf of Thailand. Data planes from within the cube are displayed on the faces of the cube. The top plane shows ancient river meanders now submerged. (Dahm and Graebner) sg-meander [ER]

assumption, however, a reconstruction of the velocity $v(x,z)$ for this data has never been done. This falls within the realm of "reflection **tomography**", a topic too difficult to cover here. Tomography generally reconstructs a velocity model $v(x,z)$ from travel time anomalies. It is worth noticing that with this data, however, the amplitude anomalies seem to give more reliable information.

EXERCISES:

1 Consider waves converted from pressure P waves to shear S waves. Assume an S-wave speed of about half the P-wave speed. What would Figure 11.2 look like for these waves?

11.2. SEISMIC RECIPROCITY IN PRINCIPLE AND IN PRACTICE

The principle of **reciprocity** says that the same seismogram should be recorded if the locations of the source and geophone are exchanged. A physical reason for the

validity of reciprocity is that no matter how complicated a geometrical arrangement, the speed of sound along a ray is the same in either direction.

Mathematically, the reciprocity principle arises because the physical equations of elasticity are self adjoint. Thinking in terms of finite differences on a gridded earth, self-adjoint means that the matrix that translates a source anywhere to a response anywhere else is a symmetric matrix. There is a reason why such a matrix turns out to be symmetric. A product of symmetric matrices is symmetric. Running a finite difference system each time step is often a symmetric matrix, so running it many steps results in a symmetric matrix. Any actual proof is much more complicated than these few words. The final result is that very complicated electromechanical systems mixing elastic and electromagnetic waves generally fulfill the reciprocal principle. To break the reciprocal principle, you need something like a windy atmosphere so that sound going upwind has a different velocity than sound going downwind.

Anyway, since the impulse-response matrix is symmetric, elements across the matrix diagonal are equal to one another. Each element of any pair is a response to an impulsive source. The opposite element of the pair refers to an experiment where the source and receiver have had their locations interchanged.

A tricky thing about the reciprocity principle is the way antenna patterns must be handled. For example, a single vertical geophone has a natural antenna pattern. It cannot see horizontally propagating pressure waves nor vertically propagating shear waves. For reciprocity to be applicable, antenna patterns must not be interchanged when source and receiver are interchanged. The antenna pattern must be regarded as attached to the medium.

I searched our data library for split-spread land data that would illustrate reciprocity under field conditions. The constant-offset section in Figure 11.7 was recorded by vertical vibrators into vertical geophones. The survey was not intended to be a test of reciprocity, so there likely was a slight lateral offset of the source line from the receiver line. Likewise the sender and receiver arrays (clusters) may have a slightly different geometry. The earth dips in Figure 11.7 happen to be quite small although lateral velocity variation is known to be a problem in this area.

In Figure 11.8, three seismograms were plotted on top of their reciprocals. Pairs were chosen at near offset, at mid range, and at far offset. You can see that reciprocal seismograms usually have the same polarity, and often have nearly equal amplitudes. (The figure shown is the best of three such figures I prepared).

Each constant time slice in Figure 11.9 shows the reciprocity of many seis-

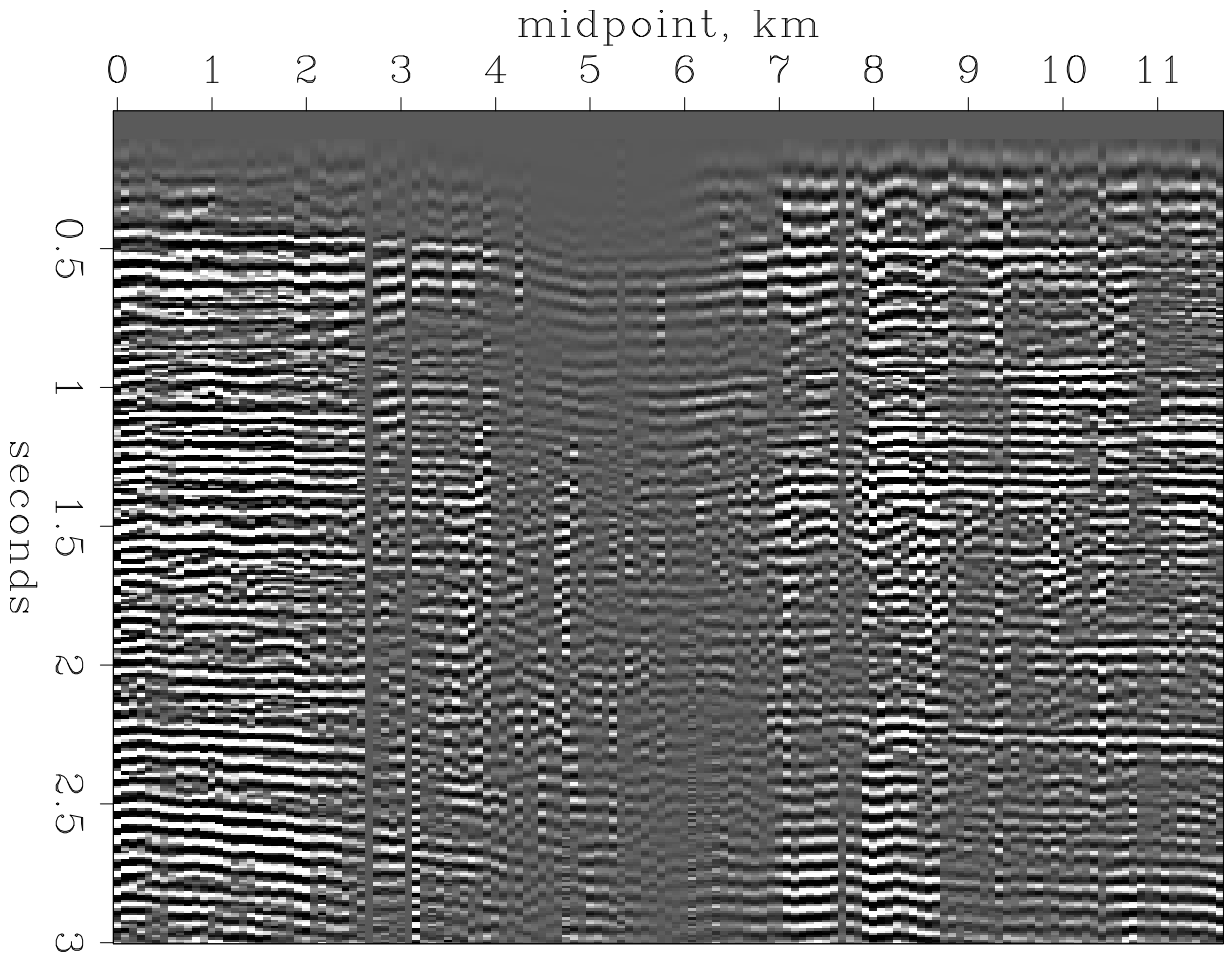


Figure 11.7: Constant-offset section from the Central Valley of California. (Chevron) sg-toldi [ER]

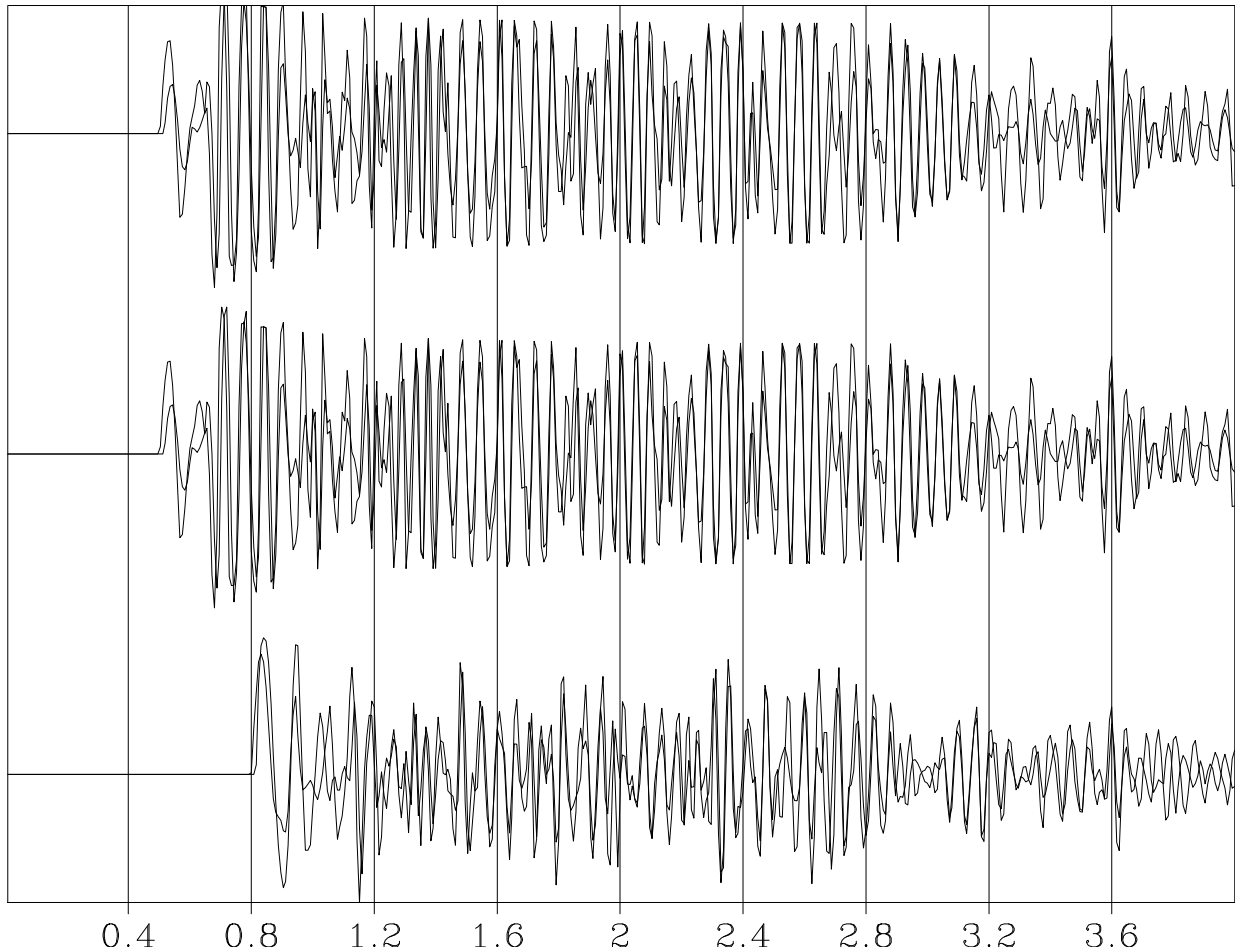


Figure 11.8: Overlain reciprocal seismograms. sg-reciptrace [ER]

mogram pairs. Midpoint runs horizontally over the same range as in Figure 11.7. Offset is vertical. Data is not recorded near the vibrators leaving a gap in the middle. To minimize irrelevant variations, moveout correction was done before making the time slices. (There is a missing source that shows up on the left side of the figure). A movie of panels like Figure 11.9 shows that the bilateral symmetry you see in the individual panels is characteristic of all times. On these slices, you notice that the long wavelengths have the expected bilateral symmetry whereas the short wavelengths do not.

In the laboratory, reciprocity can be established to within the accuracy of measurement. This can be excellent. (See White's example in FGDP). In the field, the validity of reciprocity will be dependent on the degree that the required conditions are fulfilled. A marine air gun should be reciprocal to a hydrophone. A land-surface weight-drop source should be reciprocal to a vertical geophone. But a buried explosive shot need not be reciprocal to a surface vertical geophone because the radiation patterns are different and the positions are slightly different. Under varying field conditions Fenati and Rocca found that small positioning errors in the placement of sources and receivers can easily create discrepancies much larger than the apparent reciprocity discrepancy.

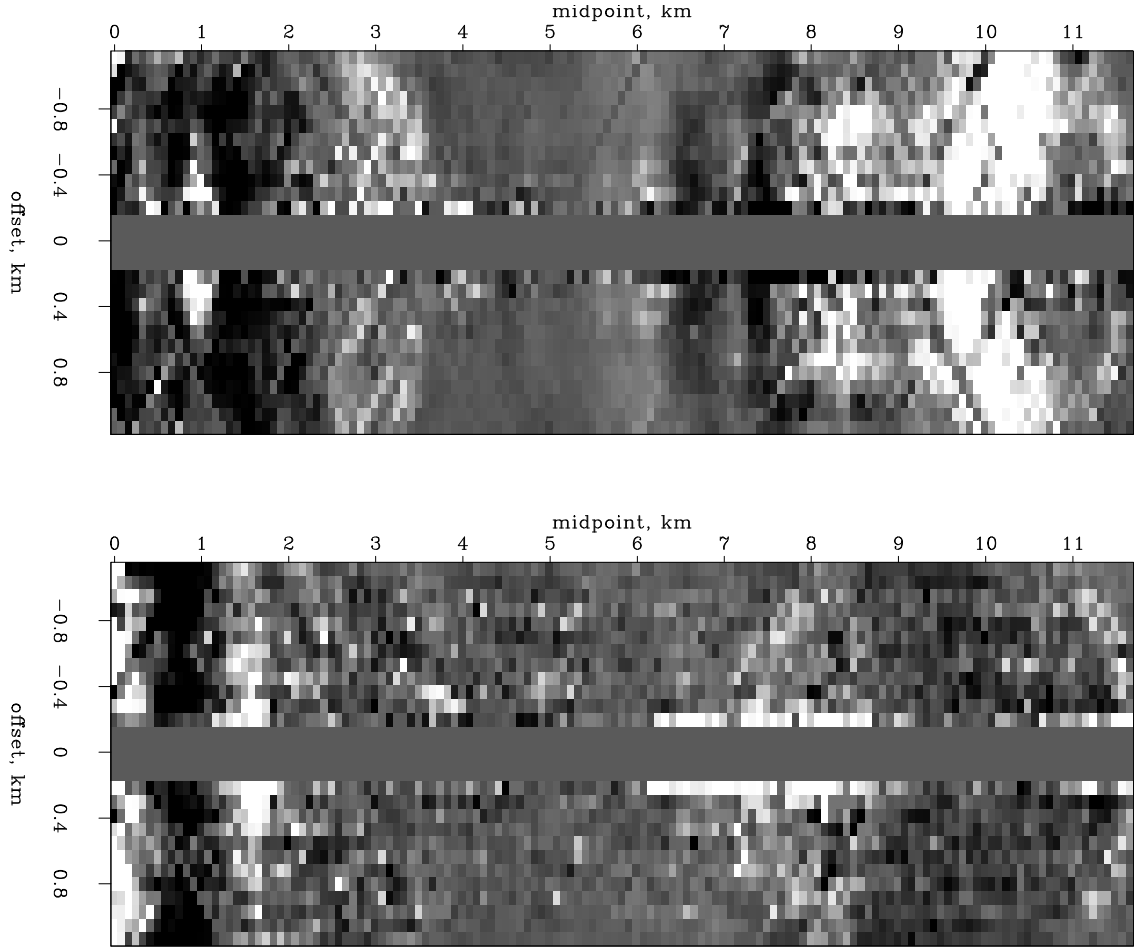


Figure 11.9: Constant time slices after NMO at 1 second and 2.5 seconds. sg-recipslice [ER]

Geometrical complexity within the earth does not diminish the applicability of the principle of linearity. Likewise, geometrical complexity does not reduce the applicability of reciprocity. Reciprocity does not apply to sound waves in the presence of **wind**. Sound goes slower upwind than downwind. But this effect of wind is much less than the mundane irregularities of field work. Just the weakening of echoes with time leaves noises that are not reciprocal. Henceforth we will presume that reciprocity is generally applicable to the analysis of reflection seismic data.

11.3. SURVEY SINKING WITH THE DSR EQUATION

Exploding-reflector imaging will be replaced by a broader imaging concept, ***survey sinking***. A new equation called the double-square-root (DSR) equation will be developed to implement survey-sinking imaging. The function of the **DSR equation** is to downward continue an entire seismic survey, not just the geophones but also the shots. Peek ahead at equation (11.13) and you will see an equation with two square roots. One represents the cosine of the wave *arrival* angle. The other represents

the *takeoff* angle at the shot. One cosine is expressed in terms of k_g, the Fourier component along the geophone axis of the data volume in (s, g, t)-space. The other cosine, with k_s, is the Fourier component along the shot axis.

11.3.1. The survey-sinking concept

The exploding-reflector concept has great utility because it enables us to associate the seismic waves observed at zero offset in many experiments (say 1000 shot points) with the wave of a single thought experiment, the exploding-reflector experiment. The exploding-reflector analogy has a few tolerable limitations connected with lateral velocity variations and multiple reflections, and one major limitation: it gives us no clue as to how to migrate data recorded at nonzero offset. A broader imaging concept is needed.

Start from field data where a survey line has been run along the x-axis. Assume there has been an infinite number of experiments, a single experiment consisting of placing a point source or shot at s on the x-axis and recording echoes with geophones at each possible location g on the x-axis. So the observed data is an upcoming wave that is a two-dimensional function of s and g, say $P(s, g, t)$.

Previous chapters have shown how to downward continue the *upcoming* wave. Downward continuation of the upcoming wave is really the same thing as downward continuation of the geophones. It is irrelevant for the continuation procedures where the wave originates. It could begin from an exploding reflector, or it could begin at the surface, go down, and then be reflected back upward.

To apply the imaging concept of survey sinking, it is necessary to downward continue the sources as well as the geophones. We already know how to downward continue geophones. Since reciprocity permits interchanging geophones with shots, we really know how to downward continue shots too.

Shots and geophones may be downward continued to different levels, and they may be at different levels during the process, but for the final result they are only required to be at the same level. That is, taking z_s to be the depth of the shots and z_g to be the depth of the geophones, the downward-continued survey will be required at all levels $z = z_s = z_g$.

The image of a reflector at (x, z) is defined to be the strength and polarity of the echo seen by the closest possible source-geophone pair. Taking the mathematical limit, this closest pair is a source and geophone located together on the reflector. The travel time for the echo is zero. This survey-sinking concept of imaging is

summarized by

$$\text{Image}(x,z) \quad = \quad \text{Wave}(s=x, g=x, z, t=0) \tag{11.1}$$

For good quality data, i.e. data that fits the assumptions of the downward-continuation method, energy should migrate to zero offset at zero travel time. Study of the energy that doesn't do so should enable improvement of the model. Model improvement usually amounts to improving the spatial distribution of velocity.

11.3.2. Survey sinking with the double-square-root equation

An equation was derived for paraxial waves. The assumption of a *single* plane wave means that the arrival time of the wave is given by a single-valued $t(x,z)$. On a plane of constant z, such as the earth's surface, Snell's parameter p is measurable. It is

$$\frac{\partial t}{\partial x} \quad = \quad \frac{\sin\theta}{v} \quad = \quad p \tag{11.2}$$

In a borehole there is the constraint that measurements must be made at a constant x, where the relevant measurement from an *upcoming* wave would be

$$\frac{\partial t}{\partial z} \quad = \quad -\frac{\cos\theta}{v} \quad = \quad -\sqrt{\frac{1}{v^2} - \left(\frac{\partial t}{\partial x}\right)^2} \tag{11.3}$$

Recall the time-shifting partial-differential equation and its solution U as some arbitrary functional form f:

$$\frac{\partial U}{\partial z} \quad = \quad -\frac{\partial t}{\partial z}\,\frac{\partial U}{\partial t} \tag{11.4}$$

$$U \quad = \quad f\left(t - \int_0^z \frac{\partial t}{\partial z}\,dz\right) \tag{11.5}$$

The partial derivatives in equation (11.4) are taken to be at constant x, just as is equation (11.3). After inserting (11.3) into (11.4) we have

$$\frac{\partial U}{\partial z} \quad = \quad \sqrt{\frac{1}{v^2} - \left(\frac{\partial t}{\partial x}\right)^2}\,\frac{\partial U}{\partial t} \tag{11.6}$$

Fourier transforming the wavefield over (x,t), we replace $\partial/\partial t$ by $-i\omega$. Likewise, for the traveling wave of the Fourier kernel $\exp(-i\omega t + ik_x x)$, constant phase means that $\partial t/\partial x = k_x/\omega$. With this, (11.6) becomes

$$\frac{\partial U}{\partial z} = -i\omega \sqrt{\frac{1}{v^2} - \frac{k_x^2}{\omega^2}}\; U \tag{11.7}$$

The solutions to (11.7) agree with those to the scalar wave equation unless v is a function of z, in which case the scalar wave equation has both upcoming and downgoing solutions, whereas (11.7) has only upcoming solutions. We go into the lateral space domain by replacing ik_x by $\partial/\partial x$. The resulting equation is useful for superpositions of many local plane waves and for lateral velocity variations $v(x)$.

11.3.3. The DSR equation in shot-geophone space

Let the geophones descend a distance dz_g into the earth. The change of the travel time of the observed upcoming wave will be

$$\frac{\partial t}{\partial z_g} \quad = \quad -\sqrt{\frac{1}{v^2} - \left(\frac{\partial t}{\partial g}\right)^2} \tag{11.8}$$

Suppose the shots had been let off at depth dz_s instead of at $z = 0$. Likewise then,

$$\frac{\partial t}{\partial z_s} \quad = \quad -\sqrt{\frac{1}{v^2} - \left(\frac{\partial t}{\partial s}\right)^2} \tag{11.9}$$

Both (11.8) and (11.9) require minus signs because the travel time decreases as either geophones or shots move down.

Simultaneously downward project both the shots and geophones by an identical vertical amount $dz = dz_g = dz_s$. The travel-time change is the sum of (11.8) and (11.9), namely,

$$dt \quad = \quad \frac{\partial t}{\partial z_g}\, dz_g + \frac{\partial t}{\partial z_s}\, dz_s \quad = \quad \left(\frac{\partial t}{\partial z_g} + \frac{\partial t}{\partial z_s}\right) dz \tag{11.10}$$

or

$$\frac{\partial t}{\partial z} = -\left(\sqrt{\frac{1}{v^2} - \left(\frac{\partial t}{\partial g}\right)^2} + \sqrt{\frac{1}{v^2} - \left(\frac{\partial t}{\partial s}\right)^2} \right) \tag{11.11}$$

This expression for $\partial t/\partial z$ may be substituted into equation (11.4):

$$\frac{\partial U}{\partial z} = +\left(\sqrt{\frac{1}{v^2} - \left(\frac{\partial t}{\partial g}\right)^2} + \sqrt{\frac{1}{v^2} - \left(\frac{\partial t}{\partial s}\right)^2} \right) \frac{\partial U}{\partial t} \tag{11.12}$$

Three-dimensional Fourier transformation converts upcoming wave data $u(t,s,g)$ to $U(\omega,k_s,k_g)$. Expressing equation (11.12) in Fourier space gives

$$\frac{\partial U}{\partial z} = -i\,\omega \left[\sqrt{\frac{1}{v^2} - \left(\frac{k_g}{\omega}\right)^2} + \sqrt{\frac{1}{v^2} - \left(\frac{k_s}{\omega}\right)^2} \right] U \tag{11.13}$$

Recall the origin of the two square roots in equation (11.13). One is the cosine of the arrival angle at the geophones divided by the velocity at the geophones. The other is the cosine of the takeoff angle at the shots divided by the velocity at the shots. With

the wisdom of previous chapters we know how to go into the lateral space domain by replacing ik_g by $\partial/\partial g$ and ik_s by $\partial/\partial s$. To incorporate lateral velocity variation $v(x)$, the velocity at the shot location must be distinguished from the velocity at the geophone location. Thus,

$$\frac{\partial U}{\partial z} = \left[\sqrt{\left(\frac{-i\omega}{v(g)}\right)^2 - \frac{\partial^2}{\partial g^2}} + \sqrt{\left(\frac{-i\omega}{v(s)}\right)^2 - \frac{\partial^2}{\partial s^2}} \right] U \quad (11.14)$$

Equation (11.14) is known as the double-square-root (DSR) equation in shot-geophone space. It might be more descriptive to call it the survey-sinking equation since it pushes geophones and shots downward together. Recalling the section on splitting and full separation we realize that the two square-root operators are commutative ($v(s)$ commutes with $\partial/\partial g$), so it is completely equivalent to downward continue shots and geophones separately or together. This equation will produce waves for the rays that are found on zero-offset sections but are absent from the exploding-reflector model.

11.3.4. The DSR equation in midpoint-offset space

By converting the DSR equation to midpoint-offset space we will be able to identify the familiar zero-offset migration part along with corrections for offset. The transformation between (g,s) recording parameters and (y,h) interpretation parameters is

$$y = \frac{g+s}{2} \tag{11.15}$$

$$h = \frac{g-s}{2} \tag{11.16}$$

Travel time t may be parameterized in (g,s)-space or (y,h)-space. Differential relations for this conversion are given by the chain rule for derivatives:

$$\frac{\partial t}{\partial g} = \frac{\partial t}{\partial y}\frac{\partial y}{\partial g} + \frac{\partial t}{\partial h}\frac{\partial h}{\partial g} = \frac{1}{2}\left(\frac{\partial t}{\partial y} + \frac{\partial t}{\partial h}\right) \tag{11.17}$$

$$\frac{\partial t}{\partial s} = \frac{\partial t}{\partial y}\frac{\partial y}{\partial s} + \frac{\partial t}{\partial h}\frac{\partial h}{\partial s} = \frac{1}{2}\left(\frac{\partial t}{\partial y} - \frac{\partial t}{\partial h}\right) \tag{11.18}$$

Having seen how stepouts transform from shot-geophone space to midpoint-

offset space, let us next see that spatial frequencies transform in much the same way. Clearly, data could be transformed from (s,g)-space to (y,h)-space with (11.15) and (11.16) and then Fourier transformed to (k_y,k_h)-space. The question is then, what form would the double-square-root equation (11.13) take in terms of the spatial frequencies (k_y,k_h)? Define the seismic data field in either coordinate system as

$$U(s,g) \quad = \quad U'(y,h) \tag{11.19}$$

This introduces a new mathematical function U' with the same physical meaning as U but, like a computer subroutine or function call, with a different subscript look-up procedure for (y,h) than for (s,g). Applying the chain rule for partial differentiation to (11.19) gives

$$\frac{\partial U}{\partial s} \quad = \quad \frac{\partial y}{\partial s}\,\frac{\partial U'}{\partial y} + \frac{\partial h}{\partial s}\,\frac{\partial U'}{\partial h} \tag{11.20}$$

$$\frac{\partial U}{\partial g} \quad = \quad \frac{\partial y}{\partial g}\,\frac{\partial U'}{\partial y} + \frac{\partial h}{\partial g}\,\frac{\partial U'}{\partial h} \tag{11.21}$$

and utilizing (11.15) and (11.16) gives

$$\frac{\partial U}{\partial s} = \frac{1}{2}\left(\frac{\partial U'}{\partial y} - \frac{\partial U'}{\partial h}\right) \tag{11.22}$$

$$\frac{\partial U}{\partial g} = \frac{1}{2}\left(\frac{\partial U'}{\partial y} + \frac{\partial U'}{\partial h}\right) \tag{11.23}$$

In Fourier transform space where $\partial/\partial x$ transforms to ik_x, equations (11.22) and (11.23), when i and $U = U'$ are cancelled, become

$$k_s = \frac{1}{2}(k_y - k_h) \tag{11.24}$$

$$k_g = \frac{1}{2}(k_y + k_h) \tag{11.25}$$

Equations (11.24) and (11.25) are Fourier representations of (11.22) and (11.23). Substituting (11.24) and (11.25) into (11.13) achieves the main purpose of this section, which is to get the double-square-root migration equation into midpoint-offset

coordinates:

$$\frac{\partial}{\partial z} U \;=\; -i \frac{\omega}{v} \left[\sqrt{1 - \left(\frac{v k_y + v k_h}{2\omega} \right)^2} + \sqrt{1 - \left(\frac{v k_y - v k_h}{2\omega} \right)^2} \right] U \tag{11.26}$$

Equation (11.26) is the takeoff point for many kinds of common-midpoint seismogram analyses. Some convenient definitions that simplify its appearance are

$$G \;=\; \frac{v\, k_g}{\omega} \tag{11.27}$$

$$S \;=\; \frac{v\, k_s}{\omega} \tag{11.28}$$

$$Y \;=\; \frac{v\, k_y}{2\, \omega} \tag{11.29}$$

$$H \;=\; \frac{v\, k_h}{2\, \omega} \tag{11.30}$$

The new definitions S and G are the sines of the takeoff angle and of the arrival angle of a ray. When these sines are at their limits of ± 1 they refer to the steepest

possible slopes in (s,t)- or (g,t)-space. Likewise, Y may be interpreted as the dip of the data as seen on a seismic section. The quantity H refers to stepout observed on a common-midpoint gather. With these definitions (11.26) becomes slightly less cluttered:

$$\frac{\partial}{\partial z}U = -\frac{i\omega}{v}\left(\sqrt{1-(Y+H)^2}+\sqrt{1-(Y-H)^2}\right)U \tag{11.31}$$

EXERCISES:

1 Adapt equation (11.26) to allow for a difference in velocity between the shot and the geophone.

2 Adapt equation (11.26) to allow for downgoing pressure waves and upcoming shear waves.

11.4. THE MEANING OF THE DSR EQUATION

The double-square-root equation is not easy to understand because it is an operator in a four-dimensional space, namely, (z, s, g, t). We will approach it through various applications, each of which is like a picture in a space of lower dimension. In this section lateral velocity variation will be neglected (things are bad enough already!).

One way to reduce the dimensionality of (11.14) is simply to set $H = 0$. Then the two square roots become the same, so that they can be combined to give the familiar paraxial equation:

$$\frac{dU}{dz} \quad = \quad -i\omega \, \frac{2}{v} \sqrt{1 - \frac{v^2 k_y^2}{4\omega^2}} \; U \tag{11.32}$$

In both places in equation (11.32) where the rock velocity occurs, the rock velocity is divided by 2. Recall that the rock velocity needed to be halved in order for field data to correspond to the exploding-reflector model. So whatever we did by setting $H = 0$, gave us the same migration equation we used in chapter 7. Setting $H = 0$ had the effect of making the survey-sinking concept functionally equivalent to the exploding-reflector concept.

11.4.1. Zero-dip stacking (Y = 0)

When dealing with the offset h it is common to assume that the earth is horizontally layered so that experimental results will be independent of the midpoint y. With such an earth the Fourier transform of all data over y will vanish except for $k_y = 0$, or, in other words, for $Y = 0$. The two square roots in (11.14) again become identical, and the resulting equation is once more the paraxial equation:

$$\frac{dU}{dz} = -i\omega \frac{2}{v} \sqrt{1 - \frac{v^2 k_h^2}{4\omega^2}}\ U \tag{11.33}$$

Using this equation to downward continue hyperboloids from the earth's surface, we find the hyperboloids shrinking with depth, until the correct depth where best focus occurs is reached. This is shown in Figure 11.10.

The waves focus best at zero offset. The focus represents a downward-continued experiment, in which the downward continuation has gone just to a reflector. The reflection is strongest at zero travel time for a coincident source-receiver pair just above the reflector. Extracting the zero-offset value at $t = 0$ and abandoning the other offsets is a way of eliminating noise. (Actually it is a way of *defining* noise). Roughly it amounts to the same thing as the conventional procedure of summation

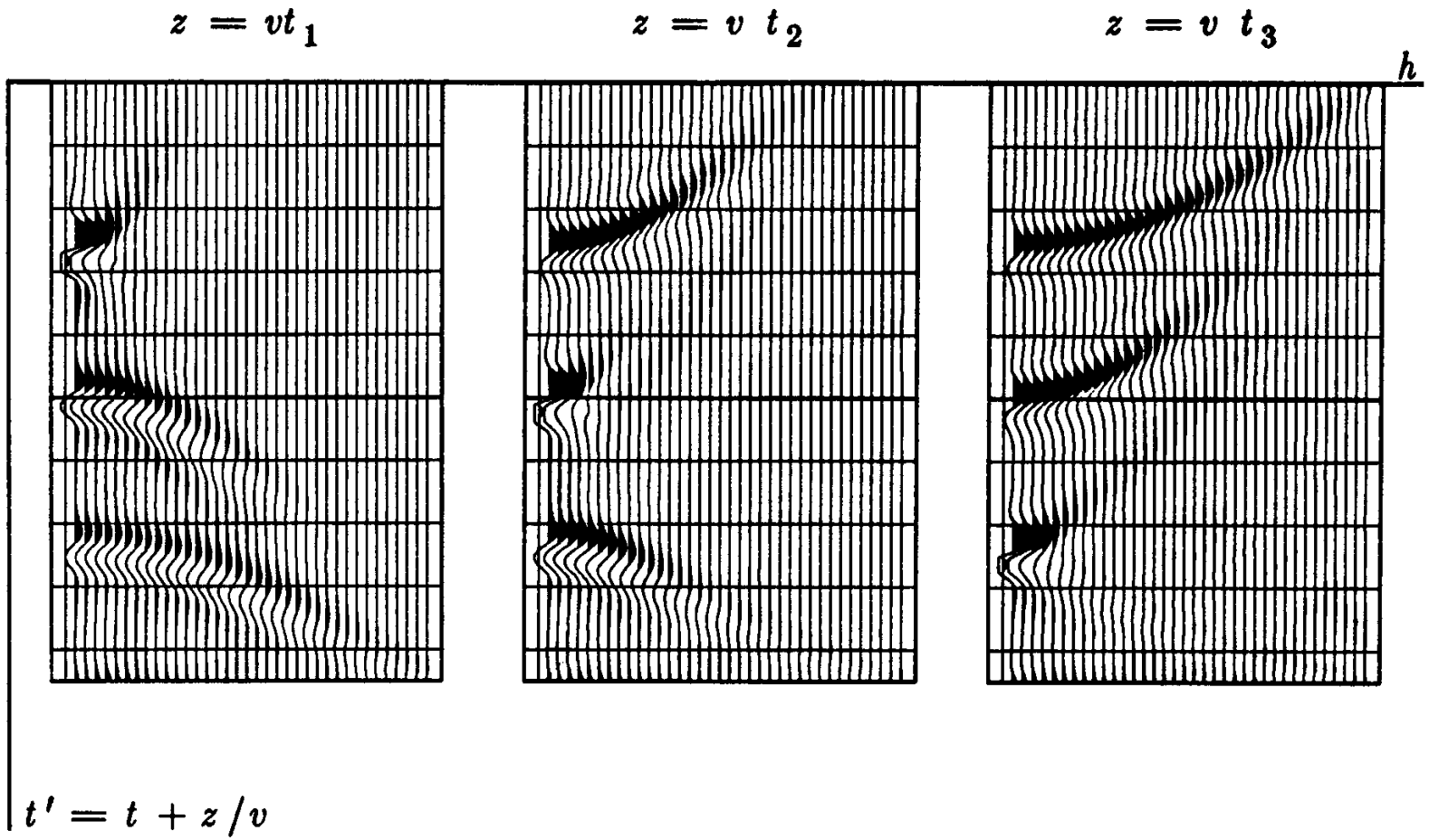


Figure 11.10: With an earth model of three layers, the common-midpoint gathers are three hyperboloids. Successive frames show downward continuation to successive depths where best focus occurs. sg-dc2 [NR]

along a hyperbolic trajectory on the original data. Naturally the summation can be expected to be best when the velocity used for downward continuation comes closest to the velocity of the earth. Later, offset space will be used to *determine* velocity.

11.4.2. Giving up on the DSR

The DSR operator defined by (11.14) is fun to think about, but it doesn't really go to any very popular place very easily. There is a serious problem with it. It is *not separable* into a sum of an offset operator and a midpoint operator. *Nonseparable* means that a Taylor series for (11.14) contains terms like $Y^2 H^2$. Such terms cannot be expressed as a function of Y plus a function of H. Nonseparability is a data-processing disaster. It implies that migration and stacking must be done simultaneously, not sequentially. The only way to recover pure separability would be to return to the space of S and G.

This chapter tells us that lateral velocity variation is very important. Where the velocity is known, we have the DSR equation in shot-geophone space to use for migration. A popular test data set is called the Marmousi data set. The DSR

equation is particularly popular with it because with synthetic data, the velocity really is known. Estimating velocity $v(x,z)$ with real data is a more difficult task, one that is only crudely handled by by methods in this book. In fact, it is not easily done by the even best of current industrial practice.

Chapter 12

RATional FORtran == Ratfor

Bare-bones **Fortran** is our most universal computer language for computational physics. For general programming, however, it has been surpassed by **C**. "**Ratfor**" is Fortran with C-like syntax. I believe Ratfor is the best available expository language

for mathematical algorithms. Ratfor was invented by the people who invented C. Ratfor programs are converted to Fortran with the Ratfor **preprocessor**. Since the preprocessor is publicly available, Ratfor is practically as universal as Fortran.[1]

You will not really need the Ratfor preprocessor or any precise definitions if you already know Fortran or almost any other computer language, because then the Ratfor language will be easy to understand. Statements on a line may be separated by ";". Statements may be grouped together with braces { }. Do loops do not require statement numbers because { } defines the range. Given that `if( )` is true, the statements in the following { } are done. `else{ }` does what you expect. We may *not* contract `else if` to `elseif`. We may always omit the braces { } when they contain only one statement. `break` will cause premature termination of the enclosing { }. `break 2` escapes from {{ }}. `while( ) { }` repeats the statements

---

[1]Kernighan, B.W. and Plauger, P.J., 1976, Software Tools: Addison-Wesley. Ratfor was invented at AT&T, which makes it available directly or through many computer vendors. The original Ratfor transforms Ratfor code to **Fortran** 66. See `http://sepwww.stanford.edu/sep/prof` for a public-domain Ratfor translator to **Fortran** 77.

in { } while the condition ( ) is true. `repeat { ... } until( )` is a loop that tests at the bottom. A looping statement more general than `do` is `for(initialize; condition; reinitialize) { }`. An example of one equivalent to `do i=0,n-1` is the looping statement `for(i=0;i<n;i=i+i)`. The statement `next` causes skipping to the end of any loop and a retrial of the test condition. `next` is rarely used, but when it is, we must beware of an inconsistency between Fortran and C-language. Where Ratfor uses `next`, the C-language uses `continue` (which in Ratfor and Fortran is merely a place holder for labels). The Fortran relational operators `.gt.`, `.ge.`, `.ne.`, etc. may be written >, >=, !=, etc. The logical operators `.and.` and `.or.` may be written & and |. Anything from a # to the end of the line is a comment. Anything that does not make sense to the Ratfor preprocessor, such as Fortran input-output, is passed through without change. (Ratfor has a `switch` statement but we never use it because it conflicts with the `implicit undefined` declaration. Anybody want to help us fix the switch in public domain Ratfor?)

Indentation in `Ratfor` is used for readability. It is not part of the `Ratfor` language. Choose your own style. I have overcondensed. There are two **pitfall**s associated with indentation. The beginner's pitfall is to assume that a `do` loop ends where the indentation ends. The loop ends after the first statement. A larger scope

for the `do` loop is made by enclosing multiple statements in braces. The other pitfall arises in any construction like `if() ... if() ... else`. The `else` goes with the last `if()` regardless of indentation. If you want the `else` with the earlier `if()`, you must use braces like `if() { if() ... } else ...`.

The most serious limitation of **Fortran-**77 is its lack of ability to allocate temporary memory. I have written a **preprocessor** to **Ratfor** or **Fortran** to overcome this memory-allocation limitation. This program, named **`sat`**, allows subroutines to include the declaration `temporary real data(n1,n2)`, so that memory is allocated during execution of the subroutine where the declaration is written. Fortran-77 forces us to accomplish something like this More recently Bob Clapp has prepared Ratfor90, a Perl-based preprocessor to Fortran 90 that incorporates the desireable features of both ratfor and Fortran 90 and is backward compatible to the codes of this book.

Chapter 13

Seplib and SEP software

Most of the seismic utility software at **SEP**[1] Stanford Exploration Project (**SEP**) software handles seismic data as a rectangular lattice or "cube" of numbers. Each

[1] Old reports of the Stanford Exploration Project can be found in the library of the Society of Exploration Geophysicists in Tulsa, Oklahoma.

cube-processing program appends to the history file for the cube. Preprocessors extend **Fortran** (or **Ratfor**) to enable it to allocate memory at run time, to facilitate input and output of data cubes, and to facilitate self-documenting programs.

At **SEP**, a library of subroutines known as **`seplib`** evolved for routine operations. These subroutines mostly handle data in the form of cubes, planes, and vectors. A cube is defined by 14 parameters with standard names and two files: one the data cube itself, and the other containing the 14 parameters and a history of the life of the cube as it passed through a sequence of cube-processing programs. Most of these cube-processing programs have been written by researchers, but several nonscientific cube programs have become highly developed and are widely shared. Altogether there are (1) a library of subroutines, (2) a library of main programs, (3) some naming conventions, and (4) a graphics library called **`vplot`**. The subroutine library has good manual pages. The main programs rarely have manual pages, their documentation being supplied by the on-line self-documentation that is extracted from the comments at the beginning of the source file. Following is a list of the names of popular main programs:

`Byte`	Scale floats to brightness bytes for raster display.
`Cat`	Concatenate conforming cubes along the 3-axis.

`Contour`	Contour plot a plane.
`Cp`	Copy a cube.
`Dd`	Convert between ASCI, floats, complex, bytes, etc.
`Dots`	Plot a plane of floats.
`Ft3d`	Do three-dimensional Fourier transform.
`Graph`	Plot a line of floats.
`In`	Check the validity of a data cube.
`Merge`	Merge conforming cubes side by side on any axis.
`Movie`	View a cube with Rick Ottolini's cube viewer.
`Noise`	Add noise to data.
`Reverse`	Reverse a cube axis.
`Spike`	Make a plane wave of synthetic data.
`Ta2vplot`	Convert a byte format to raster display with `vplot`.
`Tpow`	Scale data by a power of time *t* (1-axis).
`Thplot`	Make a hidden line plot.
`Transpose`	Transpose cube axes.
`Tube`	View a `vplot` file on a screen.
`Wiggle`	Plot a plane of floats as "wiggle traces."

`Window` Find a subcube by truncation or subsampling.

To use the cube-processing programs, read this document, and then for each command, read its on-line self-documentation. To write cube-processing programs, read the manual page for `seplib` and the subroutines mentioned there and here. To write `vplot` programs, see the references on `vplot`.

13.1. THE DATA CUBE

The data cube itself is like a Fortran three-dimensional matrix. Its location in the computer file system is denoted by `in=PATHNAME`, where `in=` is the literal occurrence of those three characters, and `PATHNAME` is a directory tree location like `/data/western73.F`. Like the Fortran cube, the data cube can be real, complex, double precision, or byte, and these cases are distinguished by the element size in bytes. Thus the history file contains one of `esize=4`, `esize=8`, or `esize=1`, respectively. Embedded blanks around the "=" are always forbidden. The cube values are binary information; they cannot be printed or edited (without the intervention of something like a Fortran "format"). To read and write cubes, see the manual pages for such routines as `reed, sreed, rite, srite, snap`.

A cube has three axes. The number of points on the 1-axis is `n1`. A Fortran declaration of a cube could be `real mydata(n1,n2,n3)`. For a plane, `n3=1`, and for a line, `n2=1`. In addition, many programs take "1" as the default for an undefined value of `n2` or `n3`. The physical location of the single data value `mydata(1,1,1)`, like a mathematical origin (o_1, o_2, o_3), is denoted by the three real variables `o1`, `o2`, and `o3`. The data-cube values are presumed to be uniformly spaced along these axes like the mathematical increments $(\Delta_1, \Delta_2, \Delta_3)$, which may be negative and are denoted by the three real variables `d1`, `d2`, and `d3`. Each axis has a label, and naturally these labels are called `label1`, `label2`, and `label3`. Examples of labels are `kilometers`, `sec`, `Hz`, and `"offset, km"`. Most often, `label1="time, sec"`. Altogether that is $2 + 3 \times 4$ parameters, and there is an optional title parameter that is interpreted by most of the plot programs. An example is `title="Yilmaz and Cumro Canada profile 25"`. We reserve the names `n4,o4,d4, and label4` (a few programs support them already), and please do not use `n5` etc. for anything but a five-dimensional cubic lattice.

13.2. THE HISTORY FILE

The 15 parameters above, and many more parameters defined by authors of cube-processing programs, are part of the "**history file**" (which is ASCI, so we can print it). A great many cube-processing programs are simple filters—i.e., one cube goes in and one cube comes out—and that is the case I will describe in detail here. For other cases, such as where two go in and one comes out, or none go in and one comes out (synthetic data), or one goes in and none come out (plotting program), I refer you to the manual pages, particularly to subroutine names beginning with `aux` (as in "auxiliary").

Let us dissect an example of a simple cube-processing program and its use. Suppose we have a seismogram in a data cube and we want only the first 500 points on it, i.e., the first 500 points on the 1-axis. A utility cube filter named `Window` will do the job. Our command line looks like `< mygiven.H Window n1=500 > myshort.H` On this command line, `mygiven.H` is the name of the history file of the data we are given, and `myshort.H` is the history file we will create. The moment `Window`, or any other `seplib` program, begins, it copies `mygiven.H` to `myshort.H`; from then on, information can only be appended to `myshort.H`. When `Window` learns that we want the 1-axis on our output cube to be 500, it does `call`

putch('n1','i',500), which appends n1=500 to myshort.H. But before this, some other things happen. First, seplib's internals will get our log-in name, the date, the name of the computer we are using, and Window's name (which is Window), and append these to myshort.H. The internals will scan mygiven.H for in=somewhere to find the input data cube itself, and will then figure out where we want to keep the output cube. Seplib will guess that someone named professor wants to keep his data cube at some place like /scr/professor/_Window.H@. You should read the manual page for datapath to see how you can set up the default location for your datasets. The reason datapath exists is to facilitate isolating data from text, which is usually helpful for archiving.

When a cube-processing filter wonders what the value is of n1 for the cube coming in, it makes a subroutine call like call hetch("n1","i",n1). The value returned for n1 will be the *last* value of n1 found on the history file. Window also needs to find a different n1, the one we put on the command line. For this it will invoke something like call getch("n1","i",n1out). Then, so the next user will know how big the output cube is, it will call putch("n1","i",n1out). For more details, see the manual pages.

If we want to take input parameters from a file instead of from the command

line, we type something like `<in.H Window par=myparfile.p > out.H`. The `.p` is my naming convention and is wholly optional, as is the `.H` notation for a history file.

`Sepcube` programs are self-documenting. When you type the name of the program with no input cube and no command-line arguments, you should see the self-documentation (which comes from the initial comment lines in the program).

SEP software supports "pipelining." For example, we can slice a plane out of a data cube, make a contour plot, and display the plot, all with the command line `<in.H Window n3=1 | Contour | Tube` where, as in UNIX pipes, the "`|`" denotes the passage of information from one program to the next. Pipelining is a convenience for the user because it saves defining a location for necessary intermediate files. The history files do flow down UNIX pipes. You may not have noticed that some location had to be assigned to the data at the intermediate stages, and when you typed the pipeline above, you were spared that clutter. To write `seplib` programs that allow pipelining, you need to read the manual page on `hclose()` to keep the history file from intermingling with the data cube itself.

A sample history file follows: this was an old one, so I removed a few anachronisms manually.

```
# Texaco Subduction Trench: read from tape by Bill Harlan
n1=1900 n2=2274
o1=2.4 it0=600 d1=.004 d2=50.  in=/d5/alaska
Window:   bill   Wed Apr 13 14:27:57 1983
input() :    in ="/d5/alaska"
output() : sets next in="/q2/data/Dalw"
Input:  float Fortran (1900,2274,1)
Output: float Fortran (512,128,1)
 n1=512 n2=128 n3=1
Swab:   root@mazama   Mon Feb 17 03:23:08 1986
# input history file /r3/q2/data/Halw
input() :    in ="/q2/data/Dalw"
output() : sets next in="/q2/data/Dalw_002870_Rcp"
#ibs=8192 #obs=8192
Rcp: paul Mon Feb 17 03:23:15 PST 1986
Copying from mazama:/r3/q2/data/Halw
to hanauma:/q2/data/Halw
in="/q2/data/Dalw"
```

```
Cp:   jon@hanauma   Wed Apr  3 23:18:13 1991
input() :    in ="/q2/data/Dalw"
output() : sets next in="/scr/jon/_junk.H@"
```

13.3. MEMORY ALLOCATION

Everything below is for Fortran 77. This approach still works, but has been superceded by a backward compatible Fortran 90 preprocessor by Bob Clapp which is called Ratfor90. `Sepcube` programs can be written in Fortran, Ratfor, or C. A serious problem with **Fortran-**77 (and hence Ratfor) is that memory cannot be allocated for arrays whose size is determined at run time. We have worked around this limitation by using two home-grown preprocessors, one called **`saw`** (Stanford Auto Writer) for main programs, and one called **`sat`** (Stanford Auto Temporaries) for subroutines. Both preprocessors transform either Fortran or Ratfor.

13.3.1. Memory allocation in subroutines with sat

The **sat** preprocessor allows us to declare temporary arrays of arbitrary dimension, such as `temporary real*4 data(n1,n2,n3), convolution(j+k-1)` These declarations must follow other declarations and precede the executable statements.

13.3.2. The main program environment with saw

The **saw** preprocessor also calls an essential initialization routine `initpar()`, organizes the self-doc, and simplifies data-cube input. See the on-line self-documentation or the manual pages for full details. Following is a complete **saw** program for a simple task:

```
# <in.H  Scale scaleval=1. > out.H
#
# Copy input to output and scale by scaleval
# keyword generic scale
#%
integer n1, n2, n3, esize
from history: integer n1, n2, n3, esize
```

```
if (esize !=4) call erexit('esize != 4')
allocate: real x(n1,n2)
subroutine scaleit( n1,n2, x)
integer i1,i2, n1,n2
real x(n1,n2), scaleval
from par: real scaleval=1.
call hclose() # no more parameter handling.
call sreed('in', x, 4*n1*n2)
do i1=1,n1
do i2=1,n2
x(i1,i2) = x(i1,i2) * scaleval
call srite( 'out', x, 4*n1*n2)
return; end
```

13.4. SHARED SUBROUTINES

The following smoothing subroutines are described in PVI and used in both PVI and BEI. boxconv triangle triangle2

```
subroutine boxconv( nb, nx, xx, yy)
# inputs:       nx,  xx(i), i=1,nx      the data
#               nb                      the box length
# output:       yy(i),i=1,nx+nb-1       smoothed data
integer nx, ny, nb, i
real xx(nx), yy(1)
temporary real bb(nx+nb)
# "||" means .OR.
if( nb < 1 || nb > nx)  call erexit('boxconv')
ny = nx+nb-1
do i= 1, ny
        bb(i) = 0.
bb(1) = xx(1)
do i= 2, nx
                bb(i) = bb(i-1) + xx(i)         # make B(Z) = X(Z)/(1-Z)
do i= nx+1, ny
                bb(i) = bb(i-1)
do i= 1, nb
                yy(i) = bb(i)
do i= nb+1, ny
                yy(i) = bb(i) - bb(i-nb)        # make Y(Z) = B(Z)*(1-Z**nb)
do i= 1, ny
                yy(i) = yy(i) / nb
return; end
```

Back

```
# Convolve with triangle
#
subroutine triangle( nr, m1, n12, uu, vv)
# input:        nr      rectangle width (points) (Triangle base twice as wide.)
# input:        uu(m1,i2),i2=1,n12      is a vector of data.
# output:       vv(m1,i2),i2=1,n12      may be on top of uu
integer nr,m1,n12, i,np,nq
real uu( m1, n12),  vv( m1, n12)
temporary real pp(n12+nr-1), qq(n12+nr+nr-2), tt(n12)
do i=1,n12 {    qq(i) = uu(1,i) }
if( n12 == 1 )
        do i= 1, n12
                tt(i) = qq(i)
else {
        call boxconv( nr, n12, qq, pp);         np = nr+n12-1
        call boxconv( nr, np , pp, qq);         nq = nr+np-1
        do i= 1, n12
                        tt(i) = qq(i+nr-1)
        do i= 1, nr-1                                   # fold back near end
                        tt(i) = tt(i) + qq(nr-i)
        do i= 1, nr-1                                   # fold back far end
                        tt(n12-i+1) = tt(n12-i+1) + qq(n12+(nr-1)+i)
        }
do i=1,n12 {    vv(1,i) = tt(i) }
return; end
```

Back

```
# smooth by convolving with triangle in two dimensions.
#
subroutine triangle2( rect1, rect2, n1, n2, uu, vv)
integer i1,i2,        rect1, rect2, n1, n2
real uu(n1,n2), vv(n1,n2)
temporary real ss(n1,n2)
do i1= 1, n1
                call triangle( rect2, n1, n2, uu(i1,1), ss(i1,1))
do i2= 1, n2
                call triangle( rect1,  1, n1, ss(1,i2), vv(1,i2))
return; end
```

Back

13.5. REFERENCES

Claerbout, J., 1990, Introduction to `seplib` and SEP utility software: **SEP–70**, 413–436.

Cole, S., and Dellinger, J., Vplot: **SEP**'s plot language: SEP-**60**, 349–389.

Dellinger, J., 1989, Why does **SEP** still use Vplot?: SEP–**61**, 327–335.

Index